Enhancing Empirical Research for Linguistically Motivated Precision Grammars

PhD Thesis

Antske Sibelle Fokkens

Presented to the Department of Computational Linguistics,
Universität des Saarlandes
in fulfillment of the requirements
for the degree of Doctor of Philosophy

Saarbrücken 2014

Deutsche Zusammenfassung

Die Komplexität und Struktur natürlicher Sprachen fasziniert Sprachwissenschaftler schon seit Jahrzehnten (Chomsky, 1957; Bierwisch, 1963, u.a.). Diese Komplexität zeigt sich auch in den syntaktischen Analysen die natürliche Sprachen modellieren sollen. Bierwisch stellte schon in 1963 Folgendes fest:

> Eine systematische Überprüfung der Implikationen einer für natürliche Sprachen angemessenen Grammatik ist sicherlich eine mit Hand nicht mehr zu bewältigende Aufgabe. Sie könnte vorgenommen werden, indem die Grammatik als Rechenprogramm in einem Elektronenrechner realisiert wird, so daß überprüft werden kann, in welchem Maße das Resultat von der zu beschreibenden Sprache abweicht. (Bierwisch, 1963, p. 163)

Eine der Ursachen für die Komplexität liegt darin, dass sprachliche Phänomene sich gegenseitig beeinflussen. Rechenprogramme können eingesetzt werden um zu überprüfen ob eine Analyse nicht nur dem Phänomen, das sie modellieren soll, gerecht wird, sondern auch ob sie mit den anderen Eigenschaften der Grammatik ein kohärentes Modell bildet. Diese Überprüfung kann aber nur die Analysen betrachten, die bei der Entwicklung der neuen Analyse bereits in der Grammatik integriert waren.

Im Allgemeinen kann für ein bestimmtes Phänomen meist mehr als eine Analyse gefunden werden, die das Verhalten des Phänomens modellieren kann. Die Wechselbeziehung zwischen Analysen unterschiedlicher Phänomene ist eines der Kriterien die dabei helfen die richtige, oder am besten geeignete, Analyse zu finden. Die Wahl zwischen alternativen Analysen wird deswegen

von den schon entwickelten Analysen beeinflusst, wobei zwei (zusammenhängende) Probleme festgestellt werden können. Erstens ist es wegen der Komplexität der Grammatik nicht möglich Phänomene, die noch nicht analysiert worden sind, auf gleicher Ebene zu betrachten, wie bereits analysierte Phänomene. Zweitens, wenn ein Grammatikentwickler überprüfen möchte, ob eine neue Analyse richtig mit dem Rest der Grammatik interagiert, werden alternative Möglichkeiten für die Phänomene, die schon analysiert worden sind, nicht betrachtet. Es ist deswegen oft unklar,0 ob eine neue Analyse tatsächlich am Besten mit den *Phänomenen* interagiert oder nur mit den bisher ausgewählten *Analysen* der Phänomene.

Die Reihenfolge, in der Phänomene betrachtet werden, beeinflusst deswegen die Grammatik. Es ist bekannt, dass sich grammatische Phänomene gegenseitig beeinflussen. Wir wissen auch, dass in der Regel mehrere Analysen für ein Phänomen möglich sind. Die Erkenntnis, dass diese Tatsachen zusammen zu einer neuen Herausforderung führen, ist meines Wissens zuerst in Publikationen formuliert worden, die Teil dieser Arbeit darstellen.

Diese Arbeit stellt eine neue Methodologie für Grammatikentwicklung vor, welche zur Lösung dieses Problems beiträgt. Die Grundidee ist neue Analysen nicht direkt in einer Grammatik zu implementieren, sondern in einer *Metagrammatik*. Diese Metagrammatik kann alternative Analysen enthalten und daraus Grammatiken *generieren*, die für jedes implementierte Phänomen eine der alternativen Analysen enthalten. Es wird dadurch möglich, neue Analysen mit alternativen Möglichkeiten aus der Vergangenheit zu überprüfen und Entscheidungen über eine bestimmte Analyse auszusetzen bis genügend Hinweise vorliegen, dass sie zur besten Modellierung des Phänomens führt.

Diese Methodologie liegt der Implementierung von CLIMB zugrunde.[1] Sie ist mittels einer Metagrammatik des Deutschen evaluiert worden, die alternative Analysen für Wortstellung und Hilfsverben enthällt. Die Ergebnisse zeigen, dass CLIMB neue Möglichkeiten für empirische Studien mit Gramma-

[1]"Comparative Libraries of Implementations with a (grammar) Matrix Basis", im Deutschen: Vergleichende Programmbibliotheken von Implementierungen mit einer Basis in der (Grammatik) Matrix.

tiken eröffnet.

Die Beobachtung, dass syntaktische Analysen von der Reihenfolge der Behandlung von Phänomenen beeinflusst werden, und die von CLIMB gebotenen Mittel dieses Problem zu reduzieren, sind die wichtigsten Leistungen dieser Arbeit. CLIMB bietet jedoch zusätzliche Vorteile, die innerhalb der Grammatikentwicklungsumgebung (DELPH-IN) und sogar innerhalb der angewendeten syntaktischen Theorie (Kopfgesteuerte Phrasengrammatik, auch bekannt als "Head-driven Phrase Structure Grammar" oder HPSG) nicht vorhanden waren. Dazu enthält diese Arbeit mehrere Untersuchungen über mehrsprachige Grammatikentwicklung. Die nächsten Seiten bieten eine Übersicht über die Inhalte der einzelnen Kapitel.

Im **ersten Kapitel** wird die oben beschriebene Hauptproblemstellung tiefergehend behandelt. Nachdem das Thema vorgestellt worden ist, werden einige allgemeine Fragen der Syntaxforschung besprochen, insbesondere die Frage ob syntaktische Modelle als wissenschaftliche oder eher philosophische Modelle betrachtet werden sollten. Ich stelle dabei fest, dass es möglich ist wissenschaftliche, empirische Experimente durchzuführen wobei überprüft wird, ob ein bestimmtes syntaktisches Modell tatsächlich grammatische Sätze analysieren, und Äußerungen, die von Muttersprachlern für ungrammatisch gehalten werden, ausschließen kann. Dazu gibt es auch die Möglichkeit mit Muttersprachlern zu überprüfen, ob Äußerungen semantisch identisch sind, wobei festgestellt werden kann, ob syntaktische Analysen zur richtigen semantischen Darstellung führen. Es gibt aber in den meisten Fällen mehrere mögliche Analysen für syntaktische Phänomene, die eine Überprüfung der Grammatikalität und semantischen Gleichwertigkeit bestehen können.

Syntaktische Theorien verwenden deswegen zusätzliche Kriterien um zu bestimmen welche die richtige (oder bessere) Analyse ist. Hierbei spielen Kriterien wie Eleganz oder Einfachheit eine Rolle genauso wie die Frage, ob eine Analyse auch erklären kann, wie das Phänomen sich in anderen Sprachen darstellt oder ob die Analyse eine psychologische Erklärung für das Phänomen bietet. Diese Kriterien können zur Zeit nicht oder nur schwierig auf wissenschaftliche Weise überprüft werden. Meiner Meinung nach sollten solche Kriterien als philosophische Annahmen betrachtet werden. Ich schließe daraus, dass syntaktische Modelle teilweise auf philosophische Annahmen basieren und deswegen eher als philosophische Modelle gesehen werden sollten. Diese Feststellung bedeutet aber nicht, dass empirische Beobachtungen keine wichtige Rolle in der Syntaxforschung spielen. Das Ziel der Syntaxforschung ist die Struktur der Sprache so gut wie möglich zu modellieren. Die Methode die in dieser Arbeit vorgestellt wird, erlaubt nicht nur besser zu überprüfen ob eine Analyse am besten mit empirischen Beobachtungen übereinstimmt, sondern auch, ob sie die beste Lösung bietet unter Verwendung anderer Vergleichskriterien.

Das **zweite Kapitel** beschreibt den Hintergrund und Kontext der CLIMB Methodologie. Die Grammatiken die von CLIMB Metagrammatiken generiert werden, sind HPSG Grammatiken die innerhalb der DELPH-IN[2] Initiative klassifiziert werden. Diese Initiative ist bemüht, frei verfügbare sprachanalytische Anwendungen zu bauen, die linguistische Analysen, die auf HPSG basiert sind, verwenden.

Das Kapitel beschreibt zuerst die Grundprinzipien von HPSG, wobei auch der in HPSG verwendete Formalismus (getypte Merkmalstrukturen) betrachtet wird. Der Beschreibung der theoretischen Eigenschaften von HPSG folgt eine Einführung von DELPH-IN. In der Beschreibung von DELPH-IN Grammatiken wird insbesondere darauf geachtet, welche Unterschiede zwischen theoretischen HPSG-Grammatiken und denjenigen Grammatiken bestehen, die als Teil der DELPH-IN Initative implementiert worden sind. Außerdem werden die Algorithmen erklärt, die zum Parsen und Generieren von natürlichen Sprachen mit DELPH-IN Grammatiken verwendet werden. Zum Schluß bietet dieses Kapitel eine ausführliche Beschreibung der LinGO Grammar Matrix (die Grammatikmatrix).

Die Grammatikmatrix bietet Grammatikentwicklern ein Sprungbett um mit neuen Grammatiken anzufangen. Sie besteht aus einem statischen Kern und einer dynamischen Komponente. Der statische Kern enthält allgemeine (nicht-sprachspezifische) Definitionen von linguistischen Phänomenen und bietet dazu Voreinstellungen mit denen die Grammatik mit dem bestehenden DELPH-IN Parser und Generator verwendet werden kann. Die dynamische

[2]DEep Linguistic Processing in HPSG INitiative

Komponente enthält sprachspezifische Analysen, die Nutzer mittels eines Fragebogens aktivieren können. Der Grammatikentwickler definiert linguistische Eigenschaften in diesem Fragebogen und auf Basis davon wird eine kleine Anfangsgrammatik generiert, die Analysen für die definierten Eigenschaften enthält. Die Idee für CLIMB ist während der Arbeit an der dynamischen Komponente der Grammatikmatrix entstanden.

Im **dritten Kapitel** wird die Beziehung zwischen der Grammatikmatrix und CLIMB weiter ausgeführt. Die Grammatikmatrix lässt Entwickler linguistische Eigenschaften als Eingabe definieren und bietet eine Anfangsgrammatik, die von Hand weiterentwickelt werden kann. Die dynamische Komponente, die die Anfangsgrammatik generiert, bleibt vor dem Grammatikentwickler verborgen. CLIMB hingegen legt den Hauptfokus auf die dynamische Generierungskomponente. In Ihr findet der Grossteil der Grammatikentwicklung statt. Die Generierungskomponente wird so zu einer Metagrammatik, aus der verschiedene konkrete Grammatiken generiert werden können. Die Metagrammatik kann alternative Analysen für die gleichen Phänomene enthalten, die dadurch systematisch verglichen werden können.

Die CLIMB Metagrammatiken, die als Teil dieser Arbeit implementiert worden sind, sind als prozedurale Funktionen in Python implementiert. Es kann für Grammatikentwickler, die es gewohnt sind ihre Grammatiken deklarativ zu programmieren, ein Hindernis sein, auf prozedurale Programmierung umzusteigen. Ich habe deswegen auch eine deklarative Variante von CLIMB entwickelt, die es Grammatikentwicklern erlaubt ihre Metagrammatik deklarativ im gleichen Formalismus wie die normale Grammatik zu programmieren. Diese deklarative Variante bietet weniger Flexibilität als die prozedurale Variante. Sie hat jedoch mehrere Vorteile, die nicht vom Standardformalismus geboten werden. Erstens können Eigenschaften eines bestimmten Typs an unterschiedlichen Stellen in der Metagrammatik definiert werden, da das CLIMB Generierungsprogram die Eigenschaften automatisch zu einer gemeinsamen Definition zusammenfügt. Zweitens können in Typdefinitionen abgekürzte Pfade verwendet werden, die von einem Pfadergänzungsalgorithmus während der Generierung der Grammatik vervollständigt werden. Drittens können Grammatikentwickler in der Metagrammatik widersprüchliche Eigenschaften implementieren, was für die Entwicklung alternativer Analysen oder Variationen für unterschiedliche Sprachen nützlich sein kann. Es muss dabei allerdings beachtet werden, dass widersprüchliche Eigenschaften bei der Generierung nicht in der gleichen Grammatik vorkommen.

Zusätzlich zu CLIMB gibt es weitere Software, die als Teil dieser Arbeit entwickelt worden ist. Der "Spring Cleaning" Algorithmus nimmt eine DELPH-IN Grammatik als Eingabe und kann Teile der Grammatik, die auf keine Weise Einfluss auf die Kompetenz der Grammatik haben, identifizieren und entfernen. Die Kompetenz der Grammatik ist deren Fähigkeit, eine bidirektionale Zuordnung zwischen Äußerungen und ihren semantischen Darstellungen herzustellen. Dazu gibt es noch drei Algorithmen, die sich mit Pfadergänzung, Pfadabkürzung und dem Identifizieren der Geometrie von Merkmalstrukturen beschäftigen. Diese Algorithmen unterstützen den Entwickler bei der Verwendung abgekürzter Pfade in deklartivem CLIMB.

Die CLIMB Software und verwandte Algorithmen bieten weitere Vorteile für die Grammatikentwicklung außer der oben beschriebenen Möglichkeit alternative Analysen nebeneinander zu unterstützen. Die Methodologie erhöht die Modularität der Grammatik, sie unterstützt die parallele Entwicklung von Grammatiken für verwandte Sprachen und vereinfacht die Entwicklung und Unterstützung alternativer Grammatiken, die dialektale Änderungen oder unterschiedliche Anwendungen unterstützen. Die verwandten Algorithmen erlauben es, Eigenschaften der Grammatik zu extrahieren und zu untersuchen. Für mehrere dieser Vorteile gilt, dass sie bereits in anderen Grammatikentwicklungsumgebungen vorgeschlagen worden sind. CLIMB ist allerdings das erste Framework, welches diese Vorteile im Rahmen von DELPH-IN bietet. Meines Wissens nach gilt dies teilweise auch für die Entwicklung von HPSG-Grammatiken im Allgemeinen.

Die Methodologie ist mittels gCLIMB, einer Metagrammatik, die Analysen des Deutschen enthält, evaluiert worden. Das **vierte Kapitel** beschreibt die Analyse deutscher Wortstellung und Hilfsverben ,die in gCLIMB vorhanden sind. Zuerst werden die Analysen vorgestellt, die in der theoretischen HPSG für deutsche Wortstellung und Hilfsverben vorgeschlagen worden sind. Danach werden alternative Analysen erläutert, die von Bender (2008a) und Bender (2010) für die australische Sprache Wambaya vorgeschlagen worden sind und auch für das Deutsche verwendet werden können. Hierbei ist besonders die alternative Analyse für Hilfsverben interessant, da sie effizienter ist, wenn sie von den DELPH-IN Parsing- und Generierungsalgorithmen für Satzanalyse oder Generierung verwendet wird. Das Kapitel erklärt die Ursache der niedrigen Effizienz der HPSG-Standardanalyse für Hilfsverben und illustriert dies an einem Beispiel. Es muss dabei beachtet werden, dass die alternative Analyse nicht ohne Weiteres mit der originalen Analyse für deutsche Wortstellung kombiniert werden kann. Die heutige Implementierung dieser Analyse in gCLIMB kann nur mit der alternativen Wortstellunganalyse in einer Grammatik kombiniert werden. Diese Tatsache zeigt, dass Entscheidungen, die am Anfang der Grammatikentwicklung getroffen werden, die Möglichkeiten für zukünftige Analysen beeinflussen: die allgemeine Akzeptanz der Standardanalyse für deutsche Wortstellung in HPSG hat dazu geführt, dass es nur eine Möglichkeit zur Analyse der Hilfsverben zu geben schien.[3]

Im **fünften Kapitel** wird die CLIMB Methodologie evaluiert. Es werden zwei Aspekte untersucht, die bei der Bewertung der Methodologie eine Rolle spielen. Erstens, die Frage, ob CLIMB für langfristige Grammatikentwicklungsprojekte genutzt werden kann. Zur Untersuchung dieses Aspektes wurde gCLIMB weiterentwickelt, so dass es alle Phänomene enthielt, die auch im Regressionskorpus von Cheetah (Cramer, 2011) enthalten waren. Es wird eine Übersicht aller behandelten Phänomene in gCLIMB präsentiert, die auch die Unterschiede zu den Analysen in Cheetah erläutert. Gleichzeitig zeigt diese Übersicht auch die Komplexität der Analysen in gCLIMB auf. In der dazugehörigen Diskussion wird argumentiert, dass es nicht möglich ist, verschiedene Methodologien der Grammatikentwicklung nach wissenschaftlichen Standards miteinander zu vergleichen. Des Weiteren wird dargelegt, wie gCLIMB eine große Bandbreite linguistischer Phänomene abdeckt, und dass es sich für die Entwicklung großer Grammatiken eignet. gCLIMB wurde in der Hälfte der Zeit entwickelt, die für Cheetah's Grundgrammatik benötigt wurde. In dieser Zeit wurden zusätzlich noch cross-linguistische Variationen

[3]Hinrichs and Nakazawa (1994) haben die Standardanalyse für Hilfsverben ins Deutschen als eine Notwendigkeit eingeführt.

mittels gCLIMB erforscht und ein komplexer Standard der Generierung der semantischen Ausgabe implementiert (Minimal Recursion Semantics). Diese Ergebnisse deuten darauf hin, dass die Nutzung von CLIMB die Grammatikentwicklung beschleunigen könnte, mindestens aber nicht übermäßig verlangsamt.

Der zweite Teil der Evaluierung vergleicht die Effizienz der unterschiedlichen Analysen. Diese Untersuchung bestätigt in einer Studie über deutsche Satzanalyse und niederländische Sprachgenerierung, dass die alternative Analyse von Bender (2010) effizienter ist als die Standardanalyse. Der Einfluss der niederländischen Wortstellung auf Sprachgenerierung wurde in dieser Studie auch untersucht. Die CLIMB Methodologie ermöglicht diese Art von Studien, weil sie es erlaubt, die Eigenschaften der Grammatik und ihre Analysen einfach und schnell anzupassen. Der dritte Teil des Kapitels präsentiert Möglichkeiten für weitere Untersuchungen mit CLIMB. Insbesondere wenn Analysen aus anderen HPSG Grammatiken für das Deutsche zu gCLIMB hinzugefügt werden, kann die Interaktion von unterschiedlichen Analysen dieser Grammatiken und ihr Einfluß auf die Effizienz untersucht werden. Ein Schritt in diese Richtung wurde bereits mit der Integration des Lexikons aus Cramer (2011) in gCLIMB gemacht (sehe Sektion 5.3). Die bisherige Ergebnisse zeigen, dass den Eigenschaften des Lexikons in der Struktur der Grammatik Rechnung getragen werden sollte.

Das **sechste Kapitel** erarbeitet die multilingualen Aspekte von CLIMB und DELPH-IN Grammatiken. Im ersten Teil des Kapitels werden die Möglichkeiten untersucht, CLIMB für multilinguale Grammatikentwicklung zu verwenden. Die erste Version von gCLIMB hat neben dem Deutschen auch das Niederländische und Dänische unterstützt. gCLIMB enthielt damals nur die Grundprinzipien von Wortstellung, intransitiven, transitiven und ditransitiven Verben. Während der Weiterentwicklung der Grammatik des Deutschen, wurden dann die anderen Sprachen nicht weiter behandelt. In Kapitel sechs wurde evaluiert ob gCLIMB verwendet werden kann, um für andere germanische Sprachen Grammatiken zu entwickeln. Dabei wurde Niederländisch, Dänisch und Nordfriesisch untersucht. Ich habe eine niederländische Version der deutschen Entwicklungsdaten gemacht und für Dänisch und Nordfriesisch wurden mir Evaluierungsdaten von Kollegen zu Verfügung gestellt. Alle Grammatiken wurden innerhalb weniger Tage erstellt. Während diese Grammatiken die meisten Daten richtig analysieren könnten, wurden bei allen noch Fehler beobachtet, die (teilweise große) Überarbeitungen der Analysen erfordern. Dieses Ergebnis zeigt einerseits, dass multilinguale Grammatikentwicklung nur dann funktioniert, wenn die Unterschiede zwischen den Sprachen während der ganzen Entwicklung betrachtet werden. Andererseits wäre es ohne gCLIMB aller Wahrscheinlichkeit nach nicht möglich gewesen, in so kurzer Zeit ähnliche Ergebnisse zu erreichen.

Die Idee, CLIMB für multilinguale Grammatikentwicklung zu verwenden und dabei stets die linguistische Variationen in einer Gruppe von Sprachen zu betrachten, wird in dem neuen Projekt "SlaviCLIMB" angewendet. Dieses Projekt folgt aus dem PaGES Projekt für multilinguale Grammatikentwicklung für Slavische Sprachen. SlaviCLIMB fügt eine dynamische Komponente zum Projekt hinzu und ich beschreibe wie die Methodologie neue empirische Verifikationen von theoretischen linguistischen Hypothesen erlauben wird. SlaviCLIMB kann im Moment die allgemeine slavische Grammatik, die von Avgustinova and Zhang (2009) innerhalb von PaGES entwickelt worden ist und die "Russian Resource Grammar" (Avgustinova and Zhang, 2010) generieren.

Der zweite Teil des Kapitels untersucht den Einfluss des statischen Kerns der Grammatikmatrix auf einzelne Grammatiken. Die Entwicklung der gCLIMB-grammatik hat zur größten Überarbeitung der statischen Komponente seit ihrer Erstveröffentlichung geführt. Diese Überarbeitungen bestehen teils aus Korrekturen von sogenannten Bugs und teils aus Korrekturen von Implementierungen, die englischspezifisch sind und nicht ins Deutsche übertragen werden können.[4] Seit der Erstveröffentlichung sind mehrere Grammatiken mit Hilfe der Grammatikmatrix entwickelt worden und es ist unwahrscheinlich, dass keiner der Grammatikentwickler dabei Fehlern in der Grammatikmatrix oder sprachabhängigen Problemen begegnet ist. Ich habe deswegen untersucht inwiefern diese Grammatiken die Implementierungen aus der Grammatikmatrix verwenden. Der erste Schritt in dieser Studie bestand dar-

[4]Die ERG, eine Grammatik für Englisch, ist die Hauptquelle für die Grammatikmatrix.

aus den “Spring Cleaning” Algorithmus auf die Grammatiken anzuwenden, damit festgestellt werden kann, welche Teile der statischen Komponente Einfluß auf die Grammatik haben. Danach habe ich die Änderungen und neue Definitionen in den Grammatiken analysiert. Diese Studie hat gezeigt, dass einige Grammatiken die gleichen Überarbeitungen in der Grammatikmatrix enthalten, die von unterschiedlichen Grammatikentwicklern unabhängig von einander gemacht worden sind. Dieses Ergebnis zeigt, dass nützliche Einsichten der Grammatikentwickler nicht (immer) an die Entwickler der Grammatikmatrix weiter gegeben werden. Die Grammatikmatrix könnte von einer engeren Zusammenarbeit mit einzelnen Entwicklern profitieren.

Im **siebten Kapitel** werden andere Initiativen für multilinguale Sprachentwicklung, Austausch zwischen Grammatiken, gemeinsame Implementierung und Verbesserung der Modularität von Grammatiken beschrieben. Mehrere der Initativen entwickeln Grammatiken in anderen Theorien und Formalismen als HPSG. Hier werden auch diese Theorie und Formalismus kurz vorgetellt und es wird erklärt, wie sie die Entscheidungen im Ansatz der gemeinsamen Implementierung oder multilingualen Sprachentwicklung beeinflussen. Jede Initiative wird mit CLIMB verglichen wobei der wichtigste Unterschied ist, dass CLIMB die einzige Initiative ist, die sich mit alternativen Analysen und empirischen Vergleichen der Analysen über längere Zeit auseinandersetzt.

Zusammenfassend kann gesagt werden, dass das Hauptaugenmerk dieser Arbeit dem verfrühten Verwerfen alternativer Analysen in der Grammatikentwicklung gilt. CLIMB ist nach meinem Kenntnisstand die erste Arbeit, die dieses Problem zum Gegenstand hat. Es bietet die Möglichkeit mehrere alternative Analysen zu implementieren und zu erhalten, während die Grammatik weiterentwickelt wird. Diese Fähigkeit erlaubt es, die Interaktion zwischen Analysen für Phänomene, die zu unterschiedlichen Zeiten erstellt wurden, miteinander zu vergleichen. So können Entscheidungen bei der Grammatikentwicklung aufgrund empirischer Belege getroffen werden. Des Weiteren bietet CLIMB und die unterstützende Software “Spring Cleaning” die Grundlage für weitere Studien zur Interaktion zwischen Grammatikanalysen, Lexika und multilingualer Sprachentwicklung.

Acknowledgements

First and foremost, I would like to thank my supervisors, Hans Uszkoreit and Emily Bender.

Hans, thank you for your support and feedback, but mostly for encouraging me to find my own way and giving me the space to do so. I enjoyed our conversations and discussions these last years.

Emily, thank you for your guidance, encouragement and enthusiasm. Your work on the Grammar Matrix, welcoming me in your group and the intense collaboration despite the distance has truly enabled the research carried out in this thesis. I am grateful for all insightful and useful feedback and help you have given me throughout these years through inspiring discussions on big questions and your eye for detail in my writings. I have learned many things during this journey, but how you manage to always provide useful support answering e-mails in negative time is still a mystery to me.

The Matrix developers played an important role in this thesis with their feedback, questions and support during weekly meetings. Safiyyah, Joshua, Glen, Varya: Thanks! In particular, I would like to thank Scott Drellishak for his help when I started to use the Grammar Matrix customisation system. Michael Goodman developed several tools that have been helpful and was always managed to provide quick responses to any questions I have had.

Special thanks go to Laurie Poulson. I have fond memories of her wit, wisdom and friendship. Her many questions and detailed feedback on my work have led to great improvements of the first papers written on CLIMB. She would have undoubtely found many ways to improve this thesis.

Yi Zhang and Tania Avgustinova: it has been great exchanging ideas with you and develop the vision of SlaviCLIMB. May many DELPH-IN grammars for Slavic languages be generated!

Working within DELPH-IN has been a joy. I am grateful to all entire DELPH-IN participants for the interaction and good times during the last eight DELPH-IN meetings. I'd particularly like to thank Francis Bond, Dan Flickinger, Montse Marimon, Petya Osenova, Rui Wang, Ann Copestake and Sanghoun Song for discussions and their interest in CLIMB.

Thanks to Christoph Zwirello and Emillia Ellsiepen for correcting and helping me polish my German summary.

Participating in DELPH-IN meetings, visiting the University of Washington and Brown University and working on my PhD at Saarland University was made possible thanks to the funding of the IRTG-PIRE program. I am grateful for having received this opportunity. I have had many good teachers and researchers who supported and encouraged me during my undergraduate studies and helped me to receive this grant. I would particularly like to thank Claude Muller, Joan Busquets, Kim Gerdes, Line Mikkelsen and Valia Kordoni.

I would also like to thank my colleagues at (and around) VU University for their encouragement. Niels, Serge, Piek, Susan, Guus, Marieke, Agata, Isa, Hennie, Roxane, Rubén, Marten, Selene, Emiel, Paul, Wouter, Jan, Willem, Thomas, Victor, Laura and Lora, I really enjoy working with you and am looking forward to continuing in the coming post-thesis time.

The IRTG was a fun crowd to be part of. I have great memories of the entire group, but would particularly like to thank Bec, Bart, Micha, Milla, Judith K., Barbara and Alexis for the shared lunches, drinks, trips and friendship. Anne-Christin, Markus, Judith B., Herr and Frau Eichner have also given me many good memories of Saarbrücken during my PhD. Peter Adolphs, Tina Klüwer and Kathrin Eichler helped to make my stay in Berlin a pleasant one and I'd like to thank David McClosky, Micha Elsner, Matt Lease, and Will Headden for making me feel at home in Providence.

Finally, I would like to thank my family for their support and patience.

Antske Fokkens, Amsterdam July 24 2014

Contents

List of Tables

List of Figures

Dedication

Chris, dziękuję za Twoją cierpliwość i wsparcie, za to, że wciąż na nowo mnie zaskakujesz, i czynisz szczęśliwą.

Chapter 1

Introduction

eine Grammatik [ist] eine Hypothese über die Struktur einer Sprache.[1]

Bierwisch (1963), p. 163

Grammars of natural language are highly complex objects (Bierwisch, 1963; Müller, 1999; Bender, 2008a; Bender et al., 2011). This complexity is reflected in formal analyses found in both syntactic theory and computational grammars. In particular, there are two factors that make it notoriously difficult to make strong assertions about analyses for natural language grammars. First, syntactic phenomena interact and therefore also their analyses interact. Second, typically more than one formal analysis can account for a given phenomenon. Both of these properties are well known. However, an additional challenge that results from the combination of these two properties has largely been ignored in previous work. Grammar engineering is an incremental process. A syntactician or grammar engineer often needs to choose between analyses without conclusive evidence as to which analysis works best. Because analyses interact, this decision will affect possible analyses for phenomena handled in the future.

The restrictions imposed by the choice of past analyses are not always easy

[1] a grammar is a hypothesis about the structure of a language.

to identify, especially after a significant amount of time has passed and the influence is the result of the interaction of several choices. As such, syntactic theories as well as computational grammars for natural language are influenced by the order in which phenomena are treated (Fokkens, 2011a).

The work presented in this thesis is, to my knowledge, the first to point out the challenge outlined above and to propose a solution that addresses this problem. The key idea, as introduced in Fokkens (2011a), is that alternative plausible analyses for linguistic phenomena are added to a metagrammar. I use *metagrammar* as a generic term to refer to a system that can generate computational grammars. This metagrammar can generate all possible combinations of these analyses automatically, creating different versions of a grammar that cover the same phenomena. The engineer can test directly how competing analyses for different phenomena interact, and determine which combinations are possible (after minor adaptations) and which analyses are incompatible. Alternative analyses can be maintained while new analyses are developed, thus reducing the influence of the order in which phenomena are investigated. The interaction between alternative analyses with phenomena can be tested empirically after new analyses have been added to the grammar. As such, the metagrammar engineering approach proposed in this thesis provides a more systematic method for grammar development that enhances empirical experiments on the grammar's behaviour.

The observation that the interaction of different analyses in future states of the grammar cannot be predicted and the proposal to address this problem using metagrammar engineering as a general methodology form the main contribution of this thesis. In particular, this thesis introduces CLIMB[2] a metagrammar environment that can generate HPSG[3] implementations. In addition to the possibility of **maintaining alternative analyses in parallel**, CLIMB supports a number of other advantageous properties found in related work using metagrammars or similar approaches (Candito (1998); Duchier et al. (2005); King et al. (2005); Ranta (2009); Bender et al. (2010),

[2]Comparative Libraries of Implementations with a (grammar) Matrix Basis (Fokkens, 2011a; Fokkens et al., 2012a; Fokkens and Bender, 2013)

[3]Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994)

among others). The methodology increases **modularity**, it supports **multilingual** grammar development and adaptations for alternative **dialects** as well as alternative versions for different **applications**. The setup facilitates a **phenomenon-based organisation**. This provides further flexibility, because it is easy to create alternative grammars that **include or exclude rare phenomena**. Furthermore, increased modularity and a phenomenon-based organisation facilitates collaborate efforts of grammar engineering. Finally, additional processing tools provide support for **investigating properties** of grammars generated by the metagrammar.

The rest of this chapter is structured as follows. Section 1.1 addresses the question of whether research on syntax can be considered to be scientific research. It discusses both empirical observations and philosophical assumptions used when designing syntactic models. Section 1.2 provides a more elaborate illustration of the idea behind grammar engineering and outlines the assumptions adopted for the grammars developed as part of this thesis. Finally, an overview of the chapters of this thesis and its main contributions are presented in Section 1.3.

1.1 Syntax, Science and Philosophy

This section discusses scientific and philosophical aspects of syntactic theory. It serves two purposes. First, I would like to elaborate on the question of whether syntactic research is science or whether it is founded on philosophical assumptions rather than observations. The main contribution of this thesis is that it provides a method to test grammars of natural language more systematically. In order to establish what this might mean for researchers working on grammars or syntactic theory, it is important to establish what these researchers aim to model. Second, this section provides an overview of the most important criteria used in syntactic research to determine the quality of an analysis. They are related to the discussion about science and philosophy, because both evidence based on observations (which can be scientific) and evidence based on assumptions (which are philosophical) are used in syntax. Moreover, what syntacticians aim to model determines which criteria they use to compare analyses. The issues discussed in this section thus relate to the underlying question of what a grammar is. In Section 1.2.2, I will present the observations and assumptions that were followed to develop the grammars for this thesis.

1.1.1 Empirical Science and Syntax

Even though philosophers of science seem to agree more or less on which disciplines are part of science and which are not, definitions of science vary (Hansson, 2012). Popper (1962) proposes falsifiability as a criterion for a statement to belong to empirical sciences: "statements or systems of statements, in order to be ranked as scientific, must be capable of conflicting with possible, or conceivable observations" (Popper, 1962, 39). Kuhn (1974) claims that the status of science is related to its puzzle-solving nature: if a prediction fails, the scientist will try to solve this by verifying measurements or a revision of the theory. Lakatos emphasises that a scientific theory should make progress: it is not the character of a theory in isolation that makes a research program scientific, but that it consists of a series of theories replacing one another making new predictions that are confirmed (Lakatos, 1970, 1974a,b, 1981). These visions on science have in common that they all rely on evidence from observations. In other words, empirical verification (or falsification) is an essential aspect of science.

Linguistics as science

Linguistics has been classified as a science by prominent linguists starting with Saussure (1916). Despite the fact that critical differences between linguistics and natural science have been pointed out by several authors (e.g. Botha (1970); Itkonen (1978, 1981); Wasow (1985); Yngve (1986, 1996); Koster (2005); Eddington (2008); Coleman (2009)), this is still a widely held view.[4] A full discussion and overview of the development of linguistics goes beyond the scope of this thesis. Because of the importance of the topic and, in my view, insufficient attention that it has received, I will provide a short discussion of some problematic aspects in scientific approaches used in linguistics and syntactic research in particular.

Coleman (1999) points out that the problems of defining linguistics as *the science of language* start with the object of study. *Language* is not a concrete object that can be observed, it is an abstraction in itself. It is therefore not possible to create a theory of language which can be proven wrong or right by observing language. This general confusion about linguistic observations and relevant data forms one of the main problems in linguistic research. Coleman (2009) provides clear demonstrations, partially inspired by Yngve (1996), of where abstractions of *observations* can be made in linguistics, where abstractions are made based on *assumptions* (as often found in philosophy) and, the highly problematic case, where abstractions based on assumptions are taken to be concrete. A scientific theory is an abstraction of observations. More specifically, the theory must be falsifiable by observations, which must be identical every time an experiment is repeated under the same conditions. Confusion about observations and assumptions are found in several subdomains of linguistics. I limit the discussion to syntax and will argue that there are scientific methods and empirical observations that support syntactic research, but that often evidence is based on assumptions. In principle, this is unproblematic, as long as syntacticians are clear about the status of their model.

Syntax as science?

The view that syntax is science is not uncommon among syntacticians. This can be seen clearly in Chomsky's highly influential *Syntactic Structures*, where he compares a grammar of English to hypothetical constructs such

[4]See Coleman (2001) for a survey of authors referring to linguistics as a science in textbooks.

as electrons in physics (Chomsky, 1957, 49).[5] More recently, Larson's book *Grammar as Science* (Larson, 2010) states not only that grammar can be seen as a science, but that "syntax offers an excellent instrument for introducing students [...] to the principles of scientific theorizing and scientific thought" (p. *xiii*). The fact that there is no mention of aspects that may make research on syntax a less scientific undertaking shows that this is not considered a real issue by Larson, nor does he seem to expect to find much doubt from his intended audience.

In my opinion, syntacticians use scientific methods in their approach, but this does not automatically mean that syntax is science. Models of syntax are generally the result of the combination of observations and philosophical assumptions. I will elaborate briefly on the most important evidence in syntactic research: namely linguistic expressions accompanied by native speaker judgements.

Syntax: scientific data

In their introduction to syntactic theory, Sag and Wasow (1999) call languages "complex phenomena, which can be studied scientifically" (p.3). They explain that hypotheses about linguistic structure can be tested through facts of language. However, linguistic structure is as abstract as language itself, which would mean that a syntactic theory cannot be a scientific model of linguistic structure. This problem can either be addressed by defining syntax as a theory rather than a scientific model, or redefining the object of study of syntax.

If syntax is considered to propose scientific models, the question rises what it models. The answer to this question is simple: a scientific theory models what scientists observe. Syntacticians generally observe linguistic expressions provided with native speaker judgements. From this point of view, syntactic theory would be a model of native speaker judgements. A similar point is

[5]Wasow (1985) points out that Chomsky questions the scientific status in Chomsky (1982), but according to Wasow, this is "clearly a minority view" (Wasow, 1985, 486).

made by Coleman (2002), who explains that data in the form of a linguistic example marked as ungrammatical is a "*supposed* 'native-speaker judgment' of the item's ungrammaticality" (46, Coleman's italics and quotes). Coleman calls the judgement *supposed*, because the exact source of such observations is typically absent from linguistic papers.

Several authors have pointed out other pitfalls of using judgements of acceptability[6] and addressed the question of how grammaticality judgements can be carried out in an experimental setting (notably, Schütze (1996); Cowart (1997); Keller (2000); Schütze (2005)). Their findings can be summarised as follows. In order to conduct scientific experiments on acceptability judgements, syntacticians should control for confounding factors such as the influence of context on acceptability or the variation across speakers depending on education, social background and dialects or other local languages spoken in the speaker's environment. Moreover, the traditional approach of classifying examples as grammatical or ungrammatical, with occasionally a *?* for doubtful or an indication of speaker dependence has been criticised by Cowart (1997) and Keller (2000). Both of these works show that grammaticality is gradient with clearly observable differences in how ungrammatical speakers consider individual expressions. An experiment determining the comparative grammaticality of examples naturally requires a certain number of subjects to judge grammaticality. Details on how to carry out experimental syntax and more thorough analyses of native speaker judgements in this field can be found in Schütze (1996), Cowart (1997) and Keller (2000). For the current discussion it suffices to say that experimental methods have been developed that in principle allow syntacticians to gather data in a scientifically sound manner. It should however be noted that these methods are only followed in a minority of syntactic studies.

[6]I will not discuss the distinction between grammaticality and acceptability here. Suffice to say that acceptability (along with influencing factors) can be observed through tests with native speakers, grammaticality cannot.

1.1.2 Philosophical aspects of syntax

If syntax uses utterances marked by judgements as observations, it can be seen as a model of native speaker judgements. Syntactic models can then be tested by verifying whether they predict the correct judgements for new utterances. Models that correctly account for known utterances and make correct predictions about new utterances can be verified against two of the three levels of adequacy described by Chomsky (1965) for evaluating grammars: **observational adequacy** and **descriptive adequacy**. But how can we compare between different models that get predictions right? The third level of adequacy, **explanatory adequacy**, can be used to distinguish between analyses that fulfil the first two. This level of adequacy is concerned with how well a theory explains its observations. Furthermore, additional criteria are set to fully satisfy the needs for observational and descriptive adequacy. Finally, grammar engineering has additional criteria of its own regardless of whether the implemented grammars are meant to verify syntactic analyses or intended to be used in applications.

Though traditionally syntactic research focused on making the right predictions on native speaker judgements, more recently research have taken the link between utterances and their semantics into account. Theories such as Lexical Functional Grammar (Kaplan and Bresnan, 1982, LFG), HPSG and the Minimalist Program (Chomsky, 1995) also aim to describe how the semantics of a sentence is built up from its parts in addition to examining native speaker judgements. This criterion also plays an important role in implemented grammars that are based on these theories.

In addition, syntacticians take **simplicity or elegance** and **crosslinguistic applicability** into account when comparing the suitability of specific analyses and grammar engineers pay attention to efficiency. Whereas the importance of making correct predictions of native speaker judgements and relating the sentence's structure to its meaning are generally accepted, syntacticians differ in the importance they assign to these additional criteria. This difference can be explained by the fact that these criteria are founded on *philosophical assumptions* rather than observations.

It makes perfect sense to narrow the search for an appropriate model by imposing desiderata based on philosophical or practical arguments, especially when observations leave a multitude of possibilities for suitable theories. Researchers should, however, always clearly distinguish between decisions based on scientific observations and those based on philosophical assumptions. The more a model is validated by assumptions rather than observations, the more it becomes a philosophical model rather than a scientific model. I believe that models of language in syntactic theory should generally be seen as philosophical models of grammar. I will elaborate briefly on each of the adopted philosophical desiderata below.

Explanatory adequacy refers to the capability of the theory to explain observations. This requirement is generally addressed with regard to the overall syntactic theory, i.e. is the formalism appropriate? What are basic assumptions of the theory that do not directly reflect analyses? And, what are the available mechanisms that can be used to exclude unacceptable utterances? Depending on theoretical assumptions, the aforementioned criteria of simplicity and crosslinguistic applicability may be considered requirements of explanatory adequacy. This discussion can go a long way, starting with the question of what syntax is supposed to be modelling in the first place. I will therefore not address this aspect here, but refer the interested reader to Wasow (1985) who points out that the typical assumption is that syntax models children's language acquisition and explains that syntacticians do not examine the right data to investigate this.

On the level of specific analyses, the criterion of explanatory adequacy seems to be what prompts linguists to criticise analyses because it is "unlikely that speakers have this in their heads". Since there currently are no experiments that allow us to examine what linguistic properties speakers may have "in their heads", such argumentations are clearly philosophical and not scientific.[7] In my experience, such comments are often made to criticise analyses with

[7]This may, of course, change in the future as technology and psycholinguistic methods evolve.

a heavy technical component, where the relation between observations and analysis is not apparent. It therefore often is justified critique, where the problem lies more in lack of **simplicity or elegance** than “likelihood of being in the heads of speakers”. Eddington (2008) points out that simplicity cannot be used as evidence to identify which theory is correct. After all, there is no reason to assume that the truth should be simple rather than complex. Linguists support the idea that analyses should be simple, because language can be learned easily by humans. However, the difference in complexity of alternative analyses is relatively small compared to the complexity of the entire grammar as a system. Furthermore, simplicity still is a philosophical criterion under this explanation for reasons explained above: at present, it is not possible (yet) to investigate which models would make the syntax of a language too complex for it to be learnable by human beings. Nevertheless, aiming for the most simple model possible (or applying Ockham’s razor) is common practice in research. Adger (2003) explains that it is easier to work with fewer concepts and that Ockham’s razor is partially applied for methodological reasons. The merits of elegance and simplicity are indeed clear (e.g. easier to understand and test analyses, aiming to prevent unnecessary additions).

Syntacticians can have different opinions on the importance of **crosslinguistic application**. The main idea behind this criterion seems to be that an analysis that can be used to explain data from more than one language is more likely to say something about language in general. Most notably, syntacticians who support the notion of a *Universal Grammar* (UG) assign high importance to crosslinguistic applicability. Some syntacticians take the point of view that the basic structure for all languages should be the same. For instance, Richard Kayne criticised an analysis by Balkiz Ozturk, because she used left-branching structures for Turkish. Kayne argued that it would not make sense to adopt a structure for Turkish based on observations of Turkish (only).[8] The result of such assumptions is that analyses concerning phenomena with large typological variation tend to get highly complex. This

[8]This discussion took place at workshop *Theoretical Approaches to Disharmonic Word Orders*, Newcastle, 31 May 2009.

seems somewhat ironic given that the proposal of UG follows from the claim that language must be innate, because there would be no other way to explain why all children manage to learn language easily despite its complexity (Chomsky, 1965).

The argumentation around UG should clearly be placed in the philosophical sphere: it may be an observation that children learn language easily, but it is currently impossible to empirically test the hypothesis of an innate grammar of linguistic structure against, for instance, a more general hypothesis that the human brain can make complex generalisations unconsciously. Scholz and Pullum (2006) point out several ways in which tests typically used to prove that language is innate fail as scientific tests.

Crosslinguistic applicability can thus not be considered a scientific argument that makes an analysis more likely to be 'true'. If we accept that syntax is better placed at the level of philosophy and comparable to mathematical models, where motivations can be built on assumptions, the crosslinguistic aspect of analyses becomes an interesting aspect in linguistic investigation. Looking at analyses that can explain linguistic behaviour (in the sense that they make correct predications concerning grammaticality) across languages can provide deeper insight into language than looking at languages in isolation.

If we try to model language to understand why particular behaviour is observed in language or how people learn language, it is important to consider which parts of our model can apply to different languages. Moreover, by sharing analyses between languages, we can make progress in describing new languages faster. This makes crosslinguistic applicability particularly interesting for implemented grammars. On the other hand, care must be taken that existing analyses for previously described languages do not determine what new grammars look like, leading to a bias in grammar design towards languages that were implemented first.

1.1.3 Summary

I have argued above that native speaker judgements can be considered observations, provided that they are obtained under the right (controlled) conditions. Syntax could then be a scientific theory about knowledge speakers have to rate utterances. However, there are many decisions to be taken to arrive at such a model. Additional assumptions have been adopted by syntacticians to guide their research. By adapting these assumptions and using them to get evidence for their models, syntacticians have moved away from science towards a more philosophical approach. In my opinion, this does not degrade syntactic research in any manner. Creating precise, philosophical models of language forms a perfectly justifiable area of research. Moreover, research on syntax has primed syntacticians to try out various variations in language and obtain grammaticality judgements. This has contributed to our knowledge about language through observations.

Even if syntax is not science, this does not mean that syntactic research is not improved by researchers following more scientifically sound methods. A philosophical model of language increases in validity and thus becomes more interesting when based on more empirical evidence. In general, care should be taken when excluding analyses based on assumptions rather than empirical evidence. It is important to clearly state the criteria for choosing among alternatives and what role these criteria play in the theory. Using evidence based on philosophical ideas is not a problem as long as they are clearly defined as such and not confused with scientific evidence. The method proposed in this thesis not only allows the grammar writers to run more empirical tests for alternative analyses, but also allows them to examine which alternative fits their philosophical assumptions best in a more systematic manner. The following section describes how metagrammar engineering allows grammar developers to base their decision on more empirical evidence and introduces the assumptions made for the grammars developed as part of this thesis.

1.2 Metagrammar Engineering

The previous section has discussed the object of study in syntactic theory and presented several criteria that are used to decide among alternative analyses. An aspect that has not been addressed is the interaction between analyses. A correct analysis should not only be able to capture data correctly in isolation, it should also interact correctly with other analyses. Interaction between analyses is not always transparent. This is the main reason why grammars of natural language are so complex and why syntacticians tend to pick one analysis and stick to it. As early as 1963, Bierwisch (1963) pointed out that phenomena interact to a point where it is not feasible to make sure a model is correct without verifying this with help of a computer. This call has since then put to practice across frameworks, notably by Müller (1999), Bender (2010) and Kinyon et al. (2006). Bender et al. (2011) illustrate how grammar engineering, and the integration of regression testing, can help linguists manage the complexity of their models. This involves both testing the interactions between analyses and testing analyses against much larger collections of data than would be feasible by hand. Bender et al. (2011) illustrate the process with the diagram in Fig. 1.1.[9]

The process depicted in Fig. 1.1 is cyclic, with new phenomena being added on successive passes through the cycle. The test suites (extended with each pass) document the analyses that have been implemented and allow the linguist to ensure that later work does not break or invalidate what has gone before. This is a strength of the methodology and key to the ability of grammar engineering to facilitate linguistic hypothesis testing. However, when we view the process of grammar engineering in this light, it also becomes apparent that phenomena considered earlier in the development of a grammar and their analyses have an asymmetrical influence on analyses of phenomena developed later (cf. Bender (2008b)).

This asymmetrical influence is unfortunate: It is fairly common for a key

[9]Note that not all efforts of grammar engineering involve compiling a grammar and adapting analyses to make sure the grammar compiles. This is however the case for the grammars implemented as part of this thesis.

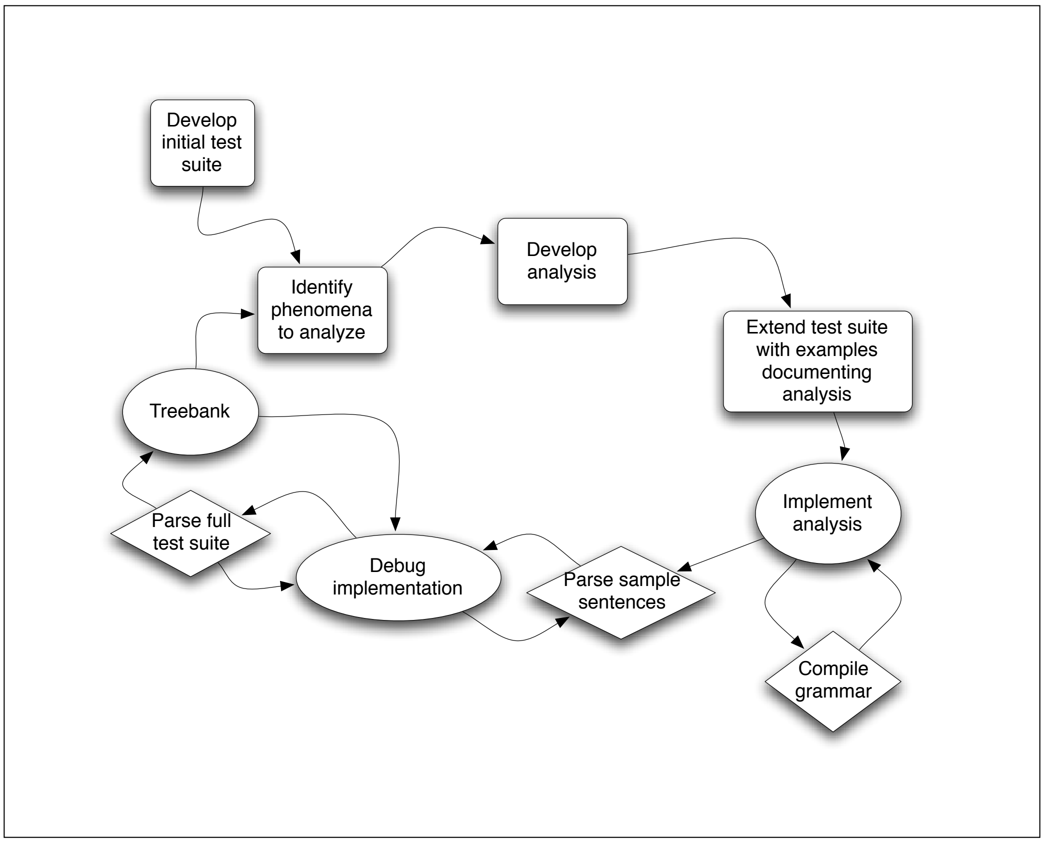


Figure 1.1: Grammar engineering workflow (Bender et al., 2011, p. 10)

phenomenon constraining the choice of analysis of another phenomenon to be only addressed after several further passes through the cycle. In the meantime, whichever analysis was chosen of the phenomenon implemented earlier may become deeply embedded in the growing grammar. This has several consequences: First, once an analysis becomes embedded in this way, it is easy to forget what alternative analyses were available. Second, the longer an analysis has been part of a grammar, the more other analyses are likely to depend on it in some way. This leads to scenarios where it becomes cumbersome or impractical to change an analysis, even when it is discovered to be suboptimal.

Challenges related to deeply embedded analyses are familiar to most grammar engineers working on large scale grammars. Francis Bond (p.c.) reports that it is often hard to identify parts of the grammar which relate to obsolete analyses no longer active in the Japanese grammar Jacy (Siegel and Bender, 2002). Montserrat Marimon (p.c.) reports that there are analyses in her Spanish grammar (Marimon, 2010) for clitics and word order that need revisions, but these phenomena interact with several other properties of the grammar. It would therefore be an elaborate undertaking to make these changes and they have been put on hold for now. Tracy King (p.c.) reports an ongoing discussion within ParGram (Butt et al., 2002) on whether adjectives have subjects or not. The English LFG grammar (Riezler et al., 2002) was changed a few times, but this was so time consuming that King decided to call the last change final.

Finally, even if a grammar engineer is inspired to go back and revise a deeply-embedded analysis, it is simply not possible to explore all alternative possibilities, that is, all the alternative analyses of the various interacting phenomena that might have been just slightly more desirable had the revised analysis been the one chosen in the first place. As a result, computational grammars are partially (or even largely) a product of the order in which phenomena are treated. For grammar engineers with practical applications in mind, this is undesirable because the resulting grammar may end up far from optimal. For grammar writers that use engineering to find valid linguistic analyses, the problem is even more serious: if there is a truth in a declarative grammar, surely, this should not depend on the order in which phenomena are treated (Fokkens, 2011a).

In order to escape from this asymmetrical influence problem, what is required is the ability to explore multiple development paths. As described in the next section, this is exactly what metagrammar engineering provides.

1.2.1 A Many-Model Interpretation

This thesis proposes to address the problem of asymmetrical influence by using a metagrammar. I will illustrate how a metagrammar in an explanation that is inspired by the many-world interpretation of quantum mechanics

Everett (1957); DeWitt (1972).[10] Quantum mechanics can predict the probability of a location of a photon and this prediction forms a wave function. However, as soon as the photon is observed, the probability function is reduced to one point and, according to the alternative Copenhagen Interpretation, the wave function collapses. The many-world interpretation rejects the idea that the wave function collapses, but maintains that the alternative worlds in which the photon is at another location than the one observed are real, implying that alternative histories and futures are real.

If we translate this vision to the grammar engineering scenario, we have a many-model interpretation, where each model considers different analyses to be correct. Each implementation decision we make places us in a given model. While making a decision in grammar engineering, the grammar engineers sets off on a specific road and the route that is taken (the order in which phenomena are considered, the choices made concerning their implementations) influences the final destination (the resulting grammar). However, unlike real life, where we are stuck in a specific world and cannot explore alternatives, we can look at alternative models for grammars. The only problem is that it is not feasible to have a clear picture of all consequences of a specific decision while following the traditional grammar engineering approach. But what if we could monitor more than one model at the same time? What if, instead of making a decision to commit to a specific model, we follow a couple of models for a while, test them with new analyses until we have gathered enough evidence to feel comfortable about a decision?

This effect can be achieved when adopting metagrammar engineering as a general methodology for grammar development. When alternative plausible analyses are at hand, they are stored in the metagrammar. When treating new phenomena, the metagrammar can generate alternative versions of the grammar each providing a different context for analyses of the new phenomenon. Choices between alternatives can thus be delayed until more evidence for or against a specific analysis has been gathered, or alternatives can coexist in the metagrammar indefinitely.

[10]This explanation is based on Fokkens and Bender (2013).

1.2.2 CLIMB metagrammars

In this subsection, I will provide a brief outline of the context and assumptions used for developing grammars in this work. The previous section explained that metagrammar engineering can help to make more informed decisions about which analysis to choose when alternatives are at hand. Section 1.1 has shown that different assumptions may be used as criteria as to which analysis is best. The only criterion syntacticians agree on is that the analyses must be able to lead to correct predictions concerning the grammaticality of data.

The discussion mainly focused on theoretical grammars. It should be noted that the grammars developed in this thesis can be seen as "practical grammars". Even though they are based on linguistic theory and aim at capturing linguistic behaviour correctly, they can also be used in applications. It is also possible to apply the methodology proposed in this thesis for syntactic hypothesis testing only, but the practical character of the grammars provides an additional set of measurable criteria related to their efficiency that can be used to evaluate alternative analyses.

As mentioned above, the proposed methodology has been tested through the development of CLIMB which can generate implemented HPSG grammars. In particular, CLIMB is closely connected to the multilingual grammar development toolkit, the LinGO Grammar Matrix (Bender et al., 2002, 2010) and, like the Grammar Matrix, part of DELPH-IN (the Deep Linguistic Processing with HPSG Initiative).[11] I will briefly elaborate on some of these aspects here. A more elaborate description of the background of these grammars (i.e. HPSG, DELPH-IN and the Grammar Matrix) will be given in Chapter 2.

The Grammar Matrix provides a starter kit for writing new DELPH-IN grammars. It consists of a static core and dynamic customisation system. The static core includes basic files to make the grammars interact correctly with DELPH-IN parsers and generators. It furthermore contains very general implementations of linguistic properties which are likely to be useful for most

[11]`http://www.delph-in.net`

languages. The customisation system allows users to define language specific properties in a web-based questionnaire. Based on this input, the customisation system generates a set of language specific implementations of linguistic phenomena. The customisation system provides basic analyses which can be corrected and extended manually by grammar engineers.

The technology used for grammar customisation forms the basis of CLIMB metagrammars. In CLIMB, grammar engineers no longer treat the customisation system as a black box only using the grammar it outputs as a starting point. Rather, they extend the original customisation system, so that it can generate all analyses they designed for their grammar. It should be noted that it is possible to write grammars with this methodology that do not make use of the Grammar Matrix, but all CLIMB metagrammars discussed in this thesis are Matrix-based. A more detailed explanation of CLIMB and how it relates exactly to the Grammar Matrix will be given in Chapter 3, Section 3.1.1.

Some of the restrictions posed on the grammars implemented to test metagrammar engineering come from their relation to DELPH-IN and the Grammar Matrix. This includes the use of typed feature structures (from HPSG) and the basic feature geometry (from the Grammar Matrix).

Grammars developed in DELPH-IN provide a bidirectional mapping between linguistic expressions and semantic representations. The main criterion that was followed to develop the grammars for this thesis is that they handle available data correctly. This means that they provide a defensible semantic representation for grammatical strings and fail to produce an analysis for ungrammatical expressions. Mappings between semantic representation and surface strings can also be verified experimentally by asking native speakers whether sentences with the same semantics are paraphrases. However, such experiments require input from several speakers and do not contribute to the main topic of this thesis. In this study, the MRS representations the grammars should produce were based on semantic representations for similar expressions by the English Resource Grammar (Flickinger, 2000, ERG) and the German Grammar (Müller and Kasper, 2000; Crysmann, 2005, GG), two other DELPH-IN grammars.

A survey of native speaker judgements and syntactic investigations to select the right data for such experiments was beyond the scope of this work. In order to prevent an undesirable bias in data selection, the main datasets used for evaluation were taken from other researchers. Particularly, the Cheetah development set (Cramer, 2011) was used as a guideline to develop CLIMB grammars for German. Extensions of this set were mostly based on literature on German syntax.

1.2.3 Summary

In this section, I argued that the choice between analyses partially depends on how each alternative interacts with the rest of the grammar. However, it is only possible to verify interaction with analyses that are already included in the grammar and phenomena that are still to be treated are (generally) ignored. This leads to an asymmetrical influence on the grammar from phenomena treated earlier in the grammar development process. Section 1.2.1 explained that this problem can be addressed by using a metagrammar. The metagrammar can contain alternatives and allows grammar engineers to maintain these alternatives while developing the grammar. The decision between alternatives can thus be postponed until more evidence is available. This idea has been implemented in the development of gCLIMB. Section 1.2.2 outlined the assumptions taken in this project. Namely, competence of the grammars is evaluated using test data that is developed independently of this thesis whenever possible and the target semantic representation for a given expression is based on the representations for equivalent expressions given by the ERG or GG.

1.3 Thesis Overview

The previous sections have introduced the topic of this thesis and outlined the basic assumptions followed in this work. This section presents the outline of this thesis in Section 1.3.1 and its main contributions in Section 1.3.2.

1.3.1 Thesis Structure

Chapter 2 introduces the background for the work carried out in this thesis. A brief introduction to HPSG is provided. The formalism of typed feature structures are defined and the main principles of the theory are explained through an example analysis. Section 2.2 describes specific properties of grammars developed as part of DELPH-IN. The basics behind parsing and generation algorithms used in DELPH-IN grammars are explained. Section 2.3 introduces the LinGO Grammar Matrix, the project that formed the main inspiration for CLIMB. The history and main goals of this project are presented as well as the basic architecture of the Grammar Matrix. The Grammar Matrix implementations provided the foundation of CLIMB, which can be seen as an extension of the Grammar Matrix.

Chapter 3 introduces the CLIMB approach and related tools to analyse grammars. Section 3.1 describes the origin of the idea behind CLIMB, the relation between CLIMB and the Grammar Matrix and the architecture and workflow used for the CLIMB grammars in this thesis. Section 3.2 presents declarative CLIMB, an alternative version of CLIMB that is easier to use for grammar engineers used to writing grammars declarative in TDL,[12] but which does not support the full flexibility of CLIMB. Additional processing tools for the grammars are introduced in Section 3.3. They include the spring cleaning algorithm (Fokkens et al., 2011) and a set of tools that support writing grammars using abbreviated paths. Finally, an overview of the possible applications of CLIMB (briefly mentioned above at the beginning of this chapter) is given in Section 3.4.

Chapter 4 introduces the alternative analyses that are examined in this thesis. Section 4.1 motivates the decision to implement a grammar for German. A basic overview of German word order and auxiliaries is given in

[12]Type Description Language used to define typed feature structures in DELPH-IN grammars (Krieger and Schäfer, 1994; Copestake, 2000).

Section 4.2. The standard HPSG analyses are described in Section 4.3. Section 4.4 describes the alternative analyses included in gCLIMB. It is explained why the alternative analyses for auxiliaries are (expected to be) more efficient in parsing and generation. It includes an illustration of what happens with the parse chart when a basic sentence is analysed with either of the two analyses.

Chapter 5 describes the evaluation of gCLIMB for German. The first part of the evaluation examines whether it is feasible to use CLIMB for large scale grammar development and what the impact is on grammar development. This is done by comparing the process of including phenomena from Cheetah's development set for regression testing in gCLIMB to the development of Cheetah's core grammar. The outcome of this evaluation indicates that CLIMB can be used for developing large scale grammars. The total development time for gCLIMB is less than half that of Cheetah, despite the facts that the gCLIMB work also includes support for other languages (to a certain extent) and that gCLIMB's semantic representations are more complex than Cheetah's. Cramer (2011), on the other hand, focused on improving efficiency. Because of this difference and the fact that there are generally too many other influential factors on development speed, it is not possible to conclude that CLIMB speeds up development. However, this result does make it clear that even if it were the case that using CLIMB generally slows grammar development down, this remains in an acceptable range.

The second part evaluates efficiency of the grammar indicating that the advantages of alternative analyses observed in Fokkens (2011a) is still found in larger grammars. As predicted, the impact increases as the metagrammar covers more phenomena, but this effect can be prevented by using additional constraints in the grammars. Section 5.3 describes future directions of research for gCLIMB including how to extend gCLIMB to carry out more elaborate studies of German syntax and how to incorporate a lexicon that is read off a treebank.

Chapter 6 discusses multilingual aspects of CLIMB. It first presents two studies that examine whether the German specific analyses created in this grammar can be used for Dutch, Danish and Northern Frisian. It examines whether additional analyses interact correctly with the language specific analyses added for Dutch and Danish at the beginning of the development of gCLIMB. The evaluation shows that decent coverage can be obtained in a relatively short time. It is unlikely the same could be achieved by only using the Grammar Matrix or starting from another grammar for German such as GG, but revisions of the metagrammar are necessary to make it work properly for these languages. Section 6.3 outlines the main goals of SlaviCLIMB, a version of CLIMB that can be used to create grammars for Slavic languages. The role of CLIMB is illustrated by comparing the implementation of a Slavic case hierarchy with a static core to the more flexible CLIMB implementation. Section 6.4 describes the CLIMB implementation of a second language learning module for adjective endings in German. This is followed by an evaluation of the spring cleaning algorithm, an algorithm which identifies inactive parts of grammars, on different DELPH-IN grammars. Finally, a study of revisions made to the Grammar Matrix and a study on the role of the Grammar Matrix as a language independent core in different DELPH-IN grammars are presented in Section 6.6.

In **Chapter 7**, I describe other approaches to improve grammar engineering by metagrammars, code sharing or crosslinguistic collaboration across frameworks. The formalism and ideas behind the following projects are addressed: MetaGrammar and XMG for TAG (Candito, 1998; Duchier et al., 2005), the GF Resource Library (Ranta, 2009, 2011) and the ParGram Project (Butt et al., 2002; King et al., 2005). Furthermore, three approaches for HPSG and PC-PATR grammars are discussed. They are PAWS (Black, 2004; Black and Black, 2009), Sygal and Wintner's (2011) modular approach to developing unification based grammar and Müller's (2013) CoreGram.[13] CLIMB is compared to each of these approaches. The main conclusion of this comparison is that even though some approaches provide the means to maintain alternative analyses in parallel (notably Candito (1998), Ranta (2011) and to a lesser extent the HPSG CoreGram), this has to my knowledge not been tried out

[13]`http://hpsg.fu-berlin.de/Projects/core.html`, accessed 30 June 2012.

on a large scale by any previous approach. CLIMB and Sygal and Wintner's modular approach reveal several similarities. Sygal and Wintner's approach mainly focuses on increasing modularity and has, to my knowledge, not been tried out for grammars of similar size to gCLIMB.

Overall conclusions and future work are presented in **Chapter 8**.

1.3.2 Contributions

The main contributions of this thesis are the following:

1. This work is, to my knowledge, the first to observe that inconclusive evidence and interaction form a major hindrance in coming to optimal solutions in syntactic models and computational grammars.
2. This thesis proposes to address this problem by adopting metagrammar engineering as a general method for grammar development. The methodology differs from other similar approaches in that all analyses are stored in syntactic libraries, so that the complete latest versions of the grammar can be generated at any time.
3. Procedural and declarative CLIMB have been developed to implement this idea. The methodology has some additional advantages including:
   - increased modularity
   - phenomenon based organisation of the grammar
   - longterm flexible multilingual grammar development
   - different versions for different dialects
   - different versions for different applications (e.g. grammar checking versus high recall on text)
   - possible speed-up for grammar development
4. A metagrammar for German has been implemented examining alternative analyses for auxiliaries and word order. This study provided the following contributions:

- Comparison between the development of gCLIMB and the Cheetah core grammar indicates that CLIMB does not seem to have a negative influence on development speed.
- Observations about the methodology while the grammars increased in size indicated that the method can easily be applied to large scale grammars.
- Experiments that would be difficult to carry out without CLIMB evaluating natural language generation involving alternative analyses and different linguistic properties are carried out.
- A setup containing implemented analyses for a range of phenomena for German is provided: This setup can be used for further research comparing analyses from other German grammars in HPSG.

5. Multilingual aspects of metagrammar engineering with DELPH-IN grammars have been explored:

    - An evaluation of Dutch, Danish and Northern Frisian grammars created with gCLIMB: This evaluation shows that CLIMB indeed facilitates adaptations of analyses to cover variations in related languages making it possible to capture a range of phenomena in short time. However, variations in linguistic properties should be taken into account throughout grammar development in order to produce high quality grammars.
    - The outline and vision on SlaviCLIMB and an implementation of its basic architecture
    - A simple method to build grammars for second language learners using the metagrammar
    - An evaluation of the Grammar Matrix describing the revisions that have been made while implementing gCLIMB as well as an evaluation of how the Matrix core is used in other grammars

The bulk of the work carried out as part of this thesis was the conceptual development of CLIMB and the technical development of tools that implement it. In particular, I developed gCLIMB to evaluate the methodology showing that CLIMB can be applied for developing large scale grammars. Building on this work, I carried out several experiments that explore the potential of the methodogy. They include comparative experiments and studies on multilingual aspects of using CLIMB. These studies show the possibilities provided by the methodology. It should be noted, however, that these experiments and studies are initial explorations. This thesis introduces the methodology, provides the tools and indicates what may be done with them. The possibilities include research to support theoretical syntax, an approach to explore the impact of analyses on efficiency in implemented grammars over time and a flexible way of multilingual grammar development that has, to my knowledge, not been proposed for HPSG-based implemented grammars before. It is my hope that the contributions of this thesis will inspire and enable other researchers to further explore properties of their grammars, compare analyses and investigate new approaches for multilingual grammar development and that it provides them with the means to carry out such research.

Chapter 2

Background

This chapter provides the theoretical background of CLIMB, as well as the context of the approach. The grammars produced by CLIMB follow the principles of Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994, HPSG). Section 2.1 provides an introduction to this theory based on Pollard and Sag (1994) and Sag et al. (2003). This is followed by an overview of the grammars developed within the DELPH-IN consortium as well as a description of how parsing and generation with these grammars works. In this section, I also point out some aspects in which analyses in DELPH-IN grammars differ from standard theoretical HPSG. Finally, Section 2.3 describes the LinGO Grammar Matrix, the project that is most closely related to the work presented in this thesis.

2.1 Head-Driven Phrase Structure Grammar

Pollard and Sag (1994) present linguistic theory as a "mathematical theory about an empirical domain" (Pollard and Sag, 1994, 6).[1] The object of study is language, which they understand to be the collective knowledge that is shared among members of a linguistic community. This knowledge should

[1]This statement can apply to both scientific theories and philosophical theories and does thus not address the questions raised in the previous chapter.

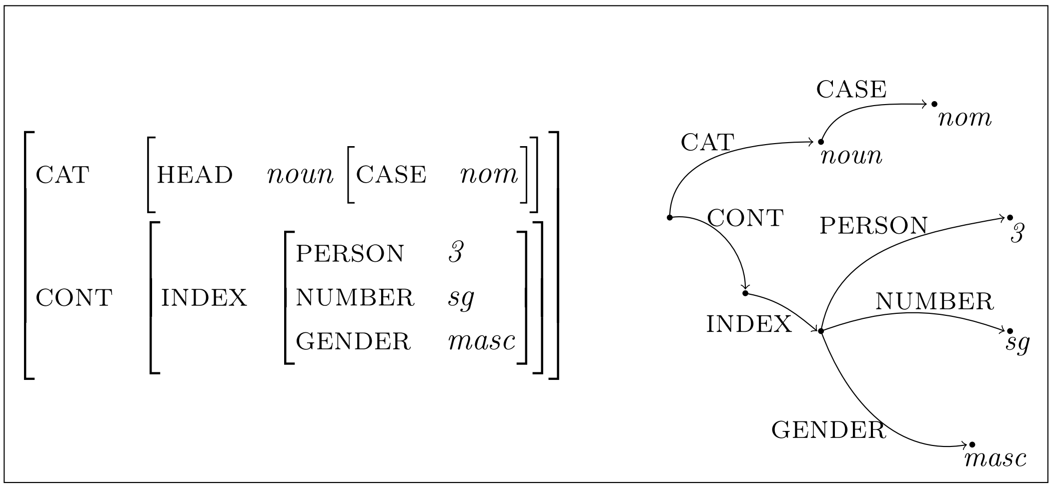


Figure 2.1: A feature structure presented as AVM (left) and DAG (right)

be sought in linguistic types rather than linguistic tokens. In other words, linguistic theory should model generalisations about linguistic utterances.

In line with seeing linguistic theory as a mathematical model of language, Pollard and Sag (1994) insist on the importance of formalising linguistic theory. In HPSG, *sorted* or *typed feature structures* (Moshier, 1988; Pollard and Moshier, 1990) are used to model language. I will use the term *typed feature structures* in this thesis. In the following subsections, I will describe typed feature structures as well as the main ideas on which HPSG theory is based.

2.1.1 Typed Feature Structures in HPSG

Feature structures

Feature structures are data structures that can be used to represent objects. They are sets of attribute-value pairs. Feature attributes can take other feature structures as values. They are therefore suitable to model objects with complex structures. Several linguistic formalisms make use of feature structures to model linguistic objects. In addition to HPSG, they are found in Functional Unification Grammar (Kay, 1979, 1984, FUG), Parse and Translate II (Shieber et al., 1983; Shieber, 1984, PATR-II), Lexical Functional Grammars (Kaplan and Bresnan, 1982, LFG) and other formalisms.

Figure 2.1 depicts two representations of a feature structure partially describing a noun or noun phrase. The figure on the left uses an attribute-value matrix (AVM) to describe the feature structure, which is the most common representation in linguistic work. On the right hand side, the same feature structure is represented as a directed acyclic graph (DAG). The feature structure partially describes the English pronoun *he* (or any other grammatically masculine singular noun bearing nominative case). It specifies that the head of the element belongs to the syntactic category *noun* and bears the nominative case. The indicated semantic values state that the referent of the object is third person singular and masculine. I will elaborate on the feature geometry (the location of specific features in the structure) in Sections 2.1.2 and 2.1.3.

Types

Feature structures in HPSG are *typed* or *sorted* (Pollard and Sag, 1994). This means that they are associated with an ontology of linguistic objects or types. Each feature structure corresponds to a type. In a graph representation, types will be indicated by labels at the nodes of the graph.

Types are organised in a finite partially ordered set called the type hierarchy. Types are ordered through subsumption. Subsumption ($\sqsubseteq$) is a reflexive, transitive and anti-symmetric relation (Carpenter, 1992).

Supertypes are more general types that subsume more specific types, their subtypes. Subsumption is a relation of inheritance: subtypes inherit all properties from their supertypes. Multiple inheritance is permitted, i.e. a type can have more than one supertype. Figure 2.2 provides an example of a type hierarchy for English agreement and case.

The type hierarchy plays an important role in HPSG's central operation: unification. Unification is used to combine words and phrases into larger ex-

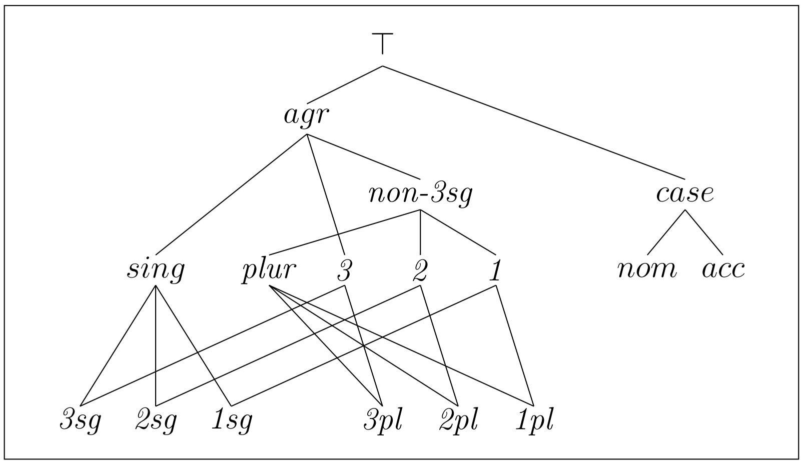


Figure 2.2: A type hierarchy for English person, number and case

pressions, as will be explained in more detail in Section 2.1.3. Grammatical constraints can also be modelled using unification: if two types bear conflicting feature values, they cannot unify. In order to unify, types must have a common subtype. Moreover, if the types that are unified are complex types, i.e. they have properties defined through feature-value pairs, the feature values of the individual types must unify as well.

Figure 2.2 shows how a type hierarchy may be used to model person number agreement in English. From the six variations between person and number, all except the type for third person singular can unify with the type *non-3sg*. Morphological marking on English verbs in present tense can restrict the agreement value of the verb's subject to *3sg* or *non-3sg*. If a verb assigns the feature value pair [AGR *non-3sg*] to its subject, a NP bearing the pair [AGR *1sg*] can become subject of this verb, whereas unification will fail when the NP is [AGR *3sg*].

In addition to their role in unification, types are used in HPSG to define principles of well-formedness of feature structures. Pollard and Sag (1994) state that feature structures must be *well-typed*, *totally well-typed* and *type-resolved* (Carpenter, 1992). I will briefly explain each of these conditions below.

Well-typedness ensures that feature structures only describe properties that are appropriate for the type they model. Feature structures may only contain attributes that are specifically defined to be appropriate for their type. Likewise, the values that these attributes can take are restricted to specific types. The AVM in Figure 2.1, for instance, contains the attribute CASE. The grammar will contain a definition that states that CASE is an appropriate attribute of the type *noun* and that values of this attribute should be of type *case*. The type hierarchy in Figure 2.2 would then allow the values *case*, *nom* and *acc* for this attribute.

Feature structures must be *totally well-typed*, which means that they must include all attributes declared appropriate for their type. In other words, where well-typedness ensures that only appropriate attribute-value pairs occur, total well-typedness ensures that all appropriate attributes occur. In practice, irrelevant attributes are often left out in representations of structures, but they are considered to be present nevertheless. This can lead to inefficiencies in implemented grammars. Flickinger (2000) provides a solution to this problem, which will be explained in Section 2.2.1.

Finally, the condition that a feature structure should be *type-resolved* ensures that no underspecified values remain in the feature structure. A feature structure is type resolved when all types it contains are maximal types, the most specific types in the hierarchy. In other words, types in a type-resolved feature structure may not have any subtypes: they are the leaves of the type hierarchy. The ambiguity of natural language is well-known and it is therefore common that even after all linguistic information is taken into account, there may be underspecified information. The idea behind this is that, in a given context, there will be only one appropriate interpretation and all information about the object is known. There are different views on this condition in the HPSG community. This approach is followed by Sag et al. (2003), but it is typically not maintained in implemented grammars, nor is it implemented in the parsing and generation tools described below in Section 2.2.2.

$$\begin{bmatrix} \textsc{phon} & \textit{string} \\ \textsc{synsem} & \begin{bmatrix} \textsc{local} & \begin{bmatrix} \textsc{cat} & \begin{bmatrix} \textsc{head} & \textit{head} \\ \textsc{val} & \begin{bmatrix} \textsc{subj} & \textit{list} \\ \textsc{comps} & \textit{list} \\ \textsc{spr} & \textit{list} \\ \textsc{spec} & \textit{list} \end{bmatrix} \end{bmatrix} \\ \textsc{cont} & \begin{bmatrix} \textsc{hook} & \textit{hook} \\ \textsc{rels} & \textit{diff-list} \\ \textsc{hcons} & \textit{diff-list} \end{bmatrix} \end{bmatrix} \\ \textsc{non-local} & \begin{bmatrix} \textsc{slash} & \textit{0-1-dlist} \\ \textsc{que} & \textit{0-1-dlist} \\ \textsc{rel} & \textit{0-1-dlist} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2.3: Basic feature-structure of the type *sign*

2.1.2 Signs

The main linguistic objects in HPSG are signs. They correspond more or less to Saussurean signs, which are arbitrary mappings between form and meaning (Saussure, 1916). Signs include words, phrases, sentences and even multi-sentence discourses (Pollard and Sag, 1994). HPSG is a lexicalist theory. This means that the bulk of syntactic and semantic properties are defined in the lexicon. As such, lexical items contain information about phonetics, syntax, semantics and discourse, just like words and phrases.

The exact feature geometry of *sign* has been revised at several occasions[2] and may differ from one researcher to another. Figure 2.3 presents the basic (incomplete) structure of a sign as typically found in DELPH-IN grammars. The next sections will explain how individual signs can be combined to form

[2]Most notably, Pollard and Sag (1994) merge the attributes SYNTAX and SEMANTICS from Pollard and Sag (1987) into SYNSEM. In constructional versions of HPSG (Ginzburg and Sag, 2000; Sag, 1997), feature structures are generally flatter than in standard HPSG.

larger expressions. Semantic properties (found under CONT) will be explained in more detail in Section 2.1.5.

2.1.3 Immediate Dominance Schemata

Signs are combined into larger units by immediate dominance schemata. In HPSG theory, schemata are seen as a small universal set of constraints that determine the immediate constituency of phrases (Pollard and Sag, 1994, 38). Likewise, the related Immediate Dominance Principle is considered a universal principle:

> THE ID PRINCIPLE (Pollard and Sag, 1994, 38,399)
> Every headed phrase must satisfy exactly one of the ID schemata.

Pollard and Sag (1994) define six schemata in total. They are listed below, followed by a brief description of their function. The reason that only six schemata are proposed is closely linked to the lexicalist character of HPSG. Lexical items contain detailed information about the arguments they take and/or the elements they may modify. Restrictions on syntactic categories or cases are therefore handled by the syntactic head or modifier of a phrase, permitting generally defined schemata to create different kinds of phrases.

> SCHEMA 1 (HEAD-SUBJECT/HEAD-SPECIFIER SCHEMA):
> Schema 1 combines a phrase with its subject or a noun with its specifier.
>
> SCHEMA 2 (HEAD-COMPLEMENT SCHEMA):
> Schema 2 combines a word or phrase with its complements.
>
> SCHEMA 3 (HEAD-SUBJECT-COMPLEMENT SCHEMA):
> Schema 3 combines a head with both its subject and a complement in a ternary structure. This schema can be used when the subject stands between a verb and its complement, interrupting the verb phrase.
>
> SCHEMA 4 (HEAD-MARKER SCHEMA):
> Schema 4 combines a syntactic marker with a phrase. The English complementiser *that* is an example of a syntactic marker.

SCHEMA 5 (HEAD-ADJUNCT SCHEMA):
Schema 5 combines phrases with modifiers.

SCHEMA 6 (HEAD-FILLER SCHEMA):
Schema 6 combines a phrase with an extracted element. This schema is used to capture long distance dependencies.

Schemata 1, 2 and 5 will be illustrated in an example derivation in Section 2.1.4. Schemata 3 and 4 are not used in the grammars discussed in this work. Examples and motivations for their use can be found in Pollard and Sag (1994). Schema 6 is used in the standard HPSG analysis for German word order and will be illustrated in Chapter 4.

The ability to apply schemata, as well as the result of their application is influenced by a set of principles. I will present the most significant ones, namely the *Head Feature Principle*, the *Subcategorisation or Valence Principle* and the *Semantics Principles* in the next section.

2.1.4 An example analysis in HPSG

This section will present an HPSG analysis for the sentence *The little girl likes strawberries.* The purpose of this example analysis is to demonstrate how schemata work, as well as to introduce some of the main principles in HPSG.

Figure 2.4 presents the structure for *likes strawberries.* A lexical rule applies to the lexical entry of *like* assigning present tense and restricting its subject to entities in third person singular. The noun *strawberries* is the daughter of a *bare-np-phrase* which creates noun phrases from nouns when they do not combine with a determiner.[3] Note that this derivation is somewhat simplified. In standard HPSG and most grammars, the grammar would include a stem for *strawberry* and a lexical rule to create the plural form. The word *likes*

[3]Bare-np-phrases are typically used in implemented grammars and not (necessarily) in theoretical HPSG.

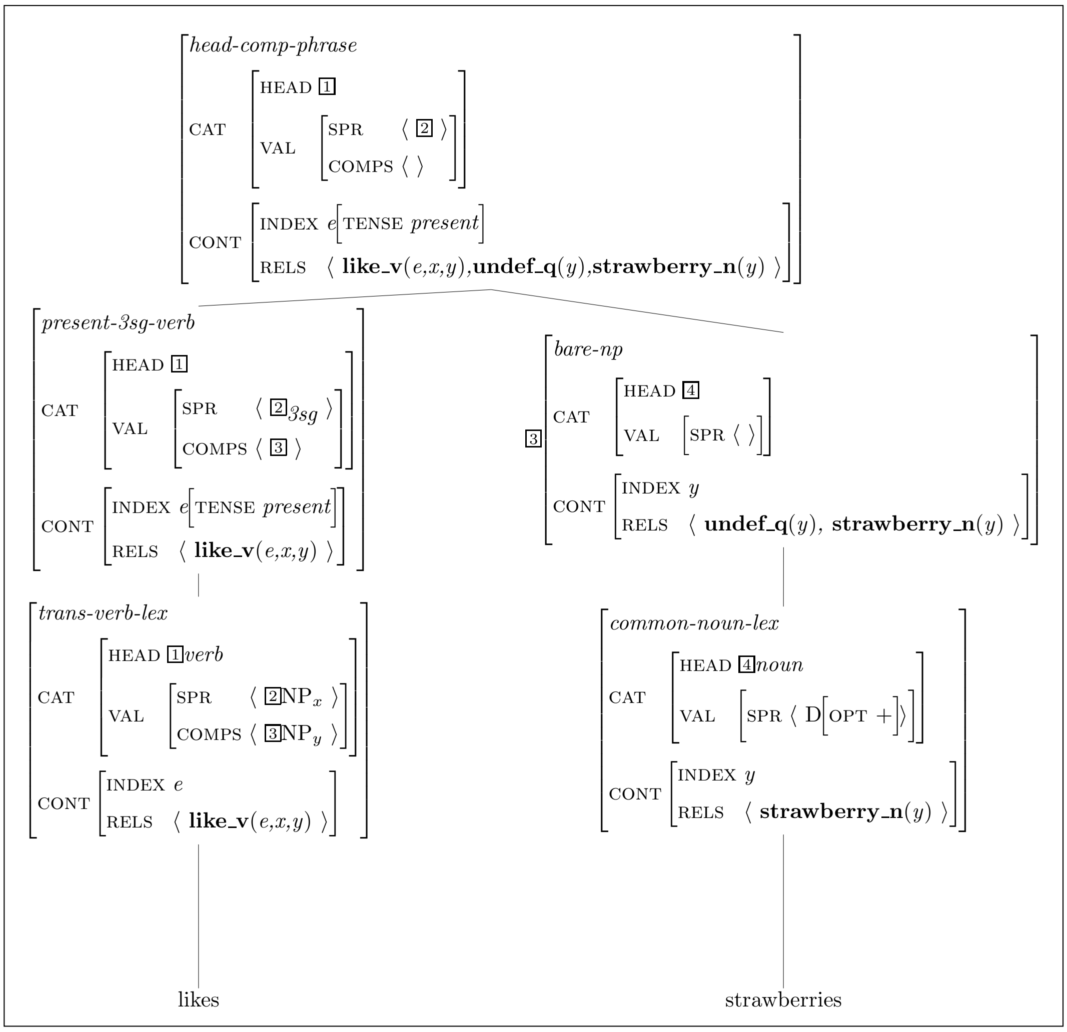


Figure 2.4: Abbreviated structure for *likes strawberries*

and phrase *strawberries* are combined with SCHEMA 2, the head-complement schema.

The head-complement schema specifies that the complement list of the resulting phrase is empty, and the elements on the complements list of the head daughter corresponds to the synsems of the non-head daughters. In Figure 2.4, the complement list of *like* has one element ($\boxed{3}$), which corresponds to the synsem of *strawberries.* The example shows that the head of *likes* ($\boxed{1}$) also is the head of the entire phrase. The noun phrase and noun *strawberries*

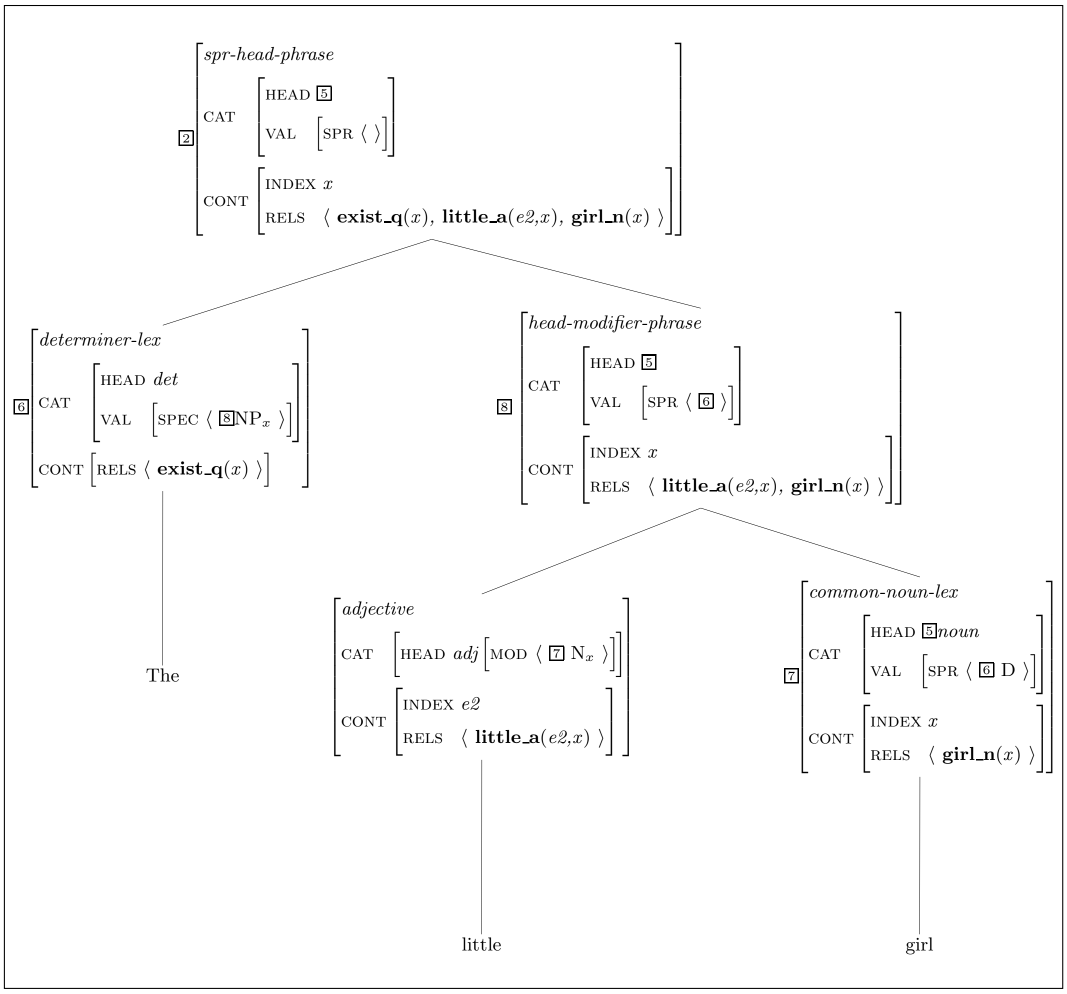


Figure 2.5: Abbreviated structure for *the little girl*

share their head value as well ($\boxed{4}$). This forms an illustration of the *Head Feature Principle* defined below (based on (Pollard and Sag, 1994, 34, 399) and (Sag et al., 2003, 55)):

> HEAD FEATURE PRINCIPLE:
> The HEAD value of any headed phrase is token identical to the HEAD value of head daughter

Figure 2.5 shows the structure of *the little girl*. The *head-modifier-phrase* is built by applying SCHEMA 5. SCHEMA 1 combines the noun with its specifier.

