

Implementierung und Erweiterung der Sprache *ALCP*

Christoph Endres, Lars Klein, Markus Meyer

Document

D-95-03



Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document

D-95-03

**Implementierung und Erweiterung der
Sprache *ALCP***

Christoph Endres, Lars Klein, Markus Meyer

März 1995

**Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH**

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
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, Siemens and Siemens-Nixdorf. 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.

Dr. Dr. D. Ruland
Director

Implementierung und Erweiterung der Sprache *ALCP*

Christoph Endres, Lars Klein, Markus Meyer

DFKI-D-95-03

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITWM-9400).

© Deutsches Forschungszentrum für Künstliche Intelligenz 1995

This work may not be copied or reproduced in whole or part for any commercial purpose. Permission to copy in whole or part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Deutsche Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, Federal Republic of Germany; an acknowledgement of the authors and individual contributors to the work; all applicable portions of this copyright notice. Copying, reproducing, or republishing for any other purpose shall require a licence with payment of fee to Deutsches Forschungszentrum für Künstliche Intelligenz.

ISSN 0946-0098

Implementierung und Erweiterung der Sprache *ALCP*

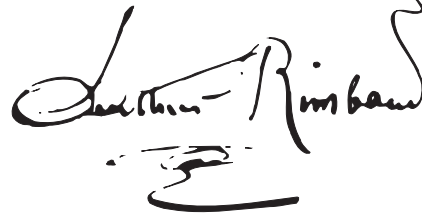
Christoph Endres, Lars Klein, Markus Meyer

email: [endres,lklein,meyer]@dfki.uni-sb.de

März 1995

~~RECHNER~~

~~"Der Dichter bestimmt das Maß des Unbekannten"~~

A handwritten signature in black ink, reading "Arthur Rimbaud". The signature is written in a cursive, somewhat stylized script with a large, sweeping flourish at the end.

Arthur Rimbaud (1854-1891)

Abstract

Diese Dokumentation beschreibt die Implementierung und Erweiterung der Sprache *ALCP*. *ALCP* ist eine von [Heinsohn 94a] entwickelte Erweiterung der terminologischen Wissensrepräsentationssprache *ALC* um eine probabilistische Komponente, so daß auch unsicheres Wissen repräsentiert und für Inferenzen verwendet werden kann. Nach einer Einführung in die grundlegenden Bereiche der Wahrscheinlichkeitstheorie und der Wissensrepräsentation erfolgt eine kurze Beschreibung der Sprache *ALCP*. Anschließend werden die bei der Implementierung verwendeten Konzepte vorgestellt und die benutzten Datenstrukturen und Algorithmen werden beschrieben und erläutert. Der praktische Aspekt der Verwendung der Sprache *ALCP* wird durch eine Beschreibung der verwendbaren Funktionen, eine Bedienungsanleitung und eine Beispielsitzung abgedeckt. Desweiteren wird die Mächtigkeit der Sprache *ALCP* anhand einiger Beispiele veranschaulicht.

Inhaltsverzeichnis

1	Übersicht	9
2	Wissensrepräsentation	11
2.1	Terminologisches Wissen und die Sprache \mathcal{ALC}	11
2.1.1	Syntax und Semantik von \mathcal{ALC}	12
2.1.2	Subsumtion und Klassifizierung	14
2.2	Unsicheres Wissen und die Sprache \mathcal{ALCP}	16
3	Wahrscheinlichkeitstheorie und \mathcal{ALCP}	19
3.1	Axiomatische Wahrscheinlichkeitstheorie	20
3.2	Bedingte Wahrscheinlichkeit	22
3.3	Das Sprachkonstrukt der p-Konditionierung	25
3.4	Trianguläre Beschränkungen (ohne logischen Bezug)	28
3.5	Trianguläre Fälle (mit logischem Bezug)	29
4	Die Umsetzung des \mathcal{ALCP} - Konzeptes	31
4.1	Die Syntax von T- und PBox	31
4.2	Zwei wichtige Begriffe	34
4.3	Zahldarstellung	34
4.4	Überblick über den Algorithmus	35
4.5	Verwendbare Funktionen	37
4.5.1	Behandlung von Wissensbasen	37
4.5.2	Behandlung von p-Konditionierungen	40
4.5.3	Propagierung	41
4.5.4	„History“ und „Trace“	41
5	Die KRIS/RAT-Schnittstelle	43
5.1	Übernahme der Terminologie	43
5.2	Klassifizierung	43
5.3	Subsumtions- und Nullkanten	44
5.4	Leere Konzepte	47

6	Datenstrukturen	49
6.1	Die Wahl der geeigneten Datenstrukturen	49
6.2	Die verwendeten Hashtabellen	51
6.3	Listen	53
6.4	Globale Variablen	55
7	Constraints und ihre Propagierung	57
7.1	Auswahl der triangulären Fälle	58
7.2	Der Propagierungsalgorithmus	59
7.3	Das Constraint CONS-LOG	61
7.4	Das Constraint CONS-PINGUIN	67
7.5	Die Abarbeitung eines triangulären Falles	68
7.6	Modifizierung von p-Konditionierungen	69
8	Beispiele, Trace	71
8.1	Nixon-Diamant	73
8.2	Einige weitere Beispiele	79
8.2.1	Barking Cats	79
8.2.2	Birds	80
8.2.3	Birds 2	81
8.2.4	Burglary	83
8.2.5	Children	85
8.2.6	Songwriter	86
8.2.7	Sprinkler	88
9	Anleitung und Oberfläche	91
9.1	Laden des Systems	91
9.2	Die Oberfläche	92
9.3	Befehlsübersicht	95
9.4	Beispielsitzung	96
9.4.1	<i>ALCP</i>	97
9.4.2	<i>XALCP</i>	99
10	Schlußbemerkung und Ausblick	103
	Literaturverzeichnis	105
	Stichwortverzeichnis	107

Kapitel 1

Übersicht

Dieses Kapitel beschreibt, wie diese Dokumentation aufgebaut ist. Die Dokumentation ist in drei Teile gegliedert. Der erste Teil, bestehend aus den nächsten beiden Kapiteln, gibt eine allgemeine Einführung in die Bereiche Wissenrepräsentation durch terminologische Logiken und Wahrscheinlichkeitstheorie unter Berücksichtigung der für *ALCP* notwendigen Gesichtspunkte. Der zweite Teil, der aus den Kapiteln vier bis sieben besteht, beschreibt die Implementierung und Erweiterung der Sprache *ALCP*. Im dritten Teil, der die letzten drei Kapitel umfaßt, werden einige Beispiele diskutiert, eine Bedienungsanleitung und schließlich eine Zusammenfassung mit Ausblick gegeben.

Im zweiten Kapitel erfolgt eine Einführung in die Wissenrepräsentation durch terminologische Logiken. Es wird erläutert, wie man Wissen über Objektklassen mit Hilfe terminologischer Logiken darstellen kann, und es wird die Sprache *ALC* vorgestellt, die es erlaubt, terminologisches Wissen zu modellieren und daraus Inferenzen zu gewinnen. Dazu werden die Syntax und die Semantik von *ALC* eingeführt und erläutert. Anschließend wird erläutert, daß terminologisches Wissen oft nicht ausreichend ist, um eine Domäne zu modellieren. Daraus ergibt sich die Notwendigkeit, unsicheres Wissen zu verwenden. Wie dieses unsichere Wissen dargestellt werden kann und wie daraus Inferenzen gezogen werden können, ist das Leitthema der folgenden Kapitel.

Das dritte Kapitel gibt zuerst eine grundlegende Einführung in die Wahrscheinlichkeitstheorie und stellt dann die Sprache *ALCP* vor, die eine Erweiterung von *ALC* darstellt, und mit der unsicheres Wissen modelliert werden kann. Es werden die Begriffe der p-Konditionierung und der triangulären Fälle eingeführt und darauf aufbauend die probabilistischen Constraints, mit deren Hilfe Inferenzen aus unsicherem Wissen gezogen werden können.

Im Kapitel 4 wird ein Überblick über die Implementierung der Sprache *ALCP* gegeben, der grundlegende Fragen beantworten und eine Basis für die nachfolgenden Kapitel bilden soll. Die Syntax von T- und PBox wird vorgestellt und an einem Beispiel motiviert und erläutert. Im weiteren Verlauf des Kapitels wird beschrieben, wie Zahlen dargestellt werden und welche Möglichkeiten zur Ein-

gabe und Ausgabe der p-Konditionierungen vorhanden sind. Abschließend wird dargelegt, welche Funktionen vorhanden sind und wie sie implementiert wurden. Im fünften Kapitel wird die Schnittstelle zu KRIS/RAT beschrieben. Es wird gezeigt, wie terminologisches Wissen aus einer TBox übernommen und in p-Konditionierungen umgewandelt wird. Weiterhin wird beschrieben, wie Subsumtions- und Nullkanten behandelt und wie leere Konzepte festgestellt werden.

Kapitel 6 stellt die verwendeten Datenstrukturen vor. Die Gründe für die gewählten Datenstrukturen werden erläutert, und anschließend erfolgt eine Beschreibung der einzelnen Hashtabellen, Listen und globalen Variablen.

Das siebte Kapitel beschreibt detailliert den verwendeten Algorithmus zur Behandlung triangulärer Fälle und zur Propagierung der probabilistischen Constraints durch eine Wissensbasis. Es wird ein weiteres Constraint vorgestellt, welches Sonderfälle behandelt. Auf das von uns erweiterte Constraint CONS-LOG wird genauer eingegangen, und die Anwendung von CONS-LOG auf einen triangulären Fall wird umfassend beschrieben.

Im achten Kapitel wird das in den Kapiteln 2 und 4 verwendete Beispiel des Nixon-Diamanten abschließend behandelt. Dazu wird beschrieben, wie die gefundenen Inferenzen mit Hilfe der probabilistischen Constraints erzielt werden. Daran anschließend werden noch einige weitere interessante Beispiele vorgestellt. Kapitel 9 enthält eine Bedienungsanleitung des Systems, die sowohl die Befehle der Oberfläche wie auch der Befehlszeile beschreibt und an einer Beispielsitzung näher erläutert.

Im letzten Kapitel wird schließlich eine Zusammenfassung der erzielten Ergebnisse und ein Ausblick auf weitere wichtige und interessante Aufgaben in diesem Bereich gegeben.

Kapitel 2

Wissensrepräsentation

Die Wissensrepräsentation stellt eine der Grundlagen der Künstlichen Intelligenz (KI) dar. Repräsentation von Wissen und seine Verarbeitung haben fundamentale Bedeutung für die meisten Bereiche des Lebens, so daß die Behandlung des Wissens in Computern ein wichtiger Forschungsgegenstand ist und auch bleiben wird. Hierbei sind vor allem zwei Fragen entscheidend: wie kann Wissen dargestellt (repräsentiert) werden und wie können aus diesem Wissen Schlüsse gezogen werden, d.h. wie kann neues Wissen aus bereits bekanntem abgeleitet werden? Für die Beantwortung dieser Fragen gibt es verschiedene Möglichkeiten, einen Überblick über diese Ansätze gibt Bibel in seinem Buch [Bibel 93]. Wir werden uns im Rahmen dieser Dokumentation auf terminologische Systeme [Heinsohn et al. 94] und eine geeignete probabilistische Erweiterung konzentrieren.

Der nachfolgende Abschnitt gibt eine Einführung in terminologisches Wissen und beschreibt eine terminologische Repräsentationssprache. Im darauffolgenden Abschnitt wird die Notwendigkeit der Behandlung unsicheren Wissens erläutert. Die mathematische Darstellung von unsicherem Wissen wird nach einer Einführung in die Wahrscheinlichkeitstheorie im nächsten Kapitel beschrieben.

2.1 Terminologisches Wissen und die Sprache *ALC*

Die Grundidee bei der Repräsentierung von Wissen durch eine *terminologische Logik* ist die Bildung einer Hierarchie von Begriffen, *Konzepte* genannt, die durch die Ober- bzw. Unterbegriffsrelation aufgebaut wird. Die Objekte einer *Welt* bzw. einer gewählten *Domäne* werden dadurch verschiedenen Klassen bzw. Konzepten zugeordnet. Terminologisches Wissen setzt sich nun ausschließlich aus diesen Konzepten und den Beziehungen zwischen ihnen zusammen. Die einzelnen Objekte werden hier nicht betrachtet. Das terminologische Wissen einer Domäne ist in der sogenannten *TBox* enthalten.

Die Ober- bzw. Unterbegriffsrelation, die die Bildung der Konzepthierarchie ermöglicht, wird *Subsumtionsrelation* genannt und ist wie folgt charakterisiert: Ein Konzept K wird durch ein Konzept K' subsumiert, wenn in allen möglichen Welten die zu dem Konzept K gehörenden Objekte auch zu dem Konzept K' gehören. Da hier eine Aussage über Mengen von Objekten gemacht wird, kann die Semantik einer terminologischen Sprache durch mengentheoretische Ausdrücke definiert werden. Die Subsumtionsrelation ist reflexiv, transitiv und antisymmetrisch, sie bildet also eine partielle Ordnung auf der Menge der Konzepte.

Eine *terminologische Repräsentationssprache* besteht aus *definierten Konzepten*, die durch notwendige und hinreichende Bedingungen festgelegt werden, aus *primitiven Konzepten*, die nur durch notwendige Bedingungen definiert sind, und schließlich aus *Rollen*. Rollen sind zweistellige Relationen zwischen Konzepten, die dazu dienen, Konzepte zu beschreiben. Zwei spezielle Konzepte sind das *TOP-Konzept* \top , welches alle Objekte der Domäne umfaßt, und das *BOTTOM-Konzept* \perp , welches kein Objekt enthält.

Durch die Klassifizierung der in einer TBox gegebenen Konzeptbeschreibungen wird eine Hierarchie der Konzepte auf der Grundlage der Subsumtionsrelation erstellt, wobei die Konzepte an den jeweils spezifischsten Stellen in die Hierarchie aufgenommen werden.

Eine terminologische Repräsentationssprache stellt \mathcal{ALC} (Attributive concept description Language with Complements) dar, die von [Smolka et al. 89] entwickelt wurde. Im folgenden sollen Syntax und Semantik von \mathcal{ALC} eingeführt und Subsumtion und Klassifizierung beschrieben werden.

In Kapitel 2.2 wird anhand des Nixon-Diamanten ein Beispiel für terminologisches Wissen gegeben.

2.1.1 Syntax und Semantik von \mathcal{ALC}

Nach der Erläuterung der terminologischen Logiken zugrundeliegenden Ideen wird nun eine spezielle terminologische Wissensrepräsentationssprache eingeführt. Die Sprache \mathcal{ALC} wird durch formale Definition sowohl der Syntax als auch der Semantik beschrieben.

Definition 2.1 (Notation) *Es heißen*

\mathcal{S} Menge der Konzeptsymbole

\mathcal{R} Menge der Rollensymbole

\mathcal{C} Menge der Konzeptbeschreibungen bzw. Konzepte

Die Syntax der terminologischen Wissensrepräsentationssprache \mathcal{ALC} wird durch folgende Definition festgelegt:

Definition 2.2 (Konzeptausdruck)

Seien $A \in \mathcal{S}$, $C, D \in \mathcal{C}$ und $R \in \mathcal{R}$.

Ein Konzeptausdruck wird nach der folgenden Syntaxregel gebildet:

$$\begin{array}{l}
 C, D \Leftrightarrow A|\top|\perp \quad \text{atomare Konzepte} \\
 | \quad C \sqcap D \quad \text{Konzeptkonjunktion} \\
 | \quad C \sqcup D \quad \text{Konzeptdisjunktion} \\
 | \quad \neg C \quad \text{Konzeptnegation} \\
 | \quad \forall R : C \quad \text{Wertrestriktion} \\
 | \quad \exists R : C \quad \text{Existenzrestriktion.}
 \end{array}$$

Die Konzeptbeschreibungen dienen dazu, Konzepte zu definieren. Es gibt zwei Arten von Konzeptdefinitionen: eine, um notwendige und hinreichende Bedingungen für ein Konzept zu beschreiben und eine, um nur notwendige Bedingungen aufzustellen. Entsprechend unterscheidet man zwischen Konzepten, die durch den Operator \doteq definiert werden und primitiven Konzepten, für deren Definition der Operator \sqsubseteq verwendet wird.

Die Bedeutung von Konzeptausdrücken wird durch folgende Definition erklärt. Es wird eine Abbildung \mathcal{E} - die sogenannte *Extensionsfunktion* - eingeführt, die jede Konzeptbeschreibung auf eine Teilmenge von \mathcal{D} und jedes Rollensymbol auf eine Teilmenge von $\mathcal{D} \times \mathcal{D}$ abbildet:

Definition 2.3 (Extensionsfunktion)

Sei die Domäne \mathcal{D} eine beliebige Menge und \mathcal{E} eine Funktion

$$\mathcal{E} : \begin{cases} \mathcal{C} \rightarrow 2^{\mathcal{D}} \\ \mathcal{R} \rightarrow 2^{\mathcal{D} \times \mathcal{D}}. \end{cases}$$

\mathcal{E} ist genau dann eine *Extensionsfunktion*, wenn die folgenden Gleichungen erfüllt sind:

$$\begin{aligned}
 \mathcal{E}[\top] &= \mathcal{D} \\
 \mathcal{E}[\perp] &= \emptyset \\
 \mathcal{E}[C \sqcap D] &= \mathcal{E}[C] \cap \mathcal{E}[D] \\
 \mathcal{E}[C \sqcup D] &= \mathcal{E}[C] \cup \mathcal{E}[D] \\
 \mathcal{E}[\neg C] &= \mathcal{D} \setminus \mathcal{E}[C] \\
 \mathcal{E}[\forall R : C] &= \{x \in \mathcal{D} \mid (x, y) \in \mathcal{E}[R] \Rightarrow y \in \mathcal{E}[C] \text{ f.a. } y \in \mathcal{D}\} \\
 \mathcal{E}[\exists R : C] &= \{x \in \mathcal{D} \mid \text{es ex. } y \in \mathcal{D} : (x, y) \in \mathcal{E}[R] \wedge y \in \mathcal{E}[C]\}
 \end{aligned}$$

Eine Menge von Konzeptdefinitionen beschreibt einen Ausschnitt aus der Welt, eine sogenannte *Domäne*. Diese Menge von vollständigen und partiellen Konzeptbeschreibungen wird *Terminologie* genannt, wenn einige Eigenschaften erfüllt werden. Zur Formalisierung von Terminologie dient die folgende Definition.

Definition 2.4 (Terminologie)

Eine Menge \mathcal{T} wohlgeformter partieller und vollständiger Konzeptdefinitionen heißt genau dann Terminologie, wenn jedes Konzeptsymbol höchstens einmal auf der linken Seite auftritt und keine terminologischen Zyklen vorhanden sind.

Ein Beispiel für einen terminologischen Zyklus ist die nachfolgende partielle Konzeptdefinition:

$$\text{Mensch} \sqsubseteq \text{Säugetier} \sqcap (\forall \text{Nachkommen} : \text{Mensch})$$

Als Abschluß dieses Abschnittes soll nun noch die Erweiterung der Extensionsfunktion von Konzepten bzw. Rollen auf Terminologien sowie die Einführung eines später benötigten Begriffes erfolgen:

Definition 2.5 (Extension und Modelle einer Terminologie)

Seien \mathcal{T} eine Terminologie und \mathcal{E} eine Extensionsfunktion. \mathcal{E} heißt genau dann Extensionsfunktion von \mathcal{T} , wenn für jedes Konzeptsymbol A und jeden Konzeptausdruck C die Bedingungen

$$\begin{aligned} \mathcal{E}[A] &= \mathcal{E}[C] \quad \text{für alle } (A \doteq C) \in \mathcal{T} \\ \mathcal{E}[A] &\subseteq \mathcal{E}[C] \quad \text{für alle } (A \sqsubseteq C) \in \mathcal{T} \end{aligned}$$

gelten. Die Menge

$$\text{mod}(\mathcal{T}) := \{\mathcal{E} \mid \mathcal{E} \text{ Extensionsfunktion von } \mathcal{T}\}$$

heißt Menge der Modelle von \mathcal{T} .

2.1.2 Subsumtion und Klassifizierung

Wie schon in der Einleitung zu diesem Kapitel beschrieben wurde, wird die Konzepthierarchie durch die Subsumtionsrelation strukturiert. Zur formalen Festlegung des Begriffes Subsumtion dient die folgende Definition:

Definition 2.6 (Subsumtion)

Sei \mathcal{T} eine Terminologie. Ein Konzept K wird genau dann von einem Konzept K' in der Terminologie \mathcal{T} subsumiert ($K \preceq_{\mathcal{T}} K'$), wenn für alle Modelle von \mathcal{T} gilt:

$$\mathcal{E}[[K]] \subseteq \mathcal{E}[[K']]$$

Die Umsetzung der durch die Konzeptausdrücke definierten Konzepte in eine Hierarchie geschieht durch die Klassifizierung der gegebenen Konzeptdefinitionen. Dabei werden die Konzepte an der Stelle in die Hierarchie eingefügt, die für sie am spezifischsten ist.

Nach der Klassifizierung einer TBox werden bestimmte Beziehungen zwischen Konzepten und Eigenschaften von Konzepten sichtbar. In der folgenden Definition werden diese Beziehungen und Eigenschaften formalisiert:

Definition 2.7 (Äquivalenz, Disjunktheit und leere Konzepte)

Sei \mathcal{T} eine Terminologie.

- Zwei Konzepte K und K' heißen genau dann äquivalent in \mathcal{T} , geschrieben $K \approx_{\mathcal{T}} K'$, wenn für jede Extensionsfunktion \mathcal{E} und jede Domäne \mathcal{D} von \mathcal{T} gilt:

$$\mathcal{E}[[K]] = \mathcal{E}[[K']]$$

- Zwei Konzepte K und K' heißen genau dann disjunkt in \mathcal{T} , wenn für jede Extensionsfunktion \mathcal{E} und jede Domäne \mathcal{D} von \mathcal{T} gilt:

$$\mathcal{E}[[K]] \cap \mathcal{E}[[K']] = \emptyset$$

- Ein Konzept K heißt genau dann leer (inkohärent) in \mathcal{T} , wenn für jede Extensionsfunktion \mathcal{E} und jede Domäne \mathcal{D} von \mathcal{T} gilt:

$$\mathcal{E}[[K]] = \emptyset$$

Diese Begriffe können auf die Subsumtionsrelation zurückgeführt werden:

Satz 2.1

Gegeben sei eine Terminologie \mathcal{T} und Konzepte K und K' . Es gelten:

- K und K' sind genau dann äquivalent, wenn $K \preceq_{\mathcal{T}} K'$ und $K' \preceq_{\mathcal{T}} K$ gelten.

- K und K' sind genau dann disjunkt, wenn $K \sqcap K' \preceq_{\mathcal{T}} \perp$ gilt.
- K ist genau dann leer, wenn $K \preceq_{\mathcal{T}} \perp$ gilt.

Der Beweis dieses Satzes wird in [Nebel 90] geführt.

2.2 Unsicheres Wissen und die Sprache \mathcal{ALCP}

Die Notwendigkeit der Behandlung unsicheren Wissens wird nach der Betrachtung eines Beispiels deutlich. Betrachten wir den bekannten Nixon-Diamant:

Beispiel 2.1

In diesem Beispiel werden die Beziehungen zwischen Quäkern, Republikanern und Pazifisten betrachtet, wobei Quäker Pazifisten und Republikaner keine Pazifisten sind. Es ergibt sich die folgende Terminologie:

$$\begin{aligned} \text{Pazifist} &\sqsubseteq T \\ \text{Quäker} &\sqsubseteq \text{Pazifist} \\ \text{Republikaner} &\sqsubseteq \neg \text{Pazifist} \end{aligned}$$

Der ehemalige Präsident der Vereinigten Staaten von Amerika, Richard Nixon, war gleichzeitig Quäker und Republikaner. Er gehörte somit zur Extension des Konzeptes $R\&Q$ (Republikaner und Quäker), das wie folgt definiert wird:

$$R\&Q \doteq \text{Quäker} \sqcap \text{Republikaner}$$

Die Klassifizierung dieser vier terminologischen Axiome ergibt eine Inkonsistenz. Das Konzept $R\&Q$ wird zum einen von *Quäker* und somit auch von *Pazifist* subsumiert, zum anderen aber auch von *Republikaner* und \neg *Pazifist*. Ein Republikaner und Quäker, wie Nixon einer war, wäre daher sowohl Pazifist als auch kein Pazifist, ein klarer Widerspruch. **Dieser Widerspruch kann mit rein terminologischem Wissen nicht aufgelöst werden.** Er könnte innerhalb der Terminologie nur dann aufgelöst werden, wenn das Konzept $R\&Q$ leer wäre. Da die Extension des Konzeptes $R\&Q$ aber mindestens ein Objekt, nämlich Nixon, enthält, ist das Konzept nicht leer.

Die Inkonsistenz wird aufgelöst, wenn man annimmt, daß nicht *alle* Quäker Pazifisten und nicht *alle* Republikaner keine Pazifisten sind. Dies ist ja auch tatsächlich der Fall: die *meisten* Quäker sind Pazifisten, aber es gibt auch *Ausnahmen*. Ähnliches gilt für die Republikaner und die meisten anderen „natürlichen“ Konzepte, im Unterschied zu mathematischen oder logischen Konzepten. „Natürliche“ Konzepte zeichnen sich gerade dadurch aus, daß die Eigenschaften zwar für die

meisten Objekte, aber doch nicht für alle gelten. Zur Repräsentierung dieser Konzepte ist also eine Modellierung von Ausnahmen nötig, da diese nicht durch terminologische Logiken beschrieben werden können.

Diese Ausnahmen stellen *nichtkategorisches* bzw. *unsicheres Wissen* dar. Die Unsicherheit liegt nicht darin, ob ein Objekt tatsächlich zur Extension eines Konzeptes gehört. Die Unsicherheit drückt vielmehr aus, daß ein Objekt nicht alle kategorischen Eigenschaften eines Konzeptes notwendigerweise erfüllen muß, z.B. müssen nicht alle Quäker Pazifisten sein.

Zur Behandlung dieses unsicheren Wissens gibt es wiederum mehrere Ansätze. Der hier verfolgte Ansatz basiert auf der Wahrscheinlichkeitstheorie und ist somit probabilistisch. Die folgenden Ausführungen sollen motivieren, warum gerade dieser Ansatz gewählt wurde:

Betrachtet man die Definition der Subsumtion, so stellt man fest, daß alle Objekte der Extension eines Konzeptes K notwendigerweise auch Objekte der Extension des Konzeptes K' sein müssen, damit K von K' subsumiert wird. Diese sichere oder kategorische Zugehörigkeit wird durch eine unsichere oder flexible Zugehörigkeit abgelöst. Das unsichere Wissen besteht also darin, untere und obere Schranken für die Zugehörigkeit anzugeben. Aus der Aussage, daß alle Quäker Pazifisten sind, wird das unsichere Wissen, daß mindestens 90 Prozent aller Quäker Pazifisten sind. Hierdurch wird gerade ausgedrückt, daß zwar die meisten, aber nicht unbedingt alle Quäker Pazifisten sind. Die untere Grenze ist hier 90 Prozent und die obere 100 Prozent. Auf das Einheitsintervall bezogen entsprechen diese Zugehörigkeiten den Werten 0.9 und 1, die die Intervallgrenzen darstellen.

In Kapitel 4 wird die Auflösung der Inkonsistenz des Nixon-Diamanten mit Hilfe des Begriffs der p-Konditionierungen beschrieben.

Im nächsten Kapitel wird nach einer Einführung in die Wahrscheinlichkeitstheorie die Verwendung von p-Konditionierungen erklärt.

Kapitel 3

Wahrscheinlichkeitstheorie und *ALCP*

Kapitel 3 beschreibt grundlegende Begriffe und Verfahren zur Sprache *ALCP* aus der als Basis verwendeten Dissertation von Heinsohn [Heinsohn 94a]. Wichtige Aspekte sind auch in [Heinsohn 94b] beschrieben.

In der Statistik werden Massenerscheinungen in Gesellschaft, Naturwissenschaften, Technik und Medizin mit Mitteln der Wahrscheinlichkeitstheorie untersucht. Die *Wahrscheinlichkeitstheorie* ist das Teilgebiet der Mathematik, das sich damit befaßt, die Gesetzmäßigkeiten unter den zufälligen Ereignissen aufzudecken und zu untersuchen.

