Entwurf und Implementierung eines Algorithmus zum wissensintensiven Lernen von Planabstraktionen nach der PABS-Methode

Wolfgang Wilke

Betreuung:
Prof. Dr. Michael M. Richter
Dipl. Inform. Ralph Bergmann

Projektarbeit
Universität Kaiserslautern

Inhaltsverzeichnis

1 Einleitung **1**

2 Planen mit Abstraktion **3**

2.1 Einsatzmöglichkeiten der Abstraktion bei der Planung 3

2.2 Beschreibung des PABS-Verfahrens 4

2.2.1 Benötigte Eingabedaten des PABS-Verfahrens 4

2.2.2 Phase-I: Simulation des konkreten Plans 5

2.2.3 Phase-II: Abbildung der konkreten auf die abstrakten Zustände . . . 5

2.2.4 Phase-III: Suche nach Zustandsübergängen der abstrakten Operatoren 6

2.2.5 Phase-IV: Die Suche nach konsistenten Pfaden vom Start- zum Zielzustand 7

2.2.6 Phase-V: Transformation des abstrakten Plans in einen Skelettplan . 7

2.3 Ein Beispiel: Towers of Hanoi (ToH) 8

3 Das PABS-Verfahren **11**

3.1 Grundlegende Definitionen zur Planabstraktion nach PABS 11

3.1.1 Die Definition einer Planungswelt 11

3.1.2 Die Definition von Abstraktionen zwischen zwei Planungswelten . . . 13

3.1.3 Definition einer wissensintensiven Theorie zur Herleitung von abstrakten Zustandsbeschreibungen 15

3.1.4 Die Definition eines Skelettplans 16

3.2 Algorithmische Beschreibung des PABS- Verfahrens 19

3.2.1 Die benötigten Eingabedaten für das PABS-Verfahren 20

3.2.2 Phase-I: Simulation des konkreten Plans 20

3.2.3 Phase-II: Die Anwendung der generischen Theorie 21

3.2.4 Phase-III: Bestimmung der relevanten Operatoren 21

3.2.5 Phase-IV: Konsistenzprüfung 23

3.2.6 Phase-V: Skelettplantransformation 25

3.2.7 Die Gesamtkomplexität des PABS-Verfahrens 26

4 Implementierung des Verfahrens **29**

4.1 Definition von Weltbeschreibungen mit PABS 29

4.1.1 Konkrete Pläne und ihre Startzustände 30

4.1.2 Die konkrete Weltbeschreibung 31

4.1.3 Die generische Theorie 33

4.1.4 Die abstrakte Weltbeschreibung 34

4.1.5 Die Definition von built-in Prädikaten 36

4.2 Realisierung des PABS-Verfahrens in Prolog 36

4.2.1 Der Beweiser des PABS-Systems 37

4.2.2 Einladen und Starten des PABS-Systems 38

4.2.3 Das PABS-System im Debug-mode 38

5 Diskussion **43**

A Domänenspezifikation Towers of Hanoi (ToH) **45**

B Domänenspezifikation Add **49**

C Beispiellauf **53**

Kapitel 1

Einleitung

Planabstraktion ist eine Möglichkeit, den Aufwand bei der Suche nach einem Plan zur Lösung eines konkreten Problems zu reduzieren. Hierbei wird eine konkrete Welt mit einer Problemstellung auf eine abstrakte Welt abgebildet. Die abstrakte Problemstellung wird nun in der abstrakten Welt gelöst. Durch die Rückabbildung der abstrakten Lösung auf eine konkrete Lösung erhält man eine Lösung für das konkrete Problem.
Da die Anzahl der zur Lösung des abstrakten Problems benötigten Operationen geringer ist und die abstrakten Zustände und Operatoren einer weniger komplexen Beschreibung genügen, wird der Aufwand zur Suche einer konkreten Problemlösung reduziert.
Das PABS-Verfahren ist ein mehrschrittiges Verfahren zur Ermittlung von abstrakten Plänen. Diese Pläne lösen in der abstrakten Planungswelt Planungsprobleme, die einem gegebenen konkreten Plan, der ein Problem in der konkreten Welt löst, entsprechen.
Zuerst wird ein vorgegebener konkreter Plan in einer konkreten Welt propagiert. Danach wird diese konkrete Welt in eine abstrakte Welt abgebildet. Anschließend werden alle abstrakten Pfade mit abstrakten Operatoren gesucht, die vom abstrakten Startzustand zum abstrakten Zielzustand führen. Die so gewonnenen Pläne werden einer Konsistenzprüfung unterworfen. Nach Auswahl eines konsistenten Plans wird dieser in einen Skelettplan transformiert. Das PABS-Verfahren ist ausführlich in [Bergmann, 1992b] beschrieben.
Gegenstand dieser Projektarbeit ist der Entwurf einer Implementation des Verfahrens und dessen Realisierung. Die Beschreibung der Arbeit gliedert sich in drei Abschnitte. Im ersten Abschnitt werden das Verfahren selbst sowie die eingehenden Parameter informell beschrieben. Im zweiten Teil werden die theoretischen Grundlagen zur Planabstraktion nach der PABS-Methode erklärt. Im dritten Abschnitt wird dargestellt, wie man die Eingabedaten definiert und mit dem Programm arbeitet. Die Implementation wurde auf einer Sparc Station SLC in Prolog durchgeführt. Anschließend werden Ausblicke und Probleme der Arbeit diskutiert. Im Anhang befinden sich zwei Domänendefinitionen zum Problem Towers of Hanoi (ToH) und eines Additionsprogramms, das von der Bit-Ebene auf die Ebene der natürlichen Zahlen abstrahiert. Außerdem befindet sich dort ein Ausdruck eines Programmlaufs, um die Eingaben, die der Benutzer machen kann, zu erläutern.
Zum Verständnis werden grundlegende Kenntnisse der Prinzipien der Künstlichen Intelligenz und des Planens vorausgesetzt. Zur Einführung in die Künstliche Intelligenz können hier Lehrbücher wie [Charniak and McDermott, 1985] oder [Richter, 1989] dienen. Um sich in den Themenbereich Planen einzuarbeiten, sei hier auf [Hertzberg, 1989] verwiesen, der einen guten Überblick über dieses Thema vermittelt und eine große Auswahl an Algorith-

men zur Planung darstellt. Bei dem Thema Abstraktion und bei anderen speziellen Themen sei auf die zitierten Artikel verwiesen.

Kapitel 2

Planen mit Abstraktion

2.1 Einsatzmöglichkeiten der Abstraktion bei der Planung

Beim Planen geht es darum, eine Folge von Operatoren zu finden, die ein System von einem Ausgangszustand in einen Zielzustand überführen. Diese Folge von Operatoren stellt dann einen Plan zur Lösung eines Planungsproblems dar. Sei m die Anzahl der anwendbaren Operatoren, die in einer Planungswelt angewendet werden können und n die Anzahl der Operationen, die minimal dazu benötigt werden, um den vorgegebenen Startzustand in einen vorgegebenen Zielzustand zu überführen. Die naive Suche nach einer Lösung für das Planungsproblem wächst dann exponentiell mit der Komplexität $O(m^n)$. Deshalb kommt es bei Planungsproblemen, die eine längere Operartorenfolge zur Lösung erfordern, schnell zu einer kombinatorischen Explosion.
Eine Möglichkeit, den Aufwand beim Planen zu reduzieren, ist die Abstraktion [Sacerdoti, 1974]. Ausgehend von einer Problemstellung in einer konkreten vorgegebenen Weltbeschreibung kann man das Problem und die konkrete Welt mit Hilfe einer Abstraktionsabbildung auf eine abstrakte Welt abbilden. In dieser abstrakten Welt ist es dann möglich, mit weniger Zustandstransformationen vom abstrakten Startzustand zum abstrakten Zielzustand zu kommen. Diese Reduzierung ist natürlich abhängig von der Wahl der Abstraktionsabbildung, die die Ebene der Abstraktion der konkreten Welt festlegt. So kann man auf der abstrakten Ebene eine Folge von konkreten Operationen auf dem konkreten Zustandsraum mit Hilfe eines abstrakten Operators beschreiben. Da sich beim Abstrahieren auch die Repräsentationssprache ändert, handelt es sich nicht um einen Makrooperator [Fikes and Nilsson, 1971] für eine Folge von konkreten Operationen, sondern um eine Beschreibung des gleichen Vorgangs, der sich jedoch durch einen geringeren Detailliertheitsgrad auszeichnet. So steht man nun vor der Aufgabe, ein abstraktes Planungsproblem zu lösen, welches jedoch zu dem konkreten Problem äquivalent ist. Dieses abstrakte Planungsproblem läßt sich jedoch mit einer kürzeren Folge von Operationen von einem abstrakten Startzustand in einen abstrakten Zielzustand überführen. Diese kürzere Planlänge führt zu einem geringeren Aufwand bei der Suche nach einem Lösungsplan, da der Aufwand für die Anwendung der Abstraktionsabbildung nur linear mit der Anzahl der abzubildenden konkreten Fakten wächst. Da abstrakte Zustände und Operatoren sich in der Regel durch eine geringere Menge von Fakten charakterisieren lassen, führt dies bei der Prüfung der Anwendbarkeit eines

Operators zu einem zusätzlichen Performanzvorteil.
Nach der Lösung des abstrakten Planungsproblems kann man die so erhaltene abstrakte Operatorenfolge mit der Umkehrung der Abstraktion wieder auf die konkrete Welt abbilden und erhält so eine Lösung für das ursprüngliche konkrete Planungsproblem.
Die oben erwähnte Reduktion bei der Beschreibung von abstrakten Plänen und Zuständen kann man sich auch beim Cased-Based-Reasoning zunutze machen. Hier kann man aus einer Datenbank mit abstrakten Plänen schneller die Anwendbarkeit eines Plans überprüfen, um so einen Plan zu finden, der das gegebene Problem auf der abstrakten Ebene löst, nachdem man den konkreten Start- und Zielzustand mit Hilfe der Abstraktionsabbildung transformiert hat. Der Performanzvorteil ergibt sich hier aus der schnelleren Überprüfung der Anwendbarkeit eines Plans, da die Start- und Zielzustände nicht so detalliert beschrieben werden wie auf der konkreten Ebene. Beim PABS-Verfahren ist die Verifizierbarkeit des abstrakten Plans durch die Repräsentation des abstrakten Plans als Skelettplan zusätzlich vereinfacht, da hier die Überprüfung auf die Anwendbarkeit ohne simulierte Ausführung des abstrakten Plans möglich ist. Wenn man nun einen abstrakten Plan, der das Problem löst, gefunden hat und diesen auf die konkrete Welt abbildet, hat man eine Lösung für das konkrete Ausgangsproblem gefunden.
Das PABS-Verfahren [Bergmann, 1992b] ist ein Algorithmus zur maschinellen Generierung von abstrakten Plänen aus konkreten Plänen. Dieses Verfahren ist in fünf Phasen gegliedert. Im folgenden Abschnitt werden diese Phasen mit den benötigten Eingabedaten, beziehungsweise mit den resultierenden Ausgabedaten informell beschrieben. Die Repräsentation der Planungswelt, der Zustände und der Operatoren basiert auf der STRIPS Notation von [Fikes and Nilsson, 1971]. Sie wird ausführlich im nächsten Kapitel, im Abschnitt über die theoretischen Grundlagen zur Planabstraktion nach PABS, erläutert.

2.2 Beschreibung des PABS-Verfahrens

In diesem Abschnitt soll das PABS-Verfahren informell beschrieben werden, damit der Leser in den folgenden Kapiteln die Notwendigkeit der einzelnen Definitionen begreift und einen Überblick über das PABS-Verfahren erhält. Im folgenden Abschnitt wird das Wissen über die Definition einer Planungswelt in STRIPS teilweise vorausgesetzt; es kann jedoch im folgenden Kapitel nachgelesen werden.

2.2.1 Benötigte Eingabedaten des PABS-Verfahrens

Die Anwendung des PABS-Verfahrens erfordert die Beschreibung einer konkreten Welt. Diese Beschreibung besteht aus:

- einer Menge von konkreten Operatoren in STRIPS Notation, die in der konkreten Welt anwendbar sind.

- einer Menge von Regeln um zusätzliche Fakten herzuleiten. Diese Menge von konkreten Regeln wird als konkrete Theorie bezeichnet.

- Ein Operationalitätskriterium, um eine Generalisierung der Beweise durchzuführen.

Analog benötigt man eine Beschreibung der abstrakten Welt, bestehend aus abstrakten Operatoren, einer abstrakten Theorie und einem abstrakten Operationalitätskriterium.
Des weiteren wird eine generische Theorie [Giordana *et al.*, 1991], bestehend aus Regeln, die die Abstraktionsabbildung realisieren, vorausgesetzt. Außerdem benötigt das PABS-Verfahren eine Folge von Operatoren, die den konkreten Plan darstellen und einen ausgezeichneten Startzustand, auf den dieser Plan angewandt wird.

2.2.2 Phase-I: Simulation des konkreten Plans

In der ersten Phase wird die Ausführung des konkreten Plans simuliert. Dabei werden die Operatordefinitionen der konkreten Operatoren in der Reihenfolge, die durch den Plan vorgegeben ist, nach der STRIPS Methode auf den vorgegebenen Startzustand angewandt. Falls ein Operator aufgrund der geltenden Fakten in einem Zustand nicht anwendbar ist, wird mit Hilfe der konkreten Theorie versucht, die benötigten Fakten zu beweisen. So entstehen Beweise für die Anwendbarkeit der Operatoren und Zwischenzustände, die man speichert. Dieser Vorgang wird in Abbildung 2.1 verdeutlicht.

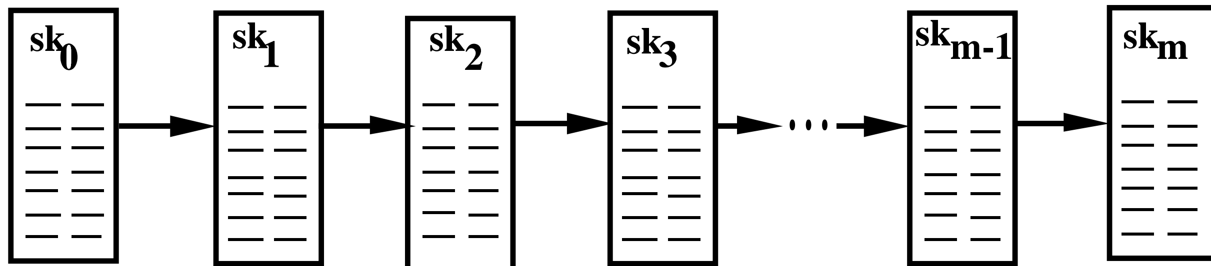

Abbildung 2.1: Simulation des konkreten Plans

2.2.3 Phase-II: Abbildung der konkreten auf die abstrakten Zustände

In der zweiten Phase wird die generische Theorie angewandt, um die durch die Ausführung des Plans entstandenen Zustandsbeschreibungen der konkreten Welt auf abstrakte Zustandsbeschreibungen abzubilden. Diese Abbildung stellt die Realisation der Abstraktion von der konkreten Welt in die abstrakte Welt dar. Auch hier wird die konkrete Theorie angewandt, um Fakten herzuleiten, die die generische Theorie als Vorbedingungen benötigt, um aus den konkreten Zustandsbeschreibungen abstrakte Zustandsbeschreibungen herzuleiten. Dieser Sachverhalt wird in Abbildung 2.2 illustriert.

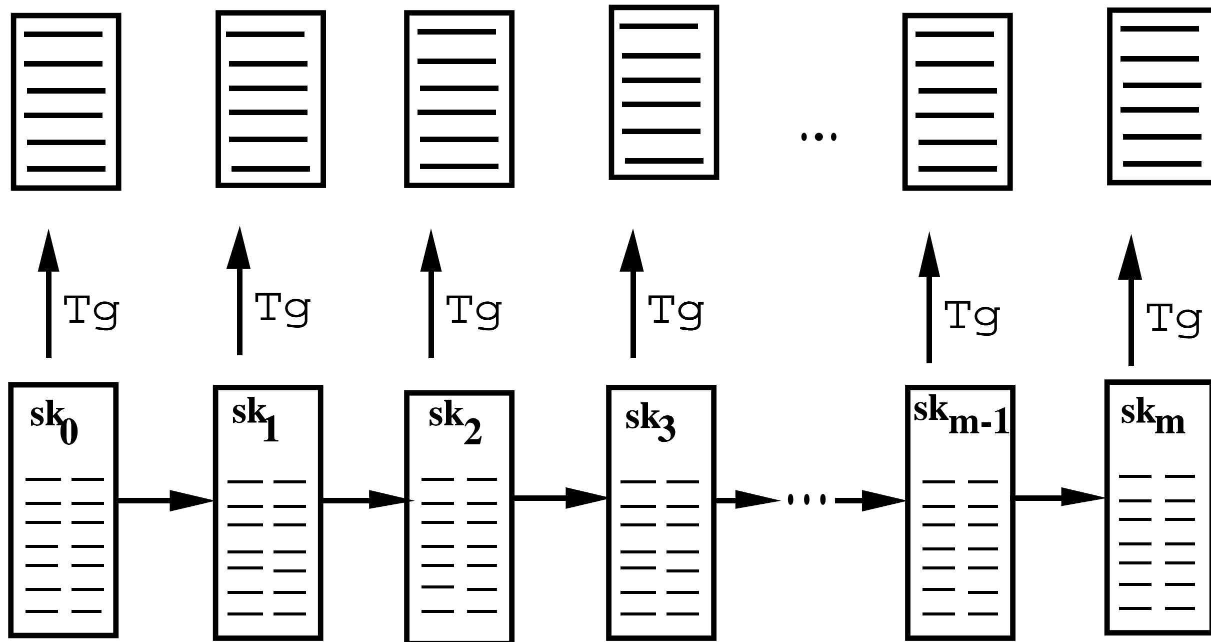

Abbildung 2.2: Abstraktion der konkreten Welt in die abstrakte Welt

2.2.4 Phase-III: Suche nach Zustandsübergängen der abstrakten Operatoren

Das Ziel der dritten Phase ist es, alle abstrakten Operatoren zu finden, die von einem abstrakten Zustand in einen beliebigen Folgezustand führen. Die Zustandsbeschreibungen müßen den STRIPS Definitionen der Operatoren genügen, das heißt, die Vorbedingungen, die ein Operator benötigt, um angewendet werden zu können, müssen in dem gewählten Startzustand gelten. In dem Zielzustand dürfen jedoch keine Fakten mehr gelten, die in der Deleteliste des Operators existieren und alle Fakten der Addliste müssen in dem Zielzustand vorhanden sein. So erhält man einen azyklischen, gerichteten Graphen, in dem die Knoten abstrakte Zustände und die Kanten mögliche abstrakte Zustandsübergänge durch abstrakte Operatoren darstellen (Abbildung 2.3).

