Diplomarbeit

Bedingte und rekursive Aktionen im Fluent-Kalkül

Hans-Peter Störr

Inhaltsverzeichnis

1. Einleitung

Ein Kennzeichen eines intelligenten Lebewesens ist es, innerhalb seiner Umgebung gezielt zu agieren. Dazu gehört es, über die Wirkungen seiner Aktionen nachzudenken und Pläne zu formen, um gewählte Ziele zu erreichen. Da die Künstlichen Intelligenz (KI) es sich zum Ziel gemacht hat, die intelligente Verhaltensweisen zu erforschen und nachzubilden, ist Planen eine der Hauptdisziplinen der KI.

In einem Planungsproblem, wie es in dieser Arbeit verstanden wird, ist der Grundgegenstand die Wechselwirkung der handelnden Einheit, im Folgenden Agent genannt, und seiner Umgebung, der Planungsdomäne, die er durch seine Aktionen beeinflussen kann. Die Planungsaufgabe besteht nun darin, für den Agenten einen Plan zu entwickeln, der eine Handlungsanweisung für den Agenten darstellt, so daß die Umgebung in einen gewünschten Zustand überführt wird. Die Charakterisierung des gewünschten Zustands wird Ziel genannt.[1]

In der KI-Forschung auf dem Gebiet der Planung wurden zwei Hauptrichtungen verfolgt [HS89]. Die eine zielt auf den Bau effizienter intelligenter Systeme, die Pläne für komplizierte Planungsdomänen generieren können. Die andere Richtung verfolgt die Anwendung von Methoden und Resultaten aus dem Bereich des automatischen Theorembeweisens, um Planungssysteme mit wohldefinierter Semantik zu bauen. Die Resultate in diesem Gebiet sind oft nützlich, um zu verstehen, was Planungssysteme eigentlich tun und wie ihre Fähigkeiten verbessert werden können. Diese Arbeit geht von letzterem Ansatz aus, und stellt Planungsprobleme mit Hilfe von klassischer Logik dar.

Bei der Bearbeitung von realen Planungsaufgaben ergibt sich eine Mannigfaltigkeit von Problemen [Geo87]. Eines davon ist die Frage, auf welche Weise Pläne dargestellt werden. Die meisten deduktiven Planer behandeln lineare oder partiell geordnete Pläne [HS89]. Kennzeichnend dafür ist, daß jede einzelne Aktion, die der Agent gemäß dem Plan ausführen muß, auch in diesem Plan erscheint.

[1] Dies ist bereits eine Einschränkung, da auch andere Ziele denkbar sind [Geo87]: Der Agent könnte z.B. daran interessiert sein, daß eine Bedingung während der Planausführung aufrechterhalten wird (z.B. daß er in einem bestimmten Zimmer bleibt), oder daß einige Aktionen in einer bestimmten Reihenfolge enthält. Derartige Ziele werden in dieser Arbeit aber nicht betrachtet.

Dies hat aber zwei Nachteile. Erstens geht dem Plan u.U. viel an Flexibilität und Allgemeinheit verloren. So kann zum Beispiel ein Plan zum Umdrehen eines Turms von Blöcken nicht ohne weiteres zum Umdrehen eines anderen Turms von Blöcken verwendet werden, wenn der Plan genau vorschreibt, wann welcher Block wohin gestellt werden muß, obwohl sich die Natur der Aufgabe nicht geändert hat. Zweitens sind Pläne häufig übermäßig detailliert. Für die Aufgabe, alle Steine, die auf dem Tisch liegen, in eine Kiste zu werfen, müsste in einem einfachen linearen oder partiell geordneten Plan für jeden Stein x die Aktion „wirf Stein x in die Kiste“ in den Plan einzutragen.

In dieser Arbeit werden daher Möglichkeiten zur angemesseneren Darstellung derartiger Pläne mit Hilfe einer Plansprache $\mathfrak{P}$ eingeführt. Zum einen gestattet es diese, einfache Entscheidungen des Agenten während der Abarbeitung des Planes darzustellen, wie zum Beispiel der Entscheidung zwischen zwei Teilplänen, je nachdem ob eine Bedingung erfüllt ist, oder der Auswahl von Parametern für die auszuführenden Aktionen gemäß dem augenblicklichen Zustand der Umwelt. Zum anderen enthält diese Sprache ein Prozedurkonzept. Damit können häufig ausgeführte Aktionssequenzen zusammengefaßt werden, so daß eine kompaktere (und damit unter Umständen leichter behandelbare) Darstellung des Planes erreicht wird. Außerdem ist mit Hilfe der Prozeduren die Darstellung von Rekursion möglich. Über die Rekursion ist auch die Iteration faßbar, d.h. z.B. die wiederholte Ausführung der gleichen Aktionssequenz solange eine Bedingung erfüllt ist.

Diese Plansprache ist so gestaltet, daß ihre Semantik sich auf einfache Weise innerhalb der klassischen Logik darstellen läßt. Dies bietet einen Vorteil gegenüber der naheliegenden Adaption von existierenden Programmiersprachen auf Planungsprobleme, deren Semantik gewöhnlich weitaus komplexer ist, so daß Fragen nach Korrektheit und Termination schwieriger zu behandeln sind [Bac86, Fra92].

Die Arbeit ist wie folgt gegliedert: In Kapitel 2 werden einige Grundlagen der Prädikatenlogik eingeführt. Kapitel 3 enthält eine Darstellung des Fluent-Kalküls, der eine bequeme Repräsentation von Aktionen und Zuständen der Planungsdomäne erlaubt. In Kapitel 4 werden Arbeiten mit vergleichbarer Zielrichtung ausgewertet, und Schlußfolgerungen zur Gestaltung einer Plansprache $\mathfrak{P}$ abgeleitet, die in Kapitel 5 zusammen mit ihrer Semantik vorgestellt wird. Kapitel 6 führt eine sogenannte Planausführungsrelation ein, die eine geeignete Darstellung von Planeigenschaften wie Korrektheit und Termination ermöglicht. Diese Relation wird dann in Kapitel 7 mit Hilfe von Prädikatenlogik dargestellt, so daß sie von automatisieren Theorembeweisern behandelt werden kann. In Kapitel 8 wird die Anwendung der Plansprache an dem sogenannten Omelettebeispiel demonstriert. Kapitel 9 faßt die Resultate der Arbeit zusammen.

2. Grundlagen

In diesem Kapitel werden die benötigten Grundlagen der Logik eingeführt, sowie die in dieser Arbeit verwendeten Konventionen beschrieben.

2.1 Logische Grundlagen

Es wird die Standardnotation und Terminologie von Lloyd [Llo87] verwendet.[1]

Definition 2.1. *Das **Alphabet** der Prädikatenlogik erster Stufe besteht aus den folgenden Arten von Symbolen:*

(i) *Variablen*
(ii) *Konstanten*
(iii) *Funktionssymbole*
(iv) *Prädikatszeichen*
(v) *Junktoren* $\neg \wedge \vee$
(vi) *Quantoren* $\exists \quad \forall$
(vii) *Begrenzer* ( ) ,

Im Folgenden werden Variablen mit x, y, z , Konstanten mit Großbuchstaben $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$, Funktionszeichen mit f, g, h und Prädikatszeichen mit p, q, r bezeichnet.

Definition 2.2. *Ein **Term** ist entweder*

(i) *eine Variable oder*
(ii) *eine Konstante oder*
(iii) $f(t_1, \ldots, t_n)$*, wobei* f *ein* n*–stelliges Funktionssymbol und* $t_1, \ldots, t_n$ *Terme sind.*

Definition 2.3. *Eine **logische Formel** ist wie folgt induktiv definiert:*

[1] Darstellung z.T. übernommen aus [Kal94].

(i) *Sei p ein n–stelliges Prädikatszeichen und $t_1, \ldots, t_n$ seien Terme. Dann ist $p(t_1, \ldots, t_n)$ eine Formel (genannt **atomare Formel** oder kürzer **Atom**).*

(ii) *Seien F und G Formeln. Dann sind auch $\neg F$, $F \wedge G$ und $F \vee G$ Formeln.*

(iii) *Sei F eine Formel und x sei eine Variable. Dann sind auch $\forall x \;\; F$ und $\exists x \;\; F$ Formeln.*

Als abkürzende Schreibweisen werden verwandt:

$F \rightarrow G$ für $(\neg F \vee G)$
$F \leftarrow G$ für $(F \vee \neg G)$
$F \leftrightarrow G$ für $(F \leftarrow G) \wedge (F \rightarrow G)$

Um die Anzahl der Klammern in den logischen Formeln zu verringern, wird in dieser Arbeit die folgende Vorrangfolge der Operatoren verwendet (nach fallender Priorität geordnet): $(), \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall \;\;, \exists$

Definition 2.4. *Ein **Literal** ist ein Atom oder die Negation eines Atoms, d.h. ein Atom mit vorangestellten $\neg$. Ein **positives Literal** ist ein Atom. Ein **negatives Literal** ist die Negation eines Atoms.*

Definition 2.5. *Für jede Formel F sei die Menge $\mathrm{var}(F)$ der **freien Variablen** von F induktiv definiert durch:*

$$\begin{aligned}
&\mathrm{var}(p(t_1, \ldots, t_n)) = \textstyle\bigcup_{i=1}^{n} \mathrm{var}(t_i),\\
&\mathrm{var}(\neg F) = \mathrm{var}(F),\\
&\mathrm{var}(F \wedge G) = \mathrm{var}(F) \cup \mathrm{var}(G),\\
&\mathrm{var}(F \vee G) = \mathrm{var}(F) \cup \mathrm{var}(G),\\
&\mathrm{var}(\exists x \;\; F) = \mathrm{var}(F) - \{x\},\\
&\mathrm{var}(\forall x \;\; F) = \mathrm{var}(F) - \{x\}
\end{aligned}$$

mit $\mathrm{var}(x) = \{x\}$, $\mathrm{var}(c) = \emptyset$ *und* $\mathrm{var}(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \mathrm{var}(t_i)$.

*Eine **abgeschlossene Formel** ist eine Formel, deren Menge der freien Variablen leer ist.*

Definition 2.6. *Eine **Prä-Interpretation** besteht aus*

(a) *Einer nichtleeren Menge D, der **Domäne** einer Prä-Interpretation,*

(b) *Eine Zuweisung von je einem Element aus D zu jeder Konstanten,*

(c) *Eine Zuweisung von Abbildungen $D^n \rightarrow D$ zu jeden n-stelligen Funktionssymbol.*

Definition 2.7. *Eine* ***Interpretation*** *besteht aus einer Prä-Interpretation J mit Domäne D und der Zuweisung einer Abbildung $D^n \to \{\mathbf{0}, \mathbf{1}\}$ zu jedem n-stelligen Prädikatssymbol. Dabei bezeichnen* $\mathbf{1}$ *und* $\mathbf{0}$ *die Wahrheitswerte „wahr“ und „falsch“.*

Definition 2.8. *Eine* ***Variablenbelegung*** *bezüglich einer Prä-Interpretation J ist die Zuweisung eines Elements der Domäne von J zu jeder Variablen.*

Definition 2.9. *Der* ***Wert eines Terms*** *bezüglich einer Interpretation I und einer Variablenbelegung V ist:*

(a) *für jede Variable das ihr zugewiesene Element entsprechend V,*
(b) *für jede Konstante das ihr zugewiesene Element entsprechend I,*
(c) *Wenn $t'_1, \ldots, t'_n$ die Werte von $t_1, \ldots, t_n$ sind, und f' die dem n-stelligen Funktionssymbol f durch I zugewiesene Abbildung, dann ist $f'(t'_1, \ldots, t'_n)$ der Wert von $f(t_1, \ldots, t_n)$.*

Definition 2.10. *Der* ***Wert einer Formel*** *bezüglich einer Interpretation I und einer Variablenbelegung V ist wie folgt definiert:*

(a) *Für ein Atom $p(t_1, \ldots, t_n)$ ist der Wert $p'(t'_1, \ldots, t'_n)$, wobei p' die Abbildung ist, die p durch I zugewiesen wird, und $t'_1, \ldots, t'_n$ die Werte der Terme $t_1, \ldots, t_n$ bezüglich I und V.*
(b) *Der Wert von $\neg F$ ist $\mathbf{1}$, wenn der Wert von F $\mathbf{0}$ ist, und umgekehrt. Der Wert von $(F \wedge G)$ ist genau dann $\mathbf{1}$, wenn die Werte von F und G $\mathbf{1}$ sind, und sonst $\mathbf{0}$. Der Wert von $(F \vee G)$ ist genau dann $\mathbf{0}$, wenn die Werte von F und G $\mathbf{0}$ sind, und sonst $\mathbf{1}$.*
(c) *Der Wert einer Formel $(\exists x \;\; F)$ ist genau dann $\mathbf{1}$, wenn ein $d \in D$ existiert, so daß F den Wert $\mathbf{1}$ bezüglich I und $V(x/d)$ hat, wobei $V(x/d)$ äquivalent zu V ist, außer daß x der Wert d zugewiesen wird. Sonst ist der Wert von $(\exists x \;\; F)$ $\mathbf{0}$.*
(d) *Der Wert einer Formel $(\forall x \;\; F)$ ist genau dann $\mathbf{1}$, wenn für alle $d \in D$ die Formel F den Wert $\mathbf{1}$ bezüglich I und $V(x/d)$ hat. Sonst ist der Wert von $(\forall x \;\; F)$ $\mathbf{0}$.*

Der Wert einer geschlossenen Formel hängt demnach nicht von der Variablenbelegung ab.

Definition 2.11. *Eine Interpretation ist ein* ***Modell*** *einer Menge von geschlossenen Formeln, wenn alle Formeln bezüglich dieser Interpretation den Wert* $\mathbf{1}$ *haben.*

Definition 2.12. *Eine geschlossene Formel F* ***folgt*** *aus einer Formelmenge (G) (Schreibweise $(G) \models F$), wenn F bezüglich jeden Modells von (G) den Wert* **1** *hat.*

Definition 2.13. *Eine* ***Substitution*** *ist eine endliche Menge der Form $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$, wobei die v_i Variablen sind, die t_i Terme, die $v_1, \ldots, v_n$ paarweise verschieden sind, sowie $v_i \neq t_i$ für $i = 1, \ldots, n$. Die Menge der v_i ist die* ***Domäne*** $\mathrm{dom}(\sigma)$, *die Menge der t_i ist die* ***Kodomäne*** $\mathrm{cod}(\sigma)$.

Definition 2.14. *Ein* ***Ausdruck*** *ist entweder ein Term, ein Literal oder eine Konjunktion oder Disjunktion von Literalen.*

Definition 2.15. *Die* Anwendung *$E\sigma$ einer Substitution $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$ auf einen Ausdruck E ist der Ausdruck, der aus E durch gleichzeitige Ersetzung jedes Vorkommens der Variablen v_i durch den entsprechenden Term t_i. $E\sigma$ wird als* ***Instanz*** *von E bezeichnet. Eine Instanz mit $\mathrm{var}(E\sigma) = \emptyset$ wird als* ***Grundinstanz*** *bezeichnet.*

Definition 2.16. *Die* Komposition *$\sigma\theta$ zweier Substitutionen σ und θ ist die Substitution*

$$\sigma\theta = \{x/t \in \theta \mid x \notin \mathrm{dom}(\sigma)\} \cup \{x/t\theta \mid x/t \in \sigma \ \wedge \ x \neq t\theta\}$$

Definition 2.17. *Wenn für zwei Terme s und t eine Substitution σ existiert, so daß $s\sigma = t\sigma$, so wird diese* ***Unifikator*** *dieser Terme genannt. Ein Unifikator von s und t wird* ***allgemeinster Unifikator*** $\mathrm{mgu}(s,t)$ *von s und t genannt, wenn für jeden Unifikator θ von s und t eine Substitution γ existiert, so daß $\theta = \sigma\gamma$.*

2.2 Notationen und Konventionen

In dieser Arbeit werden die folgenden Schreibweisen verwendet:

Für allgemeine logische Formeln:

- Variablen werden mit x,y,z ... bezeichnet.
- Konstanten werden mit großen Buchstaben $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ bezeichnet.
- Funktionen werden mit f, g, h bezeichnet.
- Prädikate werden mit p, q, r bezeichnet.

Die Priorität der logischen Operatoren ist (nach fallender Priorität geordnet): $(), \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists$.

Im Kontext von Planungsproblemen:

- Fluentnamen, Aktionsnamen und Bedingungsnamen werden schräggestellt: $on, cl, totabl$.
- Objekte werden mit Großbuchstaben $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ gesetzt.
- Namen für Strukturen (Tupel von Mengen, Relationen usw.) werden in serifenloser Schrift gesetzt: D, W .
- Mengen und Relationen werden mit kalligraphischen Buchstaben bezeichnet: $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots,$
- Multimengen werden als $\mathbb{A}, \mathbb{B}, \mathbb{C}, \ldots$ gesetzt,
- Namen für Prädikate werden kursiv gesetzt: $append, causes, doplan$.
- Variablen werden wie folgt bezeichnet:
  - a für Aktionsbezeichner,
  - b für Prozedurkörper („body“).
  - c für Bedingungsbezeichner („condition“),
  - h für Prozedurköpfe („head“),
  - o für Objekte,
  - p für Pläne (bzw. Vorhaben im Sinne von Kap. 5),
  - s für Zustände („state“),

  Diese werden zum Teil mit Index versehen.

Diese Benennungen haben in logischen Formeln Vorrang gegenüber die oben eingeführten allgemeinen Konventionen.

Im Unterschied zu Implikationen ($\rightarrow$) und Äquivalenzen in logischen Formeln werden in Sätzen die Operatoren $\Longleftrightarrow$, $\Longrightarrow$ sowie $\Longleftarrow$ im Text als Abkürzungen verwendet:

$A \iff B$ A gilt genau dann, wenn B gilt.

$A \implies B$ Aus A folgt B .

$A \impliedby B$ A gilt dann, wenn B gilt.

3. Darstellung von Zuständen und Aktionen mit dem Fluent-Kalkül

In [HS89, HS90] wurde eine reifizierte Darstellung von Zuständen mit Hilfe des Operators $\circ$ eingeführt (neuerdings unter dem Begriff ***Fluent-Kalkül*** *bekannt), die, im Gegensatz zu auf dem Situationskalkül basierenden Darstellungen, die Notwendigkeit von Rahmenaxiomen vermeidet, und damit eine effiziente Lösung des Implementationsaspekts des Rahmenproblems bildet.*

Im Unterschied zu anderen Ansätzen [Bib86, MTV90], die ebenfalls keine Rahmenaxiome benötigen, sind im Fluent-Kalkül Pläne und Zustände durch Terme, d.h. Objekte erster Klasse, dargestellt. Dadurch eignet sich dieser Ansatz sehr gut für diese Arbeit, da über die Effekte von verschiedenen Plänen geschlossen werden soll.

Im folgenden wird eine Kurzdarstellung des Fluent-Kalküls und einer einfachen Anwendung zur Aktionsbeschreibung gegeben.

3.1 Einführung

Zunächst zu einigen Grundbegriffen.

Definition 3.1. *Ein* ***Situation*** *ist der vollständige Zustand*[1] *(„Schnappschuß“) des dynamischen Systems zu einem Zeitpunkt.*[2] *Ein* ***Fluent*** *ist eine Funktion auf der Menge der Situationen. Ein* ***propositionales Fluent*** *ist ein Fluent, dessen Wertebereich die Menge der Wahrheitswerte* $\mathbf{1}, \mathbf{0}$ *ist. Fluenten mit den natürlichen Zahlen als Wertebereich werden als* ***Ressourcen*** *bezeichnet, und Fluenten, deren Wert eine Funktion ist, als* ***funktionale Fluenten****. Ein* ***Zustand*** *ist eine Wertebelegung für alle Fluenten.*

[1] im umgangssprachlichen Sinne

[2] Die Bedeutung des Begriffes „Situation“ ist in der KI-Literatur nicht einheitlich [San94].

Beispiel 3.2. *In einem dynamischen System „Schreibtisch“ seien 4 Fluenten definiert: ein Fluent stiftzahl, das die Anzahl der Stifte angibt, ein Fluent licht, daß angibt, ob das Licht angeschaltet ist, ein Fluent schreibstift, daß angibt, welchen Stift der Schreibende in der Hand hält, und ein Fluent stiftfarben, dessen Wert eine Funktion ist, die jedem Stift seine Farbe zuordnet. licht ist also ein propositionales Fluent, stiftzahl eine Ressource, und stiftfarben ein funktionales Fluent.*

Für die Situation an einem bestimmten Tag um 8.00 könnten z.B. die Fluenten folgende Werte haben: licht: $\mathbf{1}$ *, stiftzahl: 2, schreibstift: Bleistift. Der Zustand ist also*

$$\begin{aligned}\{&\mathit{licht} \rightarrow \mathbf{1},\\ &\mathit{stiftzahl} \rightarrow 2,\\ &\mathit{schreibstift} \rightarrow \mathit{Bleistift},\\ &\mathit{stiftfarben} \rightarrow \{(\mathit{Bleistift}, \mathit{schwarz}), (\mathit{Kugelschreiber}, \mathit{blau})\}\}\end{aligned}$$

Es ist möglich, daß die Situation um 9.00 Uhr eine völlig andere ist, z.B. eine andere Person am Schreibtisch sitzt, aber trotzdem der gleiche Zustand herrscht.

Eines der Schlüsselprobleme bei der Repräsentation dynamischer Systeme mit Hilfe der klassischen Logik ist die Darstellung von zeitlicher Veränderung. Da logischen Fakten mit einer Interpretation ein statischer logischer Wahrheitswert zugeordnet wird, können sie nicht ohne weiteres zur Repräsentation einer veränderlichen Große, d.h. eines Fluents, dienen. Für die Lösung dieses Problems wurden verschiedene Wege gegangen.

Eine Möglichkeit ist, den Rahmen der klassischen Logik zu verlassen. Dies führt zum Beispiel zu der Vielzahl von Modallogiken. Dort existiert nicht mehr eine einzelne Interpretation, sondern im Rahmen einer Kripkestruktur[3] eine Menge von Interpretationen, die möglichen Welten entsprechen. Es werden nun Modaloperatoren in die Logik eingeführt, mit Hilfe derer sich Beziehungen zwischen den möglichen Welten ausdrücken lassen.

Andere Autoren sahen nicht die Notwendigkeit, die klassische Logik zu verlassen. So schlug John McCarthy in [McC63] vor, einen Situationsbezeichner als zusätzliches Argument für Fluenten einzuführen, um den Wert eines Fluents in einer bestimmten Situation zu bezeichnen. So könnte z.B. der logische Fakt $\mathrm{auf}(s, \mathrm{vase}, \mathrm{tisch})$ bedeuten, daß die Vase in der Situation s auf dem Tisch steht. Dieser Ansatz ist unter dem Namen **Situationskalkül** bekannt.

In der Grundvariante des Situationskalküls wird die Anfangssituation als s_0 bezeichnet. Andere Situationen werden mit Hilfe eines funktionalen Fluents *do*

[3] Eine ausführlichere Darstellung ist z.B. in [BHS93a] zu finden.

gebildet.[4] Dieses ordnet einer Aktion a die Situation $do(a, s)$ zu, die nach Ausführung der Aktion in Situation s entsteht. Wenn eine Sequenz von Aktionen $a_1, \ldots, a_n$ ausgeführt wurde, entsteht ein Term als Situationsbezeichner, der die Aktionssequenz notiert: $do(a_n, do(a_{n-1}, \ldots, do(a_1, s_0) \ldots))$.[5] [6]

Wenn nun ein Mensch über die Veränderung der Welt nachdenkt, so macht er gewöhnlich die Annahme, daß sich Dinge, die nicht explizit verändert werden, nicht verändern. Wenn zum Beispiel am Schreibtisch das Licht angeschaltet wird, so ist anzunehmen, daß die Anzahl der Stifte danach noch die gleiche ist. Die Frage, wie dieses Trägheitsgesetz modelliert wird, ist als der Implementationsaspekt des Rahmenproblems bekannt. Im Situationskalkül wird dies durch die sogenannten Rahmenaxiome behandelt. Um z.B. auszudrücken, daß das Anschalten des Lichtes die Anzahl der Stifte nicht verändert, könnte man folgendes Axiom verwenden:

$$\forall s \quad \text{stiftzahl}(do(\text{licht\_an}, s)) = \text{stiftzahl}(s)$$

Die Anzahl dieser Axiome ist u.U. sehr groß.[7] so daß die deduktive Behandlung dynamischer Systeme erschwert wird.

Inzwischen sind mehrere Ansätze [Bib86, MTV90, HS89] bekannt, die ohne diese Rahmenaxiome auskommen.[8] Ihnen ist gemeinsam, daß sie auf eine explizite Verknüpfung der Fluenten mit der Situation verzichten. Die Ansätze von W. Bibel [Bib86] mit Hilfe der linearen Konnektionsmethode, und von Masseron et. al. [MTV90] besitzen jedoch keine explizite Repräsentation von Zuständen und Plänen, und eignen sich nach Meinung des Autors daher schlecht, um über die verschiedene Ausführungsmöglichkeiten von Plänen zu schlußfolgern, we das in dieser Arbeit (Kapitel 7) angestrebt ist. Daher wird in dieser Arbeit der inzwischen als Fluent-Kalkül bezeichnete Ansatz von Hölldobler und Schneeberger [HS89] verwendet.

[4] Statt do wird oft auch $result$ geschrieben (McCarthy's originaler Vorschlag).

[5] Der Term $do(a_n, \ldots, do(a_1, s_0))$ wird oft abkürzend geschrieben als $do([a_1, \ldots, a_n], s_0)$.

[6] Da es sich bei do um eine Funktion handelt, kann mit dieser Grundvariante Nichtdeterminismus nicht modelliert werden. Es gibt jedoch Erweiterungen / andere Interpretationen, die diesen Nachteil nicht haben.

[7] in der Größenordnung von nm, wenn n die Anzahl der Fluenten und m die Anzahl der Aktionen ist. Diese Zahl wird aber z.B. in der Formulierung von Reiter [Rei91] unter bestimmten Bedingungen auf n reduziert („successor state axioms“).

[8] Deren Äquivalenz für sogenannte konjunktive Planungsprobleme wurde in [GHS96] nachgewiesen.

