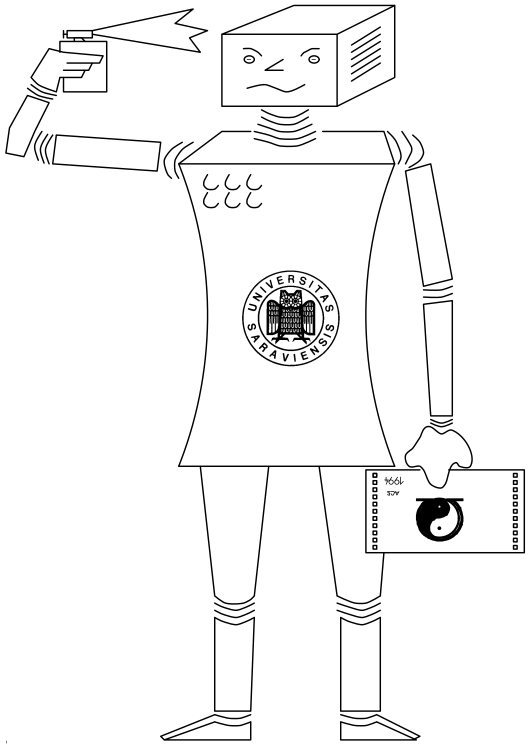
UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D-66041 SAARBRÜCKEN
GERMANY
WWW: http://js-sfbsun.cs.uni-sb.de/pub/www/



Planverfahren

Manfred Kerber

Published as: Script of a lecture, Winter 1993/94, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany

Planverfahren

– Skript zur Vorlesung –

Wintersemester 1993/94

Manfred Kerber

Fachbereich Informatik

Universität des Saarlandes

Inhaltsverzeichnis

Kapitel 1

Grundlagen des Planens

Die Fähigkeit zu planen ist ein wichtiges Merkmal menschlicher Intelligenz. In Planverfahren wird versucht, diese Fähigkeit zumindest teilweise auf dem Computer zu operationalisieren. In diesem Kapitel werden grundlegende Begriffe des Planens eingeführt. Es ist wie einige folgende an [Her89] angelehnt.

1.1 Ziel des Planens

Planen wie es in dieser Vorlesung behandelt wird, soll Aspekte dessen formalisieren, die man auch umgangssprachlich als Planung bezeichnet. Wir wollen zur Verdeutlichung ein Beispiel betrachten. Angenommen man will eine Reise von der Universität des Saarlandes in Saarbrücken zur Stanford University in Palo Alto, Kalifornien, durchführen. Das Ergebnis eines Planungsprozesses soll ein Plan sein, der nach einer Punkt-für-Punkt-Durchführung mit anfänglichem Aufenthalt hier an der Universität zu dem Ergebnis führt, daß man nach der Durchführung in Palo Alto an der Stanford University ist. Wir können zwei Phasen unterscheiden, die *Planerstellung*, die uns im folgenden vor allem interessiert, und die *Planausführung*. Die Planerstellung soll in einer Folge von Operatoren münden, zum Beispiel gehe-zur-Bushaltestelle, steige-in-Bus, steige-am-Bahnhof-aus, steige-in-Zug-nach-Frankfurt, ...
Allgemein gesprochen ist der Ausgangspunkt für einen Planungsprozeß ein *Anfangszustand*[1] („initial state“). Das allgemeine Ziel des Planens ist das Erreichen eines sogenannten *Zielzustandes* („goal state“) mit Hilfe von *Planungsoperatoren* oder auch kurz *Operatoren* genannt. Planungsoperatoren führen einen *Zustand* in einen neuen Zustand über. Zustände werden zur Formalisierung in einer formalen Sprache, in der Regel in einer Variante der Prädikatenlogik erster Stufe (oder einer Teilsprache wie zum Beispiel der Aussagenlogik), beschrieben.
Das heißt ein Plan besteht aus einer Folge von Operatoren, die den Ausgangszustand über eine Reihe von Zwischenzuständen in den Zielzustand überführen.
Vereinfachung: Vereinfachend geht man davon aus, daß man bereits zu Beginn

[1] Im Laufe der Vorlesung werden wir diese Bedingung abschwächen.

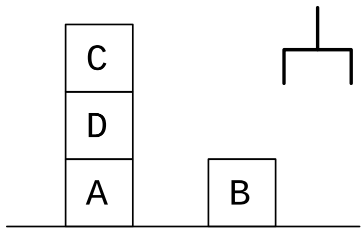
Abbildung 1.1: Ergebnis einer Planung

des Planungsprozesses vollständige Information zur Verfügung hat, das heißt daß die Beschreibung der Welt vollständig ist. Weiterhin nimmt man an, daß der Planer der einzige Agent in der Welt ist, das heißt daß die Welt während der Planausführung lediglich durch die Anwendung der Planungsoperatoren geändert wird. Dies hat zur Folge, daß man erst einen Plan im Ganzen erstellen kann, der, nachdem er erstellt ist, ausgeführt werden kann. Beide Annahmen sind natürlich für viele Anwendungen unrealistisch und es gibt Planverfahren, die ohne sie auskommen. Zum Beispiel wird man im Reiseszenario nicht alle Details im voraus planen wollen und können: zwar will man zum Beispiel die Flugroute Frankfurt nach San Fransisco im voraus planen, auch noch, bei welcher Autovermietung man ein Auto mietet, nicht aber wie man in San Fransisco zur Autovermietung kommt, da sich das vor Ort viel besser planen läßt oder gar die erforderlichen Voraussetzungen in einer sich schnell ändernden Welt noch nicht feststehen. Hinzukommt, daß Pläne bei der Ausführung scheitern können, was dann ein Umplanen erforderlich machen kann.
Aber selbst mit den erwähnten Vereinfachungen ist das Problem der Planerstellung alles andere als einfach. Diese Vereinfachungen abzuschwächen ist ein aktueller Forschungsgegenstand der KI.
Die vollständige Formalisierung des Planungsprozesses für das Reiseszenario ist überaus kompliziert, da sehr viele implizite Annahmen korrekt verarbeitet werden müßten (zum Beispiel: hat die Person einen gültigen Reisepaß, ist sie flugtauglich, hat sie einen Führerschein, ...). Deshalb hat man sich in der KI ein Laborszenario, die sogenannte *Klötzchenwelt* („blocks world“), als Untersuchungsgegenstand für Planverfahren ausgewählt.

1.2 Die Klötzchenwelt

Die Klötzchenwelt ist das klassische Beispiel für ein Planungsszenario. Es ist auch deshalb gewählt worden, da Planungsverfahren historisch eine besondere Rolle für den Einsatz von Robotern haben und viele Planverfahren zur Robotersteuerung wie für Shakey [Nil84] entwickelt wurden.
In einer Klötzchenwelt – sie ist nicht eindeutig, zum Beispiel kann sie in der Zahl, aber auch in den Ausprägungen der Klötzchen variieren – gibt es Klötzchen, die auf einem Tisch oder aufeinanderstehen und die mit einer Roboterhand manipuliert werden können. Abgesehen von der Tatsache, daß die Klötzchen auf dem Tisch stehen, aufeinander stehen oder sich im Greifarm des Roboters befinden, wird ihre gegensei-

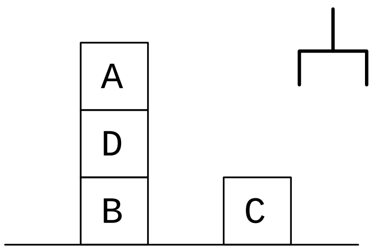
tige Lage vernachlässigt. Also es ist gleichgültig, ob sich ein Klötzchen `x` links oder rechts von einem Klötzchen `y` befindet.
Zur Beschreibung benutzt man die Prädikate `Ontable` mit der intendierten Bedeutung `Ontable(x)` für „`x` steht auf dem Tisch", `On(x,y)` für „`x` steht auf `y`", `Clear(x)` für „auf `x` steht nichts", `Handempty` für „die Roboterhand ist frei" und `Holds(x)` für „die Roboterhand hält das Klötzchen `x`".
Üblicherweise werden Situationen in der Klötzchenwelt außer durch logische Beschreibungen mit diesen Prädikaten durch zweidimensionale Abbildungen beschrieben. Ein Beispiel ist:




```
Ontable(A)
Ontable(B)
On(D,A)
On(C,D)
Clear(C)
Clear(B)
Handempty
```

Abbildung 1.2: Vier Klötzchen auf einem Tisch

Ein Planungsproblem entsteht dadurch, daß man aus einem Anfangszustand einen Zielzustand herleiten möchte, zum Beispiel aus dem Zustand in Abbildung 1.2 den in Abbildung 1.3.




```
Ontable(B)
Ontable(C)
On(D,B)
On(A,D)
Clear(A)
Clear(C)
Handempty
```

Abbildung 1.3: Ein Zielzustand

Umgangssprachlich kommt man vom Ausgangszustand zum Zielzustand, in dem man mit der freien Hand Klötzchen `C` greift und auf den Tisch stellt, dann wird Klötzchen `D` auf das Klötzchen `B` gestellt, was möglich ist, da beide nicht verstellt sind. Das dann freie Klötzchen `A` wird nun auf `D` gestellt, womit der Zielzustand erreicht ist.
Die zulässigen Operatoren zur Manipulation des Klötzchenwelt sind:

- `PICKUP(x)` für das Greifen eines Klotzes vom Tisch
- `PUTDOWN(x)` für das Absetzen auf den Tisch
- `STACK(x,y)` für das Absetzen von `x` auf `y`

- UNSTACK(x,y) für das Greifen eines Klotzes x von einem Klotz y

Um die einzelnen Zustande beschreiben zu können, aber auch bei einigen Ansätzen zur Problemlösung kommt der Prädikatenlogik erster Stufe entscheidende Bedeutung zu. Daher sollen im folgenden Abschnitt die Grundlagen kurz wiederholt werden.

1.3 Prädikatenlogik erster Stufe

Prädikatenlogik erster Stufe wird vorzugsweise benutzt, um Zustände in Planverfahren zu beschreiben. Wir wollen daher ihre Syntax und wichtige Eigenschaften hier kurz wiederholen. Auf die Semantik der Prädikatenlogik erster Stufe wird hier nicht eingegangen, sondern auf die entsprechende Logikvorlesung verwiesen.
Die logischen Zeichen der Prädikatenlogik sind:

- eine unendliche Menge von *Variablen* x, y und z (x_i),
- der einstellige *Junktor* $\neg$ („nicht“), die zweistelligen *Junktoren* $\wedge$ („und“), $\vee$ („oder“), $\Rightarrow$ („impliziert“) und $\Leftrightarrow$ („äquivalent“),
- die *Quantoren* $\forall$ („für alle“) und $\exists$ („es gibt ein“) sowie
- die *Hilfssymbole*), (und ,.

Die Signatur $\mathcal{S}$ besteht für jedes $n \in \mathbb{N}$ aus einer (möglicherweise leeren) Menge von n-stelligen *Relationssymbolen*[2], die wir in der Regel mit P, Q und R bezeichnen, sowie einer (möglicherweise leeren) Menge von n-stelligen *Funktionssymbolen*, die wir in der Regel mit f, g und h bezeichnen. Nullstellige Funktionssymbole heißen auch *Konstante* und werden mit c, d und e bezeichnet.
Bevor wir den Begriff einer Formel definieren können, brauchen wir den Begriff „Term“. Terme werden allgemein induktiv definiert.

1.1 DEFINITION (TERME): *Terme* zu einer Signature $\mathcal{S}$ sind genau die Ausdrücke, die sich durch endlichmalige Anwendung der folgenden Regeln erhalten lassen:

T1 Jede Variable ist ein Term.

T2 Jede Konstante ist ein Term.

T3 Sind $t_1, \ldots, t_n$ Terme und ist f ein n-stelliges Funktionssymbol, so ist auch $f(t_1, \ldots, t_n)$ ein Term. ■

1.2 DEFINITION (FORMELN DER PRÄDIKATENLOGIK ERSTER STUFE): *Formeln* der Prädikatenlogik erster Stufe zu einer Signature $\mathcal{S}$ sind genau die Ausdrücke, die sich durch endlichmalige Anwendung der folgenden Regeln erhalten lassen:

[2] Diese werden auch Prädikatensymbole genannt.

F1 Sind $t_1, \ldots, t_n$ Terme und ist P ein n-stelliges Relationsssymbol, dann ist der Ausdruck $P(t_1, \ldots, t_n)$ eine Formel. So gebildete Formeln heißen *atomar*.

F2 Ist φ eine Formel, so ist $(\neg\varphi)$ eine Formel.

F3 Sind φ und ψ Formeln, so sind auch $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$ und $(\varphi \Leftrightarrow \psi)$ Formeln.

F4 Ist φ eine Formel und x eine Variable, so sind auch $(\forall x \varphi)$ und $(\exists x \varphi)$ Formeln. ■

Wir vereinbaren, daß die Bindungsstärke der Junktoren $\neg$, $\wedge$, $\vee$, $\Rightarrow$ und $\Leftrightarrow$ in dieser Reihenfolge abnimmt und können somit viele überflüssige Klammern weglassen.

1.3 Beispiel: Formeln der Prädikatenlogik sind zum Beispiel „$\forall x$Mensch$(x) \Rightarrow$ sterblich(x)“, „Mensch(Sokrates)“ oder „sterblich(Sokrates)“ sind entsprechende Formeln. ■

1.4 Bemerkung: Weitere wichtige Begriffe der Logik sind die

- der *Tautologie* (Formel, die in der Semantik immer wahr ist),
- der *Folgerung* (φ ist Folgerung von Γ, $\Gamma \models \varphi$, wenn alle Formeln einer Formelmenge Γ wahr sind, dann ist auch φ wahr). ■

1.5 Bemerkung: Einige wichtige Eigenschaften der Prädikatenlogik erster Stufe sind:

1. die *Unentscheidbarkeit*: es gibt keinen terminierenden Algorithmus, der die Frage, ob eine Formel Folgerung einer gegebenen Formelmenge ist, für beliebige Eingaben richtig beantwortet. Gleiches gilt für die Frage, ob eine Formel eine Tautologie ist.
2. die *Monotonie*: folgt eine Formel φ aus einer Formelmenge Γ, so folgt φ auch aus jeder Obermenge von Γ.
3. das *Extensionalitätsprinzip*: der Wahrheitswert einer zusammengesetzten Aussage läßt sich allein aus den Wahrheitswerten ihrer Bestandteile bestimmen.
4. die *Explosivität*: aus einem Widerspruch kann man beliebige Aussagen folgern. Somit muß man sich vor Widersprüchen hüten, da sonst jeder Unsinn hergeleitet werden kann.
5. die *Zweiwertigkeit*: in der Semantik wird jeder Formel der Wert „wahr“ oder „falsch“ zugeordnet. ■

Oft ist es hinreichend, Teilsprachen der Prädikatenlogik erster Stufe zu betrachten, die angenehme Eigenschaften haben, wie zum Beispiel die Aussagenlogik, für die die Folgerungsrelation entscheidbar ist.

Wesentlich für das Rechnen mit Logiken sind Kalküle, die der Semantik entsprechen, die also korrekt und vollständig sind. Für die Prädikatenlogik erster Stufe sind eine Reihe solcher Kalküle entwickelt worden, zum Beispiel der Resolutionskalkül, der im automatischen Beweisen eine wichtige Rolle spielt. Zum vertieften Studium gibt es eine Vielzahl von einführenden Logikbüchern wie [Ric78, EFT86, Men87]. Einen Überblick über das ganze Gebiet der Logik wird in den Handbüchern [Bar77, GG89] gegeben. Wer sich für automatisches Beweisen interessiert sei auf eines der zahlreichen Lehrbücher verwiesen [Bib82, BB87, BM79, CL73, Duf91, Fit90, Lov78].

1.4 Planen durch Deduktion

In Abschnitt 1.2 haben wir Zustände durch prädikatenlogische Formeln beschrieben. Dies bereitet – zumindest für die Klötzchenwelt – keine Probleme. Will man aber den ganzen Planungsprozeß in Prädikatenlogik beschreiben, dann steht erst einmal die Monotonieeigenschaft dem im Wege: Nachdem in der Situation von Abbildung 1.2 Klötzchen `C` vom Roboterarm hochgehoben worden ist, steht es halt nicht mehr auf `D`.

Deshalb haben J. McCarthy und P. Hayes [MH69] den sogenannten Situationenkalkül entwickelt. Die Hauptidee besteht darin, daß jedem Prädikat noch ein weiteres Argument gegeben wird, in dem der jeweilige Zustand steht. Nennen wir also den Anfangszustand wie er in Abbildung 1.2 dargestellt ist `Init` und den Zielzustand von Abbildung 1.3 `Goal`, so sind die zugehörigen Beschreibungen der Zustände `Init` und `Goal` gegeben durch:

Ontable(A,Init)	Ontable(B,Goal)
Ontable(B,Init)	Ontable(C,Goal)
On(D,A,Init)	On(D,B,Goal)
On(C,D,Init)	On(A,D,Goal)
Clear(C,Init)	Clear(A,Goal)
Clear(B,Init)	Clear(C,Goal)
Handempty(Init)	Handempty(Goal)

Zur Manipulation der Zustände benötigen wir dann noch die Operatoren, die nun als Implikationen geschrieben werden können.

PICKUP(x,ξ)	PUTDOWN(x,ξ)
$\forall$x,ξ Clear(x,ξ) $\wedge$ Ontable(x,ξ) $\wedge$ Handempty(ξ) $\Rightarrow$ Holds(x,PICKUP(x,ξ)) $\wedge$ $\neg$Handempty(PICKUP(x,ξ)) $\wedge$ $\neg$Clear(x,PICKUP(x,ξ)) $\wedge$ $\neg$Ontable(x,PICKUP(x,ξ))	$\forall$x,ξ Holds(x,ξ) $\Rightarrow$ Clear(x,PUTDOWN(x,ξ)) $\wedge$ Ontable(x,PUTDOWN(x,ξ)) $\wedge$ Handempty(PUTDOWN(x,ξ)) $\wedge$ $\neg$Holds(x,PUTDOWN(x,ξ))

$\mathtt{STACK}(\mathtt{x},\mathtt{y},\xi)$	$\mathtt{UNSTACK}(\mathtt{x},\mathtt{y},\xi)$
$\forall \mathtt{x},\mathtt{y},\xi\ \ \mathtt{Holds}(\mathtt{x},\xi)\ \wedge$	$\forall \mathtt{x},\mathtt{y},\xi\ \ \mathtt{Clear}(\mathtt{x},\xi)\ \wedge$
$\quad\mathtt{Clear}(\mathtt{y},\xi)\ \wedge$	$\quad\mathtt{On}(\mathtt{x},\mathtt{y},\xi)\ \wedge$
$\Rightarrow\quad\mathtt{Clear}(\mathtt{x},\mathtt{STACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$	$\quad\mathtt{Handempty}(\xi)$
$\quad\mathtt{On}(\mathtt{x},\mathtt{y},\mathtt{STACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$	$\Rightarrow\quad\mathtt{Holds}(\mathtt{x},\mathtt{UNSTACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$
$\quad\mathtt{Handempty}(\mathtt{STACK}(\mathtt{x},\mathtt{y},\xi))\wedge$	$\quad\mathtt{Clear}(\mathtt{y},\mathtt{UNSTACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$
$\quad\neg\mathtt{Holds}(\mathtt{x},\mathtt{STACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$	$\quad\neg\mathtt{Clear}(\mathtt{x},\mathtt{UNSTACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$
$\quad\neg\mathtt{Clear}(\mathtt{y},\mathtt{STACK}(\mathtt{x},\mathtt{y},\xi))$	$\quad\neg\mathtt{On}(\mathtt{x},\mathtt{y},\mathtt{UNSTACK}(\mathtt{x},\mathtt{y},\xi))\ \wedge$
	$\quad\neg\mathtt{Handempty}(\mathtt{UNSTACK}(\mathtt{x},\mathtt{y},\xi))$

Schauen wir uns zu diesen Operatoren ein ganz einfaches Beispiel an:

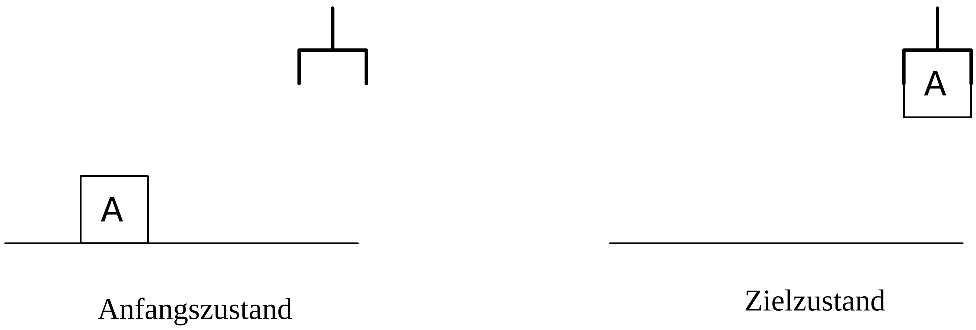


Abbildung 1.4: Ein fast triviales Planungsproblem

Die logische Beschreibung von Anfangs- und Zielzustand lautet:

`Ontable(A,Init)`	`Holds(A,Goal)`
`Clear(A,Init)`	
`Handempty(Init)`	

Durch Anwendung des Operators $\mathtt{PICKUP}(\mathtt{A},\mathtt{INIT})$ erhalten wir aus dem Anfangszustand den Zustand:

$$\begin{array}{l}\mathtt{Holds}(\mathtt{A},\mathtt{PICKUP}(\mathtt{A},\mathtt{INIT}))\\ \neg\mathtt{Handempty}(\mathtt{PICKUP}(\mathtt{A},\mathtt{INIT}))\\ \neg\mathtt{Clear}(\mathtt{A},\mathtt{PICKUP}(\mathtt{A},\mathtt{INIT}))\\ \neg\mathtt{Ontable}(\mathtt{A},\mathtt{PICKUP}(\mathtt{A},\mathtt{INIT}))\end{array}$$

Diese Formelmenge impliziert mit `Goal:=PICKUP(A,INIT)` den Zielzustand. Somit haben wir zur Erreichung des Ziels den Plan `PICKUP(A,INIT)` gefunden.
Daß diese Art des Planens Probleme bereitet, werden wir das nächstemal sehen.

Kapitel 2

Das Frameproblem

In diesem Abschnitt werden wir uns vor allem mit den Schwierigkeiten der Anwendung des Situationenkalküls beschäftigen. Von besonderer Bedeutung ist dabei das Frameproblem. Dazu wollen wir uns zuerst ein Beispiel ansehen.

2.1 Frameaxiome zur Lösung des Frameproblems

Im letzten Abschnitt haben wir gesehen, wie ein fast triviales Planungsproblem im Situationenkalkül gelöst wurde. Betrachten wir nun die leicht schwierigere Situation in Abbildung 2.1.

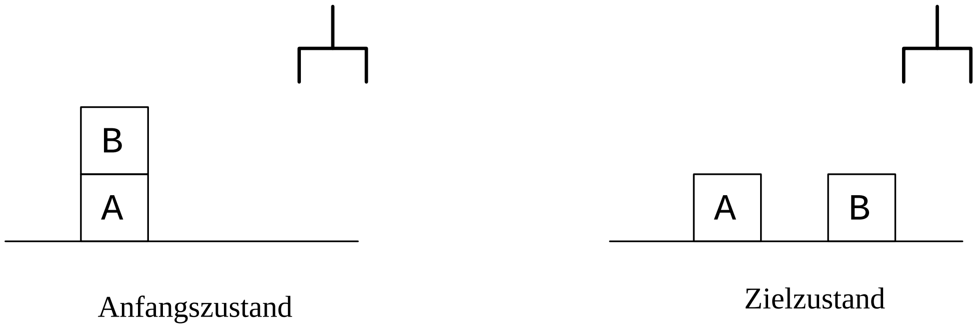


Abbildung 2.1: Planungsaufgabe zum Frameproblem

Die zugehörige Beschreibung von Anfangs- und Zielzustand lautet:

`Ontable(A,Init)`	`Ontable(A,Goal)`
`On(B,A,Init)`	`Clear(A,Goal)`
`Clear(B,Init)`	`Ontable(B,Goal)`
`Handempty(Init)`	`Clear(B,Goal)`
	`Handempty(Goal)`

Intuitiv ist klar, wie man aus dem Anfangszustand zum Zielzustand kommt. Man greift Block `B` und setzt ihn auf dem Tisch ab. In der Operatorensprache bedeutet dies, man erreicht den Zielzustand aus dem Anfangszustand mit dem Plan `Goal := PUTDOWN(B,UNSTACK(B,A,Init))`. Wie sieht das nun formal aus? Was gilt tatsächlich rein logisch in dem intendierten Zustand `PUTDOWN(B,UNSTACK(B,A,Init))`?
Der Operator `UNSTACK(B,A,Init)` ist auf den Anfangszustand anwendbar, da die Bedingungen `Clear(B,Init)`, `On(B,A,Init)` und `Handempty(Init)` erfüllt sind. Wendet man den Operator an, so ergibt sich:

```
Holds(B,UNSTACK(B,A,Init))
Clear(A,UNSTACK(B,A,Init))
¬ Clear(B,UNSTACK(B,A,Init))
¬ On(B,A,UNSTACK(B,A,Init))
¬ Handempty(UNSTACK(B,A,Init))
```

Es ist leicht zu sehen, daß dies auch alle Fakten sind, die im System mit letztem Argument `UNSTACK(B,A,Init)` ableitbar sind. Wegen `Holds(B,UNSTACK(B,A,Init))` ist als nächstes der Operator `PUTDOWN(B,UNSTACK(B,A,Init))` anwendbar. Nach der Anwendung ergibt sich:

```
Clear(B,PUTDOWN(B,UNSTACK(B,A,Init)))
Ontable(B,PUTDOWN(B,UNSTACK(B,A,Init)))
Handempty(PUTDOWN(B,UNSTACK(B,A,Init)))
¬ Holds(B,PUTDOWN(B,UNSTACK(B,A,Init)))
```

Wiederum ist sofort klar, daß mit `PUTDOWN(B,UNSTACK(B,A,Init))` als letztem Argument nur diese vier Fakten innerhalb des Systems abgeleitet werden können.
Nun vergleichen wir die Fakten mit dem Zielzustand
(für `Goal = PUTDOWN(B,UNSTACK(B,A,Init))`):

```
Ontable(A,PUTDOWN(B,UNSTACK(B,A,Init))
Clear(A,PUTDOWN(B,UNSTACK(B,A,Init))
Ontable(B,PUTDOWN(B,UNSTACK(B,A,Init))
Clear(B,PUTDOWN(B,UNSTACK(B,A,Init))
Handempty(PUTDOWN(B,UNSTACK(B,A,Init))
```

Wir sehen, daß wir zwar alle Fakten, die den gewünschten Zustand bezüglich `B` beschreiben herleiten können, daß aber die entsprechenden Fakten bezüglich `A` fehlen und auch nicht hergeleitet werden können. Das heißt

```
Ontable(A,PUTDOWN(B,UNSTACK(B,A,Init))
Clear(A,PUTDOWN(B,UNSTACK(B,A,Init))
```

ist im bisherigen System nicht herleitbar.
Woran liegt das? Es liegt daran, daß in den Formeln, die die Operatoren beschreiben, zwar genau darüber Buch geführt wird, wann die Operatoren anwendbar sind und was die Anwendung der Operatoren im Zustand der Welt verändern, daß sie aber nicht beschreiben, was durch die Anwendung der Operatoren unverändert bleibt. Dieses Problem heißt das *Frameproblem* (Rahmenproblem)[1].
Es kann dadurch gelöst werden, daß man sogenannte *Frameaxiome* zu der Beschreibung hinzufügt, die beschreiben, was sich alles *nicht* ändert.
Im Beispiel der Klötzchenwelt mit den vier Operatoren `PICKUP`, `PUTDOWN`, `STACK` und `UNSTACK` könnte das wie folgt aussehen:

Frameaxiome für `UNSTACK`	
$\forall \mathtt{x,y,z},\xi$ $\mathtt{Ontable(z},\xi)$ $\Rightarrow$ $\mathtt{Ontable(z,UNSTACK(x,y},\xi))$	$\forall \mathtt{x,y,u,v},\xi$ $\mathtt{On(u,v},\xi)$ $\wedge$ $\mathtt{u} \neq \mathtt{x}$ $\Rightarrow$ $\mathtt{On(u,v,UNSTACK(x,y},\xi))$
$\forall \mathtt{x,y,z},\xi$ $\mathtt{Clear(z},\xi)$ $\wedge$ $\mathtt{z} \neq \mathtt{x}$ $\Rightarrow$ $\mathtt{Clear(z,UNSTACK(x,y},\xi))$	...

Wo die Pünktchen stehen müssen, wenn weitere Beschreibungsmerkmale wie zum Beispiel Farbe oder Größe, für den Operator `UNSTACK` jeweils vermerkt werden, daß bei der Anwendung dieses Operators sich die zugehörige Eigenschaft nicht ändert.

Frameaxiome für `PUTDOWN`	
$\forall \mathtt{x,y},\xi$ $\mathtt{Ontable(y},\xi)$ $\Rightarrow$ $\mathtt{Ontable(y,PUTDOWN(x},\xi))$	$\forall \mathtt{x,u,v},\xi$ $\mathtt{On(u,v},\xi)$ $\Rightarrow$ $\mathtt{On(u,v,PUTDOWN(x},\xi))$
$\forall \mathtt{x,y},\xi$ $\mathtt{Clear(y},\xi)$ $\wedge$ $\Rightarrow$ $\mathtt{Clear(y,PUTDOWN(x},\xi))$	...

Für die Operatoren `STACK` und `PICKUP` benötigt man vergleichbare Axiome.

[1] Der Name nimmt Bezug auf bewegte Filmbilder, wo aufeinanderfolgende Rahmen üblicherweise gleichen Hintergrund haben und nur in einigen wenigen Merkmalen abweichen.

Kehren wir nun zu unserem Ausgangsbeispiel zurück. Nun sind in jedem Zustand erheblich mehr Fakten ableitbar.
Wenden wir wie oben den Operator `UNSTACK(B,A,Init)` auf den Anfangszustand an, so erhalten wir außer den obigen Fakten mit den zugehörigen Frameaxiomen – nur das erste ist anwendbar – noch zusätzlich:

```
Ontable(A,UNSTACK(B,A,Init))
```

Wird nun im nächsten Schritt der Operator `PUTDOWN(B,UNSTACK(B,A,Init))` angewandt, so erhält man aus den zugehörigen Frameaxiomen – die zwei links stehenden sind anwendbar – noch zusätzlich:

```
Ontable(A,PUTDOWN(UNSTACK(B,A,Init)))
Clear(A,PUTDOWN(B,UNSTACK(B,A,Init))
```

Die erste Formel ergibt sich dabei aus der zusätzlichen Formel, die aus dem Frameaxiom für `UNSTACK` abgeleitet wurde, die zweite aus dem Resultat der Anwendung von `UNSTACK`.
Somit haben wir also für den Zustandes `PUTDOWN(UNSTACK(B,A,Init))` jetzt die Beschreibung:

```
Holds(B,UNSTACK(B,A,Init))
Clear(A,UNSTACK(B,A,Init))
¬ Clear(B,UNSTACK(B,A,Init))
¬ On(B,A,UNSTACK(B,A,Init))
¬ Handempty(UNSTACK(B,A,Init))
Ontable(A,PUTDOWN(UNSTACK(B,A,Init)))
Clear(A,PUTDOWN(B,UNSTACK(B,A,Init))
```

Also ist der Zielzustand nun auch formal abgeleitet. Zur Übung kann man aus dem Zustand in Abbildung 1.2 den in Abbildung 1.3 im Situationskalkül herleiten.