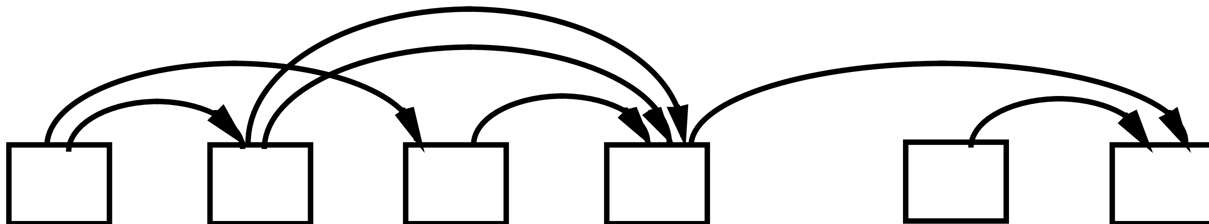

Abbildung 2.3: Graph mit abstrakten Zuständen und Operatoren

2.2.5 Phase-IV: Die Suche nach konsistenten Pfaden vom Start- zum Zielzustand

In der vierten Phase werden alle konsistenten Pfade in der abstrakten Planungswelt gesucht, die vom abstrakten Startzustand, das ist der Zustand, der in der Phase-II mit Hilfe der generischen Theorie aus dem konkreten Startzustand hergeleitet wurde, zum abstrakten Zielzustand führen, der aus dem konkreten Zielzustand hervorging.
Ein Pfad ist konsistent, falls jeder Fakt der Vorbedingungen eines STRIPS Operators, der zu diesem Pfad gehört, einer der folgenden Bedingungen genügt;

- Der Fakt galt im Startzustand und wurde bis zur Anwendung des Operators, der ihn als Vorbedingung benötigt, nicht gelöscht.
- Der Fakt wurde durch einen vorherigen Operator erzeugt und wird nicht wieder gelöscht, bis der Operator, der ihn benötigt, angewandt wird.
- Der Fakt kann aus Fakten, die einer der beiden vorherigen Bedingungen genügen, mit Hilfe der abstrakten Theorie hergeleitet werden.

So ist die Anwendbarkeit der abstrakten Operatoren durch die Anwendung der abstrakten Operatorsequenz selbst garantiert und begründet sich nicht auf Fakten, die aus der generischen Theorie und den konkreten Zuständen hergeleitet wurden.
Die so erhaltenen konsistenten Pfade in der abstrakten Welt, die vom abstrakten Startzustand zum abstrakten Zielzustand führen, sind dann mögliche abstrakte Pläne, die zum konkreten Plan korrespondieren. Aus diesen Plänen kann der Benutzer einen Plan wählen, der dann in der fünften Phase in einen Skelettplan transformiert wird (siehe Abbildung 2.4).

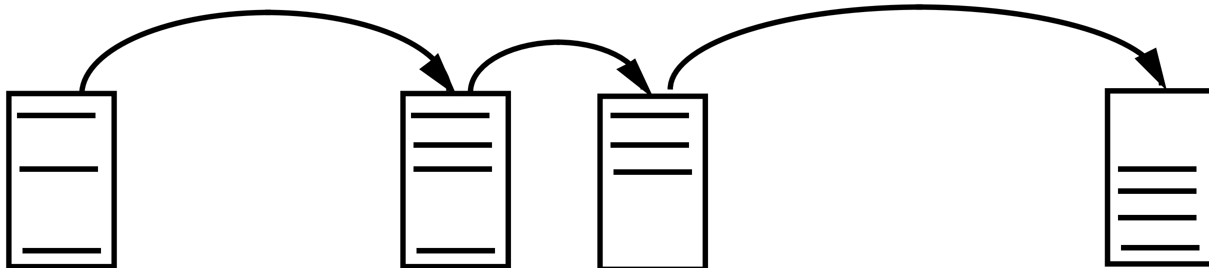

Abbildung 2.4: Die abstrakten Zustände mit einem konsistenten Pfad

2.2.6 Phase-V: Transformation des abstrakten Plans in einen Skelettplan

In der letzten Phase wird der vom Benutzer ausgewählte Plan in einen Skelettplan [Friedland and Iwasaki, 1985; Bergmann, 1992a] transformiert. Skelettpläne sind eine Folge von allgemeinen Aktionen, die, wenn sie für ein spezifisches Problem geeignet spezialisiert werden, einen Plan ergeben, der das Problem löst. Skelettpläne versuchen zum einen, schädliche

Inferenzen zwischen einzelnen Planungsschritten zu verhindern und zum anderen, nützliche darzustellen und sie gezielt auszunutzen. Die Definition der Repräsentation von Skelettplänen findet man im folgenden Kapitel im ersten Abschnitt. Das Verfahren der Transformation wird detalliert in [Bergmann, 1990] beschrieben.

2.3 Ein Beispiel: Towers of Hanoi (ToH)

In diesem Abschnitt wird noch einmal die Beispielwelt vorgestellt, an der das PABS-System erklärt wird. Es handelt sich um das Towers of Hanoi Problem, bei dem es darum geht, einen Turm mit Scheiben, von einem Stapel auf einen anderen Stapel zu bewegen. Hierbei stehen drei Stapel zur Verfügung, auf denen die Scheiben abgelegt werden können. Es darf bei der Bewegung der Scheiben jedoch niemals eine kleinere Scheibe auf einer größeren Scheibe liegen. In der Abbildung 2.5 ist das Problem und der Lösungsweg noch einmal in der konkreten Welt verdeutlicht. Die drei verschiedenen Stapel werden hier mit den Buchstaben (a,b,c) bezeichnet, die drei Scheiben mit den Zahlen (1,2,3) entsprechend ihrer Größe. Als einziger Operator läßt sich in der konkreten Welt der Operator *move(Source,Destination)* anwenden, der unter Beachtung der obigen Einschränkungen eine Scheibe von einem Stapel auf den anderen Stapel bewegt.

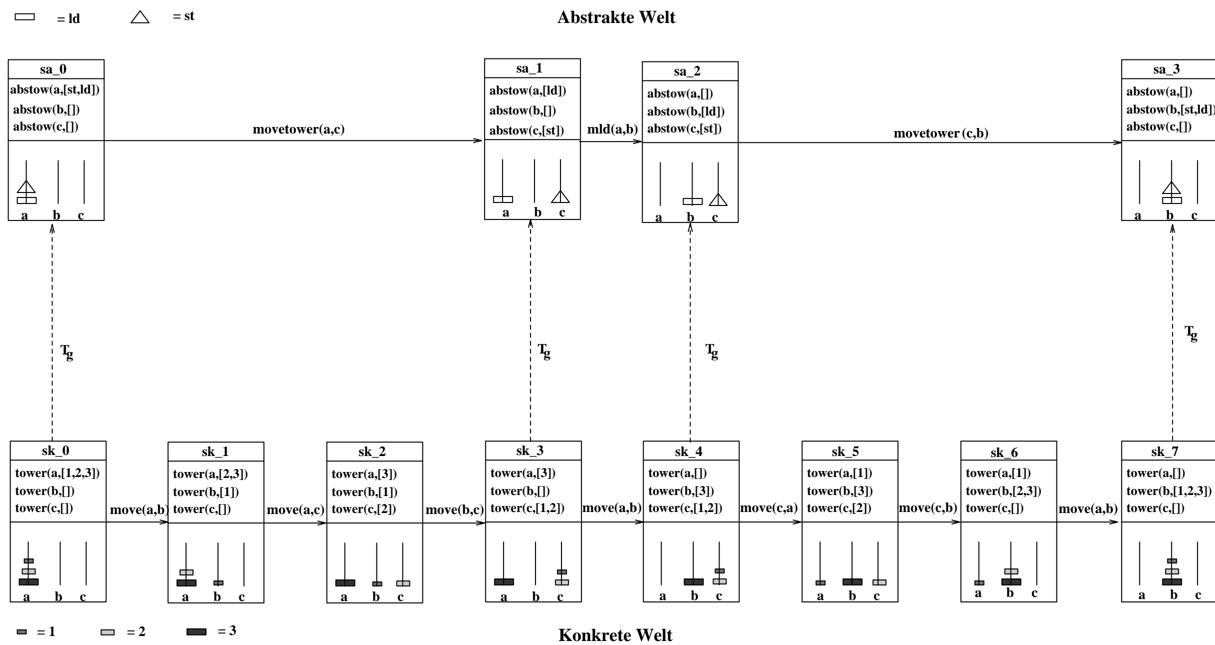

Abbildung 2.5: Beispiel: Towers of Hanoi

In der abstrakten Welt bleiben die drei verschiedenen Stapel und ihre Bezeichnungen erhalten. Die oberen Scheiben eines Turms werden jedoch zu einem small tower (st) zusammengefaßt. Beim dem hier dargestellten 3-Scheiben Problem handelt es sich um die Scheiben (1,2). Die größte Scheibe wird dort mit large Disk (ld) bezeichnet. Hier handelt es sich um die Scheibe (3). In dem ausgewählten Plan werden die abstrakten Operatoren

movetower(Source,Destination) und *mld(Source,Destination)* verwandt. Der Operator *movetower(Source,Destination)* bewegt den small tower (st) von einem Stapel auf den anderen und der Operator *mld(Source,Destination)* [1] bewegt die large Disk (ld). Im folgenden dient dieses Beispiel zur Erklärung des PABS-Verfahrens.

[1] *mld* steht für *movelargedisk*

Kapitel 3

Das PABS-Verfahren

Dieses Kapitel gliedert sich in zwei Abschnitte, wobei im ersten Abschnitt der formale Hintergrund des PABS-Verfahrens und im darauffolgenden Abschnitt die von mir gewählten Algorithmen zur Implementation beschrieben werden.

3.1 Grundlegende Definitionen zur Planabstraktion nach PABS

3.1.1 Die Definition einer Planungswelt

Das Ziel der Planabstraktion ist es, eine Beziehung zwischen einer konkreten Planungswelt und einer abstrakten Planungswelt zu finden. Dabei sollte sich die konkrete Welt durch einen größeren Detailliertheitsgrad in der Beschreibung der Zustände und der Operatoren gegenüber der abstrakten Planungswelt auszeichnen. Daraus ergibt sich die Notwendigkeit eine Sprache zu definieren, die die Operatoren und die Zustände in einer Planungswelt beschreibt.

Definition 3.1 *Eine Sprache L zur Beschreibung einer Planungswelt besteht aus drei Mengen von Termen $L = E \cup A \cup B$, die paarweise disjunkt sind, mit:*

- *E einer Menge von essentiellen Sätzen.*
- *A einer Menge von ableitbaren Sätzen.*
- *B einer Menge von built-in Prädikaten.*

Die Notwendigkeit der Aufteilung der Sprache L in drei Komponenten wird in den folgenden Definitionen verdeutlicht. Zuerst wird eine formale Beschreibung einer Planungswelt gegeben.

Definition 3.2 *Eine Planungswelt W in STRIPS Notation besteht aus einem Tripel (R, T, Op) über einer prädikatenlogischen Sprache L erster Ordnung, wobei:*

- $R \subseteq E$ *eine Menge von essentiellen Sätzen [Lifschitz, 1987] darstellt, mit denen ein Zustand in der Planungswelt zu einem festen Zeitpunkt beschrieben werden kann.*
- T *ist eine Theorie, definiert durch eine Menge von Regeln* r*, sodaß:*
  $r = l_1 \wedge l_2 \wedge \ldots \wedge l_n \rightarrow \Phi$*, wobei* $l_i \in L = E \cup A \cup B$ *für* $i = 1, \ldots, n$ *und* $\Phi \in A$*. Hierbei ist die Konjunktion* $l_1 \wedge l_2 \wedge \ldots \wedge l_n$ *die Vorbedingung der Regel und* Φ *der daraus resultierende ableitbare Satz. Um Fakten in der Theorie zu definieren, darf die Vorbedingung der Regel auch leer sein, also* $\rightarrow \Phi$ *mit* $\Phi \in A$*.*
- Op *ist eine Menge von Operatoren* $o_i \in Op$*, die durch ein Tripel* $< P_{Oi}, D_{Oi}, A_{Oi} >$ *beschrieben werden [Fikes* et al.*, 1972], wobei:*
  - P_{Oi} *eine endliche Konjunktion von Sätzen aus der zugrunde liegenden prädikatenlogischen Sprache* $L = E \cup A \cup B$ *ist, die die Vorbedingungen zur Anwendung des Operators determinieren.*
  - D_{Oi} *eine endliche Menge von essentiellen Sätzen aus* R *ist, die nach der Anwendung des Operators* $o_i \in Op$ *nicht mehr gelten, also aus dem Zustandsraum entfernt werden.*
  - A_{Oi} *eine endliche Menge von essentiellen Sätzen aus* R *ist, die nach der Anwendung des Operators* $o_i \in Op$ *gelten, also zu dem Zustandsraum hinzugefügt werden.*

Hieraus ergibt sich die Beschreibung eines einzelnen Zustands der Planungswelt mit:

Definition 3.3 *Ein Zustand* $s \in S$ *der Planungswelt* W *ist eine Untermenge der essentiellen Sätze* R *aus* W*. Die Menge* S *ist die Menge aller möglichen Zustände einer Planungswelt mit* $S = 2^R$*.*

Darauf aufbauend kann nun ein Plan in einer Planungswelt wie folgt definiert werden:

Definition 3.4 *Ein Plan* P *in einer Planungswelt* $W = (R, T, Op)$ *ist eine Sequenz von Operatoren* $(o_1, \ldots, o_n)$ *mit* $o_i \in Op$ *für* $i = 1, \ldots, n$*.*

Dieser Plan wird in der Planungswelt nach folgender Vorschrift angewandt:

Definition 3.5 *Ein Plan* $P = (o_1, \ldots, o_n)$ *in einer Planungswelt* $W = (R, T, Op)$ *und ein Initialzustand* s_0 *induzieren eine Sequenz von Zuständen* $s_1, \ldots, s_n$ *mit* $s_i \in S$ *für* $i = 1, \ldots, n$*, derart, daß :*
$s_{i+1} = (s_i \setminus D_{Oi}) \cup A_{Oi}$*, falls* $s_i \cup T \vdash P_{Oi}$ *mit* $o_i = < P_{Oi}, D_{Oi}, A_{Oi} >$*.*
Falls jedoch ein i existiert mit $s_i \cup T \not\vdash P_{Oi}$*, so ist der gesamte Plan auf den Initialzustand nicht anwendbar.*

Aus dieser Definition über die Beschaffenheit von Plänen läßt sich nun eine Äquivalenzrelation definieren, die der Gleichheit von Plänen genügt.

Definition 3.6 *Zwei Pläne* $P^1 = (o_1^1, \ldots, o_n^1)$ *und* $P^2 = (o_1^2, \ldots, o_n^2)$ *sind äquivalent genau dann, wenn für jeden Startzustand* $s_0 \in S$ *gilt:*

- *P^1 ist anwendbar genau dann, wenn P^2 anwendbar ist.*
- *$s_i^1 = s_i^2$ für $i = 1, \ldots, n$, die durch die Pläne induziert werden.*

Um noch einmal die Aufteilung der Sprache $L = E \cup A \cup B$ zu verdeutlichen: Die Menge E aller essentiellen Sätze ist die Menge, die erlaubt ist, um einen Zustand zu beschreiben. Mit ihnen wird der Startzustand beschrieben und sie dürfen sonst ausschließlich in den Deletelisten D_{Oi} und den Addlisten A_{Oi} der Operatoren $o_i \in Op$ vorkommen. Somit können die geltenden essentiellen Sätze in einem Zustand nur durch einen Operator verändert werden. Essentielle Sätze sind außerdem in den Vorbedingungen P_{Oi} eines Operators $o_i \in Op$ erlaubt, um die Anwendbarkeit eines Operators zu determinieren. Die essentiellen Sätze in den Zuständen werden also nur durch den Startzustand und die Operatorsequenz erzeugt. Die abgeleiteten Sätze dürfen nur durch Regeln der Theorie T hergeleitet werden. Ihre Gültigkeit wird also nur durch Regeln der Theorie T bestimmt. Außerdem dürfen sie in den Vorbedingungen P_{Oi} eines Operators $o_i \in Op$ vorkommen, um die Anwendbarkeit eines Operators zu determinieren.
Die build-in Prädikate dienen als Hilfsprädikate, die durch die Implementierung der Planungswelt bereitgestellt werden. Sie sollten Operationen auf den verwendeten Termen bereitstellen, die nicht durch Operationen der Planungswelt realisiert werden. Wenn zum Beispiel in der Planungswelt natürliche Zahlen als gültige Terme verwendet werden, sollten die built-in Prädikate primitive Operationen wie Vergleiche, Addition, Subtraktion, etc. bereitstellen, damit der Benutzer nicht die ganze Zahlentheorie der benötigten Operationen in seiner Planungswelt definieren muß. Built-in Prädikate dürfen sowohl in den Vorbedingungen einer Regel r_i der Theorie T auftauchen, als auch in den Vorbedingungen P_{Oi} eines Operators $o_i \in Op$.

