Diplomarbeit

Nichtmonotone Aspekte des fallbasierten Schließens und deren Fundierung in Default-Logik mit Prioritäten

Michael Mehl

15. Juni 1992

Betreuung:

Prof. Dr. Michael M. Richter

AG Künstliche Intelligenz – Expertensysteme

Universität Kaiserslautern

Erklärung

Hiermit erkläre ich, daß ich die Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

Kaiserslautern, den 15. Juni 1992

(Michael Mehl)

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Grundidee der vorliegenden Arbeit ist es, Gemeinsamkeiten der beiden großen Forschungsgebiete der künstlichen Intelligenz, *nichtmonotone Logiken* und *fallbasiertes Schließen*, zu untersuchen bzw. zu finden.

Dazu ist es zum einen nötig die vorhandenen Grundlagen beider Gebiete aufzuarbeiten. Zum anderen müssen diese Grundlagen zusammengebracht werden.

Das Gebiet des fallbasierten Schließens ist ein sehr aktuelles Forschungsgebiet der künstlichen Intelligenz und umfaßt die Entwicklung von Systemen, die in der Lage sind aus Fällen zu lernen. Als Synonym für fallbasiertes Schließen wird der Begriff Case–Based Reasoning oder abgekürzt CBR verwendet.

Die Bedeutung von Case–Based Reasoning wird in [Minsky91] sehr deutlich ausgedrückt:

> In the last 10 years there has been a great deal of research on something called Case–Based Reasoning and I predict that 20 years from now that will be the most important application

Man kann das fallbasierte Schließen als *maschinelles Lernverfahren* ansehen. Case–Based Reasoning tangiert jedoch in erheblichen Maßen auch die Gebiete *kognitive Psychologie, Common Sense Reasoning und Wissensakquisition.*

Der Begriff des Lernens ist nicht exakt zu definieren. Wir wollen hier unter maschinellem Lernen eine Zustandsänderung des Systems verstehen, die in irgendeiner Art und Weise einer Verbesserung darstellt. Das System könnte zum Beispiel schneller oder benutzerfreundlicher werden. Es könnte auch in der Lage sein eine umfassendere Problemstellung zu bearbeiten. Eine ausführliche Diskussion des Lernbegriffes bietet [RichterWendel91].

Case–Based Reasoning benutzt Fälle zur Repräsentation elementarer Wisseneinheiten. Fälle kann man als Tupel aus Problembeschreibung und Problemlösung ansehen. Sie werden in einer Speicherstruktur, genannt Fallbasis, abgelegt. Die Lösung eines Problems geschieht nun durch die Suche eines gespeicherten Falles. Dieser sollte so beschaffen sein, daß sich seine Lösung auf das Problem übertragen läßt. Die Suche wird im allgemeinen durch ein Ähnlichkeitsmaß gesteuert: Es wird der Fall gesucht, dessen Problembeschreibung ähnlich zum neuen Problem ist. Die Bestimmung eines übertragbaren Falles nennt man *Retrieval.*

In der Forschung zur nichtmonotonen Logik geht es um die Entwicklung von formalen Modellen, die es erlauben mit inkonsistentem und unvollständigem Wissen umzugehen. Insbesondere wird untersucht, wie Inferenzmechanismen aussehen könnten, d.h. wie man neues Wissen ableiten kann. Der zentrale Aspekt dabei ist, daß die Inferenzen nichtmonoton sind. Als nichtmonoton bezeichnet man einen Inferenzoperator, der durch zusätzliches Wissen u.U. weniger mögliche Ableitungen zuläßt.

Der wichtigste Punkt, der unser Interesse an dieser Arbeit begründet, ist die mangelnde logische Fundierung des Case–Based Reasoning. Es wurden sehr viele Algorithmen und Programme entwickelt, die fallbasiertes Schließen in bestimmten Fällen durchführen. Es ist nicht immer klar, was diese Programme wirklich leisten und wo ihre Grenzen sind. Aus diesem Grund ist es wünschenswert eine logische Grundlage zu finden, um das Verhalten von CBR–Systemen genauer untersuchen zu können.

Da die Themenbereiche nichtmonotone Logik und fallbasiertes Schließen zu umfangreich sind, können wir nur Teilbereiche behandeln.

Als nichtmonotone Logik legen wir die Default–Logik zugrunde. Die Default–Logik basiert auf einem zweigeteilten Wissen: einerseits einem sicheren und verläßlichen Faktenwissen, andererseits auf einem unsicheren und möglicherweise fehlerhaften Defaultwissen. Defaults sind im Prinzip Regeln mit einem zusätzlichen Bedingungsteil, der Annahmen einführt, die u.U. auch nach der Verwendung der Regel revidiert werden müssen. Das kann zu einer Rücknahme der Regel und aller daraus abgeleiteten Formeln führen.

Der Inferenzmechanismus der Default–Logik verwendet zunächst das Faktenwissen. Zusätzlich benutzt er die Defaults, die mit den Fakten und untereinander verträglich sind. Verträglich bedeutet, daß die Annahmen der Defaults zu keinen Inkonsistenzen führen. Um Inkonsistenzen gezielt auflösen zu können, werden wir ein zusätzliches Sprachelement, nämlich Prioritäten zwischen Defaults, einführen. Diese Prioritäten legen im Prinzip fest, welcher Default zu bevorzugen ist. Es wird damit eine Gewichtung der Defaults durchgeführt.

Beim fallbasierten Schließen beschränken wir uns auf die Untersuchung des Retrievalprozesses und der Speicherung der Fälle. Ziel der Arbeit ist die Bereitstellung einer logischen Grundlage zur Repräsentation des Fallspeichers und die Übertragung des Retrieval in einen Inferenzprozeß einer nichtmonotonen Default–Logik.

Die erste Aufgabe ist die Festlegung einer formalen Definition eines Fallspeichers und die Spezifikation des Retrievalprozesses.

Der zweite Schritt besteht in der Transformation des Fallspeichers in eine Default–Theorie. Fälle werden als Fakten und die Struktur des Fallspeichers als Defaults kodiert. Die Transformation ist so angelegt, daß der Retrievalprozeß des Case–Based Reasoning sich im Inferenzprozeß der Default–Logik wiederfindet.

Die Gliederung der Arbeit besteht in einer Aufteilung in zwei Teile: den Grundlagenteil und den Anwendungsteil.

Im Grundlagenteil findet eine Aufarbeitung der relevanten Literatur zur Default–Logik und zum Case–Based Reasoning statt. Es werden die Nichtmonotonie (Kapitel 2), die Default–Logik (Kapitel 3) und die Prioritäten (Kapitel 4) ausführlich eingeführt. Außerdem wird das fallbasierte Schließen (Kapitel 5) und der Aufbau des Fallspeichers (Kapitel 6) mit den zentralen Aspekten dargestellt.

Im zweiten Teil muß zunächst das formale Modell des Fallspeichers (Kapitel 7) erarbeitet werden, um darauf aufbauend eine Default–Theorie mit Prioritäten (Kapitel 8) zu entwickeln. Für diesen Teil bietet die Literatur bisher keine fundierten Ansätze.

Die abschließenden Kapitel beinhalten Ideen für mögliche Erweiterungen (Kapitel 9), den Vergleich mit verwandten Arbeiten (Kapitel 10) und eine Zusammenfassung (Kapitel 11).

Teil I

Grundlagen

Kapitel 2

Nonmonotonic Reasoning

2.1 Motivation

Das vorliegende Kapitel dient in erster Linie der Motivation der Nichtmonotonie und ist keine notwendige Grundlage für die nachfolgenden Kapitel. Es kann übersprungen werden, wenn man bereits von der Notwendigkeit nichtmonotoner Schlußweisen überzeugt ist.

Zur Einführung in das Thema „Nichtmonotonie“ werden zunächst die Gründe, die zur Entwicklung nichtmonotoner Logiken geführt haben, dargestellt. Danach wird erläutert, was eigentlich charakteristisch für nichtmonotone Logiken ist. Einige typische Beispiele sollen dann die positiven Eigenschaften und die auftretenden Probleme veranschaulichen.

Beim Versuch Weltwissen, insbesondere Common–Sense–Wissen, zu modellieren, stellt man schnell fest, daß man an Grenzen stößt, die mit klassischer Prädikatenlogik nicht mehr effektiv und adäquat zu überwinden sind. Insbesondere ist es im allgemeinen unmöglich, vollständiges Wissen zu repräsentieren, zum Teil wegen der Komplexität der Welt und zum Teil weil die genaue Einsicht und das Verständnis der Welt fehlt. Trotzdem ist es oft nötig (und der Mensch ist dazu auch in der Lage) mit unvollständigem Wissen Inferenzen zu ziehen, z.B. um Entscheidungen zu treffen oder Aktionen zu planen. Dazu werden bestimmte Annahmen getroffen, aufgrund derer dann Schlüsse gezogen werden können. Das Problem solcher Annahmen ist die Möglichkeit, daß sie sich als falsch herausstellen können und damit inklusive ihrer Konsequenzen revidiert werden müssen.

Zunächst glaubte man, durch das „Springen zu Folgerungen“ [McCarthy80], d.h. das Einführen von Annahmen, die Komplexität klassischer Beweiser erheblich reduzieren zu können, da auch der Mensch relativ schnell falsche Annahmen als solche erkennen und revidieren kann. Leider sind die bisher eingeführten nichtmonotonen Logiken, außer in Spezialfällen, unentscheidbar. Der Grund dafür ist die Notwendigkeit von Konsistenzüberprüfungen, die in der zugrundeliegenden Prädikatenlogik 1. Stufe nicht entscheidbar sind.

Eine zweite wichtige Motivation für die Einführung nichtmonotoner Logiken ist das Problem der Inkonsistenz großer Datenmengen. Die klassische Prädikatenlogik erlaubt aus Inkonsistenzen alles abzuleiten, obwohl es unter Umständen sinnvolle Kriterien für einen eingeschränkten Ableitungsbegriff gibt. Nichtmonotone Logiken erlauben Inkonsistenzen, mit denen trotzdem „vernünftige“ Ableitungen durchgeführt werden können.

Vor den grundlegenden Arbeiten zur Nichtmonotonie (1980) gab es bereits Systeme, die nichtmonotone Schlußweisen durchführten, deren Semantik jedoch informal war und die deshalb kein volles Verständnis ihres Verhaltens insbesondere bei komplexen Problemen ermöglichten [Etherington87]. Die nichtmonotonen Logiken sollten hier einen tieferen Einblick erlauben.

Erstaunlich ist die Diskrepanz zwischen der Effizienz einiger Implementierungen, z.B. von semantischen Netzen, die auch nichtmonotones Verhalten zeigen, und der Nichtberechenbarkeit von nichtmonotonen Logiken. In diesem Spannungsfeld zwischen schneller Implementierung und gutem theoretischen Verständnis ist auch die vorliegende Arbeit zu sehen, da versucht wird eine genauere Semantik des Case–Based Reasoning zu finden.

Weshalb benutzt man zur Behandlung von unvollständigem und inkonsistentem Wissen nicht die gut verstandene Prädikatenlogik, sondern eine neue „nichtmonotone" Logik? Der Grund dafür ist die Monotonie–Eigenschaft der Prädikatenlogik 1. Stufe.

Definition 2.1.1 (Monotonie)
Sei $\mathcal{L}$ eine prädikatenlogische Sprache; $\vdash$ ein Ableitungsoperator.

$\vdash$ *heißt* monoton *gdw* $\forall A, B \subseteq \mathcal{L} \;.\; \forall w \in \mathcal{L} \;.\; A \vdash w \leadsto A \cup B \vdash w$

Falls die Bedingung nicht erfüllt ist, nennt man den Operator $\vdash$ nichtmonoton*.*

Monotonie bedeutet, daß durch zusätzliches Wissen (B) bisher mögliche Folgerungen (w) weiterhin gelten. Das Schließen aufgrund von Annahmen hat diese Eigenschaft nicht mehr, da zusätzliche Informationen Annahmen ungültig machen können und damit auch Folgerungen aus dieser Annahme im allgemeinen nicht mehr gelten.

2.2 Beispiele

Die Literatur zu Nichtmonotonie enthält einige wichtige Beispiele und Anwendungsgebiete, die die Grundprobleme und die Notwendigkeit nichtmonotoner Logiken verdeutlichen sollen.

- **Ausnahmen von Regeln: Tweety**
  Das prominenteste Beispiel ist der Vogel Tweety und die Frage, ob Tweety fliegen kann [1].

  Es gilt: Alle Vögel können fliegen, außer Pinguinen, dem Vogel Strauß, usw. Das Problem hier sind die Ausnahmen: es ist im allgemeinen nicht möglich alle Fälle, in denen Vögel nicht fliegen können aufzuzählen, z.B. könnte ein pathologischer Fall, an den man bei der Spezifikation von *alle Vögel fliegen* nicht gedacht hat, ein gebrochener Flügel sein.

  Sei Tweety nun ein Vogel, von dem wir sonst nichts wissen, dann kann man ableiten, daß Tweety fliegt. Diese Ableitung ist keine Folgerung im Sinn der klassischen Logik, sondern sie ist mehr eine Annahme (engl. belief), die durch neues Wissen revidiert werden kann. Erfährt man z.B., daß Tweety einen Unfall hatte und sein Flügel gebrochen ist, dann ist die Annahme „Tweety kann fliegen" nicht mehr gültig und das Wissen muß revidiert werden.

  Das zugrundeliegende Prinzip in diesem Beispiel ist die Modellierung von „meist" oder „fast alle". Nach [Reiter87] läßt sich jedes Beispiel plausiblen Schließens darstellen als Muster von

  „Normalerweise gilt A" bzw. „Typischerweise ist A der Fall"

  Für das Tweety–Beispiel wäre die Aussage „Normalerweise fliegen Vögel" eine vernünftige Instanziierung dieses Musters.

- **Closed World Assumption (CWA): Flugverbindungen**
  In Kurzform bedeutet die Closed World Assumption: alle positiven Fakten über die Welt sind bekannt, kann eine Aussage nicht positiv bestätigt werden, dann gilt ihr Gegenteil. Fragt

[1] Angeblich **muß** Tweety in jeder Arbeit zu Nichtmonotonie auftauchen [KautzSelman89]

man zum Beispiel bei der Flugauskunft: „Gibt es eine Direktverbindung von Stuttgart nach Bonn?“ und in den Flugplänen findet sich kein entsprechender Eintrag, dann folgt gemäß CWA: es gibt keine Direktverbindung. Auch hier liegt Nichtmonotonie vor, da durch zusätzliche positive Fakten bisherige Ableitungen u.U. revidiert werden müssen.

- **Frame–Problem: Situationskalkül**
  In Beschreibungen einer sich ändernden Welt tritt das Problem auf, zu repräsentieren was sich bei Aktionen ändert und was invariant bleibt (*Frame–Problem*)[McCarthyHayes69]. Zum Beispiel ändert das Anmalen eines Objektes zwar die Farbe, aber nicht die Position, die Grösse, den Ort, usw. von allen Objekten. Insbesondere im Situationskalkül ([McCarthyHayes69]), der eine Situationsvariable in den Prädikaten mitführt, bräuchte man, um den obigen Sachverhalt auszudrücken, sehr viele Axiome der Form:

  $\forall x, y, z, s, c \,.\, location(x, y, s) \Rightarrow location(x, y, paint(z, c, s))$
  mit s ist Zustandsvariable, c Farbe, x und z sind Objekte und y eine Position.

  Solche Axiome heißen *Frame–Axiome*. Sie bestehen aus zwei Teilen: einer Aktion (*paint*) und einem Prädikat (*location*), das sich durch die Aktion nicht ändert. Da die Menge der benötigten Frame–Axiome riesig ist, nämlich für fast jedes Aktion – Prädikat Paar ein Axiom, kann man das Frame–Problem nicht sinnvoll durch die Angabe aller Frame–Axiome lösen. Durch nichtmonotone Logiken hoffte man, das Frame–Problem in den Griff zu bekommen, in dem man

  „Prädikate ändern sich nicht als Folge von Aktionen“

  als revidierbare Annahme vorgibt. Leider hat sich herausgestellt, daß diese einfache Annahme nicht die gewünschten Resultate liefert.

- **Zeitliches Schließen: Das Yale–Shooting–Problem**
  [HanksMcDermott86, HanksMcDermott87] haben am Beispiel der Modellierung des sogenannten *Yale–Shooting–Problems* gezeigt, daß unerwartete Lösungen resultieren. Das Yale–Shooting–Problem beinhaltet: eine Pistole mit den Zuständen geladen oder ungeladen und einen Menschen, der lebendig oder tot sein kann. Es gibt drei mögliche Aktionen: (1) das Laden der Pistole, (2) eine Warte–Aktion und (3) das Schießen mit der Pistole. Die Aktionen haben die intuitiven Zustandsänderungen zur Folge: z.B. wenn die Pistole geladen ist und die Aktion „Schießen“ wird ausgeführt, dann ist der Mensch anschließend tot und die Pistole ist entladen. Werden die drei Aktionen nun nacheinander ausgeführt, dann liefern die nichtmonotonen Modelle zwei mögliche Zustände, nämlich das intuitiv richtige: der Mensch ist tot und das unerwartete: der Mensch lebt, da sich die Pistole in der Wartezeit irgendwie „magisch“ entladen hat. Es gibt inzwischen viele Lösungen für dieses Problem, z.B. von [Kautz86, Lifschitz86, Shoham86]. Doch sie bieten keine allgemeine Lösung für zeitliches Schließen mit nichtmonotonen Logiken.

- **Qualifikationsproblem**
  Das Qualifikationsproblem ist eng mit dem Frame–Problem verwandt, denn beide betreffen die Spezifikation von Regeln. Während das Frame–Problem die Nach–Bedingung von Regeln als Kernproblem hat, behandelt das Qualifikationsproblem die Voraussetzungen, die zur Anwendung von Regeln (oder Aktionen) nötig sind. Es ist im allgemeinen unmöglich alle Voraussetzungen für eine korrekte Regelanwendung anzugeben. [Brewka91] bringt das Beispiel des Anlassens eines Autos. Das Umdrehen des Zündschlüssels bewirkt das Anspringen des Autos, wenn gewisse Voraussetzungen erfüllt sind: der Tank darf nicht leer sein, die Zündung muß in Ordnung sein, usw. Durch eine Annahme der Form

  „Regeln sind im allgemeinen anwendbar“

könnte man das erwünschte Verhalten erzielen: solange keine Information über die Nichtanwendbarkeit vorliegt, ist die Regel korrekt.

- **Frames**
  In Frames [Minsky75] werden Defaultwerte für Slots verwendet. Sie dienen dazu Werte zu liefern, wenn keine weiteren Informationen vorliegen. Solche Default–Werte entsprechen nichtmonotonen Annahmen, die solange benutzt werden, bis sie sich durch neue Erkenntnisse als falsch herausstellen. Die Frage der Semantik von Slots und ihren Aspekten war eine Motivation für die Entwicklung nichtmonotoner Logiken.

- **Modellbasierte Diagnose (engl. diagnosis from first principle)**
  Bei der modellbasierten Diagnose ist die Grundlage die Beschreibung, z.B. eines Gerätes und seiner Komponenten, durch das gewünschte, d.h. korrekte, Ein-/Ausgabeverhalten. Mit Hilfe des tatsächlich beobachteten Verhaltens soll der Diagnoseprozeß die Fehlerursache, also die fehlerhafte Komponente, ermitteln. Man kann die modellbasierte Diagnose so in die nichtmonotone Logik einordnen, daß man das korrekte Ein-/Ausgabeverhalten als Annahme sieht, die durch das beobachtete Verhalten u.U. revidiert werden muß. Die Diagnose besteht im Finden eines Modells in der Logik, in dem die ungültigen Annahmen den fehlerhaften Komponenten entsprechen [Reiter87b]. [Junker91] gibt eine Implementierung für die modellbasierte Diagnose mit Hilfe einer nichtmonotonen Logik an.

2.3 Verschiedene Ansätze für nichtmonotone Logiken

Um eine Grundlage für die Bewertung und Auswahl nichtmonotoner Ansätze zu legen, werden die wichtigsten nichtmonotonen Logiken vorgestellt.

Einer der ersten und prominentesten Vertreter ist Circumscription [McCarthy80, Lifschitz85, McCarthy86, Lifschitz86b]. Das Grundprinzip von Circumscription ist basierend auf der Prädikatenlogik minimale Modelle für eine Formelmenge oder ein Prädikat zu bestimmen. Das entspricht einer Art von Closed–World–Assumption. Die Berechnung der Circumscription eines Prädikates geschieht mit Hilfe einer prädikatenlogischen Formel 2. Stufe. Diese Formel entspricht im mathematischem Sinne einem Induktionsaxiom. Mit diesem Circumscriptionaxiom wird eine implizite Definition des Prädikates festgelegt. Der größte Vorteil von Circumscription ist die klare Semantik die man erreicht, da es sich um einen modelltheoretischen Ansatz handelt.

Der zweite wichtige Vertreter nichtmonotoner Schlußweisen ist ein beweistheoretischer Ansatz: die Default–Logik [Reiter80, Poole88]. Die Prädikatenlogik wird in der Default–Logik um spezielle Regeln, sogenannte Defaults, erweitert, die immer dann anwendbar sind, wenn erstens die Voraussetzungen erfüllt sind und zweitens die sogenannten Annahmen nicht widerlegt werden können. Diese Formulierung mit Regeln ist sehr anschaulich, birgt aber die Gefahr in sich, daß die Interaktion von Regeln nicht mehr überblickt werden kann. Die Default–Logik wird in Kapitel 3 ausführlich behandelt.

Weitere Ansätze basierend auf der Modallogik sind die nichtmonotone Logik I [McDermottDoyle80] und die autoepistemische Logik [Moore84, Moore85]. Es werden dabei zusätzliche Sprachelemente (Modaloperatoren) in die Prädikatenlogik eingeführt, die Aussagen über Glauben und Konsistenz von Formeln erlauben.

Die bisher erwähnten Logiken bauen alle auf der Prädikatenlogik auf und es liegt ihnen die These zugrunde, daß symbolische Ausdrucksweisen zur Modellierung der Welt genügen. Daneben gibt es jedoch eine Reihe von numerischen Ansätzen, in denen zum Beispiel das probabilistische Schließen oder eine Fuzzy–Logik im Vordergrund steht. Numerische Ansätze werden im Rahmen dieser

Arbeit nicht behandelt. Literatur dazu findet sich beispielsweise in [Pearl88].

Es wurden hier nur die wichtigsten symbolischen nichtmonotonen Ansätze beschrieben. Neben diesen Arbeiten gibt es noch eine Vielzahl von Arbeiten, die andere Arten der Modellierung von Nichtmonotonie behandeln. Exemplarisch seien hier einige neuere Veröffentlichungen genannt: P– und J–Default–Logik [DelgrandeJackson91], Disjunctive Default–Logik [GelfondLifschitz+91], Argument Systems [LinShoham89].

Kapitel 3

Default–Logik

3.1 Einführung

Es wird ein Formalismus für die Behandlung von Nichtmonotonie eingeführt, der in Kapitel 4 weiter verfeinert wird. Grundlage des Formalismus bildet die Prädikatenlogik erster Stufe, die um zusätzliche syntaktische Elemente (Defaults) erweitert wird. Parallel zu dieser Erweiterung muß auch der Gültigkeitsbegriff angepaßt werden. Dies geschieht zum einen durch den Begriff der Extension, die im Prinzip die gültigen Formeln umfaßt, und zum anderen durch die Definition eines Beweisbegriffes. Zusammen bilden diese Elemente die Fundierung der Default–Logik.

Die Grundlagen der Prädikatenlogik werden im folgenden vorausgesetzt (siehe z.B. [Richter89]). Einige wichtige Sprechweisen aus der Logik sollen hier jedoch kurz zusammengefaßt werden, um eine einheitliche Benennung zu haben.

prädikatenlogische Sprache: Als theoretische Ausgangsbasis wird eine *prädikatenlogische Sprache* $\mathcal{L}$ zugrundegelegt, die alle *Formeln* (engl. well formed formula, wff) enthält. $\mathcal{L}$ beinhaltet eine Menge von *Funktionssymbolen*, eine Menge von *Prädikatssymbolen*, eine Menge von *Variablen* und eine Menge von *logischen Zeichen*. Aus den Funktionssymbolen und Variablen werden die *Terme* konstruiert. Die Terme ohne Variablen bezeichnen wir als *Grundterme*. Die Prädikate spezifizieren Beziehungen zwischen Objekten.

geschlossene Formel: $w \in \mathcal{L}$ heißt *geschlossen* gdw w keine *freien Variablen* enthält, wobei wir nichtquantifizierte Variablen als frei bezeichen. $\mathcal{L}_0$ sei die Menge aller geschlossenen Formeln.

beweisbar: Sei $S \subseteq \mathcal{L}, w \in \mathcal{L}$ und $\vdash$ ein Ableitungsoperator eines korrekten und vollständigen Kalküls. $S \vdash w$ gdw w ist in der Prädikatenlogik 1. Stufe *beweisbar* aus S. Wir verwenden als Synonym zu beweisbar die Bezeichnung w ist *herleitbar* aus S.

konsistent: Sei $S \subseteq \mathcal{L}, w \in \mathcal{L}$. S heißt *konsistent*, wenn keine *widersprüchliche Formel* aus S herleitbar ist. w heißt konsistent mit S, wenn $S \cup \{w\}$ konsistent oder S bereits inkonsistent ist.

Theorie: Sei $S \subseteq \mathcal{L}$. $Th(S) := \{w \in \mathcal{L} \mid S \vdash w \text{ und } w \text{ ist geschlossen}\}$ heißt *Theorie* von S.

3.2 Definitionen

[Reiter80] benutzt als Ansatz für seine Default–Logik eine Erweiterung der Prädikatenlogik um spezielle Regeln (Defaults). Defaults stellen eine mögliche Form zur Repräsentation unsicheren Wissens dar. Als wesentliche Erweiterung zu gewöhnlichen Regeln, enthalten Defaults Annahmen. Diese Annahmen sind im Prinzip Voraussetzungen, die jedoch nicht herleitbar sein müssen. Annahmen müssen nur einer Konsistenzbedingung genügen, die garantiert, daß die Negation der Annahme sicher nicht herleitbar ist. Defaults bestehen aus drei Teilen: den notwendigen Voraussetzungen, den benötigten Annahmen und den Folgerungen. Der Default zur Aussage „Vögel fliegen normalerweise" hat als Voraussetzung „$Vogel(x)$". „$Fliegt(x)$" ist die Annahme und gleichzeitig auch die Folgerung. Der Default bedeutet: wenn wir wissen, daß es sich um einen Vogel handelt und wenn wir nicht wissen, daß er nicht fliegen kann, dann kann er fliegen.

Definition 3.2.1 (Default)
Seien $\alpha(x), \beta(x), \omega(x) \in \mathcal{L}$; $x = x_1, ..., x_n$ alle freien Variablen in $\alpha(x), \beta(x), \omega(x)$.

$\frac{\alpha(x):\beta(x)}{\omega(x)}$ *heißt* Default.

$\alpha(x)$ *heißt* Voraussetzung *(engl. prerequisite)*; $\beta(x)$ *heißt* Annahme *(engl. justification) und* $\omega(x)$ *heißt* Folgerung *(engl. consequent).*

Ein Default $\delta(x) = \frac{\alpha(x):\beta(x)}{\omega(x)}$ *heißt* geschlossen *gdw* $\alpha(x), \beta(x)$ *und* $\omega(x)$ *keine freien Variablen enthalten.*

Ein Default $\delta(x) = \frac{\alpha(x):\beta(x)}{\omega(x)}$ *heißt* normal *gdw* $\beta(x) \equiv \omega(x)$.

Ein Default $\delta(x) = \frac{\alpha(x):\beta(x)}{\omega(x)}$ *heißt* semi–normal *gdw* $\beta(x) \equiv \omega(x) \wedge \gamma(x)$.

Ist $\alpha(x)$ *eine Tautologie, also allgemeingültig, dann läßt man die Voraussetzung weg und schreibt:*

$$\frac{:\beta(x)}{\omega(x)}$$

$\alpha(x)$ entspricht einem monotonen und $\beta(x)$ einem nichtmonotonen Anteil, d.h. $\alpha(x)$ muß gelten und $\beta(x)$ ist nur eine Annahme.

Eine Default–Theorie besteht aus einem Tupel: einer Menge von Fakten, die man sicher weiß, und einer Menge von Defaults, die das unsichere Wissen beschreiben.

Definition 3.2.2 (Default–Theorie)
Sei D eine Menge von Defaults, $W \subseteq \mathcal{L}$. Das Tupel $\Delta = (D, W)$ heißt Default–Theorie *(DT).*

Eine Default–Theorie $\Delta = (D, W)$ heißt geschlossene Default–Theorie *gdw alle Defaults in D geschlossen sind. Entsprechend spricht man auch von normalen Default–Theorien und semi–normalen Default–Theorien.*

Im folgenden werden zunächst nur geschlossene Default–Theorien betrachtet.

Die intuitive Vorstellung einer Default–Theorie $\Delta = (D, W)$ ist: W enthält eine Menge von Fakten (gültige Formeln). Die Theorie von W ist im allgemeinen nicht vollständig. Vollständigkeit bedeutet hier: $\forall F \in \mathcal{L}_0$ gilt $F \in Th(W)$ oder $\neg F \in Th(W)$, d.h. alle geschlossenen Formeln sind aus W herleitbar oder nicht. In unvollständigen Theorien gibt es also Formeln, von denen man nicht weiß, ob sie gültig sind oder nicht. Um das fehlende Wissen zu ergänzen, benutzt man die Defaults. Sie erweitern das Wissen auf die sogenannte Extension E, die ebenfalls nicht vollständig sein muß. Die Extension ist eine Theorie, die zum einen das sichere Wissen umfaßt. Zum anderen enthält sie die Folgerungen der Defaults, die anwendbar sind. Als *anwendbar* bezeichnen wir einen Default dann, wenn seine Voraussetzung herleitbar und seine Annahme konsistent ist.

Eine Extension einer geschlossenen Default–Theorie muß die folgenden Eigenschaften erfüllen:

(E1) $W \subseteq E$, d.h. die sicheren Formeln sind in der Extension enthalten.

(E2) $Th(E) = E$, d.h. die Extension ist abgeschlossen und enthält alle aus ihr ableitbaren Formeln.

(E3) $\forall \delta = \frac{\alpha:\beta}{\omega} \in D$ aus $\alpha \in E$ und $\beta \notin E$ folgt $\omega \in E$, d.h. wenn ein Default anwendbar ist, wird er auch angewendet.

(E4) E ist minimal bezüglich Mengeninklusion.

Eine Folgerung aus den Eigenschaften (E1) und (E2) ist: $Th(W) \in E$.

Aus den Eigenschaften läßt sich die folgende Definition einer Extension als Fixpunkt eines Operators Γ motivieren. Gegeben sei eine Menge S von Formeln, die das aktuelle Wissen umfaßt. Aus dieser Menge berechnet man alle Folgerungen mit Hilfe eines Operators Γ. Sind diese Folgerungen bereits in S, dann ist S eine Extension.

Definition 3.2.3 (Extension)
Sei $\Delta = (D, W)$ eine geschlossene Default–Theorie. Definiere $\Gamma : \mathcal{L}_0 \rightarrow \mathcal{L}_0$ wie folgt:
$\forall S \subseteq \mathcal{L}_0$. $\Gamma(S)$ *sei die minimale Menge, die folgende Bedingungen erfüllt:*

(D1) $W \subseteq \Gamma(S)$

(D2) $Th(\Gamma(S)) = \Gamma(S)$

(D3) $\forall \delta = \frac{\alpha:\beta}{\omega} \in D$, $\alpha \in \Gamma(S)$ *und* $\neg\beta \notin S$ [1], *dann ist* $w \in \Gamma(S)$.

$E \subseteq \mathcal{L}_0$ *heißt* Extension *gdw* $\Gamma(E) = E$, *d.h.* E *ist Fixpunkt von* Γ

Man kann sich weitere Definitionen für Extensionen überlegen. Wichtig dabei ist, daß diese die oben genannten Eigenschaften (E1) – (E4) erfüllen.

Im Unterschied zu Reiter benutze ich für Defaults eine leicht variierte Schreibweise. Reiter schreibt Defaults in der Form:

$\frac{\alpha:M\beta_1,\ldots,\beta_n}{\omega}$

Der Unterschied zu Reiter ist:

- Der Operator M vor der Annahme entfällt, damit die Syntax einfacher wird.
- Statt einer Liste von Annahmen gibt es nur eine Annahme, da es bisher keine Anwendungen gibt die unbedingt eine Liste von Annahmen benötigen und die meisten Probleme sich äquivalent als Konjunktion ausdrücken lassen.

  Ein Beispiel das unterschiedliche Extensionen ergibt ist das folgende:

  Die Extension von $(\{\frac{:A,B}{C}\}; \{\neg A \vee \neg B\})$ enthält C

  Die Extension von $(\{\frac{:A \wedge B}{C}\}; \{\neg A \vee \neg B\})$ enthält C jedoch nicht.

[1] Beachte: S statt $\Gamma(S)$, damit die Wohldefiniertheit von Γ garantiert werden kann

3.3 Konstruktionsansatz für Extensionen

Ein Ziel ist nun, Extensionen rekursiv zu berechnen. Einen Ansatz dazu bietet der folgende Satz, der es unter Umständen erlaubt, aus W und der sukzessiven Anwendung von Defaults eine Extension zu generieren. Das Problem ist, daß bei der Konstruktion schon die entgültige Extension bekannt sein muß.

Satz 3.3.1
Sei $E \subseteq \mathcal{L}_0, \Delta = (D, W)$ eine geschlossene Default–Theorie. Definiere eine Folge $E_0, E_1, ... \subseteq \mathcal{L}_0$ mit

(1) $E_0 := W$

(2) $E_{i+1} := Th(E_i) \cup \{w \mid \exists \delta = \frac{\alpha:\beta}{\omega} \in D \textit{ mit } \alpha \in E_i \textit{ und } \neg\beta \notin E\} \textit{ für } i > 0$

E ist Extension von Δ gdw $E = \bigcup\limits_{i=0}^{\infty} E_i$.

Beweisskizze: Der Beweis des Satzes wird durch Induktion über die Teilmengen E_i durchgeführt. Zum ersten zeigt man, daß alle E_i in E sind, wenn E eine Extension ist, d.h. $\bigcup\limits_{i=0}^{\infty} E_i \subseteq E$. Der Fall $E \subseteq \bigcup\limits_{i=0}^{\infty} E_i$ ist nach Konstruktion klar. Zum zweiten muß man mit Induktion zeigen, daß alle E_i in $\Gamma(E)$ enthalten sind, d.h. $E = \bigcup\limits_{i=0}^{\infty} E_i \subseteq \Gamma(E)$. Der umgekehrte Fall $\Gamma(E) \subseteq \bigcup\limits_{i=0}^{\infty} E_i$ gilt wieder gemäß Konstruktion der E_i. $\square$

Dieser Satz bietet kein konstruktives Verfahren zur Berechnung von Extensionen, da in der Definition von E_{i+1} bereits auf die gesuchte Extension E Bezug genommen wird. Jedoch liefert er ein intuitives Verfahren für die Bestimmung von Extensionen. Man beginnt mit der Menge W und generiert mit Hilfe der Defaults solange neue Formeln, die die Konsistenz erhalten, bis kein Default mehr anwendbar ist. Damit hat man u.U. eine Extension konstruiert; nämlich dann, wenn die Annahmen aller benutzten Defaults auch mit dieser Extension noch konsistent sind. Man muß also nachträglich oder bei jedem neu hinzugenommenen Default alle benutzten Annahmen nochmals überprüfen. Bei einer festgestellten Inkonsistenz hat man keine gültige Extension und muß, z.B. durch Ändern der Reihenfolge der Default–Auswahl, versuchen eine andere Formelmenge zu finden.

Die Bestimmung der Extension soll an einem Beispiel (nach [Sandewall72]) erläutert werden.