Note that the valence requirements of the phrase do not change when the adjective and the noun are combined. This is a consequence of the valence principle (based on Sag et al. (2003), p. 143):

> Valence Principle:
> The VAL(ence) value of any headed phrase is token identical to the VAL value of head daughter, unless the applied schema specifies otherwise.

The values specified under VAL indicate the arguments an element subcategorises for. A schema that does not combine a head with an argument (or cancel an argument, like the bare-np-phrase) should therefore not change its values. Schemata that combine phrases with subjects, specifiers or complements remove the found object from the related subcategorisation list. The valence principle in combination with correctly defined schemata ensure that subcategorisation requirements are respected.

Figure 2.6 combines the verbal phrase with its subject, using SCHEMA 1. This example follows Pollard and Sag (1994), who treat subjects as specifiers. This is generally not the case in DELPH-IN grammars, where subjects have their own list under VAL (cf. Figure 2.3, p. 31). Again, the only element on the SPR list of the head daughter is identified with the synsem of the non-head daughter. The result is a phrase that no longer requires a specifier.

Now that the main syntactic properties (sharing head values and subcategorisation) have been explained, let us look at the semantics value of the sentence. They are the result of two semantic principles, namely the *Semantic Inheritance Principle* and the *Semantic Compositionality Principle*. The definition of the former is as follows (based on (Sag et al., 2003, 143)):

> Semantic Inheritance Principle:
> The INDEX of a phrase is token identical to the INDEX of its head daughter.

The index is used to build up semantics of predicates and arguments. The semantic inheritance principle ensures that the newly created phrase has the

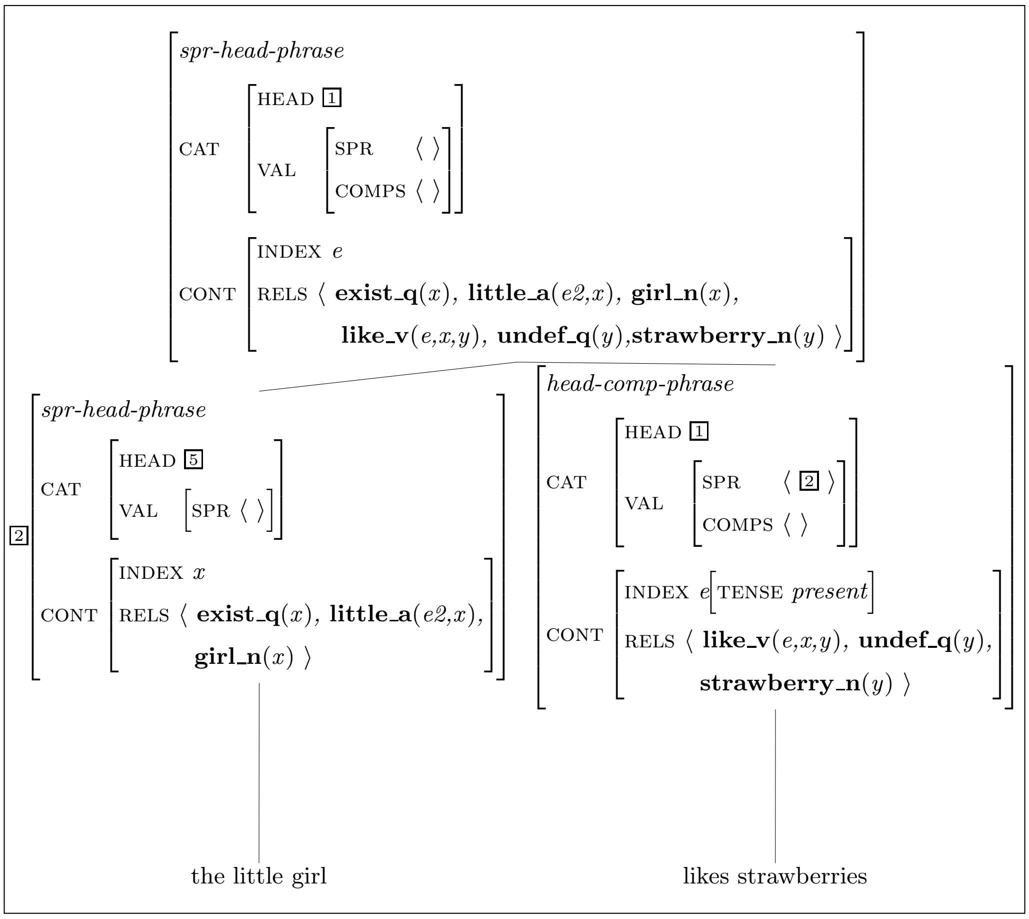

spr-head-phrase
CAT HEAD 1
VAL SPR ⟨ ⟩ COMPS ⟨ ⟩
CONT INDEX e
RELS ⟨ exist_q(x), little_a(e2,x), girl_n(x), like_v(e,x,y), undef_q(y),strawberry_n(y) ⟩
2
spr-head-phrase
CAT HEAD 5
VAL SPR ⟨ ⟩
CONT INDEX x
RELS ⟨ exist_q(x), little_a(e2,x), girl_n(x) ⟩
head-comp-phrase
CAT HEAD 1
VAL SPR ⟨ 2 ⟩ COMPS ⟨ ⟩
CONT INDEX e TENSE present
RELS ⟨ like_v(e,x,y), undef_q(y), strawberry_n(y) ⟩
the little girl
likes strawberries

Figure 2.6: Abbreviated structure for *the little girl likes strawberries*

correct index for further combinations. The example sentence *the little girl likes strawberries* reveals how this works for arguments as well as modifiers. The lexical entry *like* specifies that its subject and object correspond to its first and second argument, respectively (e is its identifying argument, also known as "distinguished variable" or ARG0, as will be explained in the next section). This is indicated by the subscribed x and y in NP_x and NP_y, which stand for the NP's indices. Similarly, N_x and x on the MOD and, respectively, RELS lists of *little* indicate that the adjective's first argument is the referent of the modified noun.

The Semantic Compositionality Principle captures the idea that the semantics of a phrase is constructed from its parts (Frege, 1892). It is defined

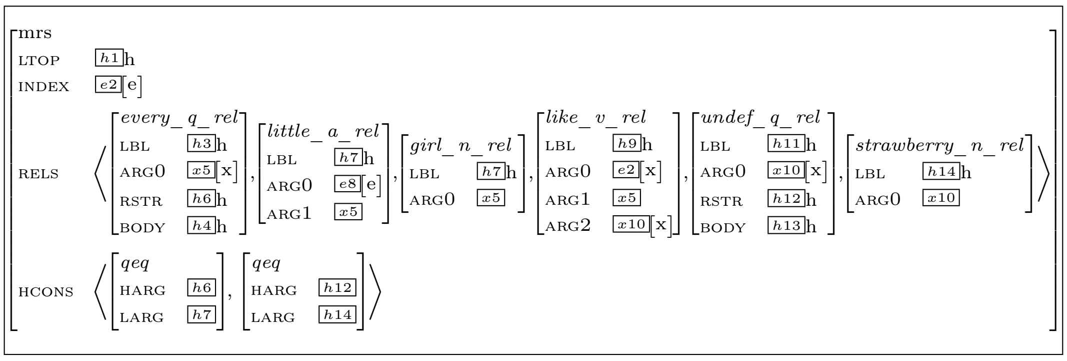


Figure 2.7: MRS Representation of *Every little girl likes strawberries*

as follows (based on (Sag et al., 2003, 144)):

> THE SEMANTIC COMPOSITIONALITY PRINCIPLE: The value of RELS of any well-formed phrase is the sum of the RELS values of its daughters.

The example analysis in this section has been used with the purpose to provide a basic indication of how HPSG works and to introduce its basic principles. The next section will describe Minimal Recursion Semantics (Copestake et al., 2005), which is used in most DELPH-IN grammars.

2.1.5 Minimal Recursion Semantics

Minimal Recursion Semantics (MRS) is a meta-language that can be used to describe semantic structures. The idea behind MRS is to provide a framework for computational semantics suitable for parsing and generation. This subsection will provide a simple example and brief explanation of MRS. For a thorough introduction, see Copestake et al. (2005). Figure 2.7 presents an MRS representation for a similar example sentence as in the previous section, *every little girl likes strawberries.*

The basic entities in MRS representations are *elementary predications* (EPs). An EP is a single relation with its associated arguments (Copestake et al., 2005, 2). The EPs in the semantics of *every little girl likes strawberries* are found under the attribute RELS in Figure 2.7.

MRS aims to provide a flat representation that still captures required information about scope. The RELS value in Figure 2.6 in the previous section presents a simplified form of the basic argument structure. The information in this simplified structure is conveyed by the values of ARGN in the MRS representation above.

In order to represent scopal information, labels are introduced with so-called *handles* as their value. Handles also appear as the value of scopal arguments, in this case they are called *holes.* Constraints on possible scopal interpretations of the expression are introduced under HCONS. The *qeq* relation indicates that either the value of LARG l fills the hole h in HARG, or l is indirectly linked to h via one or more 'floating quantifiers'. In this case, l is either the body of an argument filling h, or an argument that is the body of an argument directly or indirectly filling h (Copestake et al., 2005, 10). Two scopal relations are defined in the example above. The first element on HCONS indicates *every* has scope over *little* and *girl.* Likewise, *udef*, the quantifier introduced by the bare noun phrase rule has scope over *strawberries.*

2.1.6 Summary

This section provided an introduction into HPSG. Formal properties of typed feature structures were defined. A basic description of *sign* has been given, along with a demonstration of how signs are combined into larger expressions through unification. Finally, the basics of MRS were explained. The next section provides background information on HPSG grammars implemented as part of the DELPH-IN initiative.

2.2 DELPH-IN

The DELPH-IN (Deep Linguistic Processing with HPSG Initiative)[4] research consortium aims to create open source NLP applications that use linguistically motivated precision grammars. Research at DELPH-IN member sites ranges from grammar development and tools supporting grammar development to applications using them.

Several applications have been supported by DELPH-IN grammars, such as ontology construction (Nichols et al., 2006), machine translation (Oepen et al., 2007; Nichols et al., 2010; Bond et al., 2011; Wang et al., 2012), structured knowledge based question answering (Frank et al., 2007), extracting knowledge from scientific text (Rupp et al., 2007), textual entailment (Bergmair, 2008) and grammar checking (Bender et al., 2004; Crysmann et al., 2008; Suppes et al., 2012). A full account of NLP applications and other related research using DELPH-IN grammars is beyond the scope of this work. The only application that will be discussed in detail will be work on grammar checking in Chapter 6, as part of an introduction on using Germanic CLIMB for this purpose.

The properties of DELPH-IN grammars, as well as the tools available to support their development, on the other hand, directly influence the design choices made in the development of gCLIMB. In order to understand algorithms developed as part of CLIMB, such as the algorithm for *spring cleaning* (cf. Chapter 3, Section 3.3.1), knowledge of some of the formal properties of DELPH-IN grammars is required. The analyses proposed in Chapter 4 deviate from standard HPSG in several ways, some of which are typical deviations found in DELPH-IN grammars. In order to understand how different analyses interact with efficiency of the grammar, a basic understanding of the parsing and generation algorithms used with these grammars is needed. This section addresses these two aspects. First, DELPH-IN grammars are described. Next, the basics behind parsing and generation grammars of the DELPH-IN parsers and generators are explained.

[4]`http://www.delph-in.net`

2.2.1 DELPH-IN grammars

Name	Language	Reference	Maintainer
		Resource Grammars	
English Resource Grammar (ERG)	English	Flickinger (2000)	Dan Flickinger
Jacy Japanese Grammar (Jacy)	Japanese	Siegel and Bender (2002)	Francis Bond
GG	German	Müller and Kasper (2000), Crysmann (2005)	–
SRG	Spanish	Marimon (2010)	Montserrat Marimon
LXGram	Portuguese	Branco and Costa (2010)	Antonio Branco
KRG	Korean	Kim and Yang (2003), Song et al. (2010)	Jong Bok Kim
MCRG	Modern Greek	Kordoni and Neu (2005)	Valia Kordoni
NorSource	Norwegian	Hellan and Haugereid (2003)	Lars Hellan
gCLIMB German	German	Fokkens (2011a)	Antske Fokkens
		Treebank-trained Grammars	
Cheetah	German	Cramer and Zhang (2009)	Bart Cramer
		Medium-sized linguistic Grammars	
La Grenouille	French	Tseng (2003)	Jesse Tseng
MCG	Mandarin Chinese	Zhang et al. (2011)	Yi Zhang
BURGER	Bulgarian	Osenova (2010)	Petya Osenova
wmb (wmb)	Wambaya	Bender (2008a)	Emily Bender
HaG	Hausa	Crysmann (2012)	Berthold Crysmann
RRG	Russian	Avgustinova and Zhang (2010)	Tania Avgustinova
gCLIMB Dutch	Dutch	Fokkens (2011a)	Antske Fokkens
ManGO	Mandarin Chinese	–	Justin Chunlei Yang
		Experimental Grammars	
Min Nan Grammar	Min Nan (Taiwanese)	–	Michael Goodman
–	Turkish	Fokkens et al. (2009)	Antske Fokkens
–	Georgian	Borisova (2010)	Irina Borisova
–	Thai	–	Glenn Slayden

Table 2.1: DELPH-IN grammars as of July 20 2012

Grammars for several languages that may be used with DELPH-IN tools have been developed over the last two decades. Table 2.1 provides an overview of these grammars according to the DELPH-IN grammar catalogue,[5] which lists most of the DELPH-IN grammars.

Formal properties of DELPH-IN grammars

DELPH-IN grammars have five major components (Copestake, 2002, 107):

[5]`moin.delph-in.net/GrammarCatalogue`, accessed on 23 July 2012.

- types and constraints
- lexical entries
- grammar rules
- lexical and morphological rules
- root conditions

Types and constraints form a type hierarchy as described in Section 2.1.[6] Recall that this means a hierarchy of subsumption relations, where types can be associated with complex feature structures. The root conditions determine whether an utterance can be considered as a "completed" expression. A grammar can accept several roots at the same time (full sentences, noun phrases, only complete phrases, lexical items, etc.). The other three components *lexical entries*, *grammar rules* and *lexical and morphological rules* define instances. Parsing and generation algorithms operate on instances and types define these instances. In a grammar that closely follows theoretical HPSG, the grammar rule instances would correspond to HPSG schemata.

DELPH-IN grammars are all written in the DELPH-IN joint reference formalism (Copestake, 2000), based on Type Description Language as defined by Krieger and Schäfer (1994), and known as TDL. Figure 2.8 shows an example TDL type definition taken from one of the German grammars developed for this thesis. The type identifier (or name of the type, in this case `noun-lex`) stands left of the symbol `:=`. The symbol `:=` should always be followed by at least one supertype. Two supertypes are specified for `noun-lex`. Square brackets provide an environment to define further constraints on the type. Features and their values are separated from each other by white spaces. Dots can be used to define paths of features. `#spr` labels a reentrancy between the values of two features, i.e. it indicates that the values of these features should be unified.[7] Additional properties can be added to types by defining so-called

[6]The following text and its related examples are partially taken from Fokkens et al. (2011).

[7]Note that < > is an abbreviatory notation for lists, which are treated as feature structures containing the features FIRST and REST.

```
noun-lex := general-noun-lex & non-reflexive-noun-lex &
  [ SYNSEM.LOCAL.CAT [ HEAD.MOD <>,
                       VAL [ SPR < #spr & [ LOCAL.CAT.HEAD det ] > ]],
    ARG-ST < #spr > ] .
```

Figure 2.8: Sample type definition from German grammar

addenda. In an addendum, the type identifier is followed by the symbol `:+`. The structure of definitions defined using `:+` is similar to the structure when `:=` is used. The properties following the addendum symbol will be added to the original type definition. These properties can be additional supertypes or feature-value pairs. Addenda allow the grammar engineer to define the properties of a type over different files. This is of particular interest for multilingual approaches, where one may want to add language specific properties to a crosslinguistic type defined in a common core.

The type hierarchy must be a bounded complete partial order (BCPO) in order to be used for parsing or generation. In order to be a BCPO, the hierarchy must contain one unique most general subtype (the greatest lower bound, or glb) for any two types that have subtypes in common. Grammar engineers are not required to make sure that the type hierarchy has this property. Compilation algorithms of the parse and generation tools create glb types where necessary using an algorithm described in Penn (2000). In other words, if any two types share more than one subtype, and none of them is more general than all the others, the grammar compiler creates an additional type. This is shown in Figure 2.9.

Deviations from HPSG theory

Even though the foundation of DELPH-IN grammars lies in HPSG, several differences between theoretical HPSG and common practice in DELPH-IN grammars can be observed. Such differences may occur in order to maintain computational efficiency in the grammars. They mainly concern the treatment of word order and avoiding the use of unnecessary features.

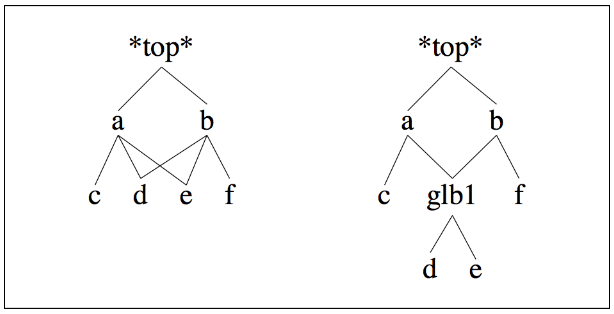


Figure 2.9: Sample hierarchy as written by grammar engineer (left) and augmented with glb type by compiler (right)

Copestake (2002) elaborates on a difference in the notion of constraints in theoretical HPSG and their treatment in implemented grammars. Recall from Section 2.1.1 that well-formed feature structures must be (*totally*) *well-typed* and *type-resolved*. Sag and Wasow (1999) distinguish typed feature structure *descriptions* from typed feature structures: descriptions need not adhere to the well-formedness conditions mentioned above.

The typed feature structures in implemented grammars are, in most cases, typed feature structure descriptions. The DELPH-IN approach uses (totally) well-typed feature structures, but does not require feature structures to be fully resolved. A typed feature structure description defines a set of typed feature structures that correspond to its description. When maximally specified, this set contains exactly one element. The unification of two descriptions is the intersection of the sets they define. If a description defines the empty set, it is considered ill-formed. This distinction between typed feature structures themselves and their descriptions is not made in implemented grammars. Operations are defined on partial orders instead of sets of objects (Copestake, 2002, 152).

Copestake (2002) explains that this difference does not seem to be significant in itself, because grammar engineers typically only use typed feature structures descriptions in their grammars. It is, however, related to a difference that does have some impact on grammar writing, namely, the way constraints are interpreted. In theoretical HPSG, features in a typed feature structure

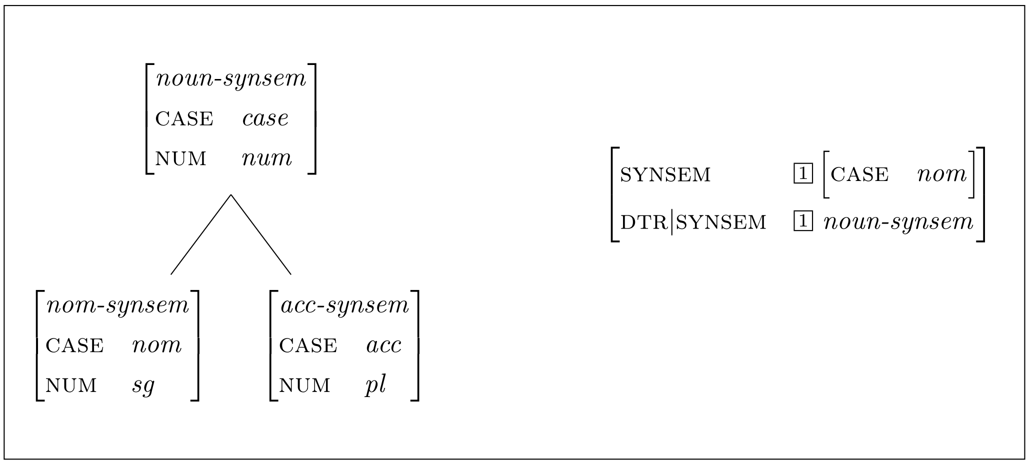


Figure 2.10: Partial type hierarchy (left) and lexical rule (right)

description are not only constrained by locally defined constraints, but also by the subtypes of the feature structure. Each feature structure description must correspond to at least one leaf type of the type hierarchy.

The constraints imposed by possible fully specified candidates influence the values of the typed feature structure, as well as it well-formedness. If a type is not a maximal type (or leaf of the type hierarchy), it is only valid if it can be specialised into one of its subtypes.

Consider the (partial) type hierarchy and basic lexical rules in Figure 2.10. Suppose the two subtypes in the hierarchy are the only subtypes of *noun-synsem.* Under the formalisation followed in theoretical HPSG, the result of applying the lexical rule represented on the right will carry the constraint [SYNSEM *nom-synsem*], automatically receiving the additional constraint [NUM *sg*]. After all, as soon as the case is constraint to the value *nom*, this is the only *noun-synsem* it can be resolved to. The value of *synsem* and its additional constraint follow from *general constraint resolution.* Constraints in DELPH-IN grammars are not resolved in this manner. In the example above, the value of SYNSEM remains *noun-synsem* with an underspecified value for the feature NUM after the rule applies.

Both DELPH-IN grammars and (most versions of) theoretical HPSG adopt a closed-world assumption. The example above illustrates that this assumption is applied uniformly in theoretical HPSG, whereas the formalisation used in DELPH-IN grammars is more flexible.

The difference may lead to underspecification if the grammar engineer does not take care when integrating analyses from theoretical work. The upside is that general constraint resolution is hardly used in HPSG (Copestake, 2002, 154). Restrictions that follow from general constraint resolution are less transparent, which may be the reason why syntacticians do not make much use of them in theoretical work.

Features

Feature structures in HPSG analyses tend to get quite large. Because all features of types to be unified are copied during unification, the size of feature structures influences both time and space in parsing and generation. It is therefore natural that grammar engineers should seek methods to reduce the size of feature structures. The aim of keeping feature structures small can also be found in theoretical HPSG. As mentioned in Chapter 1, Section 1.1.2, elegance and simplicity are generally strived for in syntactic theories and HPSG is no exception. However, there can be a difference between a theoretical justification to introduce a new feature and a computational justification.

Considerations of the feature space can inform analysis choice in grammar engineering. They can also inform the specific design of type hierarchies because the size of feature structures impacts the efficiency of processing. Flickinger (2000) proposes a specific strategy for minimising the size of feature structures while maintaining the same (analytical) feature space. Flickinger's (2000) proposal addresses inefficiencies due to the condition of total well-typedness. Because of this condition, all features assigned to a type must be verified anytime the type occurs in a structure, even if all of their values are underspecified. Flickinger (2000) proposes to introduce so-called *mintypes*. Mintypes are supertypes of basic HPSG types, where the bulk of the original type's features are omitted. Where possible, the mintype is assigned as value rather than the more elaborate type. Because of the way constraints are treated in DELPH-IN grammars, features on their subtypes will only be introduced in the structure when they are specifically defined. Features that are not relevant at a given point of a derivation are not present in the structure and need not be copied, saving both time and space in parsing (Flickinger, 2000).

Schemata and word order

The treatment of word order is probably where theoretical HPSG and DELPH-IN grammars differ most. As explained above, theoretical HPSG proposes a small set of universally valid schemata. The language dependent word order constraints are imposed by linear precedence constraints defined externally from the type hierarchy. In early HPSG, these constraints influence the order of daughters directly by ordering constituents. Later, domains were introduced which provide a method to scramble words belonging to different constituents (Reape, 1993, 1994).[8]

Grammar rule instances are the DELPH-IN equivalent of HPSG schemata: they define how signs may be combined. But unlike schemata, the arity of the rules as well as the relative order of their daughters is fixed. Moreover, the arity is generally kept small. Most rules are unary or binary rules. Occasionally, a grammar may include a ternary rule. In addition to the obvious consequence that the domain-based analysis of scrambling is not available, these conditions have a large impact on the number of rules used in the grammars. If a head can combine with its daughters in more than one order, this results in the introduction of multiple variations of a grammar rule. In the case of two daughters, the rule must be introduced in both directions. When more daughters are involved, even more rules may be necessary to allow the head to pick up its arguments in alternative orders.

According to the grammar catalogue,[9] the ERG includes 207, Jacy 51, and Wambaya (wmb) 88 grammar rules. Even though this includes some unary

[8]See Kathol (2000) for an elaborate account using domains to model German word order.

[9]`http://moin.delph-in.net/GrammarCatalogue`, accessed 24 July 2012

rules which would also be used as constructions in theoretical HPSG, this clearly surpasses the "five and maybe more, but not many more" schemata suggested in Pollard and Sag (1994). Not all of these additional rules are due to the different treatment of word order. Drellishak and Bender (2005) point out that coordination is a source of additional rules. In many cases, it can be advantageous to exclude certain structures by restrictions on grammar rules rather than waiting until ill-formed structures are excluded by lexical constraints. Such decisions may as well have an impact on the number of rules needed to get broad coverage of phenomena in a language.

2.2.2 DELPH-IN Parsing and generation

The previous section described some of the main differences between DELPH-IN grammars and theoretical HPSG. This section explains the basics behind the parsing and generation algorithms that are used with these grammars. The purpose of this explanation is to provide some background as to why certain decisions in grammar design may influence efficiency. For reasons of clarity, I will stick to the basics of the algorithms. Properties and optimisations of parsing and generation algorithms that were not used in the experiments reported in this thesis will not be addressed.

Several tools for parsing and generation have been developed that are related to research in the DELPH-IN consortium. Linguistic Knowledge Building (Copestake, 2002, LKB) can both parse and generate and was developed with the particular purpose to support grammar development. The PET parser Callmeier (2000) was developed as a platform to test unification algorithms for efficient parsing with large grammars. It serves as a core parsing component in several applications (see Oepen et al. (2007), Frank et al. (2007) and Suppes et al. (2012), among others). Several improvements have been added to PET, including subsumption based packing (Oepen and Carroll, 2000) originally developed for LKB, statistical parse ranking (Zhang et al., 2007) and unknown word handling (Zhang, 2007; Dridan, 2009).

Recently, two new tools have been developed that can be used for parsing and generation with DELPH-IN grammars. The Answer Constraint Engine (Packard, 2011, ACE) has been developed for efficient processing with DELPH-IN grammars. Parsing and generation with ACE both perform around 15 times faster than LKB. Its performance for parsing is comparable to PET, but ACE can be "significantly faster in certain common configurations" (Packard, 2011). The other recent addition to parsing tools is *agree* (another grammar engineering environment) (Slayden, 2012). The system was built as part of a project to test a new efficient unification algorithm. Performance is in league with that achieved by PET, with room to scale up performance for *agree* (Slayden, 2012).

The parsers all use bottom-up chart parsing, which will be explained through the example expression *the girls like strawberries*. This explanation is based on Copestake (2002), Malouf (2000) and Oepen and Carroll (2000) writing about LKB, and Callmeier (2000) and Dridan (2009) writing about PET. It is followed by a brief explanation of chart generation as implemented in LKB based on Carroll et al. (1999).

Parsing

Figure 2.11 presents a possible chart for parsing the expression *the girls like strawberries*. The expression has two readings, one where the expression is a sentence and one where it is a noun phrase. According to the most intuitive reading, there are girls who like strawberries (the sentence reading). The less intuitive reading is that the girls are like strawberries (the noun phrase reading). The syntactic structures are represented in Figure 2.12.

Each cell in a chart covers a given span and stores all possible structures that the grammar licenses for this span. The chart in Figure 2.11 should be read as follows: The covered span is marked in italics at the bottom of each cell. The structures are indicated by the name of lexical items or applied rules and identified with an edge number in square brackets ("[ ]") before the rule. The daughters of each rule are indicated by their edge number in regular brackets ("( )") after the rule. Items that are not followed by brackets come

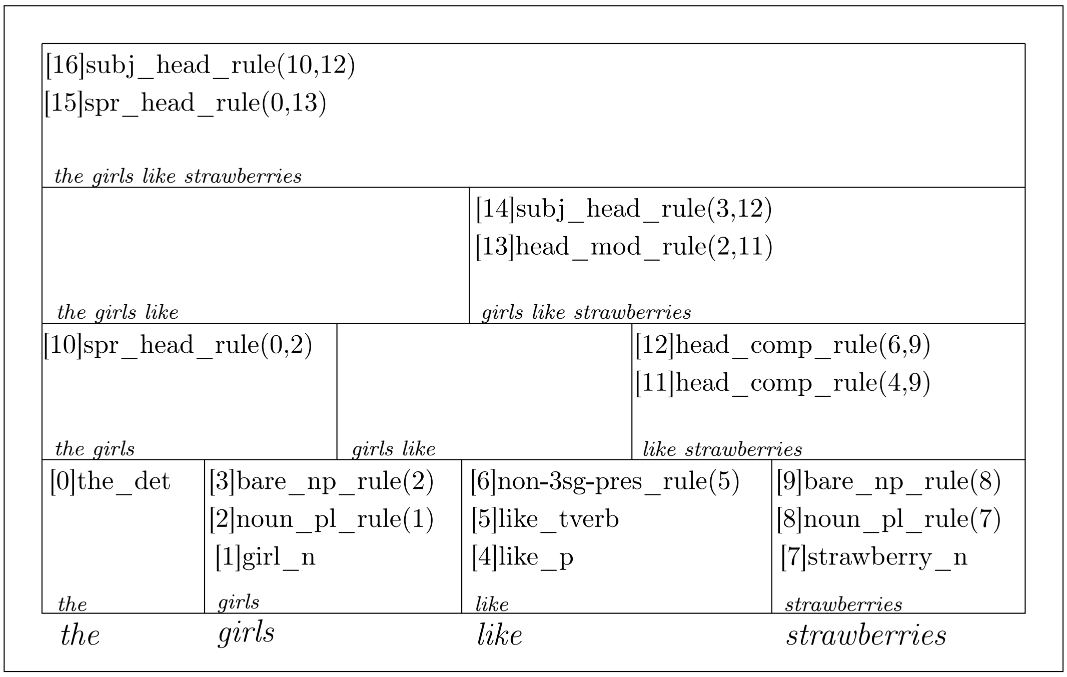


Figure 2.11: Chart for parsing *the girls like strawberries*

from the lexicon. The trees in Figure 2.12 can be linked to the chart by the edge numbers of individual nodes, marked in square brackets.

In the first stage of building up the chart, orthographic rules are applied to the input tokens of the string to produce possible lemmas of lexical items (Dridan, 2009). These lemmas are looked up in the lexicon and each candidate lexical

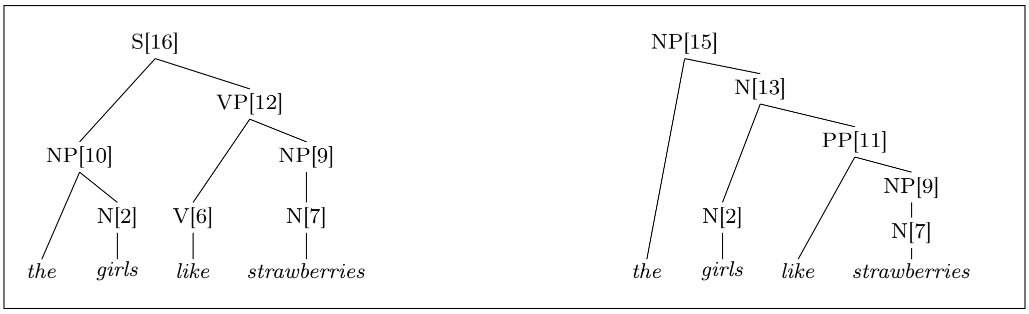


Figure 2.12: Parse trees for *the girls like strawberries*

item is added to the chart. The grammar that would produce the chart in Figure 2.11 has two lemmas for *like* and one for *the*, *girl* and *strawberry*.

Next, morphological rules are applied and added to the chart where the orthography of the surface form and constraints on the rule and its daughter permit it. For instance, a rule assigning plural to nouns is applied to both *girls* and *strawberries* (edges [2] and [8], respectively). The rule assigning singular cannot apply, because it would lead to the wrong surface forms. In the case of *like*, the rule indicating that the verb has a non third person singular subject (edge [6]) does not apply to the preposition *like*, because it is restricted to verbs.

After the possible structures for a word have been created, they can be combined by syntactic rules.[10] Syntactic rules are applied according to an agenda based parsing algorithm. This means that active edges (i.e. candidate phrases that still need one or more daughters to be complete) are placed on an agenda. The algorithm tries to combine active edges with passive edges (i.e. complete phrases) from the chart to form new edges, which may be either active edges to be placed on the agenda or passive edges to be added to the chart.

Passive edges created by unary rules are added to the same chart cell as their daughter (e.g. the bare_np_rule creating NPs in edges [3] and [9]). Binary rules can combine two edges from adjacent cells. The passive edges created by binary rules are placed in the cell that represents the span of both their daughters. The parser returns all structures that span the entire input string and are conform to the defined root conditions. If no edge is created that conforms to these two conditions, the parse fails. In the example above, two parses will be created if the root conditions permit noun phrases as well as sentences. If the root is restricted to sentences, the parser will provide only one analysis, namely where *like* is a verb.

[10]To be precise, it must be defined in the grammar that only words or phrases can be input of syntactic rules, e.g. that the verb *like* cannot be used as a daughter for a rule without the non-3sg-pres_rule having applied. Grammars generally include this restriction. Processors have a related restriction that only lexical rules may be interleaved with any spelling change rules, conform to lexical integrity.

Generation

The LKB generation algorithm (Carroll et al., 1999) generates from MRS representations to strings in three phases: lexical lookup, chart generation and modification insertion.

The lexical lookup in generation is similar to the stage in parsing where possible words in the input string are identified. All instances of the grammar (lexical items, lexical rules and grammar rules) are indexed according to the semantic relations they contribute. During lexical lookup, items that are compatible with the input semantics are selected. This means that all the items inserted either contain relations that are found in the semantics, or do not contribute semantic relations. Next, lexical rules that may take selected items as their input are applied. Just as orthography constrains possible rules in case of parsing, the semantics constrains which rules may apply for generation: the semantic contribution of lexical rules (if any) must be compatible with the input.

Carroll et al. (1999) point out four challenges for the approach. First, more than one lexical item may contribute the same semantic relation (i.e. entries for synonyms). This is similar to the ambiguity of *like* in the parsing example. In this case, the generator will create sentences with all synonyms assuming that they may lead to well-formed expressions. Second, one item may introduce several relations. This is similar to multiword expressions for parsing, which I will leave out of scope of this basic explanation.

The third challenge lies in the possibility that part of the semantics comes from a grammar rule, rather than a lexical entry or lexical rule. Therefore the semantic index of grammar rules is used to select potential contributors during lexical lookup. Finally, lexical items that do not contribute semantics form a challenge, because they could potentially be inserted for generation from any semantic representations. In most cases, this insertion would only lead to failure somewhere down the road. In order to prevent this undesirable behaviour, trigger rules can be defined that determine when lexical items without relations may be inserted.

The second phase of the generation process is chart generation. The approach taken is bottom-up head-first. It is similar to chart parsing explained above, but again with the difference that an edge must cover part of the semantic representation rather than part of a string. When active edges are to be completed, the generator looks among inactive edges for candidates whose semantic index leads to a semantic structure compatible to the input MRS after the rule is completed.

This approach can be problematic when several edges have the same index. Intersective modifiers form a well-known case where efficiency problems may occur in chart generation. There may be an indefinite number of intersective modifiers that may equally well form part of the semantic representation and their relative order cannot be determined locally. Carroll et al. (1999) explain this phenomenon through the example of *big German consultant*, where (among others) the edge *big consultant* would be created only to fail later on when it turns out that it is not possible to add *German* to the structure (assuming that the grammar is rather restrictive about the permitted order of adjectives). Furthermore, structures such as *the consultant*, *the big consultant* and *the German consultant* may all be produced only to fail when it turns out no other adjectives can be added to the NP. To circumvent this problem, intersective modification is treated in a separate phase after chart generation has completed. A description of this process (which does not play a direct role for the experiments in this thesis) can be found in Carroll et al. (1999).

2.2.3 Summary

This section provided an overview of grammars implemented as part of DELPH-IN. Differences between choices in grammar design and theoretical HPSG have been pointed out. Finally, a brief explanation was given of the parsing and generation algorithms that can be employed with these grammars.

2.3 The LinGO Grammar Matrix

This section presents the LinGO Grammar Matrix (Bender et al., 2002, 2010), a multilingual resource that supports the development of new DELPH-IN grammars. This project has a central role in the work carried out as part of this thesis. The Grammar Matrix provided the main inspiration for the idea of metagrammar engineering and its architecture provided a platform to implement the idea directly.

The section starts with a short historic overview of the project introducing the main goals and basic structure of the Grammar Matrix. This is followed by a more detailed description of the customisation system and the functions it includes. Additional attention is given to those functions that play an important role in the metagrammar. The concluding remarks on this section address the feedback loop from users to the Grammar Matrix and the question of how to get most out of grammar customisation.

2.3.1 A historic overview of the Grammar Matrix

The first version of the Grammar Matrix was presented in Bender et al. (2002). At the time, three large grammars were under development. They were the LinGO English Resource Grammar (Flickinger, 2000, ERG), the Jacy grammar of Japanese (Siegel and Bender, 2002) and the German Grammar (Müller and Kasper, 2000; Crysmann, 2005, GG).

Bender et al. (2002) point out that the grammars were mainly developed independently, despite the fact that they use the same theoretical framework and formalism. The creation of these grammars had at the time required between 5 and 15 years of development by experts. They furthermore explain that each grammar comprises between 35,000-70,000 lines of code and are documented in a manner that makes it far from trivial to extract knowledge about analyses or learn about best practices for grammar engineering. These challenges form the main motivation for developing the LinGO Grammar Matrix starter kit.

The main component of the original Grammar Matrix starter kit is a file called *matrix.tdl*. This file defines the top of the hierarchy of typed feature structures for new grammars. Bender et al. (2002) distinguish several aspects of the grammar defined in the Grammar Matrix. It includes the general feature geometry of signs and technical devices, such as lists. Basic lexical and phrasal rule types are defined, including unary and binary rule types, headed and non-headed rule types and rule types defining the relative order of head and non-head daughter. Furthermore, constructions corresponding to the HPSG schemata are present, with underspecified word order constraints. Finally, *matrix.tdl* provides the basics to construct semantic representations according to the principles of MRS. In addition, the starter kit includes configuration files to use the grammar with LKB and PET, and more recently ACE.[11]

The definitions in *matrix.tdl* were primarily based on definitions from the ERG. The Japanese grammar Jacy, a smaller grammar for Spanish and general knowledge about typological variation was used to help inform the decisions of what to keep from the ERG with the goal of making *matrix.tdl* as language independent as possible. Several grammars have been constructed that made use of the original Grammar Matrix starter kit since its introduction. They include grammars for Norwegian (Hellan and Haugereid, 2003), Spanish (Marimon, 2010), Modern Greek (Kordoni and Neu, 2005), Portuguese (Branco and Costa, 2010), Korean (Song et al., 2010) and Hausa (Crysmann, 2012).

Since 2004, the Grammar Matrix has been used in a grammar engineering course taught at the University of Washington (Bender, 2007).[12] In this course, students create initial grammars based on the Grammar Matrix. The grammars for individual languages are developed in parallel. Course instructions guide students to cover the basics for a given phenomenon depending on the properties of the language. Bender soon noticed that instructions such as 'if your language behaves like x, then implement y' could also be handled

[11] *agree* can work with grammars configured for LKB or PET.
[12] `http://courses.washington.edu/ling567/`

by a script. This inspired the Matrix developers to create an extension of the Grammar Matrix that generates language specific analyses based on user input.

Bender and Flickinger (2005) present the initial steps of what later became the Grammar Matrix customisation system (Bender et al., 2010). Their system includes initial implementations organised in modules for basic word order, yes-no questions, sentential negation and lexical items, including nouns, transitive and intransitive verbs, auxiliaries, determiners, negation markers and case marking adpositions. Drellishak and Bender (2005) describe a module for coordination that was integrated in the same system. As the customisation system grew to cover more phenomena, it soon became clear that there was not much modularity between the modules, and they were renamed *libraries*.

The original customisation system was mainly based on linguistic variations known to the developers. The extensions introduced by Drellishak and Bender (2005) were the first to be made based on a broad typological study (Drellishak, 2004). Drellishak (2009) introduces further improvements based on profound typological research for case, agreement for person, number and gender and an implementation for direct-inverse marking.[13] The possibilities of the system were further increased by the introduction of iterators. The aforementioned modules were static; only a fixed set of questions could be answered to trigger language specific implementations. Iterators allow the user to answer specific questions an arbitrary number of times, for instance to add all cases occurring in a language or a large number of lexical items. A more detailed explanation of the possibilities and structure of the customisation system will be provided in Section 2.3.2. The library for morphotactics (O'Hara, 2008; Goodman, 2012) makes use of iterators to create inflectional lexical rules.

More recently, minor revisions to ensure correct word order for verbs and

[13]Some languages mark arguments differently based on a grammatical scale, where they are ranked according to how appropriate they are to occur as the subject of the clause (Drellishak, 2009, 89). This phenomenon is called direct-inverse marking.

auxiliaries and a basic implementation for verb second word order were introduced by Fokkens (2010). Saleem (2010) added a library covering argument optionality, or pro-drop, across languages. Poulson (2011) made significant improvements to capture tense, aspect and mood. She mostly improved the possibility of building type hierarchies that represent semantic properties or syntactic features. Crowgey (2012) improved the negation library. Recently, Sanghoun Song has extended the Grammar Matrix and its customisation system to support information structure (Song and Bender, 2012; Song, 2014).

Several grammars have been created using the Grammar Matrix customisation system. They include medium-sized linguistic grammars for Wambaya (Bender, 2008a), ManGO for Mandarin Chinese, BURGER for Bulgarian (Osenova, 2010) and the RRG for Russian (Avgustinova and Zhang, 2010) together with its included core of Slavic types (Avgustinova and Zhang, 2009). Smaller experimental grammars to test the behaviour of specific phenomena have been implemented for Turkish (Fokkens et al., 2009), Georgian (Borisova, 2010), Min Nan by Goodman and Thai by Slayden. Finally, the class mentioned above has lead to the implementation of 94 small grammars for 92 languages (`http://depts.washington.edu/uwcl/twiki/bin/view.cgi/Main/LanguagesList`, accessed 28 April 2014). The customisation system was used for 54 grammars created in 2006 and later years.[14]

2.3.2 The customisation system

The previous section introduced the main components of the Grammar Matrix through a historic overview of how it was created. This section provides more detailed information concerning the structure of the customisation system, the work-flow when working with it and the methods it uses. The description below is largely based on Bender et al. (2010) and Drellishak (2009).

Figure 2.13 provides a schematic overview of the customisation system. The

[14]In principle, students should select a language that was not used in the class before. Due to the changes in the course as a result of the improvement of the customisation system, languages from the earlier years are permitted again.

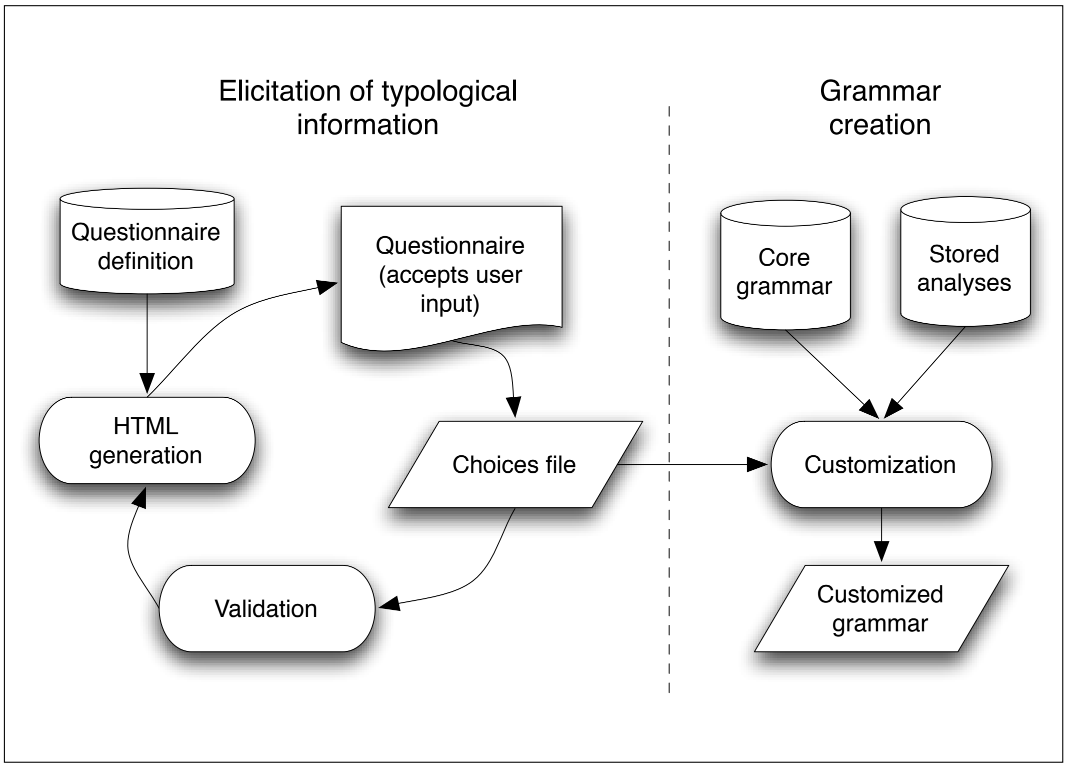
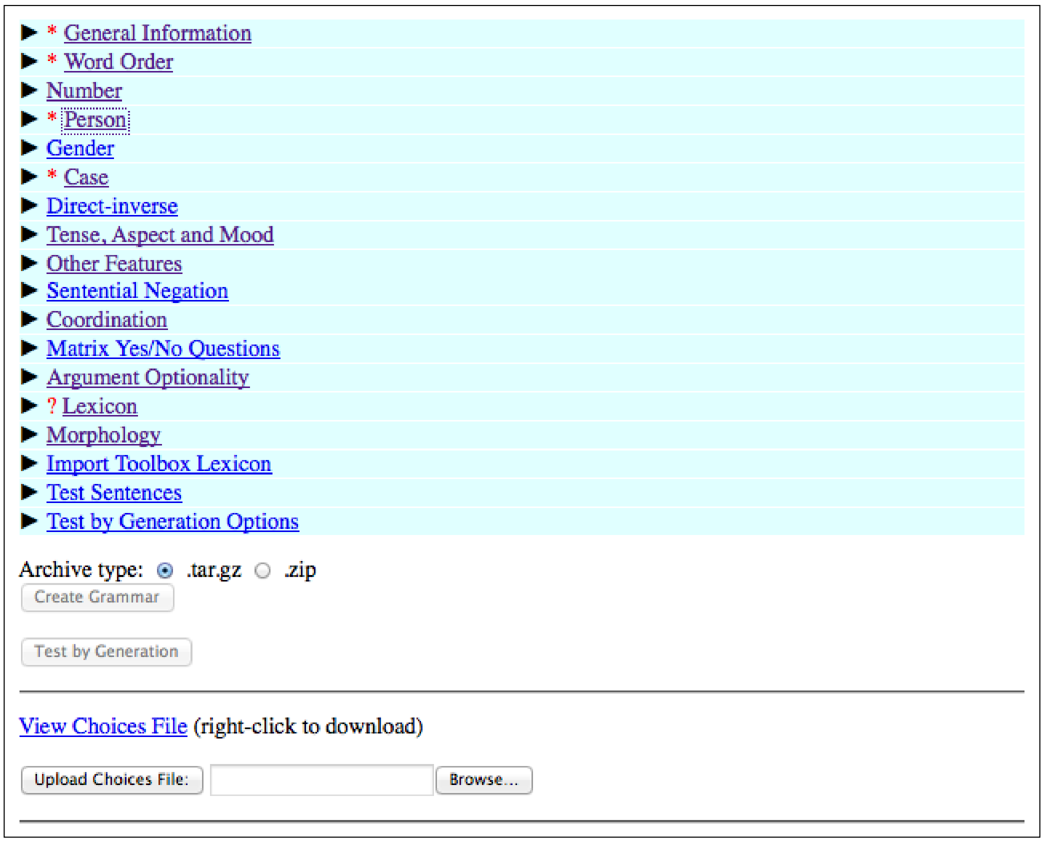


Figure 2.13: Overview of the Grammar Matrix customisation system (Bender et al., 2010, p. 7)

system is divided in two sections: on the left-hand side the components of the web-based questionnaire are depicted, the right-hand side depicts the internal structure of the customisation system itself.

Questionnaire

A static list of question definitions implemented by Matrix developers is used as input to dynamically create the HTML code for the questionnaire. Users fill this questionnaire out to define input for the customisation system. Definitions based on this input are called *choices* and stored in a so-called *choices file*. The HTML displayed on the questionnaire and its subpages change dynamically in response to user input. Input provided in one section of the questionnaire (where sections more or less correspond to individual


▶ * General Information
▶ * Word Order
▶ Number
▶ * Person
▶ Gender
▶ * Case
▶ Direct-inverse
▶ Tense, Aspect and Mood
▶ Other Features
▶ Sentential Negation
▶ Coordination
▶ Matrix Yes/No Questions
▶ Argument Optionality
▶ ? Lexicon
▶ Morphology
▶ Import Toolbox Lexicon
▶ Test Sentences
▶ Test by Generation Options

Archive type: [✓ .tar.gz] .zip

View Choices File (right-click to download)

Figure 2.14: Screenshot of questionnaire's main page, taken 13 August 2012

libraries) can lead to additional options for other sections. Features defined in sections such as case, person, number, gender, and tense, aspect and mood are available when defining lexical items or lexical rules. The sentential negation and argument optionality sections can also trigger features that can be assigned to lexical items or rules in the relevant sections.

A process called validation, which also leads to updates of the questionnaire, takes place when users leave the page of a specific library. Validation checks whether the user's choices lead to a consistent grammar that can be used for parsing. When the questionnaire is empty, validation indicates by the means of red stars that the user should provide a name of the grammar, define basic word order and indicate which person and case marking the language exhibits (both of which can be set to 'none'). The lexicon bears a red question mark: the system allows users to create a grammar with empty lexicon, but this grammar cannot parse or generate until a lexicon is provided. Users can get feedback on what red stars mean by hovering over them with their mouse. As the questionnaire is filled out, validation checks for inconsistencies. If users, for instance, define auxiliaries in the lexicon, validation makes sure they indicate that the language has auxiliaries on the word order page and define the relative word order between auxiliary and verbal complement. When no inconsistencies or missing information are identified, the user can click the button "create grammar" to customise their initial grammar.

Grammar Customisation

As depicted on the right-hand side of Figure 2.13, customisation takes a choices file as input and combines information from a component called 'core grammar' and a component labeled 'stored analyses'. The core grammar corresponds to the original components of the Grammar Matrix: an updated version of *matrix.tdl* and files to make the grammar work with DELPH-IN tools. The libraries provide stored analyses that are evoked by definitions in the choices file.

```
section=word-order                  section=lexicon
word-order=v2                       noun1_name=1st-pron-nom
has-dets=yes                          noun1_feat1_name=person
noun-det-order=det-noun               noun1_feat1_value=1st
has-aux=no                            noun1_feat2_name=case
                                      noun1_feat2_value=nom
section=case                        noun1_det=obl
case-marking=nom-acc
nom-acc-nom-case-name=nominative    verb1_name=trans
nom-acc-acc-case-name=accusative    verb1_valence=nom-acc
```

Figure 2.15: Small extract of a choices file

Figure 2.15, taken from Fokkens et al. (2012a), presents an excerpt of a

Grammar Matrix choices file. During customisation, an object containing definitions from the choices file is created. Next, TDL files for all language specific properties are initiated. They include files for phrasal rules, inflection rules, non-inflecting lexical rules, the lexicon, root conditions and the bulk of the language specific type hierarchy in a file called *language_name.tdl*,[15] where *language_name* stands for the name of the language as defined in the choices file.

Customisation then walks through the code of individual libraries. The most central function in this process is the function `add` for TDL objects. It takes at least two arguments: a TDL object and a definition written in TDL syntax. A comment and section definition (indicating where in the final TDL file the definition should appear) can be added optionally. When a type is already present in a TDL object, the function `add` merges the old and new definitions by default. This way, Matrix developers can define properties of types at different locations of the customisation system. The function `add` is crucial for capturing interactions between phenomena, especially when properties defined in different libraries must be combined to define a specific type.

The most basic functions in customisation are if-then-else conditions or iterators. In the simplest case, property x defined in the choices file leads to definition y added to a TDL object. More sophisticated methods are provided to create user defined type hierarchies, to determine the correct location of feature value pairs in the feature geometry and to create the correct relations between lexical rules. I will briefly elaborate on the customisation of type hierarchies and morphotactics, because they play an important role in the flexibility and development speed of CLIMB grammars.

Customising type hierarchies

User defined type hierarchies for case, the agreement features person-number-gender, verbal features tense-aspect-mood and features belonging to the open

[15]The root conditions can also contain language specific properties and naturally, instances included in the lexicon, inflectional rules and phrasal rules are language specific.

category ‘other features’ are formed in three stages, occurring before, during and after the treatment of lexical items and morphosyntax. First, a TDL hierarchy object is initiated for the feature values in question. This object includes definitions of individual types and their supertypes according to the user’s input. Later, these hierarchy objects are updated to include underspecified types corresponding to disjunctive feature values specified in the lexicon and morphosyntactic rules of the language. After lexical items and morphosyntax has been treated, these individual hierarchy objects are used to add the appropriate definitions to *language_name.tdl.*

Properties that can be assigned to lexical items or morphosyntactic rules have a well-defined location in the feature geometry. For instance, *person*, *number* and *gender* will be part of the index of nouns, whereas [CASE *case*] is a property of the head of the noun. For nouns, users simply define the name of the feature and its value in the questionnaire. A library called *features* determines the complete feature path based on the name of the feature and the fact that it is assigned to a noun. Verbs, on the other hand, can assign these feature values to the index or head of their subject or object. For verbs, users must define whether the feature is a property of the verb or one (and which one) of its arguments. Again, the full feature path is determined by the customisation system.

Morphotactics

The customisation system’s morphotactics library, originally implemented by O’Hara (2008) and improved by Goodman (2012) is one of the most powerful parts of the customisation system.[16]

Inflection can be defined for nouns, verbs and determiners. First, slots are defined in so-called “position classes”. As the name indicates, these classes define the location of morphemes. The position of a morpheme depends on two factors: when it is added (e.g. directly next to the root or next to another

[16]The description in this thesis is limited to the possibilities offered by the morphotactic library. See Goodman (2012) for a more detailed description and explanations about how the morphotactic implementations work.

position class) and whether it precedes or follows the stem it is attached to. When defining the position class, the user indicates what input it can take (i.e. where it can be attached to the word), whether it is a prefix or a suffix and whether it is optionally or obligatorily filled.

Each position class has one or more lexical rules for which it determines their surface position. The lexical rules are responsible for assigning grammatical properties to the word, such as tense or case values or agreement properties. A lexical rule can have one or more instances which may be an overt affix or not (e.g. a plural noun lexical rule for English may have an instance with the suffix *s*, and the singular noun rule an instance without an overt affix). Lexical rules can be defined hierarchically. It is for instance possible to define lexical rules assigning tense and lexical rules defining agreement properties. If tense and agreement are merged into one marker, the appropriate lexical rules can be defined by inheriting from the right tense and agreement markers.

Finally, constraints can be used to make sure morphological rules interact correctly. Constraints can require or forbid another morpheme. They can be defined on lexical items, position classes or individual lexical rules. Constraints can model a variety of phenomena including circumfixes (such as German participles, where the prefix requires a specific suffix), morphological classes (such as the four main conjugation classes in French) or assimilation phenomena (such as vowel harmony in Turkish).[17]

Completing customisation

After all libraries are called, the definitions stored in individual TDL objects are printed in relevant TDL files. The thus created TDL files and configuration settings form the output grammar of the customisation system, which can be downloaded, tested, inspected and improved manually.

[17]It should be noted, however, that the morphotactics page covers the interaction between morphology and syntax only and is not designed to support morphophonological phenomena. Even though vowel harmony can be modelled with the morphotactics system, Bender and Good (2005) recommend to use an external morphology for languages with a rich morphophonology such as Turkish.

Regression testing

Regression testing is important for all longterm software development projects and the Grammar Matrix, like metagrammar development, is no exception. The customisation system contains a setup for regression setting that can readily be adopted for metagrammar engineering. When adding or improving a library in the customisation system, Matrix developers add choices files and gold standard output of a test suite covering phenomena from that library to the regression tests (Bender et al., 2007). Before updating the live customisation system, regression tests are run that customise grammars from these choices files and parse all items in the associated test suite with it. The coverage, overgeneration and semantics output are compared to the gold standard using the `[incr tsdb()]` competence and performance profiling system (Oepen, 2001). Regression tests fail if coverage, overgeneration or the semantic representations changed.

2.3.3 Concluding remarks on the Grammar Matrix

This section has introduced the LinGO Grammar Matrix. A historic overview was given introducing the two main components of the Grammar Matrix: the static core of the Grammar Matrix and the dynamic customisation system. Section 2.3.2 provided a more detailed description of the technology behind the latter, walking through the process of creating a starter grammar through the web-based questionnaire.

Two aspects of the Grammar Matrix related to this thesis have not been addressed in the descriptions above. The first aspect concerns the insight that can be gained from individual grammars and may improve the Grammar Matrix. The original matrix core was mainly based on grammars for English and Japanese. The idea was that this would function as a set of hypotheses for generally crosslinguistically applicable types, which could be continuously be revised based on experience from individual grammars using the core. Investigations in Chapter 6 will show that this feedback loop has not functioned as well as initially hoped. In general, grammar engineers do not inform developers of the Grammar Matrix when they find errors or suboptimal implementations in the core.

The second aspect relates to the question of how to get the most out of the customisation system. The previous section indicated that the customisation system allows users to define type hierarchies and an indefinite amount of lexical items and morphotactic rules. Mainly because of these properties, the full range of possibilities provided by the customisation system may not always be transparent to users. The Grammar Matrix aims at theory independent descriptions, that should be easy to understand and not require elaborate explanations. Designing such questions for specific implemented analyses is, in my experience, one of the most challenging tasks in the development process of Grammar Matrix libraries. Because it is not trivial to translate specific implementations to short, logical and intuitive descriptions, it is not always clear for users which answers in the questionnaire lead to the desired result at first sight. This can clearly be seen in the evaluation carried out by Bender et al. (2010). In that paper, we present results on test data after filling out the questionnaire once for each language as well as results after optimising the customisation input. Even though this evaluation was carried out by experts, results significantly improved for all languages in their evaluation. Borisova (2010) manages to implement an analysis for Georgian polypersonal agreement almost exclusively using the customisation system. Creating the correct morphosyntactic model for this complex phenomenon required a fair amount of research.

Bender et al. (2011) provide a work flow for grammar engineering. Grammar engineering starts by a the creation of test suites containing grammatical and ungrammatical examples related to given phenomena. When an implementation is added, the grammar is compiled and tested both on data for the phenomenon that is currently treated and data of phenomena previously included in the grammar. The grammar is debugged based on observations on the test data until the phenomenon is covered correctly without breaking previous implementations. This general grammar engineering methodology should ideally be adopted from the first moment of working with the customisation system. The results from Bender et al. (2010) as well as Borisova's (2010) achievement show that it can be worthwhile to 'debug' grammars by changing the input of the questionnaire until good coverage of phenomena covered by the customisation system is reached. Fokkens et al. (2012b) provide elaborate documentation on how to fill out the questionnaire.

2.4 Summary

This chapter presented the context of the work carried out as part of this thesis. The main contribution of this thesis is a new methodology for grammar engineering that, in first place, was developed to support long term development of DELPH-IN grammars. Section 2.1 provided a basic introduction to HPSG, the theoretical framework that forms the basis of DELPH-IN grammars. Section 2.2 described formal properties of DELPH-IN grammars as well as the basics behind the algorithms used to parse and generate with these grammars. Finally, the Grammar Matrix was introduced in Section 2.3. The CLIMB methodology makes use of the software developed as part of the Grammar Matrix. The tight link between these two projects will be explained in more detail in the next chapter, which introduces CLIMB.

Chapter 3

The CLIMB Methodology

There are few areas of science in which one would seriously consider the possibility of developing a general, practical, mechanical method for choosing among several theories, each compatible with available data.
(Chomsky, 1957, p.53)

This chapter introduces CLIMB: Comparative Libraries of Implementations with a (grammar) Matrix Basis; (Fokkens, 2011a; Fokkens et al., 2012a). CLIMB's main contribution is that it introduces a new methodology for grammar engineering. This methodology allows grammar writers to monitor alternative models for grammars of natural language. As mentioned in Chapter 1, the observation that such an approach addresses a major challenge in grammar design is, to my knowledge, first made in Fokkens (2011a) and CLIMB is the first approach that tries to address this challenge.

CLIMB is closely related to the Grammar Matrix described in Chapter 2, Section 2.3. Recall that the Grammar Matrix customisation system automatically generates implementations of linguistic precision grammars. Whereas code generation is used only once to create a starter grammar in the Grammar Matrix, its application is generalised in CLIMB to be used throughout the entire grammar development process. The basic idea behind CLIMB is that code generation allows the grammar engineer to maintain alternative analyses in parallel supporting more effective and systematic exploration of the analysis space.

CLIMB uses the basic architecture of the Grammar Matrix. It should be noted, however, that the software presented in this chapter can in principle be used to develop any grammar in TDL and the basic idea is theory and framework independent.

This chapter is organised as follows. Section 3.1 introduces CLIMB in its traditional form which was used for the implementations in this thesis. The section starts with a description of the process leading to the idea behind CLIMB. Subsection 3.1.1 elaborates on the relation between the Grammar Matrix and CLIMB. An overview of the system and the workflow of applying the methodology are given in Subsection 3.1.2.

The CLIMB metagrammars developed as part of this thesis mainly consist of procedural implementations in Python that generate TDL. This means that developing CLIMB requires procedural programming which may be difficult for grammar engineers who are used to implementing their grammars declaratively in TDL. An alternative version that allows grammar engineers to write the metagrammar declaratively, mainly using TDL, has been developed to make CLIMB more accessible. This version of CLIMB, described in Section 3.2, will be called *declarative* CLIMB in this thesis. The term CLIMB alone will refer to the original setup of CLIMB, which will occasionally be specified by *procedural* CLIMB.

Section 3.3 introduces four TDL processing algorithms that were developed to support CLIMB. The spring cleaning algorithm (Fokkens et al., 2011) can identify types included in the grammar that do not have an impact on its competence. The main goal of this algorithm is to support a cleaner design of grammars. Its intended use in this context is to identify differences between grammars without taking left-overs of old analyses into account. The other three algorithms are a set that can abbreviate and complete paths in feature structure definitions. The first of the three algorithms uses the basics of the spring cleaning code to extract the feature structure of the grammar. The second and third use the extracted feature structure to create abbreviated paths for the declarative metagrammar and to complete these abbreviated paths if the declarative metagrammar is used to generate a linguistic precision grammar.

Section 3.4 presents several applications for CLIMB. In addition to monitoring alternative models for a grammar in parallel, it increases modularity, it supports multilingual grammar development and adaptations for alternative dialects as well as alternative versions for different applications. The setup facilitates a phenomenon-based organisation of the grammar, where its flexibility can manipulate whether rare phenomena are included or excluded in the grammar. Finally, the processing tools provide support to investigate properties of the grammar. The presentation of CLIMB applications is followed by a summary of the chapter in Section 3.5.

3.1 An introduction to CLIMB

The idea behind CLIMB originates in research on word order analyses for the LinGO Grammar Matrix. Fokkens (2010) describes a basic flat analysis for verb second word order based on Bender's (2008a) grammar for Wambaya. The analysis used in the Wambaya grammar differs from the standard HPSG analysis for verb second word order in Germanic languages, where a filler-gap construction is used to ensure that conjugated verbs appear in second position. The differences between these analyses will be explained in more detail in Chapter 4. The two analyses for verb second order lead to questions about how to handle such situations in the Grammar Matrix.

Multilinguality may influence the choice between alternative analyses in several ways. On the one hand, the possibility of using the same analysis to capture phenomena across languages may be a reason to adopt it rather than an alternative. On the other hand, this crosslinguistically applicable analysis may not be the best option for every language that exhibits the phenomenon it captures. If a resource like the Grammar Matrix provides an analysis that is not the right option for a language, it may harm the resulting grammar. The grammar engineer is set on a wrong path at the beginning of development. It cannot be foreseen in such a case what the burden may be to adapt the customised analysis when it is discovered that another option would work better, if this gets discovered at all. It could be that the 'wrong' analysis influenced a number of other implementations that need to be revised. Likewise, these implementations may no longer be the most suitable way to capture a phenomenon. The 'wrong' analysis may have excluded more suitable options that have become possible after it was adapted. Furthermore, the revision may have a negative impact on some interactions with other analyses. For these suboptimal solutions resulting indirectly from the original 'wrong' analysis in the customisation system, it also holds that they may remain unnoticed.

It is hard enough to verify whether a given analysis can capture a phenomenon across a set of languages. Investigating whether an analysis is ideal within a given language also forms a major challenge. It would probably take a team of researchers several years to combine these two and find out whether an analysis also forms an optimal solution across languages in large grammars. We therefore decided to make a choice between the two alternatives of a flat verb second analysis based on Wambaya and the filler-gap analysis that HPSG literature proposes for German.[1] The Wambaya based analysis was included in the Grammar Matrix following the assumption that users working on a long term project for a Germanic language would familiarise themselves with HPSG literature and make a well-informed decision on how to treat word order early in the development process. Further discussion of the matter can be found in the Upcoming Work section Grammar Matrix documentation on word order (Fokkens et al., 2012b).

The challenge came up again, when time came to integrate correct interactions between verb second word order and auxiliaries. Bender (2010) had shown in the meantime that an alternative analysis for auxiliaries significantly improved efficiency of the Wambaya grammar. Before updating the Grammar Matrix, I wanted to investigate whether this alternative analysis

[1]This decision was taken together with Emily Bender.

would also improve efficiency in grammars for Germanic languages.

For this purpose, I compared the new analysis to the standard HPSG analysis in small grammars covering Danish, Dutch and German word order (Fokkens, 2011a). For practical reasons, I extended the syntactic libraries from the Grammar Matrix customisation system. New analyses had to be shared both across languages and across different versions of the grammar (using alternative analyses for auxiliaries) and code generation was the fastest way to do this. Moreover, it guaranteed consistency among grammars for analyses that were not directly concerned with the experiment, but could influence the performance of the grammar.

As the grammars covered more phenomena, it became clear that the overall approach of using customisation could address a problem often observed in linguistic research: a hypothesis merely shown to be more likely than an alternative is taken to be 'the' correct solution and is used as a basis for future research. The adopted analysis may even be used as proof to exclude analyses for other phenomena that cannot interact correctly with it, even if the new analysis would work perfectly fine with the abandoned alternatives. When code generation is used as a general methodology, several hypotheses can be maintained in parallel as the grammar grows. Alternatives can be explored systematically until enough evidence is found to select an analysis, leading to more consistent research on linguistic structure.

The general approach of writing code that can generate an implemented grammar rather than writing the grammar itself is called *metagrammar engineering* (Fokkens, 2011a). In this thesis, the CLIMB setup is used, where the Grammar Matrix forms the basis of the metagrammar. The next subsection addresses the differences between the Grammar Matrix and CLIMB. Subsection 3.1.2 explains the workflow of this methodology.[2]

[2]The CLIMB workflow has previously been described in Fokkens et al. (2012a), from which the description below was largely taken.

3.1.1 The Grammar Matrix and CLIMB

The introduction of CLIMB has shown that it is tightly linked to the Grammar Matrix. However, both the philosophy behind the projects as well as the practice of applying them differ. The Grammar Matrix aims at lowering the hurdle of starting a new grammar. It is therefore essential that the system be easy to use and cover a wide typological range. The output of the Grammar Matrix customisation system is meant to provide the basics of an analysis. It can help starting grammar engineers and provide a quick way to try out an analysis in a small grammar (Fokkens et al., 2009). The Grammar Matrix only plays a role in the early stages of the development process.

CLIMB, on the other hand, is a methodology that particularly pays off on long term projects. It is meant to improve flexibility, investigate alternative analyses in a complex setting and provide a more modular way for implementing HPSG precision grammars. Users of CLIMB can be expected to be experts for whom it pays off to invest in the architecture of their system and in techniques that may facilitate maintenance. I will elaborate on the consequences of these differences below.

The standard approach when using the Grammar Matrix is to create a grammar through the web interface and extend this grammar manually. Most of the linguistic properties defined through the questionnaire do not reveal a direct link to the customisation system or even to HPSG theory. The only exception is the possibility of creating hierarchies of supertypes and subtypes, which points to typed feature structures and type hierarchies. The Grammar Matrix's basic approach thus emphasises the central control of "hidden" logic behind the scenes of the customisation process. The user can explore provided analyses in TDL, but no direct insight into how the customisation system came to the resulting grammar can be obtained while merely using the web interface.

CLIMB takes a radically different approach by placing the customisation source code under control of the grammar engineer, so that different layers of parameterisation can be achieved in individual grammar development projects. Users are encouraged to explore the possibilities of the customisation system and expand it for their language specific needs.

Another difference between the Grammar Matrix and CLIMB is the wide typological variations the Grammar Matrix aims to capture. Sharing implementations across languages is one of the main purposes of the Grammar Matrix. Even though CLIMB takes advantage of this possibility, the method was originally introduced to examine different analyses for the same phenomenon without (necessarily) addressing any linguistic or dialectal variation. In other words, CLIMB was created to allow grammar engineers to compare analyses that can deal with the exact same data. One could say that the Grammar Matrix explores one analysis that can be used in different manifestations of a phenomenon, whereas CLIMB explores one specific manifestation of a phenomenon that can be analysed in more than one manner. In the end, the Grammar Matrix and CLIMB complement each other. The former makes grammar engineering accessible to a wider public and provides a starting point for new grammars. The latter can be used to improve grammar development on long term projects.

The CLIMB approach has been tested through development of gCLIMB: a metagrammar that covers general word order phenomena in German, Danish and Dutch (Fokkens, 2011a) and has been extended to cover several phenomena for German only. The metagrammar contains alternative analyses for verb second word order, auxiliaries and adjectives. The metagrammar contains one regular analysis for adjectives and an alternative which supports grammar checking. Alternatives for word order and auxiliaries form the true test of the approach: word order interacts with all other phenomena and auxiliaries interact with most. An explanation and more elaborate motivation for the alternative analyses included in gCLIMB will be provided in Chapter 4. The next subsection will present the setup and workflow of CLIMB in its original form.

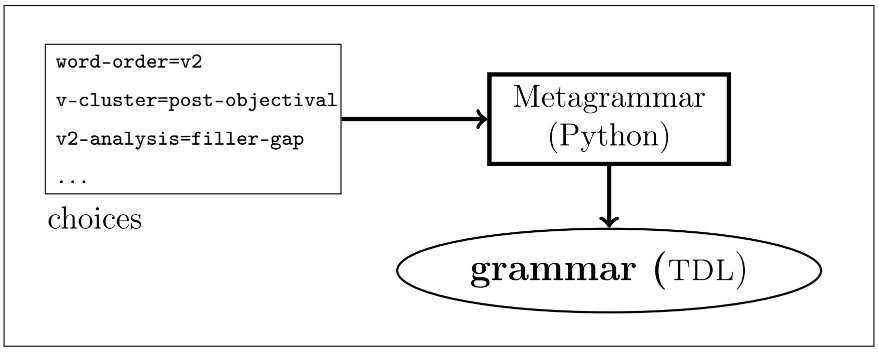


Figure 3.1: Basic overview of CLIMB

3.1.2 CLIMB overview and workflow

Basic structure of CLIMB

Figure 3.1 provides a schematic overview of CLIMB. The metagrammar takes a choices file as its input and produces a grammar in TDL. The choices file defines phenomena and properties that are generated using the metagrammar's *libraries* as described for the Grammar Matrix in Section 2.3.2. Choices are directly linked to implementations in the libraries and their parameters. The grammars reproduced by CLIMB are thus the combined result of a specific choices file and a specific version of the metagrammar. It follows that both the metagrammar and the choices file are needed to reproduce a specific grammar. Likewise, extensions and corrections to grammars are obtained by modifying the metagrammar, the choices file or, in most cases, both.

The CLIMB workflow

A simplified overview of the general workflow of CLIMB is provided in Figure 3.2. There are two general strategies for including new analyses in CLIMB depending on experience of the grammar engineer and the complexity of the phenomenon. The first strategy follows the workflow depicted on the left in Figure 3.2. After identifying a phenomenon and designing an analysis, grammar engineers generate the latest versions of the grammar using the

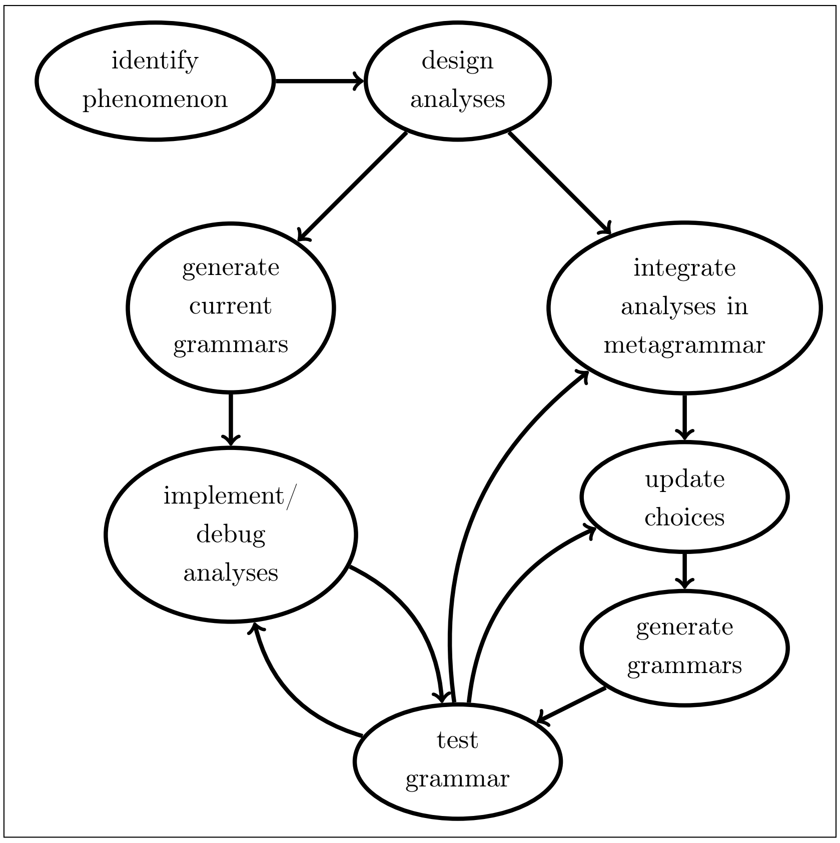

identify phenomenon
design analyses
generate current grammars
integrate analyses in metagrammar
implement/ debug analyses
update choices
generate grammars
test grammar

Figure 3.2: Simplified overview of the CLIMB workflow

latest choices files and the metagrammar. Then they follow the traditional grammar engineering approach, extending the grammar manually, testing it on test data (both data representing the new phenomenon and data for regression testing) and debugging the grammars accordingly as described in Bender et al. (2011). The analysis can be integrated in the metagrammar as soon as the grammar has achieved the desired competence. Another testing round follows to make sure that metagrammar and (updated) choices file produce the grammar as intended.

If grammar engineers follow the second strategy, they directly implement new analyses in the metagrammar. This approach is typically taken for new lexical categories and morphotactic properties. As described in Section 2.3.2, these libraries contain many general functions that can combine complex properties. Simple changes to the source code of CLIMB and a set of definitions in the choices file can give a major boost in grammar coverage. In this case, the CLIMB method can be faster than traditional grammar engineering. This is illustrated by the development of an analysis for German adjectives, which was implemented within six hours (Fokkens et al., 2012a). This included the basic setup to cover adjectives themselves and phrases to combine them with nouns (both of which are not included in the Grammar Matrix customisation system, though basic types are included in the matrix core) and German specific properties such as inflection depending on determiner, number, gender and case.[3]

As mentioned above, which of the two approaches is used depends on how experienced the grammar engineer is with CLIMB and on the complexity of the phenomenon. In many cases, the two approaches will be combined. While integrating a new analysis that does not have many interactions with the alternative versions of the grammars, a typical approach will be to implement this analysis in one version of the grammar and use the metagrammar to generate alternative versions of the grammar with the new analysis. Minor changes that need to be made to fix interactions (such as sharing a feature value) will typically be made directly in the metagrammar. It is on the other hand not uncommon to quickly try something out in a generated grammar, even if most work is done directly in the metagrammar.

Setting up a CLIMB metagrammar

The description above provided the workflow to advance the metagrammar, but we have not seen yet how a CLIMB metagrammar is initiated. The easiest way to begin developing a grammar with CLIMB is to start from an existing metagrammar and set of choices files. Both metagrammar and choices files can be adapted to remove irrelevant properties and add language specific variations.

[3]The six hours also included looking the correct behaviour up in online descriptive grammars for German.

Three resources can be used to start a CLIMB metagrammar: the Grammar Matrix, Germanic CLIMB (gCLIMB) or SlaviCLIMB, a CLIMB metagrammar for Slavic languages that covers the Russian Resource Grammar (Avgustinova and Zhang, 2010, RRG) including a Slavic core grammar (Avgustinova and Zhang, 2009).[4] The Grammar Matrix and gCLIMB are available under the MIT license and SlaviCLIMB under LGPL-LR. Section 3.2.3 provides a description of how to use CLIMB with grammars that have been developed the traditional way, either for temporarily supporting alternative versions or to convert to CLIMB step by step.

If a metagrammar for a Germanic or Slavic language is to be created, gCLIMB or SlaviCLIMB clearly form the best starting point. Both metagrammars include choices files that cover the latest versions of the grammars for German, Dutch and Danish (gCLIMB) and Russian (SlaviCLIMB), respectively. The choices files can be adapted for the new language and be used as a starting point for the development process.

If the language or languages in question do not have clear commonalities with Germanic or Slavic languages, an initial choices file can be created using the Grammar Matrix web interface. The grammar customisation system itself will form the initial metagrammar. gCLIMB and SlaviCLIMB both cover a range of lexical categories and phenomena that are currently not covered by the Grammar Matrix. They can serve as examples of how to proceed while extending the new CLIMB metagrammar. Each metagrammar will contain analyses that are not relevant for the new language(s), either because the language does not exhibit a particular phenomenon or because the grammar engineer decides not to explore a proposed analysis. I will briefly elaborate on the motivations to either keep irrelevant analyses in CLIMB or remove them.

[4]See Chapter 6, Section 6.3 for a more elaborate description of SlaviCLIMB.

Advantages and disadvantages of Matrix libraries in CLIMB

gCLIMB maintains the complete implementation of the Grammar Matrix customisation system. This decision was taken so that the project could be used to gain insight into the interaction between analyses provided by the original customisation system and new analyses added to gain analytical depth for Germanic specific phenomena. On the one hand, this possibility forms an additional strength of CLIMB. Users working on new languages may use gCLIMB to retrieve basic analyses for phenomena that are not covered by the Grammar Matrix customisation system yet, and include them in a grammar that has language specific implementations for those phenomena that are covered by the Matrix.

On the other hand, the structure of a CLIMB resource may be simplified significantly by removing options and implementations covering typological variations that do not occur in the language or languages under investigation. In a thus reduced CLIMB resource, choices files can be kept simple as well, only indicating the choice of analysis and (optionally) a list of phenomena to be included. A simpler metagrammar is more accessible to new users and grammar engineers just learning to use CLIMB. This approach is therefore considered the preferred approach for SlaviCLIMB, where several grammar engineers who do not necessarily have a strong background in programming may work with the metagrammar (cf. Section 6.3).[5]

3.1.3 Coding CLIMB

Like the Grammar Matrix customisation system, CLIMB is implemented in Python. Figures 3.3 (repeated from Section 2.3.2, p. 60) and 3.5 provide samples of a choices file and Python code found in a syntactic library, respectively.

The definitions in *choices* can be roughly divided in three categories: (i)

[5]SLAVICLIMB is currently in a transition state, where some libraries have been cleared from irrelevant Grammar Matrix analyses and others still contain the full spectrum.

```
section=word-order                section=lexicon
word-order=v2                     noun1_name=1st-pron-nom
has-dets=yes                        noun1_feat1_name=person
noun-det-order=det-noun             noun1_feat1_value=1st
has-aux=no                          noun1_feat2_name=case
                                    noun1_feat2_value=nom
section=case                      noun1_det=obl
case-marking=nom-acc
nom-acc-nom-case-name=nominative  verb1_name=trans
nom-acc-acc-case-name=accusative  verb1_valence=nom-acc
```

Figure 3.3: Small extract of a choices file

the fact that the language exhibits a specific phenomenon, (ii) the specific behaviour of a phenomenon in the language and (iii) the properties of individual elements in the language, in particular lexical items and morphemes. For instance, *choices* may include the definition *passivization=yes* which will invoke general properties needed to create passives. The definitions *pass1_marking=aux, pass1_form=participle* indicates that there is a form of passivisation that is marked by an auxiliary taking the passivised verb in participle form as its complement. The passive auxiliary itself is defined as part of the lexicon.

```
aux3_name=passive
aux3_sem=no-pred
aux3_subj=np-comp-case
aux3_aux-select=sein
  aux3_compfeature1_name=form
  aux3_compfeature1_value=pass-participle
  aux3_stem1_orth=werden
```

Figure 3.4: Example lexical entry for German passive auxiliary *werden*

Figure 3.4 provides an example definition for the auxiliary *werden* which is used to mark passives in German. Definitions of lexical items can be much more complex. Feature values can be assigned on the item itself or on its argument, capturing agreement. The same holds for the definition of morphological markings. Alternative analyses are mostly triggered by definitions

```
  wo = ch.get('word-order')

  if wo == 'v2':
    mylang.add('head-initial-head-nexus := head-initial & \
                [ SYNSEM.LOCAL.CAT.MC na & #mc, \
                  HEAD-DTR.SYNSEM.LOCAL.CAT.MC #mc ].')
    mylang.add('head-final-head-nexus := head-final & \
                [ SYNSEM.LOCAL.CAT.MC bool, \
                  HEAD-DTR.SYNSEM.LOCAL.CAT.MC na ].')

#rules shared among free and v2

  if wo == 'free' or wo == 'v2':
    mylang.add('head-subj-phrase := decl-head-subj-phrase & \
                                      head-initial-head-nexus.')
    mylang.add('subj-head-phrase := decl-head-subj-phrase & \
                                      head-final-head-nexus.')
```

Figure 3.5: Sample code from word order library: implementations triggered by word-order=v2 in choices

similar to those from category (ii) defined above, i.e. definitions that indicate the specific behaviour of a phenomenon in the language. The statement *v2-analysis=filler-gap*, for instance, selects the filler-gap analysis for verb second word order. Definitions that fall under category (iii), statements on properties of specific elements of the language such as lexical items or morphemes, also offers room for alternative analyses. It can for instance be used to create a small grammar where all forms are fully inflected or design the grammar so that it can import a lexicon and morphology from external resources.

The code presented in Figure 3.5 will be called because of the definition *word-order=v2* in the choices file. Functions in CLIMB are typically if-then-else conditions or for-loops. The code in Figure 3.5 illustrates a typical case of an if-then condition used in CLIMB. In principle, only basic programming skills as used in Figure 3.5 are needed to work with CLIMB in its original setup. The most complex components needed to use CLIMB are the functions that process and merge TDL objects. These functions are provided in the Grammar Matrix source code and need not be touched by the grammar engineer.

Nevertheless, the requirement to switch back and forth between declarative implementations written in TDL and procedural implementations in Python may become a burden to the grammar engineer. Despite the many advantages of using CLIMB on the level of organisation, consistency, flexibility and enhancement of empirically testing alternative analyses, this hurdle may prevent grammar engineers from adopting the approach. In order to address this problem, a more user-friendly version of CLIMB has been developed. The next section will describe this alternative version of CLIMB.

3.2 Declarative CLIMB

This section describes declarative CLIMB and how it can be used in grammar engineering. There are two development lines in this version of CLIMB. Section 3.2.2 describes the setup and tools for organising newly developed grammars in declarative libraries. Section 3.2.3 introduces a tool that can be used to create CLIMB libraries that interact with a large grammars developed using the traditional (non-CLIMB) method. The latter can also be used to switch from the traditional grammar engineering approach to declarative CLIMB. Both approaches have been implemented and the correctness of the algorithms have been verified through small specific tests on gCLIMB and Jacy, but they have not been used to actually develop grammars with alternative analyses at this point. Because they are easier to work with then standard CLIMB, it is expected that they can also be used to apply CLIMB to long term grammar development. This section ends with a brief summary and discussion of declarative CLIMB and a presentation of future work.

3.2.1 An introduction to declarative CLIMB

Declarative CLIMB allows users to write a metagrammar in TDL. This has several advantages over the original CLIMB approach explained above. First, the grammar engineer does not need to learn procedural programming in

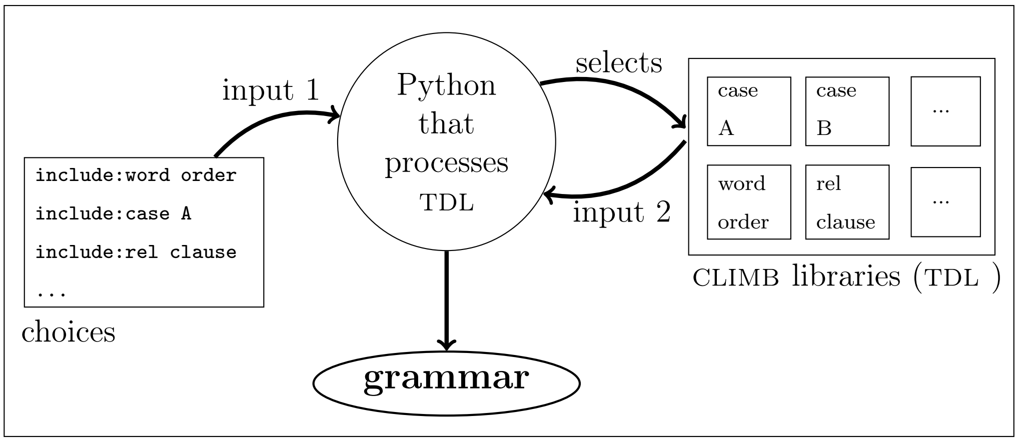


Figure 3.6: Schematic overview of declarative CLIMB

Python. Because the metagrammar is defined in TDL, it is easier to implement the metagrammar directly. This avoids the two step procedure of first manually extending one or more of the generated grammars and then adapting the metagrammar. The only additional step in writing a declarative metagrammar in TDL compared to writing a grammar in TDL is related to the organisation in libraries. I believe that most grammar writers would agree that this requirement to organise the grammar can only be seen as an advantage of the approach. The CLIMB software includes an algorithm that allows users to abbreviate paths in type definitions, which will be explained in detail in Section 3.3. This feature makes writing TDL statements in CLIMB less cumbersome than writing them directly in the grammar. Overall, declarative CLIMB significantly lowers the hurdle to use CLIMB, though it does not offer the full flexibility of the original CLIMB approach.

