Zwei induktive Konfigurierungsmodelle

O. Najmann und B. Stein
FB 17, Praktische Informatik, Universität-GH-Paderborn

Überblick: Inhalt dieses Papiers sind zwei Modelle, die einen formalen Rahmen zur Beschreibung bestimmter Konfigurierungsprobleme darstellen. Wichtiges Kennzeichen dieser Modelle ist, daß sie das zu konfigurierende technische System bzw. alle an der Konfigurierung beteiligten Objekte mit Hilfe von Funktionalitäten beschreiben. Modell M1 erlaubt die Formulierung von Konfigurierungsproblemen, wie sie bei der Ressourcen-orientierten Konfigurierung operationalisiert werden [Heinrich, 1991; Stein & Weiner, 1990]. Modell M2 ist eine Erweiterung von Modell M1 um eine Regelsprache und ermöglicht dadurch die Formulierung von Strukturwissen. Obwohl Modell M2 zusätzlich noch eine Regelsprache zur Beschreibung von Restriktionswissen besitzt, kann gezeigt werden, daß beide Ansätze bzgl. ihrer Ausdrucksstärke äquivalent sind.
Abschließend werden im letzten Abschnitt dieses Papiers auf Grundlage des formalen Rahmens mögliche Konfigurierungsprobleme formuliert und ein Komplexitätsergebnis präsentiert.

1 Motivation

Viele Ansätze und Studien im Bereich der Konfigurierung zielen auf technische Aspekte der Konfigurierung ab, wie z. B. der Beschreibung von Ideen, *wie* eine bestimmte Domäne modelliert werden kann, wie Abhängigkeiten zwischen Konfigurierungsobjekten behandelt, Hierarchien abgearbeitet werden können oder Heuristiken zum Einsatz kommen. Diese pragmatische, teleologische Sichtweise ist aufgrund der Komplexität des „allgemeinen Konfigurierungsproblems“[1] gerechtfertigt; viele Konfigurierungssysteme stellen Lösungen von Einzelfällen dar.
In diesem Zusammenhang ist PLAKON wohl das am weitesten entwickelte System. In PLAKON [Cunis et al., 1991] sind mehrere komplexe Konfigurierungsmethoden als auch eine mächtige Begriffshierarchiebeschreibungssprache operationalisiert.

Abgrenzend zu diesem und ähnlichen Ansätzen, die Methoden zum Verarbeiten und Akquirieren von Konfigurierungswissen bereitstellen (AMOR [Tank et al., 1989], DSPL [Brown & Chandrasekaran, 1989]), wird in diesem Papier eine bloße Formulierung von allgemeinen Konfigurierungsproblemstellungen durchgeführt. Eine solche Formulierung könnte dazu dienen, verschiedene Konfigurierungsansätze gegenüberzustellen und hinsichtlich ihrer Komplexität zu vergleichen. Ein Ansatz, der in eine ähnliche Richtung geht, ist das Konfigurierungsmodell von Dörner [Dörner 1991], auf das an dieser Stelle jedoch nicht näher eingegangen wird.

In unserem (in Abschnitt 2 und 3 vorgestellten) Ansatz, ist der zentrale Begriff die „Funktionalität“: Wenn Objekte ausgewählt werden, müssen ihre zugehörigen Funktionalitäten verknüpft werden, um die Gesamtfunktionalität des entstehenden Systems zu bestimmen. Diese Funktionalitäten-orientierte Herangehensweise ist besonders adäquat zur Beschreibung von Konfigurierungsproblemen, bei denen die *Selektion* von Komponenten im Vordergrund steht. Nachfolgend geben wir eine informelle Definition zum Begriff „Konfigurierungsproblem“ an,

[1] Sofern es ein solches überhaupt gibt.

die – obwohl sie sich an dem Selektionsproblem orientiert – noch ausreichend allgemein ist, um eine Vielzahl von Konfigurierungsproblemen zu umfassen.

Ein Konfigurierungsproblem besteht aus drei Mengen, (i) der Menge von Objekten, die zusammengesetzt bzw. kombiniert werden sollen, (ii) einer Menge von Funktionalitäten, die bestimmte Eigenschaften auf den Objekten definieren und (iii) einer Menge von Anforderungen, welche die gewünschten Eigenschaften des zu konfigurierenden Systems festlegen und zu erfüllen sind.

2 Modell M1

In diesem Abschnitt wird zunächst ein Konfigurierungs*problem* formalisiert und im Anschluß daran seine Lösung, die *Konfiguration* induktiv definiert.

Definition: Ein Konfigurierungsproblem Π ist ein Tupel $\langle O, F, V, P, A, T, D\rangle$, das wie folgt definiert ist:

- O ist eine beliebige endliche Menge, die *Objektmenge*.
- F ist eine beliebige endliche Menge, die *Funktionalitätenmenge*.
- Für jede Funktionalität $f \in F$ existiert eine beliebige endliche Menge v_f, die *Wertemenge* von f. $V = \{v_f | f \in F\}$ faßt diese Wertemengen zusammen.
- Für jedes Objekt o exisitiert eine *Eigenschaftsmenge* p_o, die Tupel der Form (f, x) enthält mit (i) $f \in F$ und $x \in v_f$, und (ii) jede Funktionalität $f \in F$ darf höchstens einmal in p_o auftauchen. Die Eigenschaftsmenge legt die Werte bzgl. jeder Funktionalität für ein gegebenes Objekt fest. $P = \{p_o | o \in O\}$ ist eine Zusammenfassung dieser Eigenschaftsmengen.
- Für jede Funktionalität f existiert ein *Additions-Operator* a_f, definiert durch eine partielle Funktion $a_f : v_f \times v_f \rightarrow v_f$. Ein Additions-Operator legt fest, wie zwei Werte einer Funktionalität zu einem neuen Wert verrechnet werden, falls ein weiteres Objekt zu einer Menge von Objekten hinzugenommen wird, welche ihrerseits einen Teil des zu konfigurierenden Systems beschreiben. $A = \{a_f | f \in F\}$ faßt diese Additions-Operatoren zusammen.
- Weiterhin existiert für jede Funktionalität f ein *Prädikat* t_f, definiert durch eine partielle Funktion $t_f : v_f \times v_f \rightarrow \{\text{WAHR, FALSCH}\}$. Ein Prädikat t_f gibt an, unter welchen Bedingungen eine Anforderung (siehe unten) erfüllt ist. $T = \{t_f | f \in F\}$ ist die Zusammenfassung dieser Prädikate.
- D ist eine beliebige, endliche Menge von *Anforderungen*. Jede Anforderung d ist ein Tupel (f, x) mit $f \in F$ und $x \in v_f$. Weiterhin muß die Menge der Anforderungen die Bedingung erfüllen, daß keine Funktionalität mehr als einmal in D vorkommt. Die Anforderungsmenge D enthält die gewünschten Eigenschaften des zu konfigurierenden Systems.

Anmerkungen: O enthält die Objekte, aus denen ein System zusammengesetzt werden kann, um einer bestimmten Anforderungsdefinition zu genügen. Soll zum Beispiel ein Computersystem konfiguriert werden, dann enthielte O Objekte wie Festplatten, Netzteile, eine CPU usw.

Weiterhin legen Funktionalitäten bestimmte Eigenschaften auf den Objekten in O fest. Eine Festplatte könnte zum Beispiel die Funktionalitäten „Plattenkapazität“, „Zugriffszeit“ usw. besitzen. Damit wäre $v_{\text{Plattenkapazitaet}} = \{10, 20, 30, 40\}$ [Megabytes].
In diesem einfachen Beispiel wäre eine typische Anwendung des Additions-Operators die Berechnung der gesamten Plattenkapazität einer Menge von Festplatten. Dann kann $a_{\text{Plattenkapazitaet}}$ wie nachfolgend definiert sein (das Symbol $\perp$ steht für undefiniert).

$$a_{\text{Plattenkapazitaet}}(x, y) = \begin{cases} x + y, & \text{if } x + y \leq 40; \\ \perp, & \text{sonst.} \end{cases}$$

Jedes Objekt $o \in O$ ist durch seine Eigenschaftsmenge p_o beschrieben. Für eine bestimmte Festplatte kann p folgendermaßen aussehen: $p_{\text{Festplatte}} = \{(\text{Plattenkapazität}, 10), (\text{Zugriffszeit}, 7)\}$

Ein sinnvolles Prädikat für „Plattenkapazität“ ist das „$\geq$“-Prädikat. In der Regel unterscheiden wir bei einer Menge von Anforderungen zwischen *externen* und *internen* Anforderungen. Während externe Anforderungen den Kundenwünschen entsprechen und von dem Kunden spezifiziert werden, legen die internen Anforderungen Umweltbedingungen oder andere Eigenschaften fest, die üblicherweise nicht vom Kunden vorgegeben werden.

In unserem Beispiel wäre eine Menge von externen Anforderungen $D_e = \{(\text{Plattenkap.}, 30), (\text{Hauptspeicher}, 2000), (\text{Grafik}, \text{VGA}), (\text{Coprocessor}, \text{ja})\}$ Eine Menge von internen Anforderungen sei $D_i = \{(\text{Netzspannung}, 220), (\text{Tastatur}, \text{deutsch})\}$ Somit ist die gesamte Anforderungsmenge D durch $D = D_e \cup D_i$ gegeben.

Bisher haben wir nur den Begriff des „Konfigurierungsproblems“ definiert; es muß noch definiert werden, wie eine Lösung eines solchen Problems auszusehen hat. Die Lösung eine Konfigurierungsproblems muß zwei Bedingungen erfüllen: 1. Es muß eine *Konfiguration* im wie weiter unten definierten Sinne sein. 2. Diese Konfiguration muß alle Anforderungen der Anforderungsmenge D erfüllen.

Eine Konfiguration enthält sowohl Objekte als auch Funktionalitäten. Bevor wir jedoch eine induktive Definition des Begriffs Konfiguration geben, muß definiert werden, wie die Eigenschaften von Objekten zu verknüpfen sind.

Definition: Seien (f, x), (g, y) zwei Eigenschaften.

$$\varphi((f, x), (g, y)) = \begin{cases} \{(f, a_f(x, y))\}, & \text{falls } f = g; \\ \{(f, x), (g, y)\}, & \text{sonst.} \end{cases}$$

heißt *Verknüpfung* der Eigenschaften (f, x) und (g, y).

Anmerkungen: Die Verknüpfung von zwei Eigenschaften ist eine Menge. Diese Menge enthält eine einzige Eigenschaft, falls die zu verknüpfenden Funktionalitäten gleich sind, im Falle der Ungleichheit enthält sie zwei Eigenschaften. Falls also zwei Objekte, die eine Funktionalität gemeinsam haben, sich in der Menge der selektierten Objekte befinden, müssen die zugehörigen Eigenschaftswerte (mit Hilfe des Additions-Operators) verrechnet werden. Ist diese Berechnung für die gegebene Konstellation nicht definiert, darf das neue Objekt nicht zur Konfiguration hinzugenommen werden.