3.2 Beschreibung von Zuständen im Fluent-Kalül

Die Grundidee des **Fluent-Kalküls** ist es, die Zustände des dynamischen Systems zu reifizieren, d.h. als einen sogenannten Fluent-Term darzustellen. In diesem werden Fluent-Ausdrücke zu einem Term, der den Zustand darstellt, verknüpft:

Definition 3.3. *Ein **Fluent-Term** ist ein Fluent*[9] *oder zwei mit dem Operator* $\circ$ *verknüpfte Fluent-Terme. Ein Fluent-Literal ist ein Fluent* f *oder seine Negation* $\overline{f}$ *.*

Beispiel 3.4 (Safe). *Ein (sehr einfacher) Safe habe zwei Hebel, die beide nach oben bewegt werden müssen, um den Safe zu öffnen. Der Zustand des Systems sei durch folgende Fluenten charakterisiert:*
$h1$ *: Hebel 1 ist oben,*
$h2$ *: Hebel 2 ist oben,*
o *: die Tür ist offen.*
Die dazu komplementären Fluenten (d.h. Hebel 1 ist unten, . . .) werden durch Überstreichen gekennzeichnet: $\overline{h1}, h2, \overline{o}$ *.*
Der rechts abgebildete Zustand ist also durch die Fakten $\overline{h1}$ *,* $h2$ *und* $\overline{o}$ *charakterisiert.*
Dieser Zustand läßt sich also als $\overline{h1} \circ h2 \circ \overline{o}$ *darstellen.*

Entsprechend dieser Semantik sollte es natürlich gleichgültig sein, in welcher Reihenfolge man die Fakten aufschreibt, d.h. ob man $\overline{h1} \circ h2 \circ \overline{o}$ oder $\overline{h1} \circ \overline{o} \circ h2$ oder $\overline{o} \circ \overline{h1} \circ h2$ schreibt. Der Operator sei also kommutativ und assoziativ. Man kann also Klammern weglassen.

Die leere Zustandsbeschreibung wird mit $\emptyset$ notiert. Wenn man diese zu einem Term hinzufügt, repräsentiert dieser den gleichen Zustand, d.h. $\emptyset$ bildet das Einselement von $\circ$. Man erhält also die folgenden Eigenschaften[10]:

$$
\begin{array}{rll}
x \circ (y \circ z) = (x \circ y) \circ z & \text{(Assoziativität)} & \\
x \circ y = y \circ x & \text{(Kommutativität)} & \\
x \circ \emptyset = x & \text{(Einselement)} & \\
x = x & \text{(Reflexivität)} & \text{(AC1)} \\
x = y \;\rightarrow\; y = x & \text{(Symmetrie)} & \\
x = y \;\wedge\; y = z \;\rightarrow\; x = z & \text{(Transitivität)} & \\
x = y \;\rightarrow\; (W[x] \;\leftrightarrow\; W[y]) & \text{(Substitutivität)} &
\end{array}
$$

[9] bzw. die Repräsentation eines Fluents
[10] Alle Variablen in den folgenden Formeln sind implizit allquantifiziert.

Dabei bezeichnet $W[x] \leftrightarrow W[y]$ die Äquivalenz der Formel W, die die Variable x enthält, und der Formel, die durch Ersetzen von x durch y entsteht [HT95].

Eine Gleichheit im Sinne dieser Axiome **(AC1)** wird gewöhnlich als $x =_{\mathrm{AC1}} y$ dargestellt.

Es besteht eine enge Beziehung dieser Terme zu sogenannten Multimengen. Eine Multimenge ist eine Erweiterung des Konzepts der klassischen Menge, so daß Elemente auch mehrfach Vorkommen können. (Die Anzahl des Vorkommens eines Elements ist relevant.) Formal lassen sich Multimengen wie folgt definieren:

Definition 3.5. *Eine* ***Multimenge*** $\mathbb{M}$ *ist eine Abbildung aus einer Menge* $\mathcal{M}$ *in die Menge der natürlichen Zahlen* $\mathbb{N}$ *. Die Menge aller Multimengen über* $\mathcal{M}$ *wird geschrieben als* $\mathbb{N}^{\mathcal{M}}$ *. In Analogie zu den klassischen Mengen schreibt man* $e \in_k \mathbb{M}$ *(*e *ist* k*-faches Element der Multimenge), wenn* $\mathbb{M}$ e *auf* k *abbildet. Für* $e \in_0 \mathbb{M}$ *wird auch* $e \notin \mathbb{M}$ *geschrieben.*

Notiert werden Multimengen durch Aufzählung ihrer Elemente in den Klammern $\{\!|$ *und* $|\!\}$ *. Jedes Element, daß* k*-fach in der Multimenge vorkommt, muß dabei genau* k *mal in der Aufzählung vorkommen. Die Reihenfolge der Elemente in der Aufzählung ist nicht relevant. Die leere Multimenge wird als* $\{\!\}$ *bezeichnet.*

Für zwei Multimengen $\mathbb{M}_1, \mathbb{M}_2 \in \mathbb{N}^{\mathcal{M}}$ *sind die Vereinigung* $\mathbb{M}_1 \dot\cup \mathbb{M}_2$ *, die Differenz* $\mathbb{M}_1 \dot- \mathbb{M}_2$ *sowie die Teilmengenrelation* $\mathbb{M}_1 \dot\subseteq \mathbb{M}_2$ *wie folgt definiert:*

$$\forall e \in \mathcal{M} \quad e \in_m \mathbb{M}_1 \ \wedge\ e \in_n \mathbb{M}_2 \ \rightarrow\ e \in_{m+n} \mathbb{M}_1 \dot\cup \mathbb{M}_2 .$$

$$\forall e \in \mathcal{M} \quad e \in_m \mathbb{M}_1 \ \wedge\ e \in_n \mathbb{M}_2 \ \rightarrow\ e \in_k \mathbb{M}_1 \dot\cup \mathbb{M}_2 \quad \textit{mit} \quad k = \begin{cases} m-n & \textit{für} \quad m > n \\ 0 & \textit{sonst.} \end{cases}$$

$$\mathbb{M}_1 \dot\subseteq \mathbb{M}_2 \ \leftrightarrow\ \forall e \in \mathcal{M} \ \big(e \in_m \mathbb{M}_1 \ \wedge\ e \in_n \mathbb{M}_2 \ \rightarrow\ m \leq n\big).$$

Beispiel 3.6. *Für die Multimengen* $\mathbb{M}_1 = \{\!| a, b, a |\!\}$ *und* $\mathbb{M}_2 = \{\!| a, b |\!\}$ *gelten die Beziehungen* $a \in_2 \mathbb{M}_1$ *,* $b \in_1 \mathbb{M}_1$ *,* $\mathbb{M}_2 \dot\subseteq \mathbb{M}_1$ *,* $\mathbb{M}_1 \dot\cup \mathbb{M}_2 = \{\!| a, a, a, b, b |\!\}$ *sowie* $\mathbb{M}_1 \dot- \mathbb{M}_2 = \{\!| a |\!\}$ *.*

Klassische Mengen erscheinen nun als Spezialfall von Multimengen, in dem jedes Element maximal einmal in der Menge vorkommt.[11]

[11] Dies ist allerdings keine Einbettung, da die Resultate der Mengenoperationen sich von den Resultaten der korrespondierenden Multimengenoperationen unterscheiden: $\{a\} \cup \{a\} = \{a\}$, aber $\{\!| a |\!\} \dot\cup \{\!| a |\!\} = \{\!| a, a |\!\}$. Unter bestimmten Bedingungen stimmen sie allerdings überein: so entspricht z.B. das Resultat der Vereinigung zweier Multimengen der Vereinigung der entsprechenden Mengen, falls die Multimengen disjunkt sind, d.h. $\forall e \in \mathcal{M} \quad e \notin \mathbb{M}_1 \ \vee\ e \notin \mathbb{M}_2$.

Man kann eine Abbildungen $^{\tau}$ und $^{\tau^{-1}}$ so definieren, daß die folgenden Gleichungen gelten:

$$\{\!|\ s_1, \ldots, s_n\ |\!\}^{\tau} =_{\mathrm{AC1}} s_1 \circ \ldots \circ s_n \circ \emptyset \tag{3.7a}$$

$$(s_1 \circ \ldots \circ s_n \circ \emptyset)^{\tau^{-1}} = \{\!|\ s_1, \ldots, s_n\ |\!\} \tag{3.7b}$$

Es ergeben sich folgende Eigenschaften für diese Abbildungen [HT95]:

$$(\mathbb{S}^{\tau})^{\tau^{-1}} = \mathbb{S} \tag{3.8}$$

$$(s^{\tau^{-1}}) =_{\mathrm{AC1}} s^{\tau} \tag{3.9}$$

$$\mathbb{S}^{\tau} =_{\mathrm{AC1}} s \iff \mathbb{S} = s^{\tau^{-1}} \tag{3.10}$$

$$\mathbb{S}_1{}^{\tau} \circ \mathbb{S}_2{}^{\tau} = (\mathbb{S}_1 \dot{\cup} \mathbb{S}_2)^{\tau} \tag{3.11}$$

Wegen (3.11) ist es möglich, zwei Teile x und y zu einer neuen Zustandsbeschreibung z durch die Gleichung $x \circ y =_{\mathrm{AC1}} z$ zusammenzufügen. Umgekehrt läßt sich mit Hilfe dieser Gleichung einen Teil x der Zustandsbeschreibung z abzutrennen. Dies ist auch der Grund, warum auf eine Idempotenz ($x \circ x =_{\mathrm{AC1}} x$) von $\circ$ verzichtet wird[12]. Dadurch ist garantiert, daß alle Fluenten von z entweder in x oder in y erscheinen, aber *nicht* in beiden. Die Fluenten werden als als Ressourcen behandelt, die nicht beliebig vervielfältigt werden können, sondern produziert und konsumiert werden können. Dies spiegelt sich auch im benutzten Aktionskonzept wieder, daß im folgenden Abschnitt beschrieben wird.

Es sind zwei Unterschiede des Fluent-Kalküls gegenüber dem Situationskalkül zu erkennen.

Erstens ist die Interpretation eines Fluent-Terms nicht durch die Logik festgelegt. Tatsächlich werden diese Terme in verschiedenen Arbeiten unterschiedlich genutzt. Im einfachsten Falle werden (wie die Multimengen-Interpretation nahelegt) nur Ressourcen mit den Fluent-Termen dargestellt [Höl92]: wenn das Fluent den Wert k hat, ist es k-faches Element des Terms: Wenn z.B. der Agent 5 Quarter in der Tasche hat, so wird dieser Zustand durch $q \circ q \circ q \circ q \circ q$ repräsentiert.[13] In [HS89] wird dagegen die oben erwähnte Korrespondenz von Mengen zu Multimengen mit 0 und 1-fachen Elementen zur Darstellung von Zuständen mit propositionalen Fluenten genutzt: die Menge sämtlicher propositionaler Fluenten, die im aktuellen Zustand gelten, wird als Fluent-Term dargestellt. Damit ist es allerdings nötig, dem Problem der Konsistenz einer

[12] $x \circ x =_{\mathrm{AC1}} x$ gilt hier nur dann, wenn $x =_{\mathrm{AC1}} \varepsilon$.

[13] Diese Darstellung von Ressourcen ist natürlich nur für relativ kleine Anzahlen angemessen. Für Ressourcen mit großem Wertebereich erscheint es effizienter Arithmetik zu benutzen.

Zustandsdarstellung mehr Aufmerksamkeit zu widmen, da nicht jeder entstehende Fluent-Term tatsächlich einen Zustand darstellt (zum Beispiel muß nun garantiert werden, daß Fluenten nicht mehrfach auftreten).

In [Thi96, Thi97] werden ebenfalls nur propositionale Fluenten dargestelt: ein Fluent-Term enthält aber für jedes propositionale Fluent genau ein Fluent-Literal (positiv oder negativ, je nach dem welchen Wert das Fluent im dargestellten Zustand hat). Es sind auch Mischformen denkbar: in [Leh97] werden beide erwähnten Arten von Fluent-Termen nebeneinander verwendet, um sowohl propositionale Fluenten als auch Ressourcen erfassen zu können. In [Ede95, EHT96] werden z.T. Fluent-Terme innerhalb von Fluent-Termen durch einen einstelligen Operator $\boxed{\_}$ eingekapselt, um Hierarchien für sogenanntes hierarchisches Planen darstellen zu können.

Im Gegensatz zu dieser Flexibilität des Fluent-Kalküls ist dagegen im Situationskalkül die Interpretation der Fluenten durch die Logik festgelegt: wenn ein Fluent als Atom repräsentiert wird, ist es ein propositionales Fluent, wenn es als Term repräsentiert wird, ist es ein funktionales Fluent bzw. eine Ressource. Ein Vorteil davon ist, daß logische Abhängigkeiten zwischen den Fluenten unmittelbar mit Hilfe der Logik darstellbar sind (zum Beispiel könnte ein Axiom

$$\forall s, o \;\; \text{auf}(s, o, \text{tisch}) \;\rightarrow\; \text{in}(s, o, \text{zimmer})$$

aussagen, daß sich alles, was auf dem Tisch steht, auch im Zimmer befindet). Im Fluent-Kalkül wurden logische Abhängigkeiten, falls sie nicht durch die Art der Interpretation der Fluent-Terme erfasst werden, bisher nur durch eine Art „Nachbearbeitung“ von Fluent-Termen dargestellt, wie das z.B. in [Thi96, Thi97] geschieht.

Ein zweiter Unterschied von Fluent-Kalkül und Situationskalkül ist, daß nur noch eine endliche Anzahl von Fluenten direkt dargestellt werden kann, da Fluent-Terme endlich sind, während im Situationskalkül auch die Darstellung unendlich vieler Fluenten möglich ist. [14]

3.3 Ein einfaches Aktionskonzept

Eine Aktionsdarstellung ähnlich der in **STRIPS** [FN71, Lif86] läßt sich nun sehr einfach mit Hilfe des Fluent-Kalküls darstellen. Ein Kennzeichen von STRIPS ist, daß Zustände durch eine endliche Menge von propositionalen Fluenten dargestellt werden, deren Wert **1** ist.[15] Die Mengendarstellung entspricht hierbei

[14] Die Darstellung unendlich vieler Fakten im Zusammenhang mit logischen Abhängigkeiten zwischen diesen Fakten ist allerdings auch im Situationskalkül recht problematisch: es ist dabei i.A. ein Ausweichen auf die Prädikatenlogik 2-ter Stufe notwendig [LR96].

[15] Dies ist eigentlich eine Vereinfachung in Vergleich zu STRIPS. In [FN71] ist von „wohlgeformten Formeln“ die Rede, es werden aber tatsächlich nur Konjunktionen propositionale Fluenten für die Zustandsdarstellung benutzt.

einer Kunjunktion der Fluenten der Menge. Diese Mengen, aufgefaßt als Multimengen, lassen sich nun mit Hilfe des Operators $^{\tau}$ äquivalent als Fluent-Terme darstellen.

Eine *Aktionen* wird durch ein Tripel (c, a, e) von Vorbedingungen (*C*onditions), *A*ktionsbezeichner und *E*ffekten dargestellt. c und e sind dabei Fluent-terme, entsprechend einer Menge von Fluenten. Die Aktion kann also in einem Zustand s nur ausgeführt werden, wenn alle ihre Vorbedingungen erfüllt sind, d.h. c in s enthalten ist: $c^{\tau^{-1}} \dot{\subseteq} s^{\tau^{-1}}$. Anders formuliert: Es existiert eine Multimenge $x^{\tau^{-1}}$, so daß $c^{\tau^{-1}} \dot{\cup} x^{\tau^{-1}} = s^{\tau^{-1}}$ bzw. $c \circ x =_{\mathrm{AC1}} s$. Wenn diese Gleichung erfüllt wird, enthält x sämtliche Fluenten, die den Zustand charakterisieren, außer den Fluenten, die in den Vorbedingungen enthalten sind. Dieser Nebeneffekt wird genutzt, um das Trägheitsgesetz zu modellieren: in x sind all die Fluenten enthalten, die nicht durch die Aktion verändert werden.

Im Unterschied zu STRIPS ist die Wirkung der Aktion, daß die Vorbedingungen aus dem Anfangszustand s entfernt werden, und dann die Effekte hinzugefügt werden.[16] Formal bedeutet dies, daß die Gleichungen

$$s =_{\mathrm{AC1}} x \circ c \tag{3.12}$$
$$s' =_{\mathrm{AC1}} x \circ e \tag{3.13}$$

gelöst werden.

Beispiel 3.14 (Safe, Fortsetzung). *Die Aktion „bewege Hebel 1 nach oben“ sei durch das Tripel*

$$(\overline{h1},\ \mathit{hoch(h1)},\ h1)$$

charakterisiert. Für die Ausführung von hoch(h1) im Zustand $s =_{\mathrm{AC1}} \overline{h1} \circ h2 \circ \overline{o}$ *stellen sich* (3.12) *und* (3.13) *so dar:*

$$\overline{h1} \circ h2 \circ \overline{o} =_{\mathrm{AC1}} x \circ \overline{h1}$$
$$s' =_{\mathrm{AC1}} x \circ h1$$

Diese werden genau durch $x =_{\mathrm{AC1}} h2 \circ \overline{o}$ *und* $s' =_{\mathrm{AC1}} h1 \circ h2 \circ \overline{o}$ *gelöst, so daß sich der Safe nach Ausführung von* $\mathit{hoch}(h1)$ *im Zustand* $h1 \circ h2 \circ \overline{o}$ *befindet.*

Hieran ist zu erkennen, wie das Rahmenproblem gelöst wird: sämtliche Fluenten aus s, die nicht in den Vorbedingungen c vorkommen, erscheinen statt dessen in x, und werden durch die zweite Gleichung mit in den resultierenden Zustand s' übernommen.[17]

[16] Da alle Vorbedingungen entfernt werden, sind die Vorbedingungen, die unverändert gültig bleiben, ebenfalls als Effekt der Aktion aufzuführen, damit sie wieder hinzugefügt werden.

[17] Ein Problem, das sich dabei ergeben kann ist, daß der Zustand inkonsistent werden könnte, d.h. daß sich eine widersprüchliche Aussage wie z.B: $s =_{\mathrm{AC1}} \overline{h1} \circ h1$ ergibt. Dieses Problem wird z.B. in [HS90] dadurch behandelt, daß ein einstelliges Prädikat *consistent* eingeführt wird, das nur für konsistente Zustände gilt. Dieses wird dann an geeigneten Stellen geprüft.

3.4 Darstellung von Plänen

Als Beispiel sei hier die logische Modellierung der Planausführung analog zu [HT95] dargestellt.[18]

Die Vorbedingungen und Effekte der einzelnen Aktionen werden mit Hilfe eines dreistelligen Prädikats $action(c, a, e)$ dargestellt. Dieses sei wahr, wenn (c, a, e) eine Aktion der Domäne darstellt. Wenn nun die Tripel (c_i, a_i, e_i) $(i = 1 \ldots m)$ sämtliche Aktionen der Domäne beschreiben, so läßt sich dies durch m Formeln darstellen:

$$action(c_i, a_i, e_i). \qquad \text{für} \quad i = 1 \ldots m \tag{3.15}$$

Ein linearer Plan wird als eine endliche Sequenz $a_1; a_2; \ldots; a_n; \varepsilon$ von Aktionen dargestellt, die mit Hilfe des rechtsassoziativen binären Operators ; notiert werden. Abgeschlossen wird ein Plan mit den Symbol ε als Endemarkierung, das den leeren Plan darstellt. Diese wird nun von links nach rechts ausgeführt. Das dreistellige Prädikat $causes(s, a_1; \ldots; a_n; \varepsilon, s')$ wird benutzt, um auszudrücken, daß die Aktionssequenz $a_1; \ldots; a_n; \varepsilon$ von Zustand s in Zustand s' führt. Dies wird durch die beiden folgenden logischen Formeln ausgedrückt:

$$\forall s, s' \quad causes(s, \varepsilon, s') \leftarrow s = s' \tag{3.16}$$

$$\begin{aligned} \forall s, s', a, p, c, e, x \quad causes(s,\ a; p\ , s') \leftarrow\ & action(c, a, e) \ \wedge \\ & c \circ x =_{\text{AC1}} s \ \wedge \\ & causes(x \circ e, p, s'). \end{aligned} \tag{3.17}$$

Diese entsprechen einer rekursiven Definition der Planausführung. Formel (3.16) stellt die Endebedingung dar: wenn keine Aktionen mehr auszuführen sind entspricht der Anfangszustand s dem Endzustand s'. Wenn dagegen noch ein Plan $a; p$ auszuführen ist (3.17), so sind mit $action(c, a, e)$ die Vorbedingungen und Effekte der ersten auszuführenden Aktion a zu bestimmen, diese mit Hilfe der Gleichung $c \circ x =_{\text{AC1}} s$ (analog zu (3.12) auf Seite 16) aus dem gegenwärtigen Zustand zu entfernen, die Effekte hinzuzufügen ($x \circ e$, analog zu (3.13)), und in diesem neuen Zustand ist der Rest des Plans auszuführen $causes(x \circ e, p, s')$.

Beispiel 3.18 (Safe, Fortsetzung). *Für das Safe-Beispiel stellen sich die Aktionen als logische Formeln wie folgt dar:*

$action(\overline{h1},\ hoch(h1),\ h1).$

$action(\overline{h2},\ hoch(h1),\ h2).$

$action(h1 \circ h2 \circ \overline{o},\ \ddot{o}ffnen,\ h1 \circ h2 \circ o).$

[18] Hier wird eine leichte Vereinfachung vorgenommen: Das Konzept der sogenannten Spezifität wird nicht betrachtet.

Aus (3.16) *ergibt sich mit* $s = s' = h1 \circ h2 \circ o$ *daß*

$$causes(h1 \circ h2 \circ o, \varepsilon, h1 \circ h2 \circ o)$$

gilt; aus (3.17) *ergibt sich mit*

$$s = h1 \circ h2 \circ \overline{o} \qquad p = \varepsilon \qquad s' = h1 \circ h2 \circ o$$
$$c = h1 \circ h2 \qquad a = \textit{öffnen} \qquad e = h1 \circ h2 \circ o$$

daß

$$causes(h1 \circ h2 \circ \overline{o},\ \textit{öffnen}; \varepsilon\ , h1 \circ h2 \circ o).$$

und mit

$$s = \overline{h1} \circ h2 \circ \overline{o} \qquad p = \textit{öffnen}; \varepsilon \qquad s' = h1 \circ h2 \circ \overline{o}$$
$$c = \overline{h1} \qquad a = hoch(h1) \qquad e = h1$$

ergibt sich aus (3.17)

$$causes(\overline{h1} \circ h2 \circ \overline{o},\ hoch(h1); \textit{öffnen}; \varepsilon\ , h1 \circ h2 \circ o).$$

Es läßt sich also ableiten, daß die Ausführung des Planes $hoch(h1); \textit{öffnen}; \varepsilon$ *aus dem in Beispiel 3.4 auf Seite 12 gegebenen Zustand in einen Zustand führt, in dem der Safe offen ist (d.h. in dem das Fluent* o *enthalten ist).*

3.5 Weitere Arbeiten zum Fluent-Kalkül

Es wurden Erweiterungen des Aktionsmodells vorgenommen, um das Konzept der Spezifizität [HT95] und Ramifikation [Thi97] zu integrieren, Qualifikationsprobleme behandeln zu können [Thi96] sowie zur Behandlung von gleichzeitigen Aktionen [BT96] und Nichtdeterminismus [BT97] und der Disjunktion von Fakten [BHS+93b]. Weiterhin existieren Arbeiten zum hierarchischen Planen [EHT96] und zum Partial-Order Planen [SH96].

Da bereits eine Anzahl von Arbeiten zur Darstellung von Aktionen existieren, d.h. die Probleme, die bei Berechnung der Effekte von Aktionen auftauchen, behandeln, und dieses Problem für die Thematik dieser Arbeit nicht relevant ist, wird in dieser Arbeit von der Aktionsdarstellung abstrahiert, so daß die vorhandenen Aktionsdarstellungen integriert werden können. In Beispielen wird die in den vorhergenenden Abschnitten eingefürte Darstellung verwendet.

4. Design von Plansprachen

4.1 Vorhandene Plansprachen

In diesem Kapitel werden Gemeinsamkeiten und Unterschiede verschiedener Ansätze ([LRL+97], [Lev96], [MW87], [SB95], [SB93]) zur Formulierung von rekursiven Plänen und zum Schließen über ihre Resultate betrachtet, und Anforderungen an die in dieser Arbeit konstruierte Plansprache herauskristallisiert.

Neben der Frage, wie die Umwelt und die Veränderungen, die primitive Aktionen (d.h. Aktionen, die in der Modellierung nicht weiter zerlegt werden) in der Umwelt hervorrufen, mit Hilfe der Logik repräsentiert werden, ist ein wichtiges Problem, wie komplexere Pläne (im Folgenden auch Programme genannt) behandelt werden, die aus mehreren Aktionen, Verzweigungen, Schleifen, Prozeduren u.ä. bestehen.

Es lassen sich dabei einige Kernprobleme herausstellen, deren Behandlung in diesem Zusammenhang von Interesse ist.

1. *Bedingte Ausführung von Teilplänen*: Der Agent muß in der Lage sein, zu überprüfen, ob eine Bedingung in der augenblicklichen Situation erfüllt ist, um dementsprechend zu reagieren.

2. *Rekursion und Iteration*: Bei vielen Problemen ist es bei der Formulierung von Plänen hilfreich, wenn der Agent die Lösungen kleinerer Teilaufgaben zusammenfassen und innerhalb des Gesamtplans mehrfach verwenden kann. Wenn ein Agent zum Beispiel eine Anzahl von Blöcken in eine Kiste räumen soll, so ist „Aufnehmen des Blocks; Transportieren des Blocks zur Kiste; Ablegen des Blocks“ eine solche Teilaufgabe, die zu einer Prozedur zusammengefaßt werden könnte. Da sich Iteration durch Rekursion darstellen läßt, ist es ausreichend Rekursion zu betrachten.

3. *Auswahl von Objekten*: In Planungsdomänen, in denen es viele gleichartige Aktionen gibt, kann es sinnvoll sein, Klassen von Aktionen unter einem Aktionsbezeichner zusammenzufassen, und diesen Aktionsbezeichner mit

Parametern zu versehen. Im vorhergehenden Beispiel könnte die Aktion „Aufnehmen eines Blocks“ mit dem Parameter „aufzunehmender Block“ versehen werden. In dieser Arbeit werden die möglichen Werte für die Parameter von Aktionen als **Objekte** bezeichnet.

Wenn nun der Anfangszustand vor der Planung nicht vollständig bekannt ist, bzw. der Plan so allgemein formuliert werden soll, daß er in einer ganzen Klasse von Anfangszuständen zum Ziel führt, dann können die Werte von Parametern u.U. erst während der Planabarbeitung vom Agenten ausgewählt werden.

