Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-93-20

Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus

Bernhard Herbig

August 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Friedrich J. Wendl
Director

Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus

Bernhard Herbig

DFKI-D-93-20

Diese Arbeit wurde finanziell unterstützt durch das Bundesministerium für Forschung und Technologie (ITW-FKZ ITW 8902 C4 and FKZ 413 5839 ITW 9304/3).

Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus

Bernhard Herbig

Kaiserslautern, April 1993

Zusammenfassung

Im Bereich des Constraint Logic Programming gibt es einen Ansatz, Constraint Formalismen explizit über sogenannte „Constraint Simplification Rules" zu implementieren. Daran anlehnend wird eine formale Grundlage geschaffen, die es erlaubt, Regeln zum Ersetzen und Erweitern von Basismengen zu definieren. Beim Versuch, mit Hilfe solcher Regeln den Inferenzmechanismus terminologischer Sprachen – den Konsistenztest für A-Boxen – zu implementieren, stößt man an die Grenzen des Formalismus. Die klare Formalisierung ermöglicht es aber, den Regelformalismus adäquat zu erweitern. Die Handhabung dieser Erweiterung wird an einer Reihe von ganz verschiedenen Beispielen demonstriert: der Konsistenztest für A-Boxen, logische Programme, lokale Konsistenz, Unifikation, Erfüllbarkeit von Ungleichungen. Ferner wird ein Abarbeitungsmechanismus definiert, dessen Implementation in COMMON LISP im Anhang vorgestellt wird.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Bereich des Constraint Logic Programming (CLP) gibt es einen Ansatz, Constraint Formalismen explizit über sogenannte „Constraint Simplification Rules" (CSiRs) zu implementieren.[1] Damit begegnet man den Problemen, die durch den „black box"-Charakter der Constraint Solver (CS) in existierenden CLP-Systemen entstehen. (Es ist schwierig solche CS zu erweitern, zu spezialisieren, zu optimieren, usw.) Diese CSiRs werden als redundant zum eigentlichen „Programm" betrachtet, so daß es semantisch gleichgültig ist unter welcher Strategie sie angewendet werden, und wirken über der Ebene des logischen Programms mit dem Ziel den Suchraum zu verkleinern (durch Vereinfachung von Goals, die Vermeidung von unnötigem Backtracking u.ä.). Dort gibt es zwei Arten von CSiRs:

(a) *Replacement Simplification Rules:* Hier wird eine Menge von Goals, die sog. Head-Atome, durch eine andere Menge von Goals, den sog. Body-Atomen, ersetzt, falls eine Menge von Bedingungen, sog. Guard-Atome, zutrifft. Das Ziel ist, die Head-Atome durch die Body-Atome zu vereinfachen.

(b) *Augmentation Simplification Rules:* Hier wird einer Menge von Head-Atomen um eine Menge von Body-Atomen erweitert, falls die Guard-Atome erfüllt sind. Das Ziel ist, implizit vorhandenes Wissen explizit zu machen.

Durch die Verwendung von CSiRs wird die Abstimmung eines Constraintsystems auf eine bestimmte Anwendung erheblich erleichtert, da es viel leichter ist, neue Regeln (definiert durch die Head- und Body-Atome) und Anwendbarkeitbedingungen (definiert durch die Guard-Atome) zu formulieren, als beispielsweise ein C-Programm zu modifizieren (durch das ein CS implementiert ist).

Bemerkenswerterweise wird der Inferenzmechanismus terminologischer Sprachen, der Konsistenztest für A-Boxen, mit Hilfe von Regeln definiert, die Assertionsmengen *erweitern* bzw. Assertionen *ersetzen*. Die Idee war nun diese Regeln mit Hilfe des Formalismus der CSiRs zu implementieren. Es zeigte sich jedoch, daß der Formalismus der CSiRs zu unflexibel ist, um Anwendungsstrategien und insbesondere Nichtdeterminismen abzubilden. Diese sind aber ein wichtiger Aspekt terminologischer Sprachen.

[1] Eine ausführliche Motivation für Constraint Simplification Rules findet man in [9].

Wir werden in Anlehnung an die CSiRs sogenannte „Simplification Rules“ (SiR) so definieren, daß sich die genannten Aspekte darin einbetten lassen. Mit diesen SiR beschreiben wir dann exemplarisch den Konsistenztest für A-Boxen. Dadurch wird ein Abarbeitungsmechanismus motiviert, der anschließend definiert und implementiert wird. Insbesondere wird sich dieser Abarbeitungsmechanismus als flexibel genug erweisen, weitere Wissensrepräsentationsformalismen (z.B. logische Programme à la PROLOG) zu realisieren.

Diese Arbeit gliedert sich wie folgt:
In Kapitel 2 formalisieren wir die Idee der CSiR und definieren verschiedene Sprech- und Schreibweisen, die sich im Umgang mit diesen Regeln als praktisch erwiesen haben. In Kapitel 3 erinnern wir an die Definition der Konzeptbeschreibungssprachen und zeigen auf inwieweit deren Implementation eine Erweiterung der Constraint Simplification Rules notwendig macht. Im Anschluß daran erweitern wir unseren Formalismus. In Kapitel 4 definieren wir einen Abarbeitungsmechanismus für den zuvor entwickelten Formalismus, der zur Bearbeitung terminologischer Wissensrepräsentationssysteme besonders geeignet ist. Danach zeigen wir anhand einer Reihe von Beispielen in Kapitel 5 Anwendungsmöglichkeiten für Simplification Rules. Im Anhang ist eine Implementation des in Kapitel 4 definierten Abarbeitungsmechanismus in COMMON LISP beschrieben.

Kapitel 2

Simplification Rules

2.1 Unstrukturierte Trägermengen

In diesem Kapitel werden wir die Idee der Constraint Simplification Rules formalisieren. Dabei werden wir zuerst unstrukturierte Trägermengen zugrunde legen und die so erhaltenen Definitionen danach auf Trägermengen übertragen, die eine Termstruktur haben. Weiterhin führen wir eine Reihe von Sprech- und Schreibweisen ein, die uns den Umgang mit diesen Regeln erleichtern.

Definition 2.1 (Augmentationsregeln und Ersetzungsregeln)
Sei $\mathcal{F}$ eine nichtleere Trägermenge, sei $F \subseteq \mathcal{F}$ eine Teilmenge von $\mathcal{F}$, sei SR $= (\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$ *ein Tripel von Mengen, wobei* $\mathfrak{A} = \{A_1, \ldots, A_i\}, \mathfrak{B} = \{B_1, \ldots, B_k\}$, $\mathfrak{C} = \{C_1, \ldots, C_j\}$. *Die Elemente von $\mathfrak{C}$ seien boole'sche Ausdrücke, die Elemente von $\mathfrak{A}$ und $\mathfrak{B}$ seien aus $\mathcal{F}$.*

(i) Eine durch SR *induzierte* **Ersetzungsregel** *ist eine Abbildung*

$$\begin{array}{rccl} Rep_{\text{SR}}: & \mathfrak{Pot}(\mathcal{F}) & \longrightarrow & \mathfrak{Pot}(\mathcal{F}) \\ & F & \mapsto & \begin{cases} F, \textit{ falls } Rep_{\text{SR}} \textit{ nicht anwendbar auf } F \\ (F \setminus \mathfrak{A}) \cup \mathfrak{B}, \textit{ sonst} \end{cases} \end{array}$$

(ii) Eine durch SR *induzierte* **Augmentationsregel**[1] *ist eine Abbildung*

$$\begin{array}{rccl} Aug_{\text{SR}}: & \mathfrak{Pot}(\mathcal{F}) & \longrightarrow & \mathfrak{Pot}(\mathcal{F}) \\ & F & \mapsto & \begin{cases} F, \textit{ falls } Aug_{\text{SR}} \textit{ nicht anwendbar auf } F \\ F \cup \mathfrak{B}, \textit{ sonst} \end{cases} \end{array}$$

(iii) Eine durch SR *induzierte Augmentations- bzw. Ersetzungsregel heißt* **anwendbar auf eine Menge** *F, falls*

(a) $\mathfrak{A} \subseteq F$

und

(b) für alle $C \in \mathfrak{C}$ gilt: C ist wahr

(iv) Für Ersetzungs- bzw. Augmentationsregeln heißt SR **Definitionstripel**.

[1] Augmentation = Vermehrung

Mit anderen Worten: Rep_{SR} ersetzt die Elemente von $\mathfrak{A}$ in F durch die Elemente von $\mathfrak{B}$. Aug_{SR} erweitert F um die Elemente von $\mathfrak{B}$.

Vereinbarungen: Abkürzend schreiben wir eine Ersetzungsregel auch als

$$Rep : \mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}$$

bzw.

$$Rep : A_1, \ldots, A_i \Longleftrightarrow C_1, \ldots, C_j \mid B_1, \ldots, B_k.$$

und eine Augmentationsregel als

$$Aug : \mathfrak{A} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}$$

bzw.

$$Aug : A_1, \ldots, A_i \Longrightarrow C_1, \ldots, C_j \mid B_1, \ldots, B_k.$$

Ist $\mathcal{S}$ eine durch $SR = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$ induzierte Simplification Rule[2], so nennen wir die Elemente von $\mathfrak{A}$ **Head-Atome**, die Elemente von $\mathfrak{C}$ **Guard-Atome** und die Elemente von $\mathfrak{B}$ **Body-Atome**. Die Elemente von $\mathcal{F}$ nennen wir **Fakten**. Wenn wir Aussagen über Ersetzungsregeln und/oder Augmentationsregeln machen wollen, sprechen wir allgemein von **Simplification Rules** (SiR), wenn die Unterscheidung nicht wichtig ist. Simplification Rules werden wir auch kurz **Regeln** nennen. Um deutlich zu machen auf welcher Trägermenge $\mathcal{F}$ eine Simplification Rule $\mathcal{S}$ operiert sprechen wir auch von einer Regel *über* $\mathcal{F}$.

Den Abbildungsvorgang einer Faktenmenge F auf eine Faktenmenge F' mit Hilfe einer Simplification Rule $\mathcal{S}$ ($\mathcal{S}(F) = F'$) nennen wir **Anwenden der Regel $\mathcal{S}$ auf die Faktenmenge** F. Um deutlich zu machen welche $f \in F$ den Head-Atomen der Regel entsprechen, sagen wir auch „*wir wenden die Regel auf* $f_1, \ldots, f_i$ *an*“ ($f_1, \ldots, f_i \in F$), und meinen damit $\mathcal{S}(F) = F'$ wobei die $f_1, \ldots, f_i$ den Head-Atomen der Regel entsprechen (in Anlehnung an die Definition der Anwendbarkeit). Analog sagen wir „$\mathcal{S}$ *ist auf* $\{f_1, \ldots, f_i\} \subseteq F$ *anwendbar*“, wenn S auf F anwendbar ist und $\mathfrak{A} = \{f_1, \ldots, f_i\}$.

Definition 2.2 (Auswahlstrategie)
Sei $\mathcal{F}$ eine Trägermenge. Sei $\mathfrak{S}$ eine Menge von Simplification Rules über $\mathcal{F}$. Ferner sei $F \subset \mathcal{F}$ eine Menge von Fakten. Eine **Auswahlstrategie** *α (bzgl. $\mathfrak{S}$) über $\mathcal{F}$ ist eine Abbildung*

$$\alpha_{\mathfrak{S}} : \mathfrak{Pot}(\mathcal{F}) \longrightarrow \{\mathfrak{S} \times \mathfrak{Pot}(\mathcal{F})\} \bigcup \{\uparrow\}$$

mit folgenden Eigenschaften:

- $\alpha_{\mathfrak{S}}(F) = \uparrow$, *falls kein $\mathcal{S} \in \mathfrak{S}$ auf F anwendbar ist.*
- *Ist $\alpha_{\mathfrak{S}}(F) = (\mathcal{S}, \{f_1, \ldots, f_i\})$, so ist $\mathcal{S}$ auf $\{f_1, \ldots, f_i\}$ anwendbar.*

[2](d.h. $\mathcal{S} : \mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}$ oder $\mathcal{S} : \mathfrak{A} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}$)

Ist $\alpha_{\mathfrak{S}}(F) = \uparrow$, *so heißt* F **stabil** *bezüglich* $\mathfrak{S}$ *(und* α*).*

Vereinbarungen: Statt $\alpha_{\mathfrak{S}}$ schreiben wir auch α. Sind $\mathfrak{S}$ und α gegeben, dann bedeutet $\mathfrak{S}(F) = F'$:

$$\alpha_{\mathfrak{S}}(F) \neq \uparrow \text{ und } \pi_1(\alpha_{\mathfrak{S}}(F))(F) = F'^{3}$$

Statt $\pi_1(\alpha_{\mathfrak{S}}(F))(F) = F'$ schreiben wir auch $F \longrightarrow_\alpha F'$. $\mathfrak{S}^m(F) = F^m$ ist dann kurz für $F \longrightarrow_\alpha F' \longrightarrow_\alpha F'' \longrightarrow_\alpha \cdots \longrightarrow_\alpha F^m$. Von besonderem Interesse werden solche Regelmengen $\mathfrak{S}$ sein, für die es eine Auswahlstrategie gibt, so daß sich alle Faktenmengen stabilisieren lassen. D.h. zu jedem $F \in \mathfrak{Pot}(\mathcal{F})$ muß es ein $m \in \mathfrak{N}$ [4] geben so daß $\mathfrak{S}^m(F)$ stabil ist. In Paragraph 3.1 nennen wir die Suche nach diesem m „Vervollständigung einer A-Box".

Bemerkung 2.3 *Da bei der Anwendung einer Augmentationsregel die Faktenmenge nur erweitert wird, insbesondere sich die Head-Atome, die die Anwendbarkeit entscheiden, nicht verändern, läßt sich eine Augmentationsregel immer wieder auf dieselben Fakten anwenden*[5]*. Um diese trivialen Zyklen zu vermeiden, sollte man sicherstellen (z.B. durch die Auswahlstrategie), daß eine Regel nur einmal auf* dieselben *Fakten angewendet wird. (Vergleiche dazu auch Beispiel 2.7.)*

2.2 Trägermengen mit Termstruktur

Um die Simplification Rules ausdrucksstärker zu machen, wollen wir im Folgenden auf Head-, Guard- und Body-Atomen eine Termstruktur definieren. Dies geschieht in der üblichen Form (siehe beispielsweise [25], [27]):

Definition 2.4 (Term)
Seien VAR, KONST *und* FUNC *paarweise disjunkte Mengen von Variablen-, Konstanten- bzw. Funktionssymbolen.*

1. *Jede Variable ist ein Term. Jede Konstante ist ein Term.*

2. *Sind* $t_1, \ldots, t_n$ *Terme und ist* f *ein* n*-stelliges Funktionssymbol, so ist* $ft_1, \ldots, t_n$ *ein Term.*

[3] Dabei ist π_1 die Projektion $(a, b) \mapsto a$.
[4] $\mathfrak{N}$ bezeichnet die natürlichen Zahlen.
[5] Es sei denn man fängt das in den Guard-Atomen ab.

Ein Term, der keine Variablen enthält, heißt **Grundterm**. Eine **Substitution** ist eine Abbildung von einer Menge von Variablen in die Terme. Eine **Grundsubstitution** bildet Variable in Grundterme ab. Ist t ein Term, g ein Grundterm und σ eine Substitution, so daß $\sigma(t) = g$, dann nennt man σ einen **Matcher**[6] und man sagt „t matcht g mittels σ". Die Suche nach einem Matcher nennen wir **matchen**. Damit definieren wir Simplification Rules jetzt so:

Definition 2.5 (zulässige Simplification Rules)
Ein Definitionstripel $\text{SR} = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$ *von Termmengen heißt* **zulässig**, *wenn jede Grundsubstitution der Variablen aus* $\mathfrak{A}$ *und* $\mathfrak{C}$ *auch Grundsubstitution der Variablen aus* $\mathfrak{B}$ *ist.* [7]

Eine Simplification Rule heißt **zulässig**, *wenn sie von einem zulässigen Definitionstripel induziert wird.*

Mit anderen Worten eine Simplification Rule ist genau dann zulässig, wenn jede Variable, die in den Body-Atomen vorkommt, auch in den Head- oder Guard-Atomen vorkommt.

Definition 2.6 (Augmentationsregeln und Ersetzungsregeln)
Sei $\mathcal{F}$ *eine nichtleere Menge von Termen, sei* $F \subseteq \mathcal{F}$ *eine Teilmenge von* $\mathcal{F}$, *die nur aus Grundtermen besteht. Sei* $\text{SR} = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$ *ein zulässiges Definitionstripel, wobei* $\mathfrak{A} = \{A_1, \ldots, A_i\}, \mathfrak{B} = \{B_1, \ldots, B_k\}, \mathfrak{C} = \{C_1, \ldots, C_j\}$. *Die Elemente von* $\mathfrak{C}$ *seien boole'sche Ausdrücke, die Elemente von* $\mathfrak{A}$ *und* $\mathfrak{B}$ *seien Elemente von* $\mathcal{F}$.

(i) Eine durch SR *induzierte Simplification Rule heißt* **anwendbar auf eine Menge** F, *falls es eine (Grund-)Substitution* σ *gibt, so daß*

(a) $\sigma(A_1), \ldots, \sigma(A_i) \in F$ und

(b) $\sigma(C)$ *wahr ist für alle* $C \in \mathfrak{C}$.

(ii) Eine durch SR *induzierte* **Ersetzungsregel** *ist eine Abbildung*

$$\begin{aligned} Rep_{\text{SR}}: \quad \mathfrak{Pot}(\mathcal{F}) &\longrightarrow \mathfrak{Pot}(\mathcal{F}) \\ F &\mapsto \begin{cases} F, \text{ falls } Rep_{\text{SR}} \text{ nicht anwendbar auf } F \\ (F \setminus \sigma(\mathfrak{A})) \bigcup \sigma(\mathfrak{B}), \text{ sonst} \end{cases} \end{aligned}$$

(iii) Eine durch SR *induzierte* **Augmentationsregel** *ist eine Abbildung*

$$\begin{aligned} Aug_{\text{SR}}: \quad \mathfrak{Pot}(\mathcal{F}) &\longrightarrow \mathfrak{Pot}(\mathcal{F}) \\ F &\mapsto \begin{cases} F, \text{ falls } Aug_{\text{SR}} \text{ nicht anwendbar auf } F \\ F \bigcup \sigma(\mathfrak{B}), \text{ sonst} \end{cases} \end{aligned}$$

Dabei steht $\sigma(\mathfrak{A})$ *für* $\{\sigma(A_1), \ldots, \sigma(A_i)\}$, $\sigma(\mathfrak{B})$ *für* $\{\sigma(B_1), \ldots, \sigma(B_k)\}$.

[6] Bedauerlicherweise ist es manchmal nötig englische Begriffe einzudeutschen, wenn ein adäquater deutscher Begriff nicht zur Verfügung steht.

[7] Die Grundsubstitution eines boole'schen Ausdrucks ist eine Substitution, die jede Variable des Ausdrucks durch einen Grundterm ersetzt.

Die Begriffsbildungen aus Paragraph 2.1 übertragen wir analog.

Bemerkung: Die in Definition 2.6 (i) genannte Substitution ist im allgemeinen nicht eindeutig.[8] Um eine Simplification Rule $\mathcal{S}$ auf F anwenden zu können, muß man gegebenenfalls eine Grundsubstitution auswählen. Die in Definition 2.2 eingeführte Auswahlstrategie soll in diesem Fall die anzuwendende Regel, die Fakten, auf die die Regel angewendet werden soll, *und* die Grundsubstitution liefern.

Beachte, daß Ersetzungsregeln und Augmentationsregeln nur für *zulässige* Definitionstripel definiert sind. Das ist nötig um die Grundterm-Eigenschaft auf der manipulierten Faktenmenge nicht zu verlieren.

Beispiel 2.7

gegeben:

$\underline{\text{VAR}}$	$: \{x, y, z, x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots\}$	eine Menge von Variablen
$\underline{\text{KONST}}$	$: \{a, b, c, a_1, a_2, \ldots, b_1, b_2, \ldots, c_1, c_2, \ldots\}$	Konstanten
$\underline{\text{FUNC}}$	$: \{<, \leq, =\}$	Funktionssymbolen

Sei eine Trägermenge gegeben durch $\mathcal{F} : \{(u\ f\ v) \mid f \in \underline{\text{FUNC}}, u, v \in \underline{\text{KONST}}\}$.

$$\text{Sei } \mathfrak{S} = \left\{ \begin{array}{llll} \mathit{Rep1} : (x \leq y), (y \leq x) & \Longleftrightarrow & & \mid (x = y) \\ \mathit{Rep2} : (x \leq y), (x < y) & \Longleftrightarrow & & \mid (x < y) \\ \mathit{Aug1} : (x \leq y), (y \leq z) & \Longrightarrow & (x \leq z) \notin F^* & \mid (x \leq z) \\ \mathit{Aug2} : (x < y), (y < z) & \Longrightarrow & (x < z) \notin F^* & \mid (x < z) \end{array} \right.$$

eine Menge von Simplification Rules über $\mathcal{F}$ und sei α eine Auswahlstrategie.[9] F^* bezeichne die aktuelle Faktenbasis. Wir wenden $\mathfrak{S}$ solange auf

$$F := \{(a \leq b), (a_1 < b_1), (b_1 < c_1), (a_2 \leq b_2), (b \leq c), (b_2 \leq a_2)\}$$

an, bis keine Regel aus $\mathfrak{S}$ mehr anwendbar ist. Beachte dabei, daß wir hier durch die Guard-Atome der Augmentationsregeln Aug_1 und Aug_2 eine mehrmalige Anwendung auf *dieselben* Fakten verhindern. Es ist jedoch auch möglich und im allgemeinen auch sinnvoll mehrmaliges Anwenden durch die Auswahlstrategie zu verhindern.[10] Wir wollen jedoch in diesem Beispiel die Auswahlstrategie so vage wie möglich lassen. Regeln und Fakten werden hier willkürlich ausgewählt.

---

[8] *Beispiel:* R: $x \leq y, y \leq z \Longrightarrow \mid x \leq z$, $F : \{a \leq b,\ b \leq c,\ d \leq e,\ e \leq f\}$ dann ist R sowohl auf $\{a \leq b,\ b \leq c\}$ $(\sigma = (x \leftarrow a, y \leftarrow b, z \leftarrow c))$ als auch auf $\{d \leq e,\ e \leq f\}$ $(\sigma = (x \leftarrow d, y \leftarrow e, z \leftarrow f))$ anwendbar.

[9] $\mathfrak{S}$ ist nur ein unvollständiges Regelsystem, das wir benutzen, um die Anwendung von SiR zu veranschaulichen.