Beispiel 3.3.1
Es seien die drei Defaults (1) $\frac{:C}{\neg D}$; (2) $\frac{:D}{\neg E}$ und (3) $\frac{:E}{\neg F}$ gegeben und $W = \emptyset$. Zwei Beispiele für die Konstruktion von Extensionen sind: ($E_0 = \emptyset$)

- *(1) ist anwendbar: $E_1 = \{\neg D\}$; danach ist (2) nicht mehr anwendbar, aber (3): $E_2 = Th(E_1) \cup \{\neg F\}$. Also $E = Th(\{\neg D, \neg F\})$ ist eine Extension, da die Annahmen von (1) und (3) auch unter E noch gelten.*

- *Wendet man zuerst (2) an, dann erhält man $E_1 = \{\neg E\}$ und blockiert damit (3), aber nicht (1): $E_2 = Th(E_1) \cup \{\neg D\}$. Aber $E = Th(\{\neg D, \neg E\})$ ist keine Extension, da mit E die Annahme von (2) nicht mehr erfüllt ist.*

3.4 Eigenschaften

Einige wichtige Ergebnisse der Forschung über Default–Logik werden im folgenden zusammengefaßt.

3.4.1 Einschränkungen

Verschiedene Klassen von Einschränkungen der Default–Logik wurden untersucht, insbesondere im Hinblick auf die Möglichkeiten zur Wissensrepräsentation, der Ausdruckskraft und der Komplexität der Beweisprozeduren und Implementierungen. Möglichkeiten zur Einschränkung sind:

1. geschlossene Defaults: keine freien Variablen
2. normale Defaults: Annahme ist äquivalent zur Konsequenz
3. semi–normale Defaults: Annahme impliziert Konsequenz
4. Voraussetzungsfreie Defaults: keine Vorraussetzung, nur Annahme und Konsequenz
5. Einschränkungen der prädikatenlogischen Sprache, z.B. auf Horn–Klauseln oder aussagenlogische Formeln.

Insbesondere die Einschränkung auf normale, voraussetzungsfreie Defaults, wird im folgenden genauer untersucht werden. [Poole88] und [Brewka89] benutzen für die Einführung von Prioritäten solchermaßen eingeschränkte Defaults. Interessanterweise ist es möglich, normale Defaults in voraussetzungsfreie Defaults zu transformieren:

$(1)\frac{\alpha:\omega}{\omega} \rightsquigarrow (2)\frac{:\alpha\Rightarrow\omega}{\alpha\Rightarrow\omega}$

Diese Transformation ist vollständig, aber nicht korrekt.

Beweisskizze: (Vollständigkeit)
(1) ist anwendbar, wenn α gilt und ω konsistent ist. Dann ist auch $\alpha \Rightarrow \omega$ konsistent und damit (2) anwendbar. Aus der Anwendung von (1) folgt $\omega \in E$ und aus (2) folgt $\alpha \Rightarrow \omega \in E$ und $\alpha \in E$, und damit auch $\omega \in E$. $\square$

Das Problem der Korrektheit ist die Anwendbarkeit von (2) auch ohne Wissen über α. Gilt statt α nun $\neg\omega$, dann kann man mit Hilfe von (2) durch Kontraposition $\neg\alpha$ schließen. Das bedeutet (2) ist eigentlich äquivalent zu den beiden Defaults:

$(1)\frac{\alpha:\omega}{\omega}$ und $(1')\frac{\neg\omega:\neg\alpha}{\neg\alpha}$

In der nichtmonotonen Logik nach [DelgrandeJackson91], der P–Logik, tritt dieses Problem nicht auf.

Im Beispiel soll die Problematik noch einmal verdeutlicht werden.

Beispiel 3.4.1
Der Default

$\frac{Bird(X):Flies(X)}{Flies(X)}$ *„Vögel fliegen normalerweise“*

würde transformiert in den Default

$\frac{:Bird(X)\Rightarrow Flies(X)}{Bird(X)\Rightarrow Flies(X)}$ *„Normalerweise folgt aus Vogel sein, das fliegen können“.*

Dieser Default beinhaltet aber zusätzlich die Aussage

$\frac{\neg Flies(X):\neg Bird(X)}{\neg Bird(X)}$ *„Nichtfliegende Objekte sind normalerweise keine Vögel“.*

3.4.2 Inkonsistenz zwischen Annahmen

[Poole88] zeigt das Problem auf, daß durch eine Konsistenz–Annahme für eine Defaultanwendung die Anwendung eines anderen Defaults eigentlich verhindert werden müßte. Da aber die Annahmen bei nicht normalen Defaults keine Folgerungen der Extension sind, wird die Konsistenz–Annahme des ersten Defaults bei weiteren Konsistenzprüfungen nicht mehr berücksichtigt. Das Beispiel vom gebrochenen Arm soll dieses Problem erläutern. Man weiß ein Arm ist gebrochen:

$$W = \{Broken(LeftArm) \vee Broken(RightArm)\}$$

und normalerweise sind nichtgebrochene Arme zu gebrauchen:

$$D = \{\frac{:Usable(X) \wedge \neg Broken(X)}{Usable(X)}\}.$$

Die Extension der Default–Theorie enthält sowohl $Usable(LeftArm)$ als auch $Usable(RightArm)$, obwohl man eigentlich weiß, daß ein Arm gebrochen, also nicht zu gebrauchen ist. Dieses Problem tritt bei normalen Defaults nicht auf, da hier die Konsistenz–Annahme nach Anwendung des Defaults in die Extension mit aufgenommen wird.

3.4.3 Kumulativität

[Makinson89] definiert eine wünschenswerte Eigenschaft von Inferenzrelationen, die Kumulativität, als

$$\vdash \text{ ist kumulativ gdw } \forall W \subseteq \mathcal{L}_0 \;.\; \forall x, y \in \mathcal{L} \;.\; W \vdash y \leadsto W \vdash x \text{ gdw } W \cup \{y\} \vdash x$$

Das heißt, wenn man die Prämissenmenge um eine Folgerung erweitert, ändert sich die Menge der abgeleiteten Formeln nicht. Die Kumulativität entspricht dem Deduktionstheorem der Prädikatenlogik. Sie gilt jedoch für die Default–Logik nicht. Erweitert man nämlich die Menge W der Default–Theorie $\Delta = (D, W)$ um eine Formel aus der Extension, können u.U. weitere Extensionen möglich werden. Ein Beispiel aus [Makinson89]:

$$(1)\frac{:p}{p}; (2)\frac{p \vee q : \neg p}{\neg p}; W = \emptyset$$

hat die Extension

$$E = Th(\{p\}).$$

Ist die Folgerung $p \vee q$ aus E nun auch in W, dann gibt es zwei Extensionen, nämlich die bisherige und zusätzlich

$$E' = Th(\{p \vee q; \neg p\}).$$

Dieses Problem tritt nicht auf, wenn man die Konsequenz eines Defaults zur Menge W hinzufügt. Im Beispiel ändert die Erweiterung $W = \{p\}$ die resultierenden Extensionen nicht.

[Brewka91] definiert eine Erweiterung der Default–Logik, die kumulativ ist und das Problem der Inkonsistenz zwischen Annahmen löst. Die Sprache dieser Cumulative Default Logic (CDL) enthält keine einfachen Formeln mehr, sondern sogenannte Assertionen. Assertionen sind Paare aus Formeln und Supportmengen. Die Supportmenge enthält Formeln aus den Annahmen und Konsequenzen von Defaultanwendungen. Dadurch ist es möglich, für jede durch Defaults entstandene Formel die Umgebung zu speichern und damit die Konsistenz der Annahmen nicht nur mit der Extension, sondern auch mit den Supportmengen der Formeln zu überprüfen.

3.4.4 Semi–Monotonie

Geschlossene normale Default–Theorien haben die Eigenschaft, daß eine Erweiterung der Defaults eine Extension besitzt, die durch Erweiterung einer Extension aus den bisherigen Defaults entsteht.

Seien $\Delta = (D, W); \Delta' = (D', W)$ geschlossene normale Default–Theorien mit $D \subseteq D'$ und E eine Extension von Δ, dann hat Δ' eine Extension E' mit

1. $E \subseteq E'$ und
2. die angewendeten Defaults in D für E sind eine Teilmenge der angewendeten Defaults in D' für E'.

Diese Eigenschaft ist nützlich, weil damit eine Art von Lokalitätsbedingung für die Konstruktion von Extensionen ausgedrückt wird. Man kann zum Beispiel für eine Teilmenge von D zunächst Extensionen berechnen und dann durch Hinzunahme weiterer Defaults diese Extension verfeinern.

3.5 Beweisprozeduren und Komplexität

Eine der wichtigsten Eigenschaften von Default–Logik ist die Möglichkeit, einen Beweisbegriff zu definieren. Aufbauend auf diesem Beweisbegriff ist es möglich, Beweisprozeduren zu entwickeln. Zunächst muß man festlegen, was man beweisen möchte. Dazu kann man verschiedene informelle Anforderungen stellen:

1. Gegeben: Eine Default–Theorie Δ und eine Formel $\beta \in \mathcal{L}$
   Beweise: Es gibt eine Extension E von Δ, so daß $\beta \in E$.
2. Gegeben: Eine Default–Theorie Δ und eine Formel $\beta \in \mathcal{L}$
   Beweise: Für alle Extensionen E von Δ gilt $\beta \in E$.
3. Gegeben: Eine Default–Theorie Δ, eine Formel $\beta \in \mathcal{L}$ und ein aktueller Belief, d.h. eine Menge von Formeln die als gültige Annahmen gelten.
   Beweise: Es gibt eine Extension E von Δ, die mit dem aktuellen Belief verträglich ist, und für die $\beta \in E$ gilt. Erweitere dann den aktuellen Belief um β.

(1) entspricht einer optimistischen Sichtweise: alles ist beweisbar, was in mindestens einer Extension gültig ist. Die Vorgehensweise von (2) nur Formeln zu beweisen, die in allen Extensionen gültig sind, kann als pessimistischer Standpunkt angesehen werden. (3) garantiert, daß man nicht Formeln aus verschiedenen Extensionen beweist, sondern immer noch Extensionen behält, in denen alle bisher durchgeführten Beweise weiterhin möglich sind. Wir werden uns auf die erste Art von Beweisführung beschränken.

Die Default–Logik betrachtet jede mögliche Extension als korrekt. Im Gegensatz dazu wird bei Circumscription der pessimistische Standpunkt eingenommen, in dem man eine minimale Formelmenge sucht, die in allen Extensionen enthalten ist.

Der zweite Schritt auf dem Weg zu einer Beweisprozedur in nun zu beschreiben, wie ein Beweis syntaktisch auszusehen hat, d.h. nach dem „Was“ wird nun das „Wie“ beschrieben. [Reiter80] führt dazu den Begriff des *Default–Beweises* ein. Dieser beinhaltet im Prinzip die Konstruktion einer Folge von Defaults, aus deren Konsequenz schließlich die zu beweisende Formel folgt.

Definition 3.5.1 (Default–Beweis)
Sei $\Delta = (D, W)$ eine geschlossene normale Default–Theorie, $\beta \in \mathcal{L}_0$.
Eine endliche Folge $(\delta_0, ..., \delta_k)$ von Defaults ($\forall 0 \leq i \leq k \,.\, \delta_i \in D$ und $\delta_i = \frac{\alpha_i : \omega_i}{\omega_i}$) heißt Defaultbeweis *von β bzgl. Δ gdw*

1. *$W \cup \bigcup_{i=0}^{k} \omega_i$ ist konsistent oder W ist inkonsistent.*

2. *$\forall 0 \leq i \leq k \; . \; W \cup \bigcup_{j<i} \omega_j \vdash \alpha_i$.*

3. *$W \cup \bigcup_{i=0}^{k} \omega_i \vdash \beta$.*

Ein Beispiel eines Beweises für C aus der Default–Theorie $\Delta = (D, W)$ mit den Defaults:

$\delta_1 = \frac{:A}{A}; \delta_2 = \frac{A \wedge B:C}{C}; \delta_3 = \frac{E \vee F:A \Rightarrow B}{A \Rightarrow B}$

und $W = \{A \Rightarrow E\}$ ist die Folge $(\delta_1, \delta_3, \delta_2)$.

	(1) $W \cup \delta_1 \cup \delta_3 \cup \delta_2 = \{A \Rightarrow E, A, A \Rightarrow B, C\}$ ist konsistent.
Denn	(2) $W \vdash true$ und $W \cup \{A\} \vdash E \vee F$ und $W \cup \{A, A \Rightarrow B\} \vdash A \wedge B$
	(3) $W \cup \{A, A \Rightarrow B, C\} \vdash C$

Die Default–Beweisdefinition beinhaltet zwei große Schwierigkeiten: zunächst ist die Konsistenzprüfung in der Prädikatenlogik 1. Stufe nicht entscheidbar. Also ist nur mit eingeschränkten Formelmengen ein Korrektheitsnachweis für Beweise möglich. Die zweite Schwierigkeit ist, die Reihenfolge der benötigten Defaults zu bestimmen. Dazu wird jedoch noch eine Lösung in Form einer Beweisprozedur skizziert werden.

Daß diese Definition eines Beweises wirklich dem gewünschten optimistischen Beweisbegriff genügt, muß gezeigt werden. Es ist nötig, die Vollständigkeit und Korrektheit der Definition zu zeigen, d.h. wenn eine Formel in einer Extension enthalten ist, muß es einen Beweis dafür geben und es darf nur dann einen Beweis für eine Formel geben, wenn diese Formel in mindestens einer Extension enthalten ist.

Satz 3.5.1 (Korrektheit)
Sei $\Delta = (D, W)$ geschlossene normale Default–Theorie; $\beta \in \mathcal{L}_0$.
Wenn β einen Default–Beweis $(\delta_0, ..., \delta_k)$ bzgl Δ hat, dann hat Δ eine Extension E mit $\beta \in E$.

Beweisskizze: Zeige es gibt eine Extension E' von $\Delta' = (\bigcup_{i=0}^{n} \delta_i; W)$ mit $\beta \in E'$, denn gemäß der Semi–Monotonie–Eigenschaft gilt dann: $\exists E \; . \; \beta \in E$. Da sich Extensionen nicht ändern, wenn man Konsequenzen von Defaults, die mit der Extension verträglich sind, zu der Menge W hinzufügt, [2] kann man eine Default–Theorie $\Delta_i = (\bigcup_{i=0}^{n} \delta_i; W \cup \bigcup_{0 \leq j < i} w_j)$ konstruieren, da nun für die Extension E_k von Δ_k gilt: $W \cup \bigcup_{0 \leq j < k} w_j \subseteq E_k$ und $W \cup \bigcup_{0 \leq j < k} w_j \vdash \beta$ ist also $\beta \in E_k$ und $E' = E_k$ die gesuchte Extension. $\square$

Satz 3.5.2 (Vollständigkeit)
Sei $\Delta = (D, W)$ geschlossene normale Default–Theorie; $\beta \in \mathcal{L}_0$.
Wenn Δ eine Extension E mit $\beta \in E$ hat, dann gibt es einen Default–Beweis für β bzgl. Δ.

Beweisskizze: Mit Hilfe von Satz 3.3.1 macht man einen Induktionsbeweis, in den man zeigt, daß $\forall i \geq 0$ gilt: für jede Formel in E_i gibt es einen Default–Beweis bzgl. Δ, so daß die Menge der benutzten Defaults eine Teilmenge der Defaults sind, die mit der Extension verträglich sind.

[2] Man beachte den Unterschied zum Problem der Kumulativität: es wird nur die Konsequenz und keine Folgerung aus dieser hinzugefügt

Diese Teilmengenbeziehung benötigt man, um die Konsistenz der Defaults des Beweises zu zeigen. Da $\beta \in E$ muß es einE_i geben mit $\beta \in E_i$. Hat man also die Induktion gezeigt, dann hat man auch einen Beweis für β. □

Nachdem nun die theoretische Nützlichkeit gezeigt ist, verbleibt das wichtige Problem der praktischen Anwendbarkeit des Beweisbegriffes. Zunächst besteht die Schwierigkeit die Konsistenzüberprüfung einer Formelmenge in der Prädikatenlogik 1. Stufe durchzuführen. Es gibt kein Entscheidungsverfahren, um die Erfüllbarkeit einer Formel zu zeigen oder zu widerlegen [Church36]. Für Default–Beweise bedeutet diese Nichtentscheidbarkeit, daß die Menge der mit Defaults beweisbaren Formeln nicht rekursiv aufzählbar ist. Anders ausgedrückt ist es weder möglich für alle beweisbaren Formeln zu zeigen, daß ein Default–Beweis existiert, noch ist es möglich für die nichtbeweisbaren zu zeigen, daß kein Default–Beweis existiert. Die Beweisbarkeit ist also nicht einmal semi–entscheidbar. Dieses negative Resultat ist aber kein Fehler des Beweisbegriffs, sondern es handelt sich um eine inhärente Eigenschaft der Default–Logik.

Satz 3.5.3 (Nicht–Semi–Entscheidbarkeit)
Sei $\Delta = (D, W)$ eine geschlossene normale Default–Theorie; $\beta \in \mathcal{L}_0$.
Es ist nicht semi–entscheidbar, ob für Δ eine Extension E existiert mit $\beta \in E$.

Beweisskizze: Sei $w_1, w_2, ...$ eine Aufzählung aller Formeln in $\mathcal{L}_0$. Für jede Formel definieren wir eine geschlossene normale Default–Theorie $\Delta_i = (\frac{:w_i}{w_i}; \emptyset)$. Jedes Δ_i hat eine eindeutige Extension E_i:

$E_i = \emptyset$, wenn w_i unerfüllbar ist und

$E_i = Th(\{w_i\})$, wenn w_i erfüllbar ist.

Annahme: Jedes E_i ist rekursiv aufzählbar (= semi–entscheidbar), dann ist auch die Menge der erfüllbaren Formeln $\bigcup_{i=1}^{\infty} E_i$ rekursiv aufzählbar (Diagonalisierung). Die Menge der erfüllbaren Formeln ist aber in Prädikatenlogik 1. Stufe nicht rekursiv aufzählbar, also ist die Annahme falsch. □

Dieses negative Resultat sollte nicht davon abschrecken, trotzdem Beweisprozeduren zu suchen. Man kann sich zum Beispiel auf beweisbare Teiltheorien beschränken. Des weiteren ist es möglich Heuristiken zu benutzen, die in den meisten und relevanten Fällen Beweise finden können.

Das zweite Problem der praktischen Anwendbarkeit ist die Konstruktion einer Folge von Defaults. Grundsätzlich bieten sich zwei Möglichkeiten, eine Reihenfolge der Defaults zu bestimmen:

1. **Top–Down–Idee**
   Man beginnt mit der zu beweisenden Formel β und versucht Konsequenzen von Defaults zu finden, die zusammen mit der Menge W als logische Folgerung β besitzen. Dann muß rekursiv für jeden benötigten Default die Voraussetzung entsprechend hergeleitet werden. Dies entspricht im Prinzip einer Rückwärtsverkettung, wie z.B. in Prolog, da zunächst die Konsequenzen von Defaults benutzt werden und dann versucht wird, die Voraussetzungen dieser Defaults zu bestätigen.

2. **Bottom–Up–Idee**
   Zunächst testet man, ob β bereits aus W herleitbar ist. Wenn nicht, prüft man, welcher Default anwendbar ist, d.h. ob die Voraussetzung herleitbar und die Annahme konsistent mit W ist. So erweitert man sukzessiv die Menge der hergeleiteten Formeln bis man β erhält. Beim Bottom–Up–Verfahren werden die Default–Regeln vorwärts, wie z.B. in OPS5 angewandt.

Natürlich kann man diese beiden Grundideen weiter verfeinern und eventuell kombinieren, um schnelle Beweisverfahren zu erhalten. [Reiter80] entwickelt basierend auf der Top–Down–Idee und mit Hilfe eines linearen Resolutionsbeweisers eine Beweisprozedur für Default–Beweise.

Die Komplexität von Default–Reasoning wurde besonders in [KautzSelman89] genauer analysiert. Sie definieren verschiedene Klassen von Defaults: Einerseits durch Einschränkungen der prädikatenlogischen Sprache, z.B. auf Aussagenlogik. Andererseits durch Einschränkungen der Defaults, z.B. dürfen nur unäre Literale im Default auftreten. [Etherington88] bzw. [KautzSelman89] haben gezeigt, daß

1. Für eine endliche, aussagenlogische, disjunktionsfreie, geordnete Default–Theorie [3] läßt sich eine Extension in $O(n^2)$ berechnen. Eine geordnete Default–Theorie enthält im Prinzip eine Ordnung auf den Formeln und verhindert Zykel zwischen Voraussetzungen und Annahmen.

2. Für eine endliche, aussagenlogische, unäre Default–Theorie ist die Berechnung einer Extension NP–hart. Eine unäre Default–Theorie enthält ein positives Literal als Voraussetzung, ein Literal als Konsequenz und die Annahme ist entweder die Konsequenz oder eine Konjunktion aus Konsequenz und einem positiven Literal, falls die Konsequenz ein negatives Literal ist.

3.6 Offene Defaults

Die bisherigen Ergebnisse dieses Kapitels wurden nur für geschlossene Defaults ermittelt. Um auch offene Defaults sinnvoll verwenden zu können, ist es nötig, den Begriff der geschlossenen Default–Theorie zu verallgemeinern. Es wird im folgenden aufgezeigt, wie man auch für Defaults mit freien Variablen Extensionen definieren kann. Diese Definition wird es erlauben, die bisherigen Ergebnisse entsprechend zu übertragen.

Ein Default $\delta(x) = \frac{\alpha(x):\beta(x)}{\omega(x)}$ mit freien Variablen $x = x_1, ..., x_n$, soll die intuitive Bedeutung besitzen: Für jede Einsetzung $o = o_1, ..., o_n$ von Objekten der Default–Logik für x, folgt aus $\alpha(o)$ und mit der Konsistenz der Annahme $\beta(o)$, daß $\omega(o)$ gilt. Defaults mit freien Variablen bilden eine Art von Muster und werden durch Instanziierung mit Objekten in eine Menge von geschlossenen Defaults umgewandelt.

Für eine genaue Definition muß man zunächst festlegen, was die Objekte einer Default–Logik Δ sind. Diese Menge von Objekten bezeichnen wir als Default–Universum $U(\Delta)$. Im Prinzip entspricht die Menge der Objekte dem Herbranduniversum der prädikatenlogischen Sprache unter Berücksichtigung der Defaults.

Das Herbranduniversum $H(\mathcal{L})$ einer prädikatenlogischen Sprache $\mathcal{L}$ umfaßt alle Grundterme von $\mathcal{L}$, d.h. die nullstelligen Funktionssymbole (Konstanten) [4] und die aus Grundtermen rekursiv zusammengesetzten Terme. Zusätzlich zu diesen expliziten Objekten gibt es im allgemeinen weitere implizit definierte Objekte. Diese entstehen durch existenzquantifizierte Variablen, z.B. bedeutet die Formel $\exists x \,.\, P(x)$, daß es ein Objekt o gibt mit $P(o)$.

Zur Bestimmung dieser implizit definierten Objekte, benutzt man die Skolemisierung. Skolemisierung transformiert beliebige Formeln in Formeln ohne existenzquantifizierte Variablen durch Einführung neuer Funktionssymbole. Sei $w \in \mathcal{L}$ eine Formel. Zur Bestimmung der skolemisierten Form von w, muß w in pränexe Normalform gebracht werden, damit die Existenz- und Allquantoren die „richtige" Stellung haben. Die pränexe Normalform von w zeichnet sich durch zwei

[3] geordnete Defaults werden in [Etherington88] genau definiert

[4] Es wird vorausgesetzt, daß mindestens ein nullstelliges Funktionssymbol existiert

Bedingungen aus: (1) Die pränexe Normalform muß äquivalent zu w sein und (2) Quantoren müssen vor den Negationen stehen.

Sei $w' \in \mathcal{L}$ die pränexe Normalform von w. $w'' \in \mathcal{L}$ ist eine Skolem–Form von w, wenn gilt, daß jede existenzquantifizierte Variable y durch ein neues Funktionssymbol ersetzt wird, dessen Stelligkeit der Anzahl der Allquantoren entspricht, in deren Bindungsbereich der Existenzquantor steht. Die Argumente des neuen Funktionssymbols sind gerade die allquantifizierten Variablen.

Der Algorithmus zur Herstellung der pränexe Normalform und die Skolemisierung ist z.B. in [Richter89] (Kapitel 1) genau dargestellt.

Durch die Skolemisierung erhält man eine neue Sprache $\mathcal{L}'$, die zusätzliche Funktionssymbole enthält. Die Menge der Objekte wird durch das vergrößerte Herbranduniversum der Sprache $\mathcal{L}'$ bestimmt.

Für die Default–Logik $\Delta = (D, W)$ können wir nun die Objekte durch die Grundterme der Sprache $\mathcal{L}$ und die Skolemisierung von W bestimmen. Zusätzliche Objekte können aber auch durch die Defaults eingeführt werden. Zum Beispiel wird durch den Default

$$\frac{:\exists x\ .\ P(x)}{\exists x\ .\ P(x)}$$

ein implizit definiertes Objekt o beschrieben, für das normalerweise P gilt. Folglich müssen auch die Defaults skolemisiert werden.

Definition 3.6.1 (skolemisierter Default)
Sei $\delta(x) = \frac{\alpha(x):\beta(x)}{\omega(x)}$ *ein Default.*
Die skolemisierte Form von $\delta(x)$ *ist der Default*

$\delta'(x) = \frac{\alpha(x):\beta(x)}{\omega'(x)}$ *mit*

$\omega'(x)$ *ist die skolemisierte Form des universellen Abschlusses von* $\omega(x)$*.*

Mit diesen Begriffen ist es einfach zu definieren, was eine skolemisierte Default–Theorie ist.

Definition 3.6.2 (skolemisierte Default–Theorie)
Sei $\Delta = (D, W)$ *eine Default–Theorie.*
$\Delta' = (D', W')$ *ist die skolemisierte Default–Theorie zu* Δ*, wenn gilt*

- $D' = \{\delta'(x) \mid \delta'(x)$ *ist skolemisierte Form von* $\delta(x) \in D\}$
- $W' = \{w' \mid w'$ *ist skolemisierte Form von* $w \in W\}$

Zu dieser Definition einige Bemerkungen:

- $\alpha(x)$ und $\beta(x)$ werden nicht skolemisiert, da sie nur zum Testen der Anwendbarkeit von Defaultregeln dienen und nicht in der Extension auftreten, also praktisch keinen aktiven Beitrag leisten.
- Auch Defaults, die nicht anwendbar sind, werden skolemisiert, obwohl sie in der Extension ebenfalls nicht mehr auftreten.
- Die skolemisierte Form Δ' von Δ ist nicht eindeutig, da man für die Skolemisierung beliebige neue Symbole wählen kann. Jedoch sind alle diese Skolemisierungen von Δ isomorph zueinander und durch Umbenennung direkt ineinander zu transformieren.

- Für die skolemisierte Form der Konsequenz von Defaults wird der universelle Abschluß benutzt, da die freien Variablen implizit als als allquantifiziert angesehen und benutzt werden.
- Das Herbrand–Universum ist im allgemeinen unendlich und rekursiv aufzählbar. Die Unendlichkeit ist dadurch gegeben, daß, wenn mindestens ein nichtnullstelliges Funktionssymbol existiert, man dieses beliebig oft verschachteln kann: $f(f(...(f(f(a)))))$. Rekursiv aufzählbar ist das Herbrand–Universum dann, wenn die Funktionssymbole, wie im allgemeinen vorausgesetzt, rekursiv aufzählbar sind, da die Terme rekursiv definiert sind.
- Das Default–Universum $U(\Delta)$ ist das Herbrand–Universum der Sprache $\mathcal{L}'$ der skolemisierten Default–Theorie Δ'

Mit Hilfe des Default–Universums kann man, aufbauend auf der Definition der Extension für geschlossene Defaults, den Begriff der Extension auch für allgemeine Defaults definieren.

Definition 3.6.3 (Extension)
Sei $\Delta = (D, W)$ eine Default–Theorie und $\Delta' = (D', W')$ eine skolemisierte Form von Δ.
$E \subseteq \mathcal{L}$ ist eine Extension von Δ/Δ', wenn E eine Extension von $\Delta'' = (D'', W')$ nach Definition 3.2.3 ist, mit

$$D'' = \{\delta(g) \mid \delta(x) \in D' \text{ und } g = g_1, ..., g_n \text{ mit } g_i \in H(\mathcal{L}') \text{ und } \mathcal{L}' \text{ ist Sprache von } \Delta'\}$$

Die Menge D'' entspricht der Menge aller grundinstanziierten Defaults und ist deshalb in der Regel unendlich. In den bisherigen Definitionen und Sätzen wurde die Endlichkeit nicht gefordert, also sind die Ergebnisse entsprechend auf allgemeine Defaults zu übertragen. Ein Problem verbleibt jedoch: berechnet man für eine geschlossene Default–Theorie die Extension nach Definition 3.6.3 erhält man zwar ein anderes Ergebnis als bei der Extension nach Definition 3.2.3. Da wir einmal die Default–Theorie skolemisieren und einmal nicht, ist die Extension in zwei unterschiedlichen Sprachen kodiert. Es handelt sich hierbei jedoch um kein prinzipielles, sondern um ein rein syntaktisches Problem.

[Brewka89] diskutiert einige Besonderheiten bei der Verwendung von freien Variablen, die bereits aus der Prädikatenlogik bekannt sind. Er beschreibt, welche Unterschiede zwischen freien und allquantifizierten Variablen bestehen. Damit begründet er gleichzeitig, wieso man überhaupt freie Variablen benötigt und nicht statt dessen Allquantoren benutzt. Der Unterschied besteht in der Möglichkeit, Defaults mit freien Variablen auch dann anzuwenden, wenn nicht alle Variablenbelegungen die Voraussetzung erfüllen oder die Konsistenz der Annahme garantieren. Allquantifizierte Variablen erlauben dagegen nur dann den Default zu benutzen, wenn alle Belegungen erfüllbar bzw. konsistent sind.

Beispiel 3.6.1
Sei der Default
$\delta(x) = \frac{:Vogel(x) \Rightarrow Fliegt(x)}{Vogel(x) \Rightarrow Fliegt(x)}$
und die Fakten

$$Vogel(Tweety), Pinguin(Fred), \forall x \;.\; Pinguin(x) \Rightarrow Vogel(x), \forall x \;.\; Pinguin(x) \Rightarrow \neg Fliegt(x)$$

gegeben, dann ist $\delta(x)$ für Tweety, nicht aber für Fred anwendbar. Damit ist $Fliegt(Tweety)$ in der Extension.

Würde der Default stattdessen $\delta' = \frac{:\forall x \;.\; Vogel(x) \Rightarrow Fliegt(x)}{\forall x \;.\; Vogel(x) \Rightarrow Fliegt(x)}$ heißen, wäre er überhaupt nicht anwendbar, da $\forall x \;.\; Vogel(x) \Rightarrow Fliegt(x)$ nicht mit den Fakten konsistent ist. Insbesondere wäre $Fliegt(Tweety)$ in keiner Extension.

Kapitel 4

Defaults mit Prioritäten

4.1 Einführung

Nachdem in Kapitel 3 Defaults allgemein eingeführt wurden, werden nun verschiedene Varianten diskutiert, die eine adäquatere Repräsentation von Wissen für einige Problemstellungen erlauben.

Zunächst werden wir die Hypothesen–Theorie einführen, die eine neue Sichtweise von Defaults liefert. Die Annahmen der Defaults werden als mögliche Hypothesen betrachtet, die nach einer Überprüfung für Ableitungen nutzbar sind.

Nach der Hypothesen–Theorie führen wir die Prioritäten ein. Prioritäten erlauben eine Spezifikation von Abhängigkeiten zwischen Defaults. Man führt damit ein neues Sprachkonstrukt ein, das über Formeln definiert ist. Es wird also eine Meta-Sprache zur prädikatenlogischen Sprache definiert.

4.2 Hypothesen–Theorie

Es soll nun ein Ansatz für das nichtmonotone Schließen beschrieben werden, der keine neue Logik begründet, sondern den deduktiven Ableitungsbegriff der Prädikatenlogik 1. Stufe abwandelt. Dieser Ansatz ([Poole88]) basiert auf der Idee, daß ein Inferenzsystem einen Mechanismus besitzen muß um mit Hypothesen umzugehen. Eine der größten Schwierigkeiten hypothetischen Schließens ist die Hypothesengenerierung (siehe zum Beispiel [Richter89], Kap. 10). Da wir jedoch davon ausgehen, die Hypothesen bereits in Form von Defaults durch den Benutzer vorgegeben zu haben, sind unsere Probleme reduziert auf die Auswahl der „richtigen“ Hypothesen und die Bestimmung der damit möglichen Schlüsse. Poole's Konzept beinhaltet eine Charakterisierung der Default-Logik als Konstruktion einer Theorie mit Hilfe von Defaults und Fakten:

> If one allows hypothetical reasoning, then there is no need to define a new logic to handle nonmonotonic reasoning

4.2.1 Grundlagen

Grundlage der folgenden Formalisierung ist wiederum eine prädikatenlogische Sprache $\mathcal{L}$, wobei mit $\mathcal{L}_0 \subseteq \mathcal{L}$ die Formeln ohne freie Variablen bezeichnet werden. Eine Hypothesen–Theorie besteht aus zwei Formelmengen: einer Menge von Fakten und einer Menge von Hypothesen. Man beachte den Unterschied zwischen dem Begriff der Theorie, die eine Formelmenge beschreibt, und dem Begriff der Hypothesen–Theorie, die durch ein Tupel von Formelmengen bestimmt ist.

Definition 4.2.1 (Hypothesen–Theorie)
Sei $\mathcal{L}$ eine prädikatenlogische Sprache, $H \subseteq \mathcal{L}$ und $W \subseteq \mathcal{L}_0$.
$\Delta = (H, W)$ *heißt* Hypothesen–Theorie.
H ist die Menge der möglichen Hypothesen und W ist die Faktenmenge.

Eine Unter–Theorie der Hypothesen–Theorie besteht aus den Fakten vereinigt mit einer Teilmenge der Hypothesen, die konsistent mit den Fakten sein muß. Eine solche Unter–Theorie ist eine Theorie im Sinne der Prädikatenlogik, d.h. durch die Formeln in der Unter–Theorie werden die zulässigen Modelle festgelegt.

Wir bezeichnen die Menge der Grundinstanzen einer Formelmenge $F \subseteq \mathcal{L}$ mit $G(F)$:

$$G(F) = \{f(t_1, ..., t_n) \mid f(x_1, ..., x_n) \in F \text{ und } t_1, ..., t_n \text{ sind Grundterme } \}$$

Definition 4.2.2 (Unter–Theorie)
Sei $\Delta = (H, W)$ eine Hypothesen–Theorie.
$S \subseteq \mathcal{L}_o$ ist eine Unter–Theorie (engl. scenario) *von Δ gdw*

1. *$S = W \cup H_0$ mit $H_0 \subseteq G(H)$.*

2. *S ist konsistent oder W ist bereits inkonsistent.*

Die Bedingungen besagen, daß die Unter–Theorie alle Fakten und instanziierte Hypothesen enthält und daß mit den Hypothesen keine Inkonsistenz eingeführt wird.

In der folgenden Definition wird der Begriff der Erklärung verwendet. Erklärungen gibt es in sehr vielen unterschiedlichen Bedeutungen. Wir betrachten hier nur einen sehr eingeschränkten Aspekt.

Erklärungen beinhalten hier die Beschreibung der Ursache eines beobachteten Sachverhalts. Sie können zum einen angeben warum eine Formel gilt, indem sie die benötigten Hypothesen zur Herleitung einer Formel generieren.

In unserem Zusammenhang wird Erklärung noch in einem anderen Sinn gebraucht. Sie dient zusätzlich zur Definition eines Ableitungsbegriffes in der Hypothesen–Theorie. Dieser Ableitungsbegriff ist als klassische Ableitung von maximalen Unter–Theorien definiert.

Definition 4.2.3 (Erklärung)
Sei $\Delta = (H, W)$ Hypothesen–Theorie und $w \in \mathcal{L}$.
S heißt Erklärung (engl. explanation) *von w bzgl. Δ, wenn gilt*

1. *S ist Unter–Theorie von Δ*

2. *$S \vdash w$, wobei $\vdash$ ein monotoner Ableitungsoperator der Prädikatenlogik 1. Stufe ist.*

w heißt erklärbar aus Δ, wenn es eine Erklärung S von w aus Δ gibt.

Um die Menge der möglichen Ableitungen und Erklärungen zu charakterisieren, benützt man wiederum den Begriff der Extension. Eine Extension umfaßt alle aus einer maximalen Unter–Theorie ableitbaren Formeln.

