Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-93-21

Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken

Dennis Drollinger

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

Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken

Dennis Drollinger

DFKI-D-93-21

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

Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken

Dennis Drollinger

Zusammenfassung

In der KI beschäftigt man sich zunehmend mit Terminologischen Logiken. Diese gehen auf einen logikbasierten Formalismus für Semantische Netze zurück, der von R. Brachman unter dem Name KL-ONE eingeführt wurde. In KL-ONE-basierten Systemen wird zwischen terminologischem und assertionalem Wissen unterschieden, indem für jede Komponente ein eigener Formalismus zur Verfügung steht. In der sogenannten TBOX wird das konzeptuelle Wissen beschrieben, während in der ABOX Instanzen von Konzepten gebildet werden können, die über Relationen miteinander verknüpft werden.

In den letzten Jahren lag der Schwerpunkt der Forschung im Bereich Terminologischer Logiken auf der Entwicklung vollständiger und effizienter Algorithmen für die Klassifikation, die den zentralen Inferenzdienst des TBOX-Reasoning darstellt. Heute kann man sagen, daß

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Zur Entstehung dieser Arbeit 5

1.2 Aufgabenstellung 5

1.3 Gliederung der Ausarbeitung 5

2 Einführung in Terminologische Logiken **7**

2.1 Grundgedanke Terminologischer Logiken 7

2.2 Die Konzeptsprache 7

2.2.1 Die Syntax der **TBOX** 8

2.2.2 Die Semantik der **TBOX** 8

2.2.3 Terminologische Inferenzdienste 8

2.3 Die assertionale Sprache 9

2.3.1 Die Syntax der **ABOX** 10

2.3.2 Die Semantik der **ABOX** 10

2.3.3 Assertionale Inferenzdienste 10

2.4 Terminologische Inferenzsysteme 11

2.4.1 Die Komponenten und Speicherglieder 12

2.4.2 Der Inferenzprozeß 12

3 Motivation für intelligentes Backtracking **15**

3.1 Theoretische und praktische Arbeiten auf dem Gebiet der Terminologischen Logiken 15

3.2 Backtrackingstrategien 17

4 Von der Semantik zum Entscheidungsverfahren **20**

4.1 Operationale Semantik 20

4.2 Beispiel: Die operationale Semantik von ALC 22

4.3 Regelinterpreter 23

5 Intelligentes Backtracking **26**

5.1 Problem der Konfliktursache 26

5.2 Problem der Validität der Constraints 27

5.3 Problem der Terminierung 29

5.4 Problem der Korrektheit 30

6 Der Algorithmus **31**

6.1 Darstellung 31

6.1.1 Der Propagierungsmodus 31

6.1.2 Der Backtrackingmodus 32

6.2 Formalisierung 32

6.4 Beispiel 42

6.5 Die inkrementelle Schnittstelle 45

6.6 Fakten und Annahmen 45

7 Mögliche Erweiterungen des Ansatzes **47**

7.1 Gleichheit von Termen 47

Abbildungsverzeichnis

1 Subsumptionshierarchie 9
2 Komponenten eines einfachen Constraint-Propagierungssystems 11
3 Abstrakte Darstellung 33
4 Beispiel einer Begründungsfunktion 36

1 Einleitung

Eine der wichtigsten Aufgaben der KI ist es, Verfahren zur Realisierung komplexer Operationen auf großen Datenmengen anzubieten.
Eine Möglichkeit dafür ist durch „Heuristische Suche“ den Lösungsraum so einzuschränken, daß in vertretbarer Zeit eine Lösung gefunden wird. Der Nachteil dabei ist, daß die Lösung nicht optimal sein muß oder möglicherweise, aufgrund einer zu großen Beschneidung des Suchraums, überhaupt nicht gefunden wird, obwohl sie existiert.
Bestimmte Probleme lassen jedoch auch eine andere Möglichkeit zu: Im Verlauf der Suche wird zusätzliches Wissen aufgebaut und dazu benutzt, den Lösungsraum so einzuschränken, daß, falls vorhanden, die Lösung in vertretbarer Zeit gefunden wird und auch optimal ist.

In dieser Arbeit wird ein Verfahren entwickelt, das ein Modell für eine Menge von Constraints sucht, wobei der Lösungsraum durch Wissen über die Abhängigkeiten zwischen den Constraints stark eingeschränkt wird.

1.1 Zur Entstehung dieser Arbeit

Ein Teil der „Compilations-Gruppe“ des ARC-TEC-Projekts beschäftigt sich seit Herbst 1990 mit der Verarbeitung von Taxonomischem Wissen. Dabei entstanden auch drei Implementierungen, die die praktische Relevanz der erarbeiteten Ansätze demonstrieren sollten.
Bestimmte Inferenzdienste genügten durchaus den Anforderungen der Benutzer, andere verursachten nicht vertretbare Laufzeiten.
Aus diesem Grund mußte die Implementierung optimiert werden.
Die gesammelten Erfahrungen legten die Vermutung nahe, daß die Laufzeit für alle Inferenzdienste, durch eine geeignete Behandlung der Disjunktionen, auf ein vertretbares Maß reduziert werden könnte.

Ziel der Diplomarbeit ist es, dies zu zeigen.

1.2 Aufgabenstellung

Der Schwerpunkt dieser Arbeit war die Entwicklung eines Konzepts zur Verwaltung von Abhängigkeiten in Inferenzsystemen unter besonderer Berücksichtigung Terminologischer Inferenzsysteme. Um die Zweckmäßigkeit des Ansatzes zu zeigen, sollte eine prototypische Implementierung erstellt werden.

1.3 Gliederung der Ausarbeitung

In Kapitel 2 werde ich eine kurze Einführung in Terminologische Logiken geben. Die Darstellung ist dabei sehr knapp, so daß manche „Feinheiten“ dem Leser verborgen bleiben müssen.
Am Ende des Kapitels befindet sich die Darstellung eines Constraint-Propagierungs-Systems. Ein ähnliches Constraint-Propagierungs-System fand in der letzten Implementierung Verwendung, so daß dieses den Implementierungsstand vor dieser Arbeit darstellt.

In Kapitel 3 werde ich diese Arbeit motivieren, indem ich zeige, daß eine intelligente Behandlung der Disjunktionen notwendig und ausreichend ist, um vertretbare Laufzeiten zu erhalten.

In Kapitel 4 werde ich von Terminologischen Logiken abstrahieren, indem ich anstelle einer

Terminologischen Logik einen Regelformalismus betrachte, der jedoch dazu geeignet ist, die operationale Semantik einer Terminologischen Logik adäquat zu repräsentieren. Am Ende des Kapitels steht dann eine Formalisierung des in Kapitel 2 vorgestellten Constraint-Propagierungs-Systems, auf dem die folgenden Algorithmen aufbauen werden.

Kapitel 5 beschäftigt sich mit den Problemen und deren Lösungen, die bei einer intelligenten Behandlung der Disjunktionen auftreten. Diese Lösungen repräsentieren dabei meinen Ansatz zur Realisierung einer intelligenten Backtrackingstrategie.

In Kapitel 6 wird der vollständige Algorithmus zuerst informell und dann formal vorgestellt. Die Kapitel 5 und 6 bilden den Schwerpunkt dieser Ausarbeitung.

Kapitel 7 befaßt sich mit möglichen Erweiterungen des Ansatzes.

In Kapitel 8 wird die Verwandschaft meiner Arbeit zu Arbeiten aus den Bereichen Truth-Maintenance-Systeme und Logische Programmierung untersucht.

In Kapitel 9 wird diese Arbeit zusammengefaßt und ein Ausblick auf weitere Arbeiten in dieser Richtung gegeben.

2 Einführung in Terminologische Logiken

In diesem Abschnitt werde ich einen kurzen Überblick über Terminologische Logiken geben. Dabei werde ich mich hier auf das Notwendigste beschränken. Der Leser, der an Darstellungen interessiert ist, die ausführlicher[1] sind, die Erweiterungen[2] Terminologischer Logiken beinhalten oder die die Integration Terminologischer Logiken in andere Systeme beschreiben[3], sei auf die angegebene Literatur verwiesen.

2.1 Grundgedanke Terminologischer Logiken

Terminologische Logiken gehen auf einen logikbasierten Formalismus für semantische Netze zurück, der von R. Brachman unter dem Namen **KL-ONE** eingeführt wurde. Ausgehend von semantischen Netzen, deren Semantik prozedural erklärt war, wollte er einen Wissensrepräsentationsformalismus entwickeln, der semantisch fundiert ist.

Im Laufe der Zeit entstand eine Vielzahl von Systemen, die die folgenden Gemeinsamkeiten aufweisen:

- Terminologische Logiken sind hybride Systeme, in denen zwischen terminologischem und assertionalem Wissem unterschieden wird, indem für jede Komponente ein eigener Formalismus zur Verfügung steht. In der **Terminologischen Box**, der sogenannten **TBOX**, wird das konzeptuelle Wissen beschrieben, während in der **assertionalen Box**, der sogenannten **ABOX**, Instanzen von Konzepten gebildet werden, die über Relationen, sogenannte Rollen, miteinander verknüpft werden.[4]
- Der **TBOX** -Formalismus gestattet die Strukturierung des Diskursbereichs, indem aus einfachen Begriffen komplexere gebildet werden können. Die dabei entstehende Begriffshierarchie (Subsumptionshierarchie genannt) kann dazu benutzt werden, die Inferenzdienste für den **ABOX**-Formalismus und **TBOX**-Formalismus zu unterstützen.
- Es existieren effiziente Inferenzdienste, die es gestatten, implizites Wissen explizit zu machen.
- Es läßt sich eine Tarski-Style-Semantik angeben.
- Die Semantik basiert auf der *Open World Assumption*

Die entwickelten Systeme unterscheiden sich erheblich hinsichtlich der Ausdrucksstärke. Insbesondere führte die Ausdrucksmächtigkeit des ursprünglichen **KL-ONE** zur Unentscheidbarkeit mancher Inferenzprobleme. Im folgenden wird die Terminologische Sprache ALC definiert, die in vielen terminologischen Sprachen enthalten ist.

2.2 Die Konzeptsprache

Die Konzeptsprache stellt Konstrukte zur Verfügung, mit denen die Terminologie des Diskursbereichs definiert werden kann.

[1] zum Beispiel [Brachman and Schmolze, 1985], [Nebel, 1989], [von Luck, 1991] und [Baader *et al.*, 1992a]

[2] zum Beispiel [Schmiedel, 1990], [Baader and Hanschke, 1991], [Baader and Hollunder, 1992] und [Hanschke,

2.2.1 Die Syntax der TBOX

Definition 2.1 (Konzeptterme und Terminologie von ALC) *Seien* CONC *und* ROLE *zwei disjunkte Mengen von Konzept- und Rollennamen. Die Menge der* **Konzeptterme** *von ALC ist induktiv definiert durch:*

1. *Jeder Konzeptname ist ein Konzeptterm.*
2. *Seien C und D Konzeptterme, und sei R eine Rollenname, dann sind die folgenden Ausdrücke ebenfalls Konzeptterme:*

$C \sqcap D$	*Konjunktion*
$C \sqcup D$	*Disjunktion*
$\neg C$	*Komplement*
$\exists R.C$	*Existenzrestriktion*
$\forall R.C$	*Werterestriktion*

Sei A ein Konzeptname und sei D ein Konzeptterm. Dann ist $A=D$ ein **terminologisches Axiom**. *Eine* **TBOX** $\mathcal{T}$ *ist eine endliche Menge terminologischer Axiome mit den zusätzlichen Einschränkungen, daß erstens kein Konzeptname mehrfach auf der linken Seite eines Axioms erscheint, und zweitens, daß in $\mathcal{T}$ keine zyklischen Definitionen enthalten sind.*

Eine **TBOX** besteht also aus zwei unterschiedlichen Arten von Konzeptnamen. Die *definierten Konzepte* erscheinen auf der linken Seite eines Axioms. Die anderen werden *primitive Konzepte* genannt.

2.2.2 Die Semantik der TBOX

Definition 2.2 (Interpretationen und Modelle) *Eine Interpretation I besteht aus einer Menge $dom(I)$ und einer Interpretationsfunktion. Die Interpretationsfunktion assoziiert mit jedem Konzeptnamen A eine Teilmenge A^I von $dom(I)$, und mit jedem Rollennamen R eine binäre Relation R^I auf $dom(I)$.*

Die Interpretationsfunktion — welche eine Interpretation für atomare Terme darstellt — kann auf Konzeptterme wie folgt erweitert werden:

Seien C und D Konzeptterme, und sei R ein Rollenname. Angenommen C^I und D^I sind bereits definiert. Dann ist

$$
\begin{aligned}
(C \sqcap D)^I &= C^I \cap D^I, \\
(C \sqcup D)^I &= C^I \cup D^I, \\
(\neg C)^I &= dom(I) \setminus C^I, \\
(\forall R.C)^I &= \{x \in dom(I) \mid \text{für alle } y \text{ mit } (x,y) \in R^I \text{ gilt: } y \in C^I\} \text{ und} \\
(\exists R.C)^I &= \{x \in dom(I) \mid \text{es gibt ein } y, \text{ so daß } (x,y) \in R^I \text{ und } y \in C^I\}.
\end{aligned}
$$

Eine Interpretation I ist ein Modell der **TBOX** $\mathcal{T}$ gdw *für alle Axiome $A=D$ in $\mathcal{T}$ gilt: $A^I = D^I$.*

2.2.3 Terminologische Inferenzdienste

Den wesentlichen Inferenzdienst des **TBOX**-Reasoning stellt die Subsumption dar, das heißt die Berechnung der Ober-Untermengenbeziehungen zwischen den Konzepttermen.

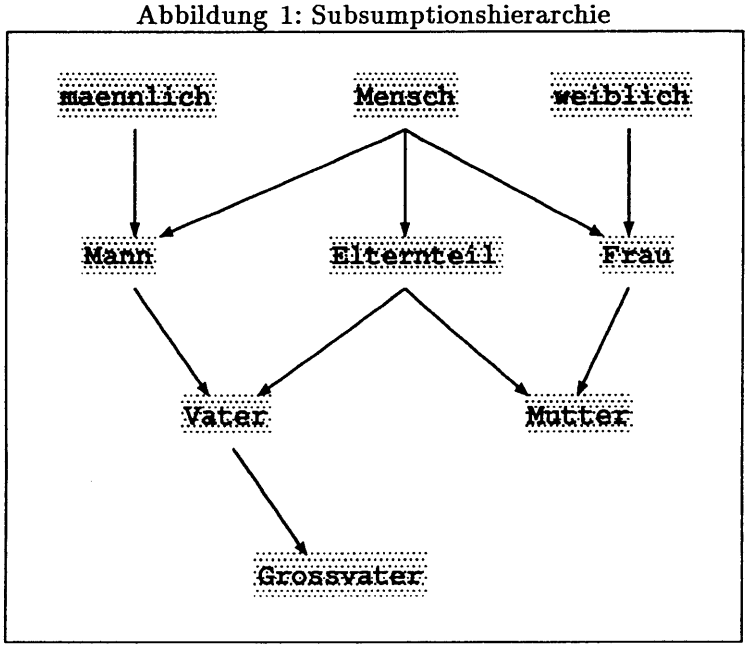
Definition 2.3 (Subsumption) *Seien C und D Konzeptterme, und $\mathcal{T}$ eine* **TBOX**. *C* **subsumiert** *D bezüglich $\mathcal{T}$* **gdw** *für alle Modelle I von $\mathcal{T}$ $D^I \subseteq C^I$ gilt. ($Notation: D \sqsubseteq_{\mathcal{T}} C$)*
Die Klassifikation ist die Berechnung der Subsumptionsbeziehung aller Konzeptterme.

Beispiel 1: Seien „Mensch" und „maennlich" primitive Konzepte und „Kind" ein Rollenname. Dann ist

$$\begin{aligned} weiblich &= \neg maennlich \\ Mann &= Mensch \sqcap maennlich \\ Frau &= Mensch \sqcap weiblich \\ Vater &= Mann \sqcap (\exists Kind.Mensch) \\ Mutter &= Frau \sqcap (\exists Kind.Mensch) \\ Elternteil &= Vater \sqcup Mutter \\ Grossvater &= Mann \sqcap (\exists Kind.Elternteil) \end{aligned}$$

eine **TBOX** aus der Domäne der Verwandschaftsbeziehungen mit der folgenden graphischen Darstellung der Subsumptionshierarchie:

Abbildung 1: Subsumptionshierarchie



2.3 Die assertionale Sprache

Während in der **TBOX** das konzeptuelle und taxonomische Wissen einer bestimmten Problemdomäne beschrieben wird, werden in der **ABOX** die Konzepte und Rollen durch konkrete Individuen instanziiert. Die auftretenden Individuen und ihre Beziehungen untereinander, sowie zu den in der **TBOX** definierten Begriffen, beschreiben einen Zustand dieser Problemdomäne.

2.3.1 Die Syntax der ABOX

Definition 2.4 (assertionale Axiome und ABOXen für ALC) *Sei* OBJ *eine Menge von Objektnamen. Die Menge aller* **assertionalen Axiome** *ist wie folgt definiert. Sei C ein Konzeptterm, R ein Rollenname und a,b Elemente von* OBJ. *Dann sind*

$$a : C \text{ und } (a, b) : R$$

assertionale Axiome. Eine **ABOX** *ist eine endliche Menge assertionaler Axiome.*

2.3.2 Die Semantik der ABOX

Definition 2.5 (Interpretationen und Modelle) *Eine Interpretation für die assertionale Sprache ist einfach eine Interpretation für ALC, die zusätzlich jedem Objekt $a^I \in dom(I)$ einen Objektnamen zuordnet. Eine Interpretation* **erfüllt** *ein assertionales Axiom*

$$\begin{aligned} a : C \quad &\text{gdw} \quad a^I \in C^I \text{und} \\ (a, b) : R \quad &\text{gdw} \quad (a^I, b^I) \in R^I. \end{aligned}$$

Eine Interpretation ist ein **Modell** *einer* **ABOX** $\mathcal{A}$ **gdw** *alle assertionalen Axiomen aus ihr folgen ($\models_I \mathcal{A}$). Sie ist ein Modell einer* **ABOX** $\mathcal{A}$ *zusammen mit einer* **TBOX** $\mathcal{T}$ **gdw** *sie ein Modell für $\mathcal{T}$ und $\mathcal{A}$ ist.*

Beispiel 2: Sei $\mathcal{T}$ die **TBOX** aus Beispiel 1. Dann lassen sich bezüglich $\mathcal{T}$ beispielsweise folgende assertionalen Axiome formulieren:

$$\begin{aligned} Iokaste &: Frau \\ Oedipus &: Mensch \\ (Iokaste, Oedipus) &: Kind \end{aligned}$$

2.3.3 Assertionale Inferenzdienste

Definition 2.6 *Eine* **ABOX** $\mathcal{A}$ *heißt* **(semantisch) konsistent** *bezüglich einer* **TBOX** $\mathcal{T}$ *falls für sie ein Modell existiert. Eine* **ABOX** $\mathcal{A}$ *und eine* **TBOX** $\mathcal{T}$ *implizieren ein assertionales Axiom α* **gdw** *alle Modelle für $\mathcal{A}$ und $\mathcal{T}$ α erfüllen ($\mathcal{A} \models_{\mathcal{T}} \alpha$).*

Das **ABOX**-Reasoning stellt folgende Inferenzdienste zur Verfügung:

- **Konsistenzproblem:** Ist eine **ABOX** $\mathcal{A}$ bezüglich einer **TBOX** $\mathcal{T}$ konsistent?
- **Instanzenproblem:** Implizieren eine **ABOX** $\mathcal{A}$ und eine **TBOX** $\mathcal{T}$ ein assertionales Axiom der Form a : C ?
- **Realisierungsproblem:** Sei a ein Objektname. Die Menge der speziellsten Konzepte, in denen a enthalten ist, ist definiert als

  $$\mathcal{MSC}_{\mathcal{A},\mathcal{T}}(a) := \{C \in CONC | \mathcal{A} \models_{\mathcal{T}} a : C \text{ und } \nexists D \in CONC \text{ mit } \mathcal{A} \models_{\mathcal{T}} a : D \text{ und } \mathcal{T} \models D \sqsubset C\}$$

  Die Berechnung der Menge $\mathcal{MSC}$ für ein α heißt Realisierung für α.

