Diplomarbeit

Realisierung eines nichtlinearen Planungssystems zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM)

Frank Weberskirch

Juni 1994

Betreuung: Prof. Dr. Michael M. Richter
Dipl. Inform. Jürgen Paulokat

Arbeitsgruppe Künstliche Intelligenz
— Expertensysteme —
Prof. Dr. Michael M. Richter

Universität Kaiserslautern
Fachbereich Informatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Planung ist ein vielfach untersuchtes Gebiet im Bereich der Künstlichen Intelligenz. Die hier vorgestellte Arbeit ist in diesem Gebiet angesiedelt: es geht um ein Planungssystem, welches auf die Unterstützung der Arbeitsplanerstellung in der computerintegrierten Fertigung abzielt. Der Bereich der computerintegrierten Fertigung ist allerdings nur als ein spezieller Anwendungsbereich für das System zu sehen.

1.1 Planung

Zur Einordnung der Arbeit soll zunächst auf den Begriff *Planung* eingegangen werden. Bei der Planung handelt es sich im Gegensatz zur Diagnose, die hauptsächlich Analyseaufgaben betrachtet, vornehmlich um die Lösung von Syntheseproblemen[1]. Die Lösung einer Aufgabe soll mit Hilfe fest vorgegebener Elementarbausteine erstellt, synthetisiert, werden und dabei gewisse Kriterien und Wertvorstellungen erfüllen. Diese allgemeine Sichtweise läßt sich noch etwas verfeinern. Man unterscheidet zwischen dem Bereich der *Konfiguration* und dem der *Aktionsplanung*. Bei der Konfiguration geht es darum, die Frage zu beantworten, aus welchen primitiven Komponenten ein komplexes Objekt zusammengesetzt werden muß, um bestimmte Anforderungen zu erfüllen, z.B. um die Konfiguration eines Fahrrades aus primitiven Komponenten wie Rahmen, Lenker, Räder, usw. Bei der Aktionsplanung besteht das zu planende Objekt aus einer Reihe von Aktionen, die eine Anfangssituation in eine Zielsituation überführen, z.B. die Planung von Roboteraktionen, wo es darum geht, durch welche primitiven Aktionen und welche Reihenfolge der primitiven Aktionen eine bestimmte Aufgabe erreicht werden kann.

Im Rahmen dieser Arbeit geht es um Planung im Sinne von Aktionsplanung. Daher soll im folgenden immer davon ausgegangen werden, daß mit dem Begriff der Planung hier speziell die Aktionsplanung gemeint ist.

1.2 Zielsetzungen der Arbeit

Diese Arbeit beschreibt die Realisierung des Planungssystems **CAPlan**[2]. Die bei der Realisierung verfolgten Zielsetzungen waren:

- Es sollte eine Architektur zur domänenunabhängigen, nichtlinearen Planung geschaffen werden. Als grundlegendes Verfahren dafür wurde ein Ansatz zur *systematischen, nichtlinearen Planung (SNLP)* [MR91] von David McAllester und David Rosenblitt realisiert und an einigen entscheidenden Stellen noch verbessert.

[1]Eine ganz strenge Trennung zwischen Analyse- und Syntheseaufgaben ist meist nicht möglich, da die Synthese in einem gewissen Ausmaß Situationen analysieren muß und umgekehrt. Bei Planung und Diagnose liegen aber die Schwerpunkte der Aufgabenstellung in den Bereichen der Synthese bzw. der Analyse.

[2]CAPlan: Computer Aided Plannning

- Das CAPlan-System sollte eine interaktive Vorgehensweise in Sinne eines *Intelligenten Plannungsassistenten* möglichst weitgehend unterstützen, dem Benutzer also ggf. alle Entscheidungen überlassen können und nur noch zur Verwaltung von Plänen und Abhängigkeiten zwischen getroffenen Entscheidungen dienen. Dies wurde durch die Entwicklung mehrerer, verschiedener Kontrollkomponenten zur Steuerung der Planung realisiert, die in beliebiger Reihefolge und Verzahnung Planungsschritte durchführen können.

- Zwei Kontrollkomponenten standen im Vordergrund, zum einen eine, die den SNLP-Ansatz in der erweiterten Version realisiert. Eine zweite sollte vollständig interaktiv arbeiten.

- Die Realisierung weiterer Kontrollkomponenten außer der sytematischen und der interaktiven sollte leicht möglich sein, z.B. die Integration einer fallbasiert arbeitenden Komponente, die die Planung mittels Fallbeispielen steuert (**CAPlan/CbC**[3]) oder einer Komponente, die dazu Kontrollregeln verwendet. Das ist jedoch nicht Gegenstand dieser Arbeit.

- Der hohe Grad an zugelassener Benutzerinteraktion machte es zwingend notwendig, Abhängigkeiten zwischen Planungsentscheidungen und ihren Auswirkungen explizit zu repräsentieren. Da ein für derartige Aufgabenstellungen von Charles Petrie entwickeltes und als Implementation ihm Rahmen des Konfigurationssystems IDAX [Rit92] entstandenes System, REDUX [Pet91, Pet92], bereits existierte, konnte bei der Realisierung darauf aufgesetzt werden.

Das System wurde in der Programmiersprache Smalltalk-80, Release 4.1 entwickelt. Das Verständnis der Implementation setzt daher gründliche Kenntnisse dieser Sprache voraus. Empfohlene Literatur dafür sind z.B. [GR83] und besonders ihm Hinblick auf die Implementierung des REDUX-System die entsprechenden Kapitel aus [Rit92].

1.3 Übersicht über die Arbeit

- Kapitel 2 stellt einige Grundlagen aus dem Bereich der Aktionsplanung vor, die für die gesamte Arbeit von Bedeutung sind.

- In Kapitel 3 werden die Konzepte der systematisch arbeitenden Kontrollkomponente des CAPlan-Systems vorgestellt, welche auf dem Ansatz zur systematischen, nichtlinearen Planung (SNLP) [MR91] beruhen.

- Da das REDUX-System als Grundlage für die Realisierung der Konzepte von SNLP dient, stellt Kapitel 4 den von Charles Petrie mit REDUX [Pet91, Pet92] verfolgten Ansatz zur Planung und Planmodifikation vor.

- Kapitel 5 beschreibt dann ausführlich, wie das Planungssystem CAPlan aufbauend auf das REDUX-System realisiert wurde.

- Kapitel 6 fäßt nochmals zusammen, was mit dieser Arbeit erreicht wurde, stellt Vergleiche mit anderen Ansätzen an und erläutert einige im Laufe der Implementierung von CAPlan aufgedeckten Zusammenhänge.

[3] CAPlan/CbC: CAPlan Case-based Control

Kapitel 2

Grundlagen

Dieses Kapitel soll einen Überblick über einige im folgenden wichtige Grundlagen zu verschiedenen Fragestellungen der Planung verschaffen. Insbesondere wird auf die Begriffe *Aktionsplanung*, *STRIPS-Planung* und *lineare/nichtlineare Planung* eingegangen.

2.1 Aktionsplanung

Bei der Aktionsplanung handelt es sich um die Lösung eines Transformationsproblems. Es wird von einer zustandsorientierten Sicht auf die Welt ausgegangen, bei der einzelne Situationen durch eine Menge von planungsrelevanten Merkmalen beschrieben werden. Außerdem gibt es eine fest vorgegebene Menge von Aktionen, welche eine gegebene Situation in einer bestimmtem Art und Weise verändern.

Es kann zwischen zwei Arten von Planungsverfahren unterschieden werden:

- *Domänenspezifische Verfahren:*
  Das Planungsverfahren benutzt direkt domänenspezifisches Wissen und Heuristiken der Domäne bei der Lösungssuche. Solche Verfahren sind aufgrunddessen meist nicht ohne großen Aufwand in anderen Bereichen einsetzbar.

- *Domänenunabhängige Verfahren:*
  Das Planungsverfahren selbst abstrahiert von speziellen domänenspezifischen Gegebenheiten und bietet eine allgemeine Problemlösemethode sowie Repräsentationsmechanismen für die Modellierung von Domänen.

Domänenunabhängige Verfahren sind in der Regel vielseitiger einsetzbar und sollen auch im folgenden im Vordergrund stehen. Der von solchen Verfahren zur Verfügung gestellte Repräsentationsmechanismus zur Modellierung von Domänen soll es dem Benutzer ermöglichen, die Domäne in einer für das System verwendbaren Weise zu definieren, andererseits ist es auch wichtig, daß für den Benutzer eine möglichst natürliche Modellierung möglich ist. Hierbei handelt es sich jedoch meist um zwei sich widersprechende Zielsetzungen für den Repräsentationsmechanismus und es ist wichtig einen guten Mittelweg zu finden.

Die Aufgabenstellung bei der Aktionsplanung bei vorgegebener Domäne ist nun die folgende: Ausgehend von einer Anfangssituation in der Planungswelt ist eine Sequenz von Aktionen gesucht, die diese Anfangssituation in eine Zielsituation transformieren, in welcher bestimmte, als Planungsziele definerte Bedingungen gelten. Diese Sequenz von Aktionen nennt man auch den Plan, der das Planungsproblem löst; er gibt Auskunft darüber, in welcher Reihenfolge welche Aktionen durchgeführt werden müssen, um die Zielsituation zu erreichen. In vielen Fällen soll die gesuchte Sequenz von Aktionen bezüglich eines definierten Kriteriums (z.B. Anzahl von Aktionen) möglichst optimal sein.

Zur Domänendefinition in einem Aktionsplanungssystem gehört sowohl eine Beschreibungssprache für Problemstellungen und Zustände der Planungswelt als auch die Menge von möglichen Aktionen in dieser Planungswelt. Konkrete Beispiele für Planungsdomänen sind:

- *Blocksworld-Domäne:*

  Die Planungswelt besteht aus Klötzchen, einem Tisch und einem oder mehreren Roboterarmen; Aktionen sind das Greifen und wieder Abstellen von Klötzchen, die Planungsaufgaben betreffen das Stapeln der Klötzchen in einer bestimmten als Ziel definierten Art und Weise.

  Dies ist das klassische Beispiel einer Domäne, die in sehr vielen Planungssystemen verwandt wird und auf den ersten Blick den Eindruck macht, eine sehr einfache Spielzeug-Domäne ohne viel Bedeutung für die Realität zu sein. Doch dieser Eindruck trügt, da viele grundlegende Probleme der Planung (z.B. die Erkennung und Auflösung von Interaktionen zwischen Aktionen) bereits dort auftauchen und untersucht werden können.

- *Transportdomäne:*

  Die Planungswelt besteht aus Orten, zu transportierenden Gütern, Transportmitteln; mögliche Aktionen sind das Be-/Entladen von Transportmitteln und das Bewegen derselben von einen Ort zu einem anderen. Ein Transportproblem besteht darin, eine Menge von Gütern zwischen verschiedenen Orten zu transportieren unter Verwendung gegebener Transportmittel. (vgl. [Vel92])

- *CNC-Arbeitsplanungsdomäne:*

  Die Planungswelt besteht aus Werkstücken, die mit einer CNC-Maschine aus einem Rohteil gefertigt werden können, z.B. der Menge der rotationssymetrischen Drehteile, und den wesentlichen Komponenten einer CNC-Maschine wie verschiedene Aufspannungen und Fertigungswerkzeuge. Ein Planungsproblem besteht darin aus einer Repräsentation des zu fertigenden Werkstücks, welche die genaue Kontur des zu fertigenden Werkstücks enthält, eine Sequenz von CNC-Bearbeitungs- und Hilfsoperatoren zu finden, die dieses Werkstück aus einem Rohteil fertigt.

Das in den folgenden Kapiteln noch ausführlicher beschriebene Planungssystem findet z.B. in der zuletzt genannten CNC-Arbeitsplanungsdomäne für rotationssymetrische Drehteile seine Hauptanwendung, ist aber als domänenunabhängiges System realisiert, womit es auch leicht für Planungsaufgaben aus anderen Domänen eingesetzt werden kann.

Für die Beispiele in den folgenden Kapiteln wird meist auf die oben erwähnte Klötzchen-Domäne zurückgegriffen werden, da dort relativ leicht und schnell geeignete Beispiele definiert und erläutert werden können.

2.2 STRIPS-Planung

Planung ist schon viele Jahre Gegenstand der Forschung. STRIPS[1] [FN71, FHN72], eines der ersten Systeme, war ursprünglich zur Planung und Steuerung eines Roboters gedacht, der Kisten durch eine Reihe von Räumen schieben sollte, die mit Türen verbundenen sind. STRIPS erreichte einen großen Bekanntheitsgrad, weil dort ein ganz charakteristisches Problem früherer Systeme gelöst wurde, welches auch unter dem Namen *Frame-Problem* bekannt ist.[2]

2.2.1 STRIPS und das Frame-Problem

Problemlöser, die auf Resolution und dem Situationskalkül basierten, hatten eine entscheidende Schwierigkeit, die unter dem Namen *Frame-Problem* bekannt ist und die sich mit folgender Frage veranschaulichen läßt: Wenn in einer Situation s eine Aktion a ausgeführt wird, welche Aussagen gelten in der daraus resultierenden Situation? Das Problem ist, daß im allgemeinen nicht ausgeschlossen werden

[1] STRIPS: STanford Research Institute Problem Solver

[2] [Geo87] gibt einen guten Überblick darüber.

kann, daß ein Operator eine bestimmte Aussage ungültig macht. Das bedeutet aber letztlich, daß für jeden Operator nicht nur angegeben werden muß, was er verändert, sondern auch alles das, was er nicht verändert.

Dies führte zu den *Frame-Axiomen*, mit deren Hilfe beschrieben wurde, welche Aussagen bei einer bestimmen Aktion unverändert bleiben. Die Schwierigkeit dabei ist jedoch, daß die Menge von notwendigen Frame-Axiomen schon bei relativ kleinen und einfachen Domänen mit nur wenigen Aktionen sehr groß wird.

Bei STRIPS wurden dieses Problem durch eine Einschränkung im Hinblick auf die Operatordefinitionen gelöst: man geht weg von einer vollständigen Beschreibung der Auswirkungen einer Operatoranwendung zu einer partiellen Beschreibung, bei der nur die sich ändernden Aspekte berücksichtigt werden. Damit spart man sowohl die Notwendigkeit der Defintion von Frame-Axiomen als auch die Notwendigkeit, sie bei der Berechnung des Resultats einer Operation verwenden zu müssen. Diese Einschränkung ist auch unter dem Namen *STRIPS-Assumption* bekannt (vgl. [Wal77]) und sagt aus, daß – wie in 2.2.3 noch näher definiert wird – ein Operator alle Aussagen unverändert gültig läßt, die nicht explizit in seiner Definition vorkommen. Auch STRIPS nutzte noch einen Resolutionsbeweiser zum Testen, ob eine Vorbedingung in einer Situation gültig ist, aber dieser wurde nicht mehr verwendet, um die Resultatsituation von Operator-Anwendungen herzuleiten, wie es frühere Systeme machten, was aber zu den oben genannten Problemen führte. Besonders diese STRIPS-artigen Operatordefintionen spielen bis heute eine große Rolle und werden hier noch näher erläutert.

2.2.2 Planungsproblem und Lösung

STRIPS-Planung baut auf einer zustandsorientierten Sicht der Welt auf. Zustände (Situationen) in der Planungswelt sind Mengen von prädikatenlogischen Formeln; im einfachsten Fall beschränkt man sich auf funktionsfreie Atomformeln. Außerdem wird meist wie z.B. im System SIPE [Wil84, Wil88] eine Form der *Closed-World Assumption* verwendet, wodurch es ausreicht, zu definieren, was in der initialen Situation gelten soll.

Ein *STRIPS-Planungsproblem* ist dann definiert als ein Tripel (DOM, Σ, Ω):

- *DOM:* Domänendefinition, d.h. eine Menge von Prädikatssymbolen für die Definition von Zuständen in der Planungswelt, eine Menge von Axiomen, die Abhängigkeiten der Prädikatssymbole ausdrücken und eine Menge von Domänenoperatoren in Form von Operator-Schemata, welche Aktionen in der Planungswelt repräsentieren
- Σ*:* Menge von Formeln, die die Startsituation beschreiben
- Ω*:* Menge von Formeln, die die Zielsituation beschreiben

Eine *Lösung zu einem solchen STRIPS-Planungsproblem* (DOM, Σ, Ω) ist eine Folge von Operatoren aus der Domäne DOM derart, daß das Ergebnis der sequentiellen Anwendung dieser Operatoren auf die Startsituation Σ eine Situation ist, in der die Zielprädikate Ω gelten.

Abbildung 2.1 zeigt ein einfaches Beispiel für eine STRIPS-Planungsproblem aus der Blocksworld-Domäne sowie eine Lösung zu diesem Problem. Zur Beschreibung von Situationen sind nur funktionsfreie Atomformeln mit den Prädikatssymbolen `On` und `Clear` erlaubt. Die hier durchgängig verwendete Notation bei der Beschreibung von Aussagen der Start- und Zielsituation ist von der Form `+P`, was bedeutet, daß die Aussage `P` in dieser Situation gelten soll.
Die Lösung für das Problem ist sehr einfach: sie besteht aus einer Aktion `PutOn(A,B)`, die die Startsituation in die Zielsituation überführt.

2.2.3 STRIPS-Operatoren

Charakteristisch bei STRIPS und in der KI-Aktionsplanung vielfach wiederverwendet sind die *STRIPS-Operatoren* zur Modellierung der möglichen Aktionen in einer Planungswelt.

Ein STRIPS-Operator ist definiert durch:

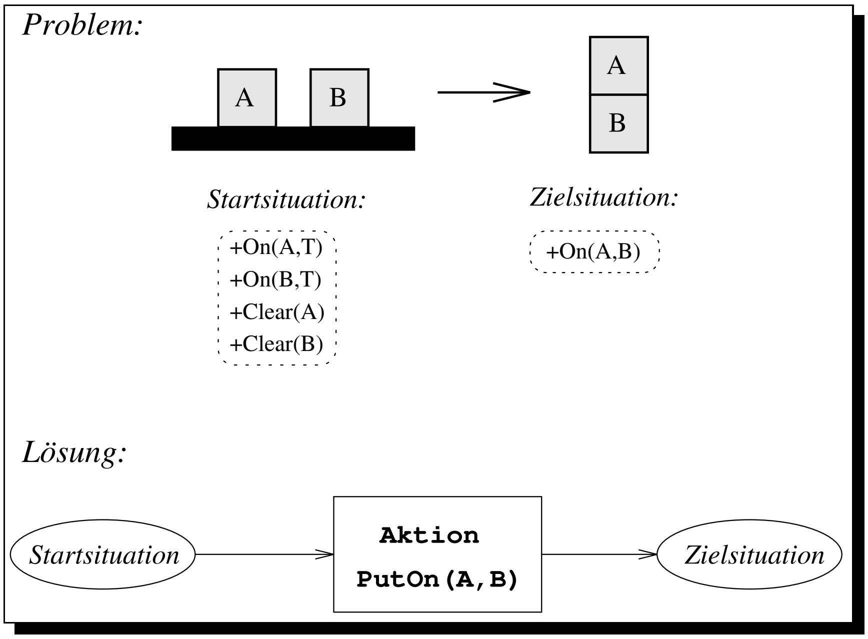

Abbildung 2.1: STRIPS-Planungsproblem aus der Blocksworld

- *Vorbedingungen:*
  Menge von Bedingungen, die erfüllt sein müssen, damit der Operator anwendbar ist
- *Add-/Delete-Liste:*
  Menge der durch den Operator bewirkten Veränderungen; sie werden zusammengefaßt als seine *Effekte* bezeichnet.

Operatoren werden in der Regel unter Verwendung von Variablen in Form von *Operator-Schemata* definiert. Damit erspart man es sich, eine sehr große Menge von konkreten Operatoren definieren zu müssen. So definiert man beispielsweise einen Operator-Schema `PutOn(x,y)` mit den Variablen `x` und `y` statt jede konkrete Ausprägung (z.B. `PutOn(A,B)` aus Abb. 2.1) einzeln zu definieren, wobei deren Anzahl exponentiell mit der Anzahl der verschiedenen Objekte steigt.
Eng mit Variablen verbunden sind Constraints, mit denen Aussagen über mögliche oder nicht mögliche Bindungen von Variablen gemacht werden. Im einfachsten Fall kann man damit auskommen, die Gleichheit und Ungleichheit von Variablenbindungen auszudrücken. Daher gehört zu einer Operatordefinition meist noch eine Menge von Constraints (*Gleichheits- und/oder Ungleichheits-Constraints*) über den in der Operatordefinition verwendeten Variablen. Sie haben folgende Bedeutung:

- `Same(x,y)` erzwingt für die Variablen `x` und `y` dieselbe Bindung
- `NotSame(x,y)` verbietet für `x` und `y` dieselbe Bindung

Abbildung 2.2 zeigt zwei Beispiele für STRIPS-Operatordefinitionen aus der Blocksworld. Die Add-/Delete-Listen sind dort als eine Liste von Effekten zusammengefaßt: `+/-` deuten an, ob es sich um Elemente der ursprünglichen Add- oder der Delete-Liste handelt.

Wilkins [Wil84] hat bezüglich der Effekte eines Operators eine Unterteilung vorgenommen: es wird zwischen *Haupteffekten* (Purposes) und *Seiteneffekten* (Side Effects) unterschieden. Dem liegt die Überlegung zugrunde, daß ein Operator in der Regel verwendet wird, um eine ganz spezifische Auswirkung hervorzurufen und daß jedoch mit einer Operatoranwendung meist noch andere, nicht explizit geforderte Auswirkungen, Seiteneffekte, verbunden sind.
Betrachtet man den `PutOn`-Operator aus Abbildung 2.2, so hat er primär den Zweck, den Effekt `+On(x,y)` hervorzurufen (dort angedeutet durch `(H)` für `Haupteffekt`). Jedoch ist z.B. `-Clear(y)`

```
PutOn-Operator
 Vorbedingungen:  +On(x,z)
                  +Clear(x)
                  +Clear(y)
 Effekte:         +On(x,y) (H)
                  +Clear(z)
                  -On(x,z)
                  -Clear(y)
 Constraints:     NotSame(x,y)
                  NotSame(x,z)
                  NotSame(y,z)
```

```
ClearTop-Operator
 Vorbedingungen:  +On(y,x)
                  +Clear(y)
                  +Clear(z)
 Effekte:         +Clear(x) (H)
                  +On(y,z)
                  -On(y,x))
                  -Clear(z)
 Constraints:     NotSame(x,y)
                  NotSame(x,z)
                  NotSame(y,z)
```

Abbildung 2.2: Beispiel für STRIPS-Operatoren

ebenfalls ein (Seiten-)Effekt dieses Operators. Bei der Ausnutzung dieser Unterteilung der Effekte im Hinblick auf die Operatorauswahl ist, wie noch erläutert werden wird, Vorsicht geboten.

Nun soll noch definiert werden, was bei der *Anwendung eines STRIPS-Operators* passiert. Die Anwendung eines STRIPS-Operators auf einen gegebenen Zustand in der Planungswelt, in dem die Vorbedingungen dieses Operators gültig sind, transformiert den Zustand in einen Nachfolgezustand, in dem folgendes gilt:

- Alle Elemente der Add-Liste sind in der Zustandsbeschreibung des Nachfolgezustandes enthalten.
- Alle Elemente der Delete-Liste sind in der Zustandsbeschreibung des Nachfolgezustandes nicht enthalten.
- Alle Aussagen, die weder in der Add- noch in der Delete-Liste vorkommen, bleiben unverändert. Dieser Punkt ist entscheidend und heißt auch die *STRIPS-Assumption*

Die Verwendung von Variablen bei der Operator-Definition steht, wie bereits erwähnt, meist in Verbindung mit der Verwendung von Constraints, was zur Folge hat, daß Operatoren in diesem Fall auch Constraints einführen.

Im Hinblick auf die Frage, welche prädikatenlogischen Formeln in den Add-/Delete-Listen und den Vorbedingungslisten zugelassen werden sollen, wurden für STRIPS in [FN71] keine Einschränkungen gemacht. Lifschitz hat jedoch gezeigt [Lif87], daß es dies falsch ist. Die Anwendung von STRIPS-Operatoren wie oben beschrieben erzeugt, wenn man beliebige prädikatenlogischen Formeln (z.B. Formeln mit Quantifizierungen) in den Situationsbeschreibungen bzw. den Add-/Delete-Listen von Operatoren zuläßt, u.U. inkonsistente Zustände, was sich dadurch äußert, daß Axiome über den Prädikatssymbolen, die zur Beschreibung von Zuständen verwendet werden, verletzt sind.

Beispiel: Gegeben sei die schon im Beispiel aus Abbildung 2.1 gezeigte Startsituation. Anders als dort sei sie hier jedoch durch folgenden drei Formeln beschrieben:

$$On(a,t), On(b,t), \forall x \in \{a,b\} : Clear(x)$$

Entscheidend ist, daß in der dritten Formel ein Quantifizierung vorkommt. Wird darauf nun der PutOn-Operator wie er in Abbildung 2.2 definiert wurde angewandt, so ergibt sich gemäß der Regel zur Anwendung von STRIPS-Operatoren folgende Menge von Formeln für die resultierende Situation:

$$On(a,b), On(b,t), Clear(t), \forall x \in \{a,b\} : Clear(x)$$

Der quantifizierte Ausdruck stand nicht in der Delete-Liste des Operators, bleibt also nach der Anwendung des Operators unverändert Bestandteil der resultierende Situationsbeschreibung.

Diese Situationsbeschreibung ist jedoch inkonsistent, da nicht gleichzeitig $On(a,b)$ und $\forall x \in \{a,b\} : Clear(x)$ gelten kann.

Ähnliche Inkonsistenzen ergeben sich, wenn die Add-/Delete-Listen quantifizierte Formeln enthalten. Als Beispiel dafür kann man sich einen Operator `PutAllBlocksOnTable` denken, dessen Effekte durch die Formel $\forall x \neq t : On(x,t)$ beschrieben sind. Wieder ergeben sich Konsistenzprobleme der Anwendung eines solchen Operators.

Lifschitz definiert nun einige Kriterien, die erfüllt sein müssen, um diese Konsistenz- bzw. Korrektheitseigenschaft nicht zu verletzen. Sie sind aber im hier betrachteten Fall unkritisch, da die hier nur funktionsfreie Atomformeln verwendet werden, wofür [Lif87] die Korrektheitseigenschaft gezeigt hat.

2.2.4 Planen als Suche

Planen kann man als ein Suchproblem verstehen: Ziel ist es, von einer Startsituation in der Planungswelt in eine Zielsituation zu gelangen unter Anwendung von Operatoren der Planungsdomäne; diese Folge von Operatoranwendungen ist gesucht. Man kann zwischen zwei Sichten auf dieses Suchproblem unterscheiden:

Suche im Raum der Zustände der Planungswelt: Man denkt sich die durch Operatoranwendung erreichbaren Übergänge zwischen Zuständen der Planungswelt als einen Graphen. Die Knoten sind Zustände in der Planungswelt; zwei Knoten sind miteinander verbunden, wenn es einen Operator gibt, der den einen Zustand in den anderen überführt. Planung ist dann die Lösung eines Graphensuche-Problems; die Suche besteht darin, einen Pfad von der Startsituation zur Zielsituation zu finden (s. Abb. 2.3).

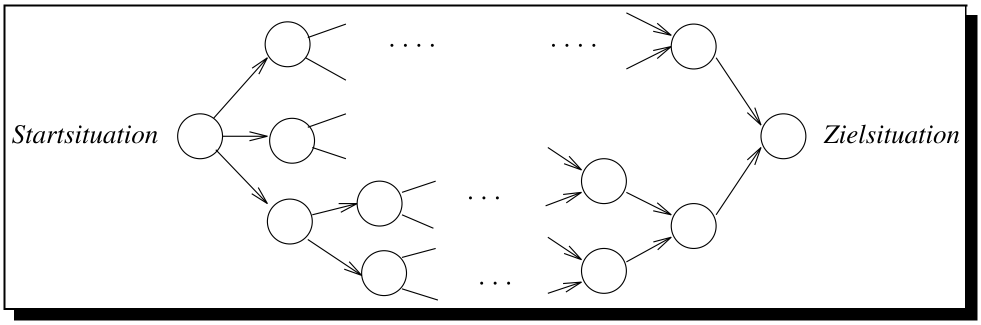

Abbildung 2.3: Suche im Raum der Zustände der Planungswelt

Suche im Raum der Planzustände: Sacerdoti's System NOAH [Sac75] ging erstmals dazu über, stattdessen im Raum der (möglicherweise noch unvollständigen) Planzustände zu suchen. Das später beschriebenen Verfahren (vgl. Kap. 3) geht ebenfalls diesen Weg.

Für die zweite Sichtweise des Planens als Suche im Raum der Planzustände muß nun genauer spezifiziert werden, was unter einem Planzustand[3] verstanden werden soll. Pläne bestehen aus einer Menge von Planschritten (Aktionen) und über diesen Aktionen ist eine meist nur partielle Ordnung definiert. Man kann einen Planzustand P beschreiben als ein Tupel (S, O, B) mit:

- S: Menge von Planschritten
- O: Menge von Ordnungsbeziehungen zwischen Planschritten, die eine partielle Ordnung auf den Planschritten S definieren
- B: Menge von Variablenbindungs-Constraints (`Same`/`NotSame`) über den in den Schritten verwendeten Variablen

[3] Die Begriffe Plan und Planzustand werden im folgenden synomym verwendet.

Außerdem muß man noch die initiale Situation Σ und die Ziele Ω eines konkreten Planungsproblems (DOM, Σ, Ω) in einen solchen Plan hineincodieren. Dies geschieht mit Hilfe zweier besonderer Planschritte s_0 und s_∞. s_0 hat die Effekte Σ und keine Vorbedingungen, s_∞ hat die Vorbedingungen Ω und soll immer hinter s_0 angeordnet sein. DOM definiert, welche Schritte zu S hinzugefügt werden können.