Auch in der KI spielt die Wahrscheinlichkeitstheorie eine immer größere Rolle. Insbesondere bei der Modellierung von Alltagswissen tritt in konkreten und realistischen Anwendungen Unsicherheit von Wissen auf, welches zunehmend mit wahrscheinlichkeitstheoretischen (probabilistischen) Verfahren gehandhabt wird. So ist die Modellierung von Unsicherheit ein aktuelles Forschungsgebiet in der KI. Die Modellierung von Unsicherheit beinhaltet natürlich nicht nur die Wahrscheinlichkeitstheorie, sondern auch verwandte Gebiete wie Fuzzy Logik, heuristische Modelle, Dempster-Shafer-Theorie, Bayessche Netze und graphbasierte Verfahren [Kruse et al. 91], von denen einige auf wahrscheinlichkeitstheoretische Grundlagen zurückgreifen.

Die Abschnitte 3.1 und 3.2 stellen einige grundlegende Ergebnisse einer axiomatischen Wahrscheinlichkeitstheorie zusammen. Die Abschnitte 3.3 bis 3.5 behandeln die Möglichkeit, die Unsicherheit von Wissen in Form von Wahrscheinlichkeitsintervallen darzustellen. Es wird das Unsicherheitsmodell für Konzepte - *ALCP* - vorgestellt. Dieses stellt eine Erweiterung der Sprache *ALC* aus Kapitel 2 dar. Mit Hilfe der probabilistischen Konditionierung (kurz: p-Konditionierung) von Konzepten kann unsicheres Wissen über Abhängigkeiten von Konzepten repräsentiert werden. Es können demnach statistisch interpretierbare Aussagen modelliert werden. Für die Sprache *ALCP*, die terminologische und probabilistische Sprachkonstrukte integriert, wird eine einheitliche Semantik entwickelt. Die Kon-

sistenz terminologischen und probabilistischen Wissens erfordert unterschiedliche Bedingungen. Gegebenenfalls werden die solchen Bedingungen entsprechenden probabilistischen Beschränkungen (*constraints*) dazu verwendet, implizite Beziehungen zwischen Konzeptbeschreibungen abzuleiten und quantitativ zu erfassen. Abschnitt 3.3 behandelt das Sprachkonstrukt der p-Konditionierung. Dies ermöglicht die Modellierung von Tendenzen und ist für den wahrscheinlichkeitstheoretischen Formalismus der Sprache \mathcal{ALCP} grundlegend. Die Abschnitte 3.4 und 3.5 stellen trianguläre Constraints über Konzepten dar, die eine lokale Berechnung implizit vorhandener p-Konditionierungen gestatten. Die angegebenen Constraints ermöglichen

- die Berechnung nicht explizit angegebener p-Konditionierungen
- die Verfeinerung von Intervallen explizit angegebener oder bereits berechneter p-Konditionierungen
- die Feststellung einer Inkonsistenz von p-Konditionierungen und damit der Inkonsistenz der gesamten Wissensbasis.

3.1 Axiomatische Wahrscheinlichkeitstheorie

Aus den axiomatischen Begründungen der Geometrie, der Algebra und anderer mathematischer Disziplinen weiß man, daß dort davon abgesehen wird, Begriffe wie Punkt und Gerade, Zahl usw. inhaltlich zu definieren. Ähnlich hat es sich gezeigt, daß für einen Aufbau der Wahrscheinlichkeitstheorie eine inhaltliche Definition von Begriffen wie “Ereignis“ und “Wahrscheinlichkeit“ nicht erforderlich, ja zur Vermeidung logischer Schwierigkeiten und im Hinblick auf eine möglichst umfangreiche und leichte Anwendbarkeit der Theorie nicht einmal erstrebenswert ist. Wie in den genannten anderen mathematischen Disziplinen kommt es auch in der Wahrscheinlichkeitstheorie nur auf die formalen Eigenschaften dieser Begriffe an. Es ist das Verdienst des russischen Mathematikers A.N. Kolmogorov [Kolmogorov 33], als erster diese Gedanken konsequent zu Ende gedacht zu haben.

Dieses Axiomensystem von Kolmogorov präzisiert den Begriff der Wahrscheinlichkeit eines zufälligen Ereignisses. Es ist die mathematische Fassung der Gesetzmäßigkeiten, die für die Häufigkeit der zufälligen Ereignisse in langen Versuchsreihen unter denselben Bedingungen beobachtet werden konnten.

Desweiteren werden in diesem Abschnitt erste Folgerungen aus den Axiomen formuliert.

Bei der Durchführung eines Versuchs (z.B. Werfen einer Münze, Werfen eines Spielwürfels) können zufällige Einflüsse einwirken, die den Ausgang des Versuchs ungewiß sein lassen. Deshalb nennt man das Ergebnis eines Versuchs, der bei

Einhaltung der Bedingungen beliebig oft wiederholt werden kann, ein „zufälliges Ereignis“ (*Elementarereignis*). Sei Ω nun eine endliche (oder abzählbare) Menge von Elementarereignissen. Ereignisse von besonderer Bedeutung sind das unmögliche Ereignis \emptyset sowie das sichere Ereignis Ω , das alle elementaren Ereignisse enthält. Sind A und B Ereignisse, so sind die Vereinigung von A und B , der Durchschnitt von A und B und die Komplemente von A und B ebenfalls Ereignisse.

Definition 3.1 (Ereignisraum)

Eine Teilmenge \mathcal{F} der Potenzmenge 2^Ω von Ω heißt Ereignisraum, wenn die folgenden Bedingungen erfüllt sind:

$$\begin{aligned}\Omega &\in \mathcal{F} \\ A, B \in \mathcal{F} &\implies A \cup B \in \mathcal{F} \\ A \in \mathcal{F} &\implies \neg A \in \mathcal{F}\end{aligned}$$

Wir nennen die Funktion $P: 2^\Omega \leftrightarrow [0, 1]$ Wahrscheinlichkeit, wenn sie die folgenden als Kolmogorov-Axiome bekannten Bedingungen erfüllt:

Definition 3.2 (Kolmogorov-Axiome)

Sei Ω eine endliche Menge. Jede Abbildung

$$\begin{aligned}P: 2^\Omega &\leftrightarrow [0, 1] \\ A &\leftrightarrow P(A)\end{aligned}$$

heißt Wahrscheinlichkeitsmaß, wenn sie die nachfolgenden Axiome erfüllt:

$$\begin{aligned}P(A) &\geq 0 \text{ für alle } A \subseteq \Omega \\ P(\Omega) &= 1 \\ P(A \cup B) &= P(A) + P(B) \text{ für } A, B \subseteq \Omega \text{ mit } A \cap B = \emptyset\end{aligned}$$

Das Tripel (Ω, \mathcal{F}, P) heißt dann Wahrscheinlichkeitsraum. Die Wahrscheinlichkeit $P(A)$ ist damit ein numerischer Index, der ein Maß dafür angibt, daß das Ereignis A eintritt. Eine Wahrscheinlichkeitsfunktion hat die folgenden Eigenschaften:

Satz 3.1 Für alle $A, B \subseteq \Omega$ gilt:

$$P(\emptyset) = 0 \tag{3.1}$$

$$P(A) \leq 1 \quad (3.2)$$

$$A \subseteq B \implies P(A) \leq P(B) \text{ (Monotonie)} \quad (3.3)$$

$$P(A \cup B) \leq P(A) + P(B) \text{ (Subadditivität)} \quad (3.4)$$

$$A \subseteq B \implies P(B \cap \bar{A}) = P(B) - P(A) \text{ (Subtraktivität)} \quad (3.5)$$

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \text{ (Additivität)} \quad (3.6)$$

$$P(\bar{A}) = 1 - P(A) \quad (3.7)$$

Die auf mehr als zwei Elemente verallgemeinerte Darstellung von (3.6) wird als Additivitätseigenschaft von Wahrscheinlichkeiten bezeichnet:

Satz 3.2 (Additivitätseigenschaft) Für alle $A_1, \dots, A_n \subseteq \Omega$ gilt:

$$P\left(\bigcup_{i=1}^n A_i\right) = \sum_{I \subseteq \{1, \dots, n\}, I \neq \emptyset} (-1)^{|I|+1} P\left(\bigcap_{i \in I} A_i\right)$$

3.2 Bedingte Wahrscheinlichkeit

Der folgende Abschnitt befaßt sich mit dem Phänomen, daß das Wissen (die Bedingung) des Eintretens eines konkreten Ereignisses mit positiver Wahrscheinlichkeit die Wahrscheinlichkeit eines weiteren Ereignisses verändern kann.

Unter der bedingten Wahrscheinlichkeit versteht man die Wahrscheinlichkeit eines Ereignisses A unter der Bedingung, daß ein anderes Ereignis mit $P(E) \neq 0$ bereits eingetreten ist. Sie wird mit $P(A|E)$ bezeichnet und „Wahrscheinlichkeit von A unter der Bedingung E “ gelesen.

Diese bedingte Wahrscheinlichkeit erfüllt die Axiome von Kolmogorov, und somit gelten für sie alle Sätze, die man aus den Axiomen herleiten kann. Die allgemeine Multiplikationsregel und die Regel der totalen Wahrscheinlichkeit führen zu der Formel für die sogenannte a-posteriori Wahrscheinlichkeit von Bayes, die in den Anwendungen eine große Rolle spielt.

Definition 3.3 (Konditionale Wahrscheinlichkeit)

Sei P eine Wahrscheinlichkeit über Ω und $E \subseteq \Omega$ ein Ereignis mit $P(E) > 0$. Dann heißt

$$P(A|E) \stackrel{\text{def}}{=} \frac{P(A \cap E)}{P(E)}$$

konditionale (bedingte) Wahrscheinlichkeit von A unter der Hypothese (Bedingung) E für jedes Ereignis $A \subseteq \Omega$.

Für die konditionale Wahrscheinlichkeit gilt nun:

Satz 3.3 Für E mit $P(E) > 0$ ist die Abbildung

$$P_{(\cdot|E)} : 2^\Omega \rightarrow [0, 1]$$

$$P_{(\cdot|E)}(A) \stackrel{\text{def}}{=} P(A|E)$$

eine Wahrscheinlichkeit über Ω .

Beispiel 3.1 Beim Würfeln mit zwei Spielwürfeln seien das Ereignis A und das Ereignis E gegeben:

- Ereignis A : Augensumme mindestens 9
- Ereignis E : Augensumme ungerade

Daraus ergibt sich:

$$P(E) = \frac{18}{36} = \frac{1}{2}$$

$$P(A \cap E) = \frac{6}{36} = \frac{1}{6}$$

aus den sechs möglichen Kombinationen $\langle 3, 6 \rangle, \langle 4, 5 \rangle, \langle 5, 4 \rangle, \langle 5, 6 \rangle, \langle 6, 3 \rangle, \langle 6, 5 \rangle$ und damit die bedingte Wahrscheinlichkeit

$$P(A|E) = \frac{P(A \cap E)}{P(E)} = \frac{1}{6} : \frac{1}{2} = \frac{1}{3}$$

Diese Methode, die Wahrscheinlichkeit von Überschneidungen zu berechnen, kann leicht auf Sequenzen von n Ereignissen generalisiert werden. Aus der Definition bedingter Wahrscheinlichkeiten folgt für $A_i \subseteq \Omega$:

$$P(A_1 \cap A_2) = P(A_2|A_1) \cdot P(A_1)$$

$$P(A_1 \cap A_2 \cap A_3) = P(A_3|A_1 \cap A_2) \cdot P(A_1 \cap A_2)$$

$$= P(A_3|A_1 \cap A_2) \cdot P(A_2|A_1) \cdot P(A_1)$$

Daraus ergibt sich für n Ereignisse die folgende Regel, die als allgemeine Multiplikationsregel bekannt ist:

Satz 3.4 (Allgemeine Multiplikationsregel)

Gegeben seien die Ereignisse $A_1, \dots, A_n \subseteq \Omega$ mit $P\left(\bigcap_{j=1}^{n-1} A_j\right) > 0$. Dann gilt:

$$P\left(\bigcap_{j=1}^n A_j\right) = P(A_n | \bigcap_{j=1}^{n-1} A_j) \cdot \dots \cdot P(A_3 | A_1 \cap A_2) \\ \cdot P(A_2 | A_1) \cdot P(A_1)$$

Dieser Satz ist auch Grundlage der Bayesschen Netze, auf die in unserer Arbeit jedoch nicht weiter eingegangen wird.

Satz 3.5 (Regel der totalen Wahrscheinlichkeit)

Sei H_1, \dots, H_n eine Partition von Ω , das heißt es gelten $H_i \cap H_j = \emptyset$ für $i \neq j$ und $H_1 \cup \dots \cup H_n = \Omega$.

Falls $P(H_i) > 0$ für $i = 1, \dots, n$, dann folgt für jedes Ereignis $A \subseteq \Omega$:

$$P(A) = \sum_{i=1}^n P(A \cap H_i) \\ = \sum_{i=1}^n P(A|H_i) \cdot P(H_i)$$

Für $A, H \subseteq \Omega$ gilt:

$$P(H \cap A) = P(H|A) \cdot P(A) \\ P(A \cap H) = P(A|H) \cdot P(H)$$

Mit $P(A) > 0$ gilt auch :

$$P(H|A) = \frac{P(A|H) \cdot P(H)}{P(A)},$$

wobei $P(H)$ eine a-priori Wahrscheinlichkeit und $P(H|A)$ eine a-posteriori Wahrscheinlichkeit ist.

Daraus folgt das bekannte Theorem von Bayes:

Satz 3.6 (Theorem von Bayes)

Sei H_1, \dots, H_n eine Partition von Ω mit $P(H_i) > 0$ für $i = 1, \dots, n$ und sei A ein Ereignis mit $P(A) > 0$. Dann gilt:

$$P(H_j|A) = \frac{P(A|H_j) \cdot P(H_j)}{\sum_{i=1}^n P(A|H_i) \cdot P(H_i)}$$

Sind zwei Ereignisse A und B mit den Wahrscheinlichkeiten $P(A)$ bzw. $P(B)$ gegeben und beeinflusst das Eintreten des Ereignisses A nicht die Wahrscheinlichkeit des Ereignisses B , so heißt das Ereignis B unabhängig vom Ereignis A . Es gilt: $P(B|A) = P(B)$.

Beim axiomatischen Aufbau der Wahrscheinlichkeitstheorie wird die Beziehung $P(B|A) = P(B)$ zur Definition der Unabhängigkeit des zufälligen Ereignisses B vom zufälligen Ereignis A benutzt:

Definition 3.4 (Unabhängigkeit)

Zwei Ereignisse A und B heißen unabhängig, wenn

$$P(B \cap A) = P(B|A) \cdot P(A) = P(B) \cdot P(A)$$

Man beachte den Unterschied zwischen den Begriffen „unabhängig“ und „disjunkt“. Sind beispielsweise die Ereignisse A und B unabhängig und gilt $P(A) > 0$ und $P(B) > 0$, dann können A und B nicht disjunkt sein.

3.3 Das Sprachkonstrukt der p-Konditionierung

Bisher haben wir nur über exakte Wahrscheinlichkeitswerte gesprochen, es soll im folgenden aber auch möglich sein, mit Wahrscheinlichkeitsintervallen zu rechnen. Unwissenheit in Form von Wahrscheinlichkeitsintervallen darzustellen, ist sinnvoll, da ein exakter Wahrscheinlichkeitswert nicht immer ausreicht, einen individuellen Überzeugungsgrad adäquat zu modellieren.

Zunächst führen wir nun das Sprachkonstrukt der probabilistischen Konditionierung (p-Konditionierung) ein. Dieses Sprachkonstrukt gestattet es, mit Unsicherheit behaftete Aussagen über Beziehungen zwischen Konzepten zu machen.

Definition 3.5 (Syntax der p-Konditionierung)

Seien K und K' Konzepte. Das Sprachkonstrukt

$$K \xrightarrow{[p_l, p_u]} K'$$

heißt p-Konditionierung genau dann, wenn $[p_l, p_u]$ ein Teilintervall der reellen Zahlen mit $0 \leq p_l \leq p_u \leq 1$ ist.

Definition 3.6 (Semantik der p-Konditionierung)

Sei \mathcal{D} eine endliche Domäne. Eine Extensionsfunktion \mathcal{E} über \mathcal{C} erfüllt eine p-Konditionierung $K \xrightarrow{[p_l, p_u]} K'$ genau dann, wenn

$$\frac{|\mathcal{E}[K \sqcap K']|}{|\mathcal{E}[K]|} \in [p_l, p_u] \quad (3.8)$$

für Konzepte $K, K' \in \mathcal{C}$ gilt, wobei $\mathcal{E}[K] \neq \emptyset$.

Kurzschreibweise: $\models_{\mathcal{E}} K \xrightarrow{[p_l, p_u]} K$

Das Sprachkonstrukt der p-Konditionierung stellt die Basis des probabilistischen Formalismus (PBox) von \mathcal{ALCP} dar.

Die Semantik der p-Konditionierung ist nicht definiert für unendliche Domänen und für p-Konditionierungen, die von Konzepten mit leerer Extension ausgehen. Deshalb entspricht es einer impliziten Existenzaussage, wenn eine p-Konditionierung $K \xrightarrow{[p_l, p_u]} K'$ ($p_l > 0$) durch eine Extensionsfunktion \mathcal{E} erfüllt wird.

Beispiel 3.2 Die p-Konditionierung $K \xrightarrow{[0.2, 0.8]} K'$ gibt an, daß zwischen 20% und 80% der Objekte aus der Extension von K auch Objekte der Extension von K' sind.

Nun folgen einige Definitionen, die nötig sind, um die in [Heinsohn 94a] entwickelten probabilistischen Constraints zur Berechnung nicht explizit angegebener p-Konditionierungen vorstellen zu können:

Definition 3.7 (Modelle von p-Konditionierungen)

Seien \mathcal{T} eine Terminologie und \mathcal{I} eine Menge von p-Konditionierungen.

1. Eine Extensionsfunktion \mathcal{E} erfüllt eine Menge \mathcal{I} von p-Konditionierungen genau dann, wenn \mathcal{E} jede p-Konditionierung $I \in \mathcal{I}$ erfüllt. ($\models_{\mathcal{E}} \mathcal{I}$)
2. Eine Menge E von Extensionsfunktionen erfüllt eine Menge \mathcal{I} von p-Konditionierungen genau dann, wenn jedes $\mathcal{E} \in E$ die Menge \mathcal{I} von p-Konditionierungen erfüllt. ($\models_E \mathcal{I}$)
3. Die Menge $\text{mod}_{\mathcal{T}}(\mathcal{I}) \stackrel{\text{def}}{=} \{\mathcal{E} \mid \models_{\mathcal{E}} \mathcal{I}\} \cap \text{mod}(\mathcal{T})$ heißt Menge der die p-Konditionierungen \mathcal{I} erfüllenden Extensionsfunktionen bez. \mathcal{T} . Man bezeichnet $\text{mod}_{\mathcal{T}}(\mathcal{I})$ auch als Menge der für p-Konditionierungen \mathcal{I} möglichen Modelle bez. \mathcal{T} .

Im folgenden wollen wir den Begriff der Konsistenz von p-Konditionierungen definieren:

Definition 3.8 (Konsistenz von p-Konditionierungen)

Eine Menge \mathcal{I} von p-Konditionierungen heißt genau dann konsistent, wenn $\text{mod}_{\mathcal{T}}(\mathcal{I}) \neq \emptyset$ gilt. Andernfalls heißt \mathcal{I} inkonsistent.

Definition 3.9 Seien \mathcal{T} eine Terminologie und \mathcal{I} eine konsistente Menge von p-Konditionierungen.

1. Die Menge

$$\text{Th}_{\mathcal{T}}(\mathcal{I}) \stackrel{\text{def}}{=} \{\mathcal{I} \mid \models_{\mathcal{E}} \mathcal{I} \text{ für alle } \mathcal{E} \in \text{mod}_{\mathcal{T}}(\mathcal{I})\}$$

heißt Gesamtmenge der bez. \mathcal{I} und \mathcal{T} folgerbaren p-Konditionierungen.

2. Im Falle ihrer Existenz heißt die Menge

$$\text{min}_{\mathcal{T}}(\mathcal{I}) \stackrel{\text{def}}{=} \bigcup_{C, D \in \mathcal{C}} \{C \xrightarrow{R_{\text{min}}} D \mid R_{\text{min}} = \bigcap_{R: C \xrightarrow{R} D \in \text{Th}_{\mathcal{T}}(\mathcal{I})} R\}$$

Gesamtmenge der bez. \mathcal{I} und \mathcal{T} minimalen p-Konditionierungen. R_{min} heißt dabei die für die p-Konditionierung von C nach D bez. \mathcal{I} und \mathcal{T} minimale Menge reeller Zahlen, $C \xrightarrow{R_{\text{min}}} D$ die bez. \mathcal{I} und \mathcal{T} minimale p-Konditionierung.

3. Eine p-Konditionierung I heißt genau dann korrekt berechnet bez. \mathcal{I} und \mathcal{T} , wenn $I \in \text{Th}_{\mathcal{T}}(\mathcal{I})$ gilt. I heißt genau dann minimal berechnet bez. \mathcal{I} und \mathcal{T} , wenn $I \in \text{min}_{\mathcal{T}}(\mathcal{I})$ gilt.
4. Ein Berechnungsverfahren für p-Konditionierungen heißt vollständig, wenn es für jede Terminologie \mathcal{T} und jede Menge \mathcal{I} die Menge $\text{min}_{\mathcal{T}}(\mathcal{I})$ berechnet. Insbesondere heißt das Berechnungsverfahren triangulär vollständig, wenn es für jede aus drei Konzepten bestehende Terminologie \mathcal{T} und jede auf \mathcal{T} beschränkte Menge vollständig ist.

Eine Mindestanforderung an das Berechnungsverfahren ist die Korrektheit der berechneten p-Konditionierungen.

3.4 Trianguläre Beschränkungen (ohne logischen Bezug)

Es werden nun probabilistische Beschränkungen (constraints) vorgestellt, mit denen nicht explizit angegebene p-Konditionierungen berechnet, Intervalle explizit angegebener p-Konditionierungen verfeinert und eine Inkonsistenz von p-Konditionierungen und damit die Inkonsistenz der gesamten Wissensbasis festgestellt werden können.

Man spricht von triangulären Beschränkungen, da sich die Constraints immer auf drei Konzeptausdrücke beziehen. Zwischen drei Konzeptausdrücken besteht kein logischer Bezug, wenn sich kein Konzeptausdruck als Negation eines anderen Ausdrucks oder als Konjunktion bzw. Disjunktion der beiden anderen Ausdrücke darstellen läßt.

Die folgenden Sätze stammen aus [Heinsohn 94a]. Diese Constraints sind auch in unserem Propagierungsalgorithmus implementiert.

Satz 3.7 (Constraint CONS-BAYES)

Seien A, B und C nichtleere Konzepte und

$\mathcal{I} = \{A \xrightarrow{[p_l, p_u]} C, A \xrightarrow{[q_l, q_u]} B, B \xrightarrow{[q'_l, q'_u]} A, C \xrightarrow{[p'_l, p'_u]} A, B \xrightarrow{[r_l, r_u]} C, C \xrightarrow{[r'_l, r'_u]} B, p'_l \neq 0, q_l \neq 0\}$ eine Menge von p-Konditionierungen.

\mathcal{I} ist inkonsistent, wenn für

$$r_l^* = r'_l \cdot \frac{p_l}{p'_u} \cdot \frac{q'_l}{q_u} \quad \text{und} \quad (3.9)$$

$$r_u^* = r'_u \cdot \frac{p_u}{p'_l} \cdot \frac{q'_u}{q_l} \quad (3.10)$$

die Ungleichung $[r_l^*, r_u^*] \cap [r_l, r_u] \neq \emptyset$ verletzt ist.

Das Constraint CONS-BAYES kann direkt aus dem Theorem von Bayes hergeleitet werden.

Satz 3.8 (Constraint CONS-TRIAN)

Seien A, B und C Konzepte und

$\mathcal{I} = \{A \xrightarrow{[p_l, p_u]} C, A \xrightarrow{[q_l, q_u]} B, B \xrightarrow{[q'_l, q'_u]} A, C \xrightarrow{[p'_l, p'_u]} A\}$ eine Menge von p-Konditionierungen. Die auf Basis dieses Wissens ableitbare und bezüglich \mathcal{I} minimale p-Konditionierung $B \xrightarrow{R_{min}} C$ hat das minimale Intervall R_{min} mit den Wahr-

scheinlichkeitsgrenzen r_l und r_u , die wie folgt berechnet werden:

$$r_l = \begin{cases} \frac{q'_l}{q_l} \cdot \max(0, q_l + p_l \Leftrightarrow 1) & \text{falls } q_l \neq 0 \\ q'_l & \text{falls } q_l = 0, p_l = 1 \\ 0 & \text{sonst} \end{cases} \quad (3.11)$$

$$r_u = \begin{cases} \min \left(1, 1 \Leftrightarrow q'_l + p_u \cdot \frac{q'_l}{q_l}, \frac{p_u}{p'_l} \cdot \frac{q'_u}{q_l}, \frac{p_u}{p'_l} \cdot \frac{q'_u}{q_l} \cdot (1 \Leftrightarrow p'_l) \right. \\ \quad \left. + q'_u, \frac{p_u}{p'_l \cdot (q_l - p_u) + p_u} \right) & \text{falls } p'_l \neq 0, q_l \neq 0 \\ \min \left(1, 1 \Leftrightarrow q'_l + p_u \cdot \frac{q'_l}{q_l} \right) & \text{falls } p'_l = 0, q_l \neq 0 \\ q'_u & \text{falls } p'_l = 1, q_l \neq 0 \\ 1 \Leftrightarrow q'_l & \text{falls } p_u = 0, q_l = 0 \\ 1 & \text{sonst} \end{cases} \quad (3.12)$$

Ist eine p -Konditionierung $B \xleftrightarrow{R} C$ bereits vorhanden, so liegt im Fall $R \cap R_{\min} = \emptyset$ eine Inkonsistenz vor, ansonsten berechnet sich das neue minimale Intervall aus diesem Durchschnitt.

3.5 Trianguläre Fälle (mit logischem Bezug)

Es werden nun die triangulären Fälle betrachtet, in denen sich ein Konzeptausdruck als Negation eines anderen Ausdrucks oder als Konjunktion bzw. Disjunktion der beiden anderen Konzeptausdrücke darstellen läßt.

Satz 3.9 (Konzeptnegation und Konzeptkonjunktion)

Für alle Konzepte $A, B, C \in \mathcal{C}$, $A, B, C \not\approx_{\mathcal{T}, \mathcal{I}} \perp$ gelten:

$$\begin{aligned} \left(A \xleftrightarrow{[p_l, p_u]} B \right) \in \min_{\mathcal{T}}(\mathcal{I}) &\Leftrightarrow \left(A \xleftrightarrow{[1-p_u, 1-p_l]} \neg B \right) \in \min_{\mathcal{T}}(\mathcal{I}) \\ \left(B \xleftrightarrow{[p_l, p_u]} \neg B \right) \in \min_{\mathcal{T}}(\mathcal{I}) &\Rightarrow p_u = 0 \\ \left(A \xleftrightarrow{[p_l, p_u]} C \right) \in \min_{\mathcal{T}}(\mathcal{I}) &\Leftrightarrow \left(A \xleftrightarrow{[p_l, p_u]} A \sqcap C \right) \in \min_{\mathcal{T}}(\mathcal{I}) \end{aligned}$$

Satz 3.10 (Konzeptdisjunktion)

Seien $A, B \in \mathcal{C}$.

Man erhält auf der Basis lokaler (triangulärer) Berechnungen:

$$B \xleftrightarrow{0} A : (A \sqcup B) \xleftrightarrow{[p_l, p_u]} A \Leftrightarrow (A \sqcup B) \xleftrightarrow{[1-p_u, 1-p_l]} B$$

Kapitel 4

Die Umsetzung des \mathcal{ALCP} - Konzeptes

Nach der Einführung in die Grundlagen von Wissensrepräsentation und Wahrscheinlichkeitstheorie und nach der Beschreibung der Sprache \mathcal{ALCP} , die in den letzten beiden Kapiteln erfolgt ist, wird nun die Umsetzung des \mathcal{ALCP} -Konzeptes in ein lauffähiges Programm erläutert.