[10] Betrachte folgendes Beispiel: Seien A: f,g $\Longrightarrow$ h $\notin F^* \mid$ h und E: h $\Longleftrightarrow \mid$ h' SiR über einer Trägermenge $\mathcal{F}$ und sei A anwendbar auf F. Dann lassen sich E und A immer abwechselnd auf F^* anwenden. So etwas muß man mit der Auswahlstrategie abfangen.

$$
\begin{array}{l}
F : \{(a \leq b), (a_1 < b_1), (b_1 < c_1), (a_2 \leq b_2), (b \leq c), (b_2 \leq a_2)\} \\
\qquad\qquad \Big| \quad \alpha(F) = (Rep_1, \{(a_2 \leq b_2), (b_2 \leq a_2)\}) \\
\downarrow \\
F' : \{(a \leq b), (a_1 < b_1), (b_1 < c_1), (b \leq c), (a_2 = b_2)\} \\
\qquad\qquad \Big| \quad \alpha(F') = (Aug_1, \{(a \leq b), (b \leq c)\}) \\
\downarrow \\
F'' : \{(a \leq b), (a_1 < b_1), (b_1 < c_1), (b \leq c), (a_2 = b_2), (a \leq c)\} \\
\qquad\qquad \Big| \quad \alpha(F'') = (Aug_2, \{(a_1 < b_1), (b_1 < c_1)\}) \\
\downarrow \\
F''' : \{(a \leq b), (a_1 < b_1), (b_1 < c_1), (b \leq c), (a_2 = b_2), (a \leq c), (a_1 < c_1)\} \\
\qquad\qquad \Big| \quad \alpha(F''') = \uparrow \\
\bot
\end{array}
$$

Fassen wir noch einmal zusammen:

- Rep: $\mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}$ heißt Ersetzungsregel.
- Aug: $\mathfrak{A} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}$ heißt Augmentationsregel.
- Die Elemente von $\mathfrak{A}$ $(A_1, \ldots, A_i)$ heißen Head-Atome, die Elemente von $\mathfrak{C}$ $(C_1, \ldots, C_j)$ heißen Guard-Atome, die Elemente von $\mathfrak{B}$ $(B_1, \ldots, B_k)$ heißen Body-Atome.
- Eine Simplification Rule ist entweder eine Ersetzungsregel oder eine Augmentationsregel.
- Eine Ersetzungsregel auf eine Faktenmenge anwenden bedeutet: Vorausgesetzt die Guard-Atome treffen zu, ersetze die Head-Atome durch die Body-Atome.
- Eine Augmentationsregel auf eine Faktenmenge anwenden bedeutet: Vorausgesetzt die Guard-Atome treffen zu, erweitere die Faktenmenge um die Body-Atome.
- Ist $\mathfrak{S}$ eine Menge von Simplification Rules, so ist eine Faktenmenge F stabil bzgl. $\mathfrak{S}$, falls keine Regel aus $\mathfrak{S}$ auf F anwendbar ist.
- Wir definieren auf den Fakten eine Termstruktur und schränken uns auf Mengen von Grundtermen ein. Durch die Einschränkung auf zulässige Regeln ist die Faktenmenge abgeschlossen bezüglich Regelanwendungen.

Kapitel 3

Anwendungen von Simplification Rules

3.1 Terminologische Wissensrepräsentationssysteme

3.1.1 Die Konzeptbeschreibungssprache $\mathcal{ALC}$

Um diese Arbeit abgeschlossen zu halten, erinnern wir kurz an die Definition der Konzeptbeschreibungssptache $\mathcal{ALC}$ wie sie von Manfred Schmitt-Schauß und Gert Smolka in [26] eingeführt wird. Wir orientieren uns dabei an den in [2] benutzten Schreibweisen.

Definition 3.1 (Konzeptterme und Termimologien von $\mathcal{ALC}$)
Sei $\mathfrak{C}$ eine Menge von Konzeptnamen, sei $\mathfrak{R}$ eine Menge von Rollennamen, die disjunkt zu $\mathfrak{C}$ ist. Die Menge der **Konzeptterme** *von $\mathcal{ALC}$ wird induktiv definiert:*

1. *Jedes $C \in \mathfrak{C}$ ist ein Konzeptterm (atomare Terme).*
2. *Seien C und D Konzeptterme, sei $R \in \mathfrak{R}$ ein Rollenname, dann sind folgende Ausdrücke ebenfalls Konzeptterme:*
   - $C \sqcup D$ *(Disjunktion)*
   - $C \sqcap D$ *(Konjunktion)*
   - $\neg C$ *(Negation)*
   - $\exists R.C$ *(Existenzrestriktion)*
   - $\forall R.C$ *(Werterestriktion)*

Sei $A \in \mathfrak{C}$ ein Konzeptname und sei D ein Konzeptterm, dann ist

$$A \doteq D$$

ein **terminologisches Axiom**[1]. *Eine* **Terminologie** *(oder auch T-Box) ist eine endliche Menge $\mathcal{T}$ terminologischer Axiome mit den Einschränkungen, daß*

1. *kein Konzeptname mehr als einmal auf der linken Seite einer Definition vorkommt und*
2. *$\mathcal{T}$ keine zyklischen Definitionen enthält.*

Beispiel 3.2 (Terminologie einer Familie)

$$
\begin{aligned}
\text{weiblich} &\doteq \neg\text{männlich} \\
\text{neutral} &\doteq \text{weiblich} \sqcap \text{männlich} \\
\text{Frau} &\doteq \text{menschlich} \sqcap \text{weiblich} \\
\text{Mann} &\doteq \text{menschlich} \sqcap \text{männlich} \\
\text{Mutter} &\doteq \text{Frau} \sqcap \exists\text{hat-Kind.menschlich} \\
\text{Vater} &\doteq \text{Mann} \sqcap \exists\text{hat-Kind.menschlich} \\
\text{Elternteil} &\doteq \text{Mutter} \sqcup \text{Vater} \\
\text{Großmutter} &\doteq \text{Mutter} \sqcap \exists\text{hat-Kind.Elternteil} \\
\text{Vater-von-Söhnen} &\doteq \text{Vater} \sqcap \forall\text{hat-Kind.männlich}
\end{aligned}
$$

Mit dieser T-Box wollen wir folgendes „Wissen" beschreiben: weiblich ist das Gegenteil von männlich, neutral ist weiblich und männlich, eine Frau ist ein weiblicher Mensch, ein Mann ist ein männlicher Mensch, eine Mutter ist eine Frau, die ein menschliches Kind hat, ein Vater ist ein Mann, der ein menschliches Kind hat, ein Elternteil ist eine Mutter oder ein Vater, eine Großmutter ist eine Mutter, die ein Kind hat, das ein Elternteil ist, ein Vater-von-Söhnen ist ein Vater, dessen Kinder alle männlich sind.

Konzeptnamen, die nicht auf der linken Seite der Definitionen erscheinen, nennen wir **primitive Konzepte**, die anderen Konzeptnamen nennen wir **definierte Konzepte**. In Beispiel 3.2 sind männlich und menschlich primitive Konzepte, weiblich, Frau, Mann, Mutter, Vater, Elternteil und Großmutter sind definierte Konzepte und hat-Kind ist ein Rollenname.

Definition 3.3 (Interpretationen und Modelle)
Eine **Interpretation** *$\mathcal{I}$ für $\mathcal{ALC}$ ist ein Paar $(dom(\mathcal{I}), \cdot^{\mathcal{I}})$ bestehend aus einer Trägermenge $dom(\mathcal{I})$ und einer Interpretationsfunktion $\cdot^{\mathcal{I}}$. Die Interpretationsfunktion hat folgende Eigenschaften:*

1. *Jedem Konzeptnamen $A \in \mathfrak{C}$ wird eine Teilmenge $A^{\mathcal{I}}$ von $dom(\mathcal{I})$ zugeordnet.*
2. *Jedem Rollennamen $R \in \mathfrak{R}$ wird eine binäre Relation $R^{\mathcal{I}}$ auf $dom(\mathcal{I})$ zugeordnet, d.h. eine Teilmenge von $dom(\mathcal{I}) \times dom(\mathcal{I})$.*

[1] Ein terminologisches Axiom nennen wir manchmal auch Definition.

3. *Für beliebige Konzepterme wird die Interpretationsfunktion so definiert: Angenommen $C^{\mathcal{I}}$ und $D^{\mathcal{I}}$ seien schon definiert, dann ist*

- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = dom(\mathcal{I}) \setminus C^{\mathcal{I}}$
- $(\forall R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid (\forall y\ (x,y) \in R^{\mathcal{I}} \longrightarrow y \in C^{\mathcal{I}})\}$
- $(\exists R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid (\exists y\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$

Eine Interpretation $\mathcal{I}$ ist ein **Modell** *für eine T-Box $\mathcal{T}$ definitionsgemäß genau dann, wenn für alle terminologischen Axiome $A \doteq D$ aus $\mathcal{T}$ gilt $A^{\mathcal{I}} = D^{\mathcal{I}}$.*

Von zentraler Bedeutung ist die Frage, ob ein Konzeptterm C erfüllbar ist, d.h. gibt es eine Interpretation $\mathcal{I}$ mit $C^{\mathcal{I}} \neq \emptyset$? Mit anderen Worten ein unerfüllbarer Konzeptterm wird von *jeder* Interpretation auf die leere Menge abgebildet, er ist also wertlos. (In Beispiel 3.2 ist neutral unerfüllbar.) Wir werden den Erfüllbarkeitstest eines Konzepttermes auf ein anderes Problem zurückführen: den Konsistenztest für A-Boxen. Dafür werden wir einen korrekten und vollständigen Algorithmus angeben. Für diesen Algorithmus ist es vorteilhaft, wenn die Konzeptterme in negationaler Normalform vorliegen. Ein Konzeptterm ist in **negationaler Normalform (NNF)**, falls Negationszeichen nur direkt vor Konzeptnamen auftreten. Jeder Konzeptterm von $\mathcal{ALC}$ läßt sich mit Hilfe folgender Transformationsregeln in einen äquivalenten Term in NNF umformen:

$$\begin{aligned}
\neg(C \sqcap D) &\longrightarrow ((\neg C) \sqcup (\neg D)) \\
\neg(C \sqcup D) &\longrightarrow ((\neg C) \sqcap (\neg D)) \\
\neg(\neg C) &\longrightarrow C \\
\neg(\exists R.C) &\longrightarrow (\forall R.\neg C) \\
\neg(\forall R.C) &\longrightarrow (\exists R.\neg C)
\end{aligned}$$

Mit Hilfe von Definition 3.3 überzeugt man sich leicht von der Äquivalenz der Transformation.

3.1.2 Terminologisches Schließen

Die wesentlichen Dienste terminologischer Repräsentationssysteme lassen sich auf den Konsistenztest für A-Boxen zurückführen. Dazu definieren wir zunächst die benötigten Begriffe, beschreiben kurz wie terminologische Inferenzdienste auf den Konsistenztest zurückgeführt werden und geben dann einen korrekten und vollständigen Algorithmus an, der eine gegebene A-Box auf Konsistenz testet.

Definition 3.4 (Assertionale Axiome und A-Boxen für $\mathcal{ALC}$)
Sei $\mathfrak{O}$ eine Menge von Objektnamen[2], seien $a, b \in \mathfrak{O}$, sei C ein Konzeptterm über $\mathcal{ALC}$, sei R ein Rollenname, dann sind

$$(i) \quad a : C \qquad \textit{und} \qquad (ii) \quad (a, b) : R$$

assertionale Axiome.[3] *Eine* **A-Box** *ist eine endliche Menge assertionaler Axiome. Die Axiome der Form (i) nennen wir auch* **membership assertions**, *die Axiome der Form (ii)* **role filler assertions.**

Die assertionale Sprache kann beispielsweise benutzt werden um auszudrücken, daß Charles der Sohn von Elizabeth ist und Diana mit Charles verheiratet ist:

CHARLES	:	Mann
(ELIZABETH, CHARLES)	:	hat-Kind
ELIZABETH	:	Mutter
(DIANA, CHARLES)	:	sind-verheiratet

Hier sind CHARLES, DIANA und ELIZABETH Elemente von $\mathfrak{O}$, Mann und Mutter sind Konzeptterme und sind-verheiratet und hat-Kind sind Rollennamen.

Definition 3.5 (Interpretationen und Modelle)
Eine **Interpretation** *für eine assertionale Sprache (A-Box) (über $\mathcal{ALC}$) ist eine Interpretation für $\mathcal{ALC}$, die zusätzlich jedem Objektnamen a von $\mathfrak{O}$ ein Objekt $a^{\mathcal{I}} \in dom(\mathcal{I})$ zuordnet. Solch eine Interpretation erfüllt ein assertionales Axiom*

$a : C$ genau dann, wenn $a^{\mathcal{I}} \in C^{\mathcal{I}}$

$(a, b) : R$ genau dann, wenn $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Eine Interpretation ist ein **Modell** *einer A-Box $\mathcal{A}$ genau dann, wenn sie alle assertionalen Axiom von $\mathcal{A}$ erfüllt. Eine Interpretation ist eine Modell einer A-Box $\mathcal{A}$ zusammen mit einer T-Box $\mathcal{T}$ genau dann, wenn sie Modell von $\mathcal{T}$ und Modell von $\mathcal{A}$ ist.*

Eine gegebene A-Box (eine A-Box zusammen mit einer T-Box) hat nicht notwendigerweise ein Modell. Betrachten wir dazu die A-Box $\mathcal{A}$, die die Axiome $a : C$

[2] Die Elemente von $\mathfrak{O}$ nennen wir auch Individuen.
[3] Assertion (engl.) bedeutet Zusicherung.

und $a : \neg C$ enthält. Diese A-Box ist widersprüchlich und kann demzufolge kein Modell haben. Wir nennen eine A-Box (eine A-Box zusammen mit einer T-Box) **konsistent**, wenn sie ein Modell hat. Anderenfalls nennen wir sie **inkonsistent**.

Wir werden im Folgenden einen Algorithmus kennenlernen, der eine gegebene A-Box (eine A-Box zusammen mit einer T-Box) auf Konsistenz testet. Dieser Konsistenztest ist deshalb von zentraler Bedeutung, weil sich andere wichtige Ableitungsprobleme auf ihn zurückführen lassen (siehe auch [13] Seite 11 ff):

1. *Erfüllbarkeit eines Konzeptes C:* Gibt es ein Modell, das C als nicht-leere Menge interpretiert? Ein Konzeptterm ist genau dann erfüllbar, wenn die A-Box $\{a : C\}$ konsistent ist.

2. *Subsumption zweier Konzepte C und D:* Kennzeichnet C in *allen* Modellen eine Obermenge von D? C subsumiert D genau dann, wenn die A-Box $\{a : \neg C, a : D\}$ inkonsistent ist.

3. *Bestimmung der Taxonomie:* Wie lassen sich die eingeführten Konzepte bzgl. Subsumption ordnen? Das läßt sich durch mehrfaches Anwenden von 2. lösen.

4. *Instanzentest:* Liegt ein Individuum a in einem Konzept C? a liegt in C im Hinblick auf eine A-Box $\mathcal{A}$ genau dann, wenn die A-Box $\mathcal{A} \cup \{a : \neg C\}$ inkonsistent ist.

5. *Realisierung:* Welches sind die kleinsten[4] Konzepte in denen ein Individuum liegt? Dieses Problem kann mit Hilfe von 2. und 4. gelöst werden.

6. *Retrieval:* Welche Individuen liegen in einem bestimmten Konzept? Dieses Problem läßt sich mit Hilfe von 4. lösen.

3.1.3 Der Konsistenztest für A-Boxen

Der Konsistenztest einer A-Box zusammen mit einer T-Box läßt sich leicht auf den Konsistenztest für A-Boxen allein zurückführen. Man muß nur die Konzeptterme der A-Box **expandieren**. D.h. man ersetzt solange sukzessive die Namen der definierten Konzepte durch ihre Definitionsterme aus der T-Box, bis nur noch Individuen, Rollen und primitive Konzepte in der A-Box auftreten.

Beispiel 3.6
Sei $\mathcal{A}$=$\{$*ELIZABETH :* Mutter, *CHARLES :* Mann$\}$ *eine A-Box und* $\mathcal{T}$ *die T-Box aus Beispiel 3.2, dann ist* $\mathcal{A}$'=$\{$*ELIZABETH :* menschlich $\sqcap$ $\neg$ männlich $\sqcap$ $\exists$hat-Kind.menschlich, *CHARLES :* menschlich $\sqcap$ männlich$\}$ *die aus* $\mathcal{A}$ *expandierte A-Box.*

[4] bzgl. Subsumption

Sei $\mathcal{A}_0$ eine (expandierte) A-Box (über $\mathcal{ALC}$). Ohne Einschränkung können wir annehmen, daß die auftretenden Konzeptterme in NNF vorliegen. Der Algorithmus beginnt mit einer gegebenen A-Box und manipuliert sie mit Hilfe bestimmter Regeln[5] bis einer der folgenden Fälle eintritt:

(a) Die modifizierte A-Box ist „offensichtlich widersprüchlich".

(b) Die modifizierte A-Box ist vollständig in dem Sinn, daß sich keine Modifikationsregel mehr anwenden läßt. In diesem Fall beschreibt die vollständige A-Box ein Modell der anfänglichen A-Box.

Da wir in unserer Konzeptsprache Disjunktionen zulassen, muß eine A-Box gegebenenfalls in zwei verschiedene neue A-Boxen transformiert werden. Deshalb arbeiten wir mit Mengen $\mathcal{M}$ von A-Boxen statt mit einer einzelnen A-Box. Um eine A-Box $\mathcal{A}_0$ auf Konsistenz zu testen, beginnen wir mit $\mathcal{M}_0 := \{\mathcal{A}_0\}$.

Definition 3.7 (Transformationsregeln)
Sei $\mathcal{M}$ eine endliche Menge von A-Boxen und sei $\mathcal{A}$ ein Element von $\mathcal{M}$. Die folgenden Regeln ersetzen die A-Box $\mathcal{A}$ durch eine A-Box $\mathcal{A}'$ oder zwei A-Boxen $\mathcal{A}'$ und $\mathcal{A}''$. In der Formulierumg der Regeln stehen a, b für die Namen von Individuen, C, D für Konzeptterme und R für einen Rollennamen.

1. Konjunktionsregel. *Angenommen $a : (C \sqcap D)$ ist in $\mathcal{A}$, dann erhält man $\mathcal{A}'$ aus $\mathcal{A}$ durch Hinzufügen der Axiome $a : C$ und $a : D$ zu $\mathcal{A}$.*

2. Disjunktionsregel. *Angenommen $a : (C \sqcup D)$ ist in $\mathcal{A}$, dann erhält man $\mathcal{A}'$ aus $\mathcal{A}$ durch Hinzufügen des Axioms $a : C$ zu $\mathcal{A}$, $\mathcal{A}''$ durch Hinzufügen des Axioms $a : D$ zu $\mathcal{A}$.*

3. Existenzrestriktionsregel. *Angenommen $a : (\exists R.C)$ ist in $\mathcal{A}$, dann erhält man $\mathcal{A}'$ aus $\mathcal{A}$ durch Hinzufügen der Axiome $(a, b) : R$ und $b : C$ zu $\mathcal{A}$. Dabei sei b ein neues Individuum (d.h. ein Individuenname, der noch nicht in $\mathcal{A}$ vorkommt).*

4. Werterestriktionsregel. *Angenommen $a : (\forall R.C)$ und $(a, b) : R$ sind in $\mathcal{A}$, dann erhält man $\mathcal{A}'$ aus $\mathcal{A}$ durch Hinzufügen des Axioms $b : C$ zu $\mathcal{A}$.*

Das Verfahren diese Regeln solange auf A-Box-Mengen $\mathcal{M}_0, \mathcal{M}_1, \ldots$ anzuwenden bis keine Regel mehr anwendbar ist nennen wir **Vervollständigung** der A-Box-Menge $\mathcal{M}_0$. Es gilt der folgende

Satz 3.8 *Jede A-Box-Menge läßt sich vervollständigen.*

Einen Beweis findet man z.B in [12].

[5] Das wird später genau der Punkt sein an dem die Simplification Rules ins Spiel kommen.

Wir präzisieren jetzt was es heißt, daß eine A-Box „offensichtlich widersprüchlich" ist.

Definition 3.9 (Clash-Regel)
Eine A-Box $\mathcal{A}$ ist **offensichtlich widersprüchlich** *(oder eine A-Box* **enthält einen Clash***), falls*[6]

- *$\mathcal{A}$ Axiome $a : C$ und $a : \neg C$ enthält für einen beliebigen Konzeptnamen C und einen beliebiges Individuum a. Das ist ein offensichtlicher Widerspruch, weil ein Objekt nicht gleichzeitig in einer Menge und ihrem Komplement liegen kann.*

Um eine A-Box $\mathcal{A}_0$ auf Konsistenz zu testen, bilden wir die A-Box-Menge $\mathcal{M}_0 := \{\mathcal{A}_0\}$, die dann mit Hilfe der Transformationsregeln 3.7 zu $\mathcal{M}_r$ vervollständigt wird. Dann gilt

Satz 3.10 *$\mathcal{A}_0$ ist genau dann konsistent, wenn es eine A-Box in $\mathcal{M}_r$ gibt, die keinen Clash enthält. Mit anderen Worten $\mathcal{A}_0$ ist genau dann inkonsistent, wenn* alle *A-Boxen in $\mathcal{M}_r$ offensichtlich widersprüchlich sind.*

Einen Beweis findet man z.B. in [2] Seite 19 ff. Der Konsistenztest läßt sich in Pseudo-Code so darstellen:

Algorithmus 3.11 (Konsistenztest) *Die folgende Prozedur entscheidet zu einer gegebenen A-Box $\mathcal{A}_0$, ob sie konsistent ist oder nicht.*

```
Procedure Konsistenztest(A_0);
    i := 0;
    M_0 := {A_0}
    while „eine Transformationsregel ist auf M_i anwendbar" do
        i := i + 1;
        M_i := wende-eine-Transformationsregel-an(M_{i-1})
    od;
    if   „es gibt ein A ∈ M_i, das keinen Clash enthält"
         then return „konsistent"
         else return „inkonsistent";
    fi
end Konsistenztest
```

Der Punkt ist, daß sich dieser Algorithmus auch für ausdrucksstärkere Konzeptsprachen (z.B. Erweiterungen von $\mathcal{ALC}$) benutzen läßt, indem man die Transformations- und Clash-Regeln erweitert.

[6] Wir wählen hier die Aufzählform um deutlich zu machen, daß bei Erweiterungen der Konzeptsprache hier *mehrere* Clash-Regeln stehen können.

3.2 Terminologisches Schließen und Simplification Rules

Fassen wir nocheinmal die wesentlichen Punkte des letzten Abschnitts zusammen.

- Wir definieren mit einer Konzeptbeschreibungssprache ($\mathcal{ALC}$) Terminologien (T-Boxen), die sich mit einfachen Rewrite-*Regeln* in eine Normalform (NNF) überführen lassen.(Seite 16)
- Mit Hilfe der Terminologien und darauf definierten Assertionen läßt sich Wissen repräsentieren, auf das sich mit verschiedenen Inferenzmechanismen (Seite 18) zugreifen läßt.
- Diese Inferenzmechanismen lassen sich auf den Konsistenztest sogenannter A-Boxen zurückführen.
- Das Herzstück des Konsistenztestes sind eine endliche Menge von Transformations- und Clash-*Regeln.*

Es liegt jetzt nahe zur Realisierung all dieser Regeln den Mechanismus der Simplification Rules heranzuziehen. Erinnern wir uns: Simplification Rules sind Abbildungen, die auf nichtleeren Mengen (den sogenannten Fakten) operieren. Bei der Realisierung der terminologischen Inferenzen mit Simplification Rules werden die Assertionen die Fakten bilden auf denen die Rewrite- Transformations- und Clash-Regeln — dargestellt als Simplification Rules — operieren. Im folgenden werden wir den Konsistenztest für A-Boxen mit Hilfe von Simplification Rules reformulieren. Wir wollen dabei den Nichtdeterminismus mit Hilfe der Simplification Rules codieren. Dazu übertragen wir die Rewrite-Regeln von Seite 16 die Clash-Regel von Seite 20 und die Transformationsregeln von Seite 19 in den Formalismus der Simplification Rules.

$$
\begin{array}{lrcll}
R_1: & ?x:\neg(?C \sqcap ?D) & \Longleftrightarrow & & \mid ?x:((\neg ?C) \sqcup (\neg ?D)) \\
R_2: & ?x:\neg(?C \sqcup ?D) & \Longleftrightarrow & & \mid ?x:((\neg ?C) \sqcap (\neg ?D)) \\
R_3: & ?x:\neg(\neg ?C) & \Longleftrightarrow & & \mid ?x:?C \\
R_4: & ?x:\neg(\exists ?R.?C) & \Longleftrightarrow & & \mid ?x:(\forall ?R.\neg ?C) \\
R_5: & ?x:\neg(\forall ?R.?C) & \Longleftrightarrow & & \mid ?x:(\exists ?R.\neg ?C) \\
R_6: & ?x:(?C \sqcap ?D) & \Longleftrightarrow & & \mid ?x:?C, ?x:?D \\
R_7: & ?x:(?C \sqcup ?D) & \Longleftrightarrow & & \mid \{?x:?C\} \bigvee \{?x:?D\} \\
R_8: & ?x:(\exists ?R.?C) & \Longleftrightarrow & ?y \notin \mathfrak{O}_F & \mid (?x,?y):?R, ?y:C \\
R_9: & ?x:(\forall ?R.?C), (?x,?y):?R & \Longrightarrow & & \mid ?y:?C \\
R_{10}: & ?x:?C, ?x:\neg ?C & \Longrightarrow & & \mid ?x:\bot
\end{array}
$$

Bemerkung 3.12

Die mit „?“ beginnenden Symbole sind Variablensymbole.