4. *Nichtdeterminismus*: Davon sind zwei Arten zu unterscheiden:

   nichtdeterministische Effekte von primitiven Aktionen: Die Umwelt eines Agenten stellt sich oft als nichtdeterministisch dar, d.h. die Ausführung einer Aktion kann mehrere mögliche Ergebnisse haben, wie z.B. beim Werfen einer Münze, so daß das Ergebnis nicht a priori, d.h. vor Ausführung der Aktion, bestimmt werden kann.

   Dies kann mehrere Ursachen haben: Einerseits könnte die reale Welt nichtdeterministisch sein[1], andererseits ist es möglich, daß die modellierten Informationen über die Welt nicht für eine Vorhersage ausreichen, da das Modell sonst zu komplex wäre. Diese Form von Nichtdeterminismus ist also vor allem eine Eigenschaft des zugrundeliegenden Aktionsmodells. Für eine ausführlichere Behandlung dieses Themas siehe z.B. [BT97].

   Nichtdeterminismus im Plan: Der Plan kann dem Agenten Spielraum für eigene Entscheidungen lassen. Wenn die Aufgabe z.B. in mehrere voneinander unabhängige Teile zerlegt werden kann, bietet es sich an, es offen zu lassen, welche Teilaufgabe er zuerst erledigt. So ist es bei der oben beschriebenen Aufgabe nicht wichtig, welchen Block der Agent zuerst zur Kiste transportiert. Hierbei handelt es sich um einen sogenannten **don't care Nichtdeterminismus**, d.h. egal welche Entscheidung der Agent trifft, er kommt immer zum Ziel. Im Gegensatz dazu steht der **don't know Nichtdeterminismus**, bei dem zwar a priori bekannt ist, daß mindestens eine der möglichen Entscheidungen zum Ziel führt, aber nicht welche. Der Agent muß also sich entweder auf sein Glück verlassen (falls er getroffene Entscheidungen nicht wieder rückgängig machen kann), oder zusätzliches Wissen in die Planausführung einbringen.

5. *Nachweis von Planeigenschaften*: Ein wichtiges Ziel ist es, deduktiv Eigenschaften des Plans wie **Korrektheit** und **Termination** bestimmen zu

[1] Es wäre auch denkbar, daß ein Agent *prinzipiell* nicht genügend Informationen sammeln kann, um eine korrekte Vorhersage zu treffen, da die Wirkung der Aktion von unzugänglichen Informationen abhängt. Dies ist aber von einem echten Nichtdeterminismus der Welt experimentiell vermutlich nicht unterscheidbar, und daher höchstens ein philosophisches Problem.

können. Dabei sind verschiedene Denkweisen möglich: Für einen **mutigen Agenten** genügt es, wenn er durch Ausführung des Plans *möglicherweise* sein Ziel erreicht. Analog bedeutet für ihn Termination eines Plans, daß die Ausführung des Planes möglicherweise endet, und Korrektheit, daß er dann das Ziel des Plans möglicherweise erfüllt hat.[2] Ein **skeptischer Agent** dagegen möchte, daß die Planausführung unter allen Umständen endet (Termination) und er damit in jedem Fall das Ziel erfüllt (Korrektheit).

6. *Planentwicklung* Das letztendliche Ziel ist es natürlich, korrekte und terminierende Pläne auf deduktivem Wege zu generieren.

In der Literatur wurden dazu verschiedene Ansätze vorgestellt. So kreierten Levesque und andere in [LRL+97] die Sprache GOLOG („alGOl in LOGic“). GOLOG ist als eine logische „high-level Programmiersprache“ gedacht. Ihr Ziel ist es, einen mit Verzweigungen, while-Schleifen und Prozeduren und verschiedenen nichtdeterministischen[3] Konstrukten formulierten Plan mit Hilfe von in den Interpreter integriertem Wissen in eine Sequenz von primitiven Aktionen umzusetzen, die dann z.B. ein Agent ausführen kann. Beispiele für GOLOG-Programme sind:

while $\exists block \;\; ontable(block)$ **do** *remove_a_block* **endWhile**

proc *remove_a_block* $(\pi x)[pickup(x); putaway(x)]$ **endProc**

Die zweite Zeile stellt eine Prozedurdeklaration dar. Sie enthält den nichtdeterministischen Auswahloperator π: $(\pi x)[\delta(x)]$ bedeutet „suche nichtdeterministisch ein Objekt x aus, und führe für dieses x den Plan $\delta(x)$ aus.“ Ein anderes Beispiel für einen nichtdeterministischen Operator ist $|$ in folgendem Programm:

if $(current\_floor \neq n)$ **then** $up(n) \,|\, down(n)$ **endIf**

Diese Zeile ist aus einem Fahrstuhlmodell entnommen, und läßt sich etwa so lesen: „Wenn sich der Fahrstuhl nicht im n-ten Stock befindet, so bewege den Fahrstuhl nach oben oder nach unten in den n-ten Stock.“ Bei beiden Operatoren handelt es sich um einen „don't know“ Nichtdeterminismus des Plans (siehe auf Seite 20): Es ist Aufgabe des Interpreters, herauszufinden, welche der sich bietenden Alternativen angemessen ist.

Eine Grundidee von GOLOG ist es, daß der Programmierer komplexe Aktionsschemata definieren kann, ohne im Detail zu spezifizieren, wie diese Aktionen

[2] Dies wird in Kapitel 6 genauer betrachtet.

[3] Hierbei ist Nichtdeterminismus des Plans gemeint. Aktionen mit nichtdeterministischen Effekten werden in GOLOG nicht berücksichtigt.

auszuführen sind. Der Interpreter (als Theorembeweiser realisiert) ist dann dafür verantwortlich, eine *ausführbare* Sequenz von primitiven Aktionen herauszufinden, die dem Programm entspricht. Im Extremfall kann man es dem Interpreter komplett überlassen, eine Aktionssequenz zu finden, die zum Ziel führt:[4]

$$\textbf{while } \neg \mathit{Goal} \textbf{ do } (\pi a)[a] \textbf{ endWhile}$$

Die Semantik eines Plans δ wird mit Hilfe des Situationskalküls durch eine Art „Makroexpansion" des Ausdrucks $Do(\delta, s, s')$ in eine Formel im Prädikatenkalkül u.U. zweiter Stufe gegeben[5]. Diese drückt aus, daß die Ausführung des Plans δ beginnend in Situation s in Situation s' terminieren kann. Die Ausführung eines Plans kann dabei nichtdeterministisch sein, d.h. es können mehrere Situationen s' existieren, für die $Do(\delta, s, s')$ gültig ist. Gemäß der Grundidee des Situationskalküls (sie Seite 10) beinhaltet dann der Situationsbezeichner $s' = do([a_0, \ldots, a_n], s)$ die Sequenz von primitiven Aktionen $a_0, \ldots, a_n$, die von s zu s' geführt haben. Dies ist dann das Resultat der Interpretation des Plans δ mit Hilfe eines Theorembeweisers.

Die Behandlung von Plänen mit Hilfe dieser Makroexpansion hat den Nachteil, daß man nicht mehr über Pläne quantifizieren kann, und damit keine deduktive Konstruktion eines Planes möglich ist, obwohl man sehr wohl innerhalb des Kalküls auf die Eigenschaften von Plänen schließen kann. In [Lev96] vermeidet Levesque dies, und behandelt Pläne dort als Objekte erster Klasse (Terme), so daß es möglich wird, Pläne deduktiv zu konstruieren, d.h. aus dem Beweis eines Theorems „Es gibt einen Plan, der das Ziel erfüllt" direkt abzuleiten. Allerdings opfert er dabei viel an Expressivität: die betrachtete Plansprache enthält keine Prozeduren mehr, sondern nur noch Iteration und Verzweigungen, und alle Aktionen sind nun parameterlos (entsprechend gibt es kein Pendant zu dem Operator π). Trotzdem muß er Prädikatenlogik zweiter Stufe benutzen, um Termination von Schleifen ausdrücken zu können. In dieser Arbeit benutzt er das Konzept von **sensing actions**: dies sind Aktionen, deren Haupteffekt es ist, das Wissen des Agenten über die Welt verändern. Allerdings wirken diese dort nur lokal als Bedingung für Verzweigungen.

Manna und Waldinger [MW87, MW91] gingen einen Weg ohne die Plansprache so weit einzuschränken. Basierend auf dem Situationskalkül entwickelten Manna und Waldinger eine „Plantheorie". Diese verwendet eine sehr komplexe Gleichungstheorie, um Pläne, die keine Referenzen zu Situationen enthalten dürfen, auf korrekte Weise mit dem im Situationskalkül notwendigen zusätzlichen Situationsargument zu versehen (eine Aufgabe, die in GOLOG das Makro *Do*

[4] Man beachte: dieser Plan ist nicht als direkte Handlungsanweisung zu interpretieren als „tue irgendetwas, bis das Ziel erfüllt ist" (was offensichtlich nicht immer von Erfolg gekrönt ist), sondern als ein Aktionsschema, daß alle Aktionssequenzen zuläßt, bei denen am Ende *Goal* erfüllt ist. Es ist die Aufgabe des Interpreters, eine derartige Sequenz herauszufinden.

[5] Prädikatenlogik zweiter Stufe wird hier notwendig, da die unbegrenzte Iteration die transitive Hülle implizit verwendet, und sich die transitive Hülle nicht in Logik erster Stufe darstellen läßt.

übernimmt), und, im Zusammenhang mit domänenspezifischen Axiomen, die Ausführung von Plänen beschreibt. Der Nachteil dabei ist, daß die verwendete Gleichungstheorie oft unendlich viele Unifikatoren für zwei Terme generiert.

Als deduktiven Mechanismus schlagen Manna und Waldinger sogenannte „Deduktive Tableaus“ vor, die mit verschiedenen zugeschnittenen Resolutionsregeln und Gleichungsunifikation entsprechend der Plantheorie arbeiten. Um nun einen Plan zu generieren, wird das Theorem „für alle Anfangssituationen gibt es eine Endsituation, die die Zielspezifikation erfüllt“ bewiesen. Die gegebenen Beweisregeln garantieren, daß der Beweis konstruktiv in dem Sinne ist, daß als Nebeneffekt des Beweises ein Plan entsteht, der die Anfangssituationen in die Endsituationen überführt.

Ein Beispiel für einen generierten Plan ist im folgenden angegeben. Dieser löst die Aufgabe, in der Blocksworld einen Block frei zu machen, d.h. eventuell auf ihm stehende Blöcke herunterzunehmen. Der entsprechende Plan lautet wie folgt:

$$makeclear(a) \Leftarrow \begin{cases} \text{if } not\ clear(a) \\ \text{then } makeclear(hat(a)); \\ \qquad put(hat(a), table). \end{cases}$$

Dies entspricht der Definition einer Prozedur $makeclear(a)$, welche prüft, ob der Block bereits frei („clear“) ist, und wenn nicht, zunächst den Block $hat(a)$ der auf ihm steht frei macht, und dann diesen herunterräumt. Im Unterschied zu GOLOG findet hier die Selektion von Objekten durch Funktionen (hier: $hat(a)$) statt. Es sind weder nichtdeterministische Aktionen, noch Nichtdeterminismus im Plan möglich.

Um für die Behandlung von Programmtermination bei rekursiven Plänen nicht auf Logik 2. Stufe ausweichen zu müssen, verwenden Manna und Waldinger das Paradigma der expliziten Induktion (siehe dazu [Wal94]). Das heißt, sie nehmen Instanzen des Induktionsaxioms

$$\big(\forall x \ \ (\forall y \ \ y \prec_P x \ \rightarrow \ P(y)) \ \rightarrow \ P(x)\big) \ \rightarrow \ \forall x \ \ P(x)$$

hinzu, wobei $\prec_P$ eine **wohlfundierte Relation** ist, d.h. es gibt keine unendlichen absteigenden Sequenzen $\cdots \prec_P a_3 \prec_P a_2 \prec_P a_1 \prec_P a_0$. In einem konkreten Beweis könnte $P(x)$ zum Beispiel die Termination eines bestimmten Planes in der Situation x sein, und $\prec_P$ eine Relation über Situationen, die z.B. die Anzahl der Blöcke, die auf Block a stehen, vergleicht (d.h. das Freiräumen eines Blockes, auf dem n Blöcke stehen, wird auf das Freiräumen des Blockes über ihm, auf dem $n-1$ Blöcke stehen, zurückgeführt). Entsprechende Relationen werden aus der Domänentheorie bestimmt, und dem Beweiser zur Verfügung gestellt.

Stephan und Biundo [SB93, SB95] spezialisieren sich auf die Verwendung von Modallogiken zur Darstellung von Planungsproblemen, insbesondere der dy-

namischen Logik (DL) [Har79]. Basierend auf zwei Modaloperatoren **add** − $\mathbf{f(d_1, \ldots, d_n)}$ und **delete** − $\mathbf{f(d_1, \ldots, d_n)}$ für jedes Fluent $f(d_1, \ldots, d_n)$, die genau dieses Fluent zum Zustand hinzufügen bzw. aus dem Zustand entfernen, werden primitive Aktionen dargestellt. Sie entsprechen damit kleinen Programmstücken, z.B.:

rec *unstack(x,y).* **if** *on(x,y)* $\wedge$ *clear(x)*
then **add**-*table(x)*;
add-*clear(y)*;
delete-*on(x,y)*
else **abort**
fi.

Dies entspricht der Definition einer Aktion *unstack(x,y)* in der Blocksworld, die einen Block x von einem anderen Block y herunternimmt. Wenn die Vorbedingungen *on(x,y)* $\wedge$ *clear(x)* erfüllt sind, dann ist die Wirkung, daß Block x auf dem Tisch steht, Block y nun frei ist, und Block x nicht mehr auf Block y steht. Aus dieser Darstellung können Rahmenaxiome algorithmisch abgeleitet werden.

Dieses Planungsverfahren wurde mit Hilfe des Karlsruhe Interactive Verifier (KIV) [HRS90, HRS91] realisiert. Dieser sogenannte taktische Theorembeweiser stellt spezielle Beweisstrategien für die dynamische Logik zur Verfügung, mit denen auch während des Beweisprozesses das Programm entstehen kann. In [SB95] wird das Thema der Plangeneration weiterverfolgt, indem eine Methode zur schrittweisen Planverfeinerung beschrieben wird, bei der aus einem vorgegebenen Planschema schrittweise der konkrete Plan erzeugt wird. Die Grundidee dabei ist, daß die Planschemata implizit bereits viele Informationen enthalten, die beim Beweis der Korrektheit des daraus erzeugten verfeinerten Plans benutzt werden können. Damit werden bei der interaktiven Erzeugung der domänenspezifischen Planschemata bereits die schwierigsten deduktiven Probleme gelöst, so daß die spätere auf ein konkretes Planungsproblem angepasste Planverfeinerung wesentlich effizienter ablaufen kann.

Das Problem bei der Verwendung von Modallogiken ist, daß dabei häufig die Logik dem Problem angepaßt wird, anstatt die Modellierung dem Problem anzupassen. (So führen z.B. Biundo und Stephan sowohl in [SB93] als auch in [SB95] neue Varianten von Modallogiken zur Behandlung ihrer Probleme ein.) Dies führt dazu, daß eine Vielzahl von Modallogiken existiert. Daher ist die klassische Logik besser erforscht, und es existieren bessere und effizientere Methoden zur automatischen Deduktion, als für nicht-klassische Logik [Gab94]. In dieser Arbeit wird daher die klassische Logik verwendet.

4.2 Schlußfolgerungen

Da es in dieser Arbeit darum geht, eine Plansprache mit zugehöriger Semantik zu gestalten, die einerseits ausdrucksmächtig genug ist, um bedingte Pläne und Rekursion zu formulieren, aber andererseits eine möglichst effiziente Bestimmung der Eigenschaften eines Planes gestattet, ist die Bestimmung der Parameter dieser Sprache freigestellt. Dieser Arbeit werden folgende Entscheidungen zugrunde gelegt:

1. *Zugrundeliegender Kalkül*: Es besteht die Möglichkeit, wie Manna und Waldinger in der Prädikatenlogik erster Stufe zu bleiben bzw. durch Hinzunahme von Induktionsaxiomen Probleme zu behandeln, die sich sonst nicht innerhalb der Logik erster Stufe beweisen lassen. Da im Fluent-Kalkül das zusätzliche Situationsargument für die Fluenten wegfällt, lassen sich die Probleme, die Manna und Waldinger mit ihrer komplexen Plantheorie lösen müssen, zum großen Teil vermeiden.

2. *Nichtdeterminismus der Aktionen*: Zur Erhöhung der Expressivität erscheint eine Behandlung von Nichtdeterminismus von Aktionen sinnvoll.

3. *Nichtdeterminismus im Plan*: Einer Vorgehensweise analog zu der von GOLOG steht entgegen, daß sich ein nicht so leicht abschätzbarer Planungsaufwand während der Laufzeit ergibt. In dieser Arbeit soll von der Herangehensweise eines skeptischen Agenten ausgegangen werden: Der Plan soll vollständige Informationen über den Weg zum Ziel liefern in dem Sinne, daß *jede* mögliche Art, den Plan auszuführen, zum Ziel führen soll. Dies schließt offenbar Nichtdeterminismus im Plan nicht aus, aber dieser sollte von der „don't care“ Art sein (siehe auf Seite 20).[6]

4. *Auswahl von Objekten*: Um die Komplexität zu begrenzen, wird auf eine Modellierung von Zuweisungen verzichtet. Die modifizierte Übernahme der Idee des π-Operators erscheint jedoch möglich, ohne explizit Substitutionen modellieren zu müssen (siehe Kapitel 5 Seite 34).

5. *Nachweis von Planeigenschaften*: Dies soll in erster Linie aus der Sicht eines skeptischen Agenten geschehen.[7]

6. *Planentwicklung*: Dies ist nicht Thema dieser Arbeit.

[6] An einigen Stellen wird auch auf mutige Agenten eingegangen.
[7] Eine ausführliche Begründung dieser Entscheidung ist in Kapitel 6 auf Seite 39 zu finden.

5. Die Plansprache $\mathcal{P}$

In diesem Kapitel wird eine Sprache zur Formulierung von bedingten und rekursiven Plänen definiert, und eine Semantik in Form eines Abarbeitungsmodells angegeben.

5.1 Ein vereinfachtes Weltmodell

In dieser der Arbeit wird von einem Weltmodell ausgegangen, daß in diesem Abschnitt beschrieben wird.

Die modellierte Welt bestehe aus einem Agenten und der Umwelt dieses Agenten. Die Umwelt sei insofern statisch, als daß sie sich nur durch Aktionen des Agenten verändern kann.

Es sollen also über das System „Umwelt + Agent“ Aussagen getroffen werden. Der Zustand dieses Gesamtsystems, im folgenden als **Welt** bezeichnet, ist durch zwei Komponenten charakterisiert: durch den Umweltzustand und durch das Vorhaben des Agenten.

Abbildung 5.1: Agent und Umwelt

Die erste Komponente ist der **Umweltzustand** („State“) s aus der **Zustandsmenge** $\mathcal{S}$. Eine Veränderung geschieht dadurch, daß der Agent eine **primitive Aktion** a aus einer Aktionsmenge $\mathcal{N}_{\mathcal{A}}(\mathcal{O})$ [1] ausführt. Dadurch geht die Umwelt in einen neuen Zustand s' über, der von s und a abhängt. Dies kann durch durch eine **Zustandsübergangsrelation**[2] $\mathcal{C}_p \subseteq \mathcal{S} \times \mathcal{N}_{\mathcal{A}}(\mathcal{O}) \times \mathcal{S}$ mit $(s, a, s') \in \mathcal{C}_p$ charakterisiert werden. Dabei handelt es sich nicht notwendigerweise um eine Funktion, da es möglich ist, daß es für ein Paar (s, a) mehrere

[1] Zur Wahl dieser Bezeichnungsweise siehe Definition 5.1 auf Seite 29.

[2] Da für den Fluent-Kalkül bereits eine Vielzahl von Arbeiten zur Aktionsdarstellung existieren, soll hier zunächst völlig von der konkreten Art von Aktionen abstrahiert werden.

Nachfolgezustände gibt, wenn die Aktion nichtdeterministische Effekte hat, bzw. unter Umständen überhaupt kein Zustand s' existiert, wenn die Aktion a im Zustand s nicht ausführbar ist (**Disqualifikation**).

Die zweite Komponente des Weltzustands ist der Zustand des Agenten. Für einen planausführenden Agenten handelt es sich einfach um sein Vorhaben, d.h. den noch nicht ausgeführten Teil des Plans.[3] Hierbei ist zwischen einem **Plan** und einem **Vorhaben** zu unterscheiden: Bei einem Plan handelt es sich um eine in einer geeigneten Sprache formulierte Handlungsanweisung, die der Agent Schritt für Schritt ausführt. Bei einem Vorhaben handelt es sich um den noch nicht ausgeführten Teil dieses Plans, es ist also im Kontext des Planes zu sehen. Der Plan eines Agenten umfaßt also das ursprüngliche Vorhaben des Agenten (in Abschnitt 5.4 kommen noch Prozedurdefinitionen zum Plan hinzu, die Bestandteil des Planes, aber nicht des Vorhabens in diesem Sinne sind.)

Die Vorhaben seien dabei in einer **Plansprache** $\mathfrak{P}$ formuliert. Aus diesem ermittelt der Agent die nächste auszuführende Aktion. Dadurch verändert sich aber auch das Vorhaben, da die soeben ausgeführte Aktion a nicht mehr zu seinem Vorhaben zählt, und nun vergessen werden kann.

Bis jetzt handelt es sich allerdings um einen blinden Agenten, der nicht die Wirkung seiner Aktionen beobachtet und nur stur seine vorgegebene Folge von Aktionen ausführen kann. Es soll aber noch die Möglichkeit modelliert werden, daß der Agent Beobachtungen anstellt. Er kann dabei Bedingungen c („condition“) aus einer **Bedingungsmenge** $\mathcal{N}_C(\mathcal{O})$ überprüfen und seine Vorhaben entsprechend dem Resultat umstrukturieren. Dies wird modelliert mit Hilfe einer **Wahrheitsrelation** $\mathcal{H}$ (von „Holds“). Diese ist die Menge aller Paare (c, s) aus einer Bedingung c und einem Umweltzustand s, so daß c in s erfüllt ist.[4] Die Prüfung von c ist also genau dann erfolgreich, wenn $(c, s) \in \mathcal{H}$. Während dieser Beobachtung führt er keine eigentliche Aktion aus, so daß die Umwelt ihren Zustand zunächst beibehält.

Das Verhalten des Agenten wird dabei in dieser Arbeit als vorhersagbar angenommen, d.h. nur von Umweltzustand s und seinem Vorhaben p bestimmt, wenn auch nicht notwendigerweise deterministisch.

Das Gesamtsystem aus Umwelt und Agent wird also durch ein Paar $(s, p) \in \mathcal{S} \times \mathfrak{P}$ charakterisiert. In jedem Schritt verändert sich dies zu einem neuen **Weltzustand** (s', p'). Dies wird durch eine **Weltrelation** $\mathcal{C}_\mathsf{W} \subseteq (\mathcal{S} \times \mathfrak{P}) \times (\mathcal{S} \times \mathfrak{P})$ dargestellt. Im folgenden soll $(s, p) \rightsquigarrow (s', p')$ ausdrücken, daß $\big((s, p), (s', p')\big) \in$

[3] Die Position des Agenten, seine vorhandenen Hilfsmittel etc. wird in diesem Zusammenhang zum Umweltzustand gerechnet. Das Problem wird hierbei aus der Sichtweise der Steuerung des Agenten betrachtet.

[4] Dieser Teil des Modells ähnelt stark einer Kripkestruktur - die Weltrelation entspräche der Erreichbarkeitsrelation, und aus $\mathcal{H}$ ließe sich für jeden Weltzustand eine Interpretation auf der Bedingungsmenge konstruieren. Im Gegensatz zu einer Modallogik ist hier aber das Ziel nicht die Ableitung von Aussagen über die Bedingungen, sondern über die Weltrelation.

$\mathcal{C}_{\mathsf{W}}$, d.h. wenn die Umwelt den Zustand s hat, und der Agent den Vorhaben p ausführen will, dann *kann* die Welt in einem Arbeitsschritt des Agenten in den Weltzustand (s', p') übergehen. Die Weltrelation faßt die Interpretation des Planes durch den Agenten und die Wirkung der Aktionen auf die Umwelt zusammen.

Da sowohl die Umwelt, als auch der Agent sich nichtdeterministisch verhalten können, ergibt sich i.A. nicht nur eine mögliche Kette (**Berechnungsfolge**) $(s_0, p_0) \rightsquigarrow (s_1, p_1) \rightsquigarrow \cdots \rightsquigarrow (s_k, p_k)$ von Weltzuständen, sondern ein nichtlinieararer Graph von möglichen Entwicklungen. Eine Berechnungsfolge kann erfolgreich enden, indem der Agent als Vorhaben das leere Vorhaben ε als Symbol für „Aufgabe erfüllt“ erreicht, sie kann aber auch endlos sein, bzw. in einen Zyklus laufen, oder in einer Sackgasse enden, wenn die nächste Aktion nicht ausführbar ist. Aber dies ist Thema des nächsten Kapitels.

Der Inhalt dieses Kapitels besteht nun darin, eine geeignete Plansprache und eine zugehörige Weltrelation $\mathcal{C}_{\mathsf{W}}$ in Abhängigkeit von $\mathcal{C}_p$ und $\mathcal{H}$ zu formulieren, so daß der Agent eine möglichst große Klasse von Aufgaben lösen kann, die aber einfach genug ist, um effizient darüber schlußfolgern zu können. Doch zunächst sollen die Anforderungen an die Relationen $\mathcal{C}_p$ und $\mathcal{H}$ konkretisiert werden.

5.2 Planungsdomänen

In vielen Umgebungen ist es erforderlich, Aktionen zu parametrisieren. So ist es z.B in der Blocksworld kaum sinnvoll, für jede move-Operation und für alle Kombinationen von Blöcken A und B einen neuen Aktionsbezeichner $move\_A\_B$ einzuführen, sondern man würde angemessenerweise Terme wie $move(A, B)$ verwenden. Dadurch ist es in Plänen möglich, einzelne Parameter der Aktionen durch Planvariablen (siehe nächster Abschnitt) zu ersetzen, die erst während der Planausführung durch konkrete Objekte aus einer **Objektmenge** $\mathcal{O}$ ersetzt werden.