2.2 Allgemeine Beschreibung

Die allgemeine Beschreibung von Aktionen im Situationenkalkül sieht wie folgt aus (nach [Rei91]):
Sei `OP` ein Operator mit Argumenten $\mathbf{x}$ und `R` ein Prädikat zur Beschreibung von Situationen (auch *Fluent* genannt), dann gibt es ein Axiom für positive Effekte der Form:

$$\pi_{\mathtt{OP}}(\mathbf{x},\xi) \wedge \varepsilon^{+}_{\mathtt{R}}(\mathbf{x},\mathbf{y},\xi) \Rightarrow \mathtt{R}(\mathbf{y},\mathtt{OP}(\mathbf{x},\xi))$$

Dabei sind $\mathbf{x}$ und $\mathbf{y}$ Variablentuple und ξ eine Zustandsvariable. All diese Variablen sind implizit allquantifiziert. In $\pi_{\mathtt{OP}}(\mathbf{x}, \xi)$ werden die Vorbedingungen des Operators abgespeichert und in $\varepsilon^+_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi)$ die des Prädikates. Zum Beispiel kann man für einen Operator `DROP`, bei dessen Ausführung ein Gegenstand fallen gelassen wird, spezifizieren, wann der Gegenstand zu Bruch geht, also ein Prädikat `Broken` erfüllt ist, in dem die Vorbedingung für den Operator $\pi_{\mathtt{DROP}}(\mathtt{x}, \xi)$ zu $\mathtt{Holds}(\mathtt{x}, \xi)$ wird und die des Prädikates $\varepsilon^+_{\mathtt{Broken}}(\mathtt{x}, \mathtt{y}, \xi)$ zu $\mathtt{y} = \mathtt{x} \wedge \mathtt{Fragile}(\mathtt{y})$ gesetzt wird. Die Regel hat dann die Gestalt:

$$\mathtt{Holds}(\mathtt{x}, \xi) \wedge \mathtt{y} = \mathtt{x} \wedge \mathtt{Fragile}(\mathtt{y}) \Rightarrow \mathtt{Broken}(\mathtt{y}, \mathtt{DROP}(\mathtt{x}, \xi))$$

Entsprechend gibt es ein Axiom für negative Effekte der Form:

$$\pi_{\mathtt{OP}}(\mathbf{x}, \xi) \wedge \varepsilon^-_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi) \Rightarrow \neg\mathtt{R}(\mathbf{y}, \mathtt{OP}(\mathbf{x}, \xi))$$

Ein Beispiel dafür ist mit dem Operator `REPAIR` gegeben, der als Vorbedingung verlangt, daß man Leim hat, also $\pi_{\mathtt{REPAIR}}(\xi)$ wird zu $\mathtt{Hasglue}(\xi) \wedge \mathtt{Broken}(\mathtt{x}, \xi)$. Die Bedingung für das Prädikat `Broken` wird mit $\varepsilon^-_{\mathtt{Broken}}(\mathtt{x}, \mathtt{y}, \xi)$ zu $\mathtt{x} = \mathtt{y}$. Also erhält man insgesamt als Regel:

$$\mathtt{Hasglue}(\xi) \wedge \mathtt{Broken}(\mathtt{x}, \xi) \wedge \mathtt{x} = \mathtt{y} \Rightarrow \neg\mathtt{broken}(\mathtt{y}, \mathtt{REPAIR}(\mathtt{x}, \xi))$$

Hinzu kommen dann noch positive Frameaxiome für jedes Prädikat `R` bezüglich einer Operation `OP`:

$$\pi_{\mathtt{OP}}(\mathbf{x}, \xi) \wedge \varphi^+_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi) \wedge \mathtt{R}(\mathbf{y}, \xi) \Rightarrow \mathtt{R}(\mathbf{y}, \mathtt{OP}(\mathbf{x}, \xi))$$

Dieses Axiom besagt, wenn die Vorbedingungen von `OP` erfüllt sind und zusätzliche Bedingungen $\varphi^+_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi)$ erfüllt sind, dann bleibt `R` für $\mathbf{y}$ nach Ausführung von `OP` wahr.
Analog gibt es für jeden Operator `OP` und jedes Prädikat `R` das negative Frameaxiom:

$$\pi_{\mathtt{OP}}(\mathbf{x}, \xi) \wedge \varphi^-_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi) \wedge \neg\mathtt{R}(\mathbf{y}, \xi) \Rightarrow \neg\mathtt{R}(\mathbf{y}, \mathtt{OP}(\mathbf{x}, \xi))$$

Man sieht, daß allein die Aufstellung der Frameaxiome nicht ganz mühelos ist.
Zur automatischen Generierung der Frameaxiome gibt es folgendes Verfahren [Ped89]:
Es ist eine Vollständigkeitsannahme notwendig:

2.1 Annahme: Die Prädikatsvorbedingung $\varepsilon^+_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi)$ spezifiziert all die Bedingungen, die gewährleisten, daß `R` im Nachfolgezustand eines Operators `OP` erfüllt ist, wenn `OP` ausgeführt wurde. Analog spezifiziert $\varepsilon^-_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi)$ all die Bedingungen, die gewährleisten, daß `R` im Nachfolgezustand eines Operators `OP` nicht erfüllt ist, wenn `OP` ausgeführt wurde. ■

Unter dieser Annahme gilt also:

$$\pi_{\mathtt{OP}}(\mathbf{x}, \xi) \wedge \mathtt{R}(\mathbf{y}, \xi) \wedge \neg\mathtt{R}(\mathbf{y}, \mathtt{OP}(\mathbf{x}, \xi)) \Rightarrow \varepsilon^-_{\mathtt{R}}(\mathbf{x}, \mathbf{y}, \xi)$$

Dies ist logisch äquivalent zu

$$\pi_{\mathtt{OP}}(\mathbf{x},\xi) \wedge \mathtt{R}(\mathbf{y},\xi) \wedge \neg\varepsilon^{-}_{\mathtt{R}}(\mathbf{x},\mathbf{y},\xi) \Rightarrow \mathtt{R}(\mathbf{y},\mathtt{OP}(\mathbf{x},\xi))$$

Analog erhält man:

$$\pi_{\mathtt{OP}}(\mathbf{x},\xi) \wedge \neg\mathtt{R}(\mathbf{y},\xi) \wedge \neg\varepsilon^{+}_{\mathtt{R}}(\mathbf{x},\mathbf{y},\xi) \Rightarrow \neg\mathtt{R}(\mathbf{y},\mathtt{OP}(\mathbf{x},\xi))$$

Diese beiden Formeln haben nun die syntaktische Gestalt von Frameaxiomen. Unter der Vollständigkeitsannahme, gibt es also eine systematische Methode, die Frameaxiome zu erzeugen. Es stellt sich also die Frage, wann kann diese Annahme verletzt sein. Grob gesprochen immer dann, wenn die Auswirkungen einer Aktion auf ein Prädikat nicht vollständig bekannt sind, zum Beispiel, wenn man ein Klötzchen unkontrolliert in eine Landschaft von Türmen aus Klötzchen fallen läßt, ist nicht klar, wie diese danach aussieht.

2.3 Beherrschbarkeit der Frameaxiome

Durch den Situationenkalkül von J. McCarthy und P. Hayes läßt sich zusammen mit den Frameaxiomen das Planungsproblem auf das Deduktionsproblem zurückführen. Es ist also kein eigenes Planungsprogramm erforderlich. Vielmehr kann man einen Standardbeweiser für Prädikatenlogik erster Stufe, zum Beispiel einen Resolutionsbeweiser, zum Planen einsetzen.
Der Hauptnachteil besteht darin, daß für die Vollständigkeit des Ansatzes die Frameaxiome notwendig werden und zwar bei n Operatoren und m beschreibenden Prädikaten $2 \cdot n \cdot m$ Axiome. Diese dienen zur Beschreibung dessen, was sich durch Anwendung von Regeln *nicht* ändert.
Um das Frameproblems zu überwinden, hat R. Reiter den folgenden Vorschlag gemacht [Rei91] der auf einem Ansatz von L.K. Schubert [Sch90] aufbaut:
Man beschreibt die Veränderungen in der Welt nicht mehr mit Hilfe von Frameaxiomen, sondern mit Hilfe von Axiomen, die einen Abschluß der Erklärungen für einen Zustand liefern.
Es wird nun ein Prädikat `Poss` eingeführt, das für $\mathtt{Poss}(\mathtt{op},\mathbf{x},\xi)$ die intendierte Semantik hat: im Zustand ξ ist es möglich den Operator `op` mit Argumenten $\mathbf{x}$ auszuführen[2].
Zu einem Prädikat `R` gibt es die folgenden Axiome:

positive Effekte für `R`: $\mathtt{Poss}(\mathtt{op},\mathbf{x},\xi) \wedge \gamma^{+}_{\mathtt{R}}(\mathtt{op},\mathbf{x},\mathbf{y},\xi) \Rightarrow \mathtt{R}(\mathbf{y},\mathtt{op}(\mathbf{x},\xi))$

negative Effekte für `R`: $\mathtt{Poss}(\mathtt{op},\mathbf{x},\xi) \wedge \gamma^{-}_{\mathtt{R}}(\mathtt{op},\mathbf{x},\mathbf{y},\xi) \Rightarrow \neg\mathtt{R}(\mathbf{y},\mathtt{op}(\mathbf{x},\xi))$

Aktionsvorbedingung für `OP`: $\pi_{\mathtt{OP}}(\mathbf{x},\xi) \Rightarrow \mathtt{Poss}(\mathtt{OP},\mathbf{x},\xi)$

[2] Mit `op` kleingeschrieben, bezeichnen wir eine allquantifizierte Variable. Durch unsere Schreibweise verlassen wir damit rein formal die Logik erster Stufe, aber de facto nicht, da man – was Reiter auch tut – statt $\mathtt{op}(\mathbf{x},\xi)$ einfach $\mathbf{DO}(\mathtt{op},\mathbf{x},\xi)$ schreiben kann.

2.2 ANNAHME: Durch die beiden Effektaxiome werden sämtliche Bedingungen charakterisiert, unter denen die Anwendung eines Operators op den Wahrheitswert eines Prädikates R von wahr zu falsch (beziehungsweise von falsch zu wahr) verändert. ■

Unter dieser Annahme kann man also dann die folgende Klasse von Axiomen annehmen:

Erklärungsabschlußaxiome:

$$\mathtt{Poss}(\mathtt{op}, \mathtt{x}, \xi) \wedge \mathtt{R}(\mathtt{x}, \xi) \wedge \neg\mathtt{R}(\mathtt{y}, \mathtt{op}(\mathtt{x}, \xi)) \Rightarrow \gamma_{\mathtt{R}}^{-}(\mathtt{op}, \mathtt{x}, \mathtt{y}, \xi)$$
$$\mathtt{Poss}(\mathtt{op}, \mathtt{x}, \xi) \wedge \neg\mathtt{R}(\mathtt{x}, \xi) \wedge \mathtt{R}(\mathtt{y}, \mathtt{op}(\mathtt{x}, \xi)) \Rightarrow \gamma_{\mathtt{R}}^{+}(\mathtt{op}, \mathtt{x}, \mathtt{y}, \xi)$$

Hinzu kommen noch technische Bedingungen, die die Eindeutigkeit von Namen garantieren.

Eindeutige Aktionsnamen:

Für verschiedene Aktionen op und op′ gilt: $\mathtt{op}(\mathtt{x}) \neq \mathtt{op}'(\mathtt{x})$

Identische Aktionen haben identische Argumente:

$$\mathtt{op}(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \mathtt{op}(\mathtt{y}_1, \ldots, \mathtt{y}_n) \Rightarrow \mathtt{x}_1 = \mathtt{y}_1 \wedge \ldots \wedge \mathtt{x}_n = \mathtt{y}_n$$

Eindeutige Zustandsnamen:

$$\mathtt{Init} \neq \mathtt{op}(\xi)$$
$$\mathtt{op}(\xi) = \mathtt{op}'(\xi') \Rightarrow \mathtt{op} = \mathtt{op}' \wedge \xi = \xi'$$

Nun sieht man nach etwas länglicher, aber rein aussagenlogischer Berechnung, daß sich die Konjunktion der beiden Effektaxiome für ein Prädikat R sowie der beiden Erklärungsabschlußaxiome für R zusammenfassen läßt zu dem folgenden Axiom:

Nachfolgezustandaxiom:

$$\mathtt{Poss}(\mathtt{op}, \mathtt{x}, \xi) \Rightarrow$$
$$\Big[\mathtt{R}(\mathtt{y}, \mathtt{op}(\mathtt{x}, \xi)) \Leftrightarrow \Big((\gamma_{\mathtt{R}}^{+}(\mathtt{op}, \mathtt{x}, \mathtt{y}, \xi) \vee \mathtt{R}(\mathtt{x}, \xi)\Big) \wedge \neg\gamma_{\mathtt{R}}^{-}(\mathtt{op}, \mathtt{x}, \mathtt{y}, \xi)\Big]$$

Sieht man von den verhältnismäßig unwichtigen Eindeutigkeitsaxiomen ab, so benötigt man also mit dieser Methode bei n Operatoren und m beschreibenden Prädikaten statt $2 \cdot n \cdot m$ lediglich noch $n + m$ Axiome, nämlich n Aktionsvorbedingungsaxiome und m Nachfolgezustandsaxiome.
Dieser Gewinn wird durch die Quantifizierung über Operatoren und über die Vollständigkeitsannahme erreicht.
Bei diesem Ansatz bleibt allerdings erst mal offen, ob sich die kompaktere Darstellung beim automatischen Beweisen auch entsprechend gutartig verhält, so daß für die Plansuche ein entsprechender Gewinn erzielt wird.

Kapitel 3

Alternative Ansätze des deduktiven Planens

War der Situationenkalkül historisch der im wesentlichen erste systematische Ansatz zum Planen, so sind in der Folge weitere Ansätze entwickelt worden, die teilweise deduktiv teilweise nicht deduktiv sind. Als Antwort auf den Situationenkalkül und das Frameproblem ist als nächster wesentlicher Beitrag das Planungssystem STRIPS [FN71] entstanden. Verwandter zum Situationenkalkül sind allerdings weitere Ansätze des *deduktiven* Planens, von denen wir in diesem Kapitel drei behandeln wollen.

3.1 Eigenschaftsvariablen

Eine Schwierigkeit des Situationenkalküls besteht in der Spezifizierung all dessen, was sich bei der Anwendung eines Operators *nicht* ändert. Üblicherweise muß man dazu Frameaxiome einführen. Im Prinzip will man sagen, daß sich alle Eigenschaften ungleich der explizit angegebenen nicht ändern. Dies ist im naiven Ansatz sehr aufwendig, da bei n Operatoren und m Prädikaten $2 \cdot n \cdot m$ Axiome notwendig sind. Im letzten Kapitel haben wir gesehen, wie R. Reiter dieses Problem gelöst hat, so daß nur $n + m$ Axiome nötig werden.
R. Kowalski [Kow79] schlägt folgendes Verfahren vor: Anstatt der im Situationenkalkül üblichen Schreibweise `On(D,A,Init)`, führt er ein neues Prädikat[1] ein, das besagt, daß `On(D,A)` im Zustand `Init` gültig ist, also `GILT(On(D,A),Init)`. Dieser ganze Ausdruck wird jetzt als eine Formel der Logik erster Stufe aufgefaßt, das heißt `GILT` ist ein Prädikatssymbol, `On(D,A)` und `Init` sind Terme, folglich muß `On` jetzt ein Funktionssymbol sein.

[1] Kowalski nennt dieses Prädikat `Holds`, was metasprachlich sagen soll, daß etwas gilt. Da wir `Holds` schon objektsprachlich dafür verwenden, daß die Roboterhand etwas hält, verwenden wir stattdessen das metasprachliche Prädikat `GILT`.

Der Vorteil dieser Art der Darstellung liegt darin, daß man nun über Beschreibungsmerkmale quantifizieren kann. Zum Beispiel ist es in dieser Sprache möglich zu sagen:
$\forall \mathtt{p}, \mathtt{x}, \xi \quad \mathtt{GILT}(\mathtt{p}, \xi) \wedge \mathtt{p} \neq \mathtt{Holds}(\mathtt{x}) \Rightarrow \mathtt{GILT}(\mathtt{p}, \mathtt{PUTDOWN}(\mathtt{x}, \xi))$
Der Operator PUTDOWN läßt sich wie folgt beschreiben:

$\mathtt{PUTDOWN}(\mathtt{x}, \xi)$
$\forall \mathtt{x}, \xi \quad \mathtt{GILT}(\mathtt{Holds}(\mathtt{x}), \xi)$ $\Rightarrow \quad \mathtt{GILT}(\mathtt{Clear}(\mathtt{x}), \mathtt{PUTDOWN}(\mathtt{x}, \xi)) \wedge$ $\quad \mathtt{GILT}(\mathtt{Ontable}(\mathtt{x}), \mathtt{PUTDOWN}(\mathtt{x}, \xi)) \wedge$ $\quad \mathtt{GILT}(\mathtt{Handempty}, (\mathtt{PUTDOWN}(\mathtt{x}, \xi))) \wedge$ $\quad \neg\mathtt{GILT}(\mathtt{Holds}(\mathtt{x}), \mathtt{PUTDOWN}(\mathtt{x}, \xi))$ $\wedge$ $\forall \mathtt{p}, \mathtt{x}, \xi \quad \mathtt{GILT}(\mathtt{p}, \xi) \wedge \mathtt{p} \neq \mathtt{Holds}(\mathtt{x}) \Rightarrow \mathtt{GILT}(\mathtt{p}, \mathtt{PUTDOWN}(\mathtt{x}, \xi))$

Der Vorteil dieser Methode besteht also darin, daß unabhängig von der Anzahl der verwendeten Prädikate, für jede Regel nur ein Frameaxiom erforderlich ist. Allerdings benötigt man nun zusätzlich zum Reiterschen Ansatz auch noch eine gute Gleichheitsbehandlung. Kowalski hat auch ausführlich die Umsetzung dieser Methode in der Suche nach Plänen diskutiert bis hin zu einer Implementierung in Prolog mit den Problemen von Negation as Failure. Wir werden diese Diskussion hier nicht verfolgen, sondern uns die entsprechenden Suchverfahren später bei STRIPS anschauen. Wen diese Diskussion näher interessiert, sei auf [Kow79, Chapter 6] verwiesen.

3.2 Dynamische Logik

Ein anderer Ansatz zur Formalisierung von Planung besteht in der Verwendung von dynamischer Logik. Für eine ausführliche Einführung in das Gebiet dynamische Logik vergleiche [Har84]. Im folgenden werden die wesentlichen Prinzipien dieser Logik eingeführt.
Zur formalen Beschreibung von zuweisungsorientierten Programmen, die ja bei der Abarbeitung wie Situationen in Planungsprozessen auch Zustände besitzen, hat man eine Logik entwickelt, die in gewisser Hinsicht Modallogiken verallgemeinert.
Modallogiken enthalten außer den in der Prädikatenlogik erster Stufe üblichen syntaktischen Konstrukten noch die beiden Modaloperatoren $\Diamond$ und $\Box$, die die intendierte Semantik haben: $\Diamond\varphi$ heißt, „Es ist *möglich*, daß φ gilt.“ und $\Box\varphi$ heißt, „Es ist *notwendig*, daß φ gilt.“[2]. Man hat zwischen den beiden Modaloperatoren die Beziehung $\Diamond = \neg\Box\neg$.
In dynamischer Logik werden die Modaloperatoren dahingehend erweitert, daß sie bezüglich eines Programms beziehungsweise bezüglich eines Programmfragments interpretiert werden. Sei also α ein Programm, so hat $[\alpha]\varphi$ die intendierte Semantik:

[2] Praktisch im ganzen Band [GG84] geht es über Modallogiken. Wer sich näher dafür interessiert, wird dort jede Menge Stoff finden.

„Nach Ausführung des Programms α gilt für alle Endzustände die Formel φ“. Entsprechend bedeutet $\langle\alpha\rangle\varphi$ „Nach Ausführung des Programms α gilt für einen Endzustand die Formel φ“. Für deterministische Programme ist die zweite Aussage stärker, denn sie beinhaltet die Terminierung des Programms.
Sehen wir uns ein kleines Beipiel an. Sei α das folgende Programm, das mit $x = n$ die n-te Fibonaccizahl F_n berechnet:

$(y, z) \leftarrow (1, 1)$
while $x \neq 0$ **do** $(x, y, z) \leftarrow (x - 1, z, y + z)$

Eine typische Aussage in dynamischer Logik könnte dann sein:
$(n \geq 0 \wedge x = n) \Rightarrow [\alpha](y = F_n)$
Dies bedeutet: Wenn n größer oder gleich 0 ist und x gleich n, dann ist y nach der Ausführung von α gleich der n-ten Fibonaccizahl (falls α terminiert).
In dynamischer Logik kommen also zwei Sprachen zusammen, eine Programmiersprache, in der Programmfragmente α formuliert werden, die innerhalb der Operatoren $[\alpha]$ und $\langle\alpha\rangle$ vorkommen und Prädikatenlogik erster Stufe, in der Aussagen über die Programme gemacht werden können. Diese Sprache ist insbesondere für zuweisungsorientierte Programmiersprachen geeignet, die ja durch die Zuweisung nicht-monoton sind. Für die dynamische Logik wurden Kalküle und zugehörige Beweiser entwickelt, in denen man formal Programme verifizieren kann.
Für die Beschreibung von Planungsvorgängen benutzt man nun die dynamische Logik, in dem man Pläne als Programme auffaßt. Dies ist von S.J. Rosenschein in [Ros81] für aussagenlogische dynamische Logik eingeführt worden und für das KIV System [HRS90] in [SB93] übertragen worden.
Bei der Darstellung beschränken wir uns hier auf den aussagenlogischen Teil. Ein Kalkül für die Logik sieht dann wie folgt aus. Die Axiome sind:

- Die Axiome der Aussagenlogik sind Axiome.
- $[\alpha](p \Rightarrow q) \Rightarrow ([\alpha]p \Rightarrow [\alpha]q)$
- $[\Lambda]p \Leftrightarrow p$
- $[p?]q \Leftrightarrow (p \Rightarrow q)$ für p nicht-modal
- $[\alpha; \beta]p \Leftrightarrow [\alpha][\beta]p$
- $[\alpha \cup \beta]p \Leftrightarrow [\alpha]p \wedge [\beta]p$

Die Inferenzregeln sind

- Aus p und $p \Rightarrow q$ folgt q (Modus Ponens)
- Aus p folgt $[\alpha]p$ (Notwendigkeitsregel)

Dabei bedeutet α ein beliebiger Plan, Λ der leere Plan, $p?$ ist ein Test, ob p erfüllt ist, mit $\alpha;\beta$ wird die Hintereinanderausführung von α und β bezeichnet und mit $\alpha \cup \beta$ die nicht-deterministische Auswahl.
Ein Beispiel sei durch folgende Situation gegeben:

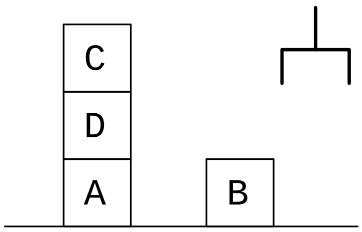
$\texttt{An(licht)} \Rightarrow [\texttt{schalte(schalter)}]\neg\texttt{An(licht)}$

$\neg\texttt{An(licht)} \Rightarrow [\texttt{schalte(schalter)}]\ \texttt{An(licht)}$

Das heißt durch diese Formeln wird beschrieben, daß ein Schaltvorgang das Licht von An auf $\neg$An und umgekehrt schaltet.
Wie funktioniert nun das Planen. Man kann eine Menge von elementaren Aktionen ausmachen, in denen die anderen beschrieben werden. Ohne dies im Detail auszuführen, sehen wir uns dazu ein Beispiel an, nämlich die Darstellung des UNSTACK Operators.

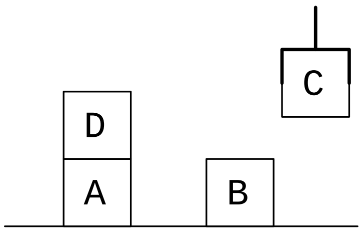
UNSTACK(x, y)	
IF	Clear(x) $\wedge$
	On(x, y)$\wedge$
	Handempty
THEN	**ADD**(Holds(x)) $\wedge$
	ADD(Clear(y)) $\wedge$
	DELETE(Clear(x)) $\wedge$
	DELETE(On(x, y)) $\wedge$
	DELETE(Handempty)
ELSE	**ABORT**
FI	

Hat man also wieder die Situation von Abbildung 1.2:



Ontable(A)
Ontable(B)
On(D,A)
On(C,D)
Clear(C)
Clear(B)
Handempty

Dann hat man nach Anwendung von UNSTACK(C, D) den Zustand



Ontable(A)
Ontable(B)
On(D,A)
Clear(D)
Clear(B)
Holds(C)

Formal gilt:

$$\left\{\begin{array}{l}\texttt{Ontable(A)} \wedge \texttt{Ontable(B)} \wedge \\ \texttt{On(D, A)} \wedge \texttt{On(C, D)} \wedge \\ \texttt{Clear(C)} \wedge \texttt{Clear(B)} \wedge \\ \texttt{Handempty}\end{array}\right\} \Rightarrow [\texttt{UNSTACK(C, D)}] \left\{\begin{array}{l}\texttt{Ontable(A)} \wedge \texttt{Ontable(B)} \wedge \\ \texttt{On(D, A)} \wedge \texttt{Clear(D)} \wedge \\ \texttt{Clear(B)} \wedge \texttt{Holds(C)}\end{array}\right\}$$

Dabei steht das Planfragment [UNSTACK(C, D)] für den Ausdruck:

$$\begin{array}{ll}\textbf{IF} & \texttt{Clear(C)} \wedge \texttt{On(C, D)} \wedge \texttt{Handempty} \\ \textbf{THEN} & \textbf{ADD}(\texttt{Holds(C)}) \wedge \textbf{ADD}(\texttt{Clear(D)}) \wedge \\ & \textbf{DELETE}(\texttt{Clear(C)}) \wedge \textbf{DELETE}(\texttt{On(C, D)}) \wedge \textbf{DELETE}(\texttt{Handempty}) \\ \textbf{ELSE} & \textbf{ABORT} \\ \textbf{FI} & \end{array}$$

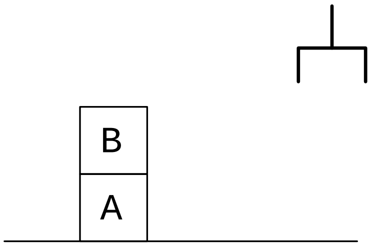
Dieserum evaluiert auf Grund dessen, daß die Vorbedingung im **IF**-Teil im entsprechenden Vorgängerzustand erfüllt ist, zum **THEN**-Zweig des Ausdrucks, das heißt zu

$$\begin{array}{l}\textbf{ADD}(\texttt{Holds(C)}) \wedge \textbf{ADD}(\texttt{Clear(D)}) \wedge \\ \textbf{DELETE}(\texttt{Clear(C)}) \wedge \textbf{DELETE}(\texttt{On(C, D)}) \wedge \textbf{DELETE}(\texttt{Handempty})\end{array}$$

Die Semantik und die zugehörige Kalkülisierung der beiden Operatoren **ADD** und **DELETE** muß nun so gewählt werden, daß diese genau das tun, was sie sollen, das heißt in diesem Fall ausgehend von der linken Zustandsbeschreibung das Faktum `Clear(D)` hinzufügen und die drei Fakten `Clear(C)`, `On(C,D)` und `Handempty` löschen.
Der große Vorteil dieser Methode besteht darin, daß nun keine Frameaxiome erforderlich sind. Man erkauft dies natürlich mit der schwierigeren dynamischen Logik, in der die Suche nach einem Plan erfolgen muß.

3.3 Linearität

Der letzte deduktive Planungsansatz, den wir hier vorstellen, läßt sich auf der einen Seite durch eine besondere Eigenschaft in der Suche nach logischen Beweisen darstellen, nämlich der Linearität. Auf der anderen Seite ist verwandt dazu eine eigene Logik, die sogenannte lineare Logik entwickelt worden. Obwohl 15 Jahre vorher von R. Meyer und M. McRobby entwickelt, ist lineare Logik erst 1987 von J.-Y. Girard wiederentdeckt, in ihrer Bedeutung erkannt und populär gemacht worden. Grob gesprochen wird im Gegensatz zur klassischen Logik in linearer Logik Buch darüber geführt, *wie oft* ein Faktum verwendet wird. Wir wollen hier darauf nicht weiter eingehen, sondern verweisen auf den ausführlichen Artikel dazu [Gir87]. Stattdessen werden wir in diesem Abschnitt näher auf den Ansatz von W. Bibel et al. der Planerzeugung durch lineare Beweise eingehen [Bib86, Fro87, BCFH89].
Betrachten wir dazu als Beispiel wieder den Anfangszustand aus Abbildung 2.1:

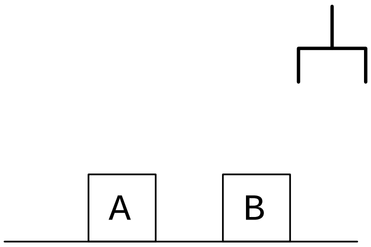


```
Ontable(A)
On(B,A)
Clear(B)
Handempty
```

Der Anfangszustand wird nun beschrieben durch die Formel:

$$\mathtt{SIT(Init)} \wedge \mathtt{Ontable(A)} \wedge \mathtt{On(B,A)} \wedge \mathtt{Clear(B)} \wedge \mathtt{Handempty} \qquad \mathtt{(Init)}$$

Die Zielsituation sieht wie folgt aus:



```
Ontable(A)
Ontable(B)
Clear(A)
Clear(B)
Handempty
```

Sie kann also durch die folgende Formel beschrieben werden:

$$\exists\zeta \ \mathtt{SIT}(\zeta) \wedge \mathtt{Ontable(A)} \wedge \mathtt{Ontable(B)} \wedge \mathtt{Clear(A)} \wedge \mathtt{Clear(B)} \wedge \mathtt{Handempty} \qquad \mathtt{(Goal)}$$

Die beiden Aktionen `UNSTACK` und `PUTDOWN` werden durch die beiden folgenden Regeln spezifiziert:

$\forall \mathtt{x}, \mathtt{y}, \xi \ \exists\zeta$ $\Rightarrow$	$\mathtt{SIT}(\xi) \wedge \mathtt{Clear(x)} \wedge \mathtt{On(x,y)} \wedge \mathtt{Handempty}$ $\mathtt{SIT}(\zeta) \wedge \mathtt{Holds(x)} \wedge \mathtt{Clear(y)}$	$\mathtt{R}_1$
$\forall \mathtt{x}, \xi \ \exists\zeta$ $\Rightarrow$	$\mathtt{SIT}(\xi) \wedge \mathtt{Holds(x)}$ $\mathtt{SIT}(\zeta) \wedge \mathtt{Ontable(x)} \wedge \mathtt{Clear(x)} \wedge \mathtt{Handempty}$	$\mathtt{R}_2$

$\mathtt{R}_1$ bedeutet: „Zu jeder Situation ξ, in der ein Klötzchen `x` auf einem Klötzchen `y` steht, in der auf `x` nichts mehr steht und in der die Roboterhand frei ist, gibt es eine Situation ζ, in der die Roboterhand `x` hält und auf `y` nichts mehr steht.“ Entsprechend liest sich die zweite Regel $\mathtt{R}_2$: „Zu jeder Situation ξ, in der die Roboterhand ein Klötzchen `x` hält, gibt es eine Situation ζ, in der `x` auf dem Tisch steht und die Roboterhand frei ist.“

Allerdings dürfen diese Regeln nicht rein logisch betrachtet werden, da man sonst unzulässige Folgerungen durchführen könnte. Wir hätten ja dann die übliche Monotonie der klassischen Logik erster Stufe, also zum Beispiel nach der Operation `UNSTACK(B,A)` hätten wir immer noch `On(A,B)`. Deshalb muß für die möglichen Herleitungsschritte im logischen Kalkül eine Einschränkung gefunden werden, die die wiederholte Verwendung ein und desselben Faktums ausschließt. Solchermaßen eingeschränkte Beweise heißen linear.