- **Retrievalproblem:** Sei C ein Konzeptterm und $\mathcal{A}$ eine **ABOX**. Das Retrievalproblem ist definiert als Berechnung der Menge aller Instanzen von C, die in $\mathcal{A}$ auftreten.

Beispiel 3: Sei $\mathcal{T}$ die **TBOX** aus Beispiel 1 und enthalte die **ABOX** $\mathcal{A}$ die assertionalen Axiome aus Beispiel 2, dann gilt für das Realisierungsproblem:

$$\mathcal{MSC}_{\mathcal{A},\mathcal{T}}(Iokaste) = \{Mutter\}.$$

Satz 1 *Alle Inferenzdienste des* **TBOX***-Reasoning und des* **ABOX***-Reasoning lassen sich auf das Konsistenzproblem für* **ABOX***en reduzieren.*

Beweis in [Hollunder, 1990]

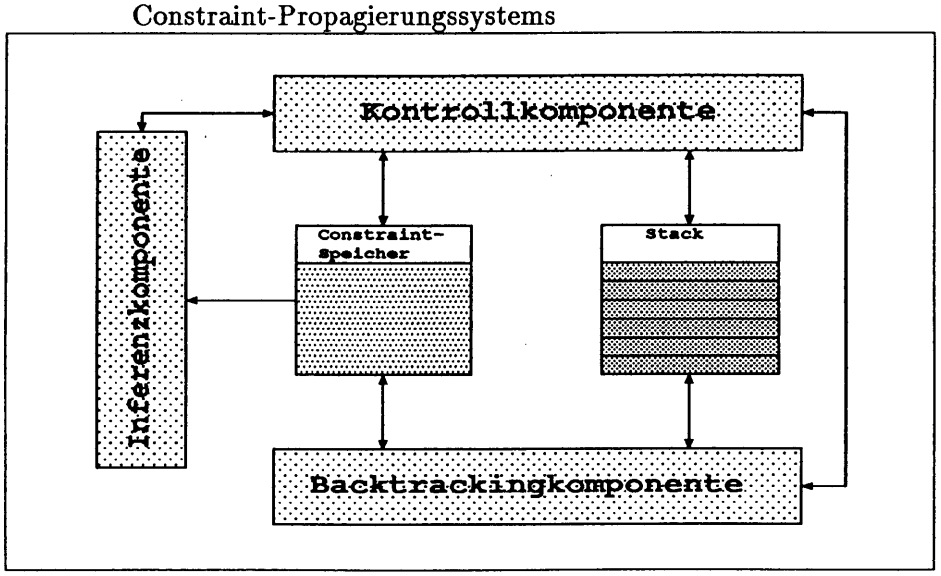
Gesucht ist also ein Entscheidungsverfahren, das zu einer gegebenen **TBOX** $\mathcal{T}$ und einer gegebenen **ABOX** $\mathcal{A}$ feststellt, ob $\mathcal{A}$ bezüglich $\mathcal{T}$ konsistent ist.
So ein Verfahren wird im nächsten Abschnitt vorgestellt und in Kaptiel 4 formalisiert.

2.4 Terminologische Inferenzsysteme

Eine Möglichkeit die Inferenzdienste zu realisieren ist, diese auf den Konsistenztest für **ABOX**en zurückzuführen. Bei diesem Konsistenztest werden assertionale Axiome als Constraints aufgefaßt, die über gewisse Inferenzregeln propagiert werden. Die eventuell vorhandenen Disjunktionen führen dazu, daß die Propagierung nicht deterministisch verläuft. Führt dieses für eine **ABOX** $\mathcal{A}$ stets zu einem offensichtlichen Widerspruch, so ist $\mathcal{A}$ inkonsistent ansonsten konsistent.

Der Inferenzprozess eines Constraint-Propagierungssystems soll nun näher untersucht werden, da er für das Verständnis der nachfolgenden Probleme wichtig ist.

Abbildung 2: Komponenten eines einfachen Constraint-Propagierungssystems



2.4.1 Die Komponenten und Speicherglieder

- Die **Inferenzkomponente** leitet aus der Menge der Constraints des Constraint-Speichers und dem „obersten" Element (Topelement) des Stacks weitere Constraints ab.
- Die **Kontrollkomponente** aktiviert die Inferenzkomponente und die Backtrackinkomponente und transportiert Daten von den Speichergliedern zu der Inferenzkomponente und umgekehrt.
- Die **Backtrackingkomponente** verwaltet die „Wahlpunkte" und restauriert frühere Zustände. Wahlpunkte sind notwendig, da eventuell vorhandene Disjunktionen Alternativen bei der Propagierung induzieren.
- Der **Constraint-Speicher** enthält die Constraints, die bereits propagiert wurden.
- Der **Stack** enthält die Constraints, die noch zu propagieren sind.

2.4.2 Der Inferenzprozeß

Initialisierung

Der Constraint-Speicher ist zu Beginn leer. Auf dem *Propagierungs*-Stack stehen alle assertionalen Axiome der **ABOX**.

Constraint-Propagierung

- Ist der *Propagierungs*-Stack nicht leer, dann wird das Topelement, im folgenden „aktueller Constraint" genannt, vom *Propagierungs*-Stack genommen.
  - Ist der aktuelle Constraint eine Disjunktion, dann wird das erste Disjunktionsglied auf den *Propagierungs*-Stack geschrieben und die restlichen Disjunktionsglieder durch die Backtrackingkomponente verwaltet. Das heißt, es wird ein sogenannter Wahlpunkt eingeführt, der die restlichen Disjunktionsglieder als Alternative für das erste Disjunktionsglied bereithält. Danach beginnt der Constraint-Propagierungsprozeß von vorn.
  - Ansonsten wird der aktuelle Constraint propagiert, das heißt, die Constraints, die aus dem Constraint-Speicher und dem aktuellen Constraint mittels der Inferenzkomponente abgeleitet werden können, werden auf den *Propagierungs*-Stack geschrieben und der Constraint-Speicher wird um den aktuellen Constraint erweitert.
    * Wird durch die Propagierung eine Inkonsistenz hergeleitet, dann wird die Backtrackingkomponente aktiviert. Diese restauriert den Zustand des Constraint-Speichers und des *Propagierungs*-Stacks, der vor der letzten Wahl einer Alternative herrschte. Dann wird die nächste Alternative des Wahlpunkts auf den *Propagierungs*-Stack geschrieben und der Propagierungsprozeß beginnt von neuem.
    * Tritt keine Inkonsistenz auf, beginnt der Propagierungsprozeß direkt von neuem.

Terminierung

- Der Algorithmus terminiert mit dem Wert „konsistent", wenn der *Propagierungs*-Stack leer ist.

- Der Algorithmus terminiert mit dem Wert „inkonsistent", wenn nach dem Auftreten einer Inkonsistenz keine weiteren Alternativen existieren, die zu prüfen wären.

Beispiel 4: Sei $\mathcal{T}$ die **TBOX** aus Beispiel 1 und $\mathcal{A}$ die **ABOX** aus Beispiel 2. Eine mögliche Anfrage wäre, ob *Iokaste* im Konzept *Mutter* enthalten ist. Das Instanzenproblem ist realisiert durch einen Widerspruchsbeweis. Bei der Anfrage, ob ein Individuum a eine Instanz eines Konzepts C ist, wird die **ABOX** um das assertionale Axiom $a : \neg C$ erweitert und auf Konsistenz getestet. Dabei ist die **ABOX** genau dann inkonsistent, wenn a eine Instanz des Konzeptes C ist.

Das heißt für dieses Beispiel, daß die **ABOX**

A= $\{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$

auf Konsistenz getestet werden muß.

1. $CS = \{\}$
   $Stack = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$
2. $CS = \{Iokaste : Frau\}$
   $Stack = \{Oedipus : Mensch, (Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$
3. $CS = \{Iokaste : Frau, Oedipus : Mensch\}$
   $Stack = \{(Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$
4. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind\}$
   $Stack = \{Iokaste : \neg Mutter\}$
5. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$
   $Stack = \{Iokaste : \neg Frau \vee \forall Kind.\neg Mensch\}$
   An dieser Stelle wird ein Wahlpunkt angelegt.
6. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$
   $Stack = \{Iokaste : \neg Frau\}$
7. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind,$
   $Iokaste : \neg Mutter, Iokaste : \neg Frau\}$
   $Stack = \{\bot\}$[5]
   Die Backtrackingkomponente findet in $Iokaste : \forall Kind.\neg Mensch$ eine Alternative für $Iokaste : \neg Frau$.
8. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind, Iokaste : \neg Mutter\}$
   $Stack = \{Iokaste : \forall Kind.\neg Mensch\}$
9. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind,$
   $Iokaste : \neg Mutter, \forall Kind.\neg Mensch\}$
   $Stack = \{Oedipus : \neg Mensch\}$
10. $CS = \{Iokaste : Frau, Oedipus : Mensch, (Iokaste, Oedipus) : Kind,$
    $Iokaste : \neg Mutter, \forall Kind.\neg Mensch, Oedipus : \neg Mensch\}$
    $Stack = \{\bot\}$
    Die Backtrackingkomponente findet keine weiteren Wahlpunkte, daraus folgt die Inkonsistenz der **ABOX**, und damit ist *Iokaste* Instanz des Konzepts *Mutter*.

[5] Dabei ist $\bot$ das Symbol für eine Inkonsistenz

Die obige Realisierung eines Terminologischen Inferenzsystems offenbart zwei Schwächen:

- Die naive Behandlung der Disjunktionen, welche in chronologischer Reihenfolge abgearbeitet werden, ohne Rücksicht auf etwaige Ursachen eines Konflikts.
- Die Nicht-Existenz einer inkrementellen Schnittstelle. Bei jeder Anfrage an die Wissensbasis werden alle Axiome der **ABOX** neu propagiert und nicht auf bereits propagierte Constraints zurückgegriffen.

Auf die naive Behandlung der Disjunktionen werde ich im folgenden näher eingehen. Im Anschluß werde ich zeigen, daß mit den zusätzlichen Informationen, die nötig sind, um eine intelligente Backtrackingstrategie zu realisieren, auch eine inkrementelle Schnittstelle realisiert werden kann.

3 Motivation für intelligentes Backtracking

3.1 Theoretische und praktische Arbeiten auf dem Gebiet der Terminologischen Logiken

Erste theoretische Arbeiten[6] über terminologische Formalismen beschäftigen sich mit Fragen der Komplexität und Entscheidbarkeit. In diesen Arbeiten wurde gezeigt, daß die inhärente Komplexität Terminologischer Logiken sehr hoch ist. Das Subsumptions-Problem für ausdrucksstärkere Sprachen (zum Beispiel ALC) ist mindestens NP-hart, und für manche Sprachen (zum Beispiel das ursprüngliche **KL-ONE**) sogar unentscheidbar.

Dieses Problem wurde auf zwei Arten umgangen:

- Die Systeme unterstützen nur eine eingeschränkte Terminologische Sprache, dabei sind die dem Konsistenztest zugrundeliegenden Algorithmen korrekt[7] und vollständig[8] .
- Die Systeme unterstützen eine ausdrucksstärkere Sprache, wobei die Algorithmen korrekt, aber unvollständig sind.

Da dies für bestimmte Anwendungen nicht ausreichend war, sind in den letzten Jahren die Systeme **Taxon**[9] und *KRIS*[10] entwickelt worden. Beide weisen die folgenden Merkmale auf:

- Die terminologische Sprache ist ausdrucksstark,
- das Konsistenzproblem ist entscheidbar und
- die zugrundeliegenden Algorithmen sind korrekt und vollständig.

Vergleiche mit Systemen, die bei ähnlicher Ausdruckstärke unvollständige Algorithmen benutzen, zeigten, daß die unvollständigen Algorithmen erheblich schneller waren. Eine Untersuchung[11] bewies jedoch, daß die Laufzeitunterschiede bei der Klassifikation im wesentlichen nicht auf die vollständigen Algorithmen zurückzuführen sind, sondern auf die unzureichende, prototypische Implementierung dieser.

Daraufhin wurden die vollständigen Algorithmen für die Subsumption wie folgt verbessert:

- In **Taxon** wurde dem eigentlichen Subsumptionstest ein „struktureller" Subsumptionstest vorgeschaltet, der als Ausgabewerte neben „subsumiert" und „subsumiert nicht" noch den Wert „Subsumptionsbeziehung unbekannt" zuließ. Im letzteren Fall wurde dann erst der eigentliche Subsumptionstest aufgerufen. Da der „strukturelle" Subsumptionstest sehr schnell war, und empirische Untersuchungen zeigten, daß der eigentliche Subsumptionstest nur noch in 20 bis 40 Prozent der Fälle aufgerufen werden mußte, kam es zu einer erheblichen Verringerung der Laufzeit.

[6] [Brachmann and Levesque, 1984], [Nebel, 1988], [Schmidt-Schauß, 1989] und [Nebel, 1990]

[7] Korrektheit bedeutet dabei: Liefert der Algorithmus den Wert „inkonsistent", dann ist die **ABOX** tatsächlich inkonsistent.

[8] Vollständigkeit bedeutet dabei: Ist die **ABOX** inkonsistent, dann liefert der Algorithmus den Wert „inkonsistent".

[9] **Taxon** wurde am DFKI von der Projektgruppe **ARC-TEC** entwickelt ([Hanschke *et al.*, 1991])

[10] *KRIS* wurde am DFKI von der Projektgruppe **WINO** entwickelt ([Baader and Hollunder, 1990])

[11] [Baader *et al.*, 1992b]

- In *KRIS* wurde ebenfalls Wissen über die Struktur der Konzepte und der Konzepthierarchie zur Effizienzsteigerung ausgenutzt. Auch dies führte zu erheblichen Verbesserungen der Laufzeit.

Trotz dieser Verbesserungen war damit ein weiteres Problem noch nicht gelöst. Die Laufzeit der vollständigen Algorithmen für das **ABOX**-Reasoning[12], welches für die meisten Applikationen die wesentlichen Inferenzdienste zur Verfügung stellt, war zu hoch, um die Algorithmen in praktisch verwendbaren Systemen einsetzen zu können.

klären, ob die Ursache der zu hohen Laufzeit entweder an der inhärenten Komplexität Terminologischer Logiken oder an der Anzahl der zu prüfenden potentiellen Modelle liegt und nicht an den ineffizienten, vollständigen Algorithmen.

Zur Klärung dieser Frage wurden anwendungsrelevante **ABOX**en und **TBOX**en im Hinblick auf ihre strukturellen Merkmale, sowie das Laufzeitverhalten des Konsistenztests für verschiedene **ABOX**en und **TBOX**en untersucht.

Dabei wurden die folgenden Merkmale festgestellt:

Merkmale der TBOX

(T1) Die sogenannten „schlimmsten Fälle“ möglicher Konzeptterme kommen in der Regel nicht vor

(T2) Die Komplexität der Konzeptterme, das heißt die Anzahl der Konzeptterme, die einen anderen definieren, ist gering (in der Regel weniger als neun).

(T3) Die Anzahl der Disjunktionen innerhalb eines Konzeptterms ist gering. Die meisten Konzeptterme enthalten überhaupt keine Disjunktionen, und wenn, dann in den seltensten Fällen mehr als zwei.

(T4) Rekursive, das heißt zyklische Definitionen von Konzepttermen sind nicht zugelassen.

Merkmale der ABOX

Folgerungen

(F1) Ist die Anzahl der Disjunktionen groß, so geschieht dies dadurch, daß viele Individuen Instanz eines Konzeptterms sind, die mindestens eine Disjunktion enthalten, und nicht dadurch, daß einige Individuen Instanz eines Konzeptterms sind, der sehr viele Disjunktionen enthält. (T3)

(F2) Die Menge der Folgerungen aus den Axiomen ist relativ klein. Im Gegensatz etwa zu Prolog[13], wo durch das Aufrufen eines „goals" selbst wieder zahlreiche „subgoals" aufgerufen werden, die ihrerseits selbst wieder „subgoals" aufrufen, usw...
(T1, T2, A2 und vor allem T4)

(F3) Die Existenz eines Axioms ist in den meisten Fällen nicht abhängig von der Existenz einer Disjunktion. Das heißt die meisten Axiome sind immer gültig. (T3)

Obige Beobachtungen belegen die These, daß die hohe Laufzeit für das **ABOX**-Reasoning nicht auf die inhärente Komplexität Terminologischer Logiken zurückzuführen ist, sondern auf die Handhabung der Disjunktionen in den Algorithmen.

Im folgenden werde ich zeigen, daß die in Terminologischen Inferenzsystemen implementierte Strategie des chronologischen Backtracking für die hohe Laufzeit verantwortlich ist. Gleichzeitig wird deutlich werden, daß eine intelligente Backtrackingstrategie die optimale Strategie beim Ziehen terminologischer Inferenzen ist. Im folgenden werden dazu die drei für diesen Kontext wesentlichsten Backtrackingstrategien beschrieben und an einem Beispiel verdeutlicht.

3.2 Backtrackingstrategien

Die Terminologie entstammt dem Artikel [Günter, 1993]. Dort finden sich noch weitere Backtrackingstrategien, die hier nicht interessant sind.