Diese Umsetzung besteht aus drei Teilen: aus den verwendeten Datenstrukturen, aus der Schnittstelle zu einer existierenden Implementierung der Sprache \mathcal{ALC} und schließlich aus der Implementierung der Constraints und Entwicklung eines Algorithmus zur Propagierung der Constraints durch eine Wissensbasis. Die in Kapitel 3 beschriebenen Constraints CONS-BAYES und CONS-TRIAN wurden von uns um die beiden Constraints CONS-LOG, das logische Abhängigkeiten modellieren kann, und CONS-PINGUIN, welches notwendig ist, um leere Konzepte behandeln zu können, erweitert. Diese Teile werden in den nachfolgenden Kapiteln beschrieben.

In diesem Kapitel wird ein allgemeiner Überblick über die Implementierung der Sprache \mathcal{ALCP} gegeben, der sowohl die Syntax von T- und PBox als auch die verwendbaren Funktionen umfaßt.

4.1 Die Syntax von T- und PBox

Die Wissensbasis von \mathcal{ALCP} besteht aus terminologischem und unsicherem bzw. probabilistischem Wissen. Dieses Wissen ist in den sogenannten T- und PBoxen enthalten.

Zur Modellierung des terminologischen Wissens verwenden wir eine am DFKI entwickelte Implementierung der Sprache \mathcal{ALC} mit dem Namen KRIS (Knowledge Representation and Inference System, siehe [Baader & Hollunder 91]). Das KRIS-System ist ein terminologisches Wissensrepräsentationssystem, welches \mathcal{ALC} um-

faßt. Alle Sprachkonstrukte von \mathcal{ALC} sind also mit KRIS modellierbar. RAT (Representation of Actions in Terminological Logic, siehe [Heinsohn et al. 92]) stellt eine Erweiterung von KRIS dar, mit der Pläne dargestellt werden können. Eine Einführung in KRIS gibt [Profitlich 93].

Folgendes Beispiel erläutert die Syntax des von KRIS übernommenen terminologischen Wissens:

Beispiel 4.1 *Betrachten wir den in Kapitel 2 eingeführten Nixon-Diamant. Um die Inkonsistenz der Terminologie zu vermeiden, verwenden wir die folgenden terminologischen Axiome:*

$$\begin{aligned} \text{Pazifist} &\sqsubseteq \top \\ \text{Quäker} &\sqsubseteq \top \\ \text{Republikaner} &\sqsubseteq \top \\ R\&Q &\doteq \text{Republikaner} \sqcap \text{Quäker} \end{aligned}$$

Diese terminologischen Axiome werden in KRIS durch folgende TBox dargestellt:

```
(defprimconcept QUAEKER)
(defprimconcept PAZIFIST)
(defprimconcept REPUBLIKANER)
(defconcept R&Q (and QUAEKER REPUBLIKANER))
```

In Kapitel 2 wurde die Notwendigkeit von unsicherem Wissen anhand dieses Beipiels erläutert. Die Auflösung der Inkonsistenz und die Möglichkeit, sinnvolle Inferenzen zu erhalten, wurden durch die Einführung von unsicherem Wissen, den sogenannten p-Konditionierungen, und durch Constraints für diese p-Konditionierungen in Kapitel 3 gegeben. Die p-Konditionierungen sind innerhalb der Wissensbasis in der PBox enthalten. Die Fortführung des obigen Beispiels beschreibt die Einführung von p-Konditionierungen, die den Nixon-Diamanten sinnvoll modellieren:

Beispiel 4.2 *Zu der oben beschriebenen TBox führen wir das folgende unsichere Wissen ein:*

- *2/5 bis 3/5 aller Quäker sind Republikaner.*
- *Ungefähr ein Drittel aller Republikaner sind auch Quäker.*
- *Fast alle Quäker (mehr als 90 Prozent) sind Pazifisten.*

- *Aber nur wenige Republikaner (höchstens ein Viertel) sind Pazifisten.*

Diesem Wissen entsprechen die durch die folgende PBox gegebenen p-Konditionierungen:

(QUAEKER REPUBLIKANER 0.4 0.6)
 (REPUBLIKANER QUAEKER 0.33 0.34)
 (QUAEKER PAZIFIST 0.9 1)
 (REPUBLIKANER PAZIFIST 0 0.25)

Auf diese vollständige Beschreibung des Nixon-Diamanten können nun die Constraints angewendet werden. Das Ergebnis dieser Anwendung wird in Kapitel 8 beschrieben. Auf die Syntax der PBox wird nun genauer eingegangen: Eine PBox besteht aus einer Folge von Tupeln, für jede p-Konditionierung existiert ein Tupel in der PBox. Jedes Tupel besteht aus vier Elementen: zwei Konzepten und den beiden Werten, die die Intervallgrenzen für die p-Konditionierung zwischen diesen beiden Konzepten darstellen. Die Werte können sowohl Dezimalzahlen als auch Brüche sein. Der Wertebereich ist auf das Einheitsintervall begrenzt, die untere Grenze darf somit nicht kleiner als Null und die obere nicht größer als Eins sein.

Beispiel 4.3 *Es sei folgende p-Konditionierung zwischen den Konzepten K und K' gegeben:*

$$K \overset{[p,q]}{\rightleftarrows} K' \text{ mit } 0 \leq p \leq q \leq 1$$

Diese p-Konditionierung wird in der PBox durch folgenden Tupel repräsentiert:

$$(K \ K' \ p \ q)$$

Falls p-Konditionierungen in beide Richtungen existieren, d.h. eine p-Konditionierung von K nach K' und eine von K' nach K , so besteht die Möglichkeit, statt zweier Quadrupel ein Tupel mit sechs Elementen anzugeben. In diesem Tupel werden nach den beiden Konzeptnamen K und K' die Werte für die p-Konditionierung vom ersten Konzept zum zweiten Konzept angegeben und dann die Werte für die umgekehrte Richtung.

Beispiel 4.4 *Es seien folgende p-Konditionierungen gegeben:*

$$K \overset{[p,q]}{\rightleftarrows} K' \wedge K' \overset{[r,s]}{\rightleftarrows} K$$

Dann gibt es zwei Möglichkeiten, diese p-Konditionierungen in der PBox zu repräsentieren:

- durch zwei Tupel: $(K \ K' \ p \ q)$ und $(K' \ K \ r \ s)$
- durch ein Tupel: $(K \ K' \ p \ q \ r \ s)$

Die triviale p-Konditionierung $[0, 1]$, die völlige Unwissenheit ausdrückt, muß nicht eingegeben werden. Für jede nicht-spezifizierte p-Konditionierung wird das Intervall $[0, 1]$ als „Default“ verwendet. Somit besteht zwischen zwei verschiedenen Konzepten der Wissensbasis genau eine p-Konditionierung: entweder eine eingegebene bzw. berechnete oder die triviale.

4.2 Zwei wichtige Begriffe

In diesem Abschnitt sollen zwei Begriffe erklärt werden, die eine wesentliche Rolle bei der Auswahl der triangulären Fälle bilden.

p-Konditionierungen bestehen aus Intervallen, die eine Teilmenge des Einheitsintervalls $[0,1]$ sind. Hierbei ergeben sich zwei grobe Unterteilungen anhand der Art der Intervalle. Diese beiden Arten von p-Konditionierungen sind:

- **nichttriviale** p-Konditionierungen
- **interessante** p-Konditionierungen

Triviale p-Konditionierungen sind p-Konditionierungen, die dem Intervall $[0,1]$ entsprechen. Diese p-Konditionierungen repräsentieren völlige Unwissenheit. Aus der p-Konditionierung

$$K \xrightarrow{[0,1]} K'$$

folgt, daß zwischen 0 und 100 Prozent der Elemente der Extension des Konzeptes K auch Elemente der Extension des Konzeptes K' sind, und dies entspricht gerade dem Fehlen jeglicher Informationen über die Beziehungen zwischen K und K' .

Interessante p-Konditionierungen werden durch alle Intervalle repräsentiert, die nichttrivial und ungleich dem Intervall $[1,1]$ sind. D.h. jedes Intervall, das nicht $[0,1]$ oder $[1,1]$ ist, stellt eine interessante p-Konditionierung dar. Dies entspricht dem Wissen, daß Information über die Beziehung der beiden Konzepte vorhanden ist, aber keine Subsumtionsbeziehung vorliegt.

4.3 Zahldarstellung

Durch die wahrscheinlichkeitstheoretische Fundierung werden für p-Konditionierungen nur Zahlen aus dem Einheitsintervall $[0, 1]$ verwendet. Die Eingabe von Werten kann wahlweise durch Dezimalzahlen oder durch Brüche erfolgen, die

im Einheitsintervall liegen müssen. Bei der internen Darstellung werden ausschließlich Brüche verwendet. Dies wird bedingt durch die Verwendung der Programmiersprache LISP. In LISP treten bei der Verwendung von Dezimalzahlen sehr schnell Rundungsfehler auf, so daß eine Verwendung von Dezimalzahlen bei rechenintensiven und an eine hohe Genauigkeit gebundenen Anwendungen ausscheidet. Die Genauigkeit der Rechenergebnisse ist bei der Umsetzung eines probabilistischen Konzeptes sehr wichtig, da z.B. $1/3 + 2/3$ immer Eins ergeben muß (wichtig z.B. für die Anwendung der Constraints oder für die Feststellung von logischen Abhängigkeiten).

Durch die ausschließliche Verwendung von Brüchen kann eine Konvertierung der Eingabe notwendig sein. Bei eingegebenen Zahlen wird zuerst geprüft, ob sie im Einheitsintervall liegen. Dezimalzahlen werden anschließend in Brüche umgewandelt, dabei wird an der neunten Nachkommastelle gerundet. Die Einstellung auf die neunte Nachkommastelle ist veränderbar und wird durch die globale Variable ***nachkommastellen*** gesteuert.

Nach Bereichsüberprüfung und Konvertierung von eingegebenen Zahlen wird noch geprüft, ob die untere Grenze kleiner oder gleich der oberen Grenze ist.

Die Ausgabe von p-Konditionierungen erfolgt in Dezimalzahlen bzw. Brüchen. Dabei wird während der Ausgabe von Dezimalzahlen bei der Konvertierung an der dritten Nachkommastelle gerundet. Diese Einstellung kann ebenso wie ***nachkommastellen*** geändert werden. Das genaue Vorgehen wird im Abschnitt über globale Variablen im Kapitel „Datenstrukturen“ beschrieben.

4.4 Allgemeiner Überblick über den Algorithmus

In diesem Abschnitt wird beschrieben, wie die Struktur der Anwendung der probabilistischen Constraints auf eine Wissensbasis ist.

Zu Beginn muß eine terminologische Wissensbasis (TBox) eingelesen werden. Dieses terminologische Wissen wird in p-Konditionierungen umgewandelt. Hierbei ergeben sich zwei verschiedene Arten von p-Konditionierungen:

- Subsumtionen: wird ein Konzept K von einem Konzept K' subsumiert, so wird daraus die p-Konditionierung $[1, 1]$ von K nach K' , eine sogenannte *Subsumtionskante*.
- disjunkte Konzepte: sind die Konzepte K und K' disjunkt, so ergibt dies die p-Konditionierung $[0, 0]$ sowohl von K nach K' als auch von K' nach K . Diese p-Konditionierungen werden *Nullkanten* genannt.

Weiterhin wird festgestellt, ob ein Konzept leer ist. In Kapitel 5.4 wird beschrieben, wie dies erkannt werden kann. Ist dies der Fall, wird es in eine Liste mit

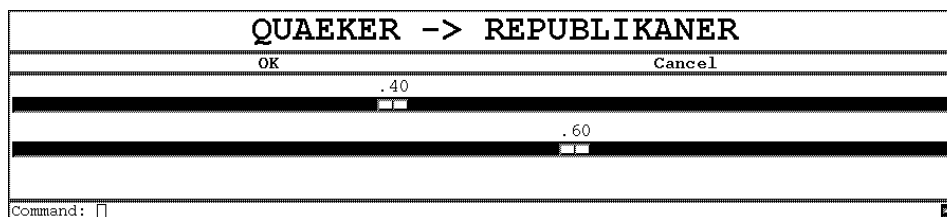


Abbildung 4.1: Der Slider

leeren Konzepten eingetragen, und es werden folgende Kanten gezogen: Subsumtionskanten von und zu allen anderen leeren Konzepten (z.B. *BOTTOM*), und eine Nullkante von *TOP* zu dem leeren Konzept. Von dem leeren Konzept nach *TOP* besteht bereits eine Subsumtionskante. Anschließend wird die probabilistische Wissensbasis (PBox) eingelesen oder die p-Konditionierungen werden eingegeben.

Das geschieht entweder durch eine Eingabe über die Tastatur, wobei es dem Benutzer überlassen bleibt, ob er die Zahlenwerte als Bruch oder als Dezimalzahl eingibt. Bei Benutzung der CLIM-Oberfläche besteht auch die Möglichkeit, p-Konditionierungen über einen sogenannten SLIDER (Schieberegler, siehe Abb. 4.1) einzugeben oder zu modifizieren. Allerdings nur bis zu einer Genauigkeit von zwei Nachkommastellen.

Die in Kapitel 3 vorgestellten Constraints und das Constraint CONS-LOG werden immer auf je drei Konzepte angewendet. Diese drei Konzepte bilden einen triangulären Fall. Die Anwendung der Constraints auf einen triangulären Fall ist rechenintensiv, so daß diese Anwendung auf die Fälle eingeschränkt werden sollte, die auch wirklich zu einem Ergebnis führen könnten. Bei einer Wissensbasis mit n Konzepten, gibt es $O(n^3)$ verschiedene trianguläre Fälle, die teilweise mehrmals betrachtet werden müssen, bis die gesamte Information durch den Graphen diffundiert ist. Es werden daher nur die triangulären Fälle betrachtet, die wirklich neue Informationen liefern können. Im Kapitel über Constraints und ihre Propagierung wird auf diesen Aspekt genauer eingegangen. Für diese Übersicht ist es ausreichend zu wissen, daß es einen „intelligenten“ Algorithmus gibt, der die Anzahl der betrachteten triangulären Fälle einschränkt.

Es werden nun alle nötigen triangulären Fälle nacheinander betrachtet. Dabei wird überprüft, ob eines der drei Konzepte leer ist. Ist dies der Fall, so wird der entsprechende Fall nicht weiter betrachtet. Zum Erkennen leerer Konzepte wird das Constraint CONS-PINGUIN verwendet. Bei jedem triangulären Fall werden die Constraints CONS-BAYES, CONS-TRIAN und CONS-LOG auf die drei Konzepte angewendet. Diese Anwendung geschieht solange, bis die Werte stabil sind und eine weitere Anwendung keine neuen Informationen bringt. Zwischen drei Konzepten gibt es sechs p-Konditionierungen. CONS-BAYES und CONS-TRIAN

berechnen jeweils ein Intervall (also eine p-Konditionierung) und müssen somit pro Durchlauf sechsmal angewendet werden. CONS-LOG muß höchstens einmal pro Durchlauf des triangulären Falles angewendet werden. Nach spätestens zwei Durchläufen sind die Werte stabil, so daß keine Endlosschleife auftreten kann. Wenn sich während dieser Anwendung der Constraints Werte verändern, bleiben die betreffenden p-Konditionierungen interessant und mit ihnen werden weitere trianguläre Fälle gebildet. Ansonsten wird dieser Fall nicht mehr betrachtet. Nach der Abarbeitung der nötigen triangulären Fälle sind alle möglichen Inferenzen aus dem bereitgestellten Wissen gezogen und ein erneutes Starten der Propagierung würde keine neuen Informationen liefern.

Einen Überblick über die abstrakte Systemarchitektur gibt Abb. 4.2.

Es besteht nun die Möglichkeit, eine p-Konditionierung zu ändern und zu betrachten, welche neuen Inferenzen daraus entstehen. Außerdem kann man sich ansehen, wie die Intervalle verfeinert wurden, indem man die sogenannte „History“ betrachtet.

4.5 Verwendbare Funktionen

In diesem Abschnitt sollen die Funktionen beschrieben werden, die der Benutzer verwenden kann. Eine Bedienungsanleitung, die die Benutzung von *ALCP* spezifiziert, befindet sich am Ende dieser Dokumentation in Kapitel 9. Interne Funktionen werden in den nachfolgenden Kapiteln beschrieben.

Dem Benutzer werden vier Gruppen von Funktionen zur Verfügung gestellt, die die folgenden Kontexte betreffen:

- Wissensbasen
- p-Konditionierungen
- Propagierung
- History und Trace

4.5.1 Behandlung von Wissensbasen

Zur Behandlung von Wissensbasen stehen fünf Funktionen zur Verfügung:

- (*t-load* “<file>“)

Diese Funktion lädt eine existierende TBox mit dem Namen <file>. Eine TBox sollte die Extension .tbox haben. Diese Funktion lädt nicht nur die spezifizierte TBox, sondern extrahiert außerdem das enthaltene terminologische Wissen und wandelt es in p-Konditionierungen um. Weiterhin

Abstrakte Systemarchitektur

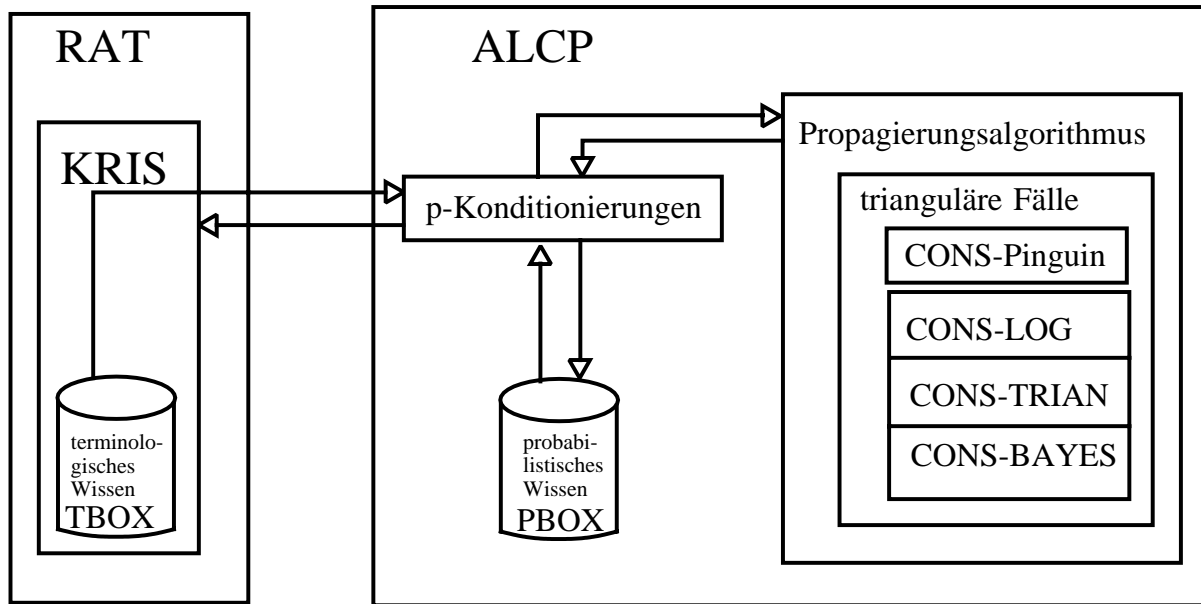


Abbildung 4.2: Die abstrakte Systemarchitektur von ALCP

wird eine Liste der existierenden Konzepte und eine Liste der leeren Konzepte angelegt. Im Kapitel über die Schnittstelle zu KRIS/RAT werden diese Aktionen ausführlich erläutert.

- (*p-load* “<file>“)

Hiermit kann eine existierende PBox geladen werden. Die PBox sollte die Extension .pbox haben. Die Syntax der PBox muß den in Kapitel 4.1 aufgeführten Spezifikationen genügen. Die beiden Konzepte einer p-Konditionierung werden daraufhin überprüft, ob sie in der Liste der existierenden Konzepte enthalten sind. Ist dies nicht der Fall, so wird eine Fehlermeldung ausgegeben. Somit können nur p-Konditionierungen zu bereits existierenden (mit einer TBox erzeugten) Konzepten eingelesen werden. Schließlich wird noch eine Kopie der eingelesenen p-Konditionierungen angelegt, die für die Funktionen, die Fragen über das ursprüngliche Wissen beantworten sollen, sowie für die „History“ gebraucht wird.

- (*p-save* “<file>“)

Die p-Konditionierungen werden in einer PBox gespeichert, die die Endung .pbox haben sollte. Dazu werden die p-Konditionierungen in eine Folge von Tupeln übersetzt, die der oben angegebenen Syntax für PBoxen genügen.

- (*detect-tbox-knowledge*)

Nach der Propagierung der Constraints durch das terminologische und probabilistische Wissen besteht die Möglichkeit, daß neues terminologisches Wissen gewonnen wurde, d.h. es können Subsumtionen oder Disjunktheiten inferiert werden, die nicht in der TBox enthalten sind. Diese Funktion zeigt dieses neue terminologische Wissen an. Die Feststellung dieses zusätzlichen terminologischen Wissens erfolgt durch Vergleich der erhaltenen p-Konditionierungen mit Antworten des KRIS/RAT-Systems auf die Frage nach Subsumtion bzw. Disjunktheit.

- (*clear*)

Mit dieser Funktion ist es möglich, sämtliche Hashtabellen gleichzeitig zurückzusetzen und neu zu initialisieren. Dabei wird das gesamte Wissen über PBox und „History“ gelöscht. Übrig bleibt nur das terminologische Wissen der aktuellen TBox. Anschließend kann man eine neue PBox laden oder p-Konditionierungen eingeben. Will man PBox und TBox löschen, so genügt es, eine neue TBox zu laden. Dabei geht das gesamte vorherige Wissen verloren.

4.5.2 Behandlung von p-Konditionierungen

Hierzu gehören die beiden Bereiche Einfügen und Anzeigen von p-Konditionierungen.

- (*insert-new-range* <konzept1> <konzept2> p q *Optional* p' q')

Zwischen <konzept1> und <konzept2> wird eine p-Konditionierung [p,q] eingefügt. Gleichzeitig kann optional ein Intervall [p',q'] für die Rückrichtung angegeben werden. Hierbei ist zu beachten, daß beim Einfügen neuer Werte Inkonsistenzen entstehen können. Insbesondere sollte man nach der Propagierung ein berechnetes Intervall nicht so modifizieren, daß ein dazu disjunktes Intervall entsteht.

Zum Anzeigen der p-Konditionierungen stehen acht verschiedene Funktionen zur Verfügung:

- (*given-ranges*)

Vorgegebene Werte werden angezeigt. Vor der ersten Propagierung sind das die durch TBox und PBox eingelesenen sowie die vom Benutzer von Hand eingegebenen Werte. Wurde bereits propagiert, so zeigt (*given-ranges*) die Startwerte der letzten Propagierung an.

- (*given-by-pbox*)

Diese Funktion zeigt vorgegebenes probabilistisches Wissen, d.h. die durch die PBox gegebenen p-Konditionierungen, an.

- (*given-by-tbox*)

Hiermit wird - analog zu den beiden vorigen Funktionen - vorgegebenes Wissen angezeigt, eingeschränkt jedoch auf terminologisches Wissen (also Subsumtionen und Disjunktheiten).

- (*new-ranges*)

Werte, die neu eingegeben oder bei der letzten Propagierung verändert wurden, werden angezeigt.

- (*all-1-ranges*)

Diese Funktion zeigt sämtliche Subsumtionen an.

- (*all-0-ranges*)

Hiermit werden alle paarweise disjunkten Konzepte ausgegeben.

- (show-range <konzept1> <konzept2>)

Hiermit kann sich der Benutzer die p-Konditionierung einer speziellen Kante, nämlich (<konzept1> <konzept2>), anzeigen lassen.

4.5.3 Propagierung

- (*start-propagation*)

Der Propagierungsalgorithmus wird initialisiert und gestartet. Dabei wird nur über die gerade in der ***agenda*** befindlichen Kanten propagiert. Dadurch kann Laufzeit eingespart werden.

4.5.4 „History“ und „Trace“

Um die internen Abläufe und Berechnungen transparent zu halten, gibt es zwei verschiedene Arten von Protokollen: zum einen die „History“, bei der mittels einer Hashtabelle zu jeder Kante die Berechnung und Verfeinerung ihrer beiden p-Konditionierungen gespeichert wird. Hier stehen dem Benutzer zwei Funktionen zur Verfügung:

- (*history-range <konzept1> <konzept2>*)

Damit wird die „History“ der Kante (<konzept1> <konzept2>) angezeigt. Diese Information umfaßt die p-Konditionierungen in beide Richtungen und zeigt sämtliche Änderungen seit dem letzten (clear) an.

- (*maphistory*)

Hier werden sämtliche Kanten, zu denen eine „History“ existiert, analog zur vorherigen Funktion aufgelistet.

Zum anderen gibt es ein während des Programmablaufs erstelltes „Trace“. Will sich der Benutzer nicht nur über die Veränderung einzelner Kanten informieren, sondern auch einen Eindruck von den Zusammenhängen und Wechselwirkungen der einzelnen Berechnungen verschaffen, so bietet ihm das Trace umfassende Information. Hierzu gibt es drei Funktionen:

- (*trace-on*)

Das „Trace“ wird eingeschaltet und auf dem Bildschirm ausgegeben.

- (*trace-off*)

Das „Trace“ wird ausgeschaltet. Dies ist die standardmäßige Einstellung.

- (*trace-in* “<filename>“)

Das „Trace“ wird in die Datei <filename> umgeleitet. Das ist insbesondere bei längeren Berechnungen zu empfehlen.

Kapitel 5

Die KRIS/RAT-Schnittstelle

Die Sprache \mathcal{ALC} wurde in Kapitel 2 als eine mögliche terminologische Repräsentationssprache eingeführt. Aufbauend auf dieser Grundlage wurde eine probabilistische Komponente von [Heinsohn 94a] hinzugefügt.

Die Systeme KRIS und RAT (im folgenden „KRIS/RAT“ genannt) sind am DF-KI entwickelte Systeme, welche die Sprache \mathcal{ALC} umfassen. Ziel dieser Arbeit war es, das System RAT (und damit auch KRIS) um die beschriebene probabilistische Komponente zu erweitern.

Aufgrund dieses Zusammenhanges verwenden wir einige Funktionen, die das KRIS/RAT-System zum Umgang mit der terminologischen Struktur einer Domäne bereitstellt. Die Verwendung dieser Funktionen wird im folgenden erklärt.

Eine Einführung in den Umgang mit terminologischem Wissen im Rahmen von KRIS/RAT wird in [Profitlich 93] gegeben. Wir beschränken uns daher hier auf die für \mathcal{ALCP} wichtigen Funktionen.

Die in den nachfolgenden Abschnitten verwendeten Hashtabellen und Listen werden im nächsten Kapitel, das die Datenstrukturen vorstellt, ausführlich erläutert.

5.1 Übernahme der Terminologie

Die Terminologie einer Domäne kann mit Hilfe von KRIS/RAT modelliert und in einer Datei abgespeichert werden. Diese TBox enthält die Namen aller erzeugten Konzepte sowie kategorische Zusammenhänge zwischen diesen Begriffen (Taxonomie). Die Übernahme dieser Information geschieht mittels der von KRIS/RAT zur Verfügung gestellten Funktion `'TL-load-tbox'`, mit der eine existierende TBox geladen wird.

5.2 Klassifizierung

Die durch eine TBox repräsentierte Wissensbasis enthält neben dem durch den Benutzer eingegebenen Wissen weitere implizite Informationen. Diese Informa-

tionen kann man durch Inferenz aus der TBox erhalten. Für die Sprache \mathcal{ALC} existieren korrekte und vollständige Inferenzalgorithmen (vgl. [Smolka et al. 89]), die aus dem expliziten und impliziten Wissen einer TBox eine auf der Subsumtionsrelation beruhende Hierarchie aufbauen.

Diese Klassifizierung des Wissens muß vor dem Start des Propagierungsalgorithmus erfolgen, da zum einen Subsumtionen spezielle p-Konditionierungen darstellen: wird ein Konzept K von einem Konzept K' subsumiert, so existiert eine Kante mit der p-Konditionierung $[1,1]$ (d.h. eine Subsumtionskante) zwischen dem Konzept K und dem Konzept K' . Zum anderen werden p-Konditionierungen auch durch disjunkte Konzepte erzeugt: sind Konzept K und Konzept K' disjunkt, so befinden sich zwischen diesen beiden Konzepten Kanten mit der p-Konditionierung $[0,0]$, sogenannte Nullkanten. Die Bestimmung dieser p-Konditionierungen wird im nächsten Abschnitt beschrieben.