3.1 DEFINITION: Ein *linearer Beweis* ist ein Beweis, in dem jede Instanz eines Atoms maximal einmal benutzt werden darf. ∎

Die Planungsaufgabe besteht nun darin einen linearen Beweis für die Formel

$$\mathtt{Init} \wedge \mathrm{R}_1 \wedge \mathrm{R}_2 \Rightarrow \mathtt{Goal}$$

zu finden. Leider steckt nun da der Teufel im Detail: In einem Kalkül kann es sein, daß eine Aussage nur einmal benötigt wird, in einem anderen kann es zur Ableitung derselben Fakten – zum Beispiel auf Grund einer Normalisierung – erforderlich werden, diese Aussage mehrmals zu verwenden. In [BCFH89] ist eine entsprechende Ausarbeitung für die Konnektionsmethode durchgeführt worden.
Betrachten wir das Beispiel informell:
Zuerst instantiieren wir die Regel $\mathtt{R}_1$ mit $\mathtt{x} \leftarrow \mathtt{B}$, $\mathtt{y} \leftarrow \mathtt{A}$ und $\xi \leftarrow \mathtt{Init}$. Die Regel darf nun auf Grund der Linearitätseigenschaft mit dieser Instantiierung für diesen Planungsprozeß nicht wieder angewendet werden. Aus naheliegenden Gründen nennen wir die auftretende Skolemkonstante für ζ $\mathtt{UNSTACK(B, A, Init)}$. Damit können wir also die Formel $\mathtt{SIT(UNSTACK(B, A, Init))} \wedge \mathtt{Holds(B)} \wedge \mathtt{Clear(A)}$ und somit in einem nächsten Schritt die entsprechenden drei Einzelformeln $\mathtt{SIT(UNSTACK(B, A, Init))}$, $\mathtt{Holds(B)}$ und $\mathtt{Clear(A)}$ ableiten. Aktiv sind nun außer diesen drei Formeln noch die Eingangsformel $\mathtt{Ontable(A)}$, die Regel $\mathtt{R}_1$, wobei eine Instantiierung verboten ist und die Regel $\mathtt{R}_2$.
Nun instantiieren wir die Regel $\mathtt{R}_2$ mit $\mathtt{x} \leftarrow \mathtt{B}$ und $\xi \leftarrow \mathtt{UNSTACK(B, A, Init)}$. Die Regel darf auf Grund der Linearitätseigenschaft mit dieser Instantiierung nicht wieder angewendet werden. Die Skolemkonstante für ζ wird mit $\mathtt{PUTDOWN(B, UNSTACK(B, A, Init))}$ bezeichnet. Da die Vorbedingungen erfüllt sind, das heißt aktive Fakten sind, ergibt sich aus dem Sukzedenz von $\mathtt{R}_2$ die Formel $\mathtt{SIT(PUTDOWN(B, UNSTACK(B, A, Init)))} \wedge \mathtt{Ontable(B)} \wedge \mathtt{Clear(B)} \wedge \mathtt{Handempty}$, also wiederum die entsprechenden vier Einzelfakten. Zu diesem Zeitpunkt sind aktiv: $\mathtt{SIT(PUTDOWN(B, UNSTACK(B, A, Init)))}$, $\mathtt{Ontable(B)}$, $\mathtt{Clear(B)}$, $\mathtt{Handempty}$ aus der letzten Regel, $\mathtt{Ontable(A)}$ vom Anfangszustand (mit dieser Aussage ist ja noch kein Deduktionsschritt durchgeführt worden) und $\mathtt{Clear(A)}$ aus dem Sukzedenz von $\mathtt{R}_1$ (auch mit dieser Aussage ist ja kein weiterer Deduktionsschritt durchgeführt worden). Dies sind aber genau die Aussagen, die für das Ziel abzuleiten waren, also ist der gewünschte Beweis erbracht.
Durch die implizite Behandlung von aktiven Formeln werden die Frameaxiome überflüssig. Schwierigkeiten ergeben sich aus der Einschränkung der für die Methode geeigneten Grundkalküle. Für diese Art von Beweisen, in denen nur eingeschränkte Ableitungen bezüglich der klassischen Ableitungsmöglichkeiten in einem fixen Kalkül zugelassen sind, stellt sich die Frage nach der Semantik. In [BCFH89] werden zwei verschiedene Semantiken angegeben, zum einen werden lineare Beweise in den Situationenkalkül eingebettet, zum anderen wird eine modallogische Semantik angegeben, die der obigen Darstellung für dynamische Logik entspricht.

Kapitel 4

Das Planungssystem STRIPS

1971 haben Richard E. Fikes und Nils J. Nilsson [FN71] das Planungssystem STRIPS entwickelt, das in gewisser Hinsicht wesentlich für die weitere Entwicklung im Planen war. STRIPS steht für *ST*anford *R*esearch *I*nstitute *P*roblem *S*olver. In STRIPS wird die Welt wiederum als eine Menge von Formel der Logik erster Stufe beschrieben. Im Gegensatz zum deduktiven Ansatz wird in STRIPS allerdings die Suche nicht innerhalb der Logik durchgeführt, so daß das Frameproblem in STRIPS nicht mehr auftritt.

4.1 Ein Planungsproblem in STRIPS

Sehen wir uns wieder die Situation aus Abbildung 1.2 als Ausgangszustand und 1.3 als Zielzustand an:

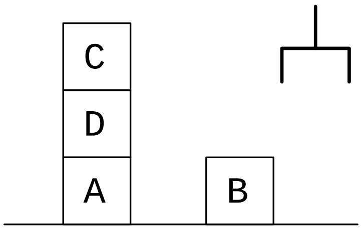


Abbildung 4.1: Ausgangszustand

```
Ontable(A)
Ontable(B)
On(D,A)
On(C,D)
Clear(C)
Clear(B)
Handempty
```

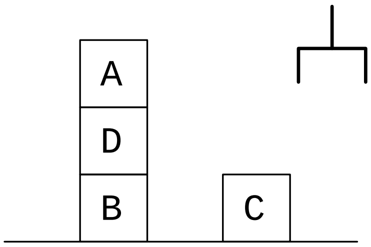


Abbildung 4.2: Zielzustand

```
Ontable(B)
Ontable(C)
On(D,B)
On(A,D)
Clear(A)
Clear(C)
Handempty
```

Die Operatoren werden nun in STRIPS – so ähnlich wie wir es im Fall der dynamischen Logik gesehen haben (historisch war STRIPS allerdings zuerst da, und in dynamischer Logik hat man das Verhalten von STRIPS nachgebaut) – durch drei Klassen von Formeln beschrieben, die Vorbedingungen **Precondition**, die Liste der zu löschenden Formeln **delete list** und die Liste der bei der Anwendung des Operators neu hinzuzufügenden Formeln **add list**. Für unsere Klötzchenwelt sieht dann eine STRIPS-Beschreibung der vier Operatoren wie folgt aus:

PICKUP(x)		PUTDOWN(x)	
Precondition	Clear(x) $\wedge$	**Precondition**	Holds(x)
	Ontable(x) $\wedge$	**delete list**	Holds(x)
	Handempty	**add list**	Clear(x) $\wedge$
delete list	Clear(x) $\wedge$		Ontable(x) $\wedge$
	Ontable(x) $\wedge$		Handempty
	Handempty		
add list	Holds(x)		
STACK(x, y)		UNSTACK(x, y)	
Precondition	Holds(x) $\wedge$	**Precondition**	Clear(x) $\wedge$
	Clear(y) $\wedge$		On(x, y) $\wedge$
delete list	Holds(x) $\wedge$		Handempty
	Clear(y)	**delete list**	Clear(x) $\wedge$
add list	Clear(x) $\wedge$		On(x, y) $\wedge$
	On(x, y) $\wedge$		Handempty
	Handempty	**add list**	Holds(x) $\wedge$
			Clear(y)

Abbildung 4.3: STRIPS-Operatoren zur Klötzchenwelt

Ein Planungsproblem in STRIPS besteht nun aus:

- einem Anfangszustand (gegeben durch eine Menge von prädikatenlogischen Formeln)
- einem Zielzustand (gegeben durch eine Menge von prädikatenlogischen Formeln)
- einer Menge von Regeln (beschrieben durch Name, **Precondition**, **delete list** und **add list**)

Sehen wir uns nun an, wie eine Lösung des obigen Planungsproblems in STRIPS aussieht. Durchgestrichene Formeln zeigen an, welche im entsprechenden Schritt gelöscht wurden, unterstrichene, welche neu hinzugekommen sind.

Ontable(A) Ontable(B) On(D,A) On(C,D) Clear(C) Clear(B) Handempty	UNSTACK(C,D) ⟶	Ontable(A) Ontable(B) On(D,A) ~~On(C,D)~~ ~~Clear(C)~~ Clear(B) ~~Handempty~~ <u>Holds(C)</u> <u>Clear(D)</u>	PUTDOWN(C) ⟶	Ontable(A) Ontable(B) On(D,A) Clear(B) ~~Holds(C)~~ Clear(D) <u>Clear(C)</u> <u>Ontable(C)</u> <u>Handempty</u>	UNSTACK(D,A) ⟶

Ontable(A) Ontable(B) ~~On(D,A)~~ Clear(B) ~~Clear(D)~~ Clear(C) Ontable(C) ~~Handempty~~ <u>Holds(D)</u> <u>Clear(A)</u>	STACK(D,B) ⟶	Ontable(A) Ontable(B) ~~Clear(B)~~ Clear(C) Ontable(C) ~~Holds(D)~~ Clear(A) <u>Clear(D)</u> <u>On(D,B)</u> <u>Handempty</u>	PICKUP(A) ⟶	~~Ontable(A)~~ Ontable(B) Clear(C) Ontable(C) ~~Clear(A)~~ Clear(D) On(D,B) ~~Handempty~~ <u>Holds(A)</u>	STACK(A,D) ⟶	Ontable(B) Clear(C) Ontable(C) ~~Clear(D)~~ On(D,B) ~~Holds(A)~~ <u>Clear(A)</u> <u>On(A,D)</u> <u>Handempty</u>

Das eigentliche Vorgehen in STRIPS ist durch die Möglichkeit beliebige prädikatenlogische Formeln für die Problembeschreibungen zu verwenden noch komplizierter; denn es ist ja nun keineswegs trivial festzustellen, ob der Zielzustand erreicht ist. Zu diesem Zweck wird in STRIPS ein automatischer Beweiser integriert. Der Ansatz funktioniert dann so:

– der automatische Beweiser guckt nach, ob der Zielzustand aus dem Anfangszustand beweisbar ist. Wenn ja: fertig, wenn nein wird heuristisch ein Operator angewandt.

– Nach jeder Operatoranwendung wird der automatische Beweiser angeworfen, um zu überprüfen, ob der Zielzustand erreicht ist.

Durch die implizite Beschreibung der Zustände benötigt man also in STRIPS keine Frameaxiome. Vielmehr wird durch die Ausführung der **add list** und **delete list** jeweils der aktuelle Zustand erzeugt. Das Problem der Planung selbst reduziert sich auf die *Suche* nach einem Plan. Was Richard E. Fikes und Nils J. Nilsson sich dazu überlegt haben, sehen wir im nächsten Abschnitt.

4.2 Suche in STRIPS

Ein einfaches Planungssystem könnte auf den Anfangszustand alle anwendbaren Operatoren der Reihe nach und mit allen möglichen Instantiierungen anwenden, dadurch erhielte man eine Menge von Zwischenzuständen, auf die man dann im Stil der Breitensuche wieder alle anwendbaren Operatoren anwendet. Man wiederholt diese Prozedur solange bis man den Zielzustand erreicht hat, genauer gesagt bis man an einem

Zustand angekommen ist, aus dem der Zielzustand ableitbar ist. Mit dieser Methode würde man allerdings in der Regel riesige Bäume, an deren Kanten die jeweils angewandten Operatoren und in den Knoten Problemzuständen stehen, erzeugen, so daß diese naive Planungsmethode meist nicht praktikabel ist.
Deshalb hat man Verfahren entwickelt, die heuristisch vorgehen. Nun ist es im allgemeinen egal, ob man Anfangszustand `Init` ausgehend einen Pfad durch den Suchbaum zum Zielzustand `Goal` sucht oder umgekehrt. In STRIPS hat man sich dennoch für die Rückwärtssuche vom Ziel zum Anfang entschieden, da das Ziel für praktische Anwendungen oft relativ einfach spezifiziert ist, während der Ausgangszustand in der Regel recht kompliziert sein kann und ausführlich beschrieben ist. Zum Beispiel kann eine konkrete Ausgangssituation mit 26 Klötzchen in der Klötzchenwelt gegeben sein, während die Zielsituation nicht vollständig beschrieben ist, sondern nur ein Fakt wie `On(A,B)` enthält. In diesem Fall ist es natürlich sinnvoll von der Zielsituation auszugehen und zu sehen, wie man diese erreichen kann. Das heißt man durchsucht den Suchraum rückwärts. Deshalb muß man natürlich die einzelnen Operatoren auch rückwärts anwenden. Um in einer bestimmten Situation weiterzusuchen, wird also ein Knoten *expandiert*, das heißt es werden alle Situationen erzeugt, aus denen man in die angestrebte Situation durch eine einzige Operatoranwendung gelangen kann. Will man zum Beispiel erreichen, daß `Holds(A)` gilt, so gibt es zwei Möglichkeiten, zum einen durch Anwendung von `PICKUP(A)` in einer Situation in der `Clear(A)`, `Ontable(A)` und `Handempty` gilt, zum anderen durch Anwendung von `UNSTACK(A,x)` in einer Situation in der `Clear(A)`, `On(A,x)` und `Handempty` gilt.
Ein solches Vorgehen heißt *means ends analysis* (Mittel-Ziel-Analyse) und wurde in STRIPS von GPS [EN69] übernommen. Wir stellen einen entsprechenden Algorithmus vor. Die Darstellung ist aus [Her89] übernommen.
Der Algorithmus erhält als Eingabe eine Menge von Zielen `Goal`, einen Anfangszustand `Init` und eine Menge von Regeln `RULES`. Als Ausgabe liefert er einen Plan `P`, das heißt eine lineare Ordnung von Operatoren. Er verwendet die Hilfsvariable `Actual` für die aktuelle Situation.

```
begin
0.    P:=NIL;Actual:=Init
1.    while noch Ziele aus Goal offen in Actual
      do
1.1.    wähle Ziel Goal_1 aus, das in Actual offen ist
        ;;; da beliebige Auswahl → Rücksetzpunkt
1.2.    wähle Instanz I eines Operators aus, die Goal_1, in der add list hat
        ;;; da beliebige Auswahl → Rücksetzpunkt
1.3    P:= append(P,MEA(Precondition(I),Actual),list(I))
1.4    Actual := P(Init)
      end_do
2.    return(P)
end
```

Abbildung 4.4: Algorithmus MEA (means-end-analysis)

Man beachte den rekursiven Aufruf des MEA-Algorithmus in der Schleife.
Im Falle unseres Eingangsbeispiels aus der Blockswelt sieht das wie folgt aus. Der Ausgangszustand und Zielzustand sind gegeben durch:

Ontable(A)	Ontable(B)
Ontable(B)	Ontable(C)
On(D,A)	On(D,B)
On(C,D)	On(A,D)
Clear(C)	Clear(A)
Clear(B)	Clear(C)
Handempty	Handempty
Actual = Init	Goal

Im aktuellen Zustand offen sind also Ontable(C), On(D,B), On(A,D) und Clear(A). Wählen wir von diesen als Ziel Goal_1 das erste, also Ontable(C) aus. Wie können wir dieses Ziel durch eine Operatoranwendung erreichen: Nur durch Anwendung des Operators PUTDOWN(C). Das heißt unser Plan setzt sich nun zusammen aus (MEA(**Precondition**(PUTDOWN(C))), PUTDOWN(C)). Wir brauchen jetzt also einen Plan zur Erzeugung der Vorbedingungen von PUTDOWN(C), also von Holds(C), dazu gibt es zwei Möglichkeiten, nämlich über ein PICKUP(C) und über UNSTACK(C,x). Wählen wir nun PICKUP(C) aus, so müßten wir, dafür als Voraussetzung unter anderem ein Ontable(C) erfüllen, was aber unser Anfangsproblem war, erkennt man dies, kann man diese Wahl ausschließen. Es bleibt also UNSTACK(C,x). Dies ist im Anfangszustand anwendbar mit x $\leftarrow$ D. Also hat P nun die Gestalt (UNSTACK(C, D), PUTDOWN(C)). Der aktuelle Zustand Actual wird nun auf den neuesten Stand gebracht und sieht wie folgt aus:

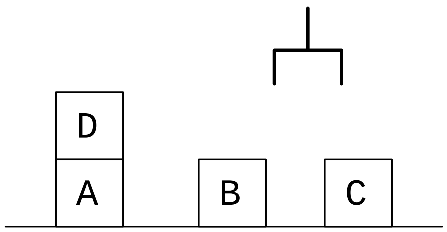
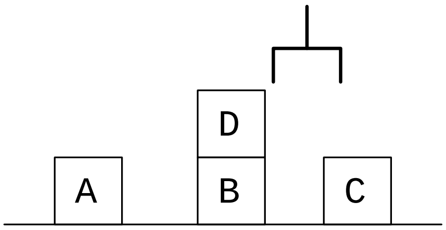


Abbildung 4.5: Erster Zwischenzustand

Ontable(A)
Ontable(B)
Ontable(C)
On(D,A)
Clear(B)
Clear(C)
Clear(D)
Handempty

Nun sind wir im ersten Durchgang durch die Schleife gelaufen. Am Zielzustand sind offen: On(D,B), On(A,D) und Clear(A). Wählen wir wieder das erste der offenen Teilziele aus, so ergibt sich als einziger Operator, der dieses Ziel erfüllt die Anwendung von STACK(D,B). Bei der Schaffung der Voraussetzungen für die Anwendung des Operators finden wir UNSTACK(D,A). Also haben wir nach dem zweiten Durchlauf der Schleife den Teilplan: (UNSTACK(C, D), PUTDOWN(C), UNSTACK(D, A), STACK(D, B)).



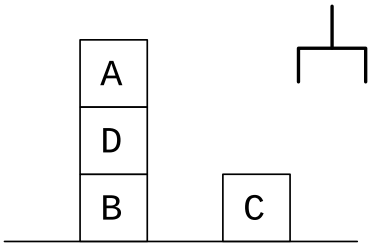

```
Ontable(A)
Ontable(B)
On(D,B)
Clear(A)
Clear(C)
Clear(D)
Handempty
```

Abbildung 4.6: Zweiter Zwischenzustand

Für den Zielzustand sind offen: `On(A,D)`. Die Auswahl der offenen Teilziele ist bei einem Teilziel einfach, ebenso die der Operatoren, wie man zu diesem Teilziel kommen kann: nur `STACK(A,D)` führt zu diesem Teilziel. Bei der Schaffung der Voraussetzungen für die Anwendung des Operators finden wir `PICKUP(A)`. Also haben wir nach dem dritten Durchlauf der Schleife den Plan:

(UNSTACK(C, D), PUTDOWN(C), UNSTACK(D, A), STACK(D, B), PICKUP(A), STACK(A, D))

Nun sind keine Ziele mehr offen, also ist dies der Plan zur Lösung des gesamten Problems.




```
Ontable(B)
Ontable(C)
On(D,B)
On(A,D)
Clear(A)
Clear(C)
Handempty
```

Abbildung 4.7: Dritter Zwischenzustand = Zielzustand

4.1 BEMERKUNG: In Schritt 1.1 des Algorithmus in Abbildung 4.4 zur Erzeugung eines Planes wird ein beliebiges Teilziel ausgewählt und nun der ganze Planungsablauf auf die Erfüllung eben dieses Teilziels abgestimmt. Dieses Vorgehen macht aber Sinn nur unter der einschränkenden Annahme, daß sich das Vorgehen beim Planen *linearisieren* läßt. Entsprechend heißt die Annahme *Linearitätsannahme*. Sie besagt, daß man zur Erfüllung mehrerer Ziele, immer eine Reihenfolge der Ziele finden kann, eine Linearisierung, in der man diese *nacheinander* erfüllen kann. ■

Leider gibt es Fälle, in denen die Linearitätsannahme nicht gegeben ist. Das heißt STRIPS ist in dieser Form nicht nur nicht praktisch, sondern auch nicht theoretisch universell einsetzbar.

4.3 Grenzen von STRIPS — Probleme der STRIPS-Strategie

Betrachten wir zuerst noch einmal kurz, wie sich in STRIPS das Frameproblem darstellt. Durch die implizite Darstellung der Zustände und eine Suche im Raum aller möglichen Zustände tritt das Frameproblem nicht mehr auf. STRIPS geht von einer Hauptannahme aus, die die Lösung des Frameproblems ermöglicht.

> Ein Operator ändert, bei seiner Ausführung genau das, was durch die Listen **add list** und **delete list** spezifiziert ist.

Die Einschränkung, die Wirkungen eines Operators genau im voraus wissen zu müssen, ist natürlich für sehr viele praktische Anwendungen nicht gegeben. In eingeschränkten klar überschaubaren Fällen macht sie allerdings Sinn. Deshalb werden wir diese Annahme noch eine Weile aufrechterhalten und uns anschauen, welche zusätzlichen Schwierigkeiten bei STRIPS auftreten. Dies sind also die Schwierigkeiten, die sich aus der MEA-Strategie (means ends analysis) und insbesondere aus der Linearitätsannahme ergeben.

4.3.1 Teilzielinteraktionen

Schwierigkeiten, die sich mit der Linearitätsannahme ergeben, kann man am folgenden Beispiel erkennen, das als Sussman-Anomalie bezeichnet wird.

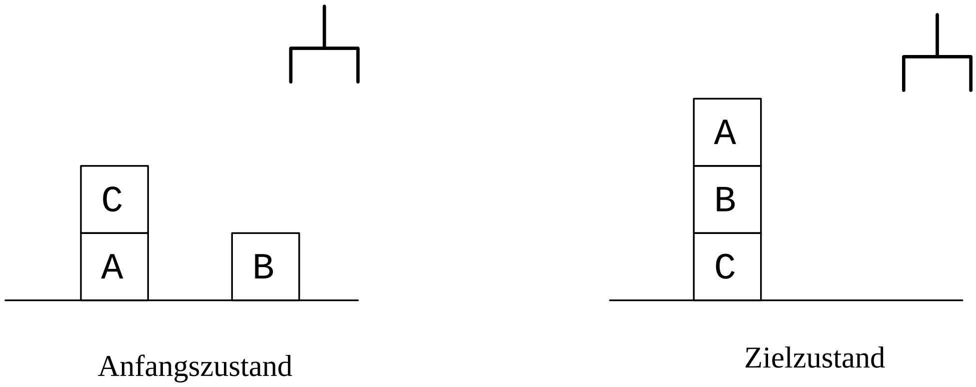


Abbildung 4.8: Die Sussman-Anomalie

Sehen wir uns nun eine Lösung des Problems in STRIPS mit MEA an. Man beachte dabei, daß der Algorithmus nicht vollständig spezifiziert ist, also daß mehrere Wahlmöglichkeiten bleiben. Die Auswahl im einzelnen orientiert sich an den Teilzielen `On(A,B)` und dann `On(B,C)`. Ansonsten ist sie jeweils optimal.

Ontable(A) Ontable(B) On(C,A) Clear(B) Clear(C) Handempty	$\xrightarrow{\text{UNSTACK(C,A)}}$	Ontable(A) Ontable(B) Clear(B) ~~On(C,A)~~ ~~Clear(C)~~ ~~Handempty~~ Holds(C) Clear(A)	$\xrightarrow{\text{PUTDOWN(C)}}$	Ontable(A) Ontable(B) Clear(A) Clear(B) ~~Holds(C)~~ Ontable(C) Clear(C) Handempty	$\xrightarrow{\text{PICKUP(A)}}$

Ontable(B) Ontable(C) Clear(B) Clear(C) ~~Ontable(A)~~ ~~Clear(A)~~ ~~Handempty~~ Holds(A)	$\xrightarrow{\text{STACK(A,B)}}$	Ontable(B) Ontable(C) Clear(C) ~~Clear(B)~~ ~~Holds(A)~~ On(A,B) Clear(A) Handempty	$\xrightarrow{\text{UNSTACK(A,B)}}$	Ontable(B) Ontable(C) Clear(C) ~~Clear(A)~~ ~~On(A,B)~~ ~~Handempty~~ Holds(A) Clear(B)	$\xrightarrow{\text{PUTDOWN(A)}}$

Ontable(B) Ontable(C) Clear(B) Clear(C) ~~Holds(A)~~ Ontable(A) Clear(A) Handempty	$\xrightarrow{\text{PICKUP(B)}}$	Ontable(A) Ontable(C) Clear(A) Clear(C) ~~Ontable(B)~~ ~~Clear(B)~~ ~~Handempty~~ Holds(B)	$\xrightarrow{\text{STACK(B,C)}}$	Ontable(A) Ontable(C) Clear(A) ~~Clear(C)~~ ~~Holds(B)~~ On(B,C) Clear(B) Handempty	$\xrightarrow{\text{PICKUP(A)}}$

Ontable(C) Clear(B) On(B,C) ~~Ontable(A)~~ ~~Clear(A)~~ ~~Handempty~~ Holds(A)	$\xrightarrow{\text{STACK(A,B)}}$	Ontable(C) On(B,C) ~~Holds(A)~~ ~~Clear(B)~~ On(A,B) Clear(A) Handempty

Wie werden diese Operatoren von MEA bestimmt. Ein offenes Teilproblem aus `Goal` ist `On(A,B)`. Der einzige Operator, der dies in der **add list** hat ist `STACK(A,B)`, um die Voraussetzungen für die Anwendung dieses Operators zu schaffen, das heißt `Clear(B)` (erfüllt) und `Holds(A)`, wird im rekursiven Aufruf der Operator `UNSTACK(C,A)` ausgewählt (um `Clear(A)` zu erreichen); dann `PUTDOWN(C)` um `Handempty` zu gewährleisten, um im folgenden `PICKUP(A)` auswählen zu können, der die Voraussetzung `Holds(A)` erreicht.

Das heißt nach dem ersten Schleifendurchlauf hat der erzeugte Teilplan die Gestalt: (UNSTACK(C,A), PUTDOWN(C), PICKUP(A), STACK(A,B)) und das Teilziel `On(A,B)` ist erfüllt.

Als zweites Teilziel wird nun `On(B,C)` ausgewählt. Zur Durchführung dieses Teilziels muß nun leider die oben gemachte Arbeit wieder zunichte gemacht werden. Sehen wir uns das im einzelnen an. Um `On(B,C)` zu erreichen, muß der Operator `STACK(B,C)` angewandt werden. Also müssen die Voraussetzungen für die Anwendungen geschaffen werden, das heißt `Clear C` (erfüllt) und `Holds(B)`, um letzteres zu erfüllen, muß man `Clear(B)` erreichen, damit ein `PICKUP(B)` oder ein `UNSTACK(B,x)` möglich wird. Dafür wird in der zweiten rekursiven Aufrufstufe `UNSTACK(A,B)` und `PUTDOWN(A)` ausgeführt[1], dann wird `PICKUP(B)` ausgewählt und der Operator `STACK(B,C)` ist anwendbar. Das heißt zu dem bisherigen Plan kommt im zweiten Schleifendurchlauf noch `(UNSTACK(A,B),PUTDOWN(A),PICKUP(B),STACK(B,C))` hinzu.
Das heißt im zweiten Schleifendurchlauf von MEA ist das erste bereits erfüllte Teilziel wieder zerstört worden und wird nun zum dritten Durchlauf der Schleife erneut ausgewählt. Dabei wird der Rest des Gesamtplanes, also `(PICKUP(A),STACK(A,B))`, erzeugt.
Wie sieht es nun aus, wenn wir die Teilziele in einer anderen Reihenfolge abarbeiten wollen, das heißt, wenn wir zuerst `On(B,C)` und dann `On(A,B)` erreichen wollen. Nicht besser, denn nun wird im ersten Durchlauf der Schleife `B` auf `C` gesetzt, was im zweiten Durchlauf wieder rückgängig gemacht werden muß.
Das Problem liegt letztendlich darin, daß das Planen über die Bearbeitungsreihenfolge von Teilzielen gesteuert wird, dies aber für das obige Problem nicht hinreichend ist. Wenn man nur ein Teilziel verfolgt, wird im zweiten Schritt das erste immer wieder kaputt gemacht. Das heißt die Lösung des Ziels `On(A,B),On(B,C)` kann nicht optimal dadurch gefunden werden, daß man zuerst eines der Teilprobleme und dann das andere erfüllt. Die Dekomposition führt zwangsläufig zu überflüssigen Schritten. Die Linearisierung der beiden Teilziele verhindert in diesem Fall zwar nicht grundsätzlich eine Lösung, aber sie führt erst einmal in die Irre, da beide Linearisierungen zu schlechten Plänen führen. Von daher haben wir einen entscheidenden Nachteil der MEA-Strategie in STRIPS gesehen. Dies ist insbesondere ein Nachteil, wenn die Durchführung des Planes erheblich teurer als dessen Erstellung ist.

4.3.2 In STRIPS unlösbare Probleme

Im Falle der Klötzchenwelt waren wir in der glücklichen Situation, daß es zu jedem Operator einen inversen Operator gibt – `STACK` versus `UNSTACK`, `PUTDOWN` versus `PICKUP`. In anderen Worten in der Klötzchenwelt kann man nichts prinzipiell falsch machen. Dies ist in allgemeinen Planungsaufgaben natürlich nicht immer gegeben. Es kann vielmehr passieren, daß durch die Anwendung eines Operators ein Zustand erreicht wird, aus dem eine im Prinzip vorhandene Lösung nicht mehr gefunden werden kann. Wir werden uns ein entsprechendes Beispiel nun näher ansehen (vgl. [Her89, S.68]).

[1] Bei nicht optimaler Ausführung könnte statt des `PUTDOWN(A)` ein `STACK(A,C)` durchgeführt werden, was natürlich die Sache erheblich verkomplizieren würde.

Im Beispiel soll der Inhalt zweier Programmvariablen `X` und `Y` vertauscht werden. Dazu ist natürlich eine dritte Hilfsvariable `Z` erforderlich. Weiterhin benötigen wir einen Zuweisungsoperator, der den Wert einer Variablen `u` an den Wert einer Variablen `v` bindet.

$\mathtt{WEISE-ZU}(\mathtt{u},\mathtt{v})$	
Precondition	$\mathtt{u} = \mathtt{a}\ \wedge$
	$\mathtt{v} = \mathtt{b}$
delete list	$\mathtt{u} = \mathtt{a}$
add list	$\mathtt{u} = \mathtt{b}$

Abbildung 4.9: Zuweisungsoperator

Betrachten wir nun den Anfangszustand und Zielzustand

Anfangszustand `Init`	Zielzustand `Goal`
`X = A`	`X = B`
`Y = B`	`Y = A`
`Z = undefiniert`	

Eine Lösung für das Problem lautet dann zum Beispiel:
`(WEISE-ZU(Z,X),WEISE-ZU(X,Y),WEISE-ZU(Y,Z))`.
Was macht STRIPS mit diesem Problem. Zuerst wählt es sich ein offenes Teilproblem aus, zum Beispiel `X = B`. Der einzige Operator, der dieses Problem in der **add list** hat ist `WEISE-ZU(X,Y)`. Dieser wird folglich ausgewählt und das erste Teilziel ist erfüllt. Der Zustand nach dem ersten Schleifendurchlauf ist nun:

```
X = B
Y = B
Z = undefiniert
```

Für den zweiten Schleifendurchlauf wird das Teilziel `Y=A` ausgewählt. Nun gibt es aber keinen Operator, der dieses Teilziel in seiner **add list** hat, folglich ist das Ziel nicht mehr lösbar. Dies ist auch klar, denn nach der ersten Zuweisung ist der ursprüngliche Wert von `X` überschrieben worden, ohne daß er gesichert worden wäre. Dasselbe Problem tritt natürlich auch auf, wenn man zuerst das Teilziel `Y = A` auswählt. In diesem Fall ist dann der ursprüngliche Wert von `Y` überschrieben worden. Folglich ist dieses Problem in STRIPS mit der Strategie MEA unlösbar. Der Grund dafür ist die zu enge Zielorientierung von MEA. Ist ein Teilziel ausgewählt, so wird ohne Rücksicht auf Verluste versucht, dieses Teilziel zu erfüllen. Dabei kann die Lösbarkeit von anderen Teilzielen zerstört werden. An sich müßte man also bei der Suche nach einem Plan flexibler vorgehen können, so daß auch scheinbare Umwege mit in die Suche einbezogen werden.