Chronologisches Backtracking

Tritt im Verlauf der Propagierung ein Konflikt auf, so wird die zuletzt getroffene Entscheidung rückgängig gemacht, bis ein Wahlpunkt angetroffen wird, der eine noch nicht untersuchte Alternative bereithält. Danach werden alle Konsequenzen zurückgenommener Entscheidungen ungültig.
Bei diesem Verfahren werden weder die möglichen Ursachen des Konflikts berücksichtigt, noch Daten, die am Konflikt unbeteiligt sind übernommen. Dabei kann die Anzahl der unnötigen Mehrfachberechnung von Daten exponentiell mit der Anzahl der Wahlpunkte steigen.

Es gibt viele Systeme, in denen die Strategie des chronologischen Backtracking berechtigt ist (PROLOG). Dies liegt daran, daß die Abhängigkeiten zwischen den Daten sehr groß sind. Aufgrund der einfachen Realisierbarkeit hat sich chronologisches Backtracking als Standardverfahren etabliert.

Abhängigkeitsgesteuertes Backtracking

Beim abhängigkeitsgesteuerten Backtracking geht man bis zu der letzten Entscheidung zurück, die am Konflikt beteiligt ist, und die eine noch nicht untersuchte Alternative bereithält. Alle

Konsequenzen zurückgenommener Entscheidungen werden ungültig. Abhängigkeitsgesteuertes Backtracking setzt voraus, daß zu jedem Konflikt Informationen über dessen Ursache explizit oder implizit vorhanden sind. Auch hier sind Mehrfachberechnungen nicht ausgeschlossen, jedoch ist deren Anzahl im Normalfall wesentlich geringer.

Intelligentes Backtracking (oder Backtracking mit Datenübernahme)

Beim intelligenten Backtracking wird nicht nur *ein* Wahlpunkt zurückgenommen, sondern die gesamte Auswahl der Wahlpunkte wird so geändert, daß der Konflikt verschwindet. Alle Konsequenzen zurückgenommener Wahlpunkte werden ungültig. Die Daten, die von der Änderung nicht betroffen sind, bleiben erhalten. Intelligentes Backtracking setzt nicht nur Informationen über die Ursachen der Inkonsistenz voraus, sondern auch über die Abhängigkeiten zwischen den Daten.

Das folgende Beispiel stellt eine stark vereinfachte **ABOX** dar, ist aber insofern typisch, als die Konstrukte die die hohe Laufzeit verursachen, darin enthalten sind. Auf jeden Fall läßt sich die Abarbeitung der Constraints mit den jeweiligen Backtrackingstrategien gut daran demonstrieren.

Beispiel Gegeben sei die folgende **ABOX**:

$$\begin{aligned}\{X_1 &: C_{11} \vee C_{12},\\ X_2 &: C_{21} \vee C_{22},\\ &\ldots\\ X_n &: C_{n1} \vee C_{n2},\\ X_1 &: \neg C_{11}\},\end{aligned}$$

wobei die X_i Objekte und die C_{ij} Konzeptterme sind.

Unabhängig von der Backtrackingstrategie werden zuerst die Disjunktionen $X_i : C_{i1} \vee C_{i2}$ $(1 \leq i \leq n)$ propagiert. Das heißt, da im allgemeinen keine Prioritätsfunktion für die Disjunktionsglieder existiert, werden die $X_i : C_{i1}$ zugesichert und die $X_i : C_{i2}$ als deren Alternativen verwaltet. Danach führt die Propagierung von $X_1 : \neg C_{11}$ zu einem inkonsisten Zustand:

Abarbeitung beim chronologischen Backtracking Der Backtrackingmechanismus setzt $X_n : C_{n1}$ zurück und sichert stattdessen die Alternative $X_n : C_{n2}$ zu. Auch dies führt nach einer erneuten Propagierung von $X_1 : \neg C_{11}$ zu einem Konflikt. Nachdem 2^{n-1} Alternativen geprüft wurden, wird die eigentliche Ursache des Widerspruchs $X_1 : C_{11}$ durch ihre Alternative ersetzt. Danach werden die Axiome $X_i : C_{i1} \vee C_{i2}$ $(2 \leq i \leq n)$ und $X_1 : \neg C_{11}$ erneut zugesichert.

Abarbeitung beim abhängigkeitsgesteuerten Backtracking Bei diesem Verfahren werden die Zusicherungen $X_i : C_{i1}$ $(2 \leq i \leq n)$ zurückgenommen, ohne daß ihre Alternativen propagiert werden, da die $X_i : C_{i1}$ nicht am Konflikt beteiligt sind. Dann wird $X_1 : C_{11}$ ersetzt und die Axiome $X_i : C_{i1} \vee C_{i2}$ $(2 \leq i \leq n)$ und $X_1 : \neg C_{11}$ erneut zugesichert.

Abarbeitung beim intelligenten Backtracking Bei diesem Verfahren wird die gesamte Auswahl der Wahlpunkte neu berechnet. Die neue Auswahl unterscheidet sich im Beispiel an einer Stelle von der alten. $X_1 : C_{11}$ wird durch $X_1 : C_{12}$ ersetzt.

Obige Ausführungen und das Beispiel verdeutlichen folgende Aussagen:

- Die hohen Laufzeiten für das **ABOX**-Reasoning liegen nicht an der inhärenten Komplexität Terminologischer Logiken. (T1 und L1)
- Beim Ziehen Terminologischer Inferenzen ist, infolge der meist lokalen Abhängigkeiten, die Anzahl der potentiellen Modelle, die man prüfen muß, um festzustellen, ob ein Modell existiert oder nicht, gering. (A2 und F1)
- Der zusätzliche Aufwand für die Verwaltung der Abhängigkeiten wiegt in fast allen Fällen nicht so schwer, wie der Effizienzgewinn, den man erhält, dadurch daß man weniger potentielle Modelle prüfen muß. (F2 und F3)

4 Von der Semantik zum Entscheidungsverfahren

Im folgenden werde ich von Terminologischen Logiken abstrahieren, indem ich stattdessen einen abstrakten Regelformalismus betrachte, mit dem dann die operationale Semantik einer Teilsprache erster Stufe beschrieben werden kann. Dieser Regelformalismus muß folgender Bedingung genügen:

> Es existiert ein Algorithmus, der für jede zulässige Instanz der Regelsprache einer gegebenen Menge von Formeln das Attribut „syntaktisch konsistent" beziehungsweise „syntaktisch inkonsistent" zuordnet.

Dabei wird die Regelsprache so gewählt, daß sich die operationale Semantik von ALC damit beschreiben läßt.
Am Ende des folgenden Abschnitts stehen die Begriffe der Inkonsistenz eines Constraint-Systems und der operationalen Semantik. Diese bilden eine Schnittstelle zwischen einem abstrakteren Semantikbegriff auf der einen und dem Regelsystem auf der anderen Seite.

4.1 Operationale Semantik

Gegeben seien eine Menge von Constraintnamen $\mathcal{CN}$, eine Menge von Funktoren $\mathcal{FU}$ und eine Menge von Variablen $\mathcal{V}$.

Definition 4.1 *Die Menge der* **Terme** *ist induktiv definiert durch:*

1. *Alle 0-stelligen Funktoren $f_i^0 \in \mathcal{FU}$ und alle Variablen $v_j \in \mathcal{V}$ sind Terme.*
2. *Sind $t_1, t_2, ..., t_n$ $(n \geq 1)$ Terme und f_i^n ein n-stelliger Funktor, dann ist auch $f_i^n(t_1, t_2, ..., t_n)$ ein Term.*

Definition 4.2 *Ein* **Grundterm** *ist ein Term, der keine Variablen enthält.*

Definition 4.3 *Die Menge der* **Formeln** $\mathcal{F}$ *wird induktiv definiert durch:*

1. $\top$ *und* $\bot$ *sind Formeln.*
2. *Sind $t_1, ..., t_n$ $(n \geq 1)$ Terme und ist C ein n-stelliger Constraintname, dann ist $C(t_1, ..., t_n)$ eine Formel.*

Definition 4.4 *Ein* **Constraint** *ist eine Formel, deren Terme Grundterme sind. Die Menge aller Constraints wird mit CT notiert.*

Definition 4.5 Regeln *Seien C_{p1}, ..., C_{pn} und C_{f1}, ..., $C_{fm}(m, n \geq 1)$ Formeln, dann ist*

$$C_{p1}, ..., C_{pn} \rightsquigarrow C_{f1}; ...; C_{fm}$$

eine Regel, wobei jede Variable, die auf der rechten Seite des Pfeils in einem Constraint auftritt, auch auf der linken Seite in einem Constraint vorkommt.

Definition 4.6 *Eine* **Auswahlfunktion** $\mathcal{A}$ *für Wahlpunkte ist eine partielle Funktion* $\mathcal{A} : \mathcal{P}(\mathcal{C}) \rightarrow \mathcal{C}$ *mit* $\mathcal{A}(\{ct_1, ..., ct_i, ..., ct_n\}) = ct_i$.

Definition 4.7 *Eine endliche Menge* $\mathcal{R}$ *von Regeln heißt* **Regelsystem.**

Definition 4.8 Substitution

1. *Eine* **Substitution** ϕ *ist eine endliche Menge der Form* $\{\langle v_1, t_1\rangle, ..., \langle v_n, t_n\rangle\}$, *wobei die* v_i *paarweise voneinander verschiedene Variablen sind und die* t_i *von* v_i *verschiedene Terme sind.*
2. *Sei* ϕ *eine Substitution und r eine Regel. Dann ist* $r\phi$ *eine* **Instanz** *von r, das heißt, die Regel, die durch gleichzeitiges Ersetzen jedes Vorkommens der Variablen* v_i *in r durch die Terme* t_i *entsteht.*

Definition 4.9 Herleitbarkeit

1. *Eine Disjunktion* ct_a; ...; ct_z *heißt* **direkt herleitbar** *aus einer Constraintmenge CS bezüglich eines Regelsystems* $\mathcal{R}$ **gdw** *es eine Regel* $r \in \mathcal{R}$ *und eine Substitution* ϕ *gibt, so daß* $r\phi \equiv ct_1, ..., ct_n \Rightarrow ct_a$; ...; ct_z *und* $\forall i : ct_i \in CS$.
   (Notation: $CS \longrightarrow_{\mathcal{R}} ct_a$; ...; ct_z*)*
2. *Ein Constraint* ct_x *heißt* **direkt herleitbar** *aus einer Constraintmenge CS bezüglich eines Regelsystems* $\mathcal{R}$ *und einer Auswahlfunktion* $\mathcal{A}$ **gdw** *es eine Regel r gibt, so daß* $r\phi \equiv ct_1, ..., ct_n \Rightarrow ct_a$; ...; ct_z *und* $\forall i : ct_i \in CS$ *und* $\mathcal{A}(\{ct_a, ..., ct_z\}) = ct_x$.
   (Notation: $CS \xrightarrow{\mathcal{A}}_{\mathcal{R}} ct_x$*)*
3. *Ein Constraint ct heißt* **herleitbar** *aus einer Constraintmenge CS bezüglich eines Regelsystems* $\mathcal{R}$ *und einer Auswahlfunktion* $\mathcal{A}$ **gdw** *es ein* $i \geq 0$ *gibt, so daß* $ct \in X_i$, *wobei* $X_0 := CS$ *und* $X_{i+1} := X_i \cup \{ct' | X_i \xrightarrow{\mathcal{A}}_{\mathcal{R}} ct'\}$.
   (Notation: $CS \xrightarrow{\mathcal{A},*}_{\mathcal{R}} ct_x$*)*

Definition 4.10 *Ein Constraintmenge CS heißt* **(syntaktisch) inkonsistent** **gdw** *für jede Auswahlfunktion* $\mathcal{A}$ *gilt:* $CS \xrightarrow{\mathcal{A},*}_{\mathcal{R}} \bot$.

Definition 4.11 *Ein Regelsystem* $\mathcal{R}$ *enthält bezüglich einer Constraintmenge CS einen* **Zyklus,** *wenn gilt: Es existiert eine Auswahlfunktion* $\mathcal{A}$ *sodaß* $CS \xrightarrow{\mathcal{A},*}_{\mathcal{R}} ct$ *und* $ct \in CS$.

Definition 4.12 *Ein Regelsystem* $\mathcal{R}$ *heißt* **zulässig** *bezüglich einer endlichen Menge* $C \subseteq CT$, *wenn für alle* $CS \subseteq C$ *gilt: Für jede Auswahlfunktion A ist die Menge aller Folgerungen Folg(CS, A):=*$\{ct | CS \xrightarrow{A,*}_{\mathcal{R}} ct\}$ *aus CS endlich.*

Definition 4.13 *Sei L eine Sprache erster Stufe. Ein Regelsystem* $\mathcal{R}$ *heißt* **operationale Semantik** *für eine Sprache L* **gdw** *eine Abbildung* $\psi : \mathcal{P}(Form(L)) \mapsto \mathcal{P}(C)$ *existiert, derart daß für alle* $F \subseteq Form(L)$ *gilt: F ist semantisch inkonsistent* **gdw** $\psi(F)$ *ist syntaktisch inkonsistent.*

Nun ist nicht klar, wie man bei gegebener Sprache L und zugehöriger Semantik $\mathcal{I}$ zu einer operationalen Semantik in Form eines Regelsystems kommen kann und ob diese überhaupt existiert. An dieser Frage sind wir hier jedoch nicht interessiert. Uns genügt es zu wissen, daß ein Regelsystem $\mathcal{R}$ existiert, welches operationale Semantik für ALC ist.

4.2 Beispiel: Die operationale Semantik von ALC

Seien

- „cterm“, „ncterm“, „role“, „definition“ und „cnamep“ Constraints,
- „and“, „or“, „not“, „some“, „forall“ und „new“ Funktoren,
- „Obj“, „C“, „C_1“, „C_2“ und „Ct“ Variablen und
- das Regelsystem $\Re$ gegeben durch

$$
\begin{aligned}
cterm(Obj,\ and(C_1,\ C_2)) &\rightsquigarrow cterm(Obj,\ C_1)\\
cterm(Obj,\ and(C_1,\ C_2)) &\rightsquigarrow cterm(Obj,\ C_2)\\
cterm(Obj,\ or(C_1,\ C_2)) &\rightsquigarrow cterm(Obj,\ C_1)\ ;\ cterm(Obj,\ C_2)\\
cterm(Obj,\ not(C)) &\rightsquigarrow ncterm(Obj,\ C)\\
cterm(Obj,\ C),\ cnamep(C),\ definition(C,\ ct) &\rightsquigarrow cterm(Obj,\ ct)\\
cterm(Obj,\ forall(Role,\ C)),\ role(Obj,\ Role,\ Obj2) &\rightsquigarrow cterm(Obj2,\ C)\\
cterm(Obj,\ some(Role,\ C)) &\rightsquigarrow cterm(new(Obj,\ Role,\ C),\ C)\\
cterm(Obj,\ some(Role,\ C)) &\rightsquigarrow role(Obj,\ Role,\ new(Obj,\ Role,\ C))\\
ncterm(Obj,\ and(C_1,\ C_2)) &\rightsquigarrow ncterm(Obj,\ C_1)\ ;\ ncterm(Obj,\ C_2)\\
ncterm(Obj,\ or(C_1,\ C_2)) &\rightsquigarrow ncterm(Obj,\ C_1)\\
ncterm(Obj,\ or(C_1,\ C_2)) &\rightsquigarrow ncterm(Obj,\ C_2)\\
ncterm(Obj,\ not(C)) &\rightsquigarrow cterm(Obj,\ C)\\
ncterm(Obj,\ C),\ cnamep(C),\ definition(C,\ ct) &\rightsquigarrow ncterm(Obj,\ ct)\\
ncterm(Obj,\ forall(Role,\ C)) &\rightsquigarrow ncterm(new(Obj,\ Role,\ C),\ C)\\
ncterm(Obj,\ forall(Role,\ C)) &\rightsquigarrow role(Obj,\ Role,\ new(Obj,\ Role,\ C))\\
ncterm(Obj,\ some(Role,\ C)),\ role(Obj,\ Role,\ Obj2) &\rightsquigarrow ncterm(Obj2,\ C)\\
cterm(Obj,\ C),\ ncterm(Obj,\ C) &\rightsquigarrow \bot,
\end{aligned}
$$

dann gilt: Das Regelsystem $\Re$ ist operationale Semantik von ALC.

Beweis

Sei die Transformationsfunktion wie folgt definiert:

$$\psi(F) := \bigcup_{f \in F} \psi_0(f),$$

wobei ψ_0 wie folgt definiert ist:

- Für das TBox-Reasoning:
  $\psi_0(\mathrm{C} := \mathrm{ct}) \mapsto \{\mathrm{definition(C,}\ \psi_1(ct)\mathrm{)},\ \mathrm{cnamep(C)}\}$,
  wobei ψ_1 wie folgt definiert ist:

$$
\begin{aligned}
\psi_1(C) &= C, \text{ falls C Konzeptname,} \\
\psi_1(C \sqcap D) &= and(\psi_1(C), \psi_1(D)), \\
\psi_1(C \sqcup D) &= or(\psi_1(C), \psi_1(D)), \\
\psi_1(\neg C) &= not(\psi_1(C)), \\
\psi_1(\exists R.C) &= some(R, \psi_1(C)) \; und \\
\psi_1(\forall R.C) &= forall(R, \psi_1(C)).
\end{aligned}
$$

- Für das ABox-Reasoning:
  ψ_0(a:C) $\mapsto$ {cterm(a, C)}
  ψ_0((a,b):R) $\mapsto$ {role(a, R, b)}

dann wird in [Baader and Hanschke, 1991] die Äquivalenz der Unerfüllbarkeit einer Formelmenge F und der Inkonsistenz eines Constraint-Systems ψ(F) gezeigt. □

Das Regelsystem $\mathfrak{R}$ entspricht im wesentlichen einem Teil der durch **Taxon** realisierten operationalen Semantik. Dort wird lediglich ausgenutzt, daß

- zusätzliches Wissen über die Konzepthierarchie vorhanden ist und,
- daß man Konzeptterme, aus denen sich keine weiteren Constraints ableiten lassen, aus der Wissensbasis entfernen kann.

Allerdings ließen sich beide Optimierungen durch ein erweitertes Regelschema, welches auch externe Prädikate und sogenannte Vereinfachungsregeln[14] kennt, realisieren.

4.3 Regelinterpreter

Was nun noch fehlt, ist die Operationalisierung des Begriffs der Herleitbarkeit einer Inkonsistenz. Dazu definieren wir einen Vorwärtsregelinterpreter, der eine einfache Tableau-Methode realisiert. Dabei wird das Constraint-System bezüglich sich chronologisch ändernder Auswahlfunktionen $\mathcal{A}_i$ auf Konsistenz getestet, bis entweder ein Modell gefunden, oder aufgrund der Nichtexistenz weiterer Auswahlfunktionen die Inkonsistenz festgestellt wird.