Gleiches gilt für die Bedingungen. Diese seien durch Terme $c_i(o_1, \ldots, o_n)$ dargestellt, wobei c_i ein Bedingungsbezeichner ist, und $o_1, \ldots, o_n$ Objekte. Damit ist $\big(c_i(o_1, \ldots, o_n), s\big) \in \mathcal{H}$ zu interpretieren als „Im Zustand s ist die Bedingung c_i für die Objekte $o_1, \ldots, o_n$ erfüllt“.

Die *Umwelt* eines Agenten, d.h. die Planungsdomäne, wird nun formal wie folgt beschrieben:

Definition 5.1.
Eine ***Planungsdomäne*** D *ist ein 7-Tupel* $(\mathcal{O}, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{C}}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$ *von Mengen. Dabei ist*

$\mathcal{O}$ *die Menge von Objekten,*

$\mathcal{N}_\mathcal{A}$ *eine endliche Menge („Names of Actions“) von Paaren* (a, n) *von Aktionsbezeichnern* a *mit Stelligkeit* $n \in \mathbb{N}$

$\mathcal{N}_\mathcal{C}$ *eine endliche Menge („Names of Conditions“) von Paaren* (b, n) *von Bedingungsbezeichnern* b *mit ihrer Stelligkeit* $n \in \mathbb{N}$ *,*

$\mathcal{S}$ *die Menge von Umweltzuständen („States“),*

$\mathcal{C}_p$ *die Zustandsübergangsrelation („*Causes*“ für* primitive *Aktionen), für die gilt:*

$$\mathcal{C}_p \subseteq \mathcal{S} \times \mathcal{N}_\mathcal{A}(\mathcal{O}) \times \mathcal{S}$$

$\mathcal{H}$ *die Wahrheitsrelation („Holds“), die*

$$\mathcal{H} \subseteq \mathcal{N}_\mathcal{C}(\mathcal{O}) \times \mathcal{S}$$

erfüllt.

Primitive Aktionen und Bedingungen sind nun die entsprechend ihrer Stelligkeit mit Parametern aus $\mathcal{O}$ *versehenen Aktions- und Bedingungsbezeichner.*[5] $\mathcal{N}_\mathcal{A}(\mathcal{O})$ *und* $\mathcal{N}_\mathcal{C}(\mathcal{O})$ *repräsentieren nun die Mengen aller primitiven Aktionen und Bedingungen:*

$$\mathcal{N}_\mathcal{A}(\mathcal{O}) = \{a(o_1, o_2, \ldots, o_n) \mid (a, n) \in \mathcal{N}_\mathcal{A},\ o_1, o_2, \ldots, o_n \in \mathcal{O}\}$$
$$\mathcal{N}_\mathcal{C}(\mathcal{O}) = \{c(o_1, o_2, \ldots, o_n) \mid (c, n) \in \mathcal{N}_\mathcal{C},\ o_1, o_2, \ldots, o_n \in \mathcal{O}\}$$

Für einfache Planungsdomänen kann auf die Parametrisierung verzichtet werden, so daß dann $\mathcal{O} = \emptyset$ gilt und nur noch 0-stellige Aktions- und Bedingungsbezeichner zugelassen sind.

Beispiel 5.2 (Blocksworld). *Eine Möglichkeit, eine Version der „Blocksworld“ darzustellen, wäre z.B. folgende Domäne* D_B *:*

$\mathcal{O}$**:** *Die (endliche) Menge der Blöcke* $\mathcal{O} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \ldots\}$ *.*

$\mathcal{S}$**:** *In der Blocksworld sind 3 Fluenten definiert:*

$ot(\mathrm{x})$**:** *Block* x *steht auf dem Tisch.*

$on(x, y)$**:** *Block* x *steht auf Block* y *.*

$cl(\mathrm{x})$**:** *Es steht kein Block auf Block* x *.*

[5] Eigentlich handelt es sich hier um Aktionsnamen und Bedingungsnamen. Im Folgenden wird allerdings der Kürze halber von Aktionen und Bedingungen gesprochen.

Ein Zustand kann z.B. durch Menge dieser Fluenten dargestellt werden, die in diesem Zustand wahr sind, wobei x und y durch Werte aus $\mathcal{O}$ ersetzt sind. So wird der nebenstehende Zustand durch $\{cl(A), on(A,B), ot(B)\}$ *beschrieben.*

$\mathcal{S}$ *ist also die Menge derartiger Zustandsbeschreibungen, denen sich ein konkreter Zustand zuordnen läßt. Dazu müssen die Beschreibungen einige Bedingungen (sog. Domain-Constraints) erfüllen, z.B. kann ein Block nicht auf zwei anderen stehen, ein Block kann nur frei sein, wenn kein Block auf ihm steht, ein Block kann nicht auf sich selbst stehen, usw.*

$$\mathcal{S} = \Big\{\{cl(A), on(A,B), ot(B)\}, \{cl(A), cl(B), ot(A), ot(B)\}, \dots\Big\}$$

$\mathcal{N}_\mathcal{A}$**:** *Es gibt zwei Aktionen:* $\mathrm{move(x,y)}$ *stellt den Block x auf Block y, und* $\mathrm{totabl(x)}$ *stellt Block x auf den Tisch.* move *ist also zweistellig, und totabl einstellig, d.h.* $\mathcal{N}_\mathcal{A} = \{(\mathrm{move}, 2), (totabl, 3)\}$ *. Damit ist*

$$\mathcal{N}_\mathcal{A}(\mathcal{O}) = \big\{\mathrm{move}(A,A), \mathrm{move}(A,B), \dots, \mathrm{move}(B,A), \dots, totabl(A), totabl(B), \dots\big\}.$$

Offenbar sind hier auch sinnlose Aktionen wie $\mathrm{move}(A,A)$ *dabei. Diese können aber durch eine entsprechende Modellierung von* $\mathcal{C}_p$ *ausgeschlossen werden (d.h. es gibt keinen Zustand s', so daß* $(s, \mathrm{move}(A,A), s') \in \mathcal{C}_p$ *).*

$\mathcal{C}_p$**:** *Die Bedeutung der Aktionen in* $\mathcal{N}_\mathcal{A}(\mathcal{O})$ *wird durch die Relation* $\mathcal{C}_p$ *modelliert.* $\mathcal{C}_p$ *ist eine Menge von Tripeln (Anfangszustand, Aktion, Endzustand). Zum Beispiel führt die Aktion* $totabl(A)$ *aus dem oben abgebildeten Zustand in den Zustand, in dem beide Blöcke auf dem Tisch stehen. Dies wird durch das Tripel*

$$\big(\{cl(A), on(A,B), ot(B)\}, totabl(A), \{cl(A), cl(B), ot(A), ot(B)\}\big)$$

ausgedrückt. $\mathcal{C}_p$ *ist nun die Menge derartiger Tripel, die den Aktionsausführungen in der modellierten Welt entsprechen:*

$$\mathcal{C}_p = \Big\{ \big(\{cl(A), on(A,B), ot(B)\}, totabl(A), \{cl(A), cl(B), ot(A), ot(B)\}\big), \big(\{cl(A), cl(B), ot(A), ot(B)\}, \mathrm{move}(B,A), \{cl(B), on(B,A), ot(A)\}\big), \dots \Big\}$$

Da sich die Blocksworld deterministisch verhält, kann man $\mathcal{C}_p$ *auch als eine partielle Funktion, die* $\mathcal{S} \times \mathcal{N}_\mathcal{A}(\mathcal{O})$ *auf* $\mathcal{S}$ *abbildet. Die Funktion ist partiell, da nicht alle Aktionen ständig ausführbar sind. Im allgemeinen ist* $\mathcal{C}_p$ *allerdings keine Funktion.*

$\mathcal{N}_\mathcal{C}$: *In der Blocksworld sind folgende Bedingungen naheliegend:*

$ot(\mathrm{x})$: *Block x steht auf dem Tisch.*

$on(\mathrm{x},y)$: *Block x steht auf Block y.*

$cl(\mathrm{x})$: *Es steht kein Block auf Block x.*

Es sind aber auch kompliziertere Bedingungen denkbar, wie z.B:

$\mathrm{over}(\mathrm{x},y)$: *Block x befindet sich oberhalb von Block y in einem Turm.*

oder zusammengesetzte Bedingungen, wie

$ttop(\mathrm{x},y)$: *Block x befindet sich an der Spitze des Turmes über Block y, d.h. $\mathrm{over}(\mathrm{x},y)$ und $cl(\mathrm{x})$ sind erfüllt.*

$\mathcal{H}$: *Während $\mathcal{N}_\mathcal{C}$ nur eine Menge von Bedingungsbezeichnern einführt, wird die Semantik dieser Bedingungen durch $\mathcal{H}$ festgelegt, indem $\mathcal{H}$ für jedem Zustand alle in diesem Zustand wahren Bedingungen zuordnet. So gelten im Zustand $s = \{cl(A), on(A,B), ot(B)\}$ z.B. die Bedingungen $cl(A), on(A,B), ot(B), \mathrm{over}(A,B)$ und $ttop(A,B)$, d.h. $\mathcal{H}$ enthält die Paare $(s, cl(A)), (s, on(A,B)), \ldots, (s, ttop(A,B))$.*

$$\mathcal{H} = \Big\{ \big(\{cl(A), on(A,B), ot(B)\}, cl(A)\big), \big(\{cl(A), on(A,B), ot(B)\}, on(A,B)\big), \ldots, \big(\{cl(A), on(A,B), on(B,C), ot(C)\}, \mathrm{over}(A,C)\big), \big(\{cl(A), on(A,B), on(B,C), ot(C)\}, ttop(A,C)\big), \ldots \Big\}$$

5.3 Sprachkonstrukte

Im diesem Abschnitt werden die benötigten Sprachkonstrukte und ihre Interpretation durch den Agenten erläutert, und daraus die entsprechende Weltrelation konstruiert. Eine formale Darstellung dieser ist im nächsten Abschnitt auf Seite 36 zu finden.

Ein Vorhaben p sei eine endliche Folge von Aktionen, die nacheinander mit dem rechtsassoziativen Operator ; getrennt aufgeschrieben werden. Die Aktionen können entweder primitive Aktionen sein, d.h. Aktionen aus der Aktionenmenge $\mathcal{N}_\mathcal{A}(\mathcal{O})$ der zugrundeliegenden Planungsdomäne $\mathsf{D} = (\mathcal{O}, \mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{C}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$, oder zusammengesetzte Aktionen, d.h. bedingte Aktionen oder Prozeduren. Abgeschlossen wird diese Folge immer durch das leere Vorhaben ε. Der nächste

Schritt bei der Planabarbeitung wird immer von der ersten Aktion bestimmt, es sei denn es handelt sich beim Vorhaben bereits um ε, d.h. der Agent ist fertig. Im folgenden werden die möglichen Typen von Aktionen beschrieben, und die Effekte, die diese hervorrufen. Zusammengefaßt wird dies dann in Definition 5.12 auf Seite 37.

5.3.1 Primitive Aktionen

Sei die Welt (das System aus Umwelt und Agent) im Zustand (s,p), und $p = a;p'$ (d.h. das gegenwärtige Vorhaben p besteht aus einer Aktion a und dem Rest p'), wobei $a \in \mathcal{N}_{\mathcal{A}}(\mathcal{O})$ eine primitive Aktion ist. Der Agent führt also die primitive Aktion a aus, wodurch die Umwelt in einen der Zustände s', für die $(s,a,s') \in \mathcal{C}_p$ ist, übergeht. Da die Aktion a ausgeführt wurde, ist das neue Vorhaben des Agenten nun p'. Auf die Weltrelation bezogen bedeutet dies, daß der Weltzustand $(s,\ a;p')$ in den Weltzustand (s',p') übergehen kann. Anders ausgedrückt: $\big((s,\ a;p),(s',p)\big) \in \mathcal{C}_{\mathsf{W}}$, oder

$$(s,\ a;p) \rightsquigarrow (s',p) \quad \text{wenn} \quad (s,a,s') \in \mathcal{C}_p \tag{5.3}$$

(Die Variablen sind dabei natürlich allquantifiziert.)

Die erste Aktion des Vorhabens wird also ausgeführt und „vergessen“.

Beispiel 5.4 (Blocksworld, Fortsetzung). [6]

Der Umweltzustand sei nun $\{cl(A), on(A,B), on(B,C), ot(C)\}$ *(dies entspricht der nebenstehenden Abbildung), und das Vorhaben* $totabl(A);\mathrm{if}(ot(B),\varepsilon,dtower(B));\varepsilon$. *Die erste auszuführende Aktion ist also* $totabl(A)$. *In der Blocksworld-domäne gilt nun*

$$\big(\{cl(A), on(A,B), on(B,C), ot(C)\},\ totabl(A),\ \{cl(A), cl(B), on(B,C), ot(A), ot(C)\}\big) \in \mathcal{C}_p$$

Damit ist der Neue Weltzustand gemäß (5.3)*:*

$$(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\},\ \mathrm{if}(ot(B),\varepsilon,dtower(B);\varepsilon);\varepsilon),$$

[6] Bei diesem Beispiel und dem folgenden Beispiel 5.7 handelt es sich hier um einen Weltzustand während der Abarbeitung des Plans $dtower(A)$ vom abgebildeten Umweltzustand aus. *dtower* ist eine Prozedur, die im Beispiel 5.9 erklärt wird.

d.h.

$$\big(\{cl(A), on(A,B), on(B,C), ot(C)\}, totabl(A); \text{if}(ot(B), \varepsilon, dtower(B); \varepsilon); \varepsilon\big) \rightsquigarrow \\ \big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\},\ \text{if}(ot(B), \varepsilon, dtower(B)); \varepsilon\big)$$

5.3.2 Bedingte Teilpläne

Bedingte Teilpläne werden ein Konstrukt mit „if-then-else“ Semantik dargestellt: $\text{if}(c, p_1, p_2)$ als erste Aktion eines Vorhabens $p = (\text{if}(c, p_1, p_2); p')$ bewirkt die Ausführung von p_1 oder alternativ von p_2 in Abhängigkeit davon, ob die Bedingung c im gegenwärtigen Zustand s erfüllt ist ($(c, s) \in \mathcal{H}$). Dies geschieht, indem das neue Vorhaben aus p_1 und p' bzw. p_2 und p' mit Hilfe der Funktion append zusammengesetzt wird. (Eine genaue Definition von append erfolgt in Gleichung 5.14 auf Seite 38).

$$(s, (\text{if}(c, p_1, p_2); p)) \rightsquigarrow (s, \text{append}(p_1, p)) \quad \text{wenn} \quad (c, s) \in \mathcal{H} \tag{5.5}$$
$$(s, (\text{if}(c, p_1, p_2); p)) \rightsquigarrow (s, \text{append}(p_2, p)) \quad \text{sonst} \tag{5.6}$$

Offenbar verändert sich nur das Vorhaben, aber nicht der Umweltzustand.

Beispiel 5.7 (Blocksworld, Fortsetzung).

Der Weltzustand sei nun

$$\big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\}, \\ \text{if}(ot(B), \varepsilon, dtower(B); \varepsilon); \varepsilon\big)$$

Als erste Aktion ist $\text{if}(ot(B), \varepsilon, dtower(B); \varepsilon)$ *auszuführen. In der Blocksworld ist die Bedingung* $ot(B)$ *in dem gegebenen Zustand nicht erfüllt, d.h. es gilt*

$$\big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\}, ot(B)\big) \notin \mathcal{H}$$

Damit ergibt sich mit $app(dtower(B), \varepsilon) = dtower(B); \varepsilon)$ *nach* (5.6)*:*

$$\big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\},\ \text{if}(ot(B), \varepsilon, dtower(B); \varepsilon); \varepsilon\big) \\ \rightsquigarrow \\ \big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\},\ dtower(B); \varepsilon\big)$$

5.3.3 Prozeduren und nichtdeterministische Auswahl

Zu jedem Plan gehört eine Menge $\mathcal{P}$ von Prozedurdefinitionen. Eine Prozedurdefinition ist ein Tripel (h, c, b) aus einem Prozedurkopf h, einer Auswahlbe-

dingung c, und einem Prozedurkörper b. Als Abkürzung für $(h, c, b) \in \mathcal{P}$ wird im Folgenden auch $h \underset{c}{\Longleftarrow} b$ geschrieben.

Die in der Prozedurdefinition vorhandene Auswahlbedingung verknüpft die in Kap. 4 motivierte nichtdeterministische Auswahl mit der Prozedurausführung. Wenn im Prozedurkopf weniger Variablen als im Prozedurkörper vorhanden sind, so werden diese durch die Auswahlbedingung durch Objekte belegt. Es geschieht also eine Anpassung des Vorhabens an die konkrete Situation.

Im Einzelnen geschieht die Interpretation eines Prozeduraufrufs wie folgt: Wenn die erste Aktion eines Vorhabens einem Prozedurkopf entspricht, so wird vom Agenten eine entsprechende Grundinstanz der Prozedurdefinition gebildet. Das heißt er findet eine Substitution σ, die alle Variablen in Prozedurkopf, -körper und Auswahlbedingung durch Objekte ersetzt. Wenn nun die Auswahlbedingung im augenblicklichen Umweltzustand erfüllt ist, so kann er die so erzeugte Instanz des Prozedurkörpers zur Weiterarbeit verwenden.

$$(s, (h\sigma; p)) \rightsquigarrow (s, \text{append}(b\sigma; p))$$
$$\text{für} \quad (h, c, b) \in \mathcal{P} \ \wedge \ h\sigma \in \mathcal{N}_{\mathcal{P}}(\mathcal{O}) \ \wedge \ c\sigma \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}) \ \wedge \ b\sigma \in \mathfrak{P} \ \wedge \ (c\sigma, s) \in \mathcal{H} \quad (5.8)$$

$\mathfrak{P}$ ist dabei die Menge aller Vorhaben. Diese sind grundinstantiiert, d.h. enthalten keine Variablen. Damit garantieren die Bedingungen $h\sigma \in \mathcal{N}_{\mathcal{P}}(\mathcal{O}) \ \wedge \ c\sigma \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}) \ \wedge \ b\sigma \in \mathfrak{P}$, daß $h\sigma, c\sigma$ und $b\sigma$ grundinstantiiert sind, d.h. das neue Vorhaben, $\text{append}(b\sigma; p)$, enthält ebenfalls keine Variablen.

Beispiel 5.9 (Blocksworld, Fortsetzung). *Eine Prozedur zum Abbau eines Turmes in der Blocksworld ist:*

$$dtower(x) \underset{on(x,y)}{\Longleftarrow} totabl(x); if(ot(y), \varepsilon, dtower(y); \varepsilon); \varepsilon$$

Wenn nun der Turm ABC (Umweltzustand $\{cl(A), on(A,B), on(B,C), ot(C)\}$ *) abgebaut werden soll, so geht man von einem Plan* $dtower(A); \varepsilon$ *aus. Die erste Aktion dieses Vorhabens ist* $dtower(A)$*. Um die Prozedurdefinition anwenden zu können, muß also eine passende Substitution der Variablen* x *und* y *gefunden werden.* x *kann nur durch* A *ersetzt werden, damit der Prozedurkopf* $dtower(A)$ *entspricht. Damit ist die Bedingung* $on(A, y)$ *zu erfüllen. Da Block A auf Block B steht, wird diese durch die Substitution* $\{x/a, y/b\}$ *erfüllt:*

$$\big(\{cl(A), on(A,B), on(B,C), ot(C)\},\ on(x,y)\{x/a, y/b\}\big) \in \mathcal{H}$$

Der Prozedurkörper ist nach Anwendung der Substitution $\{x/A\}\{y/b\}$ *zu* $totabl(A); if(ot(B), \varepsilon, dtower(B); \varepsilon); \varepsilon$ *grundinstantiiert. Nach* (5.8) *ergibt sich*

also ein Zustandsübergang

$$\big(\{cl(A), on(A,B), on(B,C), ot(C)\},\ dtower(A); \varepsilon\big) \rightsquigarrow \\ \big(\{cl(A), on(A,B), on(B,C), ot(C)\}, totabl(A); \mathrm{if}(ot(B), \varepsilon, dtower(B); \varepsilon); \varepsilon\big)$$

Analog ist im Endzustand von Beispiel 5.7 die erste auszuführende Aktion ein Prozeduraufruf:

$$\big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\},\ dtower(B); \varepsilon\big)$$

Da Block B auf Block C steht, ist die Bedingung $on(B,C)$ *in der gegebenen Situation erfüllt, d.h. die aus* (5.8) *ergibt sich mit der Substitution* $\sigma = \{x/B, y/C\}$

$$\big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\},\ dtower(B); \varepsilon\big) \rightsquigarrow \\ \big(\{cl(A), cl(B), on(B,C), ot(A), ot(C)\}, \\ totabl(B); if(ot(C), \varepsilon, dtower(C); \varepsilon); \varepsilon\big)$$

Die Auswahl kann sich dabei sowohl nichtdeterministisch verhalten, als auch deterministisch, wie im obigen Beispiel, wenn die Bedingung prinzipiell nur durch ein Objekt erfüllt werden kann (da in der Blocksworld nicht zwei Blöcke gleichzeitig auf einem Block stehen können.)

5.4 Die Plansprache

Die im vorigen Abschnitt eingeführten Sprachkonstrukte lassen sich wie folgt in zusammenfassen:

Definition 5.10. *Die* ***Menge aller Vorhaben*** $\mathfrak{P}$ *für eine Domäne* $\mathsf{D} = (\mathcal{O}, \mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{C}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$ *, eine Variablenmenge* $\mathcal{V}$ *und eine endliche Menge* $\mathcal{N}_\mathcal{P}(\mathcal{O})$ *von Paaren* (h,n) *von Prozedurnamen* h *und ihrer Stelligkeit* $n \in \mathbb{N}$ *ist definiert wie folgt:*

1. $\varepsilon \in \mathfrak{P}$
2. $\forall p \in \mathfrak{P} \quad \forall a \in \mathcal{N}_\mathcal{A}(\mathcal{O}) \quad (a; p) \in \mathfrak{P}$
3. $\forall p, p_1, p_2 \in \mathfrak{P} \quad \forall c \in \mathcal{N}_\mathcal{C}(\mathcal{O}) \quad \forall p_1, p_2 \in \mathfrak{P} \quad (\mathrm{if}(c, p_1, p_2); p) \in \mathfrak{P}$
4. $\forall h \in \mathcal{N}_\mathcal{P}(\mathcal{O}) \quad \forall p \in \mathfrak{P} \quad (h; p) \in \mathfrak{P}$
5. $\mathfrak{P}$ *ist die kleinste Menge, die diese Bedingungen erfüllt.*

Die Menge aller zulässigen Prozedurkörper $\mathfrak{P}^{(\mathcal{V})}$ *ist definiert wie folgt:*

1. $\varepsilon \in \mathfrak{P}^{(\mathcal{V})}$
2. $\forall p \in \mathfrak{P}^{(\mathcal{V})} \;\; \forall a \in \mathcal{N}_{\mathcal{A}}(\mathcal{O} \cup \mathcal{V}) \;\; (a;p) \in \mathfrak{P}^{(\mathcal{V})}$
3. $\forall p, p_1, p_2 \in \mathfrak{P}^{(\mathcal{V})} \;\; \forall c \in \mathcal{N}_{\mathcal{C}}(\mathcal{O} \cup \mathcal{V}) \;\; \forall p_1, p_2 \in \mathfrak{P}^{(\mathcal{V})} \;\; (\mathrm{if}(c, p_1, p_2); p) \in \mathfrak{P}^{(\mathcal{V})}$
4. $\forall h \in \mathcal{N}_{\mathcal{P}}(\mathcal{O} \cup \mathcal{V}) \;\; \forall p \in \mathfrak{P}^{(\mathcal{V})} \;\; (h;p) \in \mathfrak{P}^{(\mathcal{V})}$
5. $\mathfrak{P}^{(\mathcal{V})}$ *ist die kleinste Menge, die diese Bedingungen erfüllt.*

Die Schreibweisen $\mathcal{N}_{\mathcal{P}}(\mathcal{O})$, $\mathcal{N}_{\mathcal{C}}(\mathcal{O})$, $\mathcal{N}_{\mathcal{C}}(\mathcal{O} \cup \mathcal{V})$ etc. sind dabei wie in Definition 5.1 auf Seite 29 zu lesen.

Die Definition einer Welt faßt Domäne und Prozeduren zusammen, so daß dann auf dieser Welt die Weltrelation definiert werden kann.

Definition 5.11. *Eine* ***Welt*** W *ist ein Tupel* $(\mathsf{D}, \mathcal{V}, \mathcal{N}_{\mathcal{P}}, \mathcal{P})$, *wobei* D *eine Domäne* $(\mathcal{O}, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{C}}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$ *ist,* $\mathcal{V}$ *eine Menge von Variablen,* $\mathcal{N}_{\mathcal{P}}$ *eine endliche Menge von Prozedurbezeichnern und* $\mathcal{P}$ *eine endliche Menge von Prozeduren mit*

$$\mathcal{P} \subseteq \mathcal{N}_{\mathcal{P}}(\mathcal{V}) \times \mathcal{N}_{\mathcal{C}}(\mathcal{O} \cup \mathcal{V}) \times \mathfrak{P}^{(\mathcal{V})}$$
$$\mathcal{N}_{\mathcal{P}} \cap \mathcal{N}_{\mathcal{A}} = \emptyset$$

Daraus ergibt sich nun die Weltrelation:

Definition 5.12. *Für eine Domäne* $\mathsf{D} = (\mathcal{O}, \mathcal{N}_{\mathcal{A}}, \mathcal{N}_{\mathcal{C}}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$ *ist die* ***Weltrelation*** $\mathcal{C}_{\mathsf{W}} \subseteq (\mathcal{S} \times \mathfrak{P}) \times (\mathcal{S} \times \mathfrak{P})$ *die kleinste Relation, die folgenden Bedingungen erfüllt:*

$\forall s, s' \in \mathcal{S} \;\; \forall a \in \mathcal{N}_{\mathcal{A}}(\mathcal{O}) \;\; \forall c \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}) \;\; \forall p, p_a, p_b \in \mathfrak{P} \;\; \forall \sigma \quad :$

$$(s, (a;p)) \rightsquigarrow (s', p) \quad \textit{wenn} \quad (s, a, s') \in \mathcal{C}_p \tag{5.13a}$$
$$(s, (\mathrm{if}(c, p_a, p_b); p)) \rightsquigarrow (s, \mathrm{append}(p_a, p)) \quad \textit{wenn} \quad (c, s) \in \mathcal{H} \tag{5.13b}$$
$$(s, (\mathrm{if}(c, p_a, p_b); p)) \rightsquigarrow (s, \mathrm{append}(p_b, p)) \quad \textit{wenn} \quad (c, s) \notin \mathcal{H} \tag{5.13c}$$
$$(s, (h\sigma; p)) \rightsquigarrow (s, \mathrm{append}(b\sigma; p)) \quad \textit{wenn} \quad (h, c, b) \in \mathcal{P} \;\wedge\; h\sigma \in \mathcal{N}_{\mathcal{P}}(\mathcal{O}) \;\wedge\; c\sigma \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}) \;\wedge\; b\sigma \in \mathfrak{P} \;\wedge\; (c\sigma, s) \in \mathcal{H} \tag{5.13d}$$

Dabei ist $(s, p) \rightsquigarrow (s', p')$ *eine Abkürzung für* $(s, p, s', p') \in \mathcal{C}_{\mathsf{W}}$.