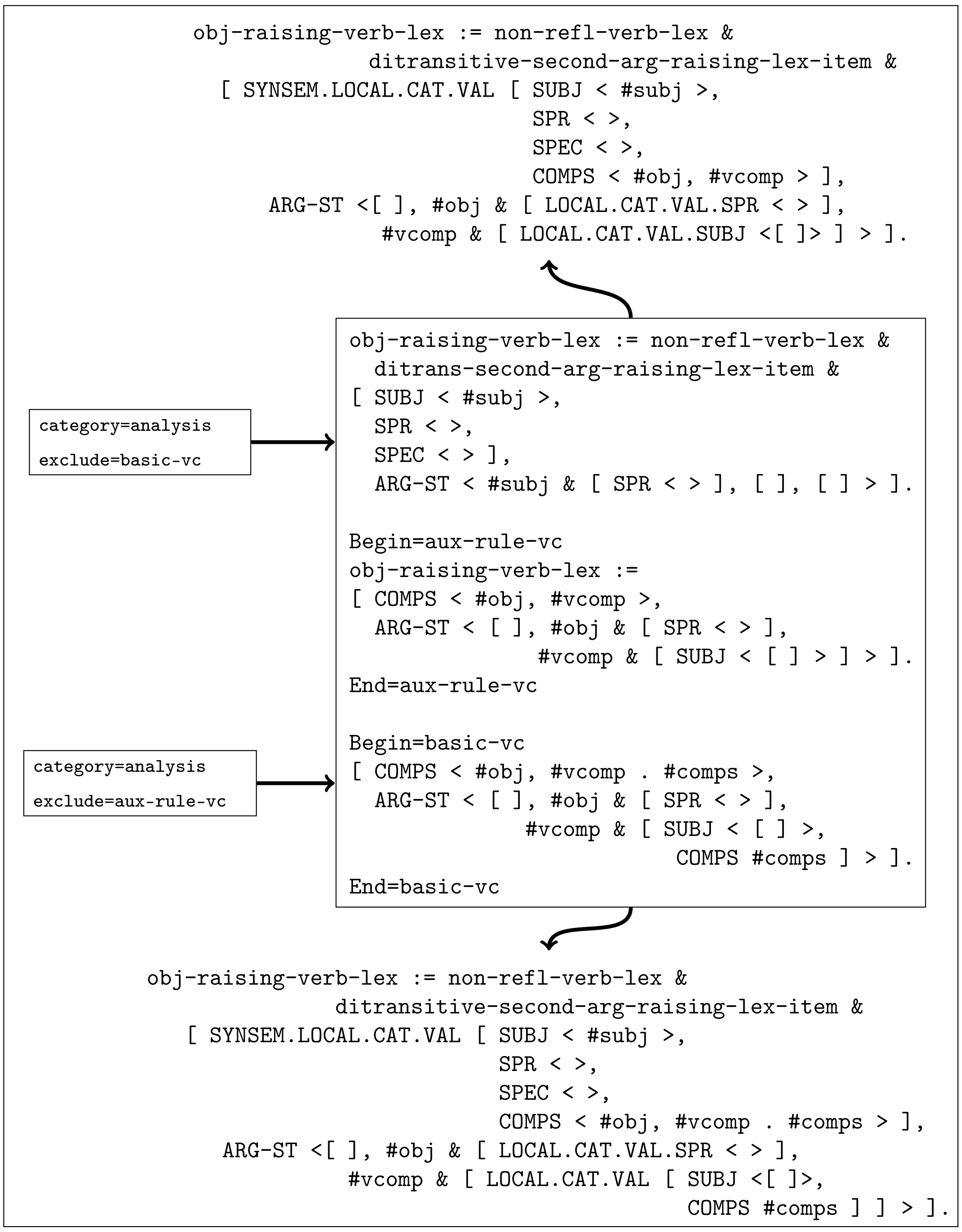
3.2.2 Declarative libraries

Figure 3.6 provides a schematic overview of declarative CLIMB. As illustrated, the original customisation part of CLIMB (cf. Figure 3.1, p.74) is split up in two components: a component that takes care of TDL processing written in Python and a component containing the actual libraries, mostly written in TDL. The Python implementations can be treated like a black box: the grammar engineer does not need to interact with them at any time. Declarative CLIMB can optionally take a choices file as input. If no *choices* are provided, all analyses in the metagrammar will be included in the generated grammar. This option can be useful for engineers who are not testing alternative analyses, but would like to use declarative CLIMB to structure their grammar differently or use abbreviated paths. Figure 3.7 shows the implementation of a basic type for object raising in declarative CLIMB. The boxes on the left of the image present two snippets of alternative *choices*. The large box provides sample code of an implementation in declarative CLIMB. The type definitions on top and on the bottom of the image indicate the output for the upper and, respectively, bottom *choices*.

The user can define which analyses should be excluded in the grammar in *choices*. The definitions in *choices* are thus direct statements about implementations (i.e. statements about chunks of code to include in the grammars) rather than statements about linguistic properties. For instance, *choices* in procedural CLIMB may contain a definition stating `word-order=v2` or `person=1-2-3`, indicating verb second word order or that the language distinguishes first, second and third person. In declarative CLIMB, one or more implementations to capture the phenomenon will be included in the metagrammar, where it is associated with an identifier which could be similar to the definitions in *choices* in procedural CLIMB (e.g. `filler-gap-v2` or `person-123`). *Choices* states whether these analyses should be included in the grammar.[6] Ideally, these two (implementations and linguistic properties) coincide and *choices* contains a list of phenomena and specific analyses that should be excluded from the resulting grammar. Each definition in *choices* points to a specific implementation in the metagrammar. This means that the metagrammar is best organised according to phenomena and their analyses rather than according to specific types as seen in the resulting grammar.

If a declarative metagrammar is organised according the phenomena it covers, some of the definitions in *choices* will be quite similar to definitions found in

[6]The current implementation of declarative CLIMB includes all analyses in the metagrammar, unless *choices* specifically states the analysis should not be included.



```
obj-raising-verb-lex := non-refl-verb-lex &
              ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >,
                           SPR < >,
                           SPEC < >,
                           COMPS < #obj, #vcomp > ],
      ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL.SUBJ <[ ]> ] > ].
```

```
category=analysis
exclude=basic-vc
```

```
category=analysis
exclude=aux-rule-vc
```

```
obj-raising-verb-lex := non-refl-verb-lex &
  ditrans-second-arg-raising-lex-item &
[ SUBJ < #subj >,
  SPR < >,
  SPEC < > ],
  ARG-ST < #subj & [ SPR < > ], [ ], [ ] > ].

Begin=aux-rule-vc
obj-raising-verb-lex :=
[ COMPS < #obj, #vcomp >,
  ARG-ST < [ ], #obj & [ SPR < > ],
               #vcomp & [ SUBJ < [ ] > ] > ].
End=aux-rule-vc

Begin=basic-vc
[ COMPS < #obj, #vcomp . #comps >,
  ARG-ST < [ ], #obj & [ SPR < > ],
              #vcomp & [ SUBJ < [ ] >,
                         COMPS #comps ] > ].
End=basic-vc
```

```
obj-raising-verb-lex := non-refl-verb-lex &
              ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >,
                           SPR < >,
                           SPEC < >,
                           COMPS < #obj, #vcomp . #comps > ],
      ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL [ SUBJ <[ ]>,
                                          COMPS #comps ] ] > ].
```

Figure 3.7: Snippet of declarative CLIMB code with alternative choices and their output

choices for procedural CLIMB, as shown above. However, statements about lexical items and morphosyntactic properties radically differ. In procedural CLIMB, lexica are defined by detailed lists of properties for each lexical type included in the grammar. Each morphosyntactic rule also has its own definition including detailed information on the features to include in *choices*. This functionality of defining a large lexicon and morphosyntactic rules in *choices* requires manipulation of typehierarchies and the feature geometry by libraries written in Python. It is therefore not supported in declarative CLIMB at this point; the possibilities when using declarative CLIMB are limited to including or excluding a definition.

Functionality and workflow of declarative CLIMB

The workflow of declarative CLIMB is closer to traditional grammar engineering compared to procedural CLIMB. Since the implementations are written in TDL, analyses can be integrated directly in the metagrammar, even if the CLIMB approach is new to the grammar engineer.

The differences between directly implementing types in a grammar and implementing them in a declarative CLIMB metagrammar are the following:

1. Types can be defined partially at different locations in the metagrammar in declarative CLIMB. This differs from adding definitions to a previously defined type using addenda,[7] because CLIMB will merge all partial definitions into one type in the resulting grammar.

2. Types in declarative CLIMB may contain abbreviated paths. Though it is possible to fully define each path, using abbreviated paths may have advantages when planning minor changes to feature geometry (see Section 3.3.2 for further details).

3. The metagrammar may contain contradictory type definitions and inconsistencies. Of course, the grammar engineer should make sure that

[7]See Section 2.2.1 for a description of addenda.

choices prevents contradictory definitions from being included in the same grammar.

Note that all three properties mentioned above can be used optionally. It is also possible to define all types at one location only, use full paths at all times and include only types that should be included in the generated grammar. In other words, a declarative CLIMB metagrammar can be identical to a traditionally implemented grammar, though there is no reason to use declarative CLIMB and not make use of any of the possibilities it offers.

The workflow for using declarative CLIMB is as follows. First, the analyses are included (directly) in the metagrammar. Declarative CLIMB is used to generate a grammar by calling the associated Python script from the command line. The grammar engineer can run new tests and regression tests, which may lead to revisions in the metagrammar. When the debugging and testing phase is finished, statements that identify the new implementations can be added to the metagrammar. New libraries can either be placed in their own file or indicated by `Library=name-of-library` in the metagrammar. Analyses that do not form a library of their own can be marked by `Begin=name-of-analysis` and ended by `End=name-of-analysis` or by a statement that another analysis begins. There is no fundamental difference between libraries and analyses in declarative CLIMB, though libraries generally form a complete set of grammatical properties dealing with a phenomenon (e.g. a library for all nominal forms in a language) and analyses tend to deal with more specific grammatical properties (e.g. an analysis for reflexive pronouns). A library can contain multiple analyses. It is thus possible to include a library but exclude some of the analyses it contains. On the other hand, if a library is excluded, all analyses it contains will be excluded as well.

The final step in grammar development with declarative CLIMB consists in documenting the new analyses and their associated choices. This particularly holds when a new analysis is incompatible with certain other analyses in the metagrammar.

Alternative analyses in declarative CLIMB

Declarative CLIMB provides several methods to include alternative accounts. The most straightforward option is to create individual libraries for alternative analyses. Libraries can be organised in separate files or by statements indicating the start of a new library within a metagrammar file. A metagrammar for German could for instance include two libraries for verb second word order, one that follows the traditional HPSG filler-gap analysis and one that follows the Wambaya based flat analysis. In this case, *choices* selects alternative analyses by including or excluding specific libraries.

It may not always be the most logical choice to place all properties associated with an analysis in an associated library. This particularly applies to small sets of constraints that ensure alternative analyses interact correctly with other analyses. For instance, the flat word order analysis for verb second languages uses features that are not included in the grammar when a filler-gap analysis is used. Passivisation rules must pass the value of these features up, but only if the flat analysis is selected. The software also supports statements that can associate small chunks that may consist of individual TDL definitions with a particular option in *choices* to facilitate the implementation of such interactions.

Finally, type names can be used to indicate an association with an analysis. The user can indicate that a given type is related to a particular analysis by a specific affix in the type name. *Choices* can state that all types with a specific affix should be excluded from the grammar if the associated analysis is not selected. Likewise, *choices* will include type names with affixes related to the selected alternative analyses. Note that this approach can only include or exclude complete type definitions. It is therefore not suitable for analyses that touch many parts of the grammar.[8] Rare phenomena that should not

[8]One way to work around this would be to use general types containing all properties that are shared across analyses and use specific subtypes that only assign analysis specific properties. However, every analysis-specific subtype is likely to be the supertype of at least one other type, whose definition will need to be adapted as well (since the name of this supertype changes according to the selected analyses). Hence, other methods of modifying types provided by declarative CLIMB are preferable in this case.

always be included in the grammar form an example of where such an approach would typically work. For instance, gCLIMB includes an analysis that accounts for a form of partial VP fronting, where part of the verbal cluster is fronted to the Vorfeld and other verbs remain in the Right Bracket, cf. Section 4.4.2. The phenomenon is quite rare and its analysis consists of a rule that introduces additional ambiguity that can only be resolved after the entire sentence is processed. Because the analysis uses additional types and does not introduce additional constraints on existing types, it can easily be identified by the type names associated to the analysis.

The idea that we can use affixes to select analyses has its origins in a DELPH-IN discussion about how to identify analyses for specific phenomena in the grammar. One of the options that was uttered during this discussion was to use type names to index phenomena.[9] Using type names to identify specific analyses thus does not only provide an indication that CLIMB can use to include or exclude a type from the grammar, but also offers a way for other grammar engineers to find types associated with a specific analysis in the resulting grammar.

Components of declarative CLIMB

Declarative CLIMB consists of the following implementations:

1. A script called *create_grammar.py.* This script is called to generate a grammar based on a *choices* file.
2. TDL processing files that take care of merging type definitions and path completion. They are:
   (a) *tdl.py.* This script takes care of merging type definitions and completing paths. It largely corresponds to the tdl processing file included in the Grammar Matrix but was extended to support path completion.

[9]This discussion took place 26 June 2011 in Suquamish, USA (`http://moin.delph-in.net/SuquamishGrammarIndexing`)

(b) *abbreviate_paths_in_climb.py.* This script can abbreviate paths automatically, but also interprets the feature geometry so it may be used for path completion.

(c) *extract_feature_geometry.py.* This script can extract the feature geometry from a grammar.

The grammar engineer needs to provide the following components to create a grammar with declarative CLIMB:

1. A feature geometry which can either be extracted from the grammar automatically using *extract_feature_geometry.py* or defined manually. It indicates where in the feature structure certain attributes may be situated.

2. A file indicating the supertypes of a specific type, if it is not a subtype of *sign*. This information is necessary for path completion.

3. A set of libraries (optionally) containing alternative analyses. The analyses are written in TDL and abbreviated paths may be used.

I will give a short description of the algorithm used to create the grammar. Section 3.3.2 describes the path completion algorithm. First, the algorithm goes through *choices* and collects all TDL files, libraries, chunks or analyses that should be excluded from the grammar. It then goes through the TDL files of declarative CLIMB that are not explicitly excluded by *choices.* If the script finds a statement indicating the beginning of a library or a chunk, it checks whether the library or chunk in question is to be excluded from the grammar according to *choices.* If this is not the case, it collects the type definitions from the metagrammar. For each type, the script verifies whether it has a suffix that is marked for exclusion. If no such suffix is identified, the definition is processed using an extended version of the TDL processing code from the Grammar Matrix and added to the type hierarchy of the grammar. This version takes type definitions with abbreviated paths as input and completes their paths using the feature-geometry and the super-subtype chain of the type in question. Like the original TDL processing algorithm, it merges definitions added to types that are already defined elsewhere. Once all TDL files are processed, the type hierarchy is printed out to a set of TDL files which can be used as a regular DELPH-IN grammar.

Setting up declarative CLIMB

The tools to use declarative CLIMB have been integrated in gCLIMB. When generating a grammar with gCLIMB, an initial setup to continue grammar development with declarative CLIMB is included. The output of gCLIMB includes a folder called *climb* in addition to the grammar, which it generates as before. The *climb* folder contains a file declaring the feature geometry of the created grammar, the declarative CLIMB processing tools including feature geometry extraction, path abbreviation and completion and a set of declarative CLIMB libraries.

The declarative libraries are organised according to the architecture of procedural gCLIMB, i.e. each library in procedural CLIMB places the analyses it generates in a corresponding declarative library. All paths in the declarative library have been abbreviated using the feature geometry extraction and path abbreviation algorithms described in Section 3.3. The output of gCLIMB has been used to test whether declarative CLIMB works properly. The algorithms included in declarative CLIMB were used to recreate grammars previously created using procedural CLIMB.

It is also possible to start using declarative CLIMB for grammars that have been developed the traditional way. Either part of the grammar can (temporarily or permanently) be stored in a metagrammar, or the complete grammar can be converted stepwise. The tools to use declarative CLIMB on large coverage grammars are presented in the next subsection.

3.2.3 Large coverage grammars and CLIMB

Introduction to SHORT-CLIMB

The CLIMB workflow provided above describes how the CLIMB approach may be used in new grammar development projects. The originally proposed version of CLIMB requires the entire grammar to be included into a meta-grammar of Python libraries. In order to get the most out of the approach, it should indeed be adopted from the first steps of grammar development. This does not necessarily mean that CLIMB can only be useful for new grammars, or grammars in an early stage of development. This section describes SHORT-CLIMB (Starting High On the Roof Top-CLIMB), a tool that has been developed to provide support for developers working on larger resource grammars (Fokkens, 2012).[10] The typical scenario for using SHORT-CLIMB would be a major revision to a large grammar. SHORT-CLIMB allows the engineer to flip back and forth between the old and revised version of the grammar until a final decision is made. The tool is available under the MIT license.[11]

The main idea behind CLIMB for large grammars is that grammar developers can create libraries for alternative analyses without converting the entire grammar into CLIMB. When using the approach, the engineer defines a set of changes (adding, removing or modifying type definitions) in a new library. This library also defines where (near which type in the TDL file) changes must be made in the grammar. SHORT-CLIMB takes the original grammar and the library as input and creates a new grammar that has been adapted according to the definitions in the library. It can also create a mirror library that reverses all changes as additional output. I will elaborate on the properties of SHORT-CLIMB below. A more elaborate guide to using SHORT-CLIMB can be found in the documentation.[12]

[10]`http://moin.delph-in.net/ClimbShortClimb`

[11]SHORT-CLIMB is available at `svn://lemur.ling.washington.edu/shared/matrix/branches/antske-germanic-development/short_climb.tar.gz`.

[12]`http://moin.delph-in.net/ClimbShortClimb`

Modifications

SHORT-CLIMB provides three kinds of modifications to the grammar: type removal, type insertion and type replacement. Removals are defined by the statement `removal=type_name`, where `type_name` stands for the type to be removed. The specification `,addendum` indicates that the original type definition should be maintained and only addenda (statement using operator :+ to add properties to a type defined elsewhere, cf. Section 2.2.1) are to be removed. Without this specification, the original statement and addenda (if present) will be removed from the grammar.

When new type definitions are to be included, the engineer simply adds their type definition in TDL to the library, immediately preceded by the statement `location=type_name`, where `type_name` indicates the location of the new type definition in the grammar in the TDL file. The type will be inserted above the type defined as `location`. Types that are to be removed can also be used as location, in which case the new definition will occur at the location of the removed type.

Type modifications can be defined in several ways. Additional constraints on a type can be inserted by including a (partial) type definition in the library. The original type and new constraints will be merged by the function `add` explained in Section 2.3.2. It must be noted that additions to type definitions should use operator `:=` rather than the addendum operator :+. When the operator :+ is used and the original grammar does not contain an addendum to the type already, the statement will be ignored. New addenda are inserted into the grammar when preceded by a `location=` statement only.

Properties can be removed from types by using a new operator :− introduced for this purpose. SHORT-CLIMB compares the type definition following :− in the library to the definitions following `:=` or :+ for the same type in the original grammar. A new type definition is created including the original definitions minus those specified to be removed by :−. Lists form a special case in this process. It is not possible to remove an element completely from a list. Properties of elements on a list, on the other hand, can be removed using :−. Examples are the type of an element on the list, or a constraint defined on a lists element. If all properties of an element on the list are removed, an underspecified element (defined by [ ]) will be found in its place.

The final way to change a type definition using SHORT-CLIMB is to provide the complete new definition of the type preceded by the statement `complete=on`. This statement will remove the original definition from the grammar and replace it by the new definition from the library. This method can be used to change the length of a list or difference list.

The algorithm used in SHORT-CLIMB is similar to the one used for declarative CLIMB. First, the algorithm goes through the modification files and collects additions, subtractions, removals and substitutions defined as described above. The algorithm then goes through the TDL files of the grammar and checks for each type whether it falls in one of the categories above. If it does, the type is modified or removed according to the indication. The algorithm also checks whether the type is mentioned as the location for a new type. In this case, it adds the new type first. If the original type is not indicated for removal, it will be added after the inserted type, either in its original version if no modifications applied or in a modified form.

Users can also request the inverse modification file when running SHORT-CLIMB. The file can be used to change the grammar back in its original state, the definitions of removed types as well as completely modified types are stored during the modification process. They are added to the inverse file as additions or modifications, respectively. Partial modifications are included by changing :+ into :− and vice versa.

I tested the algorithm and both outputs described above by making various kind of changes to Jacy using SHORT-CLIMB. These changes included removing types, adding them and changing various properties of types. They were selected to verify different parts of the algorithm, but were not related to a specific phenomenon. After running SHORT-CLIMB, the newly created grammar was inspected to see if modifications were attributed as intended. The tests also included flipping back and forth between two versions using the automatically generated file with inverted modifications and testing whether the grammar was indeed reestablished in its original state.

Starting a metagrammar with SHORT-CLIMB

SHORT-CLIMB can also be used to start to convert the structure of the grammar to a metagrammar setting.[13] As before, the grammar engineer defines a library of changes and SHORT-CLIMB modifies the grammar accordingly. Additionally, SHORT-CLIMB can create a reduced version of the grammar that no longer includes types stated to be removed, constraints and definitions removed by operator :−, or types up for complete revision (indicated by `complete=on`). The reduced grammar is accompanied by two new libraries. The first can be used to create the new version of the grammar. It contains the additions defined in the original modification library as well as all definitions meant to completely replace an old type. The second is meant to create the original version of the grammar. It includes type definitions of types removed by the revision library, removed constraints as additional constraints (replacing the :− operator by := or :+) and the original definition of types marked for complete substitution. The algorithm created this reduced grammar was also tested by making various changes to Jacy.

The reduced grammar forms the new core of the metagrammar, meaning that its definitions will all be included in the grammar regardless of what libraries and properties are selected. The newly created libraries both contain a set of definitions directly related to a specific analysis for a phenomenon. From this starting point, more libraries can be created. When SHORT-CLIMB is run, implementations related to a specific analysis will be removed from the core and placed in a library. The grammar engineer can remove an analysis for a specific phenomenon without creating an alternative by simply providing a set of removal statements (which can be either complete types or properties of types). This strategy could be of interest to developers of a small to medium-sized grammar considering whether to switch over to declarative CLIMB.

[13]See `http://moin.delph-in.net/ClimbShortClimb` for a more elaborate explanation of how to use SHORT-CLIMB.

Adjective forms following one of the following words (in any form): *der*, *dieser*, *jener*, *jeder*, *mancher*, *solcher*, *welcher*, *aller*, *sämtlicher*, *beide* (weak inflection)			
case	masculine	feminine	neutral
nom	der gut**e** Wein	die gut**e** Milch	das gut**e** Bier
gen	des gut**en** Wein(e)s	der gut**en** Milch	des gut**en** Bier(e)s
dat	dem gut**en** Wein	der gut**en** Milch	dem gut**en** Bier
acc	den gut**en** Wein	die gut**e** Milch	das gut**e** Bier
Adjective forms following one of the following words (in any form): *ein*, *kein*, *mein*, *dein*, *sein*, *ihr*, *unser*, *euer*, *Ihr* (mixed inflection)			
case	masculine	feminine	neutral
nom	ein gut**er** Wein	eine gut**e** Milch	ein gut**es** Bier
gen	eines gut**en** Wein(e)s	einer gut**en** Milch	eines gut**en** Bier(e)s
dat	einem gut**en** Wein	einer gut**en** Milch	einem gut**en** Bier
acc	einen gut**en** Wein	eine gut**e** Milch	ein gut**es** Bier
Adjective forms not preceded by a specifier (strong inflection)			
case	masculine	feminine	neutral
nom	gut**er** Wein	gut**e** Milch	gut**es** Bier
gen	gut**en** Wein(e)s	gut**er** Milch	gut**en** Bier(e)s
dat	gut**em** Wein	gut**er** Milch	gut**em** Bier
acc	gut**en** Wein	gut**e** Milch	gut**es** Bier
Adjective endings for plural adjectives			
case	"*der*" specifier	"*ein*" specifier	no specifier
nom	die gut**en** Weinen	keine gut**en** Weinen	gut**e** Weinen
gen	der gut**en** Weinen	keiner gut**en** Weinen	gut**er** Weinen
dat	den gut**en** Weinen	keinen gut**en** Weinen	gut**en** Weinen
acc	die gut**en** Weinen	keine gut**en** Weinen	gut**e** Weinen

Table 3.1: Overview of German adjective endings

3.2.4 Discussion of declarative CLIMB

This section introduced declarative CLIMB, an alternative version of CLIMB that lowers the hurdle for adopting the approach. A disadvantage of declarative CLIMB is that it does not support a number of useful functions from the customisation system. This mainly concerns iterative functions creating lexical items and morphosyntactic rules and implementations which make use of procedural functions in Python. I will illustrate the difference through the example of the aforementioned implementation of adjectives. Table 3.1

provides an overview of German adjectives.[14]

```
adj_strength = ‘’
adj_strength = self.get(‘strength-marking’)
if adj_strength:
 if adj_strength != ‘none’:
  features += [[‘strong’, ‘bool|bool’, ‘LOCAL.CAT.HEAD.STRONG’]]
  features += [[‘strong’, ‘-|-’, ‘LOCAL.CAT.HEAD.STRONG’]]
  features += [[‘strong’, ‘+|+’, ‘LOCAL.CAT.HEAD.STRONG’]]
  if adj_strength == ‘triple’:
   features += [[‘strong’, ‘luk|luk’, ‘LOCAL.CAT.HEAD.STRONG’]]
    features += [[‘strong’, ‘na-or-|na-or-’, ‘LOCAL.CAT.HEAD.STRONG’]]
   features += [[‘strong’, ‘na-or-+|na-or-+’, ‘LOCAL.CAT.HEAD.STRONG’]]
   features += [[‘strong’, ‘na|na’, ‘LOCAL.CAT.HEAD.STRONG’]]
```

Figure 3.8: Code snippet to add the feature path and values for STRONG

Implementing generic types for adjectives and rules for combining adjectives and nouns is similar in declarative CLIMB and procedural CLIMB. I will therefore focus on the implementation of specific types for adjective and morphological rules where differences are most apparent. Figures 3.8 and 3.9 illustrate the basic implementations that were added to gCLIMB for capturing German adjectives. German adjective endings depend on the determiner that is present in the noun phrase (or on the determiner being absent). The feature STRONG is used to ensure the adjective has the correct ending.[15] The path to this feature, its possible values and how these can be triggered in *choices* are defined in *choices.py*. The definitions are illustrated in Figure 3.8.

[14]Paradigms such as the one in Table 3.1 can be found in many German grammar books and online information about German grammar. In linguistic literature, information about adjective endings can also be found. For instance, Müller (2007) provides an overview of adjective endings and all grammatical properties they may have. The endings in Table 3.1 have been verified against his description (Müller, 2007, p. 214). Note however that Müller (2007) refers to strong and weak determiners rather than weak and strong inflection. Strong determiners are followed by adjectives with weak inflection and vice versa.

[15]The current implementations of the customisation system and CLIMB do take the type hierarchy into account while interpreting possible values for *choices*. Therefore all possible values including subtypes must be defined explicitly.

```
for adj in ch.get('adj',[]):
  name = get_name(adj)

  features.customize_feature_values(lang, ch, hiers, adj, atype, 'adj')

  for stem in adj.get('stem',[]):
    orth = stem.get('orth')
    orthstr = orth_encode(orth)
    pred = stem.get('pred')
    id = stem.get('name')
    id = id.replace(' ','_')
    typedef = \
     TDLencode(id) + ' := ' + atype + ' & \
       [ STEM < "' + orthstr + '" >, \
         SYNSEM.LKEYS.KEYREL.PRED "' + pred + '" ].'
    lexicon.add(typedef)
```

Figure 3.9: Code snippet for generating adjectives

The implementation presented in Figure 3.9 is responsible for generating lexical types for adjectives and adding lexical entries to the lexicon. The complete implementation for adjectives also defines more general types of adjectives which form the supertypes of the lexical types created by the code in Figure 3.9. These implementations are similar to those that would be used in declarative CLIMB, as mentioned above. Other requirements for including adjectives and their morphology in the grammar are met by generic functions in CLIMB. All other properties for including German adjectives in the grammar can be defined in *choices*. A sample of the required choices is provided in Figure 3.10.

Additional adjectives and adjective endings can be added rapidly in this implementation. The generic morphotactic and feature geometry implementations in CLIMB take care of the rest. It should furthermore be noted that it is easy to create grammars with different analyses for adjective endings. As Table 3.1 shows, several adjective endings can stand for more than one case, number, gender and "strength" combination. The *choices* in Figure 3.10 define two morphological rules. One rule assigns mixed strength, dative or

```
adj1_name=intersective
adj1_kind=int
adj1_arg-str=none
adj1_stem1_orth=riesig
adj1_stem1_pred=_riesig_m_rel
adj1_stem2_orth=herrlich
adj1_stem2_pred=_herrlich_mod_rel
adj2_name=scopal
adj2_kind=scop
adj2_arg-str=none
adj2_stem1_orth=unglaublich
adj2_stem1_pred=_unglaublich_m_rel
adj-pc1_name=adj-inflection
adj-pc1_obligatory=on
adj-pc1_order=suffix
adj-pc1_inputs=adj1, adj-pc2_lrt1
adj-pc1_lrt1_name=mix-weak-datgen
adj-pc1_lrt1_feat1_name=case
adj-pc1_lrt1_feat1_value=dat, gen
adj-pc1_lrt1_feat2_name=strong
adj-pc1_lrt1_feat2_value=na-or-
adj-pc1_lrt1_feat3_name=prd
adj-pc1_lrt1_feat3_value=-
adj-pc1_lrt1_lri1_inflecting=yes
adj-pc1_lrt1_lri1_orth=en
```

```
adj-pc1_lrt2_name=mixed-weak-nomacc-pl
adj-pc1_lrt2_feat1_name=case
adj-pc1_lrt2_feat1_value=nom, acc
adj-pc1_lrt2_feat2_name=strong
adj-pc1_lrt2_feat2_value=na-or-
adj-pc1_lrt2_feat3_name=number
adj-pc1_lrt2_feat3_value=plural
adj-pc1_lrt2_feat3_head=mod
adj-pc1_lrt2_feat4_name=prd
adj-pc1_lrt2_feat4_value=-
adj-pc1_lrt2_lri1_inflecting=yes
adj-pc1_lrt2_lri1_orth=en
```

Figure 3.10: Sample of choices defining adjectives and their morphology

genitive case and any gender or number and the other assigns mixed strength, nominative and accusative plural with the same ending. A few changes in *choices* would result in a single rule for mixed strength and plural endings, and one for dative and genitive in singular. The procedural implementation thus offers a lot of flexibility for experimenting with alternative analyses.

The equivalent rules in declarative CLIMB are represented in Figure 3.11. It is relatively easy to extend the grammar based on these two types to cover all adjective endings for German. The workload is comparable to extending *choices* in Figure 3.10. However, in order to add a rule for adjectives that can modify a noun in nominative, dative or accusative case, for instance, the case hierarchy of the grammar needs to be checked and possibly revised in order to make sure that the required type is included. This is not necessary in procedural CLIMB where hierarchies are created automatically based on

```
mixed-datgen-lrule :=  infl-lex-rule & adj-inflection-lex-rule-super &
  [ HEAD [ CASE gen+dat,
                STRONG na-or--,
                 PRD - ] ].

mixed-nomacc-pl-lrule := infl-lex-rule & adj-inflection-lex-rule-super &
  [ HEAD [ CASE nom+acc,
                STRONG na-or--,
                 MOD.NUM plural,
                 PRD - ] ].
```

Figure 3.11: Two morphotactic rules for adjectives in declarative CLIMB

properties of lexical items and morphological rules. Declarative CLIMB is thus less flexible. Moreover, constraints that make sure that obligatory morphological rules apply, that rules do not apply twice and that they interact correctly with optional rules such as comparative or superlative marking need to be created by hand for each new lexical category in declarative CLIMB. Procedural CLIMB has general rules in place that can be used directly for this purpose. The presence of general software for creating type hierarchies, paths for features and interaction of morphological rules played an important role in the rapid inclusion (six hours) of adjectives and their marking in gCLIMB. It is unlikely that this could be achieved in the same time while using declarative CLIMB.

This result indicates that procedural functions in CLIMB can help to speed up grammar development and provides additional flexibility that is not found in declarative CLIMB. On the other hand, it is easy for grammar engineers to learn how to use declarative CLIMB that are trained in writing grammars with TDL. This is an advantage that should not be underestimated. Future work will therefore focus on combining the strengths of the two approaches. In particular, this will involve the development of a tool where grammar engineers can define parameters and iterative functions with help of a GUI.

3.3 TDL processing tools

This section describes a set of TDL processing tools that have been developed as part of research related to CLIMB: The spring cleaning algorithm (Fokkens et al., 2011) and path completion. Spring cleaning was originally developed to help and compare two similar versions of a grammar in order to find which differences should be included in the metagrammar. Path abbreviation is used in declarative CLIMB and simplifies the introduction of minor changes in the feature geometry.

3.3.1 Spring cleaning

The spring cleaning[16] algorithm (Fokkens et al., 2011) identifies portions of the grammar that do not influence its competence in any way. The competence of the grammar is defined as the set of analyses that it provides for any possible input strings; for strings the grammar does not recognise, this will be the empty set. Artefacts that do not effect competence can accrue in a grammar because abandoned analyses are not completely removed or because the grammar is built on a crosslinguistic resource but does not use all of the infrastructure that resource provides. The spring cleaning algorithm takes a grammar as input and returns an equivalent grammar from which all extraneous types have been removed.[17]

As explained in the Section 2.2, the DELPH-IN processors take instances as starting points for parsing and generation. Instances thus have a direct influence on the competence of the grammar. There are several ways in which types may be **relevant**, i.e. play a role in the definition and functioning of instances. The first and most direct way is if a type is an **instantiated type**, meaning that it either has an instance or has a subtype that has an instance. Instantiated types thus directly define the instances of the grammar. The

[16]The spring cleaning algorithm was originally introduced in Fokkens et al. (2011) which was joint work with Yi Zhang and Emily Bender. The text of this section is mostly taken from there.

[17]Note that the grammar may still contain extraneous features.

second way a type has an impact on instances is by defining **features** or **values** that are part of the definition of one or more instantiated types. Finally, a type may be relevant to instances, because it defines a **lower bound** between two relevant types. In this case, the type influences the possibility of unification, i.e. if a type t forms the only lower bound between types t_1 and t_2, t permits unification between t_1 and t_2, which can no longer take place if t is removed from the grammar.[18]

Types that do not influence instances in any of the three possible ways defined above do not have an impact on the competence of the grammar. The 'spring cleaning' algorithm goes through the type definition files and identifies such irrelevant types. This process takes place in two stages. In the first stage, it identifies types that are necessary for the definition of the instantiated types. This means that they either (i) are instantiated types, (ii) define values of features of instantiated types, or (iii) introduce attributes used to define an instantiated type. The correctness of this stage of the algorithm can be verified during **compile time** of the grammar: if a type is removed that is required for the definition of an instantiated rule or lexical entry, the grammar will not load in LKB. Similarly, the PET 'flop' command, used to compile the grammar, will fail (and report an error) on a grammar missing such a type.

Types that permit unification between other types, but play no direct role to another type's definitions, are only relevant at **runtime**. These types are identified in the second stage of the process. After separating types needed to define instantiated types from those that are not, the algorithm checks for relevance at runtime of the latter types. It creates a hierarchy h of types that lead to instances and their values. For each type from the original grammar that is not defined in h, the algorithm checks whether it permits the unification of types from h that would otherwise fail.

The original spring cleaning approach verified whether a type t_1 that was marked for removal was a common subtype in the complete grammar of

[18]Recall that the DELPH-IN joint reference formalism adopts a closed-world assumption, i.e. all types that play a role in the grammar must be explicitly defined.

types t_2 and t_3 that were included in the reduced grammar. If this was the case and the reduced grammar did not include any other common subtype for types t_2 and t_3, t_1 would be included in the grammar. If the reduced grammar already contained a common subtype of t_2 and t_3, t_1 would be removed.

The implementation of spring cleaning has been modified based on a closer study of the greater lower bound (glb) completion algorithms included in LKB and PET. As it turns out, ensuring that two types that have at least one common subtype in the original grammar also have at least one common subtype in the reduced grammar does not guarantee that the reduced grammar has the same competence as the original. If the original grammar had more than one subtype and the reduced grammar has exactly one subtype, possibilities of unifying can differ, because constraints are ignored in the glb completion algorithms. Figure 3.12 illustrates the difference of a simple type hierarchy as defined (left) and after glb completion has applied (right).

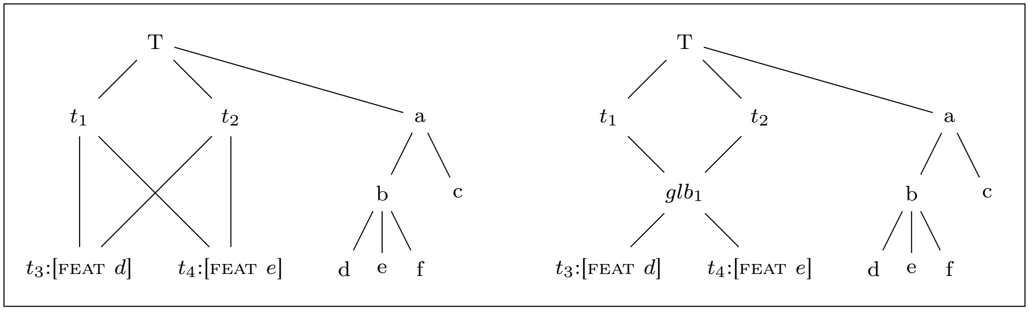


Figure 3.12: Illustration of glb completion when subtypes have features

Consider Figure 3.12: the type glb_1, introduced by the glb completion algorithm does not bear any constraints. This means that t_1 and t_2 can unify if one of these types bear the feature value [FEAT c] or [FEAT f]. If the reduced grammar contains only t_3 or t_4 this is not the case: then t_1 and t_2 may only bear [FEAT d] or [FEAT f], respectively. Note that it is not evident that features are ignored during glb completion. One may equally well introduce a common supertype of features d or e as feature value of FEAT in the grammar, or assign its greater upper bound (in this case the value b).

The current implementation of the spring cleaning algorithm checks how many common subtypes two types had in the original grammar and how many remain in the reduced grammar. If these types had more than one in the original grammar and none in the new grammar, it introduces a new glb instead of reintroducing a random common subtype originally marked for removal. If two types only have one common subtype in the reduced grammar, the algorithm checks whether this type has features and if so, a glb without features is introduced. This glb becomes the new supertype of their joint common subtype. This addition is necessary, because the original grammar had a glb which also did not have features due to the way the glb completion algorithm works. If no additional type without features is added, the reduced grammar would thus be more constrained. The decision to introduce a new glb rather than leaving the original types in the grammar was taken, because there may be cases where there may be several original types with complex definitions. Leaving them all in the grammar may misleadingly give the impression they play a role in the grammar. Introducing a special glb type provides better insight into the structure of the original grammar and leads to a cleaner grammar.

The final output of this algorithm is three sets of files: (i) modified versions of the files of the original grammar, in which the definition files include only those types that are either needed to define instances or identified as relevant at runtime, (ii) copies of the original definition files and (iii) files that list removed types for each definition file. The structure of the original grammar is preserved by this process. This includes types that are used to capture generalisations so that a given constraint need be stated in only on place. Thus the result is a grammar that is 'cleaned up' with respect to the input without being more compact in ways that might make it more opaque to the grammar engineer.[19]

[19]Fokkens et al. (2011) also present a grammar compression algorithm implemented by Yi Zhang. This algorithm indicates which types are required to create a grammar with equivalent competence. The main differences between the two algorithms are that (1) the grammar compression algorithm (currently) does not produce a reduced grammar and (2) if it did output a grammar, it would modify the structure. A more detailed explanation is given in Chapter 6 as part of the spring cleaning evaluation.

Spring cleaning can be used on any grammar in TDL. The impact is likely to be biggest on grammars written in the traditional way. Because CLIMB stimulates an organisation that clearly identifies which implementations are part of a specific analysis, chances are that pieces that belong to old analyses remain in the grammar less often. Nevertheless, spring cleaning was originally designed as part of the CLIMB approach. After a complex analysis has been integrated in a grammar manually, one way to identify which implementations should be added to the metagrammar is to generate a clean version of the grammar and compare it to the extended grammar. Spring cleaning makes sure that only relevant types of the grammar are considered, so that trials are disregarded and types that are only needed in the old analysis do not turn up in the revised analysis. In future work, spring cleaning will become part of a bigger application that automatically extracts the differences between two versions of a grammar.

3.3.2 Reduced Paths

$$\left[\begin{array}{ll}\text{SYNSEM}|\text{LOCAL}|\text{CAT}|\text{HEAD} & \boxed{1}\\ \text{HEAD-DTR}|\text{SYNSEM}|\text{LOCAL}|\text{CAT}|\text{HEAD} & \boxed{1}\end{array}\right] \quad \left[\begin{array}{ll}\text{HEAD} & \boxed{1}\\ \text{HEAD-DTR}|\text{HEAD} & \boxed{1}\end{array}\right]$$

Figure 3.13: The Head Feature Principle in full and abbreviated form

Figure 3.13 represents one attribute value matrix containing full paths (left) and one indicating the same constraint through an abbreviated path (right). Types in DELPH-IN grammars must be defined using full paths. The basic idea behind the tools described in this subsection is to let grammar writers define their types in a manner that is common practice in theoretical HPSG papers. This involves three algorithms. The **feature geometry extraction** algorithm automatically extracts the feature geometry of a grammar, i.e. it extracts where attributes are introduced (by which type) and provides chains of sub- and supertypes for types that introduce an attribute. The **path abbreviation** algorithm can automatically reduce paths to the shortest form that can still be completed deterministically. It can furthermore complete the feature geometry extraction algorithm in that it stores the default values of lists and difference lists. The **path completion** algorithm allows grammar engineers to use abbreviated paths in CLIMB.

Feature Geometry Extraction

The feature geometry extraction algorithm can identify the feature geometry of a grammar written in TDL. The algorithm reuses implementations from the spring cleaning algorithm. Feature geometry extraction first determines which type in the type hierarchy introduces an attribute. After it has listed which type introduces an attribute and what the value of this attribute is, it extracts inheritance chains for all types concerned. This information is required, because an attribute may also appear in a TDL definition as part of a constraint on a type that is a supertype or subtype of the type that introduced it.

The output of the feature geometry extraction algorithm is a file that defines for each attribute which type introduces it and the type of its values. This file furthermore contains representations of chains including super- and subtypes of all types that introduce features or are feature values.

Path Abbreviation

The path abbreviation algorithm automatically reduces the paths in type definitions. The input is a grammar where all types have fully defined types and a feature geometry file which can be defined manually or created by the feature extraction algorithm described above. The algorithm first creates two Python dictionaries from the feature geometry file. The first links every attribute defined in the grammar to the type that introduces it. The second dictionary contains types that introduce at least one attribute and links them to a list containing all attributes that may take this type as a value. This list of attributes is created using the super- subtype chains of the respective types. These two dictionaries are used to reduce paths. Figure 3.14 presents the pseudo code of the path reduction process.

```
Algorithm 3.3.1: ReducePath($fullPath, attr2itype, t2attrs$)

comment: Paths have form ATTR1.ATTR2, starting at the end.
$attributes \leftarrow fullPath.split('.').reversed()$
comment: The last attribute in the path is always included.
$new\_path \leftarrow attributes[0]$
comment:
Go through the attributes. Check preceding attribute.
The preceding attributes is the next element on the list.
for $i \leftarrow 0$ to $attributes.length() - 2$
  do
    $attr \leftarrow attributes[i]$
    $prec\_attr \leftarrow attributes[i + 1]$
    comment: Retrieve type introducing the current attribute.
    $intro\_type \leftarrow attr2itype.get(attr)$
    comment: Retrieve attributes taking intro_type as value.
    $pot\_prec\_atts \leftarrow t2attrs.get(intro\_type)$
    comment:
    If the preceding attribute is not included,
    print out a warning.
    if not $prec\_attr \in pot\_prec\_atts$
      then print “Warning: potentially ill defined path: ”
           + fullPath
    comment:
    If more than one potential preceding attributes, then
    the preceding attribute must be maintained.
      else if $pot\_prec\_atts.length() > 1$
      then $new\_path.append(prec\_attr)$
$red\_path = new\_path.reverse()$
return ($red\_path$)
```

Figure 3.14: Pseudo code for path reduction

The algorithm goes backwards through the defined path and checks whether the preceding attribute could be determined deterministically from the feature geometry. To establish this, the algorithm first retrieves which type introduces the current attribute. It then checks which attributes can take this type as their value. If there is only one such attribute and it corresponds to the attribute preceding the current attribute, then this preceding attribute need not be maintained in the reduced path. In the example in Figure 3.13 for instance, it would retrieve that HEAD is introduced by *cat* and that CAT is the only attribute in the grammar that has *cat* (or any of its subtypes) as its value. It will therefore remove CAT from the complete path.

The path abbreviation algorithm includes a special treatment for lists. If an attribute takes a list as its value, the type definition does not necessarily state what kind of elements will appear on this list. They can be anything, but in a well written grammar, a particular attribute will always have elements of compatible types on its list. For instance, SUBJ, COMPS and SPR always have a list of *synsem* as their value. When the path abbreviation algorithm encounters a list, it will try to establish the type of the elements on this list by looking at the attributes of the first element. Figure 3.15 provides the pseudo code for this algorithm.

I will illustrate how this algorithm works for a path used to indicate case marking for the second element on the COMPS list. The complete path of attributes would look as follows:

SYNSEM|LOCAL|CAT|VAL|COMPS|REST|FIRST|LOCAL|CAT|HEAD|CASE

The algorithm presented in 3.15 goes through the attributes starting at the left until it finds FIRST. It retrieves the attribute following FIRST, in this case LOCAL, which is stored as the first attribute of the list. It then goes backwards starting at FIRST going towards the beginning of the path until it finds the first attribute that is not an attribute of a list or difference list. In the example above, it skips REST and finds COMPS, identifying COMPS as the attribute that took the list as its value. Now that it has found which attribute introduces the list and the first attribute of an element of the list, it checks whether a default value for the list of COMPS is defined. If not, the algorithm performs an additional check to see whether COMPS takes a list or

```
Algorithm 3.3.2: IdentifyDefaultType($path, a2it, a2vt, t2a$)

$atts \leftarrow path.split('.')$
for $i \leftarrow 0$ to $atts.length() - 1$
  do
    if $atts[i]$ == 'FIRST'
      then
        if $atts.length > i$
          then $first\_att\_list \leftarrow atts[i+1]$
        $j = i$
        while $j \neq 0$
          do
            $j \leftarrow j - 1$
            if not $atts[j] \in$ ['REST','LIST','LAST']
              then
                $att\_with\_list\_val \leftarrow atts[j]$
                break
        if $first\_att\_list$ and $att\_with\_list\_val$
          then
            $old\_val \leftarrow a2vt[att\_with\_list\_val]$
            if not ',' $\in old\_val$
              then $itype \leftarrow a2it[first\_att\_list]$
            if 'list' $\in old\_val$
              then
                $new\_val \leftarrow old\_val +$ ',' $+ itype$
                $a2vt[att\_with\_list\_val] \leftarrow new\_val$
                $t2a[itype].append(att\_with\_list\_val)$
              else print 'Warning: found list as value of'
                         $+ att\_with\_list\_val$
```

Figure 3.15: Pseudo code for finding the default types of lists

a difference list as its value. If the value of COMPS is indeed a list or difference list, it adds *synsem*, the type introducing LOCAL, as a default value for the elements on the COMPS list. If COMPS does not have a list or difference list as its value, a warning is printed. The updated feature values (now including the default values of the list's elements) are stored in the feature geometry file.

The attribute FIRST is thrown away when reducing paths including lists, since it is always part of the path for a list. Instances of the attribute REST remain, because they indicate which element of the list bears the feature value in question.

Path Completion

Path completion works in a similar manner to path abbreviation. It takes a feature path defined in TDL and a feature geometry definition file as input. The feature geometry definition file is centred around attributes. It includes the value of each attribute, as well as the type that introduces it. For each value, a list of sub- and supertypes is provided as well. The algorithm goes backwards through the abbreviated path. I will explain the procedure for the example in Figure 3.13. If the path HEAD is provided as the abbreviated path, the algorithm will retrieve the feature that declares the attribute HEAD from the feature geometry. In Matrix grammars, HEAD is introduced on the type *cat*. The algorithm checks whether the input attribute (HEAD) is already preceded by an attribute that may take *cat* as a value. In this case, the algorithm will continue with the preceding attribute as its input. In our example, this is not the case. Because there is a unique attribute in the feature geometry that takes *cat* as a value, the algorithm inserts this attribute (CAT) in front of HEAD in the path. It then continues the search with CAT as input. The algorithm introduces LOCAL before CAT and SYNSEM before LOCAL in a similar manner. The algorithm stops looking for attributes when it reaches an attribute introduced by *sign* or by a supertype, a subtype or the type itself that bears the constraint defined through the path. If the algorithm identifies more than one possible attribute that may precede the input attribute and none of them precedes the attribute in the abbreviated path, it produces an error. Finally, the algorithm always provides the shortest path possible. Paths in typed feature structures can be recursive. This means that the grammar engineer must explicity indicate any attribute that leads to recursion. For instance, SUBJ take a list of *synsem* as its value allowing all attributes of *synsem* to occur again. If the grammar writer intends to define a property of a *synsem* of the subject list, the attribute SUBJ must therefore be present. If a property of the subject of the subject is defined, SUBJ should be indicated in the abbreviated path twice, etc.

If the preceding attribute in an incomplete path has a list as its value, the algorithm checks what the default value of this list is in the feature geometry.

It completes the path up until this default value and then includes the appropriate values to place the element at the intended location of the list. This means that it will introduce the attribute FIRST if necessary. While defining lists in abbreviated form, one thus only needs to make sure to include REST for elements that do not appear at the beginning of a list. For instance [SUBJ.CASE nom] will be completed to assign the nominative case to the subject (first element on the subject list), [COMPS.REST.CASE acc] will assign the accusative case to the second element of the complements list. An additional REST will assign the value to the third member on the list, etc.

Concerning abbreviated paths

As mentioned above, the algorithms described in this section are included in declarative CLIMB. The current implementation of (procedural) gCLIMB both generates a grammar and a declarative CLIMB version of the grammar with abbreviated paths. The feature geometry extraction algorithm and path abbreviation algorithm provide error warnings when there are problems with the type hierarchy. These error messages help to identify bugs in the metagrammar or choices file. Most of the errors lead to compiling problems of the grammar and would also be spotted while loading the grammar in LKB or flopping them with PET (cf. Section 2.2.2). These identifications, described in the following sentences, thus mainly help to identify errors before loading or flopping the grammar. The feature geometry extraction algorithm indicates when a given type has no supertypes, a type has a supertype that is not defined in the grammar or the typehierarchy contains a cycle (where a type is its own supertype or two types are both each other super- and subtypes). The path abbreviation algorithm indicates whether features are placed at the wrong location or attributes are not introduced. In addition, path abbreviation can also spot an error that are not related to a formal error in the type hierarchy, but generally is the result of a mistake and will not lead to the desirable behaviour of the grammar. It identifies which types are generally placed on a specific list and indicates cases where an incompatible type was found. Because lists do not formally constrain what kind of elements are placed on them, LKB can only spot such errors when a conflicting constraint is defined for that exact same element. If the element on the list is only defined at one location, the grammar compiles as usual, but will not lead to the desired unifications.

A potential additional advantage of abbreviated paths is that it may simplify revisions to the feature geometry. If signs are consistently defined using abbreviated paths, the engineer need only change the feature geometry behind the path completion algorithm and definitions directly related to the revision in the geometry. Likewise, path abbreviation may simplify the process of combining analyses from grammars using slightly different feature geometries. Abbreviated definitions will be completed to the correct path by the path completion algorithm, unless the difference in feature geometry is reflected in the abbreviated path (in which case it must be corrected manually). As such, path abbreviation can be used to increase the modularity of the grammar. Section 3.4 addresses increased modularity and other advantages of using CLIMB and its related tools in more detail.

3.4 CLIMB applications

Advantages of CLIMB and the TDL processing tools presented in the previous section have been mentioned in Chapter 1 and throughout this chapter. This section places several advantages next to each other as an overview of ways to use CLIMB and the other tools presented in this chapter.

3.4.1 Systematic comparison

As explained in Chapter 1, the main motivation for introducing CLIMB is to provide a more systematic methodology for grammar engineering. It addresses a joint effect of two well-known challenges in syntax and grammar engineering. Namely, the facts that (1) often more than one formal analysis can be found to account for a phenomenon and (2) phenomena (and the analyses that account for them) interact.

In both syntactic research and grammar engineering, the typical scenario when more than one plausible analysis can be found to analyse a phenomenon is to compare the analyses according to their ability to account for the data, elegance and interaction with other analyses (Bierwisch, 1963; Müller, 1999; Bender, 2008a). In grammar engineering, the criterion of efficiency may also be taken into account to decide between analyses. The best analysis according to these criteria is picked and the syntactician or grammar engineer moves on to the next phenomenon.

The problem lies in testing interaction with other analyses. This is an important distinguishing criterion: an analysis should not only work well in isolation, but also in the larger context of a grammar. However, new analyses are only tested with those analyses that were selected in the past. Alternatives that were equally or almost equally plausible at the time are not considered. It may therefore occur that an analysis is chosen that excludes the best possible analysis for a phenomenon that still needs to be accounted for in future work. Consequently, the theoretical or implemented grammar is partially the result of the order in which grammar engineers looked at individual phenomena. To my knowledge, Fokkens (2011a) and the work presented in this thesis are the first to explicitly point out this problem.

CLIMB provides grammar engineers with the means to address this challenge. In cases where no conclusive evidence is found to decide between alternative analyses, all possible alternatives can be stored in CLIMB. When adding new analyses to the grammar, the grammar engineer can test them with all alternatives maintained from past decision points. This way, evidence from phenomena treated in future work can also be taken into account in the process of deciding between alternative analyses and more generally, decisions will be given a stronger empirical foundation. The grammar generation algorithm and overall setup of CLIMB ensure that the grammar engineer has full control over which properties vary and which remain consistent across grammars. The approach is thus more systematic than the obvious alternative of simply maintaining separate versions of the grammar in parallel.

Naturally, the grammar engineer must still make sure alternative analyses all interact correctly with other implementations. During the creation of gCLIMB, time investment in adjusting interactions varied from none at all to two days in the most extreme case for including object control. Typically, less than half an hour was needed. Chapter 5 will discuss the difference between traditional grammar engineering and implementing with CLIMB as part of the evaluation carried out with gCLIMB. The result of this evaluation indicates that, even when alternative analyses are maintained, grammar development time when using CLIMB seems to be at least comparable (and possibly shorter) than when only one analysis is integrated using a traditional method.

CLIMB can however not completely prevent certain analyses from being more dominant than others in design decisions. This risk is especially present when one analysis is dominant in the literature. This analysis will have influenced the course taken in theoretical syntax and hence will influence the grammar engineer when consulting the literature in support of grammar development. It may also happen that the grammar engineer is biased towards an analysis or finds it easier to work with. In this case, the grammar engineer may tend to start looking at new phenomena using grammars containing the preferred analyses. Alternatives that work well with these analyses are more likely to be tested than those that would work better with others.

In the end, CLIMB is not a magic wand that solves all problems related to interactions and alternatives. It nevertheless greatly improves on the traditional approach, where comparison in a larger context was cumbersome and not systematic. Furthermore, CLIMB could form an essential contribution to syntactic research. One of the motivations for using grammar engineering is to test analyses empirically and CLIMB provides a platform to do so more systematically. The main message to syntacticians therefore is: *grammar engineering is good for syntax and metagrammar engineering is even better.*

3.4.2 Modularity

Ways to use CLIMB involve sharing a set of analyses and plugging in alternative analyses for syntactic research, individual languages or dialects or to highlight coverage or precision, depending on the intended application for the grammar. Plugging in alternative implementations requires a certain level of modularity. This increased level of modularity is a positive by-product of the CLIMB methodology.

CLIMB allows grammar engineers to spread type definitions over different libraries. The implementations in these libraries interact on many levels, so they cannot be developed independently as in a fully modular approach. Nevertheless, CLIMB manages to address most desiderata for a modular approach to writing typed unification grammars as defined by Sygal and Wintner (2011) (see Section 7.5.1). Namely, CLIMB fulfils **signature focus**, **partiality**, **extensibility**, **parsimony**, **associativity** and **flexibility** and procedural CLIMB can, in principle, support **privacy** and **remote reference**. The **consistency** criterion is partially fulfilled by running the feature extraction and path abbreviation algorithms when creating grammars with CLIMB. Sygal and Wintner (2011) point out that the Grammar Matrix customisation system introduces a modular setup, but has the problem that the developer has no control over the libraries. CLIMB differs from the Grammar Matrix in exactly this property. The similarities between the setup of CLIMB and the proposal by Sygal and Wintner (2011) is striking. The criteria defined by Sygal and Wintner (2011) as well as their approach and how it compares to CLIMB will be discussed in detail in Section 7.5.2 as part of related work.

Path abbreviation

Modularity can be further increased by making extensive use of path abbreviation. Moshier (1997b) illustrates the problem of lack of modularity through the revision in feature geometry that has taken place between the original introduction of HPSG by Pollard and Sag (1987) and Pollard and Sag (1994), where the original distinct features SYNTAX and SEMANTICS were merged into the single feature SYNSEM. Major principles, such as the Head Feature Principle, had to be redefined because either the feature SYN or SEM would be included in their definition. The decision to merge SYN and SEM had, however, nothing to do with the Head Feature Principle and no impact on the idea behind it. This revision in the principle was thus an undesirable consequence of the revision in feature geometry. Moshier (1997b) proposes to use category theory to redefine the formalism behind HPSG so that principles may be defined independently of the feature geometry.

Path abbreviation does not provide this mathematically sound solution, but it proposes a practical solution that avoids part of this problem. If all paths in the metagrammar are abbreviated and completed by the path completion algorithm, the paths of individual principles and language specific constraints can be changed by changing the feature geometry behind the algorithm. Merging SYN and SEM or separating SYNSEM again in two features can be done without changing principles and probably even without changing any analyses in the grammar. It is, however, less principled as an approach than Moshier's proposal. The algorithm requires that the path can be completed deterministically. Revisions in the feature geometry that change the positions where multiple completion options occur may still require changes in principles or analyses that are not related to the revision in feature geometry from a linguistic point of view. Moshier's proposal, on the other hand, is a theoretical proposal and has, to my knowledge, not been implemented to this date.

Phenomenon-based organisation

The increased modularity allows the grammar engineer to group grammatical properties according to the analysis they represent rather than according to the individual types. Within CLIMB, the grammar can thus be organised according to phenomena. For the developers of the grammar, such a structure can be advantageous when revising the grammar. Even for a well-documented grammar, one gets a better overview of an implementation if all relevant definitions for an analysis can be found at one (or few) well-defined location(s) reserved for that particular analysis alone. Developers of other grammars may benefit from the structure for the same reason. If they want to see how a specific phenomenon is implemented, they do not need to work through large files containing many irrelevant definitions.

There are two (main) challenges in organising a grammar according to phenomena or analyses of phenomena. The first challenge lies in the question of how to define phenomena in the first place. The second challenge lies in the interaction of phenomena. A full discussion about the question regarding the definition of phenomena goes beyond the scope of this work, but an elaborate study can be found in Lehmann (2000). The topic has been discussed during DELPH-IN summits at several occasions.[20] It has been suggested during these discussions to take a practical approach to classifying phenomena (i.e. where is someone trained in syntactic analyses most likely to look?) and to index classifications by examples illustrating the phenomenon.

One of the problems with this approach is that it is common for a single type to contain constraints that are relevant for different phenomena. Because CLIMB makes it possible to define different properties of a type in different libraries, the grammar can be organised according to phenomena in a larger extent. Each constraint can be assigned to a type in the library addressing the phenomenon it is related to, even if other properties of this type are defined elsewhere in the metagrammar. The grammar that is ultimately generated will contain a definition of the type including all constraints defined in individual libraries.

However, there are also cases where a specific constraint is the result of the interaction between to phenomena. This leads to the question of where constraints that are the result of the combination of two phenomena should be placed. There are several ways to deal with such cases. Implementations related to more than one phenomenon can either be located at only one of the phenomena, at the location of all phenomena involved or at a special location

[20]Most notably, 26 June 2011 in Suquamish, USA (`http://moin.delph-in.net/SuquamishGrammarIndexing`) and 2 July 2012 in Sofia, Bulgaria (`http://moin.delph-in.net/SofiaLinguisticPhenomena`)

that deals with the interaction of the phenomena involved. The decision depends on a number of factors. If phenomenon A will always be present in the grammar when phenomenon B is included, implementations that deal with interactions can be added to the implementation of phenomenon B (with documentation pointing to them at the implementation of phenomenon A). If several phenomena require the same constraint, it is best to make sure this constraint is added as part of the implementation of each phenomenon separately. The resulting grammar is not affected by doubling up implementations and it is easily forgotten that an analysis relies on a definition included elsewhere. This leads to ill-formed grammars when the phenomenon introducing the constraint is not included. A special location for a specific interaction mostly makes sense when several constraints are related to the interaction of two phenomena.

3.4.3 Grammar Investigation

The TDL processing tools that have been developed as part of the CLIMB effort allow the engineer to explore properties of the grammar. The spring cleaning algorithm forms the basis of these processes. The operations required to establish spring cleaning can be used to explore other aspects of the grammar. As suggested above, it can be used to extract the feature geometry of the grammar, which can then be used for path completion.

A more linguistically oriented investigation that could be carried out is the extraction of all reentrancies. Following an initiative of Lars Hellan, a discussion was started to simplify feature structures in DELPH-IN grammars.[21] For general constraints and contributions to the semantics, the exact location of a feature in the feature geometry is not (that) relevant. As long as all references use the right path, locations of features should not affect the competence or performance of the grammar. This does not apply for structure sharing. Features placed under HEAD are shared between mother and head daughter by the Head Feature Principle. If a feature is moved from HEAD to

[21]`http://moin.delph-in.net/SofiaCLIMBAndReducingComplexity`

CAT, this is no longer the case. When simplifying the feature geometry, it is therefore the essential that features are maintained if they bundle a group of other features that are structure-shared together through reentrancies. Retrieving all reentrancies that are actively used in the grammar thus provides significant insights to the question of whether the feature geometry can be simplified. Note that grammars must be spring cleaned first: the Grammar Matrix provides a systematic set of lexical rules that share all but one feature from sign. It should therefore first be established which of these rules are actually used in the grammar.

The study described above is planned for future work as part of the aforementioned attempt to simplify feature structures.

3.4.4 Parallel Grammar Development

The possibility of including both shared aspects of analyses as well as contradictory information makes CLIMB particularly suitable for parallel grammar development for related languages. This functionality of CLIMB can be seen as an extension of the functionality offered by the Grammar Matrix customisation system. There is, however, a major difference. In the case of CLIMB, the user has full control of the system and can extend it in any direction. By limiting the amount of typological variation, CLIMB users can move quickly towards more analytical depth and a wider range of covered phenomena.

A start in exploring this aspect of CLIMB has been made in Fokkens (2011a), where grammars cover word order and auxiliary interaction for German, Dutch (both Flemish and Northern Dutch) and Danish. The SlaviCore (Avgustinova and Zhang, 2009) project has recently started to use CLIMB (Fokkens et al., 2012a). The goal of this project is to develop a Slavic core grammar that supports the development of grammars for individual languages. In its original design, only a static core was planned containing analyses of phenomena that occur across Slavic languages. The addition of CLIMB to the project will provide more possibilities of also sharing implementations that capture the wide variation found in Slavic languages.

Chapter 6 will describe the SlaviCLIMB addition to SlaviCore in detail.

3.4.5 Dialectal differences and language change

Closely related to parallel grammar development is the possibility of capturing dialectal variations and language change. In particular, SHORT-CLIMB can be used to create grammatical alternations found in dialects of a language. The idea to model language change has been developed independently by Sygal and Wintner (2011) and Fokkens (2011b). Fokkens (2011b) particularly suggests combining research on language change and dialects, where the grammar would model new variations in related dialects as well as changes that make language grow closer to the variety considered "standard", e.g. Hochdeutsch for German.

3.4.6 Alternative versions for applications

CLIMB may also capture variations that adapt the grammar for a particular application. In general, robustness is highly valued when the grammar is used for parsing. Precision is more important for natural language generation. But even within these two applications, the desired behaviour of the grammar may depend on the final task of the system it is part of. If the grammar is used, for instance, to extract examples from a large amount of text, it may be of little importance whether it returns an analysis for each individual sentence. As long as enough examples are found in the text, it may be better to have a more precise grammar that only returns correct examples. If the grammar is used to analyse a complete text, robustness may be more important. For natural language generation, the grammar should provide more variations if it is used for paraphrasing or to support a human translator than when it is used to generate a single utterance as output of an application.

The grammars in gCLIMB are mostly tuned to reach decent coverage and precision when parsing text. However, it can also generate versions of the grammar that can identify errors in adjective endings. How implemented grammars can be used to provide detailed feedback on detected errors (Bender et al., 2004; Suppes et al., 2012) is discussed in more detail in Chapter 6.

3.4.7 One in a thousand phenomena

The final usage of CLIMB suggested in this thesis is closely related to adapting the grammar for specific applications. It relates to the possibility of completely including or excluding analyses for a given phenomenon. This property can be useful if large amounts of text need to be parsed: It may be advantageous to first use a grammar that does not cover phenomena that occur rarely, but have significant negative impact on the efficiency of the grammar. Because time-out still is a common reason for the parser to fail, this would not only reduce processing time, but could also increase overall parsing coverage.

The remaining sentences could be parsed by a grammar containing analyses for rare phenomena. Ideally, these sentences would be classified first according to the kind of rare syntactic phenomenon they may exhibit. Even though this is not straight-forward as a task, it should certainly be possible to at least exclude some possible phenomena based on the part of speech of the words in the sentence.

Ultimately, one could imagine corpus studies that rank phenomena in corpora. When generating the grammar, one could decide to exclude phenomena that occur less than a certain frequency, e.g. a small and efficient grammar excluding all phenomena that on average occur less than once in ten thousand sentences.[22] From the suggested ways to use CLIMB presented here, it is the only usage which is completely left for future work at present.

[22]This idea of creating grammars covering one-in-a-thousand, one-in-ten-thousand or one-in-a-hundred-thousand phenomena emerged from a discussion with Dan Flickinger.

3.5 Summary

This chapter has introduced CLIMB and described additional software developed as part of this thesis. Section 3.1 provided an overview of how the main idea behind CLIMB was developed, compared the CLIMB approach to the Grammar Matrix and described the CLIMB workflow. An alternative declarative version of CLIMB that lowers the hurdle for applying the approach, but currently does not support the full functionality of procedural CLIMB was introduced in Section 3.2. The same section also described SHORT-CLIMB, which allows grammar engineer to apply CLIMB advantages to grammars developed the traditional way. Section 3.3 presented the spring cleaning algorithm and path abbreviation facilities. Finally, Section 3.4 provided an overview of possible applications for CLIMB.

This chapter has proposed a wide variety of applications for CLIMB, including monitoring **alternative analyses**, increasing **modularity**, **multilingual** grammar development, supporting different **dialects** as well as alternative versions for different **applications**, **phenomena based organisation** and creating variations where **rare phenomena** are included or excluded in the grammar. Finally, the processing tools provide support to **investigate properties** of the grammar. Several of these properties, such as increased modularity, support for multilingual grammar development or phenomena based organisation are typical advantages of using a metagrammar (see Candito (1998), Ranta (2009), among others). A more detailed comparison between CLIMB and other grammar development methodologies will be provided in Chapter 7. The possibility of including alternative analyses in the metagrammar to provide systematic comparison over time truly distinguishes CLIMB from other approaches. Even though this functionality is technically supported by several systems (e.g. the eXtensible MetaGrammar (Duchier et al., 2005) and Grammatical Framework (Ranta, 2011)), I am not aware of any approach that has explored this possibility.

Chapter 4

CLIMB for Germanic languages

The CLIMB methodology has been tested through the development of resource grammars for German containing alternative analyses for word order and auxiliaries. These analyses were developed in a metagrammar for Germanic languages called gCLIMB. This chapter describes the main properties of CLIMB analyses for German.[1] The first section of this chapter explains the general motivation for a German grammar and the alternative analyses that will be compared.

The following sections describe the main phenomena and alternative analyses used in the comparative studies of this thesis. Describing all implementations in gCLIMB in detail would result in (at least) a book in itself[2] and the topic of this thesis is empirical research in grammar engineering and not German syntax. The description of German word order in Section 4.2 is therefore restricted to a presentation of basic facts on German word order with a small set of illustrative examples. Evidence for the correctness of the descriptions through contrasting negative and positive examples will not be provided in this thesis (relevant references will be provided). Similarly, as far as the analyses are concerned, descriptions are limited to those properties that are

[1]This grammar resource mainly focuses on German, but includes variations to cover word order and auxiliary interactions in Dutch and Danish as well. This chapter focuses on German exclusively.

[2]To illustrate: Müller (2002) and Müller (2007), two books that describe large fragment of German syntax, do not cover all phenomena included in gCLIMB.

directly related to coverage and efficiency. This holds both for the standard HPSG analyses presented in Section 4.3, as well as the CLIMB alternative presented in Sections 4.4.2 and 4.4.1, respectively.

4.1 Motivation of gCLIMB for German

This study started with an investigation on analyses for verb second word order for the Grammar Matrix. The first version of the metagrammar described in Fokkens (2011a) covers basic word order of Danish, Dutch and German. I decided to focus on German for the rest of this investigation, because two comparable resources for this language exist that use the same linguistic theory, the same formalism and the same tools for implementation, parsing and generation. They are GG (Müller and Kasper, 2000; Crysmann, 2005) and Cheetah (Cramer and Zhang, 2009; Cramer, 2011). Observations from comparing these resources will be presented as part of the evaluations in Chapter 5. There are several reasons why word order and auxiliary subcategorisation are interesting phenomena for investigating alternative analyses. I will elaborate on the motivation behind the chosen alternative analyses in this section.

The word order analyses contained in gCLIMB are the standard HPSG analysis for German word order (originally developed within GPSG[3] by Uszkoreit (1987)) and the Grammar Matrix customisation analysis for verb second word order (Fokkens, 2010). The latter is based on Bender's (2008a) analysis for the verb second language Wambaya. The comparison between these two analyses addresses an issue pointed out previously in Chapters 1 and 3: the two sides of crosslinguistic applicability in comparative syntax. On the one hand, crosslinguistic applicability may serve as evidence to prefer one analysis above another. On the other hand, crosslinguistically occurring phenomena can be a hindrance in determining which analysis is best, because the optimal solution may differ from one language to another. Investigating these two analyses for Germanic languages and an Australian one forms a

[3]Generalised Phrase Structure Grammar

starting point to investigate this question for verb second languages.

Similar motivations apply to the alternative analyses for auxiliaries. Bender (2010) provides convincing evidence that a new analysis using a construction to share arguments between auxiliaries and their complements is preferable for Wambaya compared to the standard HPSG argument composition analysis. The customisation system currently provides argument composition, but the question is whether this should generally change to Bender's alternative or whether both variations should be provided by the customisation system. The impact of these analyses in a German resource grammar could help to answer this question.

The two phenomena together form a suitable test case for the CLIMB methodology, because they interact heavily with other components in the grammar. They also occur frequently enough to influence efficiency in parsing open text visibly. German sentences often include auxiliaries. In fact, 40.5% of the first 200 sentences in the TiGer Treebank (Brants et al., 2002) contain an auxiliary, raising or control verb which all share arguments through different mechanisms depending on the chosen analysis. The choice of word order analysis has impact on parsing or generation of any sentence. The following section provides background information on German word order and auxiliaries. It describes the main properties that have influenced the choice of analyses in HPSG.

4.2 German word order and auxiliaries

4.2.1 Main clauses

The easiest way to describe German word order is through topological fields (Erdmann, 1886; Drach, 1937). According to this model, the sentence structure for declarative main clauses consists of five topological fields: the Vorfeld ("pre-field"), the Left Bracket (LB), the Mittelfeld ("middle field"), the Right Bracket (RB) and the Nachfeld ("after field"). The fields are represented in

Vorfeld	**LB**	Mittelfeld	**RB**	Nachfeld
Der Mann	**hat**	den Jungen	**gesehen**	nach der Party
The man.nom	*has*	*the boy.acc*	*seen*	*after the party*
Der Mann	**hat**	den Jungen nach der Party	**gesehen**	
Den Jungen	**hat**	der Mann	**gesehen**	nach der Party
Nach der Party	**hat**	der Mann den Jungen	**gesehen**	
Nach der Party	**hat**	den Jungen der Mann	**gesehen**	
Den Jungen **gesehen**	**hat**	der Mann nach der Party		
Gesehen	**hat**	der Mann den Jungen nach der Party		

Table 4.1: Basic structure of German word order in main clauses

Table 4.1 along with some variations in word order.[4]

Topological fields are defined with regard to verbal forms, which are placed in the Left and Right Brackets. Each topological field has syntactic restrictions of its own. The Vorfeld must contain exactly one constituent in an affirmative main clause.[5] The Left Bracket contains the finite verb and no other elements. These conditions on the Vorfeld and the Left Bracket create German's verb second word order. Other verbal forms (if not fronted to the Vorfeld) must be placed in the Right Bracket. The Right Bracket may only contain verbs. If more than one verb is placed in the Right Bracket they must thus been adjacent. The Right Bracket then contains a so-called verbal cluster. Most non-verbal elements are placed in the Mittelfeld. When main verbs are placed in the Vorfeld, their object(s) may stay in the Mittelfeld. This kind of partial VP fronting is illustrated by the last example in Table 4.1.

The Nachfeld typically contains subordinate clauses and sometimes adverbial phrases. Constituents placed in the Nachfeld are often 'heavy constituents'. In German, the respective order between the verbs in the Right Bracket is head final, i.e. auxiliaries follow their complements. The only exception is auxiliary flip: under certain conditions in subordinate clauses, the finite verb

[4]For reasons of space, the glosses are simplified in the table. Full glosses can be found in Example 1 below. Furthermore, the table does not provide all possible word order variations for this sentence.

[5]See Müller et al. (2012) for recent work on apparent multiple constituents in the Vorfeld.

LB	Mittelfeld	**RB**	Nachfeld
daß that	der Mann den Jungen *The man.nom the boy.acc*	**gesehen hat** *seen has*	nach der Party *after the party*
daß	der Mann den Jungen nach der Party	**gesehen hat**	
daß	den Jungen der Mann	**gesehen hat**	nach der Party
daß	nach der Party der Mann den Jungen	**gesehen hat**	
daß	nach der Party den Jungen der Mann	**gesehen hat**	

Table 4.2: Basic structure of German word order in subordinates (not exclusive)

precedes all other verbal forms. The behaviour of the auxiliary flip will be discussed in Section 4.2.3.

Polar questions can be formed by changing the word order of the clause from verb second to verb initial. Example 1 illustrates the most typical order for the sentence in Table 4.1 rephrased as a polar question.

(1)

Hat	der	Mann	den	Jungen
Have.3SG.PRES	the.M.SG.NOM	man.SG.ACC	the.M.SG.ACC	boy.SG.ACC

nach	der	Party	gesehen?
after	the.F.SG.DAT	party.SG.DAT	see.PTC

‘Did the man see the boy after the party?’ [deu]

4.2.2 Subordinate clauses

Table 4.2 presents topological fields and (a subset of) possible variations in word order for German subordinate clauses. The Left Bracket contains the word that induces the subordinate clause. The subordinate conjunction *daß* fulfils this role and position in Table 4.2. In relative clauses, the position in the Left Bracket is taken by the relative pronoun. The finite verb is placed in the Right Bracket. Because the Nachfeld does not contain any elements in most sentences, subordinate clauses in German are considered verb final.

In the linguistic literature, this change in position is often explained by the assumption that only one element may be found in the Left Bracket (see Jacobs (1986) for an analysis along these lines in GPSG and, among others, Kiss and Wesche (1991); Oliva (1992); Netter (1992) in HPSG). Since the position is taken up by the subordinate conjunction or relative pronoun, the finite verb must occur in the other position reserved for verbs: the Right Bracket. Additional evidence for this idea comes from subordinate clauses in spoken German, where the complementiser *daß* is dropped. Without the complementiser, the word order of the subordinate clause must be verb second.

4.2.3 Word order and verbal forms

This section discusses the word order of verbal forms in both main clauses and subordinates in more detail. Two properties that play a direct role in the design of the grammar are highlighted here. The first is the auxiliary flip, which has played a significant role in the analysis of auxiliaries in theoretical HPSG. The second is a form of partial VP fronting, where some verbal forms are fronted and some remain in the Right Bracket. This property provides the main linguistic evidence against the more efficient alternative analysis. The related analyses will be presented in Sections 4.4.1 and 4.4.2.

Auxiliary flip

The auxiliary flip and its consequences for analysing German word order have been pointed out by Hinrichs and Nakazawa (1994). Examples (2) and (3) are taken from their work (Hinrichs and Nakazawa, 1994, p. 12) (glosses were adapted to conform to the standard used throughout this work). Example (4) illustrates standard German word order.

(2)	Ich	wußte,	daß	er	das	Examen
	PN.1SG	know.1SG.PAST	that	PN.M.3SG.NOM	the.N.SG.ACC	exam
	hat	bestehen	können.			
	have.3SG.PRES	pass.NFIN	can.NFIN			

‘I knew that he could pass the exam.’ [deu]

(3)	Ich	wußte,	daß	er	das	Examen
	PN.1SG	know.1SG.PAST	that	PN.M.3SG.NOM	the.N.SG.ACC	exam
	würde	bestehen	können.			
	would.3SG.PRES	pass.NFIN	can.NFIN			

‘I knew that he would be able to pass the exam.’ [deu]

(4) Ich wußte, daß er das Examen
PN.1SG know.1SG.PAST that PN.M.3SG.NOM the.N.SG.ACC exam
bestehen würde.
pass.NFIN would.3SG.SJV
'I knew that he would pass the exam.' [deu]

In (2) and (3), word order in verbal cluster (situated in the Right Bracket) differs from the standard word order in German subordinate clauses. In principle, verbs follow their complement in German and the finite verb would thus be expected at the final position of the clause, as in (4). This construction, where the finite verb is followed by its infinitive complement also known under the name *double-infinitive construction* (Den Besten and Edmondson, 1983), in addition to *auxiliary-flip* (Hinrichs and Nakazawa, 1994).

Structures like the ones seen in (2) and (3) are highly restricted in German. Only a small set of auxiliaries can occur in first position of the Right Bracket while being followed by their verbal complements. The grammaticality of auxiliary flip furthermore depends on the verb governed by the auxiliary and its verbal form (i.e. its morphological marking). The governed verb also determines whether the auxiliary flip takes place obligatory or optionally (Hinrichs and Nakazawa, 1994). In Examples (2)-(4), only the represented orders are allowed. A more detailed account of the auxiliary flip in German, as well as references to other work on this topic can be found in Hinrichs and Nakazawa (1994).

Partial VP fronting

The German Vorfeld may contain verbal forms that either form a VP or a partial VP as illustrated by the last two examples in Table 4.1. There is another form of partial VPs, where auxiliaries remain in the Right Bracket, but the main verb is fronted to the Vorfeld. This form of partial VP fronting is provided in Example 5:

(5)	Bestehen	hat	er	das	Examen
	pass.BSE	have.3SG.PRES	PN.M.3SG.NOM	the.N.SG.ACC	exam
	können.				
	can.BSE				

‘He could **pass** the exam.’ [deu]

This form of partial VP fronting results into a structure with verbal forms in three topological fields: *bestehen* (‘pass’) is placed in the Vorfeld, the finite verb *hat* (‘has’) occurs in the Left Bracket and *können* (‘can’) occurs in the Right Bracket.

Crossing branches

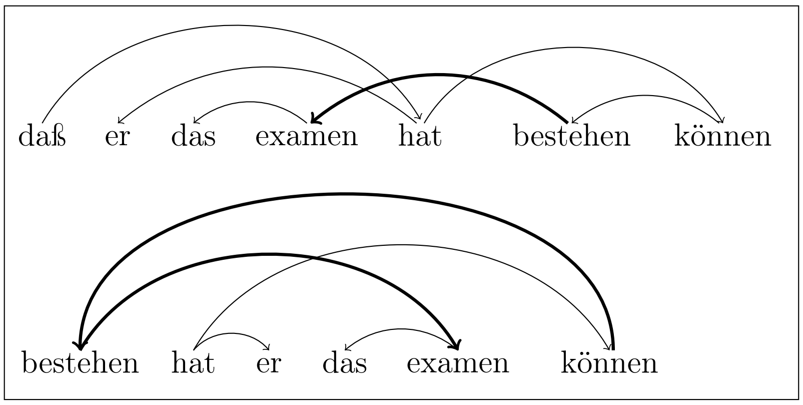

daß er das examen hat bestehen können
bestehen hat er das examen können

Figure 4.1: Dependency structures of German examples violating the adjacency principle

The dependency structures of the examples illustrating the two phenomena are represented in Figure 4.1. Within dependency representations, constituency is captured by the adjacency principle (MelÊźčuk, 1988). This principle dictates that dependency arrows should not cross and no arrow should cross over the root of the utterance. The adjacency principle is not respected in structures that exhibit the auxiliary flip or partial VP fronting.[6] The de-

[6]Violation of the adjacency principle are quite common in German clauses, e.g. object fronting when the clause contains an auxiliary. The reason why these are different will be explained in detail as the analyses for German word order are presented.

pendencies violating adjacency are marked in **bold**. When the auxiliary flip occurs, the dependency from main verbs to their object cross the dependency from the finite verb to its complement. This also occurs in the case of partial VP fronting, with the additional violation that the dependency between *können* ('can') and *bestehen* ('pass') crosses the root of the sentence *hat* ('has').

The auxiliary flip excludes analyses which propose that the main verb forms a VP with its complements in the Mittelfeld. Partial VP fronting of one verb leaving its head behind in the Right Bracket does not form a special challenge to standard analyses for German verb second word order, but it provides linguistic evidence against an efficient alternative for the standard HPSG analysis, as will be explained in Section 4.4.2. The next section will present standard HPSG analyses for word order and auxiliaries. This section will clarify why auxiliary flips have played an important role in German syntax, despite their limited occurrence.

4.3 Standard HPSG analyses for German

4.3.1 Verb secondness

The standard analysis for German word order in HPSG treats the fact that one constituent is placed in the Vorfeld as a long distance dependency. This analysis was originally worked out within Generalised Phrase Structure Grammar (Gazdar et al., 1985, GPSG) by Uszkoreit (1987). Uszkoreit (1987) based his analysis on Diderichsen (1962) and Welin (1979) working on Danish and Swedish, respectively. In HPSG, this is handled by the so-called Filler-Gap construction. I will illustrate this through the example of a basic German sentence. This is followed by a short explanation of the Filler-Gap analysis in HPSG.

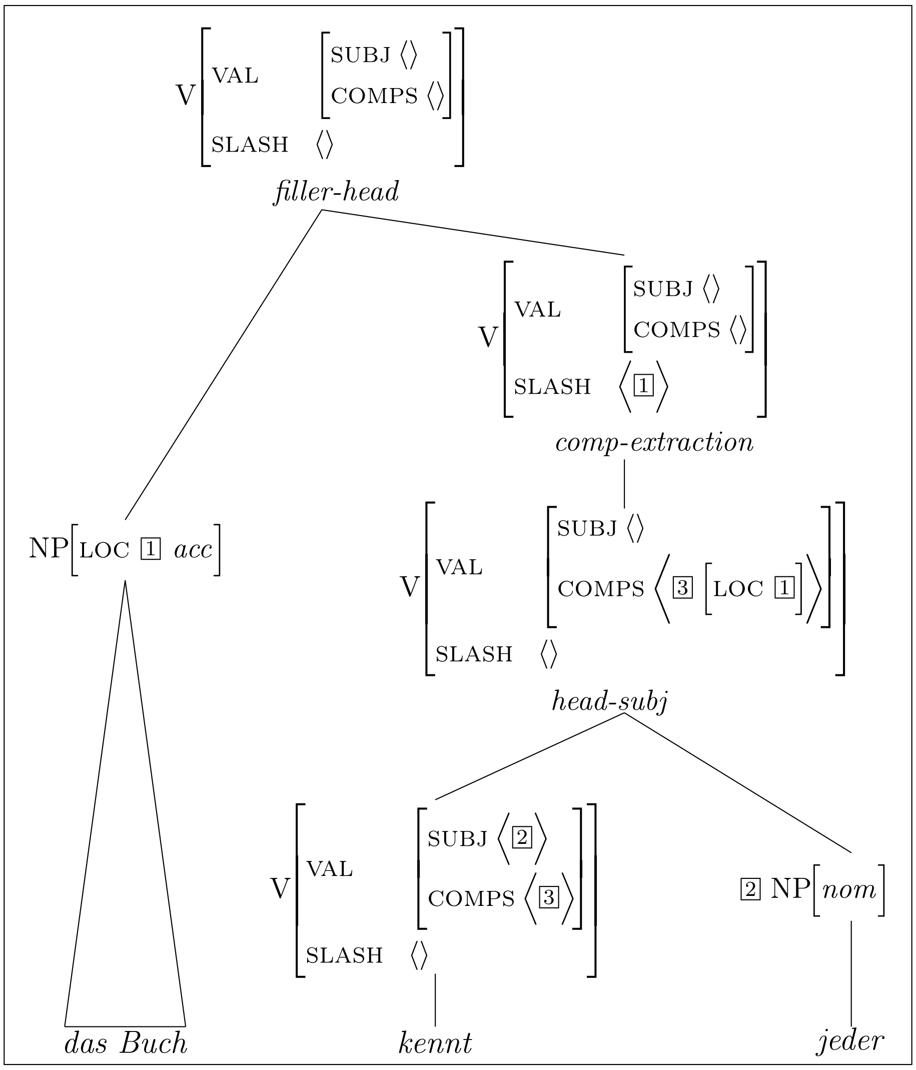


Figure 4.2: Basic analysis of *Das Buch kennt jeder*

A basic German sentence

Figure 4.2 presents the basics of a HPSG analysis for the German sentence presented in (6):

(6)	Das	Buch	kennt	jeder.
	the.N.SG.ACC	book.SG.ACC	know.3SG.PRES	everyone.NOM

‘Everyone knows the book.’ [deu]

It is inspired by an example of Müller (Müller, 2007, 168), but adapted to fit the Filler-Gap analysis as provided by the Grammar Matrix as well as the German specific assumptions used in gCLIMB. The object of the sentence *das Buch* ('the book') is found in the Vorfeld. The analysis as presented here presupposes that de object *das Buch* is extracted from the Mittelfeld, i.e. it supposes the objects canonical position is where it is found in case of a question (cf. (7)) or subject fronting (cf. (8)).