Definition 4.2.4 (Extension)
Sei $\Delta = (H, W)$ Hypothesen–Theorie.
$E \subseteq \mathcal{L}$ heißt Extension *von Δ, wenn es eine maximale (bzgl. Mengeninklusion) Unter–Theorie S gibt mit $E = Th(S)$.*

Die Menge aller Extensionen von Δ bezeichnen wir mit $\mathcal{E}(\Delta)$

Zwischen den Begriffen Erklärung und Extension gibt es einen engen Zusammenhang, der im folgenden Satz präzisiert wird. Jede erklärbare Formel ist in einer Extension enthalten und alle Formeln einer Extension sind erklärbar.

Satz 4.2.1
Sei $\Delta = (H, W)$ Hypothesen–Theorie und $w \in \mathcal{L}$.
w ist erklärbar aus Δ gdw es eine Extension E von Δ gibt mit $a \in E$.

Beweisskizze:

„$\Rightarrow$“: Ist w erklärbar, dann gibt es eine Erklärung S. Sei S' eine maximale Unter–Theorie mit $S \subseteq S'$, dann gilt wegen der Monotonie der klassischen Ableitung $S \vdash w \leadsto S' \vdash w$. Also $w \in Th(S')$ und $Th(S')$ ist eine Extension.

„$\Leftarrow$“: Sei $w \in E$, dann gibt es eine Unter–Theorie S mit $E = Th(S)$ und $S \vdash w$. Also ist w erklärbar. $\square$

4.2.2 Bezug zur Default–Logik

Wie die Schreibweise von Hypothesen–Theorien bereits andeutet, besteht ein enger Zusammenhang zur Default–Logik von Reiter. Zunächst einmal gibt es in beiden nichtmonotonen Ansätzen eine Faktenmenge, zur Modellierung des sicheren Wissens. Zusätzlich enthält die Default–Logik eine Menge von Defaultregeln, die basierend auf Annahmen neue Formeln generieren. Auch in der Hypothesen–Theorie kann man die Hypothesen als Default-Annahmen ansehen. Diese Hypothesen werden dann als gültig angenommen, wenn sie mit dem sicheren Wissen und den bisher benutzten Hypothesen konsistent sind. Genauso werden in der Default–Logik die Defaultregeln nur dann angewendet, wenn die Annahmen mit den Fakten und den abgeleiteten Formeln konsistent sind. Die Analogie zur Anwendung von Defaultregeln ist in der Hypothesen–Theorie die Sichtweise von Erklärungen als Generierung von Formeln.

Beispiel 4.2.1
Mit einer Hypothesen–Theorie soll modelliert werden, daß „Vögel normalerweise fliegen“. $\Delta = (H, W)$ bestehe aus den Formeln:

$H = \{Vogel(X) \Rightarrow Fliegt(X)\}$ *und*
$W = \{\forall X\ .\ Pinguin(X) \Rightarrow Vogel(X), \forall X. Pinguin(X) \Rightarrow \neg Fliegt(X),$
$\quad Vogel(Tweety), Pinguin(Fred)\}$

Es gilt dann:

1. *$Fliegt(Tweety)$ ist erklärbar aus $\{Vogel(Tweety) \Rightarrow Fliegt(Tweety)\}$*

2. *$Fliegt(Fred)$ ist nicht erklärbar, da $W \cup \{Vogel(Fred) \Rightarrow Fliegt(Fred)\}$ nicht konsistent ist.*

Eine wichtige Fragestellung betrifft die Ausdrucksmächtigkeit der Hypothesen–Theorie im Vergleich zur Default–Logik. Zunächst einmal ist es möglich zu zeigen, daß die Hypothesen–Theorie äquivalent zu einer Default–Theorie mit voraussetzungsfreien normalen Defaults ist.

Satz 4.2.2 (Äquivalenz)
Sei $\Delta = (H, W)$ eine Hypothesen–Theorie; $\Delta = (D, W)$ eine Default–Theorie mit den Defaults $D = \{\frac{:h}{h} \mid h \in H\}$, dann gilt:

1. *Alle Extensionen E von Δ sind auch Extensionen von Δ'.*
2. *Alle Extensionen E' von Δ' sind auch Extensionen von Δ.*

[Poole88] zeigt den ersten Teil dieses Satzes, in dem er nachweist, daß eine Extension von Δ der Definition 3.6.3 der Extension für offene Default–Theorien entspricht.

Lemma 4.2.1
Sei E Extension der Hypothesen–Theorie $\Delta = (H, W)$, dann gilt:

(E1) $W \subseteq E$

(E2) $Th(E) = E$

(E3) *Ist h_0 Grundinstanz einer Formel $h \in H$ und $\neg h_0 \notin E$, dann ist $h_0 \in E$.*

(E4) *E ist eine minimale Formelmenge, die die Bedingungen (E1) bis (E3) erfüllt.*

Beweis: Sei S eine maximale Unter–Theorie von Δ mit $E = Th(S)$.

(E1) Da $W \subseteq S \leadsto W \subseteq Th(S) = E$

(E2) $Th(E) = Th(Th(S)) = Th(S) = E$

(E3) Widerspruchsbeweis: Sei h_0 Grundinstanz von $h \in H$ mit $\neg h_0 \notin E$ und $h_0 \notin E$, dann gilt insbesondere W ist konsistent und $S \not\vdash \neg h_0$, also ist $S \cup \{h_0\}$ konsistent. $S \cup \{h_0\}$ ist aber auch Obermenge von S und damit ist S nicht maximal, also E keine Extension.

(E4) Widerspruchsbeweis: Annahme: Es existiert $E' \subset E$, das (E1) - (E3) erfüllt. Dann gilt: $S \not\subseteq E'$, denn sonst gilt mit (E2) $Th(S) \subseteq E'$, also $E \subseteq E'$ und damit haben wir einen Widerspruch zur Annahme. Also existiert $\alpha \in S$ mit $\alpha \notin E'$. Da $W \subseteq E'$ nach (E1), gilt zusätzlich, daß α Grundinstanz einer Formel $h \in H$ ist. Damit gilt nun aber $\alpha \notin E'$ und $\neg\alpha \notin E$, insbesondere $\neg\alpha \notin E'$. Also ist (E3) verletzt und E maximal. $\Box$

Der zweite Teil der Äquivalenz von Hypothesen–Theorie und Default–Theorie, nämlich daß jede Extension der Default–Theorie Δ' auch eine Extension von Δ ist, wurde bisher noch nicht gezeigt.

Lemma 4.2.2
Sei $\Delta = (H, W)$ eine Hypothesen–Theorie und $E \subseteq \mathcal{L}_0$ eine Menge, die folgende Eigenschaften erfüllt:

(E1) $W \subseteq E$

(E2) $Th(E) = E$

(E3) *Ist h_0 Grundinstanz einer Formel $h \in H$ und $\neg h_0 \notin E$, dann ist $h_0 \in E$.*

(E4) *E ist eine minimale Formelmenge, die die Bedingungen (E1) bis (E3) erfüllt.*

Dann ist E eine Extension der Hypothesen–Theorie Δ.

Beweis: Wenn W inkonsistent ist, ist der Beweis klar. Im folgenden sei also W konsistent
Sei $S = W \cup H_0$ mit $H_0 = \{h_0 \mid \exists h \in H$ mit h_0 ist Grundinstanz von h und $h_0 \in E\}$.
Beh.:

1. $W \cup H_0$ ist konsistent
2. $Th(S) = E$
3. S ist maximal

Bew.:

1. klar, da E konsistent
2. „$\subseteq$" $S \subseteq E \leadsto Th(S) \subseteq Th(E) = E$
   „$\supseteq$" zu zeigen $E \subseteq Th(S)$.
   $Th(S)$ erfüllt (E1) - (E3), also muß wegen der Minimalität von E gelten: $E \subseteq Th(S)$.
3. Annahme: $h_0 \notin S$ und $W \cup H_0 \cup \{h_0\}$ konsistent $\leadsto W \cup H_0 \not\vdash \neg h_0 \leadsto \neg h_0 \in E$ (wegen 2).
   Also ist nach (E3) $h_0 \in E$ und $h_0 \in H_0 \subseteq S\Box$

Damit ist jetzt gezeigt, daß die Hypothesen–Theorie äquivalent ist zur Default–Logik mit voraussetzungsfreien, normalen Defaults. Es bleibt das Problem, welche Ausdruckskraft die Hypothesen–Theorie im Vergleich zur Default–Logik besitzt.

Das erste Manko der Hypothesen–Theorie, die fehlende Voraussetzung, kann man als linke Seite einer Implikation in die Annahmen kodieren. Zu den Schwierigkeiten bei dieser Transformation siehe Einschränkungen der Defaults in Abschnitt 3.4.1.

Zweitens ist es möglich semi-normale Defaults zu modellieren, indem man Hypothesen benennt und Nichtanwendbarkeitsregeln einführt. Diese Benennung wird im folgenden Abschnitt genau erläutert.

4.2.3 Benennung von Hypothesen

Die Benennung einer Hypothese besteht aus einem neuen Prädikatssymbol mit einer Stelligkeit, die der Anzahl der freien Variablen in der Hypothese entspricht. Syntaktisch schreibt man eine benannte Hypothese als

$p_h(x) : h(x)$ mit $h(x) \in H, p_h$ ist ein neues Prädikatssymbol.

Ein Beispiel für einen benannten Default ist:

$NormFamilie(x) : Familie(x) \Rightarrow HatZweiKinder(x)$

Die Benennung von Hypothesen ist eigentlich nur eine abkürzende Schreibweise, also eine Art von Makro. Man kann die Benennung auch als Definition ansehen, in der ein Prädikat $p_h(x)$ einer u.U. komplexen Formel $h(x)$ gleichgesetzt wird: $p_h(x) \Leftrightarrow h(x)$.

Man hat nun drei verschiedene Ausdrucksmöglichkeiten:

1. Fakten
2. Hypothesen
3. benannte Hypothesen

Die benannten Hypothesen sind im Prinzip nur syntaktische Vereinfachungen ohne zusätzliche Ausdruckskraft, denn sie kodieren letztlich nur eine Hypothese durch eine einfachere Hypothese und einen Fakt um. Die Transformation einer Hypothese in eine benannte Hypothese ändert die Menge der erklärbaren Formeln nicht.

Satz 4.2.3
Sei $\Delta = (H, W)$ eine Hypothesen–Theorie, $g \in \mathcal{L}_0, h(x) \in H, p_h$ ein neues [1] Prädikatssymbol mit der Stelligkeit von h. Δ' sei die Hypothesen–Theorie die aus Δ durch Benennung von h entsteht:
$\Delta' = (H', W')$ *mit*
$H' = (H \backslash \{h(x)\}) \cup \{p_h(x)\}$ *und* $W' = W \cup \{p_h(x) \Rightarrow h(x)\}$
Es gilt: g ist erklärbar aus Δ gdw g ist erklärbar aus Δ'

Beweis: Der Fall, daß W inkonsistent ist, ist wieder klar. Also sei W konsistent.

„$\Rightarrow$" Sei S Erklärung von g bzgl. Δ und $V_h = \{v \mid h(v) \in S\}$.
$S' = (S \backslash \{h(v) \mid v \in V_h\}) \cup \{p_h(v) \mid v \in V_h\} \cup \{\forall x \,.\, p_h(x) \Rightarrow h(x)\}$ ist Erklärung von g bzgl. Δ'.

1. Zeige $S' \vdash g$.
   $\forall v \in V_h \,.\, S' \vdash p_h(v)$
   $\leadsto \forall v \in V_h \,.\, S', p_h(v) \vdash h(v)$
   $\leadsto S' \vdash S$ und $S \vdash g$ also $S' \vdash g$.

2. Zeige S' ist konsistent.
   $S'' = S \cup \{p_h(v) \mid v \in V_h\} \cup \{\forall x \,.\, p_h(x) \Rightarrow h(x)\}$ ist konsistent, da S konsistent ist und p_h nicht in S und $\forall v \in V_h \,.\, h(v) \in S$.
   Es gilt weiter $S' \subseteq S''$, also ist S' konsistent, wenn S'' konsistent ist.

„$\Leftarrow$" Im Prinzip analoger Beweis. $\Box$

4.2.4 Wozu benötigt man benannte Hypothesen?

Zunächst vereinfachen solche benannten Hypothesen die Struktur der Hypothesenmenge und insbesondere für Implementierungen wird die Verwaltung der Hypothesen übersichtlicher. Ein zweiter und eigentlich der wichtigste Grund Benennungen einzuführen, ist die Möglichkeit in der Faktenmenge Hypothesen explizit anzusprechen und Ausnahmen von Defaults durch Fakten explizit zu machen. Damit können semi-normale Defaults realisiert werden:

Ein semi-normaler Default $\frac{:\omega \wedge \beta}{\omega}$ kann transformiert werden in einen benannten Default $p : \omega$ und einen Fakt $\neg\beta \Rightarrow \neg p$.

Diese Transformation ist nicht ganz korrekt, wie wir im folgenden Abschnitt sehen werden. Sie liefert aber in vielen interessanten Fällen die gewünschte Bedeutung und ist ein Indiz dafür, daß man meist mit normalen Defaults auskommt.

[1] Das heißt p_h kommt nicht in H, W und g vor

Beispiel 4.2.2
Gegeben seien die folgenden Defaults:

$$\frac{:Student(x) \Rightarrow \neg HatArbeit(x)}{Student(x) \Rightarrow \neg HatArbeit(x)}, \frac{:Erwachsen(x) \wedge \neg Student(x) \Rightarrow HatArbeit(x)}{Erwachsen(x) \Rightarrow HatArbeit(x)}, \frac{:Student(x) \Rightarrow Erwachsen(x)}{Student(x) \Rightarrow Erwachsen(x)}$$

Sie lassen sich transformieren in eine Hypothesen–Theorie:

Hypothesen:	$h1(x) : Student(x) \Rightarrow \neg HatArbeit(x)$
	$h2(x) : Erwachsen(x) \Rightarrow HatArbeit(x)$
	$h3(x) : Student(x) \Rightarrow Erwachsen(x)$
Fakt:	$Student(x) \Rightarrow \neg h2(x)$

Wenn wir wissen, daß Studenten, obwohl sie erwachsen sind, keine Arbeit haben, ist es möglich die Hypothese $h2(x)$ zu blockieren. Durch die Benennung können wir explizit $h2(x)$ ausschließen, wenn $Student(x)$ gilt.

Wir werden im folgenden nur noch benannte Hypothesen verwenden, um eine einheitliche Formulierung der Hypothesen zur ermöglichen.

Erklärungen sind als Formelmengen definiert, aus denen die erklärbare Formel abzuleiten ist. Als abkürzende Schreibweise für Erklärungen von Formeln wollen wir nur noch die instanziierten Hypothesen verwenden. Die Fakten werden wir in den Erklärungen nicht mehr explizit nennen; sie sollen aber implizit enthalten sein.

4.2.5 Constraints

Wie bereits angedeutet, ist die Transformation semi-normaler Defaults in eine Hypothese und einen Fakt nicht korrekt. Problematisch ist hier die Implikation zur Verhinderung der Default-Anwendung. Diese Implikation kann mittels Kontraposition auch rückwärts angewendet werden.

Sei $\Delta = (\{p : \omega\}; \{\neg\beta \Rightarrow \neg p\})$, dann ist β mit $\{p\}$ erklärbar, denn $\neg\beta \Rightarrow \neg p$ gdw $p \Rightarrow \beta$.

Durch die Einführung des Faktums $\neg\beta \Rightarrow \neg p$ resultieren also zwei Wirkungen:

1. Blockierung der Hypothese p, wenn $\neg\beta$ bekannt ist.
2. Herleitung von β durch Kontraposition, wenn die Hypothese p angenommen wird.

[Poole88] erweitert seine Hypothesen–Theorie nun so, daß einerseits die erwünschte Blockierung von Hypothesen möglich bleibt und andererseits die Herleitung von unerwünschten Resultaten verhindert wird.

Neben der Hypothesen– und Faktenmenge gibt es zusätzlich eine Menge von Constraints. Die Aufgabe der Constraints ist es nun, Bedingungen zu spezifizieren und zu garantieren, ohne damit Ableitungen bzw. neue Erklärungen zu generieren.

Definition 4.2.5 (Hypothesen–Theorie mit Constraints)
Sei $\mathcal{L}$ eine prädikatenlogische Sprache, $H \subseteq \mathcal{L}$ und $W, C \subseteq \mathcal{L}_0$.
$\Delta = (H, W, C)$ heißt Hypothesen–Theorie mit Constraints.
C heißt Menge von Constraints.

Der Begriff der Unter–Theorie muß um eine Konsistenzbedingung, die die Constraints berücksichtigt, erweitert werden. Dabei bleibt die Menge der Formeln in der Unter–Theorie unverändert.

Definition 4.2.6 (Unter–Theorie)
Sei $\Delta = (H, W, C)$ Hypothesen–Theorie mit Constraints.
$S \subseteq \mathcal{L}_o$ ist eine Unter–Theorie *von Δ gdw*

1. *$S = W \cup H_0$ mit $H_0 \subseteq G(H)$.*

2. *$S \cup C$ ist konsistent oder $W \cup C$ ist bereits inkonsistent.*

Die Definitionen von Erklärung und Extension gelten analog. Der Satz 4.2.2 über den Zusammenhang zwischen Extension und Erklärung gilt auch für Hypothesen–Theorien mit Constraints und der Beweis dazu läßt sich direkt übertragen.

4.2.6 Implementierung

[PooleGoebelAleliunas87] beschreiben eine Implementierung für die Hypothesen–Theorie mit Constraints. Das Grundprinzip ihres Beweisers THEORIST ist vergleichbar mit der Top-Down-Idee des Default-Beweisers nach Reiter. Es werden Instanziierungen der Elemente der Hypothesenmenge gesucht, mit denen, zusammen mit der Faktenmenge, eine gewünschte Erklärung einer Formel möglich ist.

Als unentscheidbares Problem verbleibt weiterhin der Konsistenztest der Unter–Theorie. Doch ist nach den Autoren der Konsistenztest für viele interessante Problemstellungen entscheidbar. Insbesondere für die Klasse der aussagenlogischen Formeln, die im zweiten Teil dieser Arbeit eine zentrale Rolle spielen, gibt es polynomiale Konsistenztests.

4.3 Default–Theorie mit Prioritäten

4.3.1 Wozu benötigt man eine Gewichtung der Hypothesen?

[Brewka89] verfeinert die Hypothesen–Theorie um eine Ordnung zur Kontrolle der Interaktion zwischen Defaults. Mit der Hypothesen–Theorie mit Constraints ist es möglich, die Anwendung bzw. Benutzung von Hypothesen zu steuern, indem z.B. die Verwendung einer Hypothese unter bestimmten Bedingungen verhindert wird. Es kann aber keine Gewichtung der Hypothesen spezifiziert werden, um auszudrücken, daß bei konkurrierenden Hypothesen eine zu bevorzugen ist. In vielen Fällen weiß man aber, welche Hypothese zu benutzen ist, wenn zwei Hypothesen anwendbar sind, die zu einer Inkonsistenz führen. Dazu ein leicht modifiziertes Beispiel von Ulrich Junker:

Beispiel 4.3.1
Folgende Situation sei gegeben:

- *Normalerweise müssen Hiwi's Donnerstags zur Besprechung kommen.*
- *Eine Ausnahme von dieser Regel ist, wenn jemand krank ist, außer wenn es sich „nur“ um einen Schnupfen handelt.*
- *Außerdem fällt die Besprechung aus, wenn der Projektleiter in Urlaub ist.*

Mit semi-normalen Defaults kann man den gegebenen Sachverhalt einfach ausdrücken. $Ausnahme_1$ sei ein Prädikat zur Spezifikation von Ausnahmen.

- $\frac{:\neg Ausnahme_1 \wedge Besprechung}{Besprechung}$ *(normalerweise finden Besprechungen statt)*
- $\frac{Krank : \neg Erkältet \wedge Ausnahme_1}{Ausnahme_1}$ *(bei Krankheit, außer bei Erkältungen, handelt es sich um eine Ausnahme)*

- $Urlaub \Rightarrow Ausnahme_1$ *(Urlaub ist immer eine Ausnahme)*
- $Erkältet \Rightarrow Krank$ *(Erkältungen sind Krankheiten)*

Zur Transformation in eine Hypothesen–Theorie benötigt man zwei Hypothesen:

(H1) $p1 : Besprechung$ *(Besprechung ist eine Annahme)*

(H2) $p2 : Erkältet \Rightarrow Besprechung$ *(Bei Erkältung ist normalerweise auch Besprechung)*

und die Fakten

(F1) $Urlaub \Rightarrow \neg p1$ *(Im Urlaub keine Besprechung)*

(F2) $Krank \Rightarrow \neg p1$ *(Bei Krankheit keine Besprechung)*

(F3) $Urlaub \Rightarrow \neg p2$ *(Im Urlaub auch keine Besprechung bei Erkältung)*

(F4) $Erkältung \Rightarrow Krank$ *(Erkältungen sind Krankheiten)*

(H2) ist eine unschöne Hypothese, denn man hat bereits in (H1) angegeben, daß normalerweise Besprechungen sind, aber man muß hier nochmals spezifizieren, daß bei Erkältungen Besprechungen sind. Außerdem muß die Urlaubsausnahme (F1) bzw. (F3) für beide Hypothesen explizit angegeben werden. Diese Vervielfachung von Formeln und Abhängigkeiten ist in komplexen Situationen und bei verschachtelten Annahmen wesentlich umfangreicher, wie man sich leicht überlegen kann.

Eine andere mögliche Modellierung der Situation, nämlich (F2) als Hypothese (H3) $p3 : Krank \Rightarrow \neg p1$ *und den Fakt (F5)* $Erkältet \Rightarrow \neg p3$ *aufzunehmen, führt in der Situation das nur* $Krank$ *bekannt ist zu zwei Extensionen. Eine Extension ist die gewünschte und erwartete. Die zweite Extension enthält die Hypothese* $p1$ *und* $\neg p3$*, also trotz Krankheit findet die Besprechung statt.*

Ursache für dieses Problem ist die Unmöglichkeit, mit einer Hypothese eine andere Hypothese zu blockieren. Dies beinhaltet die Problematik, Präferenzen zwischen Hypothesen anzugeben. Die Idee, mit Hilfe der Constraints eine Bevorzugung einer Hypothese auszudrücken, scheitert wiederum an der Kontraposition. Möchte man z.B. Hypothese (H3) gegenüber (H1) mit dem Constraint $p3 \Rightarrow \neg p1$ bevorzugen, gilt die äquivalente Formel $p1 \Rightarrow \neg p3$ ebenfalls. Bevorzugung durch einen Fakt oder eine weitere Hypothese führt zu den gleichen Resultaten.

Durch Prioritäten führt man eine Beschreibungssprache ein, die über Prädikate der zugrundeliegenden Logik Beziehungen definiert. Damit konstruiert man eine Art Meta-Sprache zur prädikatenlogischen Sprache. Durch diese Meta-Sprache werden die Prädikate zwar nicht in ihrer Bedeutung geändert, aber sie bekommen in Zusammenhang mit den übrigen Prädikaten einen neuen Stellenwert.

[Brewka89] beschreibt zwei Ansätze, um Reihenfolgen bzw. Prioritäten zwischen Defaults festzulegen: (1) total geordnete Default–Theorien und (2) partiell geordnete Default–Theorien.

4.3.2 Total geordnete Default–Theorie

Poole verwendet zwei Mengen von Formeln: Fakten und Hypothesen. In der total geordneten Default–Theorie werden Formeln nicht mehr nur in zwei Ebenen klassifiziert, sondern eine beliebige Anzahl von Ebenen ist möglich. Die Formeln jeder Ebene sind dabei potentielle Hypothesen.

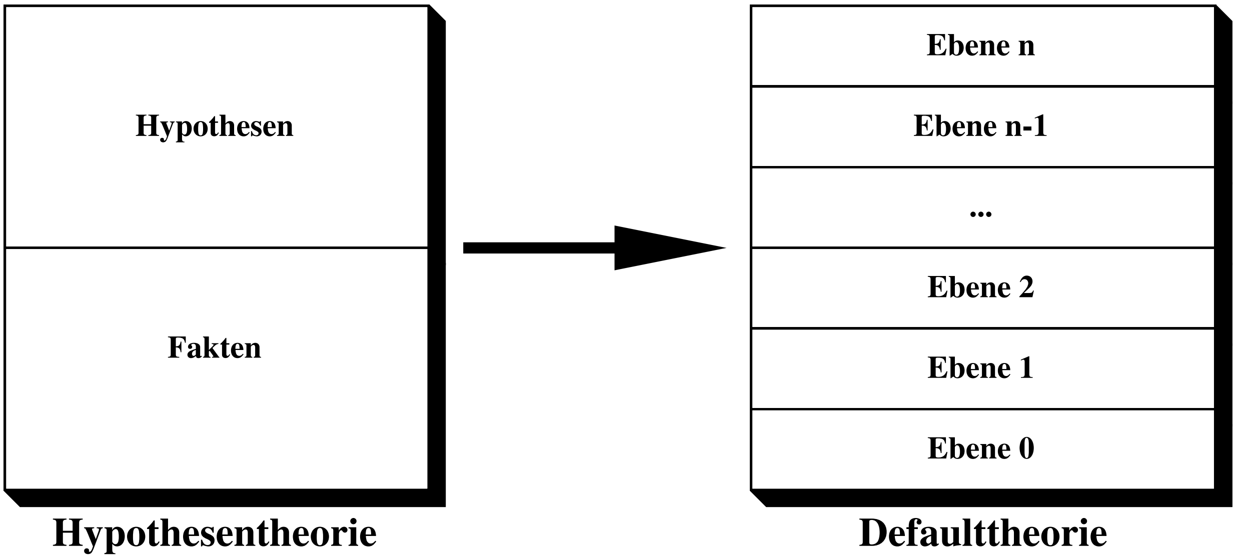

Abbildung 4.1: Die Erweiterung zur total geordneten Default–Theorie

Diese Ebenen sollen eine Bevorzugung von Formeln aus verschiedenen Ebenen ausdrücken. Hypothesen einer tieferen Ebene sollen gegenüber Formeln höherer Ebenen bevorzugt werden. Man kann auch sagen, daß Formeln einer unteren Ebene zuverlässiger sind, als solche einer darüberliegenden Ebene.

Basierend auf diesen Ebenen wird eine Theorie konstruiert, die maximal konsistente Formelmengen jeder Ebene enthält.

Definition 4.3.1 (total geordnete Default–Theorie)
Sei $\mathcal{T} = \{T_i \mid i \in I \subseteq NAT, T_i \subseteq \mathcal{L}\}$ *und* $\lhd$ *eine totale, strikte, wohlfundierte Ordnung auf* $\mathcal{T}$*.*
Das Tupel $\Delta = (\mathcal{T}, \lhd)$ *heißt total geordnete Default–Theorie.*

Die Menge $\mathcal{T}$ wird im allgemeinen endlich sein. Wir werden in einem solchen Fall $\mathcal{T}$ als Folge $(T_1, ..., T_n)$ beschreiben, mit der impliziten Ordnung $T_i \lhd T_j$ gdw $i < j$. Entsprechend zur Unter–Theorie definiert man bevorzugte Unter–Theorien, als Vereinigung der Teilmengen der Formelmengen T_i, so daß diese Vereinigung konsistent ist und Formeln aus kleineren Mengen (bzgl. $\lhd$) bevorzugt werden.

Definition 4.3.2 (bevorzugte Unter–Theorie)
Sei $\Delta = (\mathcal{T}, \lhd)$ *eine total geordnete Default–Theorie, mit* $\mathcal{T} = \{T_i \mid i \in I\}$*.*
$S \subseteq \mathcal{L}$ *heißt bevorzugte Unter–Theorie, wenn gilt*

1. $S = \bigcup_{i \in I} S_i$ *mit* $S_i \subseteq G(T_i)$*.*
2. $\forall T_k \in \mathcal{T}$. $\bigcup_{i \in J(T_k)} S_i$ *maximal, konsistente Menge, mit*
   $J(T_k) = \{i \mid T_i \lhd T_k\}$ *(Menge der Indizes von kleineren Mengen)*

Bemerkungen:

- Es gibt im Unterschied zur Definition 4.2.6 keine Menge von Fakten mehr, sondern prinzipiell ist jede Formel eine revidierbare Hypothese.
- Eigentlich handelt es sich hier um eine rekursive Definition zur Konstruktion einer Unter–Theorie. Man beginnt mit der kleinsten Menge T_{i_0} und bestimmt davon eine maximale und konsistente Teilmenge S_{i_0}. Dann nimmt man die nächstgrößere Menge T_{i_1} und bestimmt eine maximale Teilmenge S_{i_1} von T_{i_1}, so daß $S_{i_0} \cup S_{i_1}$ konsistent ist. Entsprechend fährt man mit den übrigen Elementen von $\mathcal{T}$ fort.
- Bevorzugte Unter–Theorien sind nicht eindeutig, denn die Bildung des Maximums muß nicht eindeutig sein.

Der folgende Satz soll den Zusammenhang zwischen der Unter–Theorie nach Poole und der bevorzugten Unter–Theorie nach Brewka verdeutlichen. Es stellt sich heraus, daß die Ebenen durch eine Iteration der Untertheoriebildung simuliert werden können. Die Unter–Theorie der letzten Iteration ist gerade die bevorzugte Unter–Theorie. Wir werden zur Vereinfachung der Formulierung eine endliche total geordnete Default–Theorie betrachten. Der Satz gilt aber auch für unendliche Mengen.

Satz 4.3.1
Sei $(T_1, ..., T_n)$ eine total geordnete Default–Theorie.
S ist eine bevorzugte Unter–Theorie gdw es eine Folge $\Delta_1 = (T_1, W_1), ..., \Delta_n = (T_n, W_n)$ von Hypothesen–Theorien gibt, mit

(B1) $W_1 = \emptyset$

(B2) $\forall 1 \leq i < n \,.\, W_{i+1}$ *ist maximale Unter–Theorie der Hypothesen–Theorie* $\Delta_i = (T_i, W_i)$

(B3) *S ist maximale Unter–Theorie der Hypothesen–Theorie* $\Delta_n = (T_n, W_n)$

Beweis:

„$\Rightarrow$“ Sei $S = S_1 \cup ... \cup S_n$
Behauptung: Die Folge $\Delta_1 = (T_1, \emptyset), \Delta_2 = (T_2, S_1), ..., \Delta_n = (T_n, S_1 \cup ... \cup S_{n-1})$ erfüllt die Bedingungen (B1) – (B3).
Beweis:

(B1) klar.

(B2) Sei $\hat{S}_k := \bigcup_{i=1}^{k} S_i$, dann gilt $\hat{S}_{k+1} = \hat{S}_k \cup H_k$, wobei H_k eine Menge von Grundinstanzen von T_k ist. Außerdem ist $\hat{S}_{k+1}$ konsistent und maximal, also ist $\hat{S}_{k+1}$ maximale Unter–Theorie von $(T_k, \hat{S}_k)$. Damit ist W_{k+1} maximale Unter–Theorie von (T_k, W_k).

(B3) Die Argumentation von (2) kann entsprechend übertragen werden.

„$\Leftarrow$“ Behauptung: $S = S_1 \cup ... \cup S_n$ ist bevorzugte Unter–Theorie wenn $S_1 = W_2, S_{i+1} = W_{i+2} \backslash W_{i+1}; S_n = S' \backslash W_n$ mit S' ist maximale Unter–Theorie von (T_n, W_n).

1. $S_i \subseteq G(T_i)$
2. $\bigcup_{i=1}^{k} S_i = W_{k+1}$ ist maximale Unter–Theorie von (T_i, W_i) also ist W_{k+1} maximal und konsistent. □

Die mit Hilfe der bevorzugten Unter–Theorie ableitbaren Formeln, sollen wiederum Extensionen heißen.

Definition 4.3.3 (Extension)
Sei $\Delta = (\mathcal{T}, \lhd)$ total geordnete Default–Theorie. $E \subseteq \mathcal{L}_0$ heißt Extension von Δ gdw eine bevorzugte Unter–Theorie S existiert mit $E = Th(S)$.

Die Menge aller Extensionen von Δ bezeichnen wir mit $\mathcal{E}(\Delta)$

Zwei Beispiele sollen die Eleganz und Einfachheit der Darstellung mit Hilfe der Ebenen zeigen:

Beispiel 4.3.2 (Tweety)
Wir haben wieder einmal die bekannten Aussagen über Tweety und zwei weitere Vögel Tim und Fred. Es gelte: Tweety ist ein Vogel und Tim und Fred sind Pinguine, wobei Tim trotzdem fliegen kann. Wir benötigen drei Ebenen:

$T_1 = \{Vogel(Tweety); Pinguin(Tim); Pinguin(Fred); Fliegt(Tim); \forall x \,.\, Pinguin(x) \Rightarrow Vogel(x)\}$

$T_2 = \{Pinguin(x) \Rightarrow \neg Fliegt(x)\}$

$T_3 = \{Vogel(x) \Rightarrow Fliegt(x)\}$

T_1 enthält die sichersten Informationen. Die Menge T_2 enthält Informationen, die durch T_1 überschrieben werden können, z.B. gilt, obwohl Tim ein Pinguin ist und Pinguine normalerweise nicht fliegen (nach T_2), kann er fliegen (nach T_1). T_3 enthält die unsichersten Informationen und kann durch T_1 und T_2 blockiert werden.

Beispiel 4.3.3 (Nixon)
Die Frage, ob Nixon Pazifist ist oder nicht, läßt sich auf zwei Arten lösen.

1. *Wenn wir nicht wissen welche Information stärker zu gewichten ist, kann man das Problem mit zwei Ebenen modellieren:*

   $T_1 = \{Republikaner(Nixon), Quäker(Nixon)\}$

   $T_2 = \{Republikaner(x) \Rightarrow \neg Pazifist(x), Quäker(x) \Rightarrow Pazifist(x)\}$

   Damit erhält man zwei mögliche Extensionen, nämlich eine, die $Pazifist(Nixon)$ enthält und eine, die $\neg Pazifist(Nixon)$ enthält.

2. *Ist aber bekannt, daß der Einfluß der Quäker gegenüber den Republikanern beim Pazifismus überwiegt, kann man mit drei Ebenen die unerwünschte Extension ausblenden.*

   $T_1 = \{Republikaner(Nixon), Quäker(Nixon)\}$

   $T_2 = \{Quäker(x) \Rightarrow Pazifist(x)\}$

   $T_3 = \{Republikaner(x) \Rightarrow \neg Pazifist(x)\}$

4.3.3 Partiell geordnete Default–Theorie

Ein Problem der total geordneten Default–Theorie stellt sich, wenn man eine Formel hat, die keiner Ebene direkt zuzuordnen ist. Sei im Tweety-Beispiel folgende Aussage zusätzlich bekannt:

„Genau dann wenn ein Tier Federn hat, kann es auch fliegen“

Als Formel ausgedrückt ergibt sich:

$(F)\ HatFedern(x) \Leftrightarrow Fliegt(x)$

Da wir nichts über den Zusammenhang von Federn und Pinguinen / Vögeln kennen und auch keine Präferenzen festlegen möchten, müßte man die Formel entweder in T_1 oder T_2 aufnehmen. Aber beides bereitet Probleme:

- Formel (F) in T_2 und weiterhin bekannt:

  $Vogel(Bill) \wedge HatFedern(Bill)$

  Gewünscht sind eigentlich zwei Extensionen, nämlich eine, in der Bill fliegen kann und eine, in der er es nicht kann. Aber durch die eigentlich zu vermeidende Präferenz der Aussage über Federn (F), gilt hier nur, daß Bill nicht fliegen kann.

- Analog ist die Argumentation, wenn (F) in T_3 ist. Dann hat die Default–Theorie mit der Aussage

  $Pinguin(John) \wedge HatFedern(John)$

  nur eine Extension, die $\neg Fliegt(John)$ enthält.

Diese Beobachtung führt zur Erweiterung der total geordneten Default–Theorie zu einer partiell geordneten Default–Theorie, bei der es keine Totalordnung zwischen den Ebenen gibt, sondern eine partielle Ordnung.