Bei dieser Sicht ist nun, anders als in Kapitel 2.2.2 definiert, als *Lösung zu einem Planungsproblem* ein Planzustand (S, O, B) zu finden, in dem die Vorbedingungen aller Schritte $s_i \in S$ notwendigerweise gültig sind in der s_i vorangehenden Situation.
Wie dabei konkret erreicht wird, daß jede Vorbedingung notwendigerweise gilt, ist eine in dem Zusammenhang nicht betrachtet Frage; denkbar sind z.B. Chapmans Modal Truth Criterion [Cha87] oder der in Kapitel 3 beschriebene SNLP-Ansatz [MR91].

Die *Linearisierung eines Planes P* mit $P = (S, O, B)$ ist schließlich ein Plan $P' = (S, O', B)$, der anders als P alle Planschritte total anordnet, d.h. es gilt $O \subseteq O'$ und O' ist eine totale Ordnung. Liefert ein Planungsverfahren also einen Plan als Lösung, des Ordnung über den Schritten nur partiell ist, so ist jede Linearisierung eine Lösung zum gegebenen Planungsproblem.

2.3 Nichtlineare Planung

Der Begriff der *nichtlinearen Planung* ist für die folgenden Abschnitte von Bedeutung, allerdings ist festzustellen, daß die Definition, was darunter zu verstehen sein soll, nicht ganz eindeutig ist. Der Begriff des *linearen Planens* wird in zwei Bedeutungen verwendet: (vgl. [BW93])

- Der Planer repräsentiert Pläne als total geordnete Mengen von Aktionen.
- Der Planer bearbeitet ein Teilziel komplett bevor er ein anderes zu bearbeiten beginnt[4].

Verwechselt wird hier, daß dies zwei verschiedene Aspekte sind. Beim ersten Punkt handelt es sich lediglich um ein Repräsentationsmerkmal für Pläne im Planer, welches grundsätzlich von der Frage von Bearbeitungsreihenfolgen von Teilzielen getrennt betrachtet werden sollte.

2.3.1 Partiell und total geordnete Pläne

Ein Plan besteht aus einer Menge von Planschritten (Aktionen) und einer Ordnung auf diesen Planschritten. Klar ist zunächst einmal eine Unterscheidung in Bezug auf die im Plan definierte Ordnung: (s. Abb. 2.4)

- *Total geordenete Pläne:*
  Der Plan ist also eine lineare Folge von Aktionen, d.h. die auf den Schritten definierte Ordnung ist total. Dies ist allerdings nicht zu verwechseln mit der linearen Planung.
- *Partiell geordenete Pläne:*
  Der Unterschied hier ist, daß es im Plan durchaus Schritte geben kann, die relativ zu einanderen nicht angeordnet sind, d.h. die Ordnung ist nur partiell. Wieder ist dies allerdings nicht zu verwechseln mit nichtlinearer Planung.

Ein partiell geordneter Plan kann auch als Abstraktion der Menge von total geordneten Plänen gesehen werden, die eine Linearisierung (vgl. Kap. 2.2.4) dieses partiell geordneten Planes sind.

Die Verwendung von partiell geordneten Plänen hat als Ziel, im Hinblick auf die Ordnungen möglichst wenige Festlegungen zu machen, bevor dies wirklich nötig ist. Diese Vorgehensweise ist allgemein auch als *Least-Commitment-Strategie* bekannt.

[4] Sussman's eigentliche Defintion der sog. Linearitätsannahme (*linear assumption*) [Sus75] ist noch restriktiver, indem sie annimmt, die Teilziele könnten unabhängig voneinander und in beliebiger Reihefolge gelöst werden, was aber nur von wenigen für eine durchführbare Strategie gehalten wird (vgl. [BW93])

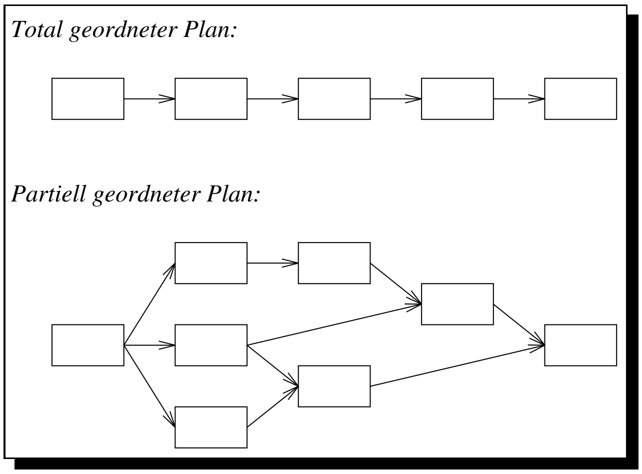

Abbildung 2.4: Ordnung auf Planschritten

2.3.2 Kategorisierung von Planungsproblemen

In [BW93] wird in Bezug auf Planungsprobleme, die aus einer Menge von zu erreichenden Teilzielen bestehen, eine Kategorisierung dieser Mengen von Teilzielen dahingehend beschrieben, wie sehr diese Teilziele miteinander interagieren. Dieser Grad an Interaktion ist ein Maß für die Schwierigkeit, das Problem zulösen. Ursprünglich stammt die Kategorisierung aus [Kor87], wurde aber in [BW93] noch etwas erweitert.

Zunächst muß dazu allerdings genauer erläutert werden, was hier genauer unter einem Teilziel zu verstehen ist. Einem Teilziel $g \in \Omega$ bzgl. eines Problems (DOM, Σ, Ω) ist eine Menge U von Planzuständen zugeordnet derart, daß für jeden Plan $(S, O, B) \in U$ gilt:

- $s_0 \in S$ und s_0 hat die Effekte Σ.
- $s_\infty \in S$ und s_∞ hat die Vorbedingungen Ω.
- s_0 bzw. s_∞ sind erster bzw. letzter Schritt in jeder Linearisierung von (S, O, B).
- P ist unmittelbar vor s_∞ notwendigerweise erfüllt.

Vereinfacht heißt das, daß man einem Teilziel G_i eine Menge U von Planzuständen zuordnet, die Lösungen für dieses Teilziel sind, da sie immer von der Startsituation ausgehen und in ihrer Zielsituation das Teilziel notwendigerweise erfüllt haben.

Geht man nun von zwei zu erreichenden Teilzielen G_1 und G_2 aus[5], so stellt sich die Frage, ob und wie sich die beiden Teillösungen kombinieren lassen. Interaktionen zwischen den beiden Teillösungen sind dabei das Hauptmerkmal, da sie u.U. bestimmte Reihenfolgen der Planschritte im Plan, der beide Teilziele erfüllt, erzwingen.

Es wird zwischen drei Gruppen unterschieden:

- *Unabhängige Teilziele:*
  Zwei Teilziele sind unabhängig voneinander, wenn jede Operatoranwendung nur zur Lösung eines der beiden Teilprobleme beiträgt und kein Schritt, der zum Teilplan des einen Teilzieles gehört, mit dem zum anderen Teilziel gehörigen Teilplan interagiert. Daraus folgt die angenehme Eingeschaft, daß jede Konkatenierung der zu den Teilzielen gehörigen optimalen Teilpläne eine optimale Gesamtlösung ist.

[5] Die Betrachtung läßt sich auch für n Teilziele machen (vgl. [Kor87] und [BW93]).

Beispiel: Sei dem Teilziel G_1 der Teilplan $s_0 \to s_1 \to s_2 \to s_\infty$[6] und G_2 der Teilplan $s_0 \to s_3 \to s_4 \to s_\infty$ zugeordnet.
Unabhängigkeit der Teilziele bedeutet, daß die beiden Teilpläne keinen gemeinsamen Schritte außer s_0 und s_∞ enthalten und kein Schritt des einen Teilplanes mit irgendeinem Schritt des andere Teilplanes interagiert. Der Gesamtplan kann die Schritte also bis auf die Restriktion s_1 vor s_2 und s_3 vor s_4 beliebig anordnen. Abbildung 2.5 (a) zeigt dies in Form eines partiell geordneten Plans, d.h. jede Linearisierung dieses Plans ist eine Lösung für ein Planungsproblem bestehend aus beiden Teilzielen.

- *Serialisierbare Teilziele:*

  Hier muß es mindenstens eine Anordnung der beiden Teilziele geben, bei der die Abarbeitung des einen Teilplans nicht wieder das vorherige Teilziel zerstört, d.h. bei mindestens einer Anordnung gibt es keine Interaktionen. An dieser Stelle wird in [BW93] nochmals eine feinere Unterscheidung gemacht in

  – *trivial serialisierbare Teilziele* und

  – *aufwendig serialiserbare Teilziele*

  gemacht, die noch etwas genauer betrachtet, ob die Reihenfolge der Abarbeitung der Teilziele bei der Lösungssuche eine Rolle spielt. Bei trivialer Serialisierbarkeit kann ohne Backtracking eine Lösung gefunden werden, bei aufwendiger Serialisierbarkeit kann eine ungünste Reihenfolge der Zielauswahl Backtracking erforderlich machen.

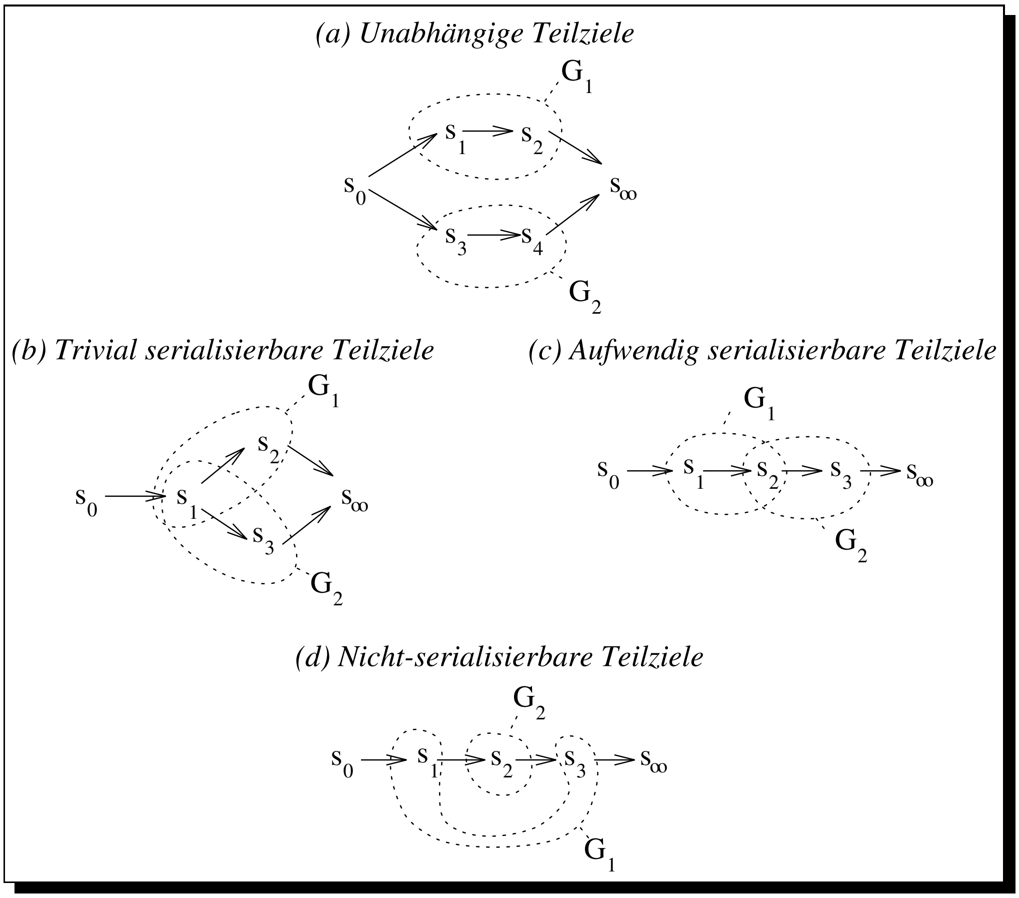

Abbildung 2.5: Beispiele für verschiedene Kategorien von Teilzielen

[6] Dies soll eine abkürzenden Schreibweise für einen Plan (S, O, B) mit $S = \{s_0, s_1, s_2, s_\infty\}$ und der durch die Pfeile angedeuteten totalen Ordnung auf S sein.

Beispiel: Trivial serialisierbar sind die beiden Teilziele G_1 mit Teilplan $s_0 \to s_1 \to s_2 \to s_\infty$ und G_2 mit Teilplan $s_0 \to s_1 \to s_3 \to s_\infty$, wenn keine Interaktionen zwischen s_2 und s_3 bestehen. Ein Gesamtplan für beide Teilziele ist hier nicht mehr einfach die Konkatenation der beiden einzelnen Teilpläne. Der Schritt s_1 kann für die Lösung beider Teilziele verwandt werden, wird aber nur einmal von dem zuerst bearbeiteten Teilziel eingeführt und von dem anderen ebenfalls verwendet (phantomisiert). Abbildung 2.5 (b) zeigt dies wieder mittels eines partiell geordneten Plans. Trivial serialisierbare Teilziele unterscheiden sich von unabhängigen Teilzielen also genau dadurch, daß Schritte für beide Teilziele verwandt werden können, daß die Reihenfolge der Abarbeitung aber unkritisch ist.

Beispiel: Bei aufwendig serialisierbaren Teilzielen ist die Reihenfolge nicht unkritisch: die Teilziele müssen in einer bestimmten Weise angeordnet werden, um eine korrekte Lösung zu liefern. Abbildung 2.5 (c) zeigt hierfür ein Beispiel. Aufgrund von Interaktionen muß hier der Teilplan von G_1 vor dem Teilplan von G_2 angeordnet sein.

- *Nicht-serialisierbare Teilziele:*

  In diesem Fall gibt es keine Anordnung der beiden Ziele, die beide Teilziele noch korrekt löst. Die beiden Teilpläne müssen ineinandern verzahnt werden, um noch eine korrekte Lösung zu liefern.

Beispiel: Abbildung 2.5 (d) zeigt als Beispiel für nicht-serialisierbare Teilziele einen Fall, in welchem der Teilplan zur Lösung von G_1 geteilt werden muß, d.h. s_1 muß vor, s_3 muß nach s_2 angeordnet werden.
Dies ist übrigens auch die charakteristische Schwierigkeit der vielzitierten *Sussman-Anomalie* aus der Blocksworld; Abbildung 2.6 zeigt die Problemstellung und die zugehörige (einzige) Lösung, wenn man von den beiden Operator-Schemata `PutOn` und `ClearTop`, wie sie in Abbildung 2.2 aus Kapitel 2.2.3 definiert wurden, ausgeht. Die beiden Teilpläne zur Erreichung der Teilziele müssen ineinander verzahnt werden, um beide Teilziele erfüllen zu können.

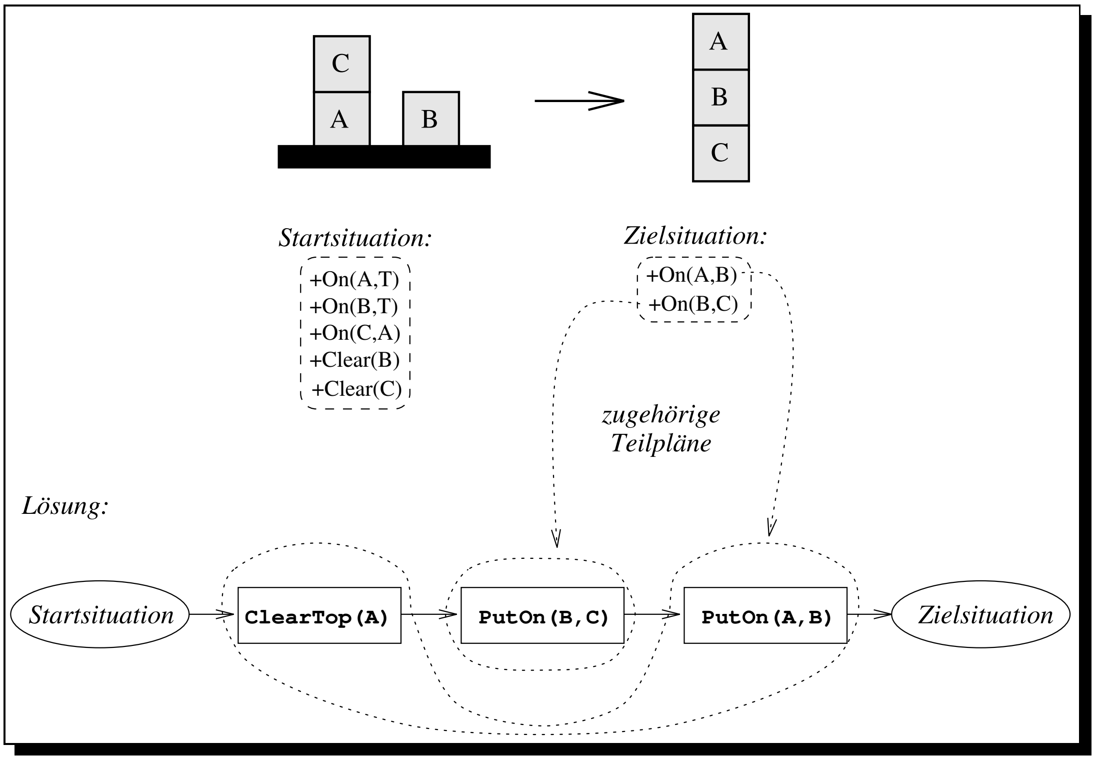

Abbildung 2.6: Die Sussman-Anomalie

Interessant ist bei diesem Beispiel noch die folgende Eigenschaft: Ändert man die Problemdefinition derart, daß die Zielsituation neben den beiden Zielen `+On(A,B)` und `+On(B,C)` auch das Ziel `+On(C,T)` enthält, so wird aus der Sussman-Anomalie als Menge von nicht-serialisierbaren Teilzielen eine (aufwendig) serialisierbare Menge, von Teilzielen. D.h. dadurch, daß die Zielsituation ausführlicher beschrieben wird, wird das Problem für das Planungssystem insgesamt leichter lösbar.

In [Kor87] und [BW93] werden noch eine ganze Reihe von Beziehungen zwischen verschiedenen Mengen von Teilzielen gemacht und bewiesen, wie die im letzten Beispiel beschriebene Beobachtung, wenn man die Ziele der Sussman-Anomalie erweitert oder daß die Teilmengen von unabhängigen Teilzielen wieder Mengen von unabhänigen Teilzielen sind. Auf diese soll hier jedoch nur verwiesen werden.

Insgesamt ergibt sich aus dieser Einteilung der Menge von Teilzielen die in Abbildung 2.7 gezeigte Hierarchie von Mengen von Teilzielen.

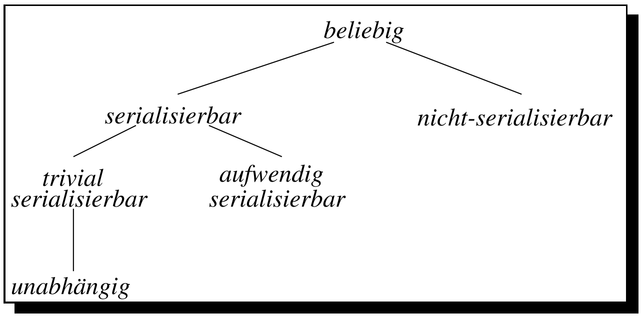

Abbildung 2.7: Hierarchie von Teilzielarten

2.3.3 Präzisierung des Begriffs der nichtlinearen Planung

Mittels dieser Kategorisierung von Mengen von Teilzielen läßt sich der Unterschied zwischen *linearem* und *nichtlinearem* Planen, wie er hier verstanden werden soll, besser erläutern:

> Nichtlineares Planen zeichnet sich gegenüber dem linearen Planen dadurch aus, daß es imstande ist, auch für nicht-serialisierbare Mengen von Teilzielen eine Lösung zu finden.

Gerade die Frage, ob nicht-serialisierbare Mengen von Teilzielen gelöst werden können, ist ein sehr entscheidendes Kriterium. Ansätze die dies nicht können, wie z.B. SIPE von Wilkins[7], müssen sich in diesem Fall einen Ausweg darin suchen, die Teilziele, die aufgrund von Interaktionen wieder zerstört werden, mehrmals zu bearbeiten, was aber als Ergebnis hat, daß dadurch der Plan unnötig groß wird und die Termination des Verfahrens nicht mehr garantiert werden kann.

Die Zielsetzung war hier:

- Nichtlineare Planung im oben definierten Sinne, d.h. das System soll Probleme bestehend aus nicht-serialisierbare Mengen von Teilzielen lösen können und
- Verwendung partiell geordneter Pläne bei der Planrepräsentation (Least-Commitment Strategie).

Der im folgenden Kapitel beschriebene Ansatz leistet genau dies.

[7] Hier können nur komplette, parallel liegende Zweige von Split-Join-Strukturen angeordnet werden (vgl. [Wil84]), was es unmöglich macht, Teilpläne ineinander zu verzahnen.

Kapitel 3

Der SNLP-Ansatz

Eine der Grundlagen des im folgenden noch genauer vorgestellten Planungssystems CAPlan bildet ein Ansatz zur *systematischen, nichtlinearen Planung*, SNLP [MR91] von David McAllester und David Rosenblitt. Der dort vorgestellte Planungsalgorithmus ist korrekt und vollständig und besitzt außerdem die Eigenschaft, den Suchraum möglicher Pläne zu einem gegebenen Problem systematisch zu durchsuchen: jeder mögliche Planzustand wird nur höchstens einmal erzeugt.

Der Ansatz ist domänenunabhängig; eine Domänendefinition gehört wie in 2.2.2 beschrieben zu einer Problemdefinition. Dadurch werden die vom Planer verwendbaren Aktionen (Operatoren) definiert sowie die zugrundeliegende Merkmale für Zustände der Planungswelt.

In seiner Grundversion, d.h. ohne die Verwendung von Variablen, leitete das Verfahren sich von Tate's NONLIN [Tat77] ab, ist jedoch einfacher. Eine Erweiterung auf den Fall, daß auch Variablen bei Operatoranwendungen vorkommen können, wird in [BW93] ausführlicher beschrieben und bildet die eigentliche Grundlage für das Planungssystem CAPlan, welche jedoch noch verbessert wurde. Die für das Verständnis des Verfahrens wesentlichen Konzepte sollen hier erläutert werden.

3.1 Repräsentation von Plänen

Ein Plan beim SNLP-Ansatz ist grundsätzlich *partiell geordnet*, er besteht aus einer Menge von Planschritten und einer auf diesen Schritten definierten (partiellen) Ordnung. Da in den Operatordefinitionen Variablen verwendet werden (vgl. Kap. 2.2.3), gehört zu einem Plan auch eine Menge von Variablenbindungs-Constraints (`Same/NotSame`).

Zur Beschreibung von Situationen in der Planungswelt und zur Definition von Operatorschemata werden nur funktionsfreie Atomformeln zugelassen; insbesondere Quantifizierungen sind also nicht möglich.

3.1.1 Planschritte

Allen Planschritten ist eindeutig ein Operator zugeordnet, ein STRIPS-Operator wie in Kap. 2.2.3 beschrieben. Durch diesen sind jedem Planschritt eindeutig Vorbedingungen und Effekte, sowie Constraints über den eingeführten Variablen zugeordnet.

Es lassen sich zwei Arten von Planschritten unterscheiden:

- *Normale Planschritte:*
  Sie haben einen eindeutigen Namen (z.B. `STEP-1`) und repräsentieren einen Operator der Domäne, also eine Aktion in der Planungswelt. Dieser Domänenoperator definiert die Vorbedingungen, Effekte und neuen Constraints des Planschrittes.
- *Künstliche Planschritte:*

Es gibt nur die zwei künstlichen Planschritte START und FINISH. Sie dienen lediglich dazu, die Problemstellung (d.h. die initiale Situation und die Zielsituation) im Plan zu repräsentieren und entsprechen den in Kapitel 2.2.4 mit s_0 und s_∞ bezeichneten Schritten.
Dem START-Schritt ist ein (Pseudo-)Operator zugeordnet, der keine Vorbedingungen hat und die Literale der initialen Situation als Effekte hat. Entsprechend dient der FINISH-Schritt dazu, die Planungsziele zu repräsentieren. Ihm ist ein Operator zugeordnet, der exakt die Vorbedingungen hat, die die Zielsituation beschreiben. Effekte hat dieser Operator keine.

Jeder Plan enthält einen START- und einen FINISH-Schritt, die in einer Initialisierungsphase erzeugt werden, und beliebig viele normale Planschritte, die im Laufe des Problemlösevorgangs entstehen.

3.1.2 Ordnungen

Betrachtet man eine Lösung (ein linearisierter Plan) zu einem STRIPS-Planungsproblem, so muß für einen korrekten Plan gelten, daß die Vorbedingungen aller vorkommenden Planschritte zur Ausführungszeit notwendigerweise erfüllt sind.

Es gibt verschiedene Strategien, dies für einen Plan sukzessive sicherzustellen, z.B. das von David Chapman entwickelte und im System TWEAK verwandte *Modal Truth Criterion* [Cha87], welches eine Aussage darüber macht, wann eine bestimmte Aussage notwendigerweise wahr ist.

Der SNLP-Ansatz geht einen anderen Weg als TWEAK, indem hier explizit der Zweck festgehalten wird, den es für die Einführung eines Schrittes in den Plan gibt und indem dieser gegen mögliche Interaktionen geschützt wird.

Dies führt zum Begriff der *Causal Links*[1]:

> Wenn s ein Planschritt ist mit dem Effekt P und t ein anderer Planschritt mit der Vorbedingung P, so kann der *Causal Link* $s \xrightarrow{P} t$ diesen Zusammenhang der beiden Schritte festhalten.

Die *partielle Ordnung auf Planschritten* beim SNLP-Ansatz ist definiert durch eine Menge von Ordnungsbeziehungen zwischen je zwei verschiedenen Schritten. Es gibt zwei Arten von Ordnungen zwischen Schritten s und t, deren Hauptunterschied der Grund ihrer Einführung in den Plan ist, beide ordnen s vor t an:

- Ein *Causal Link* $s \xrightarrow{P} t$ macht die Vorbedingung P des Schrittes t gültig, indem der Schritt s vor t angeordnet wird und durch den Causal Link explizit als Quelle für die Gültigkeit von P gekennzeichnet wird.

- Zusätzliche Ordnungen vom Typ $s \prec t$, *Protections*, sind ggf. nötig, um Interaktionen auflösen zu können (s. Kap. 3.2).

Abbildung 3.1 zeigt nochmals bildlich die Komponenten eines Planes, Schritte und Ordnungen, wie sie im folgenden verwandt werden.

3.2 Interaktionen und ihre Behandlung

Die Erzeugung von Causal Links für eine Vorbedingung P von Schritten alleine garantiert die Gültigkeit der Vorbedingung zur Ausführungszeit noch nicht. Der Causal Link hält lediglich explizit denjenigen Planschritt fest, der das jeweilige Prädikat gültig machen kann. Es können noch Interaktionen zwischen einem Causal Link und einem parallel liegenden Schritt auftreten.

[1] An anderen Stellen werden andere Begriffe für dieselbe Bedetung verwandt: *range* bei Tate, *validation* bei Kambhampati.

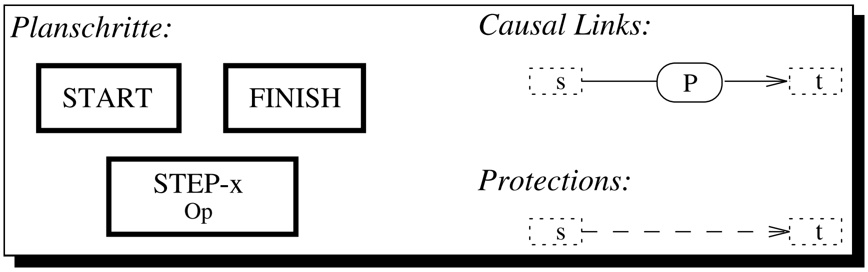

Abbildung 3.1: SNLP-Symbole

3.2.1 Threats

Interaktionen (s. Abb. 3.2) heißen im SNLP-Sprachgebrauch *Threats*; die Namensgebung läßt sich durch die Überlegung motivieren, daß solche Threats das, was ein Causal Link zu garantieren versucht – die Gültigkeit einer bestimmten Aussage an einem bestimmten Schritt – *bedrohen*. Es wird daher im folgenden auch von bedrohten Causal Links oder Bedrohungen gesprochen werden.

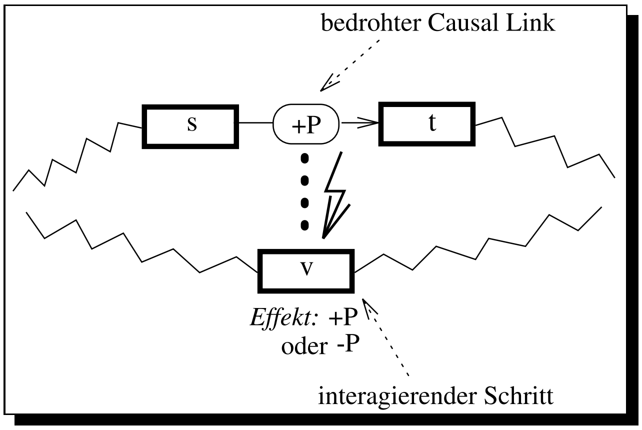

Abbildung 3.2: Interaktionen

Die generelle Definition eines Threats nach [MR91] ist die folgende:

> Ein Threat zu einem Causal Link $s \xrightarrow{P} t$ ist ein anderer Schritt v, der P oder die Negation von P als Effekt hat.

Bei dieser Definition fällt auf, daß es zwei Möglichkeiten für Interaktionen gibt:

- *Negative Interaktionen:*

  Hierbei handelt es sich um Schritte v, die die vom Causal Link zu garantierende Aussage negieren und damit bei Nichtbeachtung zu inkorrekten Plänen führen. Tate's Definition einer Interaktion [Tat77] beschränkt sich auf diesen Typ.