In Anlehnung an unsere informelle Definition aus Abschnitt 1 muß eine Konfiguration festlegen, 1. welche Objekte Bestandteil des konfigurierten Systems sind, und 2. woraus die Gesamtfunktionalität des Systems besteht.

In unserem Modell ist eine *Konfiguration* ein Paar $C = \langle I, Q \rangle$, wobei I eine Menge von *Objektvorkommen* der Form (k, o) und Q eine Menge von *Systemeigenschaften* der Form (f, x)

ist. Ein Objektvorkommen (k, o) sagt aus, daß Objekt $o \in O$ k-mal in dem konfigurierten System Verwendung findet, während (f, x) bedeutet, daß der Wert der Funktionalität f des Gesamtsystems x ist.

Obwohl Systemeigenschaften und (Objekt-)Eigenschaften syntaktisch äquivalent sind, unterscheiden wir dennoch zwischen ihnen, weil eine Eigenschaft ein Merkmal eines einzelnen Objektes ist, während eine Systemeigenschaft (f, x) das Ergebnis der Verknüpfung mehrer Objekte ist, welche die Funktionalität f in ihrer Eigenschaftsmenge besitzen.

Auf der oben eingeführten Definition der Verknüpfung basierend, sind wir nun in der Lage, den Begriff der *Konfiguration* formal einzuführen.

Definition: Sei $\Pi = \langle O, F, V, P, A, T, D \rangle$ ein Konfigurierungsproblem. Eine *Konfiguration* C sei induktiv wie folgt definiert:

1. $C = \langle \emptyset, \emptyset \rangle$ ist eine Konfiguration.
2. Falls $C = \langle I, Q \rangle$ eine Konfiguration und o_i ein Objekt aus O ist, dann ist $C' = \langle I', Q' \rangle$ eine Konfiguration, falls die folgenden Bedingungen gelten:
   (i) Für jedes $(f, x) \in p_o$ und für jedes $(g, y) \in Q$, ist die Verknüpfung $\varphi((f, x), (q, y))$ definiert oder $Q = \emptyset$.
   (ii)
   $$I' = \begin{cases} I \setminus \{(k, o)\} \cup \{(k+1, o)\}, & \exists (k, o) \in I; \\ I \cup \{(1, o_i)\}, & \text{sonst.} \end{cases}$$
   (iii)
   $$Q' = \begin{cases} \varphi((f, x), (g, y)), & Q \neq \emptyset; \\ p_o, & Q = \emptyset. \end{cases}$$
3. Nichts sonst ist eine Konfiguration.

Anmerkungen: Bedingung (i) garantiert, daß nur diejenigen Objekte $o \in O$ zu einer gegebenen Konfiguration C hinzugenommen werden, von denen alle Eigenschaften p_o mit allen Systemeigenschaften von C verknüpft werden können. Bedingung (ii) gibt an, wie ein neues Objekt zu einer gegebenen Menge von Objektvorkommen hinzugefügt werden kann. Mit Bedingung (iii) wird festgelegt, wie die neue Menge von Systemeigenschaften bei Hinzunahme eines neuen Objektes gebildet wird.

Hiermit ist es nun möglich, eine präzise Definition des Begriffs der *Lösung* eines Konfigurierungsproblems zu geben:

Definition: Eine Konfiguration $C = \langle I, Q \rangle$ ist eine *Lösung* eines Konfigurierungsproblems $\Pi = \langle O, F, V, P, A, T, D \rangle$ genau dann, wenn für jede Anforderung $d = (f, x) \in D$ eine Systemeigenschaft $q = (g, y) \in Q$ existiert so, daß $f = g$ und $t_f(x, y) = \text{WAHR}$. Die Menge $S(\Pi) = \{C \mid C \text{ ist eine Lösung von } \Pi\}$ heißt der *Lösungsraum* von Π.

Anmerkungen: Die obige Bedingung sichert, daß alle Anforderungen erfüllt sind.
In der Regel existiert mehr als eine Lösung des Konfigurierungsproblems Π. Dann wird $S(\Pi)$ manchmal auch als „Variantenraum“ bezeichnet.

Allgemeine Konfigurierungsprobleme

Auf der Grundlage der Formalisierung lassen sich eine Reihe von allgemeinen Konfigurierungsproblemstellungen definieren.

Problem CONF
Gegeben: Ein Konfigurierungsproblem Π.
Frage: Gibt es eine Lösung von Π ?

Problem FINDCONF
Gegeben: Ein Konfigurierungsproblem Π.
Aufgabe: Finde eine Lösung von Π, falls eine existiert.

Problem COSTCONF
Gegeben: Ein Konfigurierungsproblem Π, eine Kostenfunktion $C : O \to \mathbf{Q}$ und maximale Kosten $c^* \in \mathbf{Q}$.
Frage: Existiert eine Lösung $C = \langle I, Q \rangle$ von Π so, daß $\sum_{(k,o)\in I} k\, c(o) \leq c^*$ gilt ?

3 Modell M2

In diesem Abschnitt werden wir das Modell M1 in der Art ausbauen, daß wir Regeln einführen. Diese Regeln können als Einbaurestriktionen interpretiert werden. Ein Beispiel für solch eine Regel könnte sein: „Falls eine Festplatte vom Typ A benutzt wird, dann muß entweder der Festplattencontroller vom Typ B oder vom Typ C sein“. Es folgt nun eine Definition einer Konfigurierungsrestriktionssprache, die es uns ermöglicht, Regeln wie die obige zu formulieren.