Definition 4.14 *Konsistenztest $k_{\mathcal{R}}$*

1. *Sei $\mathcal{A}$ eine Auswahlfunktion. Die* **Folgerelation** $\overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}}$ *auf $\mathcal{P}(\mathcal{C})$ ist definiert durch:*

$$
CS \overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} \begin{cases} NIL & : \; falls \; \bot \in CS \\ CS \cup \{ct_i\} & : \; falls \; \bot \notin CS \; und \\ & \qquad CS \longrightarrow_{\mathcal{R}} ct_1; \ldots; ct_n \; und \\ & \qquad \forall j : ct_j \notin CS \; und \\ & \qquad \mathcal{A}(\{ct_1, \ldots, ct_n\}) = ct_i \\ T & : \; sonst \end{cases}
$$

Die transitive Hülle von $\overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}}$ wird mit $\overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}}$ notiert.

[14] [Herbig, 1993]

2. *Der* **Konsistenztest** $k_{\mathcal{R}}$ *für eine Constraintmenge CS ist definiert durch:*
$k_{\mathcal{R}}(CS)$ **gdw** *es eine Auswahlfunktion* $\mathcal{A}$ *gibt, so daß* $CS \overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} T$.

Satz 2 (Termination von $k_{\mathcal{R}}$) *Sei* $\mathcal{R}$ *ein zulässiges Regelsystem bezüglich* $\mathcal{C}$. *Dann ist* $k_{\mathcal{R}}(CS)$ *definiert für alle* $CS \subseteq \mathcal{C}$.

Beweis: Da $\mathcal{R}$ zulässig ist, ist die Menge der Folgerungen aus CS endlich. Da keine Constraints mehrfach abgeleitet werden, ist die Behauptung gültig.

Satz 3 (Korrektheit und Vollständigkeit von $k_{\mathcal{R}}$) *Sei* $\mathcal{R}$ *ein zulässiges Regelsystem bezüglich* $\mathcal{C}$. *Dann gilt für* $CS \subseteq \mathcal{C}$: *CS ist konsistent* **gdw** $k_{\mathcal{R}}(CS)$.

Beweis: CS ist syntaktisch konsistent **gdw** es eine Auswahlfunktion $\mathcal{A}$ gibt, so daß $\bot$ nicht herleitbar ist **gdw** es eine Auswahlfunktion $\mathcal{A}$ (die gleiche) gibt, so daß CS $\overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}}$ T **gdw** $k_{\mathcal{R}}(CS)$ □ .

In der Definition von $\overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}}$ ist im Bedingungsteil der zweiten Zeile die Suche nach einer Regel und einer Teilmenge des aktuellen Constraintsystems versteckt, mit der man die Disjunktion aus CS ableiten kann. Deswegen wird $\overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}}$ geeignet modifiziert, so daß diese Suche explizit ist. Der im folgenden vorgestellte Konsistenztest $\hat{k}_{\mathcal{R}}$ ist eine Formalisierung des in Kapitel 2.4.1 vorgestellten Constraint-Propagierungs-Systems. Die erste Komponente der Tupel auf denen die Folgerelation erklärt ist, entspricht dem Constraint-Speicher, die zweite dem Propagierungsstack.

Definition 4.15 *Konsistenztest* $\hat{k}_{\mathcal{R}}$

1. *Sei* $\mathcal{A}$ *eine Auswahlfunktion. Die* **Folgerelation** $\overset{\mathcal{A}}{\Longrightarrow}_{\hat{k}_{\mathcal{R}}}$ *auf* $\mathcal{P}(\mathcal{C}) \times \mathcal{P}(\mathcal{C})$ *ist definiert durch:*

$$\begin{aligned}
\langle CS \cup \{\bot\}, CS' \rangle &\overset{\mathcal{A}}{\Longrightarrow}_{\hat{k}_{\mathcal{R}}} NIL \\
\langle CS, CS' \cup \{(ct_1; ...; ct_n)\} \rangle &\overset{\mathcal{A}}{\Longrightarrow}_{\hat{k}_{\mathcal{R}}} \langle CS \cup \{ct_i\}, CS' \cup \mathcal{CP}_{\mathcal{R}}(CS, ct_i) \rangle , \\
&\qquad \textit{wobei } \mathcal{A}(\{ct_1; ...; ct_n\}) = ct_i \\
\langle CS, \{\} \rangle &\overset{\mathcal{A}}{\Longrightarrow}_{\hat{k}_{\mathcal{R}}} T, \textit{ wobei } \bot \notin CS
\end{aligned}$$

Beweis:
Aufgrund der Definition von $\mathcal{CP}_\mathcal{R}$ und der Folgerelation $\stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}}$ gilt für jede Auswahlfunktion $\mathcal{A}$:
Aus $\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS_1,CS_2\rangle$ folgt $CS_1 \subseteq \mathit{Folg}(CS,\mathcal{A})$ und $CS_2 \subseteq \mathit{Folg}(CS,\mathcal{A})$.
Da $\mathcal{R}$ zulässig ist, ist Folg(CS,$\mathcal{A}$) endlich. Nach Definition wächst das erste Argument bei jedem Übergang. Nach endlich vielen Schritten gilt dann entweder:
$\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} T,\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \mathit{NIL}$ oder $\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS_1,CS_2\rangle$ und $CS_1 = \mathit{Folg}(CS,\mathcal{A})$.
Aus der Definition von $\mathcal{CP}_\mathcal{R}$ folgt, daß diese Funktion keine weiteren Constraints ableitet. Daraus folgt, daß das zweite Argument im folgenden stetig abnimmt, bis entweder die leere Menge erreicht wird, oder der Constraint-Speicher eine Inkonsistenz enthält.

Satz 5 (Korrektheit und Vollständigkeit von $\hat{k}_\mathcal{R}$) *Sei $\mathcal{R}$ ein zulässiges Regelsystem bezüglich $\mathcal{C}$. Dann gilt für alle $CS \subseteq \mathcal{C}$: CS ist konsistent* **gdw** *$\hat{k}_\mathcal{R}(CS)$.*

Beweis:

1. Sei CS konsistent. Dann existiert eine Auswahlfunktion $\mathcal{A}$, so daß aus CS $\perp$ nicht abgeleitet werden kann. Dann gilt für jede Menge von Constraints CS' mit $\langle\{\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS',CS''\rangle : \perp \notin CS'$ und somit $\hat{k}_\mathcal{R}$(CS).

2. Es gelte $\hat{k}_\mathcal{R}$(CS). Dann gibt es eine Auswahlfunktion $\mathcal{A}$ und eine Menge CS' mit $\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS',\{\}\rangle$.
   Hilfsbehauptung:
   Aus $\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS_1,CS_2\rangle$ folgt $CS_1 \cup \{ct|CS_1 \longrightarrow_\mathcal{R} ct\} \subseteq CS_1 \cup CS_2$
   Beweis: Induktion über die Länge der Berechnung
   n=0

   $$\{\} \cup \{ct|\{\} \longrightarrow_\mathcal{R} ct\} = \{\} \subseteq \{\} \cup CS = CS$$

   $n \rightarrow n+1$
   Sei $\langle\{\},CS\rangle \stackrel{*,\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS_1,CS_2 \cup \{(ct_a;...;ct_z)\}\rangle \stackrel{\mathcal{A}}{\Longrightarrow}_{\hat{k}_\mathcal{R}} \langle CS_1 \cup \{ct_x\},CS_2 \cup \mathcal{CP}_\mathcal{R}(CS_1,ct_x)\rangle$.
   Nach I.V. gilt: $CS_1 \cup \{ct|CS_1 \longrightarrow_\mathcal{R} ct\} \subseteq CS_1 \cup CS_2 \cup \{(ct_a;...;ct_z)\}$
   Dann folgt aus der Definition von $\mathcal{CP}_\mathcal{R}$ und der Induktionsvoraussetzung

   $$CS_1 \cup \{ct_x\} \cup \{ct|CS_1 \cup \{ct_x\} \longrightarrow_\mathcal{R} ct\} \subseteq CS_1 \cup \{ct_x\} \cup CS_2 \cup \mathcal{CP}_\mathcal{R}(CS_1,ct_x)$$

   und somit die Hilfsbehauptung.

   Aus der Hilfsbehauptung folgt, daß CS' eine Obermenge von $CS' \cup \{ct|CS' \longrightarrow_\mathcal{R} ct\}$ ist, d.h. aus CS' können keine weiteren Constraints abgeleitet werden. Damit ist CS konsistent.

Die obige Realisierung des Konsistenztests $\hat{k}_\mathcal{R}$ stellt den Ausgangspunkt für meine Betrachtungen hinsichtlich des intelligenten Backtrackings dar. Im folgenden werde ich, aufbauend auf diesem, einen Regelinterpreter entwickeln, der anstatt vielfach einfach alle möglichen Auswahlfunktionen durchzuprobieren, nur diejenigen prüft, die notwendig sind, um festzustellen, ob eine Menge von Constraints konsistent ist oder nicht.

5 Intelligentes Backtracking

Wie in Kapitel 3 beschrieben, scheint eine intelligente Backtrackingstrategie von erheblichem Nutzen für Terminologische Logiken zu sein. Einer der wesentlichen Gründe für den Einsatz einer chronologischen Backtrackingstrategie ist die relativ einfache Realisierbarkeit. Genau umgekehrt verhält es sich mit einer intelligenten Backtrackingstrategie. In diesem Kapitel wird beschrieben welche zusätzlichen Probleme auftreten, wenn man statt einer chronologischen Backtrackingstrategie eine intelligente realisiert, und wie diese Probleme gelöst werden können.

Der Ansatz, der in dieser Arbeit verfolgt wird, um eine solche Strategie zu realisieren, ist der folgende:

> Die Suche nach einer Auswahlfunktion, bezüglich derer man keine Inkonsistenz herleiten kann, wird realisiert durch eine Suche im Raum der konfliktauflösenden Auswahlfunktionen. Die Eigenschaft konfliktauflösend schränkt dabei die möglichen Auswahlfunktionen erheblich ein.

Welche zusätzlichen Informationen müssen wann verfügbar sein?

Die Grundidee ist, im Falle einer Inkonsistenz, die Auswahlfunktion „minimal" zu ändern, so daß der Konflikt vermieden wird. Dazu werden Informationen über die Ursache der Inkonsistenz benötigt (*Problem der Konfliktursache*). Mit Hilfe dieser Information läßt sich dann jeder Inkonsistenz eine Menge von alternativen Änderungen der aktuellen Auswahlfunktionen zuordnen. Hat man aus dieser Menge eine Alternative ausgewählt, so müssen bestimmte Entscheidungen und deren Folgerungen zurückgenommen werden *(Problem der Validität)*; andere müssen propagiert werden. Dabei muß sichergestellt werden, daß erstens eine Auswahlfunktion nicht mehrfach geprüft wird *(Problem der Termination)* und zweitens, daß auch alle potentiellen Auswahlfunktionen untersucht werden *(Problem der Korrektheit)*.

5.1 Problem der Konfliktursache

Im Falle eines Konflikts müssen, die den Konflikt verursachenden Entscheidungen, bestimmt werden. Da prinzipiell jeder Constraint potentiell an einem Konflikt beteiligt ist, muß zu jedem Constraint die Menge von **Entscheidungen** bekannt sein, die die Existenz des Constraints rechtfertigen. Dabei kann es mehrere Begründungen für einen Constraint geben. Die leere Menge ist dabei eine Begründung für einen Constraint, der immer gültig ist. Die Menge aller Begründungen eines Constraints wird Begründungsklasse genannt. In Abhängigkeit der Konfliktursache läßt sich dann ein Mengensystem von Entscheidungen bestimmen, die sogenannte Alternativenklasse, deren Elemente Mengen von Entscheidungen sind, die Alternativen zu den Entscheidungen darstellen, die zurückgenommen werden müssen, um die Inkonsistenz (lokal) zu eliminieren.

Es existieren im wesentlichen zwei Ansätze diese Begründungsklasse zu bestimmen:

1. Zu jedem Constraint wird explizit ein Verweis, der im folgenden Begründungsfunktion genannt wird, auf dessen Begründungsklasse angelegt.

2. Zu jedem Constraint werden lediglich Rückwärtsverweise auf dessen Prämissen gespeichert. Die Begründungsklasse eines Constraints wird dann bei Anfrage mit Hilfe dieser Prämissen berechnet.

Struktur	*Aufbau*	*Berechnung der Konfliktursache*
Begründungsklasse ist explizit vorhanden	Für jeden Constraint muß die Begründungsklasse berechnet und gehalten werden. Nachteil: Die Begründungsklasse wird auch für Constraints berechnet, die keine Inkonsistenz verursachen.	Verweis
Rückwärtsverweise auf die Prämissen	Für jeden Constraint werden die Rückwärtsverweise auf die Prämissen eingetragen	Nachteil: Die Begründungsklasse muß bei jedem Konflikt neu berechnet werden

[redacted]

Eine Bewertung wird erst später vorgenommen, da beide Strukturen zur Lösung des Validitätsproblems benutzt werden können.

5.2 Problem der Validität der Constraints

Um die Konfliktauflösung zu erreichen müssen nach der Bestimmung der Alternativenklasse bestimmte Entscheidungen und deren Auswirkungen zurückgenommen werden. Dabei ändert sich die Validität mancher Constraints.

Methoden zur Bestimmung der Validität eines Constraints

1. Die Gültigkeit eines Constraints wird auf die Gültigkeit der Prämissen des Constraints zurückgeführt. Die Endpunkte dieser Rekursion stellen einerseits die Constraints dar, die keine Prämissen besitzen, also diejenigen, die immer gültig sind, andererseits Entscheidungen, auf denen eine sogenannte *Validitätsfunktion* definiert ist. Diese Validitätsfunktion ordnet jeder Entscheidung den Wert „gültig“ oder „ungültig“ zu.

2. Die Gültigkeit eines Constraints wird auf die Gültigkeit der Entscheidungen der explizit vorhandenen Begründungsklasse reduziert. Die Gültigkeit der einzelnen Begründungen der Begründungsklasse wird dabei ebenfalls mittels einer Validitätsfunktion auf Entscheidungen bestimmt.

3. Die Gültigkeit eines Constraints wird mit Hilfe der Methoden 1 oder 2 bestimmt und anschließend gespeichert. Diese Berechnug muß nach jedem Wechsel der Auswahlfunktion noch einmal durchgeführt werden. Daraus folgt, daß zusätzlich bekannt sein muß, ob seit der letzten Berechnung die Auswahlfunktion geändert wurde oder nicht.

Sei der Constraint ct eine Folgerung der Entscheidung E, die zurückgenommen wurde. Dann kann es sein, daß die Existenz des Constraints ct durch andere Entscheidungen gerechtfertigt wird. Also muß bekannt sein, ob ct auch dann noch gültig ist, wenn E zurückgenommen wird.

Diese Information kann mittels der Methoden 1 oder 2 berechnet werden. Das heißt, die dort verwendeten Strukturen müssen auch bei dieser Methode vorhanden sein.

Methode	*Anfrage, ob ein Constraint gültig ist.*	*Wechsel der Auswahlfunktion*	*Notwendige Strukturen*
1.	Die Berechnung der Gültigkeit erfolgt mittels der Rückwärtsverweise. Nachteil: teure Berechnung bei jeder Anfrage	Die Validitätsfunktion wird für die Entscheidungen, die zurückgenommen werden müssen, geändert.	Rückwärtsverweise auf die Prämissen
	−− (Zeit)	++ (Zeit)	++ (Speicherplatz)
2.	Die Gültigkeit eines Constraints wird mittels der Begründungsklasse berechnet. Nachteil: Berechnung bei jeder Anfrage	Die Validitätsfunktion wird für die Entscheidungen, die zurückgenommen werden müssen, geändert.	Die Begründungsklasse ist explizit vorhanden.
	−	++	+
3.	Nach jedem Wechsel der Auswahlfunktion wird die Gültigkeit höchstens einmal berechnet und dann gespeichert. Nachteil: Die Validität der Constraints, die am Konflikt unbeteiligt sind, muß auch neu berechnet werden.	Die Validitätsfunktion wird für die Entscheidungen, die zurückgenommen werden müssen, geändert.	1.Rückwärtsverweise auf die Prämissen oder explizit vorhandene Begründungsklasse 2.Marke (Zähler)
	+	++	+
4.	Die Information ist direkt vorhanden	Die Validitätsfunktion für Constraints muß in Abhängigkeit der ungültigen Entscheidungen neu berechnet werden. Nachteil: Die Validität wird auch für Constraints berechnet, deren Gültigkeit bis zum nächsten Wechsel der Auswahlfunktion, nicht abgefragt wird.	1.Vorwärtsverweise auf die Folgerungen 2. Es existieren Rückwärtsverweise auf die Prämissen beziehungsweise ist die Begründungsklasse explizit vorhanden.
	++	−	−

Die Anfrage, ob ein Constraint gültig ist oder nicht, tritt wesentlich häufiger auf als ein Wechsel der Auswahlfunktion. Der beanspruchte Speicherplatz kann im Prinzip vernachlässigt werden.

Da die Methoden 3 und 4 Erweiterungen der Methoden 1 und 2 darstellen und sich in fast allen Fällen als günstiger erweisen werden, werden diese getrennt betrachtet.

Kritik der Methoden 1 und 2

> Der Nachteil von Methode 2 aus dem vorherigen Abschnitt, daß die Begründungsklasse auch für Constraints berechnet und gespeichert wird, die keinen Konflikt verursachen, ist hinfällig, da die Begründungsklasse auch dazu benutzt werden kann, die Gültigkeit eines Constraints zu bestimmen. Auch der größere Speicherplatzbedarf fällt nicht ins Gewicht.
> Daher ist die explizite Existenz einer Begründungsklasse der Berechnung dieser aus den Rückwärtsverweisen auf die Prämissen eindeutig vorzuziehen[15].

Kritik der Methoden 3 und 4

> Methode 4 ist empfindlich gegen sehr häufige Wechsel der Auswahlfunktion und lange Ableitungsketten (dadurch verursacht, daß die Menge der Folgerungen aus einem Constraint groß ist).
> Methode 3 ist ungünstig, wenn der Wechsel der Auswahlfunktion gerade dann erfolgt, wenn die Gültigkeit vieler Constraints genau einmal angefragt wurde.
> Eine eindeutige Bewertung zugunsten einer der Methoden 3 oder 4 – ohne Bezug zu einer bestimmten Anwendung – kann also nicht gegeben werden.
> Die relativ kurzen Ableitungsketten lassen erwarten, daß für den Bereich Terminologischer Logiken, Methode 4, im verein mit Methode 2, die Günstigste ist.[16]

Die Ursache eines Konflikts und die Gültigkeit eines Constraints lassen sich also auf eine Begründungsklasse zurückführen. Dabei bestehen deren Elemente, die Begründungen, ausschließlich aus **Entscheidungen** und nicht aus Constraints.