- *Positive Interaktionen:*

  Damit sind Schritte gemeint, die die gleiche Aussage als Effekt haben, also ebenfalls verwendet werden könnten, um die betreffende Vorbedingung gültig zu machen. Nichtbeachtung solcher Interaktionen führt NICHT zu inkorrekten Plänen, zerstört aber die Systematik des Suchprozesses.

Die Beachtung und Auflösung von negativen Interaktionen ist für jedes Planungsverfahren obligatorisch. Der SNLP-Ansatz unterscheidet sich von einigen anderen Verfahren (z.B. [Cha87, Kam92] dadurch, daß er zusätzlich auch positive Interaktionen auflöst.

Der wesentliche Vorteil dabei ist die *Systematik* bei der Suche. Systematische Suche im Raum der möglichen Pläne heißt, daß jedes $P = (S, O, B)$ nur höchstens einmal erzeugt wird. Dazu müssen aber ALLE Interaktionen aufgelöst werden. Die Systematik hat daher u.U. den Nachteil, daß sie zu Ordnungen im Plan führt, die aus Gründen der Korrektheit des Planes nicht notwendig sind. Hier handelt es sich also um ein bewußtes Abrücken von der Least-Commitment-Strategie, die zu mehr Backtracking während der Lösungssuche führt.

3.2.2 Auflösung von Interaktionen

Beim SNLP-Ansatz müssen alle *Threats*, d.h. positive und negative Interaktionen, aufgelöst werden. Zunächst einmal ist dazu das Einführen neuer Ordnungen, *Protections*, geeignet. Betrachtet man das oben gezeigte Beispiel aus Abb. 3.2, so gibt es dazu zwei Möglichkeiten, entweder durch die Ordnung (a) $v \prec s$ oder durch (b) $t \prec v$ (s. Abb. 3.3).

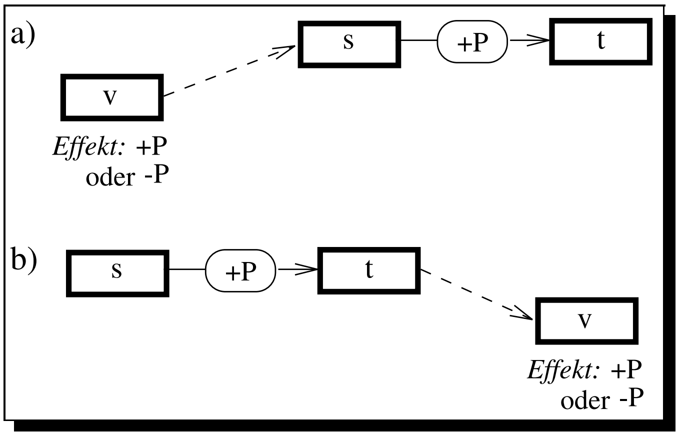

Abbildung 3.3: Auflösung von Interaktionen durch Einführen von Ordnungen

Im allgemeineren Fall besteht allerdings auch die Möglichkeit, daß die von der Interaktion betroffene Aussage P noch Variablen enthält.

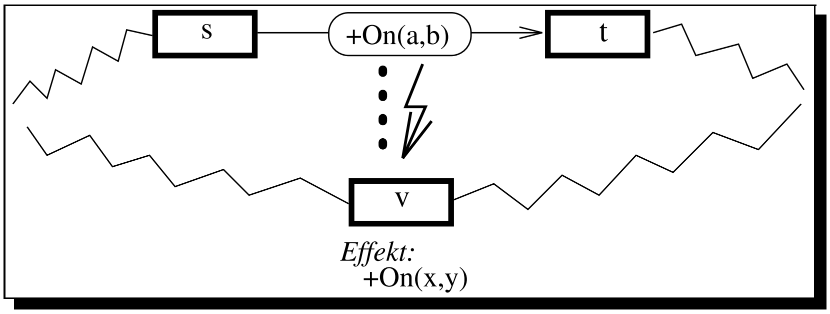

Abbildung 3.4: Interaktion, die durch Constraints aufgelöst werden kann

Beispiel: Parallel zum Causal Link $s \stackrel{+On(a,b)}{\longrightarrow} t$ liegt ein Schritt mit dem Effekt +On(x,y), wobei `x` und `y` Variablen seien. Dieser Schritt ist bei geeigneter Variablenbindung von `x` und `y` für den Causal Link ein Bedrohung (s. Abb. 3.4).

Die oben beschriebene Möglichkeit der Auflösung mittels neuer Ordnungen existiert hier natürlich weiterhin. Allerdings gibt es zusätzlich weitere Möglichkeiten, die Interaktion durch Einführen von Constraints aufzulösen, ohne die Schritte dabei anzuordnen:

- `NotSame(x,a)` und `NotSame(y,b)`

- `NotSame(x,a)` und `Same(y,b)`
- `Same(x,a)` und `NotSame(y,b)`

Auf den ersten Blick verwundert wahrscheinlich, daß es drei Möglichkeiten gibt und jede von ihnen Constraints über beide beteiligten Variablen einführt, wo doch eigentlich nur die beiden Möglichkeiten `NotSame(x,a)` und `NotSame(y,b)` reichen würden, um die Interaktion aufzulösen. Der Grund dafür ist wieder (wie die Beachtung positiver Interaktionen), daß nur so die Systematik des Suchprozesses garantiert bleibt.

3.3 Lösung von Planungsproblemen

Die vorangegangen Abschnitte haben die wesentlichen Begriffe erläutert, die für das Verständnis des eigentlichen Planungsalgorithmus von Bedeutung sind. Nun soll die Vorgehensweise des SNLP-Algorithmus aus [MR91] in der auf Variablen erweiterten Version aus [BW93] erläutert werden.

3.3.1 Konsistente Pläne

Im Hinblick auf Ordnungen zwischen Planschritten und die Variablenbindungs-Constraints kann man zwei Konsistenzeigenschaften von Plänen definieren:

- *Ordnungskonsistenz* ist gegeben, wenn der Plan folgende Eigenschaften erfüllt:
  - Kein Schritt kann vor dem START-Schritt angeordnet sein.
  - Kein Schritt hinter dem FINISH-Schritt angeordnet sein kann.
  - Es bestehen keine Zykel bzgl. der Ordnung von Schritten. Zykelfreiheit heißt, daß ein Plan ordnungsinkonsistent wird, falls es zwei Schritte s und d gibt, für die $s \prec d$ aufgrund der Ordnungen $s \prec v_1 \prec \cdots \prec v_n \prec d$ gilt, und man die Ordnung $d \prec s$ ergänzt.

  Insbesondere gilt daher für einen ordnungskonsistenten Plan immer, daß der START-Schritt vor dem FINISH-Schritt angeordnet ist.

- *Bindungskonsistenz* ist gegeben, wenn die Menge der Variablenbindungs-Constraints konsistent ist, also wenn für jede Variable mindestens eine Bindungsmöglichkeit existiert, die konsistent mit alle Constraints ist.

Ein Plan ist *konsistent*, wenn er sowohl ordnungs- als auch bindungskonsistent ist.

3.3.2 Vollständige Pläne

Ziel des SNLP-Algorithmus ist es, sukzessive einen konsistenten Plan für das gegebene Planungsproblem zu erzeugen, in welchem die Vorbedingungen aller Schritte garantiert sind. Man spricht im SNLP-Sprachgebrauch in diesem Zusammenhang von der Erzeugung von *vollständigen Plänen*.

Ein (partiell geordneter) Plan $P = (S, O, B)$, bestehend aus einer Menge S von Planschritten, einer partiellen Ordnung O (Causal Links und Protections) auf diesen Schritten und einer Menge B von Constraints über vorkommenden Variablen, heißt *vollständig*, wenn gilt:

- Für jede Vorbedingung P eines Schrittes $w \in S$ existiert ein Causal Link der Form $s \xrightarrow{P} w$, der den Schritt s als Quelle für P festhält.
- Jeder vorkommende Causal Link $s \xrightarrow{P} t$ ist gegen alle Bedrohungen (Threats) geschützt, d.h. bedroht der Schritt v einen solchen Causal Link, so enthält der Plan entweder eine Ordnung oder eine Reihe von Constraints (s. 3.2), die die Interaktion auflösen.

Man kann nun zeigen (vgl. [MR91]), daß

- jede Linearisierung eines vollständigen Plans eine korrekte Lösung des Problems ist und
- jede Lösung eines STRIPS-Planungsproblems mit einem solchen vollständigen Plan korrespondiert[2], d.h. wenn es eine Lösung gibt, so gibt es auch einen vollständigen Plan, den der Algorithmus findet.

Betrachtet man die erste Aussage, so muß folgendes festgehalten werden: Vollständige Pläne sind zwar auch korrekt, aber nicht jeder an sich schon korrekte Plan ist im Sinne von SNLP vollständig.

Beispiel: Abbildung 3.5 zeigt einen solchen Fall. FINISH hat hier die drei Vorbedingungen `+P,+Q,+R`, die im Prinzip von dem in Abbildung 3.5 gezeigten Plan erfüllt werden.

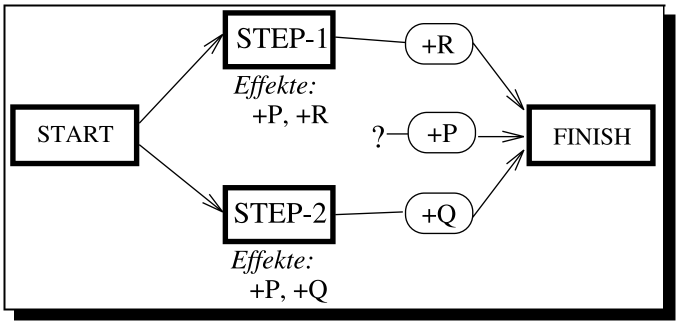

Abbildung 3.5: Korrekter, aber nicht vollständiger Plan

Hier verlangt der SNLP-Algorithmus aber noch zwei Dinge: zum einem muß eine Entscheidung getroffen werden, welcher der beiden Schritte, `STEP-1` oder `STEP-2`, als Quelle für `+P` gewählt werden soll. Ist dies dann durch einen Causal Link repräsentiert, entsteht in jedem Fall noch eine positive Interaktion, die auch eine explizite Anordnung der Schritte `STEP-1` und `SPEP-2` verlangt.

An dem Beispiel sieht man, daß die Systematik bei der Suche letztlich den Nachteil hat, daß Pläne (in Bezug auf die Korrektheit des Planes) unnötig viele Ordnungen enthalten können.

3.3.3 Der Algorithmus

Der SNLP-Algorithmus erzeugt sukzessive zu einem gegebenen Planungsproblem einen vollständigen, konsistenten Plan. Dies geschieht so, daß schrittweise für alle Vordingungen von Schritten ein gegen alle existierenden Bedrohungen geschützter Causal Link erzeugt wird.

Abbildung 3.6 zeigt den rekursiven Algorithmus im Überblick. Zum besseren Verständnis wurden bei der Notation folgenden Abkürzungen verwendet:

- Der Parameter P, mit dem der rekursive Algorithmus aufgerufen wird, ist der aktuelle Planzustand und es ist $P = (S, O, B)$ wie in Kap. 2.2.4 definiert. Vor dem ersten Aufruf wird der Plan so initialisiert, daß der nur die beiden Schritte START und FINISH enthält (vgl. Kap. 3.1.1).
- $open\_precond(P)$ bezeichnet die Menge der Vorbedingungen von Schritten des Planes P, für die noch kein Causal Link erzeugt wurde.
- $step(P)$ bezeichnet den Schritt, für den P eine Vorbedingung ist.

[2] Dabei ist mit *korrespondiert* gemeint, daß sich beide Pläne nur im Hinblick auf die Ordnungen unterscheiden: die Lösung ist also eine Linearisierung eines vollständigen Planes, den der Algorithmus finden kann.

Causal_Link_Planner(P)

1. **Termination:** Falls Plan P vollständig ist, dann fertig (Erfolg).
2. **Ziel-Auswahl:**
   Auswahl einer Vorbedingung $c \in open\_precond(P)$
   $dest := step(c)$
3. **Operator-Auswahl:**
   Sei *source* ein Schritt der den Effekt c hat (entweder ein neuer Schritt oder ein existierender, der nicht hinter *dest* angeordnet ist; u.U. sind noch Variablenbindungsconstraints $var\_bindings(source, c)$ nötig)

   Falls kein solcher Schritt *source* existiert, dann führe Backtracking durch.

   $S' := S \cup \{source\}$
   $O' := O \cup \{source \xrightarrow{c} dest\}$
   $B' := B \cup var\_bindings(source, c)$

   Backtracking-Punkt: Jede Möglichkeit, solch einen Schritt *source* so auszuwählen, daß der resultierende Plan $P' = (S', O', B')$ wieder *konsistent* ist, ist ein Punkt, zu dem durch Backtracking zurückgesprungen werden kann.
4. **Schützen von Causal Links:**
   Sei s_k ein Threat für ein $s_i \xrightarrow{p} s_j \in O'$ und sei s_k parallel zu diesem Causal Link.

   Für alle solchen Schritte s_k:

   Schütze den Causal Link durch Hinzufügen von Ordnungen oder Constraints zu O' bzw. B'.
   Ist dies nicht möglich, führe Backtracking durch.

   Backtracking-Punkt: Jede Möglichkeit, einen Causal Link konsistent zu schützen, ist ein Backtracking-Punkt.
5. **Rekursiver Aufruf:** Causal_Link_Planner(P') mit $P' = (S', O', B')$

Abbildung 3.6: Der Algorithmus

- $var\_bindings(s, c)$ ist die Menge von Variablenbindungs-Constraints vom Typ `Same`, die notwendig sind, damit ein Effekt von Schritt s mit c übereinstimmt[3].

Der Algorithmus arbeitet mit *chronologischem Backtracking*: jede Möglichkeit, einen Causal Link zu erzeugen oder gegen Bedrohungen zu schützen, ist dabei ein Backtracking-Punkt. Ist es an einer Stelle nicht mehr möglich, einen geeigneten Causal Link zu erzeugen oder einen Causal Link gegen eine Bedrohung zu schützen, ohne daß der Plan inkonsistent wird, so wird Backtracking durchgeführt.

Zunächst ist es wichtig, zu erwähnen, daß bei diesem Algorithmus die Ziel-Auswahl *kein* Backtracking-Punkt ist, wie es in anderen Systemen der Fall ist (z.B. NoLimit [Vel92]). Man kann sich auch leicht klarmachen, daß ein Verfahren, welches die Zielauswahl als Backtracking-Punkt vorsieht, die Forderung nach Systematik bei der Suche im Raum möglicher Pläne nur in Einzelfällen erfüllen kann, da es die gleiche Menge von Teilzielen u.U. mehrfach zu lösen versucht (s. Kap. 6).

Die Operator-Auswahl erzeugt für eine offene Vorbedingung einen Causal Link. Dies kann grundsätzlich auf zwei Arten geschehen:

- Es wird explizit dazu ein neuer Schritt *source* in den Plan aufgenommen, der die Vorbedingung wahr machen kann. Dies entspricht der Anwendung eines Domänenoperators.
- Es wird ein bereits existierender Schritt dazu ausgenutzt, d.h. es gilt bereits $source \in S$ und daraus ergibt sich $S' = S$. Dies entspricht der sog. *Phantomisierung eines Zieles*; es mußte keine neue Aktion in den Plan aufgenommen werden, um die Vorbedingung zu erfüllen.

[3]Beispiel: Der Effect `+Clear(x)` stimmt mit Hilfe des Constraints `Same(x,a)` mit `+Clear(a)` überein.

Durch die Operator-Auswahl und -Anwendung wird der Plan verändert: es kommt mindestens ein neuer Causal Link hinzu, evt. auch ein Schritt und einige Variablenbindungs-Constraints. Um einen vollständigen Plan zu erhalten müssen nun die daraus resultierenden Bedrohungen behandelt werden. Dies geschieht wie in Kap. 3.2 beschrieben und ist wieder ein Backtracking-Punkt.

3.3.4 Eigenschaften des Algorithmus

Die Anwendung des Algorithmus zur systematischen, nichtlinearen Planung auf ein Planungsproblem kann zu folgenden möglichen Ergebnissen führen:

- *Erfolg:*

  Der Algorithmus terminiert erfolgreich, wenn für alle Ziele ein geschützter Causal Link erzeugt wurde und damit ein vollständiger Plan gefunden wurde.

- *Mißerfolg:*

  Dies wird dann gemeldet, wenn Backtracking nicht mehr durchgeführt werden kann, da kein Backtracking-Punkt mehr existiert.

- *Nichttermination:*

  Schließlich besteht noch die Möglichkeit, daß der Algorithmus nicht terminiert, weil Teilzielrekursionen auftreten, die vom Algorithmus nicht erkannt werden.

Beim Algorithmus, wie er ursprünglich in [MR91] beschrieben ist, gibt es das Problem der Nichttermination nicht, da hier auch dann Backtracking gemacht wird, wenn die *Kosten des Planes*, die über die Kosten von Planschritten definiert werden, einen gegebenen Maximalwert überschreiten. Dieser Aspekt wurde bei [BW93] nicht berücksichtigt, konnte hier jedoch auf andere Weise behandelt werden (s. Kap. 3.4)

3.4 Verbesserungen des SNLP-Ansatzes

3.4.1 Teilzielrekursion

Wie bei der Beschreibung der Eigenschaften des SNLP-Algorithmus erwähnt, kann es sein, daß dieser Algorithmus nicht terminiert. In einem solchen Fall treten bei der Herleitung von neuen Teilzielen durch Anwendung von Operator Rekursionen auf, die den Plan beliebig größer werden lassen, aber niemals alle Teilziele wirklich erfüllen können.

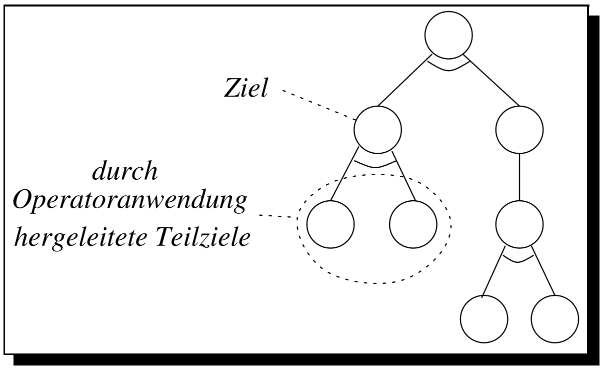

Abbildung 3.7: Teilzielhierarchie

Die Herleitung von Teilzielen bei der Anwendung von Operatoren kann man mit einer Teilzielhierarchie beschreiben: jedes Ziel ist ein Knoten in dieser Hierarchie, die Nachfolger eines Knotens sind die durch eine Operatoranwendung hergeleiteten Ziele (s. Abb. 3.7).

Teilzielrekursionen entstehen, wenn ein Ziel G in einer solchen Teilzielhierarchie einen durch Operatoranwendung hergeleiteten Nachfolger hat, der identisch mit G ist oder oder durch Variablenbindung identisch werden kann.

Beispiel: Die Bearbeitung eines Teilzieles `+Clear(a)` leitet direkt oder indirekt (d.h. über mehrere Stufen im der Teilzielhierarchie) wieder `+Clear(a)` als Unterziel her. Dieses Ziel führt unmittelbar zu einer Teilzielrekursion.
Wird stattdessen nur ein Teilziel `+Clear(x)` hergeleitet, so kann in Verbindung mit dem Variablenbindungs-Constraint `Same(x,a)` eine Teilzielrekursion entstehen.

Für die Teilzielhierarchie hat das die Konsequenz, daß dort Teilbäume rekursiv immer wieder vorkommen. Abbildung 3.8 veranschaulicht diesen Sachverhalt noch einmal.

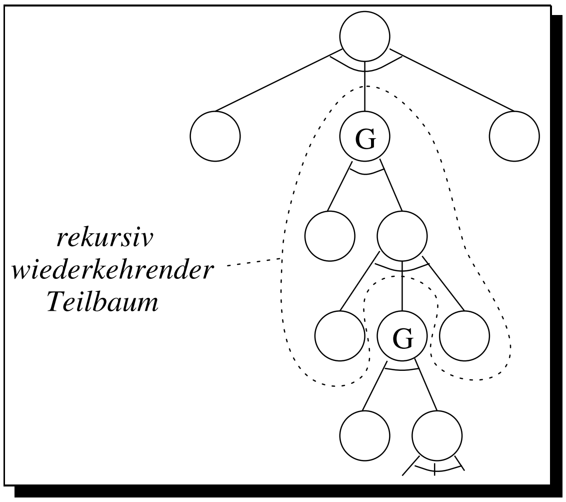

Abbildung 3.8: Teilzielrekursion

3.4.1.1 Erkennung und Behandlung von Teilzielrekursionen

Die Erkennung von Teilzielrekursionen erfordert, daß während des Problemlösevorgangs die durch Anwendung von Operatoren entstehende Teilzielhierarchie explizit verwaltet wird. Ausgehend davon kann dann bei der Herleitung von neuen Teilzielen für jedes neue Teilziel getestet werden, ob entlang des Pfades zur Wurzel der Teilzielhierarchie ein Ziel liegt, welches mit dem neuen Teilziel identisch ist oder mittels Variablenbindungs-Constraints mit diesem identisch werden kann.

Die Auflösung einer potentiellen Teilzielrekursion ist nur dann noch ohne Backtracking möglich, wenn erst noch Variablenbindungs-Constraints notwendig sind, damit sie wirklich entsteht. In diesem Fall aber kann exakt genauso vorgegangen werden wie bei der Auflösung von Interaktionen durch die Einführung von Constraints (vgl. Kap. 3.2.2), insbesondere auch im Hinblick auf den Aspekt der Systematik bei der Auflösung von Teilzielrekursionen ist die Vorgehensweise die gleiche. Bei dem zweiten Beispiel aus Abschnitt 3.4.1 würde eine Teilzielrekursion zwischen Ziel `+Clear(a)` und dem Unterziel `+Clear(x)` durch Einführung des Constraints `NotSame(x,a)` verhindert werden können.

3.4.1.2 Erweiterung des Algorithmus

Der Algorithmus aus Abbildung 3.6 läßt sich leicht so erweitern, daß auch Teilzielrekursionen erkannt und behandelt werden können. Es wird dabei davon ausgegangen, daß mittels einer Teilzielhierarchie für ein hergeleitetes Ziel entschieden werden kann, ob es sich dabei um einen Teilzielrekursion handelt.

Konkret heißt das, daß der Punkt 3 des Algorithmus zur Erkennung und Behandlung von Teilzielrekursionen, wie in Abbildung 3.9 gezeigt, erweitert werden muß. Zu beachten ist, daß damit ein

(...)

3. (a) wie Punkt 3 aus dem Algorithmus aus Abbildung 3.6
   (b) Für alle $p \in open\_precond(P') - open\_precond(P)$, die eine Teilzielrekursion verursachen können:
   Löse die Teilzielrekursion verursacht von p durch Hinzufügen von Constraints zu B' auf.
   Ist dies nicht möglich, führe Backtracking durch.

Backtracking-Punkt: Jede Möglichkeit, eine Teilzielrekursion konsistent aufzulösen, ist ein Backtracking-Punkt.

(...)

Abbildung 3.9: Erweiterung des Algorithmus

neuer Backtracking-Punkt im Algorithmus hinzukommt, da jede Möglichkeit, eine Teilzielrekursion aufzulösen, betrachtet werden muß.

3.4.2 Backtracking

Die Behandlung von Backtracking ist ein weiterer Punkt, an dem mit Verbesserungen angesetzt werden kann, indem auch die Ursachen für die Notwendigkeit von Backtracking analysiert und berücksichtigt werden.

3.4.2.1 Chronologisches Backtracking

Der SNLP-Ansatz verwendet *chronologisches Backtracking* bei der Suche nach einer Lösung. Es werden sukzessive für alle offenen Ziele (Vorbedingungen von Schritten) Causal Links erzeugt und existierende Bedrohungen aufgelöst.

Denkt man sich in diesem Suchprozeß die Backtracking-Punkte (Erzeugung und Schützen der Causal Links und das Beheben von Teilzielrekursionen) als eine Folge von zu bearbeitenden Zielen und den darauf angewandten Operatoren (Entscheidungen), so läßt sich das Vorgehen mit chronologischem Backtracking gut in Form eines Baumes darstellen. Knoten stehen für ein bearbeitetes Ziel, also für einen Backtracking-Punkt. Eine Kante zu einem Nachfolger steht für die Entscheidung[4], eine der Möglichkeiten zu wählen, das Ziel zu bearbeiten: (s. Abb. 3.10)

- Ein Durchlauf in diesem Baum über die Knoten G1, G2, ... G5, G4, G6, G7, G6, G4, G8, ... veranschaulicht die Vorgehensweise des SNLP-Algorithmus. Jede getroffene Entscheidung (D_{ij}) repräsentiert die Auswahl eines Operators, eine Möglichkeit zum Schützen eines Causal Links oder eine Möglichkeit zum Auflösen von Teilzielrekursionen im Algorithmus.
- *Chronologisches Backtracking:* Nachdem der Algorithmus G5 als nächstes zu bearbeitendes Ziel ausgewählt hatte, mußte Backtracking durchgeführt werden, da keine konsistente Möglichkeit existierte, G5 zu erfüllen. Bei G5 kam es zu einer sog. *Zielblockade*. D.h. die zuletzt getroffene Entscheidung D_{41} wird zurückgenommen und es wird zur Bearbeitung von Ziel G4 zurückgesprungen und dort eine andere Alternative gewählt.
- Es bleibt unberücksichtigt, was der Grund dafür war, daß Ziel G5 nicht erfüllt werden konnte.

War im obigen Beispiel etwa die Entscheidung D_{21} der eigentliche Grund dafür, daß die erfolgreiche Bearbeitung von G5 scheiterte, so wird durch Rückzug von D_{41} und später D_{42}, D_3, usw. ein unnötiger Teil des Suchbaumes durchlaufen.

[4]Der Begriff der Entscheidung wird im folgenden Kapitel wieder auftauchen.

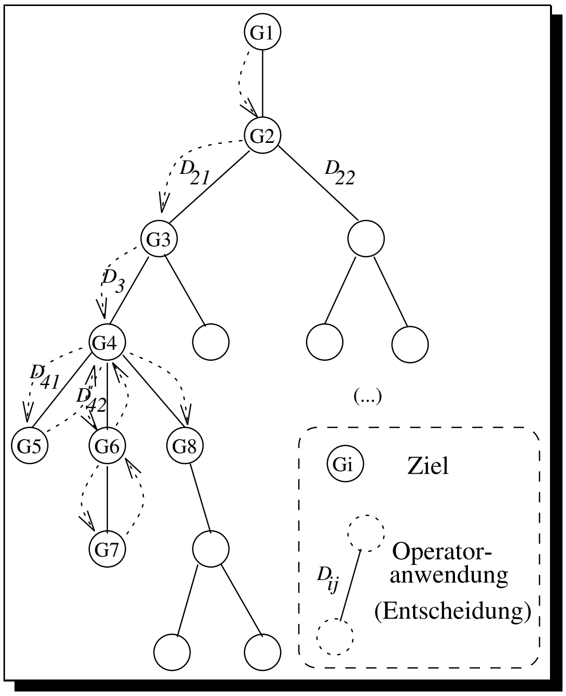

Abbildung 3.10: Chronologisches Backtracking

3.4.2.2 Verbesserung: dependency-directed Backtracking

Der SNLP-Ansatz läßt sich im Hinblick auf die Behandlung von Backtracking verbessern, wenn man durch eine Inkonsistenzanalyse nach dem Grund für die Notwendigkeit zum Backtracking sucht und abhängig davon bis zu einem bestimmten Backtracking-Punkt zurückspringt, der nicht unbedingt der letzte Backtracking-Punkt sein muß. Die Vorgehensweise ist auch in [BV92] beschrieben und unterscheidet sich von chronologischem Backtracking dadurch, daß u.U. mehr als einen Schritt zurücksprungen wird (*back-jumping*).

Abbildung 3.11 zeigt dies am Beispiel aus dem letzten Abschnitt. Nimmt man an, daß eine Inkonsistenzanalyse an Knoten G5, wo die Notwendigkeit für Backtracking erkannt wurde, ergibt, daß D_{21} der nächstliegende Grund (*culprit*) für die Inkonsistenz ist, so ist der Gewinn beim Rücksprung bis zum Backtracking-Punkt G2 (d.h. die Rücknahme von D_{41}, D_3 und D_{21}) offensichtlich. Die Inkonsistenzanalyse muß jedoch unbedingt garantieren, daß als ermittelter Grund eine Entscheidung geliefert wird, die garantiert, daß beim Backtracking damit die Lösung niemals übersprungen werden kann, daß also die Vollständigkeit des SNLP-Verfahrens davon nicht zerstört wird.

3.4.2.3 Inkonsistenzanalyse für dependency-directed Backtracking

Die Inkonsistenzanalyse ist der kritische Punkt beim dependency-directed Backtracking. Der SNLP-Algorithmus versucht an jedem Backtracking-Punkt zu erreichen, daß ein konsistenter Plan entsteht. Backtracking wird notwendig, wenn eine Zielblockade auftritt, d.h. an einem Backtracking-Punkt sind alle Möglichkeiten, das zugehörige Ziel zu erreichen (Erzeugen oder Schützen eines Causal Link oder Auflösen einer Teilzielrekursion) ohne Erfolg getestet worden oder inkonsistent oder es existiert keine Möglichkeit.