4.4 Teilzielregression

Eine Methode, das starre Planungsverfahren der Means-End-Analysis (MEA) zu erweitern, ist die Teilzielregression. Diese Methode wurde in [Wal77] eingeführt, die Darstellung im folgenden wurde von [Her89] übernommen. Bei diesem Verfahren soll das lineare Planen im Prinzip beibehalten werden. Es soll also weiterhin versucht, einen Gesamtplan dadurch zu finden, daß man sich auf Einzelziele konzentriert und diese zusammenfügt, allerdings nicht so starr wie bei MEA, wo die Pläne zur Erfüllung von neu ausgewählten Teilzielen immer an das Ende des bislang erzeugten Planes angehängt werden. Vielmehr soll in der Teilzielregression ein Punkt in der bislang gegebenen Operatorfolge ausgesucht werden, wo der Teilplan eingesetzt werden kann. Dies führt natürlich zu einer weiteren Unbestimmtheit im Algorithmus, was zu einem weiteren Rücksetztpunkt beim Scheitern einer Planungsaufgabe führt.
Man hat dabei das Problem zu lösen, daß man verhindern muß, daß gerade gelöste Teilziele durch die Lösung eines anderen Teilziels wieder zerstört werden. Deshalb geht man wie folgt vor: Vor jedem Operator im bislang erzeugten Teilplan und vor dem Planende markiert man, welche Merkmale an dieser Stelle im Plan zu gelten haben. Vor einem Operator sind das die Vorbedingungen des Operators, vor dem Planende die Planziele. Die Bedingung dafür, daß ein Teilplan `Q` in einen Teilplan `P = append(P',P")` zwischen `P'` und `P"` eingefügt werden darf, ist, daß er keine Zusicherung in `P` zerstört, das heißt, wenn in `Q` kein Merkmal `M` zerstört wird, das vor `Q` bereits gilt und nach `Q` als gefordert markiert ist, es sei denn, es wird von `Q` anschließend wieder erzeugt.
In der Praxis tritt dabei natürlich die Schwierigkeit auf, daß man, um ein Merkmal am Ende eines Planes garantieren zu können, mitten im Plan unter Umständen andere Voraussetzungen erfüllen muß, da Merkmale durch die Ausführung von Operatoren zunichte gemacht werden können. Man muß also die Regredierte des Merkmals über den entsprechenden Plan herstellen. Die Regression zu einem Operator `O` und einem Merkmal `M` ist dabei die schwächste Bedingung, die vor der Ausführung von `O` erfüllt sein muß, damit `M` nach der Ausführung gelten muß. Wir gehen dabei für das folgende von der vereinfachenden Annahme aus, daß die Regredierte eines Merkmals wieder die Gestalt einer atomaren Formel hat. Im allgemeinen Fall können dabei insbesondere Disjunktionen von Merkmalen entstehen, die durch einen etwas schwierigeren Algorithmus behandelt werden müßten. Im Gegensatz zu MEA darf bei Means-End-Analysis-mit-Teilzielregression (MEATR) auch nicht von einer aktuellen Situation ausgegangen werden, die die Anfangssituation überschreibt. Vielmehr muß die Anfangssituation immer noch zur Verfügung stehen, da man einen Teilplan an einer beliebigen Stelle in den schon erzeugten Plan einbauen können will.
Der Algorithmus für MEATR ist in Abbildung 4.10 gegeben. Er erhält als Eingabe eine Menge von Zielen `Goal`, einen Anfangszustand `Init`. Global definiert ist wieder eine Menge von Operatoren, die angewendet werden können. Ausgegeben wird ein Plan `P` als Liste von Operatoren. Als Beispiel für die Funktionsweise des Algorithmus sehen wir uns wieder die Sussman-Anomalie an, vergleiche Abbildung 4.8.

begin
0. `P:=NIL;`
1. **while** noch Ziele aus `Goal` offen in `P(S)`
   do
1.1. wähle Ziel `Goal_1` aus, das in `P(S)` offen ist
   ;;; da beliebige Auswahl → Rücksetzpunkt
1.2. wähle eine Zerlegung von `P` in `P = append(P',P");`
   ;;; da beliebige Auswahl → Rücksetzpunkt
1.3. wähle Instanz `I` eines Operators aus, die `Regr(Goal_1,P")` in der **add list** hat, so daß `append(MEATZR(`**Precondition**`(I),P'(S)),list(I))` kein markiertes Ziel bis zum Ende des Plans verletzt;
   ;;; da beliebige Auswahl → Rücksetzpunkt
1.4. `P:= append(P',MEATZR(`**Precondition**`(I),P'(S)),list(I),P")`
   end_do
2. markiere die Ziele aus `Goal` nach `P` falls noch nicht geschehen
   return`(P)`
end

Abbildung 4.10: Algorithmus MEATZR (means-end-analysis mit Teilzielregression)

Der Anfangszustand und der Zielzustand sind also wie gehabt:

`Ontable(A)`	
`Ontable(B)`	
`On(C,A)`	`On(A,B)`
`Clear(B)`	`On(B,C)`
`Clear(C)`	
`Handempty`	`Goal`
`Init`	

Die Aufrufe von MEATZR sind dann:

```
MEATZR((On(A,B),On(B,C)),Init)
 0   P:=NIL;
 1.1 Goal_1:=On(A,B)
 1.2 P':=NIL;P":=NIL;
 1.3 Regr(Goal_1,NIL)=On(A,B)
     I:=STACK(A,B)

     MEATZR((Clear(B),Holds(A)),Init)
     0   P:=NIL;
     1.1 Goal_1:=Holds(A)
     1.2 P':=NIL;P":=NIL;
     1.3 Regr(Holds(A),NIL)=Holds(A)
         I:=PICKUP(A)
```

```
            MEATZR((Ontable(A),Clear(A),Handempty),Init)
            0   P:=NIL;
            1.1  Goal_1:=Clear(A)
            1.2  P':=NIL;P":=NIL;
            1.3  Regr(Goal_1,NIL)=Clear(A)
                 I:=UNSTACK(C,A)

                 MEATZR((Handempty,Clear(C),On(C,A)),Init)
                 0   P:=NIL;
                 1    return(NIL)  ;;; alle Ziele erfüllt
                 (UNSTACK(C,A)) ;;; verletzt kein markiertes Ziel
                                ;;; bis Ende des Planes

            1.4  P:=list(UNSTACK(C,A))
            1.1  Goal_1:=Handempty
            1.2  P':=UNSTACK(C,A);P":=NIL
            1.3  Regr(Handempty,NIL)=Handempty
                 I:=PUTDOWN(C)

                 MEATZR((Holds(C)),(UNSTACK(C,A))(Init))
                 ..  ;;; leerer Plan
            1.4  P:=(UNSTACK(C,A),PUTDOWN(C))
            2    return(UNSTACK(C,A),PUTDOWN(C))
            (UNSTACK(C,A),PUTDOWN(C),PICKUP(A)) ;;; verletzt kein
             ;;; markiertes Ziel bis Ende des Planes

      1.4   P:=(UNSTACK(C,A),PUTDOWN(C),PICKUP(A))
      2     return(UNSTACK(C,A),PUTDOWN(C),PICKUP(A))
      (UNSTACK(C,A),PUTDOWN(C),PICKUP(A),STACK(A,B)) ;;; verletzt kein
       ;;; markiertes Ziel bis Ende des Planes

1.4 P:=(UNSTACK(C,A),PUTDOWN(C),PICKUP(A),STACK(A,B))
;;; Der so erzeugte Teilplan ist genau der, der auch von MEA erzeugt wurde.
;;; Vergleiche Seite 29.

1.1 Goal_1:=On(B,C)
1.2 P':=(UNSTACK(C,A),PUTDOWN(C));P":=(PICKUP(A),STACK(A,B))
     ;;; Alle anderen Zerlegungen führen zur Verletzung von Zusicherungen
1.3 Regr(On(B,C),(PICKUP(A),STACK(A,B)))=On(B,C)
    I:=STACK(B,C)
```

```
        MEATZR((Clear(C),Holds(B)),(UNSTACK(C,A),PUTDOWN(C))(Init))
        0    P:=NIL;
        1.1  Goal_1:=Holds(B)
        1.2  P':=NIL;P":=NIL;
        1.3  Regr(Holds(B),NIL)=Holds(B)
             I:=PICKUP(B)

             MEATZR((Ontable(B),Clear(B),Handempty),
                    (UNSTACK(C,A),PUTDOWN(C))(Init))
             PICKUP(B)  ;;; verletzt kein markiertes Ziel bis zum Ende
                        ;;; des Plans
        1.4  P:=(PICKUP(B))
        2    return(PICKUP(B))
        (PICKUP(B),STACK(B,C))  ;;; verletzt kein markiertes Ziel bis
                                ;;; zum Ende des Plans
    1.4 P:=(UNSTACK(C,A),PUTDOWN(C),PICKUP(B),
           STACK(B,C),PICKUP(A),STACK(A,B))  ;;; keine offenen Ziele mehr

    2    return(P)
     end
```

Das heißt, mit MEATZR kann man eine optimale Lösung für die Sussman-Anomalie finden, allerdings hat man den Preis größerer Unbestimmtheit des Algorithmus zu zahlen, man muß also auf Grund von mehr Rücksetzmöglichkeiten eine größere Suche in Kauf nehmen.
Übrigens findet MEATZR auch für das Beispiel aus Abschnitt 4.3.2 eine Lösung.
Trotz der größeren Flexibilität der Means-End-Analysis mit Teilzielregression gibt es immer noch Einschränkungen, denn man kann immer nur komplette Teilpläne in die Operatorfolge einsetzen, allerdings an beliebige Stellen. Zum Beispiel kann der folgende Plan von MEATZR nicht erzeugt werden:

$$(\text{Teilplan}_1 \text{ für Ziel } \mathtt{A}, \text{Teilplan}_2 \text{ für Ziel } \mathtt{B},$$
$$\text{Teilplan}_3 \text{ für Ziel } \mathtt{A}, \text{Teilplan}_4 \text{ für Ziel } \mathtt{B})$$

dennoch könnte ein solcher Plan natürlich Sinn machen.
Obwohl man mit der Teilzielregression das Problem der Abhängigkeiten von Teilzielen, wie es in STRIPS auftritt, lösen kann, hat das Verfahren doch einen Nachteil. Man nimmt immer eine lineare Ordnung der Pläne an, wenn diese in der Form nicht aufrechterhalten werden kann, setzt man im Suchraum auf den letzten Verzweigungspunkt zurück und versucht es mit einer anderen Ordnung, von der man hofft, daß damit ein Plan erzeugt werden kann. Ein Ausweg aus diesem umständlichen Verfahren besteht darin, daß man von der Idee der Linearität abgeht und Pläne nur noch als partiell geordnet ansieht. Man kommt dann zum nichtlinearen Planen. Wir werden uns das nächstemal damit beschäftigen.

4.5 Planen mit Heuristiken

Nun noch einige Bemerkungen zu den vielen Verzweigungsmöglichkeiten in STRIPS. An jeder Verzweigung erlaubt der Algorithmus verschiedene Möglichkeiten. Dabei kann man sich von einer Zufallsauswahl leiten lassen oder aber gewisse Heuristiken anwenden. Man kann im Extremfall sogar insgesamt auf einen Planungsalgorithmus wie STRIPS verzichten und die Planung einem wissensbasierten System überlassen, das solchen Heuristiken folgt. Dabei ist in gewisser Weise das Problem natürlich verlagert worden: In STRIPS wird für die Planung eine Auswahl von Operatoren getroffen, im Falle eines wissensbasierten Systems muß man erst einmal das Wissen in Form von Heuristiken formulieren und dann muß eine Suche innerhalb dieser Heuristiken erfolgen (zum Beispiel, welcher Heuristik soll in einer konkreten Situation gefolgt werden). Man kann die verschiedenen Formen, wie man ein und denselben Algorithmus darstellen kann, auch als die Auswahl eines Programmierparadigmas für die Implementierung ansehen, prozedural – wie MEA oben angegeben wurde – oder regelbasiert – wie es bei der Implementierung in einer Expertensystemshell geschähe. Wie sehen nun solche Heuristiken für MEA aus?
Einige Heuristiken für MEA:

- „Wenn ein Klötzchen auf dem Tisch steht und es ein Ziel ist, es in der Hand zu halten, dann wähle den Operator `PICKUP`."
- „Wenn ein Klötzchen in einem Turm enthalten ist und noch bewegt werden muß, dann setze keine anderen Klötzchen auf den Turm."
- „Wenn es kein Ziel ist, daß Klötzchen `x` auf Klötzchen `y` steht, dann vermeide den Operator `STACK(x,y)`."
- „Wenn mehrere Operatoren ein offenes Teilziel erfüllen können, dann wähle den, bei dem die meisten Vorbedingungen schon erfüllt sind."
- „Wenn ein Planziel bereits erfüllt ist, dann vermeide Operatoren, die es wieder zerstören."

Die letzte Heuristik wird von MEATZR besonders Ernst genommen.

Kapitel 5

Nichtlineares Planen

Beim linearen Planen mit STRIPS (MEA) und den entsprechenden Erweiterungen aus Kapitel 4 sind wir davon ausgegangen, daß ein Plan durch eine Liste gegeben ist, das heißt, daß ein Plan aus einer linearen Folge von Operatoren besteht. Dies führt allerdings zu Redundanzen in Plänen. Hat man zum Beispiel das folgende Planungsproblem:

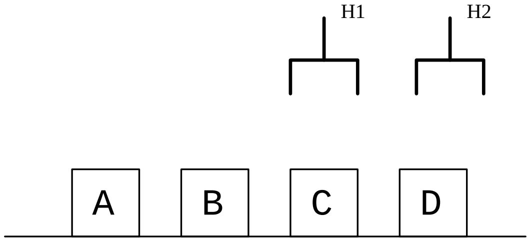


Abbildung 5.1: Ausgangszustand

```
Ontable(A)
Ontable(B)
Ontable(C)
Ontable(D)
Clear(A)
Clear(B)
Clear(C)
Clear(D)
Handempty
```

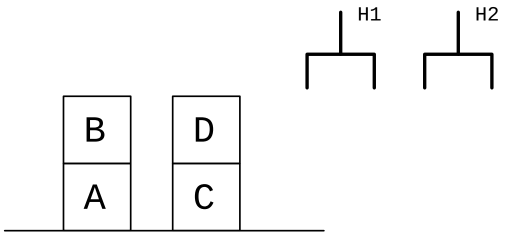


Abbildung 5.2: Zielzustand

```
On(B,A)
On(D,C)
Handempty
```

dann gibt es zur Lösung des Planungsproblems im wesentlichen zwei minimale lineare Pläne, nämlich (`PICKUP(B)`,`STACK(B,A)`,`PICKUP(D)`,`STACK(D,C)`) und (`PICKUP(D)`, `STACK(D,C)`,`PICKUP(B)`,`STACK(B,A)`). Man muß also zur Lösung des Planungsproblems Klötzchen `B` auf Klötzchen `A` stellen und `D` auf `C`. Allerdings spielt die Reihenfolge, in der man diese Aktionen ausführt, keine Rolle. Angemessener zur Darstellung des Planes wäre daher eine partielle Ordnung der Operatoren, wie zum Beispiel durch:

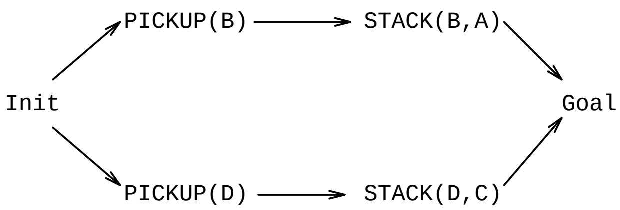


Abbildung 5.3: nichtlinearer Plan

Diese Darstellung ist für den Fall, daß man nicht nur einen Greifarm zur Verfügung hat, sondern zwei noch erheblich besser, denn nun können die beiden Teilpläne parallel ausgeführt werden. Die obige Darstellung soll verdeutlichen, daß zum einen die Operatorfolge `PICKUP(B)`,`STACK(B,A)` und zum anderen die Operatorfolge `PICKUP(D)`,`STACK(D,C)` ausgeführt werden muß. Innerhalb dieser Folgen ist die Reihenfolge festgelegt (das heißt zuerst `PICKUP` dann `STACK`), aber nicht zwischen den Folgen (das heißt, es ist egal, ob zuerst `B` auf `A` und dann `D` auf `C` gestellt wird oder umgekehrt. Die Erreichung der beiden Teilziele `On(B,A)` und `On(D,C)` ist also entkoppelt worden.
Wichtig für den Erfolg von nichtlinearem Planen ist dabei, daß eine Trennung in von einander unabhängig behandelbaren Teilzielen zumindest weitgehend möglich ist. In anderen Worten, das nichtlineare Planen wird besonders dann gut funktionieren, wenn keine Teilzielinteraktionen vorliegen oder aber die Teilziele zumindest wenig interagieren.
Bei der Interpretation von nichtlinearen Plänen wie in Abbildung 5.3 gibt es im wesentlichen drei Möglichkeiten:

schwache Nichtlinearität: parallele Zweige eines Planes dürfen als *Ganzes* in beliebiger Reihenfolge ausgeführt werden.

starke Nichtlinearität: die Operatoren in parallelen Zweigen dürfen *in jeder vollständigen Ordnung* ausgeführt werden, die mit der partiellen, nichtlinearen Ordnung verträglich ist.

Nebenläufigkeit: die parallelen Zweige dürfen parallel ausgeführt werden.

Beim obigen nichtlinearen Plan gibt es also bei angenommener schwacher Nichtlinearität genau die beiden oben angegebenen Pläne. Bei angenommener starker Nichtlinearität gibt es noch zusätzlich Pläne, wie (`PICKUP(B)`,`PICKUP(D)`,`STACK(B,A)`, `STACK(D,C)`), die nur bei der Annahme einer zweiten Greifhand Sinn machen.
Die Vorstellung der Nebenläufigkeit ist insbesondere in Anwendungsbereichen wie automatischer maschineller Fertigung untersucht worden, wo die Ausführung von einzelnen Teilaufgaben zeitkritisch ist, das heißt die Ausführung eines bestimmten Operators, der eine bestimmte Resource (wie die Greifhand oder aber einen Schraubenzieher) belegt, darf zum Beispiel höchstens 30 Sekunden dauern, damit dann die Resource anderweitig verwendet werden kann. In solchen Plänen haben die Operatoren zusätzlich immer noch einen Start- und einen Endzeitpunkt, durch die die Ausführungszeit festgelegt wird.

5.1 Die Grundidee des nichtlinearen Planens

Die Grundidee des nichtlinearen Planens besteht darin, daß man zuerst annimmt, man könne die Merkmale der Zielbeschreibung durch Unterpläne gesondert erreichen und Interaktionen träten nicht auf. Hat man solch einen unfertigen nichtlinearen Plan, löst man in einem zweiten Schritt eventuell auftretende Interaktionen auf.
Zuerst gehen wir mal davon aus, daß Pläne partiell repräsentiert werden können und zwar nicht nur durch die Angabe von Operatoren, sondern auch durch die Angabe von Teilzielen in einer partiellen Ordnung. Im Einführungsbeispiel dieses Kaptiels heißt das, man gibt einen Plan vor durch die Teilziele `On(B,A)` und `On(D,C)`. Dieser Plan muß natürlich verfeinert werden durch die Angabe von Operatoren, zum Beispiel durch `STACK(B,A,H1)` zur Erzeugung von `On(B,A)`. Dann müssen allerdings auch wieder die Vorbedingungen von `STACK` als neue Teilziele eingefügt werden, das heißt der Plan hat nun die Gestalt:

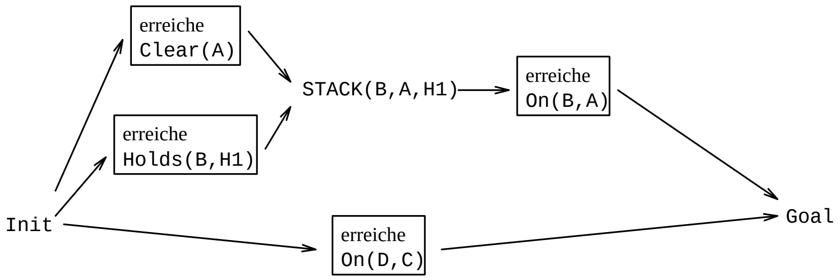


Abbildung 5.4: partieller nichtlinearer Plan

Der obige Plan wird dann der Reihe nach durch Auswahl von noch offenen Knoten vervollständigt. Ein Fragment eines Algorithmus zum nichtlinearen Planen ist gegeben durch:

begin
`1.` **while** noch Knoten ungelöst in `P` **do**
`1.1.` wähle Knoten `K` aus, der in `P` ungelöst
;;; da beliebige Auswahl → Rücksetzpunkt
`1.2.` wähle Instanz `I` eines Operators aus, die das in
`K` als zu erreichend geforderte Merkmal in der **add list** hat
;;; da beliebige Auswahl → Rücksetzpunkt
`1.3.` füge `I` als Operatorknoten direkt vor `K` in `P` ein
`1.4.` füge Erreiche-Knoten für die Vorbedingungen von `I` untereinander
angeordnet direkt vor `I` ein
`1.5.` löse Interaktionen auf
`1.6.` kritisiere den Plan
end_do
`2.` **return**`(P)`
end

Abbildung 5.5: Algorithmus NLP (nichtlineares Planen)

Die entscheidende Frage ist dabei natürlich die Frage nach der Auflösung der Interaktionen. Übrigens könnte man im Teilplan von Abbildung 5.4 zwar das Kästchen „erreiche `On(B,A)`“ löschen, aber für eventuelles Umplanen kann es nützlich sein, sich zu merken, warum man was gemacht hat.
Im folgenden wollen wir nun den Algorithmus NLP präzisieren, dazu werden wir zuerst einige grundlegenden Begriffe einführen und klären, wann ein Knoten in einem Plan als gelöst angesehen werden kann. Im weiteren werden wir betrachten, wie Interaktionen aufgelöst werden können, schließlich werden wir sehen, was man unter Kritiken von Plänen zu verstehen hat. Die Darstellung ist wieder an [Her89] angelehnt.

5.2 Grundlegende Begriffe des nichtlinearen Planens

Ein wesentlicher Bestandteil von Plänen im nichtlinearen Planen besteht darin, daß man eine neue Art von Kanten einführt, die sogenannten *Abhängigkeiten*, die die Information tragen, „wer was für wen erzeugt“. Eine Abhängigkeit verbindet zwei bezüglich der partiellen Ordnung hintereinanderliegende (nicht notwendigerweise direkt hintereinanderliegende) Planknoten, von denen der vordere K_1, die Regredierte des Merkmals M erzeugt und der hintere K_2 das Merkmal M erfordert.
Wir formalisieren nun die Begriffe durch die folgenden Definitionen weiter:

5.1 DEFINITION (PLAN): Das geordnete Paar $P = \langle \mathbf{K}, < \rangle$ ist ein *Plan*, wenn **K** eine Menge von Knoten ist und $<$ eine strikte, partielle Ordnung (d.h. irreflexive, antisymmetrische und transitive Relation) auf **K** ist mit eindeutigem kleinstem Element `Init(P)` und eindeutigem größten Element `Goal(P)`. ■

Die folgende Definition soll die Beziehung zwischen Knoten und bestimmten Merkmalen klären.

5.2 DEFINITION (ERZEUGER, NUTZER): Sei K ein Knoten und M ein Merkmal. K ist ein *Erzeuger* von M, wenn M aus den Nachbedingungen von K ableitbar ist. K ist *Nutzer* von M, wenn M eine Vorbedingung von K ist. ■

Es ergibt sich dabei allerdings, daß diese Definition für den allgemeinen Fall nicht hinreichend ist. Zum Beispiel kann es ja durchaus sein, daß für ein Merkmal mehrere Erzeuger erforderlich sind. Weiterhin muß im allgemeinen ein Merkmal nur aus der **add list** eines Erzeugers ableitbar sein, aber es muß nicht unmittelbar erzeugt werden. Zum Beispiel erzeugt der Operator `STACK(A,B)` direkt das Merkmal `On(A,B)`. Betrachten wir hingegen ein Merkmal[1] `Above(A,C)`, das besagt, daß sich A irgendwo in einem Turm oberhalb von C befindet, dann wird `Above(A,C)` nicht direkt

[1] Man definiert `Above(x,y)` üblicherweise durch:
$\forall x, y\ \mathtt{Above}(x,y) \Leftrightarrow \mathtt{On}(x,y) \vee (\exists z\ \mathtt{Above}(x,z) \wedge \mathtt{On}(z,y))$

erzeugt, läßt sich aber unter Umständen aus der Nachbedingung des Operators und dem Zustand davor herleiten. Probleme, die sich aus der Unentscheidbarkeit ergeben, werden im folgenden nicht diskutiert, bei endlichen Individuenbereichen, wie im Fall der Klötzchenwelt, treten sie auch nicht auf.
Die Definition 5.1 von Plan ist sehr allgemein, sie wird nun auf solche Pläne eingeschränkt, bei denen für alle Knoten eine Erzeuger-Nutzer-Beziehung besteht.

5.3 Definition (wohlgeformter Plan): Ein Plan $\mathtt{P} = \langle \mathbf{K}, < \rangle$ heißt *wohlgeformt*, wenn es für alle Merkmale $\mathtt{M}$ und alle Knoten $\mathtt{N} \in \mathbf{K}$ einen Knoten $\mathtt{E} \in \mathbf{K}$ gibt, so daß gilt:
Wenn $\mathtt{N}$ Nutzer von $\mathtt{M}$ ist, dann ist $\mathtt{E}$ Erzeuger von $\mathtt{M}$ und $\mathtt{E} < \mathtt{N}$.
$\mathtt{P}$ heißt *wohlgeformt bis auf* $\mathtt{K}$, wenn $\langle \mathbf{K}', <' \rangle$ wohlgeformt ist mit $\mathbf{K}' := \mathbf{K} \backslash \{\mathtt{K}\}$ und $<' = \{\langle \mathtt{k}_1, \mathtt{k}_2 \rangle \in < | \mathtt{k}_1, \mathtt{k}_2 \neq \mathtt{K}\}$. ■

Ein wohlgeformter Plan ist also ein Plan, bei dem alle Merkmale, die von Nutzern benutzt werden, auch schon vorher erzeugt werden. Da Start und Ziel als eindeutig kleinstes beziehungsweise größtes Element definiert sind, macht der Startknoten nur Zusicherungen und der Zielknoten nur Forderungen (nämlich die Zielbedingungen).
Da wir in der Regel nur wohlgeformte Pläne betrachten, lassen wir das Adjektiv „wohlgeformt" meist weg. Pläne, die bis auf einen Knoten wohlgeformt sind, treten auf, wenn wir Konflikte auflösen.
In einem wohlgeformten nichtlinearen Plan gibt es im allgemeinen mehrere Erzeuger einer Vorbedingung eines Knotens, nämlich solche Erzeuger, die in zueinander parallelen Ästen vor dem Nutzer oder solche, die irgendwo weiter vorne im Plan stehen.
Durch die folgende Definition wird geklärt, wann ein Knoten $\mathtt{N}$ von einem anderen Knoten $\mathtt{E}$ abhängig ist, nämlich, wenn $\mathtt{E}$ ein Merkmal $\mathtt{M}$ erzeugt, $\mathtt{N}$ das Merkmal $\mathtt{M}$ nutzt und kein anderer Nutzer von $\mathtt{M}$ zwischen $\mathtt{E}$ und $\mathtt{N}$ steht.

5.4 Definition (Abhängigkeit): Sei $\mathtt{P} = \langle \mathbf{K}, < \rangle$ ein Plan, $\mathtt{E}, \mathtt{N} \in \mathbf{K}$ und $\mathtt{M}$ ein Merkmal. $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ ist eine *Abhängigkeit* zwischen $\mathtt{E}$ und $\mathtt{N}$ bezüglich $\mathtt{M}$ in $\mathtt{P}$ wenn:

- $\mathtt{E}$ ist Erzeuger von $\mathtt{M}$
- $\mathtt{N}$ ist Nutzer von $\mathtt{M}$
- $\mathtt{E} < \mathtt{N}$
- $\forall \mathtt{K} \in \mathbf{K}$ mit $\mathtt{E} < \mathtt{K} < \mathtt{N}$ gilt $\mathtt{K}$ ist nicht Erzeuger von $\mathtt{M}$ ■

Aufgrund der Bedingung $\mathtt{E} < \mathtt{N}$ ist gewährleisted, daß die erzwungenen Teilordnungen mit der Operatorordnung konsistent ist.
Die erste Frage, die wir nun zur Präzisierung des Algorithmus NLP klären wollen, betrifft Punkt 1.1 des Algorithmus in Abbildung 5.5: Wann ist ein Knoten gelöst beziehungsweise ungelöst? Dann werden wir uns der Hauptaufgabe im nichtlinearen Planen zuwenden, dem Auflösen von Interaktionen.

Wenn wir prüfen wollen, ob an einem Knoten K Merkmal M gelten kann, sehen wir nach, ob eine Abhängigkeit bezüglich M nach K besteht, oder wir versuchen sie zu etablieren. Dazu suchen wir einen Knoten K_1 vor K, der M erzeugt, so daß zwischen K_1 und K das Merkmal M nicht wieder „in jedem Fall“ zerstört wird, das heißt, es muß mindestens eine Linearisierung des Planes geben, in der M zwischen K_1 und K nicht wieder zerstört wird. Es ist also durchaus möglich, daß eine Abhängigkeit verletzt wird, daraus kann dann ein Konflikt entstehen, der erkannt und aufgelöst werden muß. Die Sache kompliziert sich dadurch, daß man statt M weiter vorne im Plan immer die Regredierte betrachten muß. Weiterhin können in Merkmalen auch noch Variable auftreten wie in On(A, x), die man dann auch noch durch konkrete Instanzen ersetzen muß. Dies sei aber für das folgende ausgeklammert.
Ein Plan ist also vollständig, wenn jedes der Zielmerkmale im Zustand Goal gilt, das heißt, wenn es zu jedem dieser Zielmerkmale eine Abhängigkeit führt. Von den Startmerkmalen können Abhängikeiten in den Plan führen (es müssen aber nicht von *allen* Startmerkmalen Abhängigkeiten in den Plan führen).
Nun wollen wir an einem Beispiel sehen, wie diese Abhängigkeit gesehen werden kann. Sehen wir uns dazu den Plan aus Abbildung 5.6 an, der eine Weiterplanung aus Abbildung 5.4 ist.

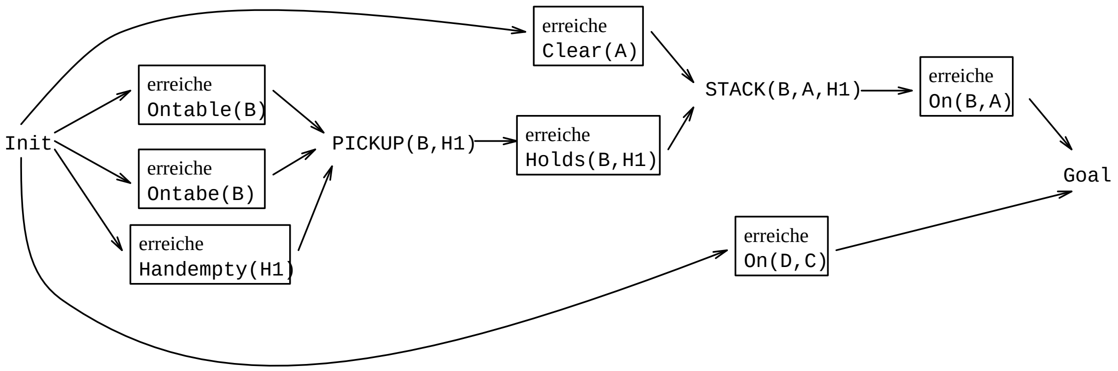


Abbildung 5.6: nichtlinearer Plan

Wie kann man nun feststellen, daß das Teilziel On(B,A) erfüllt ist? Ausgehend von On(B,A) stellt man eine direkte Kante zu STACK(B,A,H1) fest, die das Ziel etabliert; oder vorsichtiger gesagt, dadurch wird es erzeugt, es könnte noch woanders (konkret gesprochen im zweiten Zweig) wieder zerstört werden. Für die Vorbedingungen von STACK(B,A,H1) müssen wir dann entprechend Abhängigkeiten herstellen: Clear(A) gilt in Init, Holds(B,H1) wird durch die Kante zu PICKUP(B,H1) erzeugt, die Vorbedingungen dieses Operators gelten im initialen Zustand. Das Merkmal On(D,C) kann hingegen nicht nicht etabliert werden, da On(D,C) im Vorgängerzustand, nämlich Init nicht ableitbar ist.