$R_1 - R_5$. Man sieht sofort, daß R_1–R_5 den Rewriteregeln von Seite 16 entsprechen.

$\mathbf{R_6}$ ist eine direkte Übersetzung der Konjunktionsregel.

$\mathbf{R_7}$. Mit unseren bisherigen Simplification Rules können wir keine nichtdeterministischen Funktionen darstellen. In Anlehnung an Definition 2.6 erweitern wir die Regeldefinitionen wie folgt:

Definition 3.13
Sei $\text{SR} = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$[7] *ein zulässiges Definitionstripel. Sei* $\mathfrak{B}$ *partitioniert in* $\mathfrak{B}_1 \cup \mathfrak{B}_2$.

(i) Eine (durch SR *induzierte)* **nichtdeterministische Ersetzungsregel** *ist eine nichtdeterministische Funktion*

$$NRep_{\text{SR}}: \mathfrak{Pot}(\mathcal{F}) \longrightarrow \mathfrak{Pot}(\mathcal{F})$$

$$F \mapsto \left\{ \begin{array}{c} F, \text{ falls } NRep_{\text{SR}} \text{ nicht anwendbar auf F} \\ \left.\begin{array}{c} (F \setminus \sigma(\mathfrak{A})) \cup \sigma(\mathfrak{B}_1) \\ \textit{oder} \\ (F \setminus \sigma(\mathfrak{A})) \cup \sigma(\mathfrak{B}_2) \end{array}\right\} \textit{sonst} \end{array} \right.$$

(ii) Eine (durch SR *induzierte)* **nichtdeterministische Augmentationsregel** *ist eine nichtdeterministische Funktion*

$$NAug_{\text{SR}}: \mathfrak{Pot}(\mathcal{F}) \longrightarrow \mathfrak{Pot}(\mathcal{F})$$

$$F \mapsto \left\{ \begin{array}{c} F, \text{ falls } NAug_{\text{SR}} \text{ nicht anwendbar auf F} \\ \left.\begin{array}{c} F \cup \sigma(\mathfrak{B}_1) \\ \textit{oder} \\ F \cup \sigma(\mathfrak{B}_2) \end{array}\right\} \textit{sonst} \end{array} \right.$$

Ist $\text{SR} = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$ *ein Definitionstripel und ist* $\mathfrak{B}$ *partitioniert in* $\mathfrak{B}_1 \cup \mathfrak{B}_2$ *so schreiben wir auch*

$$\text{SR} = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B}_1, \mathfrak{B}_2)$$

Ist NRep eine durch $\text{SR} = (\mathfrak{A}, \mathfrak{C}, \mathfrak{B}_1, \mathfrak{B}_2)$ *induzierte nichtdeterministische Ersetzungsregel, so schreiben wir*

$$NRep : \mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}_1 \bigvee \mathfrak{B}_2.$$

Analog schreiben wir für eine nichtdeterministische Augmentationsregel

$$NAug : \mathfrak{A} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}_1 \bigvee \mathfrak{B}_2.$$

$\mathbf{R_8}$. $\mathfrak{O}_F$ ist die Menge aller in F vorkommenden Objektnamen. Beachte, daß das Guard-Atom $?y \notin \mathfrak{O}_F$ nicht so harmlos ist wie es aussieht. Es garantiert die Zulässigkeit der Regel. Im Prinzip muß die Grundsubstitution σ hier für $?y$ einen neuen Objektnamen suchen. Später verwendet wir an dieser Stelle ein selbstdefiniertes Prädikat, das eine Variablen an eine neues Objekt bindet (vgl. K_8 auf Seite 35).

$\mathbf{R_9}$ ist eine direkte Übersetzung der Werterestriktionsregel.

$\mathbf{R_{10}}$. $\perp$ ist ein Konstantensymbol für Konzeptnamen. $\perp$ kennzeichnet einen Clash.

[7] $\mathfrak{A} = \{A_1, \ldots, A_i\}, \mathfrak{B} = \{B_1, \ldots, B_k\}, \mathfrak{C} = \{C_1, \ldots, C_j\}$.

Damit läßt sich der Konsistenztest wie folgt beschreiben:

Algorithmus 3.14
Sei $\mathfrak{S} := \{R_1, \ldots, R_{10}\}$, α *eine Auswahlstrategie.* F *sei die zu überprüfende A-Box,* $\mathfrak{F}$ *stehe für eine Menge von A-Boxen.*

$i := 1$; $\mathfrak{F}_i := \{F\}$
while $\alpha(\mathfrak{F}_i) \neq \uparrow$ **do** /* *d.h. bestimme minimales* m, *so daß* */
  $\mathfrak{F}_{i+1} := \mathfrak{S}(\mathfrak{F}_i)$ /* $\mathfrak{S}^m(\mathfrak{F})$ *stabil ist* */
  $i := i + 1$
od
if $\bot \in \bigcap \mathfrak{F}_{i-1}$
  then return *„inkonsistent“*
  else return *„konsistent“*

wobei wir folgende Schreibweisen benutzen:

- $\alpha(\mathfrak{F}_i) \neq \uparrow$ *ist kurz für* $\exists F \in \mathfrak{F}_i : \alpha(F) \neq \uparrow$
- $\mathfrak{S}(\mathfrak{F}_i)$ *ist definiert als* $\{F \mid \alpha(F) = \uparrow$ *oder* $\exists F' \in \mathfrak{F}_i$ *so daß* $F' \longrightarrow_\alpha F\}$[8].

In Worten bedeutet das, wir vervollständigen die A-Box F *und überprüfen die Vervollständigung auf offensichtliche Widersprüche. Beachte, daß die Termination von Algorithmus 3.14 durch Satz 3.8 gesichert ist.*

Bemerkung: In diesem Algorithmus verarbeiten wir die Nichtdeterministische Regel durch den Übergang einer A-Box zu A-Box Mengen (vergleiche auch Algorithmus 3.11). Das ist eine praktische Darstellungsform. Im Implementierungsteil werden wir Nichtdeterminismen durch chronologisches Backtracking verarbeiten.

Angenommen wir hätten eine „Maschine“, die Simplification Rules der Form $\mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}$ bzw. $\mathfrak{A} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}$ und nichtdeterministische Simplification Rules der Form $\mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}_1 \bigvee \mathfrak{B}_2$ bzw. $\mathfrak{A} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}_1 \bigvee \mathfrak{B}_2$ auf eine Faktenmenge F anwenden kann. Dann könnten wir den Konsistenztest für A-Boxen mit Hilfe von Simplification Rules implementieren. Man beachte die Einfachheit der Implementierung! Die einzige Arbeit besteht in der (sehr einfachen) Übertragung der Rewrite-, Transformations- und Clash-Regeln. Mehr noch: Wir wissen, daß sich der Konsistenztest für A-Boxen auf Erweiterungen von $\mathcal{ALC}$ anwenden läßt allein durch Erweiterungen der Transformations- und Clash-Regeln, die sich wie gesehen in Simplification Rules umwandeln lassen. Dabei gibt es allerdings Fälle, in denen man die Anwendungsreihenfolge festlegen muß, um Vollständigkeit zu erhalten. Man hätte damit also eine Experimentierumgebung für Sprachvarianten (gegeben durch die Transformations- und Clashregeln) und Algorithmen (gegeben durch die Auswahlstrategie), die dem Experimentator erlaubt nur mit Regeln auf einer sehr hohen Ebene zu arbeiten. Es ist deshalb die Motivation dieser Arbeit

[8] Beachte, daß „$\longrightarrow_\alpha$“ nichtdeterministisch sein kann, wenn die ausgewählte Regel ein „$\bigvee$“ enthält.

folgendes Problem zu lösen:

Aufgabe: Definiere (und implementiere) einen Abarbeitungsmechanismus für Simplification Rules mit folgenden Merkmalen:

- Gegeben eine Menge von Fakten und eine Menge von Regeln (bestehend aus Simplification Rules und nichtdeterministischen Simplification Rules). Wende die Regeln nach bestimmten Strategien auf die Fakten an.
- Es soll möglich sein, den Ablauf der Regelanwendungen zu steuern.

Damit ist es dann Aufgabe des „Knowledge Engineer" geeignete Simplification Rules und Abarbeitungsstrategien für seinen jeweiligen Bedarf zu definieren.

Wir definieren in Kapitel 4 diesen Abarbeitungsmechanismus und geben in Kapitel 5 Regelsätze für verschiedene Wissensinferenzformalismen an.

Kapitel 4

Ein Abarbeitungsmechanismus für Simplification Rules

Wir entwickeln in diesem Kapitel einen Abarbeitungsmechanismus (AM) für Simplification Rules. Ausgehend von einem sehr einfachen Algorithmus-Skelett werden wir den endgültigen AM durch sukzessives Erweitern gewinnen. Der AM wird so ausgerichtet sein, daß sich der Konsistenztest terminologischer Sprachen möglichst einfach implementieren läßt. Dabei achten wir darauf, daß verschiedene terminologische Sprachen zugrunde gelegt werden können. Andererseits werden wir den AM soweit wie möglich modularisieren, so daß sich die verschiedenen Teilbereiche einzeln optimieren bzw. an andere Aufgabestellungen anpassen lassen.

Algorithmus 4.1 (Rohentwurf) *Sei α eine Auswahlstrategie über einer Trägermenge $\mathcal{F}$.*

Input	$\mathfrak{S}$ Menge von deterministischen Regeln über $\mathcal{F}$.
	F Menge von Fakten, $F \subset \mathcal{F}$
Output	$\mathfrak{S}^m(F)$ für ein $m \in \mathfrak{N}$

<u>while</u> $\alpha(F) \neq \uparrow$ <u>do</u>
    $F := \mathfrak{S}(F)$
<u>od</u>
<u>output</u> F

In diesem Algorithmus wenden wir solange Regeln aus $\mathfrak{S}$ auf F an, bis F stabil bzgl. $\mathfrak{S}$ ist. Das ist ein sinnvoller Ansatz, denn wir wissen (siehe z.B. [2] oder [13]) aus der Theorie der Konzeptsprachen, daß Stabilität bzgl. einer Regelmenge das Abbruchkriterium des Konsistenztests ist. Wie wir in Paragraph 3.1.3 gesehen haben, reichen deterministische Regeln für den Konsistenztest nicht aus.

Behandlung von Nichtdeterminismen

Im Gegensatz zu Paragraph 3.2, wo wir Nichtdeterminismen durch den Übergang von A-Boxen zu A-Box-Mengen realisiert haben, wollen wir diese jetzt mittels

chronologischem Backtracking abarbeiten. Wir benötigen dazu Mechanismen zur Erzeugung und Verarbeitung von Wahlpunkten.

Idee: Formuliere diese Mechanismen durch Regeln, die eine SiR-Struktur haben.

Definition 4.2 (Anweisungsregel)
Sei $\mathcal{A}$ eine nichtleere Trägermenge, sei $\mathcal{PROC}$ eine Menge definierter Prozeduren. Eine **Anweisungsregel** *ist eine Abbildung*

$$\begin{array}{lccc} \textit{Eval:} & \mathcal{A} & \longrightarrow & \mathcal{PROC} \\ & A & \mapsto & P \end{array}$$

Schreibe: *Eval:* $A \longleftrightarrow P$ *oder Eval:* $A \longrightarrow P$. *Die Elemente von $\mathcal{A}$ heißen* **Anweisungen**.

Eine Anweisungsregel anzuwenden bedeutet:

$\left.\begin{array}{l} \mathcal{A} := \mathcal{A} \setminus A \\ \text{Führe P aus!} \end{array}\right\}$ falls Eval: A $\longleftrightarrow$ P

$\left.\begin{array}{l} \text{Führe P aus!} \end{array}\right\}$ falls Eval: A $\longrightarrow$ P

Mit anderen Worten: A startet mittels Eval eine Prozedur P. Dabei wird gegebenenfalls die Anweisung aus $\mathcal{A}$ entfernt. Informell: Anweisungsregeln sind Simplification Rules ohne Guard-Atome, bei deren Anwendung das (die) Body-Atom(e) ausgewertet wird (werden).

Beispiel 4.3
Die nichtdeterministische Simplification Rule des Konsistenztests für A-Boxen über $\mathcal{ALC}$ läßt sich folgendermaßen eliminieren:

ersetzte	durch
NRep: $\mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \mathfrak{B}_1 \bigvee \mathfrak{B}_2$	Rep_1: $\mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \text{nondet}(\mathfrak{B}_1, \mathfrak{B}_2)$
	$Eval_1$: $\text{nondet}(\mathfrak{B}_1, \mathfrak{B}_2) \longleftrightarrow \text{make-choicepoint}(\mathfrak{B}_1, \mathfrak{B}_2)$
ergänze	um
Rep_2: $\mathfrak{A} \Longleftrightarrow \mathfrak{C} \mid \bot$ um	$Eval_2$: $\bot \longleftrightarrow$ clash

dabei seien make-choicepoint *und* clash *durch folgende Pseudo-Code Prozeduren definiert:*

procedure *make-choicepoint ($\mathcal{X}$,$\mathcal{Y} \in \mathcal{F}$);*
„erzeuge einen Wahlpunkt '$\mathcal{X}$ oder $\mathcal{Y}$'"
end *make-choicepoint.*

procedure *clash;*
  if *„es gibt einen Wahlpunkt“*
    then *„zurück zum nächsten Wahlpunkt“*
    else *STOP*
end *clash.*

Bemerkung: Wir betrachten Rep_1 und Rep_2 als SiR über $\mathcal{F} \cup \mathcal{A}$. Der wesentliche Punkt in diesem Beispiel ist Folgendes: Wir ersetzen die nichtdeterministische *SiR NRep durch die* deterministische *SiR Rep_1 und die Anweisungsregel $Eval_1$.*

Informell könnte man sagen: Anweisungsregeln sind ein Mechanismus, der der Steuerung des AM einen deklarativen Anstrich verleiht. Auf diese Weise hält man die „low-level“-Programmierung aus dem eigentlichen Abarbeitungsmechanismus heraus. Insbesondere ist man bei der Organisation der Wahlpunkte (auch im nachhinein) sehr flexibel.

Sei $\mathcal{F}$ eine Trägermenge von Fakten, sei $\mathcal{A}$ eine Trägermenge von Anweisungen, $\mathcal{A} \cap \mathcal{F} = \emptyset$, ferner sei $\mathcal{PROC}$ eine Menge von Prozeduren. Ist $\mathfrak{F} := \mathcal{F} \cup \mathcal{A} \cup \mathcal{PROC}$, so kann man Erweiterungs-, Augmentations- und Anweisungsregeln als Regeln über $\mathfrak{F}$ auffassen[1]. Sei $\mathfrak{S}$ eine Menge von Regeln über $\mathfrak{F}$. Eine Auswahlstrategie α über $\mathfrak{F}$ wählt jetzt entweder eine SiR oder eine Anweisungsregel aus, die auf $\mathfrak{F}$ anwendbar ist. Dabei heißt eine Anweisungsregel $A \overset{\longleftarrow}{\longrightarrow} P$ **anwendbar**, wenn $A \in \mathcal{A}$, d.h. $A \in \mathfrak{F}$ ist.

Algorithmus 4.4 *Sei α eine Auswahlstrategie über $\mathfrak{F}$.*

Input $\mathfrak{S}$ *Menge von Regeln über* $\mathfrak{F}$
  F *Menge von Fakten,* $F \subset \mathcal{F}$
Output $\mathfrak{S}^m(F)$

while $\alpha(F) \neq \uparrow$ **do**
  $S := \pi_1(\alpha(F))$
  if *„S ist eine Anweisungsregel“*
    then procedure-call *(body(S))*
    else $F := S(F)$
od
output F

Dabei sei body(Regel) eine Prozedur, die die Body-Atome der Regel zurückgibt.

Beachte, daß der Prozeduraufruf (d.h. insbesondere die Prozedur) keinerlei Einschränkungen unterliegt. In einer solchen Prozedur kann nahezu alles passieren (F kann manipuliert werden, das Programm kann abgebrochen werden, usw.).

Mit der Einführung der Prozeduraufrufe in den Body-Atomen haben wir den Mechanismus der SiR stark erweitert. Der Preis den man dafür zu zahlen hat, ist der Verlust der klaren deklarativen Semantik.

[1] Wir sprechen dann von SiR über $\mathfrak{F}$. Wenn wir zwischen Anweisungsregeln und Ersetzungs- bzw. Augmentationsregeln unterscheiden wollen, nennen wir letztere „klassische SiR“.

Die Auswahlstrategie

Bisher haben wir uns um die Auswahlstrategie (AS) praktisch nicht gekümmert. Tatsache ist aber, daß die AS einen wesentlichen Beitrag zum Abarbeitungsmechanismus leistet. Betrachtet man das Vorausgegangene, so stellt man fest, daß die AS sowohl für die Termination, als auch für die Abarbeitungsreihenfolge der Regelanwendungen verantwortlich ist. Es ist klar, daß die AS eng mit dem Anwendungsgebiet zusammenhängt, das man mit den SiR strukturieren will. Wir werden im Folgenden eine Auswahlstrategie erläutern, die für unser Ziel besonders geeignet ist. Erinnern wir uns: Eine AS muß aus einer gegebenen Regel- bzw. Faktenmenge eine Regel und Fakten so auswählen, daß sich die Regel auf die Fakten anwenden läßt. Dabei soll gleichzeitig verhindert werden, daß eine SiR (insbesondere eine Augmentationsregel) mehrfach auf dieselben Fakten angewendet wird. Weiterhin sind eventuell Prioritäten bezüglich der Anwendungsreihenfolge zu berücksichtigen. In unserer Anwendung ist es so, daß bestimmte Regeln vor anderen angewendet werden sollen, die Reihenfolge der beteiligten Fakten aber unwesentlich ist.

Beispiel 4.5
Seien $S_1 : x \leq y, y \leq z \Longrightarrow \mid x \leq z$ *und* $S_2 : x \leq y, x < y \Longleftrightarrow \mid x < y$ *Simplification Rules.*
Sei $F : \{\underbrace{a \leq b}_{f_1}, \underbrace{b \leq c}_{f_2}, \underbrace{b \leq d}_{f_3}, \underbrace{b \leq e}_{f_4}, \underbrace{a < b}_{f_5}\}$ *eine Faktenmenge.*
Dann liefert die Anwendung von zuerst S_1 *auf* $\{f_1, f_2\}, \{f_1, f_3\}, \{f_1, f_4\}$ *dann* S_2 *auf* $\{f_1, f_5\}$ $\mathfrak{F} : \{f_2, f_3, f_4, a \leq c, a \leq d, a \leq e, a < b\}$ *und zwar unabhängig davon in welcher Reihenfolge* S_1 *auf* $\{f_1, f_2\}, \{f_1, f_3\}, \{f_1, f_4\}$ *angewendet wird. Wendet man zuerst* S_2 *auf* $\{f_1, f_5\}$ *an, so ist* $\mathfrak{F}^* : \{f_2, f_3, f_4, a < b\}$ *und* S_1 *ist nicht mehr anwendbar. Beachte daß* $\mathfrak{F} \neq \mathfrak{F}^*$.

Wir werden künftig die Regel maximaler Priorität auswählen und diese solange wie möglich auf F anwenden. Dazu definieren wir eine Bewertungsfunktion, die jeder SiR eine Zahl zuordnet.

Teilinstantiierung von Regeln

Jetzt entwickeln wir eine Methode mit der sich die Anwendung einer beliebigen SiR auf die Anwendung einer SiR mit nur einem Head-Atom zurückführen läßt[2]. Wir betrachten folgendes Beispiel:

Beispiel 4.6

$$\mathcal{S}: \underbrace{x \leq y}_{A_1}, \underbrace{y \leq z}_{A_2} \Longrightarrow \mid x \leq y \qquad F : \{\underbrace{a \leq b}_{f_1}, \underbrace{b \leq c}_{f_2}, \underbrace{b \leq d}_{f_3}\}$$

statt $\mathcal{S}$ *auf* $\{f_1, f_2\}$ *bzw.* $\{f_1, f_3\}$ *anzuwenden bilden wir zuerst aus* $\mathcal{S}$ *durch „partielle Instantiierung" mit* f_1 *eine neue Regel* $\mathcal{S}' : b \leq z \Longrightarrow \mid a \leq z$. *D.h. das erste*

[2]Da Anweisungsregeln per definitionem nur ein Head-Atom haben, beschränken wir uns im Folgenden auf klassische SiR.

Head-Atom wird weglassen und die Substitution, die $x \leq y$ mit $a \leq b$ matcht, wird angewendet. Die Anwendung von $\mathcal{S}'$ auf f_2 danach auf f_3 führt zum gleichen Ergebnis wie vorher. Insbesondere müssen wir auf diese Weise $x \leq y$ nur einmal *mit $a \leq b$ matchen.*

Definition 4.7 (Teilinstantiierung)
Sei $\mathcal{S}$ eine durch $(\{A_1, \ldots, A_i\}, \mathfrak{C}, \mathfrak{B})$ induzierte Simplification Rule. Sei f ein Fakt, der mit A_1 matchbar ist, sei σ der Matcher von A_1 mit f[3].

Ist $\mathcal{S}'$ eine durch $(\{\sigma(A_2), \ldots, \sigma(A_i)\}, \sigma(\mathfrak{C}), \sigma(\mathfrak{B}))$ induzierte SiR und ist $\mathcal{S}'$ vom gleichen Typ[4] *wie $\mathcal{S}$, so heißt $\mathcal{S}'$* **die mittels f aus $\mathcal{S}$ teilinstantiierte Regel**. *Das Bilden von $\mathcal{S}'$ aus $\mathcal{S}$ mittels eines Faktes, der das* erste *Head-Atom matcht, nennen wir* **teilinstantiieren** *von $\mathcal{S}$.*

Beispiel 4.8
Sei $\mathcal{S}$: $\underbrace{x \leq y}_{A_1}, \underbrace{y \leq z}_{A_2} \Longrightarrow \mid x \leq z$ eine Regel und $f : a \leq b$ ein Fakt.
Dann ist $\sigma = (x \leftarrow a, y \leftarrow b)$ der Matcher von A_1 mit f, und $\mathcal{S}'$: $b \leq z \Longrightarrow \mid a \leq z$ ist die mittels f aus $\mathcal{S}$ teilinstantiierte Regel.