Jede als inkonsistent erkannte Alternative enthält nun zumindest indirekt Informationen über den potentiellen Grund für das Fehlschlagen. Ein Konsistenztest überprüft hier die Menge der Ordnungen bzw. die Menge der Variablenbindungs-Constraints im Plan und liefert bei Inkonsistenzen die Teilmenge der Ordnungen bzw. Constraints, die für das Zustandekommen der Inkonsistenz verantwortlich sind. Die zu einer solchen Teilmenge gehörige Menge von Entscheidungspunkten sind potentielle

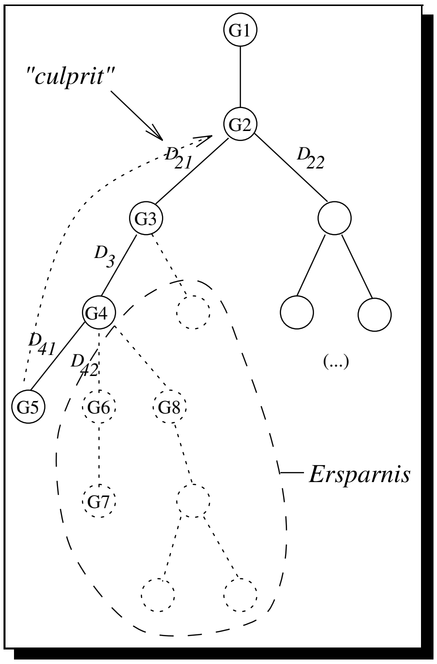

Abbildung 3.11: Dependency-directed Backtracking

Kandidaten für das Entstehen der Inkonsistenz. Wichtig ist also, daß für eine Ordnung bzw. einen Constraint der zugehörige Entscheidungspunkt explizit feststellbar ist.

Beispiel: Nimmt man an, daß ein Plan $P = (S, O, B)$ bereits die Teilmenge $O_I = \{o_1, \cdots, o_n\}$ von Ordnungen enthält, aus denen für die Schritte s und d folgt, daß $d \prec s$ gilt. Dann ist die Alternative, eine Vorbedingung P des Schrittes d durch den Schritt s zu garantieren, inkonsistent, weil $d \prec s$ zusammen mit $s \xrightarrow{P} d$ einen Zykel bilden würde.
Jede dieser Ordnungen aus O_I wurde an einem Backtracking-Punkt durch die Auswahl einer Alternative in den Plan aufgenommen. Die Menge $D_{candidates}$ dieser Backtracking-Punkte sind die potentiellen Kandidaten für ein gezieltes Backtracking. Bei Bindungsinkonistenzen ist eine solche Menge $D_{candidates}$ auf ähnliche Weise feststellbar.

Aus allen aufgrund von Konsistenztests fehlgeschlagenen Alternativen $D_{ij}, j \in J$ an einem Backtracking-Punkt G_i ergibt sich eine Menge $D_{candidates} = \{D_{ij} | j \in J\} \cup \{D_{parent}\}$ von Kandidaten für die zurückzuziehende Entscheidung. D_{parent} ist dabei die Entscheidung, die das Ziel G_i hergeleitet hat (diese darf auf keinen Fall vergessen werden). Aus der Menge $D_{candidates}$ muß eine Auswahl getroffen werden, die garantiert, daß die Vollständigkeit erhalten bleibt.

Betrachtet man die Baumstruktur zur Visualisierung des Backtrackings, so kann man feststellen, daß alle Entscheidungen aus $D_{candidates}$ Kanten auf dem Pfad von der Wurzel zum Knoten sind, an dem die Inkonsistenz entstand. Durch die Reihefolge, in der sie auf diesem Pfad liegen, lassen sich die Elemente aus $D_{candidates}$ anordnen. Die als Grund für die Inkonsistenz auszuwählende Entscheidung D_{cuprit} muß bezüglich dieser Ordnung möglichst nahe an dem Knoten liegen, an dem die Inkonsistenz entstand.

Abbildung 3.12 veranschaulicht dies: auf dem Pfad von G1 zu G4 liegen die Entscheidungen D_1, D_2 und D_3 und eine Inkonsistenzanalyse ergibt z.B. $D_{candidates} = \{D_1, D_2\}$. Hier ist $D_{cuprit} = D_2$ die zu G4 am nächsten liegende Entscheidung, bis zu der zurückgesprungen werden sollte.

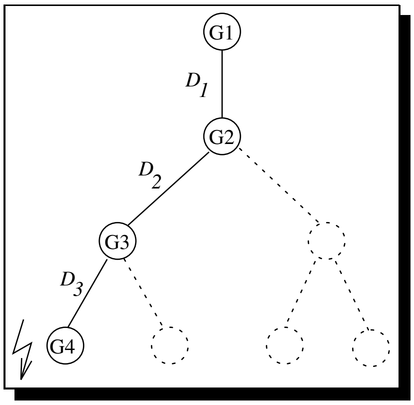

Abbildung 3.12: Auswahl des *Culprit*

3.4.2.4 Erhaltung der Vollständigkeitseigenschaft

Es ist wichtig, daß beim oben beschriebenen Vorgehen die Vollständigkeit des Algorithmus nicht zerstört wird. Um dies zu zeigen, soll dieses Vorgehen mit dem Vorgehen beim chronologischen Backtracking verglichen werden. Sei G_I das Ziel, an dem die Inkonsistenz auftrat.

Es gibt vier Möglichkeiten, die man betrachten muß. Sie betreffen die Frage, ob das Ziel G_I bei chronologischem Backtracking hätte bearbeitet werden müssen oder nicht sowie die Frage, ob D_{parent} als D_{cuprit} gewählt wurde oder nicht:

1. *Ann. G_I muß bearbeitet werden:*

   (a) $D_{parent} \neq D_{cuprit}$:
   Die nächstliegenste Entscheidung, die die Inkonsistenz verursacht haben könnte, wird zurückgenommen, um G_I erfüllbar zu machen. Keine andere, näher liegenden Entscheidungsrücknahme hätte die schon bestehende Inkonsistenz G_I auflösen können.

   (b) $D_{parent} = D_{cuprit}$:
   Die Inkonsistenz wird dadurch beseitigt, daß die Entscheidung revidiert wird, die das Ziel G_I eingeführt hat. Da aber G_I nach Ann. bearbeitet werden muß, liegt der eigentliche Grund weiter oben. Dennoch macht man keinen Fehler, wenigstens bis D_{parent} zurückzuspringen, da auch chronologisches Backtracking G_I so nicht hätte erfolgreich bearbeiten können.

2. *Ann. G_I muß nicht bearbeitet werden:*
   In diesem Fall könnte man bis zu D_{parent} zurückspringen, ohne einen Fehler zu machen, da D_{parent} das Ziel G_I herleitete, was aber nach Ann. falsch war.

   (a) $D_{parent} = D_{cuprit}$:
   Es wird genau das oben genannte getan, da Ziel G_I fehlerhafterweise von D_{parent} hergeleitet wurde.

   (b) $D_{parent} \neq D_{cuprit}$:
   Es wird weniger weit zurückgesprungen, als theoretisch möglich, aber damit ist garantiert, daß die Vollständigkeit nicht verletzt wird.

Daraus ergibt sich, daß bei der Auswahl des Rücksprungpunktes in jedem der möglichen vier Fälle noch dasselbe Ergebnis gefunden wird, das auch bei chronologischen Backtracking gefunden würde. Da aber der Algorithmus mit chronologischem Backtracking nach [MR91] vollständig ist, bleibt diese Eigenschaft auch bei dieser Erweiterung im Bezug auf Backtracking erhalten.

3.4.3 Zusammenfassung der Verbesserungsmöglichkeiten

In den vorangegangen Abschnitten wurde zwei Verbesserungsmöglichkeiten des Algorithmus aus Abbildung 3.6 diskutiert:

- Das Beheben von Teilzielrekursionen löst das Problem der Nichttermination, indem in einer Erweiterung Rekursionen in der Teilzielhierarchie erkannt und verhindert werden.
- Das Backtracking wurde durch die Einbeziehung von Informationen über potentielle Gründe für das Entstehen von Inkonsistenzen so verbessert, daß die Vollständigkeitseigenschaft des Algorithmus erhalten bleibt, aber der Suchraum verkleinert wird oder im ungünstigsten Fall gleich groß bleibt.

Kapitel 4

REDUX

Das REDUX-System [Pet91, Pet92] von Charles Petrie war für die Realisierung des im vorherigen Kapitel beschriebene Planungsverfahren die Basis. Es stellt einige für die Bereiche Konfiguration und Planung grundlegende Mechanismen zur Verfügung, wie der Titel *Ontologie der Planung und Planmodifikation* bereits suggeriert. Im Rahmen der Entwicklung des Konfigurationssystems IDAX [Rit92] entstand eine Implementierung der von Petrie vorgeschlagenen Konzepte auf Basis eines *Truth Maintenance Systems (TMS)*, speziell eines JTMS[1] [Doy79].

In den folgenden Abschnitten soll auf die Grundkonzepte von REDUX eingegangen werden, sowie auf einige Aspekte der Realisierung dieser Konzepte auf der Ebene eines JTMS, da einige Begriffe aus diesem Bereich auch im folgenden Kapitel eine Rolle spielen werden.

4.1 Grundkonzepte von REDUX

Das REDUX-System kann zunächst völlig losgelöst von seiner Realisierung mittels eines TMS betrachtet werden. Petrie hat mit seiner REDUX-Ontologie eine für den Bereich der *Planung und Planmodifikation* charakteristische Begriffswelt definiert, mit der sich die Repräsentation und Lösung von solchen Aufgaben einfacher und übersichtlicher gestaltet. Insbesondere ist dabei die explizite Repräsentation von Abhängigkeit von großer Bedeutung.

4.1.1 Schlüsselbegriffe

Zentral bei REDUX sind die Begriffe *Ziel* und *Operator*. Sie sind auch typisch für die Beschreibung von Konfigurations- und Planungsaufgaben: die Konfiguration eines Objektes oder die Planung einer komplexen Aktion ist dort das zu erreichende Ziel.

Zur Erreichung von Zielen dienen Operatoren; Operatoren werden auf Ziele angewandt, wobei u.U. neue Unterziele hergeleitet werden (Zielreduktion). Außerdem hat eine Operatoranwendung im Hinblick auf die zu lösende Aufgabe u.U. gewisse Auswirkungen. Petrie verallgemeinert dies, indem er sagt, eine Operatoranwendung hat zur Folge, daß neue Unterziele hergeleitet werden und/oder Zuweisungen (*assignments*) gemacht werden. Dadurch, daß eine Operatoranwendung wieder neue Unterziele herleiten kann, ergibt sich im Laufe eines Problemlösevorgangs eine Hierarchie von Zielen (s. Abb. 4.1), ein Baum, dessen Wurzel ein initiales Ziel zur Lösung eines Problems ist. Die Nachfolger eines Knotens in dieser Teilzielhierarchie sind die durch eine Operatoranwendung hergeleiteten Unterziele.

Die Menge der momentan gültigen Zuweisungen beschreibt die momentane Teillösung für das gegebene Problem. Dabei ist eine Zuweisung nur dann gültig, wenn der zugehörige Operator gültig ist. Bei der Aktionsplanung sind z.B. konkrete Aktionen in der Planungswelt oder die Anordnung von Aktionen

[1] Justification-based Truth Maintenance Systems

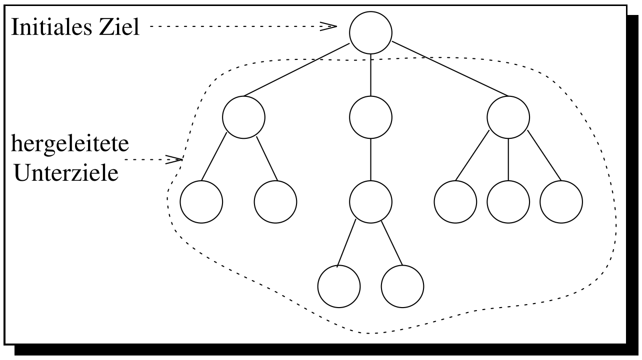

Abbildung 4.1: Teilzielhierarchie

als Zuweisungen eines Operators zu verstehen und diese Aktionen (Planschritte) und ihre Anordnung repräsentieren einen Plan.

Meist gibt es mehr als eine Möglichkeit, auf ein Ziel einen Operator anzuwenden, eine sog. *Konfliktmenge* von auf ein Ziel anwendbaren[2] Operatoren. Dies führt zum zentralen Begriff der Entscheidung (*Decision*): die Auswahl eines Operators aus einer Konfliktmenge von Operatoren zur Reduzierung eines Zieles stellt eine Entscheidung dar, für die eine Begründung definiert werden kann. Abbildung 4.2 verdeutlich dies noch einmal: aus einer Konfliktmenge wird ein Operator ausgewählt, welcher neue Unterziele herleitet und Zuweisungen macht.

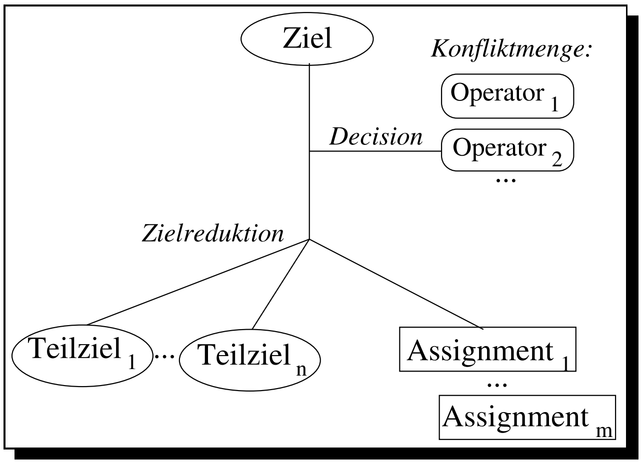

Abbildung 4.2: Zielreduktion in REDUX

Ziele, auf die ein Operator angewandt werden konnte, heißen *reduziert* (*reduced*). Sind alle Ziele reduziert, ist die gestellte Aufgabe gelöst und die mit den getroffenen Entscheidungen (Operatoranwendungen) assoziierten Zuweisungen beschreiben die Lösung. Daher kann man die Suche nach einer Lösung als eine Folge von Operatoranwendungen (Entscheidungen) verstehen.

An eine Lösung werden nun in der Regel spezielle Anforderungen gestellt, was man sich als eine Menge von Constraints über den aktuell gültigen Zuweisungen denken kann[3]. Petrie spricht auch von der Klasse der *Constrained Decision Problems*, die charakterisiert sind durch mehrere zu erreichende Zielsetzungen, viele denkbare Lösungen, von denen eine gesucht ist, generelle Constraints, die die Lösung erfüllen muß, eine einfache Repräsentation durch Ziele, die in Teilziele zerlegbar sind und die

[2]Der Begriff der *Anwendbarkeit* eines Operators hat bei REDUX eine andere Bedeutung als bei der Aktionsplanung, wo ein Operator anwendbar ist, wenn seine Vorbedingungen erfüllt sind. Hier heißt *anwendbar* lediglich, daß der Operator geeignet ist, das Ziel zu erfüllen. Die Erfüllbarkeit von Vorbedingungen (sie wird auf die Erfüllbarkeit der hergeleiteten Unterziele abgebildet) bleibt in dem Zusammenhang unberücksichtigt.

[3]Nimmt man den Bereich der Aktionsplanung, so ist die Forderung, daß das Anordnen von Aktionen A und B nicht zu einem Zykel (A vor B und B vor A) führen darf, ein Constraint über den Zuweisungen einer Lösung.

Notwendigkeit auf sich ändernde Umstände, die das System nicht manipulieren kann, durch teilweises Neuplanen zu reagieren.

Beispiel: Petrie nennt in [Pet92] als prototypisches Beispiel für ein *Constrained Decision Problems* die Planung einer Reise.

Ein einer einfachen Variante geht es z.B. darum, von einem Ort A zu einem anderen Ort B zu reisen. Es gibt drei Möglichkeiten dazu: (1) fliegen, (2) einen Bus nehmen und (3) ein Taxi nehmen. Jede dieser Möglichkeiten verursacht Kosten und nimmt eine bestimmte Zeit in Anspruch. Wenn man einen Flug wählt, zerlegt man das Problem in die Teilziele (a) zu einem Flughafen zu kommen, (b) einen Flug zu wählen und (c) vom Flughafen, wo der Flug ankommt, zu seinem Ziel zu gelangen. Es sind folgende Eventualitäten denkbar: Flüge oder Busse können gestrichen werden, Taxi's nicht erreichbar sein. Als Constraints, die über der Lösung gelten müssen, ist zum einem die Einhaltung von gewissen Maximalkosten sowie einer maximalen Reisezeit gefordert.
In diesem Szenario soll nun die Reise von Ort A nach Ort B geplant werden. Man kann sich zunächst für den Bus entscheiden, was schließlich aber aufgrund des Reisezeit-Constraints wieder zurückgewiesen werden müs. Dann entscheidet man sich, einen Flug zu buchen, was (wie oben beschrieben) neue Teilziele herleitet, usw. An einem solchen Beispiel kann man leicht alle Eventualitäten, die eintreten können, sich klarmachen.

4.1.2 Backtracking und Replanning

In allgemeinen wird es nicht gelingen, sofort eine Lösung zu finden. Vielmehr wird es so sein, daß zwischenzeitlich Entscheidungen getroffen werden, die lokal zwar gut waren, aber für die Gesamtlösung des Problems nicht geeignet sind. Dies kann sich auf zwei Arten äußern:

- Es können durch die momentan gültigen Entscheidungen Constraints über den gültigen Zuweisungen verletzt werden oder
- für ein nicht reduzierten Ziel existieren keine anwendbaren Operatoren, weil die Konfliktmenge leer ist oder nur zurückgewiesene Operatoren enthält.

Solche *Inkonsistenzen* führen zu *Backtracking* und damit zum Rückzug von Entscheidungen. Gemachte Entscheidungen können also nach Belieben wieder rückgängig gemacht werden, wenn im Laufe eines Problemlösevorgangs die Notwendigkeit dazu erkannt wird. Geht man davon aus, daß beim Rückzug von Entscheidungen die aufgetretene Inkonsistenz analysiert wird und resultierend daraus eine Entscheidung zurückgezogen wird, die eine Zuweisung betrifft, welche Teil der aufgetretenen Inkonsistenz ist, so handelt es sich hier um *dependency-directed Backtracking*.

Inkonsistenzen sind aber nicht der einzige Grund, der zum Rückzug von Entscheidungen führen kann. Zwei weitere Aspekte werden bei REDUX modelliert, die nicht zu Backtracking führen, sondern dazu, daß Teile des Problems neu geplant werden müssen:

- Der erste Aspekt berücksichtigt vom System nicht manipulierbare Annahmen über die Umgebung, in der geplant wird. Dies wird bei REDUX dadurch ausgedrückt, daß für jeden Operator und damit für jede Entscheidung ihre Zulässigkeit (*admissibility*) definiert werden kann. Eine Enscheidung kann also nicht nur zurückgewiesen werden, sondern auch aufgrund anderer Gegebenheiten unzulässig werden.

  Beispiel: Ein Flug wird abgesagt wegen schlechtem Wetter. Das schlechte Wetter ist ein vom System nicht veränderbarer Aspekt, der die Zulässigkeit eines Operators 'Fliegen' einschränkt.

  Wird eine getroffene Entscheidung unzulässig aufgrund äußerer Gegebenheiten, so kann dies zur Folge haben, daß Ziele wieder unerfüllt werden, da die getroffene Entscheidung damit ungültig wurde. Hier muß für das zugehörige Ziel neu geplant werden.

- Der zweite Aspekt betrifft die *Optimalität* eines ausgewählten Operators. Ist eine Ordnung auf den Elementen einer Konfliktmenge gegeben, so definiert diese Ordnung ein lokales Optimalitätskriterium für die verschiedenen Möglichkeiten, das Ziel zu erreichen.
  Zunächst werde der folgende Kontext betrachtet: für ein Ziel sei die lokal beste Entscheidungsmöglichkeit D_{opt} zurückgezogen und hier eine andere Entscheidung D_{n_opt} getroffen worden. Wenn nun später die Notwendigkeit des Rückzugs von D_{opt} nicht mehr gegeben ist, so ergibt sich für dieses Ziel eine lokale Optimierungsmöglichkeit, nämlich statt D_{n_opt} nun doch D_{opt} zu wählen, um das Ziel zu erfüllen.
  Generell ist es nicht ratsam, in solchen Fällen eine nicht-optimale Entscheidung automatisch zurückzunehmen und die optimale Entscheidung wieder einzusetzen, da unter Umständen schon viele von der nicht-optimalen Entscheidung abhängige Entscheidungen getroffen wurden, die dann alle wieder revidiert werden müssen ([Rit92] illustriert dies ausführlicher). Wichtig ist aber, daß solche lokalen Optimierungsmöglichkeiten erkannt werden.

Vor allem der zweite Aspekt motiviert, im Hinblick auf den Rückzug von Entscheidungen zwischen

- der Notwendigkeit zum Rückzug einer Entscheidung (*rejection*) und
- dem eigentlichen Rückzug der Entscheidung (*retraction*)

zu unterscheiden. Die Rückzugsnotwendigkeit führt in jedem Fall zum Rückzug einer Entscheidung, aber eine nicht weiter bestehende Rückzugsnotwendigkeit hebt nicht automatisch den Rückzug auf. REDUX sieht hier vor, die Möglichkeit einer solchen lokalen Optimierung zunächst dem Problemlöser zu melden und die Entscheidung, wie darauf reagiert werden soll, dort zu belassen.

Beispiel: Ein ursprünglich abgesagter, aber optimaler Flug wird wieder möglich. Soll die als Ersatz gewählte Alternative wieder zurückgenommen und der optimale Flug genommen werden oder nicht? Vielleicht wurde bereits ein anderer Flug gebucht, d.h. es wäre u.U. ein nicht unerheblicher Aufwand damit verbunden, nun den optimalen Flug zu nehmen.

Backtracking und Replanning sind beides Fälle, in denen Entscheidungen zurückgenommen werden müssen, also Spezialfälle eines allgemeinen Konzepts zum Rückzug von Entscheidungen. Der Rückzug einer Entscheidung muß in jedem Fall propagiert werden, er verlangt daher Mechanismen, solche Änderungen an abhängige Assignments und Teilziele weiterzuleiten.

4.1.3 Verarbeitungsmechanismus

Problemlösen mittels REDUX besteht darin, sukzessive alle vorhandenen Ziele zu reduzieren, dabei jedoch immer Inkonsistenzen aufgrund verletzter Constraints oder blockierter Ziele sowie lokale Optimierungsmöglichkeiten zu erkennen und zu behandelen.

Dazu gibt es die sog. *Tasks*, die in vier verschiedenen Ausprägungen vorkommen und das Auftreten verschiedener Situationen signalisieren:

- *Unreduced-Goal-Task:*
  Ein Ziel wurde noch nicht reduziert oder ist aufgrund des Rückzug einer Entscheidung nicht mehr reduziert.
- *Constraint-Violation-Task:*
  Eine Constraint wurde von den aktuell gültigen Zuweisungen verletzt.
- *Goal-Block-Task:*
  Ein Ziel ist blockiert, weil kein anwendbarer Operator existiert oder alle Operatoren zurückgezogen sind.
- *Optimality-Loss-Task:*
  Es gibt die Möglichkeit einer lokalen Optimierung.

Die Realisierung der REDUX-Konzepte erzeugt die entsprechenden Tasks durch Propagierung von Änderungen, die sich aufgrund der Abarbeitung von anderen Tasks ergeben. Hat eine Operatoranwendung z.B. neue Unterziele hergeleitet, so werden durch die Propagierung der Änderungen für jedes diesere Unterziele *Unreduced-Goal-Tasks* erzeugt.

Alle Tasks werden in einer *Agenda* verwaltet, besitzen verschiedenen Prioritäten, die die Reihenfolge der Abarbeitung beeinflussen. So hat z.B. die Bearbeitung von Inkonsistenzen (Constraint-Verletzung oder Zielblockade) höhere Priorität als die Zielreduktion. Auch innerhalb eines Typs von Tasks gibt es die Möglichkeit der Anordnung dadurch, daß auf den zugrundeliegenden Zielen eine Ordnung bzgl. der Wichtigkeit ihrer Bearbeitung definiert werden kann.

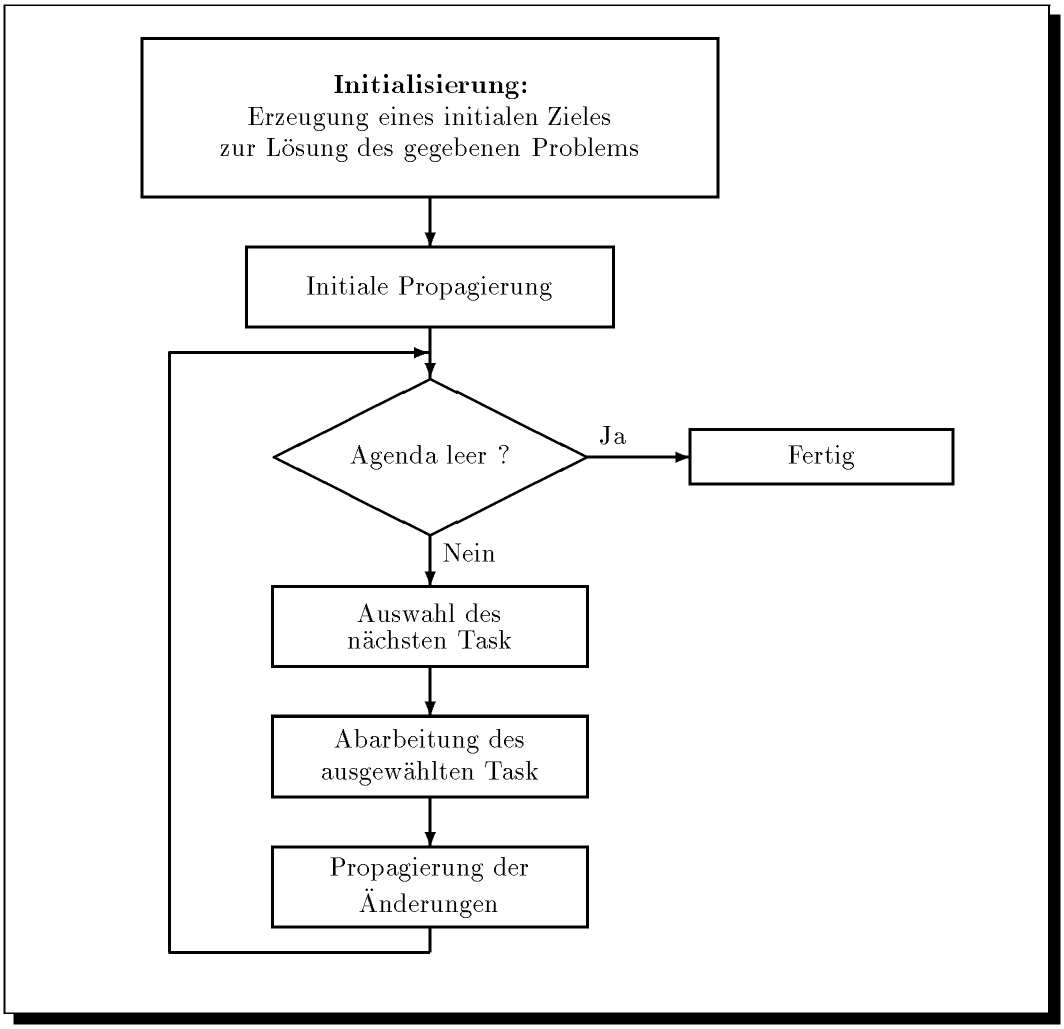

Abbildung 4.3: Problemlösealgorithmus von REDUX

Abbildung 4.3 zeigt den Problemlösealgorithmus von REDUX im Überblick. Nach einer Initialisierung werden sukzessive Tasks von der Agenda genommen, abgearbeitet und die erfolgten Änderungen werden propagiert. Dies geschieht solange, bis schließlich alle Tasks bearbeitet sind und das Problem damit gelöst ist.

Die Abarbeitung eines Tasks hat je nach Typ des Tasks verschiedene Auswirkungen:

- *Unreduced-Goal-Task:*

  Berechnung der Konfliktmenge anwendbarer Operatoren für das zugehörige Ziel, Auswahl eines Operators (Entscheidung) und Anwendung dieses Operators

- *Constraint-Violation-Task, Goal-Block-Task:*

Auflösen der entstandenen Inkonsistenz durch Rückzug einer Entscheidung (*dependency-directed Backtracking*)

- *Optimality-Loss-Task:*
  Überprüfen, ob lokale Optimierung erfolgen soll und ggf. Rückzug der nicht-optimalen Entscheidung sowie Wiedereinsetzen der lokal optimalen Entscheidung

Die Propagierung der Änderungen verursacht zum einen eine für die Realisierung von REDUX spezifische Propagierung (z.B. das Propagieren eines JTMS) zur Erzeugung der verschiedenen neuen Tasks und zum anderen eine Überprüfung der Konsistenz der aktuell gültigen Zuweisungen, die automatisch *Constraint-Violation-Tasks* für entdeckte Inkonsistenzen erzeugt.

4.1.4 Steuerung der Suche

Die Lösungssuche für ein Problem mittels REDUX kann, wie in den vorangegangenen Abschnitten bereits angeklungen ist, an zwei Stellen gesteuert werden:

1. Prioritäten der Tasks sowie eine Partialordnung auf den verschiedenen Zielen steuern die Auswahl des nächsten zu bearbeitenden Tasks.
2. Eine Partialordnung auf den verschiedenen Operatoren steuert die Auswahl eines Operators zur Zielreduktion.

An diesen beiden Stellen kann der Problemlöser eingreifen und mittels geeigneter Heuristiken verschiedene Suchstrategien realisieren.

4.2 Realisierung auf TMS-Ebene

Petrie hat vorgeschlagen, die im letzten Abschnitt beschriebenen Konzepte mittels eines *Truth Maintenance Systems* zu realisieren. Speziell hat er sich dabei auf ein JTMS [Doy79] bezogen und hat konkrete Abhängigkeitsstrukturen angegeben, die im Rahmen der hier zurgrundeliegenden Implementierung von REDUX [Rit92] realisiert wurden. Für jedes Ziel und jede Operatoranwendung (Entscheidung) werden einige Standardabhängigkeisstrukturen erzeugt, mit deren Hilfe sich die REDUX-Konzepte recht natürlich modellieren lassen. Insbesondere wird mit deren Hilfe die Erzeugung der in 4.1.3 vorgestellten Tasks gesteuert.

Diese Realisierung mittels eines JTMS soll in den folgenden Abschnitten erläutert werden. Dabei wird von grundlegenden Kenntnissen im Gebiet der Truth-Maintenance-Systeme, insbesondere dem JTMS ausgegangen (vgl. [Ric92]).