Definition: 1. Sei O eine Menge von Objekten und $N = \{1, \ldots, k\}$. Weiterhin bezeichne $\Gamma(N, O) = \{[n, o] \,|\, n \in N, o \in O\}$ eine Menge von Booleschen Variablen über N und O. Eine *Konfigurierungsrestriktionsregel* r ist eine Implikation $[n, o] \to \psi$, wobei $[n, o] \in \Gamma(N, O)$ und ψ eine logische Formel ist, die über $\Gamma(N, O)$ unter Verwendung von Klammern, '$\neg$', '$\wedge$' und '$\vee$' im konventionellen Sinne gebildet wird. Eine *Regelmenge* R ist eine endliche Menge von Konfigurierungsrestriktionsregeln über $\Gamma(N, O)$.

2. Sei O wie unter 1. gegeben. $C = \langle I, Q \rangle$ bezeichne eine Konfiguration mit $I \subseteq N \times O$. Eine *Konfigurationsbelegung* α_I ist eine Funktion $\alpha_I : \Gamma(N, O) \to \{\text{WAHR}, \text{FALSCH}\}$ so, daß für alle $[n, o] \in \Gamma(N, O)$ gilt:

$$\alpha_I([n, o]) = \begin{cases} \text{WAHR}, & \text{falls } (n, o) \in I; \\ \text{FALSCH}, & \text{sonst.} \end{cases}$$

3. Eine Konfiguration $C = \langle I, Q \rangle$ heißt *erfüllend für eine Regelmenge* R genau dann, wenn jede Regel $r \in R$ unter der Belegung von α_I wahr wird.

Anmerkungen: Die Semantik einer Restriktionsregel läßt sich durch folgendes Beispiel veranschaulichen: Es sei $r = [1, A] \to ([2, B] \wedge \neg[1, C]) \vee [3, D]$. Die Bedeutung von r ist: „Enthält eine Konfiguration genau ein Objekt A, so muß diese Konfiguration entweder zwei B's und nicht ein C oder drei D's enthalten.“

Mit obiger Definition sind wir nun in der Lage, ein erweitertes Konfigurierungsmodell zu definieren, das Restriktionsregeln enthält.

Definition: Ein Konfigurierungsproblem unter Modell M2 besteht aus einem Tupel $\Pi_R = \langle O, F, V, P, A, T, D, N, R\rangle$, wobei alle Elemente außer N und R wie in Modell M1 vereinbart seien, $N = \{1, \ldots, k\}$ und R eine Menge von Konfigurierungsrestriktionsregeln ist, die über $\Gamma(N, O)$ gebildet werden. Eine Konfiguration $C = \langle I, Q\rangle$ ist eine Lösung von Π_R genau dann, wenn für jede Anforderung $(f, x) \in D$ eine Systemeigenschaft $(g, y) \in Q$ existiert so, daß $f = g$ und $t_f(x, y) =$ WAHR und C erfüllend ist für die Regelmenge R.

4 Vergleich von M1 und M2

Während mit Hilfe von M1 „klassische“ Ressourcen-orientierte Konfigurierungsprobleme beschrieben werden können, erlaubt die Regelsprache von M2 eine explizite Formulierung von Einbau- bzw. Ausschlußwissen. Deshalb kann mit diesem Mechanismus Strukturwissen, welches typisch für Konfigurierungsprobleme ist, deren zentraler Bestandteil ein Strukturmodell darstellt (Skelettkonfigurierungsprobleme), beschrieben werden.
Ein Skelett (im Sinne von Strukturgraph) ist in M2 ein Digraph $G = (V, E)$, wobei der Knotenmenge V die Menge der selektierbaren Objekte O entspricht. Weiterhin ist eine Kante (o_i, o_j) genau dann in E enthalten, wenn in M2 eine Regel existiert, die o_i auf der linken und o_j auf der rechten Seite enthält.

Weil Modell M2 zusätzlich einen Mechanismus zur Beschreibung von Strukturwissen enthält, scheint es mächtiger als das reineRessourcen-orientierte Modell M1 zu sein. Nachfolgende Ergebnisse zeigen jedoch, daß das nicht der Fall ist:

Satz A: Es sei Π eine Instanz des Problems CONF unter Modell M1 (M2). Dann existiert eine äquivalente Instanz Π' von Problem CONF unter Modell M2 (M1), die sich in polynomieller Zeit in der Größe von Π bestimmen läßt.

Korollar: Satz A gilt genauso für die Probleme FINDCONF und COSTCONF.

Beweis: Teil I: Sei $\Pi = \langle O, F, V, P, A, T, D\rangle$ eine beliebige Probleminstanz von CONF unter Modell M1. Setze trivialerweise $R := \emptyset$, $N := \{1\}$ und $\Pi' := \langle O, F, V, P, A, T, D, N, R\rangle$.

Teil II: Sei $\Pi = \langle O, F, V, P, A, T, D, N, R\rangle$ mit $N = \{1, \ldots, k\}$ eine beliebige Probleminstanz unter Modell M2. Wir müssen zeigen, daß eine beliebige Probleminstanz unter Modell M1 existiert, so daß Π eine Lösung hat, genau dann wenn Π' eine Lösung hat.

Wir konstruieren Π' wie folgt. Die grundlegende Idee des Beweises ist, alle Regeln durch Funktionalitäten und Tests zu ersetzen, deren Verhalten äquivalent zu diesen Regeln ist.