Die Klassifizierung des terminologischen Wissens erfolgt im KRIS/RAT-System durch die Funktion *'TL-classify-tbox'*, die eine gegebene TBox klassifiziert. KRIS/RAT bietet die Möglichkeit, verschiedene Kommentare während des Klassifizierungsvorgangs anzuzeigen. In der Implementierung von \mathcal{ALCP} wurden diese Kommentare auf ein Minimum begrenzt, es werden daher nur Warnungen, z.B. über Inkonsistenzen, ausgegeben. Der Grund für diese Restriktion der Ausgabe liegt in der Intention von \mathcal{ALCP} . \mathcal{ALCP} wurde speziell für probabilistisches Wissen entwickelt, so daß nur die aus der Terminologie resultierende Hierarchie wichtig ist. Es wird also eine konsistente terminologische Modellierung der Domäne vorausgesetzt, da im anderen Fall p-Konditionierungen nichts an der Inkonsistenz ändern würden. Für ausschließlich terminologisches Wissen sollte weiterhin KRIS bzw. RAT verwendet werden.

Nach der Klassifizierung wird eine Liste ***konzepte*** angelegt, in der alle Konzepte der Domäne stehen.

5.3 Subsumtions- und Nullkanten

Eine Subsumtionskante zwischen einem Konzept K und einem Konzept K' ist eine Kante von K nach K' mit der p-Konditionierung $[1,1]$. Diese Kante gibt an, daß Konzept K von Konzept K' subsumiert wird. Sind dagegen zwei Konzepte disjunkt, so wird eine Nullkante, d.h. eine Kante mit p-Konditionierung $[0,0]$, zwischen den beiden Konzepten gezogen. Diese Kante ist ungerichtet, die p-Konditionierung $[0,0]$ besteht somit sowohl zwischen K und K' als auch zwischen K' und K .

Um die Subsumtionskanten festzustellen, betrachtet man die Liste aller Konzepte ***konzepte***. Zu jedem Konzept K dieser Liste bestimmt man durch *'all-sub K'* die Liste aller von K subsumierten Konzepte. Für jedes Element K' der so erhaltenen Liste trägt man die Kante $K' \rightarrow K$ mit der p-Konditionierung $[1,1]$ in die Hashtabelle ***p-Konditionierungen*** ein. Weiterhin wird in Hashtabelle

verbundene-Konzepte eingetragen, daß die Konzepte K und K' durch eine Kante ungleich dem Intervall $[0,1]$ verbunden sind. Auf die verwendeten Datenstrukturen wird im nächsten Kapitel noch genauer eingegangen.

Zu je zwei Konzepten K und K' wird bei der Feststellung der Nullkanten geprüft, ob die Beschreibung des Konzeptes $K'' := K \sqcap K'$ konsistent ist. Wenn diese Beschreibung inkonsistent ist und keines der Konzepte K und K' äquivalent mit ***BOTTOM*** ist, so sind K und K' disjunkt. Die Nullkante $[0,0]$ wird in Hashtabelle ***p-Konditionierungen*** eingetragen und die Konzepte K und K' werden in Hashtabelle ***verbundene-Konzepte*** als verbunden markiert.

Sind die Konzepte K und K' disjunkt, so sind alle von K subsumierten Konzepte disjunkt zu allen Konzepten, die von K' subsumiert werden. Somit folgt, daß Nullkanten propagiert werden müssen. Seien also K und K' disjunkte Konzepte. Zu K und K' werden zwei Listen L und L' mit Hilfe der Funktion 'all-subst' erzeugt, die die von K bzw. K' subsumierten Konzepte enthalten. Zwischen jedes von ***BOTTOM*** verschiedene Element von L und jedes von ***BOTTOM*** verschiedene Element von L' wird eine Nullkante gelegt.

Beispiel 5.1 Gegeben seien die folgenden terminologischen Axiome:

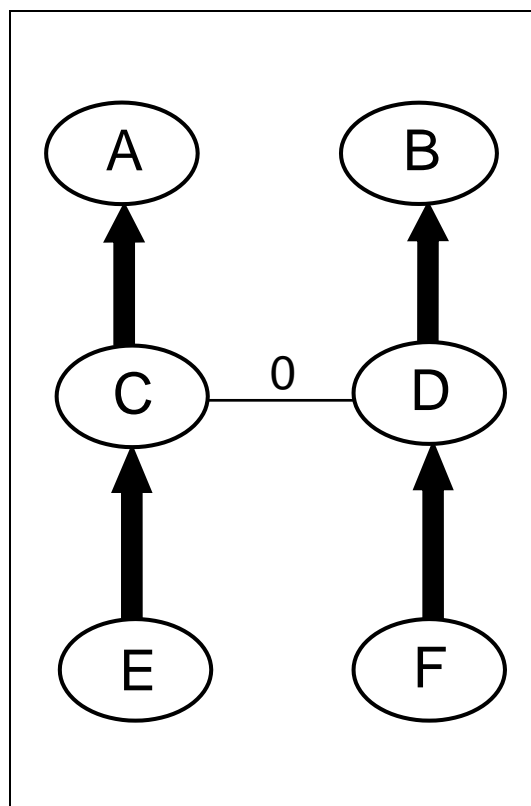
$$\begin{aligned} A &\sqsubseteq \top \\ B &\sqsubseteq \top \\ C &\sqsubseteq A \\ D &\doteq B \sqcap \neg C \\ E &\sqsubseteq C \\ F &\sqsubseteq D \end{aligned}$$

Daraus ergeben sich nach der Klassifizierung die folgenden Subsumtionskanten:

- $C \xleftrightarrow{[1,1]} A, E \xleftrightarrow{[1,1]} C, E \xleftrightarrow{[1,1]} A$
- $D \xleftrightarrow{[1,1]} B, F \xleftrightarrow{[1,1]} D, F \xleftrightarrow{[1,1]} B$

Da die Konzepte C und D disjunkt sind, ergeben sich die folgenden Nullkanten:

- $C \xleftrightarrow{[0,0]} D$
- $E \xleftrightarrow{[0,0]} D$
- $F \xleftrightarrow{[0,0]} C$
- $E \xleftrightarrow{[0,0]} F$



Man sieht, daß die Nullkanten durch die Hierarchie propagiert wurden.
Die Liste ***konzepte*** wird wie folgt initialisiert:

(A B C D E F *TOP* *BOTTOM*).

Die Hashtabelle ***p-Konditionierungen*** enthält genau die oben als Subsumtions- und Nullkanten bezeichneten p -Konditionierungen. In der Hashtabelle ***verbundene-Konzepte*** werden folgende Konzepte als verbunden markiert:

- mit A : C, E
- mit B : D, F
- mit C : A, D, E, F
- mit D : B, C, E, F
- mit E : A, C, D, F
- mit F : B, C, D, E

Nullkanten sind ungerichtet, d.h. wird eine Kante mit dem Intervall $[0,0]$ von Konzept K zu Konzept K' in Hashtabelle ***p-Konditionierungen*** eingetragen,

so wird die Kante von K' nach K - wenn möglich - auch auf $[0,0]$ verfeinert. Ist hingegen die untere Grenze der Kante $K' \rightarrow K$ größer Null, so muß die Extension des Konzeptes B leer sein. Leere Extensionen von Konzepten (oder kurz: leere Konzepte) werden im nächsten Abschnitt behandelt.

5.4 Leere Konzepte

Ein Konzept heißt genau dann leer, wenn seine Extension leer ist. (siehe Def. 2.7)

Es gibt drei verschiedene Möglichkeiten, leere Konzepte zu erkennen. Die einfachste Möglichkeit ist eine Anfrage an das KRIS/RAT-System, welche Konzepte äquivalent zu *BOTTOM* sind. Zwei weitere Möglichkeiten ergeben sich durch die p-Konditionierungen. Zum einen die direkte: Es gibt eine Nullkante von Konzept K nach Konzept K' , aber die Kante von K' nach K ist mit einem Intervall markiert, das die Null nicht enthält. Dann ist das Konzept K leer. Zum anderen die indirekte Möglichkeit: Konzept K wird von Konzept K' subsumiert, Konzept K' wird von Konzept K'' subsumiert, und von Konzept K'' nach Konzept K existiert eine Nullkante. Dann ist Konzept K ebenfalls leer.

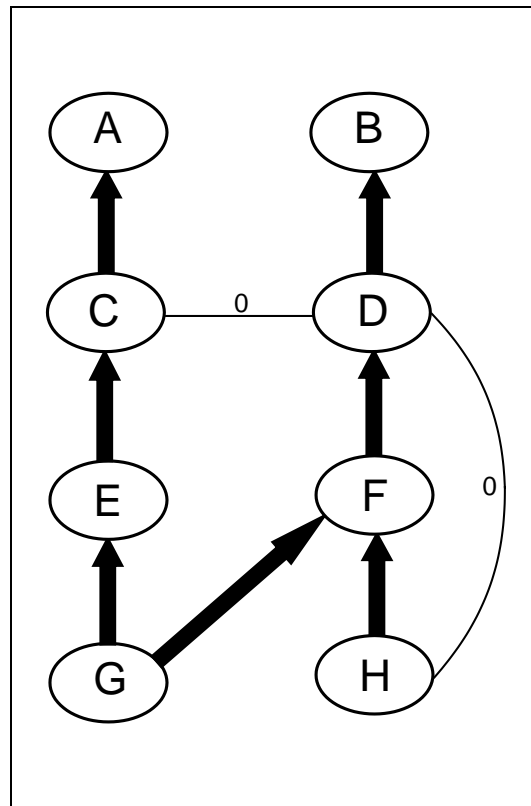
Während des Ablaufs des Propagierungsalgorithmus werden neu erkannte leere Konzepte in einer Liste ***leere-Konzepte*** gespeichert. Ansonsten werden sie „übergangen“, d.h. wenn in einem triangulären Fall ein leeres Konzept auftritt, wird dieser Fall nicht weiter behandelt. Nach dem Ablauf der Propagierung werden von jedem leeren Konzept zu jedem anderen leeren Konzept Subsumtionskanten eingetragen (da alle leeren Konzepte äquivalent sind). Desweiteren werden von jedem leeren Konzept Subsumtionskanten zu jedem nichtleeren Konzept eingetragen (da jedes leere Konzept von jedem nichtleeren Konzept subsumiert wird), und von jedem nichtleeren Konzept zu jedem leeren Konzept eine Nullkante eingetragen.

Beispiel 5.2 *Es seien die terminologischen Axiome aus dem letzten Beispiel gegeben. Diese Axiome seien durch folgende Axiome erweitert:*

$$\begin{aligned} G &\doteq E \sqcap F \\ H &\doteq F \sqcap \neg D \end{aligned}$$

Nach der Klassifizierung erkennt der Algorithmus, daß G und H leere Konzepte sein müssen, da gilt:

- $G \xrightarrow{[1,1]} F \wedge G \xrightarrow{[0,0]} F$
- $H \xrightarrow{[1,1]} F \xrightarrow{[1,1]} D \wedge H \xrightarrow{[0,0]} D$



Alle Elemente der Extension von G sind nach Definition des Konzepts G auch Elemente der Extension von F . Die Klassifizierung ergibt, daß alle Elemente der Extension von G auch in den Extensionen von C und D sind. Die Extensionen von C und D sind aber disjunkt, haben also keine gemeinsamen Elemente. Die einzige Möglichkeit, eine Inkonsistenz zu vermeiden, ist somit, daß die Extension von G keine Elemente enthält. G ist also ein leeres Konzept.

Kapitel 6

Datenstrukturen

Programme bestehen aus zwei Teilen: dem eigentlichen Algorithmus und den Datenstrukturen, die dieser Algorithmus benutzt. Es gibt viele verschiedene Möglichkeiten, die Daten eines Programmes zu verwalten. Wegen der gegenseitigen Abhängigkeit von Datenstruktur und Algorithmus und aus Effizienzgründen ist die Wahl von geeigneten Datenstrukturen sehr wichtig. Dieses Kapitel erläutert, welche Datenstrukturen bei der Implementierung von \mathcal{ALCP} verwendet wurden, und die Gründe für diese Wahl werden dargelegt. Im nächsten Kapitel wird dann detailliert auf den Algorithmus eingegangen.

Der nächste Abschnitt erläutert die Gründe, warum wir uns für die verwendeten Datenstrukturen entschieden haben. Danach soll in drei weiteren Abschnitten auf die verschiedenen Datenstrukturen detailliert eingegangen werden.

6.1 Die Wahl der geeigneten Datenstrukturen

Datenstrukturen besitzen zwei wesentliche Effizienz-Charakteristika, die für ihren Einsatz von entscheidender Bedeutung sind:

- **belegter Speicherplatz:** Datenstrukturen unterscheiden sich sehr stark in der Belegung von Speicherplatz. Bei umfangreichen Daten ist man bestrebt, die Daten möglichst platzsparend zu speichern und Redundanz in den Daten zu vermeiden.
- **Zugriffsgeschwindigkeit:** Die Art der verwendeten Datenstruktur bestimmt die Zeit, die für einen Zugriff auf die gespeicherten Daten benötigt wird. Ist ein häufiger Zugriff auf die Daten notwendig, so sollte die Zugriffszeit möglichst gering sein.

Die Algorithmen für die Propagierung der Constraints durch die Wissensbasis und die Abarbeitung eines triangulären Falles, auf die im nächsten Kapitel genauer eingegangen wird, bedingen eine hohe Anzahl an Suchvorgängen. Diese treten bei den folgenden Operationen auf:

- Wahl eines passenden dritten Konzeptes zu zwei gegebenen, um einen triangulären Fall zu bilden.
- Bestimmung der p-Konditionierungen zwischen zwei Konzepten.
- Bestimmung der logischen Abhängigkeiten in einem triangulären Fall.
- Bestimmung der „History“ einer p-Konditionierung.

Unter der „History“ einer p-Konditionierung versteht man eine Struktur, die die Entstehungsgeschichte des aktuellen Wertes der p-Konditionierung angibt. Die Bestimmung der „History“ einer p-Konditionierung ist unabhängig vom Propagierungsalgorithmus. Diese Operation tritt nur bei der Beantwortung von Benutzeranfragen zur Entwicklung einer bestimmten p-Konditionierung auf.

Während des Ablaufs des Propagierungsalgorithmus werden zur Bildung eines triangulären Falles und zur Bestimmung der Informationen, die über diesen triangulären Fall bereits bekannt sind, die ersten drei Operationen sehr häufig benötigt. Beispielsweise bestehen zwischen drei Konzepten sechs p-Konditionierungen, d.h. pro triangulärem Fall müssen diese sechs p-Konditionierungen bestimmt werden. Aufgrund dieser hohen Anzahl an Suchvorgängen ist die Verwendung einer Datenstruktur, die ein möglichst schnelles Suchen erlaubt, für die oben beschriebenen Operationen sinnvoll. Wir haben uns dazu entschieden, Hashtabellen für diese Operationen zu verwenden. Die Wahl einer Hashtabelle zur Bestimmung der „History“ einer p-Konditionierung erfolgte, um auf Benutzeranfrage möglichst schnell reagieren zu können. Prinzipiell wäre hier auch eine andere Datenstruktur möglich, dies würde aber zu einer höheren Antwortzeit führen.

Hashtabellen ermöglichen eine sehr effiziente Suche nach den in ihnen gespeicherten Elementen. Die Elemente der Hashtabelle werden dazu mit einem eindeutigen Schlüssel versehen, der einen sehr schnellen Zugriff auf das Element ermöglicht, welches mit diesem Schlüssel verknüpft ist. Auf die interne Struktur von Hashtabellen soll hier nicht eingegangen werden, da dies zum einen in jedem besseren Buch über Programmieretechniken nachgelesen werden kann und zum anderen Hashtabellen direkt von LISP bereitgestellt werden. Wichtig ist nur die Unterscheidung zwischen Schlüsseln und Elementen. Die Elemente sind die eigentlichen Daten und die Schlüssel dienen dazu, auf diese Elemente effizient zuzugreifen.

Mengen können kanonisch durch Listen repräsentiert werden. Listen sind in der Programmiersprache LISP die grundlegende Datenstruktur. Zur Behandlung von Listen stehen vielfältige Funktionen zur Verfügung. Daher liegt die Verwendung von Listen für die Datenstrukturen nahe, die Mengen repräsentieren sollen.

Weiterhin verwenden wir noch einige globale Variablen, um bestimmte Systemparameter zu kontrollieren.

In den nachfolgenden Abschnitten werden nacheinander die verwendeten Hashtabellen, Listen und globalen Variablen beschrieben.

6.2 Die verwendeten Hashtabellen

In diesem Abschnitt sollen die von uns konzipierten Hashtabellen beschrieben werden.

Hashtabelle *verbundene-Konzepte*

Schlüssel: Konzept

Rückgabewert: Liste der verbundenen Konzepte

Diese Hashtabelle enthält zu jedem Konzept eine Liste aller Konzepte, die mit diesem Konzept über eine nichttriviale p-Konditionierung verbunden sind.

Als Schlüssel für die Hashtabelle wird ein Konzept angegeben. Als Rückgabe erhält man zu diesem Konzept eine Liste aller Konzepte, die zu diesem Konzept in Verbindung stehen, d.h. für das Konzept K gehört das Konzept K' genau dann zu der Rückgabeliste, wenn eine nichttriviale p-Konditionierung (d.h. eine p-Konditionierung, die nicht gleich dem Intervall $[0,1]$ ist) von K nach K' oder von K' nach K existiert.

Wenn eine p-Konditionierung $K \rightarrow K'$ nichttrivial wird, z.B. durch eine Eingabe des Benutzers oder während der Betrachtung eines triangulären Falles, so wird K in die Liste der mit K' verbundenen Konzepte und K' in die Liste der mit K verbundenen Konzepte aufgenommen, falls sie nicht schon in der jeweiligen Liste enthalten sind.

Mithilfe dieser Hashtabelle wird während des Propagierungsalgorithmus die Anzahl der betrachteten triangulären Fälle auf die Fälle beschränkt, die neue Informationen liefern können.

Hashtabelle *p-Konditionierungen*

Schlüssel: Konzeptpaar

Rückgabewert: p-Konditionierungen

Hier werden alle nichttrivialen p-Konditionierungen gespeichert. Als Schlüssel wird eine Liste $(K \ K')$ benutzt, wobei K alphanumerisch kleiner als K' ist, um zu vermeiden, daß Kanten doppelt gespeichert oder nicht gefunden werden, d.h. die Einträge der Hashtabelle sind alphanumerisch geordnet.

Der zu einem Konzeptpaar $(K \ K')$ gehörende Eintrag ist eine Liste mit vier Werten $(p \ q \ p' \ q')$, wenn die p-Konditionierungen zwischen K und K' das folgende Format haben:

$$K \begin{array}{c} \xrightarrow{p,q} \\ \xleftarrow{p',q'} \end{array} K' \wedge K' \begin{array}{c} \xrightarrow{p',q'} \\ \xleftarrow{p,q} \end{array} K.$$

Wenn sowohl die p-Konditionierung von K nach K' als auch die p-Konditionierung von K' nach K trivial ist, so erfolgt kein Eintrag in die Hashtabelle. Sobald eine p-Konditionierung nichttrivial wird, erfolgt ein Eintrag. Wenn bei einem Zugriff auf die Hashtabelle der Wert NIL zurückgegeben wird, so sind die zu diesem Konzeptpaar gehörenden p-Konditionierungen trivial, d.h. gleich dem Einheitsintervall $[0,1]$.

Der Grund, daß zu zwei Konzepten beide p-Konditionierungen gespeichert werden, liegt darin, daß bei der Abarbeitung eines triangulären Falles beide p-Konditionierungen benötigt werden, so daß ein Zugriff auf die Hashtabelle gegenüber der Variante, in der zu einem Konzeptpaar nur eine p-Konditionierung gespeichert würde, gespart wird.

Hashtabelle *logische-Abhängigkeiten*

Schlüssel: Liste mit drei Konzepten

Rückgabewert: Art der logischen Abhängigkeit

In dieser Hashtabelle werden die logischen Abhängigkeiten innerhalb triangulärer Fälle gespeichert. Als Schlüssel wird eine Liste der drei Konzepte in alphanumerischer Reihenfolge übergeben. Als Eintrag steht dazu eine Beschreibung der logischen Abhängigkeit zwischen diesen drei Konzepten in der Hashtabelle.

Diese Beschreibung der logischen Abhängigkeit besteht aus einer Liste mit zwei Werten:

(<logic> <marker>).

<logic> ist dabei eine natürliche Zahl zwischen eins und vier, die die Art der logischen Abhängigkeit zwischen den drei Konzepten denotiert. Dabei bestehen folgende Beziehungen zwischen den Werten von <logic> und der Art der logischen Abhängigkeit:

Wert Art der Abhängigkeit

- 1 Zwei der drei Konzepte sind zueinander komplementär. Markiert wird dabei das dritte, „unbeteiligte“ Konzept.
- 2 Eines der drei Konzepte ist die Konjunktion der beiden anderen Konzepte. Dieses Konzept ist markiert.
- 3 Eines der drei Konzepte ist die Disjunktion der beiden anderen Konzepte. Dieses Konzept ist markiert.
- 4 Dieser Fall wurde bereits überprüft und es besteht keine logische Abhängigkeit. Dieser Fall ist also uninteressant.

<marker> gibt an, an wievielter Stelle in der Liste der drei Konzepte das markierte Konzept sich befindet, d.h. <marker> ist eine natürliche Zahl zwischen

eins und drei. Wenn $\langle \text{logic} \rangle$ den Wert vier hat, so ist der Wert von $\langle \text{marker} \rangle$ nicht interessant und kann somit einen beliebigen Wert annehmen.

Hashtabelle *history*

Schlüssel: Konzeptpaar

Rückgabewert: „History“ der p-Konditionierung

Diese Hashtabelle enthält nach dem Ablauf der Propagierung zu jeder p-Konditionierung, die vorgegeben oder verändert wurde, eine kurze „History“, die beschreibt, wie sich der aktuelle Wert der p-Konditionierung entwickelt hat. Dazu wird in dieser Hashtabelle gespeichert, ob die p-Konditionierung vom Benutzer eingegeben wurde und in welchen triangulären Fällen (mit Angabe der drei Konzepte, die diesen triangulären Fall bilden) mit Hilfe welcher Constraints sich die p-Konditionierung geändert hat.

Beispiel 6.1 *Betrachten wir nochmals den Nixon-Diamant aus dem vierten Kapitel. Nach Eingabe des dort beschriebenen Wissens und Propagierung der Constraints durch die so gewonnene Wissensbasis besteht die „History“ für das Konzeptpaar (R&Q Pazifist) aus den folgenden Einträgen:*

- *Der Ausgangswert ist (0 1 0 1), da der Benutzer keine Spezifikationen über die Beziehungen zwischen R&Q und Pazifisten gemacht hat.*
- *Im triangulären Fall (R&Q Pazifist Quäker) wurde durch das Constraint CONS-TRIAN der Wert auf (0 0.758 0 1) geändert.*
- *Im triangulären Fall (R&Q Pazifist Republikaner) wurde der Wert durch die Constraints CONS-TRIAN und CONS-BAYES auf (0.75 0.758 0 0.505) verfeinert.*

Man sieht also, wie sich die p-Konditionierungen zwischen R&Q und Pazifist verändert haben. Der Wert (0.75 0.758) ergibt sich, da in der PBox der Wert „ungefähr 1/3“ durch das Intervall [0.33 0.34] modelliert wurde.

6.3 Listen

Eine Liste ist eine verkettete Folge von Elementen eines gegebenen Datentyps. Mengen können kanonisch durch Listen beschrieben werden, in dem jedes Element der Menge zu einem Element der Liste wird. Der Propagierungsalgorithmus benötigt drei Mengen, die wir durch Listen repräsentiert haben.

agenda

Diese Liste ist die Grundlage des Propagierungsalgorithmus. Mit ihrer Hilfe werden die triangulären Fälle gebildet. Die ***agenda*** besteht aus Paaren von Konzepten, d.h. sie ist eine Liste von zweielementigen Listen. In der Liste befinden sich nur Konzeptpaare, zwischen denen eine interessante p-Konditionierung besteht, also eine p-Konditionierung, die ungleich den Intervallen $[0,1]$ und $[1,1]$ ist. Zu Beginn der Propagierung wird die Liste mit allen Konzeptpaaren initialisiert, zwischen denen eine interessante p-Konditionierung besteht. Aus diesen Konzeptpaaren werden dann während des Ablaufs des Propagierungsalgorithmus die triangulären Fälle gebildet, die neues Wissen inferieren können und somit betrachtet werden müssen.

Die Liste ***agenda*** ist als Schlange (queue) realisiert, d.h. ein Element wird immer vom Anfang der Liste genommen, während die Erweiterung der Liste durch Anhängen eines neuen Elementes an das Ende der Liste geschieht. Die Konzeptpaare werden dann während der Propagierung nacheinander abgearbeitet.

Wenn sich eine interessante p-Konditionierung während der Propagierung ändert bzw. wenn eine p-Konditionierung interessant wird, so wird das Konzeptpaar, zwischen dem diese p-Konditionierung besteht, an das Ende der Liste ***agenda*** angefügt. Dies hat den Vorteil, daß bei Betrachtung dieses Konzeptpaares alle Konzeptpaare, die vorher in der Liste stehen, bereits betrachtet wurden und das daraus gewonnene Wissen kann bei Betrachtung dieses Konzeptpaares einfließen. Würde es dagegegen an den Anfang der Liste angefügt, so besteht die Möglichkeit, daß das Konzeptpaar nach Abarbeitung der restlichen Elemente der ***agenda*** nochmal betrachtet werden müßte, je nach Art der gewonnenen Informationen. Durch das Anhängen an das Ende der Liste wird diese doppelte Betrachtung vermieden.

Dies ist der Grund für die Realisierung der Liste ***agenda*** als „Queue“ und nicht als „Stack“.

konzepte

In dieser Liste werden die Konzeptnamen gespeichert. Bei der Übernahme des terminologischen Wissens aus einer TBox werden die einzelnen Konzeptnamen in diese Liste aufgenommen.

Die Liste ***konzepte*** wird dazu benutzt, zu jedem Konzept die subsumierten Konzepte zu finden. Desweiteren wird geprüft, ob ein Konzept leer ist oder zu einem anderen Konzept disjunkt ist.

Bei Eingabe einer p-Konditionierung bzw. Einlesen einer PBox wird geprüft, ob die verwendeten Konzeptnamen in der Liste ***konzepte*** stehen. Falls dies nicht

der Fall sein sollte, so wird eine Warnung ausgegeben, da diese Konzepte nicht in der Terminologie vorkommen. Damit können Fehler bei der Eingabe abgefangen werden.

leere-konzepte

Diese Liste enthält alle leeren Konzepte. Sobald ein Konzept als leer erkannt wird, wird es in diese Liste aufgenommen.

6.4 Globale Variablen

In diesem letzten Abschnitt des Kapitels Datenstrukturen werden einige globale Variablen vorgestellt, die verschiedene Optionen steuern. Es handelt sich dabei um Optionen, die die Art der Ein- und Ausgabe spezifizieren. Bei der Eingabe von p-Konditionierungen sind sowohl Dezimalzahlen als auch Brüche erlaubt. Intern werden p-Konditionierungen durch Brüche repräsentiert, so daß eine Konvertierung bei Ein- und Ausgabe nötig sein kann.

nachkommastellen

Diese Variable gibt die Anzahl der Nachkommastellen an, die bei der Konvertierung der Eingabe berücksichtigt werden sollen. Als Default ist neun eingestellt, was für die meisten Domänen ausreichend sein dürfte. Bei Bedarf kann dieser Wert geändert werden.

nenneraus

Diese Variable beschreibt, wie die Ausgabe konvertiert werden soll. Durch *nenneraus* wird die Ausgabe defaultmäßig auf drei Nachkommastellen begrenzt.

device

Während der Propagierung kann ein „Trace“ angelegt werden, der angibt, welche triangulären Fälle in welcher Reihenfolge betrachtet werden, welche Constraints angewendet werden und wie die Werte sich verändern. Normalerweise ist diese Möglichkeit ausgeschaltet, da der „Trace“ schon bei kleinen Wissensbasen sehr umfangreich wird. Der Benutzer kann den „Trace“ aber anschalten und angeben, ob der „Trace“ auf den Bildschirm ausgegeben oder in ein File geschrieben wer-

den soll. Die Variable ***device*** spezifiziert das Ausgabemedium für den „Trace“ bzw. ob er ausgeschaltet ist. Die Funktionen (trace-on) und (trace-off) schalten den „Trace“ an bzw. aus. Alternativ kann auch die Option *Trace* im Menü *PBox Tools* der Oberfläche benutzt werden.

tracefile

Enthält den Namen der Datei, in die der Trace umgeleitet werden soll. Die Funktion (trace-in <filename>) kann benutzt werden, um den Namen der Datei zu spezifizieren, in die der „Trace“ geschrieben werden soll und ändert somit auch die Variable ***tracefile***. Wiederum kann auch die Option *Trace* im Menü der Oberfläche benutzt werden.