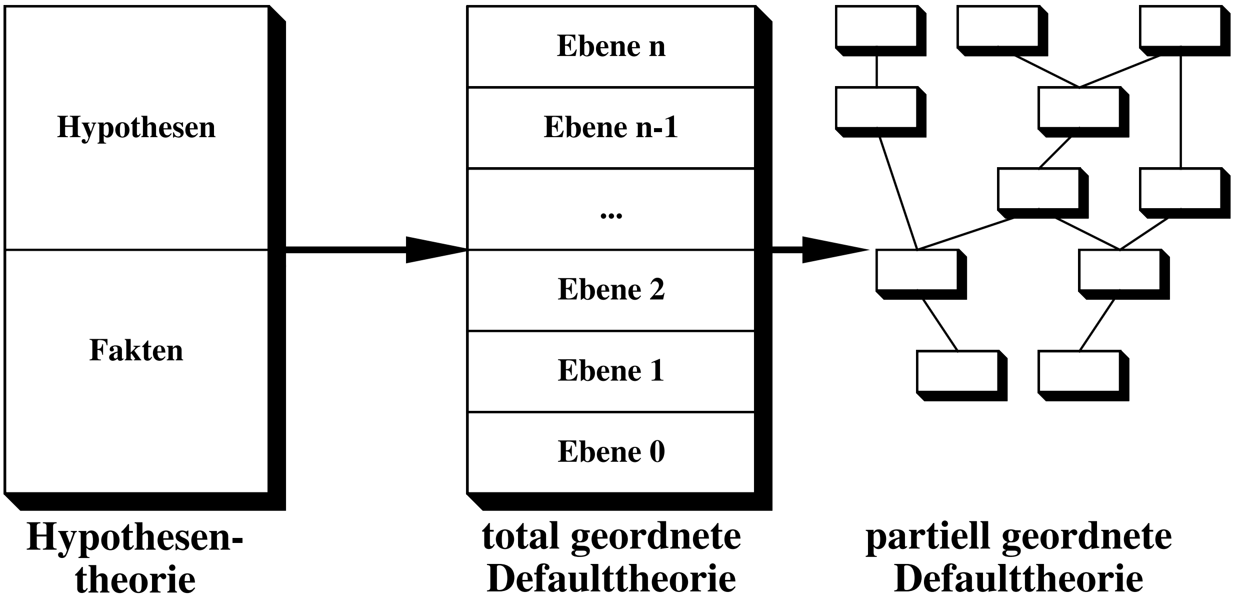

Abbildung 4.2: Die Erweiterung zur partiell geordneten Default–Theorie

Die Definition der partiell geordneten Default–Theorie kann analog zur total geordneten Default–Theorie durchgeführt werden. Der einzige Unterschied ist der, daß die Ordnung nicht mehr total sein muß, sondern nun genügt eine partielle Ordnung.

Definition 4.3.4 (partiell geordnete Default–Theorie)
Sei $\mathcal{T} = \{T_i \mid i \in I \subseteq NAT, T_i \subseteq \mathcal{L}\}$ und $\lhd$ ein partielle, strikte, wohlfundierte Ordnung auf $\mathcal{T}$. Das Tupel $\Delta = (\mathcal{T}, \lhd)$ heißt partiell geordnete Default–Theorie.

Bevorzugte Unter–Theorien zu definieren ist mit einer Partialordnung ein klein wenig schwieriger. Im Prinzip geht man so vor, daß aus der Partialordnung alle möglichen Totalordnungen konstruiert werden, die mit der Partialordnung verträglich sind. Dann kann man die Definition bevorzugter Unter–Theorien direkt übernehmen. Die Bedingung der Verträglichkeit von Partial- und Totalordnung ist

$F_1 \lhd_{partiell} F_2 \leadsto F_1 \lhd_{total} F_2$

Definition 4.3.5 (bevorzugte Unter–Theorie)
Sei $\Delta = (\mathcal{T}, \lhd)$ partiell geordnete Default–Theorie mit $\mathcal{T} = \{T_i \mid i \in I\}$.
$S \subseteq \mathcal{L}$ heißt bevorzugte Unter–Theorie gdw

(B1) *Es gibt eine Folge $(t_j \mid j \in J \subseteq NAT)$ mit*

1. $\forall i \in I \,.\, \forall t \in T_i \,.\, \exists j \in J \,.\, t = t_j$
2. $\forall j \in J \,.\, \exists i \in I \,.\, t_j \in T_i$
3. $\forall j_1, j_2 \in J \,.\, t_{j_1} \neq t_{j_2}$
4. $\forall t_j \in T_i \,.\, \forall t_{j'} \in T_{i'} \,.\, T_i \lhd T_{i'} \Rightarrow j \leq j'$

(B2) $S = \bigcup_{j \in J} S_j$ *mit* $S_j \subseteq G(\{t_j\})$

(B3) $\forall k \in J \,.\, \bigcup_{j \leq k} S_j$ *ist maximale, konsistente Menge.*

Diese Definition soll an einem Beispiel verdeutlicht werden.

Beispiel 4.3.4
Die partiell geordnete Default–Theorie $\Delta = (\mathcal{T}, \lhd)$ sei gegeben durch

$\mathcal{T} = \{T_1, T_2, T_3, T_4\}$ *mit*
$T_1 = \{A\}, T_2 = \{B\}, T_3 = \{\neg B, C\}, T_4 = \{\neg C\}$ *und der Ordnung*
$T_1 \lhd T_2; T_1 \lhd T_3; T_2 \lhd T_4$

Graphisch veranschaulicht wird die Situation in Bild 4.3.

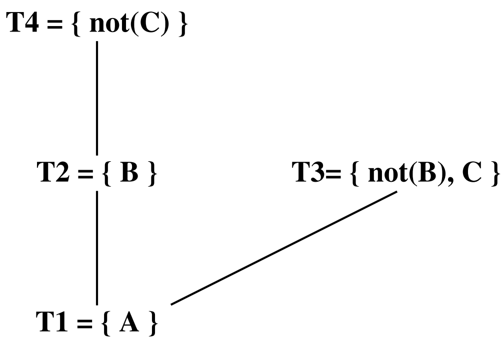

Abbildung 4.3: Ein Beispiel für eine bevorzugte Unter–Theorie

Es ergeben sich sechs mögliche Folgen von Formeln, die Bedingung (B1) der Definition genügen. Davon sind jedoch die Hälfte durch Vertauschung der unabhängigen Formeln in T_3 zu erzielen.

- $(A, \neg B, C, B, \neg C)$ *mit* $S = \{A, \neg B, C\}$ *als bevorzugte Unter–Theorie.*
- $(A, B, \neg B, C, \neg C)$ *mit* $S = \{A, B, C\}$ *als bevorzugte Unter–Theorie.*
- $(A, B, \neg C, \neg B, C,)$ *mit* $S = \{A, B, \neg C\}$ *als bevorzugte Unter–Theorie.*

Einen Nachteil sehen wir an diesem Beispiel: die Zahl der Extensionen ist gegenüber der total geordneten Default–Theorie größer. Dieses Resultat war jedoch zu erwarten, da wir weniger Information über T_3 haben und damit T_3 in der totalen Ordnung beliebig oberhalb von T_1 einordnen können. Ein wichtiges Ziel bei der Beschreibung einer Situation mit Hilfe von partiell geordneten Default–Theorien sollte sein, möglichst viel an Information auszunutzen um wenige nicht vergleichbare Formelmengen zu erhalten und sich damit der Totalordnung weitgehend anzunähern. Zwei Ideen sollen zeigen, wie das erreicht werden kann.

1. **Formelmengen zusammenfassen**
   Weiß man zum Beispiel, daß es egal ist ob die Formelmenge T_i oder T_j bevorzugt wird, kann man einfach eine neue Formelmenge $T_i \cup T_j$ stattdessen benutzen.
2. **Formelmenge einordnen**
   Ist im obigen Beispiel die Gewichtung von T_3 im Verhältnis zu T_2 und/oder T_4 bekannt, z.B. T_3 ist zuverlässiger als T_2, dann kann man das Beispiel sogar in eine Totalordnung $T_1 \lhd T_2 \lhd T_3 \lhd T_4$ umwandeln.

Es gibt zwei für uns interessante Darstellungsweisen der partiellen Ordnung:

- Definition der Partialordnung zwischen Formeln oder
- Definition der Partialordnung zwischen Formelmengen

Beide Darstellungen sind durch eine einfache Transformation äquivalent ineinander zu transformieren. Äquivalenz bedeutet hier, daß die Menge der bevorzugten Unter–Theorien sich nicht ändert. Eine Menge von Formeln mit einer Partialordnung kann als partiell geordnete Default–Theorie $\Delta = (\mathcal{T}, \lhd)$ angesehen werden. Jedes Element von $\mathcal{T}$ entspricht einer ein-elementigen Formelmenge und die Ordnung $\lhd$ entsteht durch Übertragung der Ordnung zwischen den Formeln.

Transformation 4.3.1
Sei $\mathcal{F} \subseteq \mathcal{L}$ *eine Menge von Formeln und* $<$ *eine partielle, strikte Ordnung auf* $\mathcal{F}$.
$\Delta = (\mathcal{F}, <)$ *heißt ebenfalls partiell geordnete Default–Theorie und steht als Synonym für* $\Delta' = (\mathcal{T}, \lhd)$ *mit*

$\mathcal{T} = \{\{w\} \mid w \in \mathcal{F}\}$ *und* $\lhd$ *ist definiert als*

$\forall w_1, w_2 \in \mathcal{F} \;.\; \{w_1\} \lhd \{w_2\}$ *gdw* $w_1 < w_2$

Umgekehrt kann man jede Partialordnung zwischen Mengen von Formeln in eine Partialordnung zwischen Formeln transformieren.

Transformation 4.3.2
Sei $\Delta = (\mathcal{T}, \lhd)$ *eine partiell geordnete Default–Theorie.*
Δ *wird transformiert in* $(\mathcal{F}, <)$ *mit*

$\mathcal{F} = \bigcup_{T \in \mathcal{T}} T$ *und* $<$ *ist definiert als*

$\forall w_1, w_2 \in \mathcal{F} \;.\; w_1 < w_2$ *gdw* $\exists T_1, T_2 \in \mathcal{T} \;.\; w_1 \in T_1 \land w_2 \in T_2 \land T_1 \lhd T_2$

Anschaulich bedeutet diese Transformation: wenn $T_1 \lhd T_2$, dann stehen alle Formeln aus T_1 in der $<$–Relation zu allen Formeln aus T_2. Formeln innerhalb einer Menge sind unvergleichbar (siehe Bild 4.4).

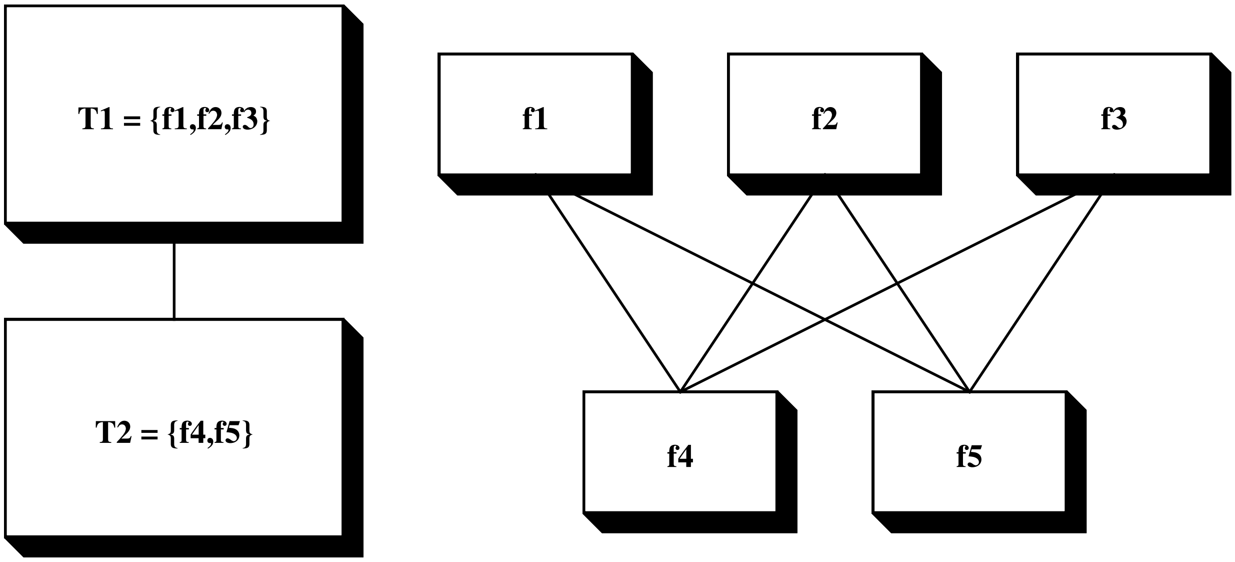

Abbildung 4.4: Beispiel für die Transformation der Ordnung

Kapitel 5

Fallbasiertes Schließen

Nachdem in den bisherigen Kapiteln die Nichtmontonie ausführlich eingeführt wurde, soll im folgenden das fallbasierte Schließen mit seinen Grundprinzipien dargestellt werden. Wir werden uns auf die Hauptaspekte beschränken und nur die für die weiteren Kapitel wichtigen Teilgebiete genauer erläutern. Der interessierte Leser sei auf die Proceedings der Workshops zu Case–Based Reasoning [DARPA87, DARPA88, DARPA89] verwiesen.

5.1 Einleitung

Nach [Slade91] hat die künstliche Intelligenz zwei grundsätzliche Forschungsschwerpunkte:

1. Verstehen des menschlichen Denkens und Problemlösens und

2. Entwicklung von Maschinen bzw. Programmen zur Realisierung intelligenten Verhaltens.

Beide Punkte sind im Paradigma des Case–Based Reasoning (CBR) zu finden, bei dem es darum geht, sich an bekannte Problemlösungen zu erinnern und diese auf neue Probleme anzuwenden. Einerseits ist das Lösen von Problemen durch den Menschen oft mit einem Rückgriff auf Erfahrungswissen verbunden. Andererseits entwickelt man Techniken, Repräsentationssprachen und Algorithmen, um mit Hilfe des Computers Erfahrungen zu verwalten und diese sinnvoll auf neue Probleme anzuwenden.

Grundlage des Case–Based Reasoning für den ersten Punkt ist die Beobachtung von Psychologen, daß sich menschliche Experten insbesondere in den Bereichen *Jura, Mathematik und strategische Planungen* oft auf eigene Erfahrungen beziehen. Diese Erfahrungen werden für die Lösung neuer Probleme modifiziert und angepaßt. Wichtiger für uns ist jedoch der zweite Punkt, in dessen Mittelpunkt die Frage steht, wie Erfahrungen in einem Computer gespeichert und organisiert werden können und wie darauf aufbauende Inferenzsysteme zu realisieren sind.

Fallbasiertes Schließen enthält als Grundprinzip die Speicherung von Fällen in einer sogenannten Fallbasis (engl. case memory). Fälle sind Beschreibungen eines Sachverhaltes zusammen mit zusätzlichen Informationen. Der Inferenzprozeß hat als Eingabe eine unvollständige Fallbeschreibung und soll mit Hilfe der Fallbasis mehr Informationen über diesen Fall bereitstellen. Dies geschieht durch die Bestimmung eines ähnlichen Falles und die Übertragung bzw. die Anpassung des darin enthaltenen relevanten Wissens.

Ein zentraler Aspekt fallbasierten Schließens ist das Ziel die Wissensakquisition zu automatisieren: Ein CBR–System soll dazu in der Lage sein neue Fälle selbständig zu integrieren und diese für weitere Inferenzen zu nutzen. Case–Based Reasoning ist damit als Lernverfahren anzusehen, bei dem neue Fälle zu einer Verbesserung des Systemverhaltens führen.

Die Speicherung von Fällen ist ein komplexes Problem, insbesondere dadurch, daß man den Suchaufwand bei großen Fallbasen optimieren möchte. Um einen effizienten Zugriff auf relevante Fälle zu erreichen, benötigt man eine clevere Speicherorganisation. Diese Speicherorganisation muß außerdem so beschaffen sein, daß es einfach ist, neue Fälle zu integrieren und, wenn es nötig wird, den Inhalt zu reorganisieren.

5.2 Ablaufmodell

Das Prozeßmodell für Case–Based Reasoning umfaßt nach [RisslandKolodnerWaltz89] sechs Phasen:

- **Lokalisierung relevanter Fälle**
  Eine Menge von potentiellen Kandidaten für den „besten" Fall wird zur Reduzierung des Aufwandes für den folgenden Schritt ausgewählt. Diese geschieht im allgemeinen durch die Bestimmung relevanter Merkmale.

- **Auswahl des besten Falles**
  Es wird ein Fall oder auch eine (kleine) Menge von Fällen gesucht, mit dessen oder mit deren Hilfe das vorliegende Problem am besten zu lösen ist. Diese Auswahl basiert auf einem Ähnlichkeitsbegriff, der ausführlich in [Wess90] und in [Tversky77] dargestellt ist.

- **Übertragung des Falles**
  Die Informationen aus dem gefundenen Fall werden auf die Problemstellung übertragen. [Wess90] beschreibt diese Transformation in Anlehnung an die Abstraktionsabbildung nach [Richter89] als die Suche nach einer Funktion g' im Modell nach Abbildung 5.1, um aus einem Problem P_{neu} eine Lösung L_{neu} zu berechnen.

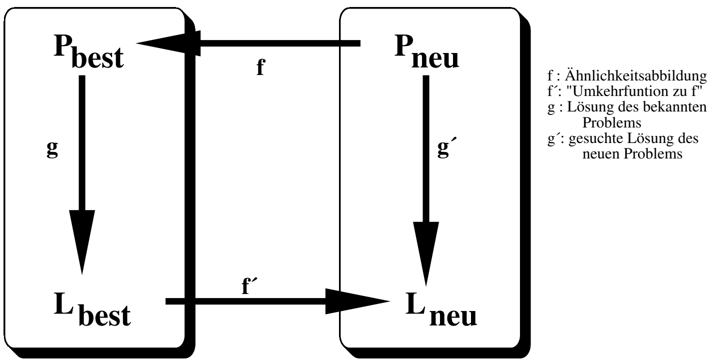

Abbildung 5.1: Abstraktionsabbildung

  g' entsteht aus f, g, f' durch eine Hintereinanderausführung:
  $g'(P_{neu}) = f'(g(f(P_{neu})))$ oder $g' = f' \circ g \circ f$

- **interne Überprüfung**
  In diesem Schritt wird die Übertragung durch eine interne Kontrolle getestet. Zum Beispiel indem man den Lösungsweg simuliert und die erhaltenen Ergebnisse mit den gewünschten vergleicht oder indem man wiederum gespeicherte Fälle zur Kontrolle benutzt.

- **externe Überprüfung**
  Durch Feedback, zum Beispiel durch den Benutzer oder als Resultat einer Aktion, findet eine zweite Überprüfung, diesmal aber in der Realität, statt. Das Feedback kann sowohl positiver als auch negativer Natur sein. Insbesondere negative Antworten erfordern eine Analyse der Ursachen für den produzierten Fehler.

- **Reorganisation des Speichers**
  Aufgrund der Ergebnisse des letzten Schrittes ist es nötig, den Speicher zu verändern. Diese Phase entspricht einem Lernschritt, da ein positives Feedback zu einer Speicherung des neuen Lösungsweges führt, der damit für weitere Aufgaben zur Verfügung steht. Entsprechend werden negative Feedbacks integriert, um gemachte Fehler in Zukunft zu vermeiden.

Im folgenden werden vor allem die ersten beiden Phasen untersucht. Die uns dabei vor allem interessierenden Aspekte sind die Repräsentation und die Fallauswahl.

5.3 Erinnern und Retrieval

In diesem Abschnitt werden einige Prinzipien des menschlichen Erinnerns nach [Schank82] dargestellt. Diese Prinzipien sollen bei der Entwicklung einer Speicherstruktur berücksichtigt werden. Wünschenswert wäre insbesondere eine Entsprechung zwischen dem Zugriff auf den Speicher, genannt Retrieval und dem Erinnern.

Das im folgenden beschriebene Modell des Gedächtnisses erhebt keinen Anspruch auf universelle Gültigkeit. Es bietet jedoch für viele Aspekte des Erinnerns adäquate Erklärungen und Ansätze zum Entwurf von Computermodellen.

Die erste Frage, die sich stellt ist: was bedeutet Erinnern?

Das Gedächtnis enthält sehr viele Informationen, die in irgendeiner Art und Weise kodiert sind. Erinnern beinhaltet, daß Ereignisse, Tatsachen, Sachverhalte, Begriffe, und ähnliches explizit gemacht werden. Die verschiedenen Arten von Informationen werden im folgenden unter dem Sammelbegriff „Ereignis“ zusammengefaßt.

Schank unterscheidet drei Arten von Erinnern:

- **Verarbeitungsbasiertes Erinnern (engl. processing based reminding)**
  Diese Art von Erinnern passiert während der Verarbeitung von Wissen. Beim Verstehen eines Eingabestromes werden in natürlicher Weise bereits gespeicherte Erfahrungen herangezogen und dabei teilweise als Erinnerungen explizit gemacht. Das verarbeitungsbasierte Erinnern wird im folgenden genauer untersucht.

- **Lexikonbasiertes Erinnern (engl. dictionary based reminding)**
  Hiermit ist der Effekt gemeint, daß bestimmte Worte oder Sätze oft ganze Episoden oder Wissenszusammenhänge assoziieren. Dieser Effekt entspricht dem Nachschlagen eines Begriffs in einem Lexikon.

- **Visuelles Erinnern (engl. visually based reminding)**
  Das Grundprinzip dieser Art von Erinnerung ist der Match eines beobachteten Musters mit einem gespeicherten. Durch diesen Match wird das Erkennen ausgelöst, z.B. kann ein Gesicht durch seine Ähnlichkeit an eine andere Person erinnern.

Es gibt zwei Ursachen, die das Erinnern veranlassen:

1. Die gezielte Suche nach einem bestimmten Ereignis.

2. Das zufällige Finden bzw. die zufällige Assoziation eines Ereignisses in einem anderen Zusammenhang.

Beide Ursachen geben Anlaß zu Fragestellungen über die Struktur des Gedächtnisses:

1. Woher wissen wir, daß ein bestimmtes Ereignis nach dem wir suchen überhaupt existiert und wie wird die Suche gesteuert?

2. Wieso kann ein „zufälliges" Erinnern stattfinden?

Eine Antwort auf diese Fragen liefert die Sichtweise der Erinnerung als Prozeß allmählicher Annäherung.

Im Prinzip läuft beim Erinnern und beim Lernen der gleiche Prozeß ab. Es wird ein Ereignis bzw. eine Anregung (Input) schrittweise in seinen verschiedenen Aspekten abgearbeitet. Dabei findet, ausgehend von allgemeinen Konzepten, eine sukzessive Spezialisierung von Konzepten statt. Im Laufe dieses Prozesses können „spontane" Erinnerungen auftreten. Diese können durch gespeicherte Ereignisse in den durchlaufenen Konzepten ausgelöst werden.

Bei der gezielten Suche wird dieser Prozeß der Spezialisierung solange fortgesetzt, bis das gewünschte Ereignis erreicht ist bzw. bis man glaubt es gefunden zu haben. Der Lernprozeß wird entsprechend neue Informationen möglichst in der Nähe zu gespeicherten ähnlichen Informationen ablegen und deshalb den gleichen Suchprozeß zur Lokalisierung durchlaufen.

Der beschriebene Prozeß der Suche ist nicht so eindeutig, wie man nach der Beschreibung vermuten könnte, sondern es gibt im allgemeinen viele mögliche Verarbeitungsreihenfolgen und -wege. Je reichhaltiger und vielfältiger der Speicher ist, desto einfacher ist das Erinnern, da viele Wege dann zum gewünschten Ereignis führen. Zum Beispiel ist es bekannt, daß man leichter lernt, wenn bereits ein Vorwissen bzw. ähnliches Wissen vorhanden ist. Es befinden sich dann bereits Zugriffspfade im Speicher, die weiter spezialisiert werden können.

Der Wunsch nach einer möglichst umfangreichen Zugriffsstruktur wird relativiert von der Forderung nach Effizienz und der Beschränkung des Speicherplatzes. Durch sehr viele Zugriffsmöglichkeiten wird insbesondere beim Lernen der Aufwand sehr hoch. Aber auch die Suche wird komplizierter, da nicht nur die Menge der erfolgreichen Zugriffe wächst, sondern proportional dazu wachsen auch die fehlerhaften Zugriffe.

Eine genauere Analyse des verarbeitungsbasierten Erinnerns führt zu verschiedenen Typen als Auslöser von Erinnerung.

- **Erwartung von Folgeereignissen**
  Während der Verarbeitung von Wissen ist es oft der Fall, daß man bestimmte Erwartungen hat, was noch geschehen wird. Entsprechend helfen Erwartungen auch beim Erinnern: man weiß oft keine genauen Details zu einem bestimmten Ereignis, aber trotzdem hat man gewisse Vorstellungen darüber, was passiert sein müßte. Diese Vermutungen können Anlaß zu einer gezielten Suche sein.

- **Zielbasiertes Erinnern**
  Folgende Beobachtung begründet diesen Typ von Erinnern: Ereignisse deren Merkmale in keinem direkten Zusammenhang stehen, können zu gegenseitiger Erinnerung führen, wenn die Ziele, Gründe oder Ursachen der Ereignisse vergleichbar sind.

- **Planbasiertes Erinnern**
  Ebenso wie die Ziele von Aktionen wechselseitiges Erinnern ermöglichen, können auch die Pläne, die Aktionen zugrundeliegen, vergleichbar sein und damit Assoziationen auslösen.

- **Erinnern durch zentrale Aspekte**
  Der Kern, der Witz oder die Moral eines Ereignisses sind ebenfalls Indizes, die zu weiteren Ereignissen führen können.

- **Erinnern über multiple Kontexte**
  Ereignisse können nicht nur wie bisher beschrieben aus der Verarbeitung eines Eingabestromes erinnert werden, sondern es ist auch möglich mit Hilfe verschiedener Kontexte ein Ereignis zu rekapitulieren.

  Ein kleines psychologisches Experiment soll erläutern, wie man auf die Idee kommt, daß es sehr viele Verbindungen zwischen eigentlich ganz verschiedenen Konzepten gibt.

  > Man stelle einer Testperson einige Fragen, bei denen die Antwort die Farbe *weiß* ist. Der Effekt sollte sein, daß sich der Proband auf das Konzept „weiße Farbe“ einstellt. Als Beispiel sei folgender typischer Dialog genannt:
  >
  > | | |
  > |---|---|
  > | F: Welche Farbe hat der Schnee? | A: Weiß. |
  > | F: Auf dem Klavier gibt es schwarze und ...? | A: Weiße Tasten. |
  > | F: Welche Farbe hat normaler Zucker? | A: Weiß. u.s.w. |
  >
  > Die letzte Frage lautet dann
  >
  > F: Was trinken Kühe?
  >
  > Und die Antwort lautet dann selbstverständlich
  >
  > A: Milch.

  Ein Ergebnis dieses Versuches ist, daß es durch die Aktivierung der Konzepte „weiß“ und „Kuh“ zu einer Aktivierung des Konzeptes „Milch“ kommt. Eine Ursache dieses Verhaltens können viele Verbindungen und Relationen zwischen Konzepten sein, die nicht alle explizit in einem Beispiel vorkommen, sondern u.U. selbständig generiert wurden. Übrigens trinken Kühe natürlich keine Milch! Die richtige Antwort „Wasser“ wird tatsächlich gegeben, wenn die Versuchsperson konkrete Erfahrungen mit Kühen besitzen.

5.4 Wissensrepräsentation

Ein zentrales Problem fallbasierten Schließens ist die effiziente Speicherung und Organisation der Fälle. Eine Grundlage von Speichermodellen für komplexe Zusammenhänge bilden semantische Netze. Nach [Richter89] dienen semantische Netze der Veranschaulichung prädikatenlogischer Formeln. Sie bestehen aus einem gerichteten Graphen, in dem die Knoten Objekte und die Kanten Relationen zwischen Objekten ausdrücken. Wichtig für uns ist, daß semantische Netze nur statisches Faktenwissen aufnehmen und ihr Aufbau bzw. ihre Generierung nicht festgelegt ist, sondern in gewisser Weise nur ad hoc geschieht. Damit ist es schwierig neues Wissen zu integrieren und semantische Netze automatisch zu reorganisieren. Semantische Netze wurden insbesondere in der Psychologie genauer untersucht.

Eine Weiterentwicklung bildet das Modell des Episodenspeichers (engl. episodic memory), in dem Episoden und Ereignisse eingetragen werden können. Neben solchen zeitlichen und räumlichen Daten, können auch zeitliche und räumliche Relationen repräsentiert werden, z.B. durch die Relationen „vorher“, „nachher“, „während“, „oberhalb“, „unterhalb“, „im gleichen Raum“.

Parallel zur psychologischen Modellbildung entwickelte auch die künstliche Intelligenz–Forschung Speichermodelle zur Repräsentation komplexen Wissens und komplexer Sachverhalte. Ursprung vieler Modelle bilden die Frames [Minsky75]. Frames bilden Informationseinheiten mit einer Vielzahl von Ausprägungen, die in sogenannten Slots eingetragen werden. Slots können verschiedenste Informationen enthalten, z.B. Zahlenwerte, Texte, Verweise auf andere Frames, Regeln, Prozeduren. Zusätzlich besitzen Slots Facetten zum Speichern von Default–Werten, datengesteuerten Prozeduren (Dämonen) und Bedingungen.

Neben Frames wurde in [SchankAbelson77] der Begriff des Scripts entwickelt. Anders als Frames, die im Zusammenhang mit Bildverarbeitungsproblemen entstanden, sollten mit Scripts primär prototypische Ereignisse, wie zum Beispiel ein Restaurantbesuch, dargestellt werden. Scripts unterscheiden sich syntaktisch nicht von Frames, haben jedoch eine spezielle kognitive Bedeutung. Scripts enthalten alle relevanten Informationen bezüglich eines Konzeptes. Das Script zum Restaurant–Besuch speichert beispielsweise Informationen darüber, wozu ein Restaurant dient und wie man eine Bestellung aufgibt. Außerdem werden mit solchen Scripts spezielle Ereignisse assoziiert, z.B. der Besuch eines Drei–Sterne–Restaurants zum Geburtstag oder ein Besuch bei McDonalds während des Einkaufs.

Die psychologische Deutung des Scripts als ein Modell des menschlichen Gedächtnisses, hat einen großen Mangel. Geht man von einer Organisation erlebter Ereignisse innerhalb eines Scripts aus, dann müßte es im Prinzip einfach sein, alle einem Script zugeordneten Ereignisse aufzuzählen. Jedoch ist der Mensch meistens dazu nicht in der Lage. Außerdem gibt es den Effekt, daß man durch ähnliche Ereignisse und Situationen an etwas erinnert wird, das dazu in keinem direkten Zusammenhang steht. Dieser Effekt ist im Scripten-Modell nicht zu erklären.

Diese Gründe führten neben anderen zum Entwurf eines weiter verfeinerten Speichermodells, nämlich des Dynamic Memory [Schank82]. Die Grundidee des Dynamic Memory ist eine Hierarchiebildung von Scripts, die hier als Konzepte betrachtet werden. Zusätzliches Merkmal ist eine Verknüpfung der Konzepte durch „Erinnerungsbrücken“, sogenannte Indizes. Außerdem ist bereits beim Entwurf berücksichtigt, daß der Speicher sich dynamisch ändern wird: neue Informationen sind integrierbar und eine Reorganisation des Speichers ist einfacher durchführbar.

Die Idee des Dynamic Memory wird in Kapitel 6 benutzt, um einen selbstorganisierenden Speicher zur Verwaltung von Fällen zu spezifizieren.

Kapitel 6

Der Fallspeicher

Da es unser Ziel ist eine Formalisierung des Fallspeichers mit Hilfe von Defaults zu realisieren, müssen wir nun genauer erläutern, wie der Fallspeicher aufgebaut ist. Zunächst wird der Begriff des Falles erklärt, um darauf aufbauend, eine konkrete Speicherstruktur zu entwickeln.

6.1 Fälle

Fälle sind die elementaren Informationseinheiten des fallbasierten Schließens. Ein Fall ist die Beschreibung eines Sachverhaltes mit einem räumlichen und zeitlichen Bezug. Einige wichtige Arten von Fällen sind:

- Ereignisse, z.B. historische Ereignisse oder Alltagserlebnisse
- beobachtetes Verhalten, z.B. das Ein-/Ausgabeverhalten von Maschinen
- Diagnosefälle, z.B. als Beschreibung fehlerhaften Maschinenverhaltens und der Fehlerursache
- Entscheidungen, z.B. Gerichtsurteile
- Pläne oder Teilpläne, z.B. zur Wiederverwendung, Vervollständigung oder Anpassung

Fälle liegen in sehr unterschiedlichen Formen vor, z.B. in natürlicher Sprache, als Protokoll, als Sensormeldung, als Diagramm und als Tabelle. Wichtig für die Anwendung von Algorithmen auf Fälle ist eine Transformation der Fälle in eine formale Sprache. Insbesondere prädikatenlogische Sprachen eigenen sich gut, da sie universell sind und eine formale logische Handhabung erlauben.

Viele Klassen von Fällen sind in Form von Mengen von Attribut–Wert–Paaren bzw. Attributvektoren beschreibbar. Als ein Beispiel seien hier die Diagnose–Fälle genannt. Diagnose–Fälle bestehen aus einer Menge von zu beobachtenden Eigenschaften mit konkreten Ausprägungen, den Werten. Ein ausgezeichnetes Attribut–Wert–Paar stellt die Diagnose dar.

Fälle in Form von Attribut–Wert–Paar lassen sich einfach in eine aussagenlogische Form transformieren.‘ Aus diesem Grund wird unsere Arbeitsgrundlage wird ein formaler Fallbegriff aufbauend auf Attribut–Wert–Paaren sein.

Definition 6.1.1 (Fall)
Sei A eine Menge von Attributen, V eine Menge von Werten.

$F \subseteq A \times V$ *heißt* Fall *gdw* $\forall (a, v), (a, v') \in F$ *gilt* $v = v'$

Man kann einen Fall auch als eine partielle Funktion von der Menge der Attribute in die Menge der Werte sehen, d.h. jedes Attribut hat höchstens einen Wert. Besitzt ein Fall für ein Attribut a keinen Wert, so heißt a *unbestimmt* oder *unbekannt*.

Die Transformation eines Falles in eine aussagenlogische Form besteht in der Umwandlung der Paare (a, v) in Gleichungen $a = v$ und deren konjunktive Verknüpfung.

Transformation 6.1.1 (Form I)
Sei $F \subseteq A \times V$ ein Fall. Die aussagenlogische Form (I) *von F ist:*

$$F \Leftrightarrow \bigwedge_{(a,v)\in F} a = v$$

Bei dieser Transformation werden die Attribute und Werte zu Objekten der Sprache und der Fall zum Prädikat über diesen Objekten.

Eine zweite mögliche Darstellung ist, ein dreistelliges Attribut, hier „*norm*“, über den Fällen, Attributen und Werten zu definieren. Damit werden dann die Fälle, die Attribute und die möglichen Werte zu Objekten der Sprache. Das Attribut „*norm*“ setzt diese Objekte zueinander in Beziehung.

Transformation 6.1.2 (Form II)
Sei $F \subseteq A \times V$ ein Fall. Die aussagenlogische Form (II) *von F ist:*

$$\bigwedge_{(a,v)\in F} norm(F, a, v)$$

Wir werden beide Formen je nach Situation verwenden:

- Form (I) ist besser geeignet, wenn wir über einzelne Fälle diskutieren, da sie leichter zu lesen ist.

- Form (II) ist dagegen zu bevorzugen, wenn eine Menge von Fällen untersucht wird, da dann Aussagen über verschiedene Fälle einfach durch eine Kombination der Teilaussagen erzeugt werden können. Zum Beispiel kann die Aussage: a hat den Wert v in Fall F oder F' einfach durch die Formel $norm(F, a, v) \vee norm(F', a, v)$ modelliert werden.

Eine weitere Darstellungsart ist möglich, wenn die Menge der Attribute endlich ist. Dann kann man einen Fall durch einen Attributvektor spezifizieren. Im Attributvektor ist jeder Position genau ein Attribut fest zugeordnet. Der Attributvektor enthält die Werte der einzelnen Attribute, wobei für unbekannte Attribute ein spezieller Wert (hier: UNBEKANNT) eingetragen wird.