4.2.1 TMS-Knoten und TMS-Rechtfertigungen

Alle beschriebenen Abhängigkeitsstrukturen sind kleine TMS-Netze bestehend aus TMS-Knoten, die mittels ihres Labelings (IN/OUT) die Gültigkeit eines bestimmten Aspekts ausdrücken sollen, und Abhängigkeiten zwischen ihnen, ausgedrückt durch TMS-Rechtfertigungen (*Justifications*). Ein TMS-Knoten kann mehrere Justications haben. Jede TMS-Rechtfertigung besteht aus IN- und OUT-Liste, welche graphisch durch eine durchgezogene bzw. eine gestrichelte Linie kennzeichnet sind (s. Abb. 4.4).

Es gibt einen Spezialfall einer TMS-Rechtfertigung, die sog. Prämissen. Hier sind IN- und OUT-Liste leer, was zur Folge hat, daß ein mit einer Prämisse begründeter TMS-Knoten immer mit IN gelabelt wird.

Der Labeling-Mechanismus des JTMS hat die Aufgabe, für alle TMS-Knoten unter Berücksichtigung aller TMS-Rechtfertigungen ein konsistentes Labeling zu berechnen, ist also von der eigentlichen REDUX-Funktionalität völlig losgelöst als ein Mittel zu dessen Realisierung zu sehen.

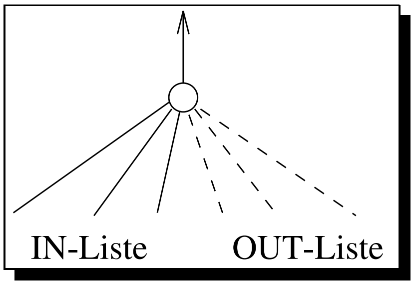

Abbildung 4.4: TMS-Rechtfertigung

4.2.2 Standardabhängigkeiten eines Zieles

Abbildung 4.5 zeigt die für jedes Ziel in REDUX erzeugte Abhängigkeitsstruktur. Jedes Ziel besitzt einen *Valid-Goal-Knoten*, welcher beschreibt, ob ein Ziel gültig ist. Dieser Knoten ist von zentraler Bedeutung und wird in anderen Abhängigkeitsstrukturen wieder auftauchen, da z.B. die Gültigkeit eines Teilziels von der Gültigkeit des dieses Teilziel einführenden Operators abhängt.

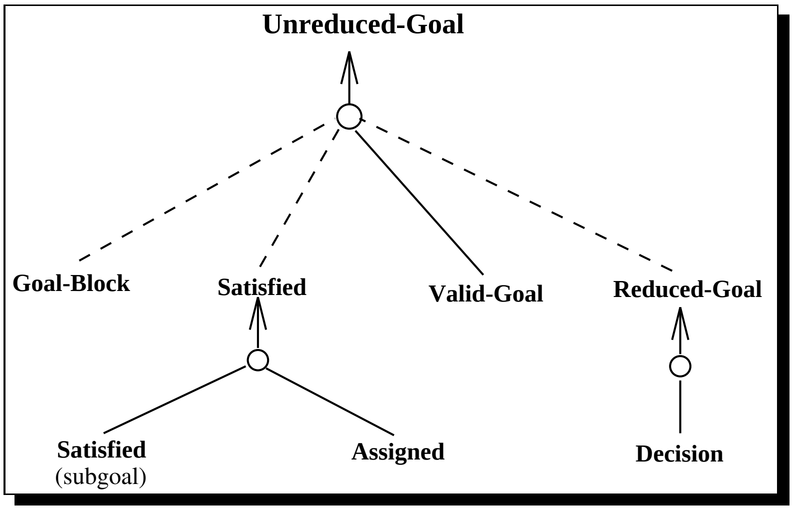

Abbildung 4.5: Abhängigkeitsstruktur eines Zieles

Der Knoten *Unreduced-Goal* repräsentiert, ob das zugehörige Ziel reduziert wurde oder nicht. Seine Rechtfertigung sagt aus, daß dies der Fall ist, falls:

- das Ziel gültig ist (*Valid-Goal-Knoten*)
- nicht bereits erfüllt ist (*Satisfied-Knoten*)
- nicht bereits reduziert ist (*Reduced-Knoten*) und
- nicht blockiert ist (*Goal-Block-Knoten*).

Wechselt der *Unreduced-Goal-Knoten* sein Labeling von OUT nach IN, so wird ein *Unreduced-Goal-Task* erzeugt, der signalisert, daß es sich hierbei um ein noch nicht reduziertes Ziel handelt.

4.2.3 Standardabhängigkeiten einer Operatoranwendung

Jedes Mal, wenn eine Entscheidung getroffen wird (Auswahl eines Operators aus einer Konfliktmenge), wird die in Abbildung 4.6 gezeigte Abhängigkeitsstruktur erzeugt.

Sie beschreibt die verschiedenen Aspekte des Zustands einer Entscheidung, wie:

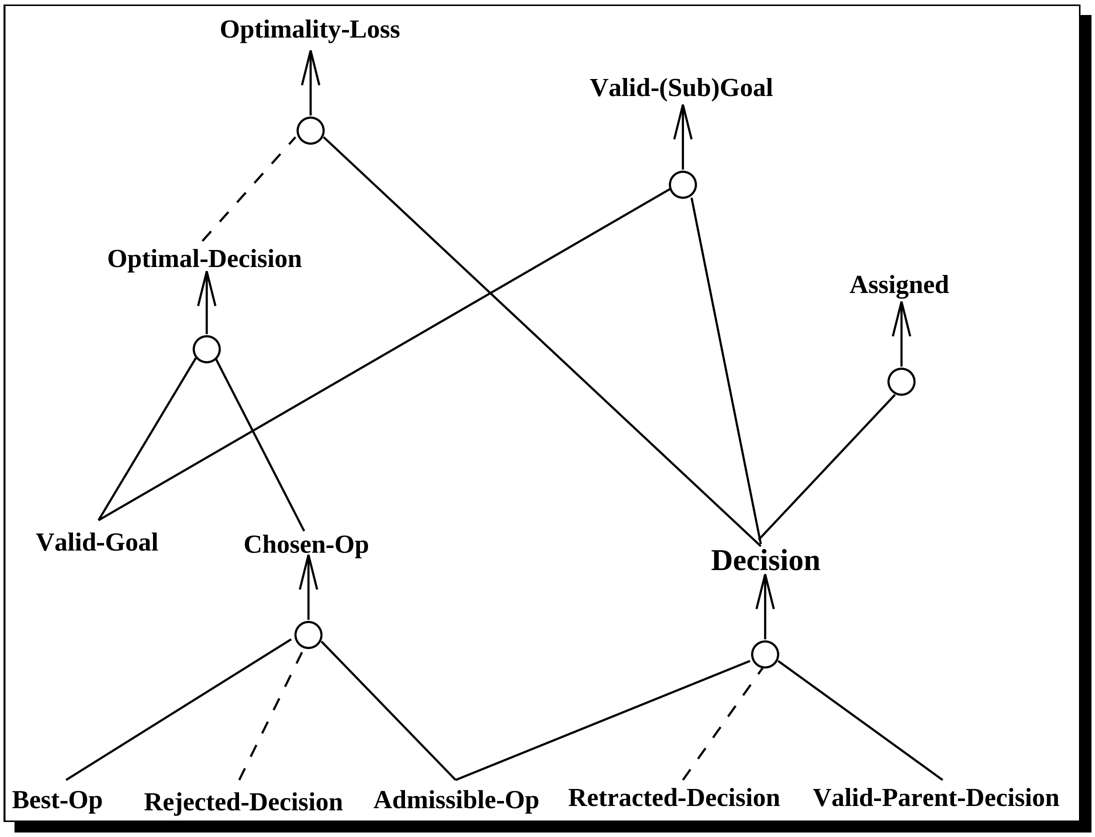

Abbildung 4.6: Abhängigkeitsstruktur einer Operatoranwendung

- Gültigkeit der Entscheidung (*Decision*)
- Rückzugsnotwendigkeit der Entscheidung (*Rejected-Decision*)
- Rückzug der Entscheidung (*Retracted-Decision*)
- Zulässigkeit des Operators (*Admissible-Op*)
- lokale Optimalität und Optimalitätsverlust

und die Beziehungen zu hergeleiteten Unterzielen und den eingeführten Zuweisungen. Jedes hergeleitete Unterziel ist nur solange gültig (*Valid-(Sub-)Goal*), wie der Operator, der es hergeleitet hat (*Decision*) und das zugehörige Ziel (*Valid-Goal*) gültig sind. Die Gültigkeit der Zuweisungen wird durch *Assigned-Knoten* ausgedrückt. Schließlich drückt der *Valid-Parent-Decision-Knoten* und seine Verwendung in der Rechtfertigung des *Decision-Knotens* aus, daß eine Entscheidung nur solange gültig sein kann, wie die Entscheidung, deren ausgewählter Operator das zugehörige Ziel hergeleitet hat.

An drei Stellen kann der Problemlöser ansetzen, indem er problemspezifische Rechtfertigungen erzeugt:

- Rechtfertigungen des *Rejected-Decision-Knoten* beschreiben Gründe für den notwendigen Rückzug einer Entscheidung aufgrund von Inkonsistenzen oder lokalen Optimierungen. Diese Rechtfertigung muß formuliert werden, falls eine Entscheidung zurückgezogen werden soll.
- Der *BestOp-Knoten* hält den Grund für die lokale Optimalität der Entscheidung fest. In der momentanen, implementierten Version von REDUX bezieht sich diese immer auf die Konfliktmenge, deren Element der Operator ist: ein Entscheidung ist lokal optimal, wenn alle lokal besseren Alternativen zurückgezogen sind. Dies setzt natürlich voraus, daß die Auswahl von Operatoren aus einer Konfliktmenge entsprechend lokale Optimalitätskriterien verwendet.

- Die Rechtfertigung des *Admissible-Op-Knotens* drückt die Anhängigkeit der Entscheidung von äußeren Gegebenheiten aus, auf die das System keinen Einfluß hat, die aber eine Entscheidung ungültig machen können.

Ein Wechsel des *Optimality-Loss-Knoten* von OUT nach IN löst einen *Optimality-Loss-Task* aus.

4.2.4 Inkonsistenzerkennung

4.2.4.1 Erkennung einer Zielblockade

Eine Zielblockade wie in Kapitel 4.1.2 beschrieben wird ebenfalls mittels eines TMS-Netzes erkannt. Für jedes Ziel wird, nachdem die Konfliktmenge berechnet ist, eine Abbhängigkeitsstruktur wie in Abbildung 4.7 aufgebaut. Diese zeigt im wesentlichen die Begründung des *Goal-Block-Knoten*.

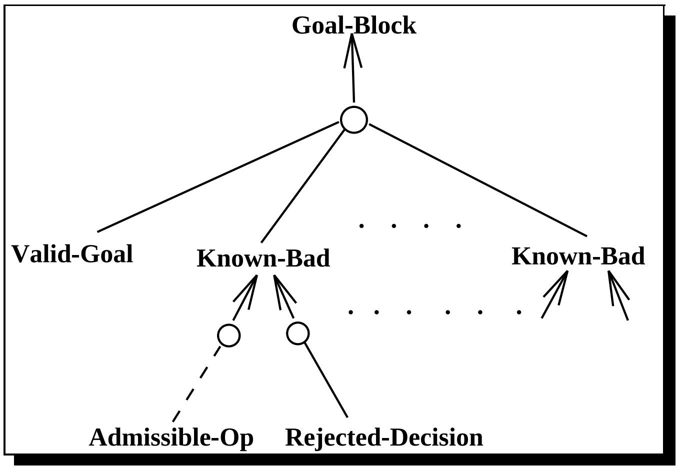

Abbildung 4.7: Abhängigkeitsstruktur für die Erkennung einer Zielblockade

Ein Ziel ist blockiert, wenn

- es noch gültig ist (*Valid-Goal*), aber
- alle Operatoren der Konfliktmenge entweder nicht zulässig oder aber zurückgezogen sind (*Known-bad*).

Wechselt dieser Knoten von OUT nach IN, so wird ein *Goal-Block-Task* ausgelöst.

4.2.4.2 Erkennung einer Constraint-Verletzung

Constraint-Verletzungen können vom System direkt festgestellt werden, so sieht es Petrie vor. Die jeweilige konkrete Anwendung prüft die Konsistenz der Constraints über den aktuellen Zuweisungen nach der Abarbeitung jedes Tasks und löst im Falle einer Inkonsistenz einen *Constraint-Violation-Task* aus.

4.3 Zusammenfassung

Wesentlich für REDUX sind die Begriffe Ziel, Operator, Konfliktmenge und Entscheidung. Eine zu lösende Aufgabe wird mittels Zielen beschrieben. Operatoren können auf Ziele angewandt werden, um diese zu reduzieren. Dabei werden u.U. neue Unterziele hergeleitet sowie Zuweisungen gemacht. Es können Konsistenzbedingungen über der Menge gültiger Zuweisungen definiert und erkannt werden. Die Menge alternativer Operatoren zur Erreichung eines Zieles heißt dessen Konfliktmenge.

Die Auswahl eines Operators aus einer Konfliktmenge heißt Entscheidung. Entscheidungen können zurückgezogen werden oder aufgrund äußerer, vom System nicht manipulierbarer Gegebenheiten unzulässig werden. Für den Rückzug können ebenso wie für die lokale Optimalität einer Entscheidung Begründungen formuliert werden.

Kapitel 5

Das Planungssystem CAPlan

Nachdem in den vorangegangenen Kapiteln zwei wesentliche Grundlagen – der SNLP-Ansatz und das REDUX-System – erläutert wurden, soll hier nun detailliert auf das im Rahmen dieser Arbeit realisierte Planungssystem CAPlan eingegangen werden.

5.1 Zielsetzungen bei der Realisierung

CAPlan ist ein domänenunabhängiges Aktionsplanungssystem, welches auf dem in Kapitel 3 erläuterten SNLP-Ansatz basiert. Dieser Ansatz ermöglicht nichtlineares Planen (vgl. Kap. 2.3.3) und stellt die grundlegende Kontrolleinheit **CLPC**[1] des Systems dar.

Das System wurde so konzipiert, daß die Kontrolle der Lösungssuche von verschiedenen Kontrollkomponenten übernommen werden kann. In der Kontrollkomponente **CLPC** wurde der SNLP-Algorithmus aus Kapitel 3.3.3 realisiert und wie in Kapitel 3.4 beschrieben durch das Erkennen und Beheben von Teilzielrekursionen und den Einsatz von dependency-directed Backtracking verbessert.

Das System hat neben dieser systematisch vorgehenden Kontrollkomponente auch eine interaktive Kontrollkomponente **UC**[2], die einen hohen Grad an Benutzerinteraktion zuläßt: im Extremfall kann jede Entscheidung vom Benutzer getroffen werden und dieser kann jeder Zeit eingreifen und von anderen Kontrollkomponeneten ausgewählte Alternativen zurückweisen. Dabei nimmt das System jedoch nur eine minimale Menge von Änderungen an der schon existierenden partiellen Lösung vor. Zwingend notwendig dafür ist die explizite Repräsentation von Abhängigkeiten zwischen Zielen, Operatoranwendungen und ihren Auswirkungen.

Die Realisierung weiterer Kontrollkomponenten für das System ist geplant und wurde berücksichtigt, z.B. die Integration einer fallbasiert arbeitenden Komponenten **CbC**[3], die die Planung mittels Fallbeispielen steuert, oder einer Komponente, die mit Kontrollregeln arbeitet. Das ist aber nicht Gegenstand dieser Arbeit.

Die Realisierung des Systems auf Basis von REDUX bot sich an, da gerade die explizite Repräsentation von Abhängigkeiten zwischen Entscheidungen und alle grundlegenden Erfordernisse für dependency-directed Backtracking Bestandteil von REDUX sind.

5.2 Systemübersicht

Abbildung 5.1 zeigt das gesamte CAPlan-System schematisch im Überblick. Es besteht aus einem Teil, der den Kern des Systems darstellt und dazu das REDUX-System verwendet und erweitert,

[1]CLPC: Causal Link Planner Control
[2]UC: User Control
[3]CbC: Case-based Control

sowie einem Teil, der die vielfältigen Benutzereingaben und -interaktionen koordiniert, an das System weitergibt und Zwischen- und Endergebnisse der Planung visualisiert.

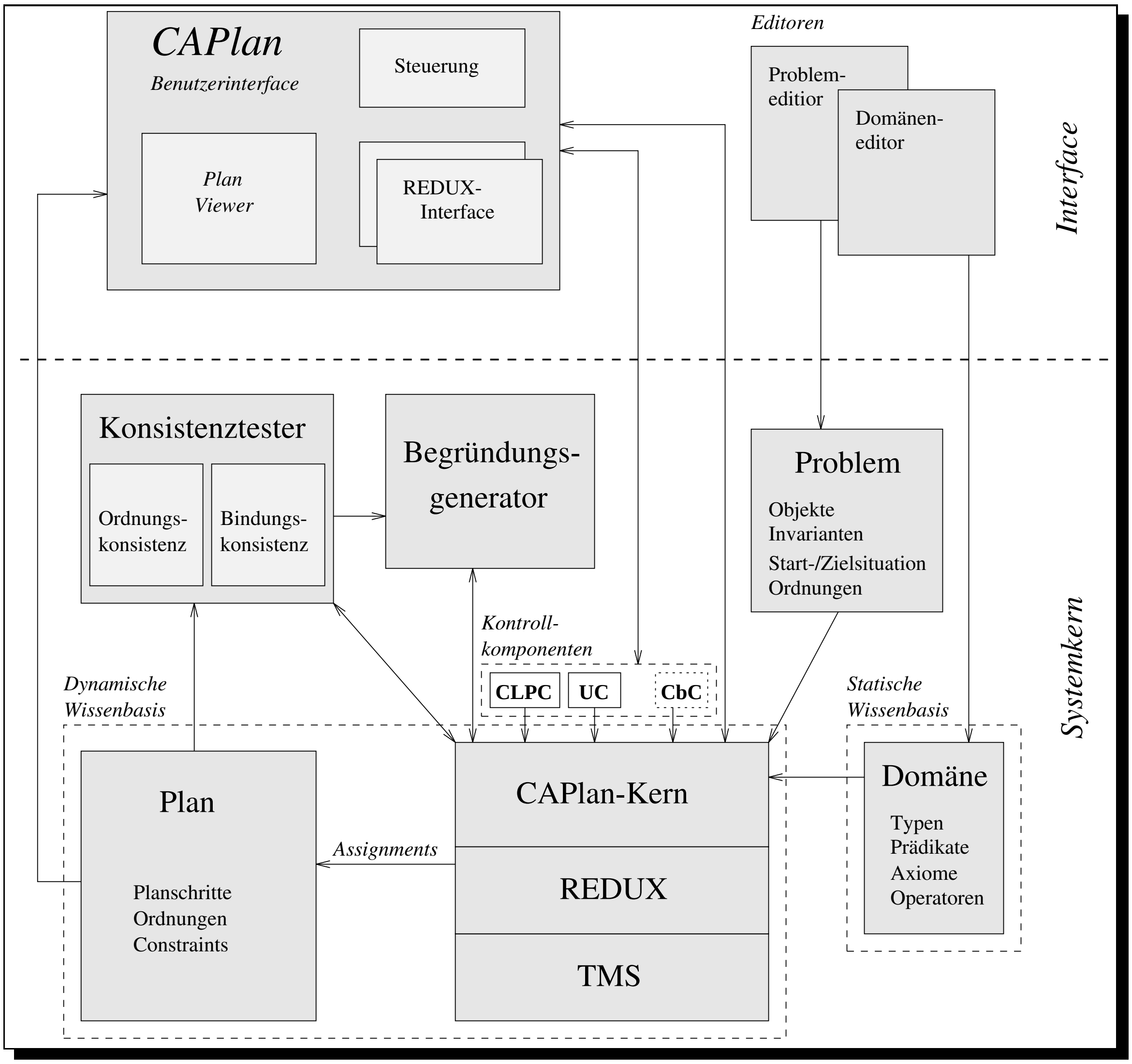

Abbildung 5.1: Architektur von CAPlan

Der Systemkern besteht aus folgenden Komponenten:

- *Statische Wissensbasis:*

  Das System ist domänenunabhängig und verwendet eine statische Wissenbasis, um Informationen über die Planungsdomäne zu speichern. Sie bleibt während eines Planungsprozesses unverändert und enthält die Objekttypen, Prädikatssymbole sowie die in der Planungswelt möglichen Aktionen (vgl. Kap. 5.3).

- *Dynamische Wissensbasis:*

  Sie besteht hauptsächlich aus einer gezielten Erweiterung des REDUX-Systems in Form des CAPlan-Kerns, welcher die Lösung von Planungsproblemen unter Verwendung der statischen Wissensbasis ermöglicht. Dies geschieht unter der Kontrolle einer der möglichen Kontrollkomponenten, wobei während der Lösung eines Planungsproblems beliebig zwischen verschiedenen Kontrollkomponenten gewechselt werden kann. Ein wichtiger Teil der dynamischen Wissesbasis ist der erzeugte Plan, der im Sinne von REDUX als eine Menge von Zuweisungen zu verstehen ist.

- *Kontrollkomponenten:*

  Im Rahmen dieser Arbeit wurde das Hauptaugenmerk auf die Realisierung von zwei Kontrollkomponenten gelegt:

  - **CLPC (Causal Link Planner Control):** Die Kontrolle des Problemlösevorgangs erfolgt so, wie der in Kapitel 3.3.3 beschriebene Algorithmus zur systematischen, nichtlinearen Planung es vorsieht. Auch die in Kapitel 3.4 beschriebenen Verbesserungen sind in dieser Kontrollkomponente angesiedelt.
  - **UC (User Control):** Der Benutzer hat die volle Kontrolle beim Problemlösevorgang; er entscheidet an allen Stellen, an denen es mehrere Entscheidungsalternativen gibt.

  Diese beiden Kontrollkomponenten können beliebig verzahnt eingesetzt werden, da es für das System aufgrund einiger festgelegter Protokolle keinen Unterschied macht, welche Kontrollkomponente Entscheidungen trifft.

- *Konsistenztester:*

  REDUX sieht vor, daß eine (Teil-)Lösung (Menge von Zuweisungen) bestimmte zu definierende Bedingungen (Constraints) erfüllt. Dies bezieht sich hier auf die Konsistenz eines Planes, d.h. Ordnungskonsistenz und Bindungskonsistenz. REDUX verwendet den Konsistenztester, um zu überprüfen, ob eine bestimmte Menge von Zuweisungen konsistent ist und um im Falle von Inkonsistenzen Rückzugsbegründungen generieren zu lassen.

- *Begründungsgenerator:*

  Spezifisch für die Realisierung der CLPC-Kontrollkomponente, welche den SNLP-Ansatz realisiert, ist, daß mittels der REDUX-Konzepte der Algorithmus Causal_Link_Planner realisiert wird. Sowohl für die einfache Version mit chronologischem Backtracking als auch für die Erweiterung zum dependency-directed Backtracking muß das System im Falle, daß Backtracking durchgeführt werden muß, Entscheidungen zurückziehen. Dazu müssen Rückzugsbegründungen (Begründungen für die Rückzugsnotwendigkeit eines Operators, s. Kap. 4.1.2) generiert werden. Der Begründungsgenerator arbeitet mit dem Konsistenztester zusammen und kann aus einer Analyse einer Inkonsistenz Begründungen generieren, die genau die einmal erkannte Inkonsistenz ohne einen erneuten Konsistenztest wiedererkennen (d.h. die Begründungen sind vollständig, sie umfassen genau diese Inkonsistenz).

Das Benutzerinterface besteht aus folgenden Komponenten:

- dem eigentlichen CAPlan-Benutzerinterface (s. Abb. 5.2); in diesem sind REDUX-spezifische Aspekte wie die Taskagenda (oben links) und die Teilzielhierarchie (oben rechts), aber auch der momentane Plan (unten) zu sehen sind und von dort aus ist die Kontrollkomponente **UC** für die Steuerung durch den Benutzer zugänglich.
- Editoren für die Domänendefinition sowie für Problembeschreibungen
- Visualisierungsmöglichkeiten für die durch REDUX erzeugten TMS-Strukturen sowie insbesondere für die automatisch generierten TMS-Rechtfertigungen

5.3 Modellierung von Planungsdomänen und Planungsproblemen

5.3.1 Domänenmodellierung

Das CAPlan-System ist ein domänenunabhängiges Planungssystem, das zur Domänenmodellierung eine STRIPS-ähnliche Notation verwendet. Sie besteht aus folgenden Komponenten: (s. Abb. 5.3)

- *Objekttypen:*

  Objektklassen zur Definition von Planungsprobleme (z.B. `Block, Table`)

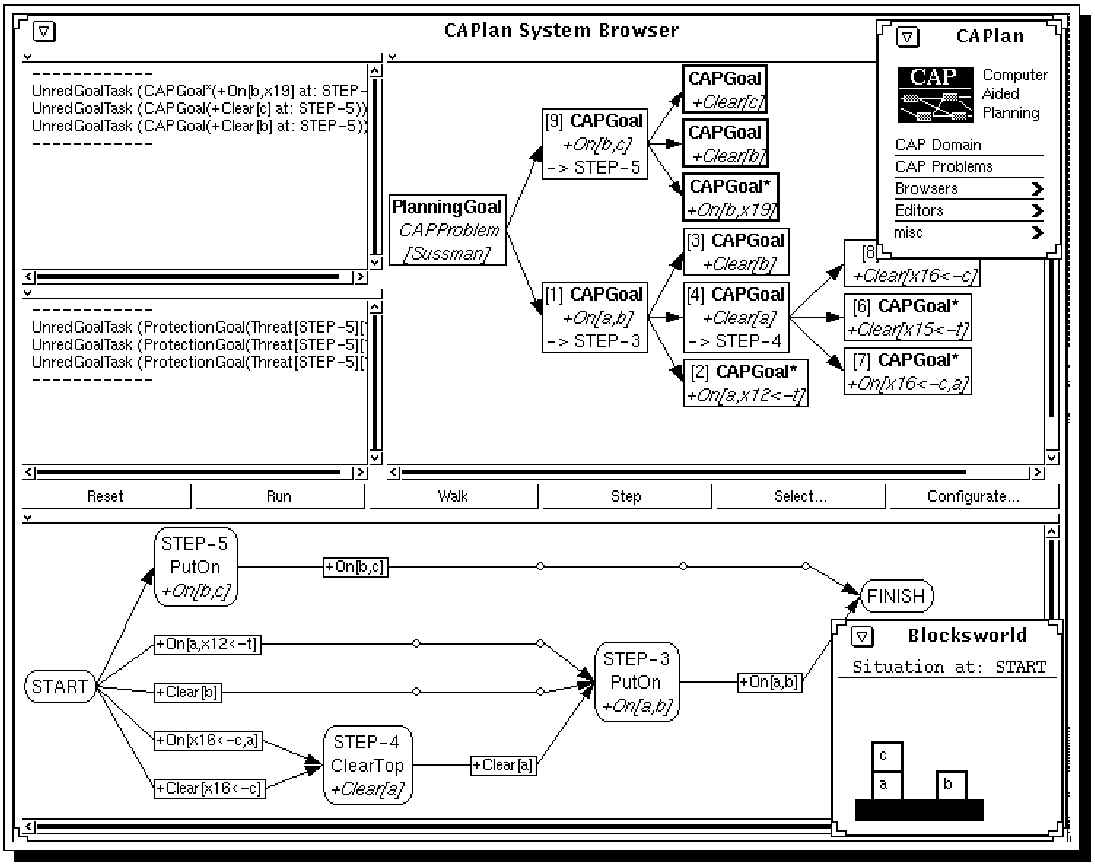

Abbildung 5.2: Benutzerinterface von CAPlan

- *Prädikate:*

  Prädikatssymbole zur Beschreibung von Situationen in der Planungswelt sowie Effekten und Vorbedingungen von Operatoren (z.B. `+On(a,b)`, `-Clear(c)`)

- *Axiome:*

  Regeln der Form *wenn <aussage1> dann <aussage2 oder aussage3 oder ..>*, die Abhängigkeiten der Prädikatssymbole ausdrücken (z.B. wenn `+Clear(x)` dann `-On(y,x)`)

- *Operatoren:*

  Ein Domänenoperator besteht aus Vorbedingungen und Effekten sowie den durch ihn eingeführten Constraints über den von ihm verwendeten Variablen.

Zur Steuerung der Operatorauswahl wird wie bei [Wil84] zwischen Haupt- und Seiteneffekten unterschieden. Dabei geht man von der Annahme aus, daß die Haupteffekte primär dazu dienen sollen, ein Ziel zu erfüllen. Haupteffekte (*Purposes*) werden im CAPlan-Sytem bei der Operatorauswahl bevorzugt[4].