Zunächst wird R in eine logisch äquivalente Menge $\tilde{R}$ umgeformt, die nur 3KNF-Formeln enthält (das sind Formeln in konjunktiver Normalform mit höchstens drei Literalen). Diese Transformation wird in zwei Schritten gemacht.

Im ersten Schritt wird eine Transformationtechnik nach Tseitin [1983] angewendet, um jede Regel in eine erfüllbarkeitsäquivalente Formel in konjunktiver Normalform zu bringen. Dieser Schritt erfordert die Einführung neuer Variablen, $\bar{\Gamma} = \{[1, \bar{o}_1], \ldots, [1, \bar{o}_T]\}$. Im zweiten Schritt wird jede in Schritt 1 erzeugte Formel in 3KNF gebracht. Dies erfordert ebenfalls die Einführung weiterer Variablen, $\hat{\Gamma} = \{[1, \hat{o}_1], \ldots, [1, \hat{o}_S]\}$. Beachte, daß beide Transformationen in quadratischer Zeit und linearem Platz ausgeführt werden können. Sei $\Gamma' = \Gamma(N, O) \cup \bar{\Gamma} \cup \hat{\Gamma}$.

Jede Regel $r \in R$ wird also in eine Menge $\text{3KNF}(r) = \{r_1, \ldots, r_u\}$ transformiert, so daß r erfüllbar ist genau dann, wenn jede Formel $r_i \in \text{3KNF}(r)$ erfüllbar ist. Sei $\tilde{R} = \bigcup_{r \in R} \text{3KNF}(r)$.

Die Einführung neuer Variablen impliziert, daß die neue Objektmenge definiert ist als $O' := O \cup \bar{O} \cup \hat{O}$, mit $\bar{O} = \{\bar{o}_1, \ldots, \bar{o}_T\}$ und $\hat{O} = \{\hat{o}_1, \ldots, \hat{o}_S\}$.

1. Für jedes $r \in \tilde{R}$ wird eine Funktionalität g_r eingeführt, dessen Werte Mengen(!) sind. Für jedes $o \in O'$, das in einer Regel $r \in \tilde{R}$ vorkommt, wird die Eigenschaftsmenge von o in Bezug auf g_r als $g_r(o) := \{(1, o)\}$ definiert.

2. Wir definieren die folgende „Vereinigung“ von einer Menge von Objektvorkommen X und einer einelementigen Menge $\{(1, o)\}$, $o \in O'$ wie folgt:

$$X \uplus \{(1,o)\} = \begin{cases} X \setminus \{(n,o)\} \cup \{(n+1,o)\}, & \text{falls } o \in O,\ o \text{ kommt in } X \text{ vor und } n \leq k; \\ X \setminus \{(n,o)\} \cup \{k+1,o)\}, & \text{falls } o \in O,\ o \text{ kommt in } X \text{ vor und } n > k; \\ X, & \text{falls } o \in \bar{O} \cup \hat{O} \text{ und } o \text{ kommt in } X \text{ vor;} \\ X \cup \{(1,o)\}, & \text{sonst.} \end{cases}$$

Diese „Vereinigungs“-Funktion wird nachfolgend zur Konstruktion der Wertemengen der neuen Funktionalitäten benutzt.

3. Die Wertemenge v_{g_r} ist induktiv wie folgt definiert; hierbei ist Δ eine Hilfsvariable.

(i) Falls o in r vorkommt, dann ist $g_r(o) \in \Delta$.

(ii) Falls $X \in \Delta$ und o kommt in r vor, dann ist $X \uplus \{(1, o)\} \in \Delta$.

(iii) Nichts sonst ist in Δ.

(iv) $v_{g_r} := \Delta \cup \{r\}$.

Beachte, daß v_{g_r} sowohl Mengen von Zahl/Objekt-Paaren als auch die Regel r selbst enthält. Die Berechnung von v_{g_r} kann offensichtlich in polynomieller Zeit gemacht werden, da $k+1$ eine obere Schranke für die Zahl n ist, die in einem Paar $(n, o) \in X \in v_{g_r}$ auftreten kann.

4. Als eine Anforderung d_r für r definieren wir $d_r := (g_r, r)$.

5. Für g_r definieren wir den Test t_{g_r} wie folgt, wobei $t_{g_r}(X, Y)$ nur für $X = r$ und $Y \in v_{g_r} \setminus \{r\}$ definiert ist.

$$t_{g_r}(r, Y) = \begin{cases} \text{WAHR}, & \text{falls } r \text{ erfüllt ist unter } \alpha_Y; \\ \text{FALSCH}, & \text{sonst;} \end{cases}$$

wobei α_Y beschränkt ist auf die Variablenmenge $\Gamma_r = \{[n, o] \,|\, n \leq k+1, \text{ und } o \text{ tritt in } r \text{ auf }\}$. Beachte, daß andere Variablen als die aus Γ_r nicht auftreten können, da stets $n \leq k+1$.

6. Als Additionsoperator a_{g_r} für die Funktionalität g_r definieren wir $a_{g_r}(X, Y) := X \uplus Y$, sofern $X \in v_{g_r} \setminus \{r\}$ und $Y \in \{g_r(o) | o \in r\}$.