3.1.2 Die Definition von Abstraktionen zwischen zwei Planungswelten

Für die Abstraktion von Plänen in zwei Planungswelten benötigt man eine konkrete Planungswelt und eine abstrakte Planungswelt. Im folgenden werden sie mit den beiden Beschreibungen $W_k = (R_k, T_k, Op_k)$ für die konkrete Planungswelt und $W_a = (R_a, T_a, Op_a)$ für die abstrakte Planungswelt assoziiert.
Die abstrakte Planungswelt zeichnet sich durch eine geringere Detailliertheit der einzelnen Planungszustände aus. Der abstrakte Plan sollte außerdem über eine geringere Anzahl von Operationen verfügen und somit, falls er in der abstrakten Planungswelt angewandt wird, mit einer geringeren Anzahl von Planungszuständen vom abstrakten Startzustand zum abstrakten Zielzustand führen.
Gilt nun $s_k \vdash^{T_g} s_a$, so ergibt sich eine Beschreibung der Zustände s_a mit einer geringeren Anzahl von abstrakten essentiellen Sätzen als die Beschreibung von s_k mit konkreten essentiellen Sätzen. Dies hat zur Folge, daß sich die abstrakten Operatoren $Op_a = < Poa_i, Doa_i, Aoa_i >$ mit weniger essentiellen Sätzen in ihren Addlisten Aoa_i und Deletelisten Doa_i, beziehungsweise ihre Vorbedingungen Poa_i mit weniger prädikatenlogischen Formeln aus der Sprache L_a beschreiben lassen als die konkreten Operatoren.
Ziel der Abstraktion ist es nun, einen konkreten Plan p_k in einer konkreten Welt $W_k = (R_k, T_k, Op_k)$ in Beziehung zu einem abstrakten Plan p_a in einer abstrakten Welt $W_a =$

(R_a, T_a, Op_a) zu setzen. Dies ist jedoch nicht für einen beliebigen Plan p_a der abstrakten Welt bei gegebenen Plan p_k in der konkreten Welt möglich. Man kann sich jedoch Relationen zwischen den beiden Plänen definieren, um von einer Abstraktion des Planes p_k durch den Plan p_a sprechen zu können, falls diese Relationen erfüllt sind. Dies motiviert folgende Definition:

Definition 3.7 *Eine Zustandsabstraktionsabbildung* $a\colon S_k \mapsto S_a$ *ist eine Abbildung von der Menge* S_k *aller möglichen konkreten Zustände in die Menge* S_a *aller abstrakten Zustände, die folgenden Bedingungen genügt:*

- *Falls* $s_k \cup T_k$ *konsistent ist, so ist auch* $a(s_k) \cup T_a$ *konsistent; das heißt a ist konsistenzerhaltend bezüglich der Theorien in der konkreten und der abstrakten Welt.*
- *Ist* $s_k \cup s'_k \cup T_k$ *konsistent, dann gilt* $a(s_k \cup s'_k) \supseteq a(s_k) \cup a(s'_k)$*; das heißt die Abbildung a ist monoton bezüglich der Hinzunahme neuer konkreter Zustandsfakten.*

Somit werden durch die Zustandsabstraktionsabbildung die konkreten Zustände $s_k \in S_k$ in die abstrakten Zustände $s_a \in S_a$ abgebildet und definieren so eine Reduktion der Zustandsbeschreibungen unter Wahrung der logischen Konsistenz.

Definition 3.8 *Eine Sequenzabstraktionsabbildung* $b\colon N \mapsto N$ *bildet eine abstrakte Zustandssequenz* $(sa_0, \ldots, sa_n)$ *auf eine konkrete Zustandssequenz* $(sk_0, \ldots, sk_m)$ *ab. Sie ist durch die Abbildung der Indizes der abstrakten Zustände auf die Indzies der konkreten Zustände definiert. Des weiteren müssen folgende Bedingungen gelten:*

- $b(0) = 0$*, das heißt der abstrakte Startzustand* sa_0 *der abstrakten Zustandssequenz wird auf den konkreten Startzustand* sk_0 *der konkreten Zustandssequenz abgebildet.*
- $b(n) = m$*, das heißt der abstrakte Endzustand* sa_n *der abstrakten Zustandssequenz wird auf den konkreten Endzustand* sk_m *der konkreten Zustandssequenz abgebildet.*
- $b(u) < b(u) \Leftrightarrow u < v$*, das heißt, daß die Abbildung b die Ordnung der konkreten Zustände in der abstrakten Welt erhalten muß.*

Mit Hilfe der beiden Abbildungen läßt sich nun eine Abstraktion definieren als:

Definition 3.9 *Ein Plan* p_a *ist eine Abstraktion eines Plans* p_k*, falls eine Zustandsabstraktionsabbildung a und eine Sequenzabstraktionsabbildung b existieren, die folgender Bedingung genügen:*
Falls p_k *und ein Initialzustand* sk_0*, für den* p_k *anwendbar ist, eine Zustandssequenz* $(sk_0, \ldots, sk_m)$ *induziert und* $(sa_0, \ldots, sa_n)$ *eine Zustandssequenz ist, mit* $sa_0 = a(sk_0)$*, die durch einen Plan* p_a *induziert wird, so gilt* $a(sk_{b(i)}) = sa_i$ *für alle* $i \in \{1, \ldots, n\}$

Diese Relation wird in bezug auf die Planungswelten in der folgenden Grafik 3.1 veranschaulicht: Die beiden Abstraktionsabbildungen (a,b) definieren so eine Äquivalenzklasse von Plänen, falls eine Abstraktion existiert, die der Planäquivalenz genügt. Die Abbildung a bildet die konkreten Zustände der konkreten Planungswelt auf die abstrakten Zustände

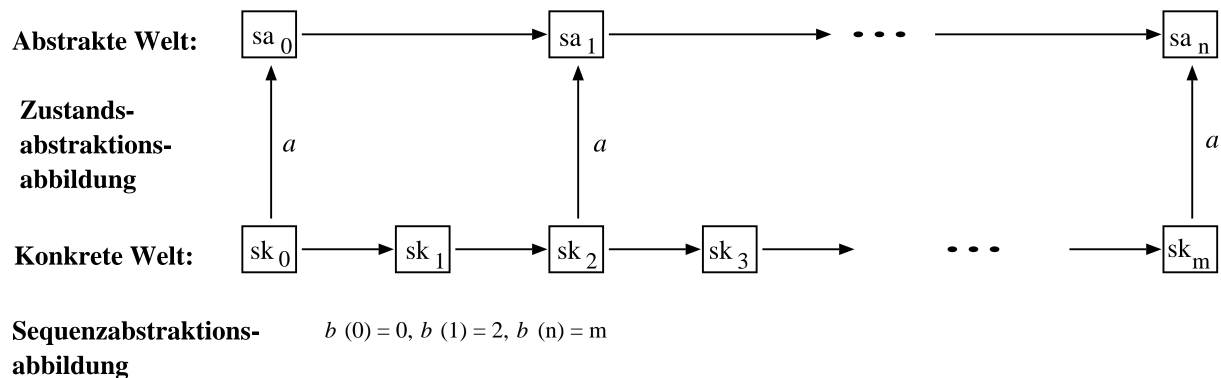

Abbildung 3.1: Konkrete und abstrakte Planungswelt mit den Abbildungen (a,b)

der abstrakten Planungswelt ab.
Die Abbildung b bildet die Indizes der abstrakten Planungszustände auf die Indizes der Zustände der konkreten Planungswelt ab, die durch einen konkreten Plan p_k und einen Startzustand sk_0 induziert werden.
Somit ist das Problem der Konstruktion einer Abstraktion zwischen zwei Planungswelten durch die Suche nach zwei Abbildungen (a,b), die den obigen Bedingungen genügen, zu charakterisieren.

3.1.3 Definition einer wissensintensiven Theorie zur Herleitung von abstrakten Zustandsbeschreibungen

Um die Abbildung der konkreten Zustände auf abstrakte Zustände zu definieren, wird hier eine generische Theorie eingeführt. Um eine Abstraktion zu formulieren, hat Giordana [Giordana *et al.*, 1991] atomare Formeln einer abstrakten Sprache L_a in Termen einer konkreten Sprache L_k ausgedrückt. Seine generische Theorie zur Abstraktion besteht aus einer Menge von Formeln der Form: $\Psi \leftrightarrow D_1 \vee D_2 \vee \ldots \vee D_n$, wobei Ψ eine atomare Formel aus L_a und $D_1 \vee D_2 \vee \ldots \vee D_n$ Konjunktionen von Sätzen aus L_k darstellt.
Beim PABS-Verfahren werden die abstrakten essentiellen Sätze durch Sätze der konkreten Welt ausgedrückt. Durch die Anwendung einer generischen Theorie kann man die Fakten eines abstrakten Zustands herleiten, die zu einem konkreten Zustand gehören.

Definition 3.10 *Eine generische Theorie T_g zur Abstraktion der Zustände ist eine Menge von Regeln der Form: $\Psi \Leftarrow K_1 \wedge K_2 \wedge \ldots \wedge K_n$, wobei Ψ ein essentieller Satz aus der abstrakten Theorie, also $\Psi \in R_a$ und $K_1 \wedge K_2 \wedge \ldots \wedge K_n$ eine Konjunktion von Sätzen aus der Sprache $L_k = E_k \cup A_k \cup B_k$ für die konkrete Theorie darstellen.*

Für eine Zustandsabstraktionsabbildung kann man nun fordern das gilt:
$\Psi \in a(sc) \Rightarrow sc \cup T_c \cup T_g \vdash \Psi$.
Somit reduziert sich die Abstraktion eines konkreten Zustands in einen abstrakten Zustand auf die Anwendung der generischen Theorie auf die Fakten des konkreten Zustands. Durch die Herleitung aller abstrakten essentiellen Sätze mit den Regeln der generischen Theorie

erhält man alle essentiellen Sätze des abstrakten Zustands sa_i, der zu einem konkreten Zustand sk_i, für $i \in \{1, \ldots, n\}$, korrespondiert.
Eine gute Zustandsabstraktionsabbildung zeichnet sich dadurch aus, die wesentlichen essentiellen Sätze zur Realisierung eines Plans, beziehungsweise zur Erfüllung des Planziels zu erhalten und die unwesentlichen zu unterdrücken. So sollten alle essentiellen Sätze, die zur Konstruktion eines abstrakten Plans benötigt werden, durch die Anwendung der generischen Theorie erzeugt werden.

3.1.4 Die Definition eines Skelettplans

Nach der Phase-IV der Planabstraktion nach dem PABS-Verfahren erhält man einen instanziierten abstrakten Plan zur Lösung des Planungsproblems. Dieser Plan besteht aus einer Operatorsequenz und einer Folge von abstrakten Zustandsbeschreibungen, die durch die abstrakte Operatorsequenz und den abstrakten Startzustand induziert wurden. Diese Daten werden nun in der Phase-V in einen Skelettplan transformiert. Vereinfacht dargestellt, geschieht dies durch die Generalisierung des gesamten instanziierten Plans. Danach werden die Beweise zur Anwendung der Operatoren auf essentielle Sätze des abstrakten Startzustandes und auf die dem Plan unterliegenden Constaints in einer Abhängigkeitsanalyse [Bergmann, 1990] reduziert.
Durch die Repräsentation der abstrakten Pläne als Skelettpläne bekommen diese eine größere Generalität und können so durch das Skelettplanrefinement auf mehrere ähnliche Anwendungssituationen spezialisert werden, so daß sie eine Lösung des gegebenen Planungsproblems darstellen.
Die Transformation eines Plans ist ausführlich in [Bergmann, 1990] nachzulesen und im zweiten Abschnitt dieses Kapitels beschrieben. Im folgenden werden die Komponenten des Skelettplans definiert.

Definition 3.11 *Ein Skelettplan* $Sp = (A_s, A_k, A_h)$ *besteht aus:*

- *einer Anwendungssituation* A_s *mit* $A_s = (Fakt_{Zini}(V), Fakt_{Zgoal}(W), R_{enable}(V, W))$.
- *einer Folge von Aktionsklassen* A_k *mit* $A_k = ([O_1(U_1) \textit{ mit } R_{o1}(V, W, U_1)], [O_2(U_2) \textit{ mit } R_{o2}(V, W, U_2)], \ldots, [O_n(U_n) \textit{ mit } R_{on}(V, W, U_n)])$.
- *einer Folge von Abhängigkeiten* $A_k = R_{plan}(V, U_0, \ldots, U_n, W, X)$.

Hierbei stellen die Mengen $V, U_0, \ldots, U_n, W, X$ mit $i \in \{1, \ldots, n\}$ Partitionen der gesamten Menge der Variablen des generalisierten abstrakten Plans dar.

Definition 3.12 *Die Mengen* V, U_i, W *und* X *sind definiert durch:*

- V *ist die Menge von Variablen, die in der Beschreibung des Startzustandes des abstrakten Plans vorkommen.*
- U_i *sind die Mengen von Variablen, die in den Parametern der Operatoren* o_i *für* $i \in \{1, \ldots, n\}$ *des abstrakten Plans* $p_a = (o_1, \ldots, o_n)$ *vorkommen.*

- *W ist die Menge von Variablen, die in der Beschreibung des Zielzustandes des abstrakten Plans vorkommen.*
- *X ist die Menge aller übrigen Variablen, die im abstrakten Plan, aber nicht in $V, U_0, \ldots, U_n, W$ für $i \in \{1, \ldots, n\}$ vorkommen.*

Im folgenden werden nun die einzelnen Komponenten des Skelettplans erklärt.

- Die Menge $Fakt_{Zini}(V)$ ist die Menge von Zustandsfakten des Startzustandes, in denen ausschließlich Variablen der Menge V vorkommen.
- Die Menge $Fakt_{Zgoal}(W)$ ist die Menge von Zustandsfakten des Zielzustandes, in denen ausschließlich Variablen der Menge W vorkommen.
- Die $O_i(U_i)$ für $i \in \{1, \ldots, n\}$ stellen die generalisierten Operationen des abstrakten Plans $p_a = (o_1, \ldots, o_n)$ dar.
- $R(V, U_0, \ldots, U_n, W, X)$ für $i \in \{1, \ldots, n\}$, kurz R, ist die Menge aller Relationen, durch die der Plan implizit in seiner Anwendung beschränkt wird.

Diese Menge läßt sich nun in die drei folgenden Teilmengen partitionieren:

Definition 3.13 *$R_{enable}(V, W)$ ist die größte Teilmenge von Relationen aus der Menge $R(V, U_0, \ldots, U_n, W, X)$ und $X \subseteq X'$, wobei alle Variablen dieser Relation aus $R(V, U_0, \ldots, U_n, W, X) - R_{enable}(V, W)$ eine Variable in X' besitzt.*

$R_{enable}(V, W)$ enthält nur Relationen, die Constraints an die Planungsaufgabe beschreiben, also an die Zustandsfakten des Start- und Zielzustands. Es kommen hier nur Variablen vor, die durch die Beschreibung der Planungsaufgabe vorgegeben sind und solche, die durch keine andere Relation in Bezug zu den Operationen gesetzt werden. Die Constraints in $R_{enable}(V, W)$ sind in sich zusammenhängend und können unabhängig von allen anderen Constraints erfüllt werden.

Definition 3.14 *$R_{oi}(V, W, U_i)$ für $i \in \{1, \ldots, n\}$ ist die größte Teilmenge von Relationen aus $R(V, U_0, \ldots, U_n, W, X) - R_{enable}(V, W)$ und $X \subseteq X_i$, wobei alle Variablen dieser Relationen in $V \cup W \cup U_i \cup X_i$ enthalten sind und keine der Relationen aus $R(V, U_0, \ldots, U_n, W, X) - R_{oi}(V, W, U_i)$ eine Variable in X_i besitzt.*

Die Relationen $R_{oi}(V, W, U_i)$ für $i \in \{1, \ldots, n\}$, mit denen die Aktionsklassen beschrieben sind, enthalten nach dieser Definition nur Constraints an die einzelne Operationen o_i für $i \in \{1, \ldots, n\}$, die wiederum unabhängig von den anderen Operationen erfüllt werden können.

Definition 3.15 *$R_{plan}(V, U_0, \ldots, U_n, W, X'') = R(V, U_0, \ldots, U_n, W, X) - (R_{enable}(V, W) \cup R_{o1}(V, W, U_1) \cup \ldots \cup R_{on}(V, W, U_n))$ mit $X'' = X - (X' \cup X_0 \cup \ldots \cup X_n)$.*

Die in $R_{plan}(V, U_0, \ldots, U_n, W, X'')$ verbleibenden Constraints sind nicht unabhängig für eine einzelne Operation zu erfüllen, da sie Relationen zwischen verschiedenen Operationen und

der Planungsaufgabe beschreiben.
Zur Verdeutlichung der einzelnen Komponenten des Skelettplans sei hier auf den Probelauf im Anhang verwiesen, wo am Ende der Phase-IV die Operationen des abstrakten Plans dargestellt werden und der daraus resultierende Skelettplan nach der Phase-V angegeben wird. Außerdem sei noch einmal auf den zweiten Abschnitt dieses Kapitels verwiesen, wo die Konstruktion eines Skelettplans aus der Beschreibung der Operationen eines Plans dargestellt wird.

3.2 Algorithmische Beschreibung des PABS- Verfahrens

Im folgenden Abschnitt werden die Algorithmen, wie sie in der Implementierung verwandt werden, in Pseudocode beschrieben. Obwohl die Implementierung in PROLOG erfolgte, ist der Pseudocode wegen der besseren Lesbarkeit in einer imperativen Notation angegeben. Außerdem werden in den einzelnen Abschnitten Aussagen über die Komplexität der einzelnen Algorithmen gemacht.
Als problematisch erweisen sich dabei Abschätzungen über die Komplexität bei der Anwendung der Theorien. Die konkrete, abstrakte und die generische Theorie werden beim PABS-Verfahren angewandt, um eine Konjunktion von Fakten zu beweisen oder um die Menge aller ableitbaren Fakten aus einem Zustand zu bestimmen. Um eine Terminierung des Verfahrens zu gewährleisten, wird vorausgesetzt, daß jeder Fakt eine endliche Herleitungslänge besitzt. Dies sollte insbesondere bei der Definition von rekursiven Theorien beachtet werden. Man könnte nun annehmen, daß die Komplexität zur Herleitung einer Konjunktion von Fakten ausschließlich mit der maximalen Herleitungslänge nach oben abgeschätzt werden kann. Dies ist im Prinzip auch richtig, aber es gehen weitere Faktoren in diese Abschätzung mit ein, die unabhängig von der maximalen Herleitungslänge sind, jedoch wesentlichen Einfluß auf die Laufzeit zum Beweisen eines Faktes haben. Dazu gehören:

- die Reihenfolge der Vorbedingungen innerhalb der Regeln
- die Abarbeitungsreihenfolge der Regeln
- die Abarbeitungsreihenfolge der gültigen Fakten in einem Zustand
- die Abarbeitungsreihenfolge der operationalen Prädikate, essentiellen Sätze und der built-in Prädikate.