Kapitel 7

Constraints und ihre Propagierung

Nach der Vorstellung der verwendeten Datenstrukturen und der Beschreibung der Schnittstelle zum KRIS/RAT-System, die in den beiden letzten Kapiteln erfolgt ist, wird nun auf die Implementierung der verwendeten Constraints und den Propagierungsalgorithmus eingegangen. Mit dem Propagierungsalgorithmus werden die probabilistischen Constraints durch die Wissensbasis propagiert, um neues Wissen zu gewinnen.

Hauptbestandteil des Propagierungsalgorithmus sind die in Kapitel 3 beschriebenen Constraints CONS-TRIAN und CONS-BAYES, die zugleich die zentralen Sätze in [Heinsohn 94a] sind.

Diese beiden Constraints wurden von uns um ein drittes Constraint CONS-LOG erweitert, dessen Aufgabe die Modellierung der logischen Abhängigkeiten zwischen Konzepten ist. In dieses Constraint sind die in Kapitel 3 beschriebenen Constraints CONS-NOT, CONS-AND und CONS-OR eingebaut.

Das Constraint CONS-PINGUIN fängt einen Sonderfall ab, der dadurch entsteht, daß die beiden Constraints CONS-TRIAN und CONS-BAYES nur auf nichtleere Konzepte angewendet werden dürfen.

Im Propagierungsalgorithmus werden trianguläre Fälle betrachtet. Auf diese triangulären Fälle werden dann die einzelnen Constraints angewendet (siehe Abb. 4.2).

Bei der Implementierung des Propagierungsalgorithmus ergaben sich folgende Probleme:

- Auswahl der triangulären Fälle
- Reihenfolge, in der die Constraints angewendet werden
- Abarbeitung eines triangulären Falles
- Auswirkungen auf das System durch vom Benutzer modifizierte p-Konditionierungen

Die folgenden Abschnitte befassen sich mit der Lösung dieser Probleme.

7.1 Auswahl der triangulären Fälle

Da die Anwendung der Constraints auf einen triangulären Fall sehr rechenintensiv ist, sollten nur die triangulären Fälle betrachtet werden, die neue Informationen für das System liefern können.

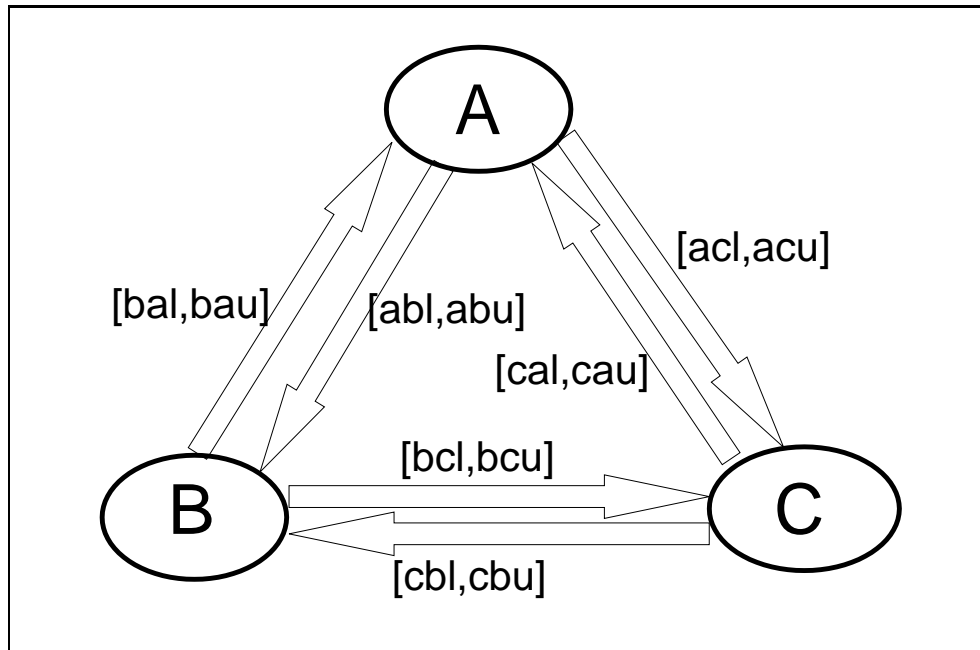


Abbildung 7.1: Triangulärer Fall

Dazu müssen sie folgenden Bedingungen genügen:

1. Es muß mindestens zwei nichttriviale p-Konditionierungen geben. Als nicht-triviale p-Konditionierungen bezeichnet man hierbei alle Intervalle außer $[0,1]$.
2. Davon muß mindestens eine p-Konditionierung interessant sein, d.h. sie darf weder trivial noch eine Subsumtionskante sein.

Einen solchen triangulären Fall bezeichnet man als „interessant“.

Es folgen nun einige Anmerkungen zur Auswahl der triangulären Fälle:

- Die p-Konditionierung $[0,1]$ drückt die völlige Unwissenheit aus und kann deshalb keine neuen Informationen für das System liefern.
- Die p-Konditionierung $[1,1]$ stellt wie in Kapitel 4 beschrieben die Subsumtionskante von einem Konzept K nach einem Konzept K' dar.

- Nullkanten $[0,0]$ werden, wie in Kapitel 5 beschrieben, gesondert behandelt.

Wie das System die „interessanten“ triangulären Fälle findet, wird im folgenden Abschnitt beschrieben.

7.2 Der Propagierungsalgorithmus

Der Propagierungsalgorithmus beschreibt, wie aus vorgegebenem Wissen neue Informationen abgeleitet werden können.

```

solange (Liste *agenda* nicht leer) do
  nimm erstes Listenelement (<konzept1> <konzept2>)
  und streiche es aus der Liste;
  suche mögliche 3. Konzepte;
  speichere sie in Liste „3. Konzepte“;
  solange (Liste „3. Konzepte“ nicht leer) do
    nimm erstes Listenelement <konzept3>
    und streiche es aus der Liste;
    bilde triangulären Fall:
    (<konzept1> <konzept2> <konzept3>);
    nimm 6 p-Konditionierungen;
    bearbeite triangulären Fall;
    speichere neue Werte;
    erweitere ggf. die Liste *agenda*;
  od;
od;

```

Der Propagierungsalgorithmus besteht im wesentlichen aus zwei ineinander verschachtelten Schleifen:

- **solange** (*Liste ***agenda*** nicht leer*) **do**
 In der äußeren Schleife wird die Liste ***agenda*** abgearbeitet, die wie in Kapitel 6 beschrieben mit allen interessanten Kanten initialisiert wird, wobei eine Kante durch eine Liste von zwei Konzepten repräsentiert wird. Z.B. wird die Kante $K \rightarrow K'$ durch die Liste $(K \ K')$ repräsentiert. Wenn die Liste ***agenda*** leer ist, wird die äußere Schleife abgebrochen.

1. *nimm erstes Listenelement (<konzept1> <konzept2>) und streiche es aus der Liste;*

Die Liste ***agenda*** hat als Elemente nur interessante Kanten, d.h. an dieser Stelle wird die 2. Bedingung für einen „interessanten“ triangulären Fall erfüllt. Hier wird das oberste Listenelement fokussiert und nach beendeter Betrachtung aus der Liste gestrichen.

2. *suche mögliche 3. Konzepte;*

Es werden nun alle möglichen dritten Konzepte gesucht, um einen interessanten triangulären Fall mit dem fokussierten Element aus der Liste ***agenda*** bilden zu können. Dazu greift man zweimal auf die in Kapitel 6 beschriebene Hashtabelle ***verbundene Konzepte*** zu und zwar einmal mit dem Schlüssel $\langle \text{konzept1} \rangle$ und einmal mit dem Schlüssel $\langle \text{konzept2} \rangle$. Man erhält eine Liste von Konzepten, die jeweils über eine nichttriviale p-Konditionierung mit dem Schlüsselkonzept $\langle \text{konzept1} \rangle$ verbunden sind und eine Liste von Konzepten, die jeweils mit dem Schlüsselkonzept $\langle \text{konzept2} \rangle$ verbunden sind.

3. *speichere sie in Liste „3. Konzepte“;*

Die beiden eben beschriebenen Listen werden zur Liste „3. Konzepte“ zusammengefügt, wobei folgendes zu beachten ist:

- Wenn ein Konzept sowohl mit $\langle \text{konzept1} \rangle$ als auch mit $\langle \text{konzept2} \rangle$ durch eine nichttriviale p-Konditionierung verbunden ist, darf es in die Liste „3. Konzepte“ nicht doppelt aufgenommen werden.
- Die beiden Konzepte $\langle \text{konzept1} \rangle$ und $\langle \text{konzept2} \rangle$ dürfen selbst nicht Elemente der Liste „3. Konzepte“ sein, da sonst kein triangulärer Fall gebildet werden kann.

• **solange** (*Liste „3. Konzepte“ nicht leer*) **do**

In der inneren Schleife wird die Liste „3. Konzepte“ abgearbeitet, die wie oben beschrieben initialisiert wurde.

1. *nimm erstes Listenelement $\langle \text{konzept3} \rangle$ und streiche es aus der Liste;*

Das erste Listenelement der Liste „3. Konzepte“ wird nun fokussiert und nach beendeter Betrachtung aus der Liste gestrichen.

2. *bilde triangulären Fall:*

$(\langle \text{konzept1} \rangle \langle \text{konzept2} \rangle \langle \text{konzept3} \rangle)$;

Die fokussierten Konzepte $\langle \text{konzept1} \rangle$, $\langle \text{konzept2} \rangle$ und $\langle \text{konzept3} \rangle$ bilden nun einen triangulären Fall. Die an dieser Stelle betrachteten triangulären Fälle erfüllen die im letzten Abschnitt aufgestellten Bedingungen für einen interessanten triangulären Fall. Auf diese Art und Weise findet also das System alle interessanten triangulären Fälle.

3. *nimm 6 p-Konditionierungen;*

Das System lädt nun die sechs p-Konditionierungen des fokussierten triangulären Falles ein, indem es auf die Hashtabelle ***p-Konditionierungen*** zugreift.

4. *bearbeite triangulären Fall;*

Auf den betrachteten triangulären Fall werden nun die einzelnen Constraints angewendet. Auf die Constraints und die Abarbeitung eines triangulären Falles wird in den folgenden Abschnitten eingegangen.

Zu beachten ist hierbei, daß dadurch nicht nur neue Werte berechnet werden können, sondern auch Inkonsistenzen festgestellt werden, falls die Schnittmenge von einem gegebenen und dem dazugehörigen berechneten Intervall leer ist.

5. *speichere neue Werte;*
Falls eine p-Konditionierung verfeinert wurde, wird sie in der Hashtabelle ***p-Konditionierungen*** gespeichert.
6. *erweitere ggf. die Liste *agenda*;*
Die verfeinerten Kanten, die jeweils durch eine Liste von zwei Konzepten repräsentiert werden, werden an die Liste ***agenda*** angehängen.

Wenn die Liste ***agenda*** vollständig abgearbeitet wurde, ist der Propagierungsalgorithmus beendet. Es sind nun alle möglichen Inferenzen aus dem bereitgestellten Wissen gezogen. Eine weitere Anwendung des Propagierungsalgorithmus würde keine neuen Informationen für das System bringen.

7.3 Das Constraint CONS-LOG

Das Constraint CONS-LOG dient der Untersuchung von logischen Abhängigkeiten zwischen Konzepten. Es umfaßt die Constraints zur Berechnung von triangulären Fällen mit logischem Bezug, die im folgenden CONS-NOT, CONS-AND und CONS-OR genannt werden.

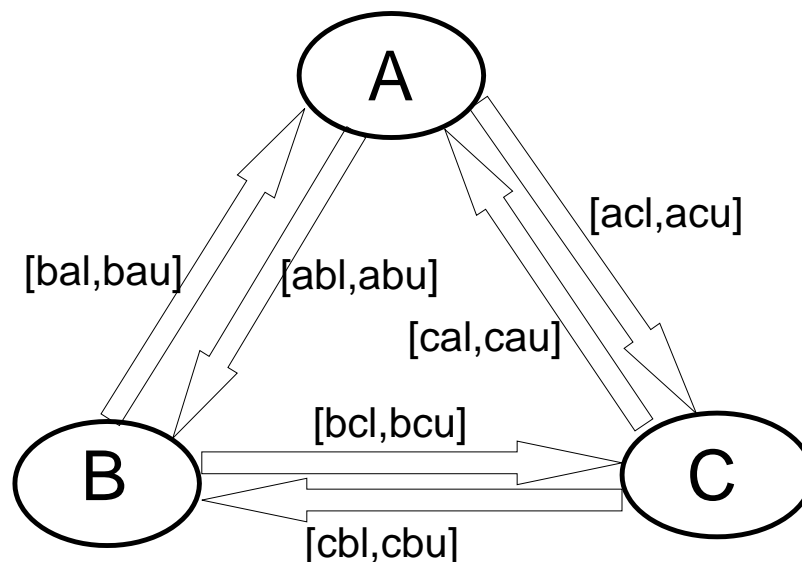


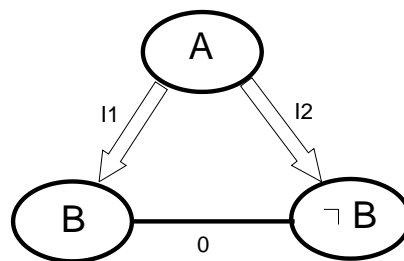
Abbildung 7.2: Der allgemeine trianguläre Fall

Damit diese Constraints, die im weiteren noch genauer beschrieben werden, auf einen gegebenen triangulären Fall (s. Abb. 7.2) angewandt werden können, müssen folgende zehn Fragen beantwortet werden:

1. Sind A und $\neg B$ äquivalent?
2. Sind B und $\neg C$ äquivalent?
3. Sind C und $\neg A$ äquivalent?
4. Sind A und $(B \sqcap C)$ äquivalent?
5. Sind B und $(A \sqcap C)$ äquivalent?
6. Sind C und $(A \sqcap B)$ äquivalent?
7. Sind A und $(B \sqcup C)$ äquivalent?
8. Sind B und $(A \sqcup C)$ äquivalent?
9. Sind C und $(A \sqcup B)$ äquivalent?
10. Existiert keiner der oben genannten logischen Bezüge?

Wenn man alle diese Fragen durch Anfragen an das KRIS/RAT-System beantworten lassen würde, so wäre CONS-LOG sehr rechenzeitintensiv und somit ineffizient. Auf diesen Punkt wird später noch eingegangen. Zunächst werden die Constraints CONS-NOT, CONS-AND und CONS-OR vorgestellt.

- Das Constraint CONS-NOT

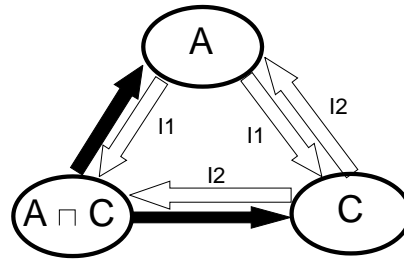


$$C = \neg B$$

$$I1 = [\max(\text{abl}, 1 - \text{acu}) ; \min(\text{abu}, 1 - \text{acl})]$$

$$I2 = [\max(\text{acl}, 1 - \text{abu}) ; \min(\text{acu}, 1 - \text{abl})]$$

- Das Constraint CONS-AND

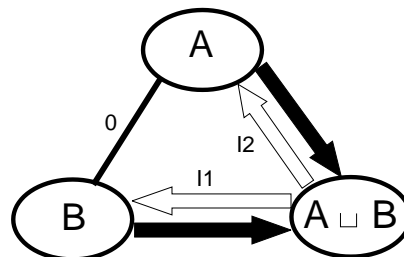


$$B = A \cap C$$

$$I1 = [\max(ab_l, ac_l) ; \min(ab_u, ac_u)]$$

$$I2 = [\max(ca_l, cb_l) ; \min(ca_u, cb_u)]$$

- Das Constraint CONS-OR



$$C = A \cup B$$

$$I1 = [\max(cb_l, 1-ca_u) ; \min(cb_u, 1-ca_l)]$$

$$I2 = [\max(ca_l, 1-cb_u) ; \min(ca_u, 1-cb_l)]$$

Diese Constraints wurden alle analog zu den Formeln (Satz 3.9, Satz 3.10) gebildet, nur mit dem Unterschied, daß Schnittmengenbildung vorgenommen wurde, wenn zu einer Kante zwei Intervalle existieren (eine gegebene und eine berechnete p-Konditionierung).

Um Laufzeit einzusparen, wird CONS-LOG nicht auf jeden triangulären Fall angewendet, sondern nur in den Fällen, in denen die notwendige Bedingung für logische Abhängigkeit erfüllt ist: Ein triangulärer Fall muß mindestens eine Nullkante oder genau zwei Subsumtionskanten enthalten, um einen logischen Bezug

der einzelnen Konzepte zueinander enthalten zu können. Diese notwendige Bedingung läßt sich leicht anhand der p-Konditionierungen ohne Anfrage an das KRIS/RAT-System überprüfen und erspart in den meisten Fällen den Aufruf von CONS-LOG.

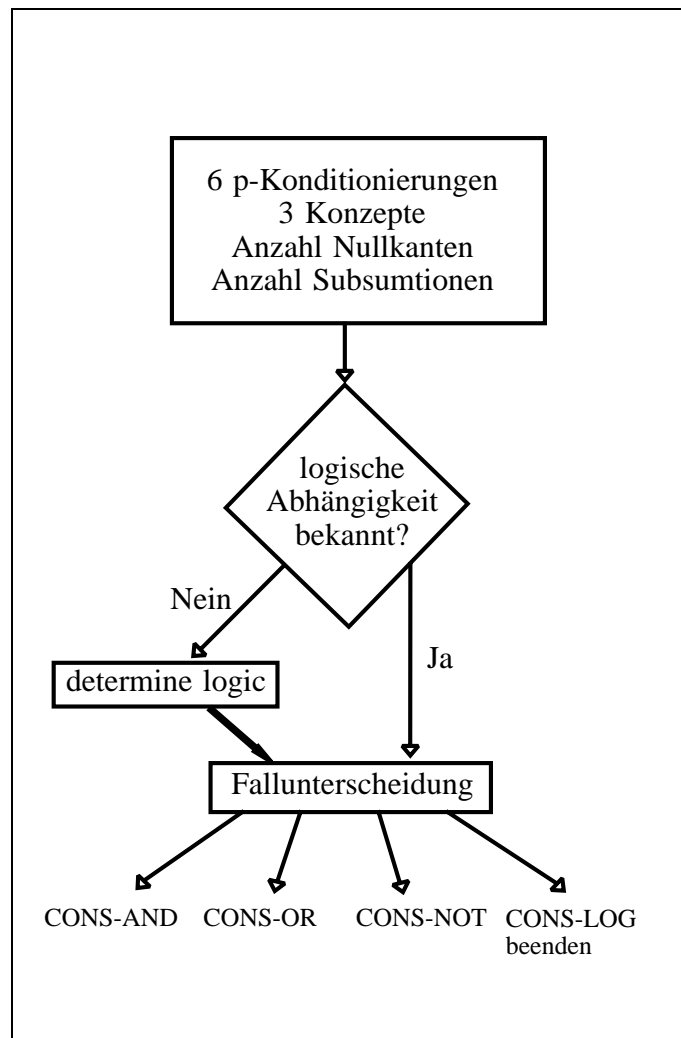


Abbildung 7.3: Das Constraint CONS-LOG

Das Constraint CONS-LOG (s. Abb. 7.3) greift auf die Hashtabelle *logische-Abhängigkeiten* zu. Wurde der aktuelle trianguläre Fall bereits überprüft, so ist hier die Art der logischen Abhängigkeit vermerkt und das entsprechende Constraint kann ausgeführt bzw. CONS-LOG verlassen werden. Ist keine logische Abhängigkeit bekannt, so wird die Funktion determine-logic aufgerufen, die die Art der logischen Abhängigkeit bestimmt und in der Hashtabelle speichert. Anschließend kann das entsprechende Constraint ausgeführt bzw. CONS-LOG beendet werden, falls keine logischen Bezüge in dem triangulären Fall bestehen. Die Speicherung in der Hashtabelle geschieht nach dem im vorigen Kapitel

erläuterten Verfahren.

Die Funktion *determine-logic* (s. Abb. 7.4) ist ein wesentlicher Bestandteil von CONS-LOG. Als Parameter werden hier die Namen der drei Konzepte des triangulären Falles, sechs p-Konditionierungen, sowie die Anzahl von Nullkanten und Subsumtionskanten übergeben. Ausgehend von *determine-logic* werden weitere Funktionen aufgerufen, auf die hier nun näher eingegangen werden soll.

- *determine-logic*
Hier wird anhand der Anzahl der Subsumtionskanten und der Nullkanten eine erste Fallunterscheidung vorgenommen. Gibt es zwei oder mehr Subsumtionen, so besteht in diesem Fall eventuell ein logischer Bezug durch Disjunktion oder Konjunktion. Die Funktion „andorgate“ wird aufgerufen. Gibt es weniger als zwei Subsumtionen, aber eine Nullkante, so wird „cons-not-call“ aufgerufen. Alle weiteren Fälle sind uninteressant.
- *andorgate*
Bekannt ist an dieser Stelle schon, daß zwei Subsumtionen vorhanden sind. Anhand der Anordnung dieser Subsumtionen wird bestimmt, ob der trianguläre Fall eine Disjunktion oder eine Konjunktion enthalten könnte. Führen beide Subsumtionskanten vom selben Konzept weg, wird „andcheck“ aufgerufen, führen sie auf dasselbe Konzept zu, und befindet sich eine Nullkante zwischen den beiden Konzepten, von denen die Subsumtionen ausgehen, so wird „orcheck“ aufgerufen. Ansonsten wird der trianguläre Fall als uninteressant für CONS-LOG markiert.
- *cons-not-call*
Anhand der Position der Nullkante, wird die Reihenfolge (d.h. die richtige Permutation der Parameter) für „notcheck“ bestimmt, und die Funktion wird aufgerufen.
- *notcheck*
„notcheck“ stellt eine Anfrage an das KRIS/RAT-System, ob die beiden Konzepte, zwischen denen sich die Nullkanten befinden, eine disjunkte Partition von *TOP* darstellen. Wenn ja, wird zu dem betrachteten Fall in der Hashtabelle ***logische-Abhängigkeiten*** vermerkt, daß CONS-NOT angewendet werden kann, ansonsten, daß er uninteressant ist.
- *andcheck*
Analog zur vorigen Funktion fragt „andcheck“ im KRIS/RAT-System an, ob das Konzept, von dem die Subsumtionskanten wegführen, der Konjunktion der beiden anderen Konzepte entspricht. Wenn ja, wird der Fall als interessant für CONS-AND, ansonsten als uninteressant markiert.

- *orcheck*
 „orcheck“ fragt im KRIS/RAT-System an, ob das Konzept, von dem die beiden Subsumtionskanten wegführen, äquivalent zur Disjunktion dieser beiden Konzepte ist. Wenn ja, ist der trianguläre Fall interessant für CONS-LOG und wird auch entsprechend in der Hashtabelle ***logische-Abhängigkeiten*** markiert. Wenn nicht, kann dieser Fall noch für CONS-NOT in Frage kommen. Es wird also „notcheck“ aufgerufen.

7.4 Das Constraint CONS-PINGUIN

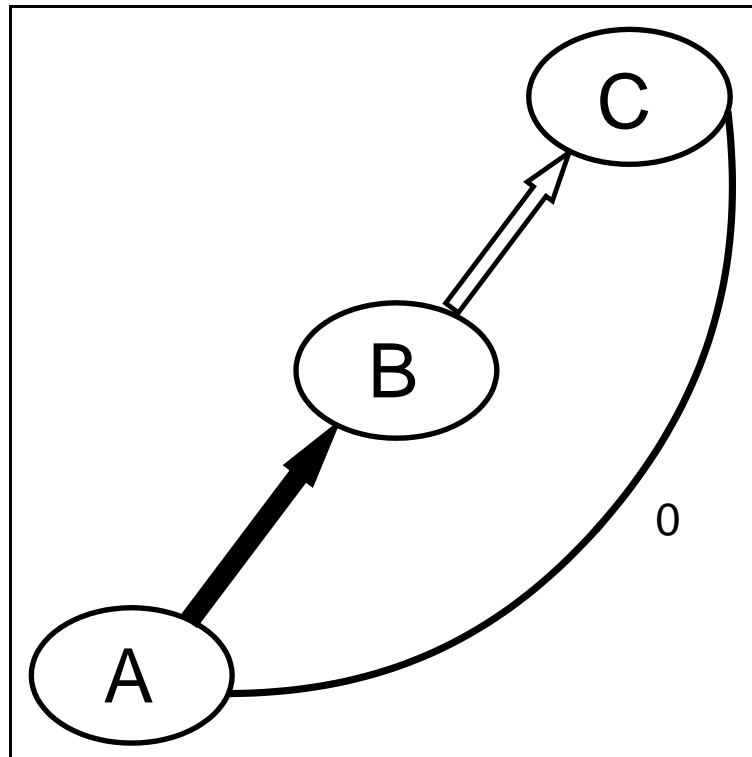


Abbildung 7.5: CONS-PINGUIN

Die Betrachtung des folgenden triangulären Falles verdeutlicht die Aufgabe des Constraints CONS-PINGUIN (siehe Abb. 7.5):

- Es gibt eine Subsumtionskante von einem Konzept *A* zu einem Konzept *B*
- Die Konzepte *A* und *C* sind disjunkt zueinander
- Die *p*-Konditionierung von Konzept *B* nach Konzept *C* ist ungleich $[1,1]$

Modifiziert nun der Benutzer die *p*-Konditionierung von Konzept *B* nach Konzept *C* zu einer Subsumtionskante oder ergibt sich dies aus der Propagierung,

so würde eine Anwendung der beiden Constraints CONS-TRIAN und CONS-BAYES auf diesen triangulären Fall eine Inkonsistenzmeldung zur Folge haben, da diese beiden Constraints nur bei Anwendung auf nichtleere Konzepte korrekt arbeiten.

Das System sollte aber erkennen, daß in diesem Fall die Extension von Konzept *A* leer sein muß. Genau diesen Sonderfall fängt das Constraint CONS-PINGUIN ab.¹

7.5 Die Abarbeitung eines triangulären Falles

Nach der Vorstellung der beiden Constraints CONS-LOG und CONS-PINGUIN in den beiden letzten Abschnitten soll nun die Abarbeitung eines triangulären Falles erklärt werden. Die beiden Constraints CONS-TRIAN und CONS-BAYES wurden bereits in Kapitel 3 ausführlich erklärt. Eine detaillierte Herleitung der beiden Constraints findet man in [Heinsohn 94a]. Wie schon in der Einleitung zu diesem Kapitel erwähnt, werden die einzelnen Constraints auf den betrachteten triangulären Fall angewendet. Es folgt nun die Abarbeitungsreihenfolge eines triangulären Falles:

1. Wende CONS-PINGUIN an
2. Wende CONS-LOG an
3. Wende CONS-TRIAN an
4. Wende CONS-BAYES an
5. Wiederhole die Punkte 2-4 solange, bis die p-Konditionierungen stabil sind

Es folgen nun einige Erläuterungen zur Abarbeitung eines triangulären Falles:

- *Wende CONS-PINGUIN an*
Es wird überprüft, ob der im letzten Abschnitt dargestellte Spezialfall in dem fokussierten triangulären Fall vorliegt. Wenn dies der Fall ist, wird das Constraint CONS-PINGUIN angewendet.
- *Wende CONS-LOG an*
Das Constraint CONS-LOG überprüft wie in Abschnitt 7.3 gezeigt die logischen Abhängigkeiten zwischen Konzepten in dem betrachteten triangulären Fall. CONS-LOG wird nur angewandt, wenn die notwendige Bedingung für logische Abhängigkeiten erfüllt ist: Ein triangulärer Fall muß mindestens eine Nullkante oder genau zwei Subsumtionskanten enthalten.