(7)	Kennt	jeder	das	Buch?
	know.3SG.PRES	everyone.NOM	the.N.SG.ACC	book.SG.ACC

'Does everyone know the book?' [deu]

(8)	Jeder	kennt	das	Buch.
	everyone.NOM	know.3SG.PRES	the.N.SG.ACC	book.SG.ACC

'Everyone knows the book.' [deu]

This analysis first combines the finite verb *kennt* ('knows') with its subject *jeder* ('everybody'). Next, the object is extracted and its local values are placed on the SLASH list of the mother.[7] The sentence is completed by the *head-filler-phrase*, where the missing element in the SLASH list is found and added to the structure. As can be observed in the coindexes, the LOCAL value of *das Buch* is identical to the value in SLASH, which in return is identical to the LOCAL value of the verb's object. Through this mechanism, the constituent in the Vorfeld is identified as the verb's object leading to the correct semantic interpretation.[8] The rest of this subsection will present the *complement-extraction* rule and the *head-filler* rule and explain how they achieve the desired syntactic behaviour.

[7]My analysis differs from the one presented in Müller (2007) in these two points. Müller introduces a phonetically empty gap first and then combines the verb with its subject.

[8]Syntactic accounts for German word order also generally assume that the finite verb in polar questions and main clauses also leaves a trace in the Right Bracket. This analysis is not used in my grammars and will not be discussed in this work for reasons mentioned in the introduction of this chapter.

Filler-Gap in HPSG

Figure 4.3[9] presents a simplified representation of complement extraction as provided by the Grammar Matrix. Similar rules are included in the grammar for subject extraction and argument extraction. The *complement-extraction* rule states that the first element on the HEAD-DTR's complement list is a *gap*, which places its LOCAL value on its own SLASH list. The mother's SLASH value is identical to the SLASH of the gap. The gap itself is no longer part of the complement list of the mother. The resulting phrase is not looking for the complement locally anymore, but stores the requirement that it must find this element somewhere else in the structure under SLASH.

$$\begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{LOCAL}|\text{COMPS} & \boxed{1} \\ \text{NON-LOCAL}|\text{SLASH} & \boxed{2} \end{bmatrix} \\ \text{HEAD-DTR}|\text{COMPS} & \left\langle \textit{gap} \begin{bmatrix} \text{LOCAL} & \boxed{3} \\ \text{NON-LOCAL}|\text{SLASH} & \boxed{2}\ <!\ \boxed{3}\ !> \end{bmatrix} \right\rangle \oplus \boxed{1} \end{bmatrix}$$

Figure 4.3: Simplified representation of complement-extraction

The value of SLASH of lexical items is the concatenation of the value of SLASH from its arguments. Figure 4.4 represents how this achieved for a lexical item taking two arguments. Phrases that combine a head with one of its arguments pass the SLASH value of the head daughter up to the mother. Because the head daughter's SLASH value is made up of the concatenation of the SLASH value of its arguments, elements extracted from one of these arguments will end up on the mother's SLASH. Head-modifier phrases use the same mechanism as presented in Figure 4.4 to collect items on the SLASH list of its daughters.[10]

[9]<! ... !> indicates that the value of a feature is a difference list, <! !> a difference list with zero elements on it. See (e.g.) Copestake (2002) for an explanation of difference lists.

[10]This approach for gathering elements in the SLASH of arguments on the head of the

$$\begin{bmatrix} \text{SYNSEM}|\text{NON-LOCAL}|\text{SLASH} & \begin{bmatrix}\text{LIST} & \boxed{1st}\\ \text{LAST} & \boxed{last}\end{bmatrix} \\ \text{ARG-ST} & \left\langle \begin{matrix} \begin{bmatrix}\text{NON-LOCAL}|\text{SLASH} & \begin{bmatrix}\text{LIST} & \boxed{mid}\\ \text{LAST} & \boxed{last}\end{bmatrix}\end{bmatrix} \\ \begin{bmatrix}\text{NON-LOCAL}|\text{SLASH} & \begin{bmatrix}\text{LIST} & \boxed{1st}\\ \text{LAST} & \boxed{mid}\end{bmatrix}\end{bmatrix} \end{matrix} \right\rangle \end{bmatrix}$$

Figure 4.4: The SLASH of a basic-two-arg lexical item in the Grammar Matrix

The slashed element will be introduced in the structure elsewhere through the Head Filler Schema. A basic version of the Head-Filler Schema for German is presented in Figure 4.5. Like the analysis in Figure 4.2, it is based on Müller (2007) but adapted to the feature geometry from the Grammar Matrix and constraints as used in gCLIMB.

$$\begin{bmatrix} \text{NON-LOCAL}|\text{SLASH} & \text{<! !>} \\ \text{HEAD-DTR} & \begin{bmatrix} \text{LOCAL}|\text{CAT} & \begin{bmatrix} \text{HEAD}|\text{FORM} & \textit{finite} \\ \text{VAL} & \begin{bmatrix}\text{SUBJ} & < > \\ \text{COMPS} & < >\end{bmatrix}\end{bmatrix} \\ \text{NON-LOCAL}|\text{SLASH} & \text{<! } \boxed{1} \text{ !>} \end{bmatrix} \\ \text{NON-HEAD-DTR} & \begin{bmatrix} \text{LOCAL} & \boxed{1} \\ \text{NON-LOCAL}|\text{SLASH} & \text{<! !>} \end{bmatrix} \end{bmatrix}$$

Figure 4.5: Simplified Head-Filler Schema for German

The head daughter must be a verb in finite verb form with an empty subject and complements list. The SLASH of the head daughter contains exactly one element. This element is token-identical to the local values of the non-head

phrase is loosely based on Bouma et al. (2001). It was implemented in the ERG and later in the Grammar Matrix.

daughter (which must have an empty SLASH list itself). The resulting phrase has an empty SLASH. These restrictions make sure that the clause is headed by a finite verb that has picked up all its arguments and modifiers except for one that was extracted. Up until this point, the structure of the sentence is (finite) verb initial. The extracted element is added left of the finite verb by the head-final *filler-head* rule. The grammar requires the *filler-head* rule to apply exactly once at the end of creating the structure to create a well-formed declarative main clause with verb second word order.

Motivation

The long distance dependency analysis for elements in the German Vorfeld is mainly motivated by the fact that it provides a uniform analysis for verb second word order. First, the structure of the sentence is similar regardless of whether a subject, object, verbal complement or adjunct is placed in the Vorfeld. The verb combines with the arguments and modifiers in the Mittelfeld and Right Bracket and either a missing argument or, when all arguments are present, an adjunct are placed in the SLASH. The verb-initial structure forms a declarative main clause by the *filler-head* phrase that combines it with the constituent left of the verb, corresponding to the element previously placed in SLASH. The only difference is found in the extraction rules used.[11]

The analysis correctly predicts that the preferred relative order for elements remaining in the Mittelfeld is the same as when the fronted element was present. Extraction from subordinated clauses are typical examples of long distance dependencies. Further linguistic evidence (and possibly the most significant evidence for this analysis) comes from the fact that elements from subordinated clauses can be placed in the Vorfeld as well, resulting in a uniform account regardless of whether the fronted constituent comes from a subordinate clause or not.

[11]Müller (2007) uses one list for all subcategorised elements, not distinguishing SUBJ and COMPS, resulting in a more uniform account using only one rule for argument extraction and one rule for adjunct extraction.

4.3.2 German Auxiliaries

The standard analysis that is currently used for German auxiliaries in HPSG was introduced by Hinrichs and Nakazawa (1989, 1994). They present their analysis as an alternative analysis to Uszkoreit's (1987) work on German in GPSG. Uszkoreit's analysis assumes that the main verb combines with its argument and modifiers in the Mittelfeld before it combines with any auxiliaries. Figure 4.6, based on Hinrich and Nakazawa's example (12), (Hinrichs and Nakazawa, 1994, 19), presents the structure of Mittelfeld and Right Bracket under this analysis.

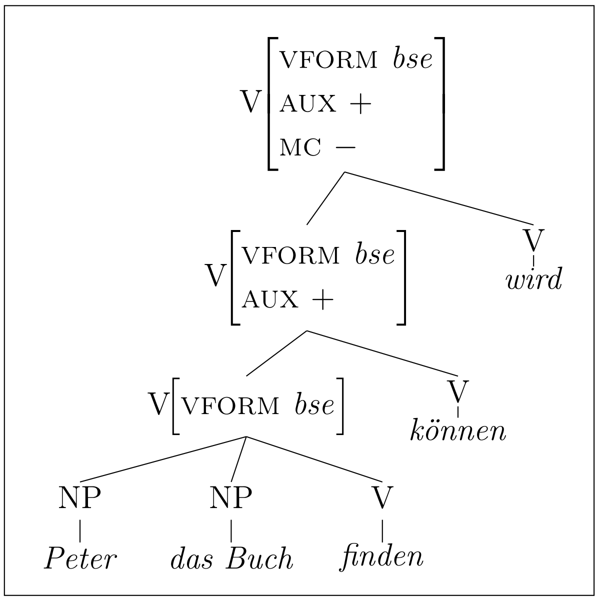


Figure 4.6: Mittelfeld and Right Bracket based on Uszkoreit (1987)

Uszkoreit's analysis addresses word order of NPs in the Mittelfeld. It does not investigate auxiliary structures in detail. Hinrichs and Nakazawa (1994) point out that under closer examination of auxiliary structures, and in particular auxiliary flip, the analysis is problematic.

Let us consider the structure of the subordinate clause in Example (2) repeated in (9).

(9)

Ich	wußte,	daß	er	das	Examen
PN.1SG	know.1SG.PAST	that	PN.M.3SG.NOM	the.N.SG.ACC	exam

hat	bestehen	können.
have.3SG.PRES	pass.NFIN	can.NFIN

‘I knew that he could pass the exam.’ [deu]

According to Uszkoreit’s analysis, the subject *er* (‘he’) and object *das Examen* (‘the exam’) would combine with the main verb *bestehen* (‘pass’) first. However, this phrase is interrupted by the head of the subordinate clause *hat* (‘has’). The formalism and theoretical assumptions in HPSG at that time[12] would not allow this structure to occur under Uszkoreit’s analysis.

$$\left[\text{VAL}\left[\begin{array}{ll}\text{SUBJ} & \boxed{1}\\ \text{COMPS} & \left\langle\left[\begin{array}{ll}\text{HEAD} & \textit{verb}\\ \text{VAL} & \left[\begin{array}{ll}\text{SUBJ} & \boxed{1}\\ \text{COMPS} & \boxed{2}\end{array}\right]\end{array}\right]\oplus\boxed{2}\right\rangle\end{array}\right]\right]$$

Figure 4.7: Subcategorisation of raising auxiliaries

Hinrichs and Nakazawa (1994) therefore propose an alternative approach where the verbal forms in the Right Bracket combine with each other before they pick up arguments in the Mittelfeld. In order to make sure the subcategorisation requirements of the main verb are maintained, auxiliaries raise all arguments of their verbal complements. Figure 4.7 presents the subject and complement values of auxiliaries under this analysis, following general assumptions concerning feature geometry found in the Grammar Matrix.

The SUBJ value of the auxiliary is identical to the SUBJ value of its verbal complement. Likewise, the complements of this verbal complement are ap-

[12]Reape (1994) proposes an alternative analysis using a structure similar to Uszkoreit’s covering the auxiliary-flip through domains. The observation from Hinrichs and Nakazawa was first made in Hinrichs and Nakazawa (1989), before domains were proposed in HPSG. Kathol (2000) argues argument composition as proposed in Hinrichs and Nakazawa (1994) and Reape’s domains work best.

pended to the auxiliary's own COMPS list. In our example, *können* ('can') combines with *bestehen* ('pass') raising its requirements of a nominative subject and accusative object. The same happens when *hat* ('has') combines with *bestehen können* ('pass can'). The structure *hat bestehen können* thus has the same subcategorisation requirements as *bestehen* started out with. The basic structure and subcategorisation values of the fragment *er das Examen hat bestehen können* are presented in Figure 4.8.

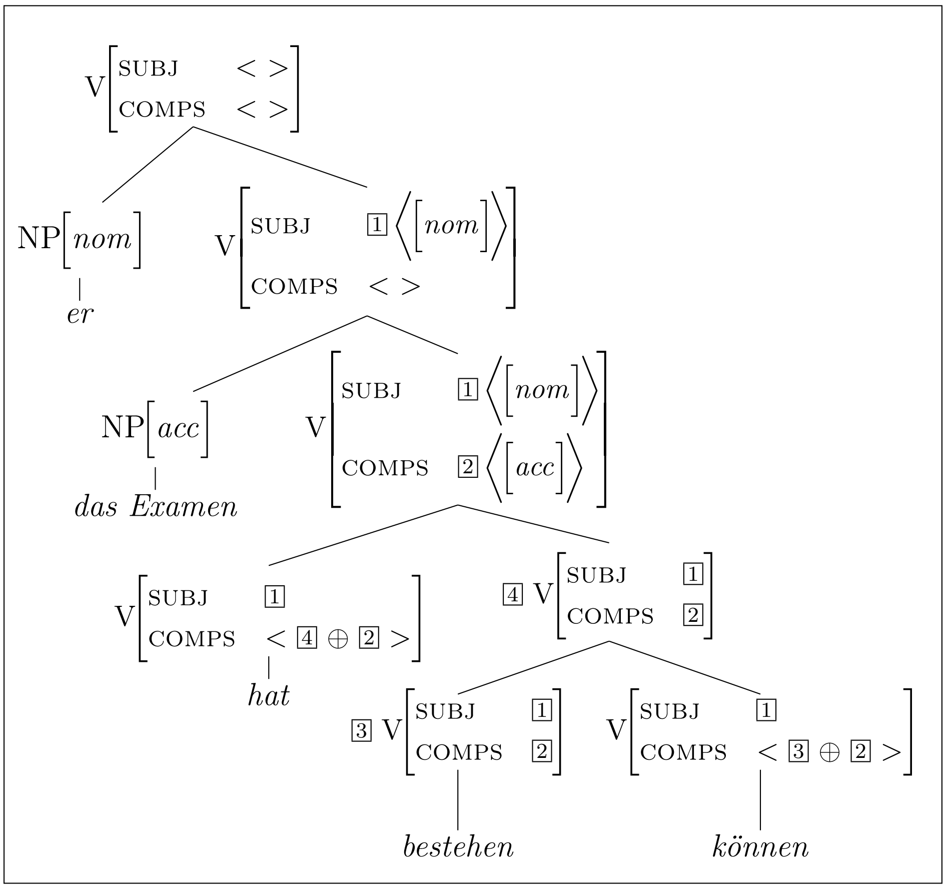


Figure 4.8: Tree for Mittelfeld and Right Bracket following Hinrichs and Nakazawa (1994)

It has been generally accepted by researchers working on German in HPSG that Hinrichs and Nakazawa (1994) show that German auxiliaries raise all their arguments (see, among others, Müller (2007); Kathol (2000); Pollard

(1994)). The problem of this analysis is that the underspecification of the auxiliary's valency leads to inefficiencies (Bender, 2010; Fokkens, 2011a). The next section will present an alternative analysis that addresses this question.

4.4 Alternative analyses in gCLIMB

In this section, alternative analyses included in gCLIMB will be described. Section 4.4.2 will present an alternative approach that uses a construction for argument raising. This alternative is attributed to Dan Flickinger[13] and shown to be more efficient in Wambaya (Bender, 2010) and for German and Dutch in small grammars (Fokkens, 2011a). Before this alternative analysis is introduced, I will present the verb second analysis provided by the Grammar Matrix customisation system. Together, the variations in analyses for verb second behaviour and variations for auxiliaries form the main alternatives explored by gCLIMB.

The descriptions in this section focus on differences between the analyses. Phenomena and interactions with other analyses that behave in the same manner for the standard HPSG analyses described above and the alternatives described in this section will not be addressed.

4.4.1 The verb second alternative in CLIMB

This subsection presents an alternative analysis for verb second word order that has been included in gCLIMB. gCLIMB also contains the Filler-Gap analysis which has been explained above. The alternative analysis presented here comes from the Grammar Matrix customisation system and was based on the Bender's (2008a) Wambaya grammar (Fokkens, 2010).[14]

[13]A partially similar solution is proposed in Müller (1997), see below.

[14]Part of this description is also included in Fokkens (2010) and can be found on `http://moin.delph-in.net/MatrixDoc/WordOrder`, accessed September 19, 2012.

The Main-Clause feature for V2 word order

The idea behind the verb second analysis provided by the customisation system is that the verb combines with all elements present on its right, and exactly one element on its left. The category feature [MC *luk*] standing for 'main-clause' is used to ensure this behaviour. The MC feature registers whether the structure is verb-initial, or verb-second. Its value *luk*, a three-valued type named after Jan Łukasiewicz and taken over from the ERG (Flickinger, 2000), can be seen as an elaborate version of *boolean.* Because this value is widely used in gCLIMB and places a prominent role in analyses for German word order, its subtypes are presented in Figure 4.9. Note that *bool* and +*-or-*− can be used interchangeably: they unify with the exact same set of values (+, −, *luk* and each other). The presence of +*-or-*− in the Grammar Matrix is an error (Bender p.c.).

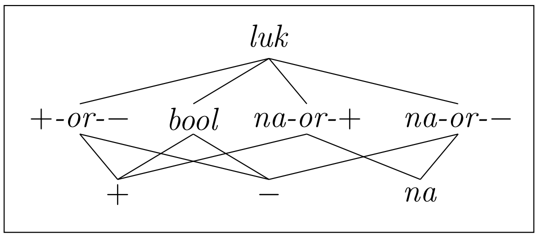


Figure 4.9: Type hierarchy for *luk* as included in the Grammar Matrix

The MC feature was originally introduced to distinguish main clauses from subordinate clauses. In the Grammar Matrix customised verb second analysis, it registers which rule was last used. The desired behaviour is that exactly one constituent appears before the verb that heads the sentence. This is achieved by making sure that the head of a clause cannot attach to any other element as soon as it has been head-final once.

Verbs start out with the value [MC *na*]. Both head-final and head-initial rules require this value on their head-daughter. The head-initial rules pass the value of MC up from their head-daughter to the mother. The head-final rules, on the other hand, assign the value + to MC,[15] so that neither head-

[15]In the original analysis for Wambaya the value *bool* (for boolean) was assigned, since

final rules nor head-initial rules can apply thereafter. The root condition in the grammar specifies that only sentences with the feature-value pair [MC +] are acceptable main clauses. The grammar will thus only accept sentences where the head-final rule has applied exactly once, placing the head of the sentence in second position.

The basic idea behind this analysis has remained unchanged in gCLIMB. However, the analysis interacts with the two alternative analyses for auxiliaries which will be presented in Section 4.4.2. Necessary changes to incorporate the alternative analysis for auxiliaries will be presented below. gCLIMB also introduced a small change in the verb second analysis for grammars using argument composition. The same principle is used, but the structure is a mirror image of the original analysis: exactly one element is picked up on the left of the finite verb and then it optionally combines with one or more elements on its right. The reasons for this change do not have an impact on the main questions addressed in the grammar and will therefore not be explained in detail here.

The verbal cluster

Verbal clusters are treated using the syntactic feature [VC *bool*] (Fokkens, 2010).[16] Words that can be part of the verbal cluster bear the value [VC +], other words are [VC −]. Phrases inherit their VC value from the non-head-daughter. Rules that build up the verbal cluster can only have non-head-daughters that are [VC +]. As soon as verbs in the Right Bracket have combined with an element that cannot be part of the verbal cluster, they can no longer be taken up in the verbal cluster themselves. In other words, all verbal forms in the Right Bracket must combine with each other before they can pick up elements in the Mittelfeld. This way all elements belonging the Right Bracket will be placed after all elements belonging in the Mittelfeld.

embedded clauses can also exhibit verb second word order in Wambaya. For German, the MC value of a clause is − if the finite verb is placed in final position and *na* otherwise.

[16]The idea of using a feature to distinguish between phrases that can occur in the verbal cluster and those that can occur in the Mittelfeld is also found in Müller (1997) who uses the feature LEX. The exact way this feature is used differs.

Verb-final and verb-initial structures

The word order analysis using MC to create verb second word order straightforwardly extends to verb-final order in subordinate clauses. Verbs and verbal phrases in the Right Bracket bear the value [MC −]. Subordinate conjunctive markers require this value on their complement. Informal verb second subordinate clauses are licensed by a unary rule.

Polar questions are formed by unary rules applying to the finite verb in the main clause. These inversion rules block *head-final* structures that place elements in the Vorfeld resulting in a verb initial sentence. They also restrict the semantics of the clause from *prop-or-ques* (proposition or question) to *ques* (question).

4.4.2 An alternative analysis for Auxiliaries

This subsection presents the alternative analysis for auxiliaries included in gCLIMB which can be used in combination with the MC word order analysis presented above. The argument composition analysis presented in Section 4.3.2 above can capture the grammatical behaviour of auxiliaries in German.[17] However, grammaticality and coverage is not all that matters for grammars of natural language. Efficiency remains an important factor, and argument composition has some undesirable properties on this level. The problem lies in the fact that lexical entries of auxiliaries have underspecified elements on their subcategorisation lists. With the current chart parsing and chart generation algorithms (Carroll and Oepen, 2005), an auxiliary in a language with flexible word order will speculatively add edges to the chart for potential analyses with the adjacent constituent as subject or complement. Because the length of the lists is underspecified as well, it can continue wrongly combining with all elements in the string. In the worse case scenario, the number of edges created by an auxiliary grows exponentially in the number of words and constituents in the string.

[17]The text of this subsection is based on Fokkens (2011a).

Figure 4.10 illustrates the problem through a simplified partial parse chart for the sentence in (10).[18]

(10)

Sie	hat	die	süßen	Erdbeeren
PN.3SG	have.3SG.PRES	thePL.ACC	sweetPL.ACC.WEAK	strawberry.PL
gerne	gegessen.			
with pleasure	eat.PTC			

‘She enjoyed eating the sweet strawberries.’ [deu]

The analysis in the illustration uses a binary structure where verb second order is achieved by picking up one constituent to the left of the finite verb and then one or more to its right (see Section 4.4.1). The finite verb *hat* may be head of a declarative verb second clause or of a head-initial polar question. It can combine with *sie* (‘she’) as its subject (HS) or as a complement (HC). The same holds for the determiner *die*. The string *sie hat die* forms 2*2 edges. Likewise, the word *süßen* can be added to *hat die* or *Sie hat die* as an additional subject or an additional complement. Moreover, it may form a constituent with the word *die*, so it is also possible to combine both structures for *sie hat* (where *sie* is subject or complement) with *die Süßen* (‘the sweet ones’) in two ways (with *die Süssen* as subject or complement). The next line of cells would continue with 62 new edges for *sie hat die süßen Erdbeeren* and 36 edges for *hat die süßen Erdbeeren gerne*.

Even in this simplified version of the grammar, it is clear that many superfluous edges are created. The auxiliary can only exclude ungrammatical structures as it combines with the main verb *gegessen* (‘eaten’). If we now take into account that the auxiliary and main verb may stand much further apart than in this relatively short example sentence, it is clear there is an undesirable effect on efficiency. In this example, lexical ambiguity was ignored and the grammar rule allowing a head to combine with the second complement on its list was excluded from the representation in Figure 4.10. The parser would actually create three edges (*head-subj*, *head-comp*, *head-second-*

[18]As for Figure 2.11, p. 2.11 the numbers in square brackets indicate the edge number, the numbers in round brackets indicate which edges are combined. Rule names are abbreviated: S stands for *subject*, C for *complement*, SP for *specifier*, M for *modifier* and H for *head*

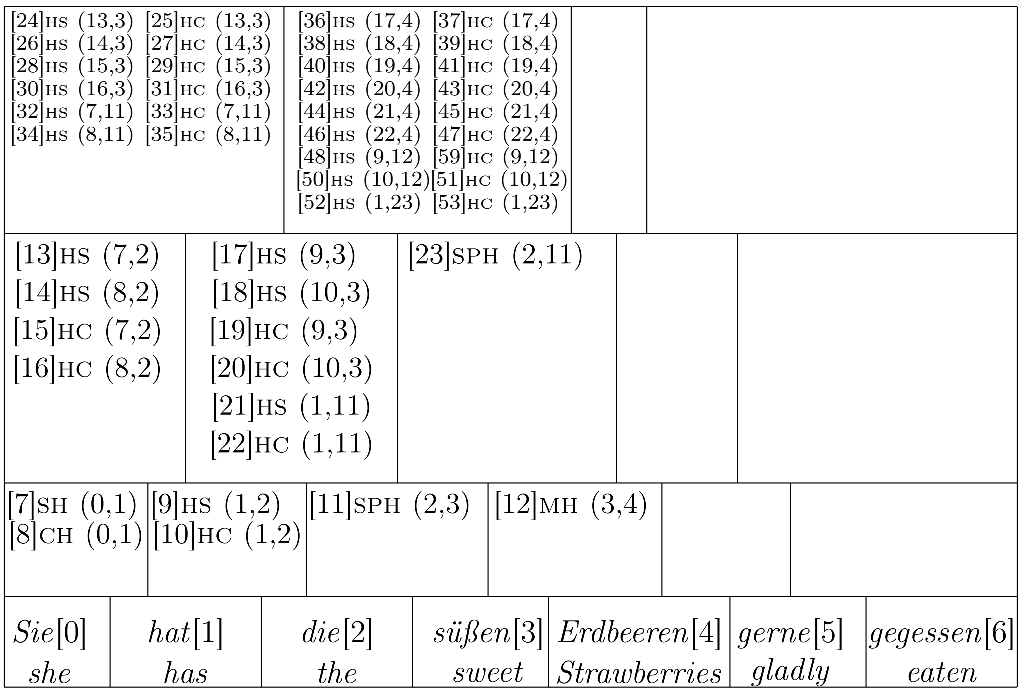

[24]HS (13,3) [25]HC (13,3)
[26]HS (14,3) [27]HC (14,3)
[28]HS (15,3) [29]HC (15,3)
[30]HS (16,3) [31]HC (16,3)
[32]HS (7,11) [33]HC (7,11)
[34]HS (8,11) [35]HC (8,11)
[36]HS (17,4) [37]HC (17,4)
[38]HS (18,4) [39]HC (18,4)
[40]HS (19,4) [41]HC (19,4)
[42]HS (20,4) [43]HC (20,4)
[44]HS (21,4) [45]HC (21,4)
[46]HS (22,4) [47]HC (22,4)
[48]HS (9,12) [59]HC (9,12)
[50]HS (10,12)[51]HC (10,12)
[52]HS (1,23) [53]HC (1,23)
[13]HS (7,2)
[14]HS (8,2)
[15]HC (7,2)
[16]HC (8,2)
[17]HS (9,3)
[18]HS (10,3)
[19]HC (9,3)
[20]HC (10,3)
[21]HS (1,11)
[22]HC (1,11)
[23]SPH (2,11)
[7]SH (0,1)
[8]CH (0,1)
[9]HS (1,2)
[10]HC (1,2)
[11]SPH (2,3)
[12]MH (3,4)
Sie[0] *she*
hat[1] *has*
die[2] *the*
süßen[3] *sweet*
Erdbeeren[4] *Strawberries*
gerne[5] *gladly*
gegessen[6] *eaten*

Figure 4.10: Chart for *Sie hat die süßen Erdbeeren gerne gegessen* using argument composition

comp) for each constituent headed by the finite auxiliary and each possible neighbouring word or constituent, including multiple edges for ambiguous words that can be linked to more than one lexical entry.

The efficiency problem is even worse for generation: while the parser is restricted by the surface order of the string, the generator will attempt to combine all lexical items suggested by the input semantics, as well as lexical items with empty semantics, in all possible orders.

The auxiliary+verb construction

Bender (2010) presents an alternative approach to auxiliary-verb structures for Wambaya. The analysis introduces auxiliaries that only subcategorise for one verbal complement, not raising any of the complementsâ arguments or its subject. Auxiliaries combine with their complement using a special aux-

$$
\text{(i)}\ \begin{bmatrix} \textsc{subj} & \langle\ \rangle \\ \textsc{comps} & \left\langle \begin{bmatrix} \textsc{head} & \textit{verb} \end{bmatrix} \right\rangle \end{bmatrix}
\qquad
\text{(ii)}\ \begin{bmatrix} \textsc{val} & \begin{bmatrix} \textsc{subj} & \boxed{1} \\ \textsc{comps} & \boxed{2} \end{bmatrix} \\ \textsc{head-dtr}|\textsc{val}|\ \textsc{comps}\ \boxed{3} & \\ \textsc{non-head-dtr}\ \boxed{3} & \begin{bmatrix} \textsc{val} & \begin{bmatrix} \textsc{subj} & \boxed{1} \\ \textsc{comps} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}
$$

Figure 4.11: Auxiliary's valence (i) and Auxiliary+verb Construction (ii)

iliary+verb construction. Figure 4.11 presents this alternative solution. In principle, the new analysis uses the same technique as argument composition in that the auxiliary+verb combination inherits the valence requirements of the verb. The difference is that the auxiliary now starts out with only one subcategorised element and can only combine with potential verbal complements that are appropriately constrained. The structure that combines the auxiliary with its complement places the remaining elements on the complement's SUBJ and COMPS lists on the respective lists of the newly formed phrase, as can be seen in Figure 4.11 (ii). The constraints on raised arguments are known when the construction applies. The efficiency problem sketched above is thus avoided.

Because arguments are shared between auxiliary and its verbal complement, it can handle phenomena such as the auxiliary-flip or subjectless main verbs showing up as a complement in a similar way as the standard lexical argument composition analysis. Related phenomena such as object control can be handled by structures similar to the one presented in Figure 4.11.[19]

Figure 4.12 shows the parse chart for the sentence *sie hat die süßen Erdbeeren gerne gegessen* using the auxiliary+verb construction. Ambiguity of lexical entries was ignored (like in Figure 4.10). The rule combining a head with the second complement, on the other hand, did not need to be excluded to maintain the overview. The length of the complement list and its syntactic properties are clear at all stages and this rule can therefore not apply. In

[19]See Section 5.1.3 p.5.2 for a brief explanation and example of object control.

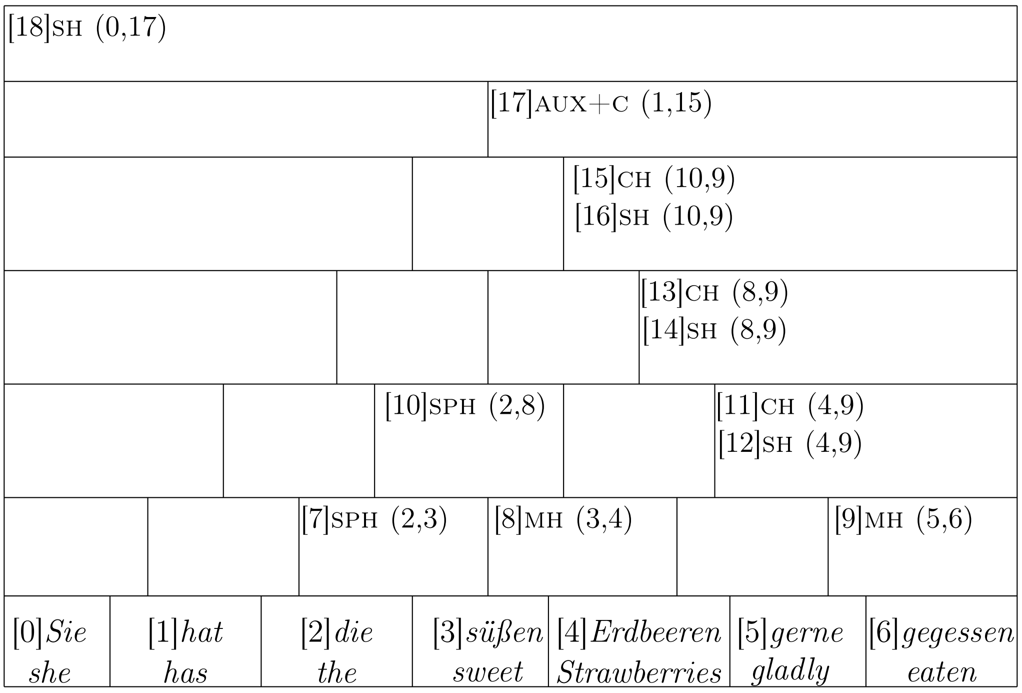


Figure 4.12: Chart for *Sie hat die süßen Erdbeeren gerne gegessen* using argument composition

fact, the chart in Figure 4.12 contains all edges that will be created under this analysis if lexical ambiguity is ignored. Note that *Erdbeeren*, *süßen Erdbeeren* and *die süsse Erdbeeren* are all potential subjects or objects, because the determiner is optional for plural noun phrases (as in English) and the determiner, adjective and noun endings can indicate either nominative or accusative case. Had the phrase been unambiguously marked as accusative, only *head-comp* edges would be created combining the NPs with the main verb.

The verb second analysis

The original MC analysis for verb second word order needed to be adapted to work correctly with the auxiliary+verb construction. As explained above, auxiliaries must combine with their verbal complement before they can combine with any of its arguments. If its verbal complement is in the Right Bracket, the finite auxiliary picks up this element on its right and then one element on its left as before. When this verbal complement is fronted on the other hand, the finite auxiliary must pick up the fronted constituent in the Vorfeld before picking up any elements on its right. The auxiliary+verb construction thus requires a separate set of rules for fronted verbal forms, just as it did for Wambaya (Bender, 2010).

Partial VP fronting: a challenge

The advantage in efficiency of the auxiliary+verb construction analysis compared to argument composition is clear. However, the analysis cannot deal elegantly with the complete range of data in German. In its basic form, the auxiliary+verb structure cannot handle partial VP fronting as presented in Section 4.2.3. In this form of partial VP fronting, the main verb is placed in first position leaving one or more verbal forms behind in the verbal cluster. Example (5) illustrating this form of partial VP fronting is repeated in (11). Its dependency structure reflecting subcategorisation as provided in the lexicon is presented in Figure 4.13.

(11)

Bestehen	hat	er	das	Examen	können.
pass.BSE	have.3SG.PRES	PN.M.3SG.NOM	the.N.SG.ACC	exam	can.BSE

‘He could **pass** the exam.’

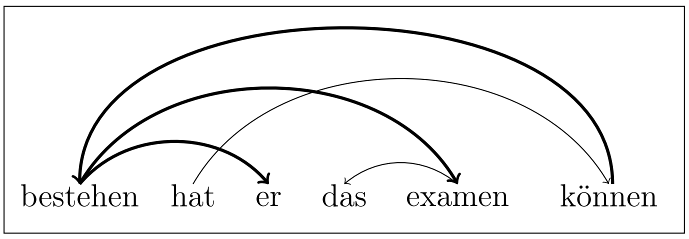


Figure 4.13: Dependency structures of partial VP fronting

The problem is that the subject and object in this sentence cannot combine with the verbs of the sentence. The verb *können* ("can") in the Right Bracket only subcategorises for a verbal complement. In this case, its complement is *bestehen* ("pass"). Under the auxiliary+verb construction analysis, *können* only raises arguments of its complement *bestehen*, when the latter is identified as its verbal complement. Auxiliary *können* thus needs to combine with *bestehen* before it can combine with *das Examen* ("the exam") or *er* ("he"). However, *können* and *bestehen* cannot combine first, because the finite verb, subject and object are placed between the two of them. Similarly, *hat* ("has") cannot combine with the subject and object or the verb *bestehen*. It only subcategorises for one verbal complement (*können* in this sentence) and the subject and object of *bestehen* are standing in between the auxiliary and its verbal complement. Auxiliary *hat* would need to raise the subject and object from *bestehen* first. However, it could only combine with *bestehen* after raising this verbal complement from *können*, i.e. only by combining with *können* which is not possible.

This shortcoming is no reason to immediately dismiss the proposal. Structures such as found in (11) are extremely rare. The difference in coverage of a parser that can and a parser that cannot handle such structures is likely to be tiny, if present at all, nor is it vital for a sentence generator to be able to produce them. However, a correct grammar should be able to analyse and produce all grammatical structures. I therefore implemented an additional version of the auxiliary+verb construction using two rather complex rules that capture examples such as (11).

The basic idea behind the analysis is that the finite verb combines with the fronted verb even though this verb is not its complement. This is achieved by the first special rule (the *noncomp-head-rule*). While it combines with this verb, it raises its arguments. It can then combine with arguments on the Mittelfeld. The second special rule (*insert-aux-rule*) adds the verbs in the Right Bracket to this structure. The original analysis provided in Fokkens (2011a) pretended that the fronted verb was the complement of the finite verb. The *noncomp-head-rule* allowed the two to combine, even if subcategorisation restrictions did not match, and marked that the *insert-aux-rule* should apply later on. The *insert-aux-rule* checked whether the inserted auxiliary (or auxiliary group) matched the subcategorisation requirements of the finite verb and whether the group itself subcategorised for a verbal form like the fronted verb. The semantics were fixed by breaking up the relation between the finite verb and the main verb, creating new relations between the finite verb and the auxiliary group and the auxiliary group and the main verb. This solution works, but radically violates the principle of semantic compositionality. The solution was therefore revised into a more elegant alternative which will be described briefly below.[20] The tree in Figure 4.14 represents the analysis for the first part of the sentence in (12).

(12)	Ausruhen	hat	er	können	(nur	schlafen	nicht).
	rest.BSE	have.3SG	PN.M.3SG.NOM	can.BSE	just	sleep.BSE	not

‘He could **rest** (just not sleep).’

The structure of the analysis remains the same as described above: the finite verb combines with the fronted verb through a special *noncomp-head* rule, presented in Figure 4.15, and auxiliaries in the Right Bracket are added higher in the tree through an *insert-aux* rule. However, the new solution does not temporarily pretend that the fronted verb is the verbal complement of the finite verb building incorrect semantics. The verbal complement of the finite verb remains on the mother's complement list. The fronted verb is identified with the complement of this verbal complement. The *insert-aux* functions as a regular *head-comp* rule, except that it explicitly creates the semantic relation between the fronted verb and the auxiliary that subcategorises for it. Figure 4.16 represents the semantics of the *insert-aux* rule.

The solution described above respects semantic compositionality, but it is still not in line with MRS algebra as described in Copestake et al. (2001). In MRS algebra, a phrase builds up semantics by filling positions to be filled or so-called *holes* in the semantic head daugther with the hook element of other daughters. The *insert-aux* daughter fills one of the holes of the daughter

[20]The basic idea for this improved solution came from Dan Flickinger.

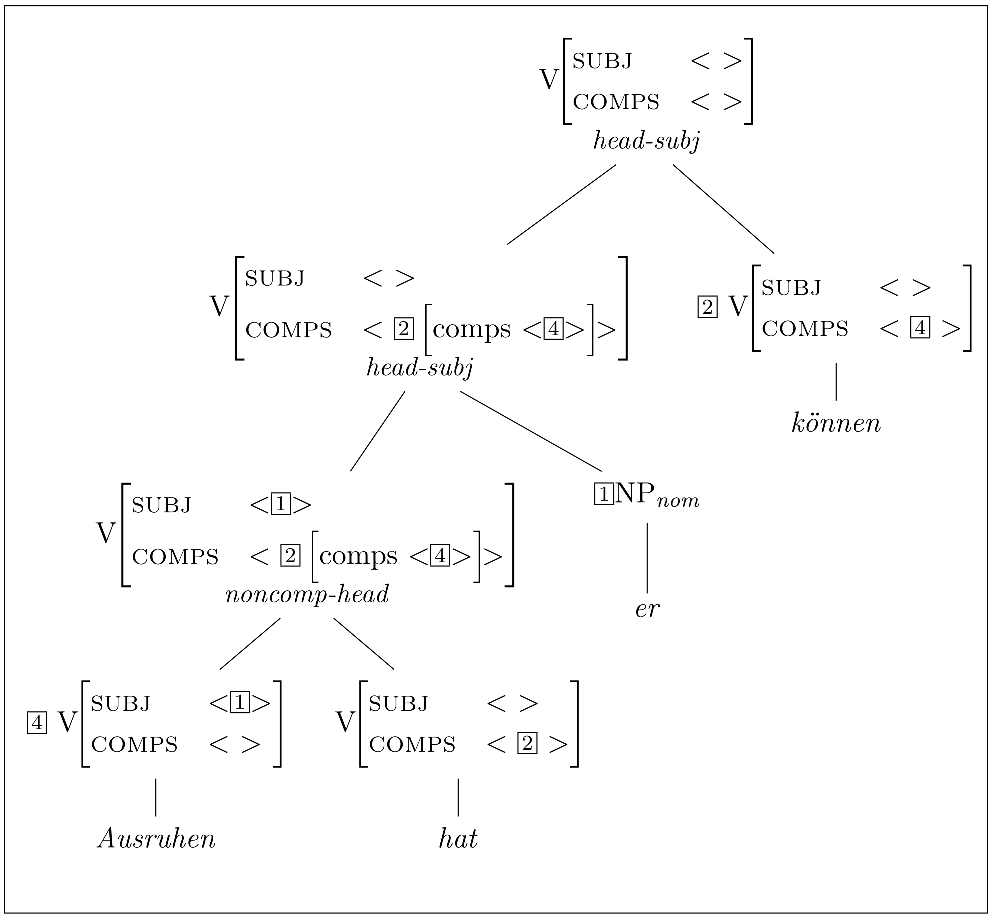


Figure 4.14: Tree structure for *Ausruhen hat er können*

that is not the semantic head with a hook that is already embedded in the semantics of the head daughter. Making an embedded hook available to build up the semantics is not in line with MRS algebra. The search for a more elegant solution will therefore be continued in future work.

Nevertheless, this solution forms a significant improvement to the previous solution, where an incorrect semantic structure was built temporarily and corrected by breaking up the incorrect semantic relation into two while inserting the auxiliary.

As far as efficiency is concerned, the full coverage version will ideally remain efficient enough as the grammar grows. But if this turns out not to be

$$\begin{bmatrix} \text{SUBJ} & \boxed{4} \\ \text{COMPS} & \left\langle \boxed{1} \oplus \boxed{2} \right\rangle \\ \text{HEAD-DTR}|\text{COMPS} & \left\langle \boxed{1} \begin{bmatrix}\text{COMPS} & \boxed{3}\end{bmatrix} \right\rangle \\ \text{NON-HEAD-DTR} & \boxed{3} \begin{bmatrix} \text{SUBJ} & \boxed{4} \\ \text{COMPS} & \boxed{2}\end{bmatrix}\end{bmatrix}$$

Figure 4.15: Simplified representation of the *noncomp-head* rule

the case, the decision can be made to exclude the additional rules from the grammar or to use them as robustness rules that are only called when regular rules fail. Given the metagrammar engineering approach, it will be straightforward to decide at a later point to exclude the special rule, if corpus studies reveal that this leads to better results.

A similar analysis

Müller (1997) proposes an analysis that is in many ways similar to the auxiliary+verb construction. His analysis also addresses the efficiency problems originating in an underspecified COMPS list and proposes to constrain the

$$\begin{bmatrix} \text{COMPS} & < > \\ \text{HCONS} & \left\langle ! \begin{bmatrix} \text{HARG} & \boxed{h1} \\ \text{LARG} & \boxed{l1} \end{bmatrix} ... \; ! \right\rangle \\ \text{HEAD-DTR} & \begin{bmatrix} \text{COMPS} & < \boxed{1} > \\ \text{HCONS} & \left\langle ! \begin{bmatrix} \text{HARG} & \boxed{h1} \\ \text{LARG} & \boxed{l2} \end{bmatrix} ... \; ! \right\rangle \end{bmatrix} \\ \text{INSERT-DTR} & \boxed{1} \begin{bmatrix} \text{HOOK.LTOP} \; \boxed{l1} \\ \text{HCONS} \; <! \begin{bmatrix} \text{LARG} \; \boxed{l2} \end{bmatrix} ... \; !> \end{bmatrix} \end{bmatrix}$$

Figure 4.16: Simplified representation of the *insert-aux* rule

list until the moment auxiliary and verbal complement are combined. This proposal particularly addresses partial VP fronting. It offers a solution for the problem sketched above where the main verb is fronted, its arguments are in the Mittelfeld and the verb that governs it is situated in the Right Bracket.

Müller's (1997) analysis uses the filler-gap strategy to capture verb second word order. If the main verb is fronted while subcategorised by an auxiliary in the Right Bracket, it is placed on the SLASH of this auxiliary by a unary rule at the beginning of the derivation. When this rule applies, the subcategorisation properties of the main verb are shared with the auxiliary. Müller (1997) states: "When the hearer of a sentence hears the words that have to be combined with a trace or introduce the nonlocal dependency in another way, he or she has already heard the phrase actually located in the Vorfeld. Therefore, the information about the nonlocal dependency is present and can be used to license the extracted element" (Müller, 1997, 15). However, the information is not present locally when building the phrase with the extracted main verb. This is solved by an implementation of the parser that allows it to search for elements non-locally. The idea is that the parser looks through the string, finds the main verb bearing the morphological markings required by the auxiliary and identifies it as the verbal complement of the auxiliary before the main verb has actually entered the structure. Having identified the auxiliary's verbal complement, it can now raise its arguments while placing it in the auxiliary's SLASH. The DELPH-IN parsers and generators do not support such operations and it is thus not possible to implement this solution in CLIMB.

4.4.3 Summary of alternative analyses in gCLIMB

This chapter has presented the main analyses included in gCLIMB to capture German word order and auxiliary structures. This subsection sums up which alternations are provided.

The previous subsections have presented two alternative analyses for word order and two alternative analyses for auxiliaries in gCLIMB. The analyses for auxiliaries can also include or exclude partial fronting of the verbal group. In total, five variations can be created for German auxiliaries and word order using CLIMB. They are four variations using the MC word order analysis: argument composition with all forms of partial VP fronting, argument composition without partial verb group fronting, the auxiliary+verb construction with all forms of partial VP fronting, and auxiliary+verb construction without partial verb group fronting. The Filler-Gap analysis is only provided with argument composition and complete coverage of VP fronting, because it cannot be combined with the auxiliary+verb construction in a straightforward manner. The following paragraph elaborates on the most important properties of the auxiliary+verb analysis and will explain why it does not necessarily work well combined with the Filler-Gap analysis.

The descriptions above have shown that the auxiliary+verb construction provides a more efficient solution to argument sharing than lexical argument composition, the standard HPSG solution. However, the auxiliary+verb analysis has two shortcomings. As pointed out above, an inelegant solution is required to account for partial fronting of the verbal group. Another drawback is that it cannot be combined straightforwardly with the standard solution for verb second word order in HPSG. Section 4.4.1 explained that two sets of rules are required when the auxiliary+verb constructions is used. The auxiliary must always combine with its verbal complement first. When the Filler-Gap analysis is used, fronted verbal forms enter the structure last through the *filler-head* rule. The uniform analysis for verb second order thus disappears if the auxiliary+verb construction were adopted in combination with the Filler-Gap analysis for verb second order.

A possible solution might be to change the Filler-Gap into a Filler-Release structure. Under this analysis, the Filler would attach low, as an identified element that must combine with another part of the sentence later. The semantics and subcategorisation cancellation take place when the canonical position of the element is found (i.e. where the gap would be introduced in the old Filler-Gap analysis). One might consider adopting this structure as a general alternative for Filler-Gap or to restrict it to fronted verbal phrases. A toy grammar incorporating a Filler-Release analysis covering main clauses, intransitive, transitive and ditransitive verbs and adverbs has been developed, but the analysis has not been integrated in gCLIMB at this point.

The difficulty of combining the auxiliary+verb construction and the Filler-Gap analysis can be seen as an illustration of the main challenge addressed in this thesis. The auxiliary+verb construction has clear advantages over argument composition, but it has not been considered seriously by syntacticians working on German in HSPG until now. The reason may very well be that the Filler-Gap analysis was introduced in the early stages of phrase structure grammar for German (recall that it was already used in Uszkoreit (1987)). It was the only analysis seriously considered when Hinrichs and Nakazawa started investigating auxiliaries. The auxiliary+verb construction was therefore excluded as an option in advance.[21]

The next chapter will present the evaluation of gCLIMB. This both includes coverage and overgeneration of phenomena included in the grammar as well as changes in efficiency.

[21]There is, of course, no guarantee that auxiliary+verb would have been considered had another analysis been chosen to capture verb second order. Even if it would have occurred to syntacticians at the time, parsing efficiency is not necessarily a major concern in theoretical syntax and argument composition fits better in the lexicalist tradition of HPSG.

Chapter 5

Evaluation

This chapter presents the main evaluation of the CLIMB methodology. Its principle purpose is to test whether the methodology can be adopted as a general method for grammar engineering. In other words, it examines whether the CLIMB methodology can be used to develop large scale grammars, while maintaining alternative analyses for a central phenomenon. It attempts to provide an indication of the effects of using CLIMB for long-term development, both concerning the resulting grammars and development time.

The second part of the evaluation illustrates the kind of research that can be carried out using CLIMB. Experiments comparing the efficiency of the alternative analyses included in CLIMB are presented. These experiments address both parsing and natural language generation. In addition, two directions for future work with CLIMB are described. The first direction is concerned with comparing CLIMB analyses to analyses found in other HPSG grammars written for German. The second direction addresses the interaction between grammar and lexicon. In particular, the question of how to use a treebank to get a decent basic lexicon is addressed. Preliminary results for these research directions indicate its potential interest, but for both it holds that a complete investigation of this topic is out of scope of this thesis.

The chapter is structured as follows. It starts with a brief description of the development process of the German grammars through CLIMB. The conditions, goals and development time of the CLIMB grammars for German are presented in this section. This is followed by a discussion about the challenges of evaluating a methodology and, particularly, the impossibility of providing exact comparison between different methodologies. Nevertheless, the outcome of this experiment indicates that the CLIMB methodology scales to long-term development for resource grammars. Section 5.2 describes the stages of grammar development in more detail, together with charts of comparative efficiency of alternative analyses as the grammars cover more phenomena. The aforementioned outlook for future research directions with CLIMB is given in Section 5.3. This is followed by a summary of the chapter.

5.1 CLIMB development

This section describes the development of gCLIMB that was carried out as part of the evaluation. The goal of this evaluation is to test whether the approach initially tested with small grammars in Fokkens (2011a) can be applied to large scale grammar development and what the influence of the method is on the development process. It is not easy to address such questions since there is no exact definition of what a large scale grammar is. Furthermore, there are many influential factors to grammar development. It is not possible to compare CLIMB grammar engineering to non-CLIMB grammar engineering in a setting where all other influential factors are equal.

The fact that it is not possible to provide a solid scientifically sound comparison does not mean that a comparative study is uninformative. If the CLIMB method can be used to create a grammar that is comparable to an existing one in more or less the same time, it shows first and foremost that CLIMB can be used to create such a grammar. Even though such a study cannot prove that grammar engineering with CLIMB has no negative impact on development speed (because speed is influenced by many factors), a similar development time does indicate that any potential negative impact remains in acceptable ranges and, in my opinion, the advantages of CLIMB would outweigh any possible slow-down (if present). On the other hand, if the grammar covers significantly less phenomena after the same development time, this outcome could indicate that using CLIMB would put an additional burden on the grammar engineer. An additional advantage of aiming to create a grammar similar to an existing grammar is that it sets a clear standard: the development time to achieve the same coverage can be measured, or (depending on the outcome) the coverage obtained in similar development time can be used as evaluation. In this evaluation, the core grammar of Cheetah (Cramer, 2011) was used to set the standard for the coverage that should be obtained by gCLIMB.

Cheetah is a grammar for German that was created as part of Bart Cramer's thesis. Creating a grammar similar to Cheetah's core grammar would make a reasonable case for arguing that CLIMB can be used to develop large scale grammars, because Cramer (2011) used this grammar to show that his approach of combining a core grammar and treebank makes large scale grammar development more feasible.

Circumstances of the development of Cheetah and CLIMB are relatively similar, but there are also differences. Cramer aimed for high coverage on the TiGer Treebank (Brants et al., 2002). He kept his output relatively close to the representations in TiGer and evaluated his grammar directly on the corpus. While improving his grammar, he created a development corpus for regression testing representing the phenomena covered by his grammar. gCLIMB maps strings to MRS representations which cannot be compared to TiGer annotations directly. I therefore focused on covering the phenomena in the development corpus, which could be evaluated manually.

Cramer reports that the development of this core grammar took two years where approximately half of the time was spent on developing the core grammar, resulting in one person year (Cramer p.c.). The goal of the present evaluation was to either implement a grammar in gCLIMB covering all phenomena in the development corpus in approximately one person year or less time, or to cover as many phenomena as possible that are also covered by the Cheetah core grammar in one year. I developed a metagrammar that covers the same phenomena in the development set created for regression testing in less than six person months using CLIMB. This outcome provides an indication that using CLIMB does not lead to a significant slow-down in grammar development and possibly speeds it up.

The rest of this section describes the process of developing a grammar for German that covers at least the 93 examples reportedly captured by Cheetah (Cramer, 2011) out of a development set of 106 examples. The Cheetah development set represents a variety of linguistically interesting phenomena (see Appendix A) and was used as a guideline for designing the Cheetah core grammar. Cramer (2011) does not provide an in depth explanation of how the examples were selected, but indicates he followed the literature on German in HPSG while he implemented Cheetah's core grammar.

This section is structured as follows. First, I will outline the overall approach in Section 5.1.1. This is followed by a description of the main differences between Cheetah and the gCLIMB grammars in Section 5.1.2. Section 5.1.3 provides an overview of the phenomena covered by gCLIMB and an indication of its performance on open data. Finally, a discussion of CLIMB as a method for developing large scale grammars is given in Section 5.1.4.

5.1.1 Methodology

The grammars described in Fokkens (2011a) cover basic word order facts of main clauses in German, Dutch and Danish. Using implementations from the Grammar Matrix customisation system, the metagrammar used in these experiments handled case marking, subject agreement, tense marking, auxiliaries, intransitive, transitive verbs and coordination for nouns, noun phrases, verbs and sentences. Required lexical items and morphosyntactic rules to support these phenomena were also included. I extended the metagrammar to cover auxiliary clusters in main clauses both clause-final as found in German and Dutch and verbal clauses that precede objects as found in Danish. Ditransitive verbs were added to the grammar and the Grammar Matrix analysis for coordination was adapted as to correctly interact with the word order phenomena of these languages. Other phenomena such as negation,

adjectives	coordination (with agreement)
polar questions	negation
adverbs	adpositions
sentential complements	copula
subject control	object raising

Table 5.1: Phenomena covered by carrying out assignments of ling567

polar questions and argument optionality have basic implementations from the Grammar Matrix, but no Germanic specific analyses were presented in the initial version of the metagrammar.

I reimplemented the metagrammar described in Fokkens (2011a) to create a clean implementation that could serve as a basis for developing a metagrammar for German. In the first stage, the course assignments of *Knowledge Engineering for NLP*[1] (Bender, 2007) were carried out, except for the assignment on information structure. The analyses developed as part of these assignments were added to the metagrammar. The phenomena covered in the course are listed in Table 5.1. Small test suites containing positive and negative examples were created for each phenomenon to evaluate the behaviour of the grammars. The total test suite at the end of this stage consisted of 258 positive examples and 334 negative examples (592 in total).

Cramer's (2011) set of 106 development examples formed the basis for the next stages of development. I primarily focused on the 93 examples that are also covered by Cheetah. The standard HPSG analysis for verb second word order, *filler-gap* as described in Section 4.3.1, was also added to the metagrammar during this stage. I created an extended development set including positive and negative examples for each phenomenon in the Cheetah development set. Additional examples were based on linguistic literature on German. The complete development set includes 508 positive examples and 630 negative examples (leading to a total of 1138).[2] Table 5.15 page 186 will

[1]`http://courses.washington.edu/ling567/2010/`, accessed 08-02-2013

[2]The development set can be found at `svn://lemur.ling.washington.edu/shared/matrix/branches/antske-germanic-development/phd_evaluation/development_process/`

provide an overview of the size of the development set at different stages of grammar development.

Throughout development, I used the output MRSs of the ERG and GG to compare and validate the output MRS produced by gCLIMB. The ERG and HPSG literature were consulted to develop the analyses. Occasionally, examples were parsed with GG or Cheetah to see what their coverage, overgeneration or output MRS was. However, I did not look at their implementations or output syntax trees at any time in this stage of the development process. Investigating implementations or the syntactic structure in previously implemented grammars could have made the task of implementing the grammar significantly easier which would have rendered the outcome of this evaluation meaningless.

In order to measure development time for the grammar, I counted the weeks in which any work on gCLIMB was carried out. This included weeks where most time was dedicated to other work or conference attendance. Weeks that were exclusively spent on work that was not related to developing gCLIMB were not counted. The full set covered by Cheetah was covered in a total development time of approximately twenty five weeks, leading to an estimated development time of less than six person months. This includes the development time of the original grammars described in Fokkens (2011a) and the reimplementation mentioned above.

Coverage of the gCLIMB grammars was slightly higher on this set than Cheetah's. All gCLIMB grammars covered a total of 98 examples from the original set and two versions covered 99 examples. The phenomena in question will be presented in Section 5.1.3 after a discussion of the differences between gCLIMB and Cheetah in the next section.

For a set of phenomena this size, it is not manageable to automatically create complete sets of all possible negative examples as was done in Fokkens (2011a). Therefore, the following approach was taken to create test suites for development. First, the examples from the Cheetah set were extended to cover interactions with other phenomena (mainly word order, coordination, and wh-questions). Second, two measures were taken to avoid overgeneration. Spurious analyses were investigated as to see whether they may lead to parses of ungrammatical sentences. Furthermore, a criterion was set that the number of negative examples had to exceed the number of positive examples. This criterion was used because not all forms of overgeneration lead to spurious edges in grammatical sentences. In particular, expressions that are highly ungrammatical and violate several constraints may remain unnoticed. As mentioned above, the total development set had 508 positive examples and 630 negative examples.

5.1.2 Differences between gCLIMB and Cheetah

There are several differences between CLIMB for German and Cheetah. The main differences are briefly discussed in this section.

Multilingual aspect of gCLIMB

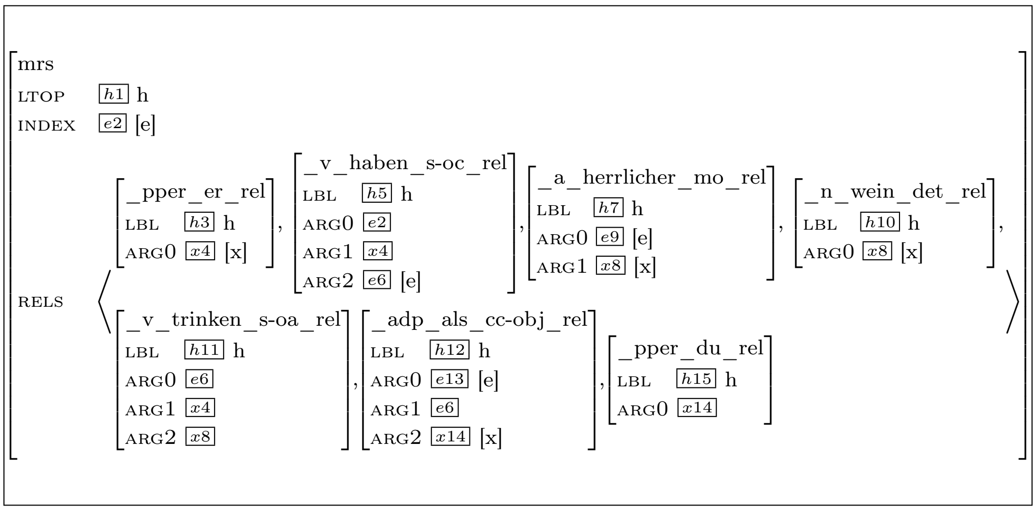
The project of building grammars with CLIMB started out as a multilingual project for Germanic languages. Word order for main clauses including auxiliaries, intransitive, transitive and ditransitive verbs was developed to correctly capture data in German, Dutch and Danish (Fokkens, 2011a). Dutch variations have been developed for adjectives, adverbs, yes-no questions and object raising in addition to these basic phenomena. The evaluation of the multilingual applicability of gCLIMB will be presented in Chapter 6.

There are two sides to this multilingual aspect of gCLIMB. On the one hand, covering Danish and Dutch variations led to extra work at the beginning of the development of the grammar. On the other hand, it led to more coverage. Cheetah does for instance not support the auxiliary flip.[3] This phenomenon could easily be implemented with gCLIMB, because the metagrammar already included complex variations in auxiliary word order as part of the standard implementation for Dutch. The auxiliary flip did not require any additions to the metagrammar, but could be integrated in the grammar by providing

[3]See Section 4.2.3, page 127 for an explanation of this phenomenon.

the right definitions in *choices*.

Semantic output



mrs
LTOP h1 h
INDEX e2 [e]
RELS
_pper_er_rel LBL h3 h ARG0 x4 [x]
_v_haben_s-oc_rel LBL h5 h ARG0 e2 ARG1 x4 ARG2 e6 [e]
_a_herrlicher_mo_rel LBL h7 h ARG0 e9 [e] ARG1 x8 [x]
_n_wein_det_rel LBL h10 h ARG0 x8 [x]
_v_trinken_s-oa_rel LBL h11 h ARG0 e6 ARG1 x4 ARG2 x8
_adp_als_cc-obj_rel LBL h12 h ARG0 e13 [e] ARG1 e6 ARG2 x14 [x]
_pper_du_rel LBL h15 h ARG0 x14

Figure 5.1: MRS output Cheetah for *Er hat herrlicheren Wein getrunken als Du.*

Perhaps the most important difference between Cheetah and gCLIMB is the difference in semantic output. Cramer (2011) explains that Cheetah adopts the overall MRS format, but otherwise stays close to syntactic dependencies, whereas gCLIMB follows MRS as it is used in the ERG. The difference between a Cheetah analysis and a gCLIMB analysis is illustrated by the representations of the sentence:

(13)	Er	hat	herrlicheren	Wein	getrunken	als	Du.
	he	have.3SG.PRES	delicious.COMP.ACC	wine	drink.PTC	than	you

‘He drank more delicious wine than you did.’ [deu]

The simple MRS output of Cheetah and gCLIMB are represented in Figures 5.1 and 5.2, respectively.

Several differences can be observed when comparing these two representations apart from minor differences such as the predicates of pronouns. First,

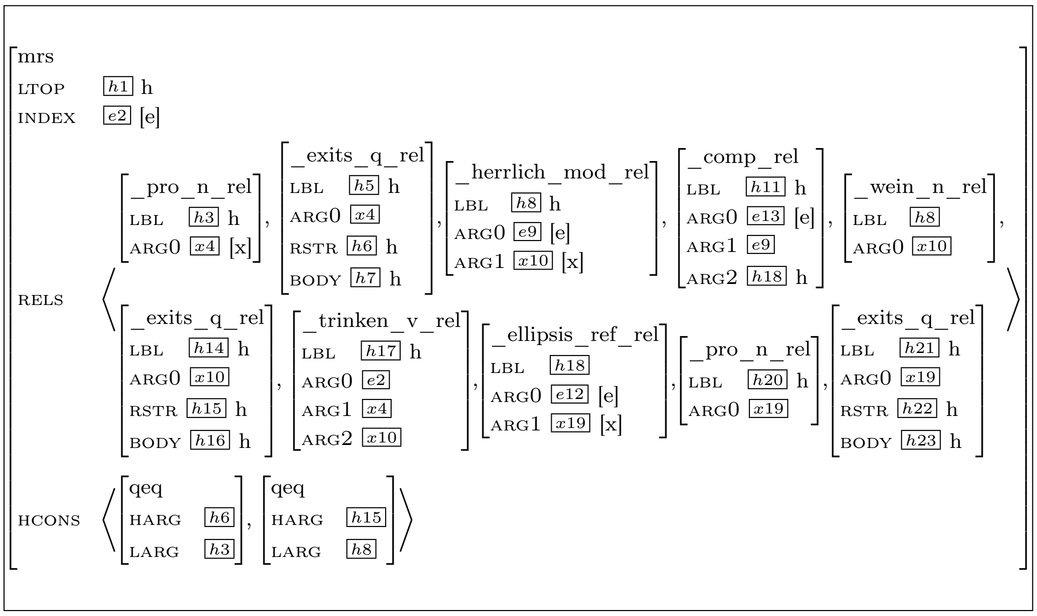

mrs
LTOP h1 h
INDEX e2 [e]
RELS
_pro_n_rel LBL h3 h ARG0 x4 [x]
_exits_q_rel LBL h5 h ARG0 x4 RSTR h6 h BODY h7 h
_herrlich_mod_rel LBL h8 h ARG0 e9 [e] ARG1 x10 [x]
_comp_rel LBL h11 h ARG0 e13 [e] ARG1 e9 ARG2 h18 h
_wein_n_rel LBL h8 ARG0 x10
_exits_q_rel LBL h14 h ARG0 x10 RSTR h15 h BODY h16 h
_trinken_v_rel LBL h17 h ARG0 e2 ARG1 x4 ARG2 x10
_ellipsis_ref_rel LBL h18 ARG0 e12 [e] ARG1 x19 [x]
_pro_n_rel LBL h20 h ARG0 x19
_exits_q_rel LBL h21 h ARG0 x19 RSTR h22 h BODY h23 h
HCONS
qeq HARG h6 LARG h3
qeq HARG h15 LARG h8

Figure 5.2: MRS output CLIMB for *Er hat herrlicheren Wein getrunken als Du.*

the Cheetah output does not include HCONS, which indicates scopal relations. The treatment of scope was left out of Cheetah, because scopal properties of lexical items (e.g. whether an adjective is scopal or intersective) are not provided in the TiGer treebank. As mentioned above, Cramer (2011) used the TiGer treebank both to derive the Cheetah lexicon and to create a gold standard for Cheetah's evaluation. Second, the treatment of syntactic function words in Cheetah differs from the CLIMB analysis. In Cheetah, all words introduce their own predicate including *hat* ("have") which marks past tense in this sentence and *als* ("than") which is selected for by the comparative. These two words are not found in the CLIMB representation. Tense marking as provided by *hat* is represented on the main verb *trinken* ("drink") (though this information is hidden in a simple MRS representation).

The grammars generally differ as it comes to introducing predicates.The gCLIMB representation introduces several predicates that do not correspond to individual words in the sentence. They include quantifiers of the nouns which are not preceded by articles, the comparative introduced by the ending

-er of *herrlicher* ("more delicious") and the VP ellipsis relation. The Cheetah representation has exactly the same number of predicates as it has words. Each word has its predicate and no new predicates are introduced. Relative clauses are a notable exception which will be explained in Section 5.1.3, p. 166. Passives form a second exception where the Cheetah analysis is not restricted to purely syntactic relations. The subject of passive sentences is analysed as the deep object (though demoted subjects are not recognised as such). Passives are discussed in more detail in Section 5.1.3, p. 174.

In the Cheetah semantic representation, *als Du* ("than you") modifies the event of *er - den Wein getrunken* ("he drank the wine"). The word *herrlicheren* ("more delicious") is treated as a normal adjective with *herrlicher* (the comparative form of *herrlich*) as its predicate. There is no explicit indication in its semantics that we are dealing with a comparative in the Cheetah semantic representation. Cheetah actually provides nearly the same semantics for the following sentence, the only difference being that the form of the adjective is *herrlichen*:

(14)	Er	hat	herrlichen	Wein	getrunken	als	Du.
	he	have.3SG.PRES	delicious.ACC	wine	drink.PTC	than	you

'He drank delicious wine as you.' [deu]

This example is also parsed by the grammar created with gCLIMB, but the fact that *herrlichen* is not a comparative is directly reflected in the (rather implausible) semantics. In the gCLIMB representation, *als Du* ("as you" or "than you") is a modifier of the sentence.[4] Furthermore, the Cheetah output does not include a VP-ellipsis to indicate that the comparative in these sentences is related to who *drank* the most delicious wine. In fact, the output of the following example is nearly identical to the representation of example (13):

(15)	Er	hat	herrlicheren	Wein	getrunken	als
	he	have.3SG.PRES	delicious.COMP.ACC	wine	drink.PTC	than

[4]The highly implausible semantics of this sentence may lead to the impression that it is not grammatical at all. The sentence means that "he" drank delicious wine when he was "you".

Dein Wein.
your wine
'He drank more delicious wine than your wine.' [deu]

The gCLIMB representation for the sentence in example (15) clearly distinguishes this structure from the VP-ellipsis in example (13). The comparative relation takes *Dein Wein* ("your wine") as its second argument, instead of scoping over the ellipsis relation as can be seen in the representation of example (13) in Figure 5.2. Section 5.1.3 will provide additional examples of differences in the semantic representation of gCLIMB and Cheetah.

From the observations above it follows that Cheetah only covers syntactic dependencies and though it can parse comparative structures and VP-ellipsis, it does not provide a full linguistic analysis for these phenomena. Cramer (2011) made a conscious decision to stick to syntactic dependencies, because the information required to create standard MRS representations was not included in TiGer. He could read the lexical information required to make syntactic dependencies off the TiGer treebank and use the treebank as a gold standard. If semantic dependencies were used, as in standard MRS, additional specifications that can only be made manually would have to be applied to the retrieved lexicon. Furthermore, additional treebanking would have been required to evaluate the output of the grammar.

The gCLIMB grammars focused first and foremost on producing correct semantic interpretations, based on MRS as it is used in the ERG. Even though the lexicon produced as part of Cheetah can be integrated in a grammar produced by gCLIMB, it was not the main goal of this project to produce a TiGer-treebank based grammar. Rather, gCLIMB was developed to test whether the CLIMB approach could be used for long-term grammar development of linguistic precision grammars. Aiming for a correct semantic representation formed an important part of this.

5.1.3 Coverage of gCLIMB

Tables 5.2 to 5.9 provide an overview of the main phenomena in the Cheetah test set that are covered by Cheetah as well as gCLIMB illustrated by an example. Each table presents a small set of related phenomena. This division was made for reasons of readability and to provide space for short explanations. It is not intended to provide a theoretically sound classification of phenomena.

A short description as well as differences between the analyses provided by Cheetah and gCLIMB grammars will be given for each set of examples. Most observations can be related back to the difference in overall goal and approach illustrated in the previous section. The goal of these description is not to further compare Cheetah and gCLIMB, but rather to illustrate the main phenomena included in the grammar and what aspects are involved when they are treated in a grammar that focuses on linguistic precision on a semantic level rather than syntactic dependencies. In other words, this section aims at illustrating the linguistic coverage and complexity of the gCLIMB grammars.

phenomenon
Example
expletives
Es gibt Wein. EXPL give.3S.PRES wine *There is wine.* [deu]
reflexives
Er hat sich auf den Wein gefreut. PRO.3S.M.NOM have.3S.PRES REFL on the.M.ACC wine rejoice.PTC *He looked forward to the wine.* [deu]
object control
Er versucht den Wein zu trinken. PRO.3S.M.NOM try.3S.PRES the.M.ACC wine to drink.INF *He tries to drink the wine.* [deu]
raising
Er sieht mich den Wein trinken. PRO.3S.M.NOM see.3S.PRES PRO.1S.ACC the.M.ACC wine drink.INF *he sees me drinking the wine.* [deu]

Table 5.2: gCLIMB phenomena in Cheetah test set: mismatches between syntactic and semantic arity

The first phenomena we consider, presented in Tabel 5.2, involve mismatches between syntactic and semantic arity. Expletives are pronouns without a referent and reflexive pronouns corefer to the subject. According to mainstream linguistic analyses, raising and control involve the covert pronouns *pro* (a phonetically empty pronoun that receives a thematic role in control structures) and *PRO* (a phonetically empty pronoun that does not receive a thematic role in raising structures). The differences in semantic representation explained in Section 5.1.2 play an important role in these structures. The expletive pronoun *es* (expletive *it*) is not semantically empty and *sich* ("himself") introduces its own predicate which is not explicitly linked to the subject in the semantic structure in the representations provided by Cheetah. Furthermore, the Cheetah MRS does not indicate that *Er* ("he") is an argument of both trying and drinking in the control structure. Pronoun *mich* ("me") is an argument of *sehen* ("see") instead of *trinken* ("drink") in Cheetah. The gCLIMB MRS representations do capture the behaviour of these phenomena.

phenomenon
Example
complement including adpositions
Er hat sich darauf gefreut. pro.3s.m.nom have.3s.pres refl on that rejoice.ptc *He looked forward to that.* [deu]
determiner including adpositions
Er geht zum Weinladen. pro.3s.m.nom go.3s.pres to the.m.dat wine store *He is going to the wine store.* [deu]
argument structure of modifiers
Er gibt mir Wein ohne ihn zu trinken. pro.3s.m.nom give.3s.pres pro.1s.dat wine without pro.3s.m.acc to drink.inf *He gives me wine without drinking it.* [deu]
NP argument structure
Er hat keine Ahnung wo der Wein liegt. pro.3s.m.nom have.3s.pres no.f.acc clue where the.m.nom wine lie.3s.pres *He has no idea where the wine is.* [deu]

Table 5.3: gCLIMB phenomena in Cheetah test set: unusual argument structures

Table 5.3 presents examples where either (a part of) the argument is contracted with the head or an item takes more arguments than it typically does. The adpositional phrase in the first example consists of one word corresponding to an adposition with demonstrative pronoun as a complement ("on that"). The second example illustrates the contraction of an adposition with the determiner of its complement. The ending of the adposition indicates a definite determiner that is masculine or neutral singular bearing dative case. The third and fourth utterances are examples of adverbs and nouns that take sentential complements.

The semantics of the examples in Table 5.3 are relatively straight-forward. The only difference between semantics provided by Cheetah and gCLIMB concerns the quantifiers introduced for determiners by gCLIMB for *zum* ("to the"), whereas Cheetah simply provides a predicate *zum*. Another minor difference is that Cheetah overgenerates and accepts the following expression:

(16)	* Im	dem	Kindes-	und Jugendalter
	In the.DAT.M.SG	PRO.M.DAT.SG	child-	and youth age

'At the age of childhood and youth' (intended) [deu]

This form of overgeneration is however not due to Cheetah not taking into consideration that *Im* ("in the") is a contraction that already includes a determiner but rather the combination of two other factors. First, Cheetah does not prevent demonstrative pronouns from taking a determiner. Second, Cheetah misses a constraint indicating the case of the phrase *Kindes- und Jugendalter* and treats it like a genitive modifier (though its case is nominative, dative or accusative).

Table 5.4 exemplifies three phenomena related to word order variations. Verbal particles are elements that are part of the verb that must be placed in the right bracket. They are prefixes of the verbal form when the verbal form is placed in the Right Bracket. When the verb is placed in the Left Bracket, the particle is separated from the verb. As illustrated in the examples below:

(17)	Er	schlägt	Wein	vor.
	PRO.3.SG.M	suggest.3.SG	wine	.PART

'He suggests wine.' [deu]

phenomenon
Example
verbal particles
Er schlägt Wein vor. PRO.3S.M.NOM propose.3S.PRES wine PART *He proposes wine.* [deu]
postpositions
Dem Mann zufolge trinkt er den Wein. the.M.DAT man according drink.3S.PRES PRO.3S.M.NOM the.M.ACC wine *According to the man, he drinks wine.* [deu]
extraposition
Er hat besseren Wein getrunken als deinen Wein. PRO.3S.M.NOM have.3S.PRES better.M.ACC wine drink.PTCP than your.M.ACC wine *He drank better wine than your wine.* [deu]

Table 5.4: gCLIMB phenomena in Cheetah test set: word order phenomena

(18) ... weil er Wein vorschlägt.
... because PRO.3.SG.M wine suggest.3.SG
'...because he suggests wine.' [deu]

Both Cheetah and gCLIMB capture the fact that *vorschlägt* never appears in the Left Bracket[5] and *schlägt* and *vor* do not appear separately in the Right Bracket. However, Cheetah treats this phenomenon by including two entries for *vorschlagen* in the lexicon: one with a particle on its argument list (with main forms that can only appear in the Left Bracket) and one with *vor* in the stem and no particle as complement (which can only appear in the Right Bracket). The gCLIMB analysis uses lexical entries with the particle on the complement list and a lexical rule that removes it from the complement list and adds it as a prefix.

As far as extraposition is concerned, Cheetah covers the restrictions applying to forms that may be placed in the Nachfeld. However, the expression *als dein Wein* ("than your wine") in the extraposition example above is treated modifier of *getrunken* ("drunk"). Table 5.5 provides examples of extraposition that are covered by both Cheetah and gCLIMB. gCLIMB covers some additional examples of extraposition that are not covered by Cheetah. These

[5]See Section 4.2, p. 124 for an explanation of topological fields (including "Nachfeld" and the Left and Right Bracket).

phenomenon
Example
wh-complements
Sie weiß bei welchem Käse er diesen PRO.3S.F.NOM know.3S.PRES with which.M.DAT cheese PRO.3S.M.NOM this.M.ACC Wein trinkt. wine drink.3S.PRES *She knows which cheese he has this wine with.* [deu]
wh-questions
Welchen Wein trinkt er? which wine drink.3S.PRES PRO.3S.M.NOM *Which wine does he drink?* [deu]

Table 5.5: gCLIMB phenomena in Cheetah test set: wh-phrases

examples will be provided in Table 5.10.

There are no major differences between the grammars as far as treatment of wh-phrases, illustrated in Table 5.5, is concerned. The only difference is that MRS representations in Cheetah do not include information on whether the clause is declarative or interrogative. The gCLIMB grammar does indicate that a question is asked when one of the arguments or modifiers is a question word.

The difference in semantics for the example with the subordinate clause is as expected based on the different goals of Cheetah and gCLIMB. First, the expletive *es* (there) is not present in the gCLIMB MRS, but represented as an underspecified argument of *gibt* (gives) in Cheetah's output. Second, the subordinate clause is analysed as a scopal modifier by the gCLIMB grammars, whereas Cheetah (which does not deal with scopal relations) represents *weil* (because) as a predicate taking the index of *trinkt* (drinks) as its first argument and the index of *gibt* (gives) as its second argument.

Relative clauses also lead to a different output in the two grammars, as expected based on the general differences in semantic representation. Figures 5.3 and 5.4 represent the semantic output for (19) provided by Cheetah and gCLIMB, respectively.

$$\begin{bmatrix} \text{mrs} & \\ \text{LTOP} & \boxed{h1}\ \text{h} \\ \text{INDEX} & \boxed{e2}\ [\text{e}] \\ \text{RELS} & \left\langle \begin{matrix} \begin{bmatrix} \text{\_d\_der\_det\_rel} & \\ \text{LBL} & \boxed{h3}\ \text{h} \\ \text{ARG0} & \boxed{x4}\ [\text{x}] \\ \text{ARG1} & \boxed{x5}\ [\text{x}] \end{bmatrix}, \begin{bmatrix} \text{\_n\_wein\_det\_rel} & \\ \text{LBL} & \boxed{h6}\ \text{h} \\ \text{ARG0} & \boxed{x5} \end{bmatrix}, \begin{bmatrix} \text{\_rel\_der\_rel} & \\ \text{LBL} & \boxed{h7}\ \text{h} \\ \text{ARG0} & \boxed{x8}\ [\text{x}] \end{bmatrix}, \begin{bmatrix} \text{\_pper\_er\_rel} & \\ \text{LBL} & \boxed{h9}\ \text{h} \\ \text{ARG0} & \boxed{x10}\ [\text{x}] \end{bmatrix}, \\ \begin{bmatrix} \text{\_v\_trinken\_s-oa\_rel} & \\ \text{LBL} & \boxed{h11}\ \text{h} \\ \text{ARG0} & \boxed{e12} \\ \text{ARG1} & \boxed{x10} \\ \text{ARG2} & \boxed{x8} \end{bmatrix}, \begin{bmatrix} \text{rel-clause} & \\ \text{LBL} & \boxed{h13}\ \text{h} \\ \text{ARG0} & \boxed{e14}\ [\text{e}] \\ \text{ARG1} & \boxed{x5} \\ \text{ARG2} & \boxed{e12} \end{bmatrix}, \begin{bmatrix} \text{\_v\_sein\_s-pd\_rel} & \\ \text{LBL} & \boxed{h15}\ \text{h} \\ \text{ARG0} & \boxed{e2} \\ \text{ARG1} & \boxed{x5} \\ \text{ARG2} & \boxed{e16}\ [\text{e}] \end{bmatrix}, \begin{bmatrix} \text{\_a\_herrlich\_mo\_rel} & \\ \text{LBL} & \boxed{h17}\ \text{h} \\ \text{ARG0} & \boxed{e16} \\ \text{ARG1} & \boxed{u18} \end{bmatrix} \end{matrix} \right\rangle \end{bmatrix}$$

Figure 5.3: MRS output Cheetah for *Der Wein den er trinkt ist herrlich. (see Table 5.6, example on "relative clauses" for glosses and a translation)*

(19)	Der	Wein	den	er	trinkt
	the.NOM.M.SG	wine	REL.ACC.M.SG	PRO.NOM.3.SG	drink.3.SG
	ist	herrlich.			
	be3.SG	delicious			

‘The wine that he is drinking is delicious.’ [deu]

Cheetah treats the two clauses, including the subject NP and its coreferring relative pronoun, as separate entities. They can be linked to each other by the additionally inserted elementary predication *rel-clause*, which takes the modified noun as its first argument and the event described by the relative clause as its second argument. This semantic representation is not strictly correct (the relative structure does not introduce an additional event other than the event expressed by the clause and the relative pronoun and modified noun have the same referent and should thus have the same index). Cheetah's representation has, however, some advantages over pure syntactic dependencies. The ARG1 of the relative clause and main verb are coindexed. Because we know that the ARG1 of the relative clause should refer to the same entity as the relative pronoun, we can infer that the subject of *ist* ("is") is the same entity as the object of *trinkt* ("drinks"). All relevant information

phenomenon
Example
subordinates
Weil es Wein gibt trinkt er Wein. because EXPL wine give.3S.PRES drink.3S.PRES PRO.3S.M.NOM wine *Because there is wine, he drinks wine.* [deu]
relative clauses
Der Wein den er trinkt ist herrlich. the.3S.NOM wine REL.M.ACC PRO.3S.M.NOM drink.3S.PRES be.3S.PRES delicious *The wine he is drinking is delicious.* [deu]
relative clauses with PP
Der Mann mit dem er Wein trinkt lacht. the.3S.NOM man with REL.M.DAT PRO.3S.M.NOM wine drink.3S.PRES laugh.3S.PRES *The man with whom he drinks wine is laughing.* [deu]
relative clause introduced by determiner
Der Mann dessen Wein er trinkt lacht. The.3S.NOM man REL.DET wine PRO.3S.M.NOM drink.3S.PRES laugh.3S.PRES *The man whose wine he drinks is laughing.* [deu]

Table 5.6: gCLIMB phenomena in Cheetah test set: subordinates and relative clauses

```
[mrs
 LTOP   [h1] h
 INDEX  [e2] [e]
 RELS   < [_def_q_rel      ], [_wein_n_rel ], [_pronoun_n_rel ], [exist_q_rel     ],
          [LBL   [h3] h    ]  [LBL  [h7] h ]  [LBL  [h8] h    ]  [LBL   [h10] h  ]
          [ARG0  [x5] [x]  ]  [ARG0 [x5]   ]  [ARG0 [x9] [x]  ]  [ARG0  [x9] [x] ]
          [RSTR  [h4] h    ]                                     [RSTR  [h11] h  ]
          [BODY  [h6] h    ]                                     [BODY  [h12] h  ]
          [_trink_v_rel    ], [_herrlich_mod_rel]                                  >
          [LBL   [h7] h    ]  [LBL  [h14]       ]
          [ARG0  [e13] [e] ]  [ARG0 [e2]        ]
          [ARG1  [x9]      ]  [ARG1 [x5]        ]
          [ARG2  [x5]      ]
 HCONS  < [qeq        ], [qeq         ] >
          [HARG  [h4] ]  [HARG  [h11] ]
          [LARG  [h7] ]  [LARG  [h8]  ]
]
```

Figure 5.4: MRS output CLIMB for *Der Wein den er trinkt ist herrlich.* (see Table 5.6, example on "relative clauses" for glosses and a translation)

phenomenon
Example
adverbs modifying PP
Wein trinkt er nur bei Käse. wine drink.3S.PRES PRO.3S.M.NOM only with cheese *He drinks wine only with cheese.* [deu]
adverbs modifying determiner
Selbst der Wein von Peter trinkt er. even the.M.NOM wine of Peter drink.3S.PRES PRO.3S.M.NOM *He drinks even Peter's wine.* [deu]
genitive modifiers
Der Wein des Mannes ist herrlich. the.M.NOM wine the.M.GEN man.GEN be.3S.PRES delicious *The man's wine is delicious.*
NP-adverbs
Dieses Mal trinkt er Wein. this time drinks he wine *This time, he is having wine.*

Table 5.7: gCLIMB phenomena in Cheetah test set: modification

can thus be derived from Cheetah's output.

The gCLIMB representation on the other hand does not explicitly indicate that there was a relative clause in the sentence, but directly uses the same index for the two arguments.

This example also indicates a difference in treatment of predicative adjectives. The gCLIMB representation makes *der Wein* ("the wine") a direct argument of *herrlich* ("delicious"), whereas Cheetah relates *herrlich* and *der Wein* by making them both arguments of *sein* ("to be"). The ARG1 of *herrlich* is underspecified in the Cheetah representation. This difference is expected given Cheetah's goal to produce syntactic dependencies.

Table 5.7 presents various forms of modification. Semantic representations of these structures are relatively straight-forward and similar between the grammars, though a small difference can be observed in the treatment of genitive modifiers. Cheetah introduces a relation called *gen-mod* to indicate the genitive modifier *des Mannes* ("of the man"). This is similar to the treatment of relative clauses, where the presence of a specific syntactic structure is also indicated by an inserted relation explicitly referring to this structure.

phenomenon
Example
comparatives
Dein Wein ist herrlicher als mein Wein. your wine is delicious.COMP than my wine *Your wine is more delicious than my wine.*
VP-ellipsis
Er hat herrlicheren Wein getrunken als Du. he has delicious.COMP wine drink.PTC than you *He drank more delicious wine than you did.*

Table 5.8: gCLIMB phenomena in Cheetah test set: comparatives

The gCLIMB representation follows a paraphrasing approach and is identical to the representation of *Der Wein von dem Mann* ("the wine of the man").

Table 5.8 represents the comparative examples that were already discussed in Section 5.1.2.

phenomenon
Example
passives
Der Wein wird von ihm getrunken. the wine AUX of him drink.PTC *The wine is being drunk by him.*
vocatives
Peter, der Wein ist herrlich! Peter, the wine is delicious *Peter, the wine is delicious!*
compound nouns
Frau Schmidt trinkt Wein. Mrs Schmidt drinks wine *Mrs Schmidt is drinking wine.*
truncated coordination
Vier- bis fünfhundert Männer lachen. Four- to five hundred man laugh *Four to five-hundred man are laughing.*
possessives
Dein Wein ist herrlich. your wine is delicious *Your wine is delicious.*

Table 5.9: gCLIMB phenomena in Cheetah test set: various

As mentioned above, passives form one of the exceptions where Cheetah diverts from providing strict syntactic dependencies. Cheetah correctly links the subject of the passive to the ARG2 of the verb (i.e. to what would be the object in the active counterpart). If the sentence includes a demoted subject such as *von ihm* ("like him") in the example above, it is treated like regular modification. The gCLIMB output includes both a representation where the demoted subject is linked to the ARG1 of the sentence as well as the Cheetah variant, where ARG1 remains underspecified and *von ihm* is a regular modifier. In addition to the differences shown above, gCLIMB introduces a possessive relation for *dein* ("your") whereas Cheetah treats it as a regular determiner with *poss-dein-rel* as its predicate.

Table 5.10 presents a set of phenomena that are not included in the Cheetah grammar, but do have a gCLIMB analysis.

Based on the evaluation presented above, it seems that gCLIMB covers more linguistic phenomena than Cheetah. However, Cheetah's development did not merely consist in covering as much data from the development set as possible. Cramer also worked on improving efficiency and coverage on the TiGer corpus. Unsurprisingly, Cheetah performs significantly better on this corpus. In order to establish whether gCLIMB can nevertheless be seen as a resource of comparable complexity, an additional evaluation was carried out on the Babel corpus (Müller, 1996, 2004). This corpus contains a set of complex linguistic phenomena and was not taken into account during the development of Cheetah or gCLIMB. I will first briefly discuss performance of gCLIMB on TiGer and then present results on the Babel corpus.