Somit ist es möglich, daß zwei logisch äquivalente Theoriedefinitionen große Unterschiede in der Laufzeit des PABS-Verfahrens verursachen. Heuristiken, um eine schnelle Laufzeit zu erreichen, werden im Abschnitt 4.1 gegeben. Somit setzen wir eine Größe $fakt_{max}$ fest, die für den Aufwand zur längsten Herleitung einer Konjunktion von Fakten steht. Außerdem gehen in diesen Wert weitere Aktionen mit ein, deren Aufwand jedoch erheblich unter dieser Größe liegt, wie zum Beispiel in der ersten Phase das Hinzufügen und Wegnehmen der Fakten in einem Zustand durch die Operatoren.
Bei der Komplexitätsabschätzung werden folgende Parameter verwandt:

- m die konkrete Planlänge
- n die maximale abstrakte Planlänge des ausgewählten Plans ($\leq m$)
- l die maximale Anzahl von geltenden abstrakten essentiellen Sätzen
- opa_{nr} die Anzahl aller abstrakten Operatoren

Bevor nun die einzelnen Phasen beschrieben werden, erfolgt eine formale Beschreibung der Eingabedaten.

3.2.1 Die benötigten Eingabedaten für das PABS-Verfahren

Als Eingabeparameter des PABS-Verfahrens werden benötigt:

- eine konkrete Weltbeschreibung $W_k = (R_k, T_k, Op_k)$ über einer Sprache $L_k = E_k \cup A_k \cup B_k$.
- ein konkreter Startzustand sk_0, auf den der konkrete Plan angewendet werden soll.
- ein konkreter Plan $Pk = (ok_1, \ldots, ok_m)$.
- Eine generische Theorie $T_g = \{rg_1, \ldots, rg_r\}$ mit $rg_i \equiv \Psi^i \leftarrow K_1^i \wedge K_2^i \wedge \ldots \wedge K_k^i$ für $i = 1, \ldots, r$.
- eine abstrakte Weltbeschreibung $W_a = (R_a, T_a, Op_a)$ über einer Sprache $L_a = E_a \cup A_a \cup B_a$.

3.2.2 Phase-I: Simulation des konkreten Plans

In der ersten Phase wird der konkrete Plan $Pk = (ok_1, \ldots, ok_m)$ auf den konkreten Startzustand sk_0 gemäß der Definition 3.5 angewendet. Es wird also eine Sequenz $sk_1, \ldots, sk_m$ berechnet mit:
$sk_{i+1} = (sk_i \setminus Dok_i) \cup Aok_i$, falls $sk_i \cup T_k \vdash Pok_i$ mit $ok_i =< Pok_i, Dok_i, Aok_i >$. Dies geschieht durch einfache Anwendung der Definition:

FOR $i = 1$ **TO** m **DO**

1. Kopiere die Zustandsbeschreibungen sk_{i-1} nach sk_i.
2. Prüfe, ob $sk_{i-1} \cup T_k \vdash Pok_i$ gilt.
3. $sk_i = sk_i \setminus Dok_i$
   (* Entferne die essentiellen Sätze der Deleteliste aus der Zustandsbeschreibung *)
4. $sk_i = sk_i \cup Aok_i$
   (* Füge die Zustandsbeschreibungen der Addliste zum neuen Zustand hinzu *)

END (* of for *)

Da die Anzahl der Fakten, die einen Zustand beschreiben, nach der Definition 3.3 endlich ist, sind im 1. Schritt nur endlich viele Fakten zu kopieren. Des weiteren ist die Anzahl der Vorbedingungen jedes Operators endlich und jede Vorbedingung muß, um das Verfahren durchführbar zu machen, einen endlichen Beweis haben. Da die Deletelisten Dok_i und die Addlisten Aok_i per Definition 3.2 auch endlich sind, müssen pro Schleifendurchlauf nur endlich viele Schritte ausgeführt werden, die man gegen ein Maximum abschätzen kann. Die Zeitkomplexität für den Beweis und für das Verändern der Zustandsbeschreibung kann mit $fakt_{max}$ abgeschätzt werden. Daraus ergibt sich ein Aufwand in der ersten Phase, der in der Landauschen Notation $O(fakt_{max} * m)$ entspricht, wobei m die Länge des konkreten Planes darstellt.

3.2.3 Phase-II: Die Anwendung der generischen Theorie

In der zweiten Phase werden durch die Anwendung der generischen Theorie T_g die Zustandsbeschreibungen der abstrakten Zustände $sa_0, \ldots, sa_m$ hergeleitet. Diese Zustandsbeschreibungen bestehen aus einer Untermenge der essentiellen Sätze $E_a = \{ea_1, \ldots, ea_l\}$ aus der Sprache L_a gemäß der Definition 3.3. Hier wird für alle konkreten Zustandsbeschreibungen geprüft, welche abstrakten essentielle Sätze sich mit der generischen Theorie aus der konkreten Zustandsbeschreibung und der konkreten Theorie herleiten lassen, also ob gilt:
$sk_i \cup T_k \cup T_g \vdash ea_j$ mit $ea_j \in E_a$.
Die Menge aller herleitbaren Sätze aus einer konkreten Zustandsbeschreibung und den Theorien ergeben dann eine Beschreibung des abstrakten Zustands.

Initalisierung: $sa_i = \{\}$ für $i = 1, \ldots, m$
FOR $i = 1$ **TO** m **DO**
  FOR $j = 1$ **TO** l **DO**
    IF $sk_i \cup T_k \cup T_g \vdash ea_j$ **THEN** $sa_i = sa_i \cup ea_j$
  END (* of for *)
END (* of for *)

Es wird also für alle essentiellen abstrakten Sätze geprüft, ob sie aus der jeweiligen Zustandsbeschreibung herleitbar sind. Die Zeitkomplexität dieser Suche wird wieder durch den Wert $fakt_{max}$ abgeschäzt. Dies wird für alle m konkreten Zustandsbeschreibungen durchgeführt und in jedem Zustand mßsen alle möglichen abstrakten essentiellen Sätze überprüft werden. Somit kann man die Komplexität der zweiten Phase mit $O(fakt_{max} * l * m)$ angeben.

3.2.4 Phase-III: Bestimmung der relevanten Operatoren

In der dritten Phase werden nach der Definition des PABS-Verfahrens alle Operatoren bestimmt, die eine Zustandstransformation von einem abstrakten Startzustand in einen abstrakten Zielzustand ermöglichen. Sei opa_{nr} die Anzahl der abstrakten Operatoren und m die Anzahl der abstrakten Zustände, die der Anzahl der konkreten Zustände entspricht. Somit muß für jeden Zustandsübergang jeder Operator auf seine Anwendbarkeit hin überprüft werden. Da der Index des Startzustandes echt kleiner sein muß als der des Zielzustandes, ergibt sich die Anzahl der zu überprüfenden Übergänge aus:

$$(m - 1 + m - 2 + \ldots + 1) = \frac{(m-1)*(m-2)}{2} = \frac{m^2-3m+2}{2} \approx O(m^2).$$

Durch die Anzahl der abstrakten Operatoren und den Aufwand zur Prüfung der Vorbedingungen ergibt sich eine Gesamtkomplexität von $O(opa_{nr} * fakt_{max} * m^2)$. Der Aufwand der dritten Phase wächst also quadratisch zur konkreten Planlänge.
Unter Berücksichtigung der Konsistenzprüfung in der vierten Phase läßt sich die Anzahl der zu überprüfenden Start- und Zielzustandskombinationen jedoch erheblich reduzieren. Es können nämlich nur Operatorübergänge zu einem konsistenten Pfad beitragen, deren Startzustand durch einen abstrakten Operatorübergang erreicht wird. Ist dies nicht der

Fall, kann dieser Übergang sowie alle seine Nachfolgerübergänge vernachlässigt werden, da sie bei der Konsistenzprüfung in der vierten Phase sowieso eliminiert würden. Außerdem brauchen nur Übergänge berücksichtigt werden, die auf einem kompletten Pfad liegen, der auch vom Startzustand bis zum Zielzustand führt. Dieser Sachverhalt wird noch einmal in der folgenden Abbildung 3.2 verdeutlicht. Die als unrelevant markierten Übergänge würden nach dem PABS-Verfahren in der dritten Phase gesucht, jedoch in der vierten Phase durch die Konsistenzprüfung eliminiert.

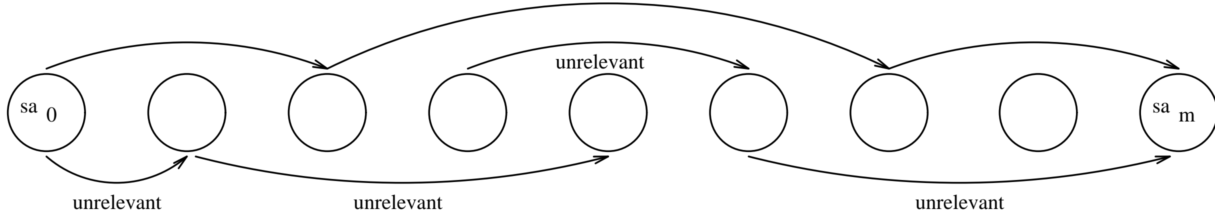

Abbildung 3.2: Unrelevante Übergänge in Phase-III

Somit wurde ein Algorithmus gewählt, der nur alle Übergänge überprüft, die auch tatsächlich für einen konsistenten Pfad relevant sind. Es handelt sich um eine Tiefensuche ausgehend vom Anfangszustand sa_0, die alle Übergänge ermittelt, die zu einem kompletten Pfad vom Startzustand zum Zielzustand beitragen. So wird die Menge der zu überprüfenden Startzustände für einen Operatorübergang eingeschränkt auf die Zustände, die tatsächlich von einer Sequenz von Operatorübergängen, ausgehend vom abstrakten Startzustand sa_0, erreicht werden. Diese Menge von zu überprüfenden Startzuständen ist minimal, da sich nicht vorausschauend ermitteln läßt, ob eine Sequenz von Operatorübergängen in einen Zustand führt, in dem kein Operator mehr anwendbar ist und der nicht dem abstrakten Endzustand sa_m entspricht. Dies liegt daran, daß die Menge der gültigen Fakten eines Folgezustands abhängig ist von den Operatordefinitionen, der Operatoren, die vom Anfangszustand zu diesem Folgezustand führen.
Sei nun $op_i \in Op_a$ für $i = 1, \ldots, opa_{nr}$ die Menge der abstrakten Operatoren und $sa_0, \ldots, sa_m$ die Menge der abstrakten Zustandsbeschreibungen, die mit Hilfe der generische Theorie in der zweiten Phase aus den konkreten Zustandsbeschreibungen hergeleitet wurden. Zur Beschreibung der dritten Phase betrachten wir folgenden Algorithmus:

Initialisierung: $START = 0$; $PFAD = [0]$

PROZEDURE FindePfad($START$,$PFAD$)
IF $START = m$
  THEN 1. Output($PFAD$) mit $PFAD = (p_0, \ldots, p_k, p_{k+1})$
    (* Vollständiger Pfad vom Start zum Ziel gefunden *)
    2. FindePfad($k, (p_0, \ldots, p_k)$)
    (* Rekursiver Aufruf um weitere Pfade zu finden *)
  ELSE IF
    1. Suche $op_i =< Poa_i, Doa_i, Aoa_i >$ mit $sa_{START} \cup T_a \vdash Poa_i$
    (* Hierbei werden Operatoren ausgewählt, die auf
    denselben Teilplan noch nicht angewendet wurden *)

2. Suche *ZWISCHENZIEL* mit $START < ZWISCHENZIEL \leq m$ und
a) $\forall a \in Aoa_i$ gilt $a \in sa_{ZWISCHENZIEL}$
b) $\forall d \in Doa_i$ gilt $d \notin sa_{ZWISCHENZIEL}$

THEN

1. $NEWPFAD = PFAD \| ZWISCHENZIEL$
(* Zustandsübergang gefunden *)
2. FindePfad(*ZWISCHENZIEL*,*NEWPFAD*)

ELSE

FindePfad$(k, (p_0, \ldots, p_k))$
(* mit $PFAD = (p_0, \ldots, p_k, p_{k+1})$ *)

Dieser Algorithmus findet, falls sie existiert, eine Reihenfolge von Operatoren, die vom Startzustand zum Endzustand führen. Die Prozedur Output(*PFAD*) gibt die Zustandsübergänge zurück, die zu dem gefundenen Pfad gehören und noch nicht in einem zuvor gefundenen Pfad vorhanden sind. Anschließend wird die Prozedur FindePfad erneut aufgerufen, um ausgehend vom letzten gefundenen Pfad weiter nach vollständigen Pfaden zu suchen. Hierbei wird durch das Entfernen des letzten Operatorübergangs aus dem gefundenen Pfad die Suche nach weiteren Pfaden fortgesetzt. Dies geschieht auch im zweiten **ELSE** Teil, falls in dem zweiten **IF** - Statement kein Operator gefunden wird, der den bis dahin verfolgten Pfad weiterführt. Hierdurch wird ein Backtracking über die bereits gefundenen Pfade realisiert. So werden alle Zustandsübergänge gefunden, die zu allen kompletten Pfaden vom abstrakten Startzustand sa_0 zum abstrakten Zielzustand sa_m beitragen. Da nur ein Backtracking in den oberen beiden Fällen angestoßen wird, werden auch keine gleichen Teilpfade, die in mehreren kompletten Pfaden vorkommen, mehrfach überprüft. Der Aufwand dieser Phase kann mit der maximalen abstrakten Pfadlänge n, der Anzahl der abstrakten Operatoren opa_{nr} und dem maximalen Aufwand zur Überprüfung der Vorbedingungen der Operatoren $fakt_{max}$, abgeschätzt werden durch $O(opa_{nr} * fakt_{max} * n^2)$.

3.2.5 Phase-IV: Konsistenzprüfung

In der vierten Phase werden alle Kombinationen von Zustandsübergängen gebildet, die einen vollständigen Pfad ergeben. Hierbei ist mit vollständig ein Pfad gemeint, der vom abstrakten Startzustand sa_0 zum abstrakten Zielzustand sa_m führt. Diese Pfade werden dann einer Konsistenzprüfung unterzogen. Sei ein abstrakter Pfad gegeben durch eine Operatorsequenz $Pa = (oa_1, \ldots, oa_n)$ mit $oa_i =< Poa_i, Doa_i, Aoa_i >$ für $i = 1, \ldots, n$, angewendet auf einen Startzustand sa'_0 gegeben, der eine Sequenz von Zuständen $sa'_{b(1)}, \ldots, sa'_{b(n)}$ induziert. Bei der Abbildung b handelt es sich um die Sequenzabstraktionsabbildung, die im vorherigen Abschnitt, in Definition 3.8 definiert wurde. Dieser Pfad ist konsistent, falls gilt:
Ist sa_i, mit $sa'_i \supseteq sa_i$, die Menge aller abstrakten essentiellen Sätze, die durch $sa_{i-1} \cup T_a \vdash Poa_i$ die Anwenbarkeit des Operators oa_i für $i = 1, \ldots, n$ garantieren, so gilt für alle $ea \in sa_i$ entweder

- $ea \in Aoa_i$ mit $k < i$ und $\forall k'$ mit $k \leq k' < i$ gilt: $ea \notin Doa_{k'}$

oder

- $ea \in sa_0$ mit $k < i$ und $\forall Doa_{k'}$ mit $0 \leq k' < i$ gilt: $ea \notin Doa_{k'}$

Diese Bedingung sichert, daß der abstrakte Plan in der abstrakten Welt auch wirklich ausführbar ist, d.h daß alle Vorbedingungen, die die Operatoren benötigen, zu der Zeit gelten, wo der Operator auf den Zustandsraum angewandt wird. Die vierte Phase wird durch folgenden Algorithmus realisiert.