¹Dieser Sonderfall begegnete uns zum ersten Mal im Beispiel „Birds“ (siehe Abb.8.3) und veranlaßte uns zur Namensgebung „CONS-PINGUIN“.

- *Wende CONS-TRIAN an*
Das Constraint CONS-TRIAN berechnet eine p-Konditionierung aus vier gegebenen Intervallen. Da es in einem triangulären Fall sechs p-Konditionierungen gibt, muß CONS-TRIAN also sechsmal pro Durchlauf angewendet werden.
- *Wende CONS-BAYES an*
Das Constraint CONS-BAYES berechnet eine p-Konditionierung aus sechs gegebenen Intervallen. Analog zu CONS-TRIAN muß CONS-BAYES daher sechsmal pro Durchlauf angewendet werden.
- *Wiederhole die Punkte 2-4 solange, bis die p-Konditionierungen stabil sind*
Die Punkte 2-4 werden solange wiederholt, bis die p-Konditionierungen in dem fokussierten triangulären Fall stabil sind, d.h. bis eine komplette Schleife ohne Veränderung von Werten durchlaufen wurde. Um Laufzeit zu sparen, ist eine „komplette Schleife“ hier nicht definiert als „von Punkt 2 bis Punkt 4“, sondern als ein vollständiger Durchlauf dieser drei Punkte (also auch 3-4-2 oder 4-2-3). Nach spätestens zwei Durchläufen sind alle p-Konditionierungen stabil.

Wenn die p-Konditionierungen in dem betrachteten triangulären Fall stabil sind, würde eine weitere Anwendung der Constraints keine neuen Informationen für das System mehr liefern.

Durch die mehrfache Anwendung der Constraints ist der Rechenaufwand bei der Bearbeitung eines triangulären Falles wie schon in Kapitel 4 erwähnt sehr groß. Daher ist es wichtig, die Anzahl der zu betrachtenden triangulären Fälle stark einzuschränken (siehe Abschnitt 7.1). Im nächsten Abschnitt wird dieser Aspekt weiter vertieft.

7.6 Modifizierung von p-Konditionierungen

Für den Benutzer besteht die Möglichkeit, nach Einlesen von T- und PBox und erfolgter Propagierung beliebig viele p-Konditionierungen mit dem SLIDER oder der Tastatur zu verfeinern.

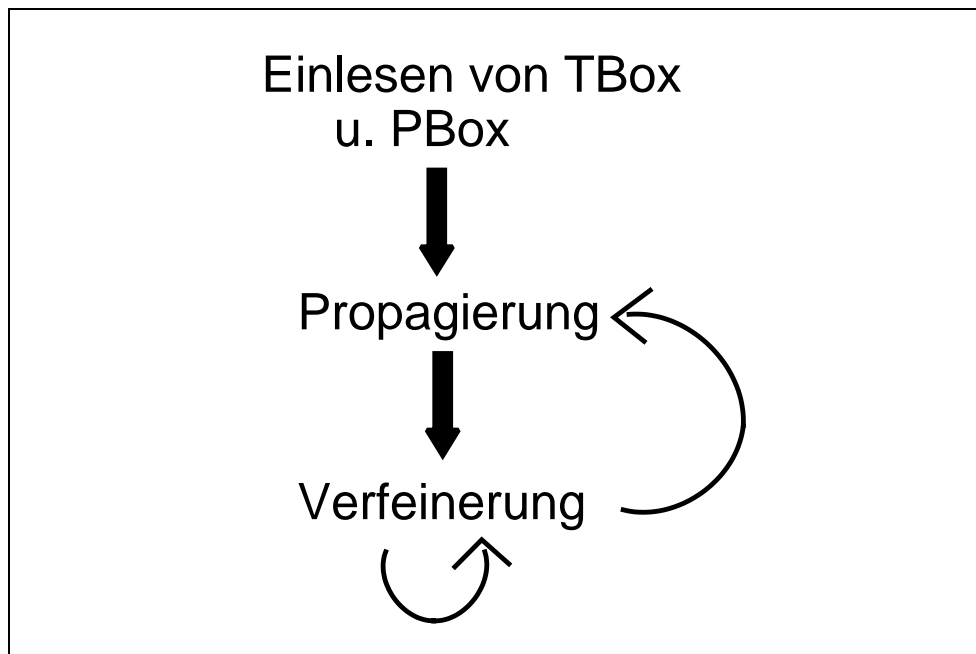


Abbildung 7.6: Modifizierung von p-Konditionierungen

Nachdem der Benutzer verschiedene p-Konditionierungen modifiziert hat, propagiert das System automatisch. Allerdings können nun die zu betrachtenden triangulären Fälle weiter eingeschränkt werden. Die Einschränkung, wie in Abschnitt 7.1 dargestellt, ist jetzt nicht mehr ausreichend, da das System alle p-Konditionierungen außer $[0,1]$ und $[1,1]$ als „interessant“ in die Liste ***agenda*** aufnehmen würde. Neue Informationen für das System können aber nur die p-Konditionierungen liefern, die vom Benutzer über Tastatur oder SLIDER modifiziert wurden. Deshalb wird die Liste ***agenda*** nur mit diesen p-Konditionierungen initialisiert.

Dadurch kann also beim Propagieren nach der Modifizierung von Werten Laufzeit gespart werden und das System arbeitet auch an dieser Stelle effizient.

Im nächsten Kapitel wird der Propagierungsalgorithmus anhand von Beispielen weiter erläutert.

Kapitel 8

Beispiele, Trace

Nachdem in den letzten Kapiteln die Theorie und Umsetzung des \mathcal{ALCP} -Konzeptes dokumentiert wurde, wird nun auf die praktische Anwendung des Systems eingegangen.

Der erste Abschnitt erläutert den Ablauf des Programmes am Beispiel des „Nixon Diamanten“ (aus [Heinsohn 94a]) Schritt für Schritt, um die theoretische Beschreibung des Propagierungsalgorithmus verständlicher zu machen.

Der zweite Abschnitt behandelt verschiedene Beispiele und stellt die von unserem System berechneten Werte vor. Die Vielfalt der Themenbereiche, aus denen die Beispiele stammen, vermittelt einen Eindruck davon, wie universell \mathcal{ALCP} eingesetzt werden kann.

Sowohl die hier vorgestellten als auch viele weitere Beispiele befinden sich im Verzeichnis `/home/endres/ALCP/KB/`, so daß der Leser diese Berechnungen auch selbst nachvollziehen kann.

Einige dieser Beispiele wurden von Heinsohn und von uns entwickelt, andere wiederum haben wir der aktuellen Fachliteratur entnommen.

Hier eine kurze Übersicht:

- **Barking Cats**

Quelle: [Güntzer et al. 91]

Fragestellung: Wie viele Katzen können bellen?

Ergebnis: Die Werte aus der Literatur wurden von \mathcal{ALCP} exakt berechnet.

- **Birds**

Quelle: Standardbeispiel aus der Fachliteratur

Fragestellung: Fast alle Vögel können fliegen - was ist mit den Pinguinen?

Ergebnis: Die erwarteten Werte wurden berechnet.

- **Birds 2**

Quelle: [Heinsohn 94a]

Fragestellung: Analog zu Birds

Ergebnis: An diesem Beispiel kann man die Ausbreitung weniger p-Konditionierungen über ein größeres Netz beobachten. Die PBox ist mit der im Birds-Beispiel identisch. Die erwarteten Werte wurden berechnet.

- **Burglary**

Quelle: [Güntzer et al. 91]

Fragestellung: Wie wahrscheinlich ist es, daß ein Einbrecher im Haus ist, wenn während eines Erdbebens die Alarmanlage ertönt?

Ergebnis: Die vorgegebenen Werte wurden, bedingt durch Rundung in der Vorgabe, von \mathcal{ALCP} etwas exakter berechnet.

- **Children**

Quelle: [Armager et al. 91]

Fragestellung: Auswertung statistischer Daten

Ergebnis: Zwei Intervallgrenzen unterscheiden sich erheblich von der Vorlage. Leider ist nicht dokumentiert, wie die Werte zustande kommen. Auf unsere Anfrage hin erklärte Amarger, daß die Werte des oben genannten Papers nicht mehr dem aktuellen Stand der Forschung entsprechen.

- **Songwriter**

Quelle: konstruiertes Beispiel

Fragestellung: Wie viele Künstler üben mehrere Kunstformen gleichzeitig aus?

Ergebnis: Dieses Beispiel wurde konstruiert, um das Constraint CONSLOG zu testen. Die erwarteten Werte wurden berechnet.

- **Sprinkler**

Quelle: [Güntzer et al. 91]

Fragestellung: Letzte Nacht hat es geregnet, und der Boden ist naß. Ist es wahrscheinlich, daß gleichzeitig die Sprinkleranlage im Garten angeschaltet war?

Ergebnis: Die vorgegebenen Werte wurden berechnet.

8.1 Nixon-Diamant

Der in Kapitel 2 vorgestellte Nixon Diamant ist hier Gegenstand der Betrachtung.

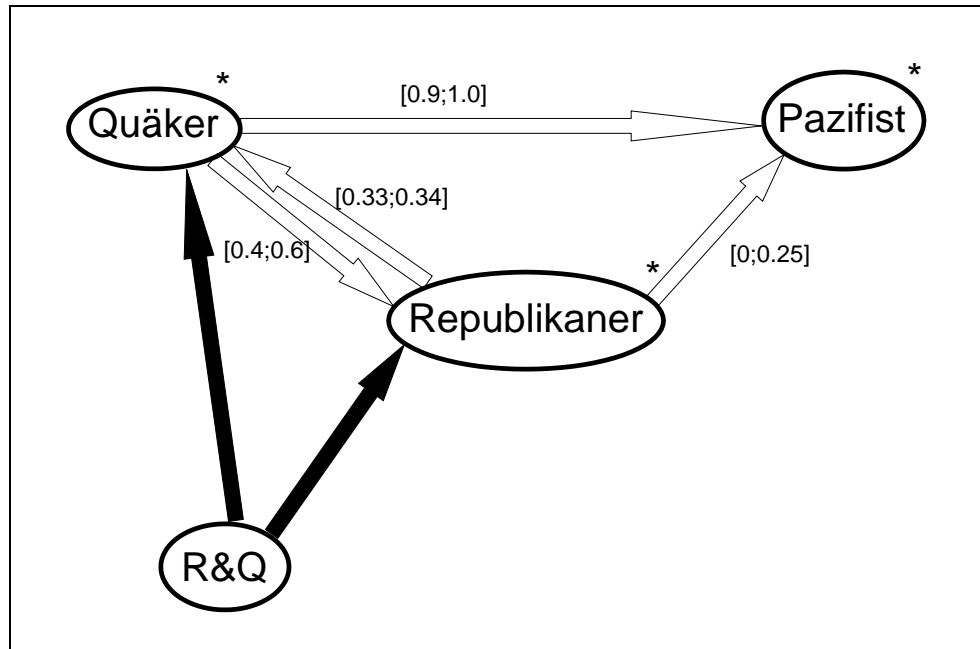


Abbildung 8.1: Der Nixon-Diamant

Die TBox

```

(defprimconcept QUAEKER)
(defprimconcept PAZIFIST)
(defprimconcept REPUBLIKANER)
(defconcept R&Q (and QUAEKER REPUBLIKANER))
  
```

und die PBox

```

(QUAEKER REPUBLIKANER 0.4 0.6)
(REPUBLIKANER QUAEKER 0.33 0.34)
(QUAEKER PAZIFIST 0.9 1)
(REPUBLIKANER PAZIFIST 0 0.25),
  
```

beschreiben das Problem hinreichend.

Der Ablauf des Programms wird zur Laufzeit in ein Tracefile gespeichert. Dieses File wird hier näher untersucht.

Zunächst versucht \mathcal{ALCP} das terminologische Wissen der TBox in p-Konditionierungen umzuwandeln, indem nach Subsumtionen und paarweise disjunkten Konzepten gesucht wird:

```
(SAVING *BOTTOM* PAZIFIST 1.0 1.0 0.0 1.0)
(SAVING QUAEKER R&Q 0.0 1.0 1.0 1.0)
(SAVING *BOTTOM* QUAEKER 1.0 1.0 0.0 1.0)
(SAVING *BOTTOM* R&Q 1.0 1.0 0.0 1.0)
(SAVING *TOP* PAZIFIST 0.0 1.0 1.0 1.0)
(SAVING *TOP* QUAEKER 0.0 1.0 1.0 1.0)
(SAVING *TOP* REPUBLIKANER 0.0 1.0 1.0 1.0)
(SAVING *TOP* R&Q 0.0 1.0 1.0 1.0)
(SAVING *BOTTOM* *TOP* 1.0 1.0 0.0 1.0)
(SAVING R&Q REPUBLIKANER 1.0 1.0 0.0 1.0)
(SAVING *BOTTOM* REPUBLIKANER 1.0 1.0 0.0 1.0)
```

In diesem Fall werden nur Subsumtionen gefunden. Danach wird die PBox geladen:

```
(SAVING QUAEKER REPUBLIKANER 0.4 0.6 0.0 1.0)
(SAVING QUAEKER REPUBLIKANER 0.4 0.6 0.33 0.34)
(SAVING PAZIFIST QUAEKER 0.0 1.0 0.9 1.0)
(SAVING PAZIFIST REPUBLIKANER 0.0 1.0 0.0 0.25)
```

Die Propagierung beginnt mit der Initialisierung der Liste ***agenda***. In diese Liste werden alle interessanten Kanten geschrieben. Dabei erfolgt die Meldung:

```
(QUEUE WAS INITIALIZED AS)
(QUAEKER - > REPUBLIKANER)
(PAZIFIST - > QUAEKER)
(PAZIFIST - > REPUBLIKANER)
```

Nun werden mit der ersten Kante in der Liste ***agenda*** (QUAEKER REPUBLIKANER) und den dazu sinnvollerweise zu betrachtenden dritten Konzepten ***TOP***, R&Q, ***BOTTOM***, PAZIFIST trianguläre Fälle gebildet und bearbeitet:

```
(REGARDED TRIANGULAR CASE - QUAEKER REPUBLIKANER *TOP*)
(CHANGING RANGES IN CONS-TRIAN)
(SAVING *TOP* QUAEKER 0.0 0.563 1.0 1.0)
```

Die Kante *TOP* QUAEKER wird zur Liste ***agenda*** hinzugefügt.

(REGARDED TRIANGULAR CASE - QUAEKER REPUBLIKANER R&Q)
 (CHANGING RANGES IN CONS-AND)
 (SAVING QUAEKER R&Q 0.4 0.6 1.0 1.0)

Durch die Berücksichtigung der logischen Abhängigkeiten werden der Kante QUAEKER R&Q, über die keine Angaben gemacht wurden, exakte Werte zugewiesen. Wenn man davon ausgeht, daß etwa die Hälfte (40 - 60%) der Quäker auch Republikaner sind, kann man folgern, daß 40 - 60% der Quäker gleichzeitig zur Gruppe R&Q (Republikaner und Quäker) gehören.

Das ist die Aussage, die hier von *CONS-AND* (einem der drei CONS-LOG-Constraints) gemacht wird. Die Kante QUAEKER R&Q wird an die Warteschlange ***agenda*** angehängt.

(REGARDED TRIANGULAR CASE - QUAEKER REPUBLIKANER *BOTTOM*)

Hier konnten keine Werte verbessert werden, was allerdings vorher noch nicht zu erkennen war. Dieser trianguläre Fall erfüllte nämlich die Voraussetzung dafür, daß eine Wertverfeinerung möglich ist: Mindestens zwei nichttriviale Kanten (nämlich genau drei!), von denen mindestens eine interessant ist (nämlich die Kante QUAEKER REPUBLIKANER).

(REGARDED TRIANGULAR CASE - QUAEKER REPUBLIKANER PAZIFIST)
 (CHANGING RANGES IN CONS-TRIAN)
 (CHANGING RANGES IN CONS-BAYES)
 (SAVING PAZIFIST QUAEKER 0.0 1.0 0.9 0.903)

Jetzt kommt noch PAZIFIST QUAEKER in die ***agenda***.

Damit sind die triangulären Fälle, die mit der Kante QUAEKER REPUBLIKANER gebildet wurden, abgearbeitet.

Die Kante QUAEKER REPUBLIKANER wird aus der Liste ***agenda*** entfernt, und die nächste Kante PAZIFIST QUAEKER, die nun am Anfang steht, wird betrachtet.

(REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER R&Q)
 (CHANGING RANGES IN CONS-TRIAN)
 (SAVING PAZIFIST R&Q 0.0 0.667 0.75 1.0)
 (REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER *BOTTOM*)

PAZIFIST QUAEKER wird an die Liste ***agenda*** angehängt.

(REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER *TOP*)

(REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER REPUBLIKANER)
 (CHANGING RANGES IN CONS-TRIAN)
 (CHANGING RANGES IN CONS-BAYES)
 (SAVING PAZIFIST REPUBLIKANER 0.0 0.505 0.248 0.25)

Die nächste Kante, die in der Liste ***agenda*** steht, ist PAZIFIST REPUBLIKANER.

(REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER *TOP*)
 (CHANGING RANGES IN CONS-TRIAN)
 (SAVING *TOP* REPUBLIKANER 0.0 0.805 1.0 1.0)

(REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER R&Q)
 (CHANGING RANGES IN CONS-TRIAN)
 (SAVING PAZIFIST R&Q 0.0 0.505 0.75 1.0)

(REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER *BOTTOM*)
 (REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER QUAEKER)

Damit wäre nach der ursprünglichen Initialisierung der ***agenda*** der Propagierungsalgorithmus beendet. Wären keine Kanten mehr verfeinert worden, so könnte an dieser Stelle abgebrochen werden.

(REGARDED TRIANGULAR CASE - *TOP* QUAEKER R&Q)
 (CHANGING RANGES IN CONS-TRIAN)
 (SAVING *TOP* R&Q 0.0 0.338 1.0 1.0)

(REGARDED TRIANGULAR CASE - *TOP* QUAEKER *BOTTOM*)
 (REGARDED TRIANGULAR CASE - *TOP* QUAEKER REPUBLIKANER)
 (REGARDED TRIANGULAR CASE - *TOP* QUAEKER PAZIFIST)
 (REGARDED TRIANGULAR CASE - QUAEKER R&Q *BOTTOM*)
 (REGARDED TRIANGULAR CASE - QUAEKER R&Q *TOP*)

(REGARDED TRIANGULAR CASE - QUAEKER R&Q REPUBLIKANER)
 (CHANGING RANGES IN CONS-AND)
 (SAVING R&Q REPUBLIKANER 1.0 1.0 0.33 0.34)

(REGARDED TRIANGULAR CASE - QUAEKER R&Q PAZIFIST)
 (REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER R&Q)
 (REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER *BOTTOM*)

(REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER *TOP*)
 (REGARDED TRIANGULAR CASE - PAZIFIST QUAEKER REPUBLIKANER)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q QUAEKER)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q *BOTTOM*)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q *TOP*)

(REGARDED TRIANGULAR CASE - PAZIFIST R&Q REPUBLIKANER)
 (CHANGING RANGES IN CONS-TRIAN)
 (SAVING PAZIFIST R&Q 0.0 0.505 0.75 0.758)

(REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER *TOP*)
 (REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER R&Q)
 (REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER *BOTTOM*)
 (REGARDED TRIANGULAR CASE - PAZIFIST REPUBLIKANER QUAEKER)

(REGARDED TRIANGULAR CASE - *TOP* REPUBLIKANER R&Q)
 (CHANGING RANGES IN CONS-TRIAN)
 (SAVING *TOP* R&Q 0.0 0.274 1.0 1.0)

Ab diesem Zeitpunkt sind alle Werte stabil. Was allerdings erst bekannt ist, wenn alle weiteren Fälle durchgerechnet wurden:

(REGARDED TRIANGULAR CASE - *TOP* REPUBLIKANER *BOTTOM*)
 (REGARDED TRIANGULAR CASE - *TOP* REPUBLIKANER QUAEKER)
 (REGARDED TRIANGULAR CASE - *TOP* REPUBLIKANER PAZIFIST)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q QUAEKER)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q *BOTTOM*)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q *TOP*)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q REPUBLIKANER)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q QUAEKER)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q *BOTTOM*)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q REPUBLIKANER)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q PAZIFIST)
 (REGARDED TRIANGULAR CASE - R&Q REPUBLIKANER *TOP*)
 (REGARDED TRIANGULAR CASE - R&Q REPUBLIKANER *BOTTOM*)
 (REGARDED TRIANGULAR CASE - R&Q REPUBLIKANER QUAEKER)
 (REGARDED TRIANGULAR CASE - R&Q REPUBLIKANER PAZIFIST)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q QUAEKER)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q *BOTTOM*)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q *TOP*)
 (REGARDED TRIANGULAR CASE - PAZIFIST R&Q REPUBLIKANER)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q QUAEKER)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q *BOTTOM*)

(REGARDED TRIANGULAR CASE - *TOP* R&Q REPUBLIKANER)
 (REGARDED TRIANGULAR CASE - *TOP* R&Q PAZIFIST)

Die Liste ***agenda*** ist damit abgearbeitet, d.h. der Propagierungsalgorithmus ist beendet, und es können keine Verfeinerungen mehr berechnet werden.

An dieser Stelle werden nun die leeren Konzepte bearbeitet und mit entsprechenden p-Konditionierungen versehen:

(SAVING *BOTTOM* PAZIFIST 1.0 1.0 0.0 0.0)
 (SAVING *BOTTOM* QUAEKER 1.0 1.0 0.0 0.0)
 (SAVING *BOTTOM* R&Q 1.0 1.0 0.0 0.0)
 (SAVING *BOTTOM* *TOP* 1.0 1.0 0.0 0.0)
 (SAVING *BOTTOM* REPUBLIKANER 1.0 1.0 0.0 0.0)

(in diesem Beispiel gab es nur ein leeres Konzept: *BOTTOM*)

Das System gibt jetzt alle neu berechneten Kanten in alphabetischer Reihenfolge aus:

(THESE RANGES ARE NEW OR UPDATED)
 (*TOP* - > QUAEKER [0.0 0.563])
 (*TOP* - > R&Q [0.0 0.274])
 (*TOP* - > REPUBLIKANER [0.0 0.805])
 (PAZIFIST - > R&Q [0.0 0.505])
 (PAZIFIST - > REPUBLIKANER [0.0 0.505])
 (QUAEKER - > PAZIFIST [0.9 0.903])
 (QUAEKER - > R&Q [0.4 0.6])
 (R&Q - > PAZIFIST [0.75 0.758])
 (REPUBLIKANER - > PAZIFIST [0.248 0.25])
 (REPUBLIKANER - > R&Q [0.33 0.34])

An dieser Stelle soll auch auf die Mächtigkeit der Constraints für die logischen Abhängigkeiten hingewiesen werden. Ohne die Anwendung dieses Constraints werden folgende Kanten schlechter berechnet:

(QUAEKER \Leftrightarrow > R&Q [0.4 1.0])
 (REPUBLIKANER \Leftrightarrow > R&Q [0.33 0.85])

8.2 Einige weitere Beispiele

8.2.1 Barking Cats

Die TBox:

```
(defprimconcept DOMESTIC)
(defprimconcept DOGS DOMESTIC)
(defprimconcept BARKING-ANIMAL)
(defprimconcept CATS DOMESTIC)
```

Die PBox:

```
(DOMESTIC DOGS 0.7 0.7)
(DOGS BARKING-ANIMAL 0.9 0.9)
(DOMESTIC CATS 0.3 0.3)
```

Mit Hilfe dieser Angaben in T- und PBox wollen wir herausfinden, wie groß der Anteil der bellenden Tiere unter den Haustieren ist (siehe Abb. 8.2).

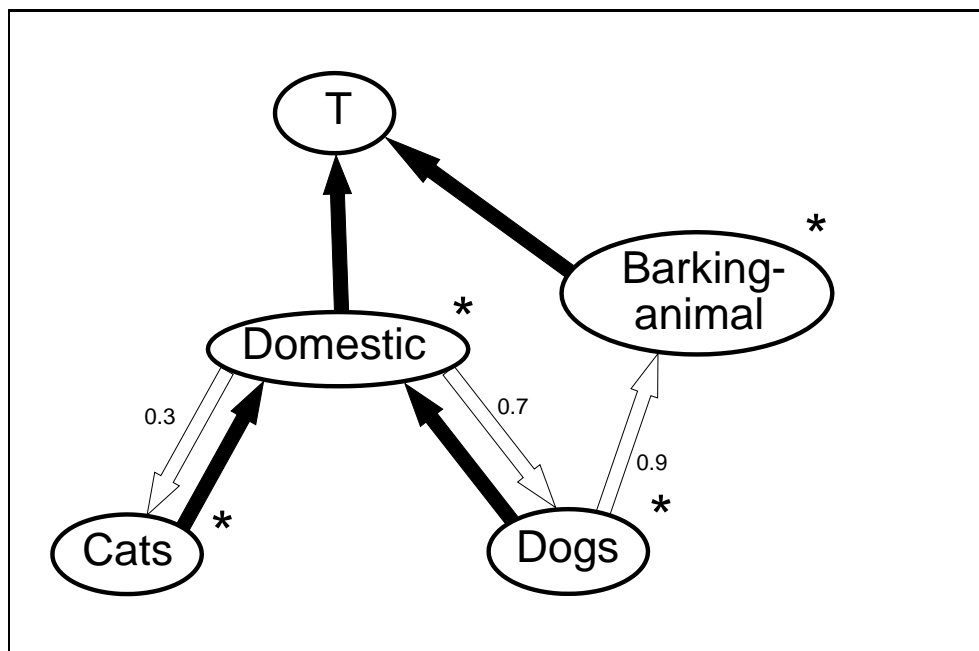


Abbildung 8.2: Barking Cats

Ergebnis der Propagation:

BARKING-ANIMAL - > CATS [0.0 0.476]

DOGS - > CATS [0.0 0.429]

DOMESTIC - > BARKING-ANIMAL [0.63 0.93]

Die Kante DOMESTIC \Leftrightarrow > BARKING-ANIMAL wird zwar wie erwartet berechnet, aber die anderen Ergebnisse zeigen an, daß diese Modellierung doch nicht unseren Vorstellungen von der Realität entsprechen. Die gegebenen oder berechneten Intervalle sollten also weiter verfeinert werden:

CATS – > DOGS : [0.0 0.0]

BARKING-ANIMAL – > DOGS : [1.0 1.0]

Erneutes Propagieren ergibt dann:

BARKING-ANIMAL – > CATS [0.0 0.0]

BARKING-ANIMAL \Rightarrow DOMESTIC (SUBSUMPTION)

CATS – > BARKING-ANIMAL [0.0 0.0]

DOMESTIC – > BARKING-ANIMAL [0.63 0.63]

Der Kante DOMESTIC \Leftrightarrow > BARKING-ANIMAL, die uns hier besonders interessierte, wird nun ein exakter Wert zugeordnet.

8.2.2 Birds

Die TBox:

(defprimconcept FLYING-OBJECT)

(defprimconcept BIRD)

(defprimconcept ANTARCTIC-BIRD BIRD)

(defprimconcept PENGUIN ANTARCTIC-BIRD)

Die PBox:

(BIRD FLYING-OBJECT 0.95 1)

(BIRD ANTARCTIC-BIRD 0.2 0.2)

(PENGUIN FLYING-OBJECT 0 0)

Bei diesem Beispiel soll der Anteil der Pinguine unter den Vögeln bestimmt werden. Den Pinguinen kommt hier eine Sonderstellung unter den Vögeln zu, da sie im Gegensatz zu den meisten anderen Vögeln nicht fliegen können.