5.3 Erkennen und Auflösen von Konflikten

Wenden wir uns nun der Hauptschwierigkeit bei NLP zu, dem Punkt 1.5 im Algorithmus aus Abbildung 5.5, dem Auflösen von Interaktionen. Ein *Konflikt* ist dabei eine *negative* Interaktion, das heißt die Behinderung von einem Teilplan durch einen anderen. Sehen wir uns dazu das folgende Beispiel an:

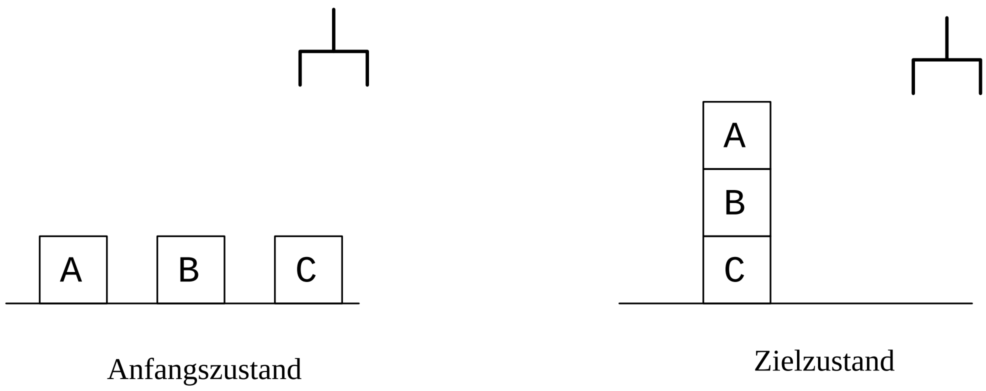


Abbildung 5.7: Interaktion

Der Zielzustand sei beschrieben durch `On(A,B), On(B,C)`. Der Algorithmus NLP liefert als Zwischenergebnis:

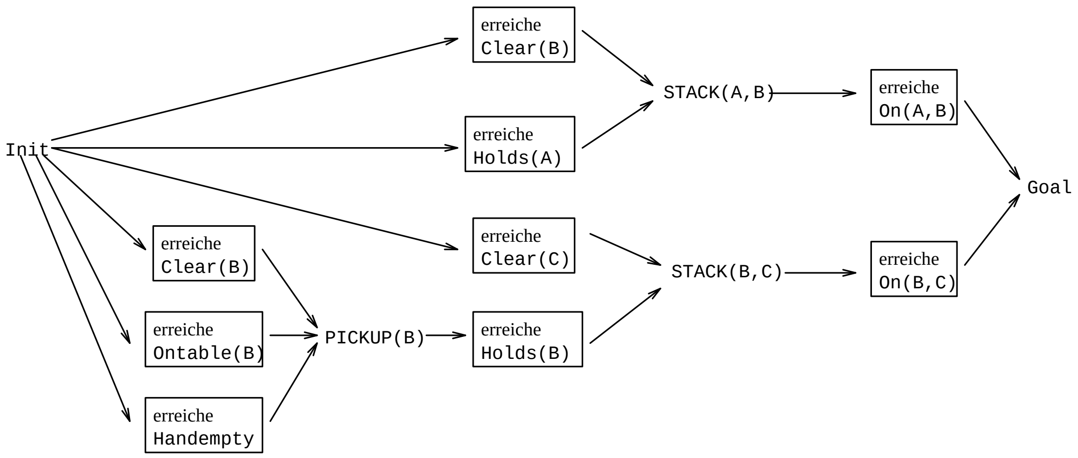


Abbildung 5.8: Zwischenergebnis einer Planung

In diesem Teilplan besteht eine Abhängigkeit des Knotens `STACK(A,B)` zum Ziel bezüglich des Merkmals `On(A,B)`. Diese Abhängigkeit wird, wenn man den oberen und dann den unteren Planzweig ausführt durch den Knoten `PICKUP(B)` verletzt, da `A` nicht auf `B` stehen kann, wenn `B` sich noch in der Hand befindet.

Woraus ergibt sich der Konflikt nun formal? Der Konflikt besteht darin, daß es im oberen Zweig eine Abhängigkeit bezüglich des Merkmals `Clear(B)` gibt, dieses Merkmal im unteren Zweig aber durch den Operator `PICKUP(B)` zerstört wird. Setzen wir also starke Nichtlinearität voraus, so ist die Reihenfolge der Ausführung der Operatoren zwischen dem oberen und dem unteren Zweig nicht mehr beliebig; denn die Reihenfolge $(\mathtt{PICKUP(B)}, \mathtt{STACK(A,B)}, \mathtt{STACK(B,C)})$ ist zum Beispiel nicht möglich. Folglich muß dieser Konflikt aufgelöst werden. Dazu wollen wir zuerst den Begriff „Konflikt“ formalisieren. Dazu brauchen wir die Begriffe der Erweiterung und der Einfügung in einen Plan.

5.5 Definition (Erweiterung): Sei $<$ eine strikte partielle Ordnung auf $\mathbf{K}$. $<'$ heißt *Erweiterung* von $<$ genau dann, wenn $<'$ eine strikte partielle Ordnung auf $\mathbf{K}$ ist mit $< \subseteq <'$. ∎

5.6 Definition (Einfügung): Sei $\mathtt{P} = \langle \mathbf{K}, < \rangle$ ein Plan, die Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ *umschließt* einen Knoten $\mathtt{K} \in \mathbf{K}$ genau dann, wenn es eine Erweiterung $<'$ von $<$ gibt, so daß $\mathtt{E} <' \mathtt{K} <' \mathtt{N}$ gibt. $<'$ heißt dann *Einfügung* von `K` in `A`. ∎

Ein Konflikt liegt nun vor, wenn ein Zerstörer von `M` (also ein Erzeuger von $\neg\mathtt{M}$) in eine Abhängigkeit bezüglich `M` so eingefügt wird, daß `M` anschließend nicht wieder hergestellt wird.

5.7 Definition (Konflikt): Ein Plan $\mathtt{P} = \langle \mathbf{K}, < \rangle$ enthält einen *Konflikt* zwischen einem Knoten $\mathtt{K} \in \mathbf{K}$ und einer Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ bezüglich des Merkmals `M` wenn `K` Erzeuger von $\neg M$ ist und es eine Einfügung $<'$ von `K` in `A` gibt, so daß für alle $\mathtt{L} \in \mathbf{K}$ gilt: Wenn $\mathtt{K} <' \mathtt{L} <' \mathtt{N}$ dann ist `L` kein Erzeuger von `M`.
Ein Plan, der keine Konflikte enthält, heißt *konfliktfrei*. ∎

Das Erkennen von Konflikten könnte man nun durch eine Breitensuche durch alle möglichen Einfügungen austesten. Für praktische Zwecke wird dies aber oft zu ineffizient sein, da gerade die Konflikterkennung eine häufig ausgeführte Operation ist. Daher ist eine Lösung dieses Problems für praktische Anwendungen sehr wichtig, wir werden sie hier aber nicht verfolgen.
Wie können nun einmal erkannte Konflikte, nachdem sie erkannt sind, aufgelöst werden? Grundsätzlich gibt es dazu zwei Möglichkeiten:

1. durch teilweise Linearisierung des Plans, so daß das gefährdete Merkmal aus der Gefahrenzone gebracht wird, oder aber

2. durch Einfügung von weiteren Erzeugern, die das gefährdete Merkmal absichern

Die eigentliche Kunst des nichtlinearen Planens besteht nun darin, dies entsprechend zu präzisieren, so daß daraus ein Algorithmus entsteht. Dazu müssen also die Fragen beantwortet werden, wie man im Falle eines Konfliktes die Ordnung verschärft und wo man welche Erzeuger neu einfügt.

Einen Weg zur Konfliktauflösung wollen wir uns nun anschauen. Die Grundidee besteht darin, daß sich jeder mögliche Konflikt in eine Menge von Elementarkonflikten auflösen läßt, die ihrerseits in drei Kategorien eingeteilt werden können, nämlich die Linearkonflikte, die Gabelkonflikte und die Parallelkonflikte. Für jede einzelne Klasse von Konflikten gibt man dann an, wie diese aufgelöst werden können. Dies wollen wir uns nun im einzelnen anschauen.
Es sollte allerdings bedacht werden, das die Transformation von Konflikten in Elementarkonflikte und die getrennte Auflösung dieser nicht immer die effizienteste Methode sein muß.
Für diese drei Grundformen von Konflikten ist die relative Lage der Knoten E, K und N innerhalb der Operatorordnung entscheidend. Zur Erinnerung E ist der Erzeuger eines Merkmals M, K ist der Erzeuger von $\neg$M, und N ist der Nutzer von M.
Da es sich bei $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ um eine Abhängigkeit handelt, gilt also $\mathtt{E} < \mathtt{N}$. Ein Konflikt kann natürlich nur dann auftreten, wenn durch eine Einfügung K umschlossen werden kann, also nicht, wenn K vor E oder hinter N liegt. Die einfachste Möglichkeit geht von einer linearen Anordnung der drei Knoten aus, das heißt alle drei Knoten sind miteinander bezüglich der Ordnung vergleichbar, bei einem Gabelkonflikt ist K nur mit einem anderen Knoten vergleichbar (also entweder mit E oder mit N) und bei einem Parallelkonflikt ist K mit keinem der beiden anderen Knoten vergleichbar.
Im folgenden geben wir für die drei verschiedenen Typen von Konflikten die Definitionen. Nächstesmal werden wir uns dann anschauen, wie die Konfliktlösung für jeden dieser Fälle aussieht.

5.8 Definition (Linearkonflikt): Sei $\mathtt{P} = \langle \mathbf{K}, < \rangle$ ein Plan, ein Konflikt in P zwischen dem Knoten $\mathtt{K} \in \mathbf{K}$ und der Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ heißt *Linearkonflikt* wenn $\mathtt{E} < \mathtt{K}$ und $\mathtt{K} < \mathtt{N}$ gilt. ■

Das Schema hat also die Gestalt einer linearen Anordnung der drei Knoten (K kann ja nicht vor oder hinter den anderen Knoten liegen):

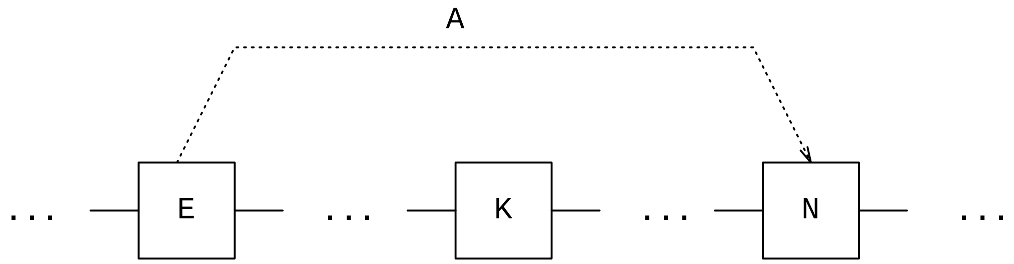


Abbildung 5.9: Schema Linearkonflikt

5.9 Definition (Gabelkonflikt): Sei $\mathtt{P} = \langle \mathbf{K}, < \rangle$ ein Plan, ein Konflikt in P zwischen dem Knoten $\mathtt{K} \in \mathbf{K}$ und der Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ heißt *Gabelkonflikt* wenn entweder $\mathtt{K} < \mathtt{N}$ oder $\mathtt{E} < \mathtt{K}$ gilt. ■

Im Fall des Gabelkonflikts gibt es also die Möglichkeiten, daß der Knoten K vor N, aber parallel zu E, oder aber hinter E, aber parallel zu K liegt.

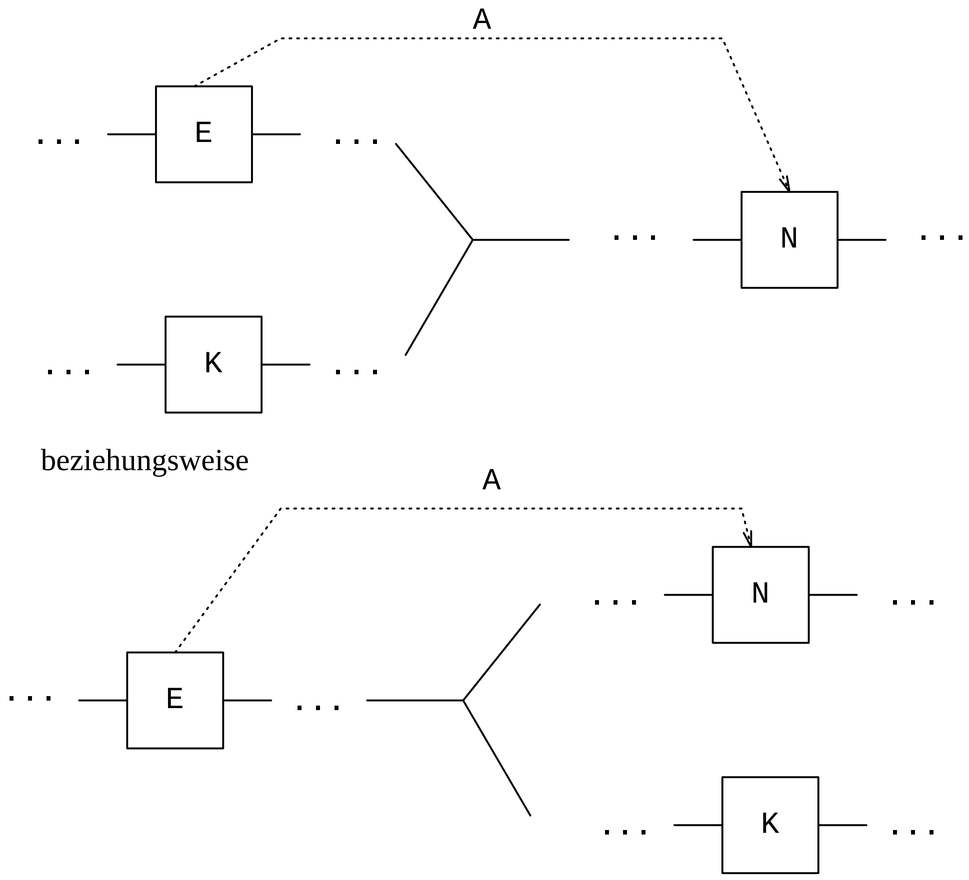


Abbildung 5.10: zwei Schemata für Gabelkonflikt

5.10 Definition (Parallelkonflikt): Sei $\mathtt{P} = \langle \mathbf{K}, < \rangle$ ein Plan, ein Konflikt in $\mathtt{P}$ zwischen dem Knoten $\mathtt{K} \in \mathbf{K}$ und der Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ heißt *Parallelkonflikt* wenn weder $\mathtt{K} < \mathtt{N}$ noch $\mathtt{E} < \mathtt{K}$ gilt. ■

Das Schema des Parallelkonflikts sieht wie folgt aus:

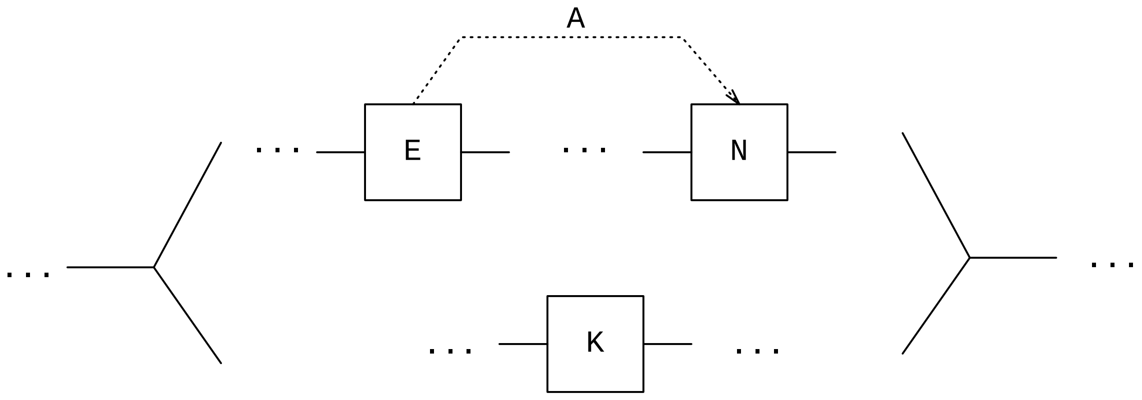


Abbildung 5.11: Schema Parallelkonflikt

Nun sehen wir uns an, wie in den drei möglichen elementaren Konflikten eine Auflösung erreicht werden kann.
Sehen wir uns zuerst den Linearkonflikt an. Da der Knoten K das Merkmal M zerstört, muß dieses Merkmal zwischen K und N neu erzeugt werden, dazu gibt es zwei Möglichkeiten. Entweder man fügt einen neuen Knoten in den Plan ein, der M erzeugt, oder aber man linearisiert den Plan so weiter, daß ein schon im Plan existierender Erzeuger von M zwischen K und N eingefügt wird. Formal bedeutet dies:

Auflösen von Linearkonflikten: Sei ein Linearkonflikt zwischen der Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ und dem Knoten K im Plan $\mathtt{P} = \langle \mathbf{K}, < \rangle$ gegeben. Der Konflikt ist in $\mathtt{P}' = \langle \mathbf{K}', <' \rangle$ gelöst mit:

1.) Auflösen durch Einsetzen eines neuen Knotens:
$\mathbf{K}' := \mathbf{K} \cup \{\mathtt{E}'\}$ mit $\mathtt{E}' \notin \mathbf{K}$ und $\mathtt{E}'$ ist Erzeuger von M.
$<'$ ist die transitive Hülle der Relation $< \cup \{\langle \mathtt{K}, \mathtt{E}' \rangle, \langle \mathtt{E}', \mathtt{N} \rangle\}$.
P ist in diesem Fall bis auf den Knoten $\mathtt{E}'$ wohlgeformt. **oder**

2.) Auflösen durch Einordnen eines bereits vorhanden Knotens:
Es gibt einen Knoten $\mathtt{E}' \in \mathbf{K}$ mit $\neg(\mathtt{E}' < \mathtt{E} \vee \mathtt{N} < \mathtt{E}')$, der M erzeugt.
$<'$ ist die transitive Hülle von $< \cup \{\langle \mathtt{K}, \mathtt{E}' \rangle, \langle \mathtt{E}', \mathtt{N} \rangle\}$.
$\mathtt{P}'$ ist in diesem Fall wohlgeformt.

Falls man einen neuen Knoten einsetzt, muß der resultierende Plan wieder zu einem wohlgeformten Plan erweitert werden, denn die Vorbedingungen des neuen Erzeugers von M müssen erst wieder erzeugt werden. Dadurch ist es dann im Gegenzug wieder möglich, daß andere Konflikte auftreten. Insbesondere können so weitere Linearkonflikte erzeugt werden.
Beim Gabelkonflikt gibt es zur Auflösung wiederum die zwei Möglichkeiten, weiter zu linearisieren – und zwar so, daß der Zerstörer K vor den Erzeuger E oder aber hinter den Nutzer N kommt – oder aber einen Erzeuger für M einzufügen, sei es durch Linearisieren, sei es durch Einfügung eines neuen Knoten. Formal:

Auflösen von Gabelkonflikten: Sei ein Gabelkonflikt zwischen der Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ und dem Knoten K im Plan $\mathtt{P} = \langle \mathbf{K}, < \rangle$ gegeben. Der Konflikt ist in $\mathtt{P}' = \langle \mathbf{K}', <' \rangle$ gelöst mit:

1.) Auflösen durch Linearisieren von E, N und K:
$\mathbf{K}' := \mathbf{K}$ mit
$<'$ ist die transitive Hülle von $< \cup \{\langle \mathtt{K}, \mathtt{E} \rangle\}$ falls $\mathtt{K} < \mathtt{N}$ und
$< \cup \{\langle \mathtt{N}, \mathtt{K} \rangle\}$ falls $\mathtt{E} < \mathtt{K}$.
$\mathtt{P}'$ ist in diesem Fall wohlgeformt. **oder**

2.) Auflösen durch Einfügen eines Erzeugers:
$\mathbf{K}' := \mathbf{K} \cup \{\mathtt{E}'\}$ mit $\mathtt{E}' \notin \mathbf{K}$ oder aber $\mathtt{E}' \in \mathbf{K}$ und $\neg(\mathtt{E}' < \mathtt{K} \vee \mathtt{N} < \mathtt{E}')$
$<'$ ist die transitive Hülle der Relation $< \cup \{\langle \mathtt{K}, \mathtt{E}' \rangle, \langle \mathtt{E}', \mathtt{N} \rangle\}$.
Falls $\mathtt{E}' \in \mathtt{P}$ ist $\mathtt{P}'$ wohlgeformt, sonst bis auf den Knoten $\mathtt{E}'$.

Um einen Parallelkonflikt aufzulösen, ergeben sich wieder die Möglichkeiten der Linearisierung oder aber des Einsetzens:

Auflösen von Parallelkonflikten: Sei ein Parallelkonflikt zwischen der Abhängigkeit $\mathtt{A} = \langle \mathtt{E}, \mathtt{M}, \mathtt{N}, \mathtt{P} \rangle$ und dem Knoten $\mathtt{K}$ im Plan $\mathtt{P} = \langle \mathbf{K}, < \rangle$ gegeben. Der Konflikt ist in $\mathtt{P}' = \langle \mathbf{K}', <' \rangle$ gelöst mit:

1.) Auflösen durch Linearisieren:
$\mathbf{K}' := \mathbf{K}$ mit
$<'$ ist die transitive Hülle von $< \cup \{\langle \mathtt{K}, \mathtt{E} \rangle\}$ falls $\mathtt{K} < \mathtt{N}$ oder aber von $< \cup \{\langle \mathtt{N}, \mathtt{K} \rangle\}$ falls $\mathtt{E} < \mathtt{K}$.
$\mathtt{P}'$ ist in diesem Fall wohlgeformt. **oder**

2.) Auflösen durch Einfügen eines Erzeugers und Linearisieren:
$\mathbf{K}' := \mathbf{K} \cup \{\mathtt{E}'\}$ mit $\mathtt{E}' \notin \mathbf{K}$ oder aber $\mathtt{E}' \in \mathbf{K}$ und $\neg(\mathtt{E}' < \mathtt{K} \vee \mathtt{N} < \mathtt{E}')$
$<'$ ist die transitive Hülle der Relation $< \cup \{\langle \mathtt{K}, \mathtt{E}' \rangle, \langle \mathtt{E}', \mathtt{N} \rangle\}$.
Falls $\mathtt{E}' \in \mathtt{P}$ ist $\mathtt{P}'$ wohlgeformt, sonst bis auf den Knoten $\mathtt{E}'$.

Zusammenfassend gibt es also die zwei Konfliktlösungsmöglichkeiten: Linearisieren (Verhindern, daß der Zerstörer von der Abhängigkeit umschlossen werden kann) oder aber Einfügung eines neuen Erzeugers zwischen Zerstörer und Nutzer.
Allerdings kann es auch noch weitere Möglichkeiten geben, zum Beispiel, wenn man Variable als Argumente in Plänen erlaubt, kann die Einschränkung von nichtlinearen Plänen auch über die Einschränkung der möglichen Variableneinsetzungen erfolgen.
Die Konfliktlösung insgesamt kann man sich also durch eine sukzessive Auflösung aller Konfliktknoten vorstellen. Jeder dieser Knoten fällt in eine der drei oben angeführten Klassen und für diese weiß man, wie man sie aufzulösen hat. Dies kann allerdings sehr umständlich sein, so daß kombinierte Auflösungen angemessener wären. Erschwerend kommt noch hinzu, daß die Reihenfolge der Auflösungen der Konflikte nicht gleichgültig ist.
Bevor wir uns nun wieder dem Beispiel aus Abbildung 5.7 zuwenden, wollen wir noch kurz auf den letzten Punkt aus dem Fragment des Algorithmus zum nichtlinearen Planen aus Abbildung 5.5 eingehen, den Kritiken.

5.4 Kritiken

Kritiken sind eine kritische Betrachtung des Resultats eines Planungsprozesses. Zum Beispiel könnte man den Punkt „Auflösung von Interaktionen“ auch unter dem Punkt Kritiken subsumieren; wegen der Wichtigkeit der Interaktionsauflösung im nichtlinearen Planen ist es jedoch durchaus angemessen, daraus einen eigenen Punkt zu machen.
Die wichtigsten Kritiken dienen der Betrachtung des Planes auf Ineffizienzen hin. Zum Beispiel trägt ein Operator, von dem keine Abhängigkeiten ausgehen, nichts zum Plan bei, er kann also gelöscht werden und braucht folglich nicht ausgeführt zu

werden. Eine solcher Operator kann in einem Plan enthalten sein, wenn Abhängigkeiten, die ursprünglich von ihm ausgegangen sind, nach Linearisierungen von anderen Operatoren ausgehen.
Es gibt zwei weitere Arten von Ineffizienzen, hilfreiche Interaktionen und Umwege. Bei hilfreichen Interaktionen, kann ein Teilziel (oder ein Teil von ihm) gelöst werden, während man die Lösung eines anderen Teilziels verfolgt. Erkennt man dies nicht, so würde man ineffizienterweise (und meist sogar fälschlicherweise) die entsprechende Arbeit doppelt machen. Bei Umwegen kann es geschehen, daß gewisse inverse Operatoren unmittelbar hintereinander vorkommen (wie bei der Sussman-Anomalie, vergleiche Seite 29), wie zum Beispiel (STACK(A, B), UNSTACK(A, B)) oder (PICKUP(C), PUTDOWN(C)). Man kann natürlich solche inversen Operatoren aus einem Plan streichen. Solche Situationen können auch beim nichtlinearen Planen durch Linearisierung entstehen. Dieses Beispiel ist ziemlich einfach, im allgemeinen ist das Erkennen und die Beseitigung von Umwegen alles andere als trivial.

5.5 Beispiel

Betrachten wir nun nochmals das Beispiel aus Abbildung 5.7 und das Zwischenergebnis aus Abbildung 5.8, die wir hier wiederholen:

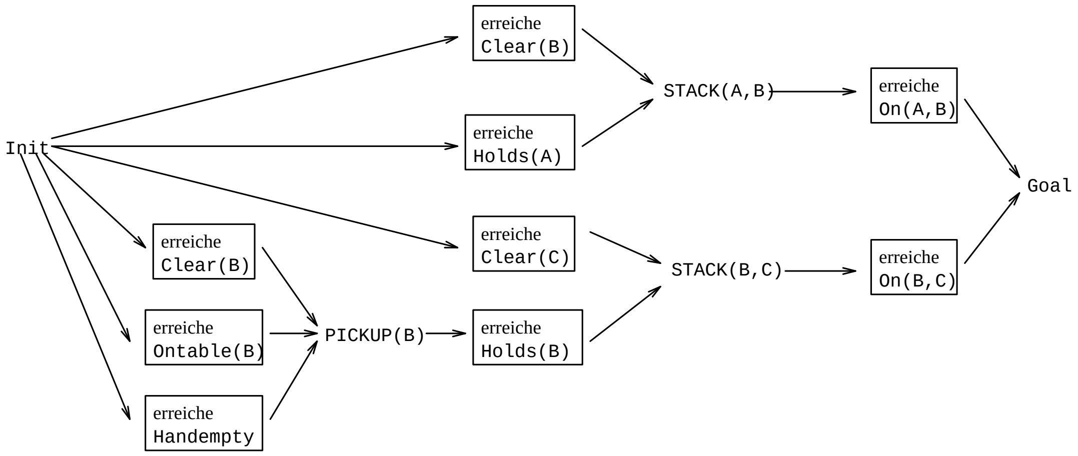


Abbildung 5.8: Konfliktauflösung

Wie bereits auf Seite 43 erwähnt, gibt es einen Konflikt zwischen dem Knoten PICKUP(B) und der Abhängigkeit A = ⟨Init, Clear(B), STACK(A, B), P⟩. Entsprechend gibt es den Konflikt zwischen dem Knoten STACK(A,B) und der Abhängigkeit A′ = ⟨Init, Clear(B), PICKUP(B), P⟩. Es handelt sich in diesem Fall um zwei Gabelkonflikte, da der Erzeuger (das heißt der Knoten Init) vor den parallelen Knoten N und K steht, das heißt in diesem Fall vor STACK(A,B) und PICKUP(B), beziehungsweise die beiden Knoten umgekehrt.
Wie kann man diese Konflikte nun auflösen? Wie wir gesehen haben, gibt es dafür wieder viele verschiedene Lösungen. Eine wäre, PICKUP(B) hinter STACK(A,B) einzuord-

nen (womit der erste Konflikt gelöst wäre) und dann zur Lösung des zweiten Konflikts `UNSTACK(A,B)` einzufügen. Allerdings erzeugt man dadurch einen weiteren Konflikt zwischen `STACK(A,B)` und `On(A,B)`. Auch diesen kann man lösen, aber man erhält erst mal einen suboptimalen Plan, der ein Teilstück `(STACK(A,B),UNSTACK(A,B))` enthält, das dann durch eine Kritik wieder entfernt wird.
Besser wäre es gewesen, `STACK(A,B)` hinter `STACK(B,C)` zu linearisieren und die Abhängigkeit ⟨`Init`, `Clear(B)`, `STACK(A,B)`, `P`⟩ durch eine andere Abhängigkeit, und zwar durch ⟨`STACK(B,C)`, `Clear(B)`, `STACK(A,B)`, `P`⟩, zu ersetzen.
Dann stellt sich natürlich sofort die Frage, wie kann man algorithmisch erreichen, daß man diese bessere Lösung findet. Leider gibt es hierauf keine allgemeine Antwort. Auch die Konfliktauflösung ist ein Problem, das im allgemeinen nur mit guten bereichsabhängigen Heuristiken angegangen werden kann.
Zum Abschluß des Kapitels über nichtlineares Planen, wollen wir kurz einen Vergleich zum linearen Planen anstellen.

5.6 Lineares versus nichtlineares Planen

Ein Hauptvorteil des nichtlinearen Planens im Vergleich zum linearen Planen liegt in der größeren Flexibilität in der Darstellung von Plänen: Ein nichtlinearer Plan repräsentiert in der Regel eine Menge von linearen Plänen, wodurch man unter Umständen nicht so oft rücksetzen muß wie beim linearen Planen. In der Regel geben lineare Pläne eine überspezifizierte Lösung an, gewisse Reihungen innerhalb der Operatoren sind notwendig, andere hingegen sind zufällig. Mit nichtlinearen Plänen ist es hingegen möglich, die Einschränkung an die Reihenfolge der Operatoren so schwach wie möglich zu spezifizieren, was einen großen Vorteil bei der Ausführung von Plänen bedeuten kann; denn unter dieser Voraussetzung ist es möglich bei der Planausführung flexibel zu entscheiden, welcher Planteil zuerst ausgeführt werden soll. Zudem ermöglichen nichtlineare Pläne oft eine parallele Ausführung, was in gewissen Anwendungsdomänen von Interesse ist.
Man darf aber bei all den Vorteilen nicht vergessen, daß nichtlineare Pläne auch besondere Probleme machen. Insbesondere ist das nichtlineare Planen nicht so einfach wie lineares Planen.
Beide Planungsansätze, lineares wie nichtlineares Planen, funktionieren natürlich dann besonders gut, wenn es sich um nicht oder aber nur schwach interagierende Anwendungsbereiche handelt.

Kapitel 6

Hierarchisches Planen

Ein Hauptproblem des Planens besteht in der Beherrschung der auftretenden Komplexität bei der Planerstellung. Dies ist ja zum Beispiel auch schon der Grund, daß man zur Planerstellung nicht einfach eine Breitensuche durchführt, sondern sich für die komplizierteren Ansätze des linearen Planens in STRIPS durch means-end-analysis (vergleiche Abbildung 4.4), means-end-analysis mit Teilzielregression (Abbildung 4.10) und des nichtlinearen Planens (Abbildung 5.5) entscheidet. Dennoch hat man mit diesen Ansätzen das Komplexitätsproblem nicht gelöst, so daß viele Probleme mit diesen Verfahren nicht sinnvoll bearbeitet werden können. Ein Hauptgrund für die weiterhin bestehende Komplexität beruht auf der unterschiedslosen Behandlung von wichtigen und weniger wichtigen Teilzielen. Wenn ein Mensch eine komplexe Aufgabe zu lösen hat, wird er seine Planung dadurch strukturieren, daß er Operatoren wichtet oder aber in der Beschreibung wichtige und weniger wichtige Merkmale verschieden behandelt. In allen bislang beschriebenen Verfahren (den deduktiven wie den auf STRIPS aufbauenden) geht das jedoch nicht.
In diesem Kapitel werden wir sehen, wie man das nichtlineare Planungsverfahren NLP, aber auch STRIPS, erweitern kann, so daß hierarchisches Planen ermöglicht wird. Dies wird zwar die Komplexitätsprobleme des Planens nicht allgemein lösen, aber viele Probleme in den Bereich einer automatischen Lösung rücken.