Initialiserung: $START = 0$; $PFAD = []$
PROZEDURE SuchePfad($START$,$PFAD$)
IF $START = m$
 THEN Output($PFAD$) mit $PFAD = (p_0, \ldots, p_k, p_{k+1})$
  (* Vollständiger Pfad vom Start zum Ziel gefunden *)
  2. FindePfad($k, (p_0, \ldots, p_k)$)
  (* Rekursiver Aufruf um weitere Pfade zu finden *)
 ELSE
  1. Suche Kante $K = (START, ZIEL, oa_i)$ von sa_{START} nach sa_{ZIEL}
  (* Kante K existiert, da nur Zustandsübergänge betrachtet werden,
  die zu einem vollständigen Pfad beitragen. Es werden nur Kanten
  ausgewählt, die in der identischen Situation noch nicht betrachtet wurden *)
  2. $\forall ea \in Poa_i$ prüfe obige Konsistenzbedingung durch Rückverfolgung des bisher gefundenen Pfades
  (* Konsistenzprüfung *)
  3. $NEWPFAD = PFAD \| K$
  (* Füge Kante K zum $PFAD$ hinzu *)
  4. SuchePfad($ZIEL$,$NEWPFAD$)

Auch hier wird nach dem Anzeigen des Pfades durch Output($PFAD$) ein rekursiver Aufruf durchgeführt, um weitere vollständige, konsistente Pfade zu finden. Die Erfüllung der Konsistenzbedingung kann jedoch nicht garantiert werden. So kann es passieren, daß es zwar vollständige Pfade gibt, die vom Start- zum Zielzustand führen, jedoch keiner dieser Pfade die Konsistenzbedingung erfüllt. Genügt ein Zustandsübergang nicht der Konsistenzbedingung wird ebenfalls ein Backtracking über den bisher gefundenen Pfad durchgeführt, um weitere vollständige, konsistente Pfade zu finden. Sind alle möglichen vollständigen, konsistenten Pfade gefunden, werden diese angezeigt und der Benutzer kann einen dieser Pfade wählen, der dann in der fünften Phase in einen Skelettplan transformiert wird.
Der Aufwand in dieser Phase ist von der Länge der vollständigen Pfade, dem Verzweigungsgrad der einzelnen Zustände, sowie der Anzahl der essentiellen Sätze, die in der Konsistenzprüfung überprüft werden müssen, abhängig. Sei n die maximale Länge eines abstrakten Pfades, opa_{nr} der maximale Verzweigungsgrad in einem abstrakten Zustand und l die maximale Anzahl der essentiellen Sätze, die zur Anwendung eines Operators überprüft werden müssen. Dann läßt sich der Aufwand der vierten Phase durch $O(l * opa_{nr} * n^2)$ nach oben abschätzen. Diese Größe resultiert daraus, daß jeder essentielle Satz in einem Zustand über alle möglichen Pfade, die diesen Zustand erreichen, zurückverfolgt werden muß. Die Länge dieser Verfolgung führt maximal über n Zustände zum abstrakten Startzustand sa_0. Dies wird bei allen Zuständen durchgeführt, die durch Operatoren des Pfades erreicht werden. Durch die Berücksichtigung aller Verzweigungen an einem Knoten werden alle möglichen Pfade erfaßt.

3.2.6 Phase-V: Skelettplantransformation

In der fünften Phase wird der vom Benutzer in der vierten Phase ausgewählte abstrakte Plan in einen Skelettplan transformiert. Dieser Algorithmus, der aus vier Phasen besteht, ist ausführlich in [Bergmann, 1990] beschrieben. In der ersten Phase wird die Anwendung des Plans auf den Startzustand simuliert, indem die durch die Operatorbeschreibung resultierenden Folgezustände berechnet werden. Diese Zustandsbeschreibungen sind jedoch nach der vierten Phase des PABS-Verfahrens bereits vorhanden, so daß auf diese Berechnug verzichtet werden kann. Es müssen nur noch die Beweise, die die Anwendbarkeit der abstrakten Operatoren sichern, unter Berücksichtigung des abstrakten Operationalitätskriteriums, generalisiert werden. Die gefundenen operationalen Konzepte beschreiben, welche Bedingungen für die Anwendung der Operatoren wichtig sind und welche Effekte aus der Anwendung der Operatoren resultieren. In der zweiten Phase der Skelettplangenerierung wird eine Abhängigkeitsanalyse durchgeführt. Hieraus resultiert ein Plankonzept für jeden Fakt aus $Fakt_{Zgoal}$, das eine Konsequenz aus der Domänentheorie und des Planungsproblems darstellt. Dieses Plankonzept stellt die durch den Plan gegebene Abhängigkeits- und Beweisstruktur durch Constraints der Mengen $Fakt_{Zini}(V), O_i(U_i)$ für $i \in \{1, \ldots, n\}$ und $R(V, U_0, \ldots, U_n, W, X)$ dar. In der dritten Phase der Skelettplantransformation wird eine operationalisierende Reduktion des Plankonzepts erzeugt. Diese Phase wurde hier nicht implementiert, da sie das Plankonzept zwar vereinfacht, aber den logischen Aussagewert des Plankonzepts nicht ändert. In der vierten Phase werden die Planungskonzepte in die Komponenten des Skelettplans gemäß Definition 3.11 transformiert. Hier werden die Elemente des operationalen Planungskonzepts $R(V, U_0, \ldots, U_n, W, X)$ in die Teilmengen $R_{enable}(V, W)$, $R_{oi}(V, W, U_i)$ und $R_{plan}(V, U_0, \ldots, U_n, W, X)$ zerlegt. Dies geschieht durch folgenden Algorithmus:

Initialisierung: $R_{enable}(V, W) = \{\}$; $R_{oi}(V, W, U_i) = \{\}$; $R_{plan}(V, U_0, \ldots, U_n, W, X) = \{\}$
FOR ALL $P \in R(V, U_0, \ldots, U_n, W, X)$ **DO**
  IF $(V \cup W \cup X) \supseteq VAR(Kontext(P, R))$ und
    $VAR(Kontext(P, R)) \cap U_i = \{\}$ für $i \in \{1, \ldots, n\}$
  THEN$R_{enable} = R_{enable} \cup \{P\}$
  ELSE
  IF $(V \cup W \cup X \cup U_i) \supseteq VAR(Kontext(P, R))$ und
    $\neg\exists j$ mit $(V \cup W \cup X \cup U_j) \supseteq VAR(Kontext(P, R))$
  THEN$R_{oi} = R_{oi} \cup \{P\}$
  ELSE$R_{plan} = R_{plan} \cup \{P\}$

Der Kontext zu einem Prädikat P bezüglich einer Menge R ist die Teilmenge von $R(V, U_0, \ldots, U_n, W, X)$, in der alle Prädikat vorkommen, in denen eine Variable vorkommt, die auch in P enthalten ist und die bezüglich dieser Eigenschaft abgeschlossen ist. Diese Menge kann definiert werden durch:
$(Kontext(P, R))$ ist eine Teilmenge von R mit:

1. $P \in (Kontext(P, R))$

2. $VAR(Kontext(P, R)) \cap VAR(R - Kontext(P, R)) = \{\}$

3. $(Kontext(P, R))$ ist minimal

Hierbei ist $VAR(M)$ die Menge aller Variablen, die in einer Menge M von Prädikaten vorkommen. Die Menge $(Kontext(P, R))$ kann leicht mit einem Hülloperator durch iterative Vergrößerung ausgehend von P bestimmt werden.
Hieraus resultiert die endgültige Repräsentation des Skelettplans, der dann in ein File geschrieben werden kann, um ihn später mit anderen Verfahren weiterverarbeiten zu können.
Bei der Betrachtung der Komplexität werden die drei implementierten Phasen der Skelettplangeneration

- Generalisierung der Beweise zur Anwendung der Operatoren
- Abhängigkeitsanalyse zur Entwicklung des Plankonzept
- Generierung der Mengen $R_{enable}(V, W), R_{oi}(V, W, U_i), R_{plan}(V, U_0, \ldots, U_n, W, X)$

getrennt betrachtet. Im ersten Schritt werden die Beweise zur Anwendung der Operatoren in ihre atomaren Bestandteile zerlegt und unter Berücksichtigung des abstrakten Operationalitätskriteriums auf eine festgelegte Ebene zur Erklärung verkürzt. Dies wird für jeden Operator des abstrakten Plans durchgeführt. Die Komplexität läßt sich durch die Länge des gewählten abstrakten Plans n und durch die maximale Beweislänge mit Berücksichtigung des Operationalitätskriteriums $fakt_{max}$ abschätzen durch $O(fakt_{max} * n)$.
Im zweiten Schritt der Skelettplangenerierung wird das Plankonzept des abstrakten Plans erzeugt. Hierzu werden alle essentiellen Sätze der Zustandsbeschreibung des abstrakten Zielzustands bis zu ihrer Erzeugung durch einen Operator, bzw. bis in den Startzustand zurückverfolgt. Diese essentiellen Sätze werden dann mit den operationalisierten Beweisstrukturen unifiziert, die für die Entstehung des Fakts verantwortlich sind. Somit ist die Komplexität dieser Phase zum einen von der Anzahl der geltenden Fakten im abstrakten Zielzustand l, zum anderen von der abstrakten Planlänge n abhängig. Die Komplexität kann somit nach oben abgeschätzt werden durch $O(n * l)$.
Im dritten Schritt werden alle Elemente der Menge $R(V, U_0, \ldots, U_n, W, X)$ in die Mengen $R_{enable}(V, W)$, $R_{oi}(V, W, U_i)$, $R_{plan}(V, U_0, \ldots, U_n, W, X)$ nach dem obigen Algorithmus einsortiert. Dazu wird die Menge $R(V, U_0, \ldots, U_n, W, X)$ gebildet, indem die Constraints aus den Beweisen zur Anwendung der Operatoren aufgesammelt werden. Somit ist dieser Schritt linear von der abstrakten Planlänge abhängig, also $O(n)$. Nun wird ein Prädikat aus der Menge $R(V, U_0, \ldots, U_n, W, X)$ entfernt und der Kontext zu diesem Prädikat gebildet. Der Aufwand für diese Kontextbildung wächst quadratisch mit der Kardinalität der Menge $R(V, U_0, \ldots, U_n, W, X)$, die durch $fakt_{max} * n$ nach oben abgeschätzt werden kann. Anschließend wird der Kontext in eine der Mengen $R_{enable}(V, W)$, $R_{oi}(V, W, U_i)$, $R_{plan}(V, U_0, \ldots, U_n, W, X)$ einsortiert. Die Komplexität dieser Phase ist also $O(n) + O((fakt_{max} * n)^2)$. Hieraus resultiert eine Gesamtkomplexität der Skelettplantransformation gegeben durch:
$O(fakt_{max} * n) + O(n * l) + O(n) + O((fakt_{max} * n)^2) \approx O(n * l) + O((fakt_{max} * n)^2)$.

3.2.7 Die Gesamtkomplexität des PABS-Verfahrens

Da die fünf Phasen des PABS-Verfahrens hintereinander ausgeführt werden kann die Gesamtkomplexität des Verfahrens angegeben werden durch die Addition der Komplexitäten

der einzelnen Phasen. Es ergibt sich eine Gesamtkomplexität von:

$$O(fakt_{max} * m) + O(fakt_{max} * l * m) + O(opa_{nr} * fakt_{max} * n^2) + O(l * opa_{nr} * n^2) + O(n * l) + O((fakt_{max} * n)^2) \approx O(fakt_{max} * l * m) + O(l * opa_{nr} * fakt_{max} * n^2) + O((fakt_{max} * n)^2)$$

mit:

- m die konkrete Planlänge
- n die maximale abstrakte Planlänge des ausgewählten Plans
- l die maximale Anzahl von geltenden abstrakten essentiellen Sätzen
- opa_{nr} die Anzahl aller abstrakten Operatoren

Kapitel 4

Implementierung des Verfahrens

Im ersten Abschnitt dieses Kapitels werden die benötigten Prolog Fakten beschrieben, die zur Definition einer konkreten, bzw. abstrakten Weltbeschreibung und einer generischen Theorie benötigt werden. Im zweiten Abschnitt wird die Implementation der Phasen in Prolog verdeutlicht. Alle beschriebenen Beispiele sind der Domäne Towers of Hanoi (ToH) entnommen, deren Definition sich vollständig im Anhang A befindet.
Um eine bessere Laufzeit des PABS-Verfahrens zu erhalten, sollten Operatoren oder Regeln, die oft benötigt werden, in der jeweiligen Definition zuerst deklariert werden. Außerdem sollten in den Vorbedingungen der Operatoren, die Sätze zuerst stehen, die am ehesten für ein Scheitern der Anwendung der Regel oder der Anwendung des Operators sorgen. Bei der Reihenfolge der Regeln, sollte man weiter darauf achten, daß die Regeln, die einen kurzen Beweis eines Faktes ermöglichen, vor den Regeln stehen, mit denen sich der Fakt zwar auch beweisen läßt, aber einen längeren Beweis zur Folge hat. Bei den übrigen Fakten zur Definition der Eingabedaten ist die Reihenfolge der Deklarationen weitgehend unabhängig von der Laufzeit.

4.1 Definition von Weltbeschreibungen mit PABS

Als Eingabedaten für das PABS-Verfahren müssen in Prolog folgende Daten definiert werden:

1. ein konkreter Plan mit einem Startzustand, auf den der Plan angewandt wird.
2. eine konkrete Weltbeschreibung, bestehend aus konkreten essentiellen Sätzen, konkreten Operatoren und einer konkreten Theorie.
3. einer generischen Theorie, bestehend aus generischen Regeln.
4. eine abstrakte Weltbeschreibung, bestehend aus abstrakten essentiellen Sätzen, abstrakten Operatoren und einer abstrakten Theorie.

Diese Definition geschieht wegen der besseren Lesbarkeit der Domänen und auf Grund einer sinnvollen Strukturierung der Eingabedaten in vier verschiedenen Files, die durch das PABS-System eingelesen werden. Die Aufteilung der Files entspricht der obigen Gliederung und sie werden mit:

1. <domäne>.plan
2. <domäne>.kon
3. <domäne>.gen
4. <domäne>.abs

bezeichnet. Es findet beim Einlesen der Dateien jedoch weder eine semantische Überprüfung des Inhalts, noch eine syntaktische Überprüfung des korrekten Filenamens statt. Um jedoch einheitliche Definitionen von Domänen zu erhalten, sollte die hier vorgeschlagene Einteilung befolgt werden. Es folgt nun eine Beschreibung des Inhalts der einzelnen Files.

4.1.1 Konkrete Pläne und ihre Startzustände

In dem File <domäne>.plan werden die konkreten Pläne mit ihren konkreten Startzuständen definiert. Hierbei ist es möglich, mehrere konkrete Pläne mit Startzuständen zu definieren, aus denen dann der Benutzer innerhalb des PABS-Systems einen zu bearbeitenden Plan auswählen kann. Hierbei wird sowohl zwischen verschiedenen Plänen mit gleichen Startzuständen, als auch zwischen verschiedenen Startzuständen mit gleichen Plänen unterschieden, da beide Fälle ein anderes Ergebnis zur Folge haben. Ein Plan mit einem Startzustand wird nun definiert durch das Prädikat plan/4 mit:

plan(<Schlüssel>,<Typ>,<Startzustand>,<Plan>).

<Schlüssel>:= Eindeutiger Schlüssel, bestehend aus einer ganzen Zahl, so daß das PABS-System bei der Auswahl des konkreten Plans auf die Pläne verweisen kann.

<Typ>:= Bezeichnet die Welt, in der der Plan gilt, also hier <Typ>= konkret. [1]

<Startzustand>:= Eine Liste von konkreten essentiellen Sätzen, die den konkreten Startzustand beschreiben, auf die der Plan angewendet wird.

<Plan>:= Eine Liste von konkreten Operatoren, die die konkrete Operatorfolge definieren. Die Reihenfolge der Anwendung der Operatoren ist durch die Position der Operatoren in der Liste bestimmt.

Der Fakt plan/4 ist der einzig zulässige Fakt in der Datei <domäne>.plan. Eine korrekte Definition von konkreten Plänen in der ToH-Domäne ist durch das folgende Beispiel gegeben:

BEISPIEL:

plan(1,konkret,[tower(a,[1,2,3]),tower(b,[]),tower(c,[])],

[1] abstrakte Pläne werden beim PABS-System nur als Skelettpläne dargestellt und anders repräsentiert. Trotzdem wurde dieses Argument in die Beschreibung aufgenommen, um die Domänendefinitionen auf andere Verfahren, wie S-PABS und hierarchisches Planen mit abstrakten Plänen, anwenden zu können.

[move(a,b),move(a,c),move(b,c),move(a,b),move(c,a),move(c,b),move(a,b)]).

plan(2,konkret,[tower(a,[1,2,3]),tower(b,[]),tower(c,[])],
[move(a,c),move(a,b),move(c,b),move(a,c),move(b,a),move(b,c),move(a,c),
move(c,b),move(c,a),move(b,a),move(c,b),move(a,c),move(a,b),move(c,b)]).

4.1.2 Die konkrete Weltbeschreibung

Die konkrete Welt wird in dem File <domäne>.kon beschrieben. Diese Beschreibung besteht aus ausgezeichneten essentiellen konkreten Sätzen, konkreten Operatoren und einer konkreten Theorie. Essentielle Sätze werden definiert durch den Prolog Fakt essential/2 mit:

essential(<Satz>,<Typ>).

<Satz>:= essentieller Satz, dessen Parameter mit anonymen Variablen angegeben werden.

<Typ>:= bezeichnet die Welt, in der der essentielle Satz gilt, also hier <Typ>= konkret.

BEISPIEL:

essential(tower(_,_),konkret).

Die konkreten Operatoren werden durch das Fakt operator/6 beschrieben mit:

operator(<Operator>,<Nr>,<Typ>,<Prelist>,<Dellist>,<Addlist>).

<Operator>:= Operatorname mit seinen Parametern. Die Parameter werden durch Prolog Variablen repräsentiert.

<Nr>:= eindeutige ganze Zahl zur Identifizierung des Operators im System. [2]

<Typ>:= gibt den Typ des Operators an, also hier <Typ>= konkret.

<Prelist>:= Konjunktion von Vorbedingungen, die die Anwendbarkeit des Operators determinieren. Hier können konkrete essentielle Sätze, abgeleitete Sätze und Systemprädikate stehen.

<Dellist>:= Konjunktion von konkreten essentielle Sätzen, die nach Anwendung des Operators nicht mehr gelten.

<Addlist>:= Konjunktion von konkreten essentielle Sätzen, die nach Anwendung des Operators gelten.