5.3 Problem der Terminierung

Beim intelligenten Backtracking werden nicht alle potentiellen Auswahlfunktionen in chronologischer Reihenfolge geprüft, sondern in bezug auf die aktuelle Inkonsistenz wird eine alternative Auswahlfunktion berechnet. Bei diesem dynamischen Prozeß müssen zusätzlich Informationen darüber bekannt sein, welche Auswahlfunktionen bereits geprüft wurden, um das permanente „Hin und Herspringen“ zwischen zwei Auswahlfunktionen, und damit eine Endlosschleife, zu vermeiden.

Lösungsansatz

> Bei jedem Wechsel der Auswahlfunktion, infolge des Auftretens eines Konflikts, werden die Entscheidungen, die geändert wurden gesperrt. Das heißt, tritt im weiteren Verlauf wieder eine Inkonsistenz auf, dann dürfen diese Entscheidungen nicht mehr zurückgenommen werden.

Diese Lösung wirft aber ein weiteres Problem auf:

[15] In der prototypischen Implementierung findet jedoch, aufgrund der relativ einfachen Realisierbarkeit, Methode 3 Verwendung.

[16] In der Darstellung des Algorithmus im nächsten Kapitel wird aus Gründen der Einfachheit Methode 2 benutzt.

5.4 Problem der Korrektheit

Wird durch das Setzen von Sperren das Prüfen bestimmter Auswahlfunktionen verhindert ?

Beispiel Es seien die folgenden Constraints gegeben:

$$\begin{aligned} X &: \neg C_0, \\ X &: C_1 \vee C_0 \textit{ und} \\ X &: \neg C_1 \vee C_2. \end{aligned}$$

Zuerst wird $x : \neg C_0$ zugesichert. Dann $X : C_1$, wofür $X : C_0$ eine Alternative darstellt. Die Propagierung von $X : \neg C_1$ führt dann zu einem Konflikt mit der Alternativenklasse $AK := \{\{X : C_0\}, \{X : C_2\}\}$, das heißt es gibt zwei Möglichkeiten die Inkonsistenz zu beheben. $X : C_0$ wird als Alternative zu $X : C_1$ ausgewählt, um den Konflikt zu vermeiden, das heißt $X : C_0$ wird gesperrt und propagiert. Dabei tritt erneut ein Widerspruch auf. Die einzige Möglichkeit diesen zu beseitigen, wäre die gesperrte Entscheidung $X : C_0$ zurückzunehmen, was aber aufgrund der Sperrung nicht möglich ist. Somit ließe sich eine globale Inkonsistenz herleiten, die gar nicht vorhanden ist. Also wäre der Algorithmus inkorrekt.

Lösung

> Es wird eine Tiefensuche auf den Sperren realisiert. Tritt eine Inkonsistenz auf, die sich nicht beheben läßt, dann wird zum letzten Konflikt zurückgesprungen, die „letzten“ Sperren zurückgenommen, und, falls möglich, der letzte Konflikt durch die Wahl einer anderen Alternative beseitigt.

Für das obige Beispiel würde dies bedeuten, daß nach dem Auftreten der zweiten Inkonsistenz, die Sperre für $X : C_0$ aufgehoben wird, $X : \neg C_1$ zurückgenommen und stattdessen $X : C_2$ gesperrt und propagiert wird.

6 Der Algorithmus

Im folgenden Abschnitt wird der Algorithmus informell dargestellt. Das Hauptaugenmerk liegt dabei auf der Beschreibung der Kontrollkomponente. Die Details der Verwaltung der Abhängigkeiten finden sich im Abschnitt „Formalisierung".

6.1 Darstellung

Die Propagierung geschieht relativ zu sogenannten *Umgebungen*. Eine Umgebung enthält

- eine vorhergehende *Umgebung* (außer der Anfangsumgebung),
- einen *Constraint-Speicher*, das heißt eine Menge von Constraints, die bereits zugesichert wurden und die (in Abhängigkeit der Validitätsfunktion und der Begründungsfunktion der Umgebung) entweder gültig oder ungültig sind.
- einen *Propagierungs-Stack*, der eine Menge von Constraints, die noch zugesichert werden müssen und die zugehörigen Begründungsklassen enthält, wobei die Constraints wiederum gültig oder ungültig sind.
- eine *Validitätsfunktion*,
- eine *Begründungsfunktion*,
- eine *Alternativenklasse*, das heißt eine Menge von Alternativen, die die Inkonsistenz der vorhergehenden Umgebung vermeiden.
- eine Menge von Entscheidungen, die sogenannten *Sperren*, die bezüglich dieser Umgebung gesperrt sind, das heißt, deren Alternativen kommen, im Falle einer Inkonsistenz, für die Alternativenklasse der nachfolgenden Umgebung nicht in Betracht.
- eine sogenannte *Konfliktklasse*, deren Elemente Mengen von Entscheidungen sind, die bezüglich des aktuellen Constraint-Speichers und jeder Validitätsfunktion, die diesen Entscheidungen den Wert „T" zuordnet und bei Propagierung zu einem nicht behebbaren Konflikt führen.

Jeder Umgebung ist zusätzlich ein Modus zugeordnet; der *Propagierungsmodus* oder der *Backtrackingmodus*. Die Anfangsumgebung enthält als Constraintmenge das auf Konsistenz zu prüfende Constraint-System CS und befindet sich im Propagierungsmodus.

6.1.1 Der Propagierungsmodus

Im Propagierungsmodus werden solange die Elemente des Propagierungs-Stacks zugesichert, bis dieser entweder leer ist oder ein Widerspruch auftritt.

- Tritt während der Propagierung eine Inkonsistenz auf, so wird eine neue Umgebung erzeugt. Die Alternativenklasse der neuen Umgebung ist dabei die Alternativenklasse der aktuellen Inkonsistenz, eingeschränkt bezüglich der Menge gesperrter Entscheidungen aller Umgebungen und eingeschränkt bezüglich der bereits geprüften Alternativen der Kon

sind zunächst leer, die restlichen Elemente werden aus der aktuellen Umgebung entnommen. Danach wird die neue Umgebung zur aktuellen erklärt und der Backtracking-Modus gesetzt.

- Ist der Propagierungs-Stack leer und existiert eine vorhergehende Umgebung, dann wird diese aktuell. Dabei werden der Constraint-Speicher, die Begründungsfunktion, die Validitätsfunktion und die Konfliktklasse übernommen. Die aktuellen Sperren der Entscheidungen werden nicht übernommen, das heißt die Entscheidungen werden wieder freigegeben.
- Ist der Propagierungs-Stack leer und existiert keine vorhergehende Umgebung, dann terminiert der Algorithmus mit dem Wert „syntaktisch konsistent".

6.1.2 Der Backtrackingmodus

Im Backtrackingmodus befindet sich der Constraint-Speicher zusammen mit der Validitätsfunktion und Begründungsfunktion in einem inkonsistenten Zustand. Diese Inkonsistenz muß behoben werden.

- Ist die aktuelle Umgebung die Anfangsumgebung und ist die Alternativenklasse leer, dann ist die Auflösung des Konflikts nicht mehr möglich; das heißt, daß die ursprüngliche Constraintmenge inkonsistent ist. Der Algorithmus terminiert mit dem Wert „syntaktisch inkonsistent".
- Ist die Alternativenklasse der aktuellen Umgebung leer, dann wird die Vorgängerumgebung aktuell. Die Konfliktklasse wird dabei um die Menge der Sperren aller vorhergehenden Umgebungen erweitert.
- Enthält die Alternativenklasse eine Menge alternativer Entscheidungen, dann werden diese auf den Propagierungs-Stack geschrieben und als „gesperrt erklärt". Die alten Entscheidungen werden durch Manipulation der Validitätsfunktion ungültig, womit die Inkonsistenz verschwindet.

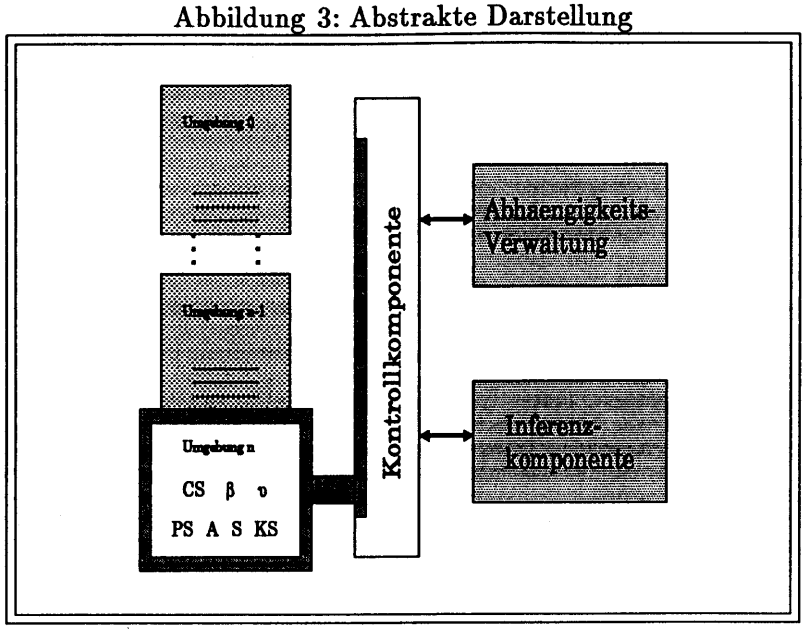
6.2 Formalisierung

In diesem Abschnitt wird der Algorithmus auf einer sehr formalen Ebene beschrieben. Zustandsänderungen werden durch eine Übergangsfunktion sowie durch Erweiterungen von partiell definierten Funktionen simuliert (vgl. [Egon Boerger, 1991]).
Der Exaktheit, die man damit erreicht, steht eine gewisse manglende Transparenz gegenüber, die aber durch die informellen Erläuterungen in den vorherigen Abschnitten ausgeglichen sein sollte.

Definition 6.1 (Entscheidung Teil 1) *Seien $ct \in C, D \subseteq C$ und $ct \in D$, dann heißt das Tripel $\langle ct, D, \{\{\}\}\rangle$ eine* **Entscheidung**. *Dabei wird ct der aktuelle Constraint und D die Menge aller Disjunktionsglieder genannt.*

Definition 6.2 *Eine* **Begründung** *ist eine endliche Menge von Entscheidungen.*
Die Menge aller Begründungen wird mit $\mathcal{B}$ notiert.

Abbildung 3: Abstrakte Darstellung



Definition 6.3 *Eine* **Begründungsklasse** *ist eine endliche Menge von Begründungen. Die Menge aller Begründungsklassen wird mit $\mathcal{BK}$ notiert.*

Definition 6.4 (Entscheidung Teil 2) *Seien $ct \in C, D \subseteq C$ und $bk \in \mathcal{BK}$, dann heißt das Tripel $\langle ct, D, bk\rangle$* **Entscheidung**. *Dabei heißt bk die Begründungsklasse der Entscheidung. Die Menge aller Entscheidungen wird mit $\mathcal{WP}$ notiert.*

Definition 6.5 *Auf den Entscheidungen sind drei Selektionsfunktionen definiert:*

$$\begin{aligned} \alpha &: \mathcal{WP} \mapsto \mathcal{C} \quad mit \quad \alpha(\langle ct, \_, \_\rangle) = ct, \\ \delta &: \mathcal{WP} \mapsto \mathcal{P}(\mathcal{C}) \quad mit \quad \delta(\langle \_, D, \_\rangle) = D \text{ und} \\ \beta_0 &: \mathcal{WP} \mapsto \mathcal{BK} \quad mit \quad \beta_0(\langle \_, \_, bk\rangle) = bk. \end{aligned}$$

Definition 6.6 *Eine Entscheidung e_1 heißt eine* **Alternative** *für eine Entscheidung e_2* **gdw** $r_1 \equiv \langle ct_1, D, bk\rangle, e_2 \equiv \langle ct_2, D, bk\rangle$ *und $ct_1 \neq ct_2$.*

Definition 6.7 *Eine* **Validitätsfunktion** *v ist eine partielle Abbildung $v : \mathcal{WP} \mapsto \mathcal{BOOL}$. Die Menge aller Validitätsfunktionen wird mit Υ bezeichnet.*

Definition 6.8 *Eine* **Begründungsfunktion** *β ist eine partielle Abbildung $\beta : \mathcal{C} \mapsto \mathcal{BK}$. Die Menge aller Begründungsfunktionen wird mit B notiert.*

Definition 6.9 Gültigkeit

1. *Eine* Entscheidung *e heißt* **gültig** *bezüglich einer Validitätsfunktion v und einer Begründungsfunktion β* **gdw** *$v(e)$ und $\exists b \in \beta_0(e) : \forall e_0 \in b : v(e_0)$.*

2. *Eine* Begründung *heißt* **gültig** *bezüglich einer Validitätsfunktion v und einer Begründungsfunktion β* gdw *jede Entscheidung der Begründung gültig ist.*

3. *Eine* Begründungsklasse *heißt* **gültig** *bezüglich einer Validitätsfunktion v und einer Begründungsfunktion β* gdw *mindestens eine Begründung der Begründungsklasse gültig ist.*

4. *Ein* Constraint *heißt* **gültig** *bezüglich einer Validitätsfunktion v und einer Begründungsfunktion β* gdw *die zugehörige Begründungsklasse gültig ist.*

Die Entscheidungen in einer Begründung sind also konjunktiv verknüpft, während die Begründungen in einer Begründungsklasse disjunktiv verknüpft sind.

Ein Entscheidung $e_1 \equiv \langle ct_1, \{ct_1, ..., ct_n\}, bk\rangle$ wird während des Inferenzprozesses immer dann kreiert, wenn eine Disjunktion $ct_1, ..., ct_n$ auftritt. Verursacht ct_1 irgendwann eine Inkonsistenz und wird e_1 zur Auflösung dieser herangezogen, dann wird e_1 für ungültig erklärt, indem eine neue Validitätsfunktion betrachtet wird, die sich von der vorherigen nur an der Stelle e_1 unterscheidet, und stattdessen eine alternative Entscheidung $e_2 \equiv \langle ct_2, \{ct_1, ..., ct_n\}, bk\rangle$ geprüft.

6.2.1 Operationen auf Begründungsklassen

Definition 6.10 *Der einstellige Operator $\triangleright$, der von einer Menge von Mengen auf eine Teilmenge von dieser abbildet, ist definiert durch:*

$$\triangleright M := \{m \in M | \nexists m_0 : m_0 \in M \text{ und } m_0 \subset m\}$$

Der Operator $\triangleright$ wird bei Vernüpfungen von Begründungsklassen benutzt, um das Ergebnis redundanzfrei zu halten.

Definition 6.11 *Die* UND-*Verknüpfung $\odot$ für Begründungsklassen ist definiert durch:*

$$bk_1 \odot bk_2 := \triangleright \{b_1 \cup b_2 | b_1 \in bk_1, b_2 \in bk_2\}$$

Die *UND*-Verknüpfung wird dazu benutzt die Begründungsklasse eines Constraints zu bestimmen, der aufgrund einer Mehrprämissenregel inferiert wurde.

Satz 6 *Seien $bk = bk_1 \odot bk_2, \beta(ct) = bk, \beta(ct_1) = bk_1$ und $\beta(ct_2) = bk_2$,*
dann gilt für jede Validitätsfunktion:

ct ist gültig gdw *ct_1 und ct_2 gültig sind.*

Definition 6.12 *Die* ODER-*Verknüpfung $\oplus$ für Begründungsklassen ist definiert durch:*

$$bk_1 \oplus bk_2 := \triangleright bk_1 \cup bk_2$$

Die *ODER*-Verknüpfung wird dazu benutzt die Begründungsklasse eines Constraints, der verschiedene Begründungen besitzt, zu erweitern.

Satz 7 *Seien $bk := bk_1 \oplus bk_2, \beta(ct) = bk, \beta(ct_1) = bk_1$ und $\beta(ct_2) = bk_2$,*
dann gilt für jede Validitätsfunktion:

ct ist gültig **gdw** *ct_1 oder ct_2 gültig ist.*

Definition 6.13 *Sei Π_{bk} die Menge aller Selektionsfunktionen bezüglich einer Begründungsklasse bk der Form:*

$$\pi : bk \longrightarrow \bigcup bk \text{ mit } \pi : b \mapsto e \in b \text{ für alle } b \in bk$$

Dann ist die „NEGATION“ $\ominus$ einer Begründungsklasse definiert durch:

$$\ominus bk := \rhd \{\{\pi b | b \in bk\} | \pi \in \Pi_{bk}\}$$

Die *„NEGATION“* findet ausschließlich bei der Bestimmung der Konfliktursache Verwendung.

Satz 8 *Sei $nbk = \ominus\, bk$, dann gilt für jede Validitätsfunktion:*

Für alle Elemente $nb \in nbk$ gilt:
Aus der Ungültigkeit aller Entscheidungen von nb folgt die Ungültigkeit von bk.

Satz 9 *Das Verknüpfungsgebilde $(\mathcal{BK}, \odot, \oplus)$ ist ein Verband.*
Dabei ist $\hat{1} := \{\{\}\}$ das Einselement bezüglich $\odot$, und $\hat{0} := \{\}$ das Nullelement bezüglich $\oplus$.