Definition 6.1.2 (Attributvektor)
Sei $F \subseteq A \times V$ ein Fall und $A = \{a_1, a_2, ..., a_n\}$ die endliche Attributmenge, $UNBEKANNT \notin V$. $(v_1, v_2, ..., v_n)$ heißt Attributvektor von F, wenn gilt:

1. $\forall 1 \leq i \leq n \;.\; v_i \in V \cup \{UNBEKANNT\}$

2. $\forall 1 \leq i \leq n \;.\; (a_i, v_i) \in F \vee v_i = UNBEKANNT$

3. $\forall (a_i, v) \in F \;.\; v_i = v$

Die Schreibweise von Fällen als Attributvektor ist besonders von Vorteil, wenn wenige Attribute vorhanden sind und die Fälle wenige unbekannte Werte enthalten. Ungeeignet ist diese Schreibweise, wenn sich die Attribute dynamisch ändern oder die Attribute nicht a priori bekannt sind. Mit Attributvektoren erhält man im allgemeinen eine kompakte Darstellung, die sich gut zu einer tabellarischen Darstellung eignet.

Die Idee des Dynamic Memory aus Abschnitt 5.4 soll nun zur Speicherung von Fällen benutzt werden. Dazu werden zunächst Kriterien für einen solchen Speicher vorgestellt. Anschließend wird ein konkretes Modell und dessen Anwendung auf Fälle dargestellt.

6.2 Kriterien für die Speicherung

In [Kolodner83b] werden einige Prinzipien genannt, die ein Modell für die Wissensrepräsentation erfüllen muß. Diese Prinzipien werden motiviert durch den Wunsch zur Nachbildung des menschlichen Gedächtnisses.

- **Der Speicher ist rekonstruktiv**
  Psychologisch kann man begründen, daß Fälle nicht als Ganzes, vollständig im Gedächtnis vorliegen. Fälle können nicht mit allen Einzelheiten einfach abgerufen werden. Ein Fall ist durch ein gewisses Grundgerüst von Informationen beschrieben. Aus diesem Grundgerüst können durch Rekonstruktion fehlende Informationen generiert werden. Ein Indiz für diese Rekonstruktionsannahme ist, daß viele Einzelheiten von Fällen oft gar nicht oder fehlerhaft erinnert werden.

- **Der Speicherinhalt ist nicht aufzählbar**
  Das Argument für diesen Punkt ist die Unfähigkeit, alle Fälle einer bestimmten Art, z.B. alle Restaurantbesuche, aufzählen zu können. Im allgemeinen wird man sich an, durch verschiedene besondere Umstände bestimmte, Fälle erinnern, aber nicht unbedingt an alle.

- **Der Speicherzugriff erfolgt durch einen Annäherungsprozeß**
  Ein solcher Prozeß entspricht der Vorgehensweise des Schließens von einem allgemeinen auf ein spezielles Konzept, das dann wiederum weiter verfeinert wird. Zum Beispiel könnte ein spezielles Rennrad in der Konzepthierarchie

  Fortbewegungsmittel — Fahrzeug — Fahrrad — Rennrad

  eingeordnet sein.

- **Der Speicher ist keine reine Konzepthierarchie**
  Mit einer reinen Konzepthierarchie ist zum einen keine Verknüpfung unterschiedlicher Konzepte möglich, die vielleicht einige vergleichbare Merkmale haben, aber in keinem direkten Zusammenhang stehen. Zum anderen ist das Prinzip des Vergessens und das manchmal fehlerhafte Verhalten beim Erinnern mit einer Konzepthierarchie nicht modellierbar.

- **Gemeinsame Speicherung ähnlicher Konzepte**
  Ähnliche Konzepte sollten unter einem gemeinsamen Oberkonzept gespeichert und über ihre Unterschiede rekonstruierbar sein. Der Grund für diese Forderung ist die Beobachtung, daß durch Konzepte Assoziationen an andere ähnliche Konzepte stattfinden und diese Assoziationen auch durch eine gemeinsame Speicherung realisiert werden sollten.

- **Konzepte enthalten in verschiedenen Kontexten unterschiedliche Bedeutungen**
  Ein Beispiel dafür sei das Konzept einer Person „Politiker“. Ein Politiker trifft in einer Regierung Entscheidungen, also kann man sagen, daß Politiker Entscheidungsträger sind.

Aber Politiker sind auch Menschen und haben vielleicht eine Familie. In der Familie könnte es sein, daß der Partner „die Hosen anhat“. Damit ist ein Politiker zuhause nicht unbedingt Entscheidungsträger.

Eine Speicherstruktur, die diese Forderungen erfüllt ist der Episodenspeicher nach Kolodner.

6.3 Episodenspeicher (E-MOP)

In diesem Abschnitt wird ein Speichermodell beschrieben, das die obengenannten Prinzipien realisieren soll. Ausführlich dargestellt wird das Modell in [Kolodner83a, Kolodner83b, Kolodner84]. Wir werden nur die für die nachfolgende Formalisierung relevanten Aspekte genauer erläutern.

6.3.1 Speicheraufbau

Speicherstrukturen, die Fälle im Speicher organisieren, werden E-MOPs (engl. episodic memory organisation package) genannt. E-MOPs stellen eine Art von Konzepten (engl. conceptual categories) für die in ihnen gespeicherten Fälle dar. Sie haben jedoch eine umfassendere Bedeutung, denn sie bilden nicht nur einen Oberbegriff für eine Menge von Fällen, sondern sie stellen spezielle Zugriffsmechanismen zur Verfügung und verwalten Verbindungen zwischen Fällen und zu anderen E-MOPs.

Der Zugriff auf die von einem E-MOP verwalteten Fälle geschieht über sogenannte *Indizes*. Diese garantieren vor allem die wichtige Eigenschaft der Nichtaufzählbarkeit, denn nur wenn der Index bekannt ist, kann er benutzt werden, um auf Fälle zuzugreifen. Indizes bilden also eine Art von Sperre, die verhindert, daß der Speicher unmotiviert, d.h. ohne bekannten Zugriffsschlüssel, durchlaufen wird.

Neben Fällen können auch weitere E-MOPs (Sub–E-MOPs) über Indizes zugreifbar sein. Damit erhält man ein E-MOP-Netzwerk, in dem die Knoten die E-MOPs und die Kanten die Indizes sind. Die Indizes induzieren damit eine Vorgänger- und Nachfolgerrelation. Ein Netzwerk von E-MOPs und Fällen heißt *Fallspeicher*.

Als Beispiel für eine Netzwerk von E-MOPs diene ein Ausschnitt aus der Wissensbasis über den Politiker Cyrus Vance in Abbildung 6.1.

Der Mechanismus der Indexierung bewirkt zusätzlich einen effizienten Zugriffsmechanismus, da nur relevante Teile des Netzwerkes, zu denen man den passenden Schlüssel hat, durchlaufen werden.

E-MOPs enthalten neben den Indizes generalisierte Informationen über die in ihnen gespeicherten Fälle und Sub–E-MOPs. Diese Informationen charakterisieren Gemeinsamkeiten oder allgemeines Wissen. Die Berechnung von Generalisierungen ist ein großes Forschungsgebiet, insbesondere ist es ein Schwerpunkt maschinellen Lernens. Die wichtigste Form der Generalisierung ist die Induktion [HollandHolyoak86]. Wir beschäftigen uns in dieser Arbeit nicht mit der Problematik der Generalisierung, sondern gehen davon aus, daß Generalisierungen der Fälle auf irgendeine Art und Weise stattfinden. Generalisierungen müssen nicht unbedingt korrekt sein, d.h. sie müssen nicht für alle Fälle gelten.

Als abkürzende Schreibweise benutzt man für Generalisierungen den Begriff NORM und die Indizes eines E-MOPs bezeichnet man als DIFFs.

Bisher haben wir erwähnt, daß Indizes Fälle bzw. Sub–E-MOPs organisieren. Nun müssen wir noch erklären, was Indizes sind und wie sie aussehen.

Indizes sind Unterscheidungsmerkmale von Fällen, die innerhalb eines E-MOPs angeordnet sind.

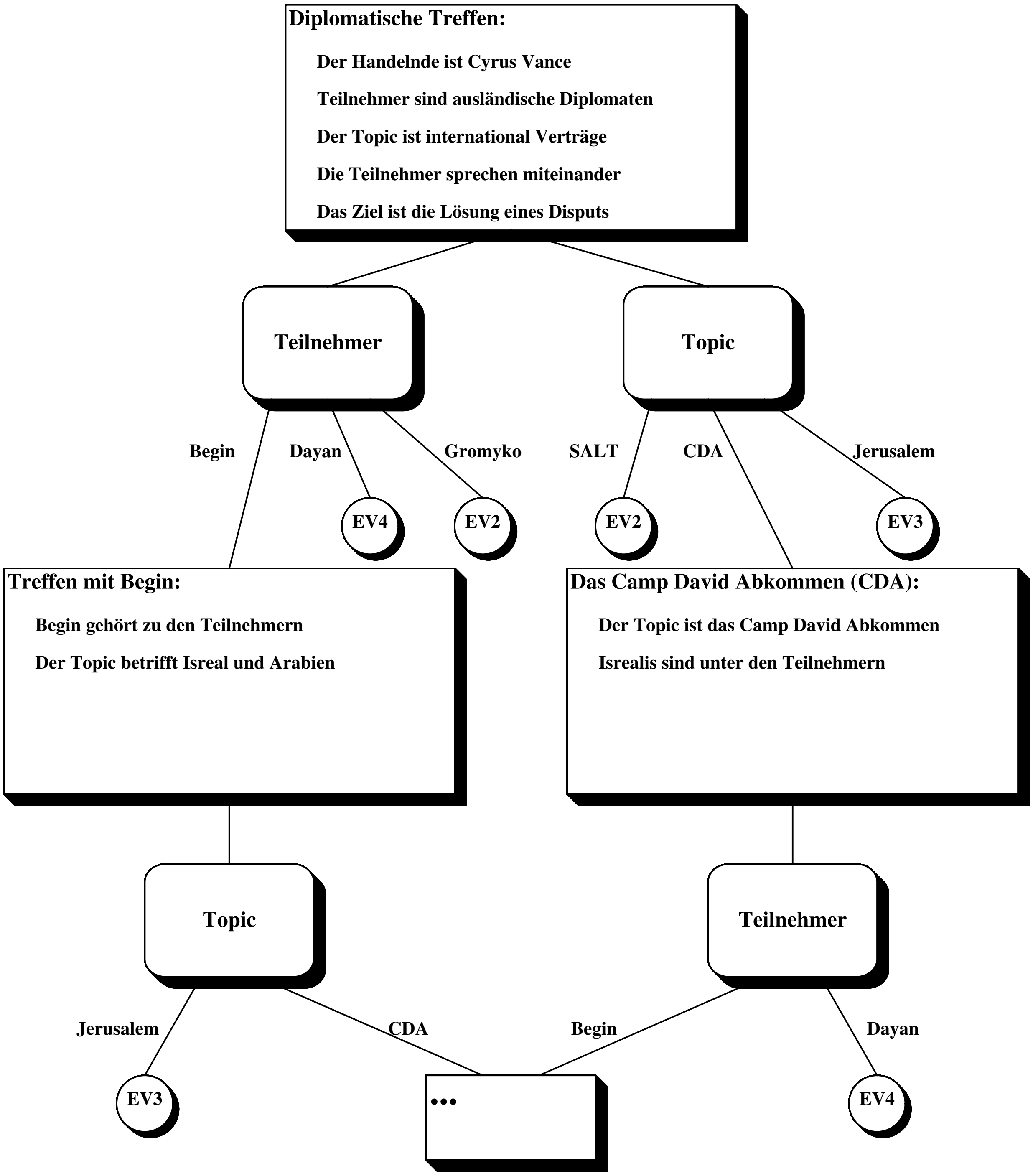

Abbildung 6.1: Eine Fallbasis

Die Menge aller Indizes eines E-MOPs wird deshalb auch unter dem Oberbegriff der differenzierenden Eigenschaften (DIFFs) zusammengefaßt.

Die Unterschiede zwischen den Fällen könnte potentiell eine sehr große Menge von Eigenschaften umfassen. Es wäre sehr aufwendig, alle diese Unterschiede als Indizes zu benutzen. Aus diesem Grund muß eine Auswahl getroffen werden, die relevante und nützliche Indizes bestimmt. „Gute“ Indizes zeichnen sich dadurch aus, daß sie vorausschauend (engl. predictive) für viele Merkmale sind. Das heißt im Zusammenhang mit diesem Index sind sehr viele andere Merkmale mitbestimmt, z.B. läßt die Universität, von der ein Absolvent kommt, oft Rückschlüsse auf einige Aspekte seines Wissensstandes und seiner Ausbildung zu.

Die Auswahl von Indizes ist im allgemeinen domänenabhängig und erfordert viel Hintergrundwissen. Manchmal ist es jedoch möglich, aus den Generalisierungen durch rein syntaktische Kriterien gute Indizes abzuleiten.

Indizes dienen vorrangig der Strukturierung der Fälle in den E-MOPs. Sie untergliedern eine unter Umständen große Fallbasis in kleinere handliche Pakete.

Auf einen speziellen Fall kann im allgemeinen auf verschiedene Arten zugegriffen werden, d.h. es gibt meist mehrere Indizes für einen Fall (engl. crossindexing).

Durch das E-MOP–Netzwerk ist ein Fall mit einer Folge von Indizes eindeutig referenzierbar (engl. subindexing). Diese Folge von Indizes entspricht einer Folge von E-MOPs, die durch diese Indizes verknüpft sind. Der spezialisierteste E-MOP enthält einen Index zu dem gesuchten Fall.

Indizes bestehen syntaktisch aus zwei Teilen: einem Attribut und einem Wert. Beispiele für solche Indizes sind: „Universität — Kaiserslautern“, „Universität — Trier“, „Farbe — lila“ und „Farbe — gelb“.

Zwei zentrale Aufgaben des Speichers, (1) die Eingliederung neuer Fälle und (2) das Retrieval bzw. Erinnern, werden nun am Konzept der E-MOPs erläutert.

6.3.2 Lernen neuer Fälle

Wenn ein neuer Fall in den Fallspeicher integriert wird, handelt es sich um ein Lernen im Sinne des maschinellen Lernens [RichterWendel91]. Um einen Fall zu lernen, muß zunächst ein passendes Konzept, also ein E-MOP, ermittelt werden. Dann müssen die Unterschiede zu den bereits in diesem E-MOP organisierten Fällen bestimmt werden, um damit Indizes zur Einordnung des Falles zu bestimmen. [Kolodner83a] beschreibt detailliert die Schwierigkeiten bei der Wahl von Indizes. Es werden dabei Lösungsansätze für die Fragen

- Was sind gute Indizes? und
- Wie können solche Indizes automatisch ermittelt werden?

gegeben.

Nach der Auswahl eines Index können drei Alternativen auftreten:

1. **Der Index ist neu**
   Dies ist der einfachste Fall: kein Fall oder Sub–E-MOP ist unter diesem Index bisher eingeordnet. Dann kann der Index neu generiert werden und der neue Fall unter diesem Index eingeordnet werden.
2. **Ein anderer Fall benutzt bereits den gleichen Index**
   Es muß ein neuer Sub–E-MOP erzeugt werden, der die Gemeinsamkeiten der beiden Fälle

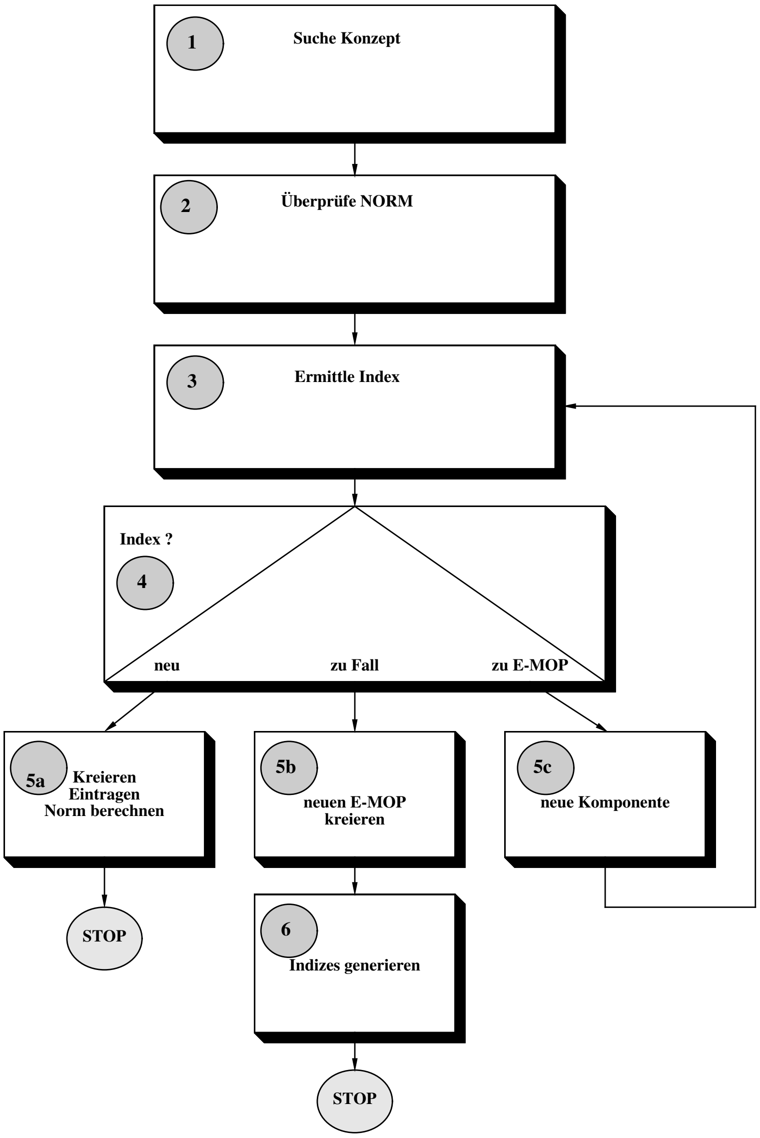

Abbildung 6.2: Der Lernprozess

mit gleichen Index beinhaltet. Unter diesem neuen E-MOP müssen dann die beiden Fälle bzgl. ihrer Unterschiede indexiert werden.

3. **Ein Sub–E-MOP ist bereits unter dem Index eingeordnet**
   In dieser Situation wird eine rekursive Behandlung angestoßen. Der neue Fall wird nach dem gleichen Verfahren in den Sub–E-MOP eingetragen.

Eine Veranschaulichung des Lernprozesses wird in Bild 6.2 gegeben.

Durch die Einordnung eines neuen Falles kann es passieren, daß die Generalisierungen, die in den NORMs eingetragen sind, nicht mehr stimmen oder unvollständig werden. Aus diesem Grund ist es nötig, in Schritt (2) die NORM zu überprüfen. Es ist möglich, sowohl Übergeneralisierungen (engl. overgeneralizations) als auch Untergeneralisierungen (engl. undergeneralizations) zu entdecken.

Bei einer Übergeneralisierung handelt es sich um eine NORM, die nicht mehr in den meisten Fällen korrekt ist, also keine Gemeinsamkeit der Fälle mehr repräsentiert.

Eine Untergeneralisierung dagegen ist eine Eigenschaft, die in den meisten Fällen gemeinsam vorkommt, aber nicht in den NORMs steht. Ein solcher Zustand zeichnet sich dadurch aus, daß viele Fälle über einen einzigen Sub–E-MOP eingeordnet sind. Dieser Sub–E-MOP wird dann aufgelöst und seine NORMs und DIFFs in den überprüften E-MOP integriert.

6.3.3 Retrieval im Fallspeicher

Hauptzweck des Speichers ist es, Informationen aufgrund von Anfragen zu finden. Im Prinzip wird dazu ein Fall gesucht, der zu der Anfrage paßt. Die darin enthaltenen relevanten Informationen werden dann als Antwort zurückgeliefert.

Eine Anfrage an den Speicher wird im folgenden als unvollständig spezifizierter Fall betrachtet, der durch gespeicherte Fälle vervollständigt werden soll.

Beginnend beim Wurzelobjekt des Fallspeichers werden bei der Suche Indizes durchlaufen und dabei neue verfeinerte E-MOPs entdeckt, bis man die gewünschten Informationen hat oder mangels weiterer Indizes die Suche abbrechen muß.

Das erste dabei auftretende Problem ist die Generierung von Indizes aus der Anfrage.

Die einfachste Art der Index–Generierung ist, aus den Informationen der Anfrage direkt den Index abzuleiten (engl. index fitting), z.B. eine Anfrage über rote Autos enthält den Index „Farbe — rot".

Eine zweite Art der Index–Generierung ist die Bildung plausibler Indizes (engl. plausible index generation). Es ist sehr schwer allgemeine Prinzipien für die automatische Generierung plausibler Indizes aufzustellen. Nur mit Hilfe von Hintergrundwissen ist es möglich, Regeln oder Heuristiken zu finden, die aus Anfragen plausible Indizes berechnen. [Kolodner83b] beschreibt das Beispiel eines Menschen, der versucht sich an Museumsbesuche zu erinnern.

> „Ich weiß, daß ich viele Museen in Europa besucht habe. Ich war in England. Dort war ich in vielen Museen — einige waren in London — das britische Museum, die National Gallery und einige kleinere Gallerien. Ich war dann in Brighton im Royal Pavillion. In Paris ... In Rom ..."

Man sieht hier, daß als Indizes zunächst europäische Länder und dann große Städte in diesen Ländern verwendet wurden.

Das zweite mögliche Problem ist, daß die Suche nach einem Fall erfolglos bleibt, d.h. es können keine weiteren Indizes gefunden werden. Bricht die Suche ab, so kann man versuchen aus dem aktuellen E-MOP trotzdem die benötigten Informationen zu beschaffen. Dazu kann man insbesondere die generalisierten Informationen verwenden.

Für unser Modell des Fallspeichers werden wir, wegen dieses Problems, keine Unterscheidung zwischen den Informationen im Fall und den Generalisierungen im E-MOP machen. Damit können wir mit einem E-MOP genauso wie mit einem Fall arbeiten. Diese Problematik wird im nächsten Kapitel ausführlicher behandelt.

Teil II

Anwendung

Kapitel 7

Der Case–Memory

Nachdem wir uns in Kapitel 6 vorrangig mit kognitiven Aspekten des Fallspeichers beschäftigt haben, steht in diesem Kapitel die Entwicklung eines formalen Modells im Mittelpunkt unseres Interesses. Ein E-MOP–Netzwerk wird hier durch eine Menge von Objekten, das sind Knoten, Attribute und Werte, und durch Relationen zwischen diesen Objekten beschrieben. Die resultierende Struktur werden wir Case–Memory nennen.

7.1 Formale Definition des Case–Memory

Formal bildet der Fallspeicher einen Graphen bestehend aus Knoten und gerichteten, markierten Kanten.

Knoten können E-MOPs oder Fälle sein. Die Unterscheidung von E-MOPs und Fällen ist nur anhand ihrer Position im Baum möglich: innere Knoten entsprechen E-MOPs und die Blätter entsprechen den Fällen. In unserem formalen Modell gibt es sonst keine Unterscheidung.

Die Markierung der Kanten bezeichnet die Indizes, die sowohl für differenzierende Eigenschaften stehen, als auch zum Durchlaufen bzw. Durchsuchen des Graphen dienen.

Die Modellierung des Graphen besteht aus einer Menge $\mathcal{M}$ von Knoten und einem ausgezeichneten Knoten M_0 aus $\mathcal{M}$. Dieser Knoten M_0 repräsentiert den Wurzelknoten, von dem der Retrievalprozeß ausgeht.

Die Kanten zwischen Knoten werden durch eine vierstellige Relation beschrieben, die die beteiligten Knoten und die Beschriftungen der Kanten in Form eines Attribut–Wert–Paares umfaßt. Dieses Attribut–Wert–Paar steht für eine differenzierende Eigenschaft des Knotens. Als intuitive Bezeichnung wird deshalb DIFF für die Relation verwendet.

Die Richtung der Kanten bestimmt den Grad der Spezialisierung und induziert eine Vorgänger- und Nachfolgerfunktion auf den Knoten. Die Nachfolgerfunktion heißt Δ und liefert zu einem Knoten alle über Indizes erreichbaren Knoten. Analog liefert die Vorgängerfunktion ∇ alle Knoten die über Indizes einen Knoten erreichen.

Die Generalisierungen in den E-MOPs und die in Fällen gespeicherten Informationen werden durch einen einheitlichen Formalismus dargestellt. Beide entsprechen formal einer Menge von Attribut–Wert–Paaren, die einem Knoten zugeordnet ist. Diese Zuordnung wird mit einer dreistelligen Relation $NORM$ realisiert, die den Knoten, das Attribut und den Wert enthält.

Damit sind alle Elemente der formalen Beschreibung genannt und wir können jetzt die Definition des ungeordneten Case–Memory angeben. Einschränkungen werden anschließend zur Definition des (geordneten) Case–Memory führen.

Definition 7.1.1 (ungeordneter Case–Memory)
Sei $\mathcal{M}$ eine endliche Menge von Knoten; $M_0 \subseteq \mathcal{M}$ ein ausgezeichneter Knoten; A eine endliche Menge von Attributen; V eine endliche Menge von Werten; $DIFF \subseteq \mathcal{M} \times A \times V \times \mathcal{M}$; $NORM \subseteq \mathcal{M} \times A \times V$.

Das 6–Tupel $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ heißt ungeordneter Case–Memory.

Der Knoten M_0 heißt Wurzelknoten *von CM.*

Die Relation $DIFF$ heißt Indizierungsrelation *von CM und $NORM$ heißt* Informationsrelation *von CM.*

Um die Formulierung von Beziehungen zwischen Knoten zu vereinfachen, definieren wir eine Vorgänger- und Nachfolgerfunktion auf dem Case–Memory.

Definition 7.1.2 (Nachfolger- und Vorgängerfunktion)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein ungeordneter Case–Memory.

Die Nachfolgerfunktion Δ ist wie folgt definiert

$\Delta : \mathcal{M} \to \mathcal{P}(\mathcal{M})$ [1]
$\Delta(M) = \{M' \in \mathcal{M} \mid \exists a \in A, v \in V \,.\, (M, a, v, M') \in DIFF\}$
$\Delta^0(M) = \{M\}$ *und* $\Delta^{(n+1)}(M) = \{\Delta(M') \mid M' \in \Delta^n(M)\}$
$\Delta^+(M) = \bigcup_{i=1}^{\infty} \Delta^i(M)$ *(transitive Hülle)*
$\Delta^*(M) = \bigcup_{i=0}^{\infty} \Delta^i(M)$ *(reflexive und transitive Hülle)*

Auf die gleiche Art und Weise wird auch die Vorgängerfunktion ∇ definiert

$\nabla : \mathcal{M} \to \mathcal{P}(\mathcal{M})$
$\nabla(M) = \{M' \in \mathcal{M} \mid \exists a \in A, v \in V \,.\, (M', a, v, M) \in DIFF\}$
$\nabla^0(M) = \{M\}$ *und* $\nabla^{(n+1)}(M) = \{\nabla(M') \mid M' \in \nabla^n(M)\}$
$\nabla^*(M) = \bigcup_{i=0}^{\infty} \nabla^i(M)$ *und* $\nabla^+(M) = \bigcup_{i=1}^{\infty} \nabla^i(M)$

Als abkürzende Schreibweise verwenden wir

$$\Delta(\{M_1, ..., M_n\}) := \bigcup_{i=1}^{n} \Delta(M_i)$$

und analog auch für ∇

$$\nabla(\{M_1, ..., M_n\}) := \bigcup_{i=1}^{n} \nabla(M_i)$$

Δ und ∇ bilden eine Art von Umkehrrelation. Die Menge der Vorgänger von Nachfolgern eines Knotens enthält den Knoten selbst. Entsprechendes gilt für die Nachfolgermenge der Vorgänger.

Lemma 7.1.1 (Zusammenhang von Δ und ∇)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein ungeordneter Case–Memory und Δ Nachfolger- und ∇ Vorgängerfunktion, dann gilt:

1. $\forall M \in \mathcal{M} \,.\, \forall M' \in \Delta(M) \,.\, M \in \nabla(M')$

[1] $\mathcal{P}(\mathcal{M})$ bezeichnet die Potenzmenge von $\mathcal{M}$

2. $\forall M \in \mathcal{M} \,.\, \forall M' \in \nabla(M) \,.\, M \in \Delta(M')$

Beweis:

1. Sei $M' \in \Delta(M)$, dann $\exists a \in A \,.\, \exists v \in V \,.\, (M, a, v, M') \in DIFF$ und somit $M \in \nabla(M')$.
2. analog. $\square$

Lemma 7.1.2 (Zusammenhang von Δ^+/∇^+ und Δ^*/∇^*)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein ungeordneter Case–Memory und Δ Nachfolger- und ∇ Vorgängerfunktion, dann gilt:

1. $\forall M \in \mathcal{M} \,.\, \Delta^+(M) = \Delta^*(\Delta(M))$

2. $\forall M \in \mathcal{M} \,.\, \nabla^+(M) = \nabla^*(\nabla(M))$

Beweis:

1. „$\Rightarrow$“ Sei $M' \in \Delta^+(M)$, dann $\exists n \geq 1 \,.\, M' \in \Delta^n(M) \leadsto \exists M'' \in \Delta(M) \,.\, M' \in \Delta^{n-1}(M'') \leadsto M' \in \Delta^{n-1}(\Delta(M)) \subseteq \Delta^*(\Delta(M))$
   „$\Leftarrow$“ Sei $M' \in \Delta^*(\Delta(M))$, dann $\exists n \geq 0 \,.\, M' \in \Delta^n(\Delta(M)) = \Delta^{n+1}(M) \subseteq \Delta^+(M)$.
2. analog. $\square$

Der Case–Memory muß sechs Bedingungen genügen:

1. **Der Case–Memory enthält M_0 als Wurzelknoten**
   Alle Knoten müssen von M_0 aus erreichbar sein und M_0 darf keinen Vorgänger haben.
2. **Der Case–Memory ist azyklisch**
   Zykel dürfen nicht auftreten, d.h. Knoten dürfen nicht in der transitiven Hülle ihrer Nachfolgerfunktion enthalten sein.
3. **Indizes sind eindeutig**
   Die Relation $DIFF$ ist eine partielle Funktion, d.h. bei Übereinstimmung der ersten drei Komponenten von zwei Tupeln aus der Relation $DIFF$, muß auch die vierte Komponente übereinstimmen.
4. **Indizes sind differenzierende Eigenschaften**
   Das Attribut–Wert–Paar eines Indexes darf nicht in der $NORM$ des Vorgängerknotens auftreten.
5. **Der Index ist eine Eigenschaft des Nachfolgers**
   In der $NORM$ des Nachfolgers muß das Attribut–Wert–Paar des Indexes auftreten.
6. **Die in einer $NORM$ gespeicherten Informationen sind eindeutig**
   Diese Forderung entspricht der Definition eines Falles nach Definition 6.1.1 und bedeutet, daß auch die Relation $NORM$ eine partielle Funktion ist.

Definition 7.1.3 (Case–Memory)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein ungeordneter Case–Memory.

CM ist ein (geordneter) Case–Memory*, wenn gilt:*
$\forall M, M', M'' \in \mathcal{M} \,.\, \forall a \in A \,.\, \forall v \in V$

1. $M \in \Delta^*(M_0)$
2. $M \notin \Delta^+(M)$
3. $(M, a, v, M') \in DIFF \wedge (M, a, v, M'') \in DIFF \Rightarrow M' = M''$
4. $(M, a, v, M') \in DIFF \Rightarrow (M, a, v) \notin NORM$
5. $(M, a, v, M') \in DIFF \Rightarrow (M', a, v) \in NORM$
6. $(M, a, v) \in NORM \wedge (M, a, v') \in NORM \Rightarrow v = v'$

Bemerkungen:

- Die Eigenschaft, daß M_0 keinen Vorgänger hat, ist nicht explizit in der Definition enthalten. Es ist jedoch möglich $\nabla(M_0) = \emptyset$ aus 1. und 2. zu schließen.

  Beweis: Annahme: es existiert ein $M \in \nabla(M_0)$.
  Es gilt: $M \in \Delta^*(M_0)$ nach 2. und nach 1., Lemma 7.1.1 und Lemma 7.1.2 gilt $M \notin \Delta^+(M) = \Delta^*(\Delta(M)) \supseteq \Delta^*(M_0)$. Damit erhalten wir einen Widerspruch, also ist $\nabla(M_0) = \emptyset$.

- Das syntaktische Kriterium zur Unterscheidung von Fällen und E-MOPs liefert die Untersuchung der Anzahl von Nachfolgern eines Knotens. Ist $\Delta(M) = \emptyset$, dann handelt es sich um ein Blatt und damit um einen Fall. Ist dagegen $\Delta(M) \neq \emptyset$, so ist M ein innerer Knoten, also ein E-MOP.

Durch die Relationen $DIFF$ und $NORM$ werden alle Knoten des Case–Memory global betrachtet. In einigen Fällen benötigt man lokale Informationen bzgl. eines Knotens, z.B. alles NORMs oder DIFFs eines Knotens. Um lokale Informationen zu erhalten, kann man Projektionen der Relationen $DIFF$ und $NORM$ definieren:

$NORM_M \subseteq A \times V$ mit
$(a, v) \in NORM_M$ gdw $(M, a, v) \in NORM$

$DIFF_M \subseteq A \times V \times \mathcal{M}$ mit
$(a, v, M') \in DIFF_M$ gdw $(M, a, v, M') \in DIFF$

7.2 Graphische Repräsentation des Case–Memory

Zur Veranschaulichung des Case–Memory werden wir eine graphische Repräsentation festlegen.

Die inneren Knoten (E-MOPs) werden als Rechtecke dargestellt, die eine Identifikation enthalten, z.B. M_0, und eine Liste von Gleichungen der Form $a = v$. Die Gleichungen sollen die Attribut–Wert–Paare der Generalisierungen repräsentieren und sind zu lesen als: das Attribut a hat normalerweise den Wert v.

Blattknoten (Fälle) erscheinen als Kreise. Sie haben den gleichen Inhalt wie die inneren Knoten, aber die Interpretation der Gleichungen ist etwas anders. Da die Blätter Fälle beschreiben, ist $a = v$ eine sichere Information, die besagt, daß in diesem Fall a den Wert v hat.

Für die Kanten wollen wir nicht nur einfache Linien verwenden, sondern wir werden die Zweiteilung der Indizes auch in der Graphik berücksichtigen. Attribute werden durch Ellipsen repräsentiert, die durch mit den Werten beschriftete Linien mit den Nachfolgerknoten verbunden sind.

Die Richtung der Kanten vom Vorgänger- zum Nachfolgerknoten wird nicht explizit durch einen Pfeil dargestellt, sondern sie ist implizit dadurch festgelegt, daß die Nachfolgerknoten unterhalb der Vorgängerknoten in der Graphik angeordnet sind.