Hierbei ist append eine Funktion, die die Teilpläne aneinander anhängt. Sie ist definiert wie folgt:

$$\begin{aligned} &\text{append} \ : \ (\mathfrak{P} \times \mathfrak{P}) \rightarrow \mathfrak{P} \ : \\ &(p_1, p_2) \mapsto \begin{cases} p_2 & \text{wenn} \quad p_1 = \varepsilon \\ (h; \text{append}(t, p_2)) & \text{wenn} \quad p_1 = (h; t) \end{cases} \end{aligned} \tag{5.14}$$

Ein Plan, im Sinne einer vollständigen Handlungsanweisung für den Agenten, besteht in diesem Kontext nicht nur aus dem Anfangsvorhaben des Agenten, sondern auch aus der Prozedurmenge. Die Definition 5.15 faßt daher zusammen, was im Folgenden unter dem Begriff „Plan“ verstanden wird.

Definition 5.15. *Ein **Plan** in einer Welt* $\mathsf{W} = (\mathsf{D}, \mathcal{V}, \mathcal{N}_{\mathcal{P}}, \mathcal{P})$ *ist ein Paar* $(p, \mathcal{P})$ *von einem Vorhaben* $p \in \mathfrak{P}$ *und der Prozedurmenge* $\mathcal{P}$ *.*

6. Die Behandlung von Plankorrektheit und -termination

In diesem Kapitel werden Anforderungen wie Planausführbarkeit, -korrektheit und -termination, die ein skeptischer Agent an den Plan stellen müßte, untersucht, und eine „Planausführungsrelation“ konstruiert, die diese Eigenschaften widerspiegelt.

6.1 Termination und Korrektheit

Im Folgenden werden Planungsprobleme betrachtet, in denen der Agent ein Ziel in Form eines Umweltzustandes, der eine bestimmte Bedingung P erfüllt, erreichen will. Dafür sind drei Grundfragen interessant: Ausführbarkeit, Korrektheit und Termination. Ausführbarkeit bedeutet, daß der Agent jederzeit die Möglichkeit hat, mit der Ausführung des Planes fortzufahren, solange er nicht fertig ist. (Der Fall, daß der Agent nicht fortfahren kann, wird im Folgenden auch als **Fehlschlag** des Planes bezeichnet.) Termination eines Planes bedeutet, daß der Agent die Ausführung des Planes beendet, d.h. der Agent als Vorhaben das leere Vorhaben erreicht. Korrektheit eines Plans bedeutet, daß, wenn der Plan terminiert, das Ziel, das der Agent anstrebt, erreicht wird.

Dabei ist der Umfang der Anforderungen für einen skeptischen und einen mutigen Agenten unterschiedlich: Ein skeptischer Agent verlangt, daß die geforderten Eigenschaften unter allen Umständen gelten, d.h. für jede mögliche Ausganssituation, für jede Ausführungsmöglichkeit des Plans, und jedem möglichen Ergebnis der Aktionen. Ein mutiger Agent kann diese Forderungen geeignet abschwächen, indem er z.B. zuläßt, daß der Plan unter Umständen fehlschlägt. Dies zeigt das folgende Beispiel.

Beispiel 6.1. *Ein Agent besitzt einen Fernseher, der plötzlich aussetzt. Sein Ziel besteht nun darin, daß der Fernseher wieder funktioniert. Angenommen er faßt nun den Plan, entweder mit der Faust gegen den Fernseher zu schlagen, oder ihn an- und auszuschalten. (Welche der beiden Varianten bei der*

Planausführung zur Anwendung kommen soll, wird also nicht spezifiziert.) Dieser Plan kann durchaus zum Ziel führen, falls der Fernseher einen einfachen Wackelkontakt hat. An diesem Plan lassen sich aber auch mehrere mögliche Arten des Fehlschlagens demonstrieren:

1. Der Plan hat in einigen Anfangssituationen Erfolg, in anderen nicht: *Wenn der Fernseher keinen Wackelkontakt hat, sondern ein Bauteil defekt ist, dann wird dieser Plan nicht zum Ziel führen.*

2. Der Plan enthält Aktionen mit nichtdeterministischen Effekten, wobei einige Ausgänge der Aktion den Plan fehlschlagen lassen, und andere nicht: *Bei einem Wackelkontakt ist es möglich, daß Faustschlagen diesen behebt - es kann aber auch ohne Wirkung bleiben.*

3. Der Plan enthält Entscheidungsmöglichkeiten für den Agenten (z.B. im Sinne der nichtdeterministischen Auswahl), bei denen einige Entscheidungen zum Ziel führen, und andere nicht: *Es kann sein, daß der Wackelkontakt so beschaffen ist, daß es helfen würde, mit der Faust gegen den Fernseher zu schlagen, aber nicht, den Fernseher aus- und wieder einzuschalten.*

Allgemein könnte ein mutiger Agent bei der Planung beschließen, die Möglichkeit bestimmter Arten von Fehlschlägen in Kauf zu nehmen, während ein skeptischer Agent auf einem Plan bestehen würde, der nie fehlschlagen kann. In diesem Beispiel müßte der Plan für einen skeptischen Agenten beinhalten, den Fernseher in eine Werkstatt zu schaffen.

Wenn die Modellierung der Planungsdomäne sehr umfangreich ist, wird es für einen skeptischen Agenten schwierig oder gar unmöglich, einen seinen Anforderungen entsprechenden Plan zu finden. Er müßte z.B. im obigen Beispiel einen Stromausfall der gesamten Stadt, oder einen Konkurs seiner Bank einkalkulieren. Daher erscheint es für reale Planungsprobleme sinnvoll, schwächere Konzepte als den skeptischen Agenten (z.B. den beschriebenen mutigen Agenten) einzusetzen.

Andererseits gibt es viele Möglichkeiten dafür, die Anforderungen eines mutigen Agenten zu definieren. Zudem erscheint eine Abschätzung der Erfolgswahrscheinlichkeit notwendig. (Wie gut ist ein Plan, der nur in einem von 10 Fällen zum Ziel führt?) Dafür scheint aber die klassische Logik, die nur „wahr“ und „falsch“ kennt, recht ungeeignet. In dieser Arbeit wird daher zunächst nur der skeptische Agent betrachtet, zumal ein Plan für einen skeptischen Agenten auch für einen mutigen Agenten akzeptabel ist, aber nicht umgekehrt. Ein mutiger Agent könnte Gegenstand späterer Untersuchungen sein.

Um nun die eingeführten Begriffe mit Hilfe der Weltrelation zu formulieren, ist zunächst die Frage zu klären, was „Ausführung“ eines Planes bedeutet. Ein Übergang $(s,p) \rightsquigarrow (s',p')$ beschreibt nun einen einzelnen möglichen Schritt bei der Ausführung eines Planes. Eine Folge von derartigen Schritten beschreibt also die gesamte Ausführung:

Definition 6.2. *Eine endliche* ***Berechnungsfolge*** *der Länge k zu einem Plan ist eine Folge von Weltzuständen $\big((s_i,p_i) \mid 0 \leq i \leq k\big)$, wobei*

$$\forall i \in \{1,2,\ldots,k\} \quad (s_{i-1},p_{i-1}) \rightsquigarrow (s_i,p_i)$$

gilt. Eine Berechnungsfolge wird auch $(s_0,p_0) \rightsquigarrow (s_1,p_1) \rightsquigarrow \ldots \rightsquigarrow (s_k,p_k)$ geschrieben.

Eine Berechnungsfolge $\big((s_i,p_i) \mid 0 \leq i \leq k\big)$ ***terminiert****, wenn $p_k = \varepsilon$.*

Bei jeder Berechnungsfolge handelt es sich nun um einen *möglichen* Verlauf der Ausführung des entsprechenden Plans durch den Agenten, und wenn die Berechnungsfolge terminiert, ist der Agent nach der Ausführung dieser Schritte bei diesem Verlauf der Ausführung fertig. Wenn Aktionen im Plan nichtdeterministische Effekte haben, oder eine Prozedur mit nichtdeterministischer Auswahlbedingung ausgeführt wird, so können von einem Weltzustand mehrere verschiedene Berechnungsfolgen ausgehen.

Davon ausgehend ergibt sich die Bedeutung der Begriffe „Ausführbarkeit“ und „Korrektheit“ eines Plans (in der Sichtweise eines skeptischen Agenten) wie folgt:

Definition 6.3 (Ausführbarkeit).
Ein Plan $(p_0,\mathcal{P})$ ist bezüglich einem Anfangsumweltzustand s_0 ***ausführbar****, wenn jede von (s_0,p_0) ausgehende endliche Berechnungsfolge $\big((s_i,p_i) \mid 0 \leq i \leq k\big)$ entweder terminiert, oder fortsetzbar ist, d.h. es einen Weltzustand (s_{k+1},p_{k+1}) mit $(s_k,p_k) \rightsquigarrow (s_{k+1},p_{k+1})$ gibt.*

Mit anderen Worten: der Agent wird bei der Ausführung des Plans nie in eine Sackgasse geraten, in der er die nächste Aktion, die ihm der Plan vorschreibt, nicht ausführen kann.

Definition 6.4 (Korrektheit). *Ein Plan $(p_0,\mathcal{P})$ ist bezüglich einem Anfangsumweltzustand s_0 und einer Eigenschaft P* ***korrekt****, wenn für jede von (s_0,p_0) ausgehende* terminierende *Berechnungsfolge $\big((s_i,p_i) \mid 0 \leq i \leq k\big)$ die Eigenschaft P für den Zustand s_k erfüllt ist.*[1]

[1] In dieser Arbeit werden nur Planungsprobleme betrachtet, in denen der Agent ein Ziel *erreichen* will. Damit braucht die Eigenschaft (die dieses Ziel darstellt) nur dann zu gelten, wenn der Agent fertig ist.

Eine Eigenschaft ist dabei eine Funktion des Zustands auf $\{\mathbf{0}, \mathbf{1}\}$. (Sie gibt also an, ob das Ziel erfüllt ist, oder nicht.)

Für die Definition von Plantermination gibt es mehrere Möglichkeiten. Man betrachte folgende Pläne für einen Agenten:

1. Gehe 5 Schritte vorwärts.
2. Gehe so viele Schritte vorwärts, wie du Münzen in der Tasche hast.
3. Wähle eine beliebige natürliche Zahl n, und gehe n Schritte vorwärts.
4. Gehe einen Schritt vorwärts und wirft eine Münze. Wiederhole dies solange du „Zahl“ wirfst.

All diese Pläne terminieren in einem intuitiven Sinn. Ihnen entsprechen folgende Terminationsbegriffe:

1. Es gibt eine Zahl n, so daß die Planausführung stets nach maximal n Schritten beendet ist.
2. Für jeden Anfangszustand gibt es eine Zahl n, so daß die Planausführung nach maximal n Schritten beendet ist.
3. Es ist nicht möglich, daß die Planausführung unendlich lange dauert.
4. Die Wahrscheinlichkeit, daß die Planausführung unendlich lange dauert, ist 0.

Die Begriffe werden in der angegebenen Reihenfolge immer schwächer, d.h. der nächste umfaßt den vorhergehenden. Welcher davon angemessen ist, ist abhängig von der Planungsaufgabe, die gelöst werden soll. Auf der anderen Seite erscheinen diese Begriffe immer schwieriger beherrschbar, je schwächer sie sind.

Bei den bisherigen Ansätzen im Fluent-Kalkül spielte die Frage nach Plantermination keine Rolle, da sämtliche Aktionen des Plans explizit aufgezählt wurden. (Ein nicht terminierender Plan ließ sich damit garnicht formulieren.) Zur Beschreibung der Plantermination eines solchen Plans ist Begriff 1 ausreichend, da ein Plan, der n Aktionen umfaßt, stets nach genau n Schritten terminiert.

Begriff 1 ist aber eine zu große Einschränkung, wenn Pläne so allgemein formuliert werden sollen, daß sie für eine Klasse von Anfangszuständen funktionieren. Zum Beispiel terminiert in diesem Sinne der Plan aus Beispiel 5.9 auf Seite 35 (Abbau eines Turms in der Blocksworld) in diesem Sinne nicht, da die Anzahl der Schritte der Planausführung proportional zur Anzahl der Blöcke im Turm ist, und daher im Prinzip beliebig groß werden kann. Die Schrittzahl ist aber

aus dem Anfangszustand ablesbar, und daher terminiert der Plan im Sinne von Begriff 2.

Auf der anderen Seite läßt sich die Termination eines Planes im Sinne von Begriff 2 bezüglich eines bestimmten Anfangszustandes für eine endliche Planungsdomäne durch ein einfaches Aufzählen der Möglichkeiten nachweisen. Bei Begriff 3 ist dies nicht unbedingt der Fall, wie sich an dem oben gegebenen Beispiel 3 ablesen läßt. (Begriff 4 besitzt den gleichen Nachteil, und benutzt noch dazu Wahrscheinlichkeiten, was ihn noch schwerer mit Hilfe der klassischen Logik fassbar macht.) Für diese Arbeit wird daher Begriff 2 benutzt.

Definition 6.5 (starke Termination).
Ein Plan ***terminiert*** *bezüglich einem Anfangszustand* (s_0, p_0)*, wenn es eine natürliche Zahl* n *gibt, so daß jede Berechnungsfolge, die in* (s_0, p_0) *beginnt, eine Länge von höchstens* n *hat.*

Dies bedeutet also, daß es einen Zeitpunkt gibt, zu dem der Agent mit Sicherheit fertig ist.

Zusammenfassung: Wenn ein skeptischer Agent sicher sein will, sein Ziel mit einem bestimmten Plan zu erreichen, so muß er sicherstellen, daß der Plan die 3 Eigenschaften Ausführbarkeit, Termination und Korrektheit bezüglich dem Ziel erfüllt. Die Ausführbarkeit garantiert, daß der Agent nicht in einer „Sackgasse“ stecken bleibt, sondern immer mit der Ausführung des Planes fortfahren kann, solange er nicht fertig ist. Die Termination garantiert, daß er in endlicher Zeit fertig wird, und die Korrektheit garantiert dann, daß er sein Ziel erreicht.

6.2 Die Planausführungsrelation

Es wäre nun recht naheliegend, die transitive Hülle $\mathcal{C}_{\mathsf{W}}{}^{\mathrm{tra}}$ der Weltrelation (im Folgenden durch den Operator $\overset{*}{\rightsquigarrow}$ dargestellt) zum Beurteilen des Plans zu nutzen. Damit drückt $(s,p) \overset{*}{\rightsquigarrow} (s',\varepsilon)$ aus, daß eine Berechnungsfolge existiert, die von (s,p) zu (s',ε) führt.[2] Auf diese Weise lassen sich die Korrektheit und Ausführbarkeit bezüglich eines Anfangszustands (s_0, p_0) leicht ausdrücken:

$$\text{Ausführbarkeit: } \forall s,p \;\; (s_0,p_0) \overset{*}{\rightsquigarrow} (s,p) \;\; \rightarrow \;\; p = \varepsilon \;\vee\; \exists s',p' \;\; (s,p) \rightsquigarrow (s',p').$$
$$\text{Korrektheit: } \forall s \;\; (s_0,p_0) \overset{*}{\rightsquigarrow} (s,\varepsilon) \;\; \rightarrow \;\; P(s).$$

[2] Dies entspricht weitgehend der Verwendung des Makros *Do* in GOLOG (siehe Seite 21).

Die Verwendung der transitiven Hülle hat jedoch Nachteile. Erstens läßt sich die transitive Hülle in Logik erster Stufe nicht endlich axiomatisieren. Zweitens läßt sich mit Hilfe von $\overset{*}{\rightsquigarrow}$ die Termination nicht direkt ausdrücken, da man dazu den Begriff der Länge einer Berechnungsfolge braucht. Damit wird es schwierig, die Termination eines Plans nachzuweisen. Daher wird eine Relation eingeführt, die den Begriff der Länge einer Berechnungsfolge integriert.

Definition 6.6. *Die* ***Planausführungsrelation*** $\mathcal{C}_n \subseteq \mathcal{S} \times \mathfrak{P} \times \mathbb{N} \times (\mathcal{S} \cup \{fail\})$ *einer Weltrelation* $\mathcal{C}_\mathsf{W}$ *sei die Menge aller Quadrupel* (s_0, p_0, n, s')*, für die eine der folgenden Aussagen gilt:*

1. *Es gibt eine Berechnungsfolge* $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ *mit* $p_k = \varepsilon$ *und* $s_k = s'$ *und einer Länge* $k \leq n$*.*

2. *Es gibt eine Berechnungsfolge* $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ *mit einer Länge* $k > n$*, und* $s' = fail$*.*

3. *Es gibt eine Berechnungsfolge* $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ *mit* $p_k \neq \varepsilon$*, für die keine Fortsetzung existiert (d.h. es gibt kein* (s_{k+1}, p_{k+1}) *mit* $(s_k, p_k) \rightsquigarrow (s_{k+1}, p_{k+1})$*), und es gilt* $s' = fail$*.*

$s_0 \xrightarrow[n]{p_0} s'$ *ist eine Abkürzung für* $(s_0, p_0, n, s') \in \mathcal{C}_n$*.*

Dabei ist $fail$ ein spezieller Zustand, der die Möglichkeit eines Fehlschlags der Planausführung ausdrückt, wenn er erreicht werden kann, d.h. $s_0 \xrightarrow[n]{p} fail$ gilt. n stellt eine Längenbegrenzung dar: $s_0 \xrightarrow[n]{p} s_1$ drückt aus, daß die Planausführung in Zustand s_1 terminieren kann, wobei die Planausführung in maximal n Schritten erfolgt, d.h. die entsprechende Berechnungsfolge eine Länge von höchstens n hat (Fall 1). Die Planausführungsrelation zeigt in zwei Fällen einen möglichen Planfehlschlag durch $s_0 \xrightarrow[n]{p} fail$ an: zum einen bei Überschreitung der Längenbegrenzung (Fall 2) oder dadurch, daß der Plan gemäß Definition 6.3 nicht ausführbar ist (Fall 3).

6.3 Eigenschaften der Planausführungsrelation

Mit Hilfe der Planausführungsrelation lassen sich nun die Anforderungen eines skeptischen Agenten wie folgt beschreiben:

Satz 6.7. *Bezüglich einem Anfangsumweltzustand* s_0 *ist ein Plan* $(p_0, \mathcal{P})$ *genau dann ausführbar, terminierend und korrekt bezüglich einer Eigenschaft* P

(im Sinne von Definitionen 6.5,6.3 und 6.4), wenn

$$\exists n \in \mathbb{N} \;\; \forall s' \in \mathcal{S} \;\; s_0 \xrightarrow[n]{p_0} s' \;\; \Longrightarrow \;\; s' \neq fail \;\wedge\; P(s'). \tag{6.8}$$

Beweis. Die Äquivalenz wird in zwei Implikationen zerlegt.

- „$\Rightarrow$“: Angenommen der Plan ist ausführbar, terminierend und korrekt. Nach Definition 6.5 auf Seite 43 für die Termination findet man ein n, so daß jede Berechnungsfolge, die von (s_0, p_0) ausgeht, eine Länge von höchstens n hat. Angenommen es gilt $s_0 \xrightarrow[n]{p_0} s'$. Fall 2 in Definition 6.6 ist nun für dieses n unmöglich, da alle von (s_0, p_0) ausgehenden Berechnungsfolgen eine Länge von höchstens n haben. Fall 3 ist wegen der Planausführbarkeit (Definition 6.3) ebenfalls ausgeschlossen. Es bleibt Fall 1 übrig, d.h. es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ mit $p_k = \varepsilon$ und $s_k = s'$. Da der Plan korrekt ist, gilt nach Definition 6.4 $P(s')$. Also ist (6.8) erfüllt.

- „$\Leftarrow$“: Angenommen es gilt (6.8). Man findet also ein n, so daß

  $$\forall s' \;\; s_0 \xrightarrow[n]{p_0} s' \;\; \Longrightarrow \;\; s' \neq fail \;\wedge\; P(s'). \tag{i}$$

  Für jede beliebige von (s_0, p_0) ausgehende Berechnungsfolge

  $$\big((s_i, p_i) \mid 0 \leq i \leq k\big)$$

  gilt also:

  - Sie ist nicht länger als n, da sonst wegen Definition 6.6 Fall 2 $s_0 \xrightarrow[n]{p_0} fail$ gelten würde, was i widerspricht. Da dies für jede von (s_0, p_0) ausgehende Berechnungsfolge gilt, terminiert der Plan nach Definition 6.5.
  - Sie terminiert oder sie ist fortsetzbar, da sonst nach Definition 6.6 Fall 3 $s_0 \xrightarrow[n]{p_0} fail$ gelten würde, was i widerspricht. Da dies für jede Folge gilt, ist Plan also nach Definition 6.3 ausführbar.
  - Da die Berechnungsfolge nach höchstens n Schritten terminiert, gilt $s_0 \xrightarrow[n]{p_0} s_k$, und aus (i) folgt damit $P(s')$. Nach Definition 6.4 ist der Plan also bezüglich P korrekt.

  Der Plan ist also ausführbar, terminierend und korrekt.

■

Für die Planausführungsrelation gelten eine Monotonieeigenschaft und eine Anti-Monotonieeigenschaft, die in den folgenden Korollaren beschrieben werden. Diese können u.U. Schlüsse vereinfachen.

Korollar 6.9. *Für eine Planausführungsrelation* $\mathcal{C}_n$ *und für* $s_2 \neq fail$ *gilt:*

$$s_1 \xrightarrow[n]{p} s_2 \quad \Longrightarrow \quad s_1 \xrightarrow[n+1]{p} s_2$$

Beweis. Aus $s_1 \xrightarrow[n]{p} s_2$ und $s_2 \neq fail$ folgt nach Definition 6.6 auf Seite 44 daß es eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ mit $p_k = \varepsilon$ gibt, deren Länge k höchstens n ist. Es gilt also auch $k < n+1$, so daß nach Definition 6.6 Fall 1 $s_1 \xrightarrow[n+1]{p} s_2$ folgt. ∎

Korollar 6.10. *Für eine Planausführungsrelation* $\mathcal{C}_n$ *gilt:*

$$s_1 \xrightarrow[n+1]{p} fail \;\rightarrow\; s_1 \xrightarrow[n]{p} fail$$

Beweis. Angenommen es gilt $s_1 \xrightarrow[n+1]{p} fail$. Gemäß Definition 6.6 sind nun zwei Fälle zu unterscheiden:

1. Es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ mit einer Länge $k > n+1$. Da damit auch $k > n$ gilt, folgt aus Definition 6.6 Fall 2, daß $s_1 \xrightarrow[n]{p} fail$.

2. Es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ mit $p_k \neq \varepsilon$, für die keine Fortsetzung existiert. Nach Definition 6.6 Fall 3 gilt also $(s_0, p_0, n, fail) \in \mathcal{C}_n$.

In beiden Fällen ist also die Behauptung erfüllt. ∎

Den Abschluß dieses Kapitels bildet ein Lemma, daß in Kapitel 7 für die Darstellung der Planausführungsrelation mit Hilfe von Logik benötigt wird. Es formt die Definition 6.6 der Planausführungsrelation in eine rekursive Form um.

Lemma 6.11. *Für eine Planausführungsrelation* $\mathcal{C}_n \in \mathcal{S} \times \mathfrak{P} \times \mathbb{N} \times (\mathcal{S} \cup \{fail\})$ *gilt:*

$$\begin{aligned}
&\forall s_0, s_n \in \mathcal{S}, p_0 \in \mathfrak{P}, n \in \mathbb{N} \quad s_0 \xrightarrow[n]{p_0} s_n \quad \Longleftrightarrow \\
&\quad (p_0 = \varepsilon \;\wedge\; s_0 = s_n) \;\vee \\
&\quad (p_0 \neq \varepsilon \;\wedge\; n = 0 \;\wedge\; s_n = fail) \;\vee \\
&\quad \big(\exists s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \quad p_0 \neq \varepsilon \;\wedge\; n > 0 \;\wedge\; (s_0, p_0) \rightsquigarrow (s_1, p_1) \;\wedge\; s_1 \xrightarrow[n-1]{p_1} s_n\big) \;\vee \\
&\quad \big(\forall s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \quad p_0 \neq \varepsilon \;\wedge\; n > 0 \;\wedge\; (s_0, p_0) \not\rightsquigarrow (s_1, p_1) \;\wedge\; s_n = fail\big).
\end{aligned}$$

Beweis. Zur Abkürzung bezeichne ich mit (a),(b),(c) und (d) die folgenden Ausdrücke:

$$(p_0 = \varepsilon \ \wedge \ s_0 = s_n). \tag{a}$$

$$(p_0 \neq \varepsilon \ \wedge \ n = 0 \ \wedge \ s_n = fail). \tag{b}$$

$$\big(\exists s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \ \ p_0 \neq \varepsilon \ \wedge \ n > 0 \ \wedge \ (s_0, p_0) \rightsquigarrow (s_1, p_1) \ \wedge \ s_1 \xrightarrow[n-1]{p_1} s_n\big). \tag{c}$$

$$\big(\forall s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \ \ p_0 \neq \varepsilon \ \wedge \ n > 0 \ \wedge \ (s_0, p_0) \not\rightsquigarrow (s_1, p_1) \ \wedge \ s_n = fail\big). \tag{d}$$

Die Behauptung lautet damit:

$$\forall s_0, s_n \in \mathcal{S}, p_0 \in \mathfrak{P}, n \in \mathbb{N} \ \ s_0 \xrightarrow[n]{p_0} s_n \iff \text{(a)} \ \vee \ \text{(b)} \ \vee \ \text{(c)} \ \vee \ \text{(d)}. \tag{i}$$

Es werden vier Fälle unterschieden.

- Es gelte

  $$p_0 \neq \varepsilon \ \wedge \ s_n = fail \ \wedge \ n > 0. \tag{ii}$$

  Die Behauptung (i) wird in zwei Implikationen zerlegt.