Lemma 4.9
Sei $\mathcal{S}$ eine durch $(\mathfrak{A}, \mathfrak{C}, \mathfrak{B})$ induzierte SiR mit $\mathfrak{A} = (A_1, \ldots, A_i)$, $\mathfrak{C} = \emptyset$.
Ist $\mathcal{S}$ (auf $f_1, \ldots, f_i$) anwendbar, dann gibt es eine Folge von Regeln $\mathcal{S} = \mathcal{S}_0 \longrightarrow \mathcal{S}_1 \longrightarrow \cdots \longrightarrow \mathcal{S}_{i-1}$ für die gilt:

- *$\mathcal{S}_m$ ist die mittels f_m aus $\mathcal{S}_{m-1}$ teilinstantiierte Regel $(1 \leq m \leq i-1)$.*
- *$\mathcal{S}_{i-1}$ ist anwendbar*

Beweis: Beachte: $\mathcal{S}_{i-1}$ ist durch $(\{\mu(A_i)\}, \mu(\mathfrak{C}), \mu(\mathfrak{B}))$ induziert, $\mu := \mu_{i-1} \circ \cdots \circ \mu_1$ wobei μ_m der Matcher von $\mathcal{S}_{m-1} \longrightarrow \mathcal{S}_m$ ist. Insbesondere hat $\mathcal{S}_{i-1}$ nur ein Head-Atom.
Sei $\mathcal{S}$ anwendbar, d.h. es gibt $f_1, \ldots, f_i \in F$ und σ, so daß $\sigma(A_1) = f_1, \ldots, \sigma(A_i) = f_i$ (beachte $\mathfrak{C} = \emptyset$).
Ist dann

$\mathcal{S}_1$ *induziert durch* $(\{\sigma(A_2), \cdots, \sigma(A_i)\}, \sigma(\mathfrak{C}), \sigma(\mathfrak{B}))$
$\mathcal{S}_2$ *induziert durch* $(\{\sigma(A_3), \cdots, \sigma(A_i)\}, \sigma(\mathfrak{C}), \sigma(\mathfrak{B}))$
$\vdots$ $\vdots$
$\mathcal{S}_{i-1}$*induziert durch* $(\{\sigma(A_i)\}, \sigma(\mathfrak{C}), \sigma(\mathfrak{B}))$,

so ist $\mathcal{S} \longrightarrow \mathcal{S}_1 \longrightarrow \cdots \longrightarrow \mathcal{S}_{i-1}$ die gesuchte Regelfolge. Ferner ist $\mathcal{S}_{i-1}$ voraussetzungsgemäß auf f_i anwendbar $(\sigma(A_i) = f_i)$. ◇

[3] Erinnerung: d.h. $\sigma(A_1) = f$
[4] Ersetzungs- bzw. Augmentationsregel

Bemerkung: Durch die Verwendung von teilinstantiierten Regeln lassen sich SiR einfacher abarbeiten, da nur noch Regeln mit *einem* Head-Atom angewendet werden. Da wir auch Ersetzungsregeln verwenden ist zum Zeitpunkt der Regelanwendung einer teilinstantiierten Regel nicht gewährleistet, daß die Fakten, die die Regeln teilinstantiiert haben, noch existieren (mit anderen Worten, die ursprüngliche Regel wäre gar nicht anwendbar). Wir müssen deshalb die Anwendbarkeitbedingung für teilinstantiierte Regeln erweitern: Existieren die Fakten, die die Regel teilinstantiiert haben noch? Ferner müssen bei der Anwendung einer teilinstantiierten Ersetzungsregel auch die Fakten ersetzt werden, die die Regel teilinstantiiert haben.

Damit sind wir in der Lage den Algorithmus weiter zu verfeinern:

Algorithmus 4.10
Sei α eine Auswahlstrategie.
Input $\mathfrak{S}$ Menge von Regeln
F Menge von Fakten
Output $\mathfrak{S}^m(F)$

```
while α(F) ≠ ↑ do
    S := π1(α(F))
    while mehr-Kopf-Regel(S) do
        S:= teilinstantiiere(S)
        𝔖:= 𝔖 ∪ {S}
    od
    anwenden(S)
od
```

Bemerkungen:

- Die Bedeutung der verwendeten Prozeduren ist intuitiv klar.
- Es werden nur Regeln angewendet, die *ein* Head-Atom haben. Wegen Lemma 4.9 ist das hinreichend.
- Bei Ersetzungsregeln muß man sicherstellen, daß auch die Fakten ersetzt werden, die die Regel teilinstantiiert haben.
- Die Auswahlstrategie sorgt dafür, daß eine Regel nicht mehrmals mit demselben Fakt teilinstantiiert wird. Insbesondere werden nur solche (teilinstantiierte) Regeln ausgewählt, deren teilinstantiierenden Fakten noch existieren. Natürlich gewährleistet die AS weiterhin, daß eine Regel nicht mehrmals auf dieselben Fakten angewendet wird.
- Da nur Ein-Kopf-Regeln angewendet werden und man zum Teilinstantiieren nur einen Fakt benötigt (nämlich einen, der das erste Head-Atom matcht), muß man eine Regel jeweils nur mit *einem* Fakt betrachten.

Auswahl der Regeln

Als nächstes schränken wir die Regeln ein, die auf Anwendbarkeit zu überprüfen sind. Dazu betrachten wir Folgendes: Sei $\mathcal{S}$ eine durch $(\{A_1, \ldots, A_i\}, \mathfrak{C}, \mathfrak{B})$ induzierte SiR. Angenommen wir haben festgestellt, daß kein $f \in F$ mit A_1 matchbar ist. Dann kann $\mathcal{S}$ nicht auf F anwendbar sein und wir müssen die Anwendbarkeit von $\mathcal{S}$ solange nicht mehr überprüfen, bis neue Fakten zu F hinzukommen. Anders gesagt: Ist $\mathcal{S}$ nicht auf F anwendbar und ist f' nicht mit A_1 matchbar, dann ist $\mathcal{S}$ auch nicht auf $F \cup \{f'\}$ anwendbar. Idee: Wir (d.h. die Auswahlstrategie) merken uns die Regeln, die als nicht anwendbar erkannt wurden, als „deaktiv" und überprüfen diese nicht noch einmal. Entsteht ein neuer Fakt, so matchen wir diesen gegen alle „deaktiven" Regeln (d.h. jeweils gegen ihr erstes Head-Atom). Finden wir einen Matcher, so „aktivieren" wir die entsprechende Regel. Beachte, daß aktivieren nicht anwenden bedeutet. Eine Regel zu aktivieren bedeutet, sie als *potentiell* anwendbar zu kennzeichnen. Über ihre Anwendung entscheidet die Auswahlstrategie. Mit anderen Worten: Durch Deaktivierung von Regeln wird der Suchraum der Auswahlstrategie nach einer geeigneten Regel eingeschränkt. D.h. sind alle Regeln deaktiv, so ist keine Regel mehr anwendbar.

Auswahl der Fakten

Wir überlegen uns jetzt was es heißt, daß ein Fakt zu einer Regel „paßt". Als erstes muß er natürlich ein Head-Atom matchen.[5] Beachte, daß wir weder Regeln mehrfach auf dieselben Fakten anwenden noch Regeln mehrfach mit denselben Fakten teilinstantiieren wollen. Ein „passender" Fakt darf also mit einer Regel noch nichts zu tun gehabt haben. Zu einer Regel $\mathcal{S}$ liefere **bekannt**$(\mathcal{S})$[6] eine Menge $\mathfrak{F}$ von Fakten für die gilt:

- $f \in \mathfrak{F}$, falls $\mathcal{S}$ mittels f aus einer Regel teilinstantiiert wurde.
- $f \in \mathfrak{F}$, falls $\mathcal{S}$ schon auf f angewendet wurde.
- $f \in \mathfrak{F}$, falls $\mathcal{S}$ nicht auf f angewendet werden konnte (das aber bereits probiert wurde).

Ein „passender Fakt" zu einer SiR $\mathcal{S}$ ist dann ein $f \in F \setminus \mathfrak{F}$, der mit A_1[7] matchbar ist.

Ergebnis

Zur Beschreibung der letzten Verfeinerung, die wir in Algorithmus 4.12 beschreiben, definieren wir folgende Ausdrucksweisen:

[5] Wie wir uns bereits überlegt haben, genügt es das erste Head-Atom zu betrachten.

[6] Eine Möglich diese Funktion zu realisieren wird in Paragraph 5.3 ab Seite 39 beschrieben.

[7] $\mathcal{S}$ sei durch $(\{A_1, \ldots, A_i\}, \mathfrak{C}, \mathfrak{B})$ induziert.

Definition 4.11
Jeder SiR sei ein „Modus-Slot" zugeordnet. Eine SiR ist entweder aktiv *oder* deaktiv. *Dabei soll eine SiR genau dann deaktiv sein, wenn wir „wissen", daß die Regel nicht anwendbar ist. Der Modus läßt sich mit den Funktionen* aktiviere*(S) bzw.* deaktiviere*(S) ändern. Weiterhin sei* passender-Fakt*(S) eine Funktion, die einer Regel S entweder einen passenden Fakt (gem. obigen Ausführungen) zuordnet oder „↑" zurückgibt, falls kein solcher existiert.*

Algorithmus 4.12
Input $\mathfrak{S}$ Menge von Regeln
F Menge von Fakten
Output $\mathfrak{S}^m(F)$

```
while „es gibt aktive SiR in 𝔖" do
      S:= wähle-aktive-Regel()
♣     f:= passender-Fakt(S)
      if f ≠ ↑
        then if ein-Kopf-Regel?(S)
                then anwenden(S)
                else S:= teilinstantiiere(S)
                     aktiviere(S)
                     𝔖:= 𝔖 ∪{S}
                     goto ♣
             fi
        else deaktiviere(S)
      fi
od
output F
```

Dabei hat die Prozedur **anwenden** folgendes zu beachten:

- Durch das Entstehen neuer Fakten ist es möglich, daß deaktive SiR anwendbar werden. Solche SiR sind gegebenenfalls zu aktivieren. Dabei gehen wir davon aus, daß der Wahrheitswert eines Guard-Atoms höchstens von den Fakten abhängt. Das ist keine wesentliche Einschränkung, da wir im Weiteren davon ausgehen, daß die Wahrheitswerte der Guard-Atome laufzeit-unabhängig sind. In den Beispielen in Kapitel 5 beziehen sich die Guard-Atome auf Eigenschaften der Fakten, die den Regel-Kopf matchen.

- Durch das Anwenden einer Ersetzungsregel werden Fakten gelöscht. Deshalb muß beim Anwenden einer teilinstantiierten Regel überprüft werden, ob sie noch gültig ist, d.h. ob die Fakten, die die Regel teilinstantiiert haben, noch existieren.

- Wird eine teilinstantiierte Ersetzungsregel angewendet, so müssen auch die Fakten ersetzt werden, die die Regel teilinstantiiert haben.

- Ist eine Regel auf Grund ihrer Guard-Atome nicht anwendbar, so wird sie deaktiviert. Dabei kommt es darauf an, ob sich der Wahrheitswert der Guard-Atome zur Laufzeit ändern kann (z.B. durch Hinzukommen neuer Fakten). Ansonsten müssen solche Regeln nicht mehr aktiviert werden. Beachte, daß nur Guard-Atome von vollständig instantiierten Regeln überprüft werden.

Fassen wir die Merkmale des Abarbeitungsmechanismus noch einmal zusammen: Gegeben sei eine Menge $\mathfrak{S}$ von Anweisungs-, Ersetzungs- und Augmentationsregeln und eine Menge F von Fakten. Der AM wendet die Regeln solange auf die Fakten an, bis F stabil bezüglich $\mathfrak{S}$ ist. Dabei dienen die Anweisungsregeln insbesondere zur Verarbeitung von Nichtdeterminismen. Die Regeln werden sukzessive teilinstantiiert, bis Regeln entstehen, die nur ein Head-Atom haben. Dadurch muß man eine Regel nur *einmal* mit einem Fakt betrachten. Matcht ein Fakt das erste Head-Atom, so wird die Regel auf den Fakt angewendet (wenn die Regel genau ein Head-Atom hat) oder mit dem Fakt teilinstantiiert. Um eine Regel anwenden zu können, ist es neben der Erfüllung der Guard-Atome jetzt auch nötig, die teilinstantiierenden Fakten zu verifizieren[8]. Die Anzahl der Fakten, die für eine Regelanwendung in Frage kommt, wird also — abgesehen vom Entstehen neuer Fakten — immer kleiner. Es ist die Aufgabe des Regel-Formulierers dafür zu sorgen, daß durch die Regelanwendungen keine Endlosschleifen entstehen können. Den Regeln wird ein Modus zugeordnet. Regeln sind entweder aktiv oder deaktiv. Dabei ist eine Regel genau dann deaktiv, wenn kein Fakt mit dem ersten Head-Atom matchbar ist oder wenn ein Guard-Atom falsifiziert wurde. Wir gehen davon aus, das der Wahrheitswert der Guard-Atome laufzeit-unabhängig ist. Damit ist F genau dann stabil bezüglich $\mathfrak{S}$, wenn alle Regeln deaktiv sind. Eine Regel anzuwenden bedeutet jetzt

(a) eine Prozedur aufrufen (Anweisungsregel).

(b) die Faktenbasis um neue Fakten erweitern (Augmentationsregel).

(c) die Faktenbasis gemäß den Head- und Body-Atomen modifizieren (Ersetzungsregel). Dabei ist darauf zu achten, daß auch die Fakten ersetzt werden, die die Regel teilinstantiiert haben.

In Anhang A befindet sich eine ausführlich kommentierte Implementation in COMMON LISP.

[8] d.h. auf Existenz zu überprufen

Kapitel 5

Regelsätze

Wir geben in diesem Kapitel eine Reihe von Regelsätzen für verschiedene Wissensrepräsentationsformalismen an. Dazu stellen wir einige allgemeine Überlegungen voran.

- Da die Regeln sukzessive teilinstantiiert werden (vgl. Kapitel 4), ist es sinnvoll die Head-Atome an den Anfang zu stellen, die schwer zu erfüllen sind. Dadurch vermeidet man Teilinstantiierungen, die zu keiner Regelanwendung führen.
- Weiterhin ist es sinnvoll diejenigen Ersetzungsregeln bevorzugt anzuwenden, die möglichst spezielles Wissen propagieren. Dadurch erspart man sich Ableitungen, die sich im nachhinein als obsolet herausstellen.
- Clash-Regeln sollten eine hohe Priorität erhalten, um Widersprüche möglichst früh zu erkennen und unnötige Ableitungen zu vermeiden.
- Nichtdeterminismen sollten solange wie möglich verzögert werden, um den Backtrackingaufwand so klein wie möglich zu halten.

Dabei finden die letzten drei Heuristiken nur dann Anwendung, wenn die Reihenfolge der Regelanwendung *semantisch* gleichgültig ist. Zur vereinfachten Darstellung von SiR vereinbaren wir:

- Prozeduraufrufe stehen direkt in den Body-Atomen. (D.h. wir verzichten auf Anweisungsregeln.) Prozeduraufrufe werden wir besonders kennzeichnen.
- In den Guard-Atomen verwenden wir (selbstdefinierte) Prädikate, deren Bedeutung sich (meistens) aus ihrem „suggestiven“ Namen ergibt.
- Wir weisen jeder SiR eine Priorität zu. Diese notieren wir nach den Body-Atomen getrennt durch „;“.
- Variable beginnen mit „?“.

Die Implementation aus Anhang A kommt mit dieser Darstellung zurecht.

5.1 Der Konsistenztest für $\mathcal{ALC}$

Wir notieren noch einmal die Regeln, die wir teilweise schon in Paragraph 3.2 kennengelernt haben (damit halten wir dieses Kapitel abgeschlossen und gewöhnen uns an die obengenannten Darstellungsweise).

K_1 :	$?x : \neg(?s \sqcap ?t)$	$\Longleftrightarrow$		$\mid ?x : (\neg ?s) \sqcup (\neg ?t)$	;	5
K_2 :	$?x : \neg(?s \sqcup ?t)$	$\Longleftrightarrow$		$\mid ?x : (\neg ?s) \sqcap (\neg ?t)$	;	5
K_3 :	$?x : \neg(\neg ?s)$	$\Longleftrightarrow$		$\mid ?x : ?s$	;	5
K_4 :	$?x : \neg(\exists ?r.?s)$	$\Longleftrightarrow$		$\mid ?x : (\forall ?r.\neg ?s)$	;	5
K_5 :	$?x : \neg(\forall ?r.?s)$	$\Longleftrightarrow$		$\mid ?x : (\exists ?r.\neg ?s)$	;	5
K_6 :	$?x : (?s \sqcap ?t)$	$\Longleftrightarrow$		$\mid ?x : ?s, ?x : ?t$	;	0
K_7 :	$?x : (?s \sqcup ?t)$	$\Longleftrightarrow$		$\mid$ NONDET$(?x : ?s, ?x : ?t)$	;	-1
K_8 :	$?x : \exists ?r.?s$	$\Longleftrightarrow$	FRESH$(?y)$	$\mid (?x, ?y) : ?r, ?y : ?s$	;	0
K_9 :	$?x : \forall ?r.?s, (?x, ?y) : ?r$	$\Longrightarrow$		$\mid ?y : ?s$	;	0
K_{10} :	$?x : ?s, ?x : \neg ?s$	$\Longleftrightarrow$		$\mid$ CLASH	;	10
K_{11} :	$?x : ?c$	$\Longleftrightarrow$	$?c \doteq ?t$	$\mid ?x : ?t$	;	6
K_{12} :	$?x : \neg ?c$	$\Longleftrightarrow$	$?c \doteq ?t$	$\mid ?x : \neg ?t$	;	6

Bemerkungen:

- Die Regeln $K_1 - K_{10}$ kennen wir bereits von Seite 21.
- Die Regeln K_{11} und K_{12} expandieren Konzeptterme.
- Das Prädikat FRESH erzeugt einen neuen Objektnamen und evaluiert zu wahr. Beachte, daß durch Binden von $?y$ im Regel-Guard die Regel zulässig ist.
- Die Funktionen NONDET und CLASH sind wie in Beispiel 4.3 definiert.
- Durch die Prioritäten kommt folgendes zum Ausdruck
  - Die Faktenbasis wird *ständig* auf offensichtliche Widersprüche untersucht (K_{10}).
  - Expandierung von Konzepttermen (K_{11}, K_{12}) hat Vorrang vor Überführung in NNF ($K_1 - K_5$), das wiederum hat Vorrang vor Vervollständigung der A-Box ($K_6 - K_9$).
  - Das Erzeugen von Wahlpunkten wird solange wie möglich verzögert (K_7).

Wir haben damit nicht nur den Konsistenztest mit zwölf handlichen Regeln definiert, sondern auch eine Basis für Verbesserungen geschaffen. Stellen wir uns beispielsweise vor, die Subsumptionshierachie der zugrundegelegten T-Box wäre uns bekannt (z.B. aus früheren Berechnungen) und wir wollten dieses Wissen jetzt benutzen. Wir defineren dazu einfach zwei neue Regeln und haben so den Algorithmus *auf Regelebene* verbessert.

$$K_{13}: \quad ?x:?s, ?x:?t \iff \text{SUBSUMES}(?s,?t) \mid ?x:?t \quad ; 20$$
$$K_{14}: ?x:?s, ?x:\neg ?t \iff \text{SUBSUMES}(?s,?t) \mid \text{CLASH} \; ; 21$$

Beachte, daß wir durch die hohen Prioritäten dieser Regeln eine Art Vorverarbeitung („preprocessing") vornehmen.

5.2 Konkrete Bereiche

Wir führen die Implementation konkreter Bereiche durch Simplification Rules mittels eines Beispiels exemplarisch vor. Dazu verwenden wir die rationalen Zahlen $\mathcal{Q}$ zusammen mit den Relationen „$<$" und „$\leq$".

$$\begin{array}{lrcl}
D_1 & : ?x \leq ?y, ?y \leq ?z & \Longrightarrow & \mid ?x \leq ?z \; ; \; 0 \\
D_2 & : \quad ?x \leq ?x & \Longleftrightarrow & \mid \text{TRUE} \; ; 20 \\
D_3 & : ?x < ?y, ?y < ?z & \Longrightarrow & \mid ?x < ?z \; ; \; 0 \\
D_4 & : ?x < ?y, ?y < ?x & \Longleftrightarrow & \mid \text{CLASH} \; ; \; 6 \\
D_5 & : \quad ?x < ?x & \Longleftrightarrow & \mid \text{CLASH} \; ; 21 \\
D_6 & : \quad \neg(?x \leq ?y) & \Longleftrightarrow & \mid ?y < ?x \; ; 10 \\
D_7 & : \quad \neg(?x < ?y) & \Longleftrightarrow & \mid ?y \leq ?x \; ; 10 \\
D_8 & : ?x \leq ?y, ?y < ?z & \Longrightarrow & \mid ?x < ?z \; ; \; 0 \\
D_9 & : ?x < ?y, ?y \leq ?z & \Longrightarrow & \mid ?x < ?z \; ; \; 0 \\
D_{10} & : ?x < ?y, ?x \leq ?y & \Longleftrightarrow & \mid ?x < ?y \; ; 15 \\
D_{11} & : ?x < ?y, ?y \leq ?x & \Longleftrightarrow & \mid \text{CLASH} \; ; 16
\end{array}$$

Dabei bringen die Regeln folgendes zum Ausdruck:

$D_1 - D_5$. Transitivität, Antisymmetrie und Reflexivität (bzw. Irreflexivität) von $\leq$ (bzw. $<$)

D_7, D_8. Negation von $\leq$ bzw. $<$

$D_9 - D_{12}$. Subsumption von $<$ bzgl. $\leq$

Falls man eine Tautologie nicht besonders kennzeichnen möchte, kann man auf TRUE verzichten. D_3 hat dann kein Body-Atom, d.h. die Tautologie $?x \leq ?x$ wird einfach gelöscht (genauer: durch „nichts" ersetzt). Die Prioritäten sind nach den anfangs genannten Heuristiken gesetzt.

5.3 Gleichheit

Bei der Handhabung von Gleichheit durch Referenzketten (Zeigerverwaltung) (wie es z.B. in PROLOG üblich ist) benutzt man das in diesen Ketten implizit vorhandene Wissen (z.B. Transitivität) zur Herleitung neuer Fakten. Dabei ist später nicht mehr nachvollziehbar welche Referenzen (i.e. welche Zeiger) benutzt wurden. Es gibt jedoch Situationen in denen diese Information benötigt wird. (Beispielsweise beim Übergang von chronologischem zu intelligentem Backtracking.) In diesen Fällen muß man Gleichheiten getrennt vom übrigen Formalismus betrachten. Durch explizieren des Wissens über Gleichheit läßt sich diese in das normale Schema einbetten. Dazu bildet man zu einem Satz von Gleichungen die transitive Hülle und propagiert das so erhaltene Wissen in die übrigen Fakten. Eine Möglichkeit das zu verwirklichen ist die Folgende: Man bestimmt zu jeder Äquivalenzklasse[1] einen Repräsentanten. Das hat man so einzurichten, daß dieser eindeutig ist. Dann ersetzt man alle Objekte (in den Fakten) durch den Repräsentanten ihrer Äquivalenzklasse[2]. Mit SiR läßt sich das Wissen folgendermaßen explizieren:

$$
\begin{array}{lrcll}
E_1 & : ?x = ?y, ?x = ?z & \Longrightarrow & \mid ?y = ?z & ; 5 \\
E_2 & : ?x = ?y, ?z = ?x & \Longrightarrow & \mid ?y = ?z & ; 5 \\
E_3 & : ?y = ?x, ?z = ?x & \Longrightarrow & \mid ?y = ?z & ; 5 \\
E_4 & : \quad ?x = ?x & \Longleftrightarrow & \mid \text{TRUE} & ; 10 \\
E_5 & : \quad ?x = ?y & \Longleftrightarrow \; ?x \rhd ?y & \mid ?y = ?x & ; 9 \\
E_{6_1} & : \quad ?x : ?c & \Longleftrightarrow \; ?y = ?x & \mid ?y : ?c & ; 0 \\
E_{6_2} & : \quad (?x, ?y) : ?r & \Longleftrightarrow \; ?z = ?x & \mid (?z, ?y) : ?r & ; 0 \\
E_{6_3} & : \quad (?x, ?y) : ?r & \Longleftrightarrow \; ?z = ?y & \mid (?x, ?z) : ?r & ; 0
\end{array}
$$

Hier beschreiben $E_1 - E_3$ die Transitivität[3], E_4 die Reflexivität der Gleichheit. E_5 bestimmt den Repräsentant *einer* Gleichung, indem das kleinere[4] Objekt auf die linke Seite der Gleichung gesetzt wird. $E_{6_1} - E_{6_3}$ propagieren das mittels $E_1 - E_5$ hergeleitete Wissen. Beachte, daß die Gleichheitsabfragen in den Guard-Atomen erfolgen müssen, da die Head-Atome ersetzt werden (und wir die Gleichheiten nicht ersetzen wollen). Ferner wird unterstellt, daß nur Fakten der Form '*Objekt* : *Konzept*' oder '(*Objekt*$_1$, *Objekt*$_2$) : *Rollenname*' in der Faktenbasis auftreten. Gibt es noch andere Muster, so muß man für jedes eine SiR definieren.