Coverage on TiGer with Cheetah and gCLIMB

I carried out an additional evaluation on sentences 1,001-2,000 from the TiGer treebank (Brants et al., 2002) with both gCLIMB and Cheetah. This experiment is described in more detail in Section 5.3.2. Here I will briefly preview those parts that are relevant to the current discussion.

The lexicon that Cheetah read off the TiGer treebank during Cramer's (2011) second development stage was incorporated into gCLIMB for this purpose.

phenomenon Example
auxiliary-flip
...daß er den Wein wird trinken können ...that he the wine AUX drink.inf can.inf *...that he will be able to drink the wine*
participle as adjective
Der getrunkene Wein ist herrlich. The drink.PTC wine was delicious *The wine that was drunk was delicious*
impersonal passive
Es wird getanzt. It AUX dance.PTC *There was dancing*
object fronting and control[6]
Den Wein versucht er zu trinken. The wine tries he to drink *He is trying to drink the wine*
extraposition
Er hat den Wein versucht zu trinken. He has the wine try.PTC to drink *He tried to drink the wine*
VP coordination
Er lacht und will den Wein trinken. He laughs and wants the wine drink *He laughs and wants to drink the wine*
Polar question coordination
Atmet der Wein und will er den Wein trinken? Breaths the wine and wants he the wine drink *Is the wine breathing and does he want to drink the wine?*
Long distance dependency, argument extraction
Was denkt Peter dass er getrunken hat? What thinks Peter that he drink.PTC has *What does Peter think he drank?*
Long distance dependency, adjunct extraction
Wie denkt Peter dass sie den Wein getrunken hat? How thinks Peter that she the wine drink.PTC has *How does Peter think she drank the wine?*

Table 5.10: Phenomena additionally covered by gCLIMB

I extended the metagrammar to generate a lexical hierarchy that cross-classifies subcategorisation lists for verbs and type names of items in Cheetah's derived lexicon and morphology were adapted to fit this hierarchy. Sentences 1,001-2,000 were then parsed with both Cheetah and gCLIMB grammars.

Cheetah covers 47.3% of the sentences, whereas gCLIMB's best grammar currently covers 21.3%. In later stages of grammar development, Cramer used the first 500 sentences as a development corpus and later carried out some optimisation based on the most frequent errors in following sentences. This may account for some of the difference, but the most common cause for not arriving at a full parse are time outs or hitting the maximum number of edges. Upon inspection of the first 100 sentences in this evaluation set, it turned out that most sentences did not exhibit phenomena not covered by gCLIMB. This strongly indicates that the difference lies in efficiency of the grammars. In particular, the interaction between analyses and lexicon is likely to play a role. Cheetah was designed to be used with a lexicon read off the TiGer treebank. The development of gCLIMB only aimed for covering analyses from the development corpus. Reproducing the entire process of Cheetah development was out of scope for this study and could not be used for comparison, because gCLIMB's output cannot be compared directly to Cheetah. As mentioned above, Section 5.3.2 will discuss the interaction between grammar and lexicon in more detail.

Evaluation on the Babel testset

The Babel test suite was created by Stefan Müller to evaluate his German grammar created in Babel (Müller, 1996, 2004). It consists of 547 positive examples and 182 negative examples illustrating a wide range of linguistic phenomena. As mentioned above, the test suite was not taken into consideration while developing Cheetah or gCLIMB. The purpose of this evaluation was to compare the coverage of these grammars on an independently developed set of linguistically motivated examples.

grammar	**Coverage**		**overgeneration**
	dependencies	**MRS**	
Cheetah	54.7%	17.2%	28.1%
gCLIMB (aux-rule)	60.5%	51.2%	25.4%

Table 5.11: Coverage and overgeneration on Babel corpus

Just like in earlier evaluations, lexical coverage was not taken into account in the evaluation. Cheetah's static lexicon was therefore extended to cover the items in the test suite and gCLIMB was adapted so that it could combine lexical items defined in *choices* with lexical items defined as part of the Cheetah lexicon. No new types were defined in Cheetah or CLIMB except for two lexical types in Cheetah. The types in question exhibit intransitive structures where the only argument bears dative or, respectively, accusative case. They could be integrated in gCLIMB easily without adding new type definitions in the metagrammar. Because they lead to a significant increase in coverage and do not reflect the complexity of the grammar, they were added to Cheetah as well.

The results of this evaluation are presented in Tables 5.11-5.13.[7] Table 5.11 provides indication of treebanked coverage and overgeneration. The first coverage column presents the percentage of positive examples for which the semantics provides the correct dependencies. The second coverage column indicates in which cases a correct MRS is provided.[8] Correct MRS representations always provide correct dependencies. The results in the second column thus form a subset of the first. The final column indicates overgeneration on ungrammatical examples of both grammars.

The number of examples associated with a specific phenomenon varies in the set. I therefore examined examples that did not get a correct analysis in

[7]The grammars that were used for this evaluation, the test suite and an indication of examples illustrating the phenomena mentioned in the tables can be found in `svn://lemur.ling.washington.edu/shared/matrix/branches/antske-germanic-development/phd_evaluation/babel-experiments`.

[8]Semantics of determiners and bare NPs were ignored in the evaluation of MRSs produced by Cheetah, because they are all different from general assumptions in MRS and occur in every sentence.

both examples and associated them with phenomena. Table 5.12 provides an overview of phenomena where performance differs between gCLIMB and Cheetah. The columns indicate whether the grammars can handle all examples related to a phenomenon in the corpus (complete) or part of the phenomena (partial).

gCLIMB outperforms Cheetah				
Phenomenon	**Cheetah**		**gCLIMB**	
	partial	complete	partial	complete
demoted subject passive				x
adjective as noun				x
raising, control (semantics)			x	
verbs as adjectives			x	
impersonal passives			x	
verb order variation			x	
partial VP fronting			x	
attachment rel. clause	x			x
flexible position adverbs	x			x
Cheetah outperforms gCLIMB				
Phenomenon	**Cheetah**		**gCLIMB**	
	partial	complete	partial	complete
particles and matrix questions		x		
embedded extraposition		x		
coordination of relative clauses	x			
VP coordination		x	x	
multiple complex arguments		x	x	
extraposition		x	x	

Table 5.12: Comparison of coverage of phenomena in Babel corpus

Table 5.13 presents phenomena that overgenerate in one language, but not the other. Phenomena marked by an asterix (*) lead to an increase of coverage as compared to the other grammar. An overview of phenomena for which neither grammar finds the correct analysis or both grammars overgenerate can be found in Appendix C.

The Babel corpus provides a set of examples that represent challenging linguistic behaviour in German. The results for both grammars can be con-

gCLIMB only
adjective inflection when taking complements
subject extraction
multiple expletive complements
case restriction when raising*
verb used as adjective*
impersonal passive*
Cheetah only
particles in the Mittelfelt
multiple relative clauses, one extracted
auxiliary selection for perfectives
adjective endings
determiner restrictions imposed by nouns
adjectives with pp complements*
irregular coordination*

Table 5.13: Comparison of overgeneration

sidered good results given that the corpus was not taken into account during their development. gCLIMB slightly outperforms Cheetah on coverage, overgeneration and number of examples handled correctly. These results show that gCLIMB can thus be considered comparable to Cheetah when looking at coverage of linguistic phenomena.

5.1.4 Discussion and Observations

The previous subsections described the process of developing grammars in CLIMB that cover the same phenomena in Cheetah's development set as Cheetah's core grammar. This undertaking served the purpose of investigating whether CLIMB can be applied to large scale grammar development and of getting a first indication of the impact on using CLIMB. As mentioned above, it is not easy to answer these questions. I will explain in this subsection why it is not possible to set up a scientific test to answer these questions. This does not mean that the implementation of gCLIMB is meaningless as an evaluation. The observations that can be made based on this effort do provide an indication of the impact of CLIMB for developing resource grammars.

Discussion

There are several problems in answering the questions addressed above. First and foremost, there is no clear definition of a large scale grammar compared to a medium sized grammar or a small grammar. Should this depend on properties of the grammar such as the number of types, lexical items, syntactic rules and lexical rules? Or rather on properties of the grammar such as the linguistic phenomena it covers or the coverage it has on open text? All three criteria provide some indication of the size and scale of the grammar, but none of them offers a well defined line as to when a grammar can be considered to be "large scale".

The challenge of comparing the method to traditional grammar engineering is even harder to overcome. It is impossible to provide conditions for a fair comparison between two methods of grammar development, because there are several factors that influence development speed and quality of the grammar that cannot be controlled for.

The engineer in charge of implementing the grammar probably has the most influence on development speed and quality of the resulting grammar. Experience and recent practice are two of the main factors that makes one engineer faster and more accurate than another. It is therefore not possible to use the same engineer in two projects (even if time would allow for this), because the engineer will be more experienced the second time. This effect is even stronger when both projects use the same language and materials: the second time will be a repetition of the earlier project, resulting in a major speed-up. The closer the language, the greater the impact of previous experience. On the other hand, the phenomena of the language and whether they are known and treated in the literature also influences development time. It is therefore not possible to compare projects for languages that have few linguistic properties in common.

Using the same engineer is therefore not a reasonable options to compare the two methods. As a result, one of the most influential factors on the development project already differs in any comparison experiment. Other factors that influence the development process such as the language, materials from the literature and other grammars that are available can be controlled for.

Ideally, several grammar development efforts using the CLIMB method would be compared with several efforts not using it. The main difference is expected to be seen on long-term grammar development and many grammar engineering efforts would need to be compared in order to get a reliable indication of the impact of the method. It is therefore not feasible within the timeframe of a dissertation to set up a scientifically sound experimental environment to compare CLIMB to standard grammar engineering. An opportunity for such a comparison may be found in the future when more researchers have adopted the CLIMB method and their efforts can be compared to those of other grammar engineers.

Observations

Despite the fact that the development of gCLIMB cannot lead to scientific proof, it can provide an indication of its scalability and the impact of using CLIMB. This section has described a wide range of phenomena covered by gCLIMB. As far as quantity is concerned,[9] gCLIMB was built on top of a version of the Grammar Matrix that had 3,651 lines of code in the linguistic libraries. The metagrammar's libraries now consist of 13,032 lines of code, compared to 6,250 lines of code in the current version of the Grammar Matrix customisation system.[10] The choices files for these grammars have 1,970 lines of definitions (for small grammars used for testing) up to 3,657 lines (for creating the complete Cheetah core grammar and incorporating its lexicon). A customised grammar consists of approximately 7,300 lines of code and 1,300 language specific types. The phenomena the grammars cover show that CLIMB can be used to write grammars of a reasonable level of complexity.

[9]These numbers have previously been reported in Fokkens and Bender (2013).

[10]The lines of code where taken from version number 26977 of the Grammar Matrix.

The size of the metagrammar and customised grammars indicate that it can be used to write grammars of a decent size.

Several observations can be made based on the impact of the method. Several influential factors for creating Cheetah's core grammar are known and were controlled for while developing gCLIMB. Cramer (2011) used literature on German in HPSG, but GG or other implementations of HPSG based grammar for German were not used. I followed the same principle for CLIMB. Moreover, Cramer also implemented his grammar as part of his PhD thesis and was a native speaker of Dutch living in Germany at the time of development. This gives reason to believe we are somewhat comparable in skills and development speed. Section 5.1.2 pointed out that variations for other Germanic languages were included during the earlier stages of developing gCLIMB and gCLIMB aims for semantic representations rather than representations that mirror syntactic dependencies. These two differences between gCLIMB and Cheetah both form additional efforts that were made for gCLIMB and not for the development of Cheetah.

Cramer (2011) on the other hand aimed for efficiency of the grammar and designed the grammar so that it could easily be used with the TiGer treebank. It is, however, not possible to determine in hindsight what portion of the one person year spent to develop Cheetah was invested in this effort.

The overall development time for gCLIMB was less than six person months compared to one year for Cheetah's core grammar (Cramer, 2011). The one year of development for Cheetah's core grammar was generally received as an acceptable time for such an undertaking by the DELPH-IN community. The gCLIMB grammars cover the same phenomena in Cheetah's development set as Cheetah partially covering variations for other languages and capturing more complex semantic aspects than Cheetah and slightly outperformed Cheetah on an external testset of linguistic phenomena. The fact that it was developed in less than half the time does not allow us to conclude that using CLIMB speeds up the grammar engineering process for reasons explained above. However, even if the shorter time is purely due to the different engineer or if using CLIMB even slowed the development process down, this result does show that a potential negative impact on development speed remains within an acceptable range.

5.2 Comparative efficiency evaluation

This section presents a comparative evaluation of the grammars included in gCLIMB. Fokkens (2011a) presented a comparison in efficiency for the toy grammars that formed the basis of this evaluation. In this section, we will see how differences in efficiency developed as the grammars produced by gCLIMB covered more phenomena. I will compare three grammars containing the argument composition analysis (henceforth **arg-comp**), the auxiliary and verbal construction analysis (**aux+verb**) both using the Grammar Matrix-based flat analysis for verb second word order or the filler-gap analysis (**filler-gap**). Descriptions of the individual analyses can be found in Chapter 4.

5.2.1 Parsing efficiency

As explained in Section 4.4.2, German auxiliaries share the arguments of their verbal complements. Under the argument composition analysis, auxiliaries are defined in the lexicon to take a verbal complement and add whatever remains on the SUBJ and COMPS list to its own argument lists. The argument lists of auxiliaries are thus (partially) underspecified. As a result, any possible constituent in the sentence is a potential subject or complement of the auxiliary. The aux+verb analysis based on Bender's analysis for Wambaya (Bender, 2010) is designed to address the inefficiency caused by this underspecified argument list. Under this alternative analysis, the auxiliary only selects a verbal complement and cannot combine with any other element in the sentence. Auxiliary and complement are combined using a special construction, which makes sure the auxiliary picks up any remaining elements on the SUBJ and COMPS list of its verbal complement. Fokkens (2011a) presents the first comparative evaluation in efficiency for Germanic languages using toy grammars that cover basic phenomena in German, Dutch, Flemish and

Danish.

Average Performed Tasks						
Language	**Complete Coverage**			**No Split Cluster**		
	arg-comp	aux+v	red.	arg-comp	aux+v	red.
Dutch	524	149	71.6%	480	134	72.1%
Flemish	529	150	71.6%	483	137	71.6%
German	684	148	78.4%	486	136	72.0%
Average	579	149	74.3%	483	135.7	71.9%
Average Created Edges						
Language	**Complete Coverage**			**No Split Cluster**		
	arg-comp	aux+v	red.	arg-comp	aux+v	red.
Dutch	58	25	57.9%	52	25	51.9%
Flemish	58	26	55.2%	52	25	51.9%
German	67	23	65.7%	52	24	53.8%
Average	61	24.7	59.6%	52	24.7	52.5%
Average Memory Use (kb)						
Language	**Complete Coverage**			**No Split Cluster**		
	arg-comp	aux+v	red.	arg-comp	aux+v	red.
Dutch	9691	6692	30.9%	8944	6455	27.8%
Flemish	9716	6717	30.9%	8989	6504	27.6%
German	10289	5675	44.8%	8315	5468	34.2%
Average	9898.7	6361.3	35.7%	8749.3	6142.3	29.8%
Average CPU Time (s)						
Language	**Complete Coverage**			**No Split Cluster**		
	arg-comp	aux+v	red.	arg-comp	aux+v	red.
Dutch	0.04	0.02	-	0.03	0.01	-
Flemish	0.04	0.02	-	0.03	0.01	-
German	0.06	0.01	-	0.04	0.01	-

Table 5.14: Differences in efficiency as found in Fokkens (2011a)

The toy grammars used in the original comparative evaluation showed a clear advantage in efficiency for the aux+verb analysis. Table 5.14 provides an overview of the results presented in Fokkens (2011a). Small grammars for German, Dutch and Flemish showed an average reduction of 73.1% in performed tasks, 56.1% in produced passive edges and 32.8% in memory when the aux+verb grammars were used as compared to the arg-comp analysis.

The average CPU time was too short for a meaningful comparison with `[incr tsdb()]`. Therefore no reduction rate and average numbers are provided for CPU time.

Stage	**Phenomena**	**repository**	**test set**	
			pos	neg
0	phenomena from Fokkens (2011a)	19578	114	157
1	adjectives	19589	135	220
2	polar questions, negation	19613	161	240
3	adverbs, adpositions and cps	19617	206	287
4	copula (predicative adjectives and pps)	19626	248	334
5	subject control	19672	279	375
6	object raising	19732	300	416
7	wh-sentences, filler-gap analysis	19978	349	463
8	argument structures, subordinates	20019	427	539
9	passives and relative clauses	20189	485	607
10	possessives, extraposition, comparatives	20975	508	630

Table 5.15: Development stages and phenomena that were treated in them

The grammars used in Fokkens (2011a) only covered basic word order facts, intransitive, transitive and ditransitive verbs and no modification, subordinates or complex predicates. This section presents an evaluation that examines how efficiency of the alternative analyses is affected as the grammars cover more phenomena. For the purpose of this evaluation, the process of developing gCLIMB up to covering Cheetah's core grammar was split up in 10 stages based on complete working versions checked into the gCLIMB repository. They are presented in Table 5.15 together with their repository version number and the number of positive and negative examples in the evaluation set for each stage.

The graphs in Figures 5.5-5.8 illustrate the performance of grammars with alternative analyses on the development set at each stage. The filler-gap analysis was implemented in stage 7 and was only taken up in the evaluation from that stage. Figure 5.5 indicates the average number of tasks carried out per sentence, Figure 5.6 provides the average number of etasks, Figure 5.7 illustrates the average memory that was used and the average CPU time

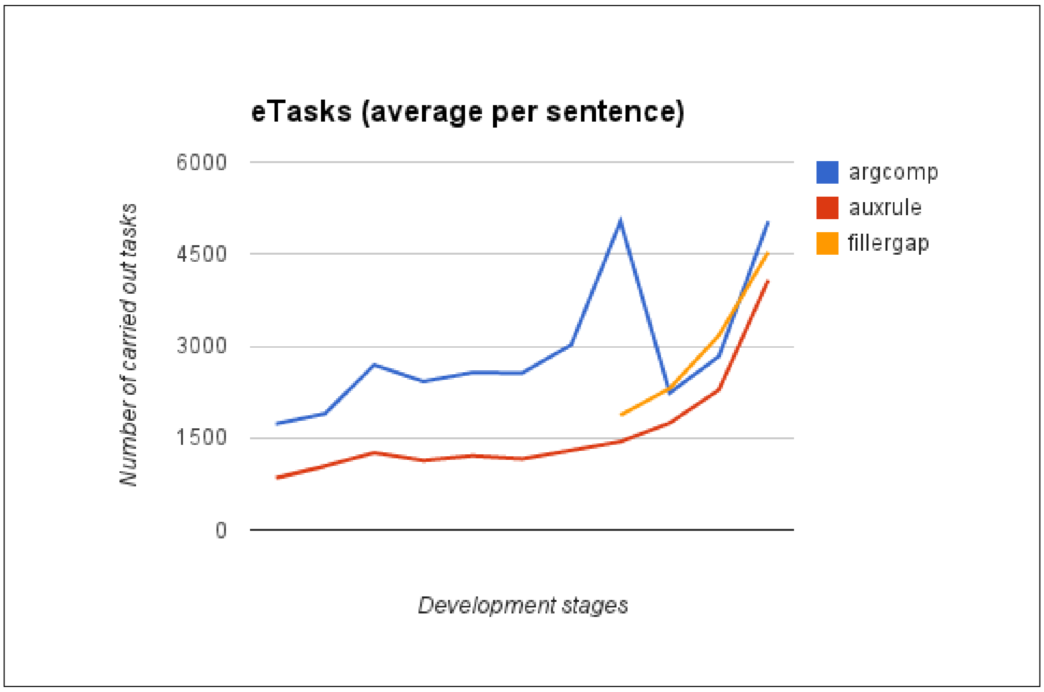


Figure 5.5: Number of tasks carried out per sentence for different grammars

per sentence is found Figure 5.8. For all performance related measures, the same pattern can be observed. I will use the average number of edges (Figure 5.5) to illustrate how each analysis influences performance in different development stages.

The number of edges used by the arg-comp analysis increases significantly as object raising and wh-clauses are added to the grammar. The efficiency of the aux+verb analysis, on the other hand, remains relatively stable. After this significant increase, additional constraints to prevent further increase of edges that do not lead to a parse were added to the grammar. Rules that combine heads with their subject or a complement where restricted to only take non-head daughters that can occur in this position in German. In other words, determiners, adjectives, adverbs, adpositions and verbs where excluded from being subjects and determiners and adverbs from becoming complements. This lead to a significant improvement in efficiency as can be seen by the sharp drop in average edges per sentence in the graph. The aux+verb analysis reveals a slow increase in edges per sentence as the grammar covers more

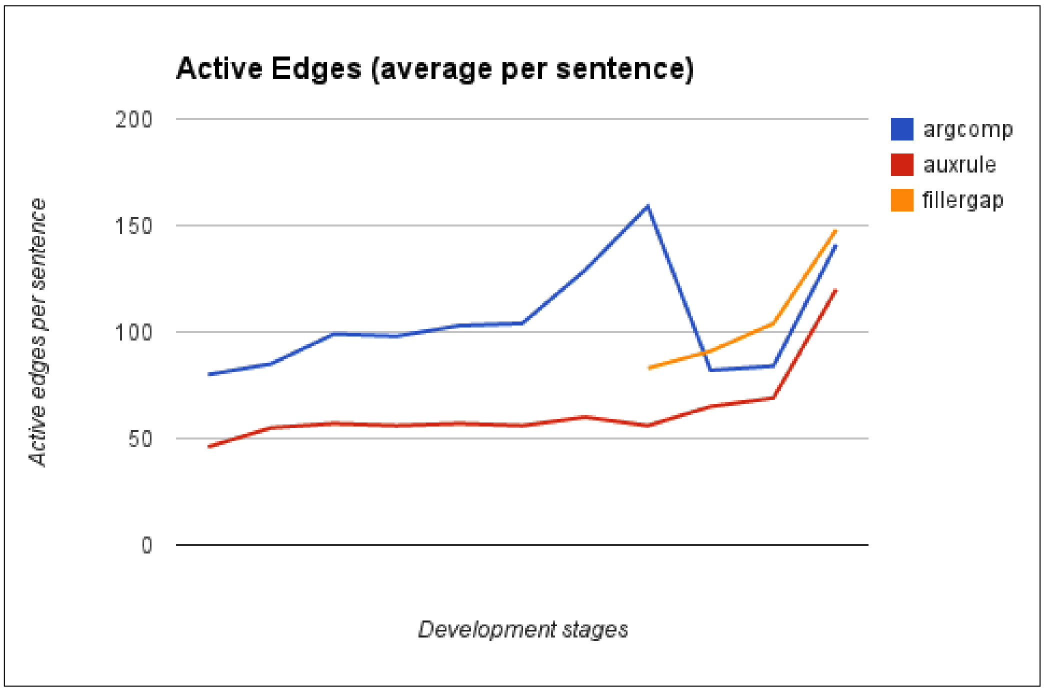


Figure 5.6: Number of active edges used per sentence for different grammars

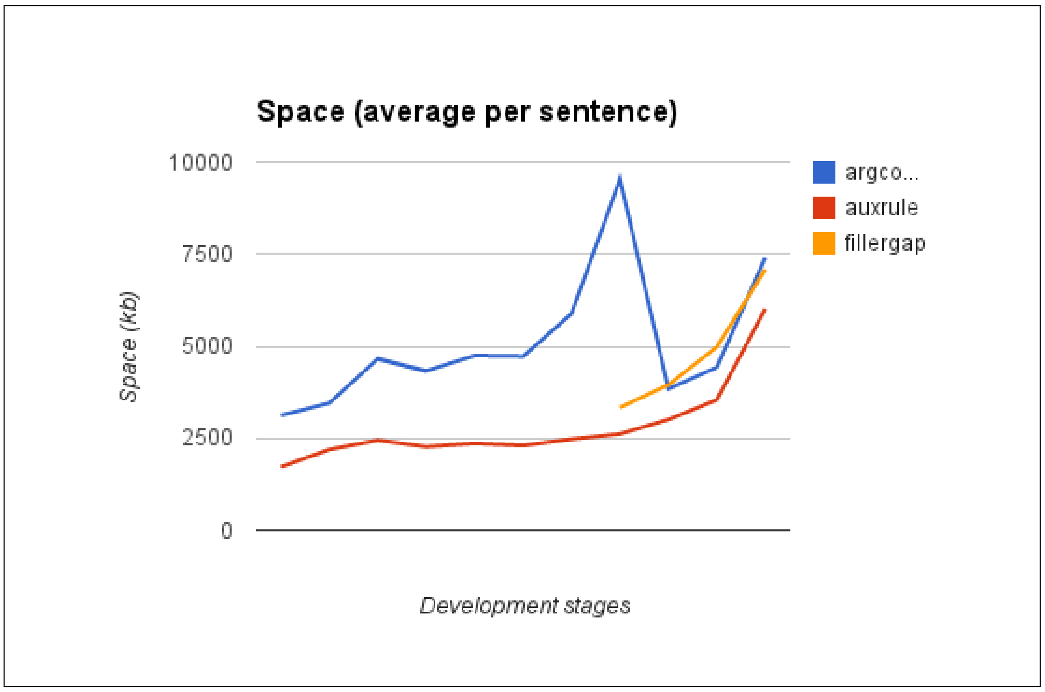


Figure 5.7: Average Memory used per sentence for different grammars

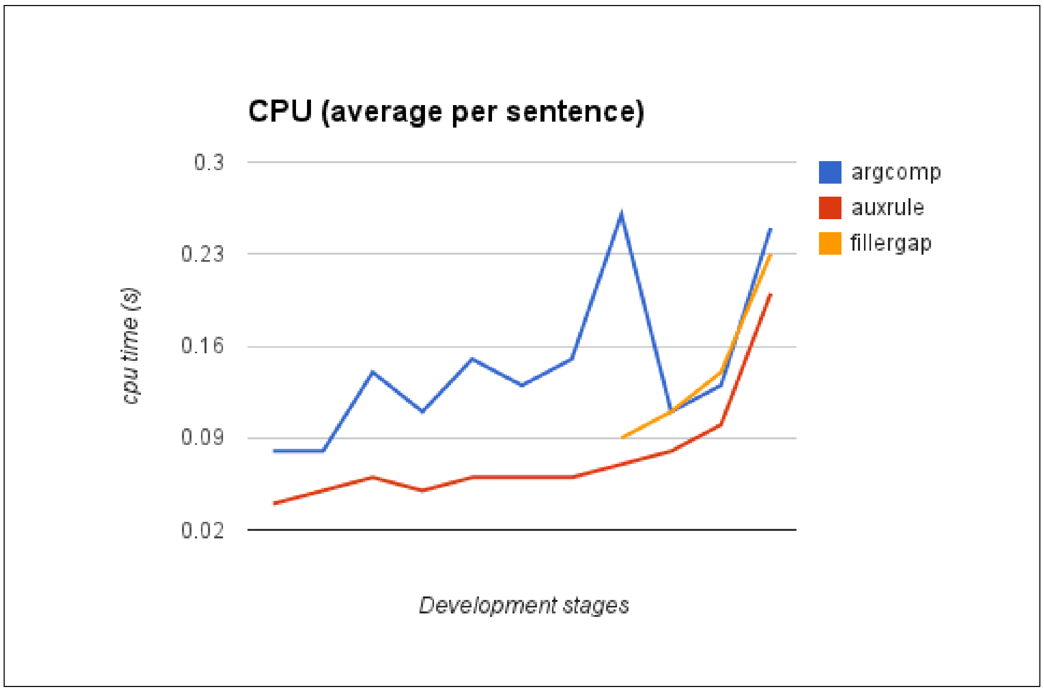


Figure 5.8: Average CPU time per sentence for different grammars

phenomena with a noticeable increase as relative clauses and extraposition are added to the grammar. The same increase for these two phenomena is observed for the arg-comp and filler-gap analysis.

The restrictions on the non-head daughter lead to significant differences in efficiency for the arg-comp analysis. A draw-back of these additional constraints is that they go against the lexicalist character of HPSG. Schemata (as the rules that combine heads with their complements or modifiers are called in theoretical HPSG) are supposed to be general and constraints on which words may be combined by them are supposed to come from the syntactic properties of the lexical items. It is on the other hand quite common for DELPH-IN grammars not to follow the idea of a small set of universally applicable schemata, as explained in Section 2.2.1, Chapter 2.

The fact that determiners do not occur as arguments or that all subjects bear nominative case are properties of the German language. It therefore seems acceptable to give the efficiency of the grammars a boost with these additional constraints. There are thus two defensible visions on whether non-head-daughters should be constraints in the grammar or not and one of the main strengths of CLIMB is that it provides a framework to test both options. Table 5.16 presents the results of this comparative study. The results compare the performance of individual analyses on the 508 positive examples of the full development set.[11]

Average Performed Tasks		
Analysis	**constraints**	**no constraints**
filler-gap	4530	11269
arg-comp	4831	14388
aux+verb	3937	4042
Average Created Edges		
Analysis	**constraints**	**no constraints**
filler-gap	156	438
arg-comp	141	461
aux+verb	120	120
Average CPU Time (s)		
Analysis	**constraints**	**no constraints**
filler-gap	0.22	0.68
arg-comp	0.23	0.92
aux+verb	0.19	0.21
Average Memory Use (kb)		
Analysis	**constraints**	**no constraints**
filler-gap	7184	26932
arg-comp	7140	33166
aux+verb	5809	5979

Table 5.16: Comparative efficiency of grammars with or without constraints on the non-head daughter

The results reveal a striking difference in impact of these additional constraints between the grammars that use argument composition (i.e. the filler-gap and arg-comp grammars) on the one hand and the grammar using the aux+verb construction (aux+verb) on the other hand. Whereas efficiency is hardly influenced by these constraints when the auxiliary+verb construction

[11]These experiments were carried out with CLIMB repository version 26558. The choices files used are included in the repository.

Average Performed Tasks		
Analysis	**constraints**	**no constraints**
filler-gap	13.1%	64.1%
arg-comp	18.5%	71.9%
Average Created Edges		
Analysis	**constraints**	**no constraints**
filler-gap	23.1%	72.6%
arg-comp	14.9%	74.0%
Average CPU Time		
Analysis	**constraints**	**no constraints**
filler-gap	13.6%	69.1%
arg-comp	17.4%	77.2%
Average Memory Use		
Analysis	**constraints**	**no constraints**
filler-gap	19.1%	77.8%
arg-comp	18.6%	82.0%

Table 5.17: Improvement in efficiency when aux+verb analysis is compared to other analyses

is used, the constraints on non-head daughters have a great impact on the results when the grammar uses argument-composition. The number of tasks decreases by a factor of 2.49, the average number of edges used by a factor of 2.81, CPU time by 3.09 and memory by 3.75 for the filler-gap analysis. Differences are even bigger when the flat verb second analysis is combined with argument composition. We see a factor 2.98 decrease in the number of tasks and a factor of 3.27 decrease in number of edges used. The CPU time is 4 times as much if the non-head daughters are not constrained and memory increases by a factor of 4.65.

The difference in efficiency between the aux+verb rule and the arg-comp or filler-gap analysis are presented in Table 5.17. The original experiments reported in Fokkens (2011a) revealed an average reduction of 73.2% in performed tasks, 56.3% in produced edges and 32.9% in memory when using the aux+verb instead of arg-comp analysis. The gain in efficiency for performed tasks is slightly lower (now 71.9%) for these bigger grammars, which may be due to the size of the feature structures which increased for all analyses. The difference in efficiency for both produced edges and memory increased to 74.0% and 82.0%, respectively. The number of edges required is most directly related to the interaction of auxiliary structures with the rest of the grammar. Moreover, hitting the maximum number of permitted edges is a common cause for the parser to fail. I therefore conclude that these results confirm that, without additional constraints on the schemata, differences in efficiency increase as predicted in Fokkens (2011a).

Conclusion on parsing efficiency

The results in this section have shown that the aux+verb analysis remains more efficient than the arg-comp analysis as the grammars generated by gCLIMB cover more phenomena. This difference can be reduced significantly by adding constraints to the grammar restricting which kind of elements (part-of-speech or case marking) may appear as a subject or complement of verbal forms. These constraints have the draw-back that they are not in line with the lexicalist character of HPSG, but seem acceptable because they capture properties of the German language.

Even though the aux+verb analysis was the most efficient grammar in all experiments presented in this section, more investigation is needed to draw firm conclusions about the relative efficiency of the two analyses cross-linguistically. Chapter 6 describes multilingual aspects of CLIMB and there we will see that the aux+verb analysis may not always be the most efficient choice for the grammars that can be generated for Dutch. Furthermore, the efficiency of the arg-comp analysis also suffers from the decision taken in the Grammar Matrix to have separate SUBJ and COMPS lists. GG (Müller and Kasper, 2000; Crysmann, 2005) has only one list called SUBCAT where both the subject and other complements a lexical item selects for are defined. The efficiency issue due to underspecified elements is thus reduced to only one list instead of two. One of the first steps in future research related to gCLIMB will therefore be to include GG implementations as alternatives in CLIMB.

5.2.2 Natural Language Generation

Fokkens (2011a) includes an evaluation of comparative efficiency when alternative analyses are used for natural language generation (NLG). The aux+verb analysis can potentially lead to bigger improvements in efficiency compared to the arg-comp analysis, when used in natural language generation. The reason the arg-comp analysis is suboptimal for efficiency lies in its underspecified argument list. In the case of parsing, this results in the auxiliary combining with any possible word or constituent in the sentence starting from the ones adjacent to it, moving on to potential constituents next to those, etc. In other words, the possible combinations are restricted by the word order in the surface string that is being parsed. When the sentence is generated from the semantics, the order of the words is still open. The auxiliary can thus combine with any word or constituent that is formed and corresponds to part of the semantics. It is thus not surprising that the evaluation in Fokkens (2011a) comparing efficiency for generation is even more significant than that found for parsing. A German sentence with one auxiliary and a ditransitive verb saw an increase in efficiency of at least 98.5% when the aux+verb analysis was used.

This section examines another phenomenon that was observed while conducting the experiments reported in Fokkens (2011a), but was not reported at the time because the result was too preliminary. It concerned variations in efficiency when different word order constraints applied in the Dutch verbal cluster. The word order between auxiliary and verbal complement is free in Dutch, except when modals are in infinitive form (then they must precede their complement) or when verbs taking participle complements are in participle form (then they must follow their complement). A more complete overview of Dutch word order will be given in Chapter 6, Section 6.2. When comparing efficiency in different Dutch structures, it turned out that it was more efficient to generate sentences containing verbs that must precede their complement than generating those with verbs that must follow their complement. The experiments in this section examine whether this observation still holds now that the grammars are more complex.

For each analysis, aux+verb, arg-comp and filler-gap, five grammars based on Dutch were created with gCLIMB. The coverage for Dutch phenomena in gCLIMB is less than for German. The grammars have around 330 language specific types and the language specific file consists of 1,300 lines of code compared to approximately 1,300 language specific types defined in 7,300 lines of code for German grammars. Each grammar has different word order properties. Namely,

- **aux-comp-only**: all auxiliaries must precede their complement
- **comp-aux-only**: all auxiliaries must follow their complement
- **both**: all auxiliaries can follow or precede their complement
- **both-a-c**: both orders are allowed in the grammar, but the auxiliaries in our example must precede their complement
- **both-c-a**: both orders are allowed in the grammar, but the auxiliaries in our example must follow their complement

The grammars are identical except for the differences in word order outlined above. This comparison provides insight into the question of whether the relative order of auxiliary and complement has any influence on efficiency for generation. This effect can be investigated for all three alternative word order analyses. Finally, efficiency differences between the three analyses when used for generation can be investigated. Note that the first two grammars, which restrict auxiliary order for all auxiliaries in the grammar, are not correct grammars of Dutch. They are included in this study to investigate whether a global restriction on auxiliary order has the same effect as a restriction on particular auxiliaries.

An important aspect of this evaluation is the fact that the experimental set-up merely required changing a few lines in the choices files to create grammars that exhibit these linguistic variations in word order. These minor changes in the choices file make sure that implementations concerned with word order are changed, but all other implementations remain stable. The same experiment can be carried out with the same guarantees about implementations not related to word order as the metagrammar grows and covers more phenomena. Facilitating studies like the one presented in this section is one of the strengths of CLIMB.

Experimental setup

The experiment is set up in similar lines as the experiment conducted in Fokkens (2011a). Each grammar was used to generate realisations for eight semantic structures with varying numbers of auxiliaries, arguments and modifiers.[12] Table 5.18 provides an overview of the main elements used in the sentences. The verbs in the table are presented in the order of their subcategorisation, i.e. the verb described on the far left is the finite verb, followed by its verbal complement, followed by the complement of the verbal complement, etc. In all sentences, the subject was a personal pronoun in first person singular and all other NPs present in the sentence consisted of a definite determiner and a noun. The modifiers in the last two structures both modify the main verb. Examples of the canonical realisation of the corresponding Dutch sentences with glosses can be found in appendix B, page 363.

MRS representations including the elements mentioned in Table 5.18 were used as input for generation.

5.2.3 Results and observations

Table 5.19 presents the number of edges required to generate all possible realisations of the input MRS with a particular grammar. Each column represents different word order conditions. The *aux-comp* and *comp-aux* columns provide the number of required edges if the grammar only accepts one fixed

[12]The experiments in Fokkens (2011a) only varied the number of auxiliaries and arguments, since modifiers where not included in the metagrammar at that time. Chapter 6 will provide a description of extensions made to Dutch analyses in the metagrammar since Fokkens (2011a).

Abbreviation	Structure
iv-2aux	auxiliary, raising verb, intransitive verb
iv-3aux	auxiliary, raising verb, auxiliary, intransitive verb
tv-2aux	auxiliary, raising verb, transitive verb
tv-3aux	auxiliary, raising verb, auxiliary, transitive verb
dv-2aux	auxiliary, raising verb, ditransitive verb
dv-3aux	auxiliary, raising verb, auxiliary, ditransitive verb
dv-3aux-1mod	as above, with adverb
dv-3aux-2mod	as above, with additional pp-modifier

Table 5.18: Overview of structures included in NLG experiment

order between auxiliaries and complements. In the third column, both orders are permitted and the auxiliaries used can either precede or follow their complement. The columns *both-a-c* and *both-c-a* represent grammars in which both orders are allowed, but the auxiliaries used in the example must precede (*both-a-c*) or follow (*both-c-a*) their complement. As mentioned above, the first two grammars are not correct grammars of Dutch, because both orders are observed in the language (and permitted depending on the auxiliary and morphological marking of the complement). It is nevertheless interesting to examine these grammars, because we can observe whether differences are caused by the existence of rules for both orders or just the restriction of the auxiliary itself. An upperbound of 100,000 edges was used: if the grammar needed more than 100,000 edges to complete generation, no further experiments were conducted to determine the exact number of edges required.

The results show that for the aux+verb analysis, structures where auxiliaries must precede their complement are indeed more efficient for generation than structures where they must follow it. This holds both for structures in grammars permitting only one order between auxiliary and complement as well as for structures where all auxiliaries impose have a particular word order, even if the grammar in principle also contains auxiliaries exhibiting different word order.

Grammars that use the arg-comp analysis (regardless of whether they use the filler-gap analysis for second word order or not) do not show such a significant difference between the two orders. When the grammar only allows for one order, generation is slightly more efficient when auxiliaries must precede their complement. However, these differences are much smaller than for the aux+verb analysis. When the grammar permits both orders, structures where the auxiliary must follow the verb may even be more efficient. In both cases, the number of required edges for complete generation are relatively close. Generation in these grammars is less efficient when both orders are

auxiliary+verb construction					
	aux-comp	**comp-aux**	**both**	**both-a-c**	**both-c-a**
iv-2aux	253	269	381	314	299
iv-3aux	563	610	1806	818	767
tv-2aux	466	494	705	569	541
tv-3aux	1051	1126	3453	1474	1371
dv-2aux	794	846	1234	975	929
dv-3aux	1827	1958	6280	2558	2383
dv-3aux-1mod	3398	7258	14263	4763	8206
dv-3aux-2mod	19334	62808	99819	26971	70906
argument-composition, no filler-gap					
	aux-comp	**comp-aux**	**both**	**both-a-c**	**both-c-a**
iv-2aux	733	743	848	796	791
iv-3aux	1560	1598	2822	1878	1854
tv-2aux	1967	1977	2163	2049	2046
tv-3aux	4039	4077	6347	4452	4439
dv-2aux	5999	6028	6307	6090	6087
dv-3aux	11706	11867	15665	12191	12178
dv-3aux-1mod	46010	47366	52918	46419	46406
dv-3aux-2mod	>100,000	>100,000	>100,000	>100,000	>100,000
filler-gap and argument composition					
	aux-comp	**comp-aux**	**both**	**both-a-c**	**both-c-a**
iv-2aux	2046	2056	2411	2157	2154
iv-3aux	8649	8681	16119	10770	10759
tv-2aux	7100	7110	8248	7381	7380
tv-3aux	31188	31220	56996	38697	38698
dv-2aux	26645	26655	30511	27522	27521
dv-3aux	>100,000	>100,000	>100,000	>100,000	>100,000
dv-3aux-1mod	>100,000	>100,000	>100,000	>100,000	>100,000
dv-3aux-2mod	>100,000	>100,000	>100,000	>100,000	>100,000

Table 5.19: NLG experiment results: minimal number of edges required

allowed as compared to only one order being allowed, but which of the two orders is permitted hardly has an impact on the efficiency.

When comparing the efficiency of the grammars according to the analysis that is used, we see that the aux+verb analysis reveals significant improvement as compared to both grammars that use argument composition. Differences do not seem to be as extreme as the ones reported in Fokkens (2011a), though it is difficult to tell what the impact is exactly for the longer sentences. For the first grammar that needed more than 100,000 edges for generation, an additional attempt was made with a maximum 200,000 edges which was not sufficient. Because this attempt took over 12 hours, the upperbound was set at 100,000 for the rest of the experiment. It thus remains unclear how many edges would be needed to complete generation for these structures. Therefore the exact impact of the different analyses remains unclear.

In Fokkens (2011a), I predicted that the difference between the argument composition analysis and the aux+verb analysis would increase as the grammars cover more phenomena and sentences become more complex. The results in Table 5.19, however, reveal smaller differences than the original study. This can be explained by the constraints on non-head-daughters mentioned in Section 5.2.1, which reduce the number of edges needed to generate with arg-comp grammars significantly. I ran an additional experiment to see whether the prediction that differences in efficiency increase for more complex grammars does hold when these constraints are not used. An alternative version of the most efficient argument composition grammar (only allowing for *aux-comp* order) without constraints on the non-head daughter was created. The results are presented in Table 5.20. The results of the argument composition analysis with constraints and the aux+verb analysis are repeated for convenience.

When the grammar with constraints is used, the number of edges required drops by 81.2% for sentences with two auxiliaries and an intransitive main verb and the reduction for sentences with three auxiliaries and a transitive main verb is at least 96%. In order to see whether the aforementioned prediction made in Fokkens (2011a) holds, I also compared these results to a

Structure	**no constraints**	**constraints on rules**	**aux+verb**
iv-2aux	3901	733	253
iv-3aux	32190	1560	563
tv-2aux	21840	1967	466
tv-3aux	100,000<	4039	1051
dv-2aux	100,000<	5999	794

Table 5.20: Required edges for NLG with arg-comp without constraints on NON-HEAD-DTR

grammar scaled up to cover additional phenomena and using aux+verb analysis. The number of required edges drops by 93.5% for sentences with two auxiliaries and an intransitive main verb. A 31.3% reduction was found for the toy grammars in Fokkens (2011a) confirming an increase in the difference in required number of edges. Furthermore, where aux+verb can generate sentences with ditransitive verbs and two auxiliaries with 794 edges, the arg-comp analysis needs more than 100,000. For this structure, the aux+verb analysis thus leads to a reduction of required edges of at least 99.2%.

The final observation that can be made from this experiment is that the filler-gap analysis seems to be less efficient than the flat Matrix-based analysis for verb second word order. It should be noted, however, that no conclusions about this matter can be drawn for the moment. The combination of the Matrix-based analysis for second word order and argument composition has been part of gCLIMB from the beginning, whereas the filler-gap analysis has been added at a later stage. Because the filler-gap analysis has never shown major issues in parsing efficiency and it has not been included in the metagrammar for long, less effort has been made to increase its efficiency. Efficiency issues related to the filler-gap analysis as compared to the Matrix-based analysis for verb second word order should be addressed in future work. The next section will describe experiments that can be used to address this question as well as other possible experiments to be conducted with gCLIMB as part of future work.

5.3 Outlook

This section describes two lines of investigation that were started during the creation of CLIMB, but whose completion would require substantial additional research. They are described here because they illustrate the kind of research that may be conducted using CLIMB. The outcome of these projects could result in new (linguistic) insight in German grammar as well as insights into the interaction of grammars and automatic approaches to lexical acquisition.

5.3.1 Comparing grammars for German

One of the reasons for choosing German as the language to use in the experiment exploring whether the CLIMB methodology scales to broader coverage grammars was that two grammars for this language have been developed within DELPH-IN. These resources thus provide material to compare gCLIMB with. Even though a comparison between these three grammars cannot lead to conclusions about the impact of using CLIMB as a methodology, a comparison does provide basic insight into the complexity of gCLIMB. This holds even more for comparing gCLIMB to GG than for comparing it with Cheetah. GG has been developed by different grammar writers over several years. Especially because of the significantly longer development time, GG will probably cover more linguistic phenomena. Insight into these differences can be used as the basis of new research. By combining analyses of the three resources, gCLIMB can be used for further experiments in comparative efficiency. Furthermore, coverage of gCLIMB can increase through additional phenomena included in GG or the more efficient and pragmatic approach followed in Cheetah. This section presents the first steps of an analysis of the differences between these three resources.

A basic comparison

In order to get some idea of how the grammars compare, the Cheetah development set and the extended set used to develop gCLIMB were parsed by all

	Cheetah	**gCLIMB development Set**			
	coverage	**overgeneration**	**coverage**	**Treebanked**	**MRS**
Cheetah	85.7%	16.6%	80.9%	78.7%	57.5%
GG	81.0%	10.5%	93.0%	92.2%	92.2%
gCLIMB-ac	94.3%	0.0%	99.8%	98.0%	98.0%
gCLIMB-ar	94.3%	0.0%	99.8%	98.6%	98.6%
gCLIMB-fg	93.3%	0.8%	99.4%	97.8%	97.8%

Table 5.21: Results of the grammars on Cheetah and gCLIMB development set

grammars.[13] Results are presented in Table 5.21.

The first column indicates coverage on Cheetah's development set. The following two columns indicate what coverage and overgeneration of the grammars are on the extended set used to develop gCLIMB. The column labeled "Treebanked" indicates how often the correct parse, as intended by the grammar, was included in the output. The final column indicates for how many sentences the grammar produced a defensible MRS representation of the sentence. GG and gCLIMB aim at producing correct MRSs, which means that they must produce a defensible MRS representation in order to produce a correct parse. Cheetah generally aims at analyses that are closer to syntactic dependencies, but as the results show this corresponds to the correct MRS in most cases; 73% of the analyses Cheetah produced as desired corresponded to MRSs as produced by GG and gCLIMB.

Considering the fact that Cheetah and gCLIMB used the dataset during development, GG's coverage is comparatively high. The quality can furthermore be seen by the fact that almost all sentences that are parsed also result in a correct analysis. Furthermore, GG covers 33.7% of 1,000 sentences taken from TiGer compared to 21.3% and 47.3% by gCLIMB and Cheetah, respectively. If we consider the fact that Cheetah and gCLIMB use lexica that are tailored for these corpora (i.e. the development corpus and 1,000 sentences from TiGer), which was not possible to do for GG, these results provide a strong indication that GG is indeed the richest resource of the three.

[13]A similar evaluation was carried out in Fokkens et al. (2012a).

Due to the significantly longer development time, another outcome of a comparative study would have been surprising. The differences in coverage and overgeneration between Cheetah and gCLIMB are also expected. A comparison between the grammars as a whole therefore does not provide any additional information on CLIMB. The interest of comparing these resources lies in their different strengths.

Expanding gCLIMB using other grammars

The main purpose of developing gCLIMB was to test whether CLIMB can be adopted as a general methodology for grammar engineering. It was not possible to use analyses of existing grammars or even closely examine them, because that would have lead to an unrealistic scenario: in most cases, there will not be a grammar developed within the same framework yet and definitely not a grammar the size of GG. However, GG contains a vast amount of knowledge. It has for instance always used the filler-gap analysis for verb second word order and a significant amount of work has gone into improving efficiency of the grammar. If gCLIMB can generate analyses closer to the ones used in GG, a clearer answer to the question raised in the previous section concerning the efficiency of the filler-gap analysis in generation may be found.

Cheetah was developed to achieve decent coverage on TiGer in a short time. Efficiency on frequent structures generally had priority over gaining coverage on complex linguistic phenomena. Another interesting property of Cheetah is the analyses used for treating German word order. Cramer (2011) uses types to define topological fields. Word order is treated as if strings (in case of parsing) are passed through an automaton that ensures that each structure is placed in a field that can indeed contain it.

The strength of CLIMB is that it provides a platform for implementing interesting variations from gCLIMB, Cheetah and GG and evaluate them while controlling for other properties of the grammar. The primary future plan for gCLIMB is therefore to investigate analyses from Cheetah and GG for word order and auxiliaries in addition to those already present in gCLIMB. One of the main challenges lies in the significant differences in feature geometry between GG and gCLIMB. Because GG only has one arguments list, all rules combining subjects, complements or specifiers to heads as well as the entire lexical hierarchy are affected. The code developed for spring cleaning and path reduction and completion described in Section 3.3.2 can be extended to handle more complex differences, but this is not a trivial task.

If gCLIMB is to be used to investigate German syntax in HPSG, BerliGram (Müller, 2007) should be considered as well. This grammar contains analyses that are primarily designed as part of linguistic investigation. Grammar engineering is used to support linguistic research in a stronger way than this was done in the case of GG, which was primarily designed with applications in mind. Müller can stay closer to HPSG theory, because his grammar runs on TRALE (Meurers et al., 2002; Penn, 2004). This platform provides a more flexible formalism than that interpreted by the DELPH-IN parsing and generation tools. The differences between DELPH-IN grammars and theoretical HPSG outlined in Chapter 2, Section 2.2 do not apply to BerliGram. In order to include analyses from BerliGram to gCLIMB, CLIMB will need to be able to produce grammars that run on TRALE. There are no fundamental differences between the way grammars are defined in both platforms, so this adaptation should be relatively straight-forward. It does however lead to an additional dimension to be considered while maintaining gCLIMB: in addition to the interaction between analyses that needs to be taken into account, the grammar writer also needs to take into consideration which analyses work on both platforms and which can only work on TRALE.

5.3.2 The grammar and the lexicon

The original implementations of lexical types in gCLIMB were an extension of the lexicon library in the customisation system. Syntactic and semantic properties of lexical items can be defined in the choices file and these definitions can be combined with an arbitrary number of STEM and PRED values for individual instances. All early grammars created with CLIMB and the

Description	Lexical items	Lexical types
Core lexicon only	534	172
Core + TiGer, no optional arguments	73,724	376
Core + TiGer, optional arguments	71,365	583

Table 5.22: Lexicon size and number of lexical types for grammars using the TiGer lexicon

core component of current CLIMB grammars use these implementations to define lexical items.

In order to increase lexical coverage, implementations were added to create grammars that can work with the lexicon and morphology derived from the TiGer Treebank (Brants et al., 2002) for Cramer's (2011) Cheetah. This lexicon and morphology will henceforth be called the TiGer lexicon and TiGer morphology. The metagrammar includes a library that can create lexical types combining all possible subjects with all possible complements creating a large variety of intransitive, transitive and ditransitive verbs as well as verbs taking four arguments. It is possible to filter the frames so that only those that are found in the TiGer lexicon are included. An overview of the size of the lexicon and number of lexical types when the TiGer lexicon is used is given in Table 5.22. This addition to gCLIMB opens a new door to address a variety of research questions. This section describes preliminary observations on how the lexicon and grammar interact and indicates related research questions that may be examined in future work.

A morphology and lexicon read off TiGer

The TiGer morphology treats all morphological variation as irregular. Each surface form is related to the lemma it received in the TiGer annotation by a rule that assigns grammatical properties of the word. This includes properties such as person and gender for nouns, which remain constant throughout morphological variation and are generally associated with lexical items rather than morphological rules. Subcategorisation frames in the lexicon are directly read off observations in the corpus. This means that for each token, its

grammar	coverage	overgeneration
arg-comp	74.5%	24.5%
arg-comp, optional args	65.1%	21.4%
filler-gap	76.9%	27.8%
filler-gap, optional args	72.2%	26.9%
aux+verb	77.6%	28.2%
aux+verb, optional args	67.6%	24.2%

Table 5.23: Results of parsing the development corpus with the TiGer lexicon

arguments or the element it modifies are checked and a new entry is added to the lexicon if this token has not been used with this subcategorisation frame or this modifiee before. As a result, certain linguistic generalisations are not captured in the lexicon. Verbs with optional arguments for instance may have several entries in the lexicon. The expletive *es* is optional for some verbs and CPs may sometimes occur instead of NPs as arguments. In all these cases, the surface subcategorisation is added to the lexicon. Adverbs form another example receiving new entries depending on the item they are modifying. Therefore, many commonly occurring words are more ambiguous in the TiGer lexicon than they would have been if more generalisations were captured.

This relatively high lexical ambiguity leads to inefficiencies in the grammar. This is shown by the results of parsing the development corpus with a grammar using the TiGer lexicon and morphology presented in Table 5.23. Coverage drops to 65.1%-77.6% and overgeneration increases to 21.4%-28.2% depending on the chosen analysis for word order and whether optional arguments are used in the TiGer lexicon. The grammars have full lexical coverage for this corpus. All loss in coverage is due to the maximum number of edges being hit before finding a complete parse. I will address the difference in coverage caused by using optional arguments in a discussion about revising subcategorisation alternation below.

Parsing TiGer

In order to get an impression of the coverage of gCLIMB grammars on a corpus, the gCLIMB grammars with the TiGer lexicon and morphology were also run on 1,000 sentences from TiGer. The most efficient word order and auxiliary analysis managed to cover 21.0% of this set. Because of problems in efficiency, some initial experiments were carried out where analyses for rare phenomena that decrease efficiency of the grammar were excluded from the grammar. As explained in Chapter 3, CLIMB facilitates creating grammars that include or exclude a specific analyses from the grammar. These experiments could thus easily be run in a clean set-up, where the metagrammar ensures that only types and constraints related to specific analyses are not included in the alternative version of the grammar.

Results remained around 20%. For instance, when excluding partial VP fronting where some auxiliaries remain in the verbal cluster, coverage goes up to 21.3%. Five new sentences are covered and two sentences are no longer covered. However, the two sentences no longer covered do not exhibit partial VP fronting. This indicates that the results that were originally produced by the grammar were probably the result of overgeneration rather than the grammar producing a correct parse. The overgeneration is likely the result of a combination of the structure of the grammar originally designed to capture linguistic generalisations where possible, the lexicon read off a corpus and the possibility of parsing phrases that are not complete sentences. This last property was included in the grammar to allow it to parse headlines.

As mentioned in Section 5.1.3, Cheetah has a coverage of 47.3% on the same 1,000 sentences from the TiGer corpus. Two aspects may have played a role in this superior coverage of TiGer. First, Cramer (2011) reports using the first 500 sentences of TiGer as a development corpus to improve the grammar. Thus, Cheetah can be expected to contain some additional corpus-specific analyses. The development of gCLIMB only covered the first stage of grammar development of Cheetah and therefore only considered the examples of Cheetah's primarily linguistically motivated development set. Second, Cheetah was developed from the start with the idea of gaining lexical coverage from the TiGer treebank and the grammar may therefore be more suited to the lexicon that was derived from the treebank.

The grammar and lexical acquisition

The interaction between grammar and the lexicon form an interesting topic for future work. Both investigation in lexical acquisition and in alternative structures of the grammar can lead to new insights into the balance between precision and lexical coverage as well as efficiency and ambiguity. Lexical acquisition is the task of automatically identifying the lexical class of unknown words for a given computational grammar. Cholakov (2012) describes the following two-step process for lexical acquisition methods (Cholakov, 2012, p.34):

1. Identify word(s) in the lexicon that are most similar to the unknown word
2. Identify and analyse the lexical categories of the similar word, so that the proper category or categories of the unknown word can be derived

When this is applied to HPSG grammars, this means that the approach aims to find words in the lexicon with similar syntactic properties and assigns a lexical type to the unknown word based on these similar words. This may lead to the identification of generalisations that apply to particular lexical types. There are three reasons that may explain why Cramer (2011) did not follow this approach for deriving a lexicon from TiGer. First, reading of a lexicon from a treebank is not the same as lexical acquisition from an unannotated corpus. The treebank provides annotations so that the syntactic properties in the particular sentence are known, unlike a lexical acquisition approach where these properties have to be derived from the context. The need to collect evidence from several structures is thus not as strong when creating a lexicon from a treebank. Furthermore, the examples with a given word are limited to those occurring in the treebank, which may not provide enough evidence to compare new lexical items to existing ones. Finally, in order to compare new lexical items to existing ones, a reliable lexicon of a decent size should be present in the first place. In the case of Cheetah, only a core lexicon of high frequency closed class lexical items (prepositions, determiners and auxiliary verbs) was present. All open class lexical items were read off the treebank. There was thus no linguistically motivated material to compare new items to.

If we want to investigate how we might create a linguistic precision grammar in relatively short time using a treebank, it would be interesting to investigate how a lexicon read off a treebank can be used as a basis for a linguistic precision lexicon. Classes of similar lexical items can be created based on the information read off the treebank or external resources. For instance, investigating which words have similar alternative subcategorisation frames can provide insight into common alternations in subcategorisation. Semantic classes may also share syntactic properties which may help to identify whether a verb has optional arguments (e.g. *eat* that may appear without an object) or assigns different roles depending on the number of arguments (e.g. *I broke the window* versus *the window broke*). In other words, we can learn from lexical acquisition approaches to create classes of words based on what was read off the treebank. These classes can be studied to form lexical types that capture general syntactic properties of the language and the entries found in the treebank can be mapped to appropriate types. Such a lexicon could be used as a basis to apply further approaches for lexical acquisition using corpora that are not annotated.

Finally, several studies have addressed deep lexical acquisition for DELPH-IN grammars (among others Baldwin (2005); Blunsom and Baldwin (2006); Marimon et al. (2007); Zhang (2007); Cholakov (2012)). Most of these approaches use the grammar itself in combination with corpora and (in some cases) linguistic information created by other resources to predict the lexical type of unknown words. These approaches have been applied to several grammars for different languages, but (to my knowledge) no studies have been carried out comparing the interaction between choices made in grammar design and lexical acquisition. In particular, the structure of the lexical type hierarchy and approaches for dealing with subcategorisation alternation may lead to new insights. Alternative approaches to subcategorisation alternation form an interesting topic of investigation independently of its interaction with lexical acquisition. I will elaborate on this below.

Revising subcategorisation alternation

Flickinger (2000) shows that efficiency in a grammar can be improved by a reduction of roughly 30% in time and 20% in heap space by using optional arguments rather than multiple lexical entries. I have created a version of the TiGer lexicon that uses optional arguments to reduce the size of the lexicon. Subcategorisation frames that occur for the same stem were compared. If the arguments of one frame were a superset of the other frame, the stem received a lexical entry with all arguments of the superset where the ones missing from the smaller frames were made optional. This reduces the average number of lexical entries per stem for verbs from 2.03 to 1.76. gCLIMB can produce grammars with this lexicon as well as with the lexicon that does not use optimal arguments. Table 5.24 provides an overview of the coverage on all sentences and average number of tasks and edges, average CPU time, average memory usage per sentence. Contrary to the findings of Flickinger (2000), results vary with a tendency towards less efficiency when optional arguments are used. The decrease in coverage, which we also observed in the evaluation data presented in Table 5.23, is caused by an increased number of examples where the parser hits the maximum number of edges. This unexpected result is caused by interactions between rules ensuring argument optionality, local word order constraints and long distance dependencies. The outcome may be different as soon as basic analyses for word order, extraction and argument optionality are revised using knowledge from GG.

Haugereid (2011) proposes an alternative analysis for alternative subcategorisation frames, where unary rules to drop optional arguments from the argument's list are avoided. The approach supposes an additional feature in lexical items that captures all possible subcategorisation frames for a given

measure	arg-comp full lex	arg-comp opt-arg	filler-gap full lex	filler-gap opt-arg	aux-rule full lex	aux-rule opt-arg
Coverage	18.2%	17.6%	18.6%	17.4%	21.2%	19.2%
Tasks	157,534	194,609	180,354	200,552	208,057	189,267
Edges	4,368	5,179	6,658	6,388	8,112	6,049
CPU (s)	1.27	1.82	2.00	2.27	1.30	1.17
Memory (kb)	23,038	26,392	32,615	31,561	31,717	26,991

Table 5.24: Difference in efficiency with and without optional arguments

word. As the verb is combined with potential arguments, it is verified against this additional feature whether they can occur as an argument of this particular verb or not. Furthermore, it can be checked whether the resulting structure forms an admissible subcategorisation frame for the verb in question after it has combined with all of its arguments in the sentence. Because this approach avoids unary rules capturing argument optionality, it may also avoid the inefficiencies found in the first preliminary experiment for reducing the size of the lexicon. Haugereid's (2011) analysis requires including analyses with fundamental differences in feature geometry. Integrating Haugereid's proposal thus involves the same challenge as the integration of GG's treatment of argument structure addressed in Section 5.3.1. Because both research related to including alternative analyses from other grammars for German as well as this investigation on efficient treatment of lexical ambiguity involve this challenge, one of the first steps in future work will be to revise CLIMB so that it provides additional support for alternative feature geometries. The path reduction and completion algorithms outlined in Section 3.3.2 provide basic functions for manipulating the feature geometry and can be extended to support this functionality.

5.3.3 Summary of the outlook

This section presented an outlook of work that can be carried out with gCLIMB. First, I provided a basic comparison between GG, Cheetah and gCLIMB. Each of these resources has different strengths. The strength of

CLIMB is that it provides the means to combine the strengths of individual sources. One of the first steps in future work will be to include analyses from Cheetah and GG in gCLIMB. Alternative word order analyses provide new material to compare efficiency of different approaches and coverage of linguistic phenomena can be extended with knowledge included in GG. It was pointed out that, when investigating German HPSG, BerliGram is particularly interesting to look at. It remains closer to HPSG theory, which is possible because it runs on TRALE. Including BerliGram analyses thus requires adapting CLIMB so that it can output grammars for TRALE requiring minor revisions of the software.

The second part of the outlook addressed possibilities of extending the gCLIMB lexicon through lexical acquisition. Results from an initial effort importing the Cheetah lexicon were presented. These results are relatively low (21% coverage on 1,000 sentences with full lexical coverage). The main reason for this results appears to be that the grammar was not designed to use lexical types that were directly read off a treebank. Future work could address the interaction between analyses in the grammar and different approaches for lexical acquisition. In particular, the question of how to deal with ambiguous subcategorisation frames can have a big impact on efficiency. The following section will provide a summary of the entire chapter.

5.4 Summary

This chapter has presented the main evaluation of CLIMB. It addressed two aspects that play a role in determining the merit of the methodology. First, the question of whether CLIMB can be used in long-term grammar development projects was addressed in Section 5.1. This was investigated through the development of gCLIMB, a metagrammar that can create German grammars covering at least all phenomena included in Cheetah's development set and covered by Cheetah's core grammar. An overview of phenomena included in gCLIMB with basic indications of how their treatment differed from Cheetah was given. This overview provided an indication of the complexity of the analyses in gCLIMB. The grammars were also evaluated on a portion of the TiGer corpus, where Cheetah outperformed gCLIMB because of efficiency and the Babel corpus where gCLIMB outperformed Cheetah handling more linguistic phenomena correctly. In the discussion of this section, I addressed the fact that it is impossible to compare different methodologies for grammar development in a scientific manner and that no clear definition can be given for when a grammar should be considered large scale. Nevertheless, it was shown that gCLIMB covers a wide range of linguistic phenomena indicating that the approach scales and is suitable for large scale grammar development. Furthermore, gCLIMB was developed in less than half the time of Cheetah's core grammar while partially addressing crosslinguistic variation and adapting a more challenging standard for its semantic output. This strongly indicates that if the method were to have a negative impact on development time, this stays within an acceptable range.

Section 5.2 provides a different means of evaluating the CLIMB methodology, this time in terms of the additional experiments that it enables. I presented two experiments addressing efficiency of the alternative analyses included in gCLIMB. The first experiment compared parsing results showing that the aux+verb analysis remains more efficient as the grammar grows, but that this difference can be greatly reduced by constraining the non-head-daughter in subject and complement rules in the arg-comp grammars. If no such constraints are assigned, the difference in efficiency of the two analyses increases significantly as the grammars grow, just as predicted by Fokkens (2011a). Similar observations are made in the natural language generation experiment. This experiment furthermore examines how specific linguistic properties of a language interact with different analyses. Grammars with slightly different word order properties were created and their performance was compared. This experiment is particularly interesting, because it is easy to set up using CLIMB. These different grammars were created by changing a few lines in choices files. It would be much harder to create all these versions of a grammar by manually adapting and maintaining them. Furthermore, the metagrammar ensures that properties that are not related to the changes under investigation remain stable. As such, this experiment shows that CLIMB facilitates research that would be extremely difficult to carry out without the tools provided by CLIMB.

Section 5.3 describes two future directions of research that can be carried out using gCLIMB. The first direction would dive deeper into the comparison of different visions on German syntax. If analyses from other HPSG based grammars for German are included in gCLIMB, the resource can truly be used to gain insight into German syntax. In particular, the inclusion of BerliGram analyses would be interesting for this purpose, since this grammar has as primary goal to provide theoretical sound analyses for German syntax. The second direction would explore the interaction between the grammar and the lexicon. This research would explore how to include generalisations when obtaining lexical items from a treebank or how to deal with ambiguous subcategorisation frames. In both directions of research, alternative analyses play a significant role. They present directions of research where CLIMB provides a framework for more systematic research.

Chapter 6

Multilingual aspects of CLIMB

This chapter describes multilingual and crosslinguistic aspects of CLIMBand DELPH-IN grammars in general. First, multilingual grammar development with CLIMB will be introduced in Section 6.1. The crosslinguistic aspects in CLIMB for Germanic languages (gCLIMB) are highlighted in Section 6.2. This is followed by the introduction of SlaviCLIMB, a version of CLIMB for Slavic languages that is part of the SlaviCore project (Avgustinova and Zhang, 2010). Section 6.4 describes CLIMB for second language learners. Finally, this chapter investigates how DELPH-IN grammars for different languages use the language independent core provided by the Grammar Matrix. Because this investigation requires insight into which types each grammar actually uses, the spring clean algorithm is revisited and applied to several grammars in Section 6.5. Changes and use of the Grammar Matrix core in gCLIMB and other grammars are described in Section 6.6.

6.1 Sharing between related languages

The general setup used in CLIMB, where partial implementations are linked to definitions in *choices*, forms a practical platform for parallel grammar development. There are several reasons why parallel grammar engineering and sharing information between grammars may be interesting. From the engineering point of view, sharing implementations helps to speed up the development of individual grammars. While describing grammars, considering commonalities and differences between a set of languages can help to better describe them and pinpoint language specific properties. Furthermore, considering multiple languages may support research that can explain the influence of the languages a student already speaks on their ability to learn a specific new language. Comparison may show what the new language has in common with languages the student already knows and reveal properties that may pose a challenge.

It is not surprising that CLIMB facilitates parallel grammar development, since the Grammar Matrix customisation system providing the basis for CLIMB was designed for this purpose. There is, however, an important difference between parallel grammar development with CLIMB and extending the Grammar Matrix customisation system to cover more phenomena. This difference lies in the open world approach of the Grammar Matrix (provided phenomena should ideally cover any natural language) versus the closed world situation when grammars for a specific set of languages are developed in parallel. Because all required variations are known when development is intended for a closed set of languages, the focus lies on analytical depth rather than typological breadth. This difference is mainly exploited in the SlaviCLIMB project discussed in Section 6.3, which will elaborate on the consequences of this difference. First, Section 6.2 will describe variations for different Germanic languages.

6.2 CLIMB for Germanic languages

The first experiments run with gCLIMB covered basic phenomena in Dutch, German and Danish. gCLIMB was further developed for German. The resulting grammar has been described in Chapter 5. This section describes the variations gCLIMB includes to cover Dutch and Danish data. I will start with a description of the main linguistic variations that were included in the early development phase of CLIMB. The next part of this section will present two small experiments examining to what extent gCLIMB can be used for creating grammars for Dutch and Danish now that gCLIMB has grown significantly focusing on German only. This is followed by a third experiment which investigates whether the analyses in gCLIMB apply to Northern Frisian, another Germanic language which was not considered at any time during the development of gCLIMB. The experiment for Dutch also includes a comparison between alternative analyses for auxiliaries and word order. This was not done for Danish and Northern Frisian, because all analyses had the same coverage and overgeneration and only minor differences in efficiency could be seen in the small test suites that were used for Danish and Northern Frisian in the evaluations.

6.2.1 Word order variations for Dutch

gCLIMB covers several main word order variations found in Germanic verb second languages. These variations include fixed order or free order of arguments in the Mittelfeld, placement of the verbal cluster before or after (indirect) objects and whether or not partial VPs can be placed in sentence initial position. They are based on linguistic properties of Danish, Dutch and German. I will discuss these variations by comparing Dutch and Danish word order to German word order described in Chapter 4. Dutch word order resembles German order most and will be presented first.[1] Table 6.1 represents topological fields and basic word order for Dutch.

Dutch word order can be modelled by the same topological fields as German. The variations found in this example for Dutch correspond to those in Table 4.1, page 125 except for one difference. The German set of sentences also contained an example where the object preceded the subject in the Mittelfeld. Even though this order is admittedly awkward in German, it is possible for objects to be placed before subjects in the Mittelfeld. Example (20) presents a more natural sounding example that exhibits this order.

[1]The description of Dutch word order is an extended version of the description given in Fokkens (2011a).

Vorfeld	**LB**	Mittelfeld	**RB**	Nachfeld
De man *The man*	**heeft** *has*	aardbeien *strawberries*	**gegeten** *eaten*	na het feest *after the party*
De Man	**heeft**	aardbeien na het feest	**gegeten**	
Aardbeien	**heeft**	de man	**gegeten**	na het feest
Na het feest	**heeft**	de man aardbeien	**gegeten**	
Aardbeien **gegeten**	**heeft**	de man na het feest		
Gegeten	**heeft**	de man aardbeien na het feest		

Table 6.1: Basic structure of Dutch word order in main clauses (not exclusive)

(20)

Nach	der	Party	hat	sie
after	the.F.SG.DAT	party.SG.DAT	have.3SG.PRES	PRO.3PL.ACC

keiner	essen	wollen.
PRO.NEG.SG.NOM	eat.BSE	want.BSE

"No one wanted to eat them after the party." [deu]

Dutch is not as tolerant about such alternations in the Mittelfeld. There is a strong requirement that the subject immediately follows the verb if it is not placed in the Vorfeld. Exceptions are possible, but hardly ever occur and are highly marked.

Example (20) has two properties that influence information structure in a manner that allow the object and subject to switch general preferred order. First, the object is a personal pronoun which preferably stands close to the left bracket. Second, the subject is an impersonal pronoun and impersonal pronouns are elements that are more flexible in their position.

Dutch generally strongly requires the subject of a main clause to immediately follow the conjugated verb if it does not precede it. The Dutch equivalent of example (20) presented in (21) is nevertheless not completely unacceptable because of the influential factors mentioned above.

The sentence in (20) will most likely be interpreted as glossed in the right context, but without context the sentence is also interpreted as if *She didn't want to eat anybody after the party* (where *ze* ("she") is interpreted as the subject) despite its implausible semantics.

(21) *? Na het feestje heeft ze niemand
after the.SG party.SG have.3SG.PRES PRO.3PL PRO.NEG.SG
willen eten.
eat.BSE want.BSE
"No one wanted to eat them after the party." (intended) [nld]

A similar difference applies to the relative order of direct and indirect object. In German, this order is flexible with a preference that depends on the verb. Dutch requires the indirect object to precede the direct object. As for subject-object order, some flexibility may occur when pronouns are involved. The preference in order is strong (despite occasional exceptions) and lack of overt case marking on Dutch noun phrases would lead to significant amounts of ambiguity if all orders were generally accepted. gCLIMB therefore includes an analysis that enforces the canonical Dutch word order of arguments in the Mittelfeld.

This analysis mainly involves restrictions on rules that pick up the second argument on the COMPS list: for fixed argument order, this is only allowed if the first argument is a verbal projection and the subject has already been found. Two additional features keep track of changes in argument order and indicate whether a main verb and argument combination are allowed to appear as a partial VP in sentence initial position. The values of these features must be passed up when several other rules apply. Finally, the grammar includes an additional rule which makes sure flexible argument order is supported when split verbal clusters[2] are permitted and the argument composition analysis is used. In total, there are ten places in the metagrammar where grammatical properties are included or excluded depending on argument order being flexible or fixed. In two cases, properties are added to ensure flexible order and the other eight properties help to constrain the order of arguments. Five constraints are added regardless of the analysis that is used for auxiliaries and word order and sixteen constraints interact with the analysis.[3] Proper-

[2]See Section 4.2.3, p. 128 for a description of this phenomenon.

[3]Note that more than one constraint can be added at the same location in the metagrammar. There can thus be more constraints than places where argument order being fixed or not plays a role.

ties related to argument order are found in three libraries: one is related to the feature geometry, one to passivisation and all others are included in the library for verb second word order.

The second difference between German and Dutch word order concerns the order of verbs in the Right Bracket. German word order in the verbal cluster is generally head-final with the exception of the auxiliary flip.[4] The order of auxiliaries and their complements is less rigid in Dutch and typically head-initial. Most Dutch auxiliaries can occur in both orders, but this may be restricted according to their verb form. Haeseryn (1997) distinguishes four groups of auxiliary verbs that have different syntactic restrictions.

1. Verbs selecting for participles which may appear on either side of their complement (e.g. *hebben* ("have"), *zijn* ("be")).

2. Verbs selecting for participles which prefer to follow their complement and must do so if they are in participle form themselves (e.g. *blijven* ("remain"), *krijgen* ("get")).

3. Modals selecting for infinitives which prefer to precede their complement and must do so if they appear in infinitive form themselves (e.g. *willen* ("want"), *kunnen* ("can")).

4. Verbs selecting for "to infinitives" which must precede their complement (e.g. *lijken, schijnen* ("seem")).

While there is some variation among speakers, the generalisations above are robust. The permitted variations for a combination of verbs of the 3rd and 1st category in the right bracket are presented in Table 6.2. The numbers behind the verbs indicate order of subcategorisation (verb1 subcategorises for verb2, etc.). The variant %*De man zou haar kunnen gezien hebben* is typical of speakers from Belgium (Haeseryn, 1997). Speakers from the Netherlands tend to regard such structures as ungrammatical. gCLIMB can both generate

[4]Part of this description was taken from Fokkens (2011a).

a Flemish grammar[5] accepting all variations in Table 6.2 and a (Northern) Dutch grammar which forces participles to appear at the edge of the verbal cluster. gCLIMB provides the possibility of including the feature EDGE (on CAT). This feature registers whether a verbal form is placed at the edge of the cluster or not. The value of this feature can be set for individual lexical items or morphological and lexical rules. It thus provides a general solution for making sure verbal forms are situated at the edge of the cluster based on the kind of verb and morphological properties.

VF	LB	MF	RB
De man *the man*	zou1 *would*	aardbeien *strawberries*	kunnen2 hebben3 gegeten4 *can have eaten*
De man %De man	zou1 zou1	aardbeien aardbeien	gegeten4 kunnen2 hebben3 kunnen2 gegeten4 hebben3
The man may have been eating strawberries			

Table 6.2: Word order variation in the right bracket

The two variations described above (fixed argument order and the order of verbal forms depending on verb class and verbal morphology) cover the main differences in word order requirements between German and Dutch. Two more variations are included to cover Danish word order.

6.2.2 Word order variations for Danish

Table 6.3 provides topological field representation for Danish as proposed by Ørsnes (2009a). The example is based on (Müller and Ørsnes, to appear, p.27) and presents a subordinate clause, a declarative main clause and an imperative.

The main difference between Danish on the one hand and Dutch and German on the other hand lies in the position of the *Verbal Field*. The verbal cluster

[5]I will not discuss the question of whether Flemish and (Northern) Dutch are the same language. The variation was included because Flemish speakers accept more variation in word order. Whether this is a dialectal difference or a difference between languages is not relevant for this discussion.

Vorfeld	T/C	Sentence Field		
			Verbal Field	
	at *that*	drengen ikke *the boy not*	har set *has seen*	filmen *the movie*
Drengen *the boy*	har *has*	ikke *not*	set *seen*	filmen *the movie*
	se *watch*	ikke *not*		filmen! *the movie*

Table 6.3: Topological model for Danish word order

in this field precedes objects. The verbal cluster forms a VP with its complements. When subjects are not fronted, they stand between the finite verb and the verbal cluster (if present). This is also the correct position for the negative adverb *ikke* ("not"). gCLIMB provides the options of a verbal cluster preceding objects (Danish) and a verbal cluster following objects (Dutch and German). The analyses for alternative placements of the verbal cluster are similar. The difference is captured by using head-initial rules to combine verbal heads in the cluster with its objects for Danish and head-final rules for Dutch and German.

Danish furthermore differs from German and Dutch in the syntactic conditions on fronted verbal elements. When verbal elements front in Danish, the complete VP must be placed in the Vorfeld. In other words, partial VP fronting is not permitted in Danish. gCLIMB provides partial VP fronting as an option that can be turned on for Dutch and German and excluded for Danish. If the Filler-Gap analysis is chosen for Danish, the correct behaviour is achieved by forcing auxiliaries to take VP complements.

The variations found in Danish have led to two more parameters in gCLIMB; verbal clusters that must precede objects (Danish) and clusters that must follow it (Dutch and German) and partial VP fronting (Dutch and German) or no partial VP fronting (Danish). Together with the two parameters based on difference between German and Dutch this leads to four parameters to capture basic properties for these three languages. The three languages received equal attention during the development of the grammars prepared for the comparative experiments carried out in Fokkens (2011a). An additional verification of how the metagrammar dealt with Dutch was carried out at stage four of the development of gCLIMB for German as presented in Table 5.15, page 186. Afterwards, the project focused on German exclusively. The following subsections present the results of experiments with grammars created with the early version of gCLIMB as well as the results of an experiment that uses the current version of gCLIMB to create grammars for Dutch and Danish.

6.2.3 Dutch Evaluation

Evaluation data

The first part of the evaluation of gCLIMB's multilingual applicability consisted of an experiment to see whether the phenomena gCLIMB covers for German are also covered for Dutch. The evaluation particularly addresses whether early implementations for Dutch still worked in the larger grammar and whether additional phenomena added for German could easily be adapted for Dutch. Linguistic coverage for German was verified by a manually created development set of 1138 examples (508 positive and 630 negative examples), as described in Section 5.1.1. The development set of 93 positive examples covered by Cheetah formed the basis of this set.

In order to examine the possibilities of covering these phenomena for Dutch with gCLIMB, I translated the German development set into Dutch. The set had to be adapted slightly to better reflect the behaviour of the phenomena in Dutch. I reduced the large variation to capture German adjective endings to adequately express Dutch variation (adjectives modifying neutral single nouns with an indefinite determiner do not have an ending, all others take the suffix -*e*) in a minimal setting of examples. Word order variations that do not occur in Dutch were adapted so that the set can be used to check for overgeneration and examples verifying German's case marking were removed. Leaving such examples in the test suite would artificially boost results for Dutch.

The Dutch set was extended to include the more complex word order variations of Dutch in the verbal group, dative shift and morphological marking of finite verbs in second person singular, which depends on word order. Dative shift in Dutch (22-23) and the word order variation for the second person singular (24-26) are illustrated below.

(22)

Marie	geeft	hem	het	boek.
Marie	give.3.SG.PRES	PRO.3.SG.ACC	DET.N.SG	book.

“Marie gives him the book.” [nld]

(23)

Marie	geeft	het	boek	aan	hem.
Marie	give.3.SG.PRES	DET.N.SG	book	to	PRO.3.SG.ACC.

“Marie gives the book to him.” [nld]

(24)

Je	hebt	hem	het
PRO.3.SG	have.2.SG.PRES.SUBJ-PREC	PRO.3.SG.ACC	DET.N.SG

boek	gisteren	gegeven.
book	yesterday	give.PTC

“You gave him the book yesterday.” [nld]

(25)

Heb	je	gisteren	hem
have.2.SG.PRES.SUBJ-FOL	PRO.2.SG	yesterday	PRO.3.SG.ACC

het	boek	gegeven?
det.N.SG	book	give.PTC

“Did you give him the book yesterday?” [nld]

(26)

Gisteren	heb	je	hem
yesterday	have.2.SG.PRES.SUBJ-FOL	PRO.2.SG	PRO.3.SG.ACC

het	boek	gegeven.
DET.N.SG	book	give.PTC

“Yesterday, you gave him the book.” [nld]

The variation of morphological marking for the verb *hebben* (‘have’) observed in Examples (24) and (25) applies to all verbs in present tense including irregular verbs, such as *zijn* (‘to be’). If a second person singular precedes a finite verb in present tense, it ends in the suffix *-t*, which is dropped if the subject follows the verb, regardless of whether this is the result of object or modifier fronting or the formation of a yes-no question.

Finally, adpositional phrases that introduce relative clauses function differently in Dutch from German. Similarly to English, Dutch uses *wh*-structures to create such relative clauses. Unlike English, these words cannot be dropped, nor can *wh*-phrases be used in relative clauses that are not introduced by an adpositional phrase. The test suite was adapted to verify correct performance of the Dutch phenomenon which is illustrated below:[6]

(27) de tafel waarop het boek ligt
the table on which the book lies
"the table the book lies on" [nld]

(28) de man waarmee Marie werkt
the man with who Marie work.3rd.sg
"the man Marie works with" [nld]

(29) de man met wie Marie werkt
the man with who Marie work.3rd.sg
"the man Marie works with" [nld]

(30) de man die met Marie werkt
the man that with Marie work.3rd.sg
"the man that works with Marie" [nld]

(31) * de man die Marie ziet
the man that Marie sees
"the man who sees Marie" [nld]

(32) * de man wie met Marie werkt
the man who with Marie works
"the man who works with Marie" (intended) [nld]

(33) * de man met die Marie werkt
the man with that Marie works
"the man Marie works with" (intended) [nld]

(34) * de man Marie ziet
the man Marie sees
"the man who sees Marie" (intended) [nld]

[6]Note that structures such as (34) are not grammatical in German either. Examples similar to (31) and (34) were also present in the original German test. The examples are included here to illustrate the difference between Dutch and English.

After all adaptations were made, the test suite consisted of 469 positive and 591 negative (a total of 1,050) examples, compared to 508 positive and 630 negative examples in the original German set. As the evaluation was carried out, corrections to the test suite were made, resulting in a set of 475 positive and 577 negative examples, a total of 1,052.[7]

Creating a grammar for Dutch

The Dutch grammars described in Fokkens (2011a) cover basic Dutch word order properties and intransitive, transitive and ditransitive verbs. A choices file used to create one of these grammars was used as the basis for creating a Dutch grammar. I added new phenomena using recent choices for German and adapting properties for Dutch where necessary. The created grammars were evaluated against the test suite described above and the choices file and metagrammar were adapted accordingly to increase the competence of the grammar.

Table 6.4 presents the competence of the grammar at different stages of this process. The first run through only attempted to get the grammar running with the *auxiliary construction* analysis.[8]

In addition to presenting the results of the Dutch grammar created with CLIMB, Table 6.4 reflects the general process of creating a new grammar with a metagrammar. In particular, the relation of adapting definitions in *choices* and adapting actual analyses is telling. After some minor corrections in the metagrammar (which took around 20 minutes in total) to make sure the metagrammar produced well-typed grammars, coverage went up from 35.8% to 96.6% without any additions to the metagrammar.[9] The bulk of

[7]The process of testing a precision grammar with a test suite (almost) always leads to new insights on both the test suite and the grammar (Bender et al., 2011). It is thus common that corrections to the test data are made.

[8]Version 23045 contained several changes to the metagrammar, including some features that allowed the *argument composition* analysis to run, the same version number is therefore repeated for each change that involved revisions in the metagrammar.

[9]Corrections in the test suite fixed typos in nine positive examples and moved six examples wrongly classified as ungrammatical to the positive set, contributing 2.0% to the increase in coverage.

attempt & additions	**coverage**	**overgen.**	**#items**	**svn**
0: (grammar did not load)	–	–	–	22129
1: removed German specifics hard coded in metagrammar	35.8%	0.3%	1050	22705
2: fixed vocabulary in choices	72.1%	2.2%	1050	
3: corrected typos in data	80.2%	3.3%	1050	
4: corrected data and choices	95.7%	5.5%	1050	
5,6: corrected data	96.6%	3.5%	1052	
7: added dative shift	97.7%	3.5%	1052	22891
8: added 2sg morphology changes	98.5%	3.1%	1052	22892
9: add adpositions introducing relative clause	99.8%	3.1%	1052	22907
10: corrected example in data	100%	3.1%	1052	
11: fixing word order settings	100%	2.9%	1052	
12: passing up passive feature	100%	2.4%	1052	23045
13: passing up feature PRD	100%	0.9%	1052	23045
14: fixing wh-rel	100%	0.5%	1052	23045
15: attempting to fix choices for passives	94.5%	0.5%	1052	
16: removing fix 15, fixing filler-gap	100%	0.3%	1052	23045

Table 6.4: Process of adapting gCLIMB for Dutch

this increase was thus due to manipulations to the definitions in *choices*. This observation is in line with Bender et al. (2010), where results from the first attempt to create and evaluate a grammar differed from the final results for all seven languages. Abkhaz revealed the smallest increase covering 27.8% more examples and the West-Greenlandic grammar covered 87.8% more of the data, both simply by adapting *choices*.

Time wise, producing these results took approximately half a day of work on the metagrammar and three and a half days of producing and correcting the definitions in *choices* for Dutch. It should be noted, however, that the 0.3% overgeneration points to a problem in the word order analyses. This small percentage in overgeneration actually refers to a fundamental problem in the current analyses for Dutch word order in the verbal group. In order to fix this problem, completely new analyses for Dutch word order need to be designed for gCLIMB, which is likely to take several days.

Overall, the results from this experiment show that the metagrammar is a suitable tool for parallel development of grammars for related languages with similar syntactic structures. The last version of the gCLIMB that was tested for Dutch variation captures only 18.9% of the positive examples in the development set. Even though Dutch variations were not taken into account during most time in the development process of gCLIMB, a grammar for Dutch capturing a large range of phenomena could be created in four days. But it also showed that not all similar phenomena in related languages can be accounted for by a small set of adaptations. In order to create a decent grammar of Dutch that can move towards a resource grammar, the word order analyses will need to be revised. It should be noted that Dutch word order was implemented and behaved correctly for the small grammars examined in Fokkens (2011a): changes to expand the grammar and capture more phenomena led to the current analyses for word order that do not handle Dutch word order correctly.

It would have been possible to maintain the old analyses in gCLIMB, but this was not done because these analyses could not handle German word order correctly as gCLIMB covered more phenomena. The possibility that they may still be the right analysis for Dutch was not taken into account at the time. This outcome shows that, in cases where the main purpose of CLIMB is to support multilinguality, the range of variations of the individual languages will need to be taken into account and be examined carefully for each phenomenon. It is important to run regression tests for all languages covered by the resource and examine whether a necessary revision for one language is indeed necessary for other languages as well. Otherwise, old analyses should be maintained in CLIMB.