7. Desweiteren sei $\rho(o) = \{(g_r, g_r(o)) \,|\, r \in \{r \in \tilde{R} \,|\, o \text{ tritt in } rauf\}\}$.

8. Die Elemente von $\Pi' := \langle O', F', V', P', A', T', D' \rangle$ sind nun wie folgt definiert:

$$\begin{aligned} O' &:= O \cup \bar{O} \cup \hat{O}, \\ F' &:= F \cup F_R, \text{ wobei } F_R := \{g_r \,|\, r \in \tilde{R}\}, \\ V' &:= V \cup V_R, \text{ wobei } V_R := \{v_{g_r} \,|\, r \in \tilde{R}\}, \end{aligned}$$

$$
\begin{aligned}
P' &:= \{p_o \cup \rho(o) | o \in O\} \cup \{\rho(o) | o \in \overline{O} \cup \hat{O}\} \\
A' &:= A \cup A_R, \text{ wobei } A_R := \{a_{g_r} \,|\, r \in \tilde{R}\}, \\
T' &:= T \cup T_R, \text{ wobei } T_R := \{t_{g_r} \,|\, r \in \tilde{R}\}, \\
D' &:= D \cup D_R, \text{ wobei } D_R := \{(g_r, r) \,|\, r \in \tilde{R}\}.
\end{aligned}
$$

Fall A. Wir haben zu zeigen: Falls $C = \langle I, Q\rangle$ eine Lösung von Π ist, dann existiert eine Lösung $C' = \langle I', Q'\rangle$ von Π'. Wir zeigen, daß eine Menge von Objektvorkommen $\triangle I$ und eine Menge von Systemeigenschaften $\triangle Q$ existieren, so daß $I' = I \cup \triangle I, Q' = Q \cup \triangle Q$ und $C' = \langle I \cup \triangle I, Q \cup \triangle Q\rangle$ eine Lösung von Π' ist. Aufgrund der Konstruktion von I' und weil $D' = D \cup D_R$, brauchen nur die "Differenzanforderungen" D_R betrachtet werden. (Die ursprünglichen Anforderungen in D sind durch I bereits erfüllt.)

Es muß nun eine Menge $\triangle I$ derart konstruiert werden, daß I' eine Menge von Systemeigenschaften $\triangle Q$ induziert, die folgende Eigenschaft hat: Zu jedem $d = (g_r, r) \in D_R$ existiert eine Eigenschaft $(g_r, Y) \in \triangle Q$ mit $t_{g_r}(r, Y) = \text{WAHR}$.

Sei $d = (g_r, r)$ eine beliebige Anforderung aus D_R, wobei $r \in \tilde{R}$ eine 3KNF-Formel der Form $r = l_1 \vee l_2 \vee l_3$ ist, mit $l_i \in \{[n, o] \,|\, [n, o] \in \Gamma'\} \cup \{\neg[n, o] \,|\, [n, o] \in \Gamma'\}$. Beachte, daß r erfüllt ist, wenn ein beliebiges l_i erfüllt ist. Da C eine Lösung Π ist, folgt, daß alle Regeln $r \in R$ erfüllt sind. Folglich müssen alle 3KNF-Formeln in $\tilde{R}$ erfüllbar sein durch eine Belegung $\alpha_{I'}$. Beachte, daß $\alpha_I \subseteq \alpha'$; dies garantiert, daß ein Objekt $o \in O$ mit Häufigkeit n in I auftritt genau dann, wenn Objekt o mit Häufigkeit n in I' auftritt. Sei o.B.d.A. $l_1 = [n_1, o_1]$ unter $\alpha_{I'}$ erfüllt.

Fall A1: Sei $o_1 \in O$. Falls $l_1 = [n_1, o_1]$, dann ist $\alpha_{I'}([n_1, o_1]) = \text{WAHR}$, und folglich muß (n_1, o_1) in I auftreten. Die Definition von a_{g_r} (vgl. den"$\uplus$-Operator") garantiert, daß ein $Y \in v_{g_r}$ mit $(n_1, o_1) \in Y$ eindeutig als Systemeigenschaftswert von g_r konstruierbar ist. Falls $l_1 = \neg[n_1, o_1]$, dann ist $\alpha_{I'}([n_1, o_1]) = \text{FALSCH}$, und folglich ist $(n_1, o_1) \notin I$. Nun ist entweder $(m_1, o_1) \in I$ mit $m_1 < n_1$ oder $m_1 > n_1$, dann ist $(m_1, o_1) \in Y$, oder o_1 tritt überhaupt nicht nicht in I auf, dann ist $(m_1, o_1) \notin Y$. Wie zuvor ist ein passendes $Y \in v_{g_r}$ konstruierbar als Systemeigenschaftswert von g_r.

Fall A2: Sei $o_1 \in \bar{O} \cup \hat{O}$. Falls $l_1 = [n_1, o_1]$, dann ist $\alpha_{I'}([1, o_1]) = \text{WAHR}$ und wir fügen $(1, o_1)$ in $\triangle I$ ein. Der Systemeigenschaftswert Y von g_r enthält dann $(1, o_1)$. Ist $l_1 = \neg[1, o_1]$, dann ist $\alpha_{I'}([1, o_1]) = \text{FALSCH}$ und o_1 wird nicht in $\triangle I$ eingefügt. Folglich enthält der Systemeigenschaftswert Y von g_r nicht das Paar $(1, o_1)$. $\triangle I$ ist damit die Menge aller Paare $(1, o_1)$, die im Fall A2 gefunden wurden. Weiterhin wird durch alle g_r (mit $r \in \tilde{R}$) und ihren korrespondierenden Y-Werten die Menge $\triangle Q$ gebildet. Wie oben gesehen, wird mit Hilfe der Definitionen von $\triangle Q$ und $\triangle I$ garantiert, daß zu jedem $d = (g_r, r) \in D_R$ eine Eigenschaft $(d_r, Y) \in \triangle Q$ existiert, so daß $t_{g_r}(r, Y) = \text{WAHR}$.