Der entscheidende Vorteil dieser Methode besteht darin, daß die durch Gleichheit entstehenden Fakten genauso behandelt werden können wie alle anderen Fakten. Insbesondere muß beim Backtracking Gleichheit nicht gesondert behandelt werden. Das mittels Gleichheit definierte Wissen ordnet sich in das allgemeine Schema ein. Wie zu erwarten ist, bekommt man das nicht umsonst. Der Preis ist ein (teilweise erheblicher) Mehraufwand an redundanten Regelanwendungen. Beachte, daß durch die SiR $E_{6_1} - E_{6_3}$ *neue* Fakten entstehen, also SiR auf diese Fakten

[1] äquivalent modulo propagierter Gleichheit

[2] Gibt es zu einem Objekt keine Gleichung, so ist das Objekt selbst der Repräsentant.

[3] Durch das Richten der Gleichungen können diese drei Fälle auftreten.

[4] Bzgl. einer künstlichen Ordnung, die mit $\rhd$ bezeichnet wird.

auch dann angewendet werden können, wenn sie bereits auf die ursprünglichen Fakten angewendet (bzw. mit ihnen teilinstantiiert) wurden.

Beispiel 5.1

$$R : ?x : \mathsf{Dreieck} \Longrightarrow \mid ?x : \mathsf{geometrisch}$$

$$F : \{\underbrace{a : \mathsf{Dreieck}}_{f_1}, \underbrace{b = a}_{f_2}\}$$

R auf f_1 angewendet ergibt

$$F : \{\underbrace{a : \mathsf{Dreieck}}_{f_1}, \underbrace{b = a}_{f_2}, \underbrace{a : \mathsf{geometrisch}}_{f_3}\}$$

E_{6_1} auf f_1 und f_3 ergibt

$$F : \{\underbrace{b = a}_{f_2}, \underbrace{b : \mathsf{Dreieck}}_{f_4}, \underbrace{b : \mathsf{geometrisch}}_{f_5}\}$$

Jetzt ist R auf f_4 anwendbar. Diese Anwendung ist jedoch redundant.

Zur Verminderung solcher „Leerschlüsse" weißt man den Gleichheitsregeln höchstmögliche Priorität zu. Allerdings kann man sie damit nicht verhindern, denn es ist ja möglich, daß die Gleichheit zweier Objekte das Ergebnis einer SiR mit niedriger Priorität ist.

Eine Möglichkeit redundante[5] Regelanwendungen zu verhindern besteht darin, die Prozedur **bekannt** (vgl. Seite 31) auch diejenigen Fakten berechnen zu lassen, die durch Anwendung einer Propagierungsregel[6] aus bereits angewendeten Fakten entstanden sind. Jedoch muß man dann auch in den teilinstantiierten Regeln die Objekte durch ihren Repräsentanten ersetzten.[7]

Beispiel 5.2 *Sei* **bekannt** *wie oben beschrieben erweitert.*

$$R : ?x < ?y, ?y < ?z \Longrightarrow \mid ?x < ?z$$

$$F : \{\underbrace{a < d}_{f_1}, \underbrace{d < c}_{f_2}, \underbrace{b < f}_{f_3}, \underbrace{b = d}_{f_4}\}$$

Teilinstantiierung von R jeweils mit f_1, f_2, f_3 liefert

$$R_1 : d < ?z \Longrightarrow \mid a < ?z$$

$$R_2 : c < ?z \Longrightarrow \mid d < ?z$$

$$R_3 : f < ?z \Longrightarrow \mid b < ?z$$

[5] bzgl. Gleichheit

[6] i.e. eine Regel wie $E_{6_1} - E_{6_3}$

[7] Das erhöht natürlich den Aufwand beim Backtracking, weil somit auch Regeln zurückgesetzt werden müssen.

somit ist nur noch R_1 auf f_2 anwendbar und liefert

$$F : \{\underbrace{a < d}_{f_1}, \underbrace{d < c}_{f_2}, \underbrace{b < f}_{f_3}, \underbrace{b = d}_{f_4}, \underbrace{a < c}_{f_5}\}$$

Jetzt sind R, R_1, R_2 und R_3 deaktiv (weil nicht mehr anwendbar). Die Propagierung von f_4 über F (mit geeigneten SiR) liefert

$$F : \{\underbrace{b < f}_{f_3}, \underbrace{b = d}_{f_4}, \underbrace{a < c}_{f_5}, \underbrace{a < b}_{f_6}, \underbrace{b < c}_{f_7}\}$$

Beachte, daß R weder auf (das neue) f_6 noch auf (das ebenfalls neue) f_7 anwendbar ist, da R bereits auf f_1 bzw. f_2 angewendet wurde. Es muß also R_1 aktiviert und dort d durch b ersetzt werden, um $a < f$ aus f_6 und f_3 ableiten zu können.

Wir wollen diese Überlegungen in einer Auswahlstrategie umsetzen. Dazu stellen wir uns folgende Datenstrukturen vor:

Fakten. Ein Fakt besteht aus drei Einträgen, die wir im Folgenden **Slot** nennen. Wir beschreiben diese Slots durch Angabe ihres Namens und die beabsichtigte Belegung.

- id: Hier steht eine *eindeutige* Identifikation.
- data: Hier wird der Datenteil also der eigentliche Fakt abgelegt. Dieser Teil wird gegen die Head-Atome der Regeln gematcht.
- rule: Hier befindet sich eine Liste von Regeln (genauer von Regel-ID's). Die Idee ist, das in diesem Slot diejenigen Regeln protokolliert werden, die auf diesen Fakt *erfolgreich* angewendet wurden.

Regeln. Eine Regel hat folgende Slots:

- id: eine *eindeutige* Identifikation
- presumed: eine Liste der Fakten, die die Regel teilinstantiiert haben[8]
- applied-to: ein Pointer auf die Fakten-Basis, desse Bedeutung wir später erläutern
- priority: die Priorität der Regel
- head: Head-Atome
- guard: Guard-Atome
- body: Body-Atome

Wir unterstellen, daß es auf jeden Slot eine Zugriffsfunktion gibt, die wir zweckmäßigerweise mit dem Slot-Namen bezeichnen. Die Fakten seien in einer rechtsoffenen Kette (der **Fakten-Basis**) angeordnet. Neue Fakten werden von rechts an diese Kette angefügt. Der applied-to Slot einer Regel zeigt zwischen zwei Elemente

[8]Beachte, daß diese Fakten bei der Regelanwendung eine wichtige Rolle spielen (vgl. dazu die Anmerkung zu Algorithmus 4.12).

dieser Kette. Dabei stehen links des Zeigers die Fakten, die der Regel bereits bekannt sind[9], rechts des Zeigers die Fakten, die der Regel unbekannt sind. Die Auswahl einer Regel mit einem anwendbaren Fakt funktioniert jetzt so:

[1] $S :=$ „wähle eine aktive Regel maximaler Priorität“ [10]
if $S = \downarrow$ **then** **STOP** [11]
[2] $f :=$ „erstes Element rechts von applied-to(S)“
[3] **if** $f = \downarrow$ **then** „deaktiviere S“ **goto** [1] [12]
[4] „verschiebe den applied-to Slot von S um eine Stelle nach rechts“
[5] **if** id(S) $\in$ rule(f) **then** **goto** [1]
[6] **if** „data(f) matcht das erste Head-Atom von S“
then „feuere S mit f“ [13]
[7] **goto** [1]

Durch Schritt [5] vermeiden wir redundante Matches. Das „feuern einer Regel S mit einem Fakt f“ bedeutet: Ist S eine Mehr-Kopf-Regel, so wird S mit f teilinstantiiert und rule(f) um id(S) erweitert. Ist S eine Ein-Kopf-Regel, so wird überprüft, ob S gültig ist (d.h. existieren die Fakten, die S teilinstantiiert haben noch) und ob die Guard-Atome wahr sind. Ist beides der Fall, so wird S auf f angewendet und rule(f) um id(S) erweitert. Ist S nicht mehr gültig, so wird die Regel gelöscht (nicht nur deaktiviert).[14] Ist (mindestens) ein Guard-Atom unerfüllt, so wird S deaktiviert, ohne daß rule(f) um id(S) erweitert wird.

Was passiert bei Gleichheitspropagierung?

Wird ein Fakt A (wie alt) mittels einer Regel, die Gleichheiten propagiert (das sind Regeln wie $E_{6_1} - E_{6_3}$ von Seite 37), durch einen Fakt N (wie neu) ersetzt, so erbt N den rule-Slot von A. Das bedeutet, daß wegen Schritt [5] eine Regel, die bereits auf A angewendet wurde, nicht noch einmal auf N angewendet wird. Gleichzeitig wird N an der rechten Seite der Fakten-Basis eingefügt, d.h. er steht allen Regeln zur Verfügung. Beachte, daß eine Regel S auch dann auf N angewendet werden kann, wenn ihr A bekannt war, beispielsweise dann, wenn das erste Head-Atom von S nicht A aber N matcht. Werden Objekte in einer teilinstantiierten Regel T durch ihre Repräsentanten ersetzt, so wird der applied-to-Slot von T auf den Anfang der Fakten-Basis gesetzt. Dadurch stehen alle Fakten dieser 'neuen' Regel zur Inspektion zur Verfügung.

[9] Hier bedeutet „bekannt sein“, daß das erste Head-Atom der Regel bereits gegen diesen Fakt gematcht wurde (erfolgreich oder nicht).

[10] Wir gehen davon aus, daß aktive und deaktive Regeln in getrennten Bereichen stehen.

[11] $S = \downarrow$ bedeutet, daß es keine aktive Regel mehr gibt.

[12] $\downarrow$ sei das letzte Element der Kette. Insbesondere ist $\downarrow$ kein Fakt.

[13] d.h. teilinstantiieren oder anwenden oder deaktivieren

[14] Die Ungültigkeit einer Regel ist etwas endgültiges. Ein gelöschter Fakt kann nie wieder entstehen, sondern höchstens ein Fakt mit gleichem Inhalt (Datum). Dieser neue(!) Fakt muß gegebenenfalls die entsprechenden Regeln (nochmal) teilinstantiieren.

5.4 Der Konsistenztest für $\mathcal{ALC}(\mathcal{Q})$

Wie wir schon in Kapitel 3 bemerkt haben, behält der Konsistenztest seine prinzipielle Einfachkeit auch dann, wenn man zu Erweiterungen von $\mathcal{ALC}$ übergeht. Wir wollen im Folgenden die Transformations- und Clashregeln für den Konsistenztest von $\mathcal{ALC}(\mathcal{Q})$ angeben. Dabei handelt es sich bei $\mathcal{ALC}(\mathcal{Q})$ um die Integration von $\mathcal{Q}$ zusammen mit den Relationen $<$ und $\leq$ in $\mathcal{ALC}$, wie sie in [2] allgemein beschrieben ist. Ich erwähne kurz die Auswirkungen der Integration eines beliebigen konkreten Bereiches $\mathcal{D}$ in $\mathcal{ALC}$.[15]

Ein **konkreter Bereich** $\mathcal{D}$ besteht aus einer Trägermenge dom($\mathcal{D}$) und einer Menge von Relationssymbolen pred($\mathcal{D}$). Dabei wird vorausgesetzt, daß pred($\mathcal{D}$) abgeschlossen unter Negation ist und einen Namen für dom($\mathcal{D}$) enthält.[16] Zu jedem Relationssymbol $P \in \text{pred}(\mathcal{D})$ gehört eine Stelligkeit n und eine n-stellige Relation $P^{\mathcal{D}} \subseteq dom(\mathcal{D})^n$. $\mathcal{ALC}(\mathcal{D})$ erlaubt funktionale Rollen (eine funktionale Rolle nennt man auch Attribut oder **Feature**[17]) in Werterestiktionen, sowie die Anwendung von Relationen über $\mathcal{D}$ auf Attribut-Ketten. Eine Attribut-Kette ist ein nichtleeres Wort über der Menge der Attribut-Namen.[18] Ferner werden bei der Integration eines konkreten Bereiches die Definitionen aus Kapitel 3 folgendermaßen erweitert:

Definition 5.3

1. *Konzeptterme (vgl. Definition 3.1)*
   - *Existenz- und Werterestriktion werden auch für Attribut-Namen zugelassen.*
   - *Für ein n-stelliges Relationssymbol $P \in dom(\mathcal{D})$ und Attribut-Ketten $u_1, \ldots, u_n$ ist $P(u_1, \ldots, u_n)$ ein Konzeptterm, die sogenannte Relationsrestriktion.*
2. *Interpretation der Konzeptterme (vgl. Definition 3.3)*
   Die Trägermenge der Interpretation dom($\mathcal{I}$) und die Trägermenge des konkreten Bereiches dom($\mathcal{D}$) müssen disjunkt sein. Zur Unterscheidnung werden wir die Elemente von dom($\mathcal{I}$) **abstrakt** *die von dom($\mathcal{D}$)* **konkret** *nennen.*
   Jedem Attribut-Namen f wird eine partielle Funktion $f^{\mathcal{I}}$ von dom($\mathcal{I}$) nach dom($\mathcal{I}$) $\cup$ dom($\mathcal{D}$) zugeordnet.
   Für eine Attribut-Kette $u = f_1, \ldots, f_n$ ist $u^{\mathcal{I}}$ die Komposition ${f_1}^{\mathcal{I}} \circ {f_2}^{\mathcal{I}} \circ \cdots \circ {f_n}^{\mathcal{I}}$ der partiellen Funktionen ${f_1}^{\mathcal{I}}, {f_2}^{\mathcal{I}}, \ldots, {f_n}^{\mathcal{I}}$. Dabei ist die Komposition von links nach rechts zu lesen, d.h. ${f_1}^{\mathcal{I}} \circ {f_2}^{\mathcal{I}} \circ \cdots \circ {f_n}^{\mathcal{I}}$ bedeutet zuerst

[15] Eine detaillierte Einführung findet man in [2].

[16] Solche Bereiche heißen in [2] zulässig (admissible).

[17] Ist r ein Rollenname und soll y durch $(x, y) : r$ *eindeutig* festgelegt sein, so nennt man r ein Attribut.

[18] Die Idee ist beispielsweise Frauen, die älter als ihr Ehemann sind, durch das Konzept Ehefrau$\sqcap <$ (Alter-des-Ehemanns, Alter) zu beschreiben. Dabei sind Alter-des-Ehemanns und Alter Attribute.

$f_1{}^{\mathcal{I}}$ dann $f_2{}^{\mathcal{I}}$ usw. anwenden. Eine Relationsrestriktion $P(u_1,\ldots,u_n)$ wird interpretiert durch

$$P(u_1,\ldots,u_n)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid \exists r_1 \ldots \exists r_n \in dom(\mathcal{D}) : u_1{}^{\mathcal{I}}(x) = r_1 \wedge \cdots \wedge u_n{}^{\mathcal{I}}(x) = r_n \wedge (r_1,\ldots,r_n) \in P^{\mathcal{D}}\}$$

3. *Wie schon in $\mathcal{ALC}$, so wollen wir auch die Konzeptterme von $\mathcal{ALC}(\mathcal{D})$ in Negationale Normalform überführen. Dazu erweitern wir die Rewrite-Regeln von Seite 16 um*

$$\begin{aligned} \neg(\forall f.C) &\longrightarrow (\exists f.\neg C) \sqcup Top_{\mathcal{D}}(f) \\ \neg(\exists f.C) &\longrightarrow (\forall f.\neg C) \sqcup Top_{\mathcal{D}}(f) \\ \neg P(u_1,\ldots,u_n) &\longrightarrow \overline{P}(u_1,\ldots,u_n) \sqcup (\forall u_1.Top) \sqcup \ldots \sqcup (\forall u_n.Top) \end{aligned}$$

Dabei sei f ein Attribut-Name, $Top_{\mathcal{D}}$ der Name für $dom(\mathcal{D})$, Top die Abkürzung für $A \sqcup \neg A$ für einen beliebigen Konzeptnamen A und $\overline{P}$ die Negation zu $P \in pred(\mathcal{D})$.

4. *Assertionale Axiome (vgl. Definition 3.4)*
Seien OC und OA zwei disjunkte *Mengen von Objektnamen. Sei f ein Attribut-Name, P ein n-stelliges Relationssymbol von $\mathcal{D}$, a, b Elemente von OA, $y, y_1, \ldots, y_n$ Elemente von OC. Dann sind, neben den in Definiton 3.4 definierten, auch folgende Ausdrücke assertionale Axiome:*
$$(a,b) : f \qquad (a,y) : f \qquad (y_1,\ldots,y_n) : P$$

5. *Interpretation assertionaler Axiome (vgl. Definition 3.5)*
Eine Interpretation für eine (erweiterte) A-Box ist eine Interpretation für $\mathcal{ALC}(\mathcal{D})$, die zusätzlich jedem Objektnamen $a \in OA$ ein Objekt $a^{\mathcal{I}} \in dom(\mathcal{I})$, jedem Objektnamen $x \in OC$ ein Objekt $x^{\mathcal{I}} \in dom(\mathcal{D})$ zuweist. Eine Interpretation $\mathcal{I}$ erfüllt ein assertionales Axiom
$$\begin{aligned} (a,b) : f \quad &gdw \quad f^{\mathcal{I}}(a^{\mathcal{I}}) = b^{\mathcal{I}} \\ (a,y) : f \quad &gdw \quad f^{\mathcal{I}}(a^{\mathcal{I}}) = y^{\mathcal{I}} \\ (y_1,\ldots,y_n) : P \quad &gdw \quad (y_1{}^{\mathcal{I}},\ldots y_n{}^{\mathcal{I}}) \in P^{\mathcal{D}} \end{aligned}$$

Attribute und Relationsrestriktionen wirken sich sowohl auf die Regeln zur Vervollständigung einer A-Box, als auch auf offensichtliche Widersprüche aus. Enthält eine A-Box Axiome $(a,b) : f$ und $(a,c) : f$, so nennt man das eine **Gabelung**. Da f als partielle Funktion interpretiert wird, müssen b und c als dasselbe Objekt interpretiert werden. Beachte, daß sich das mit den Gleichheitsregeln vom Anfang dieses Abschnitts propagieren läßt. Betrachte folgende SiR: $(?x,?y) :? f, (?x,?z) :? f \Longrightarrow \mathsf{feature?}(f) \mid ?y = ?z$. Zusammen mit geeigneten Regeln, die die Gleichheit propagieren, lassen sich so alle Gabelungen aus einer A-Box entfernen.

Da wir oben gefordert haben, daß konkrete und abstrakte Trägermenge disjunkt sind, ist ein Fakt, der die Gleichheit zwischen einem abstrakten und einem konkreten Objekt propagiert, ein offensichtlicher Widerspruch ($\rightsquigarrow R_{21}$). Ferner kann ein konkretes Objekt nicht zu einem Konzept gehören ($\rightsquigarrow R_{20}$).

Die folgende Definition erweitert Definition 3.7:

Definition 5.4
Relationsrestriktionsregel. *Angenommen* $a : P(u_1, \ldots, u_n)$ *ist in* $\mathcal{A}$. *Dann wähle man für die Attribut-Ketten* $u_i = f_{i1}, \ldots, f_{in_i}$ *neue Objektenamen* $b_{i1}, \ldots, b_{in_i-1} \in OA$ *und* $x_i \in OC$ *und erweitere* $\mathcal{A}$ *durch Hinzunahme folgender Axiome zu* $\mathcal{A}'$: $(a, b_{i1}) : f_{i1}$, $(b_{i1}, b_{i2}) : f_{i2}$, ..., $(b_{in_i-1}, x_i) : f_{in_i}$, $(x_1, \ldots, x_n) : P$.

Nach dieser langen Einleitung definieren wir jetzt die Simplification Rules, die den Konsistenztest für $\mathcal{ALC}(\mathcal{Q})$ implementieren.