BEISPIEL:

[2] Hierdurch wird es möglich einen Operator durch mehrere Operatordefinitionen zu definieren, da auf die Operatoren nicht über ihren Namen, sondern über diesen Schlüssel referiert wird

```
operator(move(X,Y),1,konkret,
          (smaller(X,Y),tower(Y,L)),
          (tower(X,[Disk|Disks]),(tower(Y,[_ |_ ])),
          (tower(X,Disks),tower(Y,[Disk|L]))).
operator(move(X,Y),2,konkret,
          (tower(X,[Disk|Disks]),tower(Y,[])),
          (tower(X,[Disk|Disks]),tower(Y,[])),
          (tower(X,Disks),tower(Y,[Disk]))).
```

Will man eine leere <Prelist>, <Dellist> oder <Addlist> definieren, so kann man den Prolog Term true verwenden. Eine Operatordefinition mit leerer <Prelist> hat dann zum Beispiel die Definition:

```
operator(dummy(),1,konkret,
          (true),
          (p(a)),
          (q(a))).
```

Kommt ein Fakt in der <Addlist> und in der <Dellist> vor, so gilt dieser nach der Anwendung des Operators, da gemäß STRIPS Definition zuerst alle essentiellen Sätze der <Dellist> entfernt werden und danach die essentiellen Sätze der <Addlist> hinzugefügt werden.
Die konkrete Theorie wird durch eine Menge von Regeln definiert. Die einzelnen Regeln werden mit dem Prolog Fakt regel/3 dargestellt:

regel(<Nr>,<Typ>,<Regel>).

<Nr>:= eindeutige ganze Zahl zur Identifizierung der Regeln im PABS-System. Diese Regelnummer sollte sich von den Regelnummern der abstrakten und der generischen Theorie unterscheiden, um dem System eine eindeutige Zuordnung der Regeln zu den Theorien, auch über die Phasen hinaus, zu ermöglichen, da der <Typ>während der Laufzeit nicht immer instanziert ist.

<Typ>:= legt die Theorie fest, zu der die Regel gehört, hier <Typ>= konkret.

<Regel>:= (<Kopf>:-<Rumpf>)
Hier ist der <Kopf> ein ableitbarer Satz, der gilt, falls alle Bedingungen in <Rumpf> erfüllt sind. Pro Regel darf hier nur ein ableitbarer Satz im <Kopf> jeder Regel stehen. Es ist aber zulässig mehrere Regeln zu definieren, die den gleichen <Rumpf> haben, sich aber durch einen anderen ableitbaren Satz im <Kopf> der Regel unterscheiden. Der Rumpf ist eine Konjunktion von essentiellen konkreten Sätzen, ableitbaren konkreten Sätzen und built-in Prädikaten. Der leere <Rumpf> wird durch den Prolog Term true deklariert.

BEISPIEL:

```
regel(10,konkret,
```

```
        (smaller(X,Y) :- tower(X,[Disk|_]),tower(Y,[DDisk|_]),Disk < DDisk)).
```

Mit Hilfe dieser drei Prädikate wird die konkrete Weltbeschreibung deklariert. Die drei Fakten essential/2, operator/6 und regel/2 sind die einzigen, die in dem File <domäne>.kon verwandt werden sollten.

4.1.3 Die generische Theorie

In dem File <domäne>.gen wir die generische Theorie deklariert. Wie die konkrete Theorie besteht die generische Theorie auch aus einer Menge von Regeln, die durch das Fakt regel/3 deklariert werden. Eine Regel der generischen Theorie wird definiert durch:

regel(<Nr>,<Typ>,<Regel>).

<Nr>:= eindeutige ganze Zahl zur Identifizierung der Regel, die nicht in den konkreten oder den abstrakten Regeln verwendet werden darf.

<Typ>:= legt die Theorie fest, zu der die Regel gehört, hier <Typ>= generisch.

<Regel>:= (<Kopf>:-<Rumpf>) mit

<Kopf>:= stellt einen abstrakten essentiellen Satz oder einen ableitbaren generische Satz dar, der gilt, falls <Rumpf> gilt.

<Rumpf>:= ist eine Konjunktion von

- essentiellen konkreten Sätzen
- abgeleiteten konkreten Sätzen
- abgeleiteten generischen Sätze
- built-in Prädikaten
- der Deklaration für den leeren Rumpf true

<u>**BEISPIEL:**</u> Generische Theorie der Welt ToH

```
regel(0,generisch,
        (abstower(X,[st]) :- tower(X,[1,2]))).
regel(1,generisch,
        (abstower(X,[ld]) :- tower(X,[3]))).
regel(2,generisch,
        (abstower(X,[st,ld]) :- tower(X,[1,2,3]))).
regel(3,generisch,
        (abstower(X,[]) :- tower(X,[]))).
regel(4,generisch,
```

```
        (abstower(X,[md]) :- tower(X,[2]))).
regel(5,generisch,
        (abstower(X,[sd]) :- tower(X,[1]))).
regel(6,generisch,
        (abstower(X,[md,ld]) :- tower(X,[2,3]))).
```

Die Fakten regel/3 zur Definition der generischen Theorie sind die einzigen, die in dem File <domäne>ġen auftauchen sollten.

4.1.4 Die abstrakte Weltbeschreibung

Die abstrakte Welt wird in dem File <domäne>.abs beschrieben. Diese Beschreibung besteht aus ausgezeichneten essentiellen abstrakten Sätzen, abstrakten Operatoren einer abstrakten Theorie und einem abstrakten Operationalitätskriterium. Essentielle Sätze werden definiert durch den Prolog Fakt essential/2 mit:

essential(<Satz>,<Typ>).

<Satz>:= essentieller Satz, dessen Parameter mit anonymen Variablen angegeben werden.

<Typ>:= bezeichnet die Welt, in der der essentielle Satz gilt, also hier <Typ>= abstrakt.

BEISPIEL:

essential(abstower(_,_),abstrakt).

Die abstrakten Operatoren werden durch das Fakt operator/6 beschrieben mit:

operator(<Operator>,<Nr>,<Typ>,<Prelist>,<Dellist>,<Addlist>).

<Operator>:= Operatorname mit seinen Parametern. Die Parameter werden durch Prolog Variablen repräsentiert.

<Nr>:= eindeutige ganze Zahl zur Identifizierung des Operators im System.

<Typ>:= gibt den Typ des Operators an, also hier <Typ>= abstrakt.

<Prelist>:= Konjunktion von Vorbedingungen, die die Anwendbarkeit des Operators determinieren. Hier können abstrakte essentielle Sätze, abgeleitete Sätze und Systemprädikate auftauchen.

<Dellist>:= Konjunktion von abstrakten essentielle Sätzen, die nach Anwendung des Operators nicht mehr gelten.

<Addlist>) := Konjunktion von abstrakten essentielle Sätzen, die nach Anwendung des Operators gelten.

BEISPIEL:

```
operator(movetower(X,Y),3,abstrakt,
          (abstower(X,[st|Rest]),abstower(Y,Disks),X\ =Y,Disks\ =[st]),
          (abstower(X,[st|Rest]),abstower(Y,Disks)),
          (abstower(X,Rest),abstower(Y,[st|Disks]))).
operator(movelargedisk(X,Y),4,abstrakt,
          (abstower(X,[ld|Rest]),abstowerleer(Y),X\ =Y),
          (abstower(X,[ld|Rest]),abstower(Y,[])),
          (abstower(X,Rest),abstower(Y,[ld]))).
```

Die abstrakte Theorie wird durch eine Menge von Regeln definiert. Die einzelnen Regeln werden mit dem Prolog Fakt regel/3 dargestellt:

regel(<Nr>,<Typ>,<Regel>).

<Nr>:= eindeutige ganze Zahl zur Identifizierung der Regeln im PABS-System.

<Typ>:= legt die Theorie fest, zu der die Regel gehört, hier <Typ>= abstrakt.

<Regel>:= (<Kopf>:-<Rumpf>)
Auch hier ist der <Kopf> ein ableitbarer Satz, der gilt, falls alle Bedingungen in <Rumpf> erfüllt sind. Der <Rumpf> ist eine Konjunktion von essentiellen abstrakten Sätzen, ableitbaren abstrakten Sätzen und built-in Prädikaten. Der leere <Rumpf> wird durch den Prolog Term true deklariert.

BEISPIEL:

```
regel(20,abstrakt,
       (abstowerleer(X) :- abstower(X,[]))).
regel(21,abstrakt,
       (abstower(X,[sd,md]) :- abstower(X,[st]))).
```

Des weiteren wird auf der abstrakten Ebene ein abstraktes Operationalitätskriterium benötigt. Dieses wird durch den Fakt operational/2 deklariert, mit:

operational(<Prädikat>,<Typ>)

<Prädikat>:= das Prädikat, welches als operational deklariert werden soll, mit den Parametern als anonymen Variablen.

<Typ>:= Ebene, auf der das Operationalitätskriterium gilt, hier <Typ>= abstrakt. [3]

[3] Um Beweise auf der konkreten Ebene für andere Verfahren generalisieren zu können, ist dieser Parameter vorhanden. Beim PABS-Verfahren werden keine konkreten generalisierten Beweise benötigt

BEISPIEL:

operational((_ =_),abstrakt).

operational(simplify_ nat(_ ,_),abstrakt).

Mit Hilfe dieser vier Prädikate wird die abstrakte Weltbeschreibung deklariert. Die vier Fakten essential/2, operator/6, regel/2 und operational/2 sind die einzigen, die in dem File <domäne>.abs verwandt werden sollten.

4.1.5 Die Definition von built-in Prädikaten

Die built-in Prädikate sind sowohl die durch das jeweilige Prolog System, auf dem das PABS-System läuft, bereitgestellten Systemprädikaten, als auch selbst geschriebene Klauselmengen, die in Prolog definiert werden können. Diese Klauselmengen können einfach zum PABS-System hinzugeladen werden.
Damit der Beweiser in PABS diese Prädikate oder Klauseln erkennt, müssen sie durch das Prädikat systemprädikat/1 gekennzeichnet werden. Diese Deklarationen stehen in dem File systemprädikate.pl im PABS-Verzeichnis. Stoßt nun der Beweiser auf ein solches Systemprädikat, wird einfach ein Call auf dieses Prädikat oder auf die Klausel ausgeführt. Die Parameter werden mit anonymen Variablen angegeben.

BEISPIEL:

systempraedikat(_ ,_).
systempraedikat(not(_)).
systempraedikat(simplify_ nat(_ ,_)). [4]

Diese Menge von Deklarationen werden beim Laden des PABS-Systems eingelesen und können vom Benutzer erweitert werden. Die Menge der Systemprädikate gilt global für alle Domänen.

4.2 Realisierung des PABS-Verfahrens in Prolog

In diesem Abschnitt wird informell die Realisierung der Phasen des PABS-Verfahrens in Prolog beschrieben. Dazu wird ein kompletter Durchlauf des PABS-Systems erläutert. Die Implementierung verfügt zur Erstellung von neuen Domänen über einen Debug-mode, der die resultierenden Daten der einzelnen Phasen anzeigt, um Fehler in den Definitionen der Eingabedaten besser korrigieren zu können. Diese Notation weicht von der internen Repräsentation der Daten ab, weil von unwesentlichen Parametern zur Erstellung von neuen Domänen abstrahiert wird. Die Beschreibung der einzelnen Ein- und Ausgabedaten bezieht sich in diesem Kapitel auch auf die kürzere Notation, die im Debug-mode ausgegeben wird. Die Realisation der Abstraktion von der internen Repräsentation zu der angezeigten

[4] mit der Definition der Klauselmenge zu simplify_ nat/2 aus add.syst

Repräsentation im Debug-mode, befindet sich im File *debug.pl* und wird durch die Klauselmenge *printphase/1* realisiert. Dort befindet sich noch eine Menge weiterer Funktionen, die es ermöglichen, die Fakten des PABS-Verfahrens anzuzeigen, die die Daten repräsentieren, oder die Fakten in ein File zu schreiben. Diese Funktionen sind selbsterklärend und bedürfen keiner weiteren Erläuterung.
Bei der Realisierung des PABS-Verfahrens ist es an mehreren Stellen nötig, Sätze mit Theorien, built-in Prädikaten, essentiellen oder abgeleiteten Sätzen zu beweisen. Dies wird durch die Klausel *prove/6* aus dem File *prove.pl*, realisiert. Diese Klausel erzeugt einen Beweisbaum, der die Herleitung des Satzes legitimiert. Bevor nun ein Durchlauf des PABS-Systems erläutert wird, wird zuerst der Aufbau dieser Beweise dargestellt.

4.2.1 Der Beweiser des PABS-Systems

In dem File *prove.pl* ist der Beweiser für die Herleitung von Sätzen durch die Klauselmenge *prove/6* definiert durch:

prove(<Goal>,<Proof>,<Typlist>,<Depth>,<StateNr>,<StateTyp>).

Bei dem Argument <Goal> handelt es sich um ein zu beweisendes Ziel oder um eine Konjunktion von Zielen, die bewiesen werden sollen. Der <Proof> ist der vom Beweiser erzeugte Beweis für <Goal>. Handelt es sich bei den Zielen um eine Konjunktion, werden die Beweise für die Ziele als Konjunktion in derselben Reihenfolge zurückgegeben, die durch die Ziele vorgegeben ist. Ein Ziel kann legitimiert werden durch

- essentielle Sätze
- built-in Prädikaten
- Regeln der Theorien, die in <Typlist> angegeben sind

Dabei werden

- essentielle Sätze durch das Prädikat *state/3* notiert, mit

  state(<StateNr>,<Typ>,<Satz>).

  <StateNr>:= Zustandsnummer, in dem der essentielle Satz gilt, der das Ziel beweist

  <Typ>:= Typ des essentiellen Satzes, also <Typ> $\in$ { konkret,abstrakt}

  <Satz>:= essentieller Satz, der zum Beweis benötigt wird

- built-in Prädikate werden durch das Prädikat mit seinen Argumenten dargestellt
- Regeln werden durch das Prädikat *regel/3* dargestellt mit

regel(<Nr>,<Typ>,<(<Teilziel>:-<Beweis>)>).

<Nr>:= Regelnummer, die angewandt wurde, um das Ziel zu beweisen

<Typ>:= Art der Regel, die angewendet wurde, mit <Typ> ∈ {konkret, generisch, abstrakt}

<(<Teilziel>:-<Beweis>)>:= Stellt einen <Beweis> für das Ziel <Teilziel> dar, der wieder aus einem Beweisbaum mit den drei beschriebenen Komponenten bestehen kann

BEISPIEL:

(regel(10, konkret, smaller(b, c) :- (state(2, konkret, tower(b, [1])) , state(2, konkret, tower(c, [2])) , 1 < 2)))

Der Parameter <Typlist> gibt an, aus welchen Theorien Regeln zum Beweisen der Ziele verwandt werden dürfen. Bei dem Parameter <Depth> handelt es sich um eine positive, ganze Zahl, die es ermöglicht, die Herleitungstiefe für die Anwendung von Regeln zu begrenzen. Wird dieser Parameter auf 0 gesetzt, ist die Herleitungstiefe unbegrenzt. <StateNr> und <Typlist> bezeichnen die Zustandsnummer und den Typ des Zustandes aus dem essentielle Sätze zum Beweisen der Ziele herangezogen werden dürfen.

4.2.2 Einladen und Starten des PABS-Systems

Nach dem Start von Prolog sollte man zuerst, falls nötig, ein File mit selbst definierten Systemprädikaten einladen. Man kann diesen Aufruf auch in den Rumpf des Prädikats *init/0* schreiben, das in dem File *start.pl* deklariert ist. Um die Dateien des PABS-Systems einzuladen und das System zu starten, muß man als nächstes das File *start.pl* in das Prolog System compilieren. Mit dem Aufruf des Prädikats *init/0* werden dann alle weiteren Dateien, bis auf die Definition der Domäne, eingeladen. Als nächstes werden die Pfade für die Definition der Eingabedaten, wie im Abschnitt 4.1 beschrieben, vom Benutzer erfragt. Diese Pfade sind relativ von der Position, von der das Prologsystem gestartet wurde, oder absolut anzugeben. Möchte man die Dateien der eingestellten Defaultpfade laden, kann man durch Eingabe von <TAB> und <CR> die angegebenen Dateien in das System laden. Die Defaultpfade werden in der Datei *misc.pl* mit dem Prädikat *defaultpath/2* für die verschiedenen Files vorgegeben und können vom Benutzer dort geändert werden.

4.2.3 Das PABS-System im Debug-mode

Nachdem die Eingabedaten für das PABS-System eingelesen wurden, werden die Pläne mit den dazugehörenden Startzuständen, auf die sie angewandt werden sollen, angezeigt. Durch die Eingabe der Plannummer kann der Benutzer einen Plan mit einem Startzustand auswählen, auf den das PABS-Verfahren angewendet wird. Die Eingabe muß mit einem Punkt abgeschlossen werden. Beispielhaft ist hier die Auswahl für die ToH Domäne angegeben, auf die sich der Beispiellauf bezieht.

Plan :1
Startstates : [tower(a, [1, 2, 3]), tower(b, []), tower(c, [])]
Operatorsequenz : [move(a, b), move(a, c), move(b, c), move(a, b), move(c, a), move(c, b), move(a, b)]
Plan :2
Startstates : [tower(a, [1, 2, 3]), tower(b, []), tower(c, [])]
Operatorsequenz : [move(a, c), move(a, b), move(c, b), move(a, c), move(b, a), move(b, c), move(a, c), move(c, b), move(c, a), move(b, a), move(c, b), move(a, c), move(a, b), move(c, b)]
Waehlen Sie :
1.

Als nächstes wird der Benutzer gefragt, ob er im Debug-mode arbeiten möchte, der die Daten angezeigt, die durch die einzelnen Phasen des PABS-Verfahrens erzeugt werden. Zu Beginn der ersten Phase werden noch einmal der ausgewählte Plan und der dazugehörende Startzustand angezeigt. Nun wird die erste Phase des PABS-Verfahrens durchgeführt, die den konkreten Plan auf den Startzustand anwendet und die Zustandsbeschreibungen der Folgezustände berechnet. Für jeden erzeugten essentiellen Satz wird ein Beweis konstruiert, der dem Beweis der Vorbedingungen des angewendeten Operators, der den jeweiligen essentiellen Satz erzeugt hat, entspricht. Diese Beweise werden intern in dem Prädikat *state/8* gespeichert. Im Debug-mode werden die Zustandsbeschreibungen nach der ersten Phase in verkürzter Form ausgegeben:

state(<Nr>,konkret,<Ess.Satz>).