Fall B. Wir müssen zeigen: Falls $C' = \langle I', Q'\rangle$ eine Lösung von Π' ist, dann existiert eine Lösung $C = \langle I, Q\rangle$ von Π. Da $C' = \langle I', Q'\rangle$ eine Lösung von Π' ist, sind alle Anforderungen in $D' = D \cup D_R$ erfüllt. Sei $I = \{[n, o] \,|\, o \text{ tritt in } O \text{ auf}\}$ und sei $\Delta Q = \{(f, x) \in Q \,|\, f \in p_o, o \in \bar{O} \cup \hat{O}\}$.

1. Offensichtlich erfüllt $C = \langle I, Q' \setminus \Delta Q\rangle$ alle Anforderungen $d \in D$, da Objekte, die in $I' \setminus I$ auftreten, keine Eigenschaften haben, für die eine Anforderung $d \in D$ existiert.

2. Um zu sehen, daß $C = \langle I, Q' \setminus \Delta Q\rangle$ alle Regeln $r \in R$ erfüllt, braucht nur obige Transformation berücksichtigt zu werden, die garantiert, daß α_I eine erfüllende Belegung ist, da $\alpha_I \subseteq \alpha_{I'}$ und $R \iff \tilde{R}$. $\diamond$

5 Skelettorientierte Konfigurierung

In diesem Abschnitt wird ein einfaches Beispiel für skelettorientierte Konfigurierung unter Modell M2 vorgestellt. Obwohl jedes Konfigurierungsproblem und seine Lösung stark von den Besonderheiten der Domäne abhängt, gibt das Beispiel eine Idee, wie ein hierarchisch strukturiertes Konfigurierungsproblem beschrieben werden kann.

Ein typisches Merkmal skelettorientierter Konfigurierung ist, daß der Lösungsraum als hierarchischer UND/ODER-Graph beschrieben werden kann (vgl. Puppe [1990]). Dieser Konfigurierungsansatz ist vor allem dann sinnvoll, wenn man System konfigurieren möchte, welches stets dieselbe Struktur hat.

Im nachfolgenden Beispiel soll ein Turm zusammengesetzt werden, der aus den drei Ebenen A, B und C besteht. Für jede dieser Ebenen stehen bestimmte Bausteine zur Verfügung, welche die folgenden Bedingungen einhalten müssen: Für Ebene A und B muß genau ein Baustein ausgewählt werden; für die C-Ebene gilt, daß C3 nicht mit einem anderen C-Baustein zusammen benutzt werden darf und daß bei der Verwendung von einem C2 der Baustein B1 nicht Bestandteil des Turms sein darf. Das Ziel ist es, einen Turm mit vorgegebener Höhe und minimalen Kosten zu bauen. Die nachfolgende Abbildung 1 zeigt die zur Verfügung stehenden Bausteine.

Baustein	Höhe	Kosten
A1	1	2
A2	2	4
B1	1	3
B2	3	5
C1	1	2
C2	2	3
C3	4	6

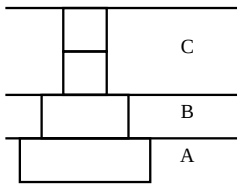

Abbildung 1: *Ein einfaches Konfigurierungsbeispiel*

Um dieses Problem als hierarchisches Konfigurierungsproblem zu beschreiben, werden spezielle „Dummy-Bausteine“ S, A, B, C eingeführt, die keine Eigenschaften besitzen. Mithilfe dieser Dummy-Bausteine können die Konfigurierungsrestriktionen nun wie folgt beschrieben werden:

[1,S] $\rightarrow$ [1,A] $\wedge$ [1,B] $\wedge$ [1,C] , [1,A] $\rightarrow$ [1,A1] $\vee$ [1,A2] , [1,B] $\rightarrow$ [1,B1] $\vee$ [1,B2] , [1,C] $\rightarrow$ [1,D] $\vee$ [1,C3] , [1,D] $\rightarrow$ [1,C1] $\vee$ [2,C1] $\vee$ [1,C2] $\vee$ [2,C2] , [1,C2] $\rightarrow \neg$ [1,B1]

Die Objektmenge O besteht aus {S,A,B,C,D,A1,A2,B1,B2,C1,C2,C3} und die Funktionalitätenmenge aus F = {Höhe}.

Mit der in Abschnitt 4 gegebenen Transformationsvorschrift von Modell M2 nach Modell M1 kann dieses Konfigurierungsproblem in ein Problem umformuliert werden, welches nur auf Funktionalitäten und ihrer Verrechnung basiert. In dem umformulierten Problem müssen die Beziehungen zwischen den Objekten über ihre Funktionalitäten abgeleitet werden. Die folgende Abbildung zeigt das ursprüngliche Konfigurierungsproblem und seine konkrete Umformulierung. Die Transformation obiger Regeln führte zu acht neuen Funktionalitäten $g_1, \ldots, g_8$. In der Abbildung 2 werden sowohl die Objekte als auch ihre Eigenschaften durch Knoten dargestellt; eine Kante (o_i, g_j) besagt, daß bei Objekt o_i die Funktionalität g_j in Eigenschaftsmenge vorkommt.