Bedeutung der Axiome bei der Domänendefinition

Axiome sind eine Menge von Regeln, mit deren Hilfe Zusammenhänge der zur Beschreibung der Planungswelt verwendeten Prädikatssymbole ausgedrückt werden können. Sie finden derzeit ausschließlich Verwendung bei der Defintion der Operatoren einer Domäne und werden an zwei Stellen eingesetzt:

[4] Die ausschließliche Berücksichtigung der Haupteffekte zur Auswahl von Operatoren ist im Zusammenhang mit dem SNLP-Ansatz falsch, da die Vollständigkeit des Algorithmus dadurch verloren geht (vgl. Kap. 6)

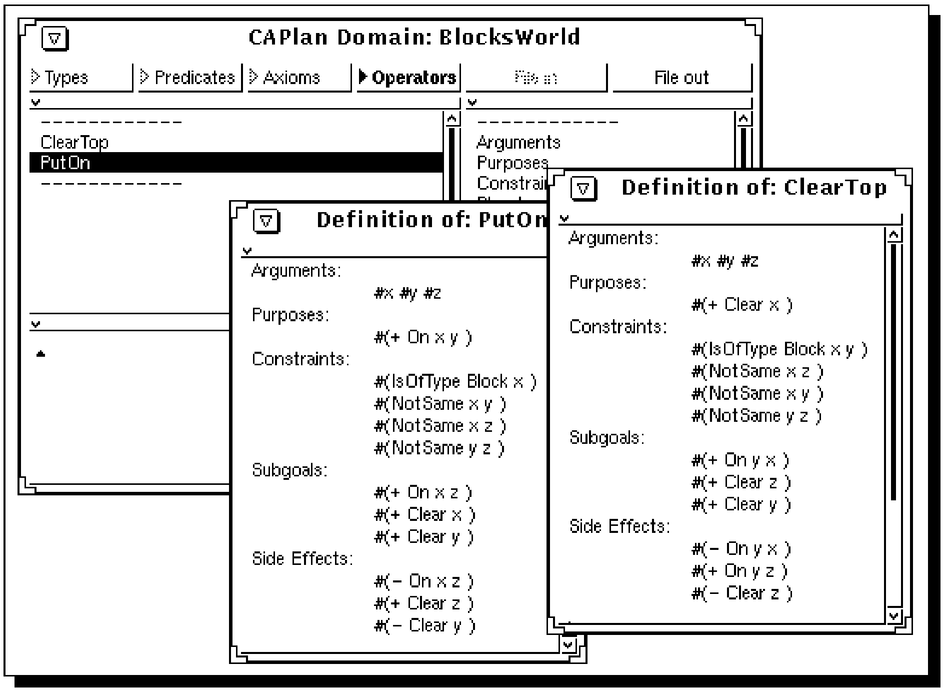

Abbildung 5.3: Definition einer Domäne

- Mit Hilfe der Axiome wird die korrekte Definition der Effekte von Operatoren erleichtert, indem das System Effekte vorschlägt, die sich aus den bereits bekannten Effekten und Vorbedinungen mittels der Axiome folgern lassen. Wichtig ist aber, das die Herleitung von Effekten nur während der Defintion eines Operators verwendet wird.

  Beispiel: Man betrachte den PutOn-Operator aus der Blocksworld aus Abbildung 5.3. Dieser Operator hat u.a. den Effekt `+On(x,y)`, der auch Hauptzweck dieses Operators ist. Mit dem Axiom *wenn +On(x,y) dann -Clear(y)* läßt sich daraus folgern, daß der Operator auch den Effekt `-Clear(y)` haben muß, wenn er `+Clear(y)` als Vorbedingung hat.

- Aus den Effekten lassen sich zusätzliche Informationen über die Vorbedingungen eines Operators herleiten, die die Lösungssuche wesentlich beschleunigen. Es kann damit zwischen zwei Arten von Vorbedingungen unterschieden werden:

  1. Vorbedingungen, deren Gültigkeit durch existierende Schritte oder u.U. erst durch Anwendung eines Domänenoperators garantiert werden kann,
  2. Vorbedingungen, deren Gültigkeit vorausgesetzt werden kann, wenn der zu einer solchen Vorbedingung gehörige Operator anwendbar ist, d.h. solche Vorbedingung repräsentieren eine Anwendbarkeitsbedingung.

  Beispiel: Man betrachte wieder den PutOn-Operator aus Abbildung 5.3. Ausgehend von der Überlegung, daß dieser Operator nur dann angewandt wird, wenn sein Haupteffekt `+On(x,y)` noch nicht gilt, kann man aus der Negation dieses Effektes mittels des Axioms *wenn -On(x,y) dann +On(x,z)*[5] schließen, daß vor der Anwendung des Operators `+On(x,z)` gelten muß. D.h. die Vorbedingung `+On(x,z)` muß durch Ausnutzen eines Effektes eines anderes Schrittes erreicht werden können (Phantomisierung), andernfalls ist der Operator nicht anwendbar.

  Worauf es ankommt ist, daß einerseits Vorbedingungen des zweiten Typs nur dazu da sind, Variablen an konkrete Objekte zu binden[6] (im Beispiel des PutOn-Operators die Variable `z`)

[5] Bei der Definition dieses Axioms wird $y \neq z$ vorausgesetzt.
[6] An anderen Stelle (NONLIN, etc.) wird diese Variablenbindung in der Regel beim Testen einer Menge von Anwendbarkeitsbedingungen durchgeführt.

und sich daß solche Vorbedingungen anderseits mittels der Axiome identifizieren lassen.

Beachtet man diesen Unterschied nicht, so ergeben keine grundsätzlichen Probleme für die Lösungssuche, jedoch ist der Suchraum größer, da auch bei einer Vorbedingung, die nur eine Variable binden soll, Domänenoperatoren statt nur Phantomisierungen getestet werden.

Variablenbindungs-Constraints

In Kapitel 2.2.3 wurde bereits die Verwendung von Constraints und Variablen zur Definition von Operator-Schemata besprochen. Im CAPlan-System gehört zu einer Domänenbeschreibung auch wie schon beschrieben eine Reihe von Objekttypen und es können für jede verwendete Variable Aussagen über den Typ gemacht werden, von dem die möglichen Bindungen sein sollen. Dazu gibt es neben den bereits erwähnten Gleichheits- und Ungleichheits-Constraints (`Same/NotSame`) noch Typ-Constraints, die eine Variablenbindung mit einem Objekt eines bestimmten Typs fordern oder verbieten:

- `IsOfType(TYPE,x)` erzwingt für die Variable `x` eine Bindung mit einem Objekt des Typs `TYPE`
- `IsNotOfType(TYPE,x)` verbietet für die Variable `x` eine Bindung mit einem Objekt des Typs `TYPE` dieselbe Bindung

5.3.2 Definition von Planungsproblemen

Nachdem eine Domäne modelliert ist, können Planungsprobleme in dieser Domäne definiert werden. Ein Planungsproblem wird durch folgenden Aspekte beschrieben:

- *Objekte:*
  Gegenstände des Planungsproblems
- *Invarianten:*
  Aussagen, die während des gesamten Planungsprozesses gültig sind
- *Start-/Zielsituation:*
  Beschreibung der Start-/Zielsituation mit Hilfe der Prädikate der Domäne
- *Ordnungen auf Zielen:*
  Menge von initial vorgegeben Ordnungen auf den Zielprädikaten

Abbildung 5.4 zeigt die Definition der Sussman-Anomalie aus der Blocksworld mittels des Editors zur Definition von Planungsproblemen. Durch die Domänendefinition wurden die beiden Prädikatssymbole `Clear` und `On` festgelegt, sowie die Objekttypen `Block` und `Table`.

Zur Verwendung von Invarianten bei der Problemdefinition soll folgendes festgestellt werden: Invarianten sind Aussagen, die während des gesamten Planungsprozesses gelten und von keinem Operator ungültig gemacht werden können. Grundsätzlich könnten sie auch zu der Definition einer Startsituation hinzugenommen werden, wobei aber sicherzustellen wäre, daß sie nicht in der Delete-Liste von Operatoren auftauchen. Hier werden solche Invarianten explizit definiert, weil dies den Vorteil hat, daß das Planungssystem diese zusätzliche Information bei der Erfüllung von Zielen dann ebenfalls explizit auszunutzen kann (vgl. Kap. 5.4.4).

Ordnungen auf den Zielprädikaten sind prinzipiell ebenfalls nicht nötig. Jedoch hat man durch sie die Möglichkeit, bereits bei der Problemdefinition Wissen über die Abarbeitungsreihenfolge dem Planungssystem zugänglich zu machen. Diese Ordnungen haben einen entsprechende Abarbeitungsreihenfolge zur Folge, und erhöhen damit die Effizienz.

Beispiel: Reiseplanung, die Teilziele seien 1. Flug buchen, 2. Unterkunft buchen; hier ist es sinnvoll Teilziel 1 vor Teilziel 2 zu bearbeiten, da im Falle des Scheiterns von Teilziel 1 das andere Teilziel erst gar nicht bearbeitet werden muß.

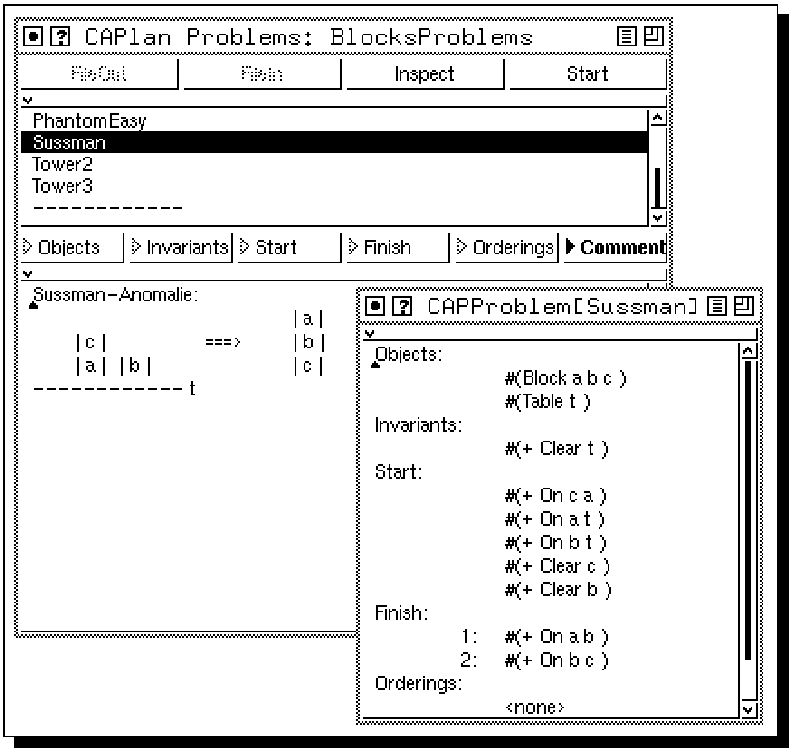

Abbildung 5.4: Definition von Planungsproblemen

5.4 Abbildung der SNLP-Begriffswelt auf REDUX-Konzepte

Die Realisierung des SNLP-Ansatzes erfordert, daß die verschiedenen dafür wichtigen Begriffe auf entsprechende Konzepte von REDUX abgebildet werden. Dies betrifft zum einen die Repräsentation von Plänen und ihren Bestandteilen, zum anderen müssen die für den Algorithmus wichtigen Begriffe wie Operatoren, Vorbedingungen und der Aspekt des Backtrackings in geeigneter Weise auf REDUX übertragen werden.

5.4.1 Backtracking-Punkte

Ein wesentlicher Aspekt bei der Realisierung des SNLP-Ansatzes mittels REDUX ist, daß alle Backtracking-Punkte auf die Abarbeitung von Zielen abgebildet werden müssen, da in REDUX die Konfliktmenge von Alternativen, um ein Ziel zu erfüllen, diese Backtracking-Punkte repräsentieren. Daraus ergibt sich, daß

- die Erzeugung eines Causal Link für eine offene Vorbedingung,
- das Schützen eines Causal Link und
- das Beheben von Teilzielrekursionen

bei der Realisierung REDUX zu Zielen werden, auf die Operatoren angewandt werden. Backtracking geschieht dadurch, daß Entscheidungen zurückgezogen werden.

5.4.2 Repräsentation von Plänen

Wie bereits in Kapitel 4 ausführlicher besprochen, können Operatoren bei REDUX neue Unterziele herleiten und/oder Zuweisungen machen; die Menge aktuell gültiger Zuweisungen beschreibt die Lösung, hier also den gesuchten Plan.

Daraus ergibt sich für den SNLP-Ansatz, daß ein Plan und die für ihn charakteristischen Konzepte wie Planschritt, Causal Links, zusätzliche Ordnungen und Constraints über Variablen, die ein Schritt einführt, zu solchen Zuweisungen eines Operators werden. Die Gültigkeit einer Zuweisung hängt dann, wie bereits erläutert, von der Gültigkeit der zugrundeliegenden Entscheidung und dem zugehörigen Ziel ab.

5.4.3 Ziele

Die Notwendigkeit, alle Backtracking-Punkte in REDUX zu Zielen zu machen, resultiert in der in Abbildung 5.5 gezeigte Hierarchie von Zieltypen. Die Namensgebung ist an der Impementierung orientiert und macht in dieser Hinsicht eine Zuordnung leicht. Die Tatsache, daß es sich um eine Hierarchie handelt, ist auf die objektorientierte Implementierung dieser Konzepte in Form von Klassen zurückzuführen, d.h. die einzelnen Zieltypen sind im System auf Klassen abgebildet und CAPAbstractGoal realisiert als gemeinsame Oberklasse die Basisfunktionalität für das Zusammenspiel mit den unterschiedlichen Kontrollkomponenten.

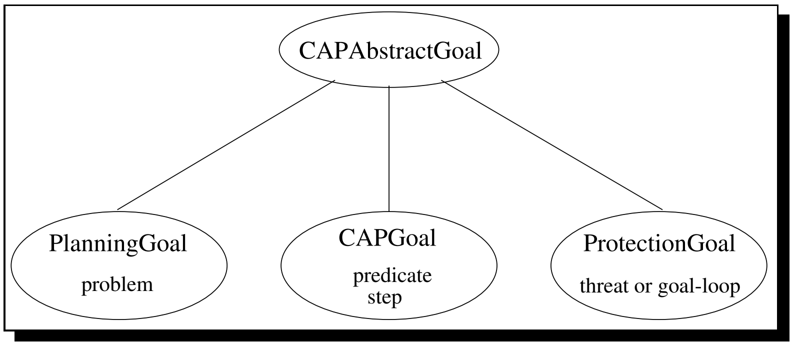

Abbildung 5.5: REDUX-Ziele zur Realisierung von SNLP

Im einzelnen lassen sich folgende Arten von Zielen unterscheiden:

PlanningGoal: Dies ist das im Sinne von REDUX initiale Ziel zur Lösung eines Planungsproblems *problem*. Ein Planungsproblem besteht dabei aus den in Kapitel 5.3.2 definierten Komponenten.

CAPGoal: Dieser Zieltyp hat immer einen Parameter *predicate*, der für eine offene Vorbedingung eines Schrittes *step* steht. Er repräsentiert das Ziel, für die Vorbedingung *predicate* eines Schrittes *step* einen Causal Link zu erzeugen.

ProtectionGoal: Solche Ziele werden einerseits erzeugt, wenn festgestellt wird, daß für einen Causal Link eine Bedrohung (Threat) existiert, gegen den ersterer zu schützen ist (*Threat-Protection*).

Eine zweite Möglichkeit, die zur Erzeugung eines ProtectionGoals führt, ist die Erkennung einer Teilzielrekursion, da diese wie in Kapitel 3.4 beschrieben im Prinzip wie Threats von Causal Links behandelt werden können (*Goal-Loop-Protection*).

5.4.4 Operatoren

Zur Bearbeitung der verschiedenen Ziele werden entsprechende Operatoren benötigt. Abbildung 5.6 zeigt die Hierarchie von Operator-Typen. Auch hier realisiert die gemeinsame Oberklasse CAPOperator in der Implementierung die gemeinsame Funktionalität für das Zusammenspiel mit den Kontrollkomponenten.

Die einzelnen Operatortypen sind:

DecomposeProblemOp: Dieser Operator ist auf Ziele des Typs PlanningGoal anwendbar und hat die Aufgabe, das System für die Lösung eines Planungsproblems zu initialisieren. Dies besteht darin, den Plan zu initialisieren und die Zielstellung durch Ziele in REDUX auszudrücken.

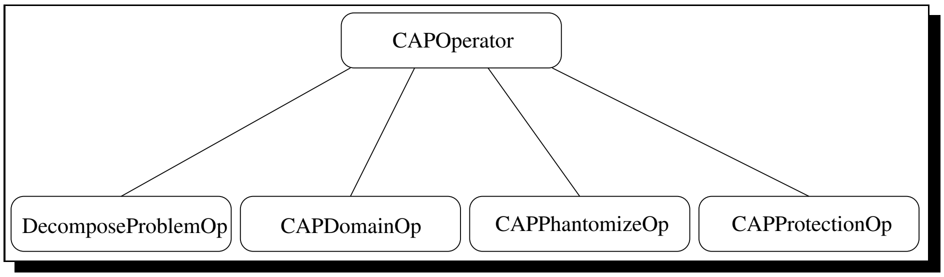

Abbildung 5.6: REDUX-Operatoren zur Realisierung von SNLP

Konkret heißt das, daß die Anwendung des Operators die folgende Auswirkungen[7] hat:

- Es werden die beiden Planschritte START und FINISH erzeugt, deren Aufgabe bereits in Kapitel 3.1.1 beschrieben wurde. Die initiale Ordnung *START*$\prec$*FINISH* braucht nicht explizit in den Plan aufgenommen zu werden, da die Definition der Konsistenz eines Planes (Kap. 3.3.1) dies bereits impliziert.
- Der Operator leitet für jede Vorbedingung des Schrittes FINISH eines neues Teilziel (CAP-Goal) her, dessen Abarbeitung dies gültig machen soll.

CAPDomainOp: Diese Art von Operatoren repräsentieren domänenspezifischen Operatoren im CAPlan-Kern. Sie sind grundsätzlich nur auf Ziele vom Typ CAPGoal anwendbar. Das zugrundeliegende Operator-Schema aus der Domäne definiert alle Details wie die Liste von Effekten, Vorbedingungen und Constraints. Die Anwendbarkeit einer konkreten Operatorinstanz ist dann gegeben, wenn es einen Effekt gibt, der (u.U. erst mittels einiger neuer Variablenbindungen) gleich dem vom gegebenen CAPGoal zu erfüllenden *predicate* ist.

Auswirkungen der Anwendung eines solchen Operators sind:

- Es wird ein neuer Planschritt s_i eingeführt, der den Domänenoperator im Plan repräsentiert, also auch dessen Effekte, Vorbedingungen und Constraints erhält.
- Es wird ein Causal Link $s_i \xrightarrow{P} s_j$ eingeführt, der die Vorbedingung P des mit dem zugehörigen Ziel assoziierten Schrittes s_j gültig macht.
- Es werden durch den neuen Schritt u.U. neue Variablen eingeführt, über denen Constraints spezifiziert sind. Diese werden ebenfalls zu Zuweisungen des Operators.
- Der Operator leitet für jede Vorbedingung des eingeführten Schrittes s_i ein neues Teilziel vom Typ CAPGoal her.

CAPPhantomizeOp: Der Operator unterscheidet sich vom vorigen hauptsächlich dadurch, daß er keinen neuen Schritt einführt, sondern eine bereits existierende Quelle für die Erfüllung der Vorbedingung P ausnutzt (*Establisher*). Als solche Quelle kommt entweder ein anderer Schritt oder eine Invariante der Problemdefinition in Betracht.

Auswirkungen der Anwendung eines Phantomisierungsoperators sind:

- Falls es sich bei der Quelle, die verwendet wird, um einen Schritt handelt, wird ein Causal Link $s_i \xrightarrow{P} s_j$ eingeführt. Falls es sich um eine Invariante handelt, ist dies nicht nötig, da Invarianten immer gelten und daher nicht durch einen Causal Link garantiert werden müssen.
- Damit der Establisher zu einem solchen wird, müssen evt. Constraints eingeführt werden, welche wieder zu Zuweisungen des Operators werden.
- Es werden keine neuen Teilziele hergeleitet.

[7] Hier und bei den folgenden Operatorbeschreibungen sind mit Auswirkungen sowohl die von den Operatoren im Sinne von REDUX gemachten Zuweisungen als auch die hergeleiteten neuen Teilziele gemeint.

CAPProtectionOp: Dieser Operator wird auf ein ProtectionGoal angewandt und ist in der Lage, eine erkannte Bedrohung für einen Causal Link oder eine Teilzielrekursion aufzulösen. Grundsätzlich kann dies geschehen, indem eine Ordnung oder eine Reihe von Constraints eingeführt werden, welche wieder die Zuweisungen des Operators werden.

Abbildung 5.7 faßt nochmal zusammen, welche Operatoren auf welche Ziele anwendbar sind und welche Zuweisungen sie machen.

Operator	anwendbar auf	Zuweisungen	neue Teilziele
Decompose-ProblemOp	PlanningGoal	• START, FINISH	Vorbedingungen von FINISH
CAP-Domain-Op	CAPGoal	• Schritt • Causal Link • Constraints	Vorbedingungen des neues Schrittes
CAP-Phantomize-Op	CAPGoal	• evt. Causal Link • Constraints	keine
CAP-ProtectionOp	CAPProtectionGoal	• Ordnung oder • Constraints	keine

Abbildung 5.7: Operatoren, Ziele und Zuweisungen im Überblick

5.4.5 Prioritäten der Ziele und Operatoren

Die in den vorangegangenen Abschnitten beschriebene Übertragung von SNLP-Begriffen auf REDUX-Konzepte alleine realisiert den SNLP-Ansatz noch nicht. Es fehlt noch die Definition der Kontrolle der Suche.

REDUX ermöglicht die Definition einer partiellen Ordnung über den Zielen und den Operatoren als Weg, die Lösungssuche zu steuern. Die Realisierung des SNLP-Ansatzes erfordert lediglich eine Ordnung, die die verschiedenen Typen von Zielen (ProtectionGoals und CAPGoals) in der folgenden Reihenfolge behandeln:

1. ProtectionGoals zur Auflösung von Teilzielrekursionen

2. ProtectionGoals zum Schützen von Causal Links

3. CAPGoals zur Erzeugung von Causal Links für offene Vorbedingungen

Innerhalb der CAPGoals wurde eine weitere Ordnung verwendet, die solche Ziele, die nur zur Bindung von Variablen dienen (vgl. Kap. 5.3.1) bevorzugt, sofern eine konsistente Bindung gefunden werden kann. Dies ist im Gegensatz zu den oben beschriebenen Prioritäten nur eine Heuristik, die sich jedoch als sehr brauchbar erwiesen hat, da Variablen mit fester Bindung in der Regel für die danach bearbeiteten Ziele die Menge der konsistenten Alternativen reduzieren.

Ebenfalls als Heuristiken zu verstehen sind die folgenden Ordnungen über verschiedenen Operatoren, die bei der Auswahl von Operatoren aus einer Konfliktmenge betrachtet werden:

- Ein CAPPhantomizeOp ist einem CAPDomainOp bei der Zielreduktion vorzuziehen, da ersterer den Plan nicht um einen weiteren Schritt vergrößert.
- Auflösung von Interaktionen sollte bevorzugt durch das Einführen von Constraints anstatt durch zusätzliche Ordnungen erreicht werden, da zusätzliche Ordnungen u.U. die Möglichkeiten für die Phantomisierung von Zielen verringern.

5.5 Änderung von REDUX im CAPlan-Kern

Die Realisierung des SNLP-Ansatzes gelang fast ohne jegliche Änderung von REDUX-Konzepten. Eine erste im CAPlan-Kern durchgeführte Änderung vereinfacht lediglich die Behandlung von Inkonsistenzen in einer Weise, daß eine für SNLP grundsätzlich zu komplexe Behandlungsmöglichkeit von Constraint-Verletzungen auf den notwendigen, spezielleren Teil reduziert wird. Damit konnte die Abarbeitungsgeschwindigkeit gesteigert werden.

Der Mechanismus zum Berechnen einer Konfliktmenge von alternativen Operatoren wurde gegenüber REDUX um die Möglichkeit erweitert, eine solche noch vergrößern zu können, nachdem sie bereits berechnet wurde. An zwei anderen Stellen mußte geringfügig in die zur Realisierung von REDUX verwendete TMS-Abhängigkeitsstruktur eingegriffen werden, um zusätzliche Aspekte von Zielen zum Schützen eines Causal Links (ProtectionGoal) ausdrücken zu können.

Bei systematischer Suche können hier keine Aussagen über die lokale Optimalität einer Entscheidung gemacht werden wie es REDUX vorsieht. Daher mußte die Generierung der TMS-Rechtfertigungen dafür modifiziert werden.

5.5.1 Behandlung von Constraint-Verletzungen

Im hier beschriebenen Planungssystem wurde die Inkonsistenzbehandlung aufgrund von Constraint-Verletzung anders realisiert als von Petrie ursprünglich intendiert und im Kapitel 4 beschrieben. Dort wird im Falle, daß durch eine Operatorauswahl eine Constraint-Verletzung auftritt, ein *Constraint-Violation-Task* erzeugt, dessen Abarbeitung die Constraint-Verletzung wieder beheben soll. Für die Realisierung des SNLP-Ansatzes würde dies bedeuten, daß, sobald ein Operator Zuweisungen macht, die zur einer Inkonsistenz im Plan führen, die Abarbeitung des dadurch erzeugten *Constraint-Violation-Task* genau diesen offensichtlich inkonsistenten Operator wieder zurückziehen würde. Die in REDUX durch diesen speziellen Task-Typ ermöglichte, viel komplexere Behandlung solcher Fälle, die im Prinzip hier dependency-directed Backtracking vorsieht, wird dabei nicht benötigt.

Diesbezüglich wurde im CAPlan-Kern die Vorgehensweise von REDUX dahingehend modifiziert, daß es grundsätzlich nicht möglich ist, inkonsistente Operatoren[8] auszuwählen. Die REDUX-Konzepte werden dadurch nicht grundlegend geändert, sondern lediglich auf den für die Realisierung von SNLP notwendigen Spezialfall eingeschränkt. Dies wurde dadurch erreicht, daß unmittelbar nach dem Berechnen der Konfliktmenge potentiell anwendbarer Operatoren jeder dieser Operatoren auf seine Konsistenz getestet werden. Für inkonsistente Operatoren wird eine Begründung für die Rückzugsnotwendigkeit generiert (vgl. Kap. 5.6.2.1), die sicherstellt, daß die Operatoren erst ausgewählt werden können, wenn die Inkonsistenz nicht mehr entstehen kann. Petrie sieht einen solchen Konsistenztest vor der Operatorauswahl ursprünglich nicht vor (Constraint-Propagierung vs. Constraint-Überprüfung).

Die oben beschriebene Modifikation hat folgende Konsequenzen:

- Es können keine Operatoren ausgewählt werden, die Constaints über den aktuell gültigen Zuweisungen verletzen, d.h. inkonsistente Pläne erzeugen.
- *Constraint-Violation-Tasks* können erst gar nicht entstehen, da Constraint-Verletzungen im REDUX-Sinne immer vor der Operatorauswahl geprüft werden und sofort durch die Generierung einer Rückzugsbegründung des beteiligten Operators behandelt werden.

[8] Das sind Operatoren, durch deren neue Zuweisungen die Menge aller aktuell gültigen Zuweisungen inkonsistent wird.

Das bedeutet insgesamt, daß es nur eine Art von Inkonsistenzen gibt, die REDUX im CAPlan-Kern mittels eines Tasks meldet, nämlich die Blockade eines Zieles mittels eines *Goal-Block-Task*. Dies entspricht aber genau den Erfordernissen für die Realisierung von SNLP.

5.5.2 Erweiterung von Konfliktmengen

In REDUX wird davon ausgegangen, daß die zu einem Ziel gehörige Konfliktmenge alternativer Operatoren sich nicht mehr ändert, nachdem sie aufgrund der Abarbeitung eines *Unreduced-Goal-Task* einmal berechnet wurde. Diese Sichtweise reicht für die Lösung von Planungsaufgaben mit Hilfe der hier beschriebenen Verfahren jedoch nicht aus.
Betrachtet man z.B. die Ziele zum Erzeugen eines Causal Link für eine offenen Vorbedingung, so können diese entweder durch Domänenoperatoren (CAPDomainOp) oder Phantomisierungsoperatoren (CAPPhantomizeOp) erfüllt werden. Gerade aber die zuletzt genannten werden aufgrund des aktuellen Planzustandes berechnet, da hier bereits existierende Schritte des Planes zum Erfüllen einer Vorbedingung verwendet werden. Betrachtet man nun die beim hier beschriebenen Planungssystem gegebene Möglichkeit, daß der Benutzer zu einem beliebigen Zeitpunkt eine getroffene Entscheidung wieder zurückziehen kann, so kann man in der Regel nicht davon ausgehen, daß sich beim erneuten Bearbeiten eines durch einen solchen Rückzug wieder offenen Zieles die Konfliktmenge nicht verändert hat. Insbesondere können nun neue Phantomisierungsmöglichkeiten hinzugekommen sein.

Daraus ergibt sich die Notwendigkeit, die bereits berechnete Konfliktmenge im Falle einer erneuten Bearbeitung eines Zieles um solche neuen Alternativen zu erweitern. Dies hat zur Folge, daß auch das TMS-Netz zur Erkennung von Zielblockaden (vgl. Kap. 4.2.4) entsprechend erweitert werden muß.

5.5.3 Ziele zum Schützen von Causal Links

Für Ziele zum Schützen eines Causal Links (ProtectionGoal) mußte auf der einen Seite der Mechanismus zur Erzeugung dieser Ziele erweitert werden, auf der anderen Seite mußte an zwei Stellen in die zur Realisierung von REDUX verwendete TMS-Abhängigkeitsstruktur eingegriffen werden, um die Gültigkeit eines solches Zieles sowie die Tatsache, daß ein solches Ziel bereits erfüllt ist, ohne daß ein Operator angewendet wurde, ausdrücken zu können.

5.5.3.1 Erzeugung eines ProtectionGoal

REDUX sieht vor, daß ein Operator nur für das Ziel, auf das er angewendet wurde, neue Teilziele herleiten kann. Die Erzeugung eines Zieles zur Behandlung von Threats (ProtectionGoal) kann im Gegensatz dazu im CAPlan-Kern auch nachträglich von einem anderen Operator angestoßen werden.

Dies hat hauptsächlich implementierungstechnische Gründe, da die Berechnung von neuen Threats nach jedem Inferenzschritt, der einen Causal Link oder einen Planschritt eingeführt hat, angestoßen wird. Ein ProtectionGoal kann damit immer als neues Teilziel des Operators, das den bedrohten Causal Link oder die Teilzielrekursion verursacht hat, eingefügt werden. Die dahinterstehende Überlegung ist, daß ein Ziel, dessen Bearbeitung einen Causal Link erzeugt, alle Ziele zum Schützen dieses Causal Link als Unterziele haben sollte. Letztere sind aber in der Regel zum Zeitpunkt der Erzeugung des Causal Links noch nicht alle bekannt und können daher nur nachträglich ergänzt werden.