R_1 :	$?x : \neg(?s \sqcap ?t)$	$\Longleftrightarrow$		$\mid ?x : (\neg ?s) \sqcup (\neg ?t)$	; 5
R_2 :	$?x : \neg(?s \sqcup ?t)$	$\Longleftrightarrow$		$\mid ?x : (\neg ?s) \sqcap (\neg ?t)$	; 5
R_3 :	$?x : \neg(\neg ?s)$	$\Longleftrightarrow$		$\mid ?x : ?s$	; 5
R_4 :	$?x : \neg(\exists ?r.?s)$	$\Longleftrightarrow$	ROLE?$(?r)$	$\mid ?x : (\forall ?r.\neg ?s)$	; 5
R_5 :	$?x : \neg(\forall ?r.?s)$	$\Longleftrightarrow$	ROLE?$(?r)$	$\mid ?x : (\exists ?r.\neg ?s)$	; 5
R_6 :	$?x : \neg(\forall ?f.C)$	$\Longleftrightarrow$	FEATURE?$(?f)$	$\mid ?x : (\exists f.\neg C) \sqcup Top_{\mathcal{Q}}(?f)$	; 5
R_7 :	$?x : \neg(\exists ?f.C)$	$\Longleftrightarrow$	FEATURE?$(?f)$	$\mid ?x : (\forall f.\neg C) \sqcup Top_{\mathcal{Q}}(?f)$	; 5
R_8 :	$?x : \neg P_<(?v_1, ?v_2)$	$\Longleftrightarrow$		$\mid ?x : P_\leq(?v_2, ?v_1) \sqcup (\forall ?v_1.Top \sqcup \forall ?v_2.Top)$	; 5
R_9 :	$?x : \neg P_\leq(?v_1, ?v_2)$	$\Longleftrightarrow$		$\mid ?x : P_<(?v_2, ?v_1) \sqcup (\forall ?v_1.Top \sqcup \forall ?v_2.Top)$	; 5
R_{10} :	$?x : P_<(?v_1, ?v_2)$	$\Longleftrightarrow$	FRESH$(?y_1, ?y_2)$	$\mid (?x, ?y_1) : ?v_1, (?x, ?y_2) : ?v_2, ?y_1 < ?y_2$	; 1
R_{11} :	$?x : P_\leq(?v_1, ?v_2)$	$\Longleftrightarrow$	FRESH$(?y_1, ?y_2)$	$\mid (?x, ?y_1) : ?v_1, (?x, ?y_2) : ?v_2, ?y_1 \leq ?y_2$	; 1
R_{12} :	$(?x, ?y) : ?f \circ ?v$	$\Longleftrightarrow$	FRESH$(?z)$	$\mid (?x, ?z) : ?f, (?z, ?y) : ?v$	; 1
R_{13} :	$?x : (?s \sqcap ?t)$	$\Longleftrightarrow$		$\mid ?x : ?s, ?x : ?t$	; 0
R_{14} :	$?x : (?s \sqcup ?t)$	$\Longleftrightarrow$		$\mid$ NONDET$(?x : ?s, ?x : ?t)$	; −1
R_{15} :	$?x : \exists ?r.?s$	$\Longleftrightarrow$	FRESH$(?y)$	$\mid (?x, ?y) : ?r, ?y : ?s$	; 0
R_{16} :	$?x : \forall ?r.?s, (?x, ?y) : ?r$	$\Longrightarrow$		$\mid ?y : ?s$	; 0
R_{17} :	$?x : ?s, ?x : \neg ?s$	$\Longleftrightarrow$		$\mid$ CLASH	; 10
R_{18} :	$?x : ?s$	$\Longleftrightarrow$	$?s \doteq ?t$	$\mid ?x : ?t$	; 6
R_{19} :	$?x : \neg ?s$	$\Longleftrightarrow$	$?s \doteq ?t$	$\mid ?x : \neg ?t$	; 6
R_{20} :	$?x : ?s$	$\Longleftrightarrow$	CONCRETE?$(?x)$	$\mid$ CLASH	; 10
R_{21} :	$?x = ?a$	$\Longleftrightarrow$	DISJ?$(?x, ?a)$	$\mid$ CLASH	; 10
R_{22} :	$?x : Top_{\mathcal{Q}}(?f)$	$\Longleftrightarrow$	FRESH$(?y)$	$\mid (?x, ?y) : ?f, ?y : \mathcal{Q}$	; 1
R_{23} :	$?x : \mathcal{Q}, ?x : ?s$	$\Longleftrightarrow$		$\mid$ CLASH	; 10
R_{24} :	$(?x, ?y_1) : ?f, (?x, ?y_2) : ?f$	$\Longrightarrow$	FEATURE?$(?f)$	$\mid ?y_1 = ?y_2$	; 8

Bemerkungen:

- $R_1 - R_9$ überführen die A-Box in NNF. Dabei sind ROLE? und FEATURE? Prädikate, die ein Symbol auf Rollen- bzw. Feature-Namen überprüfen.
- R_7, R_8. $Top_{\mathcal{Q}}$ ist ein Relationssymbol, das den konkreten Bereich ($\mathcal{Q}$) kennzeichnet. $Top_{\mathcal{Q}}$ wird in den Regeln R_{22}, R_{23} weiter ausgewertet.
- R_{10}, R_{11} schlagen die Brücke zu den in 5.2 definierten Regeln (siehe auch unten).
- R_{12} zerlegt Feature-Ketten.
- R_{20}. CONCRETE? ist ein Prädikat, das überprüft, ob ein Objektname zu OC gehört.
- R_{21}. DISJ? ist ein Prädikat, das überprüft, ob der eine Objektname zu OC der andere zu OA gehört.
- R_{23}. Ein Objekt ist entweder konkret ($?x : \mathcal{Q}$) oder abstrakt ($?x :?s$).
- R_{24} liefert die Verbindung zu den Regeln der Gleichheit aus 5.3.
- Beachte das massive Auftreten von Nichtdeterminismen ($R_6 - R_9$).

Mit den Regeln, die im konkreten Bereich schließen (vgl. 5.2),

$$
\begin{array}{rrcl}
R_{25}: & ?x \leq ?y, ?y \leq ?z & \Longrightarrow & \mid ?x \leq ?z \;;\; 100 \\
R_{26}: & ?x \leq ?x & \Longleftrightarrow & \mid \text{TRUE} \;;\; 120 \\
R_{27}: & ?x < ?y, ?y < ?z & \Longrightarrow & \mid ?x < ?z \;;\; 100 \\
R_{28}: & ?x < ?y, ?y < ?x & \Longleftrightarrow & \mid \text{CLASH} \;;\; 106 \\
R_{29}: & ?x < ?x & \Longleftrightarrow & \mid \text{CLASH} \;;\; 121 \\
R_{30}: & \neg(?x \leq ?y) & \Longleftrightarrow & \mid ?y < ?x \;;\; 110 \\
R_{31}: & \neg(?x < ?y) & \Longleftrightarrow & \mid ?y \leq ?x \;;\; 110 \\
R_{32}: & ?x \leq ?y, ?y < ?z & \Longrightarrow & \mid ?x < ?z \;;\; 100 \\
R_{33}: & ?x < ?y, ?y \leq ?z & \Longrightarrow & \mid ?x < ?z \;;\; 100 \\
R_{34}: & ?x < ?y, ?x \leq ?y & \Longleftrightarrow & \mid ?x < ?y \;;\; 115 \\
R_{35}: & ?x < ?y, ?y \leq ?x & \Longleftrightarrow & \mid \text{CLASH} \;;\; 116
\end{array}
$$

sowie den Regeln, die die Gleichheit explizieren (vgl. 5.3)[19],

$$
\begin{array}{rrcll}
R_{36}: & ?x = ?y & \Longrightarrow & & \mid ?x \leq ?y, ?y \leq ?x \;;\; 199 \\
R_{37}: & ?x = ?y, ?x = ?z & \Longrightarrow & & \mid ?y = ?z \;;\; 205 \\
R_{38}: & ?x = ?y, ?z = ?x & \Longrightarrow & & \mid ?y = ?z \;;\; 205 \\
R_{39}: & ?y = ?x, ?z = ?x & \Longrightarrow & & \mid ?y = ?z \;;\; 205 \\
R_{40}: & ?x = ?x & \Longleftrightarrow & & \mid \text{TRUE} \;;\; 210 \\
R_{40}: & ?x = ?y & \Longleftrightarrow & ?x \rhd ?y & \mid ?y = ?x \;;\; 209 \\
R_{42_1}: & ?x :?c & \Longleftrightarrow & ?y = ?x & \mid ?y :?c \;;\; 200 \\
R_{42_2}: & (?x, ?y) :?r & \Longleftrightarrow & ?z = ?x & \mid (?z, ?y) :?r \;;\; 200 \\
R_{42_3}: & (?x, ?y) :?r & \Longleftrightarrow & ?z = ?y & \mid (?x, ?z) :?r \;;\; 200
\end{array}
$$

[19] Beachte die neue Regel R_{36}.

vervollständigen wir den Konsistenztest für $\mathcal{ALC}(\mathcal{Q})$. Beachte, daß das durch Gleichheit explizierte Wissen mit höchster Priorität propagiert werden soll. Weiterhin möchte man die Ergebnisse aus dem konkreten Bereich so früh wie möglich berücksichtigen.

Die Integration eines konkreten Bereiches führt also zu einem erheblichen Mehraufwand an Regeln. Ferner treten jetzt verstärkt Nichtdeterminismen auf, deren Abarbeitung den Rechenaufwand in die Höhe schnellen lassen.

5.5 Weiterführende Beispiele

Um die Flexibilität des Simplification Rule Formalismus zu demonstrieren, zeigen wir noch Beispiele außerhalb terminologischer Sprachen.

5.5.1 Logische Programme

Es gibt einen Ansatz, terminologische Inferenzen mit logischer Programmierung zu koppeln. In *TaxLog* bettet man ein terminologisches Wissensrepräsentationssystem (*TAXON*) in einen Regelformalismus ein. Die Idee dabei ist, Schwächen von *TAXON*[20] durch einen Regelformalismus aus einfach rückwärts verketteten PROLOG-artigen Regeln aufzufangen. Eine ausführliche Motivation und Definiton von *TaxLog* findet man in [1]. In Anhang B zeigen wir wie sich ein TaxLog-Programm mit dem SiR-Formalismus ausdrücken läßt.

5.5.2 Unifikation

Wir geben im Folgenden eine Regelmenge an, die zwei Terme s und t unifiziert. Dazu fassen wir einen Term als eine geschachtelte Liste auf; die Listen selbst wiederum als durch den cons-Operator „." (wie in Lisp) aufgebaut. Ein Term ist also hier (i) eine Variable oder (ii) $f.t_1.\cdots.t_n$ wobei f ein n-stelliges Funktionssymbol und $t_1,\ldots,t_n$ Terme sind. Konstanten sind nullstellige Funktionssymbole. Folgende Regeln definieren einen Unifikationsalgorithmus (ohne Occur-Check):

$$
\begin{array}{lrcll}
U_1: & ?f = ?g & \Longrightarrow & \mathfrak{F}(?f), \mathfrak{F}(?g), ?f \neq ?g \mid \text{CLASH} & ; 5 \\
U_2: & ?t_1.?t_2 = ?s_1.?s_2 & \Longleftrightarrow & \mid ?t_1 = ?s_1, ?t_2 = ?s_2 & ; 0 \\
U_3: & ?v = ?v & \Longleftrightarrow & \mid \top & ; 9 \\
U_4: & ?v_1 = ?t_1, ?v_2 = ?t_2, & & & \\
 & ?v_1 = ?v_2 & \Longrightarrow & \mathfrak{V}(?v_1), \mathfrak{V}(?v_2), \mathfrak{T}(?t_1), \mathfrak{T}(?t_2) \mid ?t_1 = ?t_2 & ; 10 \\
U_5: & ?v = ?t_1, ?v = ?t_2 & \Longrightarrow & \mathfrak{V}(?v), \mathfrak{T}(?t_1), \mathfrak{T}(?t_2) \mid ?t_1 = ?t_2 & ; 10 \\
U_6: & ?t = ?v & \Longleftrightarrow & \mathfrak{T}(?t), \mathfrak{V}(?v) \mid ?v = ?t & ; 2 \\
U_{g1}: & ?x = ?y, ?x = ?z & \Longrightarrow & \mathfrak{V}(?x), \mathfrak{V}(?y), \mathfrak{V}(?z) \mid ?y = ?z & ; 3 \\
U_{g2}: & ?x = ?y, ?z = ?x & \Longrightarrow & \mathfrak{V}(?x), \mathfrak{V}(?y), \mathfrak{V}(?z) \mid ?y = ?z & ; 3 \\
U_{g3}: & ?y = ?x, ?z = ?x & \Longrightarrow & \mathfrak{V}(?x), \mathfrak{V}(?y), \mathfrak{V}(?z) \mid ?y = ?z & ; 3 \\
U_{g4}: & ?x = ?y & \Longleftrightarrow & \mathfrak{V}(?x), \mathfrak{V}(?y), ?x \rhd ?y \mid ?y = ?x & ; 4
\end{array}
$$

Dabei evaluiert $\mathfrak{T}(t)$ genau dann zu wahr, wenn t keine Variable , $\mathfrak{V}(v)$ wenn v eine Variable und $\mathfrak{F}(f)$ wenn f ein Funktionssymbol ist. $\top$ ist das Symbol für den leeren Fakt.

[20] keine Aggregation von Objekten durch Einführung neuer Individuen, (zu) geringe Ausdrucksfähigkeit (man beschränkt sich auf entscheidbare Teilsprachen der Prädikatenlogik erster Ordnung), Probleme beim Zulassen rekursiver Konzeptdefinitionen, Probleme transitive Hüllen von Rollen zuzulassen (beides führt in Zusammenhang mit konkreten Bereichen zu unentscheidbaren Subsumptionsproblemen)

Erläuterung

U_1. Zwei Terme, die mit verschiedenen Funktionssymbolen beginnen, sind nicht unifizierbar. Solche Gleichungen entstehen durch Anwendung von U_2.

U_2. Zwei Terme werden unifiziert, indem man den *car*- und den *cdr*-Teil unifiziert.[21]

U_3. Tautologien werden aus der Fakten-Basis entfernt.

U_4, U_5. Gleichheiten werden propagiert.

U_6. Orientierung.

$U_{g1} - U_{g4}$. Es wird die transitive Hülle der Gleichheit über den Variablen erzeugt. Das sind genau die Regeln, die wir schon im Kapitel über Gleichheit kennengelernt haben.

Bemerkung: Wenn man möchte, kann man die Typabfragen in den Guard-Atomen durch eine passende Repräsentation von Variablen und Termen vermeiden. Dadurch verschiebt man die Anwendbarkeitentscheidung in das Matching.

Zwei Terme σ und τ werden unifiziert, indem man $\{\sigma = \tau\}$ bzgl. $U_1 - U_{g4}$ stabilisiert. Das daraus resultierende Gleichungssystem definiert den Unifikator.

5.5.3 Lokale Konsistenz über finiten Domänen

Gegeben sei eine *endliche* Menge von Variablen $\mathfrak{V}$. Zu jeder Variable existiere ein *endlicher* Definitionsbereich. Schreibe dom(x,D) um zu kennzeichnen, daß D der Definitionsbereich von x ist. Sei $\mathfrak{M}$ eine endliche Menge von Atomen $R_k(x_i, x_j)$, wobei R_k eine Relation über $\mathrm{dom}(x_i) \times \mathrm{dom}(x_j)$ ist.[22] Mit Hilfe der Relationen lassen sich die Definitionsbereiche der Variablen wie folgt einschränken:

> Betrachte $R(x,y) \in \mathfrak{M}$. Sei D der Definitionsbereich von x, E der Definitionsbereich von y. Entferne alle d aus D, für die $\neg R(d,e)$ für alle e aus E gilt und umgekehrt entferne alle e aus E, für die $\neg R(d,e)$ für alle d aus D gilt.

Mit SiR läßt sich das so darstellen:

$$
\begin{array}{lrcll}
L_1: & \mathrm{chdom}(?x,?\tilde{D}), \mathrm{dom}(?x,?D) & \Longleftrightarrow & \mid \mathrm{dom}(?x,?\tilde{D}) & ;\ 5 \\
L_2: & ?R(?x,?y), \mathrm{dom}(?x,?D), \mathrm{dom}(?y,?E) & \Longrightarrow & \text{1-falsify}(?R,?D,?E,?\tilde{D}) & \\
 & & & \mid \mathrm{chdom}(?x,?\tilde{D}) & ;\ 0 \\
L_3: & ?R(?x,?y), \mathrm{dom}(?x,?D), \mathrm{dom}(?y,?E) & \Longrightarrow & \text{2-falsify}(?R,?D,?E,?\tilde{E}) & \\
 & & & \mid \mathrm{chdom}(?x,?\tilde{E}) & ;\ 0 \\
L_4: & \mathrm{dom}(?x,?D) & \Longrightarrow & ?D = \emptyset \mid \textsc{clash} & ;\ -1
\end{array}
$$

[21] Beispiel: $f.s.t = g.u.v$ wird zu $f = g$ und $s.t = u.v$, was wiederum zu $s = u$ und $t = v$ wird. Dabei sind f und g Funktionssymbole, s, t, u und v Terme.

[22] In [23] wird gezeigt, daß die Einschränkung auf zweistellige Relationen ohne Beschränkung der Allgemeinheit erfolgen kann.

wobei

```
procedure 1-falsify (R:RELATION,D,E:DOMAIN, VAR:D̃: DOMAIN): BOOLEAN;
      D̃ := ∅;
      for d ∈D do
         H:= E; /* H ist eine Hilfsvariable */
         repeat
               h:= first(H);
               if R(d,h) then D̃ := D̃ ∪ {d};
               /* hier werden die Elemente aufgesammelt, die erfüllbar sind */
               H:= rest(H);
         until R(d,h) OR H=∅;
      od
      if D̃=D /* dann läßt sich D nicht einschränken */
         then return F
         else return T
      fi;
end 1-falsify.
```

und analog für das andere Argument von R:

```
procedure 2-falsify (R:RELATION,D,E:DOMAIN, VAR:Ẽ: DOMAIN): BOOLEAN;
      Ẽ := ∅;
      for e ∈ E do
         H:= D; /* H ist eine Hilfsvariable */
         repeat
               h:= first(H);
               if R(h,e) then Ẽ := Ẽ ∪ {e};
               /* hier werden die Elemente aufgesammelt, die erfüllbar sind */
               H:= rest(H);
         until R(h,e) OR H=∅;
      od
      if Ẽ=E /* dann läßt sich E nicht einschränken */
         then return F
         else return T
      fi;
end 2-falsify.
```

Bemerkungen:

- Wir nehmen in den falsify-Prozeduren an, daß die Elemente vom Typ DOMAIN in Listenform vorliegen. **first** und **rest** sind dann die gewöhnlichen Listenoperationen, die das erste Listenelement bzw. die Liste ohne das erste Listenelement liefern.
- dom(?x,?D), chdom(?x,?$\tilde{D}$) und ?R(?x,?y) sind Muster für Fakten.

- Die Einschränkung eines Definitionsbereichs (L_1) hat höchste Priorität.

- Jede Menge $\{dom(x_i, D_i), R_j(x_k, x_l)\}$ läßt sich bzgl. dieser Regeln stabilisieren. Beachte, daß die Anwendung von L_2, L_3 stets eine Anwendung von L_1 nach sich zieht, wobei endliche Definitionsmengen kleiner werden.

- Durch Hinzunehmen der Regel
  L_5 : dom($?x, ?D$) $\Longleftrightarrow$ SPLIT($?D, ?D_1, ?D_2$) |
  dom($?x, ?D_1$) $\bigvee$ dom($?x, ?D_2$) ; -2
  erhält man sogar einen globalen Konsistenztest.[23] Das ist ein weiteres Beispiel dafür wie einfach sich SiR-Sätze zu komplexeren Anwendungen erweitern lassen.

Beachte, daß hier ein (unvollständiger) „konkreter Bereich" definiert wird.

In [17] werden Heuristiken für einen effizienten Konsistenztest über finiten Domänen beschrieben. SiR erscheinen eine geeignete Grundlage, um Auswirkungen solcher Heuristiken zu testen.

5.5.4 Erfüllbarkeit von Ungleichungen

Sei $\mathfrak{U}$ eine *endliche* Menge von linearen Ungleichungen im Polynomring der rationalen Zahlen. Sei $\{x_1, \ldots, x_n\}$ die Menge der in $\mathfrak{U}$ vorkommenden Variablen. Wir wollen wissen, ob $\mathfrak{U}$ erfüllbar ist, d.h. gibt es eine Belegung für $x_1, \ldots, x_n$, so daß alle Ungleichungen erfüllt sind?

Für eine Ungleichung $\mathfrak{u}$ heißt $b < a_1x_1 + \cdots + a_nx_n$ bzw. $b \leq a_1x_1 + \cdots + a_nx_n$ die **Standardform**. Es ist klar, daß sich alle Ungleichungen mit *Äquivalentzumformungen* auf Standardform bringen lassen. D.h. eine Menge von Ungleichungen ist genau dann erfüllbar, wenn die Standardformen der Ungleichungen erfüllbar sind. Ist $\mathfrak{u} : b\ \mathcal{R}\ a_1x_1 + \cdots + a_nx_n\ \mathcal{R} \in \{<, \leq\}$ eine Ungleichung in Standardform, so heißen $b, a_1, \ldots, a_n$ **Koeffizienten** von $\mathfrak{u}$. Ist $a_1 = a_2 = \cdots = a_n = 0$, so heißt $\mathfrak{u}$ **einfach**.

Die Idee ist jetzt, zu versuchen aus diesen Ungleichungen offensichtliche Widersprüche herzuleiten. Gelingt das, so ist $\mathfrak{U}$ unerfüllbar, sonst ist $\mathfrak{U}$ erfüllbar. Ein offensichtlicher Widerspruch ist eine einfache Ungleichung der Form $b < 0$ bzw. $b \leq 0$ für ein $b \geq 0$ bzw. $b > 0$. Zum Herleiten neuer Ungleichungen gibt es nur eine Regel:

> Seien $\mathfrak{u}$: $b_u\ \mathcal{R}_u\ u_1x_1 + \cdots + u_nx_n$ und $\mathfrak{v}$: $b_v\ \mathcal{R}_v\ v_1x_1 + \cdots + v_nx_n$ Ungleichungen ($\mathcal{R}_u, \mathcal{R}_v \in \{<, \leq\}$). Gibt es ein $i \in \{1, \ldots, n\}$ so daß $u_1 = \cdots = u_{i-1} = 0 = v_{i-1} = \cdots = v_1$ *und* $\alpha u_i + \beta v_i = 0$ für positive Zahlen α und β, so füge $\mathfrak{w}{:=}\alpha\mathfrak{u} + \beta\mathfrak{v}$ zu $\mathfrak{U}$ hinzu. Beachte, daß in $\mathfrak{w}$ die Variable x_i eliminiert wurde.

[23] Dabei ist SPLIT eine Guard-Funktion, die wahr ist, wenn $?D$ disjunkt in nichtleere Mengen $?D_1$ und D_2 zerlegbar ist.

$\mathfrak{w}$ heißt **Resolvente**. Dabei bedeutet $\alpha\mathfrak{u}$ die Multiplikation jedes Koeffizienten von $\mathfrak{u}$ mit α. Seien $\mathfrak{u}$: $b_u\ \mathcal{R}_u\ u_1x_1 + \cdots + u_nx_n$ und $\mathfrak{v}$: $b_v\ \mathcal{R}_v\ v_1x_1 + \cdots + v_nx_n$ $\mathcal{R}_i \in \{<, \leq\}, i = u, v$, dann ist $\mathfrak{u} + \mathfrak{v}$ definiert durch

$$\mathfrak{u} + \mathfrak{v} : b_u + b_v\ \mathcal{R}\ (u_1 + v_1)x_1 + \cdots + (u_n + v_n)x_n$$
$$\text{wobei } \mathcal{R} = \begin{cases} \leq, \text{ falls } \mathcal{R}_1 = \mathcal{R}_2 = \leq \\ <, \text{ sonst} \end{cases}$$

Diese Regel wird solange angewendet, bis ein offensichtlicher Widerspruch entsteht oder keine (neuen) Ungleichungen mehr hergeleitet werden können.

Wir wollen dieses Verfahren mit SiR definieren. Dazu treffen wir folgende Vereinbarungen:

Sei $\mathfrak{u}$: $b\ \mathcal{R}\ a_1x_1 + \cdots a_nx_n\ \ \mathcal{R} \in \{<, \leq\}$ eine Ungleichung, dann sei für jeden Koeffizienten eine Zugriffsfunktion definiert $Z_{a_i}(\mathfrak{u}) = a_i,\ i = 1, \ldots n$. Schreibe $a_i(\mathfrak{u})$ statt $Z_{a_i}(\mathfrak{u})$.

Zwei Ungleichungen $\mathfrak{u}$, $\mathfrak{v}$ heißen **vereinbar**, wenn es ein $i \in \{1, \ldots, n\}$ gibt, so daß $a_1(\mathfrak{u}) = a_1(\mathfrak{v}) = \cdots = a_{i-1}(\mathfrak{u}) = a_{i-1}(\mathfrak{v}) = 0$ und $a_i(\mathfrak{u}) \cdot a_i(\mathfrak{v}) < 0$ (d.h. die i-ten Koeffizienten sind die ersten von Null verschiedenen und haben unterschiedliches Vorzeichen.).

Seien $\mathfrak{u}$ und $\mathfrak{v}$ vereinbar. Wir definieren: $\mathfrak{u} \oplus \mathfrak{v} := |a_i(\mathfrak{u})|\mathfrak{v} + |a_i(\mathfrak{v})|\mathfrak{u}$. (Dabei verwenden wir das i von oben.) Mit anderen Worten: $\mathfrak{u} \oplus \mathfrak{v}$ ist eine Resolvente von $\mathfrak{u}$ und $\mathfrak{v}$.

Sei $\mathfrak{u}$ eine einfache Ungleichung $b\ \mathcal{R}\ 0\ \ \mathcal{R} \in \{<, \leq\}$. $\mathfrak{u}$ ist **falsifizierbar**, wenn $b \geq 0$ und $\mathcal{R} = <$ oder $b > 0$ und $\mathcal{R} = \leq$.