Alternative analyses for Dutch

After achieving 100% coverage and staying with a small percentage of overgeneration that presents a challenge to fix, I created grammars with the alternative analyses for auxiliaries and word order described in Chapter 4. The word order variation includes the *filler-gap* analysis for verb second order, which is the standard analysis for German and Dutch in theoretical HPSG and a flat analysis using the MC feature to make sure exactly one constituent ends op before the conjugated verb. The second analysis is based on Wambaya and included in the Grammar Matrix customisation system. The second point of variation concerns auxiliaries, which either use the argument composition analysis as proposed by Hinrichs and Nakazawa (1994) or an alternative proposed by Bender (2010) which has been shown to be more efficient for Wambaya by Bender (2010). I have shown in Chapter 5 that this analysis is also more efficient for parsing German and for generating sentences in Dutch. The analysis was more efficient for German, Dutch and Flemish in Fokkens (2011a). This alternative analysis is currently only compatible with the flat MC analysis. This resulted into three alternative grammars: *filler-gap* (standard HPSG analyses for both phenomena), *argument composition* (flat verb second analysis, but standard HPSG argument composition analysis for auxiliaries) and *auxiliary construction* (flat verb second analysis and alternative auxiliary analysis).

The additions for the interaction between the position of subjects in second person singular and morphology on the verb use features covered these phenomena for the *auxiliary construction* analysis (henceforth aux+verb). Minor revisions had to be made to the metagrammar to make sure the grammars containing the *argument composition* analysis (henceforth arg-comp) and *filler-gap* analysis were well-formed and could handle the phenomenon. The final results, including average number of edges and CPU time per sentence, for the Dutch dataset are presented in Table 6.5.

Unlike in all other experiments carried out in this thesis, the arg-comp analysis has the best results in performance when comparing the number of

analysis	**coverage**	**overgen.**	**edges**	**cpu (s)**
aux+verb	100%	0.3%	108	0.21
arg-comp	100%	0.2%	100	0.24
filler-gap	97.1%	1.0%	170	0.30

Table 6.5: Parsing results of gCLIMB for Dutch

required edges. Previous experiments on parsing German or generating sentences in Dutch (see Section 5.2.2) all showed that the aux+verb analysis was more efficient than the arg-comp analysis on all accounts (number of required edges, CPU time, memory and carried out tasks). This is an important result, because it shows that the evidence that has been collected so far does not provide conclusive evidence for the hypothesis that the aux+verb analysis is the more efficient option for every Germanic grammar. Because competence and efficiency depend on several properties of the grammar which are not independent of each other, it will always remain difficult (if it is possible at all) to provide conclusive evidence of an analysis' superior efficiency.

However, I would still be inclined to say that the results in this thesis provide strong evidence that the aux+verb analysis is more efficient than the arg-comp analysis. First, the difference in efficiency observed in Table 6.5 is small in comparison to results from previous experiments. Second, the grammars used in this experiment do not model Dutch word order completely correctly. It is possible that the bug that is leading to overgeneration in these grammars is also responsible for the higher number of edges that are needed when using the aux+verb or filler-gap analysis. The fact that overgeneration is higher for these two grammars provides further support for this hypothesis. I have pointed out in Chapter 5 that future research should further investigate under what conditions the aux+verb analysis is more efficient than the arg-comp analysis. One of the first questions to answer in this regard is whether the arg-comp analysis is still more efficient for parsing once gCLIMB is fixed and captures Dutch word order correctly again.

6.2.4 Danish Evaluation

The Danish component in gCLIMB is evaluated by using the test suite accompanying DanGram (Müller and Ørsnes, to appear)[10] of 58 positive examples, 1 example marked by a *?* and 29 negative examples (not considering examples that were commented out in the test suite). The evaluation was carried out after the evaluation on Dutch was completed. The metagrammar thus already contained all corrections that were made for Dutch. Development time was not registered in detail, but the first two stages took approximately one working day and stages three to seven were carried out in five evenings.

This evaluation is more challenging than the evaluation for Dutch for two reasons. Danish word order differs more from German than Dutch word order making it more likely that gCLIMB does not contain the required analyses. Furthermore, the Dutch test suite was based on the original development set for gCLIMB and therefore covered more or less the same phenomena. The DanGram test suite has been developed completely independently of gCLIMB.

Evaluation setup

The evaluation makes use of information included in the test suite only. Examples in this section come from the test suite.[11] Based on the descriptions accompanying the data, one (positive) example was corrected where the description said a pronoun should bear accusative case, but the nominative pronoun was found. The grammar cannot parse this sentence regardless of the case of the pronoun, so this correction did not influence the results of the evaluation. The set includes two examples which were marked as ungrammatical, but turned out to be grammatical with a pragmatically implausible

[10]`http://hpsg.fu-berlin.de/Fragments/Danish/`, accessed 3 February 2013, part of the release of 12 January 2012.

[11]The test suite did not include complete IGT, but rather only word-by-word glosses (in some cases). In order to present full IGT for the examples in this section, I made translations and glosses where they were missing in the original examples.

reading.[12]

Grammar development

Table 6.6 presents the outcome of this evaluation. The numbers in brackets represents the score if we regard the example that was marked with a question mark (presented below) as ungrammatical.

(35)	? Giver	Bjarne	Peter	den	ikke?
	Give.3SG	Bjarne	Peter	pro.N.3.SG	NEG

“Does Bjarne not give it to Peter?” [dan]

I will briefly elaborate on the individual stages of the evaluation. As mentioned above, the verbal cluster precedes objects in Danish. The first version of gCLIMB thus included a separate set of rules that places objects after the verbal cluster. These rules and all possible interactions were ignored while gCLIMB grew to cover more phenomena in German. This is reflected in the early stages of creating a grammar for Danish with gCLIMB. The first attempt failed on a bug in gCLIMB and no grammar was created. The grammar that was created during the second attempt also allowed objects to precede the verbal cluster (accidentally including rules not applicable to Danish) leading to higher overgeneration than coverage (fixed in attempt 2). The Danish experiment was interrupted for several weeks between the second and third attempt. Several changes were made to gCLIMB in this period while evaluating gCLIMB on the TiGer treebank as described in Section 5.3.2. This included the possibility of parsing fragments in addition to full sentences. Attempts 3 and 4 were run after fixing typos in *choices* and the test suite and including the possibility of parsing phrases.

The fifth attempt includes the only addition of a new phenomenon to gCLIMB that was made as part of the evaluation of Danish. Danish nouns can take suffixes that mark definiteness. These suffixes have the same effect on semantics as definite articles and are mutually exclusive to specifiers preceding

[12]This grammaticality judgement was confirmed by Bjarne Ørsnes (p.c.).

attempt & additions	coverage	overgeneration	svn
0: (no grammar created)	-	-	23313
1: fix: bugs in gCLIMB	10.2% (10.3%)	17.2% (16.7%)	23975
2: fixing position verbal cluster	5.1% (5.2%)	0% (0%)	23976
3: choices and typos	22.0% (22.4%)	27.6% (26.7%)	26689
4: allowing for phrase as root	33.9% (34.5%)	34.5% (33.3%)	
5: definite suffix and copula	50.8% (51.7%)	24.1% (23.3%)	
6: added rules for modification	54.2% (55.2%)	20.7% (20.0%)	
7: passivisation	57.6% (58.6%)	20.7% (20.0%)	26911
8: corrected judgement	59.0% (60.0%)	14.8% (14.3%)	

Table 6.6: Performance of gCLIMB grammars on Danish test suite

the noun. The following examples from the DanGram test suite illustrate this:

(36) bogen
book.DEF
"the book" [dan]

(37) den bog
DEF book
"the book" [dan]

(38) * den bogen
DEF book.DEF
"the book" (intended) [dan]

Because definite suffixes occur in several examples of the set, this addition resulted in a significant increase in coverage. Some corrections to the definitions of copula verbs in the choices file reduced overgeneration. The metagrammar did not include rules to cover modification for languages with verbal clusters that precede objects prior to svn revision 26911. This was corrected in stage 6. Finally, a minor correction to the analysis for passivisation was made in the metagrammar leading to the final results in this evaluation.

Stage 8 does not represent a new version of the grammar, but a correction in the test suite. As mentioned above, it was confirmed by Ørsnes that two examples that lead to overgeneration as measured in this evaluation are actually syntactically well-formed. They illustrate fixed word order of direct and indirect object in ditransitives. Example (39) presents one of these examples followed by its well-formed counterpart in (40).

(39)	* Manden	giver	bogen	Max.
	man.DEF	give.3.SG	book.DEF	Max

“The man gives Max the book.” (intended) [dan]

(40)	Manden	giver	Max	bogen.
	man.DEF	give.3.SG	Max	book.DEF

“The man gives Max the book.” [dan]

The sentence in (39) is not actually ungrammatical, but rather only grammatical with a pragmatically unlikely reading. It conveys the implausible reading *The man gives Max to the book.* The results in stage 8 provide the coverage and overgeneration of the grammar if the two sentences that illustrate word order for ditransitives are classified as positive examples.

Error analysis

The results for Danish are less good than those for Dutch: the final grammar has a coverage of around 59% and overgenerates on approximately 15% of the examples in the test suite. Table 6.7 provides an overview of the phenomena causing overgeneration (overgen.) and lack of coverage (undergen.). These phenomena can only be captured by including completely new analyses in gCLIMB.

Most errors are caused by lack of an adequate analysis for the position of adverbial modifiers, notably the negation marker. Another notable phenomenon that is not covered is the position of negated object NPs. Rather than their canonical position following the verbal cluster, they must precede the cluster taking positions that are usually filled by adverbs:

(41)	Bjarne	arbejder,	hvis	han	ingen	bog	læser
	Bjarne	work.3.SG,	if	pro.M.3.SG.NOM	no	book	read.3.SG

Description	overgen. (# ex)	undergen. (# ex)
Word order and negation	2	9
Negated NPs in adverbial position	1	2
Modification word order	1	1
Interrogatives and word order		4
Expletive in subordinates		2
Subject extraction		4
Coordination and extraction		1

Table 6.7: Overview of over- and undergeneration DanGram test suite

"Bjarne works if he is not reading a book." [dan]

(42)

Bjarne	arbejder,	hvis	ingen	bog	han	læser
Bjarne	work.3.SG,	if	no	book	pro.M.3.SG.NOM	read.3.SG

"Bjarne works if he is not reading a book." [dan]

(43) *

Bjarne	arbejder,	hvis	han	læser	ingen	bog
Bjarne	work.3.SG,	if	pro.M.3.SG.NOM	read.3.SG	no	book

"Bjarne works if he is not reading a book." (intended) [dan]

Clausal complements may include an additional expletive pronoun, a phenomenon that is also found in Dutch but currently not covered by gCLIMB. This phenomenon was not present in the German or Dutch development sets and therefore not addressed in the development of gCLIMB. It is illustrated in (44).

(44)

Bjarne	spørger,	hvem	der	læser	bogen.
Bjarne	ask.3.SG	who	EXPL	read.3.SG	book.DEF

"Bjarne asks who is reading the book." [dan]

Conclusions about the evaluation for Danish

This section investigated the competence of Danish grammars created with gCLIMB on a test suite consisting of 88 examples created by Müller and Ørsnes (to appear). The final grammars covered 59% of the 60 positive examples and overgenerated on 14.8% of the 28 negative examples. These numbers provide a rather limited insight into the competence of the grammars and can not be compared directly to the results for Dutch. The test suite was created as part of a grammatical description and grammar development effort that (partially) focused on different phenomena than gCLIMB. It is therefore not tailored to evaluating gCLIMB's phenomena in the way the development set for Dutch did. As a result, the Danish test suite contains phenomena that were never considered in the development of gCLIMB. Second, the set consists of a rather small number of examples. It is therefore not clear whether the Danish grammar indeed cover all variations of the included phenomena correctly. Nevertheless, the outcome of this experiment and in particular the error analysis of phenomena that gCLIMB does not cover provides insight into the current possibilities of creating grammars for Danish with gCLIMB.

The evaluation of Dutch coverage and performance presented in the previous section led to two observations. First, gCLIMB can produce a Dutch grammar that can handle a wide range of phenomena correctly. Second, if a resource is intended to create grammars for related languages, it is important to take the variations of all language into account while adapting the grammar. The Danish evaluation presented in this section mostly confirms the second observation. Even though the grammars handle most examples correctly, fundamental revisions and several additions of new phenomena are required in order to cover phenomena that are not handled correctly yet. This includes phenomena such as the word order of adverbial modifiers and negation. These revisions can only be carried out in collaboration with native speakers, given that there will always remain open questions concerning the behaviour of the language.

6.2.5 Northern Frisian Evaluation

Nigel Kilmer and Woodley Packard created a grammar for Northern Frisian in the context of the *Knowledge Engineering for NLP*[13] course. In this course (which has been mentioned at various occasions throughout this thesis), stu-

[13]`http://courses.washington.edu/ling567/`, accessed 29 November 2013.

dents create grammars for languages using the Grammar Matrix customisation system as a starting point. The grammar Kilmer and Packard created is made available together with final choices file and test suite as part of Language CoLLAGE (Bender, 2014).[14] The final test suite consists of 109 positive items and 55 negative items. The test suite is based on the description of Northern Frisian from Lasswell (1998) and 30 positive examples were taken from this work. KiIlmer and Packard, who are not (native) speakers of Northern Frisian, created additional examples they needed in order to test their grammar. These examples (55 negative and 79 positive in total) are based on Lasswell's (1998) descriptions.

Evaluation setup

The final evaluation carried out to examine the multilingual applicability of gCLIMB examined the possibility of creating a grammar for Northern Frisian based on Kilmer and Packard's test suite and choices file. Northern Frisian has not been taken into consideration at any point in time during the development, but it is a Western-Germanic language known to exhibit word order phenomena that are similar to Dutch and German (Hock and Joseph (1996)). It is spoken in the North of Germany around the Danish border and reveals similarities to these two langauges. I followed the *Knowledge Engineering for NLP* course in the initial stages of developing gCLIMB for German. This means that most phenomena included in the test suite are included in gCLIMB. The evaluation mostly tests whether the specific behaviour of the phenomenon in Northern Frisian can be captured using gCLIMB.

Grammar development

The evaluation of gCLIMB's competence for Northern Frisian was carried out after the experiments for Dutch and Danish. The metagrammar thus included all corrections and adaptations that were made during the exper-

[14]`http://www.delph-in.net/matrix/language-collage/`, accessed 29 November 2013.

iments described above. In this experiment, I investigated the possibilities of gCLIMB in a limited time frame. I therefore only fixed minor bugs in the metagrammar and did not revise analyses. All other improvements to the grammar consisted of adapting *choices*. Minor bugs are situations where, for instance, two phenomena need a feature, but only one of them introduces it. The grammar will work correctly if both phenomena are included, but not if the phenomenon that introduces the feature is missing while the other is included. The metagrammar was adapted in these cases, so that both phenomena introduce the feature. Another typical (related) case would be that the phenomenon needing (and introducing) the feature is not present in the grammar, but analyses for other phenomena still share the value of this feature. In this case, the metagrammar should check if the phenomenon requiring the feature is present in the grammar and only add the constraints sharing the feature's value in that case. These corrections only form minor revisions in the metagrammar. Similar revisions were made for Dutch and Danish, but I also actively added phenomena that were not included in gCLIMB for both of these languages (e.g. position dependent morphology for the second person singular for Dutch, morphologically marked definiteness for Danish).

The evaluation of gCLIMB's performance on Northern Frisian started by creating a grammar from Kilmer and Packard's original choices file with gCLIMB. Table 6.8 provides an overview of the results of grammars created with gCLIMB, the results of a grammar created by the Grammar Matrix and Kilmer and Packard's final grammar. I will elaborate on the individual development stages below.

The starting point of this evaluation was, as mentioned above, Kilmer and Packard's final choices file for Northern Frisian and gCLIMB without any adaptations. However, some minor revisions in the choices file were needed to make sure a grammar could be created and loaded. The latest version of the Grammar Matrix customisation system allows users to identify supertypes of lexical items. At the time, this functionality was only present for lexical rules and not for lexical items in gCLIMB. Several types for nouns in Kilmer and Packard's choices inherit information about their determiner from

Development stage	Coverage	Overgeneration	svn
Grammar Matrix customised	45.0%	9.1%	26861
Kilmer and Packard's grammar	70.6%	1.8%	-
0: gCLIMB and original choices	-	-	26911
1: gCLIMB and adapted choices	43.1%	20.0%	26911
2: adding supertypes, original choices	45.0%	5.5%	26913
3: choices & bug fixes	94.5%	7.2%	26977
4: two minor revisions	94.5%	0%	27000

Table 6.8: Overview of Northern Frisian evaluation

their supertype. Because gCLIMB ignored information about the supertype, it raised an error that information on determiners was missing for several nouns. I copied the choices concerning determiners from the supertype to its subtypes in the choices file, so that gCLIMB could create a grammar. The coordination library in gCLIMB can handle agreement between constituents. The implementation for coordination agreement in gCLIMB at the time had a bug creating agreement rules with empty values if no form of agreement was defined in the choices file resulting in an ill-formed grammar. I therefore added a choice requesting verbal forms to agree.

The revisions made in the second development stage addressed both the problems outlined above. The bug in the coordination library was fixed. I included the implementations from the Grammar Matrix allowing users to define supertypes of lexical items and use inheritance in gCLIMB. gCLIMB could now create a grammar with Kilmer and Packard's original choices file. The changes in gCLIMB led to only a small increase in coverage of the grammar, but strongly reduced overgeneration. This result is not surprising considering that supertypes were being ignored in the grammar created in the first stage.[15] Supertypes can introduce additional constraints, in this case restrictions on person, number, gender and case. The absence of these constraints leads to overgeneration, but does not stand in the way of coverage of the grammar.

[15] gCLIMB and the Grammar Matrix customisation system both ignore choices they do not understand.

The grammar has the same coverage as a grammar generated by the Grammar Matrix customisation system from the same choices. Overgeneration is reduced by 3.6% when gCLIMB is used instead of the Grammar Matrix customisation system (see Table 6.8, *Grammar Matrix customised*). This is caused by incorrect interaction between implementations for information structure and verb second analysis in the customisation system. Constraints required to ensure verb second word order are not taken up in phrasal rules introduced by the information structure library. There is no information structure library in gCLIMB and, as such, related definitions in the choices file are ignored by gCLIMB.[16]

The third stage reflects the final results on this set obtained by grammars generated using gCLIMB without revised analyses. In this stage, lexical items and grammatical properties were added to *choices* in order to cover all lexical items in the test suite (including those used in ungrammatical examples).[17] One example exhibited several complex phenomena that are not covered by gCLIMB and some lexical items only included in this example were not included. In total, 508 choices were added resulting in a choices file containing 968 choices. Minor bugs of the nature explained above were repaired in the metagrammar during this stage as well.

The evaluation of the resulting grammar led to the identification of two bugs in gCLIMB analyses which were fixed in the fourth stage of the evaluation. First, the rule changing word order requirements on finite verbs in order to form yes-no questions did not require verb forms to be fully inflected, nor did it pass on information indicating whether these forms were inflected or not. The grammar furthermore accepted sentences where the negation adverb was placed in the Vorfeld. An option that prevents scopal adverbs from occurring in this position was therefore added to the metagrammar. The remaining errors made by gCLIMB grammars after these changes are

[16]Note that this also means that the MRSs produced by gCLIMB grammars is less rich than those produced by Grammar Matrix customised grammars and the ultimate grammar created by Kilmer and Packard. MRSs produced by gCLIMB do not include ICONS. I checked MRSs produced by the gCLIMB grammars manually for plausibility, but did not compare them to the MRSs produced by Kilmer and Packard's grammar.

[17]None of the ungrammatical examples tested had illformed morphotactics.

Description	undergeneration (# ex)
Word order & optional subject	2
V2 order in subordinate	1
Asyndetic coordination	1
VPP and VP ellipsis	1
Imperatives	1

Table 6.9: Overview of remaining errors in Northern Frisian grammar

described below.

Error analysis

Table 6.9 provides an overview of the five phenomena that could not be handled properly by the gCLIMB grammars after all relevant choices were added an bugs were fixed. I will elaborate on these phenomena and why they are not covered below.

The first phenomenon that is not covered is problematic because of the interaction between word order in yes-no questions and pro-drop for subjects. The Grammar Matrix core introduces the basic rule for removing optional subjects from the SUBJ valency list. This rule requires the COMPS list to be empty. This constraint was meant to prevent spurious analyses (for instance that the rule applies to the verb, to the verb after picking up its first complement, after it picking up yet another complement, etc.). However, this constraint leads to problems in situations where word order constraints require the subject to be picked up before the complements. If the subject can be dropped in a language with strict VSO or OSV order, structures with verbs subcategorising for complements and a non-overtly expressed subject cannot be parsed. The subject cannot be removed from the SUBJ list, because the COMPS list is not saturated and the COMPS list cannot be saturated, because the verb has not combined with its subject yet. This situation occurs in Northern Frisian, where subjects must immediately follow the clause initial verb in yes-no questions and second person singular subjects can be dropped. The constraint that subjects may only be dropped if the COMPS list is saturated should be considered a language specific property (since it does not apply to all languages) and must be removed from the Grammar Matrix core.[18]

A second phenomenon that is not handled properly are subordinate clauses in Northern Frisian, which may have verb second word order even if a complementizer is present. This is illustrated in the following example:

(45)	Ik	gesi	dat	ik	iit	jam.
	pro.1.SG	guess.1.SG	COMP	pro.1.SG	eat.1.SG	pro.3.PL

“I guess that I can eat them.” [frr]

Dutch does not allow for verb second word order in subordinate clauses at all. They occur in informal German, but are only acceptable if the complementizer is dropped. The combination of complementizer and verb second order required to account for Northern Frisian was therefore not considered during the development of gCLIMB. The sentence in (45) is the only sentence that is covered in Kilmer and Packard’s final grammar, but not by the grammars created with gCLIMB.

The third phenomenon that leads to problems is asyndetic coordination, which is covered by the Grammar Matrix customisation system (Drellishak and Bender (2005)). Asyndetic coordination is a form of coordination where the coordinates are placed next to each other without a coordination marker. Because this form of coordination does not occur in German or Dutch, its interaction with phenomena added to gCLIMB was not verified during development. Including asyndetic coordination for sentences in a grammar that includes a decent range of complex phenomena without ensuring that the right features are shared and restrictions on the application of such rules are imposed leads to inefficiencies in the grammar. Coverage of the grammar drops to 85.3% when adding choices for asyndetic coordination in the choices file.[19]

[18]The constraint was removed from the Matrix core included in gCLIMB after the evaluation. This also required a revision of the word order libraries adding the constraint for languages that do not exhibit VSO and OSV word order in combination with subject drop.

[19]Kilmer and Packard’s original choices file did not include asyndetic coordination.

The forth phenomenon that is not covered is verb phrase pronominalisation (VPP), which has not been considered in gCLIMB. Furthermore, VP ellipsis has been limited to comparative structures for now. The test suite contained one example that exhibited both asyndetic coordination and VPP which could not be captured. Finally, one example formed an imperative structure, which is also not covered by gCLIMB.

As mentioned above, (45) illustrates the only example that gCLIMB currently cannot handle, but is covered by Kilmer and Packard's grammar. In this example, verb-second word order is found in combination with a complementizer. All other phenomena that gCLIMB cannot handle are also not treated in Kilmer and Packard's original grammar. The Grammar Matrix customisation system does not provide correct analyses for any of the phenomena described in this section.

Conclusion about Northern Frisian evaluation

As far as can be told from this small test suite, gCLIMB can create a decent grammar of Northern Frisian. The two phenomena that are currently not covered (the interaction between word order and dropped subject and verb second order in subordinate clauses) can be added without much effort. However, there are not enough details available in the test suite at hand to determine whether Northern Frisian word order can be captured correctly, or whether similar challenges as observed in the Dutch or Danish evaluations will be found.

This test nevertheless indicates that variations in related languages can easily be captured when using CLIMB. The grammar was created in approximately eight hours and covered 94.5% of the test suite without overgeneration. Kilmer and Packard's grammar was developed over several weeks and covered 70.6% of the test items and overgenerated on 1.8%. It should be taken into consideration that Kilmer and Packard also had to develop the test suite which is an effort that cannot be neglected. However, the significant difference in development time and superior competence of the grammar developed

Source	Covered (# ex)	Not covered (# ex)	Coverage (%)
Lasswell (1998)	8	22	26.7%
Kilmer & Packard	69	10	87.3%
Total	77	32	70.6%

Table 6.10: Coverage per source for Northern Frisian

using gCLIMB cannot be explained completely by this additional task. As explained in Chapter 5, it is not possible to compare two efforts of grammar engineering due to the multiple influential factors, but given the fact that Packard is the developer of the ACE parser and generator and has profound knowledge of DELPH-IN grammars makes it unlikely that CLIMB played no part in the difference in results (both in terms of development speed and competence of the grammar).

It should be noted that Kilmer and Packard's test suite contained a number of challenging examples. The 30 examples taken from Lasswell (1998) often exhibited more phenomena than the one they were meant to illustrate. Table 6.10 represents the coverage of Kilmer and Packard's final grammar on examples taken from Lasswell (1998) compared to the coverage on the examples they created themselves. The results in this table confirm that Lasswell's (1998) examples were comparatively challenging; only 26.7% were covered compared to 87.3% of the examples created by the authors. Moreover, six of the ten examples from Kilmer and Packard that are not covered were adaptations from examples provided by Lasswell. As I will explain below, this partially explains why the Northern Frisian grammar got so much benefit from gCLIMB. I will end this part of the evaluation with a note about the importance of a decent test suite which ideally is created with help of a native speaker or field linguist. In particular, I will address the challenges when the linguistic description of a language and grammar engineering efforts are not carried out by the same person and the people involved are not in touch.

Two examples that exhibited phenomena that were not covered at all by the grammar were all intended to evaluate other phenomena. The imperative structure illustrated a bare NP and the sentences coordinated in an asyndetic conjunction illustrated a passive sentence. The Northern Frisian test suite contains several examples that are meant to illustrate a basic phenomenon, but also include complex ones. The gCLIMB grammars for Northern Frisian include implementations of phenomena that were treated in the later stages of the development of gCLIMB, such as long distance dependencies and relative clauses. This partially explains why coverage of the gCLIMB grammars is so much higher than the coverage achieved by Kilmer and Packard's grammar (94.5% versus 70.6%).

I would be reluctant to say however that the grammars I have created for Northern Frisian include correct analyses for the phenomena they cover, even for those phenomena where all examples in the corpus were handled adequately. In many cases, more examples would be needed to know how the language truly behaves. These properties of the test suite reveal that it is extremely difficult to build a grammar based on literature alone. Grammar engineering works best if one has simple examples revealing all possible variations one can think of for a phenomenon and access to native speakers is necessary to obtain such sets. Ideally, the grammar engineer would collaborate with the field linguist (Bender et al., 2012). The grammar engineer has much to gain by such a collaboration, because finding out how the language works exactly is the most difficult step in grammar engineering and an adequate description plays an important role in that process. The fact that information is often missing when one starts implementing a grammar also shows why this collaboration may also be interesting for field linguists. Grammar engineering forces researchers to think of the details and can lead to questions that they would not have thought of asking while focusing on describing the language or carrying out pen-and-paper syntax.

6.2.6 Summary of gCLIMB evaluation

This section presented three evaluations that examined whether gCLIMB can easily be used to create grammars for Germanic languages other than German. The first notable outcome of these experiments is that the argument composition analysis turned out to be slightly more efficient in Dutch than the auxiliary+verb and filler-gap analysis. Even though this may be related to errors in the latter two (which result in more overgeneration), this outcome reveals the complexity of determining which analysis is more efficient. It underlines the importance of systematic testing in a wide range of settings.

When looking at possibilities of creating grammars for other Germanic languages, the results show that CLIMB offers a certain amount of flexibility. Creating the grammars that provided the ultimate results took a couple of days in all three experiments and all three grammars could capture at least part of the behaviour found in the test suites correctly. It is unlikely that similar results could have been achieved using the Grammar Matrix and its customisation system alone. On the other hand, all grammars had shortcomings and in all three languages, behaviour was found that was not foreseen during the development of gCLIMB. New implementations and fundamental revisions of some of the existing analyses will be required to capture this behaviour correctly. The outcome of these experiments shows that the method is in principle suitable for multilingual grammar development, but that it is essential that all variations found in the languages that the metagrammar should cover are considered during development. Focusing on one language only can push certain analyses in a direction that is not suitable for other languages. The evaluation of Dutch revealed that gCLIMB no longer supports Dutch word order correctly. The original analysis was completely abandoned based on evidence from German. It would have been possible to maintain the original analysis in CLIMB, which would have allowed me to "time travel" back to the original analysis for Dutch. This outcome shows that it may be of advantage to make use of this functionality of CLIMB more extensively, especially when more than one language is taken into consideration. The next section describes SlaviCLIMB, a direction of future work where multiple languages will be taken into account throughout the entire grammar engineering effort.

6.3 PaGES, SlaviCore and SlaviCLIMB

PaGES[20] (Parallel Grammar Engineering for Slavic languages in DELPH-IN) (Avgustinova, 2007; Avgustinova and Zhang, 2009)[21] is a project that aims to develop grammars for Slavic languages around a core of typical Slavic phenomena shared among individual languages. Initial steps have been made to combine adopt the CLIMB methodology for developing a Russian grammar. This implementation of CLIMB can also generate SlaviCore (Fokkens et al., 2012a). This section provides a brief description of PaGES and illustrates the role CLIMB can play in this project.

6.3.1 SlaviCore

PaGES is a grammar engineering effort largely inspired by the Grammar Matrix. As explained in Section 2.3, the Grammar Matrix consists of a static core and a dynamic customisation system that generates language specific analyses. Part of the static core provides generic types that are thought to be useful for most (if not all) DELPH-IN grammars. I will use the term *Matrix core* to refer to this part of the Grammar Matrix' static core. The main idea in PaGES is to provide a structure to share analyses for Slavic languages crosslinguistically through inheritance. It can thus be seen as a set of Slavic specific subtypes of the types included in the Matrix core. Figure 6.1 provides a representation of the original SlaviCore setup.[22] Note that the representation is simplified: not all Slavic languages are included, in practice individual languages may inherit some properties directly from the Matrix core and more subcores of SlaviCore are foreseen.

The simplest way to describe SlaviCore would be to see it as a Slavic specific extension of the Grammar Matrix. It should, however, be noticed that

[20] `http://moin.delph-in.net/SlavicTop`, accessed 11 April 2014.

[21] This section presents joint work with Tania Avgustinova and Yi Zhang (hence the use of *we* in this section). Some parts of it have been previously published in Fokkens et al. (2012a) and Fokkens and Avgustinova (2013).

[22] This figure is based on discussions about PaGES with Tania Avgustinova and Yi Zhang.

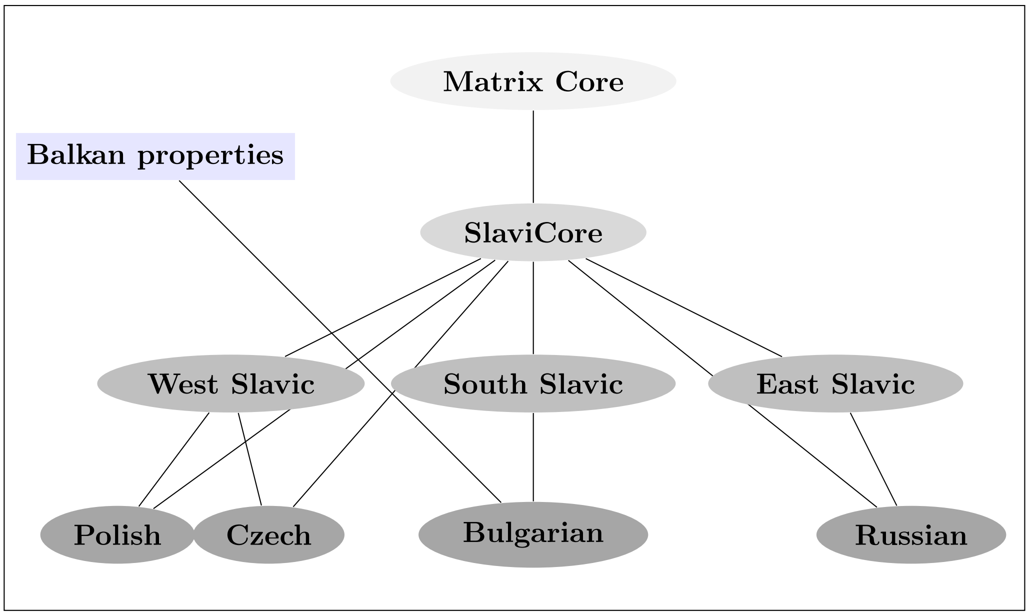


Figure 6.1: Basic architecture of original idea for PaGES

limiting the typological coverage to a closed group of languages provides a rather different perspective on crosslinguistic language modelling.

As explained in Section 2.3, p. 54, the Grammar Matrix has the ambition to provide a platform to develop grammars for any natural human language. Achieving analytical depth is a challenge even for monolingual resources. The additional ambition to provide large typological coverage imposes limitations to the analytical depth that can be achieved using the Grammar Matrix. To illustrate: the first step when working on a Grammar Matrix library is to define clear boundaries to the phenomena and the level of detail it should cover (Bender et al., 2010). This must be done, because it is not feasible to provide options that both capture the exact behaviour of individual languages and cover the full range of known typological variation.

Grammar engineers using the Grammar Matrix can start with the basic analysis provided by the customisation system and adapt it manually to handle the specifics of the language correctly. It is, however, not clear how these simplified basic analyses may influence the final grammar. On the one hand, the customisation system provides a basis the grammar writer can work with and can use to focus on other phenomena. On the other hand, it is not clear whether the basic customised analysis can be adapted easily to capture the details of the phenomenon correctly. There may be cases where analyses have to be replaced completely, with the risk of the customisation system setting the grammar developer on the wrong track. This limitation does not reflect a lack of quality of the resource, but is a direct consequence of the ambition to provide support for all languages.

When working with a closed set of languages, it becomes possible to take the fine details of variation found in individual languages into account. The resource can therefore provide more elaborate analyses that take even details in variations between languages into account.

Furthermore, by working on several Slavic languages in a multilingual grammar engineering project, we hope to improve the feedback loop between developers of the core grammar and engineers working on individual languages. Current implementation projects in SlaviCore are mainly run by three researchers, Avgustinova and Zhang working on Russian and Osenova working on Bulgarian, who collaborate to establish the core grammar. The Slavic language family does not only reveal many common properties, but also exhibits a decent range of typological variation. The development of a common core for these languages can thus lead to valuable insights into linguistic generalisations and different levels of sharing implementations. The resulting SlaviCore in itself can therefore provide important feedback to the Grammar Matrix.

6.3.2 SlaviCLIMB

The original design of SlaviCore only shared grammatical properties statically. SlaviCLIMB adds a dynamic dimension to this approach that plays a role on both a practical and a theoretical level. This section provides an outline of CLIMB's role in developing Slavic grammars around SlaviCore.

Static versus dynamic sharing

Sharing analyses statically provides a useful basis for multilingual grammar development. However, only properties that are completely identical can be shared across languages. Properties that are shared by a subset of languages can only be shared by creating subcores (or they are included in some grammars without being used). Furthermore, statically shared analyses are not suitable for capturing variations in related phenomena. Dynamic sharing provides more flexibility and can support such variations. This section will elaborate on the differences between static and dynamic sharing.

SlaviCore allows individual languages to incorporate analyses from SlaviCore or one of its subcores through inheritance from appropriate types. Minor crosslinguistic variations can be dealt with either by keeping some properties underspecified in the core or providing a set of types languages can select from. In the latter case, properties that do not occur in the language will be present in the type hierarchy without contributing to an instantiated type.

It is, however, not possible to include contradictory properties in a single type hierarchy. The possibilities of storing minor variations in CLIMB's architecture can therefore provide a superior model for cross-Slavic variations. It can include full accounts of linguistic variations of a given phenomenon, including contradictory properties, which is a more elegant and ultimately more helpful solution than underspecified analyses in a core. Likewise, the derived grammars need not contain definitions that are meant to cover properties of other languages, which forms an improvement to the second solution for capturing linguistic variation.

I will illustrate the difference between sharing analyses in a static core and using CLIMB through Avgustinova's model of case marking. Through this illustration, I will show that CLIMB provides additional means to verify linguistic hypotheses.

Modelling Slavic case

Avgustinova (2007) points out that the notion of *case*, as it is used in linguistic theory, is complex. It may refer to a range of linguistic properties at different levels of description (Avgustinova, 2007, p.25). The main distinction lies in the reference to case as morphological marking on the one hand and a syntactic notion where case is linked to a particular function on the other hand.[23]

Avgustinova proposes a general model for case that explicitly expresses this distinction. The type *case* has two subtypes: *f-case* for modelling case on the level of syntactic functions and *m-case*, which models the morphological forms of *case* that may be found. Functional case (*f-case*) is further divided into structural case (*str-case*) and inherent or lexical case (*lex-case*), following Heinz and Matiasek (1994) and Przepiórkowski (1996). Avgustinova's proposal goes beyond those of Heinz and Matiasek (1994) and Przepiórkowski (1996), because she includes adpositional marking, which plays the same double role as case in linguistic theory, in her case model.

A partial representation of how this general vision of case is used to model Slavic case is given in Figure 6.2. Only the top of the hierarchy is represented. Multiple inheritance is used to link case assignment to case marking. The part of the hierarchy that establishes this is not shown in Figure 6.2. Morphological cases that mark the form required by an assigned functional case share a subtype with this case. For instance, the type *s-nom* for nominative subjects inherits from *subj-case* and *m-nom*.

SlaviCore currently contains an elaborate case hierarchy of 117 types that captures the wide variety of case marking found in Slavic languages. The hierarchy in 6.2 is further extended to capture (among others) the range of subjective cases (most subjects in Slavic cases are nominatives, but dative subjects occur in Polish and Russian and Russian also has genitive subjects), objective cases (mostly accusative, but also genitive and instrumental in

[23]The explanation in this section is simplified from Avgustinova's (2007) original proposal (which, for instance, also mentions a semantic level where case is linked to semantic roles). See Avgustinova (2007, p. 24-35) for a more elaborate description.

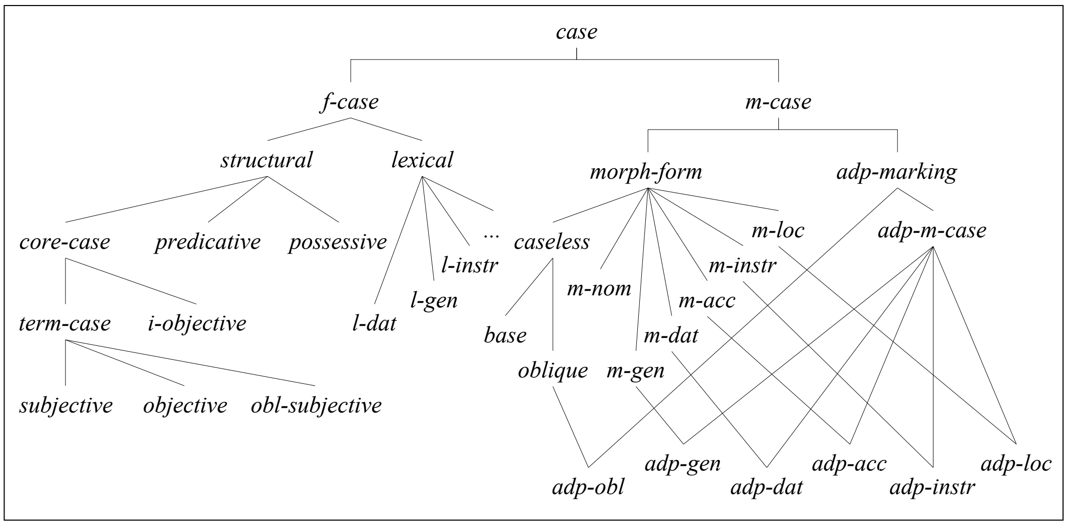


Figure 6.2: Partial representation of Slavic case, created by Avgustinova

Polish and Russian) and Bulgarian's base case.

Modelling case with CLIMB

The Grammar Matrix customisation system can add types representing underspecified case values when forms with ambiguous case are found in the language. The hierarchy starts out as a flat hierarchy, where all cases inherit directly from the type *case*. When the user defines a lexical item or rule with ambiguous case marking, the hierarchy is updated to include this ambiguous value. For instance, the German determiner *das* which can be nominative (*nom*) or accusative (*acc*) case, leads to a type *nom-or-acc*, which inherits from *case* and is a supertype of *nom* and *acc*. This functionality is also used in CLIMB.

A similar mechanism can be used to create the subtypes that link morphological cases to the right functional cases. SlaviCLIMB includes an option that makes sure the top of the case hierarchy is included in the grammar. Users can define which morphological cases can be found in a given syntactic function in *choices*. Based on these definitions, the metagrammar can adapt the hierarchy adding the common subtypes of *f-case* and *m-case* that are found in the language. SlaviCLIMB provides the cases the user can select, but only includes the ones that are defined by the user in the grammar. In other words, a Slavicist can define lexical items and morphemes with certain linguistic properties based on observations and the corresponding case hierarchy will be derived from these definitions. The case hierarchies created in this manner for individual languages can be compared to the original proposal by Avgustinova. SlaviCLIMB can provide an environment to empirically verify Avgustinova's theoretical work.

Furthermore, CLIMB's original purpose of parallel analyses can be used to leave grammar writers at liberty to include the functional-morphological distinction or not. Bulgarian only has limited overt case marking and BURGER's current implementation is a simple hierarchy where *case* has two subtypes (Osenova, 2010). Bulgarian can be modelled through Avgustinova's model and this approach makes sense when looking at Bulgarian as part of the Slavic language family, but when considered in isolation, the simple hierarchy may be the better option. SlaviCLIMB supports both strategies.

Current state of SlaviCLIMB

The first stage of integrating SlaviCore and the RRG into SlaviCLIMB is reported in Fokkens et al. (2012a). At the time 84% of the types in SlaviCore and 70% of those in RRG was covered after 36 hours of conversion. The complete process of integrating SlaviCore and RRG into CLIMB took 42 hours. The possibility of distinguishing between phenomena and properties belonging to the language family on the one hand, and language specific properties on the other hand was added to SlaviCLIMB from the start.

In this first integration, the full case hierarchy had to be defined through *choices*. The current implementation allows users to include the top of the case hierarchy in the grammar through a single choice. It also supports a set of crossclassification options and provides the possibility of defining exceptions.

Revisions to support case modelling as sketched above are currently under development. Section 6.3.3 outlines the vision we have for SlaviCLIMB on the long run.

6.3.3 Vision of SlaviCLIMB

The ultimate research goal of PaGES is to investigate crosslinguistic grammar development for a set of related languages.[24] The focus of the project lies in the development of SlaviCore and components providing common properties for subgroups of the Slavic language family. SlaviCLIMB has a supporting role in this process. The main role of this dynamic environment would be to support empirical research on shared properties of Slavic languages, as sketched above for Slavic case. Furthermore, phenomena that can be seen as parameters (in the sense that some Slavic language exhibit them and some do not) will remain part of SlaviCLIMB. Such a setup facilitates the creation of new grammars and is particularly interesting for creating grammars for dialects spoken in linguistic border areas where phenomena of both languages are expected. Finally, SlaviCLIMB is meant to serve a practical purpose. Grammar writers will be able to continuously use the customisation machinery to build new grammars and it will host variations for grammars. For the Russian Resource Grammar, for instance, two variations are planned: one that focuses on elegant linguistic analyses and one that aims at achieving decent coverage on text with the help of a Treebank derived lexicon.

The main developers of Slavic grammars will, most likely, not be Grammar Matrix experts and possibly not even be trained as programmers. The desire to use CLIMB in SlaviCore has been one of the main reasons to develop declarative CLIMB, which allows grammar engineers to write their alternating libraries in standard TDL (see 3.2, p. 81 for details). A disadvantage of declarative CLIMB, however, is that it does not provide the flexibility of procedural CLIMB to dynamically create types for lexical items and rules or hierarchies. We therefore propose to design SlaviCLIMB as a mixture between procedural CLIMB and declarative CLIMB. The main idea is that SlaviCLIMB

[24]The ideas expressed in this section were published in Fokkens and Avgustinova (2013). Some parts of the text were taken from this paper.

will provide an environment where linguists and computational linguists with different areas of expertise can contribute to SlaviCore and grammars for individual Slavic languages.

Grammar engineers can design and implement analyses for individual languages or crosslinguistically occurring phenomena. These analyses can be written in declarative CLIMB. Cases where crossclassifications plays a role (e.g. types representing declination or inflection paradigms) procedural CLIMB will be used. The functions that are used for this in procedural SlaviCLIMB all use the same basic structure. In principle, it should be possible to implement code that can generate the necessary additions to SlaviCLIMB automatically. An interface will be designed where grammar engineers can upload a declaratively defined type, indicate which feature values they wish to vary and define the range of variations. Based on this input, procedural SlaviCLIMB can then be extended automatically.

Unlike gCLIMB, where source code and choices files were manipulated directly at all times, a Slavic specific variation of the Grammar Matrix' questionnaire will be developed along side SlaviCLIMB. This should allow Slavicists to create new grammars or extend lexica and morpho-syntactic rules for existing grammars without being trained as grammar engineers (or even being HPSG experts). The initial setup can be developed from the current state of CLIMB with relatively limited effort. However, setting up the exact content is not a trivial task and will be ongoing research requiring both grammar engineering and Slavic expertise. One of the main topics of investigation would be the question of what should be included in the static core, what should be maintained in the dynamic component and how to design these components and questionnaire cleverly to support the range of Slavic variations.

Figure 6.3 illustrates the collaboration between a Slavicist and grammar engineer working on a grammar for Bulgarian.[25] The ellipses indicate components that are generated by the system, the rectangle and diamond represent components requiring active interaction with experts. Slavicists can specify linguistic properties in the web-based questionnaire and grammar engineers

[25]This figure was taken from Fokkens and Avgustinova (2013).

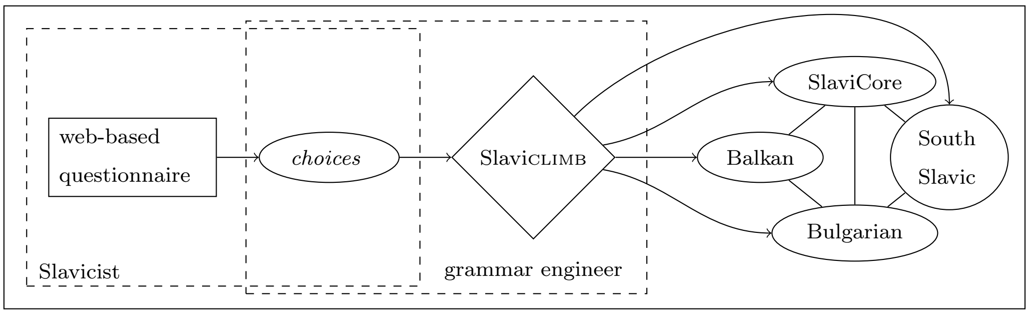


Figure 6.3: Overview of grammar development with SlaviCLIMB

are mainly responsible for developing the analyses in SlaviCLIMB. In a perfect scenario, Slavicists would be able to create high coverage precision grammars for Slavic languages through the questionnaire. It is not realistic to expect that this will be achieved. Slavicists will run into limits of the resource and analyses that do not work as intended. Interaction between Slavicists and grammar engineers is essential for creating a good resource and it is likely that grammar engineering expertise will always remain necessary to improve grammars created through SlaviCLIMB. A Slavicist who is serious about creating a large coverage grammar for a Slavic language will thus ultimately need to get familiar with grammar engineering or collaborate with a grammar engineer.

Nevertheless, a significant amount of work can be done through this method. Borisova's (2010) thesis illustrates the possibilities of using grammar customisation to model complex phenomena. She modelled polypersonal agreement in (the non-Slavic language) Georgian almost exclusively using the Grammar Matrix customisation system. Polypersonal agreement refers to the ability of the verb to agree with more than one of its arguments. Georgian verbs agree with subjects, objects and indirect objects. Georgian agreement exhibits a number of complex phenomena including zero and ambiguous morphemes, varied expressions of the same grammatical function and morphemes being dropped or altered in presence of specific other morphemes (Borisova, 2010). Georgian was, to my knowledge, not investigated when agreement was implemented in the customisation system. As explained above, SlaviCLIMB offers a better setting for creating models of complex phenomena through customisation than the Grammar Matrix, because it only needs to deal with a closed set of languages. It may not be realistic that SlaviCLIMB allows Slavicists to create resource grammars for new Slavic languages from scratch through the web-based questionnaire, but we expect that it will be possible to create medium-sized grammars that cover interesting phenomena.

The development of SlaviCLIMB requires different fields of expertise and can only be realised if Slavicists and grammar engineering collaborate. Experts in computational syntax and Slavistics should be able to explore their expertise in SlaviCLIMB with a minimum of technical support. As such, PaGES forms a good platform for investigating whether CLIMB can be adapted as a general approach to grammar engineering in the DELPH-IN community.

6.4 CLIMB for second language learners

The previous sections have discussed how CLIMB can be used to create grammars for related languages. This section addresses the possibility of creating alternative grammars for the same language so that they may be used as part of a system that helps second language learners. It is structured as follows. Section 6.4.1 will provide background information on creating DELPH-IN grammars for language learning. This is followed by an explanation of developing alternative grammars for language learning with CLIMB in Section 6.4.2. Section 6.4.3 provides a description of a prototype for language learning that is included in gCLIMB.

6.4.1 DELPH-IN grammars for language learning

In order to become fluent in another language, learners must practice. Feedback on language use is essential to improve ones language skills. However, as Bender et al. (2004) point out, there is limited room for practice and feedback for individuals in classroom settings. Learners need to practice out-side the classroom, where they often cannot get feedback on their language skills. Bender et al. (2004) describe Arboretum, a prototype system that can provide such feedback.

Arboretum includes an adapted version of the English Resource Grammar (Flickinger, 2000, ERG). In this version, so-called mal-rules (Schneider and McCoy, 1998) were added. These rules can parse sentences containing typical non-native speaker mistakes and map them to the intended semantics. Corrected sentences can be generated from the intended semantics. This approach has since been adopted in two more projects with DELPH-IN grammars. It is used in the Language Arts & Writing (LA&W) course of The Educational Program for Gifted Youth (EPGY) (Suppes et al., 2012) and in a Norwegian Grammar Sparrer[26] (Hellan et al., 2011).

The basic idea behind Arboretum and the Norwegian Grammar Sparrer can also be used to help native speakers improve their language skills. The EPGY program develops courses for American school children. The LA&W course aims at improving their writing skills. A version of the ERG with mal-rules of typical mistakes made by school children is used to support sentence and paragraph composition exercises. The children receive a selection of words and build sentences by clicking on them. The sentences they produce are parsed by the ERG. If the sentence can only be parsed using a mal-rule, it is considered ill-formed. The names of the mal-rules can give insight into the kind of errors children make. Suppes et al. (2012) show that writing skills of children improved more compared to other children as they practiced more with the system.

The goal and setup of the Norwegian Grammar Sparrer are closer to those described in Arboretum. It is meant for non-native speakers who want to learn Norwegian. Language learners can access the sparrer through a web-interface[27] where they can type in sentences in Norwegian. The sentence is parsed by NorSource (Hellan and Haugereid, 2003) augmented with mal-

[26] `http://typecraft.org/tc2wiki/A_Norwegian_Grammar_Sparrer`, accessed 14 December 2013.

[27] `http://regdili.idi.ntnu.no:8080/studentAce/parse`, accessed 15 December 2013.

rules. If a structure can only be parsed with mal-rules, a brief explanation of the error is provided, together with a link to a more elaborate explanation and the possibility of generating well-formed alternatives. Just like in Arboretum, the semantics of the ill-formed structure is used as input to generate well-formed alternatives. The Norwegian Grammar Sparrer currently supports practicing basic structures providing feedback on sentence syntax (word order, obligatory subjects), verb complementation (form and order of arguments, obligatoriness, reflexives and interaction between form and coordination) and noun phrases (form, agreement, determiners, genitives and pronouns).[28]

6.4.2 Language learning with CLIMB

The previous section outlined the basic approach taken when using DELPH-IN grammars as part of a system that helps to improve language skills. In this section, I will outline possibilities provided by CLIMB for such work.[29]

The systems described above consisted of grammars augmented with mal-rules. There are several ways in which CLIMB may facilitate the task of creating such grammars. First, it is easier to create more versions of the grammar which may help to support tasks with a different focus or aimed at people with different language backgrounds. Second, CLIMB offers a more modular implementation environment than traditional grammar writing, which makes it easier to make fundamental changes in the grammar when this is needed in order to give precise feedback. I will elaborate on these issues below.

One of the main challenges in using mal-rules is the question of what rules need to be included in the grammar. Typical errors can be collected by analysing corpora of texts produced by non-native speakers or school children (depending on the target of the system). Mal-rules can then be defined to capture these errors. The name of the mal-rule can indicate which error was made and trigger specific feedback, as done in the Norwegian Grammar

[28]`http://typecraft.org/tc2wiki/Grammar_sparring_phenomena`, accessed 15 December 2013.

[29]Several ideas presented in this section came out of discussions with Magda Wolska.

Sparrer. However, it may become harder to control the behaviour of mal-rules as the grammar grows. Interaction between rules is already challenging when writing a grammar that merely captures the correct behaviour. When this is augmented with mal-rules that need to indicate which error was made in a given sentence and where, it will become increasingly hard to control the interaction between these rules. Identifying the error that was made and making sure the efficiency of the grammar remains acceptable will become increasingly difficult as the grammar grows.

This challenge may be tackled by focusing on one (or a few) errors at the time. This approach is followed in several experiments carried out as part of the Interreg project ALLEGRO.[30] For instance, Wolska and Wilske (2010) describe a task where second language learners practice word order in German subordinate clauses, whereas their study in Wilske and Wolska (2011) allows students to practice with German datives. The grammars used in these studies are encoded in Java Speech Grammar Format and each task requires a different grammar that needs to be written from scratch (Wolska, p.c.).

In Arboretum, the EPGY program or the Norwegian Grammar Sparrer the grammars with mal-rules rely on a rich existing grammar and there is no need to create grammars from scratch for each source. However, alternative versions of the grammar that focus on different errors also make sense when DELPH-IN grammars are used. Apart from efficiency issues and challenges with interaction of many mal-rules mentioned above, it can make sense to adapt the grammar to the level of the students. German has relatively flexible word order, but it may not be a good idea to permit beginning learners to build structures that are highly unusual and can only be used under specific circumstances. More advanced speakers on the other hand should practice with variations in word order to improve their German. Furthermore, students with different native languages tend to make different mistakes. When the linguistic background of students is known, alternative grammars can focus on typical mistakes for the group. When using CLIMB, mal-rules capturing specific mistakes can be placed in their own libraries. Grammars can

[30] `http://www.allegro-project.eu/`, accessed 15 December 2013.

be created that focus on a specific error or a set of selected errors using these libraries. This setup facilitates the maintenance of large grammars meant to be used for grammar checking.

When looking at errors typically made by speakers from a specific linguistic background, multilingual CLIMB resources may be particularly interesting. The mal-rules for one language can be based on the normal rules of another language in the resource. For instance, native speakers of Dutch will not easily make mistakes with verb second word order or the relative order of words in the Mittelfeld in German (since the language show highly similar behaviour), but they may be inclined to use Dutch word order in the verbal cluster. Investigating the possibilities of such an approach is however challenging and would require extensive collaboration with experts on second language learning.

A final contribution CLIMB provides to adapting grammars for language learning lies in the increased modularity of CLIMB. When creating alternations in a lexicalist grammar that can parse ungrammatical strings, two approaches may be followed. The first option is to include alternative versions of lexical types or to use "mal-lexical types" rather than mal-rules. A specific type may leave grammatical values of its argument underspecified or a modifier will not have all its grammatical properties defined so that it can combine with heads that are otherwise forbidden. Adding such alternative elements is straight-forward in CLIMB. It is for instance possible to define instances of the supertypes of a lexical item, in addition to instances of fully defined types of the item.

A drawback of this approach is that multiple entries are required for each lexical item that may be used incorrectly. An alternative approach would include alternative versions of rules in the grammar that allows lexical types to combine despite conflicting grammatical properties. Again, CLIMB's flexibility can come in handy in this case. Constraints may be encoded in manners making it non-trivial to construct mal-rules that can capture errors related to these constraints. If a constraint is a HEAD feature for instance, it will automatically be shared between head-daughter and mother in any headed-

```
basic-head-1st-comp-no-case := basic-head-comp-phrase &
  [ SYNSEM.LOCAL.CAT.VAL.COMPS #comps,
    HEAD-DTR.SYNSEM.LOCAL.CAT.VAL.COMPS
     <[ SYNSEM [ LOCAL [ CAT [ HEAD.FORM #form,
                               VAL #val,
                               MC #mc ],
                         CONT #cont,
                         AGR #agr,
                         COORD #coord ],
                 NON-LOCAL #non-local,
                 OPT #opt ],
        ARGS #args,
        INFLECTED #inflected ] . #comps >,
    NON-HEAD-DTR [ SYNSEM [ LOCAL [ CAT [ HEAD.FORM #form,
                                          VAL #val,
                                          MC #mc ],
                                    CONT #cont,
                                    AGR #agr,
                                    COORD #coord ],
                            NON-LOCAL #non-local,
                            OPT #opt ],
                   ARGS #args,
                   INFLECTED #inflected ] ] .
```

Figure 6.4: Simplified representation of a head-complement mal-rule that ignores CASE when defined as a HEAD feature

phrase. Furthermore, when a head-daughter subcategorises for an argument with a specific head value, rules need to be created that unify all features of the subcategorised element except for this one feature value. The more embedded this value is, the more complex the rule becomes. If the feature in question (MYFEAT) is located inside of HEAD, the path to it in a standard Matrix grammar is SYNSEM.LOCAL.CAT.HEAD.MYFEAT. This means that all values of SYNSEM except LOCAL must be shared. All values of LOCAL except CAT must be shared, all values of CAT except for HEAD must be shared and finally all features of HEAD except for MYFEAT must be shared. Figure 6.4 provides a simplified illustration of what this may look like if a head-complement mal-rule ignores the case value of the complement. The structure is simplified in the sense that not all features standardly defined for signs are represented.

```
basic-head-1st-comp-no-case := basic-head-comp-phrase &
  [ SYNSEM.LOCAL.CAT.VAL.COMPS #comps,
    HEAD-DTR.SYNSEM.LOCAL.CAT.VAL.COMPS
     <[ SYNSEM [ LOCAL #local,
                 NON-LOCAL #non-local,
                 OPT #opt ],
        ARGS #args,
        INFLECTED #inflected ] . #comps >,
    NON-HEAD-DTR [ SYNSEM [ LOCAL #local,
                            NON-LOCAL #non-local,
                            OPT #opt ],
                   ARGS #args,
                   INFLECTED #inflected ] ] .
```

Figure 6.5: Simplified representation of a head-complement mal-rule that ignores CASE when it is defined as a SYNSEM feature

If the feature MYFEAT is located at SYNSEM on the other hand, the rule only needs to share everything in SYNSEM except for MYFEAT. The head-complement rule that ignores case when it is defined as a feature of SYNSEM rather than of HEAD is provided in Figure 6.5. A setup that facilitates changing the location of features makes it much easier to create mal-rules that circumvent violation of specific features.

Adapting lexical items so that they impose less constraints provides a cleaner and easier implementation than identifying special rules that ignore parts of the elements they combine. As long as the lexicon is limited to a predefined set, alternative lexical items is thus preferable to specialised rules. On the other hand, special rules form a more generic and flexible approach to cover typical mistakes by language learners. The best option thus depends on the situation. CLIMB's flexibility provides support in both scenarios.

6.4.3 A prototype in gCLIMB

German adjective endings change depending on case, number, gender of the noun and the specifier preceding it. The correct endings are provided in Table 6.11. Other endings lead to ungrammatical noun phrases. The mal-rules in the grammar should ideally identify the exact mistake made by students, i.e. whether they selected an ending based on the wrong number, gender, case or preceding specifier.

The adjective follows one of the following words (in any form): *der*, *dieser*, *jener*, *jeder*, *mancher*, *solcher*, *welcher*, *aller*, *sämtlicher*, *beide*			
case	masculine	feminine	neutral
nom	der gut**e** Wein	die gut**e** Milch	das gut**e** Bier
gen	des gut**en** Wein(e)s	der gut**en** Milch	des gut**en** Bier(e)s
dat	dem gut**en** Wein	der gut**en** Milch	dem gut**en** Bier
acc	den gut**en** Wein	die gut**e** Milch	das gut**e** Bier
The adjective follows one of the following words (in any form): *ein*, *kein*, *mein*, *dein*, *sein*, *ihr*, *unser*, *euer*, *Ihr*			
case	masculine	feminine	neutral
nom	ein gut**er** Wein	eine gut**e** Milch	ein gut**es** Bier
gen	eines gut**en** Wein(e)s	einer gut**en** Milch	eines gut**en** Bier(e)s
dat	einem gut**en** Wein	einer gut**en** Milch	einem gut**en** Bier
acc	einen gut**en** Wein	eine gut**e** Milch	ein gut**es** Bier
The adjective is not preceded by a specifier			
case	masculine	feminine	neutral
nom	gut**er** Wein	gut**e** Milch	gut**es** Bier
gen	gut**en** Wein(e)s	gut**er** Milch	gut**en** Bier(e)s
dat	gut**em** Wein	gut**er** Milch	gut**em** Bier
acc	gut**en** Wein	gut**e** Milch	gut**es** Bier
Adjective endings for plural adjectives			
case	"*der*" specifier	"*ein*" specifier	no specifier
nom	die gut**en** Weinen	keine gut**en** Weinen	gut**e** Weinen
gen	der gut**en** Weinen	keiner gut**en** Weinen	gut**er** Weinen
dat	den gut**en** Weinen	keinen gut**en** Weinen	gut**en** Weinen
acc	die gut**en** Weinen	keine gut**en** Weinen	gut**e** Weinen

Table 6.11: Overview of German adjective endings (repeated)

This is achieved by creating a mal-rule for each individual error. Figure 6.6 represents the basic phrases enforcing individual properties to agree between

```
case-sharing-adj-head := adjective-head-phrase &
        [ SYNSEM.LOCAL.CAT.HEAD.CASE #case,
          NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD.CASE #case ].

strength-sharing-adj-head := adjective-head-phrase &
        [ SYNSEM.LOCAL.CAT.HEAD.STRONG #strength,
          NON-HEAD-DTR.SYNSEM.LOCAL.CAT.HEAD.STRONG #strength ].

gend-sharing-adj-head := adjective-head-phrase &
        [ SYNSEM.LOCAL.CONT.HOOK.INDEX.PNG.GEND #gend,
          NON-HEAD-DTR.SYNSEM.LOCAL.AGR.PNG.GEND #gend ].

number-sharing-adj-head := adjective-head-phrase &
        [ SYNSEM.LOCAL.CONT.HOOK.INDEX.PNG.NUM #number,
          NON-HEAD-DTR.SYNSEM.LOCAL.AGR.PNG.NUM #number ].

 syn-agr-adj-head-phrase := case-sharing-adj-head &
                                strength-sharing-adj-head.

 sem-agr-adj-head-phrase := gend-sharing-adj-head &
                                number-sharing-adj-head.

 all-agr-adjective-head-phrase := syn-agr-adj-head-phrase &
                                      sem-agr-adj-head-phrase.

 all-but-case-adj-head-phrase := sem-agr-adj-head-phrase &
                                           strength-sharing-adj-head.
```

Figure 6.6: Mal-rules for adjectives enforcing different grammatical properties

adjective and noun. The grammar includes a rule that insists on gender, number and case, but ignores the specifier. Another rule (the *all-but-case-adj-head-phrase* in Figure 6.6) ignores the case value, but insists on the other three being correct, etc. Finally, the grammar needs one general mal-rule that ignores all grammatical aspects. This rule can parse sentences where the adjective was wrong in more than one way. gCLIMB currently includes an option that incorporates these mal-rules in the grammar. A toy system was made that can be run on a terminal. Users can type in sentences that are sent to the grammar running on a server. A basic script analyses the set of parse trees that comes back from the grammar. If the set contains a tree that does not use any mal-rules, the user receives a message that the sentence is correct. If the sentence needs a mal-rule, the system returns a message indicating that the grammatical property or properties that the user got wrong in choosing an ending. The grammars used in this setting are small and the parser returns all analyses that it finds for a given sentence. The most general mal-rule always returns a parse when the adjective ending is wrong. The algorithm going through the parses first looks for the more specific mal-rules. The message to the user will be tailored to the specific mal-rules that are identified in the parse forest. If only the most general mal-rule resulted in a parse, the user receives a generic message indicating that the ending is wrong without further specification.

The current implementation in gCLIMB does not alter the feature structure of adjectives and nouns. The features that ensure correct adjective endings in CLIMB are CASE, NUM, GEND and STRONG.[31] In standard German grammars created by gCLIMB, the features CASE and STRONG are head features, NUM and GEND are features of the person-number-gender type, which is part of the INDEX. All features are embedded relatively deeply in the sign. The syntactic features CASE and STRONG are located at SYNSEM.LOCAL.CAT.HEAD and semantic features GEND and NUM at SYNSEM.LOCAL.CONT.HOOK.INDEX.PNG. As explained above, the more a feature is embedded, the more other prop-

[31]The name STRONG is based on traditional grammars for German which use the term *strong* to refer to endings when no specifier is present and *weak* for endings of adjectives following a *der*-word.

erties need to be shared explicitly for a rule that exempts this feature from unification. Moreover, it is inconvenient for the general rule that exempts all four properties from unification that the features are not located at the same place. The possibility of changing the feature geometry using the feature geometry library taken from the Grammar Matrix customisation system and the path reduction and completion algorithms of CLIMB will be examined in future work.

6.5 Spring cleaning revisited

Fokkens et al. (2011) describe two approaches for detecting redundancy in HPSG grammars. The first approach is the spring cleaning algorithm.[32] Spring cleaning focuses on identifying portions of the grammar that do not play any role in the set of sentences it recognises or the structures it assigns to them. Such artefacts can accrue in a grammar because abandoned analyses are not completely removed or because the grammar is built on a cross-linguistic resource but does not use all of the infrastructure that resource provides. Spring cleaning is intended to be used in the course of grammar development and as such must leave the grammar in a state that is still easy to maintain. It largely maintains the original structure of the grammar.

In contrast, the second technique, Ôgrammar compressionÕ, computes the smallest subset of a type hierarchy that can assign the same structures to the same sentences as the original grammar. The grammar compression algorithm, implemented by Yi Zhang, removes not only the types taken out in spring cleaning, but also those that exist only to express generalisations over their subtypes.

This section presents the results of running the spring cleaning algorithm and grammar compression algorithm. First a brief description of the main idea behind the spring cleaning algorithm will be given followed by an explanation of grammar compression. The spring cleaning results are part of the prepar-

[32]Parts of this section are taken from Fokkens et al. (2011).

ation for the crosslinguistic grammar comparison presented in Section 6.6.

6.5.1 Spring cleaning and grammar compression

Spring cleaning

Section 2.2.2 explained how parsing and generation with DELPH-IN grammars work. The type hierarchy defines grammatical properties. During parsing and generation, the parsing and generation algorithms manipulate instances that inherit from the type hierarchy. Structures are built by unifying instances. The type hierarchy determines whether unification between two instances is allowed. This means that types that do not contribute to the definition of any instance, nor influence the possibility of instances unifying do not have an impact on the grammar.

The spring cleaning algorithm goes through the grammar starting at its instances. It identifies (1) which types contribute to the instances' definitions and (2) which types influence unifications between the types identified in (1). It returns a version of the grammar including only types that are identified in these two steps (as well as a copy of the original grammar and files indicating which types were removed). A full description of the algorithm and revisions made to the version evaluated in Fokkens et al. (2011) can be found in Section 3.3.1, p. 100.

Grammar compression

The grammar compression algorithm runs on grammars compiled by PET. The PET parser includes a compiler for TDL grammars. This compilation consists of several steps which influence which types are ultimately used by the parser. Recall from Section 2.2.1 that type hierarchies must be a bounded complete partial order (BCPO). The first step in compilation inserts greatest lower bounds in the grammar in order to fulfil this requirement. The next step expands all type definitions so that the constraints they inherit from their supertypes are directly defined on the types themselves. Finally, a bit-

	top	*a*	*b*	*c*	*d*	*e*	*f*	*glb1*
top	1	1	1	1	1	1	1	1
a	0	1	0	1	1	1	0	1
b	0	0	1	0	1	1	1	1
c	0	0	0	1	0	0	0	0
d	0	0	0	0	1	0	0	0
e	0	0	0	0	0	1	0	0
f	0	0	0	0	0	0	1	0
glb1	0	0	0	0	1	1	0	1

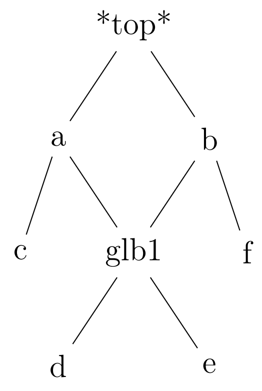


Figure 6.7: Example of a subsumption table in a compiled grammar

matrix is built that encodes all subsumption relations between types in the grammar. Because inherited constraints are explicitly defined on types and information on subsumption relations can be read off the matrix directly, supertype-subtype relations that do not influence subsumption relations do not play a role in the grammar any longer.

The table on the left hand side in Figure 6.7 displays the subsumption table for the figure on the right. The rows of a type indicates which types it subsumes (1=subsumption, 0=no subsumption). The type **top** subsumes all other types in the hierarchy and is only subsumed by itself, type *a* subsumes, types *c*, *glb1*, *d* and *e* and is subsumed by **top** and itself, etc. We can tell directly from the matrix that type *b* is equal to or more general than type *b*, *d*, *e*, *f*, and *glb1*, without looking at the complete structure provided by the type hierarchy.

Zhang's grammar compression algorithm starts with the instantiated types and marks them as well as the values of all their constraints (including the inherited ones). In the next step, it marks the glbs of the types identified in the first step. This completes the sub-BCPO. All remaining types could be removed from the grammar without having an impact on its behaviour (given that constraints defined on supertypes are pushed down on their subtypes). Zhang's proof that a compressed grammar is equivalent to the originally defined grammar can be found in Fokkens et al. (2011).

In principle, types that are identified as having no computational impact on the grammar by the compression algorithm are a superset of those identified by the spring cleaning algorithm. The grammar compression algorithm results in a flatter hierarchy. Some of the types that do not have an impact on the grammar do play a role in representing linguistic generalisations or in the maintainability of the grammar. The spring cleaning algorithm maintains the general structure of the type hierarchy. It only removes parts that do not contribute to any instances in the grammar from the bottom of the hierarchy.

6.5.2 Results

The grammars included in the spring cleaning and grammar compression experiments are presented in Table 6.12. Some preparations needed to be carried out before the grammars could be spring cleaned or compressed. Some of the more recent grammars contain comments in the type definition (indicated by quotes). Neither the compilation component of PET nor the TDL processing code in the spring cleaning grammar can parse types that are defined using this format. A script was run on the tdl files of the grammar to clear these comments out of the type definitions.

Table 6.13 presents the outcome of running grammar compression and spring cleaning on several DELPH-IN grammars.[33] The first column indicates the total size of the grammar. The second column provides the number of types identified as inactive by the grammar compression algorithm. The reduction rate that would be obtained if we were to compress grammars to only computationally active types is indicated in the first column labeled *Rate*. The

[33]There are two versions of the Wambaya grammar. They include different analyses for auxiliaries (Bender, 2010).

Name	Language	Reference
English Resource Grammar (ERG)	English	Flickinger (2000)
Jacy Japanese Grammar (Jacy)	Japanese	Siegel and Bender (2002)
GG	German	Müller and Kasper (2000), Crysmann (2005)
SRG	Spanish	Marimon (2010)
LXGram	Portuguese	Branco and Costa (2010)
KRG	Korean	Kim and Yang (2003), Song et al. (2010)
Norsyg	Norwegian	Haugereid (2011)
MCG	Mandarin Chinese	Zhang et al. (2011)
BURGER	Bulgarian	Osenova (2010)
wmb (wmb)	Wambaya	Bender (2008a)
ManGO	Mandarin Chinese	–

Table 6.12: Grammars included in spring cleaning/grammar compression experiment

number of types removed by the spring cleaning algorithm is given in the fifth column. Finally, the second column that is labeled *Rate* provides the reduction rate for the grammar when it is spring cleaned.

The grammars are divided in three groups reflecting the way the grammars were processed: The ones in the first group were spring cleaned according to the original spring cleaning methodology, starting from the instances of the grammar. The version of Norsyg containing the full lexicon cannot be compiled by PET. It is therefore not possible to run the grammar compression algorithm on this grammar.

While spring cleaning Jacy and SRG, the algorithm ran into memory issues. The algorithm had no problem interpreting the type hierarchy, but stalled while parsing the instances of the grammar. The algorithm was revised to only store unique supertypes, values and attributes rather than the complete set of instances (including the complete lexicon), but this did not solve the problem. It occurs even when parsing small files containing instances and

Grammar	Size	In active	Rate	Spring Cleaned	Rate
Norsyg (small)	5,026	4,005	79.7%	1,320	26.3%
Norsyg (+nlk)	5,026	–	–	177	3.5%
wmb (arg-comp)	3,439	2,695	78.4%	178	5.2%
wmb (aux+verb)	3,443	2,703	78.5%	187	5.4%
BURGER	2,513	1,833	72.9%	368	14.6%
ManGO	1,649	1,400	84.9%	211	12.8%
Jacy	2,548	1,308	51.3%	611	24.0%
SRG	6,118	3,827	62.6%	145	2.8%
LXGram	7,958	6,439	80.9%	201	2.5%
KRG	2,369	1,743	73.6%	260	11.0%
MCG	1,483	1,275	86.0%	435	29.3%
ERG	9,135	4,914	46.2%	–	–
GG	9,761	3,478	64.4%	–	–

Table 6.13: Results from grammar compression and spring cleaning

hence, it is likely due to a bug in the TDL processing code. This is the same code as used in the Grammar Matrix customisation system and it is rather specific about the characters it accepts in type definitions.[34] The TDL processing code forms a complex module with many interactions and solving this problem would result in a complete reimplementation of the module. Therefore, an alternative approach was taken for the remaining grammars.

As explained above, the active types identified by the grammar compression algorithm are a subset of the types that the spring cleaning algorithm maintains in the grammar. The output of the grammar compression algorithm was taken as the input of the spring cleaning algorithm. This approach did not only avoid problems in parsing the instances in Jacy and SRG, but also runs much faster than the original spring cleaning algorithm. It was therefore also used for LXGram, KRG and MCG. The spring cleaning grammar runs into similar problems as found for the instances in the SRG and Jacy when parsing the type hierarchies of the ERG and GG. These grammars can therefore not be spring cleaned until the TDL processing module is revised.

[34]To illustrate: while attempting to spring clean GG, the algorithm choked on an apostrophe in a type name.

6.5.3 Validating results

As explained in Section 3.3.1, there are two ways in which a type can contribute to the grammar. First, a type may contribute to the definition of an instance. This is the case if the type is instantiated itself (i.e. it is a supertype of an instance), introduces an attribute of an instantiated type or defines a value of an instantiated type. If a type that contributes to the definition of an instance is removed from the grammar, compilation either fails or succeeds with a notification that the related instance could not be created.

The second way a type can contribute to a grammar is when it enables the unification of instances. These types must also be present in compressed grammars, which have been proven to be equivalent to the original grammar (recall that Zhang provides a proof of equivalence in Fokkens et al. (2011)). This means that a spring cleaned grammar must be equivalent to the original grammar if it (1) can compile without notifications that were not present before spring cleaning was applied and (2) includes all types marked as active by the grammar compression algorithm. This was verified for all spring cleaned grammars except for Norsyg (+nlk), which could not be compressed.

None of the grammars produced new compilation errors. A verification algorithm was run on the first group in Table 6.13 to make sure no active types were removed.[35] The only exception that was found was the type *alts* in ManGO. This type cannot be needed for unification, because it only has one supertype. The type itself does not occur as a value and the attribute it introduces (PASSIVE) is not used in the grammar. The reason that it is marked as active by the grammar compression algorithm is probably because this algorithm marks the most specific type as active rather than the more general type. Spring cleaning, on the other hand, removes types from the bottom of the hierarchy only. The fact that it removed this one active type is thus not an error in the algorithm.

[35]Recall that the spring cleaning algorithm started of maintaining all active types for the second group. The second condition thus necessarily holds for these grammars.

The verification outlined above ensures that spring cleaned grammars exhibit equivalent behaviour to the original grammar. It does not verify if all redundant types were removed from the grammar. In the second group of grammars, there may for instance be cases similar to the ManGO example discussed above. Spring cleaning always includes all supertypes of a type it maintains. If the more specific type of a chain was marked as active by the grammar compression grammar, this type will not be removed if the active types are taken as input to the spring cleaning algorithm.

As explained above, we consider grammars to be equivalent if they have the same competence, i.e. they produce the same set of analyses for any possible input string. In Fokkens et al. (2011), the equivalent competence of grammars was verified by parsing associated test suites with both the original and reduced grammars. This experiment had the surprising outcome that some of the MRSs produced by the Wambaya grammar changed after the grammars were spring cleaned. This happened despite the fact that the grammars for Wambaya had passed both of the verification tests described above. Upon examining the differences, it turned out that the experiment did not contradict the claims made above, but rather showed that spring cleaning may also be used to identify errors in the grammar. The differences in MRSs turned out to be caused by missing types in the hierarchy of the semantic values for person and number. Some semantic feature values of the feature PERNUM (person and number) had glb-types (greatest lower bound types) as their value. These types are automatically introduced by the LKB when compiling the grammar to make sure the type hierarchy is bounded complete partially ordered (BCPO). They may play a role as a syntactic component of a derivation, but they should not appear in semantic representations. The LKB uses numbers to assign each added glb-type a unique name. These numbers, and thus the name given to a specific type, may differ if the size of the grammar changes. Because of different number of the glb-types representing person and number values in Wambaya, we were able to identify the missing types and improve the Wambaya grammars.

6.5.4 Observations

The spring cleaning algorithm removed a noticeable set of types for each grammar (the smallest number of removed types was 145 for SRG, the highest number 1,320 for NorSyg (small)). There are several reasons why such types may be found in the grammar. They may be the remains of an abandoned analysis or part of a multilingual core that represents a phenomenon that either is not covered yet by the grammar or simply does not occur. The grammar writer may be unaware of the fact that the grammar contains these types. Some types that do not have an impact on the grammar are part of an analysis that is still under development and they are likely to be used in the near future. Finally, grammars often include definitions of lexical types that occur in the language, but there are no instances of these types included in the current lexicon. In these cases, the grammar engineer is likely to be aware of their presence in the grammar and probably intends to maintain them.

Multilingual types will be addressed in Section 6.6 as part of an overview of how the Grammar Matrix is used. Only the grammar engineer can distinguish between abandoned analyses and analyses under development. Investigating the kind of types that were removed may nevertheless give some insight into whether the type is part of an analysis that is not used or whether it defines a type that is part of an active analysis, but no example of an instance happens to be included. The scenario of a correct analysis but no example of an instance is far more likely to occur with lexical items than with syntactic or lexical rules. The difference between NorSyg (small) and NorSyg (+nlk) can be explained by lexical items. The difference does not come from the size of the grammar itself (which is the same), but in the size of the lexicon. NorSyg (small) has a small lexicon that was defined by hand as part of the grammar development process. NorSyg (+nlk) includes a large lexicon derived from a treebank. The 1,143 types that are removed from the NorSyg (small) and not from NorSyg (+nlk) are all needed to define lexical items included in the nlk lexicon.

Grammar	lex. types	rule types	generic types	values	other
wmb (arg-comp)	9	3	120	45	1
wmb (aux+rule)	9	2	129	46	1
BURGER	197	1	120	49	1
ManGO	6	32	130	42	1
KRG	29	20	129	91	1
MCG	9	0	142	283	1
Jacy	383	19	99	51	58
SRG	32	39	16	57	1
LXGram	15	6	14	70	96

Table 6.14: overview of kind of types that are removed from grammars

Table 6.14 and 6.15 provide additional information on the kind of types that were removed from some of the grammars during spring cleaning. The numbers in Table 6.14 are the result of a manual count. These results provide a general indication of the kind of types that were removed during spring cleaning. The grammars in the top part of the table use the Grammar Matrix core. The SRG and LXGram also started out with the Grammar Matrix, but either moved Matrix types to other parts of the grammar and stopped including the original Matrix (SRG) or redefined a new version of the Grammar Matrix to support different theoretical assumptions (LXGram). The classes distinguished in Table 6.14 are the following: *lex. types* are types that define specific lexical items, *rule types* define language specific rules, *generic types*

Grammar	Multilingual	Language specific	Total
wmb (arg-comp)	162	16	178
wmb (aux+verb)	163	24	187
BURGER	140	228	368
ManGO	149	62	211
Jacy	90	521	611
KRG	147	113	260
MCG	174	261	435

Table 6.15: Comparing portion removed from multilingual versus language specific

define types that are generally found on top of the type hierarchy: they define basic rules and lexical types (e.g. a lexical rule that shares all values with its daughter except VAL, a zero-arg lexical item, a coordination structure). They are typically defined in the Matrix core. Finally, *values* refers to any type that defines an atomic feature value and *other* includes any type that does not fall into any of these categories, such as the nodes to label the syntax trees.

The grammars that make use of the Matrix core all reveal a similar number of types that were removed. The bulk of these removed types are defined in the Matrix core. The number of generic types removed for Jacy is lower, because the version of the Matrix core included in Jacy does not contain basic structures for coordination. Table 6.15 indicates how many removed types were included in multilingual components of the grammars and how many were defined in language specific files. Section 6.6 will provide more information on the interaction of individual grammars with the Matrix core.

The relatively high number of values removed in the MCG can be explained by the fact that this grammar does not use all syntactic head categories defined in the Matrix core. HEAD values are defined by an elaborate hierarchy representing types for all possible disjunctions of nine fundamental head types. A large chunk of this hierarchy could be removed in the MCG: 247 of the removed values in MCG were part of the hierarchy for defining HEAD values. Other grammars with a relatively high number of removed values tend to have elaborate hierarchies for expressing semantic relations. This was specifically noticeable in the KRG were 64 Korean specific relation types were removed, but it was also observed in other grammars such as Jacy, the SRG and LXGram. A possible explanation for this may be that grammar engineers tend to have a clear theoretical idea of the kind of semantic relations they want to model for their language which prompts them to include a relatively complete hierarchy earlier on in the grammar development process, before lexical items that exhibit these relations are added to the lexicon of the grammar.

The number of removed language specific lexical and rule types is relatively low for all grammars. Two grammars had a large set of types removed that did not fall in any of the categories. In Jacy, a set of types for transfer rules were removed. Most of the types removed for LXGram were included in a file called *tree-decoration.tdl* which defines the names of the nodes in syntax trees. If we take into account that (1) many of the generic types and values that are removed are inherited from a language independent source, (2) not all removed types are part of the actual grammar and (3) removed lexical types may be caused by a smaller lexicon included in the version of the grammar that was spring cleaned, it seems that DELPH-IN grammars do not contain many forgotten "left overs" from old analyses. As mentioned above, it is not possible to say with certainty by anyone but the grammar writer which types are intended for future analyses and which are forgotten. However, based on my observations as part of this evaluation, I would roughly estimate the number of types that are part of an old analyses around 10 - 50 for the grammars in Table 6.14. This is only a tiny portion of the grammars (from less than 1% to maybe 2%). Overall, these results seem to indicate that DELPH-IN grammars are generally well maintained.

It should be noted that the spring cleaning algorithm only identifies unused branches of the type hierarchy. It is not guaranteed that all unnecessary types are identified. As mentioned above, the spring cleaning algorithm maintains the original structure of the grammar in order to ensure that the cleaned grammars are at least as maintainable as the input grammars. There are cases, however, where this structure is more complex than necessary as a result of an old analysis. The remaining values for head types in the MCG are an example of such a case. A little over half of the types of this hierarchy are maintained by the spring cleaning algorithm, but as will be explained in more detail in Section 6.6.2, several types in this hierarchy no longer fulfil their intended role now that the rest of the hierarchy is removed. Furthermore, the spring cleaning algorithm only identifies uninfluential types. Grammars can also contain attributes that do not influence the behaviour of the grammar. If an attribute does not provide a contribution to the semantics of an expression and can never lead to a failure in unification, it does not have an impact on the grammar's competence. It is feasible to identify straightforward examples of such attributes and the spring cleaning algorithm may be extended in future work to remove them from the grammar. Hierarchies of supertypes that are unnecessarily complex are more difficult to identify automatically. It may be possible to help the grammar engineer to identify such cases by comparing the compressed hierarchy to the spring cleaned hierarchy and pointing out large discrepancies between the two.

6.6 The Matrix core

Section 2.3 provided a detailed description of the LinGO Grammar Matrix. As explained, the LinGO Grammar Matrix provides a starter kit for developing new DELPH-IN grammars. One of the main components of the starter kit is the file *matrix.tdl*. This file defines the top of the type hierarchy for new grammars. It includes type definitions for the feature geometry of (difference) lists, signs including lexical items and unary, binary and ternary rules and the main properties of signs. The basics needed to construct semantic principles according to the principles of MRS are provided. Furthermore, general rules for combining subjects, complements or modifiers with a head as well as structures for coordination are defined. Throughout this section, the term *Matrix core* will be used to refer to the definitions in *matrix.tdl*.

During the development of gCLIMB, several revisions were made to the Matrix core. The changes made during this project formed the biggest revision of the Matrix core since it first release.[36] This is largely due to the fact that almost no changes were made based on grammars created prior to the development of gCLIMB. It is unlikely that grammar engineers working on other languages have not encountered things that needed to be changed for their grammar. Some changes made for gCLIMB were plain errors or revisions to properties that were still English-specific. I therefore investigated how the Matrix core

[36]The coordination structures that were added by Drellishak and Bender (2005) consist of more lines of code than the revisions made for gCLIMB. However, this change forms an extension rather than a revision of the Matrix core. The same applies to more recent additions for information structure described in Song (2014).

is used in other Matrix based grammars. This study looks at two aspects. The first aspect is concerned with the actual changes that are made to types defined in the Grammar Matrix. Secondly, an analysis of the types from the Matrix core that are actually used in current grammars is given. This analysis is based on the output of the spring cleaning algorithm. Section 6.6.1 describes the changes made to the Matrix core in gCLIMB. The outcome of the investigation on other languages is presented in 6.6.2.

6.6.1 Changes inspired by gCLIMB

The revisions made to the Grammar Matrix core as part of this thesis can be divided in two categories. As more advanced phenomena were added to gCLIMB, it turned out that some types in the Matrix core did not behave as intended, regardless of the properties of German. I consider these revisions to be simple error corrections, i.e. they correct bugs in the Grammar Matrix. Other revisions had to be made, because the definitions in the core conflicted with language specific properties of German. These revisions form improvements to the multilingual applicability of the Matrix core. In total, the Matrix core was updated 20 times, resulting in 55 lines of code changed, 143 lines added and 35 removed.[37]

Table 6.16 provides an overview of the error corrections that were made in the Matrix core. In addition to the changes listed in Table 6.16, a constraint was added to the type *coord-phrase* which defines the basic structure for coordination. The value of SLASH is shared between the phrase itself and both coordinated elements. This feature is used for treating long distance dependencies. The additional constraint is based on a hypothesis that it is not possible to coordinate phrases if a particular element is extracted from one and not the other. It furthermore supposes that if an element is extracted from two coordinated examples, this is the same element and must be retrieved elsewhere in the sentence. Finally, the constraint implies that if nothing is

[37]Counts are based on manual inspection after running the unix command `diff`. Lines that were changed in order to add or remove another line where not considered to be "changed".

type name	**explanation**
basic-head-filler-phrase	supertype *headed-phrase* replaced by the more specific *head-compositional.* The semantics breaks if C-CONT is not shared with CONT of the head-daughter.
basic-extracted-subject	added *head-compositional* as a supertype, again, to ensure the semantics is well-formed.
basic-mod-adj-lex	supertype *raise-index-mod-lex-item* replaced by *norm-sem-lex-item*: adjectives have an index of type *event* and nouns have the incompatible type *index.*
basic-mod-adp-lex	supertype *raise-index-mod-lex-item* replaced by *norm-sem-lex-item*: if adpositions share their index with the element they modify, they cannot be the locative complement of copula.
head-mod-phrase-simple	removed this type from the Grammar Matrix, transferring its supertype *basic-head-mod-phrase-simple* to its subtypes. The constraint this type added was already defined for its supertype. (Based on note by Bart Cramer)
basic-adposition-lex	made the INDEX of the adposition coindexed with its second argument: providing semantics for adpositions.
s-coord-phrase	added the constraint that the COMPS list must be empty on daughters and mother.
basic-bare-np-phrase	added the supertype *head-valence-phrase*: else underspecified SLASH values occur in the structure leading to massive overgeneration and spurious structures.
zero-arg-nonslash, *zero-arg-nonque*, *norm-zero-arg*	general supertype *lex-item* replaced by the type *basic-zero-arg*, so that they have indeed zero arguments
infl-bottom-coord-rule, *infl-left-coord-rule*	replaced four of their supertypes by the common subtype of these supertypes. (Based on note by Emily M. Bender)

Table 6.16: Overview of corrections made to the Matrix core

extracted from the individual elements, this also holds for the coordinated structure as a whole. Ross (1967) first described this phenomenon for English calling the related phenomena the Coordinate Structure Constraint (it is not possible to extract elements from coordinate structures) and 'Across-the-Board' exceptions (unless the same element is extracted from both conjuncts). The corresponding HPSG analysis is based on Gazdar (1987). It is likely that these constraints apply to most languages (or even all languages), but this is still an open question. The related constraints were probably not added to the Matrix core, because long distance dependencies were not taken into consideration as coordination structures were being added to the Grammar Matrix.