5.5.3.2 TMS-Rechtfertigung für Gültigkeit eines ProtectionGoal

Für ein Ziel zum Schützen eines Causal Links wurde die TMS-Rechtfertigung des *Valid-Goal-Knotens*, der die Gültigkeit des Zieles repräsentiert, erweitert. Die Gültigkeit eines solchen Zieles hängt nicht nur von einem Oberziel und der zugehörigen Entscheidung ab, wie es bei REDUX normalerweise der Fall ist (vgl. Kap. 4.2.3), sondern neben der Entscheidung, die den Causal Link verursacht hat auch von der Entscheidung, die den Schritt einführt hat, der als Bedrohung für einen Causal Link identifiziert wurde.

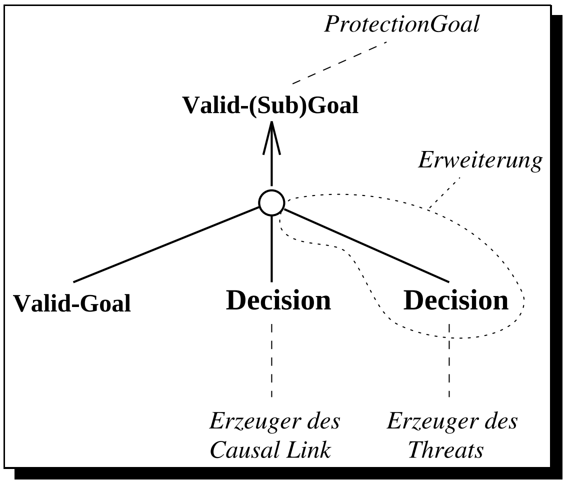

Abbildung 5.8: Gültigkeit eines Zieles zum Schützen eines Causal Link

Daraus ergibt sich, daß die TMS-Rechtfertigung des *Valid-Goal-Knotens* eines Zieles zum Schützen eines Causal Links wie in Abbildung 5.8 gezeigt aussieht. Sobald also entweder der Erzeuger des Causal Links oder der Erzeuger des Threats zurückgezogen wird, verliert das zugehörige ProtectionGoal sein Gültigkeit.

Zu bemerken ist, daß für ein ProtectionGoal zum Auflösen einer Teilzielrekursion die Abhängigkeitsstruktur nicht erzeugt werden muß, da hier die beiden betroffenen Entscheidungen aufgrund der von REDUX automatisch verwalteten Teilzielhierarchie schon abhängig voneinander sind (direkt oder indirekt über den *Valid-Parent-Decision-Knoten*).

5.5.3.3 TMS-Rechtfertigung für Erfüllung eines ProtectionGoal ohne Operatoranwendung

Nach einem Inferenzschritt werden alle Threats bestimmt, die aufgrund der Zuweisungen des Operators, der in diesem Inferenzschritt angewandt wurde, entstanden sind. Bei der Berechnung dieser Threats kann und darf jedoch noch nicht entschieden werden, ob ein gefundener Threat wirklich behandelt werden muß. Es kann vielmehr sein, daß ein Threat aufgrund der aktuell gültigen Ordnungen im Plan oder den aktuell gültigen Variablenbindungs-Constraints nicht behandelt werden muß, da er bereits aufgelöst ist. In diesem Fall ist es sinnvoll, das zugehörige ProtectionGoal zunächst als bereits erfüllt zu markieren, ohne daß ein Operator angewandt wurde.
Wichtig ist aber, daß dieses Ziele und seine Abhängigkeitsstrukturen dennoch erzeugt werden, denn spätere Operatorrückzüge (z.B. durch den Benutzer) können die Notwendigkeit hervorrufen, daß der Threat wirklich aufgelöst werden muß, weil durch den Rückzug auch eine Zuweisung ungültig wurde, die vorher den Threat implizit auflöste. Nun muß der Threat explizit durch einen ProtectionOp aufgelöst werden.

Dazu bot es sich an, eine TMS-Rechtfertigung des *Satisfied-Knotens* zu erzeugen, die genau den Grund festhält, der eine Bearbeitung dieses Threats momentan nicht erforderlich macht. Die hier gemachte Änderung erlaubt also für *Satisfied-Knoten* neben der von REDUX erzeugten TMS-Rechtfertigung weitere, von einem Begründungsgenerator erzeugte Rechtfertigungen (s. Kap. 5.6.2.2).

5.5.4 Optimalität von Entscheidungen

SNLP ist ein Algorithmus, der alleine durch systematisches Vorgehen bei der Lösungssuche zur Lösung von Problemen gelangt; er verwendet dabei kein Wissen über die Qualität von Entscheidungen. Die Auswahl von Zielen geschieht nach festgelegten, systematischen Kriterien; die Auswahl von Operatoren wird in der Implementierung durch Heuristiken über die Nützlichkeit von Operatoren gesteuert, die aber insbesondere keine lokale Optimalitätsaussage erlauben.

Daher wurde im CAPlan-Kern die automatische Begründungsgenerierung des *BestOp-Knotens* (vgl. Kap. 4.2.3) derart modifiziert, daß hier immer Prämissen als Begründung erzeugt werden. Dies hat zur Folge, daß jede Entscheidung immer als lokal optimal angesehen wird und damit niemals Optimalitätsverluste im Sinne von REDUX auftreten können.

5.6 Inkonsistenzerkennung und Begründungsgenerierung

Bei der Realisierung des SNLP-Ansatzes auf der Basis von REDUX werden die Bestandteile eines Planes als Zuweisungen von Operatoren interpretiert. Wie in Kapitel 5.5.1 bereits beschrieben, wird vor einer Operatorauswahl die Konsistenz des Planes, der durch die neuen Zuweisungen des jeweiligen Operators entsteht, überprüft. Nur solchermaßen als konsistent geprüfte Operatoren können aus einer Konfliktmenge von Operatoren ausgewählt und angewandt werden.

Zur Erkennung von solchen inkonsistenten Operatoren verwendet der CAPlan-Kern, wie bereits in der Systemübersicht aus Abbildung 5.1 zu sehen war, den *Konsistenztester* und zur Generierung von Rückzugsbegründungen ein mit diesem gekoppeltes Modul, den *Begründungsgenerator*.

5.6.1 Inkonsistenzerkennung

Konsistenz eines Planes bezieht sich hier grundsätzlich auf die Aspekte der Ordnungs- und der Bindungskonsistenz wie in Kapitel 3.3.1 definiert. Der Konsistenztester im CAPlan-System hat die Aufgabe zu überprüfen, ob die von einem Operator eingeführten neuen Zuweisungen zusammen mit den aktuell gültigen noch einen konsistenten Plan ergeben.

Testen von Ordnungskonsistenz

Ordnungskonsistenz eines Planes wurde bereits in Kapitel 3.3.1 genauer definiert. Diese Definition bildet die Grundlage für den vom Konsistenztester durchgeführten Test und soll hier nochmal wiederholt werden. Ordnungskonsistenz bedeutet:

- Kein Schritt ist vor dem START-Schritt angeordnet.
- Kein Schritt ist hinter dem FINISH-Schritt angeordnet.
- Es bestehen keine Zykel bzgl. der Ordnung von Schritten. Zykelfreiheit heißt, daß ein Plan ordnungsinkonsistent wird, falls es zwei Schritte s und d gibt, für die $s \prec d$ aufgrund der Ordnungen $s \prec v_1 \prec \cdots \prec v_n \prec d$ gilt, und man die Ordnung $d \prec s$ ergänzt.

Im Falle einer Inkonsistenz liefert der Konsistenztester als Grund für die Ordnungsinkonsistenz: (Ann. es wird $d \prec s$ getestet)

- den START-Schritt (wenn $s = START$) oder
- den FINISH-Schritt (wenn $d = FINISH$) oder
- die Ordnungen $\{s \prec v_1, v_n \prec d\} \cup \{v_i \prec v_{i+1} | i = 1, \cdots, n-1\}$.

Diese zurückgelieferten Schritte oder Ordnungen bilden die Basis für die Generierung von TMS-Rechtfertigungen aus solchen Ordnungsinkonsistenzen.

Testen von Bindungskonsistenz

Neben den Ordnungen gibt es auch Variablenbindungs-Constraints im Plan, die die Menge möglicher Bindungen von Variablen, die in den Planschritten verwendet werden, festlegen oder einschränken. Die vier Arten `Same`, `NotSame`, `IsOfType` und `IsNotOfType` von Variablenbindungs-Constraints (vgl. Kap. 2.2.3 und 5.3) können dazu verwendet werden.

Der Konsistenztester ist in der Lage zu überprüfen, ob die durch einen neuen Operator hinzukommenden Variablenbindungs-Constraints zusammen mit den aktuell gültigen noch konsistent sind, d.h. ob es noch eine Bindungsmöglichkeit für jede der Variablen gibt derart, daß keiner der Variablenbindungs-Constraints verletzt wird.

Im Falle einer Inkonisistenz liefert der Konsistenztester hier eine Teilmenge $B' \subseteq B$ der Variablenbindungs-Constraints des aktuellen Planes (S, O, B), die zu der Inkonsistenz führten. Diese Menge B' wird wiederum für die Generierung von TMS-Rechtfertigungen genutzt.

Testen konsistenter Mengen von Zuweisungen

Die bisher beschriebenen Tests, die der Konsistenztester macht, und die gelieferten Ergebnisse bezogen sich ausschließlich auf den Fall, daß die getesteten Zuweisungen als inkonsistent identifiziert wurden. Tritt dagegen der Fall ein, daß keine Inkonsistenz besteht, so liefert der Konsistenztester ebenfalls noch eine zusätzliche Information darüber, warum das Testergebnis positiv war. Dies geschieht in der Weise, daß hier die Menge der Zuweisungen des aktuellen, konsistenten Planes geliefert wird, die beim Test eine Rolle spielten.

Wird eine Ordnung $s \prec d$ als konsistent erkannt, so liefert er die Ordnungen des aktuellen Planes, die dies beweisen, also eine Menge $A = \{s \prec v_1, v_n \prec d\} \cup \{v_i \prec v_{i+1} | i = 1, \cdots, n-1\}$. Für den Test von Variablenbindungs-Constraints ergibt sich in ähnlicher Weise eine Menge von Constraints, die mit den zu testenden Constraints dadurch in Zusammenhang stehen, daß sie dieselben Variablen betreffen.

5.6.2 Begründungsgenerierung

In Kapitel 4.2.3 wurde beschrieben, daß bei REDUX grundsätzlich an drei Stellen mit problemspezifischen TMS-Rechtfertigungen die Aspekte der lokalen Optimalität, Zulässigkeit und Rückzugsnotwendigkeit von Entscheidungen ausgedrückt werden können. Bei der Realisierung des SNLP-Ansatzes wird davon nur Definition der Rückzugsnotwendigkeit einer Entscheidung benötigt. Als Erweiterung wurde in Kapitel 5.5.3 bereits beschrieben, daß auch die Tatsache, daß ein erzeugtes ProtectionGoal bereits ohne Operatoranwendung erfüllt sein kann, durch eine gegenüber REDUX zusätzliche zu erzeugende TMS-Rechtfertigung des *Satisfied-Knotens* realisiert wurde.

5.6.2.1 Rückzugsbegründung für inkonsistente Operatoren

Operatoren, die durch einen Konsistenztest wie in Kapitel 5.6.1 als inkonsistenz erkannt wurden, werden im CAPlan-Kern durch die Generierung einer TMS-Rechtfertigung für die Rückzugsnotwendigkeit des Operators als zurückgezogen begründet. Die Generierung dieser Rechtfertigung berücksichtigt, daß durch diese genau die erkannte Inkonsistenz abgedeckt wird, d.h. genau dann, wenn die Zuweisungen gültig sind, die die Inkonsistenz erzeugen, besteht die Rückzugsnotwendigkeit für diesen Operator.

Die Generierung solcher Rückzugsbegründungen baut wesentlich darauf auf, daß der Konsistenztester Informationen darüber liefert, welche Teilmenge der aktuell gültigen Zuweisungen die Inkonsistenz hervorrufen. In Kapitel 5.6.1 wurde beschrieben, daß der Konsistenztester des CAPlan-Systems genau dies leistet. Wird eine Inkonsistenz gefunden, so liefert der Konsistenztester zusätzliche Informationen, welche Zuweisungen von Operatoren diese Inkonsistenz hervorrufen.

Zur Generierung der TMS-Begründungen profitiert der CAPlan-Kern nun ganz wesentlich davon, daß REDUX alle Abhängigkeiten zwischen Entscheidungen und ihren Auswirkungen im Sinne von gemachten Zuweisungen explizit repräsentiert. Konkret geschieht dies dadurch, daß die *Assigned-Knoten* einer Entscheidung die Gültigkeit der durch die Entscheidung hervorgerufenen Zuweisungen ausdrücken.

Liefert der Konsistenztester eine Menge $A = \{a_1, \cdots, a_m\}$ von Zuweisungen als Grund für einen entdeckte Inkonsistenz, so kann REDUX für jede Zuweisung a_i den zugehörigen *Assigned-Knoten* bestimmen. Dies ergibt die Menge $A_{Assigned} = \{Assigned_1, \cdots, Assigned_{m'}\}$ von *Assigned-Knoten*[9], aus welcher die gesuchte TMS-Rechtfertigung dadurch gewonnen wird, daß $A_{Assigned}$ als IN-Liste der Rechtfertigung verwendet wird. Wird damit die Rückzugsnotwendigkeit einer Entscheidung begründet (s. Abb. 5.9), besteht für die Entscheidung genau solange Rückzugsnotwendigkeit, wie alle Elemente von A gültig sind.

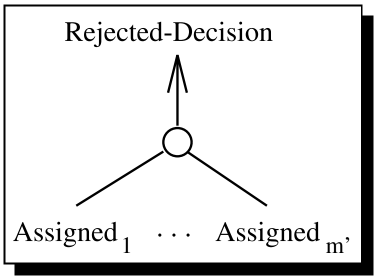

Abbildung 5.9: Rückzugsbegründung für inkonsistente Operatoren

5.6.2.2 Generierung von TMS-Begründungen für Satisfied-Knoten

In Kapitel 5.5.3 wurde bereits beschrieben, daß Ziele zum Schützen eines Causal Links (Protection-Goal) u.U. bereits ohne Operatoranwendung erfüllt sein können. Dies wird im CAPlan-Kern dadurch modelliert, daß eine entsprechende TMS-Begründung des *Satisfied-Knotens* generiert wird, die den Grund festhält, warum das Ziel erfüllt ist und damit eine Bearbeitung des Zieles solange hinauszögert, wie dies nicht notwendig ist.

Die Vorgehensweise ist ähnlich wie in Kapitel 5.6.2.1 für den Rückzug von Operatoren. Der Konsistenztester liefert im Fall, daß er eine Menge von Zuweisungen als konsistent testet, als zusätzliche Information, die Menge der Zuweisungen, die dazu getesten werden mußten (vgl. Kap. 5.6.1). War dies wieder eine Menge $A = \{a_1, \cdots, a_m\}$, so läßt sich mittels REDUX zu jedem a_i der zugehörige *Assigned-Knoten* und damit die resultierende Menge $A_{Assigned} = \{Assigned_1, \cdots, Assigned_{m'}\}$ von *Assigned-Knoten* bestimmen, welche als IN-Liste der gesuchten TMS-Rechtfertigung für den *Satisfied-Knoten* eines bereits erfüllten ProtectionGoals verwendet wird (s. Abb. 5.10).

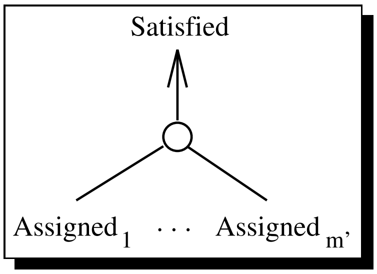

Abbildung 5.10: Begründung des *Satisfied-Knotens* für erfülltes ProtectionGoal

[9]Es gilt $m = m'$ nur in dem Fall, daß jede Zuweisung durch einen eigenen *Assigned-Knoten* repräsentiert wird. In der Implementierung können mehrere Zuweisungen durch einen solchen Knoten im TMS repräsentiert sein, weil dies den Vorteil hat, daß die Menge von TMS-Knoten damit in der Regel kleiner gehalten wird.

5.6.2.3 Rückzugsbegründung für Operatoren aufgrund von Backtracking

Der SNLP-Algorithmus arbeitet mit chronologischem Backtracking bzw. in einer Erweiterung mit dem in Kapitel 3.4.2 beschriebenen dependency-directed Backtracking. Beiden Fällen ist gemeinsam, daß Backtracking dadurch realisiert wird, daß eine oder mehrere (dies hängt von der Art des Backtrackings ab) festgelegte Entscheidungen zurückgezogen werden müssen. Wieder müssen im CAPlan-Kern dazu spezielle TMS-Begründungen für die Rückzugsnotwendigkeit generiert werden.

Für Backtracking spielen als Grund für den Operatorrückzug nun andere Entscheidungen, also *Decision-Knoten*, ein Rolle, insbesondere alle die Entscheidungen DEC_{Valid}, die in dem aktuellen Zustand gültig sind, also die Entscheidungen, deren *Decision-Knoten* das Label IN hat. Außerdem spielt die Reihefolge, in der Entscheidungen getroffen wurde, eine Rolle. Sei schließlich D_{last} die zuletzt getroffene Entscheidung. Gesucht ist nun jeweils eine Menge von Entscheidungen und eine TMS-Rechtfertigung für die Rückzugsnotwendigkeit für jede dieser Entscheidungen:

- *Chronologisches Backtracking:*
  In diesem Fall muß nur D_{last} zurückgezogen werden. Die TMS-Rechtfertigung für die Rückzugsbegründung besteht aus einer IN-Liste mit den Knoten $DEC_{Valid} - \{D_{last}\}$.
- *Dependency-directed Backtracking:*
  Hier wird zunächst durch eine Inkonsistenzanalyse wie sie in Kapitel 3.4.2 erläutert wurde eine Entscheidung $D_{culprit}$ bestimmt, bis zu der zurückzuspringen ist. Seien $DEC_{succ} = \{D_1, \cdots, D_{last}\}$ die nach $D_{culprit}$ getroffenen Entscheidungen. Um korrekt bis zum zu $D_{culprit}$ gehörigen Ziel zurückzuspringen, müssen die Entscheidungen aus DEC_{succ} sowie $D_{culprit}$ zurückgezogen werden. Dies kann man dadurch erreichen, daß man beginnend mit D_{last} solange immer die jeweils vorher getroffene Entscheidung genauso wie beim chronologischen Backtracking zurückzieht bis schließlich auch $D_{culprit}$ zurückgezogen ist.

Da REDUX automatisch eine Teilzielhierarchie generiert und verwaltet (vgl. Kap. 4.1.1), gibt es für die Implementierung die Möglichkeit, die Rückzugsbegründungen dahingehend zu optimieren, daß die keine voneinander abhängigen Entscheidungen enthalten. Dabei hängt die Entscheidung D_i genau dann von Entscheidung D_j ab, wenn das zu D_i gehörige Ziel im Teilbaum unterhalb des zu D_j gehörigen Zieles vorkommt.

5.7 Realisierung des Algorithmus

Bei der Realisierung des Algorithmus kann kann man zwischen zwei Arten von Inferenzschritten unterscheiden: eine Zielreduktion besteht darin, ein nicht reduziertes Teilziel durch Anwendung eines Operators zu bearbeiten. Das Ergebnis davon ist entweder, daß dies mit Erfolg gelingt oder es wird eine Inkonsistenz festgestellt derart, daß ein Ziel im Sinne von REDUX als blockiert erkannt wird. Die Bearbeitung einer Zielblockade ist die zweite Art von Inferenzschritt, die gemacht wird. Hier handelt es sich um die Notwendigkeit Backtracking durchzuführen. Beides, Zielreduktion und Backtracking, kann im CAPlan-Kern auf natürliche Art auf die Abarbeitung von Tasks abgebildet werden.

5.7.1 Zielreduktion

REDUX signalisiert die Tatsache, daß ein Ziel noch nicht durch eine Operatoranwendung reduziert ist, durch die Erzeugung eines Tasks zur Zielreduktion (*Unreduced-Goal-Task*). Die Abarbeitung dieser Tasks geht in folgenden Schritten vor sich:

1. Die Berechnung der momentanen Konfliktmenge des Zieles betrachtet bei der Zielreduktion folgende Operatortypen, welche in die Konfliktmenge aufgenommen werden unter Berücksichtigung der gegebenen Ordnung auf Operatoren (vgl. Kap. 5.4.5):
   (a) Phantomisierungen mittels einer Invarianten aus der Problemstellung (CAPPhantomizeOp)

   (b) Phantomisierungen mittels eines existierenden Schrittes (vgl. Kap. 5.7.2) (CAPPhantomizeOp)

   (c) Verwendung von Operator-Schemata aus der statischen Wissensbasis (CAPDomainOp)

   War für das zugehörige Ziel bereits eine Konfliktmenge berechnet worden, so wird diese lediglich bei Bedarf erweitert (vgl. Kap. 5.5.2).

2. Jeder Operator, der so berechnet wurde, wird auf seine Konsistenz getestet, d.h. es wird getestet, ob die Menge der aktuell gültigen Zuweisungen zusammen mit den neuen Zuweisungen dieses Operators noch konsistent (ordnungs- und bindungskonsistent) ist. Für inkonsistente Operatoren wird sofort eine Begründung für die Rückzugsnotwendigkeit generiert (s. Kap. 5.6.2.1), die verhindert, daß dieser Operator ausgewählt werden kann, solange die gefundene Inkonsistenz dabei entstehen kann.

3. Die aktuell aktive Kontrollkomponente wählt einen Operator aus der Menge der zulässigen, nicht zurückgewiesenen Operatoren aus.

4. Bei der Anwendung des Operators geschieht folgendes:

   (a) Zunächst werden alle REDUX-spezifischen Abhängigkeitsstrukturen aufgebaut.

   (b) Der Test, ob bei der Herleitung der neuen Teilziele Teilzielrekursionen auftreten, erzeugt ggf. entsprechende Ziele zur Auflösung von Teilzielrekursionen (ProtectionGoals).

   (c) Die Berechnung aller neuen Threats (vgl. Kap. 5.7.2) erzeugt für jeden Threat ein entsprechendes ProtectionGoal.

Nachdem der Task auf diese Art abgearbeitet wurde, bewirkt eine Propagierung des TMS, daß das gesamte System wieder in eine aus TMS-Sicht konsistenten Zustand gelangt. Dabei werden u.U. automatisch eine Reihe neuer Tasks erzeugt.

5.7.2 Suche nach Erzeugern oder Zerstörern von Effekten

Bei der Abarbeitung von Tasks zur Zielreduktion gibt es zwei Stellen, an denen Informationen aus dem aktuell gültigen Plan benötigt werden. Dies betrifft einerseits die Berechnung einer Konfliktmenge, in der Operatoren enthalten sein können, die Effekte existierender Schritte ausnutzen (1. (b)), andererseits wird bei der Suche nach Threats (4. (c)) derartige Information benötigt. Um ein andauerndes Suchen im Plan selbst zu vermeiden, verwendet der CAPlan-Kern dazu in Anlehnung an das System NONLIN [Tat77] ein sog. *TOME (Table Of Multiple Effects)*. Dieses soll die Suche nach Schritten mit bestimmten Effekten in wesentlich besser zugreifbarer Form ermöglichen, indem es den Effekt selbst als Zugriffsschlüssel auf eine Liste von Schritten, die diesen Effekt haben, verwendet.

Jede Anfrage liefert als Ergebnis eine Menge von Schritten sowie für jeden dieser Schritte eine Menge von Variablenbindungs-Constraints[10], mit deren Gültigkeit der Schritt die Anfrage erfüllt. Konkret können folgende Anfragen beantworten werden:

- *Suche nach Quellen für Phantomisierungen:*
  Es werden alle die Schritte geliefert, die einen bestimmten Effekt haben.

- *Suche nach Threats:*
  Es werden alle Schritte geliefert, die für einen gegebenen Causal Link eine Bedrohung darstellen.

Mittels der vom TOME gelieferten Informationen werden konkrete Instanzen von Phantomisierungsoperatoren und Zielen zum Schützen von Causal Links gebildet.

[10] Ist diese Menge leer, so bedeutet das, daß keine Constraints notwendig sind.

5.7.3 Backtracking

Durch die in Kapitel 5.5.1 beschriebene Modifikation von REDUX im CAPlan-Kern wurde erreicht, daß nur durch Zielblockaden, welche von REDUX durch *Goal-Block-Tasks* gemeldet werden, die Notwendigkeit zum Backtracking signalisiert wird. In einem solchen Fall muß durch den Rückzug einer oder mehrer Entscheidungen dafür gesorgt werden, daß die Zielblockade aufgelöst wird.

Die Abarbeitung eines solchen Task besteht aus folgenden Schritten:

1. Zunächst müssen von der aktuellen Kontrollkomponente eine oder mehrere Entscheidungen bestimmt werden, die als Grund für die Zielblockade identifiziert werden sollen. Hier können verschiedene Strategien realisiert werden wie chronologisches Backtracking oder dependency-directed Backtracking.

2. Für jede dieser zurückzuziehenden Entscheidungen muß durch die Kontrollkomponente eine Begründung für die Rückzugsnotwendigkeit geliefert werden. Dabei wird die Kontrollkomponente in der Regel den Begründungsgenerator verwenden, der für die Fälle von chronologischem und dependency-directed Backtracking geeignete Begründungen generieren kann.

Danach stößt REDUX die Propagierung der Änderungen an, was zur Folge hat, daß diese Zielblockade danach beseitigt ist.

5.8 Realisierung der Kontrollkomponenten

Der CAPlan-Kern selbst stellt alle Mechanismen zur Verfügung, die die Realisierung der SNLP-Konzepte mittels REDUX benötigt. Das System ist jedoch so konzipiert, daß es die eigentliche Kontrolle der Suche einer von mehreren möglichen Kontrollkomponenten übergibt. Das bedeutet, das der SNLP-Algorithmus selbst in wesentlichen Punkten in Form einer speziellen Kontrollkomponente zu finden ist.

5.8.1 Kontrollkomponenten im CAPlan

Die Kontrollkomponenten haben die Aufgabe, die Lösungssuche des Systems zu steuern. Da die Realisierung von CAPlan auf das REDUX-System aufbaut, ergeben sich folgende Möglichkeiten, an denen eine Kontrollkomponente eingreifen kann und muß:

Zielreduktion: Bearbeitung eines noch nicht reduzierten Zieles

- Auswahl des nächsten zu bearbeitenden Tasks und damit auch des Zieles
- Auswahl des anzuwendenden Operators aus einer Konfliktmenge von Alternativen

Backtracking: Realisierung eines Backtracking-Verfahrens durch Berechnung der zurückzuweisenden Entscheidung(en) und Generierung von Begründungen für die Rückzugsnotwendigkeit von Entscheidungen

5.8.2 Systematische Kontrollkomponente CLPC

Zielreduktion

Die Realisierung des SNLP-Algorithmus aus Abbildung 3.6 in Kapitel 3.3.3 erfordert zunächst die in Kapitel 5.4.5 bereits erwähnten Prioritäten der Ziele untereinander. Damit ist alles notwendige für die Zielreduktion bereits abgedeckt, da SNLP keine weiteren speziellen Auswahlmechanismen benötigt. Es wurden jedoch noch die durch die Heuristiken ausgedrückten Ordnungen auf Operatoren aus Kapitel 5.4.5 hierbei verwendet.

Backtracking

Für den Fall von Backtracking, kann das System in zwei Modi arbeiten (vgl. Kap. 3.4.2): der erste Modus realisiert das normale chronologische Backtracking, der zweite dependency-directed Backtracking (vgl. Kap. 3.4.2), bei dem mittels einer Inkonsistenzanalyse u.U. um mehr als nur einen Schritt zurückgesprungen wird (*back-jumping*).

Chronologisches Backtracking, wie es die ursprüngliche Version des SNLP-Algorithmus vorsieht, liefert die letzte getroffene Entscheidung als die zurückzuziehenden Entscheidung.

Dependency-directed Backtracking ist aufwendiger, da dazu die aufgrund von Inkonsistenzen im Plan zurückgewiesen Operatoren wie in Kapitel 3.4.2 beschrieben analysiert werden. Ergebnis davon ist, daß aufgrund dieser Analyse eine Entscheidung bestimmt wird, bis zu der zurückgesprungen werden soll. Da dies nicht immer die zuletzt getroffene Entscheidung sein muß, hat das zur Konsequenz, daß u.U. mehrere Entscheidungen in einem Schritt zurückgezogen werden.

Für beide Modi kann der Begründungsgenerator des CAPlan-Systems (vgl. Kap. 5.6.2) geeignete Rechtfertigungen für die Rückzugsnotwendigkeit von Entscheidungen generieren.

5.8.3 Interaktive Kontrollkomponente UC

Zielreduktion

Interaktive Zielreduktion heißt, daß jedes Mal, wenn eine Auswahl aus einer Menge von Möglichkeiten getroffen werden muß, diese vom Benutzer interaktiv vorgenommen wird oder zumindest vorgenommen werden kann. Es gibt drei Möglichkeiten dazu:

- Interaktive Auswahl des nächsten zu bearbeitenden Tasks und Auswahl des anzuwendenden Operators durch die Kontrollkomponente CLPC
- Auswahl des nächsten zu bearbeitenden Tasks durch die Kontrollkomponente CLPC und interaktive Auswahl des anzuwendenden Operators
- Interaktive Auswahl von Task und Operator

Zur Komuunikation mit dem Benutzer verwendet die interaktive Kontrollkomponente geeignete Elemente, die in die graphischen Benutzeroberfläche des CAPlan-Systems integriert sind.

Backtracking

Interaktives Backtracking bedeutet, daß im Falle einer Zielblockade, der Benutzer entscheiden kann, welche Entscheidungen zurückgezogen werden müssen und wie die Rückzüge zu begründen sind. Besonders die Begründung von Entscheidungsrückzügen durch den Benutzer ist allerdings kritisch, da dieser hier darauf zu achten hat, daß durch diese Begründungen keine Schleife im TMS-Rechtfertigungsnetz entstehen.