Folgende Regeln definieren den Erfüllbarkeitstest:

$$R: \quad ?u \iff \text{EINFACH}(?u),\ \text{FALSIFIZIERBAR}(?u) \mid \text{CLASH}$$
$$A: ?u, ?v \implies \text{VEREINBAR}(?u, ?v, ?w) \mid ?w$$

wobei VEREINBAR durch folgende Pseudo-Code Prozedur definiert sei:

```
procedure vereinbar (u, v: Gleichung; VAR w: Gleichung): BOOLEAN;
        n:= „Anzahl der versch. Variablen in den Ungl. u und v“
        j := 0;
        while a_j(u) = 0 = a_j(v) AND j < n do
        j := j + 1
        od
        if a_j(u) · a_j(v) < 0
              then w := u ⊕ v; return T
              else return F
        fi
end vereinbar.
```

Beispiel 5.5

$$\mathfrak{U} = \{\underbrace{0 \leq x_1}_{u_1}, \underbrace{0 \leq x_2}_{u_2}, \underbrace{-6 \leq x_2}_{u_3}, \underbrace{-18 \leq -3x_1 - 2x_2}_{u_4}, \underbrace{-4 \leq x_1}_{u_5}\}$$

folgende Ungleichungen sind vereinbar:	*daraus resultieren*
u_1, u_4	$w_1 : -18 \leq -2x_2$
u_4, u_5	$w_2 : -30 \leq -2x_2$
damit sind vereinbar:	*daraus resultieren*
u_2, w_1	$w_3 : -18 \leq 0$
u_2, w_2	$w_4 : -30 \leq 0$
u_3, w_1	$w_5 : -30 \leq 0$
u_3, w_2	$w_6 : -42 \leq 0$

$w_3 - w_6$ *sind zwar einfach, aber nicht falsifizierbar. Damit ist* $\mathfrak{U}$ *stabil (bzgl.* $\{R, A\}$*) und somit erfüllbar.*

Beispiel 5.6

$$\mathfrak{U} = \{\underbrace{0 \leq -x_2}_{u_1}, \underbrace{4 \leq 2x_1 + x_2}_{u_2}, \underbrace{7 \leq -3x_1 + x_2}_{u_3}\}$$

damit ist vereinbar:	*daraus resultiert*
u_2, u_3	$w_1 : 26 \leq 5x_2$
damit ist vereinbar:	*daraus resultiert*
u_1, w_1	$w_2 : 26 \leq 0$

w_2 *ist einfach* und *falsifizierbar, also ist* $\mathfrak{U}$ *nicht erfüllbar.*

Bemerkung: Jede endliche Menge von Ungleichungen läßt sich bzgl. $\{R, A\}$ stabilisieren. Beachte, daß die Anzahl der Variablen in resolvierten Ungleichungen streng monoton fällt.

Einen Beweis für Korrektheit und Vollständigkeit dieses Verfahrens findet man in [14].

Beachte, daß auch hier ein „konkreter Bereich“ definiert wird.

Kapitel 6

Resümee

Ausgehend von der Beobachtung, daß „Constraint Simplification Rules“ einen Großteil der Mechanismen bereitstellen, die die Grundlage terminologischer Wissensrepräsentationssysteme bilden, haben wir Simplification Rules auf eine klare formale Grundlage gestellt. Beim Versuch, den Konsistenztest für A-Boxen über $\mathcal{ALC}$ zu implementieren, sind wir an die Grenzen des Formalismus gestoßen. Es war nicht möglich, Disjunktionen adäquat darzustellen. Diese Schwäche haben wir durch die Erweiterung um nichdeterministische SiR beseitigt. Der daraus resultierende Formalismus ist zur Behandlung verschiedener Wissensrepräsentationssysteme geeignet, was durch etliche Beispiele demonstriert wurde. Wir haben gesehen, daß sogar die Verarbeitung von Gleichheit auf der Ebene der SiR geschehen kann. Dabei wurde eine Auswahlstrategie beschrieben, die eine effiziente Abarbeitung von SiR mit Gleichheit erlaubt.

Durch die Definition eines Abarbeitungsmechanismus für SiR haben wir eine Experimentierumgebung in Aussicht gestellt, die es erlaubt, Regeln auf einem sehr hohen Niveau (nämlich dem der SiR) zu definieren und ihre Dependenzen zu untersuchen. Weiterhin wurde eine einfache Implementierung des AM in COMMON LISP vorgestellt. Ein lohnendes Ziel für weitere Bemühungen wäre die Optimierung dieser Implementierung. Dabei erscheinen aus der Sicht terminologischer Sprachen folgende Ansätze besonders vielversprechend:

- Ersetze chronologisches Backtracking durch intelligentes [4, 15, 16, 18]. Beachte das massive Auftreten vom Nichtdeterminismen im Konsistenztest von $\mathcal{ALC}(\mathcal{Q})$. Durch die Behandlung von Gleichheit auf Regelebene haben wir hier wesentliche Vorarbeit geleistet.

- Finde schnellere Match-Algorithmen um zu einer Regel „passende“ (in unserem Sinn) Fakten zu finden (beispielsweise durch Indexierung der Fakten über Individuen oder Funktionssymbole (vgl. dazu [29])). Dabei erscheint eine Verwendung von RETE [8] oder TREAT [21] aufgrund vieler Ersetzungsregeln als wenig geeignet. Ein Nachteil beider Algorithmen ist der hohe Rechenaufwand, beim Löschen von Fakten (dort: „working memory elements“). Inwieweit „lazy matching“ [5] dem hier beschrieben Verfahren (über Teilinstantiierungen) überlegen ist, wäre zu untersuchen.

- Umsetzen der Auswahlstrategie, die in 5.3 beschrieben ist.
- Das Zulassen von „Mischregeln", das sind Ersetzungsregeln bei denen nur eine vordefinierte Teilmenge der Head-Atome (statt alle) ersetzt wird. Mit „Mischregeln" hätten wir die Regeln zur Gleichheitspropagierung (vgl. Seite 37) natürlicher formulieren können. Die Gleichheiten stehen dann in den Head-Atomen, die nicht ersetzt werden. Mischregeln sind eine vereinfachte Schreibweise und keine Erweiterung des Formalismus.[1] Beachte, daß sich eine Mischregel leicht in einen Augmentations- und einen Ersetzungsteil zerlegen läßt, indem man einen „Hilfsfakt" benutzt (vgl. dazu Kapitel 5.5.3 dort ist chdom(x,D) der Hilfsfakt).[2] Durch das direkte Verarbeiten von Mischregeln sind weniger Regelanwendungen nötigt, d.h. der Abarbeitungsmechanismus ist effizienter.
- Compilation der Simplificaton Rules in eine abstrakte Maschine. In [7] wurde so etwas für Vorwärtsregeln durchgeführt, allerdings ohne Ersetzungsregeln und ohne Backtracking.

Ein wichtiger Inferenzdienst terminologischer Systeme ist die Realisierung von Individuen. Um zu überprüfen, ob ein Individuum a in einem bestimmten Konzept liegt (bzgl. einer A-Box $\mathcal{A}$) fügt man $a : \neg C$ zu $\mathcal{A}$ hinzu und testet die neue A-Box auf Konsistenz. Es ist vorstellbar den SiR-Formalismus so zu erweitern, daß er die Implementierung eines Algorithmus mit folgender Charakteristik unterstützt: Mehrere solche Zugehörigkeitsabfragen werden quasi parallel bearbeitet und bei einer Erweiterung der A-Box $\mathcal{A}$ werden die durchgeführten Berechnungen im vollen Umfang wiederverwendet. (Man hätte also eine Art „Rete-Algorithmus" für terminologische Sprachen.)

Durch ihre Flexibilität sind SiR eine geeignete Implementationsgrundlage für Wissensrepräsentations- und Inferenzsysteme. Insbesondere lassen sich Systeme auf Basis von SiR leicht modifizieren und erweitern, einfach durch Angabe neuer Regeln und eventueller Modifikation der Anwendungsprioritäten. In [3] wird auf die große Bedeutung dieser Eigenschaft hingewiesen. Weitere Bemühungen sollten in das Experimentieren mit SiR investiert werden. Beispielsweise erscheint mir Folgendes reizvoll:

- Finde neue SiR, die den Regelablauf optimieren (so wie die Einführung der Subsumptionsregeln bei $\mathcal{ALC}$). Beachte, daß dieser Punkt an einer anderen Stelle ansetzt als die voherigen. Obiges beschreibt Optimierungen des *Abarbeitungsmechanismus*, hier sprechen wir von Optimierungen auf *Regelebene*.
- Integriere weitere „Konkrete Bereiche" (wie z.B. die in Paragraph 5.5.3 oder Paragraph 5.5.4 definerten) in $\mathcal{ALC}$.

[1] Man könnte Augmentations- und Ersetzungsregeln auch als spezielle Mischregeln sehen entweder mit trivialem (leerem) Ersetzungsteil oder mit trivialem Nichtersetzungsteil.

[2] Allgemein transformiert man eine Mischregel $\mathfrak{A}, A_{k1}, \ldots, A_{km} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}$, in der man nur $A_{k1}, \ldots, A_{km}$ ersetzen will in $\mathfrak{A}, A_{k1}, \ldots, A_{km} \Longrightarrow \mathfrak{C} \mid \mathfrak{B}, r(A_{k1}), \ldots, r(A_{km})$ und fügt die Regel $r(?x), ?x \Longleftrightarrow \mid \top$ mit höchster Priorität dem Regelsatz hinzu.

- [20] gibt einen Überblick über sog. „Constaint Satisfaction Problems". Dort wird zuerst ein allgemeiner (und ineffizienter) Lösungsalgorithmus vorgestellt. Danach werden Methoden beschrieben, um diesen Algorithmus zu verbessern. Implementiert man ein solches System mit SiR, lassen sich diese Methoden schrittweise einführen[3]. Ferner ist es durch den deklarativen Charakter der SiR möglich, Abarbeitungsheuristiken zu entwickeln und auf ihre Wirsamkeit zu testen.

Abschließend weise ich noch auf einige verwandte Arbeiten hin.

In [19] wird ein Parser für Graph Grammatiken beschrieben. Auch dort werden mehrfache „Regelanwendungen" vermieden und Heuristiken zur „Regelabarbeitung" verwendet. Allerdings gibt es dort weder Backtracking noch Ersetzungsregeln noch Gleichheiten zu beachten.

[10] verbindet „Constraint Logic Programming" und terminologische Inferenzen indem letztere über SiR definiert werden, die in der logischen Programmiersprache (PROLOG) implementiert werden. Die dort verwendetet Regeln unterscheiden sich von den hier definierten im Wesentlichen durch:

(a) Implementation in PROLOG (dadurch bekommt man Gleichheit geschenkt)

(b) Disjunktionen werden auf PROLOG-Ebene abgearbeitet (und nicht durch nichtdeterministische SiR)

In [11] werden Regeln benutzt, um Modelle für Klauselmengen zu konstruieren. Die dort verwendeten sogenannten „model extension rules" ähneln stark Augmentationregeln ohne Guard-Atome, „model restriction rules" entsprechen in etwa unsere Clash-Regeln. Es gibt jedoch kein Äquivalent zu unseren Ersetzungsregeln.

Danksagung

Die Themenstellung dieser Arbeit stammt aus der Wissenscompilationsgruppe des ARC-TEC Projekts, die von Dr. H. Boley geleitet wurde. Dieses Projekt wurde im Bereich „Intelligente Ingenieursysteme", den Prof. Dr. M. M. Richter am DFKI[4] leitet, verwirklicht. Philipp Hanschke war dort mein Ansprechpartner. Er nahm sich immer die Zeit, auftretende Fragen und Probleme mit mir auszudiskutieren, wofür ich ihm sehr danke. Desweiteren gilt mein Dank Andreas Abecker und Dennis Drollinger, die mir insbesondere beim Einarbeiten in die Materie wertvolle Hilfestellung geleistet haben; ferner Michael Bayer, mit dem ich meine (Zwischen-)Ergebnisse besprechen konnte und der mir beim Korrekturlesen behilflich war. Außerdem danke ich Prof. Dr. Michael M. Richter und Prof. Dr. Dietmar Schweigert, die den Rahmen für diese Arbeit geschaffen haben.

[3] Das wurde in Paragraph 5.5.3 am Beispiel lokaler Konsistenz demonstriert.

[4] Deutsches Forschungszentrum für künstliche Intelligenz

Literaturverzeichnis

[1] Andreas Abecker. TAXLOG: Taxonomische Wissensrepräsentation und logisches Programmieren. Projektarbeit. Universität Kaiserslautern, 1992

[2] Franz Baader, Philipp Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. Research Report RR-91-10, DFKI / Kaiserslautern, 1991

[3] Alex Boriga. Towards the Systematic Development of Description Logic Reasoners: CLASP reconstructed. in *KR'92 Principles of Knowledge Representation and Reasoning, Proceedings of the Third International Conference, Edited by Bernhard Nebel, Charles Rich, William Swartout*, 1992

[4] Maurice Bruynooghe. Intelligent Backtracking Revisited. 1991

[5] Daniel P. Miranker, David A. Brant, Bernie Lofaso, David Gadbois. On the Performance of Lazy Matching in Production Systems. AAAI-90

[6] Eugen Charniak, Christopher K. Riesbeck, Drew V. McDermott, James R. Meehan. Artificial Intelligence Programming, Hillsdale New Jersey 1987

[7] Christian Falter. Compilation von Vorwärtsregeln in einer hybriden Expertensystem-Shell. Diplomarbeit am Fachbereich Informatik der Universität Kaiserslautern, 1992

[8] Charles L. Forgy. Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. in *Artificial Intelligence*, 19, 1982

[9] Thom Frühwirth. Introducting Simplification Rules. GI Fachgruppentreffen 1.1.1, Goosen (bei Berlin), 1991

[10] Thom Frühwirth, Philipp Hanschke. Terminological Reasoning with Constraint Simplification Rules. D-93-01, DFKI / Kaiserslautern, 1993

[11] Masayuki Fujita, Miyuki Koshimura, Ryuzo Hasegawa, Hiroshi Fujita. Model Generation Theorem Provers on a Parallel Inference Machine. in *Proceedings of the International Conference on Fifth Generation Computer Systems, 1992.*

[12] Philipp Hanschke. Specifying Role Interaction in Concept Languages. in *Principles of Knowlegde Representation and Reasoning: Proceedings of the*

Third International Conference (KR92). Morgan Kaufmann, San Mateo, CA, 1992

[13] B. Hollunder. Hybrid Interfaces in KL-ONE-based Knowledge Representation Systems. Research Report RR-90-06, DFKI / Kaiserslautern, 1990

[14] H. W. Kuhn. Solvability and Consistency for Linear Equations and Inequalities. in *American Mathematical Monthly* 63: 217 - 232, 1956

[15] Vipin Kumar, Yow-Jian Lin. A Data-Dependency-Based Backtracking Sheme for Prolog. in *Journal of Logic Programming* 1988: 5: 165 -181

[16] Vishv M. Malhotra. An Algorithm for Optimal Back-Striding in Prolog. in *Proc. Seventh Int. Conf. on Logic Programming*, MIT Press 1990 pp. 147 - 158

[17] Manfred Meyer. Weak looking-ahead: Combining finite domain consistency techniques. in *Proceedings of ISMIS-93 Seventh International Symposium on METHODOLOGIES FOR INTELLIGENT SYSTEMS*, Springer Verlag, 1993.

[18] Vishv M. Malhotra, Tang Van To, Kanchana Kanchanasut. An Improved Data-Dependency-Based Backtracking Sheme for Prolog. in *Information Processing Letters* 31 (1989): 185 - 189

[19] Jakob Mauss. Ein heuristisch gesteuerter Chart-Parser für attributierte Graph-Grammatiken, DFKI Document D-92-10, 1992.

[20] Pedro Meseguer. Constraint Satisfaction Problems: An Overview. in *AI Communications* Vol.2 Nr.1: 3 - 17, 1989

[21] Daniel P. Miranker. TREAT: A Better Match Algorithm for AI Production Systems; Long Version. Technical Report AI TR87-58, July 1987

[22] Peter Norvig. Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp. Morgan Kaufmann, 1992

[23] B. Nudel. Consistent-Labeling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics. in *Artificial Intelligence*, 21: 135 - 178, 1983

[24] N. Eisinger, H. J. Ohlbach. Deduction Systems Based on Resolution. SEKI Report SR-90-12 (INF 114/463 - 90,12)

[25] Michael M. Richter. Prinzipien der künstlichen Intelligenz. B.G. Teubner Stuttgart, 1989

[26] Manfred Schmitt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48: 1-26, 1991

[27] Jörg H. Siekmann. Universal Unification. in *Lecture Notes in Computer Science, G.Goos, J.Hatmanis, 7th International Conference on Automated Deduction*, Springer 1984

[28] Guy L. Steele JR. . COMMON LISP The Language - second edition. Digital Equipment Corporation 1990

[29] Werner Stein, Michael Sintek. A Generalized Intelligent Indexing Method. in *Workshop "Sprachen für KI-Anwendungen, Konzepte - Methoden - Implementierungen" in Bad Honnef, 12/92-1*, Institute of Applied Mathematics and Computer Science, University of Münster, Herausgeber: H. Boley, U. Furbach und W.-M. Lippe, 1992.

Index

Anhang A

Der Abarbeitungsmechanismus - eine Implementierumg

A.1 Bemerkungen

Ich stelle im Folgenden eine Implementierung des in Kapitel 4 beschriebenen Abarbeitungsmechanismus in COMMON LISP vor.[1] Dabei habe ich nicht alle Optimierungen vorgenommen, die in Kapitel 5 hergeleitet wurden. Zum leichteren Verständnis bemerken wir:

- Nichtdeterminismen werden durch chronologisches Backtracking abgearbeitet. Entsteht ein Wahlpunkt (durch die Funktion **nondet**), so werden alle Änderungen (i.e. Aktivierung bzw. Deaktivierung von Regeln, Teilinstantiierung von Regeln, Löschen bzw. Entstehen von Fakten) getrailt, d.h. in einem Stack namens *BacktrackStack* abgelegt. Genauer gesagt werden die Umkehrfunktionen getrailt, d.h die Funktionen die den ursprünglichen Zustand wiederherstellen. Im Clash-Fall (d.h. beim Aufruf der Funktion **fail**) wird überprüft, ob ein Wahlpunkt existiert (einfach indem man feststellt, ob etwas getrailt wurde). Falls ein Wahlpunkt existiert, wird der *BacktrackStack* entsprechend abgearbeitet (d.h. die zuvor getrailten Umkehrfunktionen werden ausgeführt), ansonsten wird ein Programmabbruch vorbereitet (hier durch ein **catch** - **throw** - Paar).[2]

- **Auswahl der Regeln**
  Es gibt zwei Bereiche in denen Regeln stehen *RuleStack* und *PassiveRuleStack*.

[1] Ich habe mich bemüht die Programmierrichtlinien aus [6], [22] und [28] umzusetzen.

[2] Es gibt dazu noch eine Reihe anderer Möglichkeiten:

- alle Regeln deaktivieren
- die Fakten durch einen Fakt ersetzen, der Inkonsistenz ausdrückt und auf den keine Regel anwendbar ist
- die Auswahl der nächsten Regel so manipulieren, daß das Programm stoppt (vgl. hierzu das Flußdiagramm auf Seite 62

In ersterem stehen die aktiven, in letzterem die deaktiven Simplification Rules. Die Funktion **choose-Simplification-Rule** wählt die erste Regel, die in *RuleStack* steht. Die Idee dabei ist, daß die Regeln in *RuleStack* nach Prioritäten sortiert stehen, so daß die erste Regel maximale Priorität hat.

- **Auswahl der Fakten**
  Ein Fakt gilt als „passend" (zu einer Regel $\mathcal{S}$), wenn er das erste Head-Atom von $\mathcal{S}$ matcht.

- Die Implementation ist in folgende Bereiche unterteilt (jedem Bereich entspricht ein Programm-File):

  Main. Hier wird der AM definiert, so wie er im Flußdiagramm von Seite 62 zu sehen ist. Die Funktion **main!** wird aufgerufen mit einer Liste von Regeln und einer Liste von Fakten. Die Syntax dieser Listen wird im File **Initialize** erklärt. Zusätzlich kann man noch eine Reihe von Schaltern setzen, die entscheiden sollen, welche Stacks zurückgesetzt werden sollen (das kann nützlich sein, wenn man den AM mehrmals aufrufen will).

  Initialize. Hier werden die Funktionen definiert, die die Eingabe für die Abarbeitung mit **main!** aufbereiten. Die Eingabe der Regeln erfolgt als Liste fünfelementiger Vektoren (i.e. fünfelementiger Listen). Diese gliedern sich so:

  1.Stelle Head-Atome

  2.Stelle => für Augmentationsregel, <=> für Ersetzungsregel

  3.Stelle Guard-Atome

  4.Stelle Body-Atome

  5.Stelle Priorität

  Dabei wird ein Head- bzw. Guard- bzw. Body-Atom jeweils durch eine Liste repräsentiert. Einzelne Atome werden zu Listen zusammengefaßt (evtl. einelementig oder leer). Variable werden dadurch gekennzeichnet, daß sie mit „?" beginnen. Die Fakten werden als Liste von Einzelfakten eingegeben. Ein Fakt ist in der Regel eine Liste. Beachte, daß die Head-Atome der Regeln und die Fakten das gleiche Format haben müssen, da sie gegeneinander gematcht werden. Da Variable eine eigene Datenstruktur bekommen, ist es nötig die Regeln durch einen Parser zu schicken.

  Parser. Hier wird der oben erwähnte Parser definiert. Jedes Symbol, das mit „?" beginnt, wird durch eine Variable[3] ersetzt. Natürlich werden in einer Regel die gleichen Symbole durch die gleichen Variablen ersetzt.

  DataStructures. Hier werden die Datenstrukturen für Regeln, Fakten und Variable definiert.

  Stacks. Hier werden globale Speichervariable und ihre Zugriffsfunktionen definiert.

[3] i.e. eine in **DataStructures** definierte Datenstruktur

Match. Hier wird eine Funktion zum Matching definiert.