  „$\Rightarrow$“ Angenommen es gilt $s_0 \xrightarrow[n]{p_0} s_n$ sowie (ii). Daraus folgt

  $$s_0 \xrightarrow[n]{p_0} fail.$$

  Nach Definition 6.6 gibt es dafür zwei Möglichkeiten:

  – Es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ mit einer Länge $k > n$. Da $k > n > 0$ ist $\big((s_i, p_i) \mid 1 \leq i \leq k\big)$ eine Berechnungsfolge der Länge $k - 1 > n - 1$, d.h. $s_1 \xrightarrow[n-1]{p_1} fail$ ist erfüllt. Mit (ii) ergibt sich

  $$p_0 \neq \varepsilon \ \wedge \ n > 0 \ \wedge \ (s_0, p_0) \rightsquigarrow (s_1, p_1) \ \wedge \ s_1 \xrightarrow[n-1]{p_1} s_n,$$

  so daß (c) erfüllt ist, und damit (a) $\vee$ (b) $\vee$ (c) $\vee$ (d).

  – Es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ für die keine Fortsetzung existiert.
  Wenn nun $k > 0$ dann ist $\big((s_i, p_i) \mid 1 \leq i \leq k\big)$ eine Berechnungsfolge für die keine Fortsetzung existiert, d.h. nach Definition 6.6 Fall 3 folgt $s_1 \xrightarrow[n-1]{p_1} fail$. Mit (ii) ergibt sich

  $$p_0 \neq \varepsilon \ \wedge \ n > 0 \ \wedge \ (s_0, p_0) \rightsquigarrow (s_1, p_1) \ \wedge \ s_1 \xrightarrow[n-1]{p_1} s_n,$$

so daß (c) erfüllt ist, und damit (a) $\vee$ (b) $\vee$ (c) $\vee$ (d).
Wenn dagegegen $k = 0$ gilt, gibt es also keine Fortsetzung für $\big((s_0, p_0)\big)$, d.h

$$\forall s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \;\; (s_0, p_0) \not\rightsquigarrow (s_1, p_1).$$

Mit (ii) ergibt sich

$$p_0 \neq \varepsilon \;\wedge\; n > 0 \;\wedge\; (s_0, p_0) \not\rightsquigarrow (s_1, p_1) \;\wedge\; s_n = fail,$$

so daß (d) erfüllt ist, und damit ebenfalls (a) $\vee$ (b) $\vee$ (c) $\vee$ (d).

„$\Leftarrow$“ Angenommen es gilt (a) $\vee$ (b) $\vee$ (c) $\vee$ (d). Wegen (ii) können (a) sowie (b) nicht gelten, d.h. (c) $\vee$ (d) ist erfüllt. Danach lassen sich zwei Fälle unterscheiden:

- Es gilt (c). Man findet also s_1 und p_1, so daß

$$(s_0, p_0) \rightsquigarrow (s_1, p_1) \;\wedge\; s_1 \xrightarrow[n-1]{p_1} fail \qquad \text{(iii)}$$

  Nach Definition 6.6 gibt es für $s_1 \xrightarrow[n]{p_1} fail$ zwei Möglichkeiten:

  - ▷ Es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 1 \leq i \leq k\big)$ mit einer Länge $k-1 > n-1$. Wegen (iii) ist nun $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ eine Berechnungsfolge der Länge $k > n$, d.h. nach Definition 6.6 Fall 2 ist $s_0 \xrightarrow[n]{p_0} fail$ erfüllt, und damit auch

$$s_0 \xrightarrow[n]{p_0} s_n,$$

    d.h. die Behauptung ist erfüllt.
  - ▷ Es gibt eine Berechnungsfolge $\big((s_i, p_i) \mid 1 \leq i \leq k\big)$, für die keine Fortsetzung existiert.
    Wegen (iii) ist nun $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ eine Berechnungsfolge, und diese hat ebenfalls keine Fortsetzung. Nach Definition 6.6 Fall 3 ist $s_0 \xrightarrow[n]{p_0} fail$ erfüllt, und damit auch

$$s_0 \xrightarrow[n]{p_0} s_n,$$

    d.h. die Behauptung ist erfüllt.
- Es gilt (d). Damit ist also die Berechnungsfolge $\big((s_0, p_0)\big)$ nicht fortsetzbar, und wegen (ii) ist Fall 3 von Definition 6.6 gilt $s_0 \xrightarrow[n]{p_0} s_n$, d.h. die Behauptung ist erfüllt.

- $p = \varepsilon$: Damit gibt es von (s_0, p_0) aus nur die Berechnungsfolge $\big((s_0, p_0)\big)$ der Länge 0. In der Definition 6.6 auf Seite 44 der Planausführungsrelation

kann damit der Fall 2 nicht gelten, und Fall 3 wegen $p = \varepsilon$ ebenfalls nicht, d.h. $s_0 \xrightarrow[n]{p_0} s_n$ gilt genau dann, wenn Fall 1 gilt, d.h.

$$s_0 \xrightarrow[n]{p_0} s_n \iff (p_0 = \varepsilon \wedge s_0 = s_n). \tag{iv}$$

Wegen $p = \varepsilon$ sind (b),(c) und (d) falsch. Die Konjunktion von (a),(b),(c) und (d) ist daher äquivalent zu (a), d.h.

$$\text{(a)} \vee \text{(b)} \vee \text{(c)} \vee \text{(d)} \iff (p_0 = \varepsilon \wedge s_0 = s_n),$$

woraus mit (iv) die Behauptung (i) folgt.

- $p \neq \varepsilon \wedge s_n \neq fail$: Wegen $s_n \neq fail$ sind in Definition 6.6 Fall 2 und 3 ausgeschlossen.

  „$\Rightarrow$“ Angenommen es gilt $s_0 \xrightarrow[n]{p_0} s_n$.
  Nach Definition 6.6 Fall 1 findet man also eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ mit $p_k = \varepsilon$ und einer Länge $k \leq n$. Aus $p \neq \varepsilon$ folgt nun $p_0 \neq p_k$ und damit $n \geq k > 0$, d.h. $\big((s_i, p_i) \mid 1 \leq i \leq k\big)$ ist ebenfalls eine Berechnungsfolge mit einer Länge $k - 1 \leq n - 1$, so daß nach Definition 6.6 Fall 1 $s_0 \xrightarrow[n-1]{p_0} s_1$ gilt. Ebenso folgt aus Definition 6.2 auf Seite 41, daß $(s_0, p_0) \rightsquigarrow (s_1, p_1)$. Es gilt also

  $$p_0 \neq \varepsilon \wedge n > 0 \wedge (s_0, p_0) \rightsquigarrow (s_1, p_1) \wedge s_1 \xrightarrow[n-1]{p_1} s_n.$$

  Damit ist (c) erfüllt, woraus die Behauptung (a) $\vee$ (b) $\vee$ (c) $\vee$ (d) folgt.

  „$\Leftarrow$“ Angenommen es gilt (a) $\vee$ (b) $\vee$ (c) $\vee$ (d) .
  Wegen $p \neq \varepsilon \wedge s_n \neq fail$ können nun weder (a), (b) noch (d) erfüllt sein, d.h. es muß

  $$\big(\exists s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \;\; p_0 \neq \varepsilon \wedge n > 0 \wedge (s_0, p_0) \rightsquigarrow (s_1, p_1) \wedge s_1 \xrightarrow[n-1]{p_1} s_n\big).$$

  gelten. Nach Definition 6.6 Fall 1 findet man also eine Berechnungsfolge $\big((s_i, p_i) \mid 1 \leq i \leq k\big)$ mit $p_k = \varepsilon$ und einer Länge von $k - 1 < n - 1$. Wegen (c) gilt aber auch $(s_0, p_0) \rightsquigarrow (s_1, p_1)$, d.h. die Folge $\big((s_i, p_i) \mid 0 \leq i \leq k\big)$ ist ebenfalls eine Berechnungsfolge. Sie hat die Länge $k \leq n$ und endet mit $p_k = \varepsilon$, so daß nach Definition 6.6 Fall 1 $s_0 \xrightarrow[n-1]{p_0} s_n$ folgt, und damit die Behauptung.

- $p \neq \varepsilon \wedge s_n = fail \wedge n = 0$ Nach Definition 6.6 Fall 2 bzw. 3 ist nun $s_0 \xrightarrow[n]{p_0} s_n$ erfüllt. Da b ebenfalls erfüllt ist, gilt die Behauptung.

■

7. Beschreibung mit Hilfe von Prädikatenlogik

In diesem Kapitel wird aus der Welt-Beschreibung eine Beschreibung mit Hilfe von Logik generiert, und deren Äquivalenz zu der $\mathcal{C}_n$-Relation bewiesen.

7.1 Abbildung der Domäne

Den folgenden Betrachtungen wird ein Alphabet der Prädikatenlogik gemäß Definition 2.1 auf Seite 3 zugrundegelegt, daß die Objekte der betrachteten Domäne (Definition 5.1 auf Seite 29) als Konstanten enthält, die Aktionsbezeichner und Bedingungsbezeichner als Funktionssymbole entsprechender Stelligkeit; ebenso enthalte das Alphabet die Prozedurnamen einer Welt (Definition 5.11 auf Seite 37) als Funktionssymbole und die Variablen der Welt als Variablen. Damit sind die Elemente von $\mathcal{N}_{\mathcal{A}}(\mathcal{O}), \mathcal{N}_{\mathcal{A}}(\mathcal{O} \cup \mathcal{V}), \mathcal{N}_{\mathcal{C}}(\mathcal{O}), \mathcal{N}_{\mathcal{C}}(\mathcal{O} \cup \mathcal{V})$, $\mathcal{N}_{\mathcal{P}}(\mathcal{O})$ und $\mathcal{N}_{\mathcal{P}}(\mathcal{V})$ Terme der Logik. Weiterhin enthalte das Alphabet das Funktionssymbol ; (das allerdings im Folgenden weiterhin in Infix-Notation geschrieben wird), sowie geeignete Konstanten und Funktionssymbole zur Darstellung der Zustände.

Inhalt dieses Kapitels ist es nun, eine Menge von logischen Formeln ($\mathcal{F}_{\mathsf{W}}$) aufzustellen, so daß die Relation $\mathcal{C}_n$ durch ein Prädikat *doplan* beschrieben wird, d.h. in einem Modell von ($\mathcal{F}_{\mathsf{W}}$) $doplan(s^\tau, p, s^n(0), s'^\tau)$ [1] genau dann wahr ist, wenn $s \xrightarrow[n]{p} s'$ gilt.

Analog zu Kapiteln 5 und 6 wird wieder von einer Beschreibung der Domäne ausgegangen, aber diesmal in Form einer Menge von logischen Formeln[2], die als Domänenmodellierung bezeichnet wird. Zu dieser gehören einerseits eine Abbildung $^\tau$, die es gestattet, Zustände als Terme darzustellen, und andererseits

[1] $^\tau$ ist eine Abbildung von $\mathcal{S}$ auf Terme, sowie $s^n(0)$ eine Termdarstellung für natürliche Zahlen. Diese werden später besprochen.

[2] Zum Beispiel als Vervollständigung eines logischen Programms.

eine Menge von logischen Formeln (Dom), die die Prädikate *causes* und *holds* definieren, die die Relationen $\mathcal{C}_p$ und $\mathcal{H}$ der Domäne darstellen:

Definition 7.1.
Eine ***Domänenmodellierung*** *einer Domäne* $\mathsf{D} = (\mathcal{O}, \mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{C}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$ *ist ein Paar* $((Dom), ^\tau)$, *wobei* (Dom) *eine Menge von Formeln ist,* $^\tau$ *eine Abbildung von* $\mathcal{S}$ *auf Terme des Alphabets, und es gilt:*

$$(Dom) \models {s_1}^\tau = {s_2}^\tau \iff s_1 = s_2. \tag{7.2a}$$
$$(Dom) \models causes({s_1}^\tau, a, {s_2}^\tau) \iff (s_1, a, s_2) \in \mathcal{C}_p. \tag{7.2b}$$
$$(Dom) \models holds(c, {s_1}^\tau) \iff (c, s_1) \in \mathcal{H}. \tag{7.2c}$$

Weiterhin sei die Domänenmodellierung vollständig, d.h.

$$(Dom) \not\models {s_1}^\tau = {s_2}^\tau \iff (Dom) \models {s_1}^\tau \neq {s_2}^\tau. \tag{7.2d}$$
$$(Dom) \not\models causes({s_1}^\tau, a, {s_2}^\tau) \iff (Dom) \models \neg\, causes({s_1}^\tau, a, {s_2}^\tau). \tag{7.2e}$$
$$(Dom) \not\models holds(c, {s_1}^\tau) \iff (Dom) \models \neg\, holds(c, {s_1}^\tau). \tag{7.2f}$$

Für den Fluent-Kalkül existieren bereits mehrere als vollständig und korrekt bewiesene Modellierungen [HT95, Thi97, Thi96, BT96, BT97, Leh97], die für die Verwendung als Domänenmodellierung angepaßt werden könnten, so daß in dieser Arbeit verzichtet wird, eine weitere anzugeben. Als Beispiel, wie eine solche Domänenmodellierung aussehen könnte, ist eine Übertragung des einfachen Aktionsmodells aus Kapitel 3 in ein logisches Programm angegeben.

Beispiel 7.3 (Blocksworld). *Ein Beispiel für eine Domänenmodellierung der Blocksworld-Domäne (siehe auch Beispiel 5.2 auf Seite 30) besteht aus der Abbildung* $^\tau$ *gemäß Gleichung* (3.7a) *auf Seite 14 und der folgenden Menge von Axiomen:*

1. *Es wird im Sinne von Kapitel 3 ein Prädikat action definiert, so daß* $action(c, a, e)$ *genau dann erfüllt ist, wenn* ${c}^{\tau^{-1}}$ *die Effekte und* ${e}^{\tau^{-1}}$ *der Aktion* a *sind.*[3]

[3] Diese Formel bildet die sogenannte vervollständigte Form des *action*-Prädikats analog zu Kapitel 3 3.4. (Zur Vervollständigung siehe auch [Llo87]).

$$\begin{aligned}
&\forall c, a, e \quad action(c, a, e) \;\leftrightarrow \\
&\quad \Big(\exists x, y \quad a =_{\text{AC1}} totabl(x) \;\wedge\; c =_{\text{AC1}} cl(x) \circ on(x, y) \;\wedge \\
&\qquad e =_{\text{AC1}} cl(x) \circ cl(y) \circ ot(y)\Big) \;\vee \\
&\quad \Big(\exists x, y \quad a =_{\text{AC1}} move(x, y) \wedge \Big(\\
&\qquad \big(c =_{\text{AC1}} cl(x) \circ ot(x) \circ cl(y) \;\wedge\; e =_{\text{AC1}} cl(x) \circ on(x, y)\big) \;\vee \\
&\qquad \big(\exists z \quad c =_{\text{AC1}} cl(x) \circ on(x, z) \circ cl(y) \;\wedge\; e =_{\text{AC1}} cl(x) \circ on(x, y) \circ cl(z)\big) \\
&\quad \Big)\Big).
\end{aligned}$$

2. *Das Prädikat* $causes(s_1, a, s_2)$ *ist genau dann erfüllt, wenn die Aktion* a *von Zustand* s_1 *in Zustand* s_2 *führen kann.*

$$\begin{aligned}
&\forall s_1, a, s_2 \quad causes(s_1, a, s_2) \;\leftrightarrow \\
&\quad \big(\exists c, e, r \quad action(c, a, e) \;\wedge\; c \circ r =_{\text{AC1}} s_1 \;\wedge\; e \circ r =_{\text{AC1}} s_2\big).
\end{aligned}$$

3. *Die sogenannte unifikationsvollständige Theorie (AC1*) (siehe [HT95]). Diese beinhaltet die Axiome* (AC1) *auf Seite 12 und eine Menge von Axiomen, so daß für zwei Terme* s *und* t*, die nicht gleich nach* (AC1) *sind,* $\neg\, s =_{\text{AC1}} t$ *folgt.*[4]

7.2 Beschreibung der Planausführungsrelation

Aufbauend auf der Domänenbeschreibung werden nun einige Hilfsprädikate definiert, und später zum Prädikat *doplan* zusammengesetzt.[5]

Das Prädikat *append* bildet das Pendant zur append-Funktion (Gleichung (5.14) auf Seite 38). Es wird von der folgenden Formel beschrieben. Die die beiden lezten Zeilen entsprechen dabei dem ersten bzw. dem zweiten Fall in Gleichung (5.14).

$$\begin{aligned}
&\forall p_1, p_2, p_n \quad append(p_1, p_2, p_n) \quad\longleftrightarrow\quad \Big(\\
&\quad \big(p_1 = \varepsilon \;\wedge\; p_2 = p_n\big) \;\vee \\
&\qquad \exists h, t, t_n \quad \big(p_1 = (h; t) \;\wedge\; p_n = (h; t_n) \;\wedge\; append(t, p_2, t_n)\big)\Big). \qquad (7.4)
\end{aligned}$$

[4] Aus diesen Axiomen folgen damit auch die sogenannten „unique names“ Axiome.
[5] Im Folgenden wird der Kürze halber für $=_{\text{AC1}}$ das Symbol $=$ benutzt.

Die Prozedurmenge $\mathcal{P}$ einer Welt werden mit Hilfe des Prädikats *pexpand* kodiert. $pexpand(h, c, b)$ ist genau dann wahr, wenn h, c und b Grundinstantiierungen von Prozedurkopf, Auswahlbedingung und Prozedurkörper einer Prozedur aus $\mathcal{P}$ sind.

$$\forall h, c, b \;\; pexpand(h, c, b) \iff \Big(\bigvee_{(\tilde{h},\tilde{c},\tilde{b})\in\mathcal{P}} \exists\big(\mathrm{var}(\tilde{h}) \cup \mathrm{var}(\tilde{c}) \cup \mathrm{var}(\tilde{b})\big) \;\; (h = \tilde{h} \;\wedge\; c = \tilde{c} \;\wedge\; b = \tilde{b})\Big). \quad (7.5)$$

Dabei bedeutet $\exists\big(\mathrm{var}(\tilde{h}) \cup \mathrm{var}(\tilde{c}) \cup \mathrm{var}(\tilde{h})\big)$, daß jede freie Variable in $\tilde{h}, \tilde{c}$ sowie $\tilde{b}$ existenzquantifiziert wird.

Das Prädikat *world* repräsentiert die Weltrelation gemäß Definition 5.11 auf Seite 37. Die einzelnen existenzquantifizierten Ausdrücke entsprechen den Zeilen 5.13a bis 5.13d.

$$\begin{aligned}
&\forall s_1, p_1, s_2, p_2 \;\; world(s_1, p_1, s_2, p_2) \iff \\
&\quad \big(\exists h, t \;\; p_1 = (h; t) \;\wedge\; p_2 = t \;\wedge\; causes(s_1, h, s_2)\big) \;\vee \\
&\qquad \big(\exists h, t, p_a, p_b \;\; p_1 = (\mathrm{if}(h, p_a, p_b); t) \;\wedge\; append(p_a, t, p_2) \\
&\qquad\qquad \wedge\; holds(h, s_1) \;\wedge\; s_1 = s_2\big) \;\vee \\
&\qquad \big(\exists h, t, p_a, p_b \;\; p_1 = (\mathrm{if}(h, p_a, p_b); t) \;\wedge\; append(p_b, t, p_2) \;\wedge \\
&\qquad\qquad \neg\, holds(h, s_1) \;\wedge\; s_1 = s_2\big) \;\vee \\
&\qquad \big(\exists h, t, c, b \;\; p_1 = (h; t) \;\wedge\; pexpand(h, c, b) \;\wedge\; holds(c, s_1) \;\wedge \\
&\qquad\qquad append(b, t, p_2) \;\wedge\; s_1 = s_2\big) \\
&\quad \Big).
\end{aligned} \quad (7.6)$$

Das Prädikat *doplan* repräsentiert nun die Relation $\mathcal{C}_n$. Zur Abbildung der natürlichen Zahlen wird ein Funktionssymbol s eingeführt, daß den Nachfolger bezeichnet, und eine Konstante 0 , die Zahl 0 bezeichnet. Eine natürliche Zahl n wird also durch den Term $s^n(0)$ bezeichnet, wobei

$$\forall n \in \mathbb{N} \;\; s^n(0) = \begin{cases} 0 & \text{wenn} \quad n = 0 \\ s(s^{n-1}(0)) & \text{sonst.} \end{cases} \quad (7.7)$$

Die logische Formel für *doplan* entspricht der rekursiven Darstellung der Planausführungsrelation nach Lemma 6.11. Dabei stellt $\exists n_1 \;\; n = s(n1)$ nach der oben eingeführten Darstellung für natürliche Zahlen dar, daß n der Nachfolger

von $n1$ ist. Damit wird einerseits die Bedingung $n > 0$ dargestellt, und andererseits wird der Wert $n - 1$, der in der Definition nach Lemma 6.11 benötigt wird, erzeugt.

$$
\begin{array}{l}
\forall s_0, p_0, n, s_n \;\; doplan(s_0, p_0, n, s_n) \;\; \longleftrightarrow \;\; \Big(\\
\quad \big(p_0 = \varepsilon \;\wedge\; s_0 = s_n\big) \;\vee \\
\quad \big(p_0 \neq \varepsilon \;\wedge\; n = 0 \;\wedge\; s_n = fail\big) \;\vee \\
\quad \exists n_1 \;\; \big(p \neq \varepsilon \;\wedge\; n = s(n_1) \;\wedge \\
\qquad \exists s_1, p_1 \;\; \big(world(s_0, p_0, s_1, p_1) \;\wedge\; doplan(s_1, p_1, n_1, s_n)\big) \;\vee \\
\qquad \forall s_1, p_1 \;\; \big(\neg\, world(s_0, p_0, s_1, p_1) \;\wedge\; s_n = fail\big) \\
\quad \big) \\
\Big).
\end{array}
\tag{7.8}
$$

Zusammengefasst bilden diese Formeln mit der Domänenbeschreibung einer Planungsdomäne eine logische Weltmodellierung:

Definition 7.9. *Eine* ***logische Weltmodellierung*** $(\mathcal{F}_{\mathsf{W}})$ *einer Welt* W *mit entsprechender Domänenmodellierung* $((Dom), ^{\tau})$ *ist die Menge von logischen Formeln* (Dom), (7.5), (7.4), (7.6) *und* (7.8).

Wie im folgenden Abschnitt bewiesen wird, erzielt eine logische Weltmodellierung eine äquivalente Darstellung der Planausführungsrelation. [6]

7.3 Äquivalenzbeweis

In diesem Abschnitt wird nachgewiesen, daß die logische Weltmodellierung einer Planungsdomäne tatsächlich die Planausführung in dieser Domäne darstellt, d.h. die Planausführungsrelation zu dem Prädikat *doplan* korrespondiert.[7] Dazu werden zunächst einige Lemmata bewiesen, die dafür benötigt werden.

Das erste Lemma befasst sich mit Darstellung der Funktion append durch das Prädikat *append*.

[6] Die angegebenen Formeln sind entsprechen zwar einem vervollständigten logischen Programm, aber sie sind nicht mit Hilfe von SLDENF abarbeitbar, da die Abarbeitung z.B. bei dem Literal $\neg\, world(s_1, p_1, s_2, p_2)$ in (7.8) floundern würde. Wie in Kapitel 9 begründet wird, ist dies aber unerheblich.

[7] Es wird beim Beweis vorausgesetzt, daß logische Weltmodellierung ein Modell besitzt.

Lemma 7.10.

$$\forall p_1, p_2, p_n \in \mathfrak{P}$$
$$(\mathcal{F}_{\mathsf{W}}) \models \mathit{append}(p_1, p_2, p_n) \iff p_n = \text{append}(p_1, p_2). \tag{7.11}$$

Beweis. Der Beweis erfolgt per Induktion über die Länge von p_1 .

Induktionsanfang Es gelte $p_1 = \varepsilon$, d.h. die Länge von p_1 ist 0. Damit ist entsprechend Formel 7.4 $(\mathcal{F}_{\mathsf{W}}) \models \mathit{append}(p_1, p_2, p_3)$ genau dann wahr, wenn $p_2 = p_n$. Nach Definition 5.14 auf Seite 38 ist aber $\text{append}(\varepsilon, p_2) = p_2$, d.h. $p_n = \text{append}(p_1, p_2)$ gilt ebenfalls genau dann, wenn $p_2 = p_n$. Daraus folgt, daß die Behauptung (7.11) für $p_1 = \varepsilon$ gilt.

Induktionsschritt Angenommen die Behauptung ist eine Länge n von p_1 erfüllt. Für $p_1 = (h; t)$ mit der Länge $n + 1$ gilt nun: Aus der Induktionsvoraussetzung folgt nun für alle t_n daß

$$(\mathcal{F}_{\mathsf{W}}) \models \mathit{append}(t, p_2, t_n) \iff t_n = \text{append}(t, p_2).$$

Nach Formel 7.4 gilt nun $\mathit{append}(p_1, p_2, p_n)$ genau dann, wenn ein T_n existiert, so daß $p_n = (h; T_n)$ und $T_n = \text{append}(t, p_2)$, d.h. genau dann, wenn $p_n = (h; \text{append}(t, p_2))$. Damit ist gezeigt, daß die Behauptung (7.11) gilt, wenn p_1 die Länge $n + 1$ hat.