Beispiel 7.2.1
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ *(Bild 7.1) gegeben als*

$\mathcal{M} = \{M_0, ..., M_7\}; A = \{a, b, c, d\}; V = \{0, 1, 2, 3\}$

$NORM = \{(M_0, a, 1), (M_1, b, 0), (M_2, b, 1), (M_2, d, 1), ...\}$

$DIFF = \{(M_0, b, 0, M_1), (M_0, b, 1, M_2), ..., (M_4, b, 2, M_7)\}$

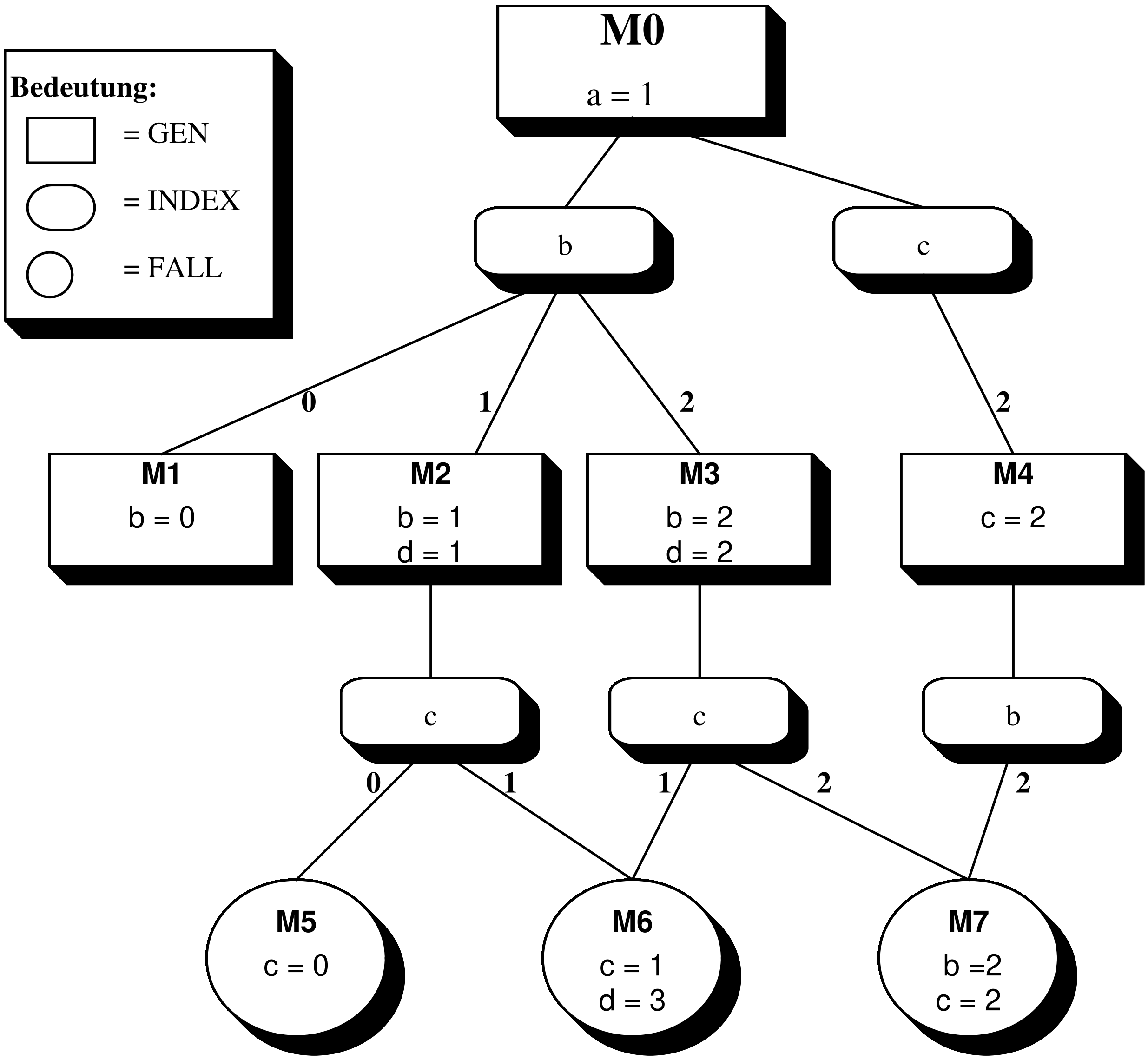

Abbildung 7.1: Beispiel eines Case–Memory

7.3 Informationsgehalt des Case–Memory und Retrieval

Nach der exakten Definition des Case–Memory können wir nun seine Bedeutung und seine Aufgabe festlegen. Die Aufgabe des Case–Memory ist die Bereitstellung von Informationen, die in irgendeiner Form der Lösungsfindung eines Problems dienen.

Zunächst ist die Beschreibung des Problems und der Lösung notwendig. Problemstellung sind für uns möglicherweise unvollständige Fallbeschreibungen, die, entsprechend den Fällen, aus einer Menge von Attribut–Wert–Paaren bestehen. Aufgabe des Case–Memory ist die Bestimmung von weiteren Attribut–Wert–Paaren, die zusätzliche Werte für Attribute, detaillierte Informationen oder Lösungen bzw. Lösungshinweise für den Eingabefall bereitstellen.

Wir bezeichnen die Eingabe als unvollständige Fallbeschreibung, die mit Hilfe des Case–Memory vervollständigt wird. Die Vervollständigung kann u.U. nur Teilaspekte liefern, mit denen andere Inferenzmechanismen arbeiten können oder die zur Abfrage anderer Wissensquellen eingesetzt werden können.

Den Prozeß der Vervollständigung nennen wir entsprechend Abschnitt 5.3 Retrieval. Der Retrievalprozeß soll nun genauer analysiert werden.

Prozedural läßt sich der Retrievalprozeß wie folgt beschreiben:

Eingabe:	$P \subseteq A \times V$ mit $P = \{(a_1, v_1), ..., (a_n, v_n)\}$ und $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$
Start:	Der aktuelle Knoten K sei M_0. $A := M_0$
Schleife:	Wähle ein Attribut–Wert–Paar aus P mit $\exists (a, v) \in P \, . \, (K, a, v, M') \in DIFF$.
	Setze M' als aktuellen Knoten. $K := M'$.
	Solange eine Index existiert, wiederhole die Schleife.
Ende:	Liefere die im letzten aktuellen Knoten K enthaltenen Informationen zurück.

Eine Einschränkung des Algorithmus ist zu beachten. Es werden nur die Indizes benutzt, die direkt aus der Eingabe generiert werden können. Eine Generierung plausibler Indizes, wie in Abschnitt 6.3.3 vorgeschlagen, findet hier nicht statt. Wir werden in Abschnitt 9.3 auf die Indexgenerierung zurückkommen und Lösungsansätze vorstellen.

Bisher ist noch nicht definiert worden, was der im letzten Schritt des Algorithmus erwähnte Informationsgehalt eines Knotens ist. Zunächst einmal gehört zum Informationsgehalt, die im Knoten gespeicherte Information, d.h. bei Fällen/Blattknoten das sichere Wissen und bei E-MOPs/inneren Knoten das generalisierte Wissen. Zusätzlich umfaßt der Informationsgehalt, die Generalisierungen der auf dem Weg von der Wurzel zum Endknoten durchlaufenen Knoten, wobei Generalisierungen u.U. überschrieben werden können.

Das Retrieval besteht nach den vorausgegangen Informationen aus zwei Teilen. Zunächst wird der Graph beginnend von der Wurzel vorwärts zum gesuchten Knoten durchlaufen. Anschließend wird der gefundene Lösungspfad zur Bestimmung des Informationsgehalt rückwärts abgearbeitet. Diesen zweistufigen Prozeß werden wir bei der Modellierung mit Defaults wiederfinden.

Der Informationsgehalt läßt sich wie folgt beschreiben. Sei $(M_0, M_1, ..., M_n)$ die Folge von Knoten, die der Algorithmus mit der Eingabe P durchläuft. Diese Folge heißt Lösungspfad. Der Informationsgehalt I von M_n ist dann bestimmt als

$$I = P \oplus (NORM_{M_n} \oplus (NORM_{M_{n-1}} \oplus (... \oplus NORM_{M_0}))),$$

wobei $\oplus$ eine Art von Vereinigungsoperation darstellt, die den linken Wert bevorzugt, wenn Attribut–Wert–Paare mit gleichem Attribut, aber unterschiedlichen Werten verknüpft werden.

Intuitiv bedeutet $F = F_1 \oplus F_2$, daß F alle Attribut–Wert–Paare aus F_1 enthält und zusätzlich die Paare aus F_2, deren Attribute keinen Wert in F_1 haben.

Definition 7.3.1 ($\oplus$)
Sei A endliche Attributmenge und V endliche Wertemenge.
Definiere $\oplus : (A \times V) \times (A \times V) \rightarrow (A \times V)$ als

1. $\forall F \subseteq A \times V \, . \, F \oplus \emptyset := F$

2. $\forall F, F' \subseteq A \times V \ . \ \forall a \in A \ . \ \forall v \in V$

$$F \oplus (F' \cup \{(a,v)\}) := \begin{cases} F \oplus F', & \textit{wenn } \exists v' \in V \ . \ (a,v') \in F \\ (F \oplus F') \cup \{(a,v)\}, & \textit{sonst.} \end{cases}$$

Zum Verständnis von $\oplus$, soll folgende Eigenschaft dienen

$$F \subseteq F \oplus F' \subseteq F \cup F'$$

Der erste Teil $F \subseteq F \oplus F'$ läßt sich durch eine Induktion nach der Anzahl der Elemente in F' beweisen. Der zweite Teil $F \oplus F' \subseteq F \cup F'$ ist offensichtlich.

Die Eingrenzung von $F \oplus F'$ ist scharf, d.h. es gibt Mengen F_1, F_2 so, daß die Gleichheit gilt: $F = F \oplus F'$, z.B. $F = \{(a,v)\}; F' = \{(a,v')\}$ und $F \oplus F' = F \cup F'$, z.B. $F = \{(a,v)\}; F' = \{(b,v')\}$.

Der Operator $\oplus$ ist nicht kommutativ, z.B. ist für $v \neq v'$:

$$\{(a,v)\} \oplus \{(a,v')\} = \{(a,v)\} \neq \{(a,v')\} = \{(a,v')\} \oplus \{(a,v)\}.$$

Es gilt jedoch das Assoziativgesetz:

$$\forall F, F', F'' \subseteq A \times V \ . \ (F \oplus F') \oplus F'' = F \oplus (F' \oplus F'')$$

Der Beweis der Assoziativität ist sehr technisch und basiert auf mehreren Induktionen. Er wird deshalb hier nicht explizit dargestellt.

Als nächstens wird der Begriff des Pfades in einem Case–Memory und der Informationsgehalt von Pfaden definiert.

Definition 7.3.2 (Pfad)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein Case–Memory.

Ein Pfad P von M nach M' in CM ist eine endliche Folge von Knoten $P = (M_1, ..., M_n)$ mit

1. $\forall 1 \leq i \leq n \ . \ M_i \in \mathcal{M}$
2. $M_1 = M$ *und* $M_n = M'$
3. $\forall 1 \leq i \leq n \ . \ M_{i+1} \in \Delta(M_i)$

Wir benutzen die Schreibweise $M \in P$ gdw $\exists 0 \leq i \leq n \ . \ M = M_i$

Definition 7.3.3 (Informationsgehalt)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein Case–Memory und $P = (M_1, ..., M_n)$ ein Pfad in CM.

Der Informationsgehalt $I(P) \subseteq A \times V$ ist rekursiv definiert:

1. $I(P) = \emptyset$, *falls* $n = 0$, *also* $P = ()$
2. $I(P) = NORM_{M_n} \oplus I(P')$, *falls* $n \geq 1$ *mit* $P' = (M_1, ..., M_{n-1})$

Im Algorithmus zum Retrieval wurde irgendein passender Index zur Lösungsfindung ausgewählt. Es soll aber i.a. nicht eine beliebige Lösung gefunden werden, sondern eine beste Lösung, in dem Sinn, daß sie den Anforderungen der Problemstellung weitgehend entspricht. Auf die Optimierung der Lösungen wollen wir jedoch zunächst verzichten und alle möglichen Lösungen bestimmen. Auf einige Aspekte bester Lösungen werden wir in Abschnitt 9.4 eingehen.

Ein Lösungspfad für ein Problem Q ist ein Pfad vom Wurzelknoten zu einem inneren Knoten, der nicht weiter detailliert werden kann. Unter Detaillierung ist die Generierung eines Indexes aus Q und die Bestimmung eines Nachfolgerknotens zu verstehen. Alle benachbarten Knoten im Lösungspfad müssen über Indizes verknüpft sein, die aus der Problembeschreibung Q generierbar sind.

Definition 7.3.4 (Lösungspfad)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ *ein Case–Memory und* $Q \subseteq A \times V$ *ein Problem.*

Ein Pfad $LP(Q) = (M_1, ..., M_n)$ *heißt Lösungspfad von* Q *bzgl.* CM*, wenn gilt*

1. $M_1 = M_0$
2. $\forall 1 \leq i < n \ . \ \exists (M_i, a, v, M_{i+1}) \in DIFF \ . \ (a, v) \in Q$
3. $\forall (a, v) \in Q \ . \ \forall M \in \mathcal{M} \ . \ (M_n, a, v, M) \notin DIFF$

Die Menge der Lösungspfade von Q *bzgl.* CM *bezeichnen wir mit* $\mathcal{LP}(Q)$.

Die Bedingung 2 enthält eine Art von Korrektheitsanforderung für Lösungpfade. Die 3. Bedingung drückt die Forderung nach Maximalität aus. Nur wenn sich der Knoten M_n nicht weiter detaillieren läßt, ist der Pfad ein Lösungspfad.

Aus der Definition des Lösungspfades und des Informationsgehalts von Pfaden läßt sich nun auf eine einfache Weise definieren, was eine Lösung eines Problems mit Hilfe des Case–Memory ist.

Definition 7.3.5 (Lösung)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ *ein Case–Memory und* $Q \subseteq A \times V$ *ein Problem.*

$L(Q) \subseteq A \times V$ *heißt Lösung von* Q *bzgl.* CM*, wenn gilt*

1. *Es existiert ein Lösungspfad* $LP(Q)$ *von* Q *bzgl.* CM.
2. $L(Q) = Q \oplus I(LP(Q))$

Die Menge aller Lösungen wird als $\mathcal{L}(Q)$ *notiert:*

$$\mathcal{L}(Q) = \{Q \oplus I(LP(Q)) \mid LP(Q) \in \mathcal{LP}(Q)\}$$

7.4 Beispiel

Die Begriffe Pfad, Informationsgehalt, Lösungspfad und Lösung sollen an einem Beispiel veranschaulicht werden. Wir legen den Fallspeicher nach Abschnitt 7.4 zugrunde.

Beispiele für Pfade sind

$$(), (M_4), (M_0, M_1), (M_0, M_2), (M_3, M_7), (M_0, M_3, M_6).$$

Keine korrekten Pfade sind

$$(M_1, M_0) \text{ und } (M_0, M_1, M_5).$$

Der Informationsgehalt wird exemplarisch für die Pfade $P_1 = (M_2, M_5)$ und $P_2 = (M_0, M_2, M_6)$ berechnet.

$$I(P_1) = NORM_{M_5} \oplus I((M_2)) = NORM_{M_5} \oplus NORM_{M_2} =$$
$$\{(c, 0)\} \oplus \{(b, 1), (d, 1)\} = \{(c, 0), (b, 1), (d, 1)\}$$

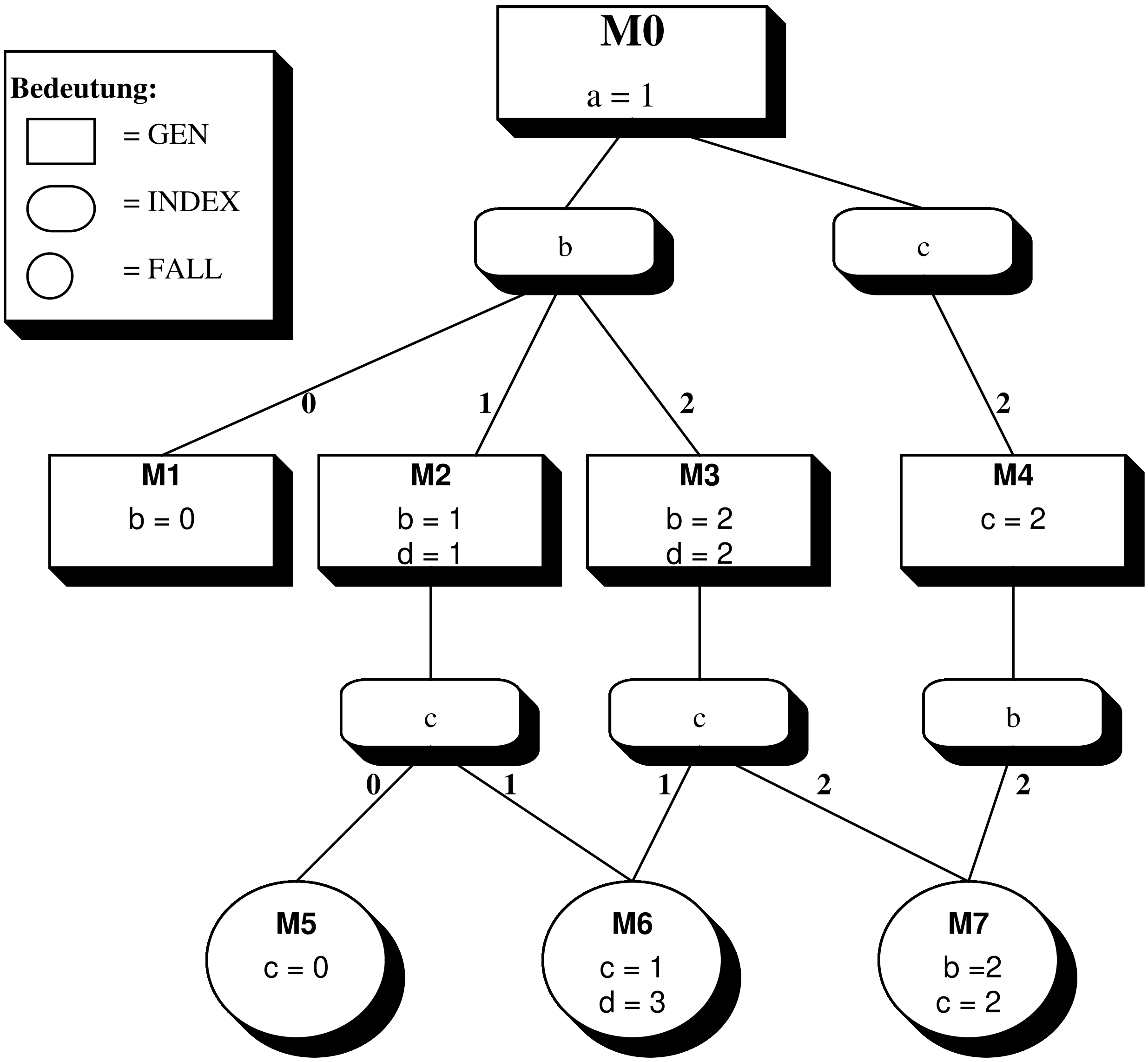

Abbildung 7.2: Beispiel eines Case–Memory

$I(P_2) = NORM_{M_6} \oplus I((M_0, M_2)) = NORM_{M_6} \oplus (NORM_{M_2} \oplus NORM_{M_0}) =$
$\{(c,1),(d,3)\} \oplus (\{(b,1),(d,1)\} \oplus \{(a,1)\}) = \{(c,1),(d,3)\} \oplus \{(b,1),(d,1),(a,1)\} =$
$\{(c,1),(d,3),(b,1),(a,1)\}$

Das Problem $Q = \{(b,2),(c,1),(f,5)\}$ hat als einzigen Lösungspfad $LP(Q) = (M_0, M_3, M_6)$. Die Lösung von Q ergibt sich damit als

$L(Q) = Q \oplus (M_6 \oplus (M_3 \oplus M_0)) = \{(b,2),(c,1),(f,5),(d,3),(a,1)\}.$

Kapitel 8

Der Case–Memory als Default–Theorie

Die Ausführungen aus den vorangegangenen Kapiteln werden nun zu den wichtigsten Ergebnissen der Arbeit zusammengebracht. Es ist möglich eine prädikatenlogische Sprache zu entwickeln, die zur Beschreibung des Case–Memory sinnvoll verwendet werden kann. Mit Hilfe dieser prädikatenlogischen Sprache definiert man eine partiell geordnete Default–Theorie, deren Extension den Informationsgehalt des Case–Memory für einen Eingabefall in einer kodierten Form enthält.

8.1 Prädikatenlogische Sprache

Ein typisches Szenario für die Verwendung der zu definierenden prädikatenlogischen Sprache besteht aus zwei Teilen. Zum einen hat man den Case–Memory, der durch Formeln und Defaults zu beschreiben ist. Zum anderen müssen die zu lösenden Probleme ebenfalls durch Formeln dargestellt werden. Durch ein Deduktionsverfahren werden weitere Formeln hergeleitet, z.B. indem eine Extension bestimmt wird. Aus dem Ergebnis der Deduktion sind zum Schluß die gewünschten Informationen abzulesen.

Die prädikatenlogische Sprache zur Beschreibung des Case–Memory enthält die üblichen Sonderzeichen und einige spezielle Objekte. Die Objekte sind Repräsentanten der Knoten, der Attribute und der Werte des Case–Memory. Zusätzlich gibt es Objekte zur Identifizierung der Eingabeprobleme. Alle genannten Objekte werden durch nullstellige Funktionssymbole, also Konstanten, dargestellt.

Neben den Funktionssymbolen gibt es Prädikatssymbole, die in der Sprache enthalten sein müssen, um die Objekte in Beziehung zu setzen. Ein dreistelliges Prädikatssymbol „*norm*“ dient der Zuordnung von Attribut–Wert–Paaren zu Knoten bzw. Eingabeproblem. Die Kanten des Case–Memory werden durch ein vierstelliges Prädikat „*diff*“ dargestellt. Daneben gibt es ein zweistelliges Subsumptionsprädikat „*sub*“, das die Zuordnung von Eingabeproblemen zu Knoten realisiert. Der Begriff der Subsumption wird im nächsten Abschnitt genau definiert. Hier soll das intuitive Verständnis von Subsumption, als Zuordnung der Eingabe zu Knoten im Lösungspfad, genügen.

Definition 8.1.1 (prädikatenlogische Sprache für Case–Memory)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ *ein Case–Memory und* $\mathcal{Q} \subseteq \mathcal{P}(A \times V)$ *eine Menge von Eingabefällen.*

$\mathcal{L}_{CM}$ *ist eine prädikatenlogische Sprache für* CM*, wenn gilt*

1. *$\mathcal{L}_{CM}$ ist eine prädikatenlogische Sprache.*
2. *$\forall a \in A$ existiert eine Konstante in $\mathcal{L}_{CM}$, die wir ebenfalls mit a bezeichnen wollen.*
3. *$\forall v \in V$ existiert eine Konstante in $\mathcal{L}_{CM}$, die wir ebenfalls mit v bezeichnen wollen.*
4. *$\forall M \in \mathcal{M}$ existiert eine Konstante in $\mathcal{L}_{CM}$, die wir mit m bezeichnen wollen.*
5. *$\forall Q \in \mathcal{Q}$ existiert eine Konstante in $\mathcal{L}_{CM}$, die wir mit q bezeichnen wollen.*
6. *Die obengenannten Mengen von Konstanten sind disjunkt.*
7. *$\mathcal{L}_{CM}$ enthält ein dreistelliges Prädikat norm, ein vierstelliges Prädikat diff und ein zweistelliges Prädikat sub.*

Ein Beispiel soll zeigen, wie man mit dieser Sprache umgehen kann. Der Zusammenhang zwischen Case–Memory bzw. Eingabeproblem und der prädikatenlogischen Darstellung wird im nächsten Abschnitt exakt dargestellt.

Beispiel 8.1.1
Sei $Q = \{(a,1),(b,2)\}$ ein unvollständiger Fall mit der zugeordneten Konstante f. $a, b, 1, 2$ sind ebenfalls Konstanten. Mit Hilfe des Prädikates norm läßt sich z.B. die Formel $norm(q,a,1) \wedge norm(q,b,2)$ konstruieren. Diese Formel ist zu lesen als: der Fall Q hat für das Attribut a den Wert 1 und für b den Wert 2.

Ist M ein Knoten des Case–Memory mit dem zugehörigen Funktionssymbol m, dann lassen sich weitere Formeln notieren:

$norm(m,a,2)$: *Das Attribut a hat für den Knoten M normalerweise den Wert v.*

$diff(m,b,1,m')$: *Der Knoten M hat als Nachfolger M' mit dem Index $(b,1)$.*

$sub(m,q)$: *Der Fall Q ist unter dem Knoten M zu subsumieren.*

8.2 Partiell geordnete Default–Theorie

Es wird nun vorgestellt, wie man mit der Sprache $\mathcal{L}_{CM}$ aus Abschnitt 8.1 den Case–Memory und die Eingabeprobleme in eine Menge von Defaults und Fakten transformieren kann. Zusätzlich werden Prioritäten für die Defaults angegeben, so daß eine partiell geordnete Default–Theorie entsteht. Des weiteren wird dargestellt, wie eine Vervollständigung der Eingabe bestimmt wird.

Es wird im folgenden ein Case–Memory $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ und eine Menge von unvollständigen Fällen $\mathcal{Q}$ vorausgesetzt. Wir werden nacheinander die benötigten Formeln bzw. Defaults vorstellen um sie dann abschließend zu einer Transformation zusammenzufassen. Mit $\mathcal{T}$ bezeichnen wir die Menge der Formeln zur Beschreibung des Case–Memory.

8.2.1 Formeln und Defaults

Zunächst sehen wir uns an, wie die Knoten mit den in ihnen gespeicherten Informationen transformierbar sind. Für jeden Knoten und jedes Attribut–Wert–Paar in den *NORM*s der Knoten wird eine Atomformel generiert, die genau das Tripel aus Knoten, Attribut und Wert beschreibt.

(F1) $\forall (M,a,v) \in NORM \;.\; norm(m,a,v) \in \mathcal{T}$

Der zweite Schritt, die Transformation der Kanten des Case–Memory, ist analog zum vorherigen Schritt.

(F2) $\forall (M, a, v, M') \in \mathit{DIFF} \,.\, \mathit{diff}(m, a, v, m') \in \mathcal{T}$

In der prädikatenlogischen Sprache $\mathcal{L}_{CM}$ ist für jeden Eingabefall eine Konstante definiert, um ihn eindeutig zu identifizieren. Die Identifizierung dient in erster Linie zur Spezifikation der Eingabefälle. Um unnötige Prädikate zu sparen und wegen der vergleichbaren Bedeutung, wollen wir die Zuordnung der Attribut–Wert–Paare zu den Fällen auch mit dem Prädikat *norm* realisieren.

(F3) $\forall Q \in \mathcal{Q} \,.\, \forall (a, v) \in Q \,.\, \mathit{norm}(q, a, v) \in \mathcal{T}$

Die Identifizierung der Eingabefälle ist weiterhin zur Definition der Subsumptionsrelation nötig, denn wir werden den Lösungspfad eines unvollständig beschrieben Falles als die Menge der Knoten bestimmen, die den Fall subsumieren.

Damit steht die Initialisierung der Subsumption fest: alle Eingaben werden vom Wurzelknoten subsumiert.

(F4) $\forall Q \in \mathcal{Q} \,.\, \mathit{sub}(m_0, q) \in \mathcal{T}$

Um Fälle weiter zu detaillieren, muß man Indizes traversieren und neue Knoten generieren, d.h. in unserer Sprache: man bestimmt einen Nachfolgeknoten der den Fall ebenfalls subsumiert. Dazu muß man zu den Eigenschaften des Eingabefalles eine passende differenzierende Eigenschaft finden.

Zuerst geben wir die Defaultregel $\mathit{spec}(x_m, x_a, x_v, x_q)$ an, um sie anschließend zu erklären.

(D5) $\forall M \in \mathcal{M}$.

$\mathit{spec}_M(x_m, x_a, x_v, x_q) : \mathit{diff}(m, x_a, x_v, x_m) \wedge \mathit{sub}(m, x_q) \wedge \mathit{norm}(x_q, x_a, x_v) \Rightarrow \mathit{sub}(x_m, x_q) \in \mathcal{T}$

Die Regel besagt im Prinzip: Wenn wir wissen es gibt eine Verbindung zwischen dem Knoten M und irgendeinem Nachfolgerknoten x_m mit dem Index (x_a, x_v) ($\mathit{diff}(m, x_a, x_v, x_m)$) und ein Fall x_q wird von M subsumiert ($\mathit{sub}(m, x_q)$), der für das Attribut x_a den Wert x_v hat ($\mathit{norm}(x_q, x_a, x_v)$), dann wird der Fall auch von x_m subsumiert ($\mathit{sub}(x_m, x_q)$).

Jetzt wird beschrieben, was es bedeutet, daß ein Fall unter einem Knoten subsumiert wird. Die intuitive Bedeutung ist die Übertragung der Knoteninformationen (*NORM*s) auf den Fall. Da die Attribut–Wert–Paare in den verschiedenen subsumierenden Knoten unterschiedlich sein können, handelt es sich um Defaults. Um die „richtigen" Werte für ein Attribut zu erhalten, werden wir weiter unten Prioritäten für diese Defaults festlegen.

(D6) $\forall M \in \mathcal{M} \,.\, \mathit{sol}_M(x_q, x_a, x_v) : \mathit{sub}(m, x_q) \wedge \mathit{norm}(m, x_a, x_v) \Rightarrow \mathit{norm}(x_q, x_a, x_v) \in \mathcal{T}$

Es fehlen noch zwei Formeln, die zum einen garantieren, daß genau ein Lösungspfad bestimmt wird und daß die Lösungen für jedes Attribut höchstens einen Wert enthalten. Diese beiden Formelschemata sind sehr interessant, da erst sie zu Inkonsistenzen in Zusammenhang mit den Defaults führen.

Das erste Formelschema drückt aus, daß ein Fall keine unterschiedlichen Werte für ein Attribut besitzen darf.

(F7) $\mathit{norm}(x_q, x_a, x_v) \wedge \mathit{norm}(x_q, x_a, x_w) \Rightarrow x_v = x_w \in \mathcal{T}$ [1]

Schließlich sollte eine Eingabefall nicht von Knoten subsumiert werden, die in keinem gemeinsamen Lösungspfad enthalten sind.

(F8) $\forall M, M' \in \mathcal{M}$ mit $M' \notin \Delta^*(M) \cup \nabla^*(M) \,.\, \neg \mathit{sub}(m, x_q) \vee \neg \mathit{sub}(m', x_q) \in \mathcal{T}$

Die Formulierung der Bedingung ist kompliziert, denn es ist zu verhindern, daß ein Fall von Knoten aus unterschiedlichen Pfaden subsumiert wird. Umgangssprachlich formuliert lautet die

[1] Mit = ist hier die syntaktische Gleichheit gemeint

Bedingung: Zwei Knoten, die in keiner Vorgänger- oder Nachfolgerbeziehung stehen, können keinen Fall gemeinsam subsumieren.

Folgendes Problem ist durch die Formel (F8) nicht berücksichtigt:

Gegeben sei ein Case–Memory mit $M_1, M_2 \in \Delta(M_0)$ und $M_2 \in \Delta(M_1)$ nach Abbildung 8.1.

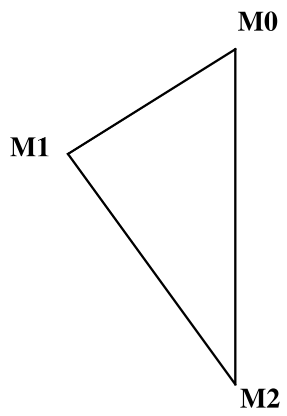

Abbildung 8.1: Beispiel für problematische Lösungspfade

Mögliche Lösungspfade könnten (M_0, M_1, M_2) oder (M_0, M_2) sein. Da mit die Extension der Default–Theorie eine Maximierung der Prädikate durchführt, wird man mit den bisherigen Fakten und Defaults nur die erste Lösung erhalten. Um auch die zweite Lösung zu erzeugen, führt man einen Default ein der die Lösung mit M_1 und M_2 gleichzeitig im Lösungpfad inkonsistent macht.

(D9) $\forall M \in \mathcal{M}\ .\ \forall M', M'' \in \Delta(M)\ .\ M' \neq M''$
$exc_{M,M',M''}(x_q) : \neg sub(m', x_q) \vee \neg sub(m'', x_q) \in \mathcal{T}$

In Zusammenhang mit Default (D5) kann es nun im obigen Beispiel drei Lösungen geben: die Defaults $spec_{M_0}(m_1, a, v, q), spec_{M_0}(m_2, b, w, q)$ und $exc_{M_0,M_1,M_2}(q)$ sind inkonsistent. Jeweils einer der Defaults darf nicht in der Extension auftreten. Damit erhält man die Lösungspfade: $(M_0, M_2), (M_0, M_1), (M_0, M_1, M_2)$. Durch eine entsprechende Vergabe der Prioritäten, kann man den Pfad (M_0, M_1) als Lösungspfad ausschließen.

Die Formeln (F7), (F8) und der Default (D9) heißen *Ausschlußbedingungen.*

8.2.2 Prioritäten

Der nächste Schritt auf dem Weg zu einer partiell geordneten Default–Theorie nach Abschnitt 4.3.3 erfordert die Festlegung von Prioritäten.

Die höchste Priorität geben wir den bisher als Fakten bezeichneten Formeln (F1) – (F4) und (F7) – (F8). Sie bilden die Menge T_0 der sicheren Formeln.

Wir werden zeigen, daß T_0 konsistent ist. Somit handelt es sich bei diesen Formeln um Fakten, die nicht revidiert werden müssen.

Die Prioritäten der Defaults (D5),(D6) und (D8) werden aus der Vorgehensweise des Retrievalalgorithmus abgeleitet. Beim Retrieval bestimmt man zunächst den Lösungspfad. Dazu wird der Case–Memory ausgehend von der Wurzel durchlaufen. Entsprechend zu diesem Verhalten bekommen die Defaults (D5) und (D8) die höhere Priorität gegenüber den Defaults (D6).

Nicht alle Defaults $spec_M(x_m, x_a, x_v, x_q)$ bzw. $exc_{M,M',M''}(x_q)$ erhalten die gleiche Priorität, sondern ausgehend von der Wurzel nimmt die Priorität zu den Blättern hin ab.

Sei n die Anzahl der Knoten im Case–Memory und $\mathcal{M} = \{M_1, ..., M_n\}$ eine Aufzählung der Knoten, dann kann man die Mengen $T_1, ..., T_n$ als $T_i = \{spec_{M_i}(x_m, x_a, x_v, x_q), exc_{M,M',M''}(x_q)\}$ festlegen.

Die partielle Ordnung dieser T_i ist mit Hilfe der Nachfolgerfunktion definiert:

$$\forall 1 \leq i, j \leq n \;.\; T_i \lhd T_j \text{ gdw } M_j \in \Delta^+(M_i)$$

Also gilt: T_i hat eine höhere Priorität als T_j, wenn M_i vor M_j im Case–Memory angeordnet ist. Die Ordnung ist sinnvoll, denn sie ist transitiv, wegen der Transitivität von Δ^+ und sie ist antisymmetrisch, wegen der Zykelfreiheit des Case–Memory.

T_0 hat eine höhere Priorität als die T_i:

$$\forall 1 \leq i \leq n \;.\; T_0 \lhd T_i$$

Im Algorithmus zum Retrieval wird als letzter Schritt der Informationsgehalt des Lösungspfades berechnet. Im Prinzip das gleiche leisten die Defaults (D6). Aus diesem Grund sind die Prioritäten für die „$sol_M(x_q, x_a, x_v)$“–Prädikate am niedrigsten.

T_{n+1} bis T_{n+n} enthalten jeweils einen Default aus (D6). $T_{n+i} = \{sol_M(x_q, x_a, x_v)\}$ mit der Bezeichnung für Knoten wie vorher.

Die Ordnung auf diesen Mengen ist umgekehrt zur Ordnung auf $T_1, ..., T_n$, denn nun werden die spezialisierten Informationen der Blätter stärker gewichtet, als die allgemeinen Informationen der inneren Knoten in Richtung Wurzel.

$$\forall 1 \leq i, j \leq n \;.\; T_{n+i} \lhd T_{n+j} \text{ gdw } M_i \in \Delta^+(M_j)$$

Da wir diese Mengen mit den niedrigsten Prioritäten versehen, gilt weiterhin

$$\forall 1 \leq i, j \leq n \;.\; T_i \lhd T_{n+j}$$

Ein kleines Beispiel zur Veranschaulichung der Prioritäten wird in Bild 8.2 gezeigt.

8.2.3 Zusammenhang zwischen Lösungen und Extensionen

Bevor der Abschnitt in einer Transformation zusammengefaßt wird, soll kurz auf die Bestimmung der Lösung mit Hilfe der partiell geordnete Default–Theorie im Vergleich zum Retrievalalgorithmus eingegangen werden. Formal wird der Zusammenhang in den Korrektheits- und Vollständigkeitssätzen hergestellt.

Um für einen Eingabefall F die Lösung zu bestimen, schaut man sich alle mit den Fakten und Defaults ableitbaren Formeln — eine Extension — an. Insbesondere sind die Atomformeln $norm(f, -, -)$ von Bedeutung, die als erstes Argument die Konstante zur Identifikation des Eingabefalles haben. Die Konjunktion dieser Formeln beschreibt gerade eine Lösung.

Verschiedene Lösungen können in verschiedenen Extensionen resultieren. Wir werden zeigen, daß die verschiedenen Extensionen genau die Lösungen der verschiedenen Lösungspfade enthalten.

8.2.4 Zusammenfassung der Transformation

Die Transformation des Case–Memory in eine partiell geordnete Default–Theorie $\Delta = (\mathcal{T}, \lhd)$ besteht in der Bestimmung von $\mathcal{T} = \{T_0, ..., T_{n+n}\}$ und einer strikt partiellen Ordnung $\lhd$ auf $\mathcal{T}$.