HigherFunctions. Hier werden folgende Funktionen definiert:

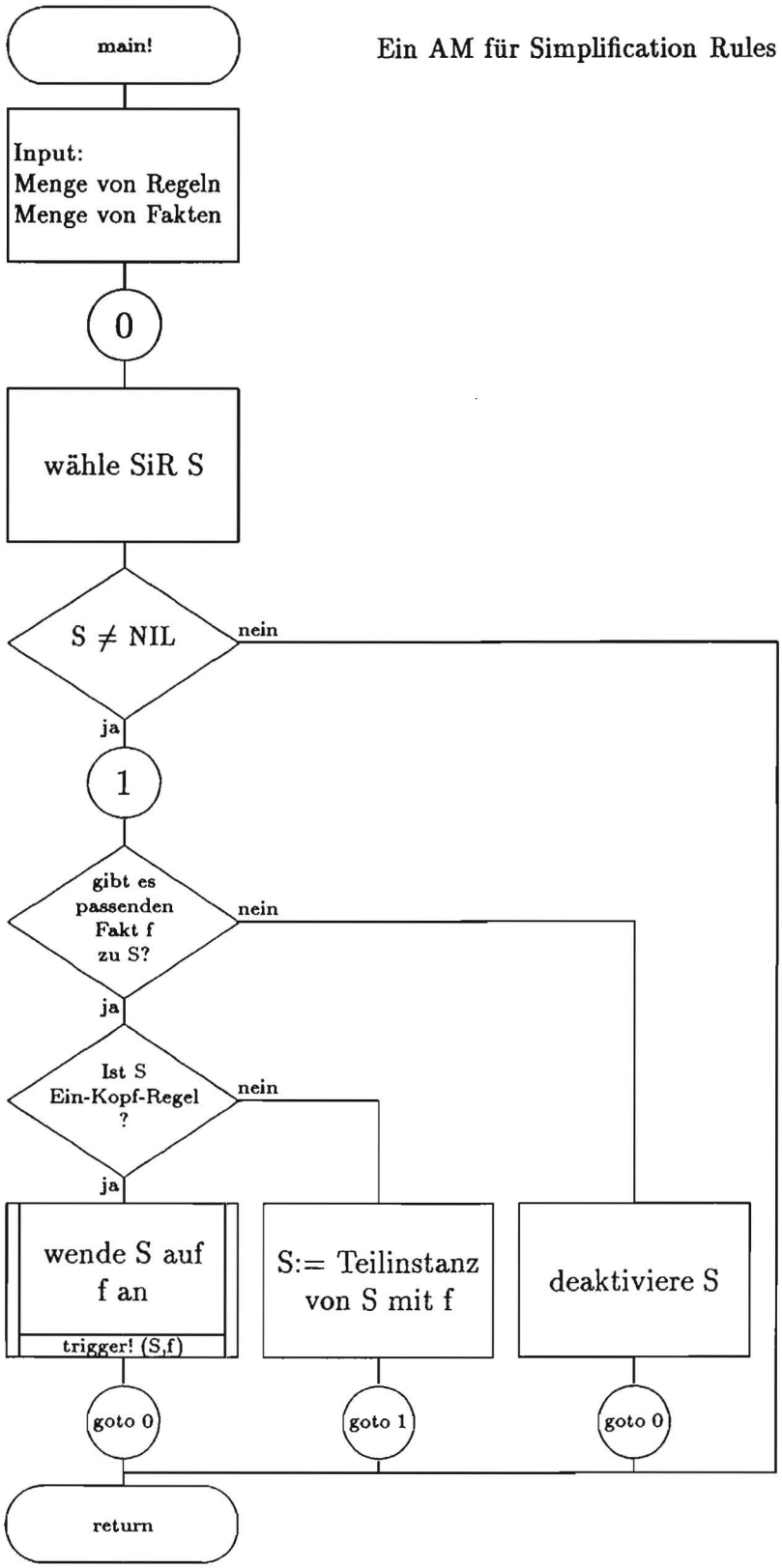
- choose-Simplification-Rule. Wählt eine aktive SiR maximaler Priorität.
- approriate-fact(Regel). Durchsucht den Stack *Facts* solange bis ein Fakt gefunden wird, der das erste Head-Atom der Regel matcht. falls kein solcher existiert, wird NIL[4] zurückgegeben.
- teilinstanz(r,f). Erzeugt eine neue Regel: die mittels f aus r teilinstantiierte Regel.
- trigger!(r,f). Wendet die Regel r auf den Fakt f so an, wie im Flußdiagramm auf Seite 63 beschrieben.

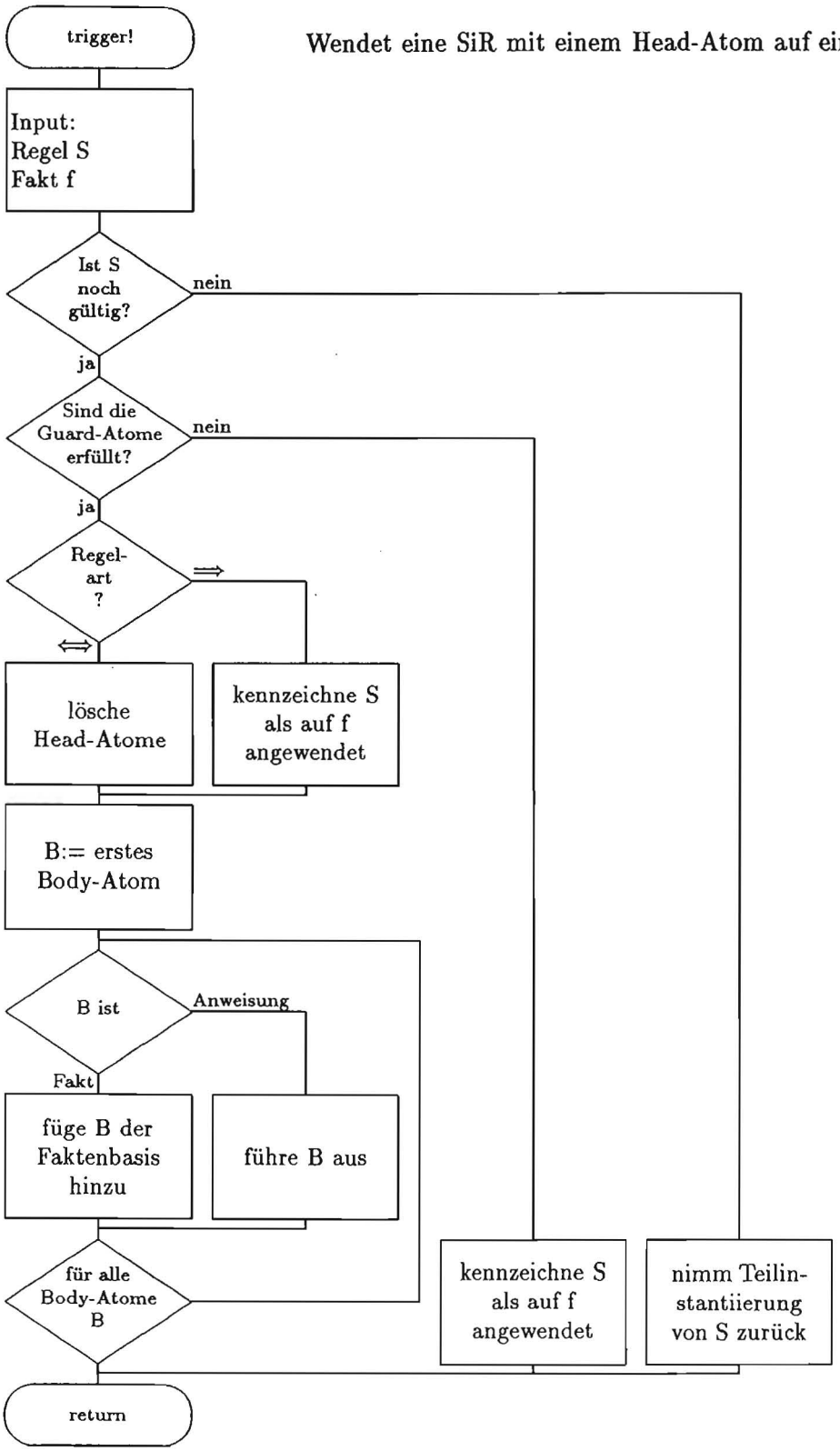
BasicFunctions. Hier werden alle Funktionen definiert, die sonst noch gebraucht werden, insbesondere zur Definition der „höheren Funktionen".

Spezielle Anwendungen werden in eigenen Files definiert. (Bsp.: terminology, declaration, experiment)

[4]das Lisp-Atom, das die leere Liste kennzeichnet

Ein AM für Simplification Rules




trigger!
Wendet eine SiR mit einem Head-Atom auf einen Fakt an.
Input:
Regel S
Fakt f
Ist S noch gültig?
nein
ja
Sind die Guard-Atome erfüllt?
nein
ja
Regel-art ?
⟹
⟺
lösche Head-Atome
kennzeichne S als auf f angewendet
B:= erstes Body-Atom
B ist
Anweisung
Fakt
füge B der Faktenbasis hinzu
führe B aus
für alle Body-Atome B
kennzeichne S als auf f angewendet
nimm Teilin-stantiierung von S zurück
return

A.2 Sourcecodes

Anhang B

Logische Programme mit SiR - ein Beispiel

Wir zeigen anhand eines Beispiels wie sich TaxLog-Fakten und -Regeln in Simplification Rules übersetzen lassen . Zum leichteren Verständnis wählen wir hier die PROLOG-Darstellung der Regeln.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  1.) the concept definitions                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     BICONIC isa GEOMETRIC and COMPOSED
                           and exists LEFT in TRUNCONE
                           and exists RIGHT in TRUNCONE
                           and REAL-=((LEFT C-B)(RIGHT C-A))
                           and REAL-=((LEFT R-B)(RIGHT R-A)).
     RING isa TRUNCONE and REAL-=(C-A C-B).
     ASCENDING isa TRUNCONE and REAL-<(R-A R-B).
     CYLINDER isa TRUNCONE and REAL-=(R-A R-B).
     TRUNCONE isa GEOMETRIC and ATOMIC and WF-TC(C-A C-B R-A R-B).
     COMPOSED isa (nota ATOMIC).
     GEOMETRIC isa (nota HUMAN).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  2.) the Taxlog rules: an example                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    WP(A) :- TRUNCONE(A), (A,0):R-A, (A,20):R-B, (A,0):C-A, (A,20):C-B.
    WP(B) :- TRUNCONE(B), RING(B), (B,20):R-A, (B,40):R-B, (B,20):C-A,
            (B,20):C-B.
    WP(C) :- TRUNCONE(C), (C,40):R-A, (C,50):R-B, (C,20):C-A, (C,30):C-B.
    WP(D) :- TRUNCONE(D), (D,50):R-A, (D,60):R-B, (D,30):C-A, (D,60):C-B.
    ASC-SEQ(X) :- WP(X), TRUNCONE(X), (ALL NEXT BOTTOM)(X), ASCENDING(X).
    ASC-SEQ(X) :- WP(X), ASC-SEQ(Y), TRUNCONE(Y), TRUNCONE(X),
                  ASCENDING(X), (X,Y):NEXT, BICONIC(Z), (Z,X):LEFT,
                  (Z,Y):RIGHT, (X,Y):UNEQUAL.
```

Wir übersetzen jetzt obige Konzeptdefinitionen:

```
BICONIC isa GEOMETRIC and COMPOSED
                      and exists LEFT in TRUNCONE
                      and exists RIGHT in TRUNCONE
                      and REAL-=((LEFT C-B)(RIGHT C-A))
                      and REAL-=((LEFT R-B)(RIGHT R-A)).
```

wird zu:

```
(conc biconic (and geometric
                   (and composed
                        (and (exists right truncone)
                             (and (exists left truncone)
                                  (exists2 left right neighbouring))))))
```

Analog übersetzen wir

```
RING isa TRUNCONE and REAL-=(C-A C-B). ; zu
(conc ring (and truncone (exists2 c-a c-b real-=)))

ASCENDING isa TRUNCONE and REAL-<(R-A R-B). ; zu
(conc ascending (and truncone (exists2 r-a r-b real-<)))

CYLINDER isa TRUNCONE and REAL-=(R-A R-B). ; zu
(conc cylinder (and truncone (exists2 r-a r-b real-=)))

TRUNCONE isa GEOMETRIC and ATOMIC and WF-TC(C-A C-B R-A R-B). ; zu
(conc truncone (and geometric
                    (and atomic (exists4 c-a c-b r-a r-b wf-tc))))

COMPOSED isa (nota ATOMIC). ;zu
(conc composed (nota atomic))
```

Beachte, daß die zugrundegelegte Terminologie in einer Hash-Tabelle abgelegt wird, auf die die Guard-Funktion ISA zugreift (vgl. dazu das Hilfsmodul *terminology.lsp* von Seite ??). *and* und *exists* werden mit den normalen $\mathcal{ALC}$-Regeln abgearbeitet. *exists2* und *exists4* mit speziellen Taxlog-Regeln, die später definiert werden.

Taxlog-Fakten und -Regeln werden nach folgendem Schema übersetzt:

$$\left.\begin{array}{l} \mathrm{P} :\!- G_1 \\ \mathrm{P} :\!- G_2 \end{array}\right\} \longrightarrow \mathrm{P} \Longleftrightarrow \mid G_1 \vee G_2.$$

Den Taxlog-Fakten

```
WP(A) :- TRUNCONE(A), (A,0):R-A, (A,20):R-B, (A,0):C-A, (A,20):C-B.
WP(B) :- TRUNCONE(B), RING(B), (B,20):R-A, (B,40):R-B, (B,20):C-A,
         (B,20):C-B.
WP(C) :- TRUNCONE(C), (C,40):R-A, (C,50):R-B, (C,20):C-A, (C,30):C-B.
WP(D) :- TRUNCONE(D), (D,50):R-A, (D,60):R-B, (D,30):C-A, (D,60):C-B.
```

entspricht die folgende Regel (so wie sie sich mit dem Abarbeitungsmechanismus von Anhang A verarbeiten läßt):

```
(((in ?x wp)) <=> '()
 ((meta-or (meta-and (in ?x truncone) (in (?x 0) r-a) (in (?x 20) r-b)
                     (in (?x 0) c-a) (in (?x 20) c-b))
           (meta-and (in ?x truncone) (in ?x ring) (in (?x 20) r-a)
                     (in (?x 40) r-b) (in (?x 20) c-a) (in (?x 20) c-b))
           (meta-and (in ?x truncone) (in (?x 40) r-a) (in (?x 50) r-b)
                     (in (?x 20) c-a) (in (?x 30) c-b))
           (meta-and (in ?x truncone) (in (?x 50) r-a) (in (?x 60) r-b)
                     (in (?x 30) c-a) (in (?x 60) c-b)))) 0)
```

Den Taxlog-Regeln:

```
ASC-SEQ(X) :- WP(X), TRUNCONE(X), (ALL NEXT BOTTOM)(X), ASCENDING(X).
ASC-SEQ(X) :- WP(X), ASC-SEQ(Y), TRUNCONE(Y), TRUNCONE(X),
              ASCENDING(X), (X,Y):NEXT, BICONIC(Z), (Z,X):LEFT,
              (Z,Y):RIGHT, (X,Y):UNEQUAL.
```

entspricht die Regel:

```
(((in ?x asc-seq)) <=> ((fresh ?y) (fresh ?z))
   (meta-or (meta-and (in ?x wp) (in ?x (every next bottom)) (in ?x ascending))
            (meta-and (in ?x wp) (in ?y asc-seq) (in ?y truncone)
                      (in ?x ascending) (in ?z biconic) (in (?x ?y) next)
                      (in (?z ?x) left) (in (?z ?y) right)
                      (in (?x ?y) real-/=))) 0)
```

Beachte, daß *meta-and* und *meta-or* hier mehrstellig sind, in nachfolgenden Regeln aber zweistellig. Diese Darstellung wählen wir zugunsten der Lesbarkeit. In der Zusammenfassung haben wir obige Regeln mit zweistelligem *meta-and* und *meta-or* so definiert, wie nachfolgende Regeln das voraussetzen. Um Beispiele aus diesem Bereich abarbeiten zu können, benötigen wir noch die folgenden Regeln (wiederum in der direkt verarbeitbaren Form):

```
(((meta-and ?a ?b)) <=> '() (?a ?b) 0)

(((meta-or ?a ?b)) <=> '() ((nondet ?a ?b)) 0)

(((in ?x (exists2 ?r1 ?r2 ?p))) <=> ((fresh ?y) (fresh ?z))
 ((in (?x ?y) ?r1) (in (?x ?z) ?r2) (in (?y ?z) ?p)) 0)

(((in ?x (exists4 ?r1 ?r2 ?r3 ?r4 ?p))) <=>
 ((fresh ?y1) (fresh ?y2) (fresh ?y3) (fresh ?y4))
 ((in (?x ?y1) ?r1) (in (?x ?y2) ?r2) (in (?x ?y3) ?r3) (in (?x ?y4) ?r4)
  (in (?y1 ?y2 ?y3 ?y4) ?p)) 0)

(((in (?x ?y) neighbouring)) <=>
 ((fresh ?r1) (fresh ?r2) (fresh ?c1) (fresh ?c2))
 ((in (?x ?c1) c-b) (in (?y ?c2) c-a) (in (?c1 ?c2) real-=)
  (in (?x ?r1) r-b) (in (?y ?r2) r-a) (in (?r1 ?r2) real-=)) 0)

(((in (?c1 ?c2 ?r1 ?r2) wf-tc)) <=> '()
 ((in ?r1 real->=0) (in ?r2 real->=0) (in (?c1 ?c2) real-<=)
  (meta-or (meta-and (in (?c1 ?c2) real-=) (in (?r1 ?r2) real-/=))
      (meta-and (in (?c1 ?c2) real-/=)
           (meta-or (in ?r1 real->0) (in ?r2 real->0))))) 0)

(((in (?x ?y1) ?f) (in (?x ?y2) ?f)) => ((attribute? ?f)) ((real-= ?y1 ?y2)) 0)
```

Zusammengefaßt erhalten wir:

```
#|*****************************************************************************
*                                                                             *
*                        File: taxlogregeln                                   *
*                                                                             *
*****************************************************************************|#

; Diese Regeln braucht man, um Beispiele aus diesem Bereich abzuarbeiten:

(setf allgemeineregeln
'(
  (((meta-and ?a ?b)) <=> '() (?a ?b) 0)
  (((meta-or ?a ?b)) <=> '() ((nondet ?a ?b)) 0)
  (((in ?x (exists2 ?r1 ?r2 ?p))) <=> ((fresh ?y) (fresh ?z))
   ((in (?x ?y) ?r1) (in (?x ?z) ?r2) (in (?y ?z) ?p)) 0)
  (((in ?x (exists4 ?r1 ?r2 ?r3 ?r4 ?p))) <=>
   ((fresh ?y1) (fresh ?y2) (fresh ?y3) (fresh ?y4))
   ((in (?x ?y1) ?r1) (in (?x ?y2) ?r2) (in (?x ?y3) ?r3) (in (?x ?y4) ?r4)
    (in (?y1 ?y2 ?y3 ?y4) ?p)) 0)
  (((in (?x ?y) neighbouring)) <=>
   ((fresh ?r1) (fresh ?r2) (fresh ?c1) (fresh ?c2))
   ((in (?x ?c1) c-b) (in (?y ?c2) c-a) (in (?c1 ?c2) real-=)
    (in (?x ?r1) r-b) (in (?y ?r2) r-a) (in (?r1 ?r2) real-=)) 0)
  (((in (?c1 ?c2 ?r1 ?r2) wf-tc)) <=> '()
   ((in ?r1 real->=0) (in ?r2 real->=0) (in (?c1 ?c2) real-<=)
    (meta-or (meta-and (in (?c1 ?c2) real-=) (in (?r1 ?r2) real-/=))
        (meta-and (in (?c1 ?c2) real-/=)
             (meta-or (in ?r1 real->0) (in ?r2 real->0))))) 0)
  (((in (?x ?y1) ?f) (in (?x ?y2) ?f)) => ((attribute? ?f)) ((real-= ?y1 ?y2)) 0)
)) ; setf allgemeineregeln

; Diese Regeln entsprechen den PROLOG-Regeln des Beispiels:

(setf beispielregeln
'(
  (((in ?x asc-seq)) <=> ((fresh ?y) (fresh ?z))
   (meta-or (meta-and (in ?x wp)
            (meta-and (in ?x (every next bottom)) (in ?x ascending)))
       (meta-and (in ?x wp)
            (meta-and (in ?y asc-seq)
                 (meta-and (in ?y truncone)
                      (meta-and (in ?x ascending)
                           (meta-and (in ?z biconic)
                                (meta-and (in (?x ?y) next)
                                     (meta-and (in (?z ?x) left)
                                          (meta-and (in (?z ?y) right)
                                               (in (?x ?y) real-/=)))))))))) 0)
  (((in ?x wp)) <=> '()
   ((meta-or (meta-and (in ?x truncone)
             (meta-and (in (?x 0) r-a)
```

```
                (meta-and (in (?x 20) r-b)
                     (meta-and (in (?x 0) c-a) (in (?x 20) c-b)))))
        (meta-or (meta-and (in ?x truncone)
                 (meta-and (in ?x ring)
                      (meta-and (in (?x 20) r-a)
                           (meta-and (in (?x 40) r-b)
                                (meta-and (in (?x 20) c-a) (in (?x 20) c-b))))))
            (meta-or (meta-and (in ?x truncone)
                     (meta-and (in (?x 40) r-a)
                          (meta-and (in (?x 50) r-b)
                               (meta-and (in (?x 20) c-a) (in (?x 30) c-b)))))
                (meta-and (in ?x truncone)
                     (meta-and (in (?x 50) r-a)
                          (meta-and (in (?x 60) r-b)
                               (meta-and (in (?x 30) c-a) (in (?x 60) c-b)))))
            )))
    ) 0)
)) ; beispielregeln

(setf taxlog-to-concdomain-regeln
'(
  (((in (?x ?y) real-=)) <=> '() ((real-= ?x ?y)) 0)
  (((in (?x ?y) real-/=)) <=> '() ((real-/= ?x ?y)) 0)
  (((in (?x ?y) real-<=)) <=> '() ((real-<= ?x ?y)) 0)
  (((in (?x ?y) real-<)) <=> '() ((real-< ?x ?y)) 0)

;; Diese Regeln entscheiden ob Fakten auswertbares Wissen repraesentieren.
;; Beachte, dass die numerischen Relationen in den Guard-Atomen
;; LISP-Funktionen sind!

(setf concdomain-decide-regeln
'(
  (((in ?x real->0)) <=> ((numberp ?x) (<= ?x 0)) ((fail)) 0)
  (((in ?x real->0)) <=> ((numberp ?x) (> ?x 0)) '() 0)
  (((in ?x real->=0)) <=> ((numberp ?x) (< ?x 0)) ((fail)) 0)
  (((in ?x real->=0)) <=> ((numberp ?x) (>= ?x 0)) '() 0)
  (((real-= ?x ?y)) <=> ((numberp ?x) (numberp ?y) (/= ?x ?y)) ((fail)) 0)
  (((real-= ?x ?y)) <=> ((numberp ?x) (numberp ?y) (= ?x ?y)) '() 0)
  (((not ((real-= ?x ?y)))) <=> '() ((real-/= ?x ?y)) 0)
  (((real-/= ?x ?y)) <=> ((numberp ?x) (numberp ?y) (= ?x ?y)) ((fail)) 0)
  (((real-/= ?x ?y)) <=> ((numberp ?x) (numberp ?y) (/= ?x ?y)) '() 0)
  (((real-<= ?x ?y)) <=> ((numberp ?x) (numberp ?y) (> ?x ?y)) ((fail)) 0)
  (((real-<= ?x ?y)) <=> ((numberp ?x) (numberp ?y) (<= ?x ?y)) '() 0)
  (((real-< ?x ?y)) <=> ((numberp ?x) (numberp ?y) (>= ?x ?y)) ((fail)) 0)
  (((real-< ?x ?y)) <=> ((numberp ?x) (numberp ?y) (< ?x ?y)) '() 0)
)) ; setf concdomain-decide-regeln
```

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or per anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster: WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler: Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi: Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann: Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt: On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo: A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder: How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist: Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi: An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit: GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz: Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert: Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta: A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof [illegible], Volker Schölles, Markus A. Thies, Wo[illegible] Wahlster: PLUS - Plan-based User Supp[illegible] Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: [illegible] Oriented Concurrent Constraint Prog[illegible] Oz
17 pages

RR-93-17
Rolf Backofen: Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminologica[illegible] Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard [illegible]: Embedding Defaults into Terminologies Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller: Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy: Comparative Study of Connectionist Simu[illegible]
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel: Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Ap[illegible] Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger: Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agen[illegible] Worlds viaTerminological Logics
35 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.):
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf:
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

DFKI Technical Memos

TM-91-15
Stefan Busemann: Prototypical Concept Formation An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein: Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

DFKI Documents

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.):
Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch:
Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth: Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz Saarbrücken 29.-30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch: Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden, Christoph Jung: Natural Language Software Registry (Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-15
Robert Laux: Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus

Bernhard Herbig

D-93-20
Document