A Language for the Statement of Binary Relations over Feature Structures

Graham Russell Afzal Ballim Dominique Estival Susan Warwick-Armstrong
ISSCO, 54 rte. des Acacias
1227 Geneva, Switzerland
elu@divsun.unige.ch

Abstract

Unification is often the appropriate method for expressing relations between representations in the form of feature structures; however, there are circumstances in which a different approach is desirable. A declarative formalism is presented which permits direct mappings of one feature structure into another, and illustrative examples are given of its application to areas of current interest.

1. Introduction

Benefits arising from the adoption of unification as a tool in computational linguistics are well known: a declarative, monotonic method of combining partial information expressed in data structures convenient for linguistic applications permits the writing of sensible grammars that can be made independent from processing mechanisms, and a growing familiarity, in both theoretical and computational circles, with the techniques of unification fosters fruitful interchange of ideas and experiences. There are, however, occasions when unification alone is not an appropriate tool. In essence, unification is a ternary relation in which two structures, when merged, form a third; it is less attractive in circumstances where the relation to be expressed is binary – when one would like to manipulate a single feature structure (FS), perhaps simulating the direct transformation of one FS into another.[1] The present paper introduces a declarative formalism intended for the expression of such relations, and shows how it may be applied to some areas of current interest.

The formalism in question is based upon a notion of 'transfer rule'; informally, a set of such rules may be considered as characterizing a binary relation over a set of feature structures, the properties of that relation depending on the content of the particular rule set in use. Transfer rules associate the analysis of one FS with the synthesis of another; they may be thought of as a specialized variety of pattern-matching rule. They are local in nature, and permit the recursive analysis and synthesis of complex structures according to patterns specified in a format closely related to that widely employed in unification-based computational linguistics. Indeed, the interpretation of transfer rules involves unification, albeit in a context which restricts it to the role of a structure-building operation.[2]

In the remainder of this paper we provide a brief specification of the transfer rule formalism, discuss its interpretation, outline two alternative rule application regimes, and illustrate the use of the formalism in the areas of machine translation and reduction of FSs to canonical form. We conclude with an overview of continuing strands of research.

2. Rule Format and Interpretation

2.1. General Remarks

A transfer rule consists of four parts:

(i) a rule name;[3]
(ii) a set of constraint equations describing a FS;
(iii) a set of constraint equations describing a FS;[4]

We are indebted to Jacques Jayez for comments on an earlier draft of this paper.

[1] Clearly there is a sense in which such relations can be viewed as ternary: $T(F_1, R, F_2)$, where F_1 and F_2 are FSs, and R is the rule set which relates them.

[2] The rule formalism is thus monotonic, being unable to effect changes in the input representation, and constructing the output by means of unification.

[3] The rule name plays no part in the interpretation of rules, but provides a convenient reference for tracing their ordering and application.

[4] The equations in each of (ii) and (iii) must be uniquely rooted. The current implementation disallows disjunction in the equation sets for this reason.

(iv) a (possibly empty) set of 'transfer correspondence statements' – equations describing transfer correspondences that must hold between variable bindings established in (ii) and (iii).

A transfer rule relates the two FSs it describes either directly or indirectly, via the rule's transfer correspondence statements; in order for the relation to hold between the source and destination FS, it must hold between the FSs to which any transfer-variables are bound. An example of a transfer rule is given below:

```
:T:  example-1
:L1: <* a b> = X1
     <* c d> = Y1
:L2: <* p q> = X2
     <* p r> = Y2
:X:  X1 <=> X2
     Y1 <=> Y2
```

This rule establishes a correspondence between the two feature structures shown below, (1) being the FS described by the equations under 'L1' and (2) by those under 'L2':

$$(1)\ \begin{bmatrix} a & b & X1 \\ c & d & Y1 \end{bmatrix} \qquad (2)\ \begin{bmatrix} p & \begin{bmatrix} q & X2 \\ r & Y2 \end{bmatrix} \end{bmatrix}$$

The correspondence is licensed provisionally for this FS pair by "example-1"; it is licensed absolutely for a pair of FSs (1′) and (2′) having the same root as (1) and (2) respectively only if:

(i) (1′) contains sub-FSs α unified with X1 and β unified with Y1 in (1),
(ii) (2′) contains sub-FSs γ unified with X2 and δ unified with Y2 in (2), and
(iii) the same type of correspondence is licensed, possibly by some other rule, between α and γ and between β and δ.