Definition 6.14 *Die n-stellige* UND*-Verknüpfung $\bigodot$ ist definiert durch:*

$$\bigodot\{bk_1, bk_2, \ldots, bk_n\} := bk_1 \odot bk_2 \odot \ldots \odot bk_n$$

6.2.2 Ordnungen auf Begründungsklassen

Definition 6.15 *Eine Begründungsklasse bk_1 heißt* **schwächer** *als eine Begründungsklasse bk_2* **gdw** *$\forall b_1 \in bk_1 : \exists b_2 \in bk_2 : b_2 \subseteq b_1$ (Notation: $bk_1 \sqsubseteq_{bk} bk_2$)*
Eine Begründungsklasse bk_1 heißt **echt schwächer** *als eine Begründungsklasse bk_2* **gdw** *$bk_1 \sqsubseteq_{bk} bk_2$ und nicht $bk_2 \sqsubseteq_{bk} bk_1$. (Notation: $bk_1 \sqsubset_{bk} bk_2$)*

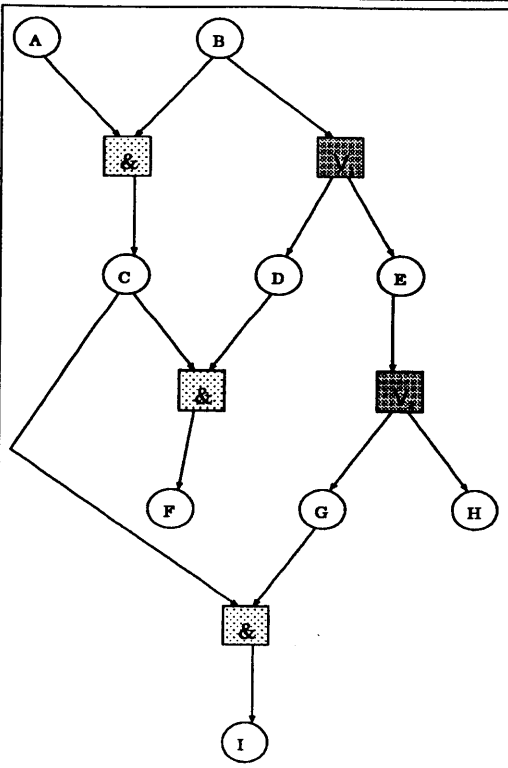
Satz 10 *Seien bk_1 und bk_2 Begründungsklassen und es gelte: $bk_1 \sqsubset_{bk} bk_2$, dann gilt für jede Validitätsfunktion: Aus der Gültigkeit von bk_1 folgt die Gültigkeit von bk_2.*

6.3 Umgebungen

Im folgenden seien

- CS $\subseteq \mathcal{C}$ eine Menge von Constraints, (Constraint-Speicher)
- PS $\subseteq \mathcal{C} \times \mathcal{P}(\mathcal{C}) \times \mathcal{BK}$, (Propagierungs-Stack)
- v eine Validitätsfunktion,
- β eine Begründungsfunktion,
- $\mathcal{A}$ ein Mengensystem von Entscheidungen, (Alternativenklasse)

Abbildung 4: Beispiel einer Begründungsfunktion



Die zugehörige Begründungsfunktion ist:

$\beta(A)$=î
$\beta(B)$=î
$\beta(C)$=î
$\beta(D)$={{ $\langle D, \{D,E\}, \hat{1}\rangle$ }}
$\beta(E)$={{ $\langle E, \{D,E\}, \hat{1}\rangle$ }}
$\beta(F)$={{ $\langle D, \{D,E\}, \hat{1}\rangle$ }}
$\beta(G)$={{ $\langle G, \{G,H\}, \{\{\langle E, \{D,E\}, \hat{1}\rangle\}\}\rangle, \langle E, \{D,E\}, \hat{1}\rangle\}\}$
$\beta(H)$={{ $\langle H, \{G,H\}, \{\{\langle E, \{D,E\}, \hat{1}\rangle\}\}\rangle, \langle E, \{D,E\}, \hat{1}\rangle\}\}$
$\beta(I)$={{ $\langle G, \{G,H\}, \{\{\langle E, \{D,E\}, \hat{1}\rangle\}\}\rangle, \langle E, \{D,E\}, \hat{1}\rangle\}\}$

Der Constraint I besitzt als Rechtfertigung die zweite Alternative von v_1 und die erste von v_2.

- $\mathcal{S} \subseteq \mathcal{WP}$ eine Menge von Entscheidungen und (Sperre)
- $\mathcal{KS}$ ein Mengensystem von Entscheidungen. (Konfliktmengensystem)

Definition 6.16 *1. Die Menge der* **Umgebungen** *U definiert durch:*

(a) $\langle p, \epsilon, CS, PS, v, \beta, A, S, KS\rangle$ *und* $\langle b, \epsilon, CS, PS, v, \beta, A, S, KS\rangle$ *sind Umgebungen.*

(b) Ist u eine Umgebung, dann sind auch $\langle p, u, CS, PS, v, \beta, A, S, KS\rangle$ *und* $\langle b, u, CS, PS, v, \beta, A, S, KS\rangle$ *Umgebungen.*

2. Sei $\underline{0}$ *die nirgends definierte Funktion.Dann heißt*

$$u_{CS} := \langle \epsilon, \{\}, \{\langle ct_1, \{ct_1\}, \hat{1}\rangle, ..., \langle ct_n, \{ct_n\}, \hat{1}\rangle\}, \underline{0}, \underline{0}, \{\}, \{\}, \{\}\rangle$$

die **Anfangsumgebung** *der Constraint-Menge* $\{ct_1, ..., ct_n\}$.

3. u_0 *heißt* **Vorgängerumgebung** *von* u_2 **gdw** $u_2 \equiv \langle \_, u_1, \_, \_, \_, \_, \_, \_, \_\rangle$ *und* $u_0 = u_1$ *oder* u_0 *ist Vorgängerumgebung von* u_1.

Auf einer abstrakten Ebene lassen sich die einzelnen Komponenten einer Umgebung wie folgt betrachten:

- Die Begründungsfunktion β steht für die Abhängigkeitsverwaltung,

- die Vorgängerumgebung, die Alternativenklasse, die Sperren und die Konfliktklasse stehen für die Kontrollkomponente und
- der Constraint-Speicher, der Propagierungs-Stack und die Validitätsfunktion stehen für die Inferenzkomponente. In diesem Kapitel wird auf diese nun nicht näher eingegangen, da diese in Kapitel 4 ausführlich untersucht wurde.

Bemerkung zur Notation

Im folgenden wird $p(\langle u, CS, PS, v, \beta, \text{A,S,KS}\rangle)$ statt $\langle p, u, CS, PS, v, \beta, \text{A,S,KS}\rangle$ und $b(\langle u, CS, PS, v, \beta, \text{A,S,KS}\rangle)$ statt $\langle b, u, CS, PS, v, \beta, \text{A,S,KS}\rangle$ geschrieben.
Dabei erscheinen die Modi „p" beziehungsweise „b" innerhalb der Umgebung u nicht mehr, da sie unwichtig sind.

In der folgenden Definition steht $\uparrow$ für undefiniert und $\downarrow$ für definiert.

Definition 6.17 Erweiterung der Begründungsfunktion
Die Funktionale $\tilde{\beta} : \mathcal{B} \times \mathcal{C} \times \mathcal{BK} \mapsto \mathcal{B}$ *und* $\hat{\beta} : \mathcal{B} \times \mathcal{WP} \mapsto \mathcal{B}$ *sind definiert durch:*

$$\tilde{\beta}(\beta, ct, bk).(x) = \begin{cases} \beta(ct) \oplus bk & : \quad x = ct \text{ und } \beta(ct) \downarrow \\ bk & : \quad x = ct \text{ und } \beta(ct) \uparrow \\ \beta(x) & : \quad \text{sonst} \end{cases}$$

Obige Definition realisiert die Erweiterung der Begründungsfunktion an der Stelle ct um die Begründungsklasse bk.

$$\hat{\beta}(\beta, e).(x) = \begin{cases} \beta(\alpha(e)) \oplus (\beta_0(e) \odot \{\{e\}\}) & : \quad x = e \text{ und } \beta(\alpha(e)) \downarrow \\ \beta_0(e) \odot \{\{e\}\} & : \quad x = e \text{ und } \beta(\alpha(e)) \uparrow \\ \beta(x) & : \quad \text{sonst} \end{cases}$$

Diese Definition realisiert die Erweiterung an der Stelle $\alpha(e)$.
Der aktuelle Constraint $\alpha(e)$ einer Entscheidung e ist genau dann gültig, wenn die Begründungsklasse $\beta_0(e)$ und die Entscheidung e gültig sind.

Definition 6.18 Erweiterung der Validitätsfunktion
Die Funktionale $\tilde{v} : \Upsilon \times \mathcal{WP} \mapsto \Upsilon$ *und* $\hat{v} : \Upsilon \times \mathcal{P}(\mathcal{WP}) \mapsto \Upsilon$ *sind definiert durch:*

$$\tilde{v}(v, e).(x) = \begin{cases} T & : \quad x = e \\ v(x) & : \quad \text{sonst} \end{cases}$$

Wird eine neue Entscheidung getroffen, dann wird die Validitätsfunktion v durch $\tilde{v}$ ersetzt, und die Entscheidung e ist fortan gültig.

$$\hat{v}(v, E).(x) = \begin{cases} NIL & : \quad \exists e \in E : x \text{ ist Alternative von } e \\ v(x) & : \quad \text{sonst} \end{cases}$$

Nach einem Wechsel der Auswahlfunktion müssen bestimmte Entscheidungen ungültig werden. Dabei gibt es immer nur eine Alternative für obige Entscheidung e, die bezüglich v gültig ist.

Definition 6.19 *Die* **Constraint-Propagierungsfunktion** $\mathcal{CP}_{\mathcal{R}} : \mathcal{P}(C) \times C \times \Upsilon \times B$ *ist definiert durch:*

$$\mathcal{CP}_{\mathcal{R}}(CS, ct, v, \beta) := \{\langle ct_1, \{ct_1, ..., ct_n\}, bk\rangle | \exists CS_0 \subseteq CS :$$

$$CS_0 \cup \{ct\} \longrightarrow_{\mathcal{R}} ct_1, ..., ct_n, \quad (1)$$

$$\forall ct_0 \in CS_0 \cup \{ct\} : ct_0 \textit{ ist gültig}, \quad (2)$$

$$\nexists CS_2 \subset CS_0 \cup \{ct\} : CS_2 \not\longrightarrow_{\mathcal{R}} ct_1, ..., ct_n, \quad (3)$$

$$bk \equiv \bigodot\{\beta(ct_0) | ct_0 \in CS_0 \cup \{ct\}\} \textit{ und} \quad (4)$$

$$\textit{für } 1 \leq i \leq n \textit{ gilt}: \beta(ct_i) \sqsubset bk\} \quad (5)$$

Die Constraint-Propagierungsfunktion $\mathcal{CP}_{\mathcal{R}}$realisiert im Prinzip die gleiche Abbildung wie die gleichnamige Funktion aus Kapitel 4. Das Ergebnis ist die Menge der gültigen Entscheidungen, die aus gültigen Constraints des Constraint-Speichers CS und dem Constraint ct, falls dieser gültig ist, abgeleitet werden können.
Die erste Zeile stellt dabei sicher, daß die Disjunktion aus $CS_0 \cup \{ct\}$ hergeleitet werden kann, die zweite Zeile garantiert, daß alle Constraints aus $CS_0 \cup \{ct\}$ gültig sind,
die dritte Zeile fordert, daß es keine Teilmenge von $CS_0 \cup \{ct\}$ gibt, aus der man bereits die Disjunktion herleiten kann, um die korrekte Berechnug der Begründungsklasse bk zu gewährleisten,
die vierte Zeile Bestimmt die Begründungsklasse bk und
die fünfte Zeile verhindert unnötige Propagierungen und fängt Zykel ab, indem gefordert wird, daß alle Begründungsklassen von bereits propagierten Disjunktionsgliedern echt schwächer sind als die Begründungsklasse bk.

Bei einer chronologischen Backtrackingstrategie ist es nicht notwendig, Constraints zu folgern, die schon im Constraint-Speicher vorhanden sind. Bei einer intelligenten Backtrackingstrategie kann dies notwendig werden, wenn dadurch die Begründungsklasse stärker wird.

Definition 6.20 *Die Menge aller* **Sperren** $\sigma(u)$ *bezüglich einer Umgebung u ist gegeben durch:*

$\sigma(u) := \{S | S$ *ist die Menge der aktuellen Sperren der Umgebung u oder eines Vorgängers von u* $\}$

Die Menge aller Sperren repräsentiert den Teil der Auswahlfunktion, der „fest“ ist.

Definition 6.21 Alternativenklasse

Die Menge der Verursacher einer Inkonsistenz $\perp$ *in Abhängigkeit einer Validitätsfunktion v und einer Begründungsfunktion* β *ist die Menge der gültigen Begründungen, die keine gesperrten Entscheidungen enthalten.*

$$KV_{v,\beta,S,u} := \{B | B \in \beta(\perp), \forall e \in B : v(e) \textit{ und } e \notin S \cup \sigma(u)\}$$

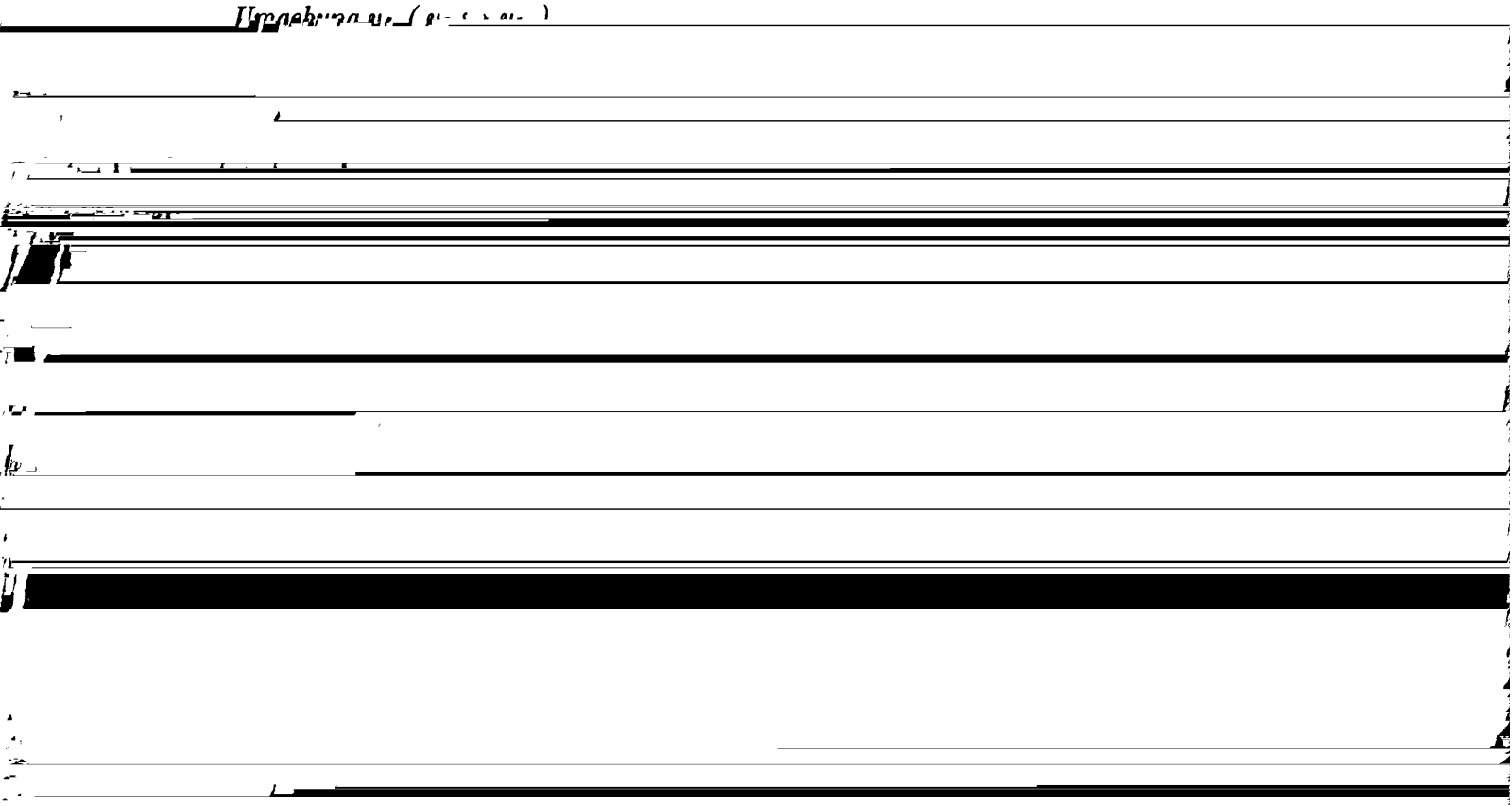
Jedes $A \in \ominus KV_{v,\beta,S,S_u}$ *ist eine minimale Menge von Entscheidungen, die ungültig werden müssen, um die Inkonsistenz* $\perp$ *zu beheben. Die Konfliktklasse KS enthält Mengen von Entscheidungen, die, wenn sie alle gültig sind, zu einer nicht behebbaren Inkonsistenz führen*[17]*. Deswegen muß man diejenigen A nicht prüfen, die zusammen mit den aktuellen Sperren und der Menge aller Sperren der Vorgängerumgebung eine Obermenge eines Elements der Konfliktklasse bilden.*

$$\tilde{A}_{v,\beta,u,S,KS} := \{A | A \in \ominus KV_{v,\beta,S,u} \textit{ und } \nexists k \in KS : k \subseteq A \cup S \cup \sigma(u)\}$$

[17] Im Bereich der Truth-Maintenance-Systeme nennt man die Konfliktklasse die „Nogoods“.

Definition 6.22 Folgeumgebung

1. *Sei* $\uplus$ *die disjunkte Vereinigung. Eine Umgebung* u_1 *heißt* **direkte Folgeumgebung** *einer*



(a) Konfliktregel

$$\frac{p(\langle u, CS \uplus \{\bot\}, PS, v, \beta, AK, S, KS\rangle)}{b(\langle\langle u, CS, PS, v, \beta, AK, S, KS\rangle, CS, \{\}, v, \beta, \tilde{A}_{v,\beta,u,S,KS}, S, KS, \{\}, KS\rangle)}$$

Die Konfliktregel bewirkt aufgrund der Entdeckung einer Inkonsistenz den Übergang vom Propagierungsmodus zum Backtrackingmodus. Dabei wird eine neue Umgebung erzeugt, deren Alternativenklasse $\tilde{A}_{v,\beta,u,S,KS}$ gerade diejenige der aktuellen Inkonsistenz ist.

(b) Rücksprungregel

$$\frac{p(\langle\langle u, \_, PS, \_, \_, \_, AK, S, \_\rangle, CS, \{\}, v, \beta, \_, \_, KS\rangle)}{p(\langle u, CS, PS, v, \beta, AK, S, KS\rangle)}$$

Die Rücksprungregel bewirkt ein Zurückspringen zur Vorgängerumgebung, um den dortigen Propagierungs-Stack abzuarbeiten.

(c) Constraint-Propagierungsregel

$$\frac{p(\langle u, CS, PS \uplus \{\langle ct_1, \{ct_1\}, bk\rangle\}, v, \beta, AK, S, KS\rangle)}{p(\langle u, CS \cup \{ct_1\}, PS \cup \mathcal{CP}_{\mathcal{R}}(CS, ct_1, v, \tilde{\beta}_{\beta,ct_1,bk}), v, \tilde{\beta}_{\beta,ct_1,bk}, AK, S, KS\rangle)}$$

Die Constraint-Propagierungsregel propagiert einen Constraint ct_1, für den die Begründungsfunktion β um die Begründungsklasse bk erweitert wird.