6.1 Planen mit Situationsabstraktion

Die ursprüngliche Idee zur Einführung von Abstraktion in STRIPS ist in [Sac74] beschrieben. Es wird dort auch ein System vorgestellt, nämlich ABSTRIPS (Abstraction-Based STRIPS), das in Experimenten ein erheblich besseres Komplexitätsverhalten zeigt. Die Hauptidee besteht darin, daß man die in den Operatoren auftretenden Literale (Elementarmerkmale) gewichtet, in einem ersten Planungsdurchgang nur die wichtigsten betrachtet, im nächsten diesen abstrakten Plan dadurch verfeinert, daß man auch die zweitwichtigsten berücksichtigt, und so weiter. Sehen wir uns das an Hand eines Klötzchenweltbeispiels aus [Her89, S.132 ff] näher an.
Die Merkmale seien wie folgt gewichtet:

On	4
Clear	3
Holds	2
Ontable	2
Handempty	1

Größere Zahlen stehen dabei für wichtigere Merkmale. Diese Gewichtung zieht sich auf die Operatoren durch, daß heißt die normalen STRIPS-Operatoren werden nun durch solche ersetzt, in denen die Gewichte auch in den Vorbedingungen der Operatoren vermerkt sind.

PICKUP(x) **Precondition** $\mathtt{Clear}^3$(x) ∧ $\mathtt{Ontable}^2$(x) ∧ $\mathtt{Handempty}^1$ **delete list** Clear(x) ∧ Ontable(x) ∧ Handempty **add list** Holds(x)	PUTDOWN(x) **Precondition** $\mathtt{Holds}^2$(x) **delete list** Holds(x) **add list** Clear(x) ∧ Ontable(x) ∧ Handempty
STACK(x, y) **Precondition** $\mathtt{Holds}^2$(x) ∧ $\mathtt{Clear}^3$(y) ∧ **delete list** Holds(x) ∧ Clear(y) **add list** Clear(x) ∧ On(x, y) ∧ Handempty	UNSTACK(x, y) **Precondition** $\mathtt{Clear}^3$(x) ∧ $\mathtt{On}^4$(x, y) ∧ $\mathtt{Handempty}^1$ **delete list** Clear(x) ∧ On(x, y) ∧ Handempty **add list** Holds(x) ∧ Clear(y)

Abbildung 6.1: ABSTRIPS-Operatoren zur Klötzchenwelt

Wir wollen uns nun ansehen, wie man den Algorithmus des nichtlinearen Planens aus Abbildung 5.5 auf das abstrakte Planen übertragen kann.

```
begin
0.    if Index=0 then return(P)
1.    else
1.1     blende alle Vorbedingungen von Operatoren aus,
        deren Gewicht kleiner als Index ist.
1.2.    wähle P':=NLP(P)
        ;;; da beliebige Auswahl → Rücksetzpunkt
      endif
1.3.    return(NLPSA(P',Index-1))
end
```

Abbildung 6.2: Algorithmus NLPSA (nichtlineares Planen mit Situationsabstraktion)

Das Programm erhält als Eingabe einen nichtlinearen Plan `P` und einen Index `Index`. Initial ist der nichtlineare Plan trivial und der Index das maximal vorkommende Gewicht. Das Programm setzt voraus, daß die Merkmale in den Operatoren, sowie die Operatoren selbst wie oben mit Indizes gewichtet sind.[1]
Ausblenden von Vorbedingungen bedeutet, daß man die entsprechenden Bedingungen bei der Berechnung, ob die Vorbedingung erfüllt ist, einfach ignoriert. Damit man einen so gefundenen Plan auf der nächsten Stufe weiterverwenden kann, bleiben die Merkmale allerdings in **add list**, **delete list** und Zielbeschreibung unverändert.
Ein Mangel dieser Art von Planung besteht darin, daß ein Zurücksetzen im Falle des Backtrackings in Schritt 1.2 insoweit blind erfolgt, als der nichtlineare Planer keine Information darüber erhält, warum die von ihm präsentierte Lösung nicht genehm war.
Sehen wir uns jetzt mal an einem Beispiel an, wie die Planung erfolgt. Gegeben sei das folgende Problem, das wir schon aus Abbildung 5.7 kennen:

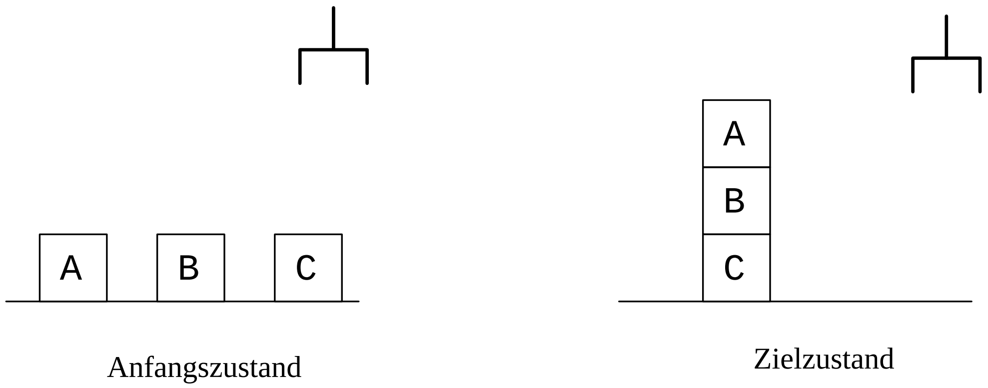


Abbildung 6.3: Beispiel abstraktes Planen

Auf Stufe 4 wir nur das Merkmal `On` berücksichtigt, also werden die Vorbedingungen im einzigen Operator, der `On` in der **add list** hat, alle nicht berücksichtigt, und NLP erzeugt einfach die folgende Lösung:

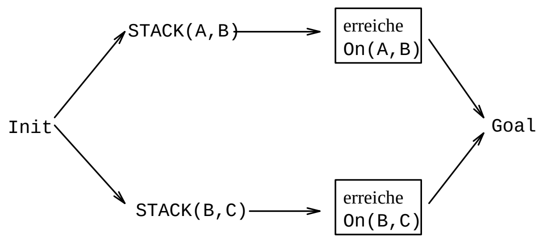


Abbildung 6.4: Zwischenergebnis Gewicht 4

[1] A priori ist es also möglich, daß in verschiedenen Operatoren dasselbe Merkmal verschieden gewichtet wird.

Dieser Plan ist im rekursiven Aufruf auf der Stufe 3 zu verfeinern. Wir sind nicht fertig, da Vorbedingungen weggelassen wurden. Auf der Stufe 3 sind nun also die Vorbedingungen mit Gewicht 3, das heißt die `Clear` zu berücksichtigen. Da diese initial erfüllt sind, werden nur entsprechende „erreiche"-Knoten zwischen `Init` und den `STACK`-Knoten eingefügt. Auf der Stufe 2 kommen nun die Merkmale `Holds` und `Ontable` hinzu. Entsprechend müssen die erweiterten Vorbedingungen für den Operator `STACK` jeweils erfüllt werden, ebenso müssen die von `PICKUP` erfüllt werden. Weiterhin muß eine Konfliktlösung (hier durch weitere Linearisierung) erfolgen.

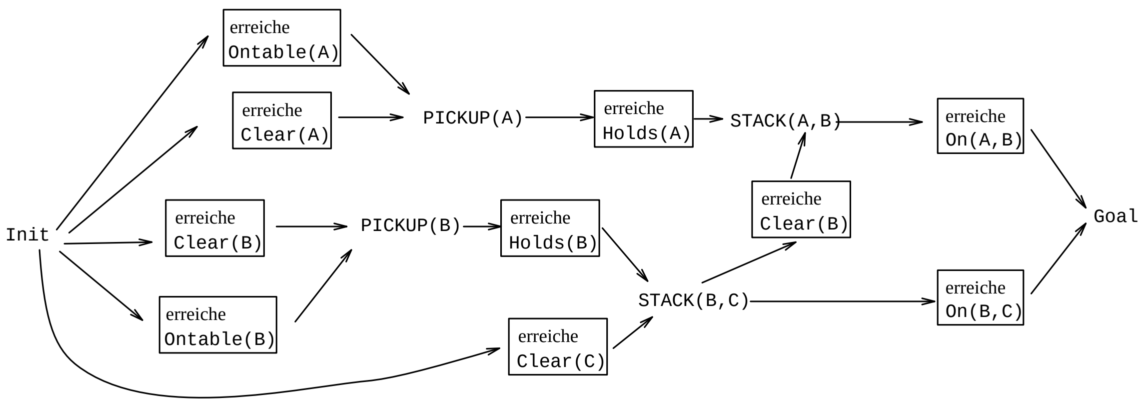


Abbildung 6.5: Zwischenergebnis Gewicht 2

Auf der Stufe 1 kommt nun noch das Prädikat `Handempty` hinzu. Das heißt es ergibt sich nach Konfliktlösung die folgende Lösung des Planungsproblems:

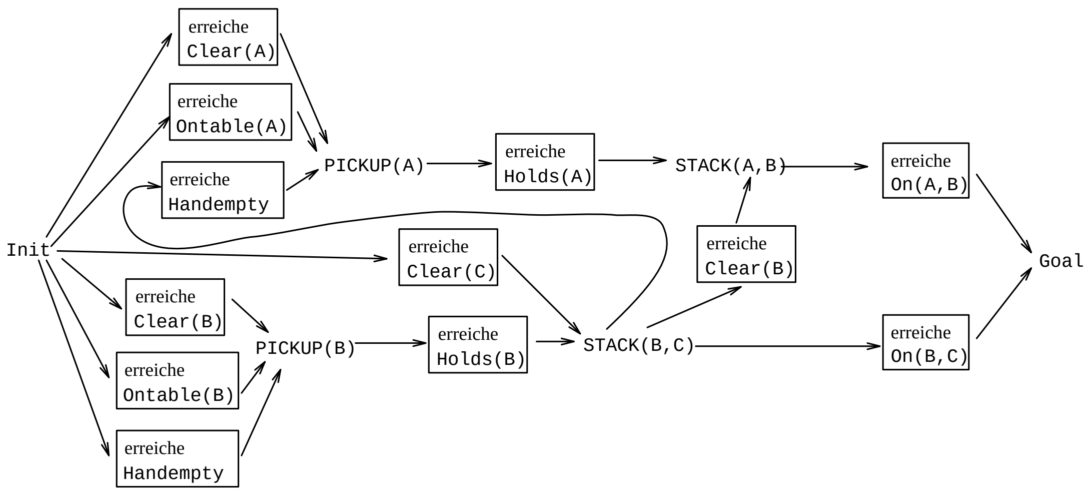


Abbildung 6.6: Endergebnis

Ein Problem, das bei dieser Art des abstrakten Planens auftreten kann, beruht auf den Konfliktlösungsstrategien des nichtlinearen Planens. Es kann ja erforderlich werden, daß neue Knoten in den abstrakten Plan eingefügt werden müssen. Dabei kann aber der Fall eintreten, daß man erst auf der konkreteren Ebene sieht, welcher

Knoten auf der abstrakten Ebene am besten eingefügt werden sollte. Zum Beispiel, wenn man `Holds(A)` erreichen muß, gibt es ja zwei Möglichkeiten `UNSTACK(A,x)` und `PICKUP(A)`. Welcher soll ausgewählt werden? Das hängt natürlich von der Situation ab. Steht `A` zum Beispiel auf dem Tisch, so muß ein `PICKUP(A)` gemacht werden, was man aber auf einer abstrakteren Stufe gar nicht weiß, da das Prädikat `Ontable` dort unsichtbar ist. Für dieses Problem gibt es im wesentlichen die folgenden Lösungen:

- Man kümmert sich nicht darum, sondern nimmt ineffiziente Pläne in Kauf.
- Man schaut auf niedrigere Stufen voraus. (Dies ist natürlich von der Idee des abstrakten Planens her nicht schön.)
- Man läßt den Konflikt auf der höheren Stufe erst einmal stehen und löst ihn erst später auf, wenn hinreichend viel Information von den niedrigeren Stufen angesammelt ist. Dazu muß allerdings der NLP-Algorithmus angepaßt werden, so daß er mit offenen Konflikten umgehen kann.

Der Erfolg der Planung mit Situationsabstraktion hängt natürlich wesentlich von der Wahl der richtigen Gewichtung der Operatoren ab, so ist in unserem Beispiel klar, daß die Eigenschaft `Handempty` relativ unwichtig ist, da sie sich praktisch von Schritt zu Schritt ändert. Hingegen ist `On` eine wichtige Eigenschaft, zum Beispiel ist sie wesentlich zur Beschreibung komplizierterer Situationen. Von daher macht es Sinn, `Handempty` mit 1 und `On` mit 4 zu gewichten. Allerdings ist die Frage, ob `Holds` und `Ontable` dieselbe Wichtigkeit haben oder nicht, nicht so klar zu beantworten.

Für das abstrakte Planen ist eine „richtige“ Entscheidung für Operatoren auf der abstrakten Ebene natürlich viel wichtiger als im nichtabstrakten Planen; macht man nämlich einen Umweg auf hoher Ebene, sagen wir von fünf Schritten, so müssen diese fünf Schritte weitergeplant werden und können auf der konkreten Ebene beliebig aufwendig werden. Andererseits kann es natürlich auch passieren, daß auf den konkreteren Stufen nichts Wesentliches mehr passiert und man von daher unütze Berechnungen durchführen muß. Deshalb hat Sacerdoti, obwohl sein System auf STRIPS aufbaut, nicht das Standardprogramm MEA hergenommen, sondern es so erweitert, daß Umwege im abstrakteren Bereich vermieden werden. Bei hinreichend geschickter Abstraktion ist für gewisse Problemklassen durchaus denkbar, das abstraktes Planen mit Breitensuche betrieben werden könnte (was den Vorteil hätte, daß man garantiert kürzeste Pläne erhält).

Die These, daß Abstraktion eine wesentliche Rolle beim menschlichen Problemlösen spielt, ist wohl unbestritten. Allerdings scheint die Frage weitgehend offen, wie man dies umfassend in den Bereich Planung übertragen kann. Insbesondere stellt sich bei den verschiedenen Problemen, die man mit dieser Art von Abstraktion hat, die Frage, inwieweit beziehungsweise für welche Problemklassen sie günstig ist. Sacerdoti zieht in [Sac74] eine positive Bilanz, die er in Experimenten belegt hat. Der Grund, weshalb ABSTRIPS im Verglich zu STRIPS besser abschneidet, liegt nach seinen Ergebnissen daran, daß durch die Suchabstraktion erreicht wird, daß der Suchbaum weniger buschig wird. Man sucht zwar noch, aber nur auf plausiblen Pfaden (was gleichbedeutend ist mit, nur auf Pfaden, die auf höherer Ebene schon erzeugt sind). Sacerdoti nennt dies length-first search (Längensuche).

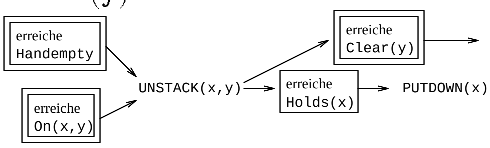
6.2 Planen mit Operatorabstraktion

Eine wichtige Frage, die man für konkrete Planungsaufgaben beantworten muß, ist die Ebene, auf der die Operatoren angegeben sind. Zum Beispiel beim Planen in der Klötzchenwelt haben wir die vier Operatoren `PICKUP`, `PUTDOWN`, `STACK` und `UNSTACK` kennengelernt. Man kann sich aber auch noch detailliertere Operatoren vorstellen, zum Beispiel solche, bei denen der Operator `PICKUP` in drei Teile aufgespalten wird, in denen durch einen Operator `POSITION` erst einmal die Roboterhand positioniert wird, dann wird das Objekt mit einem Operator `CATCH` gegriffen und schließlich mit einem Operator `LIFT` hochgehoben. Diese Operatoren könnten dann wiederum näher spezifiziert werden. Von dieser Warte aus ist der Operator `PICKUP` eine Operatorabstraktion der Operatoren `POSITION`, `CATCH` und `LIFT`.
Auf der anderen Seite kann man die bis jetzt betrachteten Operatoren `PICKUP`, `PUTDOWN`, `STACK` und `UNSTACK` aber auch weiter abstrahieren. Dies wollen wir im folgenden tun. Dazu definieren wir einen Operator `FREE`, der die Operatoren `PICKUP` und `STACK` abstrahiert.

FREE(x, y)	
Precondition	Handempty $\wedge$
	On(x, y)
delete list	On(x, y)
add list	Clear(y)
plot	(UNSTACK(x, y), PUTDOWN(x))

Sehen wir uns nun diesen Operator genauer an, er hat im ersten Eintrag Vorbedingungen **Precondition**, dann eine **delete list** und eine **add list**, so daß er genau wie die bisher in STRIPS aber auch im nicht-linearen Planen betrachteten Operatoren zum Planen benutzt werden kann. Ein Unterschied zu den bisher für die Klötzchenwelt betrachteten Operatoren besteht darin, daß nicht alle in der Vorbedingung auftretenden Bedingungen durch **delete list** gelöscht werden, in diesem Fall ist es für die Anwendbarkeit von `FREE` erforderlich, daß die Roboterhand leer ist, aber nach der Ausführung ist sie das noch immer.
Wirklich neu ist nur der letzte Eintrag **plot**, der besagt, daß man mit der Einfügung von diesem Operator noch nicht fertig ist, sondern ihn – wenn man mit der abstrakten Planung fertig ist – noch expandieren muß und zwar das in diesem Eintrag stehende Planfragment. Im Unterschied zum bisherigen Planen müssen die abstrakten Operatoren auf eine bestimmte Plandatenstruktur abgestimmt sein, in diesem Fall auf das Format einer Liste, die STRIPS als Ergebnis einer Planung liefert. Für das nichtlineare Planen erhält man als Struktur den Plan als Operatormenge mit einer partiellen Ordnung, also informell als die Struktur:

FREE(x, y)

Precondition	Handempty $\wedge$
	On(x, y)
delete list	On(x, y)
add list	Clear(y)
plot	

Die in den Plan hinein- und herausgehenden Abhängigkeiten sind markiert und werden durch **Precondition** und **add list** gekennzeichnet. Eine wichtige Eigenschaft beim abstrakten Planen besteht darin, daß man die Vor- und die Nachbedingungen nicht genau spezifizieren muß. So ist im vorliegenden Beispiel nicht spezifiziert, daß als Nachbedingung der Operatoranwendung `FREE(x,y)` stets `Ontable(x)` gilt. Dies kann man deshalb machen, da der Ansatz der Planung mit Operatorabstraktion vorsieht, daß nach der Erstellung eines abstrakten Planes, dieser konkretisiert wird und auf der konkreten Ebene für die entsprechenden Operatoren wieder geprüft wird.
Eine Hauptfrage, die man bei der Definition von abstrakten Operatoren beantworten muß, ist der Grad der Abstraktion, den man wählen soll. Auf der einen Seite ist es ja gut, daß man gewisse unwichtige Details vergißt – zum Beispiel, daß ein Klotz als Nebeneffekt der Anwendung von `FREE` auf dem Tisch landet. Auf der anderen Seite kann gerade dies in konkreten Planungsaufgaben ärgerlich sein, da es auf abstrakter Ebene zu umständlichen Plänen führen kann. Zum Beispiel wenn `On(A,B)` gilt und man zur Erreichung von `Clear(B)` den abstrakten Operator `FREE(A,B)` anwendet, dann müßte man – wenn man zusätzlich `Ontable(A)` haben will – fälschlicherweise einen Teilplan zur Erzeugung von `Ontable(A)` entwerfen. Dies liegt daran, daß man nach der Ausführung des abtrakten Operators `FREE(A,B)` nicht mehr weiß, welche Zustandsbeschreibung für `A` gilt. Ein Ausweg aus dieser Situation besteht darin, abzuwarten, bis der Operator `FREE(A,B)` soweit konketisiert ist, daß man weiß, was mit `A` passiert ist. Dies läuft allerdings der Idee der Abstraktion zuwider.
Eine andere Möglichkeit, wie abstrakte Operatoren zu nichtoptimale Pläne führen, sei durch das folgende Beispiel gegeben:

Abbildung 6.7: Nichtoptimalität abstrakter Operatoren

Das Beispiel sei wie folgt beschrieben:

Init	Goal
Ontable(B)	On(A,C)
On(A,B)	Ontable(C)
Clear(A)	Clear(A)
Clear(C)	Clear(B)
Handempty	Handempty

Will man nun zuerst das Ziel `Clear(B)` erreichen und wählt dazu den abstrakten Operator `FREE(A,B)` aus, so landet bei der Ausführung das Klötzchen `A` auf dem Tisch, um im nächsten Schritt auf `C` gestellt zu werden. Linearisiert ergibt sich also – gehen wir mal davon aus, daß `FREE` der einzige neue Operator ist – der Plan `(FREE(A,B),PICKUP(A),STACK(A,C))`, der nach der Expansion von `FREE(A,B)` – was problemlos geht – im Plan `(UNSTACK(A,B),PUTDOWN(A),PICKUP(A),STACK(A,C))` mündet. Dieser Plan enthält den Umweg `(PUTDOWN(A),PICKUP(A))`.

Nun sehen wir uns an, wie man den Algorithmus zum nichtlinearen Planen erweitern muß, damit auch abstrakte Operatoren behandeln kann.

begin
1. **while** noch Knoten ungelöst oder unexpandiert in `P` **do**
1.1. wähle Knoten `K` aus, der in `P` ungelöst oder unexpandiert
;;; da beliebige Auswahl → Rücksetzpunkt
1.2 **if** `K` ist zum Lösen ausgewählt
then
1.2.1 wähle Instanz `I` eines Operators aus, die das in
`K` als zu erreichend geforderte Merkmal in der **add list** hat
;;; da beliebige Auswahl → Rücksetzpunkt
1.2.2 füge `I` als Operatorknoten direkt vor `K` in `P` ein
füge Erreiche-Knoten für die Vorbedingungen von `I` untereinander
angeordnet direkt vor `I` ein
else
1.2.3 ersetze `K` durch seinen **plot** und ergänze Vorbedingungen
end_if
1.3. löse Interaktionen auf
1.4. kritisiere den Plan
end_do
2. **return**(P)
end

Abbildung 6.8: Algorithmus NLPOA (nichtlineares Planen mit Operatorabstraktion)

Eingegeben wird wieder ein initialer nichtlinearer Plan, Ausgabe ist ein nichtlinearer Plan (modulo einer fixen Menge von Operatoren).
Neu hinzugekommen ist also die Expansion von Knoten. Entscheidet man sich einen Knoten zu expandieren – nach diesem Algorithmus gibt es a priori keine Trennung in

eine Phase, in der abstrakt geplant wird und dann eine Phase, in der die abstrakten Operatoren expandiert werden – so wird er einfach durch seinen **plot** ersetzt. Eleganter wäre eine Lösung, bei der man bei der Expansion eines abstrakten Operators, diesen nur als gelöst markiert, aber im Plan beläßt, damit man den Plan zum Beispiel einfacher erklären kann oder auch besser zurücksetzten kann.
Sehen wir uns nun den Algorithmus anhand der Sussman-Anomalie an.

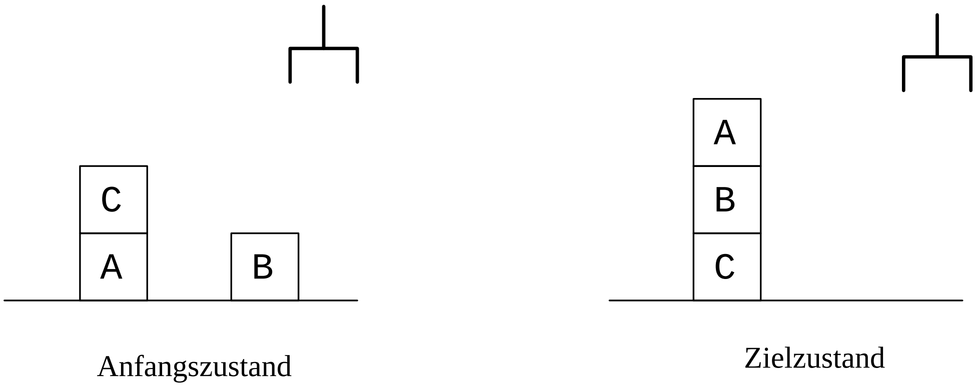
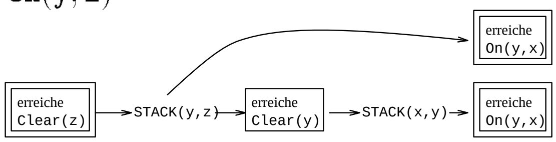


Abbildung 6.9: Die Sussman-Anomalie

Wir gehen davon aus, daß wir zusätzlich zu den vier altbekannten Opertoren den abstrakten Operator `FREE(x,y)` und einen Operator `STACKTHREE(x,y,z)` haben, mit dem man drei Klötzchen aufeinanderstellen kann und der die folgende Gestalt hat:

`STACKTHREE(x, y, z)`

Precondition	`Clear(z)`
delete list	`Clear(z)`
add list	`On(x, y)` `On(y, z)`
plot	



Um das Ziel zu lösen kann man nun einfach den Operator `STACKTHREE(A,B,C)` aufrufen. Der Operator löst das Ziel, da er `On(A,B)` und `On(B,C)` in der **add list** hat. Schon etwas merkwürdiger ist die Anwendbarkeit: er ist anwendbar, da `Clear(C)` gilt. Im nächsten Durchlauf durch die Schleife des Algorithmus bleibt nur die Expansion des abstrakten Operators wie oben angegeben, das heißt wir haben nun einen Teilplan der Form:

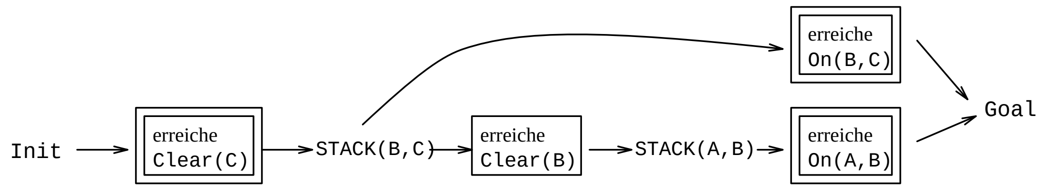


Abbildung 6.10: Teilplan zur Sussman-Anomalie

Im nächsten Schritt müssen nun die Vorbedingungen der expandierten Operatoren betrachtet werden. Dies sind zu `STACK(B,C)` die drei Bedingungen `Handempty`, `Clear(C)` und `Holds(B)`, analog für `STACK(A,B)` die drei Bedingungen `Handempty`, `Clear(B)` und `Holds(A)`. Um die Vorbedingungen `Holds(x)` zu erfüllen, wird sinnvollerweise jeweils der Operator `PICKUP(x)` mit `x=A` beziehungsweise `x=B` ausgewählt, was wiederum unter anderem die Bedingungen `Clear(A)` und `Clear(B)` nach sich zieht, wovon die erste mit `FREE(C,A)` erfüllt werden kann, wohingegen die zweite initial gilt. Das heißt wir haben die Situation:

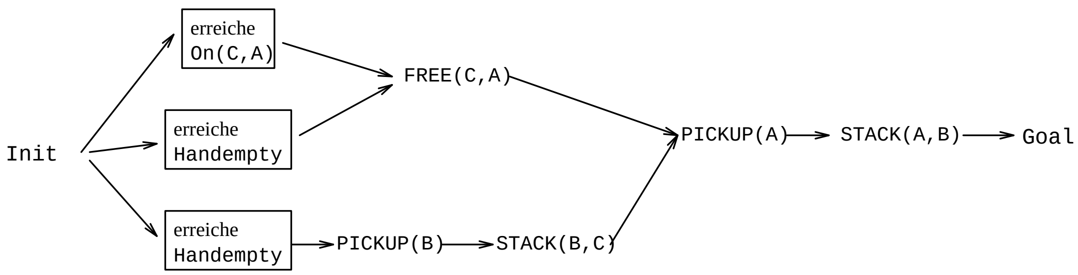


Abbildung 6.11: erweiterter Teilplan zur Sussman-Anomalie

In diesem Plan sind der Übersichtlichkeit halber nur einige Erreiche-Knoten eingezeichnet. Was man aber auch so sieht, ist der Konflikt zwischen den Operatoren `FREE(C,A)` und `PICKUP(B)` bezüglich der Eigenschaft `Handempty`. Es stellt sich also die Frage, wie man diesen Konflikt auflösen kann. Dies kann durch eine Linearisierung geschehen, in dem man entweder `FREE(C,A)` vor oder aber hinter dem parallelen Planteil einordnet. In diesem Fall wäre es natürlich sinnvoll, den `FREE(C,A)`-Operator nach vorne zu verlagern, aber auf der abstrakten Ebene ist es nicht möglich, zu sehen, daß dies die bessere Entscheidung ist; denn man sieht auf der abstrakten Ebene die versteckte Voraussetzung `Clear(C)`, die aus dem `UNSTACK`-Operator kommt, nicht. Man kann dies erst nach der Expansion von `FREE(C,A)` erkennen. In diesem Fall wäre es also besser gewesen, die Konfliktauflösung aufzuschieben und erst nach der Expansion der abstrakten Operatoren durchzuführen.
Abschließend kann man sagen, daß die Einführung abstrakter Operatoren sicherlich von großem Nutzen sein kann, aber in der Praxis auch viele zusätzlichen Schwierigkeiten mit sich bringt.

Kapitel 7

Planen mit äußeren Ereignissen

Die bislang betrachteten Planverfahren gingen wie am Anfang der Vorlesung gesagt (vgl. S. 1) von verschiedenen Vereinfachungen aus aus:

Konstanz der Welt: Der Planer ist der einzige Agent in der Welt, das heißt die externe Welt bleibt während des Planungsprozesses unverändert, nur durch die Planausführung darf sich etwas ändern.

vollständige Information: Ein Planungsproblem ist anfangs vollständig durch einen Anfangszustand, einen Zielzustand und eine Menge von Operatoren gegeben. Der Plan soll angeben, wie man vom Anfangszustand mithilfe der Operatoren zum Zielzustand gelangen kann.

Interaktionsfreiheit: Bislang ließen sich alle Pläne – wenn sie nicht schon linear gegeben waren – linearisieren, also durch eine Sequenz von Operatoren ausdrücken. Das heißt eine Interaktion von mehreren Operatoren war in Plänen nicht erforderlich

Die beiden ersten Annahmen, Konstanz der Welt und vollständige Information zu Beginn der Planung, sind wesentlich für alle bislang vorgestellten Planungsverfahren. Die letzte dieser drei Annahmen wurde zumindest teilweise im Zusammenhang mit linearem versus nichtlinearem Planen diskutiert. Unter Nichtlinearität kann (vgl. S. 38) kann man ja auch die parallele Durchführung paralleler Zweige eines Planens verstehen, was eng verwandt mit der von uns angenommen starken Nichtlinearität ist. Allerdings gibt es in diesem Zusammenhang auch subtilere Formen von Interaktionen, zum Beispiel kann es zum Hochheben eines Tisches erforderlich sein, daß zwei Operationen parallel ausgeführt werden, nämlich das Hochheben an einem Ende und das Hochheben am anderen Ende.
In realen Anwendungen steht einem Planer oft keine vollständige Information zur Verfügung, vielmehr muß er sich mit einem eingeschränkten Blick auf die Welt zufrieden geben. Was kann dann aber noch ein Plan sein? Der Plan kann dann Abfragen enthalten, die bei der Ausführung je nach augenblicklichen Weltzustand entscheiden,

welche Zweige des Planes ausgeführt werden sollen. Zum Beispiel könnte eine solche Anweisung lauten: „Wenn die auszuführende Aktion das Bezahlen eines Gegenstandes ist und die Kosten den Bargeldvorrat übersteigen, dann soll mit Kreditkarte bezahlt werden, andenfalls bar."

$$\mathtt{Try(Pay(Sum,I))} \Rightarrow \textbf{if } \mathtt{Sum} > \mathtt{Cash} \textbf{ then } \mathtt{PayCard(Sum,I)} \textbf{ else } \mathtt{PayCash(Sum,I)}$$

Eine andere Möglichkeit besteht darin, die Planungsschritte zurückzustellen bis für die Planung hinreichend viel Information zur Verfügung steht.
Sehen wir uns nun externe Vorgänge etwas näher an. Es gibt Ereignisse, die der planende Agent nicht beeinflussen kann, wie zum Beispiel das Wetter, es gibt Ereignisse, die durch eine Präventionsmaßnahme verhindert werden können, wie das Eindringen in einen Raum durch das Abschließen der Tür, oder aber Ereignisse, die durch eine Aktion ermöglicht werden.
Im folgenden wollen wir nun Probleme und eine mögliche Lösung betrachten, die eine Erweiterung des Planungsparadigmas betrachten. Dazu werden wir einen deduktiven Planungsansatz verfolgen, der auf einer Intervallogik basiert. Der Ansatz ist aus [Pel91] übernommen. Im Gegensatz zu STRIPS und den verwandten Verfahren sind entsprechende Fragen erst am Anfang einer Erforschung und von daher sind die Lösungsansätze noch nicht allgemein als die bestmöglichen akzeptiert.