Complex FSs are analysed and constructed recursively as a result of the passage of control through transfer variables.

In the abstract, transfer rules have no inherent directionality; the two FSs above may be visualized interchangeably as input and output, or 'source' and 'destination'. When compiled for a particular application, however, they are interpreted directionally, the domain of the transfer relation being collectively characterized by the equation sets labelled 'L1' and the range by those labelled 'L2', or vice versa. One may then think of compiled transfer rules as having a 'left-hand' or 'input' and a 'right-hand' or 'output' side, the former describing a source FS and the latter a destination FS. We shall use these terms freely in contexts where directionality is at issue, and assume that the rules have been compiled accordingly.

2.2. Interpretation

The relation of transfer between a source FS Σ and a destination FS Δ is defined recursively in terms of the quintuple $\langle R, \Phi_\lambda(R), \Phi_\rho(R), T(R), \Theta(\Sigma)\rangle$, where R is a rule, $\Phi_\lambda(R)$ and $\Phi_\rho(R)$ are, respectively, the FSs induced by the left-hand and right-hand equation sets in R, T(R) is the set of transfer correspondence statements in R, and $\Theta(\Sigma)$ is the result of converting any path-final variables in Σ to constants:[5]

Σ stands in the transfer relation to Δ with respect to R iff:

(i) $\Phi_\lambda(R)$ subsumes $\Theta(\Sigma)$, and
(ii) $\Phi_\rho(R)$ unifies with Δ, and
(iii) for each $\tau \in T(R)$, the sub-FSs of Σ and Δ unifying with the transfer variables mentioned in τ stand in the transfer relation with respect to some rule in the currently accessible rule set.

The first clause of this definition states the condition under which a rule is a candidate for application to a given input FS. The second states the condition under which a rule is a candidate for application to a given output FS. Note that the operations differ; whereas the matching in (i) is based on subsumption, the action in (ii) employs unification. As a consequence, the FS $\Phi_\rho(R)$ is added to the output FS Δ. The third clause imposes the further condition that, in order for Σ and Δ to be related by R, any FSs they contain which are explicitly connected via variable binding and a transfer correspondence statement in T(R) are also related.

As will be seen from clause (iii) of the definition, a complex FS is traversed from root to terminals, control being passed via variables in transfer equations, and the extent of each sub-transfer (i.e. how much of the input FS is consumed at each stage) being determined by

[5] It may well be the case that, in certain applications or environments, source FSs will not contain such variables; the possibility must be acknowledged nevertheless, since non-declarative rule interactions may otherwise occur.

the path specifications in the left-hand side equation set of the currently active rule. Possible paths through the FS from a given point are determined collectively by the left-hand side equations of all rules, together with their transfer correspondence statements.

Because FSs are finite and acyclic, termination is guaranteed as long as there is no rule of the form shown below. This is able to apply (in the 'L1 → L2' direction – we ignore the converse) without consuming part of the source FS:

```
:T:  infinite-recursion
:L1: <*> = X
:L2: ...
:X:  X <=> Y
```

Coherence of a destination FS with respect to a source FS and a set of transfer rules is ensured by the formalism; material can only be introduced into a destination FS by the right-hand side of transfer rules which have successfully applied. Completeness, on the other hand, must be verified explicitly; every part of the source FS must be subsumed by a subpart of the FS obtained by unifying the FSs induced by the left-hand side patterns of every rules that has successfully applied. In the current implementation, it is possible to declare that certain subparts of a source FS are not to be transferred; in this case, it is the remainder of that FS which must be covered by the rules.

3. Applications of the Formalism

We now illustrate how the transfer rule formalism may be exploited, and indicate briefly how the rule invocation regime may vary. The machine translation example in the following section assumes parallel invocation of the rule set, while that involving reductions to canonical form seems most amenable to the serial invocation of individual rules or subsets of rules.