(d) Entscheidungs-Propagierungsregel

$$\frac{p(\langle u, CS, PS \uplus \{e\}, v, \beta, AK, S, KS\rangle)}{p(\langle u, CS \cup \{\alpha(e)\}, PS \cup \mathcal{CP}_{\mathcal{R}}(CS, \alpha(e), \tilde{v}_{v,e}, \hat{\beta}_{\beta,e}), \tilde{v}_{v,e}, \hat{\beta}_{\beta,e}, AK, S, KS\rangle)}$$

Die Entscheidungs-Propagierungsregel übernimmt vom Propagierungs-Stack eine Entscheidung, erweitert für diese die Validitätsfunktion, das heißt e wird für gültig er-

2. *Eine Umgebung u_n heißt* **Folgeumgebung** *einer Umgebung u_0 ($u_0 \overset{*}{\hookrightarrow} u_n$), falls Umgebungen $u_1, ..., u_{n-1}$ existieren mit $u_0 \hookrightarrow u_1 \hookrightarrow \ldots \hookrightarrow u_n$.*

3. *$p(\langle \epsilon, -, \{\}, -, -, -, -, - \rangle)$ heißt* **positive Endumgebung** *und $b(\langle\langle \epsilon, -, -, -, -, -, -, - \rangle, -, -, -, -, \{\}, -, - \rangle)$ heißt* **negative Endumgebung.**
Die Endumgebungen sind Umgebungen, auf die man keine Regel anwenden kann.

Definition 6.23 *Der* **Konsistenztest** $K_{\mathcal{R}}$ *ist definiert durch:*

$$K_{\mathcal{R}}(CS) = \begin{cases} \text{„syntaktisch konsistent“} & : \quad u_{CS} \overset{*}{\hookrightarrow} p(\langle \epsilon, -, \{\}, -, -, -, -, - \rangle) \\ \text{„syntaktisch inkonsistent“} & : \quad u_{CS} \overset{*}{\hookrightarrow} b(\langle\langle \epsilon, -, -, -, -, -, -, - \rangle, -, -, -, -, \{\}, -, - \rangle) \end{cases}$$

Satz 11 Terminierung
Sei $\mathfrak{R}$ ein bezüglich C zulässiges Regelsystem. Dann ist $K_{\mathcal{R}}(CS)$ definiert für alle $CS \subseteq C$.

Beweis-Skizze

1. Es gelte $u_{CS} \overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} u$. Dann folgt aus der „Regeldefinition“, daß u entweder Endumgebung ist oder eine Regel auf u anwendbar ist.

2. Es läßt sich eine wohlfundierte Ordnung $\sqsubseteq_u$ auf U angeben.
Für jede Folge $u_1 \overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} u_2$ mit $u_{CS} \overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} u$ gilt dann:
$u_2 \sqsubseteq_u u_1$.
Diese Ordnung berücksichtigt folgende Größen, die in der Reihenfolge ihrer Priorität wiedergegeben sind:

   Größe der Konfliktklasse,
   Anzahl der bereits geprüften Alternativen,
   Anzahl aller Sperren,
   Anzahl der Vorgängerumgebungen und
   eine Ordnung auf Constraints, die existiert, falls das zugehörige Regelsystem zulässig ist, das heißt die Menge der herleitbaren Constraints endlich ist. □

Satz 12 Vollständigkeit
Aus $K_{\mathcal{R}}(CS)$ = „syntaktisch konsistent“ folgt die Konsistenz der Constraintmenge CS.

Beweis-Skizze

Seien $PS_g^*(u)$ die Menge aller gültigen aktuellen Constraints, die auf dem Propagierungsstack der Umgebung u, oder auf dem Propagierungsstack eines Vorgängers von u liegen und $CS_g(u)$ die Menge der gültigen Constraints der Umgebung u.
Dann läßt sich zeigen, daß für jede Umgebung u mit $u_{CS} \overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} u$ gilt:

$$\begin{aligned} CS &\subseteq CS_g(u) \cup PS_g^*(u) \\ CS_g(u) \cup \{ct | CS_g(u) \longrightarrow_{\mathcal{R}} ct\} &\subseteq CS_g(u) \cup PS_g^*(u) \end{aligned}$$

Dann gilt dies auch für eine positive Endumgebung u_e mit $PS_g^*(u_e) = \{\}$. Daraus folgt: Aus $CS_g(u_e)$ läßt sich nichts mehr ableiten, und da $CS \subseteq CS_g(u_e)$ ist, ist $CS_g(u_e)$ ein Modell für CS und CS ist somit konsistent. □

Satz 13 Korrektheit
Aus $K_{\mathcal{R}}(CS)$ = „syntaktisch inkonsistent" folgt die Inkonsistenz der Constraintmenge CS.

Beweis-Skizze

Die Constraintmenge CS sei konsistent. Dann existiert eine Auswahlfunktion $\mathcal{A}$ sodaß $\perp$ nicht hergeleitet werden kann.
Sei u mit $u_{CS} \overset{*,\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} u \overset{\mathcal{A}}{\Longrightarrow}_{k_{\mathcal{R}}} u'$ die erste Umgebung, die eine Inkonsistenz enthält – falls so ein u nicht existiert, terminiert der Algorithmus mit dem Wert „syntaktisch konsistent". Dann enthält die Alternativenklasse von u' auch eine Menge von Entscheidungen E, die eine Teilauswahl von $\mathcal{A}$ darstellen. Wird nun in der Folge eine andere Alternative geprüft und ein Modell gefunden, dann terminiert der Algorithmus mit dem Wert „syntaktisch konsistent". Ist dies nicht der Fall, wird irgendwann die Alternative E geprüft, und deren Elemente gesperrt. Dieser Prozeß läßt sich solange weiterführen bis die vollständige Auswahlfunktion $\mathcal{A}$ bezüglich derer man keine Inkonsistenz herleiten kann, aufgebaut ist. □

6.4 Beispiel

Die Konsistenz der Constraintmenge CS :=$\{a : \neg C_2, a : C_1 \vee C_2, a : C_3 \vee C_4, a : \neg C_1 \vee \neg C_2\}$ soll geprüft werden.
Die Validitätsfunktion wird beschrieben durch eine Menge. Diese Menge enthält genau die Entscheidungen, die gültig sind. Die Begründungsfunktion wird ebenfalls durch eine Menge dargestellt. Dabei steht $ct \mapsto bk$ für $\beta(ct) = bk$.
Die einzelnen Komponenten werden nur dann aufgeführt, wenn sie sich gegenüber der letzten Umgebung geändert haben.

Initialisierung

Der Propagierungs-Stack enthält die Constraints in der korrekten Syntax.

$$
\begin{aligned}
m_0 &= p \\
z_0 &= \epsilon \\
CS_0 &= \{\} \\
PS_0 &= \{\langle a : \neg C_2, \{a : \neg C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_1 \vee C_2, \{a : C_1 \vee C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_3 \vee C_4, \{a : C_3 \vee C_4\}, \hat{1}\rangle, \\
&\quad \langle a : \neg C_1 \vee \neg C_2, \{a : \neg C_1 \vee \neg C_2\}, \hat{1}\rangle \\
&\quad \} \\
\upsilon_0 &= \{\} \\
\beta_0 &= \{\} \\
A_0 &= \{\} \\
S_0 &= \{\} \\
KS_0 &= \{\}
\end{aligned}
$$

Constraint-Propagierungsregel
$a : \neg C_2$ wird propagiert und die Begründungsfunktion wird aktualisiert.

$$
\begin{aligned}
CS_1 &= \{a : \neg C_2\} \\
PS_1 &= \{\langle a : C_1 \vee C_2, \{a : C_1 \vee C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_3 \vee C_4, \{a : C_3 \vee C_4\}, \hat{1}\rangle, \\
&\quad \langle a : \neg C_1 \vee \neg C_2, \{a : \neg C_1 \vee \neg C_2\}, \hat{1}\rangle\} \\
\beta_1 &= \{a : \neg C_2 \mapsto \hat{1}\}
\end{aligned}
$$

Constraint-Propagierungsregel
$a : C_1 \vee C_2$ wird propagiert und die Begründungsfunktion wird aktualisiert.

$$
\begin{aligned}
CS_2 &= CS_1 \cup \{a : C_1 \vee C_2\} \\
PS_2 &= \{\langle a : C_3 \vee C_4, \{a : C_3 \vee C_4\}, \hat{1}\rangle, \\
&\quad \langle a : \neg C_1 \vee \neg C_2, \{a : \neg C_1 \vee \neg C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle\} \\
\beta_2 &= \beta_1 \cup \{a : C_1 \vee C_2 \mapsto \hat{1}\}
\end{aligned}
$$

Constraint-Propagierungsregel
$a : C_3 \vee C_4$ wird propagiert und die Begründungsfunktion wird aktualisiert.

$$
\begin{aligned}
CS_3 &= CS_2 \cup \{a : C_3 \vee C_4\} \\
PS_3 &= \{\langle a : \neg C_1 \vee \neg C_2, \{a : \neg C_1 \vee \neg C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle\} \\
\beta_3 &= \beta_2 \cup \{a : C_3 \vee C_4 \mapsto \hat{1}\}
\end{aligned}
$$

Constraint-Propagierungsregel
$a : \neg C_1 \vee \neg C_2$ wird propagiert und die Begründungsfunktion wird aktualisiert.

$$
\begin{aligned}
CS_4 &= CS_3 \cup \{a : \neg C_1 \vee \neg C_2\} \\
PS_4 &= \{\langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle, \\
&\quad \langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle, \\
&\quad \langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
\beta_4 &= \beta_3 \cup \{a : \neg C_1 \vee \neg C_2 \mapsto \hat{1}\}
\end{aligned}
$$

Entscheidungs-Propagierungsregel
$a : C_1$ wird propagiert, die Entscheidung $\langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle$ für gültig erklärt und β aktualisiert.

$$
\begin{aligned}
PS_5 &= CS_4 \cup \{a : C_1\} \\
CS_5 &= \{\langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle, \\
&\quad \langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
\upsilon_5 &= \{\langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle\} \\
\beta_5 &= \beta_4 \cup \{a : C_1 \mapsto \\
&\qquad \{\{\langle a : C_1, \\
&\qquad\quad \{a : C_1, a : C_2\}, \\
&\qquad\quad \hat{1}\}\}\}
\end{aligned}
$$

Entscheidungs-Propagierungsregel
$a : C_3$ wird propagiert, die Entscheidung $\langle a : C_1, \{a : C_3, a : C_4\}, \hat{1}\rangle$ für gültig erklärt und β aktualisiert.

$$
\begin{aligned}
PS_6 &= PS_5 \cup \{a : C_3\} \\
CS_6 &= \{\langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle \\
&\quad\ \} \\
v_6 &= v_5 \cup \{\langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle\} \\
\beta_6 &= \beta_5 \cup \{a : C_3 \mapsto \\
&\qquad \{\{\langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle\}\}\}
\end{aligned}
$$

Entscheidungs-Propagierungsregel
$a : \neg C_1$ wird propagiert, die Entscheidung $\langle a : C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle$ für gültig erklärt und β aktualisiert.

$$
\begin{aligned}
CS_7 &= CS_6 \cup \{a : \neg C_1\} \\
PS_7 &= \{\langle \bot, \\
&\quad \{\bot\}, \\
&\quad \{\{\langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle \\
&\qquad \langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle \\
&\quad \}\}\rangle\} \\
v_7 &= v_6 \cup \{\langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
\beta_7 &= \beta_6 \cup \{a : \neg C_1 \mapsto \\
&\qquad \{\{\langle a : \neg C_1, \\
&\qquad\ \{a : \neg C_1, a : \neg C_2\}, \\
&\qquad\ \hat{1}\rangle\}\}\}
\end{aligned}
$$

Konfliktregel
Die Inkonsistenz des Constraint-Speichers wird festgestellt und die Begründungsfunktion erweitert.

$$
\begin{aligned}
CS_8 &= CS_7 \cup \{\bot\} \\
PS_8 &= \{\} \\
\beta_8 &= \beta_7 \cup \{\bot \mapsto \\
&\qquad \{\{\langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle, \\
&\qquad\ \langle a : \neg C_1, \\
&\qquad\ \{a : \neg C_1, a : \neg C_2\}, \\
&\qquad\ \hat{1}\rangle\}\}\}
\end{aligned}
$$

---

trial-Regel
Die Alternativenklasse enthält die potentiellen Alternativen zur Vermeidung der Inkonsistenz $a : C_1$ und $a : \neg C_1$.

$$
\begin{aligned}
m_9 &= b \\
z_9 &= \langle z_0, CS_7, PS_8, v_8, \beta_8, A_0, S_0, KS_0\rangle \\
CS_9 &= CS_7 \\
PS_9 &= \{\} \\
v_9 &= v_8 \\
\beta_9 &= \beta_8 \\
A_9 &= \{\{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}, \\
&\qquad \{\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\}\} \\
S_9 &= \{\} \\
KS_9 &= KS_0
\end{aligned}
$$

Entscheidungs-Propagierungsregel
Die Alternative $\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle$ wird auf den Propagierungs-Stack geschrieben, gesperrt und ...

$$
\begin{aligned}
m_{10} &= p \\
PS_{10} &= \{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\} \\
v_{10} &= \{\langle a : \neg C_2, \{a : \neg C_2\}, \hat{1}\rangle, \\
&\qquad \langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle, \\
&\qquad \langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle \\
&\qquad \} \\
A_{10} &= \{\{\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\}\} \\
S_{10} &= \{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}
\end{aligned}
$$

Constraint-Propagierungsregel
propagiert.

$$
\begin{aligned}
CS_{11} &= CS_9 \cup \{a : C_2\} \\
PS_{11} &= \{\langle \bot, \\
&\qquad \{\bot\}, \\
&\qquad \{\{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}\}\rangle\} \\
v_{11} &= v_{10} \cup \{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\} \\
\beta_{11} &= \beta_9 \cup \{a : C_2 \mapsto \\
&\qquad \{\{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}\}\}
\end{aligned}
$$

Konflikt-Regel
Dadurch entsteht erneut eine Inkonsistenz,

$$
\begin{aligned}
CS_{12} &= CS_{11} \cup \{\bot\} \\
PS_{12} &= \{\} \\
\beta_{12} &= \beta_{10} \cup \{\bot \mapsto \\
&\qquad \{\{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}\}\}
\end{aligned}
$$

---

trial-Regel
die nicht mehr zu beheben ist.

$$
\begin{aligned}
m_{13} &= b \\
z_{13} &= \langle z_9, CS_{11}, PS_{12}, v_{11}, \\
&\quad \beta_{12}, A_{10}, S_{10}, KS_0 \rangle \\
CS_{13} &= CS_{11} \\
PS_{13} &= \{\} \\
v_{13} &= v_{12} \\
\beta_{13} &= \beta_{12} \\
A_{13} &= \{\} \\
S_{13} &= \{\} \\
KS_{13} &= KS_0
\end{aligned}
$$

error-Regel
Infolgedessen wird „zurückgesprungen“ und aus der Alternativenklasse A_9 die zweite Alternative geprüft.

$$
\begin{aligned}
z_{14} &= z_9 \\
CS_{14} &= CS_7 \\
PS_{14} &= \{\} \\
v_{14} &= v_7 \\
\beta_{14} &= \beta_8 \\
A_{14} &= \{\{\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\}\} \\
S_{14} &= \{\} \\
KS_{14} &= \{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}
\end{aligned}
$$

trial-Regel
Die Alternative $\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle$ wird auf den Propagierungs-Stack geschrieben, gesperrt und ...

$$
\begin{aligned}
m_{15} &= p \\
CS_{15} &= CS_{13} \\
PS_{15} &= \{\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
v_{15} &= \{\langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle, \\
&\quad \langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle \\
&\quad \} \\
\beta_{15} &= \beta_{14} \\
A_{15} &= \{\} \\
S_{15} &= \{\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
KS_{15} &= KS_{14}
\end{aligned}
$$

Entscheidungs-Propagierungsregel
propagiert.

$$
\begin{aligned}
CS_{16} &= CS_{15} \cup \{a : \neg C_2\} \\
PS_{16} &= \{\} \\
v_{16} &= v \cup \{\langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
\beta_{16} &= \beta_8 \cup \{a : \neg C_2 \mapsto \hat{1}\}
\end{aligned}
$$

Rücksprungregel
Danach ist der Propagierungs-Stack leer und Vorgängerumgebung wird aktiv, wobei der Constraint-Speicher, die Validitätsfunktion, die Begründungsfunktion und die Konfliktklasse übernommen werden.

$$
\begin{aligned}
z_{17} &= \epsilon \\
CS_{17} &= \{a : \neg C_2, a : C_1 \vee C_2, a : C_3 \vee C_4, \\
&\quad a : \neg C_1 \vee \neg C_2, a : C_1, a : C_3, a : \neg C_1\} \\
PS_{17} &= PS_8 = \{\} \\
v_{17} &= \{\langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle, \\
&\quad \langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle, \\
&\quad \langle a : \neg C_2, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
\beta_{17} &= \{a : \neg C_2 \mapsto \hat{1}, a : C_1 \vee C_2 \mapsto \hat{1}, \\
&\quad a : C_3 \vee C_4 \mapsto \hat{1}, a : \neg C_1 \vee \neg C_2 \mapsto \hat{1}, \\
&\quad a : C_1 \mapsto \langle a : C_1, \{a : C_1, a : C_2\}, \hat{1}\rangle, \\
&\quad a : C_3 \mapsto \langle a : C_3, \{a : C_3, a : C_4\}, \hat{1}\rangle, \\
&\quad a : \neg C_1 \mapsto \langle a : \neg C_1, \{a : \neg C_1, a : \neg C_2\}, \hat{1}\rangle\} \\
A_{17} &= A_8 = \{\} \\
S_{17} &= S_8 = \{\} \\
KS_{17} &= \{\langle a : C_2, \{a : C_1, a : C_2\}, \hat{1}\rangle\}
\end{aligned}
$$

Obige Umgebung ist eine positive Endumgebung.

„konsistent“

Die Menge der gültigen Constraints des Constraint-Speichers CS_{17} in Abhängigkeit von v_{17} und β_{17} $\{a : \neg C_2, a : C_1 \vee C_2, a : C_3 \vee C_4, a : \neg C_1 \vee \neg C_2, a : C_1, a : C_3\}$ ist ein Modell für die ursprüngliche Constraintmenge CS.

6.5 Die inkrementelle Schnittstelle

Die Realisierung einer inkrementellen Schnittstelle ist sehr einfach, denn es gilt:

Satz 14 *Seien* $u_{CS} \overset{*,A}{\Longrightarrow}_{k_R} u_{e1}$ *und* $u_{e1} \equiv \langle u, CS^*, PS, v, \beta, AK, S, KS\rangle$ *Endumgebung, dann ist* $CS \cup \{ct\}$ *konsistent* **gdw** $\langle u, CS^*, \{\langle ct_1, \{ct_1\}, \hat{1}\rangle\}, v, \beta, AK, S, KS\rangle \overset{*,A}{\Longrightarrow}_{k_R} u_{e2}$, *wobei* u_{e2} *positive Endumgebung ist.*

Ist u_{e1} eine positive Endumgebung, dann ist der Progierungs-Stack PS leer.