7.1 Intervallogik und ihre Grenzen

Im ersten Kapitel haben wir Planen mit Deduktion kennengelernt. Dort haben alle Zustandsbeschreibungen und alle Operatoren ein zusätliches Argument ξ, das den Zustand angibt. In erster Näherung kann man sich eine Intervallogik so vorstellen, daß dieser Zustand durch ein Intervall $\mathtt{I}$ gegeben wird, das angibt, in welchem Zeitbereich die Beschreibung zutrifft. Zum Beispiel kann man die Regel: „Wenn der Wind bläst während man versucht zu segeln, dann bewegt sich das Boot die ganze Zeit." durch die folgende Formel beschreiben:

$$\mathtt{Try(Sail(I))} \land \mathtt{WindBlowing(I_1)} \land \mathtt{I} \subseteq \mathtt{I_1} \Rightarrow \mathtt{BoatMoving(I)}$$

Durch die Verwendung einer Erweiterung dieser Logik ist es nun möglich auch externe Ereignisse zu modellieren. Wir werden darauf später zurückkommen.
Was ist nun ein Plan in dieser Logik. Die Welt wird durch Weltbeschreibungsformeln gegeben, zusammengefaßt in einer Menge $\mathtt{W}$, wie im deduktiven Planen werden die Regeln in Form von Formeln spezifiziert, diese bilden zusammen die Menge $\mathtt{AS}$. Der Zielzustand wird wiederum durch eine Formel $\mathtt{goal}$ gegeben. Ein Plan ist dann der Beweis für eine Formel der Gestalt $\mathtt{ass_1} \land \ldots \land \mathtt{ass_n} \Rightarrow \mathtt{goal}$. Da die Implikation $\Rightarrow$ eine materielle Implikation ist, also die Implikation wahr ist, wenn eine der Vorbedingungen falsch ist, fordert man für einen korrekten Plan:

1. $\mathtt{W} \cup \mathtt{AS} \vdash \mathtt{assum_1} \land \ldots \land \mathtt{assum_n} \Rightarrow \mathtt{goal}$ und

2. $\mathtt{W} \cup \mathtt{AS} \cup \{\mathtt{assum_1} \wedge \ldots \wedge \mathtt{assum_n}\}$ ist konsistent,

wobei $\vdash$ die Beweisbarkeit in der Intervallogik bedeutet.
Dieser Ansatz hat allerdings gewisse Grenzen, die im wesentlichen durch Erweiterungen gelöst werden sollen. Zum einen läßt sich in dieser Logik nicht direkt ausdrücken, was mögliche zukünftige Zustände sein werden, so daß sich nicht planen läßt, wie man mögliche zukünftige Ereignisse verhindern beziehungsweise ermöglichen soll. Auf der anderen Seite verhindert die klassische Konsistenzannahme die Behandlung partieller oder ungenauer Information.
Sehen wir uns nun solche Probleme näher an:
Deduktiv kann es der Fall sein, daß zwei verschiedene Aktionen ausführbar sind, aber nicht beide zusammen. Zum Beispiel kann ein Roboter, wenn die Vorbedingungen gegeben sind – Weg ist frei – nach vorne gehen, das heißt die Aktion $\mathtt{MoveForward(I)}$ ist ausführbar, er kann aber auch rückwärts gehen, das heißt $\mathtt{MoveBackward(I)}$ ist ebenfalls ausführbar, aber $\mathtt{MoveForward(I)} \wedge \mathtt{MoveBackward(I)}$ ist natürlich nicht ausführbar. Man könnte die Ableitbarkeit dieser Formel dadurch verhindern, daß man eine Formel der Gestalt $\neg(\mathtt{MoveForward(I)} \wedge \mathtt{MoveBackward(I)})$ in die Weltbeschreibung aufnimmt. Für den allgemeinen Fall setzt dieses Vorgehen allerdings voraus, daß man eine vollständige Weltbeschreibung hat und Interaktionen dieser Art nicht vergessen werden dürfen. In einem wesentlich restriktiveren Ansatz würde man konservativer vorgehen und aus der Tatsache, daß eine Interaktion nicht explizit angegeben ist, nicht schließen, daß sie nicht vorliegt.
Aber es gibt auch noch andere Probleme, was die Vergabe von Ressourcen betrifft. Nehmen wir mal an, wir haben die Situation, daß ein Heizöfchen eine Einheit Strom verbraucht, eine elektrische Säge dieselbe Menge Strom verschlingt und daß das Stromnetz eine Belastung von zwei Einheiten Strom verkraftet. Wir könnten diese Situation mit dem folgenden Axiom beschreiben:

$$\begin{array}{l}\mathtt{Try(RunHeater, I)} \wedge \mathtt{Try(RunSaw, I)} \wedge \mathtt{CircuitProvides(2, I)} \Rightarrow \\ \mathtt{HeaterRunning(I)} \wedge \mathtt{SawRunning(I)}\end{array}$$

Dieses Axiom besagt aber nichts darüber, was passiert, wenn man nur ein Gerät betreiben will. Dazu braucht man dann noch die folgenden beiden Axiome:

$$\mathtt{Try(RunHeater, I)} \wedge \mathtt{CircuitProvides(1, I)} \Rightarrow \mathtt{HeaterRunning(I)}$$

$$\mathtt{Try(RunSaw, I)} \wedge \mathtt{CircuitProvides(1, I)} \Rightarrow \mathtt{SawRunning(I)}$$

Nehmen wir diese beiden Regeln dazu, haben wir aber das Problem, daß nun bei einer Energieversorgung von $\mathtt{1}$ mit der ersten Regel ableitbar ist, daß das Heizöfchen läuft und mit der zweiten, daß die Säge läuft. Das wollten wir aber gar nicht, deshalb müssen wir die Regeln durch die folgenden komplizierteren ersetzen:

$$\mathtt{Try(RunHeater, I)} \wedge \neg\mathtt{Try(RunSaw, I)} \wedge \mathtt{CircuitProvides(1, I)} \Rightarrow \mathtt{HeaterRunning(I)}$$

$$\mathtt{Try(RunSaw, I)} \wedge \neg\mathtt{Try(RunHeater, I)} \wedge \mathtt{CircuitProvides(1, I)} \Rightarrow \mathtt{SawRunning(I)}$$

Diese Regeln haben aber den entscheidenden Nachteil, daß sie nicht lokal sind. Es ist nicht mehr möglich die Aktionen isoliert zu beschreiben, vielmehr ist es bei der Planung von `HeaterRunning` erforderlich, auch noch `RunSaw` zu berücksichtigen. Bei vielen Merkmalen erhalten wir also eine unübersichtliche Regelmenge. Ziel ist es also, die Regeln isoliert zu beschreiben, obwohl sie parallel ausgeführt werden können sollen.
Dies soll dadurch erreicht werden, daß man die Bedingung abändert, daß zwei Aktionen nicht interferieren, wenn sie konsistent angenommen werden können, das heißt $\mathtt{A}_1(\mathtt{I})$ und $\mathtt{A}_2(\mathtt{I})$ interferieren nicht, wenn $\mathtt{A}_1(\mathtt{I}) \wedge \mathtt{A}_2(\mathtt{I})$ konsistent sind. Dies soll dadurch geschehen, daß man die Eigenschaft des Interferierens explizit verwaltet.

7.2 Ein erweiterter Rahmen des Planens

Da der klassische Rahmen zu eingeschränkt ist, soll er erweitert werden, so daß er das folgende unterstützt:

1. externe Ereignisse, die auch während der Ausführung einer Aktion auftreten können,
2. Bedingungen, die für die Ausführung notwendig sind und die davon abhängen, was während der Ausführung passiert,
3. die explizite Behandlung dessen, was eine Aktion *nicht* betrifft,
4. gleichzeitige Interaktionen zwischen Aktionen, wie Interferenzen (auch bedingte) und Gleichzeitigkeitseffekte,
5. die sequentielle und parallele Zusammensetzbarkeit von Aktionen.

Um dies zu ermöglichen, muß man eine Erweiterung einführen, die sich von der Determiniertheit der Zukunft verabschiedet. Bislang hatten wir (implizit oder explizit) ein lineares Zeitmodell betrachtet. Nun wird als Erweiterung der Intervallogik eine Struktur über den Zeitintervallen eingeführt, eine sogenannte Historie, die es erlaubt über *mögliche Zustände* zu reden. Jede Historie ist ein möglicher Ablauf in der Zeit. Zusätzlich gibt es eine Relation über den Historien, die diese in einen Baum gliedert, der in der Zukunft verzweigt, das heißt Zeit ist nicht linear, sondern die Zukunft ist insoweit offen, als verschiedene Abläufe denkbar sind. Dieses Modell entspricht ziemlich genau dem Kripke-Modell einer auf Modallogik aufbauenden Temporallogik. Aktionen selbst werden nie absolut, sondern immer im Rahmen der augenblicklichen Zeit ausgeführt.
Zwischen Zeitintervallen gibt es dabei die Allenschen Beziehungen, die sich aus der Beziehung $\mathtt{Meets}(\mathtt{I}_1, \mathtt{I}_2)$ zusammensetzen lassen. $\mathtt{Meets}(\mathtt{I}_1, \mathtt{I}_2)$ heißt, daß $\mathtt{I}_1$ ganz vor $\mathtt{I}_2$ liegt, sich die Intervalle aber berühren. Andere Beziehungen sind:

$\mathtt{I}_1 \subseteq \mathtt{I}_2$ für $\mathtt{I}_1$ ist in $\mathtt{I}_2$ enthalten.

Starts$(\mathtt{I}_1, \mathtt{I}_2)$ für $\mathtt{I}_1$ fängt gleichzeitig mit $\mathtt{I}_2$ an, hört aber vor $\mathtt{I}_2$ auf.

Finishes$(\mathtt{I}_1, \mathtt{I}_2)$ für $\mathtt{I}_1$ endet gleichzeitig mit $\mathtt{I}_2$, fängt aber später an.

$\mathtt{I}_1 = \mathtt{I}_2$ für $\mathtt{I}_1$ und $\mathtt{I}_2$ sind gleich.

EndsBeforeEqual$(\mathtt{I}_1, \mathtt{I}_2)$ für $\mathtt{I}_1$ endet bevor or gleichzeitig mit $\mathtt{I}_2$.

Die Zeit ist dabei in die Zukunft verzweigend, das heißt sie hat die Struktur:

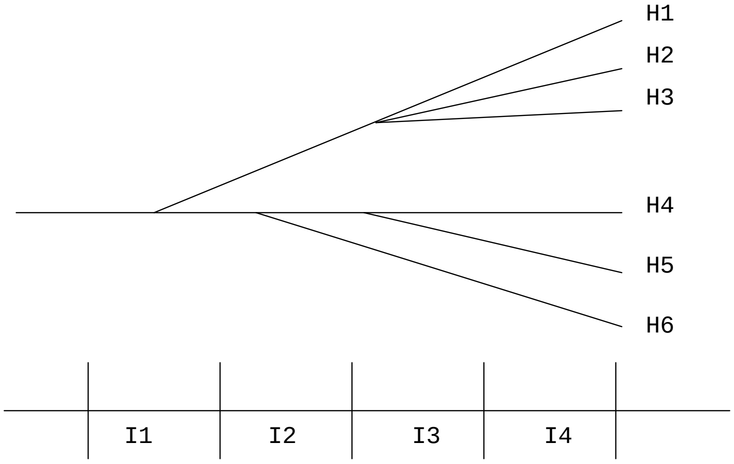


Abbildung 7.1: Historien in Relation R und Zeitintervalle

Es gibt dabei eine sogenannte Übergangsrelation R, die angibt in welcher Weise Historien zusammenhängen. $\mathtt{R}(\mathtt{I}, \mathtt{H}_1, \mathtt{H}_2)$ bedeutet, daß $\mathtt{H}_1$ und $\mathtt{H}_2$ eine gemeinsame Vergangenheit haben, die (mindestens) bis durch das Ende von I geht.
Für R gibt es die folgenden Einschränkungen:

R0 Für alle Historien $\mathtt{H}_1$ und $\mathtt{H}_2$ und alle Intervalle $\mathtt{I}_1$ und $\mathtt{I}_2$ gilt, wenn EndsBeforeEqual$(\mathtt{I}_1, \mathtt{I}_2)$ und $\mathtt{R}(\mathtt{I}_2, \mathtt{H}_1, \mathtt{H}_2)$, dann auch $\mathtt{R}(\mathtt{I}_1, \mathtt{H}_1, \mathtt{H}_2)$.

R1 R ist reflexiv, das heißt für alle Historien H und alle Intervalle I gilt $\mathtt{R}(\mathtt{I}, \mathtt{H}, \mathtt{H})$.

R2 R ist symmetrisch, das heißt für alle Historien $\mathtt{H}_1$, $\mathtt{H}_2$ und alle Intervalle I gilt: wenn $\mathtt{R}(\mathtt{I}, \mathtt{H}_1, \mathtt{H}_2)$, dann $\mathtt{R}(\mathtt{I}, \mathtt{H}_2, \mathtt{H}_1)$.

R3 R ist transitiv, das heißt für alle Historien $\mathtt{H}_1$, $\mathtt{H}_2$ und $\mathtt{H}_3$ sowie alle Intervalle I gilt: wenn $\mathtt{R}(\mathtt{I}, \mathtt{H}_1, \mathtt{H}_2)$ und $\mathtt{R}(\mathtt{I}, \mathtt{H}_2, \mathtt{H}_3)$, dann $\mathtt{R}(\mathtt{I}, \mathtt{H}_1, \mathtt{H}_3)$.

R4 Für alle Historien $\mathtt{H}_1$ und $\mathtt{H}_2$, jede Eigenschaft P sowie alle Intervalle $\mathtt{I}_1$ und $\mathtt{I}_2$ gilt: wenn EndsBeforeEqual$(\mathtt{I}_1, \mathtt{I}_2)$ und $\mathtt{R}(\mathtt{I}_2, \mathtt{H}_1, \mathtt{H}_2)$ dann ist $\langle \mathtt{I}_1, \mathtt{H}_1 \rangle \in \mathtt{P}$ genau dann wenn $\langle \mathtt{I}_1, \mathtt{H}_2 \rangle \in \mathtt{P}$

Wie man in der letzten Einschränkung sieht, werden Eigenschaften P dabei als Mengen geordneter Paare aufgefaßt. Das heißt $\langle \mathtt{I}, \mathtt{H} \rangle \in \mathtt{On}(\mathtt{A}, \mathtt{B})$ bedeutet, daß in der Historie H während des Intervalls I Klötzchen A auf Klötzchen B steht.

7.3 Planinstanzen und elementare Aktionen

Zur Beschreibung der Möglichkeiten eines Agenten zeichnen wir eine Menge von Formeln aus, die die *elementaren Aktionen* des Agenten umfassen. Informell verstehen wir dabei unter elementaren Aktionen die möglichen Aktionen des Agenten, die auf der feinst möglichen Ebene gegeben sind, die zur vollständigen Beschreibung der Domäne erforderlich ist. Eine Aktion heißt elementar, wenn es keine andere Aktion (Folge von Aktionen) gibt, durch die sie erreicht werden kann. Zum Beispiel braucht man zur Planung eines Schachprogramms nicht mehr zu spezifizieren, wie die Figuren effektiv versesetzt werden, sondern die Züge in der Form `Dd2d5` sind im Sinne einer Planung des Schachspiels elementar. Die leere Aktion, durch die nichts geschieht, sehen wir dabei auch als elementar an, es kann ja für einen Plan wichtig sein, eine Zeitlang nichts zu tun, beziehungsweise bestimmte Sachen nicht zu tun.
Planinstanzen `pi` sind Paare $\langle \mathtt{FI}, \mathtt{EA} \rangle$ bestehend aus Mengen von Formelinstanzen $\{\mathtt{fi}\}$ und Mengen von Instanzen elementarer Aktionen $\{\mathtt{ea}\}$, wobei die Menge der Formelinstanzen eine Weltbeschreibung bestehend aus Paaren der Form $\langle \mathtt{On(A,B)}, \mathtt{I} \rangle$ liefert und die Instanzen der elementaren Aktionen die Schritte beschreibt, die im Plan ausgeführt werden sollen. Dabei haben diese elementaren Aktionen wie die Formelinstanzen auch wieder Zeitconstraints, die besagen, in welchem Zeitintervall die elementare Aktionen ausgeführt werden soll. Das heißt die elementraren Aktionen haben die Gestalt $\langle \mathtt{STACK(A,B)}, \mathtt{I} \rangle$.
Für Planinstanzen haben wir die folgenden Constraints:

7.1 Annahme: Sei $\mathtt{Ints(S)}$ definiert als $\{I | \langle P, I \rangle \in S\}$ und $\mathtt{Covers(I,S)}$ wahr genau dann wenn für alle Intervalle $\mathtt{I_x} \in \mathtt{S}$ gilt $\mathtt{In(I_x, I)}$ ist wahr und es gibt Intervalle $\mathtt{I_y}$ und $\mathtt{I_z}$ in $\mathtt{S}$, so daß sowohl $\mathtt{EqualStarts(I_y, I)}$ als auch $\mathtt{EqualFinishes(I_z, I)}$ gelten. Wir fordern dann, daß für jede Planinstanz $\langle \mathtt{FI}, \mathtt{EA} \rangle$ Intervalle $\mathtt{I_{c_1}}$ und $\mathtt{I_{c_2}}$ existieren, so daß $\mathtt{Covers(I_{c_1}, Ints(FI))}$ und $\mathtt{Covers(I_{c_2}, Ints(EA))}$ gelten, wobei $\mathtt{I_{c_1}}$ und $\mathtt{I_{c_2}}$ die Beziehung $\mathtt{EqualStarts(I_{c_2}, I_{c_1})}$ haben. ■

In anderen Worten für Planinstanzen soll gelten, daß es ein Intervall gibt, das die Menge der elementaren Aktionen überdeckt und gleichzeitig mit einem anderen Intervall beginnt, das die Menge der Formelinstanzen überdeckt, aber vor ihm endet. In der STRIPS-Sprache gesprochen soll also gewährleistet werden, daß zur Zeit der Ausführung der elementaren Aktionen die Vorbedingungen für die Ausführung noch gewährleistet sind.
Nun soll formalisiert werden, was es heißt, daß eine Planinstanz *ausführbar* ist. Dies ist der Fall wenn der Versuch der Ausführung nicht fehlschlägt, sondern in einer Historie `H` zu seinem Vorkommen führt, daß heißt wenn für jede Formelinstanz $\langle \mathtt{P}, \mathtt{I} \rangle$ in $\mathtt{FI} \cup \mathtt{EA}$ gilt $\langle \mathtt{I}, \mathtt{H} \rangle \in \mathtt{P}$ (geschrieben als $\mathtt{OccursIn(\langle FI, EA \rangle, H)}$)
Das Zusammenfügen von Plänen besteht einfach aus einer Menge von elementaren Aktionen. Diese werden im Gegensatz zum linearen und nichtlinearen Planen nicht mehr extern geordnet (sei durch die Reihenfolge in einer Liste oder eine explizite Ordnung), sondern sie sind durch die Zeitintervalle, mit denen sie markiert sind, im-

plizit geordnet. Somit ergibt sich die Komposition von zwei Planinstanzen $\langle \mathtt{FI}_1, \mathtt{EA}_1 \rangle$ und $\langle \mathtt{FI}_2, \mathtt{EA}_2 \rangle$ einfach durch $\langle \mathtt{FI}_1 \cup \mathtt{FI}_2, \mathtt{EA}_1 \cup \mathtt{EA}_2 \rangle$. Damit dies problemlos möglich ist, wird folgende allgemeine Einschränkung erhoben:

7.2 Annahme: Für beliebige Planinstanzen $\langle \mathtt{FI}_1, \mathtt{EA}_1 \rangle$ und $\langle \mathtt{FI}_2, \mathtt{EA}_2 \rangle$ ist auch $\langle \mathtt{FI}_1 \cup \mathtt{FI}_2, \mathtt{EA}_1 \cup \mathtt{EA}_2 \rangle$ eine Planinstanz. ■

Für solchermaßen zusammengesetzte Pläne haben wir die Beziehung:
Für jede Historie $\mathtt{H}$ und für alle Planinstanzen $\langle \mathtt{FI}_1, \mathtt{EA}_1 \rangle$ und $\langle \mathtt{FI}_2, \mathtt{EA}_2 \rangle$ gilt:
$\mathtt{OccursIn}(\langle \mathtt{FI}_1 \cup \mathtt{FI}_2, \mathtt{EA}_1 \cup \mathtt{EA}_2 \rangle, \mathtt{H})$ genau dann wenn $\mathtt{OccursIn}(\langle \mathtt{FI}_1, \mathtt{EA}_1 \rangle, \mathtt{H})$ und $\mathtt{OccursIn}(\langle \mathtt{FI}_2, \mathtt{EA}_2 \rangle, \mathtt{H})$.
Was ist nun durch diese Darstellung im Verhältnis zur bisherigen Beschreibung von Operatoren und Zuständen gewonnen. Durch die explizite Repräsentation von Zeit sowohl bei Zuständen (durch Ausdrücke wie $\langle \mathtt{On(A,B)}, \mathtt{I} \rangle$) als auch von Operatoren (durch Ausdrücke wir $\langle \mathtt{STACK(x,y)}, \mathtt{I} \rangle$) hat man die Möglichkeit sich von einigen der Annahmen zu lösen. Die Konstanz der Welt ist nicht mehr vorausgesetzt, da zu gewissen Zeiten, gewisse Beschreibungen neu auftreten können. Weiterhin wurde durch die letzte Annahme eine gewisse Interaktionsfreiheit gefordert. A priori spricht auch nichts mehr dagegen, daß während des Planungsprozesses neue Informationen hinzukommen.

7.4 Die Nähe von Historien

In diesem Abschnitt wollen wir uns damit beschäftigen, wie verschiedene Historien bezüglich einer elementaren Aktion in Beziehung gesetzt werden können. Dazu definieren wir eine Funktion $\mathcal{N}$, die aus einer elementaren Aktion $\mathtt{ea}$, einen Intervall $\mathtt{I}$ und einer Historie $\mathtt{H}$, die Menge von Historien $\mathcal{N}(\mathtt{ea}, \mathtt{I}, \mathtt{H})$ bestimmt, die von $\mathtt{H}$ nur insoweit abweichen als sie der elementaren Aktion $\mathtt{ea}$ im Intervall $\mathtt{I}$ Rechnung tragen. Diese Historien werden als die zu $\mathtt{H}$ nächsten Historien bezeichnet bei denen $\mathtt{ea@I}$ auftritt. Dabei steht $\mathtt{ea@I}$ für „Elementaraktion $\mathtt{ea}$ mit Ausführungszeit $\mathtt{I}$“. Wir sagen „$\mathtt{ea@I}$ ist ausgeführt“, wenn $\mathtt{ea@I}$ vorkommt.
Formal gelten für die Funktion $\mathcal{N}$ die folgenden Beziehungen:

1. Für jede Elementaraktion $\mathtt{ea}$, jedes Intervall $\mathtt{I}$ und jede Historie $\mathtt{H}$ gilt, wenn $\langle \mathtt{I}, \mathtt{H} \rangle \in \mathtt{ea}$, dann gilt $\mathcal{N}(\mathtt{ea}, \mathtt{I}, \mathtt{H}) = \{\mathtt{H}\}$.

   Dies besagt, eine Elementaraktion $\mathtt{ea@I}$ kommt in einer Historie $\mathtt{H}$ vor (d.h. $\langle \mathtt{I}, \mathtt{H} \rangle \in \mathtt{ea}$) oder nicht.

2. Für alle Historien $\mathtt{H}_1$ und $\mathtt{H}_2$, jede Elementaraktion $\mathtt{ea}$ und alle Intervalle $\mathtt{I}_1$ und $\mathtt{I}_2$ gilt: Wenn $\mathtt{H}_2 \in \mathcal{N}(\mathtt{ea}, \mathtt{I}_2, \mathtt{H}_1)$ und $\mathtt{Meets}(\mathtt{I}_1, \mathtt{I}_2)$, dann gilt $\mathtt{R}(\mathtt{I}_1, \mathtt{H}_2, \mathtt{H}_1)$.

   Diese Einschränkung besagt, daß wenn eine Historie $\mathtt{H}_2$ von einer Historie $\mathtt{H}_1$ lediglich in Bezug auf die Ausführung der Elementaraktion $\mathtt{ea}$ während des Intervalls $\mathtt{I}_2$ abweicht, daß dann die Historien in jedem Intervall $\mathtt{I}_1$, das endet, wenn $\mathtt{I}_2$ beginnt, übereinstimmen.

3. Für alle Historien $\mathtt{H}_1$, $\mathtt{H}_2$ und $\mathtt{H}_{\mathcal{N}_1}$, jede Elementaraktion $\mathtt{ea}$ und alle Intervalle $\mathtt{I}$ und $\mathtt{I_x}$ gilt: Wenn $\mathtt{R}(\mathtt{I_x},\mathtt{H}_1,\mathtt{H}_2)$ und $\mathtt{H}_{\mathcal{N}_1} \in \mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H}_1)$ dann gibt es eine Historie $\mathtt{H}_{\mathcal{N}_2} \in \mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H}_2)$, so daß $\mathtt{R}(\mathtt{I_x},\mathtt{H}_{\mathcal{N}_1},\mathtt{H}_{\mathcal{N}_2})$.

   Diese Bedingung sagt, daß wenn zwei Historien $\mathtt{H}_1$ und $\mathtt{H}_2$ mindestens bis zum Ende von $\mathtt{I_x}$ übereinstimmen und $\mathtt{ea}$ eine Elementaraktion ist, die zur Zeit $\mathtt{I}$ ausgeführt wird, daß dann die Nachbarschaftsbeziehungen $\mathcal{N}$ für $\mathtt{H}_1$ und $\mathtt{H}_2$ in der folgenden Form kompatibel sein müssen: Zu jedem Element $\mathtt{H}_{\mathcal{N}_1}$ von $\mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H}_1)$ existiert ein Element $\mathtt{H}_{\mathcal{N}_2}$ von $\mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H}_2)$, so daß $\mathtt{H}_{\mathcal{N}_1}$ und $\mathtt{H}_{\mathcal{N}_2}$ vor $\mathtt{I_x}$ übereinstimmen.

4. Für alle Historien $\mathtt{H}$, Elementaraktionen $\mathtt{ea}$ und Intervalle $\mathtt{I}$ gilt:
   Wenn $\mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H}) \neq \{\mathtt{H}\}$ dann ist $\mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H}) \subseteq \{\mathtt{H}_2 | \langle \mathtt{I},\mathtt{H}_2\rangle \in \mathtt{ea}\}$.

   Diese letzte Bedingung betrifft die sogenannte Standardbedingung von Elementaraktionen, die besagen, wann sie überhaupt anwendbar sind. Diese Bedingung wird gefordert, um zu erreichen, daß der einzige Fall, in dem $\mathtt{ea@I}$ nicht in jeder Historie von $\mathcal{N}(\mathtt{ea},\mathtt{I},\mathtt{H})$ vorkommt, der ist, in dem die Standardbedingung verletzt ist.

Wie im klassischen Fall des Planens spielt natürlich die Zusammensetzung und Interaktion von Elementaraktionen eine große Rolle. Man muß auch hier die Fälle behandeln, in denen Aktionen interferrieren, eine Aktionsinstanz die Standardbedingung einer anderen zerstört, eine Aktionsinstanz Voraussetzung für die Standardbedingung einer anderen ist oder beide Standardbedingungen nicht gelten. Zu diesem Zweck wird eine Funktion $\mathtt{F}_{\mathcal{N}}$ definiert, die eine Menge von Elementaraktionen $\mathtt{EA}$ und eine Historie $\mathtt{H}$ als Argumente erhält und als Ergebnis die Menge aller Historien liefert, die von $\mathtt{H}$ nur in Bezug auf die Vorkommen der Elementaraktionen in $\mathtt{EA}$ abweichen. Dann muß im Detail geklärt werden, wann eine Komposition von Aktionen möglich ist. Beides soll hier nicht ausgeführt werden. Die Prinzipien sind ähnlich denen, die wir im nichtlinearen Planen kennengelernt haben (vergleiche [Pel91, p.154 ff]).

7.5 Die Sprache

Die zugrundeliegende Sprache ist eine sortierte Logik erster Stufe mit Modaloperatoren.

Syntax

Die primitiven Symbole sind Variable, Funktionssymbole (inklusive Konstanten), Prädikatssymbole (mit aussagenlogischen Atomen), Modaloperatoren, Junktoren und Quantoren. Terme sind wie üblich rekursiv definiert und bei der Definition von Formeln in Definition 1.2 kommen noch die folgenden beiden Fälle hinzu:

F5 Ist $\mathtt{I}$ ein Intervall und φ eine Formel, so ist $\Box_{\mathtt{I}}\varphi$ eine Formel.

F6 Ist $\texttt{pi}$ eine Planinstanz und φ eine Formel, so ist $\text{IFTRY}_{\texttt{pi}}\varphi$ eine Formel.

Der Temporalaspekt kommt dadurch zum Tragen, daß das letzte Argument von Prädikatssymbolen ein Intervall sein kann, es handelt sich dann um ein temporales Prädikat, andernfalls um ein zeitloses Prädikat. Zu den letzteren zählen die fest interpretierten Prädikate Meets und Succ.
$\Box_{\texttt{I}}$ und $\text{IFTRY}_{\texttt{pi}}$ sind Modaloperatoren. $\Box_{\texttt{I}}\varphi$ hat die intuitive Semantik, daß am Ende des Intervalls I die Formel φ unausweichlich gilt. Dual dazu definiert man wie üblich $\Diamond_I\varphi$ als $\neg\Box_I\neg\varphi$ mit der intuitiven Semantik, daß φ am Ende von I möglicherweise gilt (es ist nicht notwendigerweise der Fall, daß φ nicht gilt.)
Die intuitive Bedeutung von $\text{IFTRY}_{\texttt{pi}}\varphi$ ist: „Wenn die Planinstanz pi versucht worden wäre, dann wäre φ wahr“. Das Ausdruck Succ(pi) hat die Semantik, daß pi erfolgreich ausgeführt werden kann. Damit kann man zum Beispiel definieren, daß pi ausführbar ist durch: $\texttt{Executable(pi)} := \text{IFTRY}_{\texttt{pi}}\texttt{Succ(pi)}$.

Semantik

Die Semantik dieser Logik wird wie üblich definiert, wobei allerdings die Historien die möglichen Welten darstellen, bezüglich denen zeitabhängige Ausdrücke interpretiert werden. Wir gehen hier nur kurz auf die fest interpretierten Prädikate und die Modaloperatoren ein.
Es wird definiert:
$\mathcal{I}(\texttt{P}(\texttt{t}_1,\ldots,\texttt{t}_\texttt{n},\texttt{I}),\texttt{H}) = \textsf{wahr}$ genau dann wenn $\langle\mathcal{I}(I),H\rangle \in \mathcal{I}(P)(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_n))$.