Damit folgt über vollständige Induktion die Behauptung. ∎

Lemma 7.12. *Es gilt:*

$$\forall h \in \mathcal{N}_{\mathcal{P}}(\mathcal{O}), c \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}), b \in \mathfrak{P}$$
$$(\mathcal{F}_{\mathsf{W}}) \models \mathit{pexpand}(h, c, b) \iff$$
$$\exists(\tilde{h}, \tilde{c}, \tilde{b}) \in \mathcal{P}, \sigma \quad h = \tilde{h}\sigma \ \wedge \ c = \tilde{c}\sigma \ \wedge \ b = \tilde{b}\sigma$$

Beweis. Aus Formel 7.5 folgt, daß $\mathit{pexpand}(h, c, b)$ genau dann gilt, wenn es ein Tripel $(\tilde{h}, \tilde{c}, \tilde{b}) \in \mathcal{P}$ gibt, so daß $\exists\big(\text{var}(\tilde{h}) \cup \text{var}(\tilde{c}) \cup \text{var}(\tilde{b})\big) \ (h = \tilde{h} \ \wedge \ c = \tilde{c} \ \wedge \ b = \tilde{b})\big)$ gilt, d.h. genau dann, wenn es eine Belegung der Variablen in $\tilde{h}$, $\tilde{b}$ und $\tilde{c}$ gibt, so daß dann $(h = \tilde{h} \ \wedge \ c = \tilde{c} \ \wedge \ b = \tilde{b})$. Dies ist äquivalent dazu, daß eine Substitution σ existiert, die $H = \tilde{h}\sigma \ \wedge \ c = \tilde{c}\sigma \ \wedge \ b = \tilde{b}\sigma$ erfüllt. Daraus folgt die Behauptung. ∎

Lemma 7.13.

$$\forall s_1, s_2 \in \mathcal{S}, p_1, p_2 \in \mathfrak{P}$$
$$(\mathcal{F}_{\mathsf{W}}) \models \mathit{world}({s_1}^{\tau}, p_1, {s_2}^{\tau}, p_2) \iff (s_1, p_1) \rightsquigarrow (s_2, p_2)$$

Beweis. Da die Weltrelation gemäß Definition 5.12 auf Seite 37 die kleinste Relation ist, die 5.13a - 5.13d erfüllt, ist ein Quadrupel $((s_1,p_1),(s_2,p_2))$ genau dann in der Weltrelation (d.h. $(s_1,p_1) \rightsquigarrow (s_2,p_2)$), wenn einer dieser 4 Fälle erfüllt ist. Anders formuliert: Für alle $s_1,s_2 \in \mathcal{S}, p_1,p_2 \in \mathfrak{P}$ gilt: $(s_1,p_1) \rightsquigarrow (s_2,p_2)$ gilt genau dann, wenn

$$\Big(\exists a \in \mathcal{N}_{\mathcal{A}}(\mathcal{O}), p \in \mathfrak{P} \;\; p_1 = (a;p) \;\wedge\; p_2 = p \;\wedge\; (s_1,a,s_2) \in \mathcal{C}_p\Big) \;\vee$$

$$\Big(\exists c \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}), p, p_a, p_b \in \mathfrak{P} \;\; p_1 = (\mathrm{if}(c,p_a,p_b);p) \;\wedge\; p_2 = \mathrm{append}(p_a,p) \;\wedge$$
$$(c,s_1) \in \mathcal{H} \;\wedge\; s_1 = s_2\Big) \;\vee$$

$$\Big(\exists c \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}), p, p_a, p_b \in \mathfrak{P} \;\; p_1 = (\mathrm{if}(c,p_a,p_b);p) \;\wedge\; p_2 = \mathrm{append}(p_b,p) \;\wedge$$
$$(c,s_1) \notin \mathcal{H} \;\wedge\; s_1 = s_2\Big) \;\vee$$

$$\Big(\exists p \in \mathfrak{P}, h, \sigma \;\; p_1 = (h\sigma;p) \;\wedge\; (h,c,b) \in \mathcal{P} \;\wedge\; h\sigma \in \mathcal{N}_{\mathcal{P}}(\mathcal{O}) \;\wedge\; c\sigma \in \mathcal{N}_{\mathcal{C}}(\mathcal{O}) \;\wedge$$
$$b\sigma \in \mathfrak{P} \;\wedge\; (c\sigma,s) \in \mathcal{H} \;\wedge\; p_2 = \mathrm{append}(b\sigma,p) \;\wedge\; s_1 = s_2\Big)$$

Nach 7.10, 7.12 und Definition 7.1 auf Seite 52 ist der folgende Ausdruck dazu äquivalent.

$$\begin{aligned}
&(\mathcal{F}_{\mathsf{W}}) \models \\
&\Big(\exists h,t \;\; p_1 = (h;t) \;\wedge\; p_2 = t \;\wedge\; causes(s_1,h,s_2)\Big) \;\vee \\
&\quad\Big(\exists h,t,p_a,p_b \;\; p_1 = (\mathrm{if}(h,p_a,p_b);t) \;\wedge\; append(p_a,t,p_2) \\
&\qquad\qquad \wedge\; holds(h,s_1) \;\wedge\; s_1 = s_2\Big) \;\vee \\
&\quad\Big(\exists h,t,p_a,p_b \;\; p_1 = (\mathrm{if}(h,p_a,p_b);t) \;\wedge\; append(p_b,t,p_2) \;\wedge \\
&\qquad\qquad \neg\, holds(h,s_1) \;\wedge\; s_1 = s_2\Big) \;\vee \\
&\quad\Big(\exists h,t,c,b \;\; p_1 = (h;t) \;\wedge\; pexpand(h,c,b) \;\wedge\; holds(c,s_1) \;\wedge \\
&\qquad\qquad append(b,t,p_2) \;\wedge\; s_1 = s_2\Big) \\
&\Big).
\end{aligned}$$

Mit Formel 7.6 folgt daraus die Behauptung. ∎

Satz 7.14. *Gegeben sei eine Welt* W*, eine zugehörige Domänenmodellierung* $((Dom),^{\tau})$ *und die logische Weltmodellierung* $(\mathcal{F}_{\mathsf{W}})$ *gemäß Definition 7.9 auf Seite 55. Dann gilt:*

$$\forall s_0 \in \mathcal{S}, s_n \in (\mathcal{S} \cup \{fail\}, p_0 \in \mathfrak{P}, n \in \mathbb{N}$$
$$(\mathcal{F}_{\mathsf{W}}) \models doplan({s_0}^{\tau}, p, s^n(0), s_n) \iff s_0 \xrightarrow[n]{p} s_n \quad (7.15)$$

Dabei ist $s^n(0)$ *eine Termdarstellung von* $n \in \mathbb{N}$ *(Gleichung* (7.7) *auf Seite 54).*

Beweis. Der Beweis erfolgt über vollständige Induktion über n.

Induktionsanfang $n = 0$: Wegen Lemma 7.13 auf Seite 56 und Definition 7.1 auf Seite 52 sind für $n = 0$ folgende Behauptungen äquivalent

$$
\begin{aligned}
(\mathcal{F}_{\mathsf{W}}) \models\ & \\
& \big(p_0 = \varepsilon \ \wedge\ {s_0}^{\tau} = {s_n}^{\tau}\big) \ \vee \\
& \big(p_0 \neq \varepsilon \ \wedge\ n = 0 \ \wedge\ {s_n}^{\tau} = \mathit{fail}\big) \ \vee \\
& \exists n_1 \ \big(p \neq \varepsilon \ \wedge\ n = s(n_1) \ \wedge \\
& \qquad \exists s_1, p_1 \ \big(\mathit{world}({s_0}^{\tau}, p_0, s_1, p_1) \ \wedge\ \mathit{doplan}(s_1, p_1, n_1, {s_n}^{\tau})\big) \ \vee \\
& \qquad \forall s_1, p_1 \ \big(\neg\, \mathit{world}({s_0}^{\tau}, p_0, s_1, p_1) \ \wedge\ {s_n}^{\tau} = \mathit{fail}\big) \\
& \big) \qquad \text{(i)}
\end{aligned}
$$

und

$$
\begin{aligned}
& (p_0 = \varepsilon \ \wedge\ s_0 = s_n) \ \vee \\
& (p_0 \neq \varepsilon \ \wedge\ n = 0 \ \wedge\ s_n = \mathit{fail}) \ \vee \\
& \big(\exists s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \ \ p_0 \neq \varepsilon \ \wedge\ n > 0 \ \wedge\ (s_0, p_0) \rightsquigarrow (s_1, p_1) \ \wedge\ s_1 \xrightarrow[n-1]{p_1} s_n\big) \ \vee \\
& \big(\forall s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \ \ p_0 \neq \varepsilon \ \wedge\ n > 0 \ \wedge\ (s_0, p_0) \not\rightsquigarrow (s_1, p_1) \ \wedge\ s_n = \mathit{fail}\big). \\
& \qquad \text{(ii)}
\end{aligned}
$$

Nun ist $(\mathcal{F}_{\mathsf{W}}) \models \mathit{doplan}({s_0}^{\tau}, p_0, s^n(0), {s_n}^{\tau})$ nach 7.8 äquivalent zu (i), und $s_0 \xrightarrow[n]{p_0} s_n$ nach 6.11 äquivalent zu (ii). Also gilt die Behauptung (7.15) für $n = 0$.

Induktionsschritt Angenommen es gilt

$$
(\mathcal{F}_{\mathsf{W}}) \models \mathit{doplan}({s_1}^{\tau}, p, s^{n-1}(0), x) \quad \Longleftrightarrow \quad s_1 \xrightarrow[n-1]{p} s_2. \qquad \text{(iii)}
$$

Zusammen mit Lemma 7.13 und Definition 7.1 folgt nun ebenfalls, daß folgende Behauptungen äquivalent sind:

$$
\begin{aligned}
(\mathcal{F}_{\mathsf{W}}) \models\ & \\
& \big(p_0 = \varepsilon \ \wedge\ {s_0}^{\tau} = {s_n}^{\tau}\big) \ \vee \\
& \big(p_0 \neq \varepsilon \ \wedge\ n = 0 \ \wedge\ {s_n}^{\tau} = \mathit{fail}\big) \ \vee \\
& \exists n_1 \ \big(p \neq \varepsilon \ \wedge\ n = s(n_1) \ \wedge \\
& \qquad \exists s_1, p_1 \ \big(\mathit{world}({s_0}^{\tau}, p_0, s_1, p_1) \ \wedge\ \mathit{doplan}(s_1, p_1, n_1, {s_n}^{\tau})\big) \ \vee \\
& \qquad \forall s_1, p_1 \ \big(\neg\, \mathit{world}({s_0}^{\tau}, p_0, s_1, p_1) \ \wedge\ {s_n}^{\tau} = \mathit{fail}\big) \\
& \big) \qquad \text{(iv)}
\end{aligned}
$$

und

$$\begin{aligned}
&(p_0 = \varepsilon \ \wedge \ s_0 = s_n) \ \vee \\
&(p_0 \neq \varepsilon \ \wedge \ n = 0 \ \wedge \ s_n = fail) \ \vee \\
&\big(\exists s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \ \ p_0 \neq \varepsilon \ \wedge \ n > 0 \ \wedge \ (s_0, p_0) \rightsquigarrow (s_1, p_1) \ \wedge \ s_1 \xrightarrow[n-1]{p_1} s_n\big) \ \vee \\
&\big(\forall s_1 \in \mathcal{S}, p_1 \in \mathfrak{P} \ \ p_0 \neq \varepsilon \ \wedge \ n > 0 \ \wedge \ (s_0, p_0) \not\rightsquigarrow (s_1, p_1) \ \wedge \ s_n = fail\big).
\end{aligned} \tag{v}$$

Nun ist $(\mathcal{F}_{\mathsf{W}}) \models doplan({s_0}^\tau, p_0, s^n(0), {s_n}^\tau)$ nach 7.8 äquivalent zu (i), und $s_0 \xrightarrow[n]{p_0} s_n$ nach 6.11 äquivalent zu (ii). Also folgt daraus, daß die Behauptung (7.15) für $n-1$ gilt, daß die Behauptung (7.15) auch für n gilt.

Mit Hilfe von vollständiger Induktion folgt daher, daß (7.15) für alle $n \in \mathbb{N}$ gilt. ∎

Satz 7.16. *Wenn für logische Weltmodellierung $(\mathcal{F}_{\mathsf{W}})$ zu einer Welt $\mathsf{W} = (\mathsf{D}, \mathcal{V}, \mathcal{N}_{\mathcal{P}}, \mathcal{P})$, einen Anfangsumweltzustand s_0, eine natürliche Zahl n und ein Anfangsvorhaben p_0 gilt:*

$$(\mathcal{F}_{\mathsf{W}}) \models \forall s_n \ \ doplan({s_0}^\tau, {p_0}^\tau, s^n(0), s_n) \ \rightarrow \ s_n \neq fail \ \wedge \ P(s_n), \tag{7.17}$$

dann der Plan $(p_0, \mathcal{P})$ ausführbar, terminierend und bezüglich der Eigenschaft P korrekt.

Beweis. Angenommen $s' \in \mathcal{S}$ ist ein Zustand, für den

$$s_0 \xrightarrow[n]{p_0} {s'}^\tau \tag{i}$$

gilt. Dann folgt aus Satz 7.14

$$(\mathcal{F}_{\mathsf{W}}) \models doplan({s_0}^\tau, p_0, s^n(0), {s'}^\tau). \tag{ii}$$

Wegen (7.17) folgt daraus

$$(\mathcal{F}_{\mathsf{W}}) \models {s'}^\tau \neq fail \ \wedge \ P({s'}^\tau). \tag{iii}$$

Wegen der Vollständigkeit von (AC1*), das in $(\mathcal{F}_{\mathsf{W}})$ enthalten ist, ist $(\mathcal{F}_{\mathsf{W}}) \models {s'}^\tau \neq fail$ äquivalent zu $s' \neq fail$. Damit folgt aus (iii)

$$s' \neq fail \ \wedge \ P(s'). \tag{iv}$$

Damit ist also gezeigt, daß für jeden Zustand s', der (i) erfüllt, (iv) gilt. Nach Satz 6.7 auf Seite 44 folgt also, daß der Plan $(p_0, \mathcal{P})$ bezüglich des Anfangszustands s_0 ausführbar, terminierend und bezüglich der Eigenschaft P korrekt. ∎

Satz 7.14 zeigt nun, daß eine auf dem Fluent-Kalkül basierende logische Weltmodellierung nach Definition 7.9 auf Seite 55 die Planausführungsrelation korrekt abbildet. Die logische Weltmodellierung bildet damit eine Semantik für die in Kapitel 5 definierte Plansprache. Wegen Satz 7.16 können also Eigenschaften wie Termination, Ausführbarkeit und Korrektheit aus der Sichtweise des skeptischen Agenten nachgewiesen werden.

8. Ein Anwendungsbeispiel

In diesem Kapitel wird als Anwendungsbeispiel eine vereinfachte Version des Omelettebeispiels aus [Lev96] beschrieben und nachgewiesen, daß sich mit der Plansprache $\mathfrak{P}$ ein angemessener Plan aufstellen läßt, dessen Ausführbarkeit, Termination und Korrektheit sich mit einer logischen Weltmodellierung nachweisen läßt.

Das Omelettebeispiel, so wie es in [Lev96] beschrieben wird, lautet in etwas vereinfachter Form wie folgt:[1]

Beispiel 8.1 (Omelettebeispiel). *Ein Koch hat einen Vorrat an Eiern, von denen einige schlecht sind, und einige gut. Es sei aber mindestens ein Ei gut. Der Koch hat weiterhin eine leere Untertasse. Er kann den Inhalt eines Eis in die Untertasse geben, an der Untertasse riechen (um festzustellen, ob das Ei schlecht ist) und den Inhalt der Untertasse wegschütten. Sein Ziel ist es nun, in der Schüssel genau ein gutes Ei (und kein weiteres) zu haben.*

Dieses Problem ist aus zwei Gründen nicht mit einem linearen Plan, der sämtliche auszuführenden Aktionen aufzählt, zu lösen. Erstens kann der Koch nicht voraussehen, ob das Ei, das er zerbricht, gut oder schlecht ist. Der Koch muß also auf das Resultat seiner Aktionen reagieren. Zweitens ist die Länge der Planausführung nicht vorauszusehen: wenn der Koch n Eier hat, so ist es möglich, daß erst das $n-1$. Ei, das er zerbricht, ein gutes ist.

Um dieses Problem nun mit der in dieser Arbeit beschriebenen Plansprache zu lösen, wird zunächst die entsprechende Planungsdomäne D formal gemäß Definition 5.1 auf Seite 29 beschrieben.

Beispiel 8.2 (Omelettebeispiel, Fortsetzung). *Die Planungsdomäne* $\mathsf{D} = (\mathcal{O}, \mathcal{N}_\mathcal{A}, \mathcal{N}_\mathcal{C}, \mathcal{S}, \mathcal{C}_p, \mathcal{H})$ *für das Omelettebeispiel ist wie folgt:*

[1] Der Kürze halber wurde die Menge von Aktionen gegenüber [Lev96] auf die zur Lösung des Problems notwendigen Aktionen eingeschränkt. Im Original hat der Koch zudem eine Untertasse und eine Schüssel, und sein Ziel ist es, 3 gute Eier in der Schüssel zu haben.

$\mathcal{O}$: *In dieser Domäne werden keine Parameter für die Aktionen und Bedingungen benötigt, daher ist die Objektmenge leer.*

$\mathcal{S}$: *Es können die folgenden Fluenten identifiziert werden:*

e_g : *Ein gutes Ei ist im Vorrat. („*good egg*“).*

e_b : *Ein schlechtes Ei ist im Vorrat. („*bad egg*“).*

s_g : *Ein gutes Ei befindet sich in der Untertasse. („*saucer*“).*

s_b : *Ein schlechtes Ei befindet sich in der Untertasse.*

Dabei handelt es sich um Ressourcen, da jeweils mehrere Eier im Vorrat oder der Untertasse sein können. Ein Zustand wird daher als Multimenge dargestellt, wobei eine Ressource mit dem Wert k als k-faches Element der Multimenge dargestellt wird.

$$\mathcal{S} = \mathbb{N}^{\{e_g, e_b, s_g, s_b\}}$$

$\mathcal{N}_\mathcal{A}$: *Die Aktionen, die dem Koch zur Verfügung stehen, sind:*

break : *Der Inhalt eines Eis wird in die Untertasse gegeben.*

throw : *Der Inhalt der Untertasse wird weggeschüttet.*

Diese Aktionen sind parameterlos (d.h. 0-stellig), daher gilt:

$$\mathcal{N}_\mathcal{A} = \{(break, 0), (throw, 0)\}.$$

Daraus ergibt sich

$$\mathcal{N}_\mathcal{A}(\mathcal{O}) = \{\text{break}, throw\}.$$

$\mathcal{C}_p$: *Die Aktionen beim Omeletteproblem lassen sich analog den Aktionen in Kapitel 3 Abschnitt 3.3 durch ihre Vorbedingungen und Effekte beschreiben: zum Beispiel entfernt* break *ein gutes bzw. schlechtes Ei aus dem Vorrat (dies ist die Vorbedingung $\{\!\{ e_g \}\!\}$ bzw. $\{\!\{ e_b \}\!\}$), und hat als Resultat, daß ein gutes bzw. schlechtes Ei in der Untertasse ist (dies ist der Effekt $\{\!\{ s_g \}\!\}$ bzw. $\{\!\{ s_b \}\!\}$). Die Menge aller Tripel (Aktionsbeschreibungen) (c, a, e) von Vorbedingung, Aktionsname und Effekt, wird im Folgenden mit $\mathcal{A}$ bezeichnet. Für das Omelettproblem ist $\mathcal{A}$ nun:*[2]

$$\begin{aligned}\mathcal{A} = \big\{&(\{\!\{ e_g \}\!\}, \text{break}, \{\!\{ s_g \}\!\}), && (\{\!\{ e_b \}\!\}, \text{break}, \{\!\{ s_b \}\!\}),\\ &(\{\!\{ s_g \}\!\}, throw, \{\!\{\}\!\}), && (\{\!\{ s_b \}\!\}, throw, \{\!\{\}\!\})\big\}. \end{aligned} \tag{8.3}$$

[2] Die hier gewählte Modellierung ist an dieser Stelle etwas vereinfacht: sie ist nur unter der Bedingung dem Ausgangsbeispiel entsprechend, daß sich in der Untertasse nicht mehr als ein Ei befindet. Dies wird aber während der später beschriebenen Planausführung eingehalten.

Die eigentliche Zustandsübergangsrelation wird nun wie folgt modelliert:

$$\mathcal{C}_p = \{\!\!\{ (x \dot\cup c, a, x \dot\cup e \mid (c, a, e) \in \mathcal{A} \ \wedge \ x \in \mathcal{S} \}\!\!\}$$

Dies entspricht den Gleichungen (3.12) *und* (3.13) *auf Seite 16 in Multimengenschreibweise.*[3]

$\mathcal{N}_\mathcal{C}$ **:** *Das Riechen des Kochs an der Untertasse wird in einem Weltmodell gemäß Kapitel 5 nicht als Aktion aufgefasst, da sie die Umwelt nicht verändert, sondern als Bedingung, die der Koch prüfen kann, dargestellt.*

Die Domanenmodellierung enthält damit die 0-stellige Bedingung badegg *, sowie die stets erfüllte 0-stellige Bedingung* $\mathbf{1}$ *, die zur Prozedurdefinition benötigt wird (siehe unten). Entsprechend gelten:*

$$\mathcal{N}_\mathcal{C} = \{(badegg, 0), (\mathbf{1}, 0)\},$$
$$\mathcal{N}_\mathcal{C}(\mathcal{O}) = \{badegg, \mathbf{1}\}.$$

$\mathcal{H}$ **:** *Die Bedeutung der Bedingungen wird nun durch die Wahrheitsrelation* $\mathcal{H}$ *festgelegt. Die Bedingung* badegg *ist dann erfüllt, wenn ein schlechtes Ei in der Untertasse ist, d.h.* $\{\!\!\{ s_b \}\!\!\}$ *im Zustand enthalten ist.* $\mathbf{1}$ *ist stets erfüllt. Damit ist die Wahrheitsrelation wie folgt festgelegt:*

$$\mathcal{H} = \left\{(badegg, s) \mid s \in \mathcal{S} \ \wedge \ \{\!\!\{ s_b \}\!\!\} \dot\subseteq s\right\} \cup \{(\mathbf{1}, s) \mid s \in \mathcal{S}\} \qquad (8.4)$$

Der Plan des Agenten (d.h. des Kochs) könnte es nun sein, ein Ei zu zerbrechen und in die Untertasse zu geben (*break*), daran zu riechen, und wenn es schlecht ist (Bedingung *badegg*) den Inhalt der Untertasse wegzuschütten (*throw*) und von vorn zu beginnen. Sollte das Ei gut sein, so ist er fertig. Mit den Hilfsmitteln der Plansprache $\mathfrak{P}$ wird dieser Plan mit Hilfe der folgenden Prozedur formuliert:

$$eggtosaucer \underset{\mathbf{1}}{\Longleftarrow} break; if(badegg, \ \ throw; eggtosaucer; \varepsilon, \ \ \varepsilon); \varepsilon \qquad (8.5)$$

Es wird also die Prozedur *eggtosaucer* definiert, die die Schleife im obigen Plan durch einen rekursiven Aufruf ersetzt: nach Ausführung von *break* wird die Bedingung *badegg* getestet, und wenn sie erfüllt ist, nach *throw* wieder *eggtosaucer* aufgerufen.[4] Das Anfangsvorhaben des Agenten ist *eggtosaucer* $; \varepsilon$, d.h. es beinhaltet lediglich einen Prozeduraufruf.

Um nun mit Hilfe der Planausführungsrelation die Eigenschaften dieses Plans untersuchen zu können, muß die Domäne D mit den definierten Prozeduren zu einer Welt im Sinne der Definition 5.11 auf Seite 37 zusammengesetzt werden:

[3] Hier wird ein einfaches Modell von Nichtdeterminismus verwendet: wenn in einem Zustand die Vorbedingungen verschiedener Aktionsbeschreibungen zutreffen (wie dies bei der Aktion *break* meist der Fall ist), so kann die Aktion verschiedene Auswirkungen gemäß den verschiedenen Aktionsbeschreibungen haben.

[4] Der Mechanismus der nichtdeterministischen Auswahl wird hier nicht benötigt, daher ist die Auswahlbedingung die stets erfüllte Bedingung $\mathbf{1}$.

Beispiel 8.6 (Omelettebeispiel, Fortsetzung). *Die Welt* $\mathsf{W} = (\mathsf{D}, \mathcal{V}, \mathcal{N}_\mathcal{P}, \mathcal{P})$ *für das Omelettebeispiel sei wie folgt:*

D**:** *ist die Domäne* D *nach Beispiel 8.2,*

$\mathcal{V}$**:** *ist* $\emptyset$ *, da hier keine Variablen benötigt werden,*

$\mathcal{N}_\mathcal{P}$**:** *ist* $\{eggtosaucer\}$ *, da nur diese Prozedur benötigt wird,*

$\mathcal{P}$**:** *enthält die Prozedurdefinition für* $eggtosaucer$ *:*

$$\mathcal{P} = \big\{\big(eggtosaucer, \mathbf{1}, break; if(badegg, throw; eggtosaucer; \varepsilon, \varepsilon); \varepsilon\big)\big\}.$$

Im Folgenden wird nun bewiesen, daß dieser Plan für die in Betracht kommenden Anfangszustände korrekt ist. Der Beweis erfolgt hier mit Hilfe der Berechnungsfolgen Planausführungsrelation, er könnte aber wegen Satz 7.14 auch mit Hilfe des entsprechenden logischen Programms erfolgen. Zur Abkürzung wird folgende Eigenschaft definiert, die charakterisiert, daß ein Weltzustand aus einem Anfangszustand gemäß Beispiel 8.1 und dem Anfangsvorhaben $eggtosaucer; \varepsilon$ besteht:

Definition 8.7. *Ein Weltzustand* $(s, p) \in \mathcal{S} \times \mathfrak{P}$ *ist ein Anfangszustand für das Omelettproblem, wenn in* s *die Untertasse leer ist, wenigstens ein gutes Ei vorhanden ist, und* p *das Anfangsvorhaben ist, d.h.*

$$\{\!|\, e_g \,|\!\} \dot{\subseteq} s \quad \wedge \quad \{\!|\, s_b \,|\!\} \dot{\not\subseteq} s \quad \wedge \quad \{\!|\, s_g \,|\!\} \dot{\not\subseteq} s \quad \wedge \quad p = eggtosaucer; \varepsilon. \tag{8.8}$$

Die Anforderungen an den Umweltzustand s eines Anfangszustands (s, p) entsprechen der in 8.1 genannten Anforderungen für den Startzustand; diese wurden nur durch die Forderung $p = eggtosaucer; \varepsilon$ ergänzt.

Zuerst werden nun zwei Lemmata über die Eigenschaften aller möglichen Berechnungsfolgen in dieser Welt bewiesen, und daraus werden dann die Eigenschaften der Planausführungsrelation bestimmt.