<Nr>:= Zustandsnummer, in dem der Fakt gilt

<Ess.Satz>:= instanziierter essentieller Satz, der in diesem konkreten Zustand gilt

States nach Phase1:
state(0, konkret, tower(a, [1, 2, 3])).
state(0, konkret, tower(b, [])).
state(0, konkret, tower(c, [])).
state(1, konkret, tower(c, [])).
state(1, konkret, tower(a, [2, 3])).
state(1, konkret, tower(b, [1])).
state(2, konkret, tower(b, [1])).
state(2, konkret, tower(a, [3])).
state(2, konkret, tower(c, [2])).
⋮
In der zweiten Phase wird nun die generische Theorie auf die konkreten Zustandsbeschreibungen angewandt, um die abstrakten Zustandsbeschreibungen zu erzeugen. Dort werden alle beweisbaren, abstrakten, essentiellen Sätze durch den Beweiser hergeleitet und in die Prolog Datenbasis eingefügt. Diese abstrakten essentiellen Sätze mit ihrem Beweis werden nach der zweiten Phase angezeigt. Die Beweise haben den Aufbau, wie er im vorherigen Abschnitt beschrieben wurde. Die Beschreibung der abstrakten essentiellen Sätze entspricht

der nach der ersten Phase, wobei der Typ des essentiellen Satzes mit dem Term abstrakt ausgegeben wird.
state(0, abstrakt, abstower(a, [st, ld])).
<— regel(2, generisch, abstower(a, [st, ld]) :- state(0, konkret, tower(a, [1, 2, 3])))
state(0, abstrakt, abstower(b, [])).
<— regel(3, generisch, abstower(b, []) :- state(0, konkret, tower(b, [])))
state(0, abstrakt, abstower(c, [])).
<— regel(3, generisch, abstower(c, []) :- state(0, konkret, tower(c, [])))
state(1, abstrakt, abstower(c, [])).
<— regel(3, generisch, abstower(c, []) :- state(1, konkret, tower(c, [])))
state(1, abstrakt, abstower(b, [sd])).
<— regel(5, generisch, abstower(b, [sd]) :- state(1, konkret, tower(b, [1])))
state(1, abstrakt, abstower(a, [md, ld])).
<— regel(6, generisch, abstower(a, [md, ld]) :- state(1, konkret, tower(a, [2, 3])))
state(2, abstrakt, abstower(a, [ld])).
<— regel(1, generisch, abstower(a, [ld]) :- state(2, konkret, tower(a, [3])))
state(2, abstrakt, abstower(c, [md])).
<— regel(4, generisch, abstower(c, [md]) :- state(2, konkret, tower(c, [2])))
state(2, abstrakt, abstower(b, [sd])).
<— regel(5, generisch, abstower(b, [sd]) :- state(2, konkret, tower(b, [1])))
⋮
In der dritten Phase werden alle möglichen abstrakten Pfade gesucht, die vom abstrakten Startzustand zum abstrakten Zielzustand führen. Alle Kanten, die zu einem abstrakten Pfad beitragen, werden durch das Prädikat *applicable/4* repräsentiert, mit:

applicable(<Source>,<Dest>,<Op>,<OpNr>).

wobei

<Source>:= die Nummer des Ausgangszustands ist, indem der Operator angewendet wird.

<Dest>:= die Zustandsnummer ist, zu dem der Operator hinführt.

<Op>:= der instanziierte Operator ist, der den Übergang realisiert.

<OpNr>:= die Operatornummer der Operatordeklaration ist, die den Bedingungen des Übergangs genügt.

Applicable Operators after Phase3 :
applicable(0, 3, movetower(a, c), 3).
applicable(3, 4, movelargedisk(a, b), 4).
applicable(4, 7, movetower(c, b), 3).
applicable(0, 2, difftower(a), 5).
applicable(2, 5, swap(b, a), 7).
applicable(5, 7, undifftower(b), 6).
applicable(2, 5, swap(a, b), 7).

In der vierten Phase werden dann alle möglichen Wege vom abstrakten Startzustand zum abstrakten Zielzustand auf deren Konsistenz überprüft. Dem Benutzer werden dann im Debug-mode alle konsistenten, vollständigen Pfade angezeigt, die die Menge der abstrahierten Pläne darstellen.

Planauswahl:
Plan 1
[applicable(0, 3, movetower(a, c), 3), applicable(3, 4, movelargedisk(a, b), 4),
applicable(4, 7, movetower(c, b), 3)]
Plan 2
[applicable(0, 2, difftower(a), 5), applicable(2, 5, swap(b, a), 7), applicable(
5, 7, undifftower(b), 6)]
Plan 3
[applicable(0, 2, difftower(a), 5), applicable(2, 5, swap(a, b), 7), applicable(
5, 7, undifftower(b), 6)]

Außerdem werden dem Benutzer die Beweise für die Übergänge angezeigt. Von diesen Plänen wählt der Benutzer nun einen Plan aus, der in der fünftem Phase in einen Skelettplan transformiert wird. Nach der fünften Phase wird der gewonnene Skelettplan dem Benutzer mit seinen Komponenten angezeigt.

Phase5: Skelettplangenerierung [5]
ZINIT:
[abstower(_g6921, [st, ld]), abstower(_g6939, []), abstower(_g6949, [])]
ZGOAL:
[abstower(_g6949, [st, ld]), abstower(_g6939, []), abstower(_g6921, [])]
RENABLE:
[_g6939 \= _g6949, _g6921 = _g7009]
RAI:
[[difftower(_g7021), _g6921 = _g7021], [swap(_g7043, _g7045), _g7045 = _g6921,
_g7043 = _g6949], [undifftower(_g7077), _g7077 = _g6949]]
RPLAN:
[]

Der Benutzer kann sich diesen Plan nun in ein File schreiben lassen, um ihn dann in weiteren Verfahren, wie zum Beispiel dem hierarchischen Planen mit abstrakten Plänen, weiterverwenden zu können. Somit ist das PABS-Verfahren beendet und der Benutzer wird gefragt, ob er eine andere Domäne einladen möchte oder ob er erneut mit derselben Domäne eine Planabstraktion nach der PABS-Methode durchführen möchte. Nach dem Verlassen des PABS-Systems oder bei der Wahl einer neuen Domäne werden die Definitionen der alten Domäne gelöscht, so daß es möglich ist, die Definitionen der Domänen zu ändern und das Verfahren erneut zu starten. Das Löschen der Domänendefinitionen ist auch jeder Zeit in Prolog mit dem Prädikat *clear/0* möglich.

[5]Dieser Skelettplan gehört zu dem Plan 2 aus der vierten Phase

Kapitel 5

Diskussion

Mit Hilfe der Prolog Implementierung des PABS-Verfahrens und den Definitionen für die ToH-Domäne, beziehungsweise die der add-Domäne, war es bei einer akzeptabelen Laufzeit möglich, abstrakte Skelettpläne zu generieren, die den erwarteten Plänen bei der manuellen Durchführung des Verfahrens entsprachen. Im Vergleich zu realen Anwendungen sind diese Domänen jedoch sehr klein. Es wäre interessant, die Konstruktion einer Domänendefinition und die Laufzeit der Implementation bei einer realen Anwendung zu betrachten.
Die von der Implementierung des PABS-Verfahrens erzeugten Skelettpläne liegen in einer Form vor, so daß sie durch einen hierarchischen Planer zur Lösung eines konkreten Planungsproblems verfeinert werden können. Dieser Planer wird im Rahmen einer Projektarbeit [Surmann, 1993] erstellt. Des weiteren werden dort die Laufzeiten bei der Suche nach einer konkreten Lösung mit der Unterstützung eines abstrakten Plans und bei der Suche nach einer konkreten Lösung ohne abstrakten Skelettplan verglichen. Die Ergebnisse dieser Arbeit werden zeigen, ob die Lösung von konkreten Planungsproblemen durch Planabstraktion gegenüber anderen Verfahren einen Performancevorteil aufweist, der die Implementierung des PABS-Verfahrens motivierte.
Des weiteren wäre eine Zusammenführung dieser beiden Implementationen unter einer gemeinsamen Benutzeroberfläche wünschenswert. Diese Oberfläche sollte die beiden Verfahren auch graphisch veranschaulichen, um eine komfortable Erstellung von Domänenspezifikationen zu ermöglichen.
Ein weiteres Problem liegt in der Ausdrucksschwäche der STRIPS Notation für die Planungswelten, die eine Formulierung von komplexen Domänen erschweren oder unmöglich machen. Hier könnte eine Lösung darin bestehen, eine einheitliche Schnittstelle zwischen den Planungswelten und dem eigentlichen Prozeß der Abstraktion zu finden, die unabhängig von der Repräsentation der Planungswelten, die Abstraktion realisiert.
Als Erweiterung des PABS-Verfahrens wird in einer zukünftigen Arbeit das S-PABS-Verfahren [Bergmann, 1993b; Bergmann, 1993a] implementiert. Dabei handelt es sich um ein EBL-Verfahren, in dem Abstraktion und hierarchische Cluster-Verfahren kombiniert werden, um automatisch eine Hierarchie von abstrakten Plänen zu erzeugen. Diese Hierarchie aus abstrakten Plänen, die sich aus dem Grad der Abstraktion der einzelnen Pläne ergibt, hat beim Lernen aus mehreren Beispielen beim shared-Abstraction-Verfahren eine Reduzierung der Komplexität bei der Überprüfung der Anwendbarkeit zur Folge.

Anhang A

Domänenspezifikation Towers of Hanoi (ToH)

hanoi.plan:

```
% Konkrete Pläne


plan(1,konkret,[tower(a,[1,2,3]),tower(b,[]),tower(c,[]),start],
      [move(a,b),move(a,c),move(b,c),move(a,b),move(c,a),move(c,b),move(a,b)]).
plan(2,konkret,[tower(a,[1,2,3]),tower(b,[]),tower(c,[])],
      [move(a,c),move(a,b),move(c,b),move(a,c),move(b,a),move(b,c),move(a,c),
      move(c,b),move(c,a),move(b,a),move(c,b),move(a,c),move(a,b),move(c,b)]).
```

hanoi.kon:

```
% Definition konkreter essential Sentence
essential(tower(_,_),konkret).
% Definition der konkreten Operatoren
operator(move(X,Y),1,konkret,
          (smaller(X,Y),tower(Y,L)),
          (tower(X,[Disk|Disks]),tower(Y,[_,_])),
          (tower(X,Disks),tower(Y,[Disk|L]))).
operator(move(X,Y),2,konkret,
          (tower(X,[Disk|Disks]),tower(Y,[])),
          (tower(X,[Disk|Disks]),tower(Y,[])),
          (tower(X,Disks),tower(Y,[Disk]))).


% Definition einer konkreten Theorie
regel(10,konkret,
      (smaller(X,Y) :- tower(X,[Disk|_]),tower(Y,[DDisk|_]),Disk < DDisk)).
```

hanoi.gen:

```
% Definition einer generischen Theorie
regel(0,generisch,
      (abstower(X,[st]) :- tower(X,[1,2]))).
regel(1,generisch,
      (abstower(X,[ld]) :- tower(X,[3]))).
regel(2,generisch,
      (abstower(X,[st,ld]) :- tower(X,[1,2,3]))).
regel(3,generisch,
      (abstower(X,[]) :- tower(X,[]))).
regel(4,generisch,
      (abstower(X,[md]) :- tower(X,[2]))).
regel(5,generisch,
      (abstower(X,[sd]) :- tower(X,[1]))).
regel(6,generisch,
      (abstower(X,[md,ld]) :- tower(X,[2,3]))).
```

hanoi.abs:

```
% Definition der operationalen Prädikate zur Generalisierung

operational((_ =_ ),abstrakt).
operational((_ \ =_ ),abstrakt).

% Definition abstrakte essentieller Sätze

essential(abstower(_ ,_ ),abstrakt).

% Definition der abstrakten Operatoren


operator(movetower(X,Y),3,abstrakt,
          (abstower(X,[st|Rest]),abstower(Y,Disks),XY,Disks[st]),
          (abstower(X,[st|Rest]),abstower(Y,Disks)),
          (abstower(X,Rest),abstower(Y,[st|Disks]))).

operator(movelargedisk(X,Y),4,abstrakt,
          (abstower(X,[ld|Rest]),abstowerleer(Y),XY),
          (abstower(X,[ld|Rest]),abstower(Y,[])),
          (abstower(X,Rest),abstower(Y,[ld]))).

operator(difftower(X),5,abstrakt,
          (abstower(X,[st,ld]),abstower(Y,[]),abstower(Z,[]),Y\ =Z),
          (abstower(X,[st,ld]),abstower(Y,[]),abstower(Z,[])),
          (abstower(X,[ld]),abstower(Y,[md]),abstower(Z,[sd]))).
```

```
operator(undifftower(X),6,abstrakt,
        (abstower(X,[ld]),abstower(Y,[md]),abstower(Z,[sd])),
        (abstower(X,[ld]),abstower(Y,[md]),abstower(Z,[sd])),
        (abstower(X,[st,ld]),abstower(Y,[]),abstower(Z,[]))).

operator(swap(Y,X),7,abstrakt,
        (abstower(X,[Disk1]),abstower(Y,[Disk2]),Disk1[],Disk2[]),
        (abstower(X,[Disk1]),abstower(Y,[Disk2])),
        (abstower(X,[Disk2]),abstower(Y,[Disk1]))).

operator(firststep(X,Y),8,abstrakt,
        (abstower(X,[st,ld]),abstower(Y,[])),
        (abstower(X,[st,ld]),abstower(Y,[])),
        (abstower(X,[md,ld]),abstower(Y,[sd]))).


% Definition einer abstrakten Theorie


regel(20,abstrakt,
    (abstowerleer(X) :- abstower(X,[]))).
regel(21,abstrakt,
    (abstower(X,[sd,md]) :- abstower(X,[st]))).
```

Anhang B

Domänenspezifikation Add

add.sytem:

```
simplify_nat(L,SL) :-
        normterm(L,L1),
        term_to_list(L1,L2),
        sort_term_list(L2,L3),
        list_to_term(L3,SL),!.


normterm( (T1 + T2), (T1 + T2) ) :- ground(T1), ground(T2),!.
normterm(T,T) :- prod_term(T),!.
normterm( (T1 + T2), (T1R + T2R) ) :-
          normterm(T1,T1R), normterm(T2,T2R).
normterm( (T1 * (T3 + T4)), (T1R + T2R)) :-
          !,normterm( (T1 * T3) , T1R),
          normterm( (T1 * T4) , T2R).
normterm( ((T3 + T4) * T1), (T1R + T2R)) :-
          !,normterm( (T3 * T1) , T1R),
          normterm( (T4 * T1) , T2R).
normterm( (T1 * T2), TR) :-
          normterm(T1,T1R),
          normterm(T2,T2R),
          normterm((T1R * T2R),TR).


ground(X) :- number(X).
ground(X) :- atom(X).
ground(nat_of(_)).

prod_term(X) :- ground(X).
prod_term(X*Y) :- prod_term(X),prod_term(Y).
```

```
term_to_list( (X + Y) , L) :-
        term_to_list(X,Lx), term_to_list(Y,Ly), append(Lx,Ly,L).
term_to_list( (X*Y), [L] ) :-
        term_to_list(X,[Lx]),term_to_list(Y,[Ly]), append(Lx,Ly,L).
term_to_list( X, [[X]]) :- ground(X).


sort_term_list(L,L1) :-
        sort_elements(L,LL),
        sort(0,=¡,LL,L2),
        red_add(L2,L1).
sort_elements([],[]).
sort_elements([X|Y],[Xs|Ys]) :-
        sort(0,=¡,X,XXs),
        red(XXs,XXXs),
        red2(XXXs,XXXXs),
        red1(XXXXs,Xs),
        sort_elements(Y,Ys).
red([X,Y|L],L1) :-
        number(X), number(Y),!,
        Z is X * Y,
        red([Z|L],L1).
red(X,X).
red2([nat_of(T),nat_of(T)|Rest],L) :- !,
        red2([nat_of(T)|Rest],L).
red2([X|L],[X|LL]) :-
        red2(L,LL).
red2([],[]).
red1([N|L],m(L,N)) :- number(N),!.
red1(L,m(L,1)).
red_add([m(X,N1),m(X,N2)|Z],R) :-
        M is N1 + N2, M 0,!,
        red_add([m(X,M)|Z],R).
red_add([m(X,N1),m(X,N2)|Z],R) :-
        0 is N1 + N2,!,
        red_add(Z,R).
red_add([X|Z],[X|L]) :-
        red_add(Z,L).
red_add([],[]).
list_to_term([m(L,N)],T) :-
        list_prod(L,LTerm),
        ( (LTerm = 1, T = N);
        (T=(N*LTerm))).
list_to_term([m(L,N)|Rest],T + RestTerm) :-
        list_prod(L,LTerm),
        list_to_term(Rest,RestTerm),
        ( (LTerm = 1, T = N);
```

```
            (T=(N*LTerm))).
list_to_term([],0).
list_prod([X],X).
list_prod([X|Y],(X * YT)) :-
            list_prod(Y,YT).
list_prod([],1).
```

add.plan:

```
% Konkrete Pläne
plan(1,konkret,[biteq(x1,t1),biteq(x2,t2),biteq(x3,t3),biteq(x4,t4)],
      [xor(x1,x3,x5),and(x1,x3,x8),xor(x2,x4,x6),xor(x6,x8,x6),
      and(x2,x4,x7),and(x2,x8,x9),or(x7,x9,x7),and(x4,x8,x9),or(x7,x9,x7)]).
plan(2,konkret,[biteq(x1,t1),biteq(x2,t2),biteq(x3,t3),biteq(x4,t4)],
      [xor(x1,x3,x5)]).
```

add.kon:

```
essential(biteq(_,_),konkret).


operator(and(X,Y,Z),1,konkret,
      (biteq(X,T1),biteq(Y,T2)),
      (biteq(Z,_)),
      (biteq(Z,and(T1,T2)))).

operator(or(X,Y,Z),2,konkret,
      (biteq(X,T1),biteq(Y,T2)),
      (biteq(Z,_)),
      (biteq(Z,or(T1,T2)))).

operator(xor(X,Y,Z),3,konkret,
      (biteq(X,T1),biteq(Y,T2)),
      (biteq(Z,_)),
      (biteq(Z,xor(T1,T2)))).

operator(set(X,Z),4,konkret,
      (biteq(X,T1)),
      (biteq(Z,_)),
      (biteq(Z,T1))).


regel(10,konkret,
      (simplify_bool(X,X))).
```

add.gen:

```
regel(0,generisch,
      (nateq(y(X1,X2),T) :-
            biteq(X1,T1), biteq(X2,T2), bitvec(X1,X2,_),
            map_to_n(T1,TN1), map_to_n(T2,TN2),
            simplify_nat(TN1+2*TN2,T))).
regel(1,generisch,
      (nateq(y(X1,X2,X3),T) :-
            biteq(X1,T1), biteq(X2,T2), biteq(X3,T3),
            bitvec(X1,X2,X3),
            map_to_n(T1,TN1), map_to_n(T2,TN2), map_to_n(T3,TN3),
            simplify_nat(TN1+(2*TN2)+(4*TN3),T))).
regel(2,generisch,
      (map_to_n(and(T1,T2), TN1 * TN2) :-
      map_to_n(T1,TN1), map_to_n(T2,TN2))).
regel(3, generisch,
      (map_to_n(or(T1,T2), ((TN1+TN2) + ((-1)*TN1*TN2))) :-
            map_to_n(T1,TN1), map_to_n(T2,TN2))).
regel(4, generisch,
      (map_to_n(xor(T1,T2), ((TN1+TN2) + ((-2)*TN1*TN2))) :-
            map_to_n(T1,TN1), map_to_n(T2,TN2))).
regel(5, generisch,
      (map_to_n(T,nat_of(T)) :- atom(T))).


regel(6, generisch, bitvec(x1,x2,x3)).
regel(7, generisch, bitvec(x2,x3,x4)).
regel(8, generisch, bitvec(x3,x4,x5)).
regel(9, generisch, bitvec(x4,x5,x6)).
regel(10,generisch, bitvec(x5,x6,x7)).
regel(11,generisch, bitvec(x6,x7,x8)).
regel(12,generisch, bitvec(x7,x8,x9)).
```

add.abs:

```
essential(nateq(_,_),abstrakt).
operational(simplify_nat(_,_),abstrakt).


operator(add(X,Y,Z),10,abstrakt,
         (nateq(X,T1), nateq(Y,T2), simplify_nat((T1+T2),T3)),
         (nateq(dummy,dummy)),
         (nateq(Z,T3))).
```