3.1. Machine Translation

Perhaps the most obvious application for the formalism presented here lies in the domain of machine translation. The transfer model of MT may be thought of as involving three distinct mappings; from the source language expression to a source linguistic representation, from the source representation to a target representation, and from this to an expression in the target language. The first and last of these are to be performed by parsing and generation with natural language grammars, but, while proposals have been made to combine some of the three stages (e.g. Kaplan et al., 1989), there are advantages in treating the intermediate, transfer, stage independently.

As an example, consider the FSs shown below:[6]

(3)
$$\begin{bmatrix} \text{sem} & \begin{bmatrix} \text{pred} & \text{schwimmen} \\ \text{args} & \langle \text{<1> sem pred Maria} \rangle \\ \text{mod} & \text{sem pred gern} \end{bmatrix} \end{bmatrix}$$

(4)
$$\begin{bmatrix} \text{sem} & \begin{bmatrix} \text{pred} & \text{aimer} \\ \text{args} & \langle \text{<1> sem pred Maria,} \\ & \quad \text{<2> sem} \begin{bmatrix} \text{pred} & \text{nager} \\ \text{args} & \langle \text{\#1} \rangle \end{bmatrix} \rangle \end{bmatrix} \end{bmatrix}$$

(3) and (4) are possible representations for the German sentence *Maria schwimmt gern*, and the French sentence *Maria aime nager*, both of which might translate into English as 'Maria likes swimming'. Note that, whereas (3) has the predicate which translates 'swim' at the top level, and contains a modifier *gern* which might be glossed as 'gladly', (4) embeds the 'swim' predicate within an argument to the main predicate *aimer* 'like', and links the first argument of *aimer* to the first argument of *nager* by means of a re-entrancy.[7]

The set of rules given below together establish a transfer relation between (3) and (4):[8]

---

[6] Note the use of a list, indicated by '⟨...⟩', to encode arguments in these FSs, the identification of elements on such a list by e.g. '<1>', and re-entrancy flagged by '#'.

[7] Clearly, one could employ a similar analysis for the German sentence by making *gern* an 'equi' predicate like *aimer* – this would amount to simplifying transfer by shifting complexity from the transfer rules into the German grammar.

[8] This is not quite true; the variables 'Tf' and 'Tg' in the rule "gem-aimer" will bind to lists (the empty list in this case), and we therefore require additional generic list-transfer rules that will have the effect of passing through a list, recursively transferring heads and tails. Implementations for systems that lack the list data type will naturally be able to dispense with this. In addition, the lexical transfer rules assume the presence in the current set of a rule consuming the '<* sem pred>' paths terminating in *Paul* and *Maria*.

```
:TA: Paul Paul
:TA: Maria Maria
:T:  schwimmen-nager
:L1: <* sem pred> = schwimmen
     <* sem args> = [Xg]
:L2: <* sem pred> = nager
     <* sem args> = [Xf]
:X:  Xg <=> Xf
:T:  gern-aimer
:L1: <* sem pred> = Rg
     <* sem args> = [Ag|Tg]
     <* sem mod sem pred> = gern
:L2: <* sem pred> = aimer
     <* sem args> = [Af,Vf]
     <* sem args> = [Af,Vf]
     <Vf sem args> = [Af|Tf]
:X:  Rg <=> Rf
     Ag <=> Af
     Tg <=> Tf
     <Vf sem pred> = Rf
```

The pair of rules ':TA: Paul Paul' and ':TA: Maria Maria' are 'lexical transfer rules'; they state a transfer relation between atomic FSs (i.e. words, in the context of MT), rather than complex ones, and, further, do so without reference to the context of these FSs. They are equivalent to e.g.

```
:T:  Maria Maria
:L1: <*> = Maria
:L2: <*> = Maria
:X:  -
```

The re-entrancy in FS (4), in which the first argument associated with the predicate *aimer* is also the argument associated with the embedded predicate *nager*, is of some interest in connection with transfer. Taking (4) as the source, application of "gern-aimer" results in the binding of both instances of the variable 'Af' to the sub-FS indexed as '<1>' which is subject to the relevant transfer correspondence statement and whose corresponding destination sub-FS (in this case identical) will be present in the overall destination FS as the first element on the argument list of *schwimmen*. Reversing the direction, with (3) as the source, the variable 'Ag' is bound to the sub-FS indexed as '<1>', whose corresponding destination sub-FS is similarly present in the overall destination FS, this time as the first element in both argument lists, and, moreover, owing to the identity of variables in "gern-aimer", unified rather than duplicated. Re-entrancy may thus be detected in the source FS and created in the destination; naturally, responsibility for correctly analysing structures containing re-entrancies, and enforcing them where desired in output structures, lies with the writer of transfer rules.