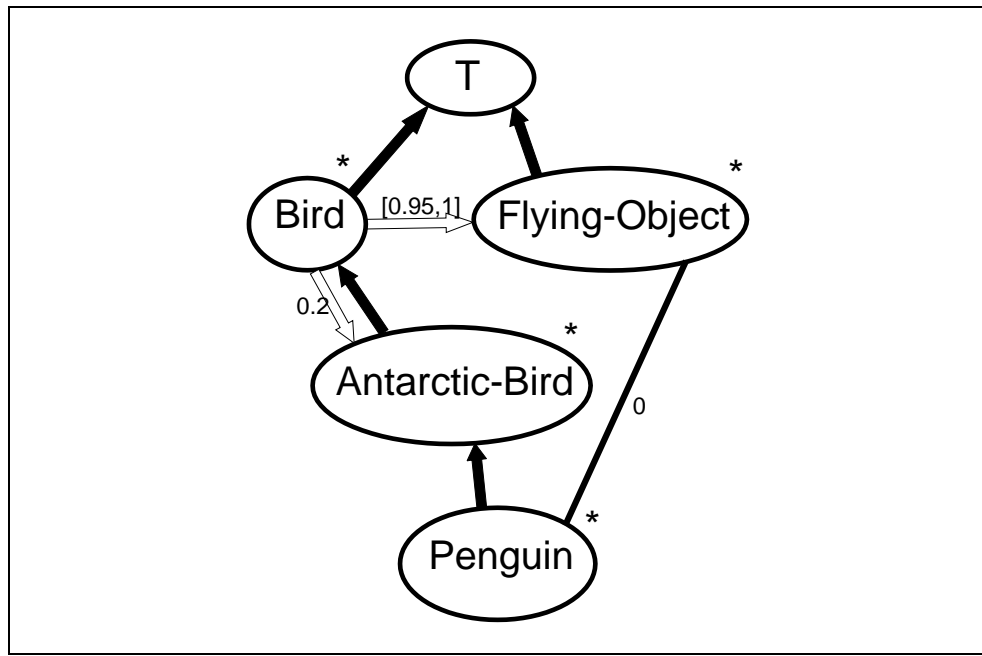


Abbildung 8.3: Birds

Ergebnis der Propagierung:

ANTARCTIC-BIRD - > FLYING-OBJECT [0.75 1.0]

ANTARCTIC-BIRD - > PENGUIN [0.0 0.25]

BIRD - > PENGUIN [0.0 0.05]

FLYING-OBJECT - > ANTARCTIC-BIRD [0.0 0.211]

Das Ergebnis besagt, daß bis zu 5% der Vögel Pinguine sein können. Das entspricht genau dem Wert, den man intuitiv aus der Aussage folgern würde, daß mindestens 95% der Vögel fliegen können, die Pinguine jedoch nicht.

8.2.3 Birds 2

Die TBox:

```
(defprimconcept ANIMAL)
```

```
(defprimconcept MOVEMENT)
```

```
(defprimconcept FLYING MOVEMENT)
```

```
(defconcept MOVING-OBJECT (all MOVE-BY MOVEMENT))
```

```
(defconcept FLYING-OBJECT (all MOVE-BY FLYING))
```

```
(defprimconcept BIRD ANIMAL)
```

```
(defprimconcept ANTARCTIC-ANIMAL ANIMAL)
```

```
(defconcept ANTARCTIC-BIRD (and ANTARCTIC-ANIMAL BIRD))
```

```
(defconcept NOT-ANTARCTIC-BIRD (and BIRD (not ANTARCTIC-BIRD)))
```

```
(defprimconcept PENGUIN ANTARCTIC-BIRD)
```

Die PBox:

ANTARCTIC-BIRD \rightarrow FLYING-OBJECT [0.75 1.0]

ANTARCTIC-BIRD \rightarrow PENGUIN [0.0 0.25]

BIRD \rightarrow PENGUIN [0.0 0.05]

FLYING-OBJECT \rightarrow ANTARCTIC-BIRD [0.0 0.211]

Hier wurden - unter Beibehaltung der PBox - die terminologischen Axiome erheblich erweitert. Interessant ist hier die Beobachtung, wie sich die - anfänglich nur drei - p-Konditionierungen über die gesamte Taxonomie ausbreiten.

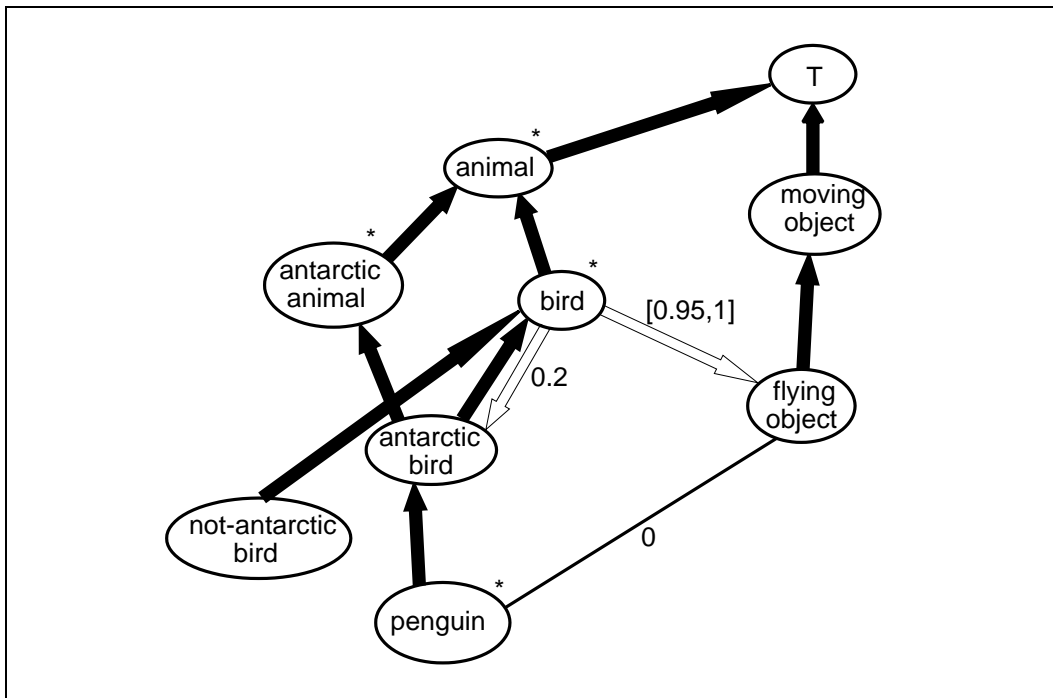


Abbildung 8.4: Birds2 (ohne Rollen)

Ergebnis der Propagierung:

ANIMAL \rightarrow ANTARCTIC-BIRD [0.0 0.2]

ANIMAL \rightarrow NOT-ANTARCTIC-BIRD [0.0 0.8]

ANIMAL \rightarrow PENGUIN [0.0 0.05]

ANTARCTIC-ANIMAL \rightarrow PENGUIN [0.0 0.25]

ANTARCTIC-BIRD \rightarrow FLYING-OBJECT [0.75 1.0]

ANTARCTIC-BIRD \rightarrow MOVING-OBJECT [0.75 1.0]

ANTARCTIC-BIRD \rightarrow PENGUIN [0.0 0.25]

BIRD \rightarrow ANTARCTIC-ANIMAL [0.2 0.2]

BIRD - > MOVING-OBJECT [0.95 1.0]
 BIRD - > NOT-ANTARCTIC-BIRD [0.8 0.8]
 BIRD - > PENGUIN [0.0 0.05]
 FLYING-OBJECT - > ANTARCTIC-BIRD [0.0 0.211]
 FLYING-OBJECT - > NOT-ANTARCTIC-BIRD [0.0 0.842]
 MOVING-OBJECT - > ANTARCTIC-BIRD [0.0 0.211]
 MOVING-OBJECT - > NOT-ANTARCTIC-BIRD [0.0 0.842]
 MOVING-OBJECT - > PENGUIN [0.0 0.053]
 NOT-ANTARCTIC-BIRD - > FLYING-OBJECT [0.938 1.0]
 NOT-ANTARCTIC-BIRD - > MOVING-OBJECT [0.938 1.0]

8.2.4 Burglary

A: Alarm event
 B: Earthquake event
 C: Burglary Assume ¹

Die TBox:

(defprimconcept A)
 (defprimconcept B)
 (defprimconcept C)
 (defconcept AB (and A B))
 (defconcept BC (and B C))
 (defconcept AC (and A C))

Die PBox:

(A B 0.06 0.06 0.7 0.7)
 (A C 0.95 0.95)
 (B C 0.2 0.2)

Die Fragestellung bei diesem Beispiel aus der Literatur lautet: Wie wahrscheinlich ist es, daß ein Einbrecher im Haus ist, wenn während eines Erdbebens die Alarmanlage ertönt?

¹Wir benutzen hier die Begriffe aus [Güntzer et al. 91], obwohl sie für unsere Anwendung eigentlich zu ungenau sind: Ein Alarm kann kein Erdbeben sein, etc. Gemeint ist hier eigentlich: Zeitpunkt des Alarms, Zeitpunkt des Erdbebens und Zeitpunkt, von dem angenommen wird, daß ein Einbruch stattfindet

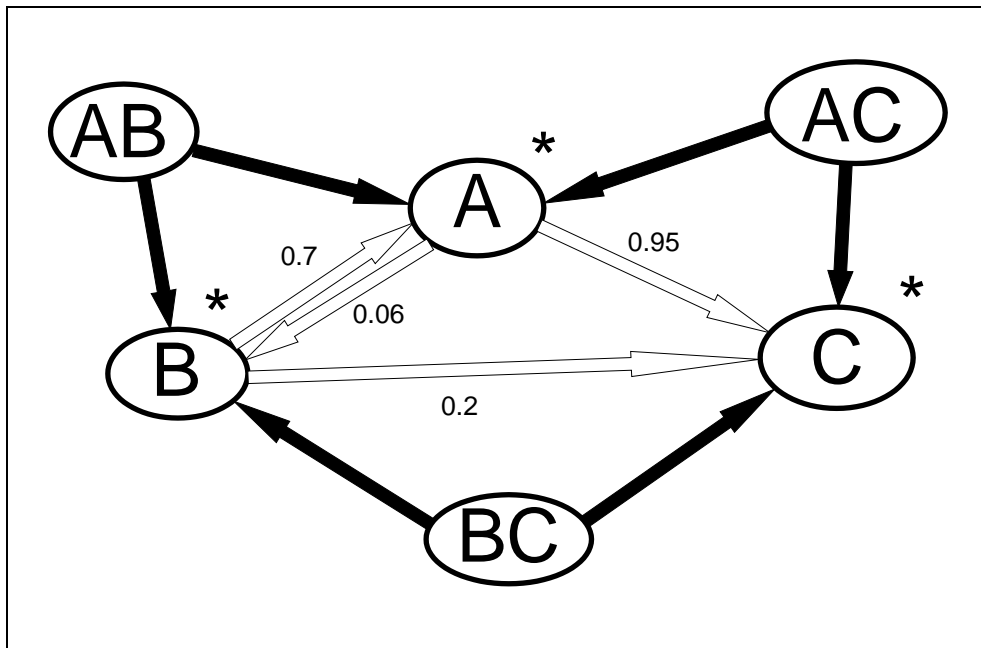


Abbildung 8.5: Burglary

Ergebnis der Propagierung:

AB - > AC [0.167 1.0]
 AB - > BC [0.0 0.286]
 AB - > C [0.167 0.286]
 AC - > AB [0.011 0.063]
 AC - > BC [0.0 0.018]
 AC - > B [0.011 0.063]
 A - > AB [0.06 0.06]
 A - > AC [0.95 0.95]
 A - > BC [0.0 0.017]
 B - > AB [0.7 0.7]
 B - > AC [0.117 0.7]
 B - > BC [0.2 0.2]
 C - > AB [0.0 0.018]
 C - > BC [0.0 0.018]
 C - > B [0.0 0.018]

Die Antwort auf diese Frage gibt das System durch die Kante $AB \Leftrightarrow C$ [0.167 0.286]. So abwegig, wie man spontan vermuten würde, ist diese Überlegung also gar nicht. Das kann u.a. darauf zurückgeführt werden, daß der Kante $B \Leftrightarrow C$ ein ziemlich hoher Wert (0.2) zugeordnet wurde.

8.2.5 Children

Die TBox:

```
(defprimconcept CHILDREN)
(defprimconcept YOUNG)
(defprimconcept STUDENT)
(defprimconcept SINGLE)
(defprimconcept SPORT)
```

Die PBox:

```
(CHILDREN YOUNG 0 0.05 0 0.05)
(STUDENT YOUNG 0.85 0.95 0.25 0.35)
(SINGLE YOUNG 0.6 0.8 0.9 1)
(SPORT YOUNG 0.9 1 0.8 0.9)
(SINGLE SPORT 0.7 0.9 0.8 0.85)
(CHILDREN SINGLE 0 0.05 0.05 1)
(SPORT STUDENT 0.4 0.6 0.7 0.9)
```

Hier werden dem System eine größere Menge statistischer Daten über Studenten zur Auswertung übergeben.

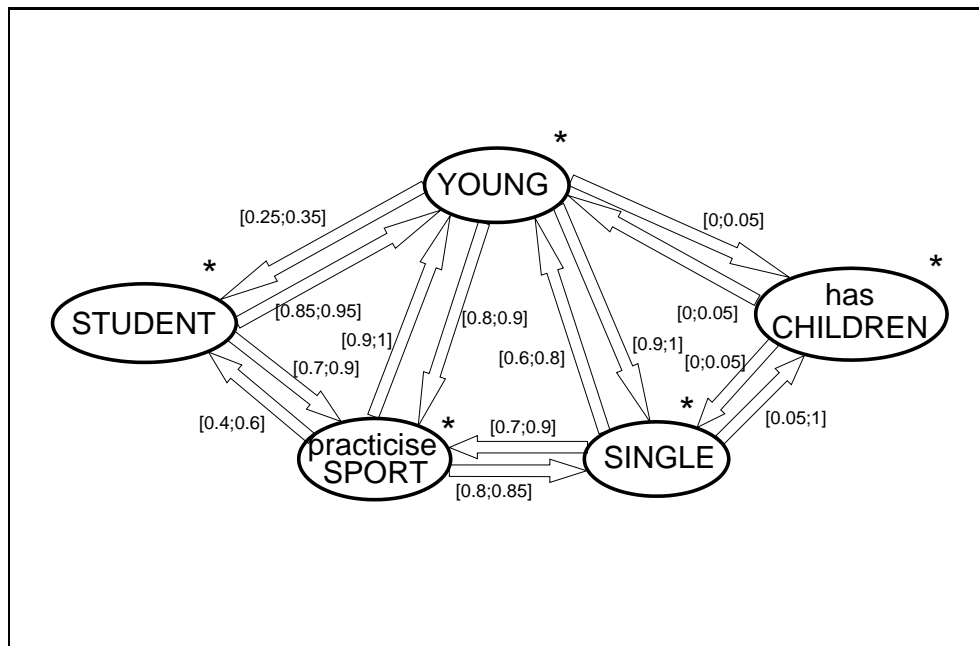


Abbildung 8.6: Children

Ergebnis der Propagierung:

CHILDREN - > SPORT [0.0 0.143]
 CHILDREN - > STUDENT [0.0 0.173]
 CHILDREN - > YOUNG [0.0 0.044]
 SINGLE - > CHILDREN [0.05 0.29]
 SINGLE - > SPORT [0.7 0.745]
 SINGLE - > STUDENT [0.217 0.366]
 SINGLE - > YOUNG [0.752 0.8]
 SPORT - > CHILDREN [0.0 0.154]
 SPORT - > STUDENT [0.4 0.426]
 SPORT - > YOUNG [0.9 0.958]
 STUDENT - > CHILDREN [0.0 0.279]
 STUDENT - > SINGLE [0.592 1.0]
 STUDENT - > SPORT [0.846 0.9]
 STUDENT - > YOUNG [0.85 0.905]
 YOUNG - > SINGLE [0.9 0.954]
 YOUNG - > SPORT [0.834 0.888]
 YOUNG - > STUDENT [0.329 0.35]

An diesem Ergebnis sieht man, daß nicht nur gegebene Werte erheblich verfeinert werden konnten, sondern auch nicht angegebene Kanten, wie z.B. STUDENT \Leftrightarrow CHILDREN genaue Werte zugeordnet werden konnten.

8.2.6 Songwriter

Die TBox:

```
(defprimconcept PERSON)
(defprimconcept ARTIST PERSON)
(defconcept NO_ARTIST (and PERSON (not ARTIST)))
(defprimconcept MUSICIAN ARTIST)
(defprimconcept OTHER_ARTIST ARTIST)
(defprimconcept WRITER OTHER_ARTIST)
(defconcept PAINTER (and OTHER_ARTIST (not WRITER)))
(defconcept SONGWRITER (and MUSICIAN WRITER))
```

Die PBox:

```
(PERSON ARTIST 0.2 0.4)
(ARTIST MUSICIAN 0.3 0.6)
(ARTIST OTHER_ARTIST 0.5 0.6)
(OTHER_ARTIST PAINTER 0.5 0.7)
(WRITER SONGWRITER 0.1 0.5)
(MUSICIAN SONGWRITER 0.4 0.6)
```

Die Motivation bei diesem Beispiel, das statistische Daten über Künstler ausgewertet, besteht aus zwei Punkten: Zum einen sollten die Constraints für logische Bezüge getestet werden, und zum anderen wollten wir wissen, wie genau mit wenigen - grob geschätzten - statistischen Angaben Werte berechnet werden können, die sich schlecht abschätzen lassen, wie z.B. die Kante PERSON \Leftrightarrow SONGWRITER.

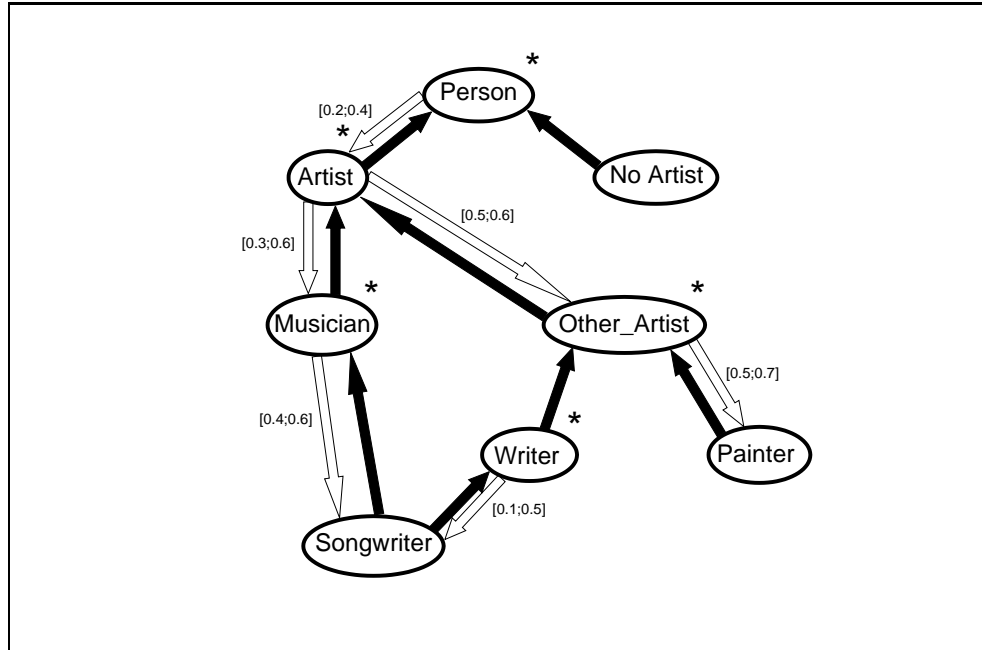


Abbildung 8.7: Songwriter

Ergebnis der Propagierung:

ARTIST - > MUSICIAN [0.3 0.375]
 ARTIST - > PAINTER [0.25 0.36]
 ARTIST - > SONGWRITER [0.12 0.15]
 ARTIST - > WRITER [0.24 0.3]
 MUSICIAN - > OTHER_ARTIST [0.4 1.0]
 MUSICIAN - > PAINTER [0.0 0.6]
 MUSICIAN - > SONGWRITER [0.4 0.5]
 MUSICIAN - > WRITER [0.4 0.5]
 OTHER_ARTIST - > MUSICIAN [0.2 0.625]
 OTHER_ARTIST - > PAINTER [0.5 0.6]
 OTHER_ARTIST - > SONGWRITER [0.2 0.25]
 OTHER_ARTIST - > WRITER [0.4 0.5]
 PAINTER - > MUSICIAN [0.0 0.9]
 PERSON - > MUSICIAN [0.06 0.15]

PERSON - > NO_ARTIST [0.6 0.8]
 PERSON - > OTHER_ARTIST [0.1 0.24]
 PERSON - > PAINTER [0.05 0.144]
 PERSON - > SONGWRITER [0.024 0.06]
 PERSON - > WRITER [0.048 0.12]
 WRITER - > MUSICIAN [0.4 0.5]
 WRITER - > SONGWRITER [0.4 0.5]

Der Kante PERSON \Leftrightarrow SONGWRITER wurde ein ziemlich exakter Wert zugeordnet. Interessant bei der Konstruktion dieses Beispiels war auch, daß die ersten Versuche inkonsistentes Wissen beinhalten, d.h. daß unsere intuitiven Schätzungen widersprüchlich sein können, ohne daß wir es bemerken.

8.2.7 Sprinkler

Die TBox:

(defprimconcept GROUND_IS_WET_EVENT)

(defprimconcept SPRINKLER_WAS_ON_EVENT GROUND_IS_WET_EVENT)

(defprimconcept IT_RAINED_LAST_NIGHT_EVENT GROUND_IS_WET_EVENT)

Die PBox:

(GROUND_IS_WET_EVENT SPRINKLER_WAS_ON_EVENT 0.1 0.1)

(GROUND_IS_WET_EVENT IT_RAINED_LAST_NIGHT_EVENT 0.9 0.9)

Es hat in der letzten Nacht geregnet und der Boden ist naß. Mit welcher Wahrscheinlichkeit war die Sprinkleranlage angeschaltet?

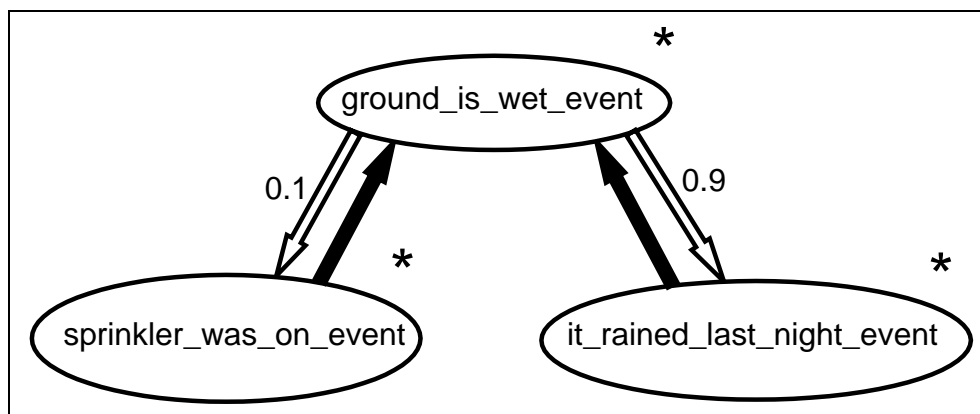


Abbildung 8.8: Sprinkler

Ergebnis der Propagierung:

IT_RAINED_LAST_NIGHT_EVENT - > SPRINKLER_WAS_ON_EVENT [0.0 0.111]

Kapitel 9

Bedienungsanleitung und Benutzeroberfläche

In diesem Kapitel werden Bedienung und graphische Benutzeroberfläche des Systems erläutert. Zunächst wird das Laden des Programms erklärt. Es folgt eine Beschreibung der Oberfläche und der \mathcal{ALCP} -relevanten Funktionen. Ein weiterer Abschnitt befaßt sich mit den Funktionen, die im Kommando-Modus ohne die Oberfläche eingegeben werden können. Abschließend wird für die Benutzung des Systems sowohl mit als auch ohne Oberfläche eine „Sitzung“ für das Beispiel „Birds 2“ (siehe Abb.8.4) vorgeführt.

9.1 Laden des Systems

Das Starten des Systems setzt die Zugriffsberechtigung auf die LISP-Version „Allegro CL 4.2“ voraus. Dazu kann man sich z.B. auf einem der serv-Rechner am DFKI (serv-100, serv-201, serv-202 etc.) einloggen. Dort wechselt man zunächst mit `cd /home/endres/ALCP/` das Verzeichnis.

Danach lädt man mit dem Kommando `/opt/acl/bin/clim2xm_composer` die oben genannte LISP-Version.

Dem Benutzer stehen nun zwei verschiedene Laderoutinen zur Verfügung:

- `(load "alcp")`
zum Laden des Systems ohne CLIM-Oberfläche - im folgenden \mathcal{ALCP} genannt
- `(load "Xalcp")`
zum Laden des Systems mit CLIM-Oberfläche unter XWindow - im folgenden \mathcal{XALCP} genannt
(Am Rechner, an dem man sich befindet, muß „access control disabled“ sein. Dazu verwendet man das Unix-Kommando `'xhost +'`.)

Das System ist jetzt betriebsbereit.

9.2 Die Oberfläche

Die Oberfläche wurde in CLIM (Common Lisp Interface Manager) implementiert und besteht aus einer Menüleiste mit sechs Pull-Down-Menüs und vier Fenstern (*panes*) (siehe Abb. 9.1). Das Fenster links oben (*graphic pane*) ist für graphische Darstellungen reserviert. Falls eine TBox geladen ist, wird sie hier dargestellt. Rechts daneben ist das *status pane*. Hier befinden sich die Status-Anzeigen und verschiedene Schalter. In der unteren Bildschirmhälfte befindet sich links ein Fenster für Textausgaben (*output pane*) und rechts ein Fenster für Benutzereingaben (*listener pane*). Der äußerste rechte Menüpunkt (PBox Tools) enthält die PBox-spezifischen Funktionen.

Ebenfalls wichtig für die Benutzung des Systems sind die Menüpunkte unter „STATUS“, u.a.:

- KB-Path (Knowledge Base-Path)
Eingeben eines neuen Pfades, aus dem die T- und PBoxen geladen werden sollen. Eingegeben werden kann nach Auswahl dieses Punktes z.B.: `"/home/endres/ALCP/KB/"`. Das ist der Pfad, in dem sich u.a. die Beispiele aus dem vorigen Kapitel befinden.
Pfadnamen müssen mit einem „/“ (Slash) enden.
- TBox
Laden einer TBox. Ein Auswahlmenü wird angezeigt.
- PBox
Laden einer PBox. Auch hier erscheint ein Auswahlmenü.

Die eingestellten Werte (Pfadnamen, Namen der Boxen etc.) werden im *status pane* angezeigt.

Die Menüpunkte unter „PBox Tools“ greifen auf die in Kapitel 4.4 beschriebenen Funktionen zu, daher kann an dieser Stelle auf eine ausführliche Beschreibung verzichtet werden.

Hier eine kurze Übersicht:

- Load PKB
Laden einer PBox. Es erscheint ein Fenster mit allen im aktuellen Pfad befindlichen PBoxen. Die gewünschte PBox kann durch Anklicken ausgewählt werden.
- Save PKB
Speichern der aktuellen PBox in der Datei "`<filename>`". Der Dateiname wird im Fenster *listener pane* eingegeben.