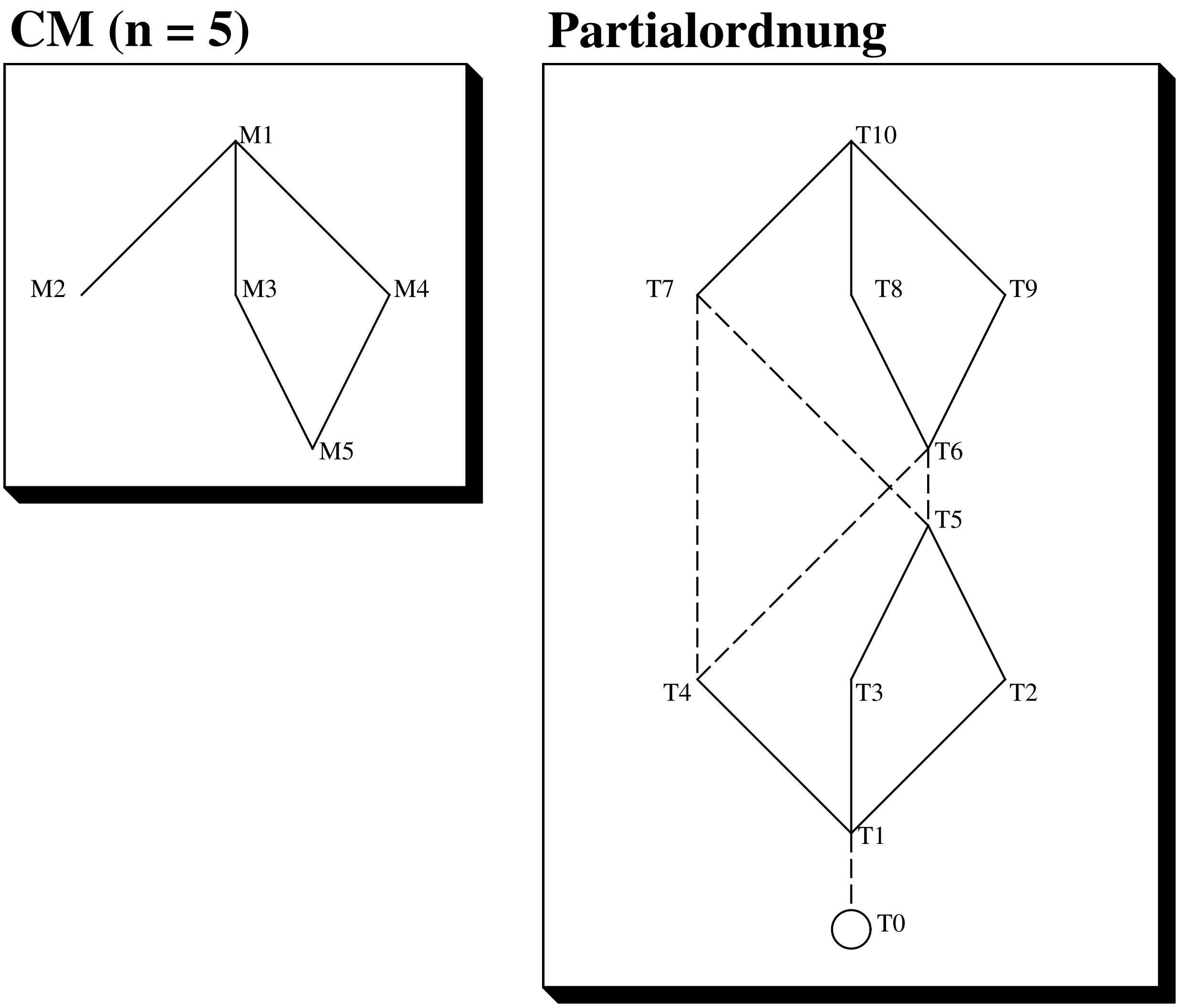

Abbildung 8.2: Prioritäten

Transformation 8.2.1 (partiell geordnete Default–Theorie für einen Case–Memory)
Sei $CM = (\mathcal{M}, M_0, A, V, NORM, DIFF)$ ein Case–Memory, $\mathcal{Q} \subseteq \mathcal{P}(A \times V)$ eine Menge von unvollständigen Fällen, d.h. $\forall Q \in \mathcal{Q}$ gilt: Q ist partielle Funktion, $\mathcal{L}_{CM}$ eine prädikatenlogische Sprache und n die Anzahl der Knoten in $\mathcal{M}$.

$\Delta = (\mathcal{T}, \lhd)$ ist eine partiell geordnete Default–Theorie des Case–Memory CM bzgl. $\mathcal{Q}$, wenn gilt.

1. $\mathcal{T} = \{T_0, ..., T_{n+n}\}$
2. T_0 *enthält folgende Formeln*
   (a) (F1) $\forall (M, a, v) \in NORM \;.\; norm(m, a, v) \in T_0$
   (b) (F2) $\forall (M, a, v, M') \in DIFF \;.\; diff(m, a, v, m') \in T_0$
   (c) (F3) $\forall Q \in \mathcal{Q} \;.\; \forall (a, v) \in Q \;.\; norm(q, a, v) \in T_0$
   (d) (F4) $\forall Q \in \mathcal{Q} \;.\; sub(m_0, q) \in T_0$
   (e) (F7) $norm(x_q, x_a, x_v) \wedge norm(x_q, x_a, x_w) \Rightarrow x_v = x_w$

(f) (F8) $\forall M, M' \in \mathcal{M}$ *mit* $M' \notin \Delta^*(M) \cup \nabla^*(M) \;.\; \neg sub(m, x_q) \vee \neg sub(m', x_q) \in \mathcal{L}$

3. *Die T_i ($1 \leq i \leq n$) sind bestimmt durch einelementige Mengen von Defaults.*

   (D5),(D9) $T_i = \{$
   $spec_{M_i}(x_m, x_a, x_v, x_q) : diff(m_i, x_a, x_v, x_m) \wedge sub(m_i, x_q) \wedge norm(x_q, x_a, x_v)$
   $\Rightarrow sub(x_m, x_q),$
   $exc_{M,M',M''}(x_q) : \neg sub(m', x_q) \vee \neg sub(m'', x_q)\}$

4. *Für $n+1 \leq i \leq n+n$ sind die T_i bestimmt als*

   (D6) $T_i = \{sol_{M_i}(x_q, x_a, x_v) : sub(m_i, x_q) \wedge norm(m_i, x_a, x_v) \Rightarrow norm(x_q, x_a, x_v)\}$

5. *Die strikte partielle Ordnung $\lhd$ der T_i ist definiert als*

   (a) $\forall 1 \leq i \leq n \;.\; T_0 \lhd T_i$
   (b) $\forall 1 \leq i, j \leq n \;.\; T_i \lhd T_{n+j}$
   (c) $\forall 1 \leq i, j \leq n \;.\; T_i \lhd T_j$ *gdw* $M_j \in \Delta^+(M_i)$
   (d) $\forall 1 \leq i, j \leq n \;.\; T_{n+i} \lhd T_{n+j}$ *gdw* $M_i \in \Delta^+(M_j)$

8.3 Eigenschaften

In diesem Abschnitt werden einige Eigenschaften der Transformation 8.2.1 angegeben. Zentrale Eigenschaften werden in den Sätzen zur Korrektheit und Vollständigkeit der partiell geordnete Default–Theorie zum Case–Memory ausgedrückt.

Wir werden die Beweise so führen, daß sie die wesentlichen Aspekte für das Verständnis beinhalten. Durch Argumente für die Korrektheit der Sätze und Lemmata werden die Grundideen hinter den Aussagen verständlicher gemacht. Mit Hilfe dieser Ideen und Beweisskizzen sollte es dem Leser möglich sein, die Beweise selbst zu vervollständigen.

Die erste Eigenschaft, die bereits vorher erwähnt wurde, ist die Konsistenz der Menge der Grundterme von T_0.

Lemma 8.3.1 (Konsistenz von $G(T_0)$)
Sei Δ wie in Transformation 8.2.1 gegeben.

$G(T_0)$ *ist konsistent.*

Beweis: Durch eine Induktion über alle Formeln in T_0 zeigt man die Konsistenz.

1. Die Menge der grundinstanziierten Formeln aus T_0 sind endlich.
2. Die Formeln (F1) – (F4) sind atomare Formeln, die zu keiner Inkonsistenz untereinander führen können.
3. Die Formel (F7) drückt aus, daß alle Projektionen $NORM_M (M \in \mathcal{M})$ und alle $Q \in \mathcal{Q}$ partielle Funktionen sind.
   (a) $\forall Q \in \mathcal{Q} \;.\; \forall (a, v), (a, w) \in Q \;.\; v = w$, also ist $norm(q, a, v), norm(q, a, w)$ und (F7) konsistent.
   (b) $\forall M \in \mathcal{M} \;.\; \forall (a, v), (a, w) \in NORM_M \;.\; v = w$, also ist $norm(m, a, v), norm(m, a, w)$ und (F7) konsistent.

4. (F8) kann ebenfalls zu keiner Inkonsistenz führen, da nur $sub(m_0, f)$ für alle Fälle gilt und für keinen weiteren Knoten $sub(m, f)$ ableitbar ist.

Definition 8.3.1 (Verträglichkeit)
Seien $CM, \mathcal{Q}, \Delta$ wie in Transformation 8.2.1 gegeben.

Wir bezeichnen einen Lösungspfad $LP(Q)$ und eine Extension E als verträglich, *wenn gilt:*

$\forall M \in \mathcal{M} \; . \; M \in LP(Q) \; gdw \; sub(m, q) \in E$

Wir bezeichnen eine Lösung $L(Q)$ und eine Extension E als verträglich, *wenn gilt:*

$\forall a \in A \; . \; \forall v \in V \; . \; (a, v) \in L(Q) \; gdw \; norm(q, a, v) \in E$

Der mit dem folgenden Satz definierte Korrektheitsbegriff zeigt, wie Lösungen des Case–Memory und die Verwendung der Transformation zusammenhängen. Die Aussage über die Korrektheit der Transformation ist: für jede Extension der partiell geordneten Default–Theorie Δ gibt es einen Lösungspfad im Case–Memory, so daß ein Attribut–Wert–Paar genau dann in der Lösung ist, wenn das Prädikat *norm*, das das Tripel aus Eingabefall, Attribut und Wert in Beziehung setzt, in der Extension enthalten ist.

Satz 8.3.1 (Korrektheit)
Seien $CM, \mathcal{Q}, \Delta$ wie in Transformation 8.2.1 gegeben.

$\forall Q \in \mathcal{Q} \; . \; \forall E \in \mathcal{E}(\Delta) \; . \; \exists L(Q) \in \mathcal{L}(Q)$

$\forall a \in A \; . \; \forall v \in V \; . \; norm(q, a, v) \in E \; gdw \; (a, v) \in L(Q).$

Beweis:

1. Es gilt $\forall E \in \mathcal{E}(\Delta) \; . \; \exists LP(Q) \in \mathcal{LP}(Q)$ so, daß E und $LP(Q)$ verträglich sind (Nach Lemma 8.4.3 im folgenden Abschnitt).

2. Nach Lemma 8.4.4 gilt zusätzlich, daß die Lösungen mit der Extension verträglich sind.

Analog zur Korrektheit definieren wir die Vollständigkeit, als die Eigenschaft jeder maximalen Lösung eine Entsprechung in einer zugehörigen Extension zu besitzen.

Satz 8.3.2 (Vollständigkeit)
Seien $CM, \mathcal{Q}, \Delta$ wie in Transformation 8.2.1 gegeben.

$\forall Q \in \mathcal{Q} \; . \; \forall L(Q) \in \mathcal{L}(Q) \; . \; \exists E \in \mathcal{E}(\Delta)$

$\forall a \in A \; . \; \forall v \in V \; . \; (a, v) \in L(Q) \; gdw \; norm(q, a, v) \in E$

Beweis:

1. Es gilt $\forall LP(Q) \in \mathcal{LP}(Q) \; . \; \exists E \in \mathcal{E}(\Delta)$ so, daß E und $LP(Q)$ verträglich sind (Nach Lemma 8.4.2 im folgenden Abschnitt).

2. Nach Lemma 8.4.4 gilt zusätzlich, daß die Lösungen mit der Extension verträglich sind.

8.4 Beweise und weitere Eigenschaften

Der Beweis der Vollständigkeit und Korrektheit wird zusammen durchgeführt, indem bestimmte Eigenschaften der Extensionen untersucht werden. Diese Eigenschaften werden wir als Lemmata formulieren.

Zunächst stellen wir fest, daß die Formeln in T_0 tatsächlich Fakten sind, also in jeder Extension enthalten sind.

Lemma 8.4.1
Seien $CM, \mathcal{Q}, \Delta$ wie in Transformation 8.2.1 gegeben.

Für alle Extensionen E von Δ gilt: $G(T_0) \in E$.

Beweis: Dieses Lemma ist eine Folgerung aus Lemma 8.3.1, denn wenn $G(T_0)$ konsistent ist und T_0 die höchste Priorität hat, dann muß jede bevorzugte Unter–Theorie wegen der Maximalität $G(T_0)$ enthalten.

Im folgenden Lemma wird der Zusammenhang zwischen dem Subsumptionsprädikat und dem Lösungspfad genauer beschrieben. Sei $Q \in \mathcal{Q}$ ein Eingabefall, dann subsumiert jeder Knoten im Lösungspfad zu Q in der Extension den Fall. Jeder Knoten, der Q subsumiert, ist auch im Lösungspfad zu Q enthalten.

Lemma 8.4.2 (Vollständigkeit der Lösungspfade)
Seien $CM, \mathcal{Q}, \Delta$ wie in Transformation 8.2.1 gegeben.

$\forall Q \in \mathcal{Q} \ . \ \forall LP(Q) \in \mathcal{LP}(Q) \ . \ \exists E \in \mathcal{E}(\Delta)$ *mit*

1. $\forall M \in \mathcal{M} \ . \ M \in LP(Q) \Rightarrow sub(m, q) \in E$
2. $\forall M \in \mathcal{M} \ M \notin LP(Q) \Rightarrow sub(m, q) \notin E$

Beweis:

1. Durch eine Induktion über die Knoten im Lösungspfad $LP(Q)$ zeigen wir den ersten Teil. Sei $LP(Q) = (M_0, ..., M_n)$.

   I.A.: $M_0 \in LP(Q) \leadsto sub(m_0, q) \in E$ nach Lemma 8.4.1

   I.V.: $\forall 1 \leq i \leq k < n \ . \ sub(m_i, q) \in E$ und $\forall 1 \leq j < k \ . \ \forall (M_j, a, v, M_{j+1}) \in DIFF \ .$
   $spec_{M_j}(m_{j+1}, a, v, q) \in E.$

   I.S.: $k \rightarrow k+1$: zu zeigen ist $sub(m_{k+1}, q) \in E$

   Wir zeigen, daß der Default (D5) $spec_{M_k}(m_{k+1}, a, v, q)$ konsistent ist und $sub(m_{k+1}, f)$ ableitbar ist.

   Es gilt:

   (a) $sub(m_k, q) \in E$

   (b) $\exists (M_k, a, v, M_{k+1}) \in DIFF$ mit $(a, v) \in Q$ (nach Definition 7.3.4)
   $\leadsto diff(m_k, a, v, m_{k+1}) \in E$ (nach Lemma 8.4.1).

   (c) aus $(a, v) \in Q$ folgt $norm(q, a, v) \in E$ (ebenfalls mit Lemma 8.4.1).

(d) Beh: $spec_{M_k}(m_{k+1}, a, v, q) \in E$
Bew:
$spec_{M_k}(m_{k+1}, a, v, q) : \mathit{diff}(m_k, a, v, m_{k+1}) \wedge sub(m_k, q) \wedge norm(q, a, v) \Rightarrow sub(m_{k+1}, q)$ ist konsistent und es gibt keine Grundinstanz von (D5) mit einer höheren Priorität, die konsistent ist und nicht bereits in E enthalten. Damit folgt die Behauptung $spec_{M_k}(m_{k+1}, a, v, q) \in E$.

Mit den Aussagen (a) – (d) folgt damit die Behauptung $sub(m_{k+1}, a, v, q) \in E$. □

2. Ann.: Für alle Extensionen E, die 1. erfüllen gilt:
$\exists M \in \mathcal{M} \;.\; M \notin LP(Q) \wedge sub(m, q) \in E$
Es gilt $M \neq M_0$, also muß $sub(m, q)$ durch (D5) eingeführt werden.
Das bedeutet: $\exists M', M'' \in LP(Q) \;.\; M \in \Delta(M')$ und $M'' \in \Delta(M') \leadsto$
   (a) (D8) $exc_{M',M,M''}(q) \notin E$.
   (b) (D5) $spec_{M'}(m, a, v, q) \in E$.
   (c) (D6) $spec_{M'}(m'', b, w, q) \in E$.

   Es muß aber auch eine Extension E' geben, in der (D8) und (D6) herleitbar sind und (D5) nicht herleitbar ist. Damit besteht ein Widerspruch zur Annahme. □

Mit Lemma 8.4.2 haben wir gezeigt, daß zu jedem Lösungspfad eine zugehörige Extension existiert. Den umgekehrten Fall behandeln wir im nachfolgenden Lemma. Zu jeder Extension gibt es auch einen Lösungspfad.

Lemma 8.4.3 (Korrektheit der Lösungspfade)
Sei Δ wie in Transformation 8.2.1 gegeben.
$\forall Q \in \mathcal{Q} \;.\; \forall E \in \mathcal{E}(\Delta) \;.\; \exists LP(Q) \in \mathcal{LP}(Q)$ *mit*

1. $\forall sub(m, q) \in \mathcal{L}_{CM} \;.\; sub(m, q) \in E \leadsto M \in LP(Q)$
2. $\forall sub(m, q) \in \mathcal{L}_{CM} \;.\; sub(m, q) \notin E \leadsto M \notin LP(Q)$

Beweis:

1. Der Beweis wird durch eine Induktion über den Abstand der Knoten zur Wurzel durchgeführt.
   I.A. $sub(m_0, q) \in E$ und $M_0 \in LP(Q)$
   I.V. $\forall M \in \bigcup_{i=0}^{k} \Delta^i(M_0) \;.\; \forall sub(m, q) \in \mathcal{L}_{CM} \;.\; sub(m, q) \in E \leadsto M \in LP(Q)$
   I.S. Wenn $M \in \Delta^{k+1}(M_0)$ und $sub(m, q) \in E$, dann muß (D5) angewendet worden sein, also $\mathit{diff}(m_k, a, v, m) \wedge sub(m_k, q) \wedge norm(q, a, v) \in E$. Aus $(M_k, a, v, M) \in DIFF$ folgt $M_k \in \bigcup_{i=0}^{k} \Delta^i(M_0)$. Damit gilt $M_k \in LP(Q)$, da zusätzlich $sub(m_k, q) \in E$. Mit $(M_k, a, v, M) \in DIFF$ und $(a, v) \in Q$ folgt $M \in LP(Q)$.
2. Ann: $sub(m, q) \notin E$ und $M \in LP(Q)$.
   Es gilt: $M \neq M_0$. Aus $M \in LP(Q)$ folgt, daß alle Voraussetzungen von (D5) erfüllt sind, also muß (D9) benutzt worden sein. Es gibt folglich ein $LP'(Q) \in \mathcal{LP}(Q)$, das M nicht enthält.

Mit Lemma 8.4.2 und 8.4.3 wurde gezeigt daß die Extensionen genau die richtigen Lösungspfade enthalten. Im nächsten Schritt ist nun zu zeigen, daß auch die Lösungen richtig berechnet werden. Auch dazu sind wiederum zwei Teile zu beweisen. Zum einen muß für jedes Tupel aus dem Lösungspfad das zugehörige $norm(q,-,-)$ – Prädikat in der Extension enthalten sein. Zum anderen dürfen nur die Prädikate zu denen es eine Lösung gibt, in der Extension sein.

Lemma 8.4.4 (Korrektheit und Vollständigkeit der Lösungen)
Seien $CM, \mathcal{Q}, \Delta$ wie in Transformation 8.2.1 gegeben.

$\forall LP(Q) \in \mathcal{LP}(Q)\ .\ \forall E \in \mathcal{E}(\Delta)$. *Wenn E mit $LP(Q)$ verträglich ist, dann gilt:*

Sei $L(Q) = Q \oplus I(LP(Q))$ die zugehörige Lösung.

1. $\forall (a,v) \in I(Q) \rightsquigarrow norm(q,a,v) \in E$
2. $\forall norm(q,a,v) \in E \rightsquigarrow (a,v) \in L(Q)$

Beweis:

1. Induktion über die Knoten des Lösungspfades, beginnend am Ende des Pfades.
   I.A.: $k = n\ .\ \forall (a,v) \in Q \rightsquigarrow norm(q,a,v) \in E$
   I.V.: $\forall n \geq i > k \geq 0\ .\ \forall (a,v) \in NORM_{M_i} \wedge (a,v) \in L(Q) \rightsquigarrow norm(q,a,v) \in E$
   I.S.: $k+1 \rightarrow k$: Sei $(a,v) \in NORM_{M_k}$
   (a) $\exists (a,w) \in L(Q)\ .\ w \neq v \rightsquigarrow (a,v) \notin L(Q)$
   (b) Der Default (D6) $sol_{M_k}(q,a,v)$ ist konsistent und es gibt keinen Default $sol_M(q,a,w)$ mit einer höheren Priorität. Dann ist $sol_{M_k}(q,a,v) \in E$. Es gilt außerdem $sub(m_k,q) \in E$ und $norm(m_k,a,v) \in E$, also folgt $norm(q,a,v) \in E$.
2. Sei $norm(q,a,v) \in E$, dann kann diese Formel auf zwei Arten herleitbar sein:
   (a) (aus (F3)): $norm(q,a,v) \in E \rightsquigarrow (a,v) \in Q \rightsquigarrow (a,v) \in L(Q)$.
   (b) (aus (D6)): $sol_M(q,a,v) \in E \rightsquigarrow sub(m,q) \wedge norm(m,a,v) \in E$. Aus $sub(m,q) \in E$ folgt $\exists 1 \leq k \leq n\ .\ M = M_k \in LP(Q)$ und aus $norm(m,a,v) \in E$ folgt $(a,v) \in NORM_M$. Da der Default anwendbar ist, kann es keinen Default mit höherer Priorität geben, der ebenfalls einen Knoten des Lösungspfades betrifft und einen Wert für a berechnet:
   $\forall i > k\ .\ M_i \in LP(Q) \rightsquigarrow sol_{M_i}(q,a,w) \in E$, wegen der höheren Priorität.
   Aber $norm(q,a,v) \notin E$, also $norm(m_i,a,w) \notin E$, also $(a,w) \notin NORM_{M_i}$. Das bedeutet $(a,v) \in L(Q)$.

Kapitel 9

Mögliche Erweiterungen und Ausblick

Die Formalisierung des Case–Memory mit einer partiell geordneten Default–Theorie soll ein genaueres Verständnis des Case–Memory erlauben. Darüber hinaus möchten wir auch Ideen und Verallgemeinerungen aus der Forschung zur Logik, insbesondere zu nichtmonotonen Logiken, auf das fallbasierte Schließen übertragen.

Einige Ansätze für mögliche Erweiterungen werden im folgenden diskutiert. Diese sind vor allem als Anregungen gedacht und müßten im einzelnen noch genauer analysiert werden. Die detaillierte Analyse ist im Rahmen dieser Arbeit jedoch nicht möglich.

9.1 Total geordnete Default–Theorie

Partiell geordnete Default–Theorien haben durch ihre mangelnde Fähigkeit zur eindeutigen Konfliktlösung bei inkonsistenten Defaults den Nachteil, viele Extensionen zu liefern (vgl. Abschnitt 4.3.3). Auch unsere Modellierung beinhaltet dieses Problem. Man ist deshalb an Kriterien interessiert, die angeben, wann es möglich ist, statt einer partiell geordneten Default–Theorie eine total geordnete Default–Theorie zu verwenden. Um solche Kriterien zu finden, sehen wir uns die Klassen $T_1, ..., T_n$ bzw. $T_{n+1}, ..., T_{n+n}$ von Defaults nach Transformation 8.2.1 an.

Im folgenden benötigen wir den Begriff der *Erweiterung* einer Partialordnung. Eine Totalordnung $\lhd_t$ heißt *Erweiterung* der partiellen Ordnung $\lhd$, wenn alle Elemente, die bereits partiell geordnet sind, in der Totalordnung in gleicher Weise geordnet sind:

$w \lhd w' \leadsto w \lhd_t w'$

Für die Defaults $spec_M, exc_{M,M',M''}$ zum Durchlaufen des Graphen, kann man die partielle Ordnung $\lhd$ auf $T_1, ..., T_n$ beliebig zu einer totalen Ordnung erweitern, denn es gilt:

1. Liegen zwei Knoten M_i, M_j auf einem gemeinsamen Pfad, o.B.d.A. $M_i \in \Delta(M_j)$, dann liefert die transitive Hülle der Nachfolgerfunktion, wie in der partiellen Ordnung auch, die Priorisierung von T_j über T_i.

2. Liegen M_i, M_j nicht auf einem gemeinsamen Pfad, dann ist auch die Bevorzugung beliebig zu wählen, da es zu keinem Konflikt zwischen den Defaults T_i und T_j kommen kann.

Die gleiche Argumentation gilt auch für die Defaults $T_{n+1}, ..., T_{n+n}$ zur Berechnung der Lösung. Die wichtigste Bedingung ist, daß die Defaults, die zu Knoten auf einem gemeinsamen Pfad gehören, geordnet sind. Durch die Partialordnung wird diese Bedingung bereits erfüllt, so daß die Totalordnung eine beliebige Erweiterung sein kann.

9.2 Verallgemeinerung des Fallbegriffes

Eine erste Art der Generalisierung von Fällen wurde bereits eingeführt. Fälle werden normalerweise mit einer bestimmten Absicht bzw. für eine spezielle Aufgabe verwendet. Zu diesem Zweck bestehen Fälle meist aus zwei Teilen: (1) einer Beschreibung des Falles, die insbesondere beim Retrieval benutzt wird und (2) den gewünschten Informationen. Der in Abschnitt 6.1 eingeführte Fallbegriff macht diese Unterscheidung nicht, sondern prinzipiell kann jede Teilinformation des Falles zum Retrieval verwendet werden oder vom Retrievalprozeß als Ausgabe geliefert werden.

Im Case–Based Reasoning benutzt man im allgemeinen einen Fall aus dem Fallspeicher, um diesen zur Lösung eines Problems zu verwenden. Um die optimalen Informationen aus dem Fallspeicher zu bekommen, kann es u.U. sinnvoll sein, mehrere gespeicherte Fälle zu betrachten. Dabei kann man z.B. aus verschiedenen Fällen Informationen verwenden, die jeweils für das Problem am besten geeignet sind. Es ist auch möglich aus mehreren Fällen die Informationen zu benutzen, die allen gemeinsam sind, um zum Beispiel eine höhere Sicherheit zu haben, daß die Informationen auch korrekt sind.

Im Prinzip werden beim Retrieval nach Abschnitt 7.3 bereits gemeinsame Informationen aus verschiedenen Fällen verwendet. Falls ein Lösungspfad nicht zu einem Blatt führt, sondern an einem inneren Knoten endet, dann werden die Generalisierungen, die Informationen aus verschiedenen Fällen enthalten, als Lösung zurückgeliefert.

Eine weitere Verallgemeinerung kann man durch eine andere Sichtweise von Fällen gewinnen. Unser Fallbegriff basiert auf der Annahme, daß viele Probleme und ihre Lösungen bzw. Teile ihrer Lösungen sich durch eine Menge von Attribut–Wert–Paaren darstellen lassen. Es gibt aber auch Probleme, die eine komplexere Beschreibung erfordern oder die sich durch Attribut–Wert–Paare nicht adäquat darstellen lassen. Ein Beispiel für einen komplexen Fall aus einer Diagnosesituation könnte sein: Eine Maschine ist defekt und es riecht verbrannt. Man weiß dann, daß eine Komponente einen Fehler in der Mechanik hat, aber man weiß nicht welche Komponente das ist. Eine prädikatenlogische Beschreibung könnte z.B. folgendermaßen aussehen:

$$norm(F, Maschine, defekt) \wedge norm(F, riecht, verbrannt) \wedge \exists x \,.\, norm(F, MechanikDefektIn, x)$$

Im Prinzip läßt die Darstellung des Case–Memory mit Defaults jede beliebige prädikatenlogische Formel zur Repräsentation von Fällen zu. Falls man für die Fallbeschreibung komplexe Formeln verwendet, muß man auch für die generalisierten Informationen und für die Indizes alle Formeln erlauben. Damit müssen sich auch die Defaults zur Bestimmung des Lösungspfades ändern.

Ein Index sei nun eine beliebige prädikatenlogische Formel. Er kann traversiert werden, wenn die Index–Formel aus der Fallbeschreibung herleitbar ist. Als Beispiel sei der folgende Index gegeben:

$$norm(x, Komponente, A) \vee norm(x, Komponente, B)$$

Der Index ist dann mit dem Fall F

$$norm(F, Maschine, defekt) \wedge norm(F, Komponente, B)$$

traversierbar.

Wesentlich schwieriger ist die Situation bei der Verwendung von Generalisierungen zur Berechnung von Lösungen. Bisher gab es ein relativ einfaches syntaktisches Kriterium, wann eine Generalisierung korrekt ist, nämlich dann, wenn ein Attribut, das bisher unbekannt ist, einen Wert erhält. Im verallgemeinerten Case–Memory handelt es sich um ein komplexes Problem, zu ermitteln, wann eine Generalisierung mit bisher bekanntem Wissen konsistent ist.

9.3 Indexgenerierung

Wir werden hier Möglichkeiten aufgezeigen, wie man die Suche im Case–Memory weiter unterstützen kann. Dabei liegt unser Hauptgewicht auf der Frage: welche syntaktischen Möglichkeiten bietet die Default–Logik? Eine ganz andere Fragestellung, die hier nicht untersucht wird, ist: wie findet man überhaupt Heuristiken zur Generierung von Indizes?

In Abschnitt 6.3.3 wurden zwei Arten der Indexgenerierung genannt:

1. **Die Ableitung von Indizes aus dem Eingabefall**
   In unserem Modell des Case–Memory werden Indizes ohne Ableitung direkt aus der Eingabe übernommen. Eine mögliche Verallgemeinerung kann mit Hilfe von Regeln realisiert werden, die Zusammenhänge zwischen verschiedenen Eigenschaften herstellen. Zum Beispiel sei ein direkter Zusammenhang zwischen der Warnlampe W13 und dem Zustand einer Komponente A einer Maschine gegeben, dann läßt sich eine einfache Regeln angeben:

   $norm(x, W13, an) \Rightarrow norm(x, A, abgeschaltet)$

   Damit kann man $norm(q, A, abgeschaltet)$ als Index verwenden, wenn $norm(q, W13, an)$ bekannt ist.

   Die Regeln können global oder lokal formuliert werden. Eine globale Formulierung geschieht in der Weise, wie im Beispiel angegeben. Globale Regeln bedeuten, daß ein Zusammenhang immer besteht. Bestimmte Zusammenhänge können jedoch auch nur lokal, d.h. nur in gewissen Kontexten, die wir mit Knoten des Case–Memory identifizieren, gültig sein.

   Eine lokale Regel hat z.B. die folgende Form:

   $sub(m, x) \wedge norm(x, W13, an) \Rightarrow norm(x, A, abgeschaltet)$

2. **Die Generierung plausibler Indizes**
   Durch die Beschreibung des Case–Memory mit Defaults, ist es einfach die Erzeugung plausibler Indizes zu modellieren. Ein plausibler Index wird durch einen Default mit einer geringen Priorität beschrieben, der nur dann benutzt wird, wenn sonst kein passender Index mehr gefunden werden kann. Es handelt sich hier wie im vorherigen Punkt um Regeln, jedoch nicht um sichere.

   Eine zweite Möglichkeit um weitere Informationen zu bekommen, wenn keine Indizes mehr traversiert werden können, eröffnen Defaults, die zu anderen Knoten springen. Im Beispiel der Museumsbesuche in Abschnitt 6.3.3 springt man vom Knoten „Museen“ zum Knoten „Reisen“, um dort nach Museumsbesuchen auf den verschiedenen Reisen zu suchen. Eine Formulierung mit Defaults könnte z.B. so aussehen:

   $sub(Museen, x) \wedge norm(x, Plan, findeMuseumsbesuche) \Rightarrow sub(Reisen, x).$

9.4 Optimale Lösungen

In Kapitel 7 definierten wir als Lösungen die Informationen in solchen Pfaden, die bestimmten Bedingungen genügen mußten. Um die Menge der gültigen Lösungspfade einzuschränken, kann man sich Bedingungen überlegen, die weitere Restriktionen der Lösungspfade beschreiben. Eine weitere Möglichkeit, um aus vielen Lösungen eine oder wenige zu bestimmen, bietet die Definition einer Ordnung auf den Lösungen bzw. Lösungspfaden. Diese Ordnung sollte ausdrücken, wann eine Lösung besser ist als eine andere.

Am einfachsten zu behandeln sind Ordnungen, die auf syntaktischen Kriterien beruhen. Oft hat man jedoch nur inhaltliche Kriterien, wann eine Lösung gut ist.

Wenn man an ganz bestimmten Informationen interessiert ist, z.B. an den Diagnosen zu Fehlerbeschreibungen, dann sind solche Lösungen optimal, die die gewünschte Information direkt enthalten, so daß keine weiteren Schritte mehr zur Problemlösung notwendig sind. Ein anderes Kriterium ist die Bevorzugung von Lösungspfaden, die Fälle enthalten, d.h. Lösungspfade, die von der Wurzel bis zu den Blättern führen.

Zwei mögliche syntaktische Kriterien sind:

1. **Die Pfadlänge**
   Optimale Pfade sind die kürzesten Pfade. Der Vorteil der Bevorzugung kurzer Pfade ist, daß Algorithmen mit Breitensuche schnell die beste Lösung finden. Problematisch ist nur, daß diese Lösung u.U. zuwenig an Information enthält.

2. **Die Informationsmenge**
   Die Untersuchung der Informationsmenge dreht die Vor- und Nachteile von (1) gerade um. Es werden nun alle Lösungen betrachtet und i.a. auch berechnet. Bevorzugt sind dann die Lösungen, die die meisten Informationen enthalten. Das kann zweierlei bedeuten: Einerseits kann man die Ordnung aus der Teilmengenbeziehung der Lösungen ableiten. Dann sind alle maximalen Lösungsmengen optimal. Andererseits kann man die Ordnung auch über die Anzahl der Attribut–Wert–Paare in den Lösungen definieren.

Kapitel 10

Verwandte Arbeiten

10.1 Multiple Vererbungshierarchien mit Ausnahmen

Eine große Forschungsrichtung innerhalb der Arbeiten zur Nichtmonotonie ist die Fundierung von multiplen Vererbungssystemen mit Ausnahmen durch nichtmonotone Inferenzsysteme.

Die bisherigen Untersuchungen zum Case–Memory zeigen, daß es Parallelen zu Vererbungshierarchien gibt. Die Verbindungen der Knoten über Indizes können als Unterklassenrelation aufgefaßt werden, denn Generalisierungen werden von Knoten auf ihre Nachfolger vererbt. Es kann jedoch sein, daß die Nachfolger die ererbten Generalisierungen überschreiben. Also finden wir im Case–Memory sowohl Vererbung, als auch eine Art von Ausnahmen bei der Vererbung. Da wir keine Baumstruktur, sondern eine azyklische Graphenstruktur als Case–Memory erlauben, kann es auch multiple Vererbungen geben.

Um zu zeigen wo Unterschiede und Gemeinsamkeiten zwischen Case–Memory und multiplen Vererbungssystemen bestehen, werden im folgenden einige wichtige Probleme und Lösungsansätze in Vererbungssystemen dargestellt.

Eine gute Übersicht über die wichtigsten Aspekte geben [Touretzky+87] und [Donini+90]. Auf diese Artikel werden wir uns insbesondere beziehen. Es gibt noch eine Reihe weiterer Arbeiten, die sich teilweise mit speziellen Gesichtspunkten beschäftigen. Genannt seien hier [Fahlman+81, EtheringtonReiter83, Sandewall86, Horty+87, Etherington87].