5.8.4 Weitere Kontrollkomponenten

Die Erweiterung des System um andere als die soeben vorgestellen Kontrollkomponenten ist recht einfach, da im wesentlichen die in Kapitel 5.8.1 definierten Auswahlentscheidungen von der Kontrollkomponente gemacht werden müssen. D.h. eine Kontrollkomponente muß den nächsten zu bearbeitenden Task sowie einen Operator aus einer Konfliktmenge ausgewählen. Außerdem kann sie noch ein Backtracking-Verfahren realisieren, wenn die als Default vorhandenen Verfahren (chronologisches oder dependency-directed Backtracking) nicht verwendet werden sollen.

Kapitel 6

Diskussion und Ausblick

Mit dem System CAPlan wurde eine generische Architektur für domänenunabhängige, nichtlineare Aktionsplanung konzipiert und realisiert. Das System baut auf zwei Grundpfeilern auf: zum einen ist das ein Ansatz zur *systematischen, nichtlinearen Planung (SNLP)* [MR91] von David McAllester und David Rosenblitt. Zum anderen ist es das *REDUX-System* [Pet91, Pet92] von Charles Petrie.

Der SNLP-Ansatz zeichnet sich durch eine Reihe erwähnenswerter Merkmale aus. Zunächst ist dieser Ansatz ein Vertreter der Sicht des Planens als *Suche im Raum der möglichen Planzustände* im Gegensatz zur (ältern) Sicht als Suche im Raum möglicher Weltzustände. Der Algorithmus ist vollständig und korrekt, d.h. wenn eine Lösung existiert, so findet der Algorithmus sie, und jede Lösung ist korrekt in dem Sinne, daß die Ausführung des Planes den Startzustand in den Zielzustand überführt. Außerdem zeichnet sich dieser Ansatz dadurch aus, daß er sytematisch im Raum möglicher Planzustände sucht, d.h. es wird garantiert, daß kein Planzustand mehr als einmal erzeugt werden kann. Dies kann allerdings nur auf Kosten eines Abrückens von der *Least-Commitment-Strategie* erreicht werden, welche nur dann explizite Entscheidungen zu treffen versucht, wenn dies unumgänglich ist. Dies spielt besonders bei der Auflösung von Interaktionen eine wichtige Rolle.
Eine wesentliche Schwäche des Algorithmus aus [BW93], der dieser Arbeit zugrundeliegt, die Nicht-Termination im Falle von Teilzielrekursionen, konnte, wie in Kapitel 3.4.1 ausführlicher beschrieben, gelöst werden. Außerdem wurde das chronologischen Backtracking des Algorithmus durch eine Form von dependency-directed Backtracking (*back-jumping*) ersetzt, die einerseits die entstandene Inkonsistenz analysiert und daraus gewonne Informationen bei der Auswahl des Rücksprungpunkten berücksichtigt, um den Suchraum zu reduzieren, andererseits die Vollständigkeit des Algorithmus unberührt läßt.

Das REDUX-System von Charles Petrie als Grundlage für die Realisierung von CAPlan bietet für den Bereich Planung mit der REDUX-Ontologie eine charakteristische Begriffswelt, mit der sich die Repräsentation und Lösung von Planungsaufgaben leicht realisieren lassen. Problemstellungen werden durch zu erreichende Ziele beschrieben; Operatoren werden auf Ziele angewandt, um diese zu reduzieren, und leiten u.U. neue Unterziele her und machen Zuweisungen, die die Komponenten sind, aus denen ein Plan besteht. REDUX zeichnet sich besonders dadurch aus, daß vielfältige Abhängigkeiten, wie z.B. die Abhängigkeiten zwischen Zielen und Operatoren, Operatoren und ihren ihren Zuweisungen, explizit repräsentiert sind.

CAPlan wurde aufbauend auf REDUX entwickelt, indem die Konzepte des SNLP-Ansatzes auf entsprechende REDUX-Konzepte übertragen wurden. Dies gelang fast ohne jegliche grundsätzliche Änderungen an REDUX. Die Behandlung von Constraint-Verletzung wurde lediglich so vereinfacht, wie es die Realisierung des SNLP-Ansatzes benötigte. Die Erweiterbarkeit von Konfliktmengen ist jedoch neu und die Behandlung von Zielen zum Schützen von Causal Links erforderte sogar eine geringfügige Erweiterung der TMS-Abhänigkeitsstrukturen. Letztes war aber nur deshalb notwendig, da das System grundsätzlich so konzipiert ist, daß es an beliebiger Stelle von die aktive Kontrollkomponente wechseln kann und z.B. den Benutzer Entscheidungen zurückziehen lassen kann. Die vielfältigen Benutzerinteraktionsmöglichkeiten waren jedoch ein ausdrückliches Ziel bei der Realisierung von CAPlan als einem *Intelligenten Plannungsassistenten*, der auf der einen Seite zwar selbständig, systematisch nach einer Lösung suchen kann, aber mittels der interaktiven Kontrollkomponente auch vollständig vom Benutzer gesteuert werden kann.

6.1 Vorteile von REDUX bei der Realisierung von CAPlan

Es stellt sich dennoch die Frage, welche konkreten Vorteile REDUX bei der Realisierung gebracht hat. Die Realisierung des einfachen SNLP-Ansatzes in der vorgestellten Form erfordert bei weitem nicht die Möglichkeiten, die die Abhängigkeitsverwaltung in REDUX bietet. Um das Verfahren jedoch wie in Kapitel 3.4 beschrieben verbessern zu können, ist REDUX von großem Wert:

- Die Verwaltung einer Teilzielhierarchie ist für die Erkennung von Teilzielrekursion unbedingt notwendig und wird bei REDUX automatisch dadurch mitverwaltet, daß dort immer die Abhängigkeit zwischen Operatoren und den von ihnen hergeleiteten Zielen repräsentiert werden.

- Dependency-directed Backtracking wie in [BV92] (*back-jumping*) benötigt für die Inkonsistenzanalyse eine explizite Repräsentation von Abhängigkeiten zwischen einer Operatoranwendung und ihren Zuweisungen. Dadurch lassen sich Entscheidungen identifizieren, die für aufgetretene Inkonsistenzen durch von ihnen gemachte Zuweisungen verantwortlich sind.

- Besondere Vorteile ergeben sich jedoch im Hinblick auf die Möglichkeit, an jeder beliebigen Stelle der Lösungsfindung Benutzerinteraktionen zuzulassen, insbesondere den Rückzug beliebiger Entscheidungen. Die Abhängigkeitsverwaltung von REDUX nimmt hier automatisch alle von einer zurückgezogenen Entscheidung abhängigen Entscheidungen zurück.

Zusammenfassend kann man also sagen, daß für den einfachen SNLP-Ansatz REDUX nicht unbedingt gebraucht würde, daß allerdings für alle hier gemachten Verbesserungen wie die Behandung von Teilzielrekursionen, dependency-directed Backtracking und das Zulassen beliebiger Benutzerinteraktionen gerade die Abhängigkeitsverwaltung von REDUX die notwendige Infrastruktur zur Verfügung stellt.

6.2 Systematik bei der Suche

Bei der Behandlung von Interaktionen unterscheidet sich der hier verwendete SNLP-Ansatz von einigen anderen Verfahren dadurch, daß die Definition von Interaktionen hier etwas strenger ist: es wird neben den üblicherweise betrachteten negativen Interaktionen, die aus Gründen der Korrektheit eines Planes aufgelöst werden müssen, auch die Auflösung von positiven Interaktionen gefordert. Dies hat den Vorteil, daß damit die Suche nach der Lösung systematisch in der Hinsicht wird, daß kein Planzustand mehr als einmal erzeugt wird. Systeme wie z.B. NOAH [Sac75], NONLIN [Tat77], SIPE [Wil84, Wil88] oder TWEAK [Cha87] lösen nur negative Interaktionen auf und können die Systematik daher nicht garantieren. Nachteil der Systematik ist jedoch eindeutig die Tatsache, daß dies in der Regel zu mehr Backtracking führt, was hier dazu geführt hat, das Backtracking des SNLP-Algorithmus zu verbessern.

Gerade im Hinblick auf das Planen mittels Causal Links gibt es in [Kam92] von S. Kambhampati eine interessante Erweiterung. Um die Vorbedingung P eines Schrittes gültig zu machen werden hier sog. *Multi-contributor Causal Links* verwendet, die statt eines Ausgangsschritts, der den Effekt P hat, eine Menge von solchen Schritten betrachten. Damit wird Redundanz im Hinblick auf potentielle Garanten einer Vorbedingung durch weniger Ordnungsbeziehungen im Plan ausgenutzt. Allerdings wird die Erkennung von Interaktionen damit um einiges aufwendiger, womit sich insgesamt keine klaren Vorteile ergeben.

6.3 Aufgedeckte Zusammenhänge

6.3.1 Unterscheidung zwischen Haupt- und Seiteneffekten

Wilkins' Ansatz [Wil84], bei den Effekten eines Operators zwischen Haupt- und Seiteneffekten zu unterscheiden und nur Haupteffekte zur Operatorauswahl zu verwenden, hat beim SNLP-Ansatz die unangenehme Eigenschaft gezeigt, daß dadurch die Vollständigkeit des Algorithmus verlorengeht. Ein ungünstige Reihenfolge der Abarbeitung von Zielen kann dann verhindern, daß eine Lösung gefunden wird. Daher wurde lediglich die Heuristik verwendet, die Haupteffekte eines Operator bevorzugt bei der Operatorauswahl zu betrachten.

6.3.2 Auswahl von Zielen

Der SNLP-Algorithmus ist vollständig, d.h. er findet eine Lösung, sofern eine solche existiert. Die Frage, ob der Algorithmus bei einer beliebigen Zielauswahl immer dieselbe Lösung findet kann allerding klar verneint werden.

Dies kann man sich an einem einfachen Beispiel klarmachen. Angenommen, die Vorbedingungen des FINISH-Schrittes (Zielsituation) sind $+P$ und $+Q$ und es gebe die drei Schritte $s1$ mit den Effekten $+P$ und $-Q$, $s2$ mit den Effekten $+P$ und $+Q$ und $s3$ mit dem Effekt $+Q$.
Wird zuerst $+P$ bearbeitet und dadurch $s1$ in den Plan aufgenommen, so muß für die Bearbeitung von $+Q$ auch $s2$ oder $s3$ in den Plan aufgenommen werden. Der Plan enthält also in jedem Fall zwei Schritte. Bei anderer Reihenfolge der Bearbeitung von Zielen kann Ziel $+Q$ auch direkt mit $s2$ erfüllt werden, wodurch auch $+P$ erfüllt wird. Bei dieser Reihenfolge der Zielauswahl kann also ein Plan mit nur einem Schritt entstehen, d.h. die Reihefolge bei der Bearbeitung von Zielen beeinflußt, welche der möglichen Lösungen gefunden werden.

6.4 Einfluß der Domänenmodellierung

Einige andere Beobachtungen wurde im Zusammenhang mit der Domänenmodellierung gemacht.

Zunächst stellt sich die Frage, wie sich falsche Definitionen von Operator-Schemata auf den Algorithmus auswirken. Falsch meint hier, daß die Definition mehr Effekte enthält, als der Operator tatsächlich hat. Im Grunde handelt es sich dabei um einen Verstoß gegen die Definition von STRIPS-Operatoren, die hier vorausgesetzt wird und die verlangt, daß nur die sich ändernden Aspekte zu Effekten eines Operators werden.
Resultat einer fehlerhaften Definition von Operatoren kann es sein, daß der SNLP-Algorithmus keine Lösung finden kann, da nicht alle Interaktionen, die sich aus den Effekten der verschiedenen Operatoren ergeben, aufgelöst werden können. Ein Beispiel für einen solchen Fall ist ein Operator, der einerseits die Vorbedingung P hat, aber andererseits auch den Effekt P, obwohl er die Gültigkeit von P nicht selbst bewirkt. Gegen solche Fehler ist der SNLP-ALgorithmus nicht resistent.

Es wurde bereits bei der Beschreibung von Operatordefinitionen in CAPlan (vgl. Kap. 5.3.1) erläutert, daß Axiome über den verwendeten Prädikatssymbolen zur Beschreibung von Effekten bei der Herleitung von Effekten sowie der Identifizierung spezieller Teilziele (Ziele zum Binden von Variablen) verwendet werden. Gerade der letzte Punkt kann die Suche nach der Lösung u.U. erheblich beschleunigen, da für solche Ziele, die letztlich Anwendbarkeitsbedingungen repräsentieren, die Menge der möglichen Operatoren, aus denen gewählt werden muß, keiner ist. Außerdem hat es sich als lohnenswert herausgestellt, diese Ziele so früh wie möglich zu bearbeiten, da dadurch in der Regel viele Interaktionen verhindert werden.

6.5 Ausblick

Weitere Arbeiten im Zusammenhang mit CAPlan beziehen sich momentan in erster Linie auf die Entwicklung und Integration weiterer Kontrollkomponenten für die Steuerung der Suche.

Konkret wird momentan an zwei anderen Kontrollkomponenten gearbeitet:

CbC (Case-based Control): Die Lösungssuche wird mit Hilfe von geeigneten Fallbeispielen gesteuert. Werner Schirp arbeitet daran, Lösungen von Planungsproblemen als Fälle zu speichern und bei neuen Planungsproblem wiederverwenden zu können. Er setzt dabei mit Erweiterungen bei REDUX an; dies hat u.a. auch den Vorteil, daß andere, auf REDUX aufbauende Systeme wie das Konfigurationssystem IDAX, diese Mechanismen so einsetzen können.

Kontrollregeln: An einer Steuerung der Planung mit Hilfe von Kontrollregeln arbeitet zur Zeit Vera Kettnaker. Hier ist es Ziel, im Laufe einer Trainingsphase eine Reihe von Regeln zu generieren, die die Task-Auswahl sowie die Operator-Auswahl steuern.

Anhang A

Beispiele

Im folgenden soll anhand von zwei Beispielen die Vorgehensweise des CAPlan-Systems bei der Planung illustriert werden. Dabei wird wesentlich auf die vom System zur Verfügung gestellten Visualisierungsmöglichkeiten zurückgegriffen, die in Form einer Reihe von Abbildungen aus verschiedenen Planungsstadien zeigen, wie die in den vorangegangenen Kapiteln beschriebenen Verfahren hier realisiert sind.

A.1 Beispiel 1 – Sussman-Anomalie

In diesem Abschnitt geht es um die schon in Kapitel 2.3.2 angesprochene Sussman-Anomalie; die Problemstellung ist nochmal in Abbildung A.1 gezeigt.

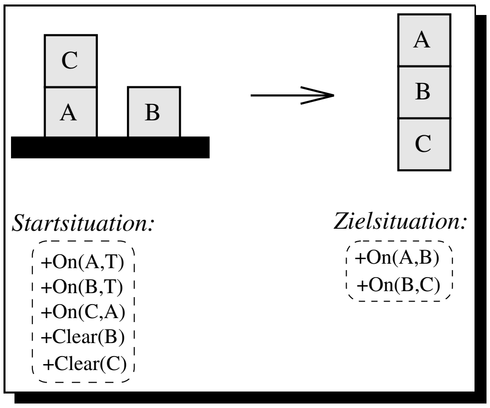

Abbildung A.1: Problemstellung bei der Sussman-Anomalie

Anhand von einigen Abbildungen wird nun gezeigt, wie das CAPlan-System dieses Problem lößt. Es wird im wesentlichen der *System Browser* des CAPlan-Systems in verschiedenen Stadien während der Lösungssuche gezeigt.

Abbildung A.2 zeigt schematisch, was diese Benutzerschnittstelle an Informationen über den Zustand des Planes liefert:

- Oben links gibt es zwei Listen, in denen die noch zu bearbeitetenden Tasks enthalten sind. Es wird zwischen normalen Tasks, welche sich auf Ziele beziehen, die für eine Vorbedingung eines Schrittes einen Causal Link erzeugen sollen, und Meta-Tasks unterschieden; letztere beziehen sich ausschließlich auf Ziele zum Auflösen von Teilzielrekursionen oder zum Schützen von Causal Links.

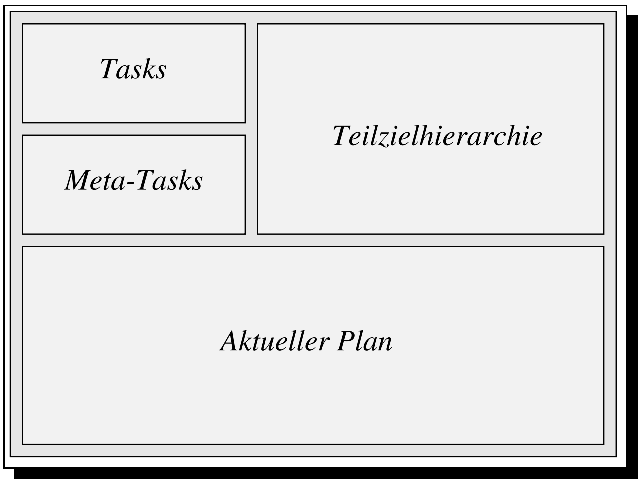

Abbildung A.2: System Browser von CAPlan (schematisch)

- Oben rechts ist einen komprimierte Version der momentanen Teilzielhierarchie zu sehen. Sie ist komprimiert, weil dort nur ein Teil der Teilzielhierarchie gezeigt wird; alle Ziele vom Typ ProtectionGoal sind aus Gründen der besseren Übersicht hier ausgeblendet, so daß nur Ziele, die direkt durch Operatoranwendung hergeleitet wurden hier enthalten sind.

- Im unteren Bereich des Fensters sieht man schließlich den aus Planschritten, Causal Links und Protections bestehenden momentanen Plan.

Abbildung A.3 zeigt zunächst die Initialisierungsphase: Das initiale Ziel *PlanningGoal(Sussman)* wurde reduziert und hat im Plan die Schritte START und FINISH erzeugt sowie beiden Vorbedingungen `+On(a,b)` und `+On(b,c)` von FINISH als neue Teilziele hergeleitet. Das System hat für jedes dieser noch unreduzierten Teilziel einen entsprechenden Task auf der Agenda.

Abbildung A.4 zeigt den Zustand, nachdem einige Inferenzschritte gemacht wurden, welche offene Ziele bearbeitet haben und dabei sowohl den Plan als auch die Teilzielhierarchie verändert haben. Im Plan wurden z.B. bereits die beiden Planschritte STEP-0 und STEP-1[1] eingeführt und ein Teil der Vorbedingungen wurde durch Causal Links erfüllt.

In Abbildung A.5 sieht man das System in einem Zustand, in dem es neben noch drei nicht reduzierten Zielen zwei Threats erkannt hat, die bearbeitet werden müssen (zwei Meta-Tasks auf der Agenda). Der erste Meta-Task (in der Abbildung selektiert) betrifft eine Bedrohung des Causal Link $START \stackrel{+Clear(x4)}{\longrightarrow} STEP-1$ durch den Schritt STEP-2. Konkret bedroht hier der Effekt `-Clear(c)` von STEP-2 den Causal Link. Dieser Threat kann nur dadurch aufgelöst werden, daß die Ordnung (*Protection*) $STEP\text{-}1 \prec STEP\text{-}2$ eingeführt wird (s. Abb. A.7).

Ziele zum Schützen von Causal Links (ProtectionGoal) gehören ebenfalls zur Teilzielhierarchie, da sie jeweils als Unterziele des Zieles erzeugt werden, dessen Abarbeitung den zu schützenden Causal Link erzeugt hat. Abbildung A.6 zeigt die vollständige Teilzielhierarchie, die zum Zeitpunkt aus Abbildung A.5 besteht; sie zeigt gegenüber der Teilzielhierarchie auch ProtectionGoals an. Selektiert ist wieder das durch den Threat aus Abbildung A.5 erzeugte ProtectionGoal.

Abbildung A.8 zeigt schließlich das System, wenn es das Problem gelöst hat; es gibt keine Tasks mehr, die noch bearbeitet werden müßten.

[1] Die Namensgebung der Planschritte geschieht automatisch durch das Hochzähen eines Zählers

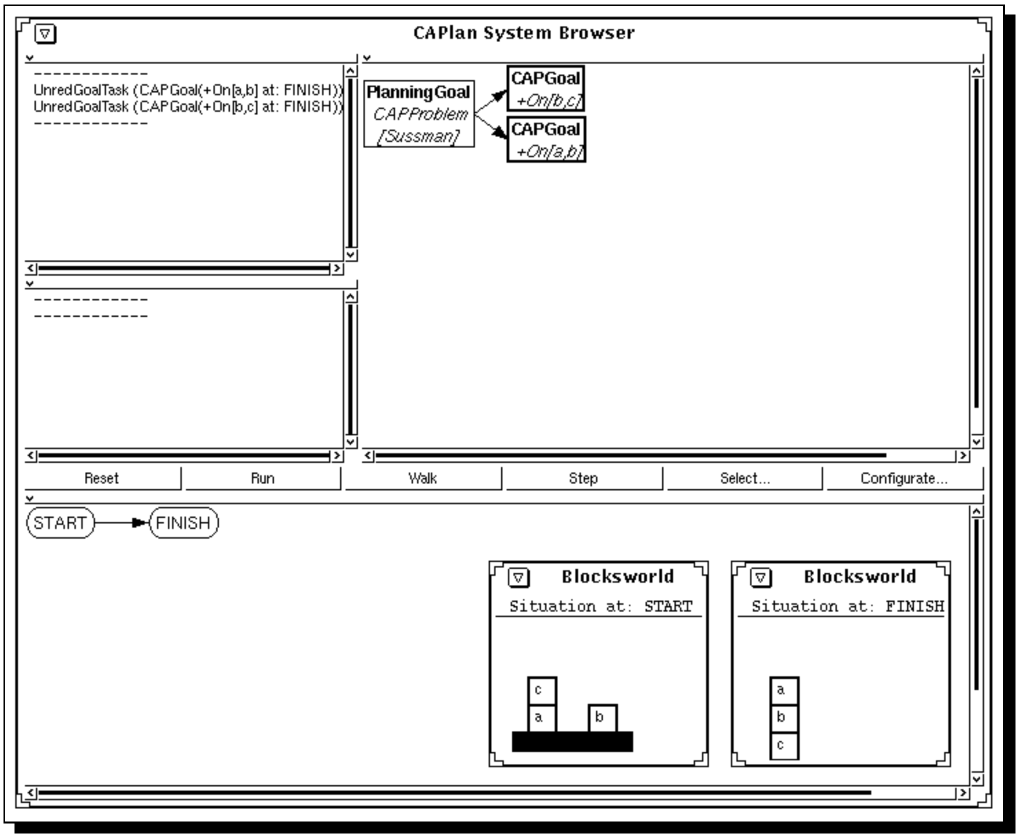

Abbildung A.3: Sussman-Anomalie – Initialisierung

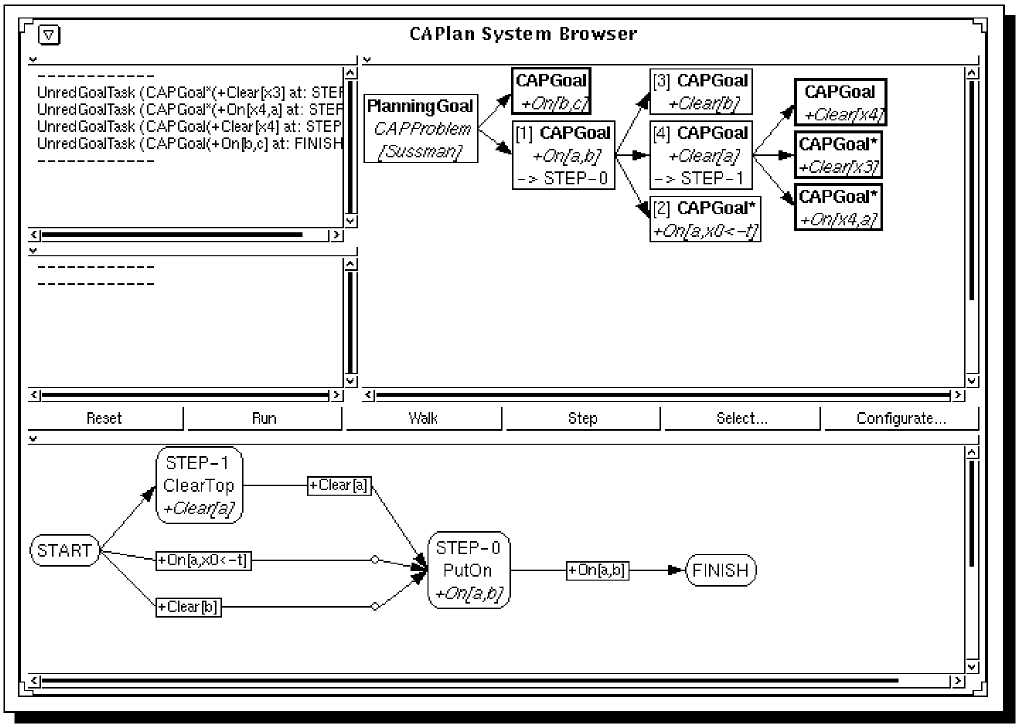

Abbildung A.4: Sussman-Anomalie – Inferenzschritt

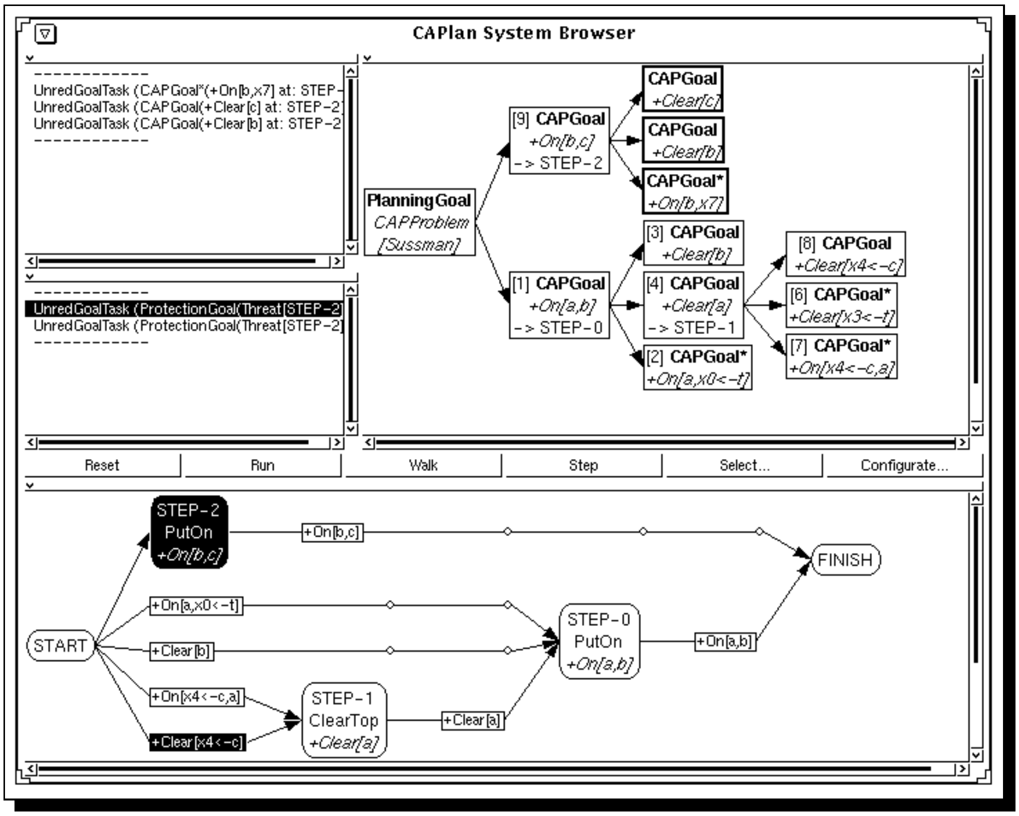

Abbildung A.5: Sussman-Anomalie – Interaktionen

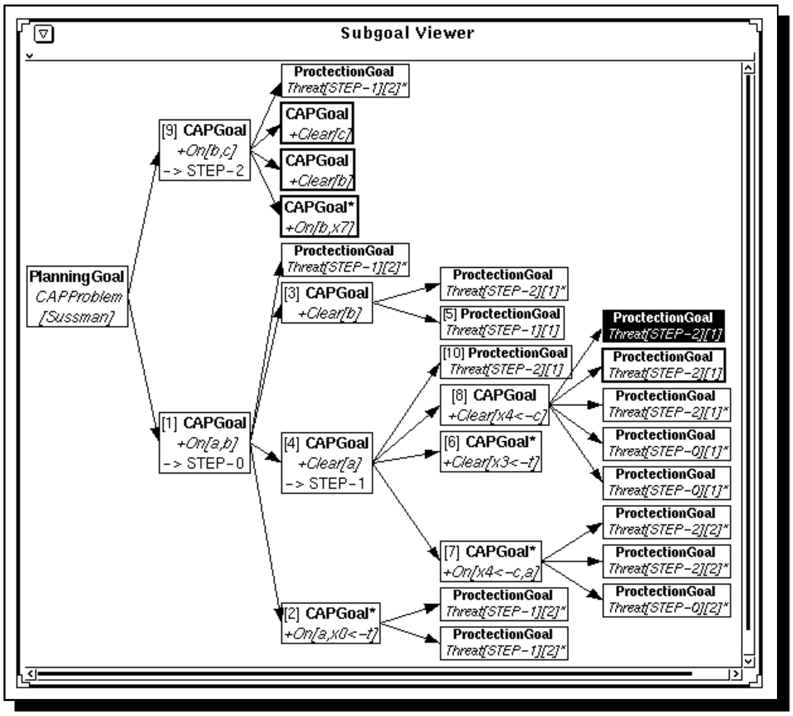

Abbildung A.6: Sussman-Anomalie – Vollständige Teilzielhierarchie