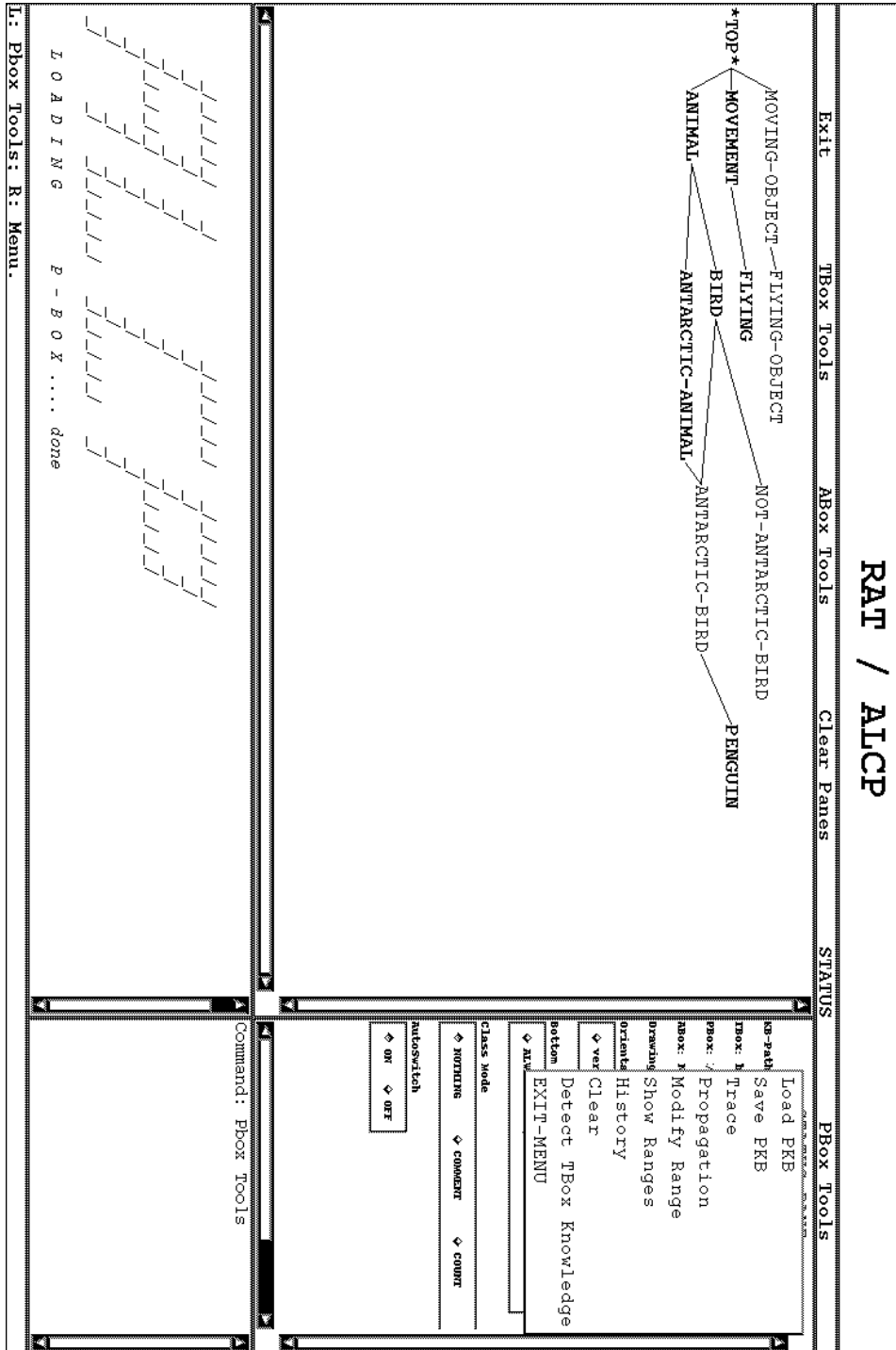


Abbildung 9.1: Die Benutzeroberfläche

- Trace
 - Hier wird der „Trace“-Modus eingestellt. Es erscheint ein Untermenü:
 - ON
 - „Trace“ wird angeschaltet (Ausgabe im *output pane*).
 - OFF
 - „Trace“ wird abgeschaltet.
 - in file
 - „Trace“ wird in die Datei “<filename>“ umgeleitet. Der Dateiname wird im *listener pane* eingegeben.
- Propagation
 - Der Propagierungsalgorithmus wird gestartet.
- Modify Range
 - by slider
 - Manual

Der Benutzer kann über den SLIDER (siehe Abschnitt 4.3) oder über die Tastatur neue Werte für ein Intervall angeben. Zunächst muß er dabei durch Anklicken von zwei Konzeptnamen der Taxonomie im *graphic pane* angeben, welche Kante er verändern will. Anschließend wird er gefragt, ob er weitere Werte verändern möchte. Hat der Benutzer alle gewünschten Veränderungen durchgeführt, wird mit den neuen Werten wieder propagiert und das Ergebnis angezeigt.

- Show Ranges
 - Der Benutzer kann nun auswählen, welche p-Konditionierungen angezeigt werden sollen:
 - given ranges
 - Die von TBox und PBox vorgegebenen Werte.
 - given by TBox
 - Von der TBox vorgegebene Werte.
 - given by PBox
 - Von der PBox vorgegebene Werte.
 - new
 - Neu berechnete Werte.
 - all
 - Alle Werte mit Ausnahme der Subsumtionen.

- all subsumtions
Die Subsumtionen mit Ausnahme jener, die von BOTTOM wegführen oder auf TOP zuführen.
- all 0-ranges
Die paarweise disjunkten Konzepte.
- pairwise
Eine einzelne p-Konditionierung, die durch Anklicken von zwei Konzeptnamen in der Taxonomie angegeben wird.
- History
Die „History“ der p-Konditionierungen wird angezeigt. Auch hier hat der Benutzer verschiedene Möglichkeiten:
 - all ranges
Die „History“ aller p-Konditionierungen.
 - pairwise
Die „History“ einer bestimmten p-Konditionierung, die durch Anklicken von zwei Konzeptnamen in der Taxonomie angegeben wird.
- Clear
Löschen aller probabilistischen Daten.
- Detect TBox Knowledge
Anzeigen von neuerworbenem terminologischem Wissen.
- EXIT-MENU
Verlassen dieses Menüs.

9.3 Befehlsübersicht

Nachfolgend eine Übersicht über die Schnittstellenfunktionen von \mathcal{ALCP} (Nähere Informationen entnehmen Sie bitte Kapitel 4.4).

- Wissensbasen
 - (*t-load* “<file>“)
 - (*p-load* “<file>“)
 - (*p-save* “<file>“)
 - (*detect-tbox-knowledge*)
 - (*clear*)
- p-Konditionierungen eingeben

- (*insert-new-range* *<konzept1>* *<konzept2>* *p q* *&optional p' q'*)
- p-Konditionierungen anzeigen
 - (*given-ranges*)
 - (*given-by-pbox*)
 - (*given-by-tbox*)
 - (*new-ranges*)
 - (*all-1-ranges*)
 - (*all-0-ranges*)
 - (*show-range* *<konzept1>* *<konzept2>*)
- Propagierung
 - (*start-propagation*)
- History ausgeben
 - (*history-range* *<konzept1>* *<konzept2>*)
 - (*maphistory*)
- Trace umschalten
 - (*trace-on*)
 - (*trace-off*)
 - (*trace-in* “*<filename>*“)

9.4 Beispielsitzung

Anhand des Beispiels „Birds 2“ (siehe Abbildung 8.4) wird nun die Bedienung des Programmes demonstriert. Der Ablauf der Sitzung gliedert sich jeweils wie folgt:

1. Laden der TBox
2. Ausgabe der TBox-Werte
3. Laden der PBox
4. Ausgabe der PBox-Werte
5. Propagierung

6. Verändern der Kante *antarctic-bird* \rightarrow *flying-object* auf den Wert [1;1]
7. Erneut propagieren
8. Ausgabe des neuerworbenen terminologischen Wissens

9.4.1 *ALCP*

1. Laden der TBox

```
(t-load “/home/endres/ALCP/KB/birds2.tbox“)
; Loading /home/endres/ALCP/KB/birds2.tbox.
T
```

2. Ausgabe der TBox-Werte

```
(given-by-tbox)
THESE P-CONDITIONINGS WERE GIVEN BY T-BOX
ANTARCTIC-ANIMAL => ANIMAL (SUBSUMPTION)
ANTARCTIC-ANIMAL AND NOT-ANTARCTIC-BIRD ARE DISJOINT
ANTARCTIC-BIRD => ANIMAL (SUBSUMPTION)
ANTARCTIC-BIRD => ANTARCTIC-ANIMAL (SUBSUMPTION)
ANTARCTIC-BIRD => BIRD (SUBSUMPTION)
ANTARCTIC-BIRD AND NOT-ANTARCTIC-BIRD ARE DISJOINT
BIRD => ANIMAL (SUBSUMPTION)
FLYING-OBJECT => MOVING-OBJECT (SUBSUMPTION)
FLYING => MOVEMENT (SUBSUMPTION)
NOT-ANTARCTIC-BIRD => ANIMAL (SUBSUMPTION)
NOT-ANTARCTIC-BIRD AND ANTARCTIC-ANIMAL ARE DISJOINT
NOT-ANTARCTIC-BIRD AND ANTARCTIC-BIRD ARE DISJOINT
NOT-ANTARCTIC-BIRD => BIRD (SUBSUMPTION)
NOT-ANTARCTIC-BIRD AND PENGUIN ARE DISJOINT
PENGUIN => ANIMAL (SUBSUMPTION)
PENGUIN => ANTARCTIC-ANIMAL (SUBSUMPTION)
PENGUIN => ANTARCTIC-BIRD (SUBSUMPTION)
PENGUIN => BIRD (SUBSUMPTION)
PENGUIN AND NOT-ANTARCTIC-BIRD ARE DISJOINT
T
```

3. Laden der PBox

```
(p-load “/home/endres/ALCP/KB/birds2.pbox“)
T
```

4. Ausgabe der PBox-Werte

(given-by-pbox)

THESE P-CONDITIONINGS WERE GIVEN BY PBOX

BIRD - > ANTARCTIC-BIRD [0.2 0.2]

BIRD - > FLYING-OBJECT [0.95 1.0]

FLYING-OBJECT - > PENGUIN [0.0 0.0]

PENGUIN - > FLYING-OBJECT [0.0 0.0]

T

5. Propagierung

(start-propagation)

THESE RANGES ARE NEW OR UPDATED

TOP* - > ANTARCTIC-BIRD [0.0 0.2]

TOP* - > NOT-ANTARCTIC-BIRD [0.0 0.8]

TOP* - > PENGUIN [0.0 0.05]

ANIMAL - > ANTARCTIC-BIRD [0.0 0.2]

ANIMAL - > NOT-ANTARCTIC-BIRD [0.0 0.8]

ANIMAL - > PENGUIN [0.0 0.05]

ANTARCTIC-ANIMAL - > PENGUIN [0.0 0.25]

ANTARCTIC-BIRD - > FLYING-OBJECT [0.75 1.0]

ANTARCTIC-BIRD - > MOVING-OBJECT [0.75 1.0]

ANTARCTIC-BIRD - > PENGUIN [0.0 0.25]

BIRD - > ANTARCTIC-ANIMAL [0.2 0.2]

BIRD - > MOVING-OBJECT [0.95 1.0]

BIRD - > NOT-ANTARCTIC-BIRD [0.8 0.8]

BIRD - > PENGUIN [0.0 0.05]

FLYING-OBJECT - > ANTARCTIC-BIRD [0.0 0.211]

FLYING-OBJECT - > NOT-ANTARCTIC-BIRD [0.0 0.842]

MOVING-OBJECT - > ANTARCTIC-BIRD [0.0 0.211]

MOVING-OBJECT - > NOT-ANTARCTIC-BIRD [0.0 0.842]

MOVING-OBJECT - > PENGUIN [0.0 0.053]

NOT-ANTARCTIC-BIRD - > FLYING-OBJECT [0.938 1.0]

NOT-ANTARCTIC-BIRD - > MOVING-OBJECT [0.938 1.0]

T

6. Verändern einer Kante

(insert-new-range 'antarctic-bird 'flying-object 1 1)

T

7. Erneut propagieren

(start-propagation)

EMPTY CONCEPT DETECTED @c[PENGUIN]

THESE RANGES ARE NEW OR UPDATED

TOP* - > PENGUIN [0.0 0.0]

ANIMAL - > PENGUIN [0.0 0.0]

ANTARCTIC-ANIMAL - > PENGUIN [0.0 0.0]

ANTARCTIC-BIRD => MOVING-OBJECT (SUBSUMPTION)

ANTARCTIC-BIRD - > PENGUIN [0.0 0.0]

BIRD - > PENGUIN [0.0 0.0]

FLYING - > PENGUIN [0.0 0.0]

MOVEMENT - > PENGUIN [0.0 0.0]

MOVING-OBJECT - > PENGUIN [0.0 0.0]

PENGUIN => FLYING-OBJECT (SUBSUMPTION)

PENGUIN => FLYING (SUBSUMPTION)

PENGUIN => MOVEMENT (SUBSUMPTION)

PENGUIN => MOVING-OBJECT (SUBSUMPTION)

PENGUIN => NOT-ANTARCTIC-BIRD (SUBSUMPTION)

T

8. Ausgabe des neuerworbenen terminologischen Wissens

(detect-tbox-knowledge)

THE FOLLOWING NEW TBOX-KNOWLEDGE HAS BEEN DETECTED

ANTARCTIC-BIRD => FLYING-OBJECT (SUBSUMPTION)

ANTARCTIC-BIRD => MOVING-OBJECT (SUBSUMPTION)

PENGUIN => *BOTTOM* (SUBSUMPTION)

PENGUIN => FLYING-OBJECT (SUBSUMPTION)

PENGUIN => FLYING (SUBSUMPTION)

PENGUIN => MOVEMENT (SUBSUMPTION)

PENGUIN => MOVING-OBJECT (SUBSUMPTION)

PENGUIN => NOT-ANTARCTIC-BIRD (SUBSUMPTION)

T

9.4.2 XALCP

1. Laden der TBox

- Menü „STATUS“ anklicken
- Es erscheint ein Fenster mit verschiedenen TBoxen. Hier nun *birds2.tbox* anklicken.
- Die TBox erscheint im *graphic pane*.

2. Ausgabe der TBox-Werte

- Menü „PBox Tools“ anklicken
- darunter den Menüpunkt „Show ranges“ auswählen
- hier wiederum das Untermenü „given by TBox“ anklicken
- Im *output pane* erscheint die Meldung:
THESE P-CONDITIONINGS WERE GIVEN BY T-BOX
 ... (gefolgt von denselben Systemmeldungen wie im vorherigen Abschnitt)

3. Laden der PBox

- Menü „STATUS“ anklicken
- Es erscheint ein Fenster mit verschiedenen PBoxen. Hier nun *birds2.pbox* anklicken.
- Im linken, unteren Fenster (*output pane*) erscheint die Meldung:
L O A D I N G P - B O X done

4. Ausgabe der PBox-Werte

- Menü „PBox Tools“ anklicken
- darunter den Menüpunkt „Show ranges“ auswählen
- hier wiederum das Untermenü „given by TBox“ anklicken
- Im *output pane* erscheint die Meldung:
THESE P-CONDITIONINGS WERE GIVEN BY P-BOX
 ...

5. Propagierung

- Menü „PBox Tools“ anklicken
- darunter den Menüpunkt „Propagation“ auswählen
- Im *output pane* erscheint die Meldung:
P R O P A G A T I O N PLEASE WAIT ...
 und später:
THESE RANGES ARE NEW OR UPDATED
 ...

6. Verändern einer Kante

- Menü „PBox Tools“ anklicken
- darunter den Menüpunkt „Modify range“ auswählen
- darunter wiederum die Option „by slider“ anklicken

- Es erscheint die Meldung:
PLEASE ENTER THE FIRST CONCEPT
 - Konzept ANTARCTIC-BIRD in der Taxonomie (graphic pane) anklicken
 - Es erscheint die Meldung:
PLEASE ENTER THE SECOND CONCEPT
 - Konzept FLYING-OBJECT in der Taxonomie anklicken
 - Es erscheint die Meldung
ANTARCTIC BIRD \Leftrightarrow > FLYING-OBJECT HAS ACTUALLY THE VALUE 0.75 1.0
sowie der bereits in Abschnitt 4.3 erwähnte SLIDER mit den entsprechenden Werten.
 - den oberen Regler des SLIDERS anklicken und auf 1 ziehen
 - den „OK“-Button im Sliderfenster anklicken
 - Es erscheint die Meldung (output pane):
INSERT ANOTHER P-CONDITIONING? (Y/N)
 - N tippen, wobei „NO“ auf dem Bildschirm erscheint
7. Erneut propagieren
Erneutes Propagieren erfolgt nun automatisch.
8. Ausgabe des neuerworbenen terminologischen Wissens
- Menü „PBox Tools“ anklicken
 - darunter den Menüpunkt „Detect TBox Knowledge“ auswählen
 - Es erscheint die Meldung:
THE FOLLOWING NEW T-BOX KNOWLEDGE HAS BEEN DETECTED
...

Kapitel 10

Schlußbemerkung und Ausblick

Der vorliegende Bericht beschreibt, wie das in [Heinsohn 94a] eingeführte Konzept der Sprache *ALCP* implementiert und erweitert wurde. Dieses Sprachkonzept wurde eingebettet in ein bestehendes System, welches terminologisches Wissen und Wissen über Pläne modelliert. Dem Benutzer des Programms *XALCP* wurden Funktionen zur Verfügung gestellt, um mit probabilistischem Wissen eine Domäne realitätsnäher als mit rein terminologischem Wissen zu modellieren und sowohl eine größere Anzahl an Inferenzen als auch schärfere Schlußfolgerungen zu gewinnen, sowie Inkonsistenzen zu vermeiden. Wir haben einige Beispiele aus der Literatur getestet und bei den meisten erhielten wir stärkere Einschränkungen der Intervalle, d.h. genauere Informationen.

Mit unserer Arbeit ist der behandelte Themenbereich jedoch noch nicht vollständig abgeschlossen. Es bietet sich an, das bestehende System dahingehend zu erweitern, daß

- neuerworbenes terminologisches Wissen an das KRIS/RAT-System übergeben und unmittelbar in die TBox eingefügt werden kann.
- Modellierung von Unsicherheit bei *assertionalem* Wissen ebenfalls möglich ist. (Die Theorie für ein Unsicherheitsmodell für Objekte ist im fünften Kapitel in [Heinsohn 94a] beschrieben.)
- weitere Constraints für logische Bezüge von mehr als drei Konzepten, z.B. *Additivität* oder *Fokussierungsregel*, eingefügt werden können.

Das erstellte System stellt eine geeignete Basis für diese weitergehenden Arbeiten dar.

Literaturverzeichnis

- [Armager et al. 91] S. **Armager**, D. **Dubois**, and H. **Prade**. *Constraint Propagation with Imprecise Conditional Probabilities*. In: 7th Conference on Uncertainty in artificial intelligence, pp. 26–34, 1991.
- [Baader & Hollunder 91] F. **Baader** and B. **Hollunder**. *KRIS: Knowledge Representation and Inference System*. SIGART Bulletin, 2(3):8–14, 1991.
- [Bibel 93] W. **Bibel**. *Wissensrepräsentation und Inferenz. Eine grundlegende Einführung*. Vieweg Verlag, 1993.
- [Güntzer et al. 91] U. **Güntzer**, W. **Kießling**, and H. **Thöne**. *New Directions For Uncertainty Reasoning In Deductive Databases*. In: ACM SIGMOD International Conference on Management of Data, pp. 178–187, 1991.
- [Heinsohn et al. 92] J. **Heinsohn**, D. **Kudenko**, B. **Nebel**, and H.-J. **Profitlich**. *RAT – Representation of Actions in Terminological Logics*. pp. 16–22. Saarbrücken: DFKI Document D-92-08, German Research Center for Artificial Intelligence, February 1992.
- [Heinsohn et al. 94] J. **Heinsohn**, D. **Kudenko**, B. **Nebel**, and H.-J. **Profitlich**. *An Empirical Analysis of Terminological Representation Systems*. ai journal, 68, 1994.
- [Heinsohn 94a] J. **Heinsohn**. *ALCP – Ein hybrider Ansatz zur Modellierung von Unsicherheit in terminologischen Logiken*. Sankt Augustin: DISKI-55, INFIX-Verlag, 1994.
- [Heinsohn 94b] J. **Heinsohn**. *Probabilistic Description Logics*. In: Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence, Seattle, Washington, July 1994.
- [Kolmogorov 33] A.N. **Kolmogorov**. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Berlin: Springer, 1933.
- [Kruse et al. 91] R. **Kruse**, E. **Schwecke**, and J. **Heinsohn**. *Uncertainty And Vagueness in Knowledge Based Systems*. Heidelberg: Springer, 1991.

- [Nebel 90] B. **Nebel**. *Reasoning and Revision in Hybrid Representation Systems*. In: Lecture Notes in Artificial Intelligence LNAI, volume 422. Springer, 1990.
- [Profitlich 93] H.-J. **Profitlich**. *KRIS Version 2.0*, 1993.
- [Smolka et al. 89] M. **Schmidt-Schauss** and G. **Smolka**. *Attributive Concept Descriptions with Unions and Complements*. Technical report, IBM Wissenschaftliches Zentrum, Institut für Wissensbasierte Systeme, Stuttgart, 1989.

Index

- ALCP*, 19, 20, 37
 - Umsetzung, 31
- ALC*, 12, 43, 44
- Äquivalenz, 15
- Abhängigkeit
 - logische, 50
- Algorithmus
 - intelligenter, 36
- Alltagswissen, 19
- Axiom
 - terminologisches, 32
 - von Kolmogorov, 21
- Axiomensystem, 20
- Bayes, 24
 - Bayessches Netz, 19
 - Theorem von, 24
- Bedienungsanleitung, 37, 91–96
- Beispiel, 71
 - Barking cats, 71, 79
 - Birds, 71, 80
 - Birds 2, 81
 - Birds-2, 72
 - Burglary, 72, 83
 - Children, 72, 85
 - Nixon-Diamant, 12, 16, 17, 32, 53, 73
 - Songwriter, 72, 86
 - Sprinkler, 72, 88
- Beispielsitzung, 96–101
- Berechnungsverfahren, 27
 - vollständiges, 27
- Bereichsüberprüfung, 35
- Beschränkung
 - trianguläre, 28
- BOTTOM, 12, 45, 47
- Bruch, 34–36
- Clear, 95
- Constraint, 20, 26, 33, 36, 37, 57
 - CONS-AND, 57, 61–63
 - CONS-BAYES, 28, 31, 36, 57
 - CONS-LOG, 31, 36, 37, 61, 62, 64
 - CONS-NOT, 57, 61, 62
 - CONS-OR, 57, 61–63
 - CONS-PINGUIN, 31, 36, 57, 67
 - CONS-TRIAN, 28, 31, 36, 57
 - Implementierung, 31
- Datenstrukturen, 31, 45, 49
- Dempster-Shafer-Theorie, 19
- Detect TBox Knowledge, 95
- Detect-tbox-knowledge, 95
- Dezimalzahl, 34–36
- Disjunktheit, 15, 39
- Disjunktion, 29
- Domäne, 14, 25
- Domäne, 43, 44
 - Terminologie, 43
- Durchschnitt, 21
- Eingabe
 - Konvertierung, 35
- Einheitsintervall, 33–35
- Elementarereignis, 21
- Ereignis, 21, 25
- Ereignisraum, 21
- Existenzrestriktion, 13
- Extension, 17, 47
 - Extensionsfunktion, 13, 25, 26

- Funktion
 - andcheck, 65
 - andorgate, 65
 - cons-not-call, 65
 - determine-logic, 65
 - notcheck, 65
 - orcheck, 67
- Fuzzy-Logik, 19
- Genauigkeit, 35, 36
- Graphbasiertes Verfahren, 19
- Hashtabelle, 50, 51
 - *history*, 53
 - *logische-Abhängigkeiten*, 52
 - *p-Konditionierungen*, 51
 - *verbundene-Konzepte*, 45, 51
 - Element, 50
 - Schlüssel, 50
- Hierarchie, 11, 44
- History, 37, 41, 50, 95, 96
 - all ranges, 95
 - Anzeigen, 41
 - pairwise, 95
- Inferenz, 37
 - Algorithmus, 44
- Information
 - implizite, 43
- Inkohärenz, 15
- Inkonsistenz, 16, 20, 29, 45
 - Auflösung, 32
- Künstliche Intelligenz, 19
- Klassifizierung, 12, 14, 16, 43
- Kolmogorov, A.N., 20
- Kolmogorov-Axiom, 21
- Komplement, 21
- Konjunktion, 29
- Konsistenz, 27, 45
- Konzept, 11, 15
 - definiertes, 12
 - drittes, 49
 - leeres, 15, 35, 47, 68
 - primitives, 12
 - subsumiertes, 54
- Konzeptausdruck, 13
- Konzeptdefinition, 14
- Konzeptdisjunktion, 13
- Konzepte
 - disjunkte, 35
- Konzepthierarchie, 12
- Konzeptkonjunktion, 13, 29
- Konzeptnamen, 54
- Konzeptnegation, 13, 29
- Korrektheit, 27
- KRIS/RAT, 43, 44
 - Anfrage, 47
 - Schnittstelle, 43
- Laden, 91
- LISP, 35, 50
- Liste, 50, 53
 - *agenda*, 54
 - *konzepte*, 54
 - *leere-konzepte*, 55
- Load PKB, 92
- Logik
 - terminologische, 11
- Modell, 14, 26
 - heuristisches, 19
- Modify Range, 94
 - by slider, 94
 - Manual, 94
- Monotonie, 21
- Multiplikationsregel, 23
- Nachkommastelle, 35, 36
- Negation, 29
- Netz
 - bayessches, 24
- Nullkante, 35, 44, 58
- Oberbegriffsrelation, 12
- Oberfläche, 36, 92
- Ordnung
 - partielle, 12

- p-Konditionierung, 25
 - anzeigen, 40, 96
 - Behandlung, 40
 - einfügen, 40
 - eingeben, 95
 - interessante, 34
 - Konsistenz, 27
 - Modell, 26
 - nichttriviale, 34
 - verändern, 94
 - verändern, 69
- PBox, 26, 32, 33, 36
 - löschen, 39, 95
 - laden, 39, 92, 95
 - speichern, 39, 92, 95
 - Syntax, 31
- Pfadnamen
 - einstellen, 92
- Propagation, 94
- Propagierung, 36, 37, 57, 96
 - automatische, 70
 - starten, 41
- Propagierungsalgorithmus, 31, 50, 54, 57, 94
 - Implementierung, 57, 59
- Queue, 54
- Repräsentationssprache
 - terminologische, 12, 43
- Rolle, 12
- Save PKB, 92
- Schnittstelle, 31
- Semantik, 12, 25
- Show Ranges, 94
 - all, 94
 - all 0-ranges, 95
 - all subsumptions, 95
 - given by PBox, 94
 - given by TBox, 94
 - given ranges, 94
 - new, 94
 - pairwise, 95
- SLIDER, 36, 70
- Speicherplatz, 49
- Stack, 54
- Statistik, 19
- Subadditivität, 21
- Subsumtion, 12, 14, 35, 39, 44
- Subsumtionskante, 35, 36, 44, 58
- Subsumtionsrelation, 12, 15
- Suchvorgänge, 50
- Syntax, 25
- Systemarchitektur
 - abstrakte, 37
- Taxonomie, 43
- TBox, 32, 35, 43, 54
 - laden, 37, 92, 95
 - Syntax, 31
- TBox-Wissen
 - entdecken, 39
- Terminologie, 14–16, 27
 - Übernahme, 43
- TOP, 12
- Trace, 41, 71, 74–78, 94, 96
 - in file, 94
 - off, 94
 - on, 94
 - umschalten, 41
- Triangulärer Fall, 36, 49
 - Abarbeitung, 61, 68
 - Auswahl, 58
 - interessanter, 58
- Unabhängigkeit, 25
- Unsicherheit, 17, 19
- Unsicherheitsmodell, 19
- Unterbegriffsrelation, 12
- Unwissenheit, 58
- Variable
 - globale, 35, 50, 55
 - *agenda*, 41
 - *device*, 55
 - *konzepte*, 44
 - *nachkommastellen*, 35, 55

nenneraus, 55

tracefile, 56

Wahrscheinlichkeit

a-posteriori, 24

a-priori, 24

bedingte, 22

konditionale, 22

totale, 24

Wahrscheinlichkeitsfunktion, 21

Wahrscheinlichkeitsgrenzen, 29

Wahrscheinlichkeitsintervalle, 19

Wahrscheinlichkeitsmaß, 21

Wahrscheinlichkeitsraum, 21

Wahrscheinlichkeitstheorie, 11, 17, 19,
20

Wertrestriktion, 13

Wissen, 11

nichtkategorisches, 17

terminologisches, 11, 12, 35

unsicheres, 17, 32

Wissensbasis, 20, 31, 36, 43

Behandlung, 37

probabilistische, 36

terminologische, 35

Wissensrepräsentation, 11

Wissensrepräsentationssprache, 12

Zugehörigkeit, 17

Zugriffsgeschwindigkeit, 49