Vererbungsnetzwerke mit Ausnahmen entsprechen im Prinzip semantischen Netzen, die eine Klassenhierarchie beschreiben und die Vererbung von Eigenschaften dieser Klassen erlauben. Die Klassenhierarchie wird durch eine „class–of“ – Relation bestimmt. Zusätzlich gibt es eine „is–a“ – Relation, die Individuen und Klassen in Beziehung setzt. Eine Besonderheit von Vererbungshierarchien sind die Ausnahmen, die durch eine weitere Relation zwischen Klassen angegeben werden können.

Formal ist ein Vererbungsnetzwerk ein markierter und gerichteter Graph, dessen Knoten Individuen oder Klassen repräsentieren. Die Kanten beschreiben die verschiedenen Relationen zwischen Knoten.

1. **Positive Verbindungen**
   Diese Verbindungen entsprechen den „is–a“ – bzw. „class–of“ – Relationen. Sie werden durch $a \rightarrow b$ notiert, wenn a, b Knoten bezeichnen. Positive Verbindungen sind als *a ist ein b* zu lesen.

2. **Negative Verbindungen**

Durch negative Verbindungen werden die Ausnahmen beschrieben. Die Schreibweise für *a ist kein b* lautet: $a \not\rightarrow b$. In den Abbildungen werden die negativen Verbindungen durch punktierte Linien angegeben.

An einem Beispiel soll nun erklärt werden, wie ein Vererbungsnetzwerk zu verstehen ist und welche Probleme dabei auftreten.

Beispiel 10.1.1
Das bekannte Wissen ist:

Clyde ist ein Elefant. Clyde ist ein Königselefant. Königselefanten sind Elefanten. Elefanten sind graue Tiere. Königselefanten sind keine grauen Tiere.

Als Graph wird das Beispiel in Abbildung 10.1 gut veranschaulicht.

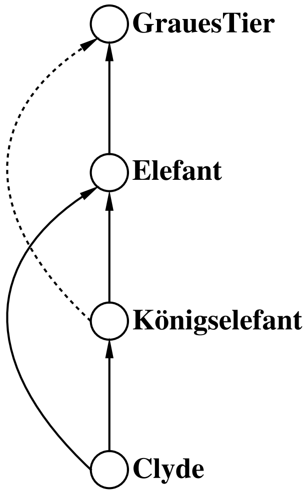

Abbildung 10.1: Clyde der Königselefant

Die zentrale Frage der Vererbung ist nun, ob Clyde ein graues Tier ist oder nicht.

Die Interpretation des Vererbungsnetzwerkes wird durch eine sogenannte *Extension* bestimmt, die durch eine Menge von Pfaden festgelegt wird. Extensionen garantieren die Konsistenz der Pfadmenge. Es gibt zwei Arten von Pfaden:

1. $x_1 \rightarrow x_2 \rightarrow ... \rightarrow x_{n-1} \rightarrow x_n$ ist ein Pfad mit nur positiven Verbindungen.

2. $x_1 \rightarrow x_2 \rightarrow ... \rightarrow x_{n-1} \not\rightarrow x_n$ ist ein Pfad, bei dem nur die letzte Verbindung negativ ist.

Im Prinzip wird durch den Pfad bestimmt, ob x_1 *ein* x_n *ist oder nicht.* Man bezeichnet $x_1 \rightarrow x_n$ bzw. $x_1 \not\rightarrow x_n$ als *Folgerung.*

Beispiel 10.1.2 (Fortsetzung)
In unserem Beispiel von Clyde gibt es drei verschiedene Pfade von „Clyde“ zu „graues Tier“:

1. $Clyde \rightarrow Königselefant \rightarrow Elefant \rightarrow grauesTier$

2. $Clyde \rightarrow Elefant \rightarrow grauesTier$

3. $Clyde \rightarrow Königselefant \not\rightarrow grauesTier$

Wie man an diesem Beispiel sieht, gibt es oft viele mögliche Pfade. Ziel der Forschungsarbeit zur Vererbung ist es vor allem eine Semantik für Netze zu finden, die „richtige“ Pfade bestimmt.

In unserem Beispiel wissen wir, daß der 3. Pfad der gewünschte und richtige ist. Clyde ist also kein graues Tier. Generell ist es aber bisher noch nicht gelungen Kriterien anzugeben, die eindeutig für alle gewünschten Fälle die richtigen Pfade charakterisieren.

Es gibt eine Reihe von Ansätzen, die unterschiedliche Anforderungen an die Pfade stellen, aber für jeden Ansatz gibt es Beispiele von Netzwerken, deren Semantik nicht mit der intendierten Bedeutung übereinstimmt.

Einige Ansätze werden nun kurz skizziert:

1. **Kürzeste Pfade**
   Kürzeste Pfade können durch Breitensuche auf dem Graphen bestimmt werden und sind deshalb schnell zu berechnen. Für Bäume ist die Bestimmung kürzester Pfade beweisbar korrekt, aber für allgemeine Graphen gibt es mindestens zwei Probleme.

   Durch die Transitivität der Unterklassenbeziehung kann man redundante Verbindungen in den Graphen einzeichnen und so neue kürzeste Pfade bekommen, ohne daß sich die Bedeutung des Graphen geändert hätte.

   Der zweite Kritikpunkt ist die fehlende Behandlung von nicht eindeutigen Aussagen. Ein Beispiel für eine nicht eindeutige Aussage ist der bereits in Kapitel 2 erwähnte Nixon–Diamant (Bild 10.2).

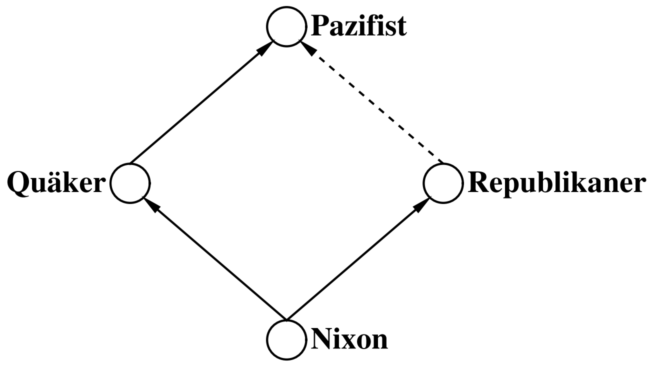

Abbildung 10.2: Der Nixon–Diamant

   Einfache Algorithmen zur Bestimmung kürzester Pfade werden entweder einen möglichen kürzesten Pfad liefern oder mehrere kürzeste Pfade gleichzeitig zulassen und so u.U. Inkonsistenzen erzeugen. Keiner der beiden Möglichkeiten ist theoretisch sauber.

2. **Ordnung der Pfade nach der Ableitungslänge**
   In diesem Ansatz werden die Pfade mit negativer letzter Verbindung $x_0 \rightarrow ... \rightarrow x' \not\rightarrow x$ bevorzugt vor Pfaden $x_0 \rightarrow ... \rightarrow x'' \rightarrow x$, wenn x' eine Unterklasse von x'' ist, d.h. ein Pfad $x' \rightarrow ... \rightarrow x''$ existiert (siehe Abbildung 10.3).

   Doch auch durch diese Ordnung, die in [Touretzky86] ausführlich dargestellt ist, werden nicht alle Wünsche an eine Vererbungssemantik erfüllt. Einige Wünsche an die Interpretation bzgl. bester Pfade werden nun als Gegensatzpaare dargestellt.

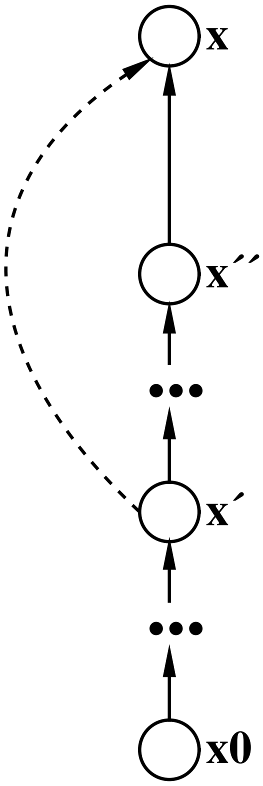

Abbildung 10.3: Die Ableitungslänge

(a) **Kritische oder leichtgläubige Interpretation**
Die kritische Interpretation verhindert im Nixon–Beispiel von oben jede Aussage über Nixons Pazifismus. Eine leichtgläubige Interpretation wird in solchen nicht eindeutigen Fällen mehrere Extensionen generieren. Damit hat man den Nachteil, daß verschiedene Extensionen parallel betrachtet werden müssen, während man bei einer kritischen Interpretation immer eine eindeutige Extension bekommt.

(b) **Top–Down – oder Bottom–Up – Interpretation**
Die normale Sichtweise von Vererbung ist eine Weitergabe von Eigenschaften von Klassen an ihre Unterklassen, die durch eine Vererbungsausnahme (eine negative Verbindung) unterbunden werden kann. Dieser Top–Down – Interpretation kann man jedoch auch die Bottom–Up – Sichtweise entgegensetzen, in der ein Objekt Eigenschaften von seinen Eltern erbt. Beide Interpretationen resultieren in unterschiedlichen Folgerungen aus den Pfaden in bestimmten Fällen.

(c) **Klassische oder intuitive Interpretation der Inkonsistenz**
Durch die klassische Logik werden als Folge einer Inkonsistenz alle Formeln ableitbar und jede unter Umständen sinnvolle Teilinformation geht dadurch verloren. Bei Interpretation von Vererbungsnetzwerken kann man auch Inkonsistenzen feststellen. Die Inkonsistenzen in Teilnetzen stören jedoch nicht die anderen Teile des Netzes.

Beispiele für Inkonsistenzen in Vererbungsnetzen, die noch eine sinnvolle Handhabung erlauben, sind in Abbildung 10.4 angegeben.

Mit (1) können die Sätze „Clyde ist ein Elefant“ oder „Clyde ist kein Elefant“ gebildet werden. Für (2) können beide Pfade korrekt sein, denn es gibt u.U. Elefanten die grau sind und auch andere Elefanten, die es nicht sind.

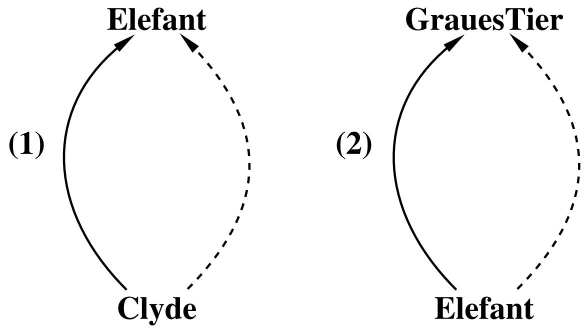

Abbildung 10.4: Inkonsistenzen

Wir haben nun einige Aspekte von Formalisierungen von multiplen Vererbungssystemen mit Ausnahmen gesehen. Daß die Semantik von solchen Systemen nicht vollständig klar ist, ist als Fazit klar herausgekommen. Es gibt jedoch eine Reihe von Ansätzen und Fortschritten im Verständnis der Vererbung mit Ausnahmen.

Der Stand der Forschung ist durch die Entwicklung neuer Semantiken und ihrer Überprufung an Beispielen bestimmt.

Welche Erkenntnisse können wir nun von den Vererbungssystemen auf den Case–Memory übertragen?

Zunächst einmal können wir keine fertigen „Kochrezepte“ erwarten, die genau sagen, wie die Semantik eines Fallspeichers auszusehen hat. Es ist jedoch möglich aus den auftretenden Problemen zu lernen, sowie aus den möglichen Semantiken die Grundideen zu extrahieren und auf den Case–Memory zu übertragen.

Ein Schluß der damit naheliegt ist der folgende: Unsere Formalisierung des Case–Memory ist nicht die einzig mögliche, sondern es gibt andere, die vielleicht in irgendeiner Weise besser sind. Eine andere Formalisierung betrachten wir im nächsten Abschnitt.

10.2 Der Ansatz von Koton und Chase

In [KotonChase89] wird eine Formalisierung des Inferenzmechanismus im fallbasierten Schließen basierend auf der Default–Logik von Reiter angegeben. Wir nennen diesen Ansatz KC–Case–Memory. Es wird in [KotonChase89] zunächst ein informelles Modell des Case–Memory beschrieben, das weitgehend dem formalen Modell nach Abschnitt 7.1 entspricht.

Die Kodierung des KC–Case–Memory in eine prädikatenlogische Sprache ist eine leichte Abwandlung der Formulierung nach Definition 8.1.1. Um eine einheitliche Schreibweise zu verwenden, werden wir die Formeln aus dem Ansatz von Koton und Chase in unsere Syntax transformieren.

Die Default–Logik $\Delta = (D, W)$ umfasst zum einen das sichere Wissen (W):

(F1) $\forall x \;.\; sub(m_0, x) \in W$

(F2) $\forall x \;.\; norm(x, a, v) \Rightarrow \neg norm(x, a, v') \in W$, für alle $v \neq v'$.

Der Fakt (F1) drückt aus, daß M_0 die allgemeinste Generalisierung ist, die jeden Fall subsumiert. (F2) fordert die Funktionalität von $norm$.

Es gibt vier Arten von Defaults (D):

(D1) $sub(m,x) \Rightarrow norm(x,a,v) \in D$, wenn $(a,v) \in NORM_M$ und $\nexists(M,a,v',M') \in DIFF$

(D2) $sub(m,x) \wedge norm(x,a,v) \Rightarrow sub(m',x) \in D$, wenn $(M,a,v,M') \in DIFF$.

(D3) $sub(m,x) \wedge \neg norm(x,a,v_1) \wedge ... \wedge \neg norm(x,a,v_n) \Rightarrow norm(x,a,v) \in D$, wenn $(a,v) \in NORM_M$ und $\forall 1 \leq i \leq n \; . \; \exists M_i \in \mathcal{M} \; . \; (M,a,v_i,M_i) \in DIFF$.

(D4) $sub(m,x) \wedge \neg norm(x,a_1,v_1) \wedge ... \wedge \neg norm(x,a_n,v_n) \Rightarrow \neg norm(x,a,v) \in D$, wenn $\exists(a,v') \in NORM_M$ und $\exists M' \in \mathcal{M}$ mit $\forall 1 \leq i \leq n \; . \; (M,a_i,v_i,M') \in DIFF$ und $(M,a,v,M') \in DIFF$

Die Regeln (D1) und (D3) drücken aus, daß die Generalisierungen Informationen über den Fall enthalten, die aber u.U. überschrieben werden können. (D2) realisiert die Traversierung von Indizes und damit die Bestimmung spezieller subsumierender Knoten. (D4) soll nach Koton und Chase eine eingeschränkte Art der Closed–World–Assumption bewirken.

Beispiel 10.2.1
In Bild 10.5 auf Seite 85 ist ein Case–Memory skizziert.

Eingabe:
$norm(Tweety, cover, feathers), norm(Tweety, feet, claw),$
$norm(Tweety, habitat, water)$

Default–Theorie:
$D = \{sub(m_0,x) \Rightarrow norm(x, cover, feathers), sub(m_0,x) \wedge norm(x, feet, web) \Rightarrow sub(m_2,x), ...,$
$sub(m_1,x) \wedge \neg norm(x, habitat, water) \Rightarrow \neg norm(x, diet, fish), ...\}$[1]

Extension:
$E \subseteq \{sub(m_0, Tweety), sub(m_1, Tweety), sub(m_2, Tweety), sub(m_3, Tweety),$
$norm(Tweety, cover, feathers), norm(Tweety, feet, claw), norm(Tweety, habitat, water),$
$norm(Tweety, diet, fish)\}$

Eine ausführliche Erörterung der oben genannten Defaults würde den Rahmen dieser Arbeit sprengen. Einige Bemerkungen sollen jedoch die Gemeinsamkeiten bzw. Unterschiede zu Transformation 8.2.1 verdeutlichen.

Die prädikatenlogische Sprache des KC–Case–Memory lieferte das Vorbild für die in Definition 8.1.1 schließlich verwendete Sprache. Die Fakten wurden ebenfalls sinngemäß übernommen und erweitert. Von den Defaults (D1) und (D2) finden sich die Grundideen, nämlich das Durchlaufen des Graphen von der Wurzel zu den Blättern (D2) und die Berechnung des Informationsgehalts (D1) in Transformation 8.2.1 wieder.

Die vorliegende Arbeit enthält jedoch entscheidende Unterschiede. Als wichtigster Unterschied ist die formale Definition des Case–Memory zu nennen, die es erlaubt Korrektheits- und Vollständigkeitsaussagen über die Defaulttheorie zu formulieren und zu beweisen. Ein weiterer Punkt ist die Beschränkung auf Pfade. Im KC–Case–Memory gibt es keine Ausschlußbedingungen, die garantieren, daß alle subsumierenden Knoten auf einem Pfad liegen. Im Beispiel gilt:

$$sub(m_0, Tweety), sub(m_1, Tweety), sub(m_2, Tweety), sub(m_3, Tweety)$$

und M_0, M_1, M_2, M_3 liegen nicht auf einem Pfad.

Außerdem behaupten Koton und Chase, es gebe in ihrem Ansatz eine eindeutige Extension, während es nach unserm Ansatz mehrere Extension gibt. Dabei wird nicht genau klar, welche

[1] Es werden nur einige Defaults beispielhaft dargestellt. Die Übrigen lassen sich einfach aus Bild 10.5 herleiten.

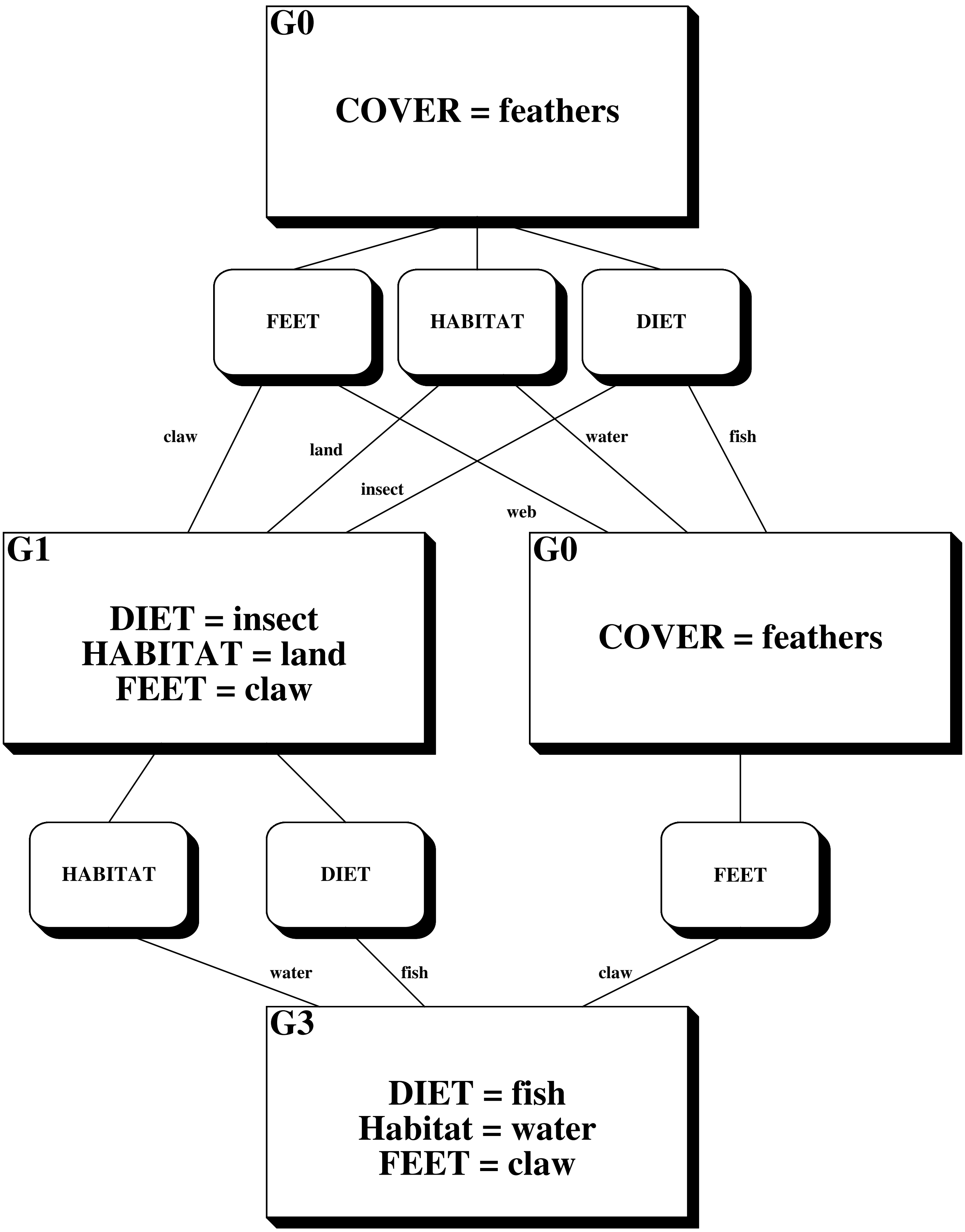

Abbildung 10.5: Ein Beispiel für den KC–Case–Memory

Einschränkungen des Case–Memory notwendig sind, um die Eindeutigkeit der Extension garantieren zu können, denn für den allgemeinen Case–Memory kann es viele mögliche Lösungen und damit auch Extensionen geben.

Da Koton und Chase nicht genau definieren, welches Modell des Case–Memory ihrer Formalisierung zugrunde liegt, ist es schwierig fundierte Kritik zu üben. Es ist jedoch möglich Beispiele für Fallspeicher anzugeben, in denen die Defaults des KC–Case–Memory falsche Ergebnisse liefern.

Die Defaults, insbesondere (D3) und (D4), sind sehr komplex, da sie alle möglichen Ausnahmen explizit nennen. Eine Umstrukturierung des Case–Memory erfordert deshalb sehr aufwendige Änderungen der Default–Theorie. Die Verwendung von Prioritäten in unserem Ansatz erlaubt eine wesentlich einfachere Formulierung der Defaults.

Kapitel 11

Zusammenfassung und Ausblick

Das Ziel der Arbeit bestand in der logischen Fundierung des fallbasierten Schließens. Ein Schwerpunkt lag auf der Repräsentation des Fallspeichers durch eine prädikatenlogische Sprache, die eine adäquate Darstellung der Graphenstruktur, der Indizes und der Generalisierungen lieferte. Das zweite eng mit der Repräsentation verknüpfte Thema war die Übertragung des Retrievalprozesses auf den Inferenzprozeß einer nichtmonotonen Default–Logik mit Prioritäten.

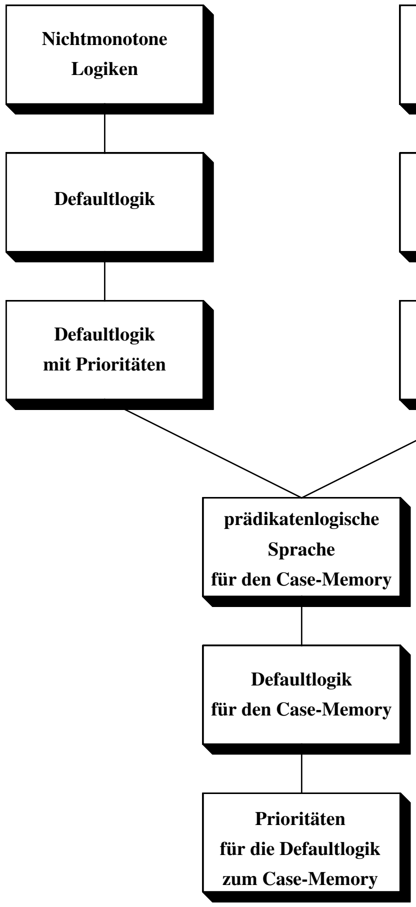

Abbildung 11.1: Eine Übersicht

Die Vorgehensweise zur Erreichung des oben genannten Ziels ist in Bild 11.1 dargestellt.

Zunächst mußten bereits vorhandene Ergebnisse aus der Forschung aufgearbeitet und die relevanten Resultate extrahiert werden. Der erste auffällige Punkt, der neuer Ideen bedurfte, war die fehlende Formalisierung des Fallspeichers und des Retrieval. Aufbauend auf dem entwickelten formalen Modell des Case–Memory und der Definition eines Lösungsbegriffes, der im Prinzip eine Spezifikation des Retrievalalgorithmus ist, konnte eine Default–Theorie mit Prioritäten entwickelt werden. Es wurde als zentrales Ergebnis gezeigt, daß diese Default–Theorie einer Vollständigkeits- und einer Korrektheitsanforderung genügt.

Einige Möglichkeiten zur Erweiterung der Default–Logik wurden bereits in Kapitel 9 genannt. Hier wird noch eine zusätzliche Problematik erörtert. Das Zusammenwirken und die Interaktion von verschiedenen Defaults ist im allgemeinen nicht leicht zu überblicken. Teilweise merkt man erst bei der Durchführung von Beweisen welche Abängigkeiten zwischen Defaults bestehen. Das Problem tritt nicht nur in unserer Default–Theorie auf, aber hier bietet sich die Gelegenheit nach einfachen Defaults zu suchen und eventuell eine Partitionierung in voneinander unabhängige Teiltheorien durchzuführen, da wir Informationen über den Aufbau des Case–Memory u.U. gezielt ausnützen können. Ein wichtiges Hilfsmittel zur Reduzierung der Komplexität sind die Prioritäten. Sie haben in unserem Ansatz dazu beigetragen die Anzahl der Formeln pro Default zu reduzieren, da die Bevorzugung nicht in die Defaults kodiert werden muß.

Weitere Ansätze zur Erweiterung bietet die folgende Beobachtung: Das fallbasierte Schließen besteht nicht nur aus dem Retrieval, sondern eine weitere zentrale Aufgabe ist die Organisation und die Reorganisation des Speichers. Es wäre eine interessante und anspruchsvolle Arbeit auch diesen Teilaspekt aufbauend auf unserer Modellierung logisch zu untermauern. Es stellt sich die Frage, ob die vorhandenen Ausdrucksmittel der Default–Logik ausreichen, denn eine Reorganisation erfordert u.U. die Änderung, das Löschen und die Generierung von Defaults.

Zusammenfassend läßt sich sagen, daß das angestrebte Ziel mit dieser Arbeit erreicht wurde. Es wird jedoch nicht der Anspruch erhoben, es handele sich um die optimale Formalisierung des fallbasierten Schließens. Vielmehr haben wir einen kleinen Schritt gemacht, der durch die vorhandenen CBR–Systeme verifiziert, verbessert und ergänzt werden muß. Die vorliegende Arbeit kann als Ausgangsbasis für weitere Forschungen im Spannungsfeld von fallbasiertem Schließen und nichtmonotonen Logiken gesehen werden.

Literaturverzeichnis

[AAAI86] AAAI. *Proceedings of the Fifth National Conference of the AAAI.* Morgan Kaufmann Publishers, 1986.

[BrachmanLevesque89] Ronald J. Brachman und Hector J. Levesque (Herausgeber). *Proceedings of the First International Conference on Priciples of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, Inc, 1989.

[BrachmanLevesque91] Ronald J. Brachman und Hector J. Levesque (Herausgeber). *Proceedings of the Second International Conference on Priciples of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, Inc, 1991.

[Brewka89] Gerhard Brewka. Preferred Subtheories: An Extended Logical Framework for Default Reasoning. In: *Proceedings of the Eleventh International Joint Conference on Artifical Intelligence*, Seite: 1043 – 1048. IJCAI, 1989.

[Brewka91] Gerhard Brewka. Cumulative Default Logic: In Defense of Nonmonotonic Inference Rules. to appear AI, Summer 1991.

[Church36] A. Church. An unresolvable problem of number theory. *American Journal of Mathematics*, 58:345 – 363, 1936.

[DelgrandeJackson91] James P. Delgrande und W. Ken Jackson. Default Logic Revisited. In: Brachman und Levesque [BrachmanLevesque91], Seite: 118 – 127.

[Donini+90] F.M. Donini, M. Lenzerini, D. Nardi, F. Pirri und M. Schaerf. Nonmonotonic Reasoning. *Artificial Intelligence Review*, 4:163 – 210, 1990.

[EtheringtonReiter83] David W. Etherington und Raymond Reiter. On Inheritance Hierarchies with Exceptions. In: *Proceedings of the AAAI 1983*, Seite: 104 – 108. AAAI, 1983.

[Etherington87] David W. Etherington. Formalizing Nonmonotonic Reasoning Systems. *Artificial Intelligence*, 31:41 – 85, 1987.

[Etherington88] David Etherington. *Reasoning with Incomplete Information.* Morgan Kaufmann, 1988.

[Fahlman+81] S. E. Fahlman, D.S. Touretzky und W. van Roggen. Cancellation in parallel semantic network. In: *Proceedings of the Seventh International Joint Conference on Artifical Intelligence*, Seite: 257 – 262. IJCAI, 1981.

[GelfondLifschitz+91] Michael Gelfond, Vladimir Lifschitz, Halina Przymusinska und Miroslaw Truszczynski. Disjunctive Defaults. In: Brachman und Levesque [BrachmanLevesque91], Seite: 230 – 237.

[HollandHolyoak86] John H. Holland, Keith J. Holyoak, Richard E. Nisbett und Paul R. Thagard. *Induction: Process of Inference, Learning, and Discovery.* MIT Press, 1986.

[HanksMcDermott86] S. Hanks und Drew McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In: *Proceedings of the Fifth National Conference of the AAAI* [AAAI86], Seite: 328 – 333.

[HanksMcDermott87] S. Hanks und Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379 – 412, 1987.

[Horty+87] John F. Horty, Richmond H. Thomason und David S. Touretzky. A skeptical theory of inheritance in nonmonotonic semantic nets. In: *Proceedings of the AAAI-87.* AAAI, Morgan Kaufmann Publishers, 1987.

[Junker91] Ulrich Junker. Prioritized Defaults: Implementation by TMS and Application to Diagnosis. Accepted for the IJCAI 91, 1991.

[Kautz86] Henry A. Kautz. The logic of persistence. In: *Proceedings of the Fifth National Conference of the AAAI* [AAAI86], Seite: 401 – 405.

[KotonChase89] Phyllis Koton und Melissa P. Chase. Knowledge Representation in a Case-Based Reasoning System: Defaults and Exceptions. In: Brachman und Levesque [BrachmanLevesque89], Seite: 203 – 211.

[Kolodner83a] Janet L. Kolodner. Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science*, 7:243–280, 1983.

[Kolodner83b] Janet L. Kolodner. Reconstructive Memory: A Computer Model. *Cognitive Science*, 7:281–328, 1983.

[Kolodner84] Janet L. Kolodner. *Retrieval and organizational strategies in conceptual memory: A computer model.* Erlbaum, 1984.

[KautzSelman89] Henry A. Kautz und Bart Selman. Hard Problems for Simple Default Logics. In: Brachman und Levesque [BrachmanLevesque89], Seite: 189 – 197.

[Lifschitz85] Vladimir Lifschitz. Computing Circumscription. In: *Proceedings of the Ninth International Joint Conference on Artifical Intelligence*, Seite: 121 – 127. IJCAI, Morgan Kaufmann, 1985.

[Lifschitz86] Vladimir Lifschitz. On the satisfiability of circumscription. *Artificial Intelligence*, 28:17 – 27, 1986.

[Lifschitz86b] Vladimir Lifschitz. Pointwise circumscription: A preliminary report. In: *Proceedings of the Fifth National Conference of the AAAI* [AAAI86], Seite: 406 – 410.

[LinShoham89] Fangthen Lin und Yoav Shoham. Argument Systems: a uniform basis for nonmonotonic reasoning. In: Brachman und Levesque [BrachmanLevesque89], Seite: 245 – 255.

[Makinson89] David Makinson. General Theory of Cumulative Inference. In: *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, number 346 in LNAI, Seite: 1 – 18. Springer, 1989.

[McCarthy80] John McCarthy. Circumscription – A Form of Non-monontonic Reasoning. *Artificial Intelligence*, 13:27 – 39, 1980.

[McCarthy86] John McCarthy. Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artificial Intelligence*, 28:89 – 116, 1986.

[McDermottDoyle80] Drew McDermott und John Doyle. Non-Monotonic Logic I. *Artificial Intelligence*, 13:41 – 72, 1980.

[McCarthyHayes69] John McCarthy und Patrick J. Hayes. Some philospohical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463 – 502, 1961.

[Minsky75] Marvin Minsky. A framework for representing knowledge. In: Patrick H. Winston (Herausgeber), *The Psychology of Computer Vision*, Seite: 211 – 277. McGraw-Hill, 1975.

[Minsky91] Marvin Minsky. Interview. *KI*, 2, 1991.

[Moore84] Robert C. Moore. Possible World Semantics for Autoepistemic Logic. In: *Proceedings of the 1984 Nonmonotonic Reasoning Workshop*, Seite: 344 – 354. NMR, 1984.

[Moore85] Robert C. Moore. Semantical Considerations on NML. *Artificial Intelligence*, 25:75 – 94, 1985.

[Pearl88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publisher, Inc., 1988.

[PooleGoebelAleliunas87] David L. Poole, R. G. Goebel und R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In: N. Cercone und G. McCalla (Herausgeber), *The Knowledge Frontier: Essays in the Representation of Knowledge*, Seite: 331 – 352. Springer, 1987.

[Poole88] David Poole. A Logical Framework for Default Reasoning. *Artificial Intelligence*, 36:27 – 47, 1988.

[Reiter80] Raymond Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81 – 132, 1980.

[Reiter87b] Raimond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:952 – 957, 1987.

[Reiter87] Raymond Reiter. Nonmonotonic Reasoning. *Annual Review of Computer Science*, 2:147 ff, 1987.

[Richter89] Michael M. Richter. *Prinzipien der künstlichen Intelligenz.* B.G. Teubner, 1989.

[DARPA87] Edwina Rissland, Janet Kolodner und David Waltz (Herausgeber). *Proceedings of the 1987 DARPA Worhshop on Case-Based-Reasoning.* DARPA, 1987.

[DARPA88] Edwina Rissland, Janet Kolodner und David Waltz (Herausgeber). *Proceedings of the 1988 DARPA Worhshop on Case-Based-Reasoning.* DARPA, Clearwater Beach , FL, May 1988.

[RisslandKolodnerWaltz89] Edwina Rissland, Janet Kolodner und David Waltz. Proceedings from the Case-Based Reasoning Workshop, Pensacola Beach, Florida. In: *Proceedings from the Case-Based Reasoning Workshop, Pensacola Beach, Florida* [DARPA89], Seite: 1 – 13.

[DARPA89] Edwina Rissland, Janet Kolodner und David Waltz (Herausgeber). *Proceedings of the 1989 DARPA Worhshop on Case-Based-Reasoning.* Morgan Kaufmann, 1989.

[RichterWendel91] Michael M. Richter und Oliver Wendel. Lernende Systeme. (Unveröffentlichtes Vorlesungsskript), 1991.

[SchankAbelson77] Roger C. Schank und R. P. Abelson. *Scripts, plans, goals and understanding.* Erlbaum, 1977.

[Sandewall72] E. Sandewall. An approach to the frame problem and its implementation. *Machine Intelligence*, 7:195 – 204, 1972.

[Sandewall86] E. Sandewall. Non-monotonic inference rules for multiple inheritance with exceptions. In: *Proceedings of the IEEE*, volume 74, Seite: 1345 – 1353, 1986.

[Schank82] Roger C. Schank. *Dynamic Memory: A theory of reminding and learning in computers and people.* Cambridge University Press, 1982.

[Shoham86] Yoav Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* Dissertation, Yale University, 1986.

[Slade91] Stephen Slade. Case-Based Reasoning: A Research Paradigm. *AI Magazine*, 1:42 – 55, 1991.

[Touretzky+87] David S. Touretzky, John F. Horty und Richmond H. Thomason. A Clash of Intuitions: The Current State of Nonmonotonic Multiple Inheritance Systems. In: *Proceedings of the Tenth International Joint Conference on Artifical Intelligence*, Seite: 476 – 482. IJCAI, 1987.

[Touretzky86] David S. Touretzky. *The Mathematics of Inheritance Systems.* Morgan Kaufman Publishers, 1986.

[Tversky77] Amos Tversky. Features of Similarity. *Psychological Review*, 84:327 – 352, 1977.

[Wess90] Stefan Weß. PATDEX/2 - ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen. Diplomarbeit, Universität Kaiserslautern, 1990.