Lemma 8.9. *Bei einer Berechnungsfolge* $\big((s_i, p_i) \mid 0 \le i \le l\big)$ *, bei der* (s_0, p_0) *ein Anfangszustand ist, ist jeder vierte Zustand ein Anfangszustand. Die Anzahl der Eier in diesen Anfangszuständen sinkt jeweils um eins.*

Beweis. Der Induktionsanfang ist trivial, da (s_0, p_0) nach Voraussetzung Anfangszustand ist. Der Induktionsschritt wird gezeigt wie folgt: Angenommen (s_k, p_k) ist ein Anfangszustand. Zu zeigen ist nun, daß (s_{k+4}, p_{k+4}) ebenfalls ein Anfangszustand ist, sofern $k + 4 \le l$. Wegen (8.8) ist also

$$p_k = eggtosaucer; \varepsilon.$$

Wegen (5.13d) auf Seite 37 und (8.5) ist also

$$p_{k+1} = break; if(badegg,\ throw; eggtosaucer; \varepsilon,\ \varepsilon); \varepsilon \quad \wedge \quad s_{k+1} = s_k. \tag{i}$$

Nach (5.13a) und (8.3) sowie (8.2) lassen sich nun zwei Fälle unterscheiden:

1. Es gilt:

$$p_{k+2} = if(badegg,\ \ throw; eggtosaucer; \varepsilon,\ \ \varepsilon); \varepsilon \quad \wedge$$
$$s_{k+1} = x \,\dot\cup\, \{\!\!\{\ e_g\ \}\!\!\} \quad \wedge \quad s_{k+2} = x \,\dot\cup\, \{\!\!\{\ s_g\ \}\!\!\}.$$

   für ein $x \in \mathcal{S}$.

   Mit (5.13c) und (8.4) ergibt sich dann

$$p_{k+3} = \varepsilon \quad \wedge \quad s_{k+3} = s_{k+2}$$

   Da $p_{k+3} = \varepsilon$ ist, kann die Berechnungsfolge nicht fortgesetzt werden, der Beweis ist in diesem Falle also beendet.

2. Es gilt:

$$p_{k+2} = if(badegg,\ \ throw; eggtosaucer; \varepsilon,\ \ \varepsilon); \varepsilon \quad \wedge$$
$$s_{k+1} = x \,\dot\cup\, \{\!\!\{\ e_b\ \}\!\!\} \quad \wedge \quad s_{k+2} = x \,\dot\cup\, \{\!\!\{\ s_b\ \}\!\!\}. \qquad \text{(ii)}$$

   für ein $x \in \mathcal{S}$.

   Mit (5.13b) und (8.4) ergibt sich dann

$$p_{k+3} = throw; eggtosaucer; \varepsilon \quad \wedge \quad s_{k+3} = s_{k+2} \qquad \text{(iii)}$$

   und daraus mit (5.13a) und (8.3) sowie (8.2)

$$p_{k+4} = eggtosaucer; \varepsilon \quad \wedge$$
$$s_{k+3} = s_{k+4} \,\dot\cup\, \{\!\!\{\ s_b\ \}\!\!\}. \qquad \text{(iv)}$$

   Aus (i),(ii),(iii) und (iv) ergibt sich damit:

$$p_{k+4} = eggtosaucer; \varepsilon \quad \wedge$$
$$s_k = s_{k+4} \,\dot\cup\, \{\!\!\{\ e_b\ \}\!\!\}. \qquad \text{(v)}$$

   Da für (s_k, p_k) (8.8) erfüllt ist, ist (8.8) also auch für (s_{k+4}, p_{k+4}) erfüllt. Zudem enthält s_{k+4} ein Ei weniger als s_k.

Damit ist der Induktionsschritt bewiesen, und via Induktionsschluß folgt die Behauptung. ■

Da die also (endliche) Anzahl der Eier nach 4 Planausführungsschritten um ein Ei abnimmt, folgt hieraus, daß die Länge einer Berechnungsfolge begrenzt ist. Dies ist eine wichtige Eigenschaft für den Nachweis der Termination. Das folgende Lemma beschäftigt sich nun mit terminierenden und fehlschlagenden Berechnungsfolgen (d.h. Berechnungsfolgen, für die keine Fortsetzung existiert, die aber nicht terminieren.)

Lemma 8.10. *Eine Berechnungsfolge* $\big((s_i, p_i) \mid 0 \leq i \leq l\big)$, *bei der* (s_0, p_0) *ein Anfangszustand ist und für die keine Fortsetzung existiert, terminiert (d.h.* $p_k = \varepsilon$*), und es gilt* $\{\!|\, s_g \,|\!\} \dot{\subseteq} s_k$.

Beweis. Für eine Berechnungsfolge der Länge l mit $4k \leq l < 4(k+1)$ ist nach Lemma 8.9 (p_{4k}, s_{4k}) ein Anfangszustand. Ich kann mich also O.b.d.A. auf Berechnungsfolgen der Länge $0 \leq l < 4$ beschränken.

Wegen (8.8) ist also

$$p_0 = \mathit{eggtosaucer}; \varepsilon.$$

Wegen (5.13d) auf Seite 37 und (8.5) ist die Berechnungsfolge $\big((s_0, p_0)\big)$ fortsetzbar durch

$$p_1 = \mathit{break}; \mathit{if}(\mathit{badegg},\ \ \mathit{throw}; \mathit{eggtosaucer}; \varepsilon,\ \ \varepsilon); \varepsilon \quad \wedge \quad s_1 = s_0. \tag{i}$$

Nach (5.13a) und (8.3) sowie (8.2) ist die Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq 1\big)$ ebenfalls fortsetzbar, und es lassen sich nun zwei Fälle unterscheiden:

1. Es gilt:

   $$p_2 = \mathit{if}(\mathit{badegg},\ \ \mathit{throw}; \mathit{eggtosaucer}; \varepsilon,\ \ \varepsilon); \varepsilon \quad \wedge \tag{ii}$$
   $$s_1 = x \,\dot{\cup}\, \{\!|\, e_g \,|\!\} \quad \wedge \quad s_2 = x \,\dot{\cup}\, \{\!|\, s_g \,|\!\}. \tag{iii}$$

   für ein $x \in \mathcal{S}$.

   Mit (5.13c) und (8.4) läßt sich diese Berechnungsfolge weiter fortsetzen durch:

   $$p_3 = \varepsilon \quad \wedge \quad s_3 = s_2 \tag{iv}$$

   Da $p_3 = \varepsilon$ ist, kann die Berechnungsfolge $\big((s_i, p_i) \mid 0 \leq i \leq 3\big)$ nicht fortgesetzt werden. Dies ist also eine Berechnungsfolge, die die Voraussetzungen des Lemmas erfüllt, sie erfüllt aber auch die Behauptung: sie ist terminiert, und wegen (ii) und (iv) ist $s_3 = x \,\dot{\cup}\, \{\!|\, s_g \,|\!\}$, d.h. $\{\!|\, s_g \,|\!\} \dot{\subseteq} s_3$.

2. Es gilt:

   $$p_2 = \mathit{if}(\mathit{badegg},\ \ \mathit{throw}; \mathit{eggtosaucer}; \varepsilon,\ \ \varepsilon); \varepsilon \quad \wedge$$
   $$s_1 = x \,\dot{\cup}\, \{\!|\, e_b \,|\!\} \quad \wedge \quad s_2 = x \,\dot{\cup}\, \{\!|\, s_b \,|\!\}.$$

   für ein $x \in \mathcal{S}$.

   Mit (5.13b) und (8.4) läßt sich dies durch

   $$p_3 = \mathit{throw}; \mathit{eggtosaucer}; \varepsilon \quad \wedge \quad s_3 = s_2$$

   fortsetzen und weiter mit (5.13a) und (8.3) sowie (8.2)

   $$p_4 = \mathit{eggtosaucer}; \varepsilon \quad \wedge$$
   $$s_3 = s_4 \,\dot{\cup}\, \{\!|\, s_b \,|\!\}.$$

   Damit ist aber $k \geq 4$, d.h. dieser Fall braucht nicht betrachtet zu werden.

■

Aus diesen beiden Lemmata läßt sich nun mit Hilfe von Definition 6.6 auf Seite 44 die Planausführungsrelation ermitteln. Damit lassen sich die Planeigenschaften bestimmen:

Proposition 8.11. *Sei* W *die Welt nach Beispiel 8.6. Für einen Anfangszustand* (s_0, p_0) *ist das Vorhaben* $eggtosaucer; eggtosaucer; eggtosaucer; \varepsilon$ *ausführbar, terminierend und bezüglich der Bedingung*

$$P(s) \iff \{\!|\, s_g \,|\!\} \dot{\subseteq} s \ \wedge\ \{\!|\, s_b \,|\!\} \dot{\not\subseteq} s \tag{8.12}$$

korrekt.

Beweis. Aus Lemma 8.9 folgt, daß jede Berechnungsfolge, die von einem Anfangszustand (s_0, p_0) mit m Eiern ausgeht, eine Länge von weniger als $4(m+1)$ hat, da sonst die Anzahl der Eier 0 unterschreiten müßte. Ich wähle nun ein $n > 4(m+1)$, so daß es keine Berechnungsfolge länger als n gibt.

Angenommen, es gilt $s_0 \xrightarrow[n]{p_0} s'$. In Definition 6.6 können nun weder Fall 2 zutreffen, da die Länge einer Berechnugsfolge nicht größer als n sein kann, noch Fall 3 (wegen Lemma 8.10). Es trifft also Fall 1 zu, d.h. man findet eine Berechnungsfolge $\big((s_i, p_i) \mid 0 \le i \le k\big)$ mit $k \le n$. Nach Lemma 8.10 ist nun Bedingung 8.12 erfüllt.

Damit wurde also bewiesen, daß für das gewählte n gilt:

$$\forall s' \ \ s_0 \xrightarrow[n]{p_0} s' \ \ \rightarrow \ \ s \neq fail \ \wedge\ P(s'),$$

d.h. nach Satz 6.7 auf Seite 44 ist der Plan ausführbar, terminierend und bezüglich $P(s)$ korrekt. ■

Der Plan erfüllt also die Ansprüche eines skeptischen Agenten. Damit ist auch gezeigt, daß die Plansprache $\mathfrak{P}$ ausdrucksmächtiger als die linearen oder partiell geordneten Pläne ist, die bisher in den auf dem Fluent-Kalkül basierenden Arbeiten verwendet wurden.

9. Zusammenfassung und Ausblick

Im Gegensatz zu den bisherigen Arbeiten zum Planen mit Hilfe des Fluent-Kalküls, die sich mit Aktionsbeschreibung und Generation von Plänen für einen gegebenen Anfangszustand beschäftigten, schafft diese Arbeit die Möglichkeit zur Formulierung von Plänen, die für eine ganze Klasse von Anfangszuständen funktionieren. Dazu wird ein Plan nicht mehr als eine Sequenz[1] von primitiven Aktionen dargestellt (bzw. als ein partiell geordneter Plan, der einer Menge derartiger Sequenzen entspricht), sondern in einer komplexeren Plansprache formuliert. Diese enthält Ausdrucksmittel für bedingte Ausführung von Teilplänen, für Auswahl von Parametern von Aktionen während der Planausführung (gemäß dem aktuellen Zustand der Umgebung des Agenten), und für Prozeduraufrufe, mit denen sich mehrfach benötigte Sequenzen von Aktionen zusammenfassen lassen sowie Rekursion (und damit auch Iteration) darstellen läßt.

Dabei wurde von einem Modell für ein dynamisches System (als „Welt“ bezeichnet) ausgegangen, das einen Agenten und seine Umwelt umfaßt (Kapitel 5). Der Agent wird dabei als ein planausführendes System verstanden: der Plan stellt dabei das Vorhaben (Intention) des Agenten in Form eines Satzes in einer Plansprache $\mathfrak{P}$ dar, welcher während der Ausführung des Planes verändert wird, um den Ausführungsstand des Planes und die Entscheidungen des Agenten bei bedingten Teilplänen und bei der Auswahl von Aktionsparametern zu reflektieren. Die Veränderungen der Umwelt und des Vorhabens des Agenten wurde durch eine als „Weltrelation“ bezeichnete Zustandsübergangsfunktion charakterisiert. In Anlehnung an die Automatentheorie wird eine Folge von Arbeitsschritten des Agenten als „Berechnungsfolge“ bezeichnet.

Die Frage, ob ein Plan für einen Agenten akzeptabel ist, erweitert sich nun in diesem Kontext um einen neuen Aspekt (Kapitel 6). Bei bei linearen und partiell geordneten Plänen stehen die Fragen, ob der Plan ausführbar ist und ob nach Ausführung das Ziel des Planes erreicht wird (Korrektheit) im Mittelpunkt. Bei Plänen, die Rekursion enthalten, kommt die nun Frage hinzu, ob die Ausführung des Planes endet (Termination), oder ob der Agent eine unendliche Folge von Schritten durchlaufen kann. In dieser Arbeit wurde von einem „skeptischen“ Agenten ausgegangen, d.h. der Agent verlangt, daß der Plan stets ausführbar

[1] auch total geordneter Plan genannt

ist, daß das Ziel des Planes erreicht wird, wenn der Plan ausgeführt wurde, und daß der Plan stets terminiert. Diese Anforderungen könnten von einem „mutigen“ Agenten abgeschwächt werden, wie in Abschnitt 6.1 diskutiert wird.

Für die Beurteilung der genannten Planeigenschaften sind die Weltzustände[2] interessant, die vom Anfangszustand erreicht werden können. Diese werden von der transitiven Hülle der Weltrelation beschrieben. Da die transitive Hülle einer Relation aber innerhalb von Logik erster Stufe nicht endlich axiomatisierbar ist, wird in dieser Arbeit statt dessen von der n-ten Approximation der transitiven Hülle ausgegangen[3]. Diese wird um einen speziellen Zustand $fail$ zur Planausführungsrelation (Abschnitt 6.2 auf Seite 43), mit deren Hilfe sich die Planeigenschaften, die der skeptische Agent fordert, sehr kompakt darstellen lassen. Die Eigenschaft der Termination wird dabei zur starken Termination verschärft, die verlangt, daß man für jeden Anfangszustand *vor* der Planausführung eine Grenze für die Schrittzahl, die die Ausführung benötigt, angeben kann.[4] Die Schrittzahl ist also nur von Plan und Anfangszustand der Umwebung abhängig. Diese Eigenschaft ist im Gegensatz zur einfachen Termination auch ohne die Verwendung von Induktion nachweisbar.[5]

Diese Planausführungsrelation wurde basierend auf dem Fluent-Kalkül in Prädikatenlogik erster Stufe abgebildet, und die Korrektheit dieser Abbildung nachgewiesen (Kapitel 7, Satz 7.14 auf Seite 57). Satz 7.16 auf Seite 59 zeigt, daß sich die Anforderungen eines skeptischen Agenten an Ausführbarkeit, Termination und Korrektheit eines Plans mit Hilfe logischer Folgerung nachweisen lassen. Damit existiert eine wohldefinierte Semantik der Plansprache $\mathfrak{P}$ und es ist nun möglich, Theorembeweiser zum Nachweis dieser Eigenschaften einzusetzen.

Als Abschluß wurde in Kapitel 8 für eine einfache Version des Omelettebeispiels [Lev96] ein Plan aufgestellt, und Ausführbarkeit, Termination sowie Korrektheit für diesen Plan nachgewiesen. Damit wurde demonstriert, daß die Ausdruckskraft der Plansprache $\mathfrak{P}$ die Ausdruckskraft von linearen Plänen (im Sinne einer Sequenz von primitiven Aktionen) übersteigt.

In Zukunft könnte diese Arbeit in mehrere Richtungen erweitert werden.

Erstens ist die Implementation auf einem existierenden Theorembeweiser erforderlich um praktische Versuche anstellen zu können. Wie in der Einleitung angedeutet, könnte bei einigen Planungsproblemen die Anwendung von Induktion den Beweis der Korrektheit vereinfachen, so daß der verwendete Theo-

[2] d.h Zustände der Umwelt *und* des Agenten

[3] Die n-te Approximation der transitiven Hülle einer binären Relation M ist definiert als $\bigcup_{i=0}^{n} M^i$.

[4] Die starke Termination unterscheidet sich nur für nichtdeterministische Planausführung von der einfachen Termination.

[5] Die Nicht-Termination ist allerdings im allgemeinen nicht ohne Induktion nachweisbar, da dies eine Betrachtung aller $n \in \mathbb{N}$ für die Planausführungsrelation erfordert.

rembeweiser dies unterstützen sollte, wie z.B. INKA [Wal94]. Da die $(AC1*)$-Gleichungstheorie in allgemeiner Form recht unhandlich ist, und unendlich viele Axiome enthalten kann [HT95], aber für die auftretenden AC1-Unifikationsprobleme effiziente Unifikationsalgorithmen existieren [Thi92], erscheint die Integration dieser Unifikationsalgorithmen in den Beweiser erforderlich. Eine Verwendung des SLDENF-Verfahrens [Mic94], daß für andere Arbeiten im Fluent-Kalkül verwendet wurde, erscheint jedoch nicht wünschenswert, da es menschliche Schlußweisen wie Induktion nicht nachbilden kann.

Zweitens wird im vorliegenden Ansatz aus Effizienzgründen der Plan nur teilweise als Term dargestellt, und ein anderer Teil (die Menge von Prozedurdefinitionen) als ein Axiom zum logischen Modellierung hinzugefügt. Damit ist eine Plangeneration durch Beweis eines Theorems „Es existiert ein Plan, der zum Ziel führt“ nicht zur Erzeugung von Prozedurdefinitionen nutzbar. Dafür sind also andere Methoden erforderlich.

Drittens berücksichtigt das Weltmodell nur Veränderungen der Umwelt durch den Agenten. In einer realen Umwelt können jedoch Veränderungen stattfinden, die nicht vom Agenten selbst ausgelöst werden, wie z.B. das Fortschreiten der Zeit, oder etwa durch andere Agenten ausgelöste Veraenderungen.

Und viertens könnte eine Betrachtung des Konzepts eines „mutigen Agenten“ erfolgen. In der Realität ist ein Plan, der immer zum Ziel führt, kaum denkbar. Die Modellierung könnte also dahingehend erweitert werden, daß die Ursachen für mögliche Fehlschläge des Plans erkannt und bewertet werden können.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie Zitate kenntlich gemacht habe.

Dresden, den 24.März 1997

Hans-Peter Störr

Literaturverzeichnis

[AHT90] Allen, J., J. Hendler und Austin Tate (Herausgeber): *Readings in planning.* The Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[Bac86] Backhouse, Roland Carl: *Program construction and verification.* Prentice Hall International (UK) Ltd, 1986.

[BHS93a] Bibel, Wolfgang, Steffen Hölldobler und Torsten Schaub: *Wissensrepräsentation und Inferenz: eine grundlegende Einführung.* Vieweg, Wiesbaden, 1993. ISBN 3-528-05374-7.

[BHS+93b] Brüning, S., S. Hölldobler, J. Schneeberger, U. Sigmund und M. Thielscher: *Disjunction in Resource–Oriented Deductive Planning.* In: Miller, D. (Herausgeber): *Proceedings of the International Logic Programming Symposium*, Seite 670, 1993.

[Bib86] Bibel, W.: *A Deductive Solution for Plan Generation.* New Generation Computing, 4:115–132, 1986.

[BT96] Bornscheuer, Sven-Erik und Michael Thielscher: *Representing Concurrent Actions and Solving Conflicts.* Journal of the Interest Group in Pure and Applied Logics (IGPL). Special issue of the ESPRIT project MEDLAR, 4(3):355–368, 1996.

[BT97] Bornscheuer, S.-E. und M. Thielscher: *Explicit and Implicit Indeterminism.* Journal of Logic Programming, Special Issue 'Action and Change', 1997. (to appear spring 97).

[Ede95] Eder, Kerstin: *A resource oriented deductive approach towards hierarchical planning.* Diplomarbeit, TU Dresden, 1995.

[EHT96] Eder, Kerstin, Steffen Hölldobler und Michael Thielscher: *An Abstract Machine for Reasoning about Situations, Actions, and Causality.* In: *Proceedings of the International Workshop on Extensions of Logic Programming*, Lecture Notes in Artificial Intelligence 1050, Seiten 137–151. Springer, 1996.

[FN71] Fikes, Richard E. und Nils J. Nilsson: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence, 2:189–208, 1971.

[Fra92] Francez, Nissim: *Program verification.* Addison-Wesley, 1992.

[Gab94] Gabbay, D. M.: *Classical vs non-classical logics (the universality of classical logic).* In: Gabbay, DOV M., C. J. Hogger und J. A. Robinson (Herausgeber): *Handbook of Logic in Artificial Intelligence and Logic Programming*, Band 2 - Deduction Methodologies. Clarendon Press, Oxford, 1994.

[Geo87] Georgeff, Michael P.: *Planning.* Ann. Rev. Comput. Sci., 2:359–400, 1987. Ebenfalls abgedruckt in [AHT90].

[GHS96] Grosse, Gerd, Steffen Hölldobler und Josef Schneeberger: *Linear Deductive Planning.* Journal of Logic and Computation, 6(2):233–262, 1996.

[Har79] Harel, D.: *First Order Dynamic Logic*, Band 68 der Reihe *LNCS*. Springer, 1979.

[Höl92] Hölldobler, S.: *On Deductive Planning and the Frame Problem.* In: Voronkov, A. (Herausgeber): *Proceedings of the Conference on Logic Programming and Automated Reasoning*, Seiten 13–29. Springer, LNCS, 1992.

[HRS90] Heisel, M., W. Reif und W. Stephan: *Tactical Theorem Proving in Program Verification.* In: *Proceedings of the 10th International Conferenve on Automated Deduction*, Band 449 der Reihe *LCNS*, Seiten 117–131. Springer, 1190.

[HRS91] Heisel, M., W. Reif und W. Stephan: *Formal Software Development in the KIV-System.* In: McCartney, R. und M. R. Lowry (Herausgeber): *Automating Software Design.* AAAI Press, 1991.

[HS89] Hölldobler, S. und J. Schneeberger: *A new Deductive Approach to Planning.* In: *Proceedings of the German Workshop on Artificial Intelligence*, Seiten 63–73, 1989.

[HS90] Hölldobler, S. und J. Schneeberger: *A New Deductive Approach to Planning.* New Generation Computing, 8:225–244, 1990. A short version appeared in the Proceedings of the German Workshop on Artificial Intelligence, Informatik Fachberichte *216*, pages 63-73, 1989.

[HT95] Hölldobler, S. und M. Thielscher: *Computing Change and Specificity with Equational Logic Programs.* Annals of Mathematics and Artificial Intelligence, 14:99–133, 1995.

[Kal94] KALINKE, YVONNE: *Ein massiv paralleles Berechnungsmodell für normale logische Programme.* Diplomarbeit, TU Dresden, 1994.

[Leh97] LEHMANN, H.: *Aktionen, Bedingungen und Ressourcen.* Diplomarbeit, TU Dresden, Fakultät Informatik, Institut KI, 1997.

[Lev96] LEVESQUE, HECTOR J.: *What is planning in the presence of sensing?* Technischer Bericht Department of Computer Science, University of Toronto, Canada, 1996. email: `hector@cs.toronto.edu`.

[Lif86] LIFSCHITZ, V.: *On the Semantics of STRIPS.* In: GEORGEFF, M. P. und A. L. LANSKY (Herausgeber): *Proceedings of the Workshop on Reasoning about Actions & Plans.* Morgan Kaufmann, 1986.

[Llo87] LLOYD, J. W.: *Foundations of Logic Programming.* Springer-Verlag, second Auflage, 1987.

[LR96] LIN, F. und R. REITER: *How to Progress a Database.* Artificial Intelligence, 1996. (to appear), avail. from http://www.cs.toronto.edu/ cogrobo/.

[LRL+97] LEVESQUE, H.J., R. REITER, Y. LESPRANCE, F. LIN und R. SCHERL: *GOLOG: A Logic Programming Language for Dynamic Domains.* Journal of Logic Programming, Special Issue 'Action and Change', 1997. (to appear spring 97), siehe http://www.cs.toronto.edu/ cogrobo.

[McC63] MCCARTHY, J.: *Situations and Actions and Causal Laws.* Stanford Artificial Intelligence Project, Memo 2, 1963.

[Mic94] MICHAEL: *Automatisiertes Schließen über Kausalbeziehungen mit SLDENF-Resolution.* Doktorarbeit, TH Darmstadt, 1994.

[MTV90] MASSERON, M., C. TOLLU und J. VAUZIELLES: *Generating Plans in Linear Logic.* In: *Foundations of Software Technology and Theoretical Computer Science*, Seiten 63–75. Springer, LNCS *472*, 1990.

[MW87] MANNA, Z. und R. WALDINGER: *How to Clear a Block: A Theory of Plans.* Journal of Automated Reasoning, ???:???, ??? 1987.

[MW91] MANNA, ZOHAR und RICHARD WALDINGER: *Fundamentals of Deductive Program Synthesis.* In: BAUER, F. L. (Herausgeber): *Logic, Algebra and Computation*, Band 79 der Reihe *NATO Advanced Science Institutes Series, subseries F: Computer and System Sciences*, Seiten 41–107. Springer Verlag, 1991. http://theory.stanford.edu/people/zm/papers/pap-fundamentals92.ps.Z, ebenfalls in [MW92].

[MW92] Manna und Richard Waldinger: *Fundamentals of Deductive Program Synthesis.* In: *IEEE Transactions on Software Engineering*, Band 18 No. 8, Seiten 674–704. August 1992.

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

[San94] Sandewall, Erik: *Features and Fluents - The Representation of Knowledge about Dynamical Systems*, Band 1 der Reihe *Oxford Logic Guides 30.* Oxford Science Publications, 1994.

[SB93] Stephan, Werner und Susanne Biundo: *A New Logical Framework for Deductive Planning.* In: *Proceedings of the International Joint Conference on Artificial Intelligence*, Seite 32. 1993.

[SB95] Stephan, Werner und Susanne Biundo: *Deduction-based Refinement Planning.* Technischer BerichtRR-95-13, Germen Research Center for Artificial Intelligence (DFKI), {stephan,biundo}ddfki.uni-sb.de, 1995.

[SH96] Steffen Hölldobler, Josef Schneeberger: *Constraint Equational Logic Programming and Resource-Based Partial Order Planning.* Technischer BerichtWV-96-08, TU Dresden, Institut KI, Fachgebiet Wissensverarbeitung, 1996.

[Thi92] Thielscher, Michael: *AC1-Unifikation in der linearen logischen Programmierung.* Diplomarbeit, Technische Hochschule Darmstadt, Fachgebiet Intellektik, Fachbereich Informatik, aug 1992.

[Thi96] Thielscher, Michael: *Qualification and Causality.* Technischer BerichtTR-96-026, International Computer Science Institute (ICSI), Berkeley, CA, Juli 1996.

[Thi97] Thielscher, Michael: *Ramification and Causality.* Artificial Intelligence Journal, 1997. (To appear. A preliminary version is available as Technical Report TR-96-003, ICSI, Berkeley, CA).

[Wal94] Walther, Christoph: *Mathematical Induction.* In: Gabbay, Dov M., C. J. Hogger und J. A. Robinson (Herausgeber): *Handbook of Logic in Artificial Intelligence and Logic Programming*, Band 2. Clarendon Press, Oxford, 1994.

Index