6.6 Fakten und Annahmen

Der Algorithmus gestattet neben der Repräsentation von Constraints, die immer gültig sind und im folgenden Fakten genannt werden, auch die Repräsentation von Constraints, deren Validität man auch außerhalb des Konsistenztestes, oder sogar nur außerhalb dessen, ändern kann. Diese Constraints werden Annahmen genannt. Ohne den Algorithmus erweitern zu müssen ist es möglich die Annahmen in den Inferenzprozeß zu integrieren, indem die Annahmen als „Disjunktionsglieder ohne Alternative“ geführt werden.

Definition 6.24 *Sei ct ein Constraint. Dann heißt das Tripel* $\langle ct, \{ct\}, \hat{1}\rangle$ *eine* **Annahme.**

Definition 6.25 *Eine Annahme A heißt bezüglich einer Validitätsfunktion v gültig* **gdw** $v(A)$.

Der Definitionsbereich der Validitätsfunktion muß nicht erweitert werden, da die Menge der Annahmen eine Teilmenge von $\mathcal{WP}$ ist.

Satz 15 *Seien CS eine Menge von Fakten, A eine Menge von Annahmen und v eine Validitätsfunktion, die für alle Annahmen aus A definiert ist, dann gilt :*

$$p(\langle \epsilon, \{\}, PS_F \cup PS_A, v, \underline{0}, \{\}, \{\}, \{\}\rangle) \overset{*,A}{\Longrightarrow}_{k_R} u_{pos}$$

gdw

$$CS \cup \{ct | \langle ct, \{ct\}, \hat{1}\rangle \in A \text{ und } v(\langle ct, \{ct\}, \hat{1}\rangle)\} \text{ konsistent ist, wobei}$$

$PS_F := \{\langle ct, \{ct\}, \hat{1}\rangle | ct \in CS\}$,
$PS_A := \{\langle ct, \{ct\}, \{\{\langle ct, \{ct\}, \hat{1}\rangle\}\}\rangle | \langle ct, \{ct\}, \hat{1}\rangle \in A\}$ *und*
u_{pos} *eine positive Endumgebung ist.*

Das heißt, der Algorithmus terminiert mit dem Wert „syntaktisch konsistent“ **gdw** *die Vereinigung der Menge der Fakten und der Menge der gültigen Annahmen konsistent ist.*

Im verein mit einer inkrementellen Schnittstelle stellt die Möglichkeit Annahmen zu treffen, und diese wieder fallen zu lassen, ein mächtiges Werkzeug dar.

Beispielsweise lassen sich Widerspruchsbeweise in geeigneter Weise führen:

> Die Wissensbasis wird repräsentiert durch Fakten. Dabei erscheint es sinnvoll, diese auf Konsistenz zu prüfen, da man aus einer inkonsistenten Wissensbasis „alles" herleiten kann. Im Anschluss an den Konsistenztest hat man, aufgrund der inkrementellen Schnittstelle, Zugang zu dem noch vorhandenen Modell der Wissensbasis. Soll nun per Widerspruchsbeweis gezeigt werden, daß der Constraint ct aus der Wissensbasis gefolgert werden kann, dann wird das Komplement kct des Constraints als gültige Annahme eingeführt, und mit dem noch vorhandenen Modell auf Konsistenz getestet. Unabhängig vom Ergebnis des Konsistenztests wird im Anschluß die Annahme, und damit auch ihre Folgerungen, für ungültig erklärt.

Findet der Algorithmus ein Modell, dann ist es offensichtlich, daß dieses Modell weiter benutzt werden kann. Ist dies auch so, wenn eine globale Inkonsistenz auftritt ? Ja, denn der Algorithmus kehrt, im Falle einer globalen Inkonsistenz, bis zur ersten Inkonsistenz zurück, die in diesem Fall durch die Annahme verursacht wurde. Wird nun die Annahme für ungültig erklärt, verschwindet die Inkonsistenz, und man hat das gleiche Modell wieder wie zu Beginn des Konsistenztests.

7 Mögliche Erweiterungen des Ansatzes

In Kapitel 4 wurde ein Regelformalismus angegeben mit dem sich die operationale Semantik von ALC beschreiben läßt.
Für diesen Formalismus wurde ein Interpreter entwickelt, der eine intelligente Backtrackingstrategie realisiert. Im folgenden wird beschrieben werden, wie man den Formalismus und den Interpreter erweitern muß, so daß sich damit auch die operationale Semantik von **Taxon** darstellen läßt.

In **Taxon** besteht gegenüber ALC noch die Möglichkeit die Gleichheit von Termen zu fordern und die Konsistenz einer Menge sogenannter konkreter Prädikate durch ein externes Entscheidungsverfahren prüfen zu lassen.

7.1 Gleichheit von Termen

Es bieten sich zwei Möglichkeiten an, die Gleichheit von Termen zu realisieren.

1. Auf der Regelebene wird eine transitive und symmetrische Relation beschrieben,

$$\begin{aligned} gleich(Obj_1, Obj_2) &\rightsquigarrow gleich(Obj_2, Obj_1) \\ gleich(Obj_1, Obj_2), gleich(Obj_2, Obj_3) &\rightsquigarrow gleich(Obj_1, Obj_3) \end{aligned}$$

   deren „Reflexivität" dadurch simuliert wird, daß für jede Regel mit mehreren Prämissen die gleiche Regel modulo Gleichheit eingeführt wird.

$$\begin{aligned} cterm(Obj,\ forall(Role,\ C)),\ role(Obj,\ Role,\ Obj_2) &\rightsquigarrow cterm(Obj_2,\ C) \\ cterm(Obj,\ forall(Role,\ C)),\ role(Obj_1,\ Role,\ Obj_2),\ gleich(Obj, Obj_1) &\rightsquigarrow cterm(Obj_2,\ C) \end{aligned}$$

   Nachteil:

   (a) Der „Zykeltest" innerhalb der Constraint-Propagierungsfunktion $\mathcal{CP}_{\mathcal{R}}$ muß durchgeführt werden.

   (b) Die Repräsentation der Gleichheit als Menge von einzelnen Gleichheitsconstraints ist nicht besonders effektiv.

2. Der Regelformalismus wird erweitert durch die Einführung eines speziellen Constraintnamens „=", der die Menge der Formeln wie folgt vergrößert:

   Sind t_1 und t_2 Terme, dann ist $t_1 = t_2$ eine Formel.

   Der Regelinterpreter interpretiert das spezielle Symbol „=" als Äquivalenzrelation.
   Nützlich erscheint dabei:

   (a) die Gleichheitsconstraints an eine spezielle Komponente zu übergeben, die diese geeignet repräsentiert und in Abhängigkeit einer Begründungsfunktion, auf Anfrage die Begründungsklasse eines Gleichheitsconstraints $obj_1 = obj_2$ berechnet und

   (b) die Gleichheitsconstraints mit in die Begründungsklasse aufzunehmen. Dies hat den Vorteil, daß wenn weitere Begründungen die Gleichheitsbeziehung $obj_1 = obj_2$ rechtfertigen, die Begründungsklasse des Constraints, und all seiner Folgerungen, nicht erweitert werden muß.

7.2 Konkrete Prädikate

Das größte Problem bei konkreten Prädikaten ist, daß das externe Entscheidungsverfahren ebenfalls die Abhängigkeiten verwalten muß. Ist nämlich die Menge der konkreten Prädikate inkonsistent, dann müssen diejenigen Prädikate bestimmt werden, die den Konflikt verursachen, und der Kontrollkomponente von **Taxon** übergeben werden.
Auch das Zurücknehmen bestimmter Entscheidungen, und die damit verbundene Ungültigkeit bestimmter konkreter Prädikate muß für das Entscheidungsverfahren transparent sein.

Am einfachsten ist, sofern dies möglich ist, aus dem externen Entscheidungsverfahren ein internes zu machen, das heißt die operationale Semantik des Entscheidungsverfahrens mit Hilfe der hier vorgestellten Regeln zu beschreiben, und die Konsistenz einer Menge von Prädikaten auf den Konsitenztest $K_{\mathcal{R}}$ zurückzuführen.

Für das zum „Lieferumfang" von **Taxon** gehörende Entscheidungsverfahren für Ordnungen auf reellen Zahlen ist dies problemlos möglich.

8 Verwandte Arbeiten

In den folgenden zwei Abschnitten wird kurz die Verwandschaft zu den Gebieten Truth-Maintenance-Systeme und Logische Programmierung aufgezeigt.

8.1 Truth-Maintenance-Systeme

Zu Beginn der Arbeit war klar, daß die Verwaltung der Abhängigkeiten und die Schlußfolgerungsmechanismen zu denen ähnlich sind, die in Truth-Maintenance-Systemen Verwendung finden. Jedoch sind die Probleme, die man mit einem Truth-Maintenance-System lösen will, und damit die Anforderungen an diese, von denen verschieden, die wir hier betrachtet haben.

Die beiden bekanntesten Vertreter der Truth-Maintenance-Systeme sind das Justification based TMS (JTMS) von Doyle[18] und das Assumption based TMS (ATMS) von de Kleer[19]. Der wesentliche Unterschied zu einem JTMS ist, daß dieses in der Lage ist,

- Nicht-Monotonie zu behandeln jedoch
- keine Disjunktionen direkt repräsentieren kann.

Zwar ist aufgrund der Disjunktionen auch in dem hier beschriebenen Konzept eine gewisse Nicht-Monotonie vorhanden, diese ist jedoch nicht nach außen transparent, da ein Modell einer Menge von Constraints aufgebaut werden soll, um deren Konsistenz zu zeigen, und nicht um aus diesem Modell irgendwelche Schlüsse zu ziehen. Die Behandlung der Nicht-Monotonie in einem JTMS erfordert einen erheblichen Aufwand, so daß es nicht adäquat erschien, die für unsere Anwendung sehr wichtigen Disjunktionen durch ein JTMS zu simulieren.

Ein ATMS ist dazu konzipiert mehrere sogenannte Kontexte parallel zu verwalten. Um ein Modell für eine Menge von Constraints zu finden, ist dies jedoch nicht notwendig. Also konnte auch hier zusätzlicher Aufwand eingespart werden.

8.2 Intelligentes Backtracking in PROLOG

In der Logischen Programmierung versteht man unter Intelligentem Backtracking, das was hier unter dem Namen „Abhängigkeitsgesteuertes Backtracking“ vorgestellt wurde, das heißt das Zurückspringen zu der letzten Entscheidung, die am Konflikt beteiligt ist.

Backtracking in PROLOG findet gebenüber Terminologischen Logiken unter gänzlich anderen Voraussetzungen statt:

- PROLOG-Systeme sind Rückwärtsregelsysteme, das heißt die Abarbeitung ist zielgerichtet. Implizites Wissen das nicht zum Beweis eines „goals“ benötigt wird, wird nicht abgeleitet; im Gegensatz zu Vorwärtsregelsystemen, die vielfach Widerspruchsbeweiser sind

- Die Ableitungsketten werden im allgemeinen sehr lang. Dies liegt insbesondere daran, daß die rekursive Regeldefinition in PROLOG nicht nur erlaubt, sondern ein fundamentales Konzept dieser Sprache darstellt. In terminologische Logiken sind sogenannte zyklische Definitionen der Konzeptterme nicht gestattet.

Dies führt dazu, daß die Methoden, die entwickelt wurden, um im Bereich der Logischen Programmierung „abhängigkeitsgesteuert backtracken" zu können, für diese Arbeit nur eine untergeordnete Rolle spielen. Der interessierte Leser sei auf die Literatur[20] verwiesen.

[20][P.Codognet, 1990], [Malhorta, 1990] und [M.Bruynooghe, 1991]

9 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Konzept zur Verwaltung von Abhängigkeiten entwickelt. Dieses wurde in einen Algorithmus integriert, der ein Modell für eine Menge von Constraints sucht, und mit dem Wissen über die Abhängigkeiten eine intelligente Backtrackingstrategie realisiert.

Die wesentlichen Merkmale des Algorithmus sind:

- direkte Repräsentation von Disjunktionen (im Gegensatz zu den Truth-Maintenance-Systemen),
- Konzentration auf die Entscheidungen bei der Verwaltung der Abhängigkeiten,
- die Existenz einer inkrementellen Schnittstelle und
- die Möglichkeit zur Repräsentation von Fakten und Annahmen.

Um die theoretisch zu erwartende Effizienzsteigerung bei terminologischen Inferenzsystemen zu überprüfen, wurde eine prototypische Implentierung erstellt. Die zugrundeliegende terminologische Sprache war dabei ALC.

> Die Ergebnisse übertrafen sogar die Erwartungen, zumindest für getesteten Beispiele. Der zusätzliche Aufwand für die Verwaltung der Abhängigkeiten war sehr gering. Auch wenn keine Disjunktionen auftraten, war der „intelligente“ Algorithmus nur unwesentlich langsamer als der, der eine chronologische Backtrackingstrategie verfolgt. Allerdings muß man sagen, daß ALC nicht repräsentativ ist. Die **ABOX**en und **TBOX**en, die in Anwendungen benutzt werden, verlangen jedoch die volle Funktionalität des terminologischen Inferenzsystems **Taxon** und stehen deswegen als Testbeispiele noch nicht zur Verfügung.

Zukünftige Arbeiten

- Zu Überprüfen wäre die Einsetzbarkeit der Algorithmen in anderen Bereichen, in denen naturgemäß Disjunktionen vorkommen wie zum Beispiel Planen.
- Auf dem Gebiet der terminologischen Logiken wäre zu klären, wie sich die Algorithmen zur Realisierung der Inferenzdienste ändern, unter Benutzung eines Konsistenztests, der eine inkrementelle Schnittstelle besitzt und die Möglichkeit zur Repräsentation von Fakten und Annahmen bietet. Die neuen Möglichkeiten zur Einbettung einer terminologischen Logik in ein hybrides Wissensrepräsentationssystem[21] müßten untersucht werden, da zu erwarten ist, daß dies nun viel einfacher und effizienter zu realisieren ist als das bisher der Fall war.
- Auf der praktischen Seite steht die Realisierung der vollen Funktionalität von **Taxon** an, das heißt die Implementierung der Komponente zur Behandlung der Gleichheit.

[21] zum Beispiel TaxLog [Abecker, 1993]

Literatur

[Abecker, 1992] Andreas Abecker. TaxLog: Taxonomische Wissensrepräsentation und logisches Programmieren. In S. Hölldobler, editor, *Logische Programmierung, Beiträge zum 8. Workshop Logische Programmierung, Darmstadt.* AIDA-Report AIDA-92-10, 1992.

[Abecker, 1993] Andreas Abecker. Taxlog: A flexible architecture for logic programming with structured types and constraints. In Manfred Meyer, editor, *Workshop on Constraint Processing*, 1993.

[Baader and Hanschke, 1991] F. Baader and Ph. Hanschke. A scheme for integrating concrete domains into concept languages. Research Report RR-91-10, DFKI / Kaiserslautern, 1991.

[Baader and Hollunder, 1990] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system - system description -. Technical Memo TM -90-3, DFKI / Kaiserslautern, November 1990.

[Baader and Hollunder, 1992] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In *[?]*, 1992.

[Baader *et al.*, 1992a] F. Baader, H.-J. Bürckert, B. Hollunder, A. Laux, and W. Nutt. Terminologische Logiken. *KI*, (3):23–33, 1992.

[Baader *et al.*, 1992b] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological systems. In *[?]*, 1992.

[Boley *et al.*, 1991] H. Boley, Ph. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: a hybrid compilation theory. In *3rd International Workshop on Data, Expert Knowledge and Decisio ns*, September 1991. Also available as DFKI ...

[Brachman and Schmolze, 1985] R.J. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April 1985.

[Brachmann and Levesque, 1984] Ronald L. Brachmann and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proceedings of the 4th National Conference on Artificial Intelligence.* AAAI, 1984.

[de Kleer, 1986] Johann de Kleer. An assumption-based tms. *Artificial Intelligence*, (28):127–162, 1986.

[Doyle, 1979] John Doyle. A truth maintenance system. *Artificial Intelligence*, (12):231–272, 1979.

[Egon Boerger, 1991] Elvinia Riccobene Egon Boerger. Logical operational semantics of parlog. In M. M. Richter H. Boley, editor, *Processing Declarative Knowledge*, 1991.

[Günter, 1993] Andreas Günter. Verfahren zur Auflösung von Konfigurationskonflikten in Expertensystemen. *KI*, (1), 1993.

[Hanschke *et al.*, 1991] Ph. Hanschke, A. Abecker, and D. Drollinger. TAXON: A concept language with concrete domains. In *[?]*, 1991. System Description.

[Hanschke, 1992] Ph. Hanschke. Specifying role interaction in concept languages. In *[?]*, 1992.

[Hanschke, 1993] Philipp Hanschke. *A Declarative Integration of Terminological, Constraint-based,Data-driven, and Goal-directed Reasoning.* PhD thesis, University of Kaiserslautern, 1993.

[Herbig, 1993] Bernhard Herbig. Implementierung terminologischer Inferenzen mit „simplificati on rules". Diplomarbeit, Universität Kaiserslautern, 1993.

[Hollunder, 1990] Bernhard Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. Research Report RR-90-06, DFKI, P.B.2080, D-675 Kaiserslautern, May 1990.

[Malhorta, 1990] V. Malhorta. An algorithm for optimal back-striding in prolog. MIT-Press, 1990.

[M.Bruynooghe, 1991] M.Bruynooghe. Intelligent backtracking revisited. MIT-Press, 1991.

[Nebel, 1988] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, April 1988.

[Nebel, 1989] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems.* PhD thesis, University of Saarbrücken, 1989.

[Nebel, 1990] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[P.Codognet, 1990] P.Codognet. Determining the causes of unsolvability of a system of equations and disequations. MIT-Press, 1990.

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In Ronald Brachman, editor, *First International Conference On Principles of Knowledge Representation and Re asoning*, May 1989.

[Schmiedel, 1990] Albrecht Schmiedel. A temporal terminological logic. In *Proceedings Eight National Conference on Artificial Intelli gence*, volume one, pages 640–645, Menelo Park, Cambridge,London, July 1990. AAAI, AAAI Press / The MIT Press.

[von Luck, 1991] K. von Luck. Hybride logikbasierte Systeme. *KI*, (2):27–31, 1991.

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 Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

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

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder: Embedding Defaults into Terminological 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 Simulators
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 Approach to 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-Agent 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

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

Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken

Dennis Drollinger

D-93-21
Document