$\mathcal{I}(\texttt{Succ(pi)},\texttt{H}) = \textsf{wahr}$ genau dann wenn für jede Formelinstanz $\langle\texttt{x},\texttt{I}\rangle$ in $\mathcal{I}(\texttt{pi}|_1)\cup\mathcal{I}(\texttt{pi}|_2)$ gilt $\langle\texttt{I},\texttt{H}\rangle \in \texttt{x}$. (Dabei benutzen wir die Notation, daß $\texttt{pi}|_1$ das erste Element des Paares pi darstellt und $\texttt{pi}|_2$ das zweite).

Für Meets gilt:
$\mathcal{I}(\texttt{Meets}(\texttt{I}_1,\texttt{I}_2),\texttt{H}) = \textsf{wahr}$ genau dann wenn $\texttt{Meets}'(\mathcal{I}(\texttt{I}_1),\mathcal{I}(\texttt{I}_2))$ historienunabhängig gilt.

Der Modaloperator $\Box_{\texttt{I}}$ hat die Semantik:
$\mathcal{I}(\Box_{\texttt{I}}\varphi,\texttt{H}) = \textsf{wahr}$ genau dann wenn für jede Historie $\texttt{H}'$ mit $\texttt{R}(\mathcal{I}(\texttt{I}),\texttt{H},\texttt{H}')$ gilt $\mathcal{I}(\varphi,H')$ $= \textsf{wahr}$.

Der Modaloperator $\text{IFTRY}_{\texttt{pi}}$ hat die Semantik:
$\mathcal{I}(\text{IFTRY}_{\texttt{pi}}\varphi) = \textsf{wahr}$ genau dann wenn für jede Historie H mit $\texttt{H} \in \texttt{F}_{\mathcal{N}}(\mathcal{I}(\texttt{pi})|_2)$ gilt $\mathcal{I}(\varphi,H) = \textsf{wahr}$.
Damit sind nun die grundlegenden Beschreibungselemente für die Planung zusammen, insbesondere eine modallogische Sprache mit ihrer temporallogischen Semantik.

7.6 Der Planungsvorgang

Wir wollen nun den Planungsbegriff an den temporallogischen Ansatz anpassen. Dazu definieren wir:

7.3 DEFINITION (PLANUNGSPROBLEM, PLAN): Ein *Planungsproblem* ist gegeben durch

- eine Beschreibung der externen Welt, in der die Planausführung stattfinden soll, der sogenannten *Weltbeschreibung*[1] $\mathbf{W}$,
- eine Beschreibung der Aktionen, die der Agent ausführen kann, und ihrer Interaktion, der sogenannten *Aktionsspezifikation* $\mathbf{A}$ und
- einer Beschreibung der gewünschten Bedingungen, dem sogenannten *Ziel*, $\mathbf{Z}$.

Die gewünschte Ausgabe eines Planungsalgorithmus ist eine temporal geordnete Menge von Aktionen, dem *Plan*, der bei einer Ausführung in jeder möglichen Welt, die die Weltbeschreibung erfüllt, das Ziel erreicht. ■

In STRIPS-artigen Systemen ist die temporale Ordnung durch die Position der Operatoren im Plan gegeben, im nichtlinearen Planen durch die explizite Ordnung und im temporallogischen Ansatz durch die Angabe der Intervalle.
Ein Planungssystem heißt *korrekt*, wenn die Zielbeschreibung $\mathbf{Z}$ aus der Weltbeschreibung $\mathbf{W}$ und der Aktionsspezifikation $\mathbf{A}$ für jede mögliche externe Welt ableiten läßt. Konkret heißt das, wir suchen eine Menge von Planinstanzen $\mathtt{pi}_1, \ldots, \mathtt{pi}_\mathtt{n}$ mit $\mathtt{pi}_\mathtt{n} = \langle \mathtt{FI}_\mathtt{n}, \mathtt{EA}_\mathtt{n} \rangle$ so daß gilt:

C1 $\mathbf{W} \cup \mathbf{A} \models \Box_\mathtt{I}\mathtt{Executable}(\mathtt{Comp}(\mathtt{pi}_1, \ldots, \mathtt{pi}_\mathtt{n}))$

C2 $\mathbf{W} \cup \mathbf{A} \models \Box_\mathtt{I}\mathtt{Succ}(\mathtt{Comp}(\mathtt{pi}_1, \ldots, \mathtt{pi}_\mathtt{n})) \Rightarrow \mathbf{Z}$

Dabei bedeutet $\mathtt{I}$ die Zeit, während der der Plan ausgeführt wird. $\mathtt{Comp}$ steht für die Zusammensetzung von Planinstanzen.
Die erste Bedingung gewährleistet, daß in jedem Weltmodell, das sowohl die Weltbeschreibung als auch die Aktionsspezifikation erfüllt, der Plan ausführbar ist; die zweite, daß nach erfolgreicher Ausführung das Ziel gilt.
Weltbeschreibungen sind dabei in der Regel modallogisch mit den Modalopertoren $\Box$ oder $\Diamond$ versehen, das heißt sie können die Gestalt haben:
$\Box_\mathtt{I}\forall\mathtt{I}_1\ \mathtt{VaseDrops}(\mathtt{I}_1) \Rightarrow \exists\mathtt{I}_2\ \mathtt{Meets}(\mathtt{I}_1, \mathtt{I}_2) \wedge \mathtt{VaseBreaks}(\mathtt{I}_2)$ oder
$\Diamond_\mathtt{I}\varphi$, wobei φ eine Formel der Intervallogik ist.
Zur *Spezifikation* von Aktionen gibt es verschiedene Möglichkeiten. Zum einen sind Spezifikationen vergleichbar den Vorbedingungen (**Precondition**) von STRIPS nun

[1]Die Bezeichnung `Init` für die Weltbeschreibung in STRIPS ist nicht mehr allgemein zutreffend, da die Beschreibungsmerkmale nicht initial gelten müssen.

Ausführbarkeitsbedingungen genannt, und *Effekte* (**add list** und **delete list**) notwendig. Weiterhin muß man sagen können, was von einer Aktion unberührt bleibt. Dies geschieht entweder durch die Angabe von Frameaxiomen oder durch eine entsprechende Annahme wie in STRIPS, die besagt, daß alles, was nicht explizit geändert wird, unverändert bleibt.
Die Ausführbarkeitsbedingung für eine Planinstanz `pi` sind gegeben durch eine Formel der Art:

$$\Box_{\mathtt{I}}(\mathtt{Vor} \Rightarrow \mathtt{Executable(pi)})$$

wobei `Executable(pi)` definiert war als $\text{IFTRY}_{\mathtt{pi}}\mathtt{Succ(pi)}$.
Zum Beispiel hat die Ausführbarkeitsbedingung des Operators `STACK` die Gestalt:
$\Box_{\mathtt{I}}(\mathtt{Clear(A, I_A)} \wedge \mathtt{Meets(I_A, I)} \wedge \mathtt{Holds(B, I_B)} \wedge \mathtt{Meets(I_B, I)} \Rightarrow$
$\mathtt{Executable(STACK(B, A, I)))}$
Ausführbarkeitsbedingungen sind insoweit allgemeiner als die Vorbedingungen von STRIPS, als sie sich nicht auf die Beschreibung eines Zustandes *vor* Ausführung eines Operators beschränken müssen. Zum Beispiel kann eine solche Bedingung die Gestalt haben:
$\Box_{\mathtt{I}}(\mathtt{WindBlowing(I_1)} \wedge \mathtt{I} \subseteq \mathtt{I_1} \Rightarrow \mathtt{BoatMoving(I)})$
oder aber Bedingungen können sich auf andere Planinstanzen beziehen wie in:
$\Box_{\mathtt{I}}(\mathtt{Succ(PICKUP(A, I_0))} \wedge \mathtt{Clear(x, I_0)} \wedge \mathtt{Meets(I_0, I)} \Rightarrow \mathtt{Executable(STACK(A, x, I)))}$
Effekte eines Plans sind auch durch Aussagen in der Intervallogik gegeben. Sie haben die Gestalt:

$$\Box_{\mathtt{I}}(\mathtt{Succ(pi)} \Rightarrow \mathtt{EFF})$$

Zum Beispiel für den Operator `STACK` können wir spezifizieren:

$$\begin{aligned}\Box_{\mathtt{I}}(\mathtt{Succ(STACK(x, y, I))} \Rightarrow (\exists \mathtt{I_1}\mathtt{Meets(I, I_1)} \wedge \neg\mathtt{Holds(x, I_1)} \wedge \neg\mathtt{Clear(y, I_1)} \wedge \\ \mathtt{Clear(x, I_1)} \wedge \mathtt{On(x, y, I_1)} \wedge \mathtt{Handempty(I_1)}))\end{aligned}$$

Ein wesentlicher Punkt der in der STRIPS-Sprache nicht direkt ausgedrückt werden kann, ist die Beschreibung von simultanen Effekten, wie zum Beispiel das Hochheben eines Tisches durch zwei Agenten:

$$\begin{array}{llll}\Box_{\mathtt{I}}(& \mathtt{Succ(PushLeft(I))} \wedge & \neg\mathtt{Succ(PushRight(I))} & \Rightarrow \mathtt{BoxSlidesLeft(I)}) \\ \Box_{\mathtt{I}}(& \neg\mathtt{Succ(PushLeft(I))} \wedge & \mathtt{Succ(PushRight(I))} & \Rightarrow \mathtt{BoxSlidesRight(I)}) \\ \Box_{\mathtt{I}}(& \mathtt{Succ(PushLeft(I))} \wedge & \mathtt{Succ(PushRight(I))} & \Rightarrow \mathtt{BoxLifted(I)})\end{array}$$

In dieser Darstellung werden weder die Effekte von `PushRight` noch diejenigen von `PushLeft` isoliert beschrieben. In STRIPS kann ein solches Zusammenspielen von Operatoren nicht beschrieben werden.
Wie beschreibt man was eine Planinstanz nicht berührt? Für eine Bedingung `P` kann man dies durch die folgende Formel ausdrücken:

$$\mathtt{P} \Rightarrow \text{IFTRY}_{\mathtt{pi}}\ \mathtt{P}$$

Man beachte, daß sowohl $\mathtt{P}$ als auch $\mathtt{pi}$ jeweils mit Intervallen zeitbeschränkt gegeben sind, deshalb sehen konkrete Instanzen wie folgt aus:

$$\Box_{\mathtt{I}}(\mathtt{TableStationary(I)} \Rightarrow \text{IFTRY}_{\mathtt{PLACE(Obj,Table,I)}}\ \mathtt{TableStationary(I)}))$$

was besagt, daß etwas auf einen Tisch zu stellen den Tisch als solchen nicht bewegt. Da automatisch gewährleistet ist, daß Aktionen keinen Einfluß in die Vergangenheit haben, brauchen entsprechende Zusammenhänge nicht explizit spezifiziert zu werden. Um eine überschaubare Axiomatisierung zu erreichen bieten sich Repräsentationsmöglichkeiten an, wie sie im Abschnitt 2.3 behandelt wurden.
Der vorgestellte Rahmen bietet auch die Möglichkeit Interaktionen zwischen dem Agenten und der externen Welt direkt zu beschreiben. Was damit gemeint ist, sehen wir uns wieder an einem Beispiel an:
Angenommen wir wollen sagen, daß ein Agent $\mathtt{agt}$ ein Terminal mit einem anderen Agenten $\mathtt{agt_1}$ sich teilen soll. Wir benutzen das Prädikat $\mathtt{UseTerm(x, I)}$ für „Der Agent $\mathtt{x}$ benutzt das Terminal während des Intervalles $\mathtt{I}$". Wir können dann ausdrücken, daß der Agent $\mathtt{agt}$ das Terminal nicht gleichzeitig mit dem Agenten $\mathtt{agt_1}$ benutzen kann durch:

$$\forall \mathtt{I_1, I_2}\ \neg\mathtt{Disjoint(I_1, I_2)} \Rightarrow \Box_{\mathtt{I}}(\neg\mathtt{UseTerm(agt_1, I_1)} \wedge \mathtt{Succ(UseTerm(agt, I_2)))}$$

In diesem Fall richtet sich der Agent $\mathtt{agt}$ ganz nach dem Agenten $\mathtt{agt_1}$ und benutzt das Terminal nur, wenn es frei ist. Ja mehr noch, wann immer $\mathtt{agt_1}$ zugang zum Terminal will, kann er $\mathtt{agt}$ vertreiben. In der Sprache kann man aber genausogut andere Interaktionen ausdrücken, wie solche, die besagen, daß der Agent nur wartet, wenn das Terminal schon besetzt ist, sich aber nicht vertreiben läßt, wenn es einmal frei war.
Wie lassen sich nun Interaktionen zwischen Planinstanzen beschreiben? Wir können definieren, daß zwei Planinstanzen einander nicht betreffen durch eine Definition wie:

$$\begin{aligned}\mathtt{NoConflict(pi_1, pi_2)} :\Leftrightarrow & (\mathtt{Executable(pi_1)} \Rightarrow \text{IFTRY}_{\mathtt{pi_2}}\mathtt{Executable(pi_1)}) \wedge \\ & (\mathtt{Succ(pi_1)} \Rightarrow \text{IFTRY}_{\mathtt{pi_2}}\mathtt{Succ(pi_1)}) \wedge \\ & (\mathtt{Executable(pi_2)} \Rightarrow \text{IFTRY}_{\mathtt{pi_1}}\mathtt{Executable(pi_2)}) \wedge \\ & (\mathtt{Succ(pi_2)} \Rightarrow \text{IFTRY}_{\mathtt{pi_1}}\mathtt{Succ(pi_2)})\end{aligned}$$

Wollen wir nun ausdrücken, daß die beiden Planinstanzen $\mathtt{pi_1}$ und $\mathtt{pi_2}$ einander nicht stören, so können wir sagen:

$$\Box_{\mathtt{I}}\mathtt{NoConflict(pi_1, pi_2)}$$

Wie können eine solche Aussage natürlich auch an Bedingungen knüpfen durch:

$$\Box_{\mathtt{I}}(\mathtt{C} \Rightarrow \mathtt{NoConflict(pi_1, pi_2)})$$

Im Beispiel des Heizöfchens und der Säge liest sich das als:

$$\Box_{\mathtt{I'}}\mathtt{MinCircuitProvides(2, I)} \Rightarrow \mathtt{NoConflict(HeaterRunning(I), SawRunning(I))}$$

Zum Abschluß wollen wir uns kurz einen Planungsalgorithmus ansehen. Die Eingabe des Algorithmus besteht dabei aus:

W der Weltbeschreibung

Z dem Ziel

sowie den Aktionsspezifikationen bestehend aus einer

PI Menge von einfachen Planinstanzen, die zur Erreichung des Ziels verwendet werden können

EF Funktion, die die Effekte zu jeder einfachen Planinstanz bestimmt.

AB Funktion, die die Ausführungsbedingungen zu jeder einfachen Planinstanz bestimmt.

NI Relation, die die Nichtinterferenz zwischen einfachen Planinstanzen angibt.

Ein Planungsprozeß kann jetzt dadurch erfolgen, daß man einen temporallogischen Beweiser das Ziel aus der Weltbeschreibung und den Aktionsspezifikation herleitet. Alternativ kann man sich ein System vorstellen, das STRIPS-artig auf means-end-analysis aufbauend, sich jeweils ein Teilziel verschafft und durch Rückwärtsschließen dieses auf neue Teilziele zurückführt, bis man bei der Weltbeschreibung angelangt ist.

Die Behandlung des Frameproblems kann, wie wir gesehen haben, durch die explizite Angabe des Nichtbetreffens von Aktionen geschehen. Es kann aber auch erfolgen, in dem man die Annahme implizit macht, daß alle Effekte einer Aktion spezifiziert sein müssen. In diesem Fall hat man: Wenn die Bedingung `Bed` in einer Historie `H` gilt und man kann nicht ableiten, daß `Bed` in der Historie `H'` nicht gilt, die durch Anwendung von `pi'` aus `H` entsteht, so nehmen wir ein, daß `C` in `H'` gilt.

Was geht nun in diesem Rahmen nicht? Die gemachte Planungsannahme war, daß die Zielbeschreibung **Z** aus der Weltbeschreibung **W** und der Aktionsspezifikation **A** temporallogisch so hergeleitet werden soll, daß der Plan unter allen möglichen Umständen ausführbar ist; insbesondere beschränkt sich der Planungsprozeß nicht auf eine einzige spezifische Weltbeschreibung. So gesehen haben wir zwar nicht direkt vorausgesetzt, daß die Information vollständig ist, sind aber im Hinblick auf typisch menschliches Planen immer noch ziemlich inflexibel. Dies hängt allerdings eng mit dem sogenannten *Qualifikationsproblem* zusammen, das besagt, daß eine vollständige Spezifikation eines Operators im allgemeinen nicht möglich ist, sondern daß man immer nur bestimmte Möglichkeiten betrachtet. Zum Beispiel beim Operator `PICKUP(A,I)` betrachtet man, daß `A` auf dem Tisch steht, frei ist und die Hand leer ist. Man müßte aber an sich auch noch andere Bedingungen berücksichtigen, die besagen, daß die Hand nicht defekt ist, daß der Strom nicht ausgefallen ist, daß der Block nicht festgeklebt ist usw. usf.

Diese Probleme können teilweise durch die Berücksichtigung von Aspekten der Nichtmonotonie, wie sie in [Sho88] beschrieben sind und die sich übrigens sehr gut im vorgestellten temporallogischen Ansatz beschreiben lassen, behoben werden. Zusätzlich kann man den Rahmen um „wissensproduzierende Aktionen“ erweitern, so daß eine bessere Behandlung unvollständiger Informationen möglich ist.

Kapitel 8

Ausblick

8.1 Was wurde nicht behandelt – ein unvollständiger Überblick

Welches sind wichtige Fragen im Planen, die gar nicht, nur teilweise oder nur unzureichend behandelt wurden? Wir werden in diesem Absatz einige Probleme auflisten, die in dieser Vorlesung trotz ihrer Wichtigkeit nicht behandelt worden sind. Die Probleme sind teilweise aus [Sha87] übernommen.

Planerkennung: Hier handelt es sich um das Problem, zu erkennen, was das Ziel eines Planes ist. Ein mögliches Anwendungsszenario besteht darin, daß ein Auskunftssystem das Interesse eines Kunden erkennt, um entsprechend reagieren zu können. Zu unterscheiden ist dabei kooperatives Handeln und nicht-kooperatives Handeln. Beim kooperativen Handeln wird das System auf die Frage: „Wissen Sie, wie spät es ist?“ nicht mit „Ja“ antworten, sondern mit der Uhrzeit. Bei nicht-kooperativem Handeln, verfolgt das System „eigene“ Interessen, wie zum Beispiel den Verkauf einer Ware und wird versuchen, sie im für den Kunden besten Licht erscheinen zu lassen.

Reaktives Planen: Es geht darum, auf unvorhergesehene beziehungsweise unvorhersehbare Umstände zu reagieren. Im Extremfall verändert sich die Sichtweise des Planens vollständig. Während man klassisch von einer Planungsphase und einer Ausführungsphase spricht, geht es nun eher darum, auf gewisse Situationen zu reagieren und dabei ein Ziel zu verfolgen.

Umplanen: Scheitert ein Plan oder ändern sich Ziel und/oder Ausgangssituation leicht, so kann man versuchen, einen alten Plan weitgehend zu übernehmen und diesen an die neue Situation anzupassen. Ein Problem dabei ist, daß eine kleine Änderung in der Beschreibung natürlich große Änderungen im Plan nach sich ziehen kann.

Lernen: Die meisten bestehenden Planungsverfahren sind nicht in der Lage aus bereits erfolgten Problemlösungen zu lernen.

Backtracking: Beim Planen ist sehr oft ein Backtracking erforderlich, es stellt sich die Frage, wie man intelligent zurücksetzen kann. Um dies tun zu können, müßte man im Prinzip wissen, warum ein Planungsversuch gescheitert ist.

Theorie: Viele theoretische Fragen sind für einzelne Planungsverfahren erst im Ansatz geklärt, wie Fragen der Vollständigkeit, der Komplexität oder der Optimalität.

Verteiltes Planen: Klassischerweise besteht Planen darin, daß der Planungsvorgang von einem Prozeß aus erfolgt. Im üblichen menschlichen Verfahren werden schwierigere Aufgaben aber nicht von einer einzigen Person geplant, sondern von mehreren, so daß die Erstellung eines Planes verteilt erfolgt. Dies erfordert natürlich die Erweiterung der vorgestellten Ansätze, zum Beispiel um die explizite Behandlung von Kommunikation.

8.2 Anwendung von Planen am Beispiel des automatischen Beweisens

Im automatischen Beweisen gibt es den Ansatz der vom interaktiven Beweisen auf sehr niedriger Ebene durch die direkte Angabe von Kalkülregeln ausgeht. Da dies sehr mühsam und länglich wird, ist man recht bald zu sogenannten Taktiken übergegangen. Taktiken sind dabei Programme, die in einem bestimmten Beweiszustand aufgerufen in eine Folge von Kalkülregeln expandieren; man kann also durch eine einzige Benutzerinteraktion eine ganze Menge von Kalkülschritten durchführen. Durch dieses Vorgehen erreicht man zudem, daß Taktiken zwar scheitern können, daß sie aber nicht dazu führen können, daß man falsche Sachen ableitet. Sie haben allerdings als allgemeine Programme den Nachteil, daß sie nicht beziehungsweise nur sehr schwer einer automatischen Behandlung zugänglich sind. Der Benutzer kann sie aufrufen, aber ein System kann nur schwer *planen*, welche Taktiken es aufrufen soll. Deshalb hat Alan Bundy [Bun88] den Begriff der Taktik zum Begriff der *Methode* so erweitert, daß zu jeder Taktik eine Spezifikation hinzugefügt wird, so daß man sich eine Methode vorstellen kann als einen Planungsoperator plus eine Taktik.

$$\texttt{Methode} = \texttt{Planungsoperator} + \texttt{Taktik}$$

Das automatische Beweisen wird nun als ein Planungsprozeß aufgefaßt.
Sehen wir uns ein Beispiel für eine Methode an, wie wir sie in Ω-MKRP verwenden wollen (vergleiche [HKK+92]).

Method: `hom1`					
pre	(exline 1) $\wedge$ (exline 2)				
post	(exline 5)				
dec-cont	1	1	$\vdash$	$\forall x \text{Formula}_f$	J1
	2	2	$\vdash$	$\forall x P(x) \Leftrightarrow \text{Formula}'$	J2
	3	3	$\vdash$	$P(c)$	?
	4	4	$\vdash$	Eq_Elim_r(Univ_Elim(2, $f(c)$), (5))	? 1 3
	5	2,4	$\vdash$	$P(f(c))$	Def_Intro 2 4
proc	`schema-interpreter`				

Abbildung 8.1: Planungsoperator im automatischen Beweisen

Ein Beweis in Ω-MKRP besteht aus einer Folge von Beweiszeilen (in etwa wie die Zeilen 1-5 in der Methode). Sind alle Fragezeichen eliminiert, so ist der Beweis vollständig. Die vorgestellte Methode soll informell gesprochen eine Homomorphieeigendschaft in den Beweis einfügen, das heißt konkret: die Zeile 5 soll gezeigt werden, also $P(f(c))$, dies kann mithilfe der Definition von P in Zeile 2 und Zeile 4 gezeigt werden. Nun ist allerdings noch Zeile 4 zu zeigen, was mithilfe einer Eigenschaft von f aus Zeile 1 und der entsprechenden Eigenschaft $P(c)$ geschehen soll, die selbst auch wiederum gezeigt werden muß. Wie dies wiederum geschehen soll, sagt die Methode nicht. Vielmehr müssen darüber nun andere Methoden Auskunft geben.
Das Planen selbst geschieht nun darüber, daß man versucht mithilfe der Nachbedingungen post das Theorem zu zeigen, unter der Voraussetzung, daß man die Zeilen, die in den Vorbedingungen pre angezeigt sind, schon hat. In diesem Planungsschritt würden also bei der obigen Methode nur die Zeilen 1, 2 und 5 berücksichtigt, wobei man die Zeilen 1 und 2 voraussetzt und die Zeile 5 zeigen will. Hat man auf einer solchen Ebene das Theorem auf bekannte Voraussetzungen zurückgeführt, so hat man einen Plan für den Beweis des Theorems gefunden. Im nächsten Durchgang wird dieser Plan ausgeführt, das heißt nun werden auch die Zeilen 3 und 4 aus obiger Methode in den aus Beweiszeilen bestehenden partiellen Beweis eingetragen. Dadurch können neue Ziele entstehen, das heißt Zeilen, die mit einem Fragenzeichen versehen sind. Diese müssen dann in einer weiteren Runde gelöst werden. Dieses Verfahren wird solange weitergeführt, bis alle Fragezeichen eliminiert sind.
Wichtige Fragen in diesem Zusammenhang sind, wie muß die Sprache genau aussehen, mit der man die Planungsoperatoren beschreibt. In den Vor- und Nachbedingungen stehen im obigen Beispiele nur die Nummern von Zeilen. Im allgemeinen wird man an solche Zeilen allerdings auch noch gewisse Bedingungen knüpfen müssen. Weiterhin kann man die Operatoren wie sie angegeben sind, mehr oder weniger in STRIPS-Operatoren übersetzen. Weitgehend offen ist, welches Planungsverfahren für welche Probleme verwendet werden sollte. Eine weitere wichtige Frage besteht darin, wie man bestehende Operatoren auf neue Probleme anpassen kann, so daß a priori mit der gegebenen Operatormenge unlösbare Probleme gelöst werden können. Diese und verwandte Fragen werden zur Zeit im Projekt Ω-MKRP im SFB 314 (D2) untersucht.

Literaturverzeichnis

[Bar77] Jon Barwise (Herausgeber). *Handbook of Mathematical Logic*, Band 90 in *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, Niederlande, 1977.

[BB87] Karl Hans Bläsius und Hans-Jürgen Bürckert, Herausgeber. *Dedukionssysteme – Automatisierung des logischen Denkens*. Oldenbourg, München, 1987.

[BCFH89] Wolfgang Bibel, Luis Fariñas del Cerro, Bertram Fronhöfer und Andreas Herzig. Plan generation by linear proofs: On semantics. In D. Metzig, Herausgeber, *Proceedings of GWAI*, Seiten 49–62, Eringerfeld, September 1989. Springer Verlag, Berlin.

[Bib82] Wolfgang Bibel. *Automated Theorem Proving*. Vieweg, Wiesbaden, 1982.

[Bib86] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, **4**:115–132, 1986.

[BM79] Robert S. Boyer und J Strother Moore. *A Computational Logic*. Academic Press, New York, USA, 1979.

[Bun88] Alan Bundy. The use of explicit plans to guide inductive proofs. In *Proceedings of the 9th CADE*, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin. LNCS 310.

[CL73] Chin-Liang Chang und Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, USA, 1973.

[Duf91] David Duffy. *Principles of Automated Theorem Proving*. John Wiley & Sons, Chichester, Großbritannien, 1991.

[EFT86] Heinz-Dieter Ebbinghaus, Jörg Flum und Wolfgang Thomas. *Einführung in die mathematische Logik*. Wissenschaftliche Buchgesellschaft, Darmstadt; zweite Auflage, 1986.

[EN69] G. Ernst und Alan Newell. *GPS: A Case Study in Generality and Problem Solving*. ACM Monograph Series. Academic Press, New York, USA, 1969.

[Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving.* Springer Verlag, New York, USA, 1990.

[FN71] Richard E. Fikes und Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**:189–208, 1971.

[Fro87] Bertram Fronhöfer. Linearity and plan generation. *New Generation Computing*, **5**:213–225, 1987.

[GG84] Dov M. Gabbay und Franz Guenthner. *Handbook of Philosophical Logic – Volume II: Extensions of Classical Logic*, Band 165 in *Studies in Epistemology, Logic, Methodology, and Philosophy of Science.* D. Reidel Publishing Company, Dordrecht, Niederlande, 1984.

[GG89] Dov M. Gabbay und Franz Guenthner. *Handbook of Philosophical Logic*, Band 164–167 in *Studies in Epistemology, Logic, Methodology, and Philosophy of Science.* D. Reidel Publishing Company, Dordrecht, Niederlande, 1983-1989.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, **50**:1–102, 1987.

[Har84] David Harel. Dynamic logic. In D. Gabbay und F. Guenthner, Herausgeber, *Handbook of Philosophical Logic*, Kapitel II.10, Seiten 497–604. D.Reidel Publishing Company, Dordrecht, Niederlande, 1984. Band II: Extensions of Classical Logic.

[Her89] Joachim Hertzberg. *Planen – Einführung in die Planerstellungsmethoden der Künstlichen Intelligenz.* B.I. Wissenschaftsverlag, Mannheim, 1989.

[HKK+92] Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts und Jörg Siekmann. Ω-MKRP – a proof development environment. SEKI Report SR-92-22, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, Saarbrücken, 1992.

[HRS90] Marita Heisel, Wolfgang Reif und Werner Stephan. Tactical theorem proving in program verification. In Mark E. Stickel, Herausgeber, *Proceedings of the 10th CADE*, Seiten 117–131, Kaiserslautern, 1990. Springer Verlag, Berlin. LNAI 449.

[Kow79] Robert Kowalski. *Logic for Problem Solving.* North Holland, New York, USA, 1979.

[Lov78] D.W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, New York, USA, 1978.

[Men87] Elliott Mendelson. *Introduction to Mathematical Logic.* Wadsworth & Brooks/Cole, Monterey, Kalifornien, USA; dritte Auflage, 1987.

[MH69] John McCarthy und Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, **4**:463–502, 1969.

[Nil84] Nils J. Nilsson. Shakey the robot. Technischer Bericht SRI-323, SRI International, Menlo Park, Kalifornien, USA, 1984.

[Ped89] E.P.D. Pednault. ADL – exploring the middle ground between STRIPS and the situation calculus. In R.J. Brachman, H. Levesque und R. Reiter, Herausgeber, *Proceedings of KR'89*, Seiten 324–332, San Mateo, Kalifornien, USA, 1989. Morgan Kaufmann.

[Pel91] Richard N. Pelavin. Planning with simultaneous actions and external events. In James F. Allen, Henry A. Kautz, Richard N. Pelavin und Josh D. Tenenberg, Herausgeber, *Reasoning about Plans*, Kapitel 3, Seiten 127–211. Morgan Kaufmann, San Mateo, Kalifornien, USA, 1991.

[Rei91] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, Herausgeber, *Artificial Intelligence and Mathematical Theory of Computation – Papers in Honor of John McCarthy*, Seiten 359–380. Academic Press, Boston, Massachusetts, USA, 1991.

[Ric78] Michael M. Richter. *Logikkalküle.* Leitfäden der angewandten Mathematik und Mechanik LAMM. Teubner, Stuttgart, 1978.

[Ros81] Stanley J. Rosenschein. Plan synthesis: A logical perspective. In *Proceedings of the 7th IJCAI*, Seiten 331–337, Karlsruhe, 1981. Morgan Kaufmann, San Mateo, Kalifornien, USA.

[Sac74] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, **5**:115–135, 1974.

[SB93] Werner Stephan und Susanne Biundo. A new logical framework for deductive planning. In Ruzena Bajcsy, Herausgeber, *Proceedings of the 13th IJCAI*, Seiten 32–38, Chambéry, Frankreich, 1993. Morgan Kaufmann, San Mateo, Kalifornien, USA.

[Sch90] L.K. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyberg, R.P. Loui und G.N. Carlson, Herausgeber, *Knowledge Representation and Defeasible Reasoning*, Seiten 23–67. Kluwer Academic Press, Boston, Massachusetts, USA, 1990.

[Sha87] Stuart C. Shapiro (Herausgeber). *Encyclopedia of Artificial Intelligence.* John Wiley and Sons, New York, USA, New York, USA, 1987.

[Sho88] Yoav Shoham. *Reasoning about Change – Time and Causation from the Standpoint of Artificial Intelligence.* The MIT Press, Cambridge, Massachusetts, USA, 1988.

[Wal77] Richard Waldinger. Achieving several goals simultaneously. *Machine Intelligence*, **8**, 1977. auch in B.L. Webber and N.J. Nilsson: Readings in Artificial Intelligence, Morgan Kaufman, San Mateo, Kalifornien, USA, 1981.

Index

Abbildungsverzeichnis