The second category involves changes that remove constraints from the Matrix core because they are not crosslinguistically applicable or add new phrases because some crosslinguistic variations are not covered. Most changes falling in this category are related to word order. First, one of the analyses used for capturing verb second word order uses the MC feature to ensure the verb is in second place. The MC stands for "main clause" and takes *luk*, a three-fold variant of *boolean*, as its value. The analysis is based on Bender's (2008a) Wambaya grammar. Even though it is not the standard HPSG analysis for German word order, the MC feature is a natural choice for capturing this phenomenon, because German word order is clause final in subordinate clauses. The *basic-head-comp-phrase* and *head-mod-phrase-simple* no longer share the value of MC between head daughter and mother to support this analysis.

Second, I added a number of phrases for analysing *wh*-questions to the Matrix core. *Wh*-questions are questions formed by using a question word. It is named after the first two letters of most question words in English (*who*, *what*, *when*, *whether*, *which*). Question words are often involved in long distance dependencies. English question words, for instance, tend to stand at the beginning of the sentence, even if the word is embedded in the structure. Compare the following example with question word *what* and its counterpart where the object of *saw* is *him* instead of *who.*

(46) What did Kim say Sandy thought John saw yesterday?

(47) Kim said Sandy thought John saw him yesterday.

The Matrix core offered the NON-LOCAL feature QUE (Ginzburg and Sag, 2000). This feature is similar to the feature SLASH used for extraction as explained in Section 4.3.1. Apart from the feature itself, the Grammar Matrix did not contain phrases for *wh*-questions. The type hierarchy for phrases in the Matrix core was revised in gCLIMB. For all phrases combining a head with a specific element (*head-subj*, *head-comp*, *head-spr*, *head-mod*), a generalised type was introduced. These general types have two subtypes: one type that corresponds to the original phrase and one that allows the head to combine with a question word. With this addition, the Matrix core provides basic support for *wh-questions.* Furthermore, the hierarchy was extended so that different combinations of NON-LOCAL features could be shared. Previously, the *head-valence-phrase*, which shares the SLASH value between a phrase and its head daughter inherited from a type sharing the other two NON-LOCAL features (QUE and REL). The supertype was replaced by the more general type *headed-phrase* and a new subtype *head-valence-head-nexus* was introduced that shares all three features.[38]

The Grammar Matrix developers have not looked into long distance dependencies prior to gCLIMB. It is thus not surprising that several revisions were required to make sure long distance dependencies work properly for German. The adaptations mentioned in Table 6.16 already included corrections of some minor errors in the original analyses. With these corrections, instances inheriting from extraction phrases and head-filler-phrases can analyse long distance dependencies resulting in well-formed semantics. More revisions were needed to make sure the analyses can be used in grammars for Germanic languages. This particularly holds for grammars using the filler-gap analysis for verb second word order, which treats every element in sentence initial position as a long distance dependency. The main changes made in the Matrix core to improve its provisions for long distance dependencies are the following:

[38]The name of this phrase is based on the names of its immediate supertypes *head-valence-phrase* and *head-nexus-phrase.*

1. The type defining the *gap* cannot have its INDEX restricted to *event-or-ref-index*. The filler-gap analysis uses extraction to place expletives in sentence initial position and expletives' INDEX is not compatible with this type. This constraint is probably based on English.

2. The restriction forbidding adjuncts to be extracted from subjects was removed.

3. Several phrases and lexical items left their values of SLASH underspecified. Constraints where added to make sure that the value of SLASH for each lexical item is the concatenation of SLASH values of its arguments, that all phrases pass up the SLASH values of their head-daughter and that the value of a head-modifier phrase's SLASH is the concatenation of the SLASH of its daughters.

4. The restrictions placed on VAL in the *basic-filler-phrase* were removed. These constraints (requiring saturated valencies) leads to problems for the filler-gap analysis when dealing with extraposed elements.

5. The restriction [MC *na*] was removed from the subject extraction phrase. This does not interact well with the analyses for word order that use MC, nor with MC as it registers whether we are dealing with a main clause or subordinate.

6. The restriction on adjunct extraction phrases limiting extraction to intersective modifiers was removed. This restriction does not work well with the filler-gap analysis, since adjunctive clauses with scopal reading may be found in initial position in German.[39]

The Matrix core did not provide ready-to-use analyses for relative clauses. The analyses for relatives in gCLIMB make use of *marker-phrases*. A *marker-phrase* in the Matrix core is a phrase that behaves in the exact same manner as a *headed-phrase* except that it does not share the head values of mother

[39]In the examples I encountered, these clauses had scope over the entire main clause regardless of their position. No elaborate investigation was carried out to examine the full range of this phenomenon.

and daughter. The *basic-marker-comp-phrase* was introduced in the Matrix core to support an analysis for disharmonic word order between auxiliary and verbal complement and verb and object. The phrase was included in the Matrix core, because it defines a generic type with properties that may be useful for other languages.[40] It has not been examined how many languages exhibit phenomena that would make use of this type. An explanation of the motivation and working of this analysis can be found in Fokkens (2010) and Fokkens et al. (2012b).

A phrase where the head value of the mother is not identical with the main daughter is useful for relative clauses, because as soon as a verb combines with a relative pronoun, the clause becomes a modifier. The verb remains the head of the clause, but its MOD value must now be non-empty. The feature MOD is part of the verb's HEAD and a regular headed phrase shares its value between mother and head-daughter. If we use *marker-phrases* as basic structures for analysing relative clauses, we can have relative pronouns combine with their head and adapt its MOD feature. For this reason, the types *basic-marker-subj-phrase* and *basic-marker-mod-phrase* were introduced in the Matrix core. Their inclusion in the core was based on the previous inclusion of the *basic-marker-comp-phrase*. In fact, the description of relative clauses provided above is not Germanic specific. This observation thus provides evidence that these phrases may be crosslinguistically applicable. Further research would be needed however to determine whether they are indeed suitable to serve as basic types for relative clauses.

Correct coverage of relative clauses required a few more minor changes to the Matrix core. Similarly to the other non-local phenomena (using SLASH and WH), constraints were added to several types making sure the values of REL were passed on appropriately to the mother phrases.

Two further additions were made to the Matrix core. First, the Matrix core defines several relation types. It distinguishes *event-relations* (with INDEX *event* for verbs, adjectives and adverbs) and *noun-relations* (with INDEX *ref-ind* for nouns). The type *named-relation* is a subtype of *noun-relation* which

[40]This decision was made during a discussion with Bender.

is meant to capture the semantics of names. The predicate of this relation is "named-rel" and it has an additional feature CARG taking a string as its value. The lemma of the name can be indicated as the value of CARG. Similar semantic structures are used for numbers. I added a new supertype for *named-relation* to the Matrix core. This type called *carg-relation* introduces the feature CARG. Furthermore, I included a new type for expressing the semantics of numbers that inherits from this type. The second addition concerns a lexical type that takes four arguments. This type is introduced in order to create types for ditransitive verbs with particles compatible with the TiGer lexicon described in Section 5.3.2.

The final revision of the Matrix core was made based on an observation on Northern Frisian. As explained in Section 6.2.5, the phrase removing optional subjects from the SUBJ list assumes that the verb has picked up all its other complements. This assumption is incompatible with languages that exhibit VSO or OSV order. If the subject stands in between verb and objects, it has to be combined with the verb before elements on the COMPS list are combined. Under the previous analysis in the Matrix core, the verb could not combine with its complements, because it has to combine with its subject first and it could not drop its subject, because it had to combine with its complements first. As a result, structures with dropped subjects could not be analysed (or created) for language with VSO or OSV order. This problem was first observed while creating a grammar for Northern Frisian, which exhibits VSO order for yes-no questions and allows for second person singular subjects to be dropped.

Summary

This section described several revisions of the Matrix core that were carried out during the development of gCLIMB. Some revisions were plain error corrections and others were made to improve the Matrix core's crosslinguistic applicability. The revisions described in this summary form the most elaborate revisions of the Matrix core since its first release. This can partially be explained by the fact that the Matrix developers mostly work with the

Grammar Matrix in relation to the customisation system. Phenomena that are supported by this system tend to be examined carefully by Matrix developers. Other phenomena such as modification, long distance dependencies, relative clauses and wh-questions often consist of analyses adapted from the ERG taking some variations known from Jacy and a small Spanish grammar into account. The types related to phenomena such as modification, long distance dependencies, relative clauses and wh-questions have not been tested in a wide range of languages by the Matrix developers. It is therefore not surprising that most of the revisions described in this section are related to these phenomena. However, even though they have not been tested by Matrix developers, several of these phenomena are included in grammars that use the Matrix core. The next section addresses the question of how the Matrix core is used in different grammars.

6.6.2 The Matrix core in different grammars

The previous section described a number of changes that were made to the Matrix core in order to make it work for grammars created with gCLIMB. These changes consisted partially of simple corrections of errors, partially extensions to the Matrix core and partially of removing constraints or assumptions that do not apply to German. This section examines how other grammars interact with the Matrix core. We hereby distinguish two aspects. First, I examine to what extent types from the Matrix core are actually used. This analysis is based on the output of the spring cleaning algorithm presented in Section 6.5. Second, I investigate changes made to the original Matrix core.

Used types

There may be several reasons why a type defined in the Matrix core is not used by a specific grammar. In order to get a better idea of how redundant types in the core should be interpreted, a manual study was carried out. Tables 6.17 and 6.18 present the outcome of this study. For all removed types, I investigated whether it was related to a phenomenon that a) is not included in the grammar ($\not\ni$), b) does not occur in the language (n.a.) or c) is analysed differently in the grammar ($\neq$). Furthermore, the Matrix core contains one example type for appending elements to difference lists (ex), a set of generic types that are not related to a specific analysis (gen) or types that are used for defining labels or increasing efficiency (other). Naturally, most types in the Matrix core can be called "generic". The types that are called *generic types* in this classification are types that are not linked to a phenomenon, a specific HPSG phrase, rule or lexical items or building up semantics. For instance, the hierarchy of lexical rules where all but a specific part of *sign* is shared are considered generic types. These rules can be of practical use for various definitions and it is often unclear whether they may be useful for some future phenomenon ($\not\ni$) or will not be useful in this grammar at any time (n.a.).

It should be noted that the distinction between types related to phenomena that are not included yet and those that are not applicable cannot always be made with certainty without profound knowledge of the language. For instance, *head-initial* types are not used in the KRG or Jacy. Korean and Japanese are both head-final languages, but head-initial rules may still be needed for certain structures (Siegel and Bender, 2004).[41] A joint study with the grammar engineers would lead to a more accurate classification of unused Matrix types. This lies out of the scope of the present work. The only exception is the analysis of the Wambaya grammars which was verified against a manual classification carried out by Bender as part of the spring clean evaluation presented in Fokkens et al. (2011). However, it is still difficult to classify types for this language correctly. The full description of Wambaya provided in Nordlinger (1998) was used to create the grammars for Wambaya and even though the language is not officially considered to be extinct,[42] it is difficult to obtain information about further syntactic phenomena that occur

[41] Jacy actually includes some head-initial rules. It is unclear why they do not inherit from the head-initial types in the Matrix core.

[42] According to the Ethnologue, there were 89 speakers in 2006 (`https://www.ethnologue.com/language/wmb/`, accessed 12 April 2014).

Removed from	$\not\ni$	n.a.	ex	$\neq$	gen.	other	various	Total
all	14	-	1	6	1	3	-	25
all but one	8	18	-	3	5	1	8	43
all but two	17	5	-	3	22	1	4	52
three grammars	10	7	-	1	15	-	9	42
two grammars	2	3	-	5	13	-	10	33
Wambaya (only)	2	12	-	3	-	-	-	17
BURGER (only)	-	-	-	1	1	-	-	2
ManGO (only)	1	-	-	-	-	-	-	1
MCG (only)	3	6	-	5	22	-	-	36
Jacy (only)	-	-	-	9	24	-	-	33
KRG (only)	-	-	-	6	2	-	-	8
Total	57	52	1	41	105	5	31	292

Table 6.17: Classification of redundant types from Matrix core in groups of grammars

in the language.

Table 6.17 provides a classification of removed types across grammars. It shows how many types in the core are not used in an individual grammar for a specific reason and in how many cases this applies to more grammars. This information can help to generalise over removed types and hence provide better insight into the suitability of the types in the Grammar Matrix. For instance, the Matrix core includes six types that are not used in any of the grammars, because they use an alternative implementation. There are eighteen types that are not applicable to five out of six grammars. It is worthwhile to investigate whether these types lack crosslinguistic applicability or they are excluded from the grammars for another reason. One grammar engineer choosing an other analysis gives less reason for such an investigation than five or six. The class *various* in this table indicates types that are not included or not applicable for some grammars and analysed differently in others.

A full classification of removed types in a particular grammar is given in Table 6.18. The number of removed types in Jacy is significantly lower than that in other grammars. This can be largely explained by the fact that Jacy uses a relatively early version of the Matrix core. This version of the

Removed from	$\not\ni$	n.a.	ex	$\neq$	gen	other	Total
wmb (arg-comp)	54	43	1	25	33	6	162
wmb (aux+verb)	54	44	1	24	34	6	163
BURGER	34	27	1	23	50	5	140
ManGO	47	34	1	17	44	6	149
MCG	60	35	1	24	48	6	174
Jacy	21	3	1	27	33	5	90
KRG	51	30	1	28	31	6	147

Table 6.18: Classification of redundant types from Matrix core per grammar

core has been adapted to stay close to later versions of the Matrix core at several stages. Basic definitions for lexical types, many of the aforementioned generic types and rules for coordination were not added to the Matrix core in Jacy during these upgrades. The basic types for coordination represent a variety of coordination strategies. Types that are not used in individual grammars generally support forms of coordination that do not occur in the grammar. The absence of these rules in Jacy's Matrix core explains the relatively low number of not applicable types in the Matrix core. Jacy also has a relatively low number of cases where the phenomenon is not included in the grammar. This may partially be the case because Jacy has been developed longer and covers more phenomena than the other grammars and partially because several of the types classified as "not included" represent complex lexical types, which are not defined in Jacy's Matrix core.

The following section will elaborate on some notable cases where grammar engineers decided to follow other analyses than those provided by the Matrix core. This section will also address observations based on grammars other than Jacy.

Changes to the Matrix core

There are three methods grammar engineers use to include analyses that differ from those provided by the Matrix core. As presented above, definitions provided by the Matrix core may be ignored and another analysis included in the language specific files. Grammar engineers also use two other techniques to change analyses provided by Matrix core. They either directly change, remove or add definitions in the file *matrix.tdl* or they redefine a type that is already defined in the Matrix core in a language specific file. In the latter case, the definition in the language specific file overwrites the definition in *matrix.tdl*. Some notable changes made by grammar engineers will be outlined below.

Several of the phenomena that led to changes in the Matrix core used in gCLIMB also required changes in other grammars. ManGO and BURGER removed the constraints on the feature QUE on *head-subj-phrase* and *head-comp* phrases so that they can be used for question words in canonical positions. The basic phrase combining head and modifier was changed in the Matrix core of the Wambaya grammars for the same reason. The constraints identifying INDEX of adjectives and adverbs with the element they modify had been removed in several grammars. Finally, the *head-adjunct-phrase* was adapted in BURGER and the Wambaya grammar to remove English specific constraints.

Other changes were also made for more than one grammar. ManGO and BURGER have several adaptations in common. They both use a different definition for determiners. The Matrix core's standard definition introduces one relation scoping over the element the determiner specifies. Possessive pronouns introduce a possessive relation scoping the element they specify. ManGO and BURGER generalised the definition for determiners so that it provides a common supertype for regular determiners and possessives. The gCLIMB libraries introduce a new type in the language specific files for possessive, which could have inherited from this more general type for determiners. ManGO and BURGER furthermore contain an adapted definition for optional complements, so that it can also be used to cancel elements of the complement list of nouns. The Matrix core restricts this rule to heads which have an EVENT index, a constraint that needs to be removed to allow it to apply to nouns.

The KRG flipped constraints on the left daughter and right daughter of coordination structures to change them to left-branching structures rather than right branching structures. The same change was made for Turkish as described in Fokkens et al. (2009). Drellishak and Bender (2005) created types for coordination based on the assumption that coordinated phrases are generally symmetric and that therefore either left-branching or right-branching rules could work for any language. This hypothesis is falsified by head-final languages where coordination rules do reveal asymmetry such as Turkish.

The changes described above all reveal cases where the solution offered by the Matrix core does not apply crosslinguistically. Incorporating them in the Matrix core in such a way that both the old solution and the newly required solution are covered would improve the Grammar Matrix. A number of other changes were made in individual grammars for which it is less straightforward to see how the Matrix core can be adapted to offer these functions for future grammars. I will illustrate this through the example of the new definitions for head-types introduced in the KRG and the MCG.

In the KRG, several new features are defined for the syntactic head, including a part-of-speech feature which takes values such as *verb* and *noun*. The hierarchy for *head* in the MCG still uses part of the original hierarchy provided by the Matrix core, but introduced new definitions for all basic head types except *comp*. The Grammar Matrix provides an elaborate hierarchy for head types including definitions for any possible combination of syntactic heads. Because the MCG still uses the original definition for *comp*, a little more than half of this hierarchy is maintained (254 types out of a total of 501). However, this complex hierarchy of supertypes for *comp* cannot fulfil their intended role, because all other head-types (except for two new subtypes of *comp*) are defined completely separately of the original hierarchy. The type *+vc* intended to unify with head value *verb* and *comp*, for instance, no longer unifies with the newly defined type *verb*. It would therefore have been better if all head types had been redefined. I will elaborate on MCG's hierarchy for head values below.

It is not straightforward to incorporate the analyses used in the MCG and

KRG in the Matrix core, because there is no consensus about a universal set of part-of-speech. Furthermore, classifications of which part-of-speech may need to be combined (because certain features or structures apply to several syntactic heads) may differ significantly from one language to another. The Matrix core currently includes a set of types that are likely to be useful for many languages together with a hierarchy that can capture any possible combination of these types. The revision included in the MCG forms a more elegant solution using theoretically founded distinctions such as *content* heads and *functional* heads, adapting the name of certain heads to better reflect their behaviour in Chinese (e.g. *dem* rather than *det*) and adding a handful of Chinese specific types. For instance, the grammar includes a phrase for which the head-daughter must be a content word. The distinction between content heads and functional heads provides an natural way to restrict the phrase in question.

In the examples outlined above, only the additions at the bottom of the hierarchy are strictly necessary to make the analyses in the grammar work as intended. Instead of *content* and *functional* head values, the corresponding disjunctive types provided by the Matrix core could have been used. Elegance, more mnemonic type names and better representation of syntactic theory provide a strong motivation for modifying an analysis and the changes made in the MCG and KRG are thus sensible changes. However, they cannot be ported to the Matrix core. They would change a general analysis which can likely be used for many language to an analysis that is based on theoretical assumptions made for Mandarin Chinese in the MCG and Korean in the KRG.

Developing the Matrix core

The goal of the Grammar Matrix is to share knowledge between grammar engineers so that they are not reinventing the wheel. It achieves this goal by providing basic types providing partial or general analyses for phenomena that occur in many languages. However, the changes described above were mostly applied to the Matrix core of more than one grammar. This means that the wheel is still being reinvented if it comes to identifying errors or language specific constraints in the Matrix core. For some of the changes described above, it is clear that they form a general improvement to the Matrix core. A part from a handful of changes made for the development of Wambaya and those that were also made in gCLIMB, they are not taken up in the current version of the Grammar Matrix. Both the Wambaya grammars and gCLIMB are developed by people that are directly involved with developing the Grammar Matrix. We currently do not have a good mechanism to obtain feedback from other grammar engineers leading to valuable information remaining limited to individual grammars.

As pointed out above, some changes that improve the core for a specific language cannot be ported to the Matrix core because they are likely to reduce the crosslinguistic applicability of the Matrix core. On the other hand, large hierarchies consisting of many generic types that each introduce few properties do not necessarily improve the transparency and maintainability of the resource. Some of the types removed by the spring cleaning algorithm defined properties that were introduced elsewhere in the grammar indicating that the grammar engineer may not have noticed that the Matrix core already offered a relevant type. There can thus be a trade off between modularity and covering more variation and transparency. The closer the Matrix core comes to providing variation where it is needed and compact, simple hierarchies where possible, the more useful it is as a resource. Changing the Grammar Matrix towards this ideal is only possible by looking at a variety of languages and gaining good understanding of the analyses that are used and the motivation behind changes to the present core. This is challenging when looking at the grammars alone, because not all adaptations are well documented. Feedback from individual grammar writers is essential for improving the Matrix core.

The Matrix core currently provides a set of basic types that help grammar engineers develop their grammars and at the same time gives them space to introduce changes where they see fit. A drawback of the way the core is currently used is that feedback from such changes does not come back to developers of the Grammar Matrix in many, or even most, cases. Chapter 7 will describe two approaches for sharing information across grammars that address this problem. In the ParGram project, grammar developers meet every six months to compare their grammars and agree on the features they use. In the CoreGram project, which also uses HSPG as its theoretical framework, the common core is shared by all grammars at all times. Changes made for one grammar can thus not go unnoticed for other grammars. Both of these projects have the advantage that they help to avoid grammar engineers having to solve the exact same problem. Moreover, grammars stay as comparable as they can be, whereas changes in the Matrix core made in individual grammars may make it hard to share otherwise similar analyses directly. On the other hand, both approaches leave less flexibility to individual engineers compared to the Grammar Matrix as it is used now. Furthermore, neither meeting every six months nor always sharing a common core is feasible for developers of Matrix grammars. The DELPH-IN grammar development efforts are too diverse and (partially) not funded well enough to allow for regular meetings. The number of researchers and opinions on individual analyses are too many for the grammars to share the exact same core. I will therefore describe some alternative steps that may be taken to improve the Grammar Matrix core.

This section has provided a basic analysis of changes made to the Matrix. In order to gain deeper insight into the role the Matrix core plays and how it can be improved, more elaborate analyses are needed where developers of the core discuss changes with the authors of the grammars. Even if it is not feasible to have meetings every six months or even to include a session during the annual DELPH-IN summit, it may be worthwhile to try and carry out such an analysis every couple of years. In this case, we could communicate directly with grammar engineers about changes they have made and alternative analyses they propose via e-mail using the Grammar Matrix mailing list. Once the major changes in early development stages are discussed, further changes are likely to be less frequent. It may thus be possible to keep the discussion going once the main changes in the Matrix core the grammar have been examined. Furthermore, it may be useful to reinvestigate which definitions should be included in the static core and which definitions would better be moved to Grammar Matrix customisation libraries. Most languages will not use all attested techniques to create coordinated structures leading to types that will not be used in the Matrix core. At the same time, addenda such as case marking on specific head-types are quite common across languages. Investigating addenda and other analyses across grammars may provide interesting comparison material for research on crosslinguistic applicability. Including such a study has the additional advantage that it may become more interesting for individual grammar engineers to participate in such a meeting. In addition to the interest in their own grammar shown by other grammar writers, they may gain new insights or ideas about how additional phenomena may be addressed by looking at other grammars.

6.7 Summary

In this chapter, I have addressed several multilingual aspects of CLIMB and the Grammar Matrix. Section 6.1 described how the CLIMB method may be used for parallel grammar engineering. This approach is similar to that already offered by the Grammar Matrix customisation system. The main difference is that in such an approach, analyses need not be limited to the basic properties of linguistic phenomena. The resource can include detailed analyses for individual languages, because only a closed group of languages is considered in this approach. If the languages in question are well-documented, the full range of variation for a phenomenon is known and due to their relatedness, it will in most cases be less than the full typological variation found in all languages.

Section 6.2 investigated the possibilities of creating grammars for Germanic languages other than German. Grammars for Dutch, Danish and Northern Frisian were created and evaluated. The earliest versions of gCLIMB contained variations for Dutch and Danish, but these were not maintained as the metagrammar was further developed to cover the Cheetah test suite. Dutch was evaluated on a test suite based on the original suite created for

German. For Danish, the test suite accompanying DanGram Müller and Ørsnes (to appear) was used.

Neither of the experiments led to perfect coverage of the test data, but the Dutch results clearly outperformed those for Danish, with Dutch leading to coverage of 97.1%-100% and overgeneration of 0.2%-1.0% and Danish resulting in 57.6% coverage and 20.7% overgeneration. The difference in results can be explained by the fact that the Dutch test suite largely includes the same phenomena as the test suite used to develop gCLIMB and that German syntax is closer to Dutch syntax than to Danish. It is furthermore possible that properties of Dutch have influenced the implementations in gCLIMB, because it is my native language.

Despite the similarities between Dutch and German, however, a few fundamental revisions would be required to correct the errors in the Dutch grammars currently created with gCLIMB. The fact that Dutch word order was handled correctly in earlier versions of gCLIMB shows how important it can be to make use of CLIMB's possibility of maintaining old analyses. If I had kept the original analysis in parallel with the revised one, I could have "time travelled" back to grammars that could handle word order in Dutch and possibly avoided fundamental revisions in a complex grammar. This observation has led to the conclusion that a multilingual resource can only support multiple languages correctly if the variations found in these languages are taken into consideration at all stages of development. On the other hand, the Dutch grammars handle a large part of the phenomena correctly and even the Danish grammars revealed correct behaviour for several phenomena. It is unlikely that a similar result could have been obtained by using only the Grammar Matrix customisation system and extending the grammars manually. The customisation system covers significantly less phenomena and it implementing all phenomena covered by gCLIMB from scratch would probably take several weeks. Adapting a traditionally developed DELPH-IN grammar that contains these analysis such as GG for Dutch and Danish is a highly complex task and would most likely not lead to a similar result in such a short time.

The results on Northern Frisian, which was not considered at any time during the development of gCLIMB, confirm this outcome. The final result using gCLIMB revealed coverage of 94% and no overgeneration and was obtained in one day of work. The grammar created with help of the Grammar Matrix customisation system during the Knowledge Engineering course covers 70.6% and has 1.8% overgeneration. This result and the outcome of the Dutch and Danish evaluations show that gCLIMB indeed offers an increased level of modularity that makes it easier to share its implementations across languages.

PaGES forms an interesting project to further explore the possibilities for parallel grammar development with CLIMB. Section 6.3 described research questions that can be addressed by developing SlaviCLIMB, a dynamic component supporting the development of a static SlaviCore. The section describes how a dynamic component may help to empirically verify Avgustinova's (2007) theoretical model for cross-Slavic grammar development. SlaviCLIMB can currently generate the SlaviCore and RRG described in Avgustinova and Zhang (2009) and Avgustinova and Zhang (2010). The section furthermore explained how SlaviCLIMB can serve as a platform for Slavicists and grammar engineers to collaborate on creating resources for Slavic grammars in future work.

The final section addressing how the CLIMB method may be used to create alternative grammars explained how CLIMB may be used to create grammars for second language learners.

In addition to applications of CLIMB related to multilinguality, this chapter has addressed how different DELPH-IN grammars relate to the Matrix core. Because this question can only be answered if we do not only look at how the Matrix core is changed, but also at what portion of the Grammar Matrix is used, an explanation of the spring cleaning algorithm and its results were provided in Section 6.5. Section 6.6 presented the outcome of a first study of the kind of types that are removed from the Grammar Matrix and what changes are made. This led to the conclusion that several changes are made independently for different grammars and that some changes clearly lead to an improvement of the Matrix core that goes beyond the benefit of the individual grammar that introduced it. A proposal was made to carry out a follow up study leading to revisions of the Matrix core based on individual analyses.

The next chapter presents related work using metagrammars, sharing knowledge across grammars or improving modularity. This chapter also addresses how other initiatives deal with the challenge of sharing information from different grammars through more extensive communication or constant sharing of a common core.

Chapter 7

Related Work

The challenges in grammar development are well-known and many proposals have been made over the years to improve methodologies, the structure of grammars, modularity, generalisations and the possibility of sharing information across grammars. This chapter discusses some of the most prominent efforts across frameworks in this direction.

The methodology used for code-sharing or making grammars more systematic is generally related to the formalism or linguistic theory used as a foundation of the grammar. I will therefore give a basic introduction to the formalisms and theories of three projects before presenting their methodologies. In particular, I will describe Tree Adjoining Grammar (Joshi and Schabes, 1997, TAG) and its related MetaGrammar project (Candito, 1998) and the eXtensible MetaGrammar (Duchier et al., 2005). This is followed by a description of Ranta's (2004) Grammatical Framework (GF) and his GF Resource Grammar Library (Ranta, 2009). The description of the ParGram (Parallel Grammar) project Butt et al. (2002); King et al. (2005) is preceded by a brief introduction to Lexical Functional Grammars (Kaplan and Bresnan, 1982; Dalrymple, 2001, LFG).

A complete introduction to any of these formalisms or theories would require at least a chapter of its own. I therefore provide basic, simplified descriptions. They serve the purpose of giving the reader sufficient background to follow the motivation behind the related projects, rather than introducing the frameworks themselves.

Three more projects are presented without an introduction to their formalism. PAWS (Black, 2004; Black and Black, 2009) generates PC-PATR (McConnel, 1995) grammars for field linguists. Their approach does not focus on improving grammar engineering and little attention is given to the implementation and interactions with the formalism in the literature on PAWS. The final two projects described in this chapter are both related to HPSG grammars. Sygal and Wintner (2011) propose a modular approach to developing unification based grammars with typed feature structures. In Section 7.5, I will show that, from a technical point of view, their approach is by far the most similar to CLIMB of the work presented in this chapter, despite the difference in focus. The final section will present CoreGram (Müller, 2013) which is similar to both the static core in the Grammar Matrix as well as declarative CLIMB.

Each of the sections below introduces a different project. The description of the project is followed by a comparison between the project and CLIMB. In some cases, differences from the Grammar Matrix are also discussed. The final section focuses on CLIMB. This section provides a brief global comparison between CLIMB and the other approaches.

7.1 MetaGrammar and XMG for TAG

The MetaGrammar (Candito, 1998, MG) and the follow-on eXistensible MetaGrammar (Duchier et al., 2005, XMG) projects are efforts to improve the development of Lexicalised Tree Adjoining Grammars (Joshi and Schabes, 1991, LTAG). An initial foundation for these projects was made by Vijay-Shanker and Schabes (1992). Because the motivation for the MG project is tightly linked to the formalism of TAG, I will provide a formal definition of TAGs and a (somewhat simplified) description. This is followed by a brief explanation of the challenges the work by Vijay-Shanker and Schabes (1992), Candito (1998) and Duchier et al. (2005) addresses. I then briefly explain the basic ideas and setup of Candito's MG and the main extensions found in XMG.

7.1.1 TAG

As defined in Joshi and Schabes (1997), a TAG is a quintuple (Σ,NT,I,A,S), where

- Σ is a finite set of terminal symbols
- NT is a finite set of non-terminal symbols: $\Sigma \cap NT =$
- S is a distinguished non-terminal symbol: $S \in NT$
- I is a finite set of finite trees called *initial trees* or *elementary trees*, where
  - interior nodes have non-terminal symbols as labels
  - nodes at the frontier can be labeled by a terminal or non-terminal symbol. Non-terminal symbols at the frontier are marked for substitution, indicated by the symbol $\downarrow$
- A is a finite set of finite trees called *auxiliary trees*, where
  - interior nodes have non-terminal symbols as labels
  - nodes at the frontier can be labeled by a terminal or non-terminal symbol. Non-terminal symbols at the frontier are marked for substitution, indicated by the symbol $\downarrow$, except for one note the *foot node*, marked *.
  - The labels of the foot node and the root node of an auxiliary tree must be identical

The main operations within TAG are substitution and adjunction. Figure 7.1 provides a schematic illustration of both operations. In the case of substitution, depicted on the left, an initial tree is inserted in another initial tree at

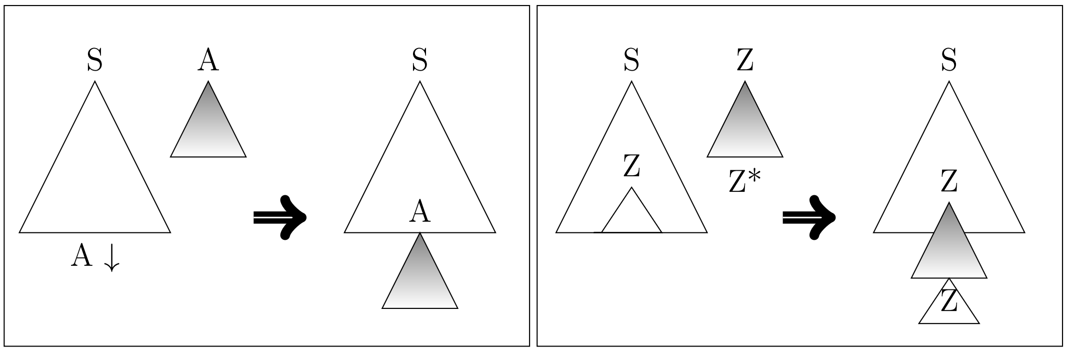


Figure 7.1: Substitution (left) and adjunction (right) in TAG

a frontier node marked for substitution. The root node of the inserted tree must have the same category as the node marked for substitution. Auxiliary trees can be inserted into initial, auxiliary or derived trees using adjunction. Auxiliary trees can only be inserted at nodes that are not marked for substitution and whose category is identical to the root and foot node of the auxiliary tree in question. The original node (together with its subtree) is inserted at the foot node of the auxiliary tree as can be seen in the right hand image of Figure 7.1.

7.1.2 Inheritance in TAG

The MG project and XMG support *lexicalised* tree-adjoining grammars (LTAGs). An LTAG is a TAG in which each initial or auxiliary tree has at least one terminal symbol, its *anchor*.

Figure 7.2 provides an example of substitution in LTAG. This figure demonstrates the rich information that lexicalised trees in TAG contain. A lexicalised tree specifies the subcategorisation requirements of a lexical item as well of as the location where arguments should be inserted in the tree. In the standardly used Feature Structure based TAG (FTAG), where nodes are labeled with feature structures rather than atomic labels,[1] initial and auxiliaries trees also contain morphosyntactic information about the arguments of

[1]Candito (1999) refers to lexicalised TAGs augmented with features as "TAG standards" (standard TAGs).

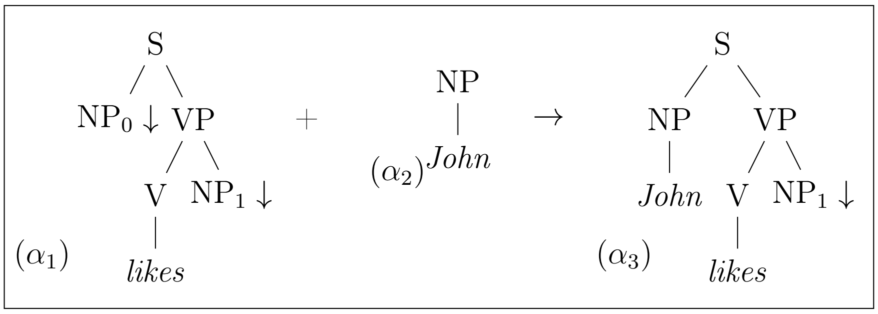


Figure 7.2: Example of substitution in LTAG

the anchor.

Originally, implemented TAGs were organised using two levels. The grammar engineer would define initial and auxiliary trees with uninstantiated anchors. Following Candito (1998), I will refer to such structures by the term *tree sketches*. Related tree sketches are grouped into families in this approach. The lexicon consists of lemmas which can be combined with anchors of tree sketches to form initial and auxiliary trees. The definition of each lemma contains a selection of families of tree sketches the lemma may combine with to form an initial or auxiliary tree.

Engineers working in this original setting faced a major challenge in the lack of generalisations over tree sketches, resulting in large lexica that were difficult to maintain (Becker, 1990; Vijay-Shanker and Schabes, 1992). When part of the grammar was modified, each individual tree sketch interacting with this phenomenon had to be adapted manually. Vijay-Shanker and Schabes (1992) introduce 'structure-sharing' for LTAGs inspired by Flickinger (1987).[2] The basic structure they introduce involves inheritance in a similar way as found in HPSG type hierarchies. Their work can be seen as the basis of Candito's MG.

[2]Vijay-Shanker and Schabes (1992) use the term 'structure-sharing' for combining information from lexical items and elementary trees. What they call 'structure-sharing' is similar to inheritance in HPSG.

7.1.3 The MetaGrammar Project

Candito (1999) addresses the same basic problem of sharing information between tree sketches. The main goal of her MG is to provide a setup that can capture linguistically motivated generalisations in a grammar. Properties of tree sketches are encoded in three distinguished dimensions (each using hierarchies and inheritance) which are combined to form tree structures using code generation. The dimensions contain the following information (Candito, 1996):

dimension 1 *the canonical subcategorisation frame* defines the arguments of predicates and the argument's index, possible category and canonical syntactic function.

dimension 2 *the redistribution of syntactic functions* defines possible diatheses of a lexical item, e.g. argument alternation, passivisation, causatives.

dimension 3 *the syntactic realisation of functions* defines the realisation of syntactic functions on the surface, their constraints on word order and morphology

The generation of a well-formed tree in TAG takes place in two steps. First, a generator creates a cross-classification by combining terminal classes (corresponding to types in HPSG) from each of the three dimensions in a controlled manner. The cross-classification is restricted by language independent *principles of well-formedness* and language specific *compatibility constraints*. The systematically generated classes contain information from all three dimensions.

The second step converts the thus obtained classes into well-formed TAG tree sketches. Classes can contain flat tree structures where dominance or precedence are underspecified. Tree structures with underspecified dominance can be compared to *head-final* or *head-initial* phrases in HPSG, which state the relative order between the head of the phrase and other daughters, but provides no indication of the exact relations between the respective phrases. If precedance is underspecified, the exact location of arguments is not defined. This underspecification makes it possible to disassociate dominance from precedance, a property that can capture flexibility in the final order of constituents on the surface. A class can be converted in one or more tree sketches representing an initial or auxiliary tree. As before, the lexical definitions of lemmas specify with which anchors they may combine.

7.1.4 Extensions to MG

Several extensions have been proposed to Candito's MG. Gaiffe et al. (2002) propose an alternative compiler that leaves the number of dimensions used open. Their experimental proposal is worked out and scaled up by Duchier et al. (2005) and Crabbé and Duchier (2004), referring to the updated version of MG as eXtensible MetaGrammar (XMG). The original motivation for this extension was that the three dimensions proposed by Candito (1999) focused on verbs in French and Italian, but could be suboptimal for other languages or words belonging to other categories than verbs. Gerdes (2002) investigates the possibility of using an adapted metagrammar for TAG to capture German word order, known to go beyond the generative power of basic TAG (Becker et al., 1991). He concludes that a compromise must be made between a "minimal violation of the TAG principles [...], a maximal coverage of the grammar, a maximal usefulness for simple language generation systems, and a maximal simplicity in the metagrammar description" (Gerdes, 2002, p.5-6).

Clément and Kinyon (2003) propose a methodology to develop grammars using the frameworks of TAG and LFG in parallel from a metagrammar. This approach is possible thanks to Candito's original classification, where the first dimension encodes information relevant for lexical items in LFG, the second dimension can be used to derive lexical rules, and the structure of tree sketches can serve as a basis for generating LFG's context free rules.[3] The main purpose of using MG in Clément and Kinyon (2003) is to find a 'middle-way' between hand-crafting grammars and extracting grammars

[3]For a basic introduction to LFG, see Section 7.3.

from a treebank. The basic part of the grammar is hand-crafted and the bulk of the grammar is automatically generated off-line. The engineer need only write a small grammar and verify the derived rules.

Kinyon et al. (2006) explore the use of XMG in a multilingual context. In principle, the hierarchies in the metagrammar can contain crosslinguistically invariable properties. Their study uses XMG to perform a crosslinguistic comparison of verb-second structures, investigating the applicability of their analysis to German and Yiddish.

7.1.5 Comparing MG and XMG for TAG to CLIMB

Metagrammars for TAG were developed to provide a level of linguistically motivated generalisation and thereby improve the maintainability of the grammar. They also contribute to the overall structure of the grammar and have a multilingual ambition (Candito, 1999).

The (X)MG approach clearly has a lot in common with the approach proposed in this thesis. First, they both use libraries (or dimensions) of basic implementations in combination with code generation to develop grammars. Second, they share the goal of improving maintainability, modularity and the systematic structure of the grammar. Finally, both approaches support multilingual grammar development, even though MGs and XMGs for TAG are generally monolingual (Kallmeyer, p.c.).

On the other hand, the differences between the approaches are also significant. The main purpose of MG and XMG is to allow the grammar engineer to organise the grammar in a manner that is generally done by the type hierarchy and multiple inheritance in HPSG grammars. Code generation thus plays different roles in (X)MG and CLIMB. Whereas (X)MG uses code generation to capture generalisations, CLIMB uses code generation to include alternative analyses for the same phenomenon (within a language or crosslinguistically).

Furthermore, there is little overlap between XMG's organisation in dimensions and the organisation in linguistic libraries in the Grammar Matrix or CLIMB. On the one hand, CLIMB represents basic subcategorisation and surface constraints together at several places in the metagrammar. On the other hand, XMG does not, to my knowledge, use the more fine grained architecture found in CLIMB libraries to distinguish between different morphosyntactic properties, such as case assignment or person-number-gender agreement. With CLIMB, linguists can define observations they make on the surface (e.g. ambiguous morphological markings) and automatically generate a grammar containing a corresponding type hierarchy. Section 6.3.2 explained how this property of CLIMB can be used to verify linguistic hypotheses. To my knowledge, (X)MG does not provide such support to grammar writers.

Finally, I am not aware of any work in MG or XMG that investigates maintaining different versions of a grammar. This includes both versions meant for different applications as well as systematic investigations on the impact of specific analyses. The original goal of CLIMB is therefore not explored in this approach. The overall setup would, however, allow engineers to use MG for comparison. This is exactly where the research that is closest related to my research within XMG differs from the work in this thesis. Kinyon et al. (2006) investigate verb-second analyses through a metagrammar. They explore the possibility of using one analysis across two verb-second languages. Alternative analyses are not explored in their approach. This work is therefore more closely related to the Grammar Matrix than to CLIMB.

7.2 The GF Resource Library

The GF Resource Grammar Library (Ranta, 2009, 2011) is a multilingual linguistic resource that contains a set of syntactic analyses implemented in Grammatical Framework (Ranta, 2004, GF). The purpose of the library is to allow engineers working on NLP applications to write simple grammar rules that can call more complex syntactic implementations from the grammar library. The grammar library is written by researchers with linguistic expertise. It makes extensive use of code sharing across languages. Ranta

(2011) reports comprehensive fragments for twenty languages, the GF website[4] indicates implementations for thirty six languages.

The GF Resource Library shares the multilingual ambition of the Grammar Matrix, as well as the desire to use code sharing between languages found in both the Grammar Matrix and CLIMB. However, the architecture of the GF Library and its use differ. I will provide a brief description of the GF formalism and its Resource Library. Descriptions will be limited to what is needed to get a basic picture of how the system works and how it differs from the Grammar Matrix and CLIMB.

7.2.1 Grammar Framework (Ranta, 2004)

The origin of GF lies in Type Theoretical Grammar (Ranta, 1994) using semantics from Intuistionistic Type Theory (Martin-Löf, 1984). GF can be seen as the practical successor of Type Theoretical Grammar. It is designed to provide both a simple tool for engineers who want to build natural language applications as well as a reliable grammar formalism for linguists.

Ranta (2004) situates GF as closest to Montague grammar (Montague, 1974). More distantly, it is also related to unification based formalisms such as DCG (Pereira and Warren, 1980), PATR (Shieber, 1986), HPSG and LFG. There are, however, several differences between GF and unification grammars. I will elaborate on GF's distinction between abstract and concrete syntax and the distinction between parameters and inherent features. The descriptions provided below are mostly based on Ranta (2009) and, to a lesser extent, on Ranta (2004) and Ranta (2011).

Abstract versus Concrete Syntax

The distinction between abstract syntax and concrete syntax is fundamental in GF (Ranta, 2009). Abstract syntax is used to define sets of tree structures. Concrete syntax deals with mapping trees to surface strings and records,

[4]http://www.grammaticalframework.org/lib/doc/status.html, accessed 25 June 2012.

which are similar to feature structures. The following examples taken from (Ranta, 2009, 5-6) illustrate both levels.

The judgments[5] below define properties of **abstract syntax**:

```
cat CN

fun House :  CN

fun Mod :  AP → CN → CN
```

The first judgment introduces the category CN for common nouns. Categories are basic types in abstract syntax and represent a set of trees. The second and third judgment define functions that build trees. `fun House :  CN` is a zero argument function stating that *House* is a tree belonging to the category CN. The function `Mod` recursively builds CN trees by adding adjectives.

Assuming the category AP is defined by a judgment and *Big* is assigned to this category, the following trees can be built:

```
Mod Big House

Mod Big (Mod Big House)

Mod Big (Mod Big (Mod Big House))

...
```

Concrete syntax defines linearisation of both types and functions, as demonstrated below:

```
lincat CN = Str

lin House = ‘‘house’’

lin Mod ap cn = ap ++ cn
```

[5]The word *judgment* in GF refers to what in other frameworks might be called “definitions” or “statements”.

The judgments above assign linearisation to the three abstract judgments provided above. The first judgment assigns the linearisation type *string* to the category of common nouns. This type must be respected by judgments that define the linearisation functions linked to abstract functions. The second judgment links the type `House` to its corresponding string. The linearisation of `Mod` is defined as the concatenation of a tree belonging to AP and a CN tree, with the AP tree preceding the tree of category CN.

Parameters and inherent features

Another distinguishing property of GF is the emphasis it lays on distinguishing between parameters and inherent features. Parameters can be used to represent different forms observed for entities of given category. Typical examples are number marking for nouns, or agreement in number, case and gender for adjectives.

Through a judgment stating that the linearisation of CN assigns strings to number, it is now possible to define `House` as a lexeme, with a singular and plural form rather than a simple word.

On the other hand, a category may have features that are inherent to specific instances of the category. Gender in German or French is a typical example of such an inherent property. Apart from a small set of exceptions, a given noun always carries the same gender. The judgment for CN below illustrates the difference between a parameter and inherent feature:

```
lincat CN = {s :  Number ⇒ Str ; g :  Gender}
```

Separating inherent features from variable properties forms one of the principle tasks in designing a grammar in GF (Ranta, 2009). Ranta (2009) notes that features that are inherent for a category in a given language are often also inherent in other languages that use the same feature. This property forms one of the bases for sharing implementations across grammars in GF.

Auxiliary operations

Being a functional programming language, GF uses functions identified by the keyword `oper` to incorporate generalisations within concrete syntax. The operation below defines a boolean parameter to indicate surface order for strings. The linearisation judgment captures the fact that some French adjectives follow and others precede the noun they modify.

```
oper prefixIf :  Bool → Str → Str → Str =
\p,x,y → case p of {
True ⇒ x ++ y;
False ⇒ y ++ x };

lin Mod ap cn = { s = table {n ⇒
prefixIf ap.isPre (ap !  cn.g !  n) (cn.s !  n)};
g = cn.g }
```

Operation `prefixIf` can be used for all kinds of variation in word order within categories, such as clitics in Romance languages or distinguishing postpositions and prepositions in Finnish (Ranta, 2009).

7.2.2 The Resource Library

GF has been oriented towards application and multilingualism from its early days. The observation that complex linguistic phenomena occur in any application dealing with natural language motivated the development of software libraries to support engineers.

The GF resource library consists of a user API and a core grammar. Both layers contain language independent as well as language specific modules. The following modules are provided by the API for developers of application grammars.

- `SyntaxX` for access to the syntax in the core

- `ParadigmsX` providing inflection paradigms to build a lexicon
- `ExtraX` allowing engineers to extend the core grammar

The categories and functions provided in the `Syntax` module are the same for all languages. Paradigms are language dependent, though an effort is made to share name conventions across languages. Extensions can be either crosslinguistically applicable or language specific.

The core grammar is structured as follows:

- `GrammarX` containing syntactic combination rules
- `LexiconX` a lexicon to test the core independently from applications
- `LangX` combines `GrammarX` with `LexiconX`
- `AllX` combines `LangX` with additions from `ExtraX` and a language specific lexicon

The language independent library provides a core syntax and lexicon. The core syntax has approximately 60 categories and 200 functions. The core lexicon consists of around 100 structural words, 50 words to create numbers and 350 basic words resulting in a total of around 500 lexemes. The crosslinguistic component of the lexicon defines the general meaning and semantic function of words. Language specific definitions are provided capturing the form, inflection and basic word order patterns of the basic words in individual languages. Additionally, there are modules that capture extensions of syntactic properties for individual languages. The core lexicon includes, for instance, an extension for Romance tense and extensions for irregular verbs for several languages.

The examples below illustrate how the sentence *we walk* can be generated in English and Finnish (*me kävelemme*) using the Library Resource.

```
mkCl we_NP (mkV ‘‘walk’’)
mkCl we_NP (mkV ‘‘kävellä’’)
```

Pronouns are included in the core lexicon. The core syntax takes care of properties such as agreement, resulting in well-formed sentences by combining the basic type for the pronoun with a function taking the infinitive form of the verb as an argument. The next subsection describes the multilingual character of the GF Resource Library in more detail.

7.2.3 Multilingualism in GF

GF shares implementations crosslinguistically as much as possible. At the same time, it provides several methods to incorporate linguistic differences where necessary, both at the level of abstract as well as the level of concrete syntax.

Abstract syntax is generally language independent. It represents semantically relevant tree structures abstracting away from the surface up to a point comparable to MRS: functional words, morphology and surface order are not found in the abstract representation. There are, of course, exceptions where utterances equivalent on a higher level use different semantic structures in individual languages. Abstract syntax can, despite its abstraction and (partial) similarity to MRS,[6] not be seen as a purely semantic representation, since it also includes syntactic categories and basic rules for combining syntactic elements. These syntactic components are comparable to some of the types included in the Grammar Matrix core.[7] They provide generic statements about how words are combined with arguments or modifiers with their heads, but no information on how this is expressed exactly on the surface. As explained in Section 2.3, the Grammar Matrix provides generic types that combine a phrase with a subject or a complement as well as types that define the arity of lexical items.

As mentioned above, the core includes basic lexical items that are found across languages (the surface form naturally being defined specifically for each

[6]See section 2.1.5 for an explanation of MRS.
[7]See Section 2.3 for a description of the Grammar Matrix.

language). These basic lexical items can be compared to instances of basic lexical types in HPSG where only the predicate and not the form is defined. In the example of *we walk* and its Finnish equivalent *me kävelemme* for instance, the pronoun and verb will have common properties in both languages. These common properties are included in the language independent core lexicon. Where possible, the same syntactic structure is used to describe as many languages as possible.

GF provides three basic strategies in case either abstract or concrete syntax of a given language differs from other languages. First, the developer can decide to define the given function in language specific parts of the grammar rather than the common core. The second possibility is to define a general functor suitable for most languages, but apply restricted inheritance for those languages that reveal different behaviour. The general behaviour can then be overwritten by language specific properties. The final, and according to Ranta (2009) preferable, method is to use parameters to define the different options found in individual languages. This method can also be applied to incorporate variations within a language. For instance, the level of politeness in imperatives can be defined in a flexible way using parameters.

7.2.4 Comparing GF and CLIMB

Even though the GF resource grammar and CLIMB both use libraries containing syntactic analyses, the approaches differ in many ways. GF Libraries primarily address engineers who are interested in building NLP applications. They can define a domain specific grammar (e.g. by defining a domain specific vocabulary) without dealing with complex syntactic properties of the language. Syntactic properties are implemented by linguistic experts. Linguistic insights are thus taken seriously during the development of GF, but it does not come with a linguistic theory. The approach can provide insight into the potential to share grammatical definitions across languages, but Ranta (2009) carefully claims that the approach 'may' be of typological interest.

The focus of CLIMB mainly differs in two aspects. The libraries serve in first place to facilitate the development process of linguistic analyses. The libraries make it possible to adapt grammars according to domains or applications, but these adaptation still require linguistic expertise. CLIMB does provide an interface to work on the grammar without grammar engineering expertise, but the intended users of this interface (editors of *choices*) are linguists and not application oriented engineers.

The most significant difference concerns the original goal of CLIMB, which can be seen as opposite to that of the GF libraries. Where CLIMB aspires to stimulate developers to explore different implementations to achieve a particular goal, GF aims to use the same implementation for as many structures as possible.

This difference in interest can partially be explained by the application oriented setup of GF and the fact that GF does not focus on developing a linguistic theory: comparison and creating optimal models is not the purpose of GF. On the other hand, optimal solutions are also relevant for purely engineering goals. In particular, the starting point of sharing between languages may lead to suboptimal solutions for individual languages.

GF parameters or restricted inheritance can in principle be used to incorporate alternative analyses. This would, however, render the core grammar more complex. If there were an interest in grammar comparison within GF, it would probably be a more suitable solution to have parallel core grammars. The benefit would be that the API can remain unchanged, but parallel development of the core would increase the work load significantly. In the end, code generation seems to be a key ingredient for a successful approach to grammar comparison. Given the main purpose of GF, this is unlikely to be developed.

7.3 The ParGram Project

The ParGram (Parallel Grammar) project is a multilingual grammar development approach developing LFG grammars using the XLE parser and grammar platform (Maxwell III and Kaplan, 1993; Crouch et al., 2011). The main idea behind the project is to share knowledge and increase consistency across grammars.

7.3.1 Lexical Functional Grammars

Lexical Functional Grammar[8] takes a modular approach to grammatical description using (minimally) two levels of representation: f-structures (functional structures) and c-structures (constituent structures). C-structures represent the phrase structure through dominance and precedence of surface strings. They are licensed by a context free grammar (CFG).

F-structures encode syntactic dependencies of a structure by representing predicate arguments in attribute-value matrices (AVMs). The values of attributes can be f-structures, atomic values or sets. Feature structures in LFG are not typed: the operation or ($\vee$) can be used to indicate underspecification. Figure 7.3 represents an example of a basic f-structure and c-structure in LFG. F-structures are linked to c-structures through a functional projection function ϕ that maps constraints from c-structure nodes to attribute-value pairs in the f-structure. The constraints on c-structure nodes are defined on context-free rules and lexical items. The following rules and (simplified) lexical items that can derive the structures in Figure 7.3:

$$\begin{array}{llcc} \mathrm{S} & \rightarrow & \mathrm{NP} & \mathrm{VP} \\ & & (\uparrow \textsc{subj}) = \downarrow & \uparrow = \downarrow \end{array}$$

$$\begin{array}{llcc} \mathrm{VP} & \rightarrow & \mathrm{V} & (\mathrm{AP}) \\ & & \uparrow = \downarrow & \downarrow \in (\uparrow \textsc{adj}) \end{array}$$

Mary	NP	$(\uparrow \textsc{pred}) = $ '*Mary*'
smiles	VP	$(\uparrow \textsc{pred}) = $ '*smile*< SUBJ >'
pleasantly	AP	$(\uparrow \textsc{pred}) = $ '*pleasant*'

[8]This basic description of LFG is based on Dalrymple (2001) and Butt et al. (1999).

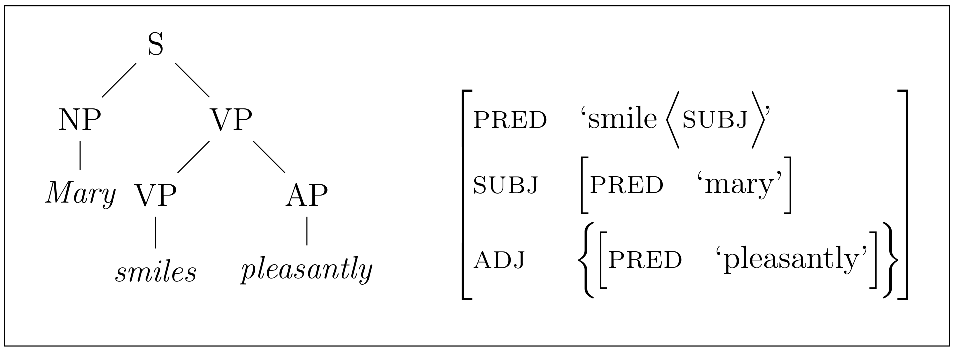


Figure 7.3: Basic c-structure and f-structure of *Mary smiles pleasantly*

The arrows in the annotation indicate how information associated with individual nodes is linked to the f-structure. Phrases inherit their functional properties from their head according to the head convention. This is indicated by the annotation $\uparrow=\downarrow$, where $\uparrow$ refers to the f-structure related to the mother node and $\downarrow$ to the f-structure of the current node.

The annotation $(\uparrow \text{SUBJ}) = \downarrow$ associated with the NP in the first rule above indicates that the functional properties of the current node are identical to those of the subject of the mother node. Similarly, $\downarrow \in (\uparrow \text{ADJ})$ includes the functional properties of the current node in the set of adjuncts of the mother node. Figure 7.4 represents the c- and f-structure together with function ϕ for *Mary laughs pleasantly.*

There are several criteria that determine whether an expression is accepted by an LFG grammar. First, its c-structure has to be licensed by the CFG. Second, function ϕ must derive a well-formed f-structure. An f-structure must fulfil three conditions to be well-formed (Butt et al., 1999, 6):

1. The f-structure must fulfil the condition of uniqueness: each attribute in an f-structure may have at most one value

2. The f-structure must be complete: An f-structure is *locally complete* iff it contains all grammatical functions that its predicate governs. An f-structure is *complete* iff it and all its subsidiary f-structures are complete.

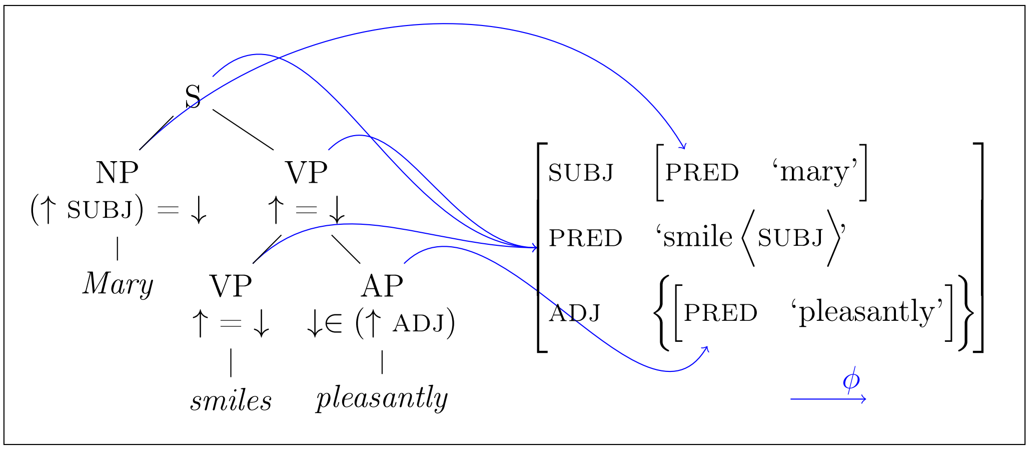


Figure 7.4: Indicating ϕ for analysis *Mary smiles pleasantly*

3. The f-structure must be coherent: An f-structure is *locally coherent* iff all the governable grammatical functions it contains are governed by the predicate. An f-structure is *coherent* iff it and all its subsidiary f-structure are coherent.

The first condition makes sure that an attribute does not receive conflicting values. This can be used to capture grammatical properties such as agreement. Completeness and coherence together make sure that subcategorisation requirements are fulfilled. Completeness dictates that all arguments a predicate subcategorises for be present in the structure, i.e. it excludes expressions like **the lion devoured.* Coherence implies that the f-structure does not contain governable grammatical functions that are not selected by a predicate, excluding ill-formed expressions such as **Mary smiled a table.*

The idea behind different levels of representation in LFG is to represent individual aspects of language in suitable forms. In addition to the c-structure and f-structure, it is for instance possible to add semantic structures using logical forms. The distinction between c-structure and f-structure in syntactic representations is, however, not merely introduced to describe different aspects of language in a convenient manner. It also captures the idea that languages vary on the level of morphology and constituency, but often the same generalisations can be made on a functional level.

Many phenomena across languages can be described with notions such as subject, object, modification, complements or anaphoric binding (Butt et al., 1999, 7). In other words, f-structures provide descriptions that are (largely) language independent. C-structures represent the highly varying surface properties of individual languages. This distinction is reflected in the general setup of ParGram described in the following subsection.

As mentioned above, feature structures in LFG are not typed. XLE (Crouch et al., 2011) allows users to create templates that can be used for generalisations. Templates can group equations associated to lexical items or rules. The basic format of templates and the way they are called is presented below:

```
template name(parameters) = equations.

@(template name parameters)
```

The following example (King et al., 2005, 20) demonstrates how templates may be used to provide a parameter for tense and assign indicative mood. The equation in (1) declares the template. (2) shows how the template is called to assign past tense. The equations in (3) are equivalent to the equation in (2) calling (1).

1. TENSE-IND (TNS) =
   (↑ TNS-ASP TENSE) = _TNS
   (↑ TNS-ASP MOOD) = `indicative.`

2. @(TENSE-IND `past`)

3. (↑ TNS-ASP TENSE) = `past`
   (↑ TNS-ASP MOOD) = `indicative`

Templates can call templates themselves. They can therefore be used to build hierarchies that are used in a similar manner as type hierarchies in HPSG.

The hierarchies built by templates in XLE grammars generally use less layers than HPSG type hierarchies. Hierarchies using more than three layers are rare (Butt, p.c.).

7.3.2 ParGram

The ParGram project (Butt et al., 2002; King et al., 2005) is a collaborative effort to develop LFG grammars for a variety of languages. The related ParSem initiative focuses on deriving semantic representations from syntactic structures in ParGram grammars. I will focus on ParGram in this discussion. The main goals are maintaining consistency among grammars, sharing knowledge between grammars and crosslinguistic research on syntax. Individual grammars are in principle developed independently. ParGram uses two main mechanisms to achieve its goals: tools for shared implementation and discussion during meetings held twice a year.

Grammar engineers writing ParGram grammars try to use the same features and atomic values that are used in other ParGram grammars as much as possible. Both shared implementations and the ParGram meetings are used for this purpose. King et al. (2005) describe a feature checking mechanism that is used as a filter to verify consistency among grammars. The process uses two tools: feature declarations and a feature table.

Feature declarations determine what features may be used in the grammar. Crosslinguistic research is used to decide which features are declared. Feature declarations take the following form:

1. FEAT:$\rightarrow\in$ $\{\texttt{value}_1\ \texttt{value}_2\ \ldots\texttt{value}_n\}$.
2. FEATA:$\rightarrow$<< [FEATB$_1$ . . . FEATB$_n$].

The first definition is used to declare atomic features and their permitted values, the second can declare complex features as value. The example below forms the feature declarations for tense and aspect.

1. TNS-ASP: →<< [MOOD PERF PROG TENSE].
2. MOOD: →∈ { `imperative indicative subjunctive` }.
3. PERF: →∈ {`+_ -_`}.
4. PROG: →∈ {`+_ -_`}.
5. TENSE: →∈ {`fut null past pres` }.

Defining a valid set of features for all languages is not a trivial task (Poulson, 2011). Tense and aspect, for instance, vary across languages. The features above are the result of a practical approach to capturing basic variations in tense and aspect that languages may express, leaving the final interpretation of tense and aspect to a semantic component (King et al., 2005).

When the grammar is loaded, XLE verifies whether all features used in the grammar rules are declared. If the grammar rules contain undeclared features, XLE prints a warning pointing to the feature in question. New features must be added to the declarations in order for the grammar to load.

It is possible to define multiple feature declarations that are called according to a priority order. This allows grammar writers to declare common, language independent features and provide declarations for the specifics of the language in question. More specific declarations are given higher priority, i.e. they can overwrite those from the common core. There are three operations to manipulate more general feature spaces: +, & and !. The operator + adds a value, operator & can restrict the number of values a feature can take and the ! operator can replace the entire set of values.

It is general practice to add all features to the common feature space that are introduced in one of the ParGram grammars for sound linguistic or implementation reasons. This means that the + operation should be used seldomly and, in principle, temporarily. Feature comparison between grammars is one of the main tasks of the ParGram meetings. A feature table is produced based on the declarations of individual grammars. For each new feature, the feature committee consisting of developers of individual grammars carefully checks whether it is required. The committee will always prefer alternative solutions that use the existing common feature space to introducing new features. Only if they are convinced of its absolute necessity, will the new feature be added to the common feature declarations. The comparison of feature spaces in grammars also provides an insight into the set of phenomena covered by individual grammars and the analyses they use (Butt et al., 2002).

Analyses provided for specific sentences are compared during the meeting. A shared set of sentences (a fable or a set of sentences representing a specific phenomenon) is translated and parsed by the participating grammars. For each sentence, differences between grammars are examined and it is discussed whether the difference is justified, or whether analyses could be adapted to increase parallelism. On this level of providing descriptive analyses, templates can be used to share implementations. King et al. (2005) report on 270 templates that are shared among ParGram grammars.

Finally, the meetings serve as a platform to discuss phenomena that do not have a fixed analysis yet. If a general solution is found and accepted by the participants, it is integrated into all grammars. In other cases, a tentative analysis is implemented in one grammar or a couple of grammars and the results are reported in the next meeting. The process of comparing analyses and discussing them also serves the purpose of helping smaller grammars to increase their coverage based on grammars that are more advanced.

7.3.3 Comparing ParGram and CLIMB

King et al. (2005) provide an elaborate discussion about the similarities and differences between feature declarations in ParGram grammars and type hierarchies in HPSG. The main difference they point out is that feature declarations and the checking mechanism are more lenient than the conditions on well-typedness in HPSG.

F-structures are minimalistic in the sense that only those features which are explicitly defined in the lexicon or grammar rules appear in the f-structure. In other words, an f-structure does not necessarily contain all features that are declared appropriate. HPSG features are always present on the types for which they are declared.[9] Furthermore, the authors claim that feature declaration is less theory driven than HPSG type hierarchies. This would have the advantage that the grammar engineer is not limited by theoretical constraints which may be revised, or have not been proven to be crosslinguistically applicable.

This second difference seems only partially valid when looking at current practice in DELPH-IN grammars. The challenge HPSG faces due to lack of modularity in its feature structures is well-known (Moshier, 1997a,b) and has been addressed in Section 3.4.2. However, it is not the case that grammar developers are prevented from integrating new features in their grammar that are not reflected in HPSG theory. Nor does the Grammar Matrix dictate a basic type hierarchy that must under all circumstances be respected. It is common practice to adapt definitions in the Matrix core or overwrite them with new language specific definitions. In fact, grammar writers are more at liberty to do so than members of ParGram: the Grammar Matrix and HPSG theory form a starting point and guideline in their development practice, but new features are not 'skeptically reviewed' by a committee as in ParGram. As the evaluation of the Grammar Matrix has revealed, DELPH-IN grammars could benefit from a similar approach as found in ParGram. Contrary to what has been claimed in King et al. (2005), this approximation to the ParGram method would lead to a stricter approach for writing DELPH-IN grammars.

ParGram shares the goal of achieving consistency in grammars with CLIMB. It differs in its focus on multilinguality and vision of multiple possible analyses. Butt et al. (2002) directly refer to the known problem of multiple ways to analyse a phenomenon. Comparison between grammars at ParGram meetings serve the purpose of making decisions in such cases. The purpose of sharing code is to reduce the possibilities in analysing phenomena rather than exploring alternatives extensively. This does not mean that there is no interest in examining various approaches or the possibility of using a method

[9]Flickinger (1987) proposes a method which avoids superfluous features, where supertypes with less features are introduced that can be used as values in cases where not all features are needed (cf. Section 2.2.1, p.46).

similar to CLIMB. King (p.c.) mentioned the example of adjectives in the English grammar. It was revised and changed back and forth based on a theoretical discussion. The experiment ended in King deciding she would not make the elaborate change again, rather than an agreement on how adjectives should be analysed. King's comment suggests that there is a need for a CLIMB-like approach within ParGram, but, to my knowledge, it has not been attempted so far.

Code sharing within ParGram is static: it consists of a set of feature declarations and predefined templates. Adapting metagrammar engineering would therefore require the effort of designing a suitable metagrammar. In principle, a similar approach as used in the Grammar Matrix customisation system and CLIMB can be adapted for LFG and could be beneficial for ParGram grammars.

7.4 PAWS

The basic setup of PAWS (Black, 2004; Black and Black, 2009, Parser And Writer for Syntax) is relatively close to the Grammar Matrix. The system consists of a questionnaire where linguistic properties can be defined. Consequently, the PAWS Starter Kit automatically creates a draft of grammatical description as well as a basic implementation that can be used by McConnel's PC-PATR syntactic parsing program (McConnel, 1995).

The focus of PAWS differs from the Grammar Matrix in several ways going in quite an opposite direction from CLIMB. I will focus on the differences in purpose and method. For a more detailed description of differences between the Grammar Matrix and PAWS, see Bender et al. (2010).

7.4.1 PAWS setup

PAWS is developed by SIL (Summer Institute of Linguistics) as support for field linguists. The questionnaire provides basic descriptions of known linguistic behaviour that can be observed in different languages. The user answers questions on specific phenomena and is encouraged to provide examples. This input is used as a basis to generate (prose) descriptive grammars. As such, PAWS can be seen as a guide to grammar description.

The descriptions are also linked to implementation in PC-PATR. The questionnaire provides feedback on how to define lexical items for the implemented grammar. Alternatively, AMPLE (Weber et al., 1988) can be used to create the lexicon. The information required for the PC-PATR implementation is clearly distinguished from that intended for grammatical description. This setup allows users who are not interested in an implemented grammar to focus on grammatical description alone. The motivation behind the distinction clearly shows that grammatical description is the primary goal of PAWS.

7.4.2 Comparing PAWS and CLIMB

Black and Black (2009) do not elaborate on the motivation behind implementing a system that can generate PC-PATR grammars. The interest seems to lie mainly in providing a tool for visualising syntactic analyses for descriptive linguists. The grammars are, to my knowledge, not meant to be used in applications nor to provide a theoretical model of language. The latter is not surprising, given that PATR is a formalism for linguistic description and not a framework for developing a linguistic or syntactic theory. Grammatical comparison as carried out in CLIMB has therefore not been carried out within PAWS. Nevertheless, they may still be interested in comparative research. The multilingual nature of their project and their goal of supporting field linguists can lead to regular revisions of analyses. A setup that facilitates comparison can be helpful in such cases.

7.5 Modular typed unification grammars

Sygal and Wintner (2011) present foundations of a modular construction for unification grammars using typed feature structures. This research is part of a project on mathematical and computational infrastructure for grammar engineering.[10] Engineers can implement modules defining a part of a type hierarchy. These modules can be combined to form a complete grammar by specially defined functions. This basic setup is similar to declarative CLIMB. One of the main differences between the two approaches is the fact that improving modularity is the purpose of Sygal and Wintner's (2011) work, where improvement in modularity in CLIMB can be seen as an advantageous by-product of practical adaptations in grammar engineering serving other purposes. A more detailed comparison will be provided in Section 7.5.2

This section will explain the modular design for unification grammars proposed by Sygal and Wintner (2011). Because the formal background of typed feature structures was presented in detail in Section 2.1.1, the description in this section will mostly remain informal. Related definitions and detailed examples explaining how individual operations work can be found in Sygal and Wintner (2011) and Sygal (2011).

7.5.1 Modularity for typed feature structures

Requirements and applications

The following goals are set for the modularised approach to grammar engineering with typed feature structures (Sygal and Wintner, 2011, p. 3-4):

- **Signature focus**, where *signature* refers to the type hierarchy (including appropriateness specification and constraints). This goal expresses that a modular signature is essential for a modular grammar and therefore the modularisation of the grammar should focus on the signature. This can be achieved by distributing definitions making up the type hierarchy over individual modules.
- **Partiality**: Modules can provide partial information about the com-

[10] `http://cl.haifa.ac.il/projects/grameng/index.shtml`

ponents of a grammar.

- **Extensibility**: One module must be deterministically extensible into a full grammar (through combinations with other modules).
- **Consistency**: Inconsistencies between modules must be identified when they are combined.
- **Flexibility**: Unnecessary restrictions on modules should be avoided.
- **(Remote) Reference**: A module must be able to refer to entities defined in another.
- **Parsimony**: A combination of modules should contain all information from individual modules. Additional information may be added, but only if needed to render the resulting combined module well-defined.
- **Associativity**: In case of simple union of data, the operation of combining modules should be associative and commutative.
- **Privacy**: It must be possible to encapsulate information in modules, i.e. make it unavailable to other modules.

The advantages of a more modular approach can be seen in applications such as the development of a single (large) grammar by a team, multilingual grammar development as done in the Grammar Matrix, ParGram or CoreGram and sequences of grammars modelling diachronic change in languages (Wintner et al., 2009).

Signature modules

Sygal and Wintner (2011) introduce **signature modules**, which may be used for modular development of type hierarchies. These modules define partial properties of a type hierarchy. Information can both be partial on the level of the hierarchy itself (partial subsumption relations) and on the properties of specific types (partial appropriateness conditions). The information in modules is defined on nodes in a graph. Modules can define which nodes are exposed to other modules and how other modules may interact with a node. Nodes can be seen as parameters that may allow information to be imported from or exported to other modules.

The partial descriptions of a type hierarchy in a module are defined as a partially specified signature (PSS). This is a finite directed graph, where nodes correspond to types and edges can either indicate subsumption relations or appropriateness conditions. Nodes may optionally be **marked** which means that the are associated with specific types, but they can also remain **anonymous**. Anonymous nodes are used to refer to types that are defined in another module. Within a module, there may be only one node referring to a specific type, i.e. the relation T used to mark some nodes with types must be one to one. A PSS must be partially ordered, but need not be a BCPO (see Section 2.2.1 for an explanation on BCPOs).

A signature module consists of a PSS, a set of internal types, a set of imported parameters and a set of exported parameters. The members of each set correspond to nodes in the PSS. Nodes which are not a member of any set are external nodes. Internal types[11] can be compared to local variables in programming: they cannot communicate with other modules. All other nodes (imported, exported and external) do communicate with other modules. Imported nodes receive information, exported nodes provide information and external nodes function similarly to global variables in programming.

Combining signature modules

Signature modules can be combined by two operations: **merge** and **attachment**. Both operations may only apply when the signature modules that are to be combined are consistent, i.e. they may not include types that are internal to the other module and they may not have common nodes.[12]

[11]Internal nodes are always marked (i.e. associated with a specific type): anonymous nodes should (at least) receive information from other modules.

[12]It may happen that the modules do have common nodes or internal nodes that have the same name. In this case, they "can be renamed without affecting the operation" (Sygal

When two modules are merged, their information is combined into a new module. An essential step in this process is the coalescence of nodes marked by the same type. Anonymous nodes may also be coalesced if they are indistinguishable. Merging starts with the simple union of the graphs defined by the respective modules. Then coalescence for identically marked nodes and indistinguishable anonymous nodes takes place by applying a compactness algorithm. This algorithm also removes redundant arcs from the graph (i.e. arcs that denote information that can also be deducted from other definitions). The next step inserts appropriate conditions to ensure the *relaxed upward closure condition*. This condition ensures that appropriate conditions defined on a supertype also apply to its subtypes (i.e. inheritance of appropriate conditions). This operation may result in new indistinguishable anonymous nodes and new redundant arcs. Therefore the compactness algorithm is applied again. Merge can only apply if the output constitutes a subsumption hierarchy that is a partial order.

The attachment operation is an asymmetric operation where a signature module S_1 receives another module S_2 as input. Similar to the merge operation, information from S_2 and S_1 is combined (that of S_2 is added to S_1). However, an additional asymmetric operation applies where the exported parameters from S_2 are linked to the imported parameters from S_1. The respective associated nodes from both modules are coalesced. Unlike the merge operation, this coalescence does not require nodes to be indistinguishable. This operation can only apply if the number of importing parameters in S_1 is equal to the number of exporting parameters in S_2. Furthermore, parameters that are linked to each other may not be associated with distinct types. Finally, the modules must be mergeable and the attachment must lead to a partial order without subsumption cycles.

and Wintner, 2011, p. 42).

Creating a well-defined grammar

The partial signature modules need not conform to all requirements of a type hierarchy defining a grammar. After all relevant modules are combined, a set of operations must be carried out to ensure a well-defined type hierarchy is formed. First, the combined modules may still contain anonymous nodes. These nodes are compared to nodes[13] that are marked (i.e. associated with a type) and if two nodes are found to be indistinguishable except for the markedness of one node and anonymity of the other, the nodes are coalesced. Anonymous nodes for which no indistinguishable marked node can be found receive an arbitrary type name. Then the signature is turned into a BCPO using Penn's (2000) BCPO completion algorithm. Next, appropriate conditions are passed down to newly created subtypes. This is followed by an algorithm that consolidates appropriateness conditions restricting the value of the same feature on the same type (assigning the most specific type as appropriate to the feature). Penn's (2000) feature introduction algorithm is applied to ensure the feature introduction condition. According to this condition, a feature may be introduced to only one type in the hierarchy, which subsumes all types that make reference to this feature. This algorithm can only be applied to a BCPO with correct appropriateness inheritance conditions, but at the same time, it may disrupt the bounded completeness. The BCPO algorithm and two algorithms passing on appropriateness conditions and collapsing conditions for identical features on the same type must therefore be applied again.

Grammar Modules

A grammar consists of instances (including both grammar rules and lexical items) and a start symbol which are associated with typed feature structures in a type hierarchy. A grammar module $M = < S, G >$ is defined in a similar manner: rules and words of the grammar G may be associated to nodes in

[13]When compactness applies for merge or attachment, both nodes must be anonymous to be indistinguishable.

the signature module S. When the merge and attachment operations are applied to grammar modules rather than signature modules, the signatures of the grammar modules are combined as before. Additionally, the grammar components are combined using set union.

Testing and Conclusion

The basic type hierarchy provided as an appendix in Pollard and Sag (1994) was implemented in signature modules. This hierarchy was chosen because it is big enough to contain the kind of knowledge found in linguistically motivated grammars and small enough to make the reorganisation in modules practical. Grammars that can analyse phenomena in natural language are, however, much larger and exhibit much more complex interactions.[14] Sygal and Wintner (2011) claim the extension of their approach to such grammars to be "feasible, but [...] beyond the scope of this preliminary work" (Sygal and Wintner, 2011, p. 36).

They conclude that all desirable properties of a modular approach to typed unification grammars are obtained by their proposal: It **focuses** on **signatures** allowing for a **partial** definition in modules. These modules can be **extended** deterministically in a well-defined type hierarchy. Modules must be **consistent** in order to be combined and are only restricted in that subcycles are not allowed (i.e. they are **flexible**). Parameters provide the possibility of **(remotely) referring** to nodes. All information from both modules is included when two modules are combined and only information to ensure well-definedness is added ensuring the desideratum of **parsimony**. Attachment is not, but the merge operation is **associative**. Finally, internal nodes are comparable to **private** variables.

[14]The basic type hierarchy only provides types from the top of a HPSG hierarchy. It does not provide complete analyses of phenomena in a specific language.

7.5.2 Modular typed unification grammars and CLIMB

When comparing modular typed unification grammars to CLIMB, similarities in setup and overall ideas are immediately apparent, especially for declarative CLIMB. As a matter of fact, Sygal and Wintner (2011) address the LinGO Grammar Matrix and, in particular the customisation system, in their related work as a similar approach that improves modularity. They point out that the approach is different from theirs, because only prewritten code is divided over different modules and the grammar writer has no control over the customisation system. This is exactly the point in which CLIMB differs from the Grammar Matrix. As explained in Section 3.1.1, the grammar engineers using CLIMB directly manipulate the libraries in the customisation system, and are thus fully in control of the language specific customisation system they are creating.

Both approaches allow the engineer to provide partial definitions of a type hierarchy in separate modules which can be combined to form a full grammar. CLIMB can, in principle, fulfil all nine desiderata set by Sygal and Wintner (2011) and fulfils five of them for the same reasons their own approach does, namely: **signature focus**, **partiality**, **extensibility**, **parsimony** and **associativity**. **Flexibility** is even stronger in CLIMB than in Sygal and Wintner's (2011) modular typed unification grammars, because partial descriptions in CLIMB need not constitute well-defined signatures. The feature geometry extraction and path abbreviation algorithm that run when a grammar is created for gCLIMB perform several **consistency** checks, as explained in Section 3.3.2, page 110. The Python part of CLIMB supports **remote reference** and **privacy**, but these properties are not supported within libraries of declarative CLIMB, where reference is achieved through identical naming only.

As mentioned above, one of the main differences between the two approaches is that improving modularity is the main goal of Sygal and Wintner's (2011) approach. CLIMB, on the other hand, is the result of a practical solution to a theoretical problem. Modularity improves through CLIMB because it is necessary for sharing grammar components and maintaining alternative analyses at the same time (regardless of whether alternatives are meant to support syntactic research or to capture crosslinguistic differences). For Sygal and Wintner (2011), modularity is a goal in itself, with a predefined set of desiderata. This makes their approach more principled (there are no formal restrictions on CLIMB modules), but it remains to be seen whether this will be advantageous in practice.

The first clear difference lies in the requirement that modules are partially ordered, which is absent from CLIMB. CLIMB may be considered less modular from this point of view, because definitions in a library must always be linked to the rest of the type hierarchy. There are no formal restrictions on consistency when combining information from two libraries with CLIMB. Well-formedness therefore only applies to the grammar as a whole.

In general, it is difficult to provide a critical comparison between the two approaches, because advantages and shortcomings will mostly become apparent in practice. This holds for the difference in requirements on modules expressed above, as well as for the two desiderata supported by Sygal and Wintner (2011) and not by CLIMB. It is not clear to me how internal nodes in modules facilitate grammar engineering. Anonymous nodes can be used to underspecify a part of the type hierarchy within a module. This feature clearly improves modularity in a manner that CLIMB does not, but I have not experienced challenges because of this lack of modularity throughout the development of gCLIMB. Of course, this does not mean that gCLIMB may not have been better had remote reference been supported.

A true comparison could only be provided after experience with both approaches on a large scale grammar. At present, CLIMB has the advantage that it has been used to develop such a grammar. This experiment must, to my knowledge, still be undertaken for Sygal and Wintner's (2011) approach. Even though I agree with the authors that the approach should scale to large projects in principle, I believe this may not be straightforward. Especially the scenario of multiple engineers working on a similar grammar is likely to become a challenge. Furthermore, Sygal and Wintner (2011) do not address what grammar engineers need to learn to adopt the approach. It may suffer from the same challenge as the original version of CLIMB: defining grammars in a new manner may form a hurdle for even experienced grammar engineers. Collaboration between the two approaches may lead to further improvements on both sides. The fact that a purely practical approach and a mainly theoretically motivated approach turn out this similar, despite being developed completely independently, is a good sign for both projects.

7.6 CoreGram

7.6.1 Basic description of CoreGram

CoreGram is the common core of a set of HPSG grammars developed at the Freie Universität Berlin (Müller, 2013). The CoreGram website[15] lists grammars for eight languages, namely German (Müller, 2007), Danish (Ørsnes, 2009b; Müller, 2009a; Müller and Ørsnes, 2011), Yiddish (Müller and Ørsnes, 2011), Persian (Müller, 2010; Müller and Ghayoomi, 2010), Maltese (Müller, 2009b), Mandarin Chinese (Lipenkova, 2008; Müller and Lipenkova, 2009), Spanish and French. All grammars are implemented using TRALE (Meurers et al., 2002; Penn, 2004).

The primary focus in these grammars lies on implementing accurate HPSG analyses. They are generally closer to standard HPSG theory than DELPH-IN grammars, which often prefer practical or more efficient approaches. In fact, Müller (1999) is one of the first works that puts Bierwisch's (1963) proposal into practice and uses implementations to verify correct interactions of linguistic analyses. This is in line with the references for the grammar, which all refer to papers on syntactic phenomena. This does not mean that engineering aspects are not considered in the development of CoreGram grammars. Müller takes the point of view that the choice between linguistic motivation and efficiency is comparable to the choice between Java and C programming. Whereas Java used to be avoided for lack of efficiency, it became more and

[15]`http://hpsg.fu-berlin.de/Projects/core.html`, accessed 30 June 2012.

more dominant later because it is cleaner (Müller, p.c.).

CoreGram is intended to support sharing implementations across grammars. It consists of a core containing definitions that hold for all languages as well as subcores containing definitions for a subgroup of languages. The developers take a bottom-up approach, where languages are analysed independently. The implementations of individual languages are compared and, where possible, generalisations are added to the core grammar. It is determined empirically whether an analysis or parts of it can be included in the core or a file shared by a subgroup of languages. In principle, analyses that are already used in other grammars are preferred, but only if there is no contradictory evidence. Crosslinguistic applicability is thus stimulated, but language specific evidence always comes first. Müller calls the approach a "*bottom-up approach with cheating*" (Müller, 2013, p. 96), where the approach is "cheating" in the sense that a top-down approach is used to choose among alternative analyses. Even though such research may provide insight into universals in language, no strong claims are made on this account.

Technically, the CoreGram setup uses the possibility provided by TRALE of defining properties of types at different locations of the grammar. This functionality corresponds to the use of addenda (definitions using `:+` to add properties to a type defined elsewhere in the grammar, see Section 2.2.1) used in DELPH-IN grammars. Implementations belonging to the core, subcores and language specific parts of the grammar are placed in different files. Grammars for individual languages are created by loading a selection of files. Grammars thus share identical files at all time. CoreGram differs in this aspect from the Grammar Matrix, where grammar engineers take the common core and use this as a basis to create their individual grammars. There is no impact on other grammars when a change to the core is made.[16] The advantage of the CoreGram approach is that common components of grammars are constantly updated when new information from individual grammars is

[16]As mentioned above, the original idea behind the Grammar Matrix was that such changes would be reported and the core adapted accordingly. Grammar engineers could then decide to upgrade the core in their grammar so that revisions are used in individual grammars.

found. The Grammar Matrix, on the other hand, avoids situations where individual grammars have to be revised in order to work with the changes in the core.

Another consequence of this difference is that grammar writers using CoreGram should try to agree on fundamental, language independent properties of their grammars, whereas grammar writers using the Grammar Matrix can make fundamental changes to the core if they have other theoretical ideas. The advantage of sharing the same principles is that all grammar writers contribute to the same theoretical foundation increasing its validity. Leaving grammar writers free to make fundamental changes has the advantage that they do not need to compromise their point of view on linguistic theory, which may become more difficult when the resource is used by more linguists.

7.6.2 A Metagrammar approach for CoreGram?

Despite its HPSG background, interest in linguistic precision and facilitation of sharing implementations, CoreGram is a rather different approach from CLIMB. The approach does not use code generation. The organisation of parts of the grammars across files that may be included or excluded from the grammar is similar to declarative CLIMB, but the absence of code generation leaves out the option of both working on the level of phenomena (partial definitions of several types) and types (mostly complete type definitions with constraints related to several phenomena) at the same time. CoreGram does not (to my knowledge) contain an equivalent of the dynamic component of CLIMB that can speed up grammar development and capture similarities across languages where individual feature values differ by generating lexical items or rules.