Anhang C

Beispiellauf

SEPIA Version 3.0.5, Wed Jul 25 16:33 1990 Copyright ECRC GmbH
sepia: compile("/home.jaspis/wilke/pabs/start.pl").
start.pl compiled 8388 bytes in 0.10 seconds

yes.
sepia: init.
debug.pl compiled 14172 bytes in 0.12 seconds
misc.pl compiled 1188 bytes in 0.02 seconds
prove.pl compiled 2144 bytes in 0.02 seconds
generalize.pl compiled 5108 bytes in 0.03 seconds
Domains/add.syst compiled 9044 bytes in 0.10 seconds
systempraedikate.pl compiled 200 bytes in 0.00 seconds
phase1.pl compiled 2864 bytes in 0.02 seconds
phase2.pl compiled 764 bytes in 0.02 seconds
phase3.pl compiled 3420 bytes in 0.05 seconds
phase4.pl compiled 4772 bytes in 0.05 seconds
phase5.pl compiled 38320 bytes in 0.33 seconds

Geben Sie den Pfadnamen fuer die konkrete Theorie ein
DEFAULTPATH : <Domains/hanoi.kon> TAB tippen <CR>

Domains/hanoi.kon compiled 2800 bytes in 0.02 seconds
Geben Sie den Pfadnamen fuer die generische Theorie ein
DEFAULTPATH : <Domains/hanoi.gen> TAB tippen <CR>

Domains/hanoi.gen compiled 700 bytes in 0.02 seconds
Geben Sie den Pfadnamen fuer die abstrakte Theorie ein
DEFAULTPATH : <Domains/hanoi.abs> TAB tippen <CR>

Domains/hanoi.abs compiled 2744 bytes in 0.07 seconds
Geben Sie den Pfadnamen fuer die konkreten Plaene ein

DEFAULTPATH : <Domains/hanoi.plan> TAB tippen <CR>

Domains/hanoi.plan compiled 3436 bytes in 0.02 seconds

Plan :1

Startstates : [tower(a, [1, 2, 3]), tower(b, []), tower(c, [])]

Operatorsequenz : [move(a, b), move(a, c), move(b, c), move(a, b), move(c, a), move(c, b), move(a, b)]
Plan :2
Startstates : [tower(a, [1, 2, 3]), tower(b, []), tower(c, [])]
Operatorsequenz : [move(a, c), move(a, b), move(c, b), move(a, c), move(b, a), move(b, c), move(a, c), move(c, b), move(c, a), move(b, a), move(c, b), move(a, c), move(a, b), move(c, b)]
Waehlen Sie :
1.
Debugging ? (j/n)
j.
Phase1: Plan propagieren
loading the library /usr/local/lib/sepia/lib/lists.pl

Konkret Start State:
loading the library /usr/local/lib/sepia/lib/strings.pl
state(0, konkrete, tower(a, [1, 2, 3])).
state(0, konkrete, tower(b, [])).
state(0, konkrete, tower(c, [])).
state(0, konkrete, start).

Konkreter Plan:

plan(1, konkret, [tower(a, [1, 2, 3]), tower(b, []), tower(c, []), start], [move(a, b), move(a, c), move(b, c), move(a, b), move(c, a), move(c, b), move(a, b)]).

Weiter mit <RETURN>

States nach Phase1:
state(0, konkret, tower(a, [1, 2, 3])).
state(0, konkret, tower(b, [])).
state(0, konkret, tower(c, [])).
state(0, konkret, start).
state(1, konkret, tower(c, [])).
state(1, konkret, start).
state(1, konkret, tower(a, [2, 3])).
state(1, konkret, tower(b, [1])).

state(2, konkret, start).
state(2, konkret, tower(b, [1])).
state(2, konkret, tower(a, [3])).
state(2, konkret, tower(c, [2])).
state(3, konkret, start).
state(3, konkret, tower(a, [3])).
state(3, konkret, tower(b, [])).
state(3, konkret, tower(c, [1, 2])).
state(4, konkret, start).
state(4, konkret, tower(c, [1, 2])).
state(4, konkret, tower(a, [])).
state(4, konkret, tower(b, [3])).
state(5, konkret, start).
state(5, konkret, tower(b, [3])).
state(5, konkret, tower(c, [2])).
state(5, konkret, tower(a, [1])).
state(6, konkret, start).
state(6, konkret, tower(a, [1])).
state(6, konkret, tower(c, [])).
state(6, konkret, tower(b, [2, 3])).
state(7, konkret, start).
state(7, konkret, tower(c, [])).
state(7, konkret, tower(a, [])).
state(7, konkret, tower(b, [1, 2, 3])).

Weiter mit <RETURN>

Phase2: Generische Theorie anwenden
Abstrakte States nach Phase2:

state(0, abstrakt, abstower(a, [st, ld])). <— regel(2, generisch, abstower(a, [st, ld]) :- state(0, konkret, tower(a, [1, 2, 3])))
state(0, abstrakt, abstower(b, [])). <— regel(3, generisch, abstower(b, []) :- state(0, konkret, tower(b, [])))
state(0, abstrakt, abstower(c, [])). <— regel(3, generisch, abstower(c, []) :- state(0, konkret, tower(c, [])))
state(1, abstrakt, abstower(c, [])). <— regel(3, generisch, abstower(c, []) :- state(1, konkret, tower(c, [])))
state(1, abstrakt, abstower(b, [sd])). <— regel(5, generisch, abstower(b, [sd]) :- state(1, konkret, tower(b, [1])))
state(1, abstrakt, abstower(a, [md, ld])). <— regel(6, generisch, abstower(a, [md, ld]) :- state(1, konkret, tower(a, [2, 3])))
state(2, abstrakt, abstower(a, [ld])). <— regel(1, generisch, abstower(a, [ld]) :- state(2, konkret, tower(a, [3])))
state(2, abstrakt, abstower(c, [md])). <— regel(4, generisch, abstower(c, [md]) :- state(2, konkret, tower(c, [2])))

state(2, abstrakt, abstower(b, [sd])). <— regel(5, generisch, abstower(b, [sd]) :- state(2, konkret, tower(b, [1])))
state(3, abstrakt, abstower(c, [st])). <— regel(0, generisch, abstower(c, [st]) :- state(3, konkret, tower(c, [1, 2])))
state(3, abstrakt, abstower(a, [ld])). <— regel(1, generisch, abstower(a, [ld]) :- state(3, konkret, tower(a, [3])))
state(3, abstrakt, abstower(b, [])). <— regel(3, generisch, abstower(b, []) :- state(3, konkret, tower(b, [])))
state(4, abstrakt, abstower(c, [st])). <— regel(0, generisch, abstower(c, [st]) :- state(4, konkret, tower(c, [1, 2])))
state(4, abstrakt, abstower(b, [ld])). <— regel(1, generisch, abstower(b, [ld]) :- state(4, konkret, tower(b, [3])))
state(4, abstrakt, abstower(a, [])). <— regel(3, generisch, abstower(a, []) :- state(4, konkret, tower(a, [])))
state(5, abstrakt, abstower(b, [ld])). <— regel(1, generisch, abstower(b, [ld]) :- state(5, konkret, tower(b, [3])))
state(5, abstrakt, abstower(c, [md])). <— regel(4, generisch, abstower(c, [md]) :- state(5, konkret, tower(c, [2])))
state(5, abstrakt, abstower(a, [sd])). <— regel(5, generisch, abstower(a, [sd]) :- state(5, konkret, tower(a, [1])))
state(6, abstrakt, abstower(c, [])). <— regel(3, generisch, abstower(c, []) :- state(6, konkret, tower(c, [])))
state(6, abstrakt, abstower(a, [sd])). <— regel(5, generisch, abstower(a, [sd]) :- state(6, konkret, tower(a, [1])))
state(6, abstrakt, abstower(b, [md, ld])). <— regel(6, generisch, abstower(b, [md, ld]) :- state(6, konkret, tower(b, [2, 3])))
state(7, abstrakt, abstower(b, [st, ld])). <— regel(2, generisch, abstower(b, [st, ld]) :- state(7, konkret, tower(b, [1, 2, 3])))
state(7, abstrakt, abstower(c, [])). <— regel(3, generisch, abstower(c, []) :- state(7, konkret, tower(c, [])))
state(7, abstrakt, abstower(a, [])). <— regel(3, generisch, abstower(a, []) :- state(7, konkret, tower(a, [])))
Weiter mit <RETURN>
Phase3: Suche nach abstrakten Pfaden

Applicable Operators after Phase3 :
applicable(0, 3, movetower(a, c), 3).
applicable(3, 4, movelargedisk(a, b), 4).
applicable(4, 7, movetower(c, b), 3).
applicable(0, 2, difftower(a), 5).
applicable(2, 5, swap(b, a), 7).
applicable(5, 7, undifftower(b), 6).
applicable(2, 5, swap(a, b), 7).

Weiter mit <RETURN>

Phase4: Konsistenzpruefung der Pfade

Planauswahl:

Plan 1
[applicable(0, 3, movetower(a, c), 3), applicable(3, 4, movelargedisk(a, b), 4), applicable(4, 7, movetower(c, b), 3)]
Plan 2
[applicable(0, 2, difftower(a), 5), applicable(2, 5, swap(b, a), 7), applicable(5, 7, undifftower(b), 6)]
Plan 3 [applicable(0, 2, difftower(a), 5), applicable(2, 5, swap(a, b), 7), applicable(5, 7, undifftower(b), 6)]
Welcher Plan ?
2.
Nach Phase4 :
plan(abstrakt, [applicable(0, 2, difftower(a), 5), applicable(2, 5, swap(b, a), 7), applicable(5, 7, undifftower(b), 6)]).
applicable(0, 3, movetower(a, c), 3, (state(0, abstrakt, abstower(a, [st, ld])) , state(0, abstrakt, abstower(c, [])) , a \= c , [] \= [st]), (abstower(a, [st, ld]) , abstower(c, []) , a \= c , [] \= [st])).
applicable(3, 4, movelargedisk(a, b), 4, (state(3, abstrakt, abstower(a, [ld])) , regel(20, abstrakt, abstowerleer(b) :- state(3, abstrakt, abstower(b, []))) , a b̄), (abstower(a, [ld]) , abstowerleer(b) , a \= b)).
applicable(4, 7, movetower(c, b), 3, (state(4, abstrakt, abstower(c, [st])) , state(4, abstrakt, abstower(b, [ld])) , c \= b , [ld] \= [st]), (abstower(c, [st]) , abstower(b, [ld]) , c \= b , [ld] \= [st])).
applicable(0, 2, difftower(a), 5, (state(0, abstrakt, abstower(a, [st, ld])) , state(0, abstrakt, abstower(c, [])) , state(0, abstrakt, abstower(b, [])) , c \= b , a = a , a = a , a = a), (abstower(a, [st, ld]) , abstower(c, []) , abstower(b, []) , c \= b , a = a , a = a , a = a)).
applicable(2, 5, swap(b, a), 7, (a = a , b = b , _g14187 = _g14187 , state(2, abstrakt, abstower(a, [ld])) , state(2, abstrakt, abstower(b, [sd])) , ld \= [] , sd \= []), (a = a , b = b , _g14187 = _g14187 , abstower(a, [ld]) , abstower(b, [sd]) , ld \= [] , sd \= [])).
applicable(5, 7, undifftower(b), 6, (b = b , state(5, abstrakt, abstower(b, [ld])) , state(5, abstrakt, abstower(c, [md])) , state(5, abstrakt, abstower(a, [sd]))), (b = b , abstower(b, [ld]) , abstower(c, [md]) , abstower(a, [sd]))).
applicable(2, 5, swap(a, b), 7, (b = b , a = a , _g14187 = _g14187 , state(2, abstrakt, abstower(b, [sd])) , state(2, abstrakt, abstower(a, [ld])) , sd \= [] , ld \= []), (b = b , a = a , _g14187 = _g14187 , abstower(b, [sd]) , abstower(a, [ld]) , sd [] , ld [])).

Weiter mit <RETURN>

Phase5: Skelettplangenerierung

ZINIT:
[abstower(_g6921, [st, ld]), abstower(_g6939, []), abstower(_g6949, [])]

ZGOAL:
[abstower(_g6949, [st, ld]), abstower(_g6939, []), abstower(_g6921, [])]
RENABLE:
[_g6939 \= _g6949, _g6921 = _g7009]
RAI:
[[difftower(_g7021), _g6921 = _g7021], [swap(_g7043, _g7045), _g7045 = _g6921, _g7043 = _g6949], [undifftower(_g7077), _g7077 = _g6949]]
RPLAN:
[]
plan(abstrakt, [applicable(0, 2, difftower(a), 5), applicable(2, 5, swap(b, a), 7), applicable(5, 7, undifftower(b), 6)]).
In ein File schreiben : (j/n)
n.
Noch ne Runde ? (j/n)
n.
yes.

Literaturverzeichnis

[Bergmann, 1990] R. Bergmann. Generierung von Skelettplänen als Problem der Wissensakquisition. Master's thesis, Universität Kaiserslautern, Erwin-Schrödinger-Straße, Postfach 3049, W-6750 Kaiserslautern, Germany, 1990.

[Bergmann, 1992a] R. Bergmann. Knowledge acquisition by generating skeletal plans. In F. Schmalhofer, G. Strube, and Th. Wetter, editors, *Contemporary Knowledge Engineering and Cognition*, pages 125–133, Heidelberg, 1992. Springer.

[Bergmann, 1992b] R. Bergmann. Learning plan abstractions. In H.J. Ohlbach, editor, *GWAI-92 16th German Workshop on Artificial Intelligence*, volume 671 of *Springer Lecture Notes on AI*, pages 187–198, 1992.

[Bergmann, 1993a] R. Bergmann. Integrating abstraction, explanation-based learning from multiple examples and hierarchical clustering with a performance component for planning. In Enric Plaza, editor, *Proceedings of the ECML-93 Workshop on Integrated Learning Architectures (ILA-93)*, Vienna, Austria, 1993.

[Bergmann, 1993b] R. Bergmann. Learning hierarchically clustered shared plan abstractions as problem solving knowledge with high utility for planning. In A. Horz, editor, *Proceedings of GI-Workshop on 'Planung und Konfigurierung' PuK*, number 723 in Arbeitspapiere der GMD, pages 97–108, Bonn, 1993. GMD.

[Charniak and McDermott, 1985] E. Charniak and D. McDermott. *Introduction to artificial intelligence.* Addison-Wesley Publishing, 1985.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. Strips: A new apporach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Friedland and Iwasaki, 1985] P. E. Friedland and Y. Iwasaki. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, pages 161–208, 1985.

[Giordana *et al.*, 1991] A. Giordana, D. Roverso, and L. Saitta. Abstracting background knowledge for concept learning. In Y. Kodratoff, editor, *Lecture Notes in Artificial Intelligence: Machine Learning-EWSL-91*, volume 482, pages 1–13, Berlin, 1991. Springer.

[Hertzberg, 1989] Joachim Hertzberg. *Planen: Einführung in die Planerstellungsmethoden der künstlichen Intelligenz.* B-I Wissenschaftsverlag, 1989.

[Lifschitz, 1987] V. Lifschitz. On the semantics of strips. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 1–9, Timberline, Oregon, 1987.

[Richter, 1989] Michael M. Richter. *Prinzipien der Künstlichen Intelligenz.* Teubner Verlag, 1989.

[Sacerdoti, 1974] E.D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Surmann, 1993] Dagmar Surmann. Implementierung und vergleichende Untersuchung verschiedener Varianten abstraktionsbasierter Planungsverfahren. Projektarbeit, Universität Kaiserslautern, 1993.