3.2. Reduction to Canonical Form

It is often the case that a grammar assigns just one of a range of logically equivalent representations to a sentence; designers of grammars for use in analysis generally take care to ensure that the result of parsing a non-ambiguous sentence is a unique semantic representation, and multiple representations are seen as the hallmark of (pre-theoretical) ambiguity. In generation, as Shieber (1988) and Appelt (1989) observe, a situation may arise in which the representation supplied as input to the process (perhaps by another program) is not itself directly suitable, but is logically equivalent to one that is. The use of distinct grammars for parsing and generation could provide a solution to this problem, but it raises others connected with management of the resulting system. An alternative is to define equivalence classes of representations, and reduce all members of a class to the single canonical form which the grammar can map into a sentence. Exactly how the classes and reductions are defined will doubtless depend on many factors; we consider here some of the standard logical equivalences exploited in reducing arbitrary expressions of the propositional calculus to disjunctive normal form.

```
:T:  not-not
:L1: <* op> = not
     <* val 1 op> = not
     <* val 1 val 1> = Y
:L2: <*> = X
:X:  X <=> Y
:T:  not-or
:L1: <* op> = not
     <* val 1 op> = or
     <* val 1 val 1> = X1
     <* val 1 val 2> = X2
:L2: <* op> = and
     <* val 1 op> = not
     <* val 1 val 1> = Y1
     <* val 2 op> = not
     <* val 2 val 1> = Y2
:X:  X1 <=> Y1
     X2 <=> Y2
```

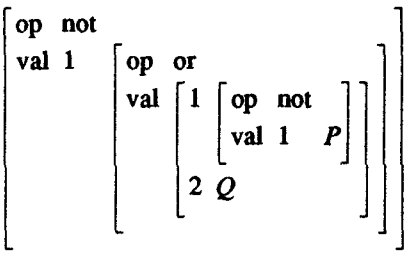
The two rules shown above express the equivalences which are more familiar as:

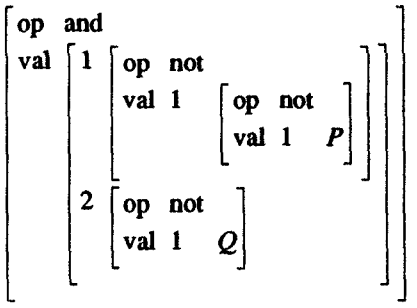
$$\neg(\neg p) \leftrightarrow p$$

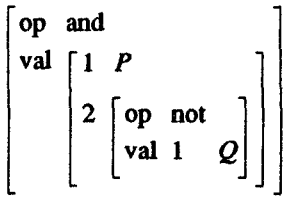
and

$$\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q).$$

The mode of application required here is rather different from that described in the preceding section, for a context in which "not-not" applies may not exist prior to the application of "not-or". Consider the three FSs below:

(5)


(6)


(7)


Given (5), the desired result is (7), by way of (6). A suitable context for the rule "not-not" is created by "not-or"; note, however, that this context exists only in the destination FS, and not in the source. What is required is a serial mode of invocation, as opposed to the parallel mode assumed for the MT application, with the 'output' of one rule serving as the 'input' to another. An alternative would be to formulate transfer rules that encompass a wider context; drawbacks of such an approach would be that it is not possible to cater for all contexts, and that, in attempting to do so, one would diminish the locality and thus the transparency of the rules.

There are several possibilities for implementing serial rule invocation; the most straightforward involves taking an output FS as the input to another pass through the rule set. In this case, vacuous application of the rule set must be detected in order to ensure termination.

It will not normally be desirable to apply canonicalization rules 'in reverse': the effect will be to derive all forms that are logically equivalent to the input, and, if the relevant equivalence classes are not finite, the process will not terminate. Consider the rule "not-not"; its presence in a rule set compiled with 'L2' as the left-hand side will result in the derivation of forms involving, at each point, an embedding of the source FS under a progressively higher even number of *nots*. This is as it should be, however, given the semantics of transfer rules outlined in section 2, since, in this direction, the rule characterizes a relation whose range is not finite. Individual applications of the rule terminate, nevertheless.