Systematic comparison of analyses through implementation has not been carried out on a large scale within CoreGram. In the literature discussing analyses of the grammars mentioned above, explanatory adequacy and generalisation are used as primary criteria to choose between analyses. Implementations serve the purpose of verifying the correct behaviour of analyses and their interaction with the rest of the grammar, but are not use to explore alternatives. The basic setup of CoreGram would allow for such research by including alternatives in separate files. In principle, a large number of variations could be captured this way, though it may turn out to be a suboptimal form of organisation[17] (imagine several levels of subcores to cover variations in Slavic languages, variations in analyses and variations for different applications).

Given the goal of providing correct syntactic models of language, metagrammar engineering is expected to be of interest to researchers working on these grammars. After all, a systematic test of a model against possible alternatives augments its validity. The CLIMB approach can thus be recommended to the developers of CoreGram. There are no technical reasons why the CoreGram grammars could not be generated from a metagrammar like CLIMB. As a matter of fact, it should be straightforward to adapt the tools provided by the Grammar Matrix for these grammars, because they use the same formalism. CoreGram and its related projects could provide an excellent platform to test the linguistic contributions of metagrammar engineering.

7.7 An overview of related work and CLIMB

The previous sections have described several approaches to grammar development that use metagrammars, code generation, code sharing or aim at consistency across grammars. In this section, I will briefly repeat the main similarities and differences between these approaches and CLIMB.

The main purpose of CLIMB is to improve the possibility of empirically exploring alternative analyses in linguistic precision grammars. As we have seen, none of the projects described above have aspired to this goal. Nor has, to my knowledge, any research in this direction been carried out in other projects. However, several other advantages of CLIMB can be found in

[17]Lars Hellan expressed concerns about the proposal to increase the number of files to capture variations or distinguish phenomena from each other, because it would reduce the maintainability of the grammar. DELPH-IN Summit, Sofia, 5 July 2012.

the other projects.

7.7.1 Maintainability and Consistency

Both (X)MG (Section 7.1) and the GF Resource Libraries (Section 7.2) aim to improve maintainability of the grammar through the organisation of the metagrammar or libraries, respectively. This aspect is directly related to the improvement in consistency within the grammar and across related grammars, an aspect that is also found in ParGram (Section 7.3) and, presumably,[18] in CoreGram (Section 7.6).

7.7.2 Multilinguality

Multilinguality is found in most of the approaches above. All except Sygal and Wintner (2011) (Section 7.5) explicitly mention multilinguality as part of their approach, even though it is not the main focus of (X)MG and research on multilingual applicability has only taken place on a limited ground in that project. In addition to the degree to which multilinguality is pursued, individual projects differ in the approach taken towards multilinguality. Projects differ in technical choices as well as the question of whether they prefer a top-down or bottom-up approach establishing which implementations are shared crosslinguistically.

CoreGram can be placed on one extreme, where analyses for individual grammars are developed independently. At a later stage, it is investigated whether common factors between analyses exist that may be placed in the common core. PAWS, on the other hand, is a typical example of a top-down approach. Implementations are provided based on properties known to exist in languages, but (as far as one can tell from the literature) no profound study of individual languages is performed to develop the analyses. The other approaches are somewhere in between these two extremes.

[18]The structure and methodology of CoreGram are very suitable to improve consistency, but I am not aware of the developers addressing this aspect of their project.

ParGram carefully looks at the grammars under current development to decide on crosslinguistic analyses. Sometimes elaborate test cases are implemented to learn more about an analysis. From this point of view, it is bottom-up. On the other hand, the aim to fit new analyses into the existing system is more in line with a top-down approach. Analyses in GF are based on linguistic observations of individual languages (bottom-up), but each new language uses existing analyses which are only adapted if they are found not work (top-down).

Both elements can also be observed in the Grammar Matrix. Like PAWS (Section 7.4), it provides basic analyses covering a large typological range for new languages. On the other hand, both the creation of the original core grammar as the design of libraries in the customisation system involve studying the behaviour of individual languages. The same can be said about CLIMB in multilingual setting. In principle, it can be used both in a bottom-up and a top-down manner. In the original Germanic project, the focus lies more on bottom-up grammar development. SlaviCLIMB has a bottom-up aspect, because it involves development of precision grammars for individual languages, but the main idea of providing a Slavic core grammar based on theoretical assumption can be seen as top-down.

7.7.3 Modularity

Sygal and Wintner's (2011) modular typed unification grammars are the most relevant related work at this point. Based on the detailed comparison above, this work shows that CLIMB manages to cover many desiderata of a modular approach to defining typed feature structures. The main technical difference is that CLIMB is more flexible (or less principled) in its requirements on modules. This leads to more freedom for the developers as to how to organise the grammar, but a greater risk of introducing errors in individual modules. Because Sygal and Wintner's (2011) approach has not been applied to a large scale grammar at this point, it is not clear how the two approaches compare in their influence on the grammar development process on the long run.

MG explicitly mentions improving modularity as one of its motivations (Candito, 1999). It is achieved by the division into dimensions. This aspect of MG is highly similar to CLIMB, where modularity is increased by organising the grammar into libraries. The focus in the two approaches differs in that MG separates different levels of information in its dimensions, whereas CLIMB separates parts of the grammar according to the phenomenon they analyse.

Modularity has also been addressed in relation to ParGram. It is an aspect that has been addressed extensively in the framework of LFG. It is therefore not as much a purpose of ParGram, but more a feature of the linguistic theory ParGram uses from which the ParGram project benefits.

7.7.4 The CLIMB idea for other projects

The proposal made in this thesis can in principle be adopted for each of the projects above. However, the amount of effort that would be involved as well as the potential interest in adopting metagrammar engineering differs from one project to another. Feasibility is directly related to the method used for sharing implementations. It is not possible to make strong claims about potential interest from researchers involved in the other projects, but this is likely to be related to whether or not the project is based on linguistic theory.

As far as implementation methods and feasibility are concerned, two manners of sharing code can be distinguished: static code sharing and dynamic code sharing. ParGram, GF and CoreGram develop sets of definitions that can be used by individual grammars. They share code statically. (X)MG and PAWS use code generation in their systems, just like CLIMB. Their method is more dynamic. Even though it is in principle possible to systematically compare implementations using different versions of static code, it is code generation that provides a basic methodology that facilitates and stimulates this. (X)MG would not require many changes to its architecture to be able to carry out systematic comparative research.

The second question is whether systematic comparison may be of interest to the projects. As far as they address the question of alternative possibilities, they tend to go for restricting possibilities based on crosslinguistic research. This is specifically mentioned in Butt et al. (2002). Ranta (2009) also aims to share as much as possible between grammars. In many ways, this attitude towards alternatives makes sense. If data cannot provide clear evidence for a particular analysis, any other indications to help and choose are welcome. But in the end, it seems clear that testing more analyses integrated into a large grammar gives a more reliable indication than merely discussing alternatives or only trying things out at the moment they first come up. In principle, all approaches that are interested in optimising their grammars could therefore benefit from a methodology that can help to compare analyses systematically.

PAWS does not seem to focus much on its PC-PATR implementations. It neither makes claims about linguistic theory, nor are there explicit references to intentions of building grammars for applications. Even here, it could make sense to maintain multiple alternatives in parallel for some time while extending the system, but most observations that may come out of such a practice would not be of primary interest to PAWS users.

There may be more of an interest in using a metagrammar development approach in the GF Resource Library. Systematic exploration can lead to more efficient grammars, which is generally interesting for application-oriented grammar engineering. Moreover, it might be possible to get more out of sharing analyses between different application grammars if code generation were introduced. For now, it seems that most of the sharing is done through the core grammar.

The other three projects may have a clearer interest in systematic exploration of alternatives. All three are grounded in linguistic theories. Despite the occasional influence of practical aspects, both (X)MG and ParGram aim for representing theoretical assumptions from TAG and, respectively, LFG correctly. Systematic exploration would make the grammars more flexible to changes in the theory and increase the support the respective theories can get from implemented grammars. As mentioned above, ParGram already ran into a situation where code generation to adapt the grammar would have been extremely useful.

Of all projects other than DELPH-IN,[19] the proposal made in this thesis is in my opinion most relevant for CoreGram. The grammars in this project are all intended to provide correct HPSG models of language. To my knowledge, they are not application driven and will therefore not make practical decisions to improve grammar performance if this is not in line with HPSG theory. Even though the CoreGram developers are probably not interested in experiments comparing computational efficiency, they seem to care more than the other approaches about finding a correct model of language. Theoretical syntax suffers as much from multiple alternative analyses and potential interactions as grammar engineering does, if not more. Adapting an approach similar to that of CLIMB for the CoreGram grammars would therefore, in my opinion, mean a significant improvement in their methodology.

[19]Recall that CLIMB can be used to develop any grammar in TDL. DELPH-IN thus refers to all grammars developed within DELPH-IN regardless of whether they use the Grammar Matrix or not.

Chapter 8

Conclusion and Future Work

In this final chapter, I will recapitulate the main contributions and observations made in this thesis. The goal of this investigation was to enhance empirical research on linguistic precision grammars. I will explain how this is achieved with CLIMB in Section 8.1. In Section 8.2, I will summarise additional contributions of CLIMB and discuss related contributions such as the spring cleaning algorithm. Section 8.3 discusses open issues and outlines future work. Final concluding remarks are provided in Section 8.4.

8.1 CLIMB: Enhancing empirical research

This thesis started with the statement that grammars of natural language are complex objects. This complexity forms a challenge in any effort to represent such grammars, both as part of syntactic theory and as part of computational grammars. Chapter 7 has presented several approaches across theories that aim at providing support in such efforts by sharing knowledge, increasing modularity and improving maintainability and consistency. This thesis introduced CLIMB, a methodology that addresses these aspects for DELPH-IN grammars. The main idea behind the approach is that analyses are added to a metagrammar that can automatically generate computational grammars. The CLIMB methodology and software are, however, more than a

HPSG and DELPH-IN specific version of grammar engineering support found in other frameworks. It is to my knowledge the first approach that is developed to facilitate maintenance of multiple alternative analyses in parallel.

This property of CLIMB addresses an important challenge faced in both theoretical syntax and grammar engineering, which is the result of two well-known challenges combined: phenomena and thus their analyses interact and often more than one analysis can be found that can account for the data. Because of the interaction, the choice of analysis for a given phenomenon can influence the possibilities for linguists to be able to analyse other phenomena. Grammars are too complex to foresee how a choice at a given time may influence possibilities of future analyses or even to maintain a clear overview of how analyses restrict each other exactly. Grammar engineering allows syntacticians to check whether analyses interact correctly, but traditional grammar engineering does not provide a straightforward method to see whether another choice in the past may have led to different options at present.

Because CLIMB allows grammar engineers to maintain alternative analysis in parallel, they can systematically test how these alternative analyses interact with analyses of other phenomena. When no conclusive evidence is found, the grammar engineer can postpone the decision and continue testing the alternative as the grammar covers more phenomena. The CLIMB methodology thus enhances the possibility of empirically testing analyses.

The challenge that follows from the combination of inconclusive evidence and interaction between phenomena is, to my knowledge, first formulated in work carried out as part of this thesis. It follows that the CLIMB methodology is the first proposal to address this problem, as far as I am aware. Because this observation has, in my opinion, important theoretical and practical implications, they form the main contribution of this work.

The most important step in evaluating the approach involved investigating whether it is possible to maintain multiple analyses while developing a large scale grammar with CLIMB. This was confirmed by developing gCLIMB, a metagrammar that can generate grammars for German with alternative analyses for word order and auxiliaries that cover (a little more than) all phenomena in the development set of Cramer's (2011) Cheetah that Cheetah covers as well. The development of gCLIMB took less than half the time that had been needed to create Cheetah's core, but as explained in Chapter 5, there are many factors that influence development speed. This result does therefore not allow us to conclude that CLIMB speeds up grammar development. The fact that gCLIMB also focused on other languages in early stages, includes alternative analyses, aimed for more complex semantic representations than Cheetah and slightly outperforms it when comparing coverage on linguistic phenomena does strongly indicate that using CLIMB in principle does not have a significant negative impact on grammar development speed and may even have a positive effect. It should be noted that Cramer, on the other hand, spent additional time on improving efficiency of his grammar and outperforms gCLIMB on the TiGer corpus. It should furthermore be noted that the current result only applies to the question of whether using a metagrammar in the form of CLIMB slows you down when using a handful of alternative analyses. As the number of alternative analyses increases, grammar development and testing will naturally require more time.

Experiments carried out with gCLIMB investigated the efficiency of alternative analyses as the grammars covered more analyses. They confirmed the prediction made by Fokkens (2011a) that the difference in efficiency between the aux+verb analysis compared to the standard HPSG argument composition analysis increases as the grammar covers more phenomena. However, this difference can be reduced by introducing additional constraints on grammar rules that go against HPSG's lexicalist character, but do capture properties of the German language. Additionally, language generation experiments were carried out that investigate the interaction of word order constraints and alternative analyses for Dutch. These experiments show that it is easy to carry out experiments with CLIMB in a controlled environment that would be time consuming and difficult when traditional grammar engineering is used. The evaluation thus showed that CLIMB indeed provides a method that facilitates and stimulates experimenting with grammars.

8.2 Other aspects of CLIMB and related tools

Throughout this thesis, several related topics to the CLIMB solution for systematic testing were addressed. This section provides an overview of results found in this thesis other than the main result addressed in Section 8.1. They involve additional software that was developed to support CLIMB, properties of CLIMB other than its support of systematic testing and studies on the Grammar Matrix core. I will briefly elaborate on each of these aspects below.

Chapter 3 introduced additional software related to CLIMB. This software mostly supports practical functions. The spring cleaning algorithm identifies types in DELPH-IN grammars that do not have an impact on the grammar's competence. It was originally developed to help and compare two similar versions of a grammar in order to find which differences should be included in the metagrammar. In this case, only properties that have an impact on the grammar's behaviour should be taken into account. The algorithm creates a version of the grammar that no longer contains the types that have no impact on the grammar's competence, but otherwise maintains the original structure of the grammar. The algorithm also indicates which types were removed and thus provides insight into properties of the grammar. Furthermore, a feature extraction, path abbreviation and path completion algorithm were introduced. These algorithms mainly make the grammar engineer's life easier by supporting the use of abbreviations in the grammar, but they also provide a way to introduce minor changes in feature geometry in a straightforward manner.

The CLIMB method and its (related) software have several advantages over traditional grammar engineering other than providing a setup for systematic testing. They include increasing modularity, flexibly sharing analysis across languages, supporting alternative analyses for different applications or different dialects, a phenomenon-based organisation of phenomena and the possibility of including or excluding rare phenomena. As we have seen in Chapter 7, several of these advantages are also found in other methodologies and approaches to grammar engineering. Even though these properties are not unique to CLIMB, they still form an important contribution for grammars developed within DELPH-IN and, more generally, computational grammars developed within the framework of HPSG. The Grammar Matrix Bender et al. (2010) and CoreGram Müller (2013) both support code sharing across languages. However, as explained in Chapter 6, the Grammar Matrix cannot offer as much analytical depth, because it aspires to provide useful implementations for any language. CoreGram shares implementations statically and does not make use of dynamic code sharing that allows for parametrisation. Sygal and Wintner (2011) focus on improving modularity and provide a more mathematically sound approach for this than CLIMB, but their approach has, to my knowledge, not been tested yet in project of a scale comparable to gCLIMB.

Chapter 6 investigated multilingual aspects of the approach. The metagrammar mainly developed for German was used to create grammars for Dutch, Danish and Northern Frisian. The outcome of these studies shows that in order to create a metagrammar that truly captures crosslinguistic variation in a group of languages, it is important to take these variations into account throughout grammar development. If you focus completely on one language (as was done in gCLIMB), it is likely that at least some fundamental changes are needed in grammars for other languages. On the other hand, several phenomena were covered correctly with no or only minor changes to the metagrammar. This indicates that CLIMB indeed increases modularity. Analyses mainly developed for German could easily be adapted to cover variations found in other languages. It is unlikely that a similar result could have been obtained with a grammar implemented in the traditional way, which leads me to conclude that the approach is suitable for crosslinguistic grammar development.

It was clear even before these studies were conducted that CLIMB can be used for multilingual grammar development, since it makes use of the same technology as the Grammar Matrix customisation system. We have shown the capacities of the Grammar Matrix customisation system in Bender et al.

(2010). However, CLIMB takes these possibilities in a whole new direction by continuously using this technology for a small group of languages. It is unlikely the results obtained for Dutch, Danish and Northern Frisian with CLIMB would also have been obtained while only using the Grammar Matrix. This is confirmed by the results for Northern Frisian. The grammar developed by gCLIMB managed to reach 94.5% coverage and no overgeneration in one day, whereas Kilmer and Packard's grammar reached 70.6% of coverage and 1.8% overgeneration by the end of the *Knowledge Engineering for NLP* course. As has been pointed out at several occasions in this thesis, no firm conclusions can be drawn from comparing two grammar engineering efforts and it should be taken into consideration that Kilmer and Packard also had to develop the test suite. Nevertheless, the significant difference in time and coverage and the fact that Packard, who has developed the ACE parser and generator for DELPH-IN grammars and knows the formalism very well indicate that gCLIMB provided a major boost while developing the grammar for Northern Frisian.

Another context for exploring advantages of applying CLIMB technology to a closed set of related languages is provided by SlaviCLIMB, where CLIMB is used to support the development of a core grammar for Slavic analyses. SlaviCLIMB is currently in its initial stages and its potential will be discussed in more detail in Section 8.3.

The development of gCLIMB led to the most extensive revisions of the Grammar Matrix core since its release. Several revisions were bug fixes and some removed English specific properties. Because it seemed unlikely that none of the other grammars using the Matrix core ran into similar issues, Chapter 6 also investigated how individual languages use the Matrix core. The Grammar Matrix core provides general (mostly) language independent implementations for DELPH-IN grammars. The spring cleaning algorithm described in Section 3.3 was used to support this investigation. As mentioned above, this algorithm identifies types that do not have an impact on the competence of the grammar and removes them. This leads to a cleaned up version of the grammar, where types that do not have an impact on the grammar are removed, but the structure of the grammar is preserved. The impact of the Grammar Matrix was investigated by looking at changes that were made to the Grammar Matrix core and the types that were removed by the spring cleaning grammar.

The most important observation out of this investigation is that several grammar engineers made changes to the Grammar Matrix core that form a general improvement to the Grammar Matrix. The current setup of the Grammar Matrix, where grammar writers typically go their own way after using the Grammar Matrix for a jump start, provides complete flexibility to grammar writers. They can make any change that suits them without needing to explain their motivation to fellow grammar writers. The disadvantage is that there is no need for grammar writers to discuss the changes they make with fellow grammar writers and often changes remain unnoticed by developers of other grammars and of the Grammar Matrix. Valuable information on how the Grammar Matrix core works is thus lost and grammar engineers need to reinvent the wheel when it comes to dealing with shortcomings of the resource. The study carried out in Chapter 6 is, to my knowledge, the first overview of the Matrix core types that are used and changes made to the core in different Matrix-based grammars. The spring cleaning algorithm played an important role in this investigation revealing that this algorithm can also be used to support empirical investigation for linguistic precision grammars.

Finally, this thesis provided an overview of efforts to share knowledge, improve modularity and facilitate multilingual grammar engineering in different frameworks. It was explained how the linguistic theory or formalism that forms the foundation of the grammar development effort and overall goals of the project influence the approach that is taken.

8.3 Discussion and Future work

Sections 8.1 and 8.2 have provided an overview of the most important contributions of this thesis. In this section, I will reflect on a few critical aspects of CLIMB and the studies that have been carried out with this resource so far and explain how they can be addressed in future work.

8.3.1 Accessibility

As explained in Section 3.2, using CLIMB involves a learning curve that may prevent grammar engineers from adopting the method. It requires writing procedural code while writers of HPSG grammars are used to declarative programming. Even programmers that have experience in procedural programming may feel uncomfortable flipping back and forth between declarative programming in TDL and procedural programming in Python. Declarative CLIMB addresses this issue by allowing grammar engineers to write their metagrammar in TDL, but it does not offer the full flexibility of procedural CLIMB. In particular, the research questions set out to be addressed by SlaviCLIMB involve automated adaptations of type hierarchies that declarative CLIMB cannot handle. Future work around CLIMB should therefore first and foremost focus on making CLIMB more accessible to grammar engineers. I will briefly outline my vision of CLIMB below.

Ideally, grammar writers would be able to make use of the full functionality of (procedural) CLIMB without the burden of learning a new way of programming and constantly flipping back between declarative and procedural programming. They should be able to write their code declaratively in TDL as much as possible and increase flexibility through simple statements. Procedural code responsible for the flexibility remains hidden to the users as is currently the case for declarative CLIMB. I would like to achieve this by implementing functions that generate the necessary Python code for the metagrammar based on the grammar writer's input through a user interface.

Figure 8.1 provides a schematic overview of the idea behind this future form

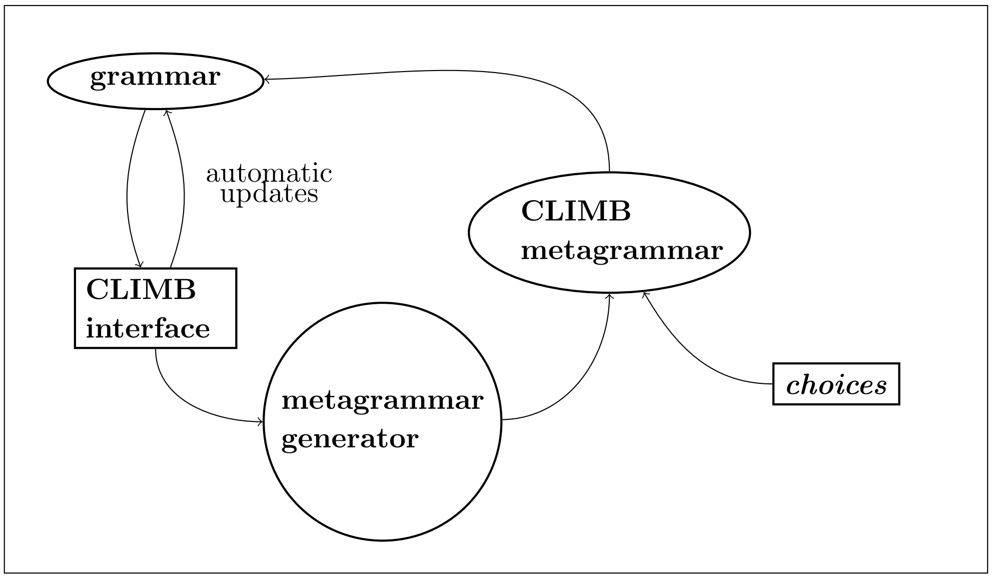
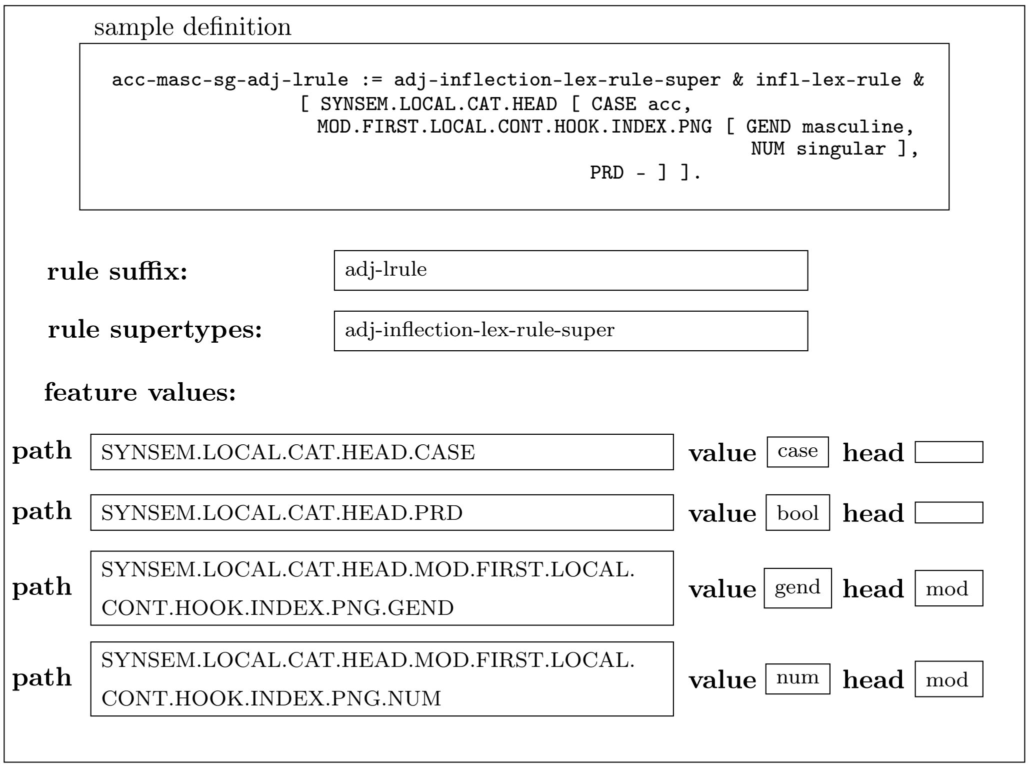


Figure 8.1: Schematic overview of future form of CLIMB

of CLIMB. Grammar developers can work on their grammars using a specially designed interface. Most changes (additional constraints on existing types and new types) will be added to the grammar they are working on immediately for testing. This functionality can be provided by the existing declarative CLIMB code with only minor adaptations. The interface offers the additional possibility of associating these properties with a particular analysis and or phenomenon. Implementations together with their associations are used as input to generate an updated version of the metagrammar.

It is relatively straightforward to generate functions supporting the functionalities offered by declarative CLIMB. Basic general functions needed to define flexible type hierarchies (as required for developing SlaviCLIMB) are already present in the Grammar Matrix customisation system and gCLIMB. This code can be generalised easily so that users can define new lexical items, locations in feature geometry and basic hierarchies that should be considered for automatic revision. The main challenge of this further development of CLIMB lies in designing a good user interface that is intuitive and allows users to make maximal use of CLIMB's advantages. An initial idea of how a grammar writer may provide the basis for a new morphotactic rule is provided in Figure 8.2.



sample definition

```
acc-masc-sg-adj-lrule := adj-inflection-lex-rule-super & infl-lex-rule &
                [ SYNSEM.LOCAL.CAT.HEAD [ CASE acc,
                  MOD.FIRST.LOCAL.CONT.HOOK.INDEX.PNG [ GEND masculine,
                                                        NUM singular ],
                                    PRD - ] ].
```

rule suffix: adj-lrule

rule supertypes: adj-inflection-lex-rule-super

feature values:

path	SYNSEM.LOCAL.CAT.HEAD.CASE	**value**	case	**head**	
path	SYNSEM.LOCAL.CAT.HEAD.PRD	**value**	bool	**head**	
path	SYNSEM.LOCAL.CAT.HEAD.MOD.FIRST.LOCAL. CONT.HOOK.INDEX.PNG.GEND	**value**	gend	**head**	mod
path	SYNSEM.LOCAL.CAT.HEAD.MOD.FIRST.LOCAL. CONT.HOOK.INDEX.PNG.NUM	**value**	num	**head**	mod

Figure 8.2: Sketch of fields for defining new lexical types in future CLIMB

The grammar engineer can define a prototype rule or lexical item, e.g. the *acc-masc-sg-adj-lrule* in the example. The interface will then generate a list of questions with default answers concerning the properties of the rule. The first question allows the grammar engineer to define the ending of the rule identifiers (the default being the last two dash separated parts, if applicable). The second question provides the opportunity to define one or more standard supertypes. If the definition includes supertypes, *infl-lex-rule* or *const-lex-rule* will be ignored, because CLIMB already contains the necessary code to assign either of these types as supertypes. Finally, all the properties assigned to the type are analysed and the path to the feature value and basic value are assigned.

All these properties can be adapted manually by the grammar writer in case there are errors. The metagrammar generator will then add the necessary rules to assign values to the right feature in the feature geometry and to assign the correct supertypes. The metagrammar then contains the necessary machinery to generate types similar to the sample type.

The definitions in *choices* to create the sample type from a metagrammar are the following:

```
adj-pc1_lrt3_name=acc-masc-sg
      adj-pc1_lrt3_feat1_name=case
      adj-pc1_lrt3_feat1_value=acc
      adj-pc1_lrt3_feat2_name=gender
      adj-pc1_lrt3_feat2_value=masculine
      adj-pc1_lrt3_feat2_head=mod
      adj-pc1_lrt3_feat3_name=number
      adj-pc1_lrt3_feat3_value=singular
      adj-pc1_lrt3_feat3_head=mod
      adj-pc1_lrt3_feat4_name=prd
      adj-pc1_lrt3_feat4_value=-
      adj-pc1_lrt3_lri1_inflecting=yes
      adj-pc1_lrt3_lri1_orth=en
```

Other inflection rules for adjectives can then be defined in a similar manner in *choices*. It should be noted that all these properties can optionally be defined for a type. The user thus needs to give only one sample of a complete type definition and can then define a hierarchy using supertypes for which not all values are specified. Moreover, when more than one value is assigned to a certain feature, the code to adapt the type hierarchy is already present. For instance, when the value *masculine, feminine* is assigned to *gender*, the gender hierarchy will be adapted to contain a value *masculine+feminine*. The metagrammar generator will be able to provide the full flexibility offered by procedural CLIMB to the grammar engineer based on the definition of a sample type. Naturally, the grammar engineer can also directly define basic properties by providing answers to the questions proposed in Figure 8.2 without defining a sample type first.

The implementations needed to generate the metagrammar involve generalisations and some restructuring of the current libraries used in CLIMB. It is, however, still an open question whether grammar engineers will feel comfortable writing grammars through a CLIMB interface and whether the proposed approach truly enables them to take full advantage of the possibilities offered by CLIMB. In the end, the method can only be made more accessible to grammar writers while collaborating with them. SlaviCLIMB provides an excellent setting for developing and testing a CLIMB interface, if work is continued on this project. Furthermore, initial steps are currently being made to start a collaborative project on developing grammars for Chinese using CLIMB (Bond p.c.). This project will focus on Mandarin Chinese and aim to share Mandarin analyses for Cantonese. Finally, Sanghoun Song has plans for writing a grammar for Korean using CLIMB (Song, p.c.) providing an additional opportunity for collaboration.

8.3.2 Further explorations with CLIMB

The evaluation in Chapter 5 has shown that CLIMB can be used to develop a large scale grammar and test alternative analyses for phenomena that are central to the grammar. If we do not take the infinite possibilities provided by dynamic parts of CLIMB for defining morphotactic rules and hierarchies for lexical items into account, gCLIMB supports three fundamentally different analyses of linguistic phenomena. These analyses cover German word order and auxiliaries. Comparative analyses addressed the efficiency of the grammars and, to some extent, their capability of accounting for the data (though this was highly similar for the individual accounts). The purpose of the evaluation was not to end up with the best possible German grammar in HPSG, but rather to examine whether it is feasible to write grammars of a certain complexity with CLIMB. This naturally involved including correct analyses and make the grammars as good as possible, but it did not lead to a result that provided new insights into German syntax.

In order to come to such a result, many more alternative analyses need to be considered. More importantly, grammar development must then be combined with linguistic investigation of the data. Such an investigation provides enough research questions for several PhD theses in itself and it was naturally out of scope for this work. However, a broader application of CLIMB involving more alternative possibilities would lead to better insight into the potential of CLIMB. Particularly, it could address the question of whether it can provide a platform for testing alternatives as part of syntactic research.

A complete answer to this question can probably only be given if CLIMB were to be adopted by a syntactician working on a particular language for many years, but gCLIMB does provide an interesting starting point for research in this direction. Two other DELPH-IN grammars for German exist, namely GG (Müller and Kasper, 2000; Crysmann, 2005) and Cheetah. A third German grammar based on HPSG, BerliGram (Müller, 2007), exists that runs on TRALE (Meurers et al., 2002; Penn, 2004). Each of these grammars include different analyses for word order and other phenomena. A lot of insight into the impact of individual analyses and their interaction with other parts of the grammar could be obtained if analyses from these grammars are included in gCLIMB. Adapting gCLIMB so that it also outputs grammars that run on TRALE should be relatively straightforward, since TRALE and DELPH-IN grammars both use typed feature structures to define their grammars. Existing validation mechanisms that CLIMB has adopted from the Grammar Matrix can be used to ensure that no properties that are only supported by TRALE end up in grammars intended to run on DELPH-IN tools. The biggest challenge in comparing analyses of these grammars is that they make some fundamentally different assumptions on the feature geometry of signs in German. These assumptions interfere with almost all definitions in the grammar. Flexible feature geometry is currently limited in CLIMB to definitions for lexical rules and specific lexical items. A significant revision is required to allow CLIMB to output a different feature geometry depending on the chosen analysis for any type in the grammar. The implementations of the path abbreviation algorithm and path completion algorithm provide a basis for such a revision. An additional advantage for supporting alternative feature geometries would be that it allows for Haugereid's (2011) alternative subcategorisation approach to be integrated in gCLIMB as suggested in Section 5.3.

Further possibilities for testing CLIMB on a larger scale than has been done so far are provided by SlaviCLIMB (cf. Section 6.3). This project has the advantage that it is based on an elaborate theoretical study by Avgustinova (2007). SlaviCLIMB is intended to provide a platform for grammar engineers and Slavicists to work together on linguistic precision grammars. These grammars can partially be written through the definition of linguistic observations by Slavicists. These observations will be used as input for SlaviCLIMB which generates a grammar based on these observations. The generated grammars can be compared to Avgustinova's proposal. Hence SlaviCLIMB provides a platform for empirically testing theoretical work.

In addition to the research mentioned above, two more research directions for CLIMB were explored in this thesis. The first direction concerned an initial study where the lexicon that Cramer (2011) derived from the TiGer Treebank (Brants et al., 2002) for Cheetah was included in gCLIMB. Initial observations show that integrating such a lexicon is not trivial. Coverage on 1,000 test sentences from the TiGer corpus was only 21.3% for the best performing gCLIMB grammar despite the fact that the grammars had full lexical coverage for these sentences. One of the problems was that the lexicon did not take generalisations captured in the grammar into account. For instance, gCLIMB defines lexical types for adverbs that capture the flexible behaviour they exhibit in German (in the sense that they can modify verbs, adpositional phrases, adjectives, other adverbs, etc.), whereas the lexicon derived from TiGer contains a separate entry for each different category the adverb may modify resulting in multiple entries for a single adverb. This mismatch between the lexical type hierarchy and the lexicon read off the treebank leads to inefficiencies in the grammar. Future work will need to address both how we can introduce generalisations in the lexicon that is read off a treebank and how the lexical hierarchy can be defined in a way that is compatible to the information the treebank provides.

The second direction that was explored involved using CLIMB for alternative applications. The current version of gCLIMB can create grammars for German that allow users to practice adjective endings. Grammars for language learning typically involve so-called mal-rules that can parse ungrammatical structures and identify the error that was made. For DELPH-IN grammars, this means creating alternative versions of existing grammars that include the necessary mal-rules. Because CLIMB makes it easier to create alternative versions of a grammar, its potential for this application is clear.

8.4 Concluding remarks

Section 8.3 has addressed some critical aspects of the approach proposed in this thesis. The main issues left open in this thesis are the hurdle grammar engineers may experience when learning how to work with CLIMB and the fact that the scope of the evaluation with gCLIMB is too small to lead to new insights into German syntax. Nevertheless, this thesis has presented several results that show the potential of CLIMB. It is easier to adapt a grammar written in CLIMB to cover phenomena that behave slightly differently in a related language. Three alternative analyses for German auxiliaries and word order were compared in grammars covering a wide range of phenomena. Their efficiency was compared and a study involving the interaction between linguistic properties, alternative analyses and efficiency in natural language generation was conducted using CLIMB. These results show that CLIMB can be used to conduct empirical research that would be significantly less feasible (if not virtually impossible) in traditional grammar engineering. The spring cleaning algorithm developed to support CLIMB also led to new insights into linguistic precision grammars. It identifies types that are not used by the grammar. The algorithm was applied to a number of grammars using the Grammar Matrix core resulting in a unique study on the way the Matrix core is used in individual grammars.

Overall, the methodology proposed in this thesis and the software developed to support it provide a platform for research on linguistic precision grammars that could not be carried out before, or only with great difficulty. This thesis thus succeeded in its objective to enhance empirical research for linguistically motivated precision grammars. This work presented several indicative results which suggest topics to investigate in future work, which is exactly the outcome one would want when introducing a new methodology. I believe that a new method for carrying out research should first and foremost offer new directions of investigation. It is my hope that this work will inspire fellow research to take the tools offered by this thesis, explore and learn more about grammars of natural language.

Appendix A

Cheetah test set (with coverage indication)

Overview of sentences in the Cheetah development set. The most important phenomena of this set are presented in Chapter 5, Section 5.1.3 with translation. The sentences as well as the indications of coverage by Cheetah are taken from Cramer (2011), Appendix A, p. 156.

Sentence	Cheetah	gCLIMB
Es gibt Käse	x	x
Der Käse stinkt	x	x
Antje isst den Käse	x	x
Antje schenkt mir den Käse	x	x
Der Käse ist Käse	x	x
Der Käse ist herrlich	x	x
Antje freut sich auf den Käse	x	x
Antje freut sich auf darauf	x	x
Antje schlägt Käse vor	x	x
Antje denkt dass Käse herrlich ist	x	x
Antje sagt der Käse stinkt	x	x
Antje weiß wo der Käse liegt	x	x
Antje weiß wer stinkt	x	x
Antje weiß was Peter isst	x	x
Antje weiß auf welchen Käse Peter sich freut	x	x
Antje weiß mit welcher Butter Peter das Brot isst	x	x

Bestimmt stinkt der Käse	x	x
Seit gestern stinkt der Käse	x	x
Weil es Käse gibt isst Antje den Käse	x	x
Antje schenkt mir den Käse ohne ihn zu essen	x	x
Den Käse schenkt Antje mir	x	x
Mir schenkt Antje den Käse	x	x
Antje versucht den Käse zu essen	x	x
Den Käse versucht Antje zu essen		x
Bestimmt versucht Antje den Käse zu essen	x	x
Antje will den Käse essen	x	x
Den Käse will Antje essen	x	x
Antje sieht mich den Käse essen	x	x
Mich sieht Antje den Käse essen	x	x
Den Käse sieht Antje mich essen	x	x
Den Käse essen sieht Antje mich	x	x
Antje hat den Käse gegessen	x	x
Der Käse wird von Antje gegessen	x	x
Der Käse soll von Antje gegessen werden	x	x
Der Käse ist von Antje gegessen worden	x	x
Antje hat den Käse zu essen versucht	x	x
Antje hat den Käse versucht zu essen		x
Antje hat versucht den Käse zu essen	x	x
Antje hat den Käse essen wollen	x	x
Peter denkt dass Antje den Käse wird essen können		x
Peter denkt dass Antje den Käse essen können wird	x	x
Der Käse ist herrlicher Käse gewesen	x	x
Isst Antje den Käse	x	x
Wird Antje den Käse essen	x	x
Wo isst Antje den Käse	x	x
Welchen Käse isst Antje	x	x
Wer isst den Käse	x	x
Und der Käse stinkt	x	x
Käse ist Antje nur mit Brot	x	x
Dem Mann zufolge isst Antje den Käse	x	x
Selbst der herrliche Käse von Antje stinkt	x	x
Der gegessene Käse stinkt		x
Der von Antje gegessene Käse stinkt		
Antje der Käse stinkt	x	x
Der Käse des Mannes stinkt	x	x

Frau Antje stinkt	x	x
Dieses Mal isst Antje den Käse	x	x
Antje isst dieses Mal den Käse	x	x
Antje hat keine Ahnung wo der Käse liegt	x	x
Antje hat keine Neigung den Käse zu essen	x	x
Der Kampf um den herrlichen Käse	x	x
Antje hat Sorgen dass der Käse stinkt	x	x
Die 3 liegen	x	
Der Käse den Antje isst stinkt	x	x
Der Käse auf den Antje sich gefreut hat stinkt	x	x
Antje isst den Käse der stinkt	x	x
Das Brot mit dem Antje den Käse isst stinkt	x	x
Der Mann dessen Brot Antje gegessen hat stinkt	x	x
Die Frau deren Brot Antje gegessen hat stinkt	x	x
Die Männer deren Brot Antje gegessen hat stinken	x	x
Der Mann auf dessen Brot Antje sich freut stinkt	x	x
Der Käse ist eine Woche alt		
Das ist sich entwickelender Käse		
Das ist der mir zustehende Käse		
Der Käse ist bestimmt fast unglaublich herrlich		
Der Käse stinkt und das Brot stinkt	x	x
Der Käse stinkt und Antje will den Käse essen	x	x
Stinkt der Käse und will Antje den Käse essen		x
Antje stinkt und will den Käse essen		x
Antje will mir den Käse schenken und das Brot essen	x	x
Bestimmt will Antje mir den Käse schenken und das Brot essen	x	x
Antje und Peter essen den Käse	x	x
Antje isst den Käse und das Brot	x	x
Der Käse und Käse stinkt	x	x
Der Käse liegt bei dem Brot und neben der Butter	x	x
Der Käse ist herrlich und kostbar	x	x
Der herrliche und kostbare Käse stinkt	x	x
Früher oder später wird Antje den Käse essen	x	x
Vier- bis fünfhundert Männer	x	x
Im Kindes- und Jugendalter	x	x
Was denkt Antje dass sie gegessen hat		x
Wer denkt Antje dass den Käse gegessen hat[1]		
Wie denkt Antje dass Peter den Käse gegessen hat		x

[1]According to native speakers, this utterance is not grammatical

Wer sagt Peter hat den Käse gegessen	x	x
Was sagt Peter hat Antje gegessen	x	x
Wie sagt Peter hat Antje den Käse gegessen	x	x
Wir sagt Peter haben den Käse gegessen	x	x
Den Käse sagt Peter haben wir gegessen	x	x
Bestimmt sagt Peter haben wir den Käse gegessen	x	x
Peter hat den Käse gegessen auf den Antje sich gefreut hat	x	x
Peter hat sich auf den Käse gefreut den Antje gegessen hat	x	x
Peter hat die Neigung den Käse zu essen auf den Antje sich freut	x	x
Antje denkt dass dein Käse herrlicher ist als mein Käse	x	x
Antje denkt dass Peter herrlicheren Käse isst als ich	x	x
Antje hat herrlicheren Käse als du gegessen	x	x
Antje hat herrlicheren Käse gegessen als du	x	x

Appendix B

Sentences for Natural Language Generation

Overview of the Dutch sentences used as input for the experiment in Natural Language Generation. All sentences are presented in their canonical word order in standard Dutch. They were created using words that happened to be included in the vocabulary that had the desired syntactic properties.

(48)	Ik	zal	willen	slapen.
	pro.1.sg	will.1.SG	want.INF	sleep.INF

‘I’ll want to sleep.’ [nld]

(49)	Ik	zal	willen	kunnen	slapen.
	pro.1.sg	will.1.SG	want.INF	can.INF	sleep.INF

‘I’ll want to be able to sleep.’ [nld]

(50)	Ik	zal	de	vrouw	willen	kussen.
	pro.1.sg	will.1.SG	the	woman	want.INF	kiss.INF

‘I’ll want to kiss the woman.’ [nld]

(51)	Ik	zal	de	vrouw	willen	kunnen	kussen.
	pro.1.sg	will.1.SG	the	woman	want.INF	can.INF	kiss.INF

‘I’ll want to be able to kiss the woman.’ [nld]

(52)	Ik	zal	de	vrouw	de	brief	willen	sturen.
	pro.1.sg	will.1.SG	the	woman	the	letter	want.INF	send.INF

‘I’ll want to send the woman the letter.’ [nld]

(53)	Ik	zal	de	vrouw	de	brief	willen	kunnen	sturen.
	pro.1.sg	will.1.SG	the	woman	the	letter	want.INF	can.INF	send.INF

‘I’ll want to be able to send the woman the letter.’ [nld]

(54)

Ik	zal	de	vrouw	de	brief	zeker	willen	kunnen	sturen.
pro.1.sg	will.1.SG	the	woman	the	letter	definitely	want.INF	can.INF	send.INF

'I'll definitely want to be able to send the woman the letter.' [nld]

(55)

Ik	zal	de	vrouw	de	brief	zeker	met	het	bier	willen	
pro.1.sg	will.1.SG	the	woman	the	letter	definitely	with	the	beer	want.INF	
kunnen	sturen.										
can.INF	send.INF										

'I'll want to be able to send the woman the letter definitely with the beer.' [nld]

Appendix C

Babel phenomena not handled by either grammar

Overview of phenomena which neither Cheetah or gCLIMB covers or where they both overgenerate.

Phenomena that neither grammar covers
Lexically selected objects
Extraction from NPs (fronted)
“Wer X (der) Y” (*who is doing X, does Y*)
Dative passives
Argument agreement (two arguments must bear the same case)
Partially complement including adpositions
Adverbial use of adjectives
Semi-particles
“Weder...noch” (*neither...nor*)
Phenomena where both grammars overgenerate
Too flexible in fronting (e.g. expletives from embedded clauses)
Extraposition across multiple clauses
Restriction on prepositional form and its case
Word order flexibility in the Mittelfelt
Extraposed relative clauses modifying a subject person name
Lexical selection for fixed expressions

Bibliography

Adger, David. 2003. *Core syntax: A minimalist approach*, volume 33. Oxford University Press Oxford.

Avgustinova, Tania. 2007. *Language Family Oriented Perspective in Multilingual Grammar Design*. Linguistik International: Band 17, Frankfurt am Main, Germany: Peter Lang - Eurpopäischer Verlag der Wissenschaft.

Avgustinova, Tania and Zhang, Yi. 2009. Parallel Grammar Engineering for Slavic Languages. In *Proceedings of GEAF*, Singapore.

Avgustinova, Tania and Zhang, Yi. 2010. Conversion of a Russian dependency treebank into HPSG derivations. In *Proceedings of TLT'9*.

Baldwin, Timothy. 2005. Bootstrapping deep lexical resources: Resources for courses. In *Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition*, pages 67–76, Ann Arbor, USA: Association for Computational Linguistics.

Becker, Tilman. 1990. Meta-rules on Tree Adjoining Grammars. In *Proceedings of the 1st International Workshop on Tree Adjoining Grammars*, Dagstuhl Castle, Germany.

Becker, Tilman, Joshi, Aravind and Rambow, Owen. 1991. Long distance scrambling and tree adjoining grammars. In *Proceedings of the 5th Conference of the European Chapter of the ACL (EACL 1991)*, pages 21–26, Berlin, Germany.

Bender, Emily M. 2007. Combining Research and Pedagogy in the Development of a Crosslinguistic Grammar Resource. In Tracy Holloway King and Emily M. Bender (eds.), *Proceedings of the GEAF07 Workshop*, pages 26–45, Stanford, CA: CSLI.

Bender, Emily M. 2008a. Evaluating a Crosslinguistic Grammar Resource: A Case Study of Wambaya. In *Proceedings of ACL-08: HLT*, pages 977–985, Columbus, Ohio: Association for Computational Linguistics.

Bender, Emily M. 2008b. Grammar Engineering for Linguistic Hypothesis Testing. In *Proceedings of the Texas Linguistics Society X Conference: Computational Linguistics for Less-Studied Languages*, pages 16–36, Stanford: CSLI Publications.

Bender, Emily M. 2010. Reweaving a Grammar for Wambaya: A Case Study in Grammar Engineering for Linguistic Hypothesis Testing. *Linguistic Issues in Language Technology* 3(3), 1–34.

Bender, Emily M. 2014. Language CoLLAGE: Grammatical Description with the LinGO Grammar Matrix. In *Proceedings of LREC 2014*, Reykjavik, Iceland.

Bender, Emily M., Drellishak, Scott, Fokkens, Antske, Poulson, Laurie and Saleem, Safiyyah. 2010. Grammar Customization. *Research on Language & Computation* 8(1), 23–72.

Bender, Emily M. and Flickinger, Dan. 2005. Rapid Prototyping of Scalable Grammars: Towards Modularity in Extensions to a Language-Independent Core. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing IJCNLP-05 (Posters/Demos)*, Jeju Island, Korea.

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan.

Bender, Emily M., Flickinger, Dan and Oepen, Stephan. 2011. Grammar Engineering and Linguistic Hypothesis Testing: Computational Support for Complexity in Syntactic Analysis. In *Language from a Cognitive Perspective: Grammar, Usage and Processing*, pages 5–29, Stanford, USA: CSLI Publications.

Bender, Emily M., Flickinger, Dan, Oepen, Stephan, Walsh, Annemarie and Baldwin, Tim. 2004. Arboretum: Using a precision grammar for grammar checking in CALL. In *Proceedings of the InSTIL/ICALL Symposium 2004*, Venice, Italy.

Bender, Emily M., Ghodke, Sumukh, Baldwin, Timothy and Dridan, Rebecca. 2012. From Database to Treebank: On Enhancing Hypertext Grammars with Grammar Engineering and Treebank Search. In Sebastian Nordhoff and Karl-Ludwig G. Poggeman (eds.), *Electronic Grammaticography*, pages 179–206, Honolulu, USA: University of Hawaii Press.

Bender, Emily M. and Good, Jeff. 2005. Implementation for Discovery: A Bipartite Lexicon to Support Morphological and Syntactic Analysis. In *Edwards, Midtlyng, Sprague and Stensrud*, The Panels, No. 41, Chicago Linguistic Society.

Bender, Emily M., Poulson, Laurie, Drellishak, Scott and Evans, Chris. 2007. Validation and Regression Testing for a Cross-linguistic Grammar Resource. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 136–143, Prague, Czech Republic: Association for Computational Linguistics.

Bergmair, Richard. 2008. Monte Carlo semantics: McPIET at RTE4. In *Text Analysis Conference (TAC 2008) Workshop-RTE-4 Track*, National Institute of Standards and Technology.

Bierwisch, Manfred. 1963. *Grammatik des deutschen Verbs*, volume II of *Studia Grammatica*. Akademie Verlag.

Black, Cheryl A. 2004. Parser And Writer for Syntax, paper presented at the International Conference on Translation with Computer-Assisted Technology: Changes in Research, Teaching, Evaluation, and Practice, University of Rome "La Sapienza", April 2004.

Black, Cheryl A. and Black, H. Andrew. 2009. PAWS: Parser And Writer for Syntax: Drafting Syntactic Grammars in the Third Wave. In *SIL Forum for Language Fieldwork*, volume 2.

Blunsom, Phil and Baldwin, Timothy. 2006. Multilingual Deep Lexical Acquisition for HPSGs via Supertagging. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 164–171, Stroudsburg, PA, USA: Association for Computational Linguistics.

Bond, Francis, Oepen, Stephan, Nichols, Eric, Flickinger, Dan, Velldal, Erik and Haugereid, Petter. 2011. Deep Open-Source Machine Translation. *Machine Translation* 25(2), 87–105.

Borisova, Irina. 2010. *Implementing Georgian Polypersonal Agreement through the LinGO Grammar Matrix*. Masters Thesis, Saarland University.

Botha, Rudolph P. 1970. *The methodological status of grammatical argumentation*. The Hague, the Netherlands: Mouton.

Bouma, Gosse, Malouf, Robert and Sag, Ivan A. 2001. Satisfying constraints on extraction and adjunction. *Natural Language and Linguistic Theory* 19, 1–65.

Branco, AntÃşnio and Costa, Francisco. 2010. A Deep Linguistic Processing Grammar for Portuguese. In *Computational Processing of the Portuguese Language*, volume LNAI6001 of *Lecture Notes in Artificial Intelligence*, pages 86 – 89, Berlin, Germany: Springer.

Brants, Sabine, Dipper, Stefanie, Hansen, Silvia, Lezius, Wolfgang and Smith, George. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol.

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi and Rohrer, Christian. 2002. The Parallel Grammar Project. In John Carroll, Nelleke Oostdijk and Richard Sutcliffe (eds.), *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 1–7.

Butt, Miriam, King, Tracy Holloway, Niño, Maria-Eugenia and Segond, Frédérique. 1999. *A Grammar Writer's Cookbook*. Stanford, USA: CSLI Publications.

Callmeier, Ulrich. 2000. PET - a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6 (1), 99 – 107.

Candito, Marie-Hélène. 1996. A principle-based hierarchical representation of LTAGs. In *Proceedings of the 16th conference on Computational linguistics*, pages 194–199, Kopenhagen, Denmark: Association for Computational Linguistics.

Candito, Marie-Hélène. 1998. Building Parallel LTAG for French and Italian. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 211–217, Montreal, Canada: Association for Computational Linguistics.

Candito, Marie-Hélène. 1999. *Organisation modulaire et paramétrable de grammaires électroniques lexicalisées. Application au francais et á l'italien*. Ph. D.thesis, l'université Paris 7.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theorertical Computer Science, No. 32, New York, USA: Cambridge University Press.

Carroll, John, Copestake, Ann, Flickinger, Dan and Poznański, Victor. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop on Natural Language Generation*, pages 86–95, Toulouse, France.

Carroll, John and Oepen, Stephan. 2005. High efficiency realization for a wide-coverage unification grammar. In *IJCNLP*, Jeju Island, South Korea: Springer-Verlag LNCS.

Cholakov, Kostadin. 2012. *Lexical Acquisition for Computational Grammars*. Ph. D.thesis, Rijksuniversiteit Groningen.

Chomsky, Noam. 1957. *Syntactic Structures*. The Hague, The Netherlands: Mouton.

Chomsky, Noam. 1965. *Aspects of the theory of syntax*. Cambridge, USA: MIT Press.

Chomsky, Noam. 1982. *The Generative Enterprise*. Dordrecht, the Netherlands: Floris Publications.

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, USA: the MIT Press.

Clément, Lionel and Kinyon, Alexandra. 2003. Generating parallel multilingual LFG-TAG grammars with a MetaGrammar. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, Sapporo, Japan: Association of Computational Linguistics.

Coleman, Douglas W. 1999. Assumptions, Hypotheses, and Theories in 'applied' vs. 'theoretical' linguistics. *The LACUS Forum* 25, 461–472.

Coleman, Douglas W. 2001. *Data* and *Science* in introductory linguistics textbooks. *the LACUS Forum* 27, 75–85.

Coleman, Douglas W. 2002. A corpus study on the (non-)physicality of linguistic observations. *The LACUS Forum* 28, 43–50.

Coleman, Douglas W. 2009. There are three kinds of abstractions: abstractions, damned abstractions, and damned lying abstractions. *The LACUS Forum* 35, 109–121.

Copestake, Ann. 2000. Appendix: Definitions of Typed Feature Structures. *Natural Language Engineering* 6, 109–112.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. Stanford, CA: CSLI Publications.

Copestake, Ann, Flickinger, Dan, Sag, Ivan and Pollard, Carl. 2005. Minimal Recursion Semantics. An Introduction. *Journal of Research on Language and Computation* 3(2–3), 281 – 332.

Copestake, Ann, Lascarides, Alex and Flickinger, Dan. 2001. An Algebra for Semantic Construction in Constraint-based Grammars. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 140–147.

Cowart, Wayne. 1997. *Experimental Syntax: Applying Objective Methods to Sentence Judgments*. SAGE Publications.

Crabbé, Benoît and Duchier, Denys. 2004. Metagrammar Redux. In *Proceedings of the International Workshop on Constraint Solving and Language Processing*, Roskilde, Denmark.

Cramer, Bart. 2011. *Improving the feasibility of precision-oriented HPSG parsing*. Ph. D.thesis, Saarland University.

Cramer, Bart and Zhang, Yi. 2009. Constructon of a German HPSG grammar from a detailed treebank. In *Proceedings of the ACL 2009 Grammar Engineering across Frameworks workshop*, pages 37–45, Singapore, Singapore.

Crouch, Dick, Dalrymple, Mary, Kaplan, Ronald M., King, Tracy Holloway, Maxwell III, John T. and Newman, Paula. 2011. XLE Documentation.

Crowgey, Joshua. 2012. An a priori Typology of Sentential Negation from an HPSG Perspective. In Stefan Müller (ed.), *Proceedings of the 19th International Conference on Head-Driven Phrase Structure Grammar, Chungnam National University Daejeon*, pages 107–122.

Crysmann, Berthold. 2005. Relative Clause Extraposition in German: An Efficient and Portable Implementation. *Research on Language and Computation* 3(1), 61–82.

Crysmann, Berthold. 2012. HaG – a computational grammar of Hausa. In Tristan Purvis and Michael Marlo (eds.), *Selected Proceedings of the 42nd Annual Conference on African Linguistics*.

Crysmann, Berthold, Bertomeu, Nuria, Adolphs, Peter, Flickinger, Dan and Klüwer, Tina. 2008. Hybrid processing for grammar and style checking. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, Manchester, UK.

Dalrymple, Mary. 2001. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. New York, USA: Academic Press.

Den Besten, Hans and Edmondson, Jerold A. 1983. The Verbal Complex in Continental West Germanic. In Werner Abraham (ed.), *On the Formal Syntax of the Westgermania: Papers from the III Groninger Grammar Talks*, Amsterdam, The Netherlands: Benjamins.

DeWitt, Bryce Seligman. 1972. The Many-Universes Interpretation of Quantum Mechanics. In *Proceedings of the International School of Physics "Enrico Fermi" Course IL: Foundations of Quantum Mechanics*.

Diderichsen, Paul. 1962. *Elementar Dansk Grammatik*. Copenhagen, Denmark: Gyldendal.

Drach, Erich. 1937. *Grundgedanken der Deutschen Satzlehre*. Frankfurt am Main, Germany: Diesterweg.

Drellishak, Scott. 2004. *A Survey of Coordination Strategies in the WorldâĂŹs Languages*. Masters Thesis, University of Washington.

Drellishak, Scott. 2009. *Widespread but Not Universal: Improving the Typological Coverage of the Grammar Matrix*. Ph. D.thesis, University of Washington.

Drellishak, Scott and Bender, Emily M. 2005. A Coordination Module for a Crosslinguistic Grammar Resource. In Stefan Müller (ed.), *The Proceedings of the 12th International*

Conference on Head-Driven Phrase Structure Grammar, Department of Informatics, University of Lisbon, pages 108–128, Stanford: CSLI Publications.

Dridan, Rebecca. 2009. *Using Lexical Statistics to Improve HPSG Parsing*. Ph. D.thesis, Saarland University.

Duchier, Denys, Le Roux, Joseph and Parmentier, Yannick. 2005. The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture. In Peter Van Roy (ed.), *Multiparadigm Programming in Mozart/OZ*, volume 3389 of *Lecture Notes in Computer Science*, pages 175–187, Heidelberg/Berlin, Germany: Springer.

Eddington, David. 2008. Linguistics and the Scientific Method. *Southwest Journal of Linguistics* 27, 1–16.

Erdmann, Oskar. 1886. *Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung dargestellt. Erste Abteilung*. Stuttgart, Germany: Verlag der Cotta'schen Buchhandlung.

Everett, H. 1957. "Relative State" Formulation of Quantum Mechanics. *Reviews of Modern Physics* 29, 454–462.

Flickinger, Dan. 1987. *Lexical Rules in the Hierarchical Lexicon*. Ph. D.thesis, Stanford University.

Flickinger, Dan. 2000. On Building a More Efficient Grammar by Exploiting Types. *Natural Language Engineering* 6 (1), 15 – 28.

Fokkens, Antske. 2011a. Metagrammar engineering: Towards systematic exploration of implemented grammars. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1066–1076, Portland, Oregon, USA: Association for Computational Linguistics.

Fokkens, Antske. 2011b. Thesis Proposal. Unpublished manuscript.

Fokkens, Antske. 2012. SHORT-CLIMB Documentation.

Fokkens, Antske and Avgustinova, Tania. 2013. SlaviCLIMB: Combining exp ertise for Slavic grammar development using a metagrammar. In Denys Duchier and Yannick Parmentier (eds.), *Proceedings of the Workshop on High-Level Methodologies for Grammar Engineering at ESSLLI 2013*, Düsseldorf, Germany.

Fokkens, Antske, Avgustinova, Tania and Zhang, Yi. 2012a. CLIMB grammars: three projects using metagrammar engineering. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk and

Stelios Piperidis (eds.), *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, Instanbul, Turkey: European Language Resources Association (ELRA).

Fokkens, Antske and Bender, Emily M. 2013. Time Travel in Grammar Engineering. Using a Metagrammar to Broaden the Search Space. In Denys Duchier and Yannick Parmentier (eds.), *Proceedings of the Workshop on High-Level Methodologies for Grammar Engineering at ESSLLI 2013*, Düsseldorf, Germany.

Fokkens, Antske, Bender, Emily M. and Gracheva, Varya. 2012b. LinGO Grammar Matrix Customization System Documentation.

Fokkens, Antske, Poulson, Laurie and Bender, Emily M. 2009. Inflectional morphology in Turkish VP-coordination. In Stefan Müller (ed.), *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar*, pages 110–130, Stanford, CA: CSLI Publications ONLINE.

Fokkens, Antske, Zhang, Yi and Bender, Emily. 2011. Spring Cleaning and Grammar Compression: Two Techniques for Detection of Redundancy in HPSG grammars. In *Proceedings of the 25th PACLIC*, Singapore, Singapore.

Fokkens, Antske S. 2010. Documentation for the Grammar Matrix word order library. Technical Report, Saarland University.

Frank, Annette, Krieger, Hans-Ulrich, Xu, Feiyu, Uszkoreit, Hans, Crysmann, Berthold, Jörg, Brigitte and Schäfer, Ulrich. 2007. Question answering from structured knowledge sources. *Journal of Applied Logic* 5, 20–48, special Issue on Questions and Answers: Theoretical and Applied Perspectives.

Frege, Gottlob. 1892. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik* 100, 25–50.

Gaiffe, Bertrand, Crabbé, Benoît and Roussanaly, Azim. 2002. A new metagrammar compiler. In *Proceedings of TAG+6*, Venice, Italy.

Gazdar, Gerald. 1987. Unbounded dependencies and coordinate structure. In *The Formal complexity of natural language*, pages 183–226, Springer.

Gazdar, Gerald, Klein, Ewan, Pullum, Geoffrey K. and Sag, Ivan A. 1985. *Generalized Phrase Structure Grammar*. Cambridge, USA: Harvard University Press.

Gerdes, Kim. 2002. DTAG. attempt to generate a useful TAG for German using a metagrammar. In *Proceedings of TAG+6*, Venice, Italy.

Ginzburg, Jonathan and Sag, Ivan. 2000. *Interrogative Investigations: The form, meaning, and use of English interrogatives*. Stanford, USA: CSLI.

Goodman, Michael Wayne. 2012. Generation of Machine-Readable Morphological Rules from Human-Readable Input.

Haeseryn, Walter. 1997. De gebruikswaarde van de ANS voor tekstschrijvers, taaltrainers en taaladviseurs. *Tekst[blad]* 3.

Hansson, Sven Ove. 2012. Science and Pseudo-Science. In Edward N. Zalta (ed.) (ed.), *The Stanford Encyclopedia of Philosophy*.

Haugereid, Petter. 2011. A grammar design accommodating packed argument frame information on verbs. In *Proceedings of the 25th PACLIC*, pages 31–40.

Heinz, Wolfgang and Matiasek, Johannes. 1994. Argument structure and case assignment in German. In John Nerbonne, Klaus Netter and Carl Pollard (eds.), *German in HPSG*, pages 199–236, Stanford, USA: CSLI.

Hellan, Lars, Bruland, Tore, Sandøy, Mads, Aamot, Elias, Beermann, Dorothee and Flickinger, Dan. 2011. A Grammar Sparrer for Norwegian. Technical Report, NTNU Trondheim, Trondheim, Norway.

Hellan, Lars and Haugereid, Petter. 2003. NorSource: An exercise in Matrix grammar-building design. In Emily M. Bender, Dan Flickinger, Frederik Fouvry and Melanie Siegel (eds.), *Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Development*, ESSLLI 2003, pages 41–48, Vienna, Austria.

Hinrichs, Erhard and Nakazawa, Tsuneko. 1989. *Flipped out: AUX in German*, volume 25. Chicago, USA: Chicago Linguistic Society.

Hinrichs, Erhard and Nakazawa, Tsuneko. 1994. Linearizing AUXs in German Verbal Complexes. In John Nerbonne, Klaus Netter and Carl Pollard (eds.), *German in HPSG*, Chapter 1, Stanford, USA: CSLI.

Hock, Hans Henrich and Joseph, Brian D. 1996. *Language history, language change, and language relationship: An introduction to historical and comparative linguistics*. Berlin, Germany: Mouton de Gruyter.

Itkonen, Esa. 1978. *Grammatical theory and metascience: A critical investigation into the methodological and philosophical foundations of 'autonomous' linguistics*. Amsterdam, the Netherlands: John Benjamins.

Itkonen, Esa. 1981. The concept of linguistic intuition. In Florian Coulmas (ed.), *A Festschrift for Native Speaker*, pages 127–140, The Hague, the Netherlands: Mouton.

Jacobs, Joachim. 1986. The Syntax of Focus and Adverbials in German. In W. Abraham and S. de Meij (eds.), *Topic, Focus, and Configurationality. Papers from the 6th Groningen Grammar Talks, Groningen, 1984*, pages 103–127, Amsterdam, the Netherlands: John Benjamins Publishing.

Joshi, Aravind and Schabes, Yves. 1991. Tree-Adjoining Grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski (eds.), *Definability and Recognizability of Sets of Trees*, Elsevier.

Joshi, Aravind and Schabes, Yves. 1997. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa (eds.), *Handbook of Formal Languages*, volume 3, pages 69–124, Berlin, Germany: Springer.

Kaplan, Ronald and Bresnan, Joan. 1982. Lexical Functional Grammar: A formal system for grammatical representation. In Joan Bresnan (ed.), *The Mental Representation of Grammatical Relations*, pages 173–âĂŞ281, Cambridge, USA: the MIT Press.

Kathol, Andreas. 2000. *Linear Syntax*. New York, USA: Oxford University Press.

Kay, Martin. 1979. Functional Grammar. In Christine Chiarello (ed.), *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, Berkeley, USA.

Kay, Martin. 1984. Functional unification grammar: a formalism to machine translation. In *Proceedings of the 10th International Conference on Computational Lingusitics*, pages 75–78, Stanford, USA.

Keller, Frank. 2000. *Gradience in Grammar: Experimental and Computational Aspects of Degrees of Grammaticality*. Ph. D.thesis, University of Edinburgh.

Kim, Jong Bok and Yang, Jaehyung. 2003. Korean Phrase Structure Grammar and Its Implementations into the LKB System. In Don Hong Ji and Kim Teng Lua (eds.), *Proceedings of the 17th Asia Pacific Conference*, pages 88–97, COLIPS Publications.

King, Tracy Holloway, Forst, Martin, Kuhn, Jonas and Butt, Miriam. 2005. The Feature Space in Parallel Grammar Writing. *Research on Language and Computation, Special Issue on Shared Representations in Multilingual Grammar Engineering* 3(2), 139–163.

Kinyon, Alexandra, Rambow, Owen, Scheffler, Tatjana, Yoon, SinWon and Joshi, Aravind K. 2006. The Metagrammar Goes Multilingual: A Cross-Linguistic Look at the V2-Phenomenon. In *Proceedings of the 8th International Workshop on Tree Adjoining*

Grammar and Related Formalisms, pages 17–24, Sydney, Australia: Association for Computational Linguistics.

Kiss, Tibor and Wesche, Birgit. 1991. Verb Order and Head Movement. In O. Herzog and C.-R Rollinger (eds.), *Text Understanding in LILOG*, pages 216–242, Berlin, Germany: Springer-Verlag.

Kordoni, Valia and Neu, Julia. 2005. Deep analysis of Modern Greek. In Keh-Yih Su, Jun'ichi Tsujii and Jong-Hyeok Lee (eds.), *Lecture Notes in Computer Science*, volume 3248, pages 674–683, Berlin, Germany: Springer.

Koster, Jan. 2005. Is linguistics a natural science. In Hans Broekhuis, Norbert Corver, Riny Huybregts, Ursula Kleinhenz and Jan Koster (eds.), *Organizing Grammar: Linguistic Studies in Honor of Henk van Riemsdijk*, pages 350–358, Mouton de Gruyter.

Krieger, Hans-Ulrich and Schäfer, Ulrich. 1994. TDL - A Type Description Language for constraint-based grammarsanguage for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 893–899, Kyoto, Japan.

Kuhn, Thomas S. 1974. Logic of Discovery or Psychology of Research? In Paul A. Schilpp (ed.), *The Philosophy of Karl Popper*, volume xiv of *The Library of Living Philosophers*, pages 798–819, La Salle: Open Court.

Lakatos, Imre. 1970. Falsification and the Methodology of Research program. In Imre Lakatos and Alan Musgrave (eds.), *Criticism and the Growth of Knowledge*, pages 91–197, Cambridge, UK: Cambridge University Press.

Lakatos, Imre. 1974a. Popper on Demarcation and Induction. In Paul A. Schilpp (ed.), *The Philosophy of Karl Popper*, volume xiv of *The Library of Living Philosophers*, pages 241–273, La Salle: Open Court.

Lakatos, Imre. 1974b. Science and pseudoscience. *Conceptus* 8, 5–9.

Lakatos, Imre. 1981. Science and pseudoscience. In Stuart C. Brown, John Fauvel and Ruth H. Finnegan (eds.), *Conceptions of Inquiry: A Reader*, pages 114–121, London, UK: Methuen.

Larson, Richard K. 2010. *Grammar as Science*. London, UK: the MIT Press.

Lasswell, Steven Theophilos. 1998. *An Ecological Reference Grammar of Solring North Frisian (Germany)*. Ph. D.thesis, University of California, Santa Barbara.

Lehmann, Sabine. 2000. *Towards a Theory of Syntactic Phenomena. SaarbrÃijcken Dissertations in Computational Linguistics and Language Technology, Volume 11*. Ph. D.thesis, Universität des Saarlandes, Fachbereich Computerlinguistik, Saarbrücken.

Lipenkova, Janna. 2008. *Serienverben im Chinesischen und ihre Analyse im Formalismus der HPSG*. Masters Thesis, Freie Universität Berlin.

Malouf, Robert. 2000. Efficient feature structure operations without compilation. *Natural Language Engineering* 6 (1), 29 – 46.

Marimon, Montserrat. 2010. The Spanish Resource Grammar. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner and Daniel Tapias (eds.), *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, Valetta, Malta: European Language Resources Association (ELRA).

Marimon, Montserrat, Bel, Núria, Espeja, Sergio and Seghezzi, Natalia. 2007. The spanish resource grammar: pre-processing strategy and lexical acquisition. In *Proceedings of the Workshop on Deep Linguistic Processing*, pages 105–111, Association for Computational Linguistics.

Martin-Löf, Per. 1984. *Intuitionistic Type Theory*. Napoli, Italy: Bibliopolis.

Maxwell III, John T. and Kaplan, Ron. 1993. The interface between phrasal and functional constraints. *Computational Lingusitics* 19, 571–589.

McConnel, Stephen. 1995. *PC-PATR Reference Manual*. Summer Institute for Linguistics.

MelÊźčuk, IgorÊź Aleksandrovič. 1988. *Dependency syntax: theory and practice*. State University of New York Press.

Meurers, Detmar, Penn, Gerald and Richter, Frank. 2002. A Web-Based Instructional Platform for Constraint- Based Grammar Formalisms and Parsing. In D. Radev and C. Brew (eds.), *Effective Tools and Methodologies for Teaching NLP and CL*, pages 18–25.

Montague, Richard. 1974. *Formal Philosophy. Collected Papers edited by Richmond Thomason*. New Haven, USA: Yale University Press.

Moshier, Drew. 1988. *Extensions to Unification Grammars for the Description of Programming Languages*. Ph. D.thesis, University of Michigan, Ann Arbor, USA.

Moshier, M. Andrew. 1997a. Featureless HPSG. In Peter Blackburn and Maarten de Rijke (eds.), *Specifying Syntactic Structures*, pages 115–155, Stanford, USA: CSLI Publications.

Moshier, M. Andrew. 1997b. Is HPSG Featureless or Unprincipled? *Linguistics and Philosophy* 20, 669–695.

Müller, Stefan. 1996. The Babel-System—An HPSG Fragment for German, a Parser, and a Dialogue Component. In *Proceedings of the Fourth International Conference on the Practical Application of Prolog*, pages 263–277, London.

Müller, Stefan. 1997. Yet Another Paper about Partial Verb Phrase Fronting in German. Technical Report, DFKI, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

Müller, Stefan. 1999. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Tübingen: Max Niemeyer Verlag.

Müller, Stefan. 2002. *Complex Predicates: Verbal Complexes, Resultative Constructions, and Particle Verbs in German*, volume 13 of *Studies in Constraint-Based Lexicalism*. Stanford, USA: CSLI Publications.

Müller, Stefan. 2004. An Analysis of Depictive Secondary Predicates in German without Discontinuous Constituents. In Stefan Müller (ed.), *Proceedings of the HPSG-2004 Conference, Center for Computational Linguistics, Katholieke Universiteit Leuven*, pages 202–222, Stanford: CSLI Publications.

Müller, Stefan. 2007. *Head-Driven Phrase Structure Grammar: Eine Einführung*. Stauffenburg Einführungen, No. 17, Tübingen: Stauffenburg Verlag.

Müller, Stefan. 2009a. On Predication. In *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar, University of Göttingen, Germany*, pages 213–233, Stanford, USA: CSLI Publications.

Müller, Stefan. 2009b. Towards an HPSG Analysis of Maltese. In Bernard Comrie, Ray Fabri, Beth Hume, Manwel Mifsud, Thomas Stolz and Martine Vanhove (eds.), *Introducing Maltese linguistics. Papers from the 1st International Conference on Maltese Linguistics (Bremen/Germany, 18–20 October, 2007)*, volume 113 of *Studies in Language Companion Series*, pages 83–112, Amsterdam, Philadelphia: John Benjamins Publishing Co.

Müller, Stefan. 2010. Persian Complex Predicates and the Limits of Inheritance-Based Analyses. *Journal of Linguistics* 46(3), 601–655.

Müller, Stefan. 2013. The CoreGram Project: A Brief Overview and Motivation. In Denys Duchier and Yannick Parmentier (Eds) (eds.), *Proceedings of the Workshop on High-level Methodologies for Grammar Engineering (HMGE 2013)*, Düsseldorf, Germany.

Müller, Stefan, Bildhauer, Felix and Cook, Philippa. 2012. Beschränkungen für die scheinbar mehrfache Vorfeldbesetzung im Deutschen. In Colette Cortès (ed.), *Satzeröffnung. Formen, Funtionen, Strategien*, Eurogermanistik, No. 31, pages 113–128, Tübingen: Stauffenburg Verlag.

Müller, Stefan and Ghayoomi, Masood. 2010. PerGram: A TRALE Implementation of an HPSG Fragment of Persian. In *Proceedings of 2010 IEEE International Multiconference on Computer Science and Information Technology*, pages 461âĂŞ–467, Polish Information Processing Society.

Müller, Stefan and Kasper, Walter. 2000. HPSG analysis for German. In Wolfgang Wahlster (ed.), *Verbmobil: Foundations of Speech-to-Speech translation*, pages 238 – 253, Berlin, Germany: Springer.

Müller, Stefan and Lipenkova, Janna. 2009. Serial Verb Constructions in Chinese: An HPSG Account. In Stefan Müller (ed.), *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar, University of Göttingen, Germany*, pages 234–254, Stanford, USA: CSLI Publications.

Müller, Stefan and Ørsnes, Bjarne. 2011. Positional Expletives in Danish, German, and Yiddish. In Stefan Müller (ed.), *The Proceedings of the 18th International Conference on Head-Driven Phrase Structure Grammar*, pages 167–187, Stanford, USA: CSLI Publications.

Müller, Stefan and Ørsnes, Bjarne. to appear. Danish in Head-Driven Phrase Structure Grammar. Ms, Freie Universität Berlin.

Netter, Klaus. 1992. On Non-Head Non-Movement. An HPSG Treatment of Finite Verb Position in German. In G. Görz (ed.), *Konvens 92. 1. Konferenz „Verarbeitung natürlicher Sprache"*, pages 218–227, Nürnberg, Germany: Springer-Verlag.

Nichols, Eric, Bond, Francis, Appling, Darren Scott and Matsumoto, Yuji. 2010. Paraphrasing Training Data for Statistical Machine Translation. *Journal of Natural Language Processing* 17(3), 101–122.

Nichols, Eric, Bond, Francis, Tanaka, Takaaki, Fujita, Sanae and Flickinger, Dan. 2006. Multilingual ontology acquisition from multiple MRDs. In *Proceedings of the 2nd Workshop on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, pages 10–âĂŞ17, Sydney, Australia: Association for Computational Linguistics.

Nordlinger, Rachel. 1998. *A Grammar of Wambaya, Northern Territory (Australia)*. Canberra, Australia: Pacific Linguistics.

Oepen, Stephan. 2001. [incr tsdb()] — Competence and Performance Laboratory. User Manual. Technical report, Saarland University, Saarbrücken, Germany.

Oepen, Stephan and Carroll, John. 2000. Parser engineering and performance profiling. *Natural Language Engineering* 6 (1), 81 – 97.

Oepen, Stephan, Velldal, Erik, Lønning, Jan Tore, Meurer, Paul, Rosén, Victoria and Flickinger, Dan. 2007. Towards hybrid quality-oriented machine translation – on linguistics and probabilities in MT. In *Proceedings of the 11th Conference on Theoretical and Methodological Issues in Machine Translation (TMI-07)*, Skövde, Sweden.

O'Hara, Kelly. 2008. *A Morphotactic Infrastructure for a Grammar Customization System*. Masters Thesis, University of Washington.

Oliva, Karel. 1992. Word Order Constraints in Binary Branching Syntactic Structures. Technical Report, Universität des Saarlandes, Saarbrücken, Germany, cLAUS-Report 20.

Ørsnes, Bjarne. 2009a. Das Verbalfeldmodell âĂŞ ein Stellungsfeldermodell für den kontrastiven DaF-Unterricht. *Deutsch als Fremdsprache* 46(3), 143âĂŞ–149.

Ørsnes, Bjarne. 2009b. Preposed Sentential Negation in Danish - an HPSG Approach. In Stefan Müller (ed.), *Proceedings of the international HPSG Conference 2009, University of Göttingen, Germany*, pages 255–275, Stanford, USA: CSLI Publications.

Osenova, Petya. 2010. BUlgarian Resource Grammar âĂŞ EfiňĄcient and Realistic (BURGER).

Packard, Woodley. 2011. ACE, the Answer Constraint Engine. `http://sweaglesw.org/linguistics/ace/`, accessed, 9 August 2012.

Penn, Gerald. 2000. *The algebraic structure of attributed type signatures*. Ph. D.thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA.

Penn, Gerald. 2004. Balancing Clarity and Efficiency in Typed Feature Logic Through Delaying. In D. Scott (ed.), *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACLÕ04)*, pages 239–246, Barcelona, Spain.

Pereira, Fernando and Warren, David. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13, 231–278.

Pollard, Carl. 1994. Toward a Unified Account of Passives in German. In John Nerbonne, Klaus Netter and Carl Pollard (eds.), *German in HPSG*, Chapter 8, Stanford, USA: CSLI.

Pollard, Carl and Moshier, Drew. 1990. *Unifiying partial descriptions of sets*, volume 1 of *Vancouver Studies in Cognitive Science*. University of British Colombia Press, information, language and cognition edition.

Pollard, Carl and Sag, Ivan. 1987. *Information-Based Syntax and Semantics, Volume 1: Fundamentals*, volume 13 of *CSLI Lecture Notes*. Stanford, USA: CSLI/Chicago Press.

Pollard, Carl and Sag, Ivan. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, USA: University of Chicago Press.

Popper, Karl. 1962. *Conjectures and refutations. The growth of scientific knowledge*. New York, USA: Basic Books.

Poulson, Laurie. 2011. Meta-modeling of Tense and Aspect and in a Cross-linguistic Grammar Engineering Platform. *University of Washington working papers in linguistics* 28, 1–67.

Przepiórkowski, Adam. 1996. Case assignment in Polish: towards an HPSG analysis. In Claire Grover and Enric Vallduví (eds.), *Studies in HPSG*, Edinburgh Working Papers in Cognitive Science, No. 12, pages 191–228, Edinburgh, Centre for Cognitive Science.

Ranta, Aarne. 1994. *Type Theoretical Grammar*. Oxford, UK: Oxford University Press.

Ranta, Aarne. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming* 14(2), 145–189.

Ranta, Aarne. 2009. The GF Resource Grammar Library. *Linguistic Issues in Language Technology* 2(2).

Ranta, Aarne. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. Stanford, USA: CSLI Publications.

Reape, Mike. 1993. *A Formal Theory of Word Order: A Case Study in West Germanic*. Ph. D.thesis, University of Edinburgh.

Reape, Mike. 1994. Domain union and word order in variation in German. In John Nerbonne, Klaus Netter and Carl J. Pollard (eds.), *German in Head-Driven Phrase Structure Grammar*, pages 151–198, Stanford, USA: CSLI Publications.

Riezler, Stefan, King, Tracy Holloway, Kaplan, Ronald M., Crouch, Richard, III, John T. Maxwell and Johnson, Mark. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of ACL*.

Ross, John R. 1967. *Constraints on Variables in Syntax*. Ph. D.thesis, MIT.

Rupp, CJ, Copestake, Ann, Corbett, Peter and Waldron, Benjamin. 2007. Integrating general-purpose and domain-specific components in the analysis of scientific text. In *Proceedings of the UK e-Science Programme All Hands Meeting (AHM2007)*, pages 446–453, Nottingham, UK.

Sag, Ivan. 1997. English Relative Clause Constructions. *Journal of Linguistics* 33(2), 431–484.

Sag, Ivan A. and Wasow, Thomas. 1999. *Syntactic Theory: A formal introduction*. Stanford, USA: CSLI Publications.

Sag, Ivan A., Wasow, Thomas and Bender, Emily M. 2003. *Synactic Theory: A Formal Introduction*. Stanford, CA: CSLI Publications, second edition.

Saleem, Safiyyah. 2010. *Argument Optionality: A New Library for the Grammar Matrix Customization System*. Masters Thesis, University of Washington.

Saussure, Ferdinand de. 1916. *Cours de linguistique générale*. Paris, France: Payot.

Schneider, David and McCoy, Kathleen. 1998. Recognizing syntactic errors in the writing of second language learners. In *Proceedings of Coling-ACL'98*, pages 1198–1204, Montreal, Canada.

Scholz, Barbara C. and Pullum, Geoffrey K. 2006. Irrational nativist exuberance. In Robert Stainton (ed.), *Contemporary Debates in Cognitive Science*, pages 59–80, Oxford, UK: Basil Blackwell.

Schütze, Carson. 1996. *The empirical base of linguistics: Grammaticality judgments and linguistic methodology*. University of Chicago Press.

Schütze, Carson. 2005. Thinking about what we are asking speakers to do. In Stephan Kepser and Marga Reis (eds.), *Linguistic evidence: Empirical, theoretical, and computational perspectives*, pages 457–485, Berlin, Germany: Mouton de Gruyter.

Shieber, Stuart M. 1984. The design of computer language for linguistic information. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 362–366, Stanford, USA.

Shieber, Stuart M. 1986. *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes Series*. Stanford, USA: CSLI publications.

Shieber, Stuart M., Uszkoreit, Hans, Pereira, Fernando, Robinson, Jane and Tyson, Mabry. 1983. The formalism and implementation of PART-II. In *Research on Interactive Acquisition and Use of Knowledge*, volume 1894 of *SRI Final Report*, Menlo Park, USA: SRI International.

Siegel, Melanie and Bender, Emily M. 2002. Efficient Deep Processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization at the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Siegel, Melanie and Bender, Emily M. 2004. Head-Initial Constructions in Japanese. In Stefan Müller (ed.), *Proceedings of the 11th International Conference on Head-Driven Phrase Structure Grammar*, pages 244–260, Stanford, USA: CSLI Publications.

Slayden, Glenn. 2012. *Array TFS Storage for Unification Grammars*. Masters Thesis, University of Washington.

Song, Sanghoun. 2014. *A Grammar Library for Information Structure*. Ph. D.thesis, University of Washington.

Song, Sanghoun and Bender, Emily M. 2012. Individual Constraints for Information Structure. In Stefan Müller (ed.), *The 19th International Conference on Head-Driven Phrase Structure Grammar*, Daejeon, Korea.

Song, Sanghoun, Kim, Jong-Bok, Bond, Francis and Yang, Jaehyung. 2010. Development of the Korean Resource Grammar: Towards Grammar Customization. In *The 8th Workshop on Asian Language Resources (in conjunction with COLING2010)*, Beijing, China.

Suppes, P., Flickinger, D., Macken, B., Cook, J. and Liang, T. 2012. Description of the EPGY Stanford University Online Courses for Mathematics and Language Arts. In *International Society for Technology in Education Annual (ISTE) 2012 Conference*, San Diego CA.

Sygal, Yael. 2011. *Modular Development of Typed Unification Grammars*. Ph. D.thesis, University of Haifa PhD thesis.

Sygal, Yael and Wintner, Shuly. 2011. Towards Modular Development of Typed Unification Grammars. *Computational Linguistics* 37(1), 29–74.

Tseng, Jesse. 2003. LKB Grammar Implementation: French and beyond. In *Proceedings of the Workshop on Ideas and Strategies for Multilingual Grammar Development*, ESSLLI 2003, pages 91–97, Vienna, Austria.

Uszkoreit, Hans. 1987. *Word Order and Constituent Structure in German*. Stanford, USA: CSLI Publications.

Vijay-Shanker, K. and Schabes, Yves. 1992. Structure sharing in Lexicalized Tree-Adjoining Grammars. In *Proceedings of the 14th conference on Computational linguistics*, COLING '92, pages 205–211, Stroudsburg, USA: Association for Computational Linguistics.

Wang, Rui, Osenova, Petya and Simov, Kiril. 2012. Linguistically-Augmented Bulgarian-to-English Statistical Machine Translation Model. In *Proceedings of the EACL 2012 Joint Workshop on Exploiting Synergies between Information Retrieval and Machine Translation (ESIRMT) and Hybrid Approaches to Machine Translation (HyTra)*, Avignon, France.

Wasow, Thomas. 1985. The Wizards of Ling. *Natural Language & Linguistic Theory* 3(4).

Weber, David J., Black, H. Andrew and McConnel, Stephen R. 1988. AMPLE: A Tool for Exploring Morphology. In *Occasional Publications in Academic Computing*, volume 12, Dallas, USA: Summer Institute of Linguistics.

Welin, Carl Wilhelm. 1979. *Studies in Computational Text Comprehension*, volume 5 of *Milus Monograph*. Stockholm: Sweden: Institute for Linguistics.

Wilske, Sabrina and Wolska, Magdalena. 2011. Meaning versus Form in Computer-assisted Task-based Language Learning: A Case Study on the German Dative. *Journal Language Technology and Computational Linguistics (JLCL)* 26(1), 23–37.

Wintner, Shuly, Lavie, Alon and MacWhinney, Brian. 2009. Formal Grammars of Early Language. *Languages: from Formal to Natural* 5533, 204–227.

Wolska, Magdalena and Wilske, Sabrina. 2010. German subordinate clause word order in dialogue-based CALL. In *Proceedings of the Computational Linguistics - Applications (CLA-10) Section of the International Multiconference on Computer Science and Information Technology*, pages 553–559, Wisła, Poland.

Yngve, Victor H. 1986. *Linguistics as a science*. Bloomington, USA: Indiana University Press.

Yngve, Victor H. 1996. *From grammar to science: New foundations for general linguistics*. Philadelphia, USA: John Benjamins.

Zhang, Yi. 2007. *Robust Deep Linguistic Processing*. Ph. D.thesis, Saarland University, Saarbrücken, Germany.

Zhang, Yi, Oepen, Stephan and Carroll, John. 2007. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th international conference on parsing technologies (IWPT 2007)*, pages 48–59, Prague, Czech Republic.

Zhang, Yi, Wang, Rui and Chen, Yu. 2011. Engineering a Deep HPSG for Mandarin Chinese. In *Proceedings of ALR*, Chiang Mai, Thailand.