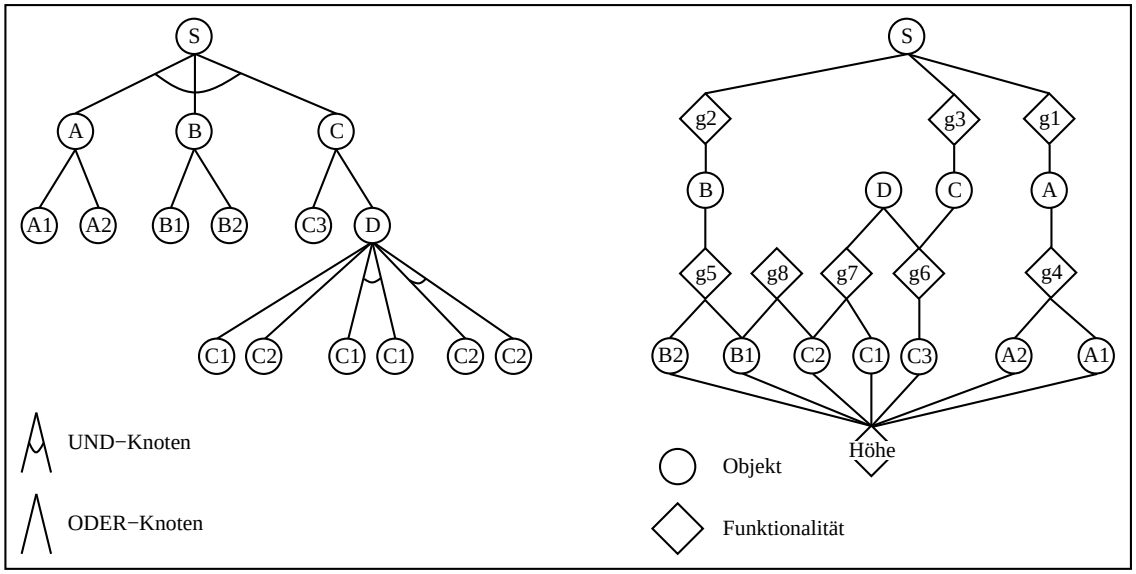

Abbildung 2: *Transformation des Konfigurierungsbeispiels von M1 nach M2*

6 Eine Komplexitätsbetrachtung

Auf Basis der in den Abschnitten 2 und 3 beschriebenen Modelle ist es nun möglich, Komplexitätsbetrachtungenen für oben definierte Konfigurierungsproblemstellungen durchzuführen. Es gilt folgender Satz:

Satz: Problem CONF ist NP-vollständig.

Auf diesen Satz wird in [Najmann & Stein, 1992] eingegangen. Die Beweisidee sei kurz skizziert:

Zunächst wird gezeigt, daß das Problem CONF mit Regeln (= CONF_R) NP-vollständig ist. Dies geschieht durch Reduktion von 3SAT auf CONF_R. Hierbei wird jede Klausel von 3SAT in eine logisch äquivalente (Einbau-) Regel transformiert. Aus der Äquivalenz von CONF und CONF_R folgt unmittelbar obiger Satz.

Es sei noch darauf hingewiesen, daß die anderen Konfigurierungsprobleme mindestens so schwer wie CONF sind.

7 Zusammenfassung

Es wurden die zwei Konfigurierungsmodelle M1 und M2 vorgestellt. Während M1 die Beschreibung typischer Ressourcen-orientierter Konfigurierungsprobleme erlaubt, kann mit M2 zusätzlich auch Strukturwissen formuliert werden.
Modell M2 enstand aus Modell M1 durch die Einführung einer Regelsprache. Obwohl Modell M2 bzgl. der Formulierung von Konfigurierungsproblemen mächtiger zu sein scheint, konnte die Äquivalenz beider Ansätze gezeigt werden. Weiterhin wurde gezeigt, daß ein grundsätzliches Problem, nämlich die Entscheidung, ob überhaupt eine Konfiguration für eine gegebene Anforderungsdefinition existiert, NP-vollständig ist.

Literatur

Brown, D. C.; Chandrasekaran, B. [1989] *Design Problem Solving,* Pitman.

Cunis et al. [1991] *Das PLAKON-Buch – Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen,* Springer Verlag, Berlin.

Dörner, H. [1991] "Ein Modell des Konfigurierens" *Beiträge zum 5. Workshop "Planen und Konfigurieren",* LKI-M-1/91

Heinrich, M. [1991] "Ressourcenorientierte Modellierung als Basis modularer technischer Systeme," *Beiträge zum 5. Workshop "Planen und Konfigurieren"* LKI-M-1/91

Najmann, O., Stein, B. [1992] "A Theoretical Framework for Configuration", erscheint in *Proceedings of the 5th IEA AIE 1992*, Belli, F. (Ed.)

Puppe, F. [1990] *Problemlösungsmethoden in Expertensystemen,* Springer Verlag.

Stein, B.; Weiner, J. [1990] *MOKON,* Interner Report, Universität-Gesamthochschule-Duisburg, SM-DU-178

Tank et al. [1989] "AMOR: Eine Beschreibungssprache für technische Systeme zur Unterstützung der Wissensakquisition für Konfigurationsprobleme", *Beiträge zum 3. Workshop "Planen und Konfigurieren",* Arbeitspapiere der GMD 388

Tseitin, G . [1983] "On the complexity of derivations in propositonal calculus", *Automation of Reasoning 2: Classical Papers on Computational Logic*, pp. 466-483. Springer Verlag, Berlin.