4. Conclusion

We have presented what is to our knowledge the first formalization and implementation of a type of rule and control regime intended for use in situations where it is desired to produce the effect of transforming one feature structure into another.[9]

The formalism described above has been implemented as part of ISSCO's ELU[10], an enhanced PATR-II style (Shieber, 1986) unification grammar environment, based on the UD system presented by Johnson and Rosner (1989). ELU incorporates a parser and generator, and is primarily intended for use as a tool for research in machine translation. Use of transfer rules in translation has not so far brought to light instances where the serial rule invocation regime described in section 3.2 proves necessary. ELU grammars permit the use of typed feature structures (cf. Johnson and Rosner, op. cit., Moens et al., 1989) in grammars; although the present transfer rule format does not, they are clearly a desirable addition, since they would provide a means of exerting control over rule interactions.

A third area in which the transfer rule formalism might be applied concerns the manipulation of re-entrant structures. While re-entrancy is in general a useful property of FSs, the complexity entailed by its presence is in some cases unwelcome; the method of genera-

[9] Van Noord (1990) describes the use of a standard unification grammar to successively instantiate a single feature structure embodying meaning representations for both source and target language expressions in a machine translation application. Similarly, the transfer rules of Zajac (1990) express a relation between subparts of a single complex structure. Such an approach does not appear suitable for the application discussed in section 3.2 above.

[10] "Environnement Linguistique d'Unification"

tion proposed by Wedekind (1988), for example, requires that the LFG-style f-structures which form the input to the generation process be 'unfolded' into unordered trees. This may be done with a suitably formulated rule set of the kind introduced here. The present rule format is unable to preserve the information that distinct sub-FSs in a destination FS arise from the duplication of a single, re-entrant, sub-FS in the source. Ways of incorporating this ability into the rule formalism are under consideration, one possibility being the addition of an indexing mechanism that would flag sub-FSs as originating in a re-entrancy.

A companion paper describes an interpretation of transfer rule sets in terms of a partial ordering with respect to the specificity of rules, and discusses linguistic and computational motivations for this view; it also comments in greater detail on the rule interaction problems referred to in fn. 3, and on issues of termination, completeness and coherence in transfer. Here, we simply note that, in the current implementation, it is possible to declare to the system the path set of a source FS that is to be subject to transfer, so as to provide run-time notification if inadequacies in the rule set result in a specified sub-FS being neglected. With respect to a given rule set and source FS, however, correctness of the transfer process is assured.

References

Appelt, Douglas E. (1989) "Bidirectional Grammars and the Design of Natural Language Generation Systems", in Y. Wilks (ed.) *Theoretical Issues in Natural Language Processing*; 199–205. Hillsdale, NJ: Laurence Erlbaum.

Johnson, Rod and Mike Rosner (1989) "A Rich Environment for Experimentation with Unification Grammars". *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, UK, April 10th–12th 1989; 182–189.

Kaplan, Ronald M., Klaus Netter, Jürgen Wedekind, and Annie Zaenen (1989) "Translation by Structural Correspondence". *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, UK, April 10th–12th 1989; 272–281.

Moens, Marc, Jo Calder, Ewan Klein, Mike Reape, and Henk Zeevat (1989) "Expressing Generalizations in Unification-based Grammar Formalisms". *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, UK, April 10th–12th 1989; 174–181.

Shieber, Stuart M. (1986) *An Introduction to Unification-Based Theories of Grammar*. CSLI Lecture Notes no. 4, CSLI, Stanford.

Shieber, Stuart M. (1988) "A Uniform Architecture for Parsing and Generation". *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, August 22nd–27th, 1988; 614–619.

van Noord, Gertjan (1990) "Reversible Unification Based Machine Translation". *Proceedings of the 13th International Conference on Computational Linguistics, vol.2*, Helsinki, Finland, August 20th–24th, 1990; 299–304.

Wedekind, Jürgen (1988) "Generation as Structure-Driven Derivation". *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, August 22nd–27th, 1988; 732–737.

Zajac, Rémi (1990) "A Relational Approach to Translation". *Proceedings of the Third International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, Austin, Texas, June 11th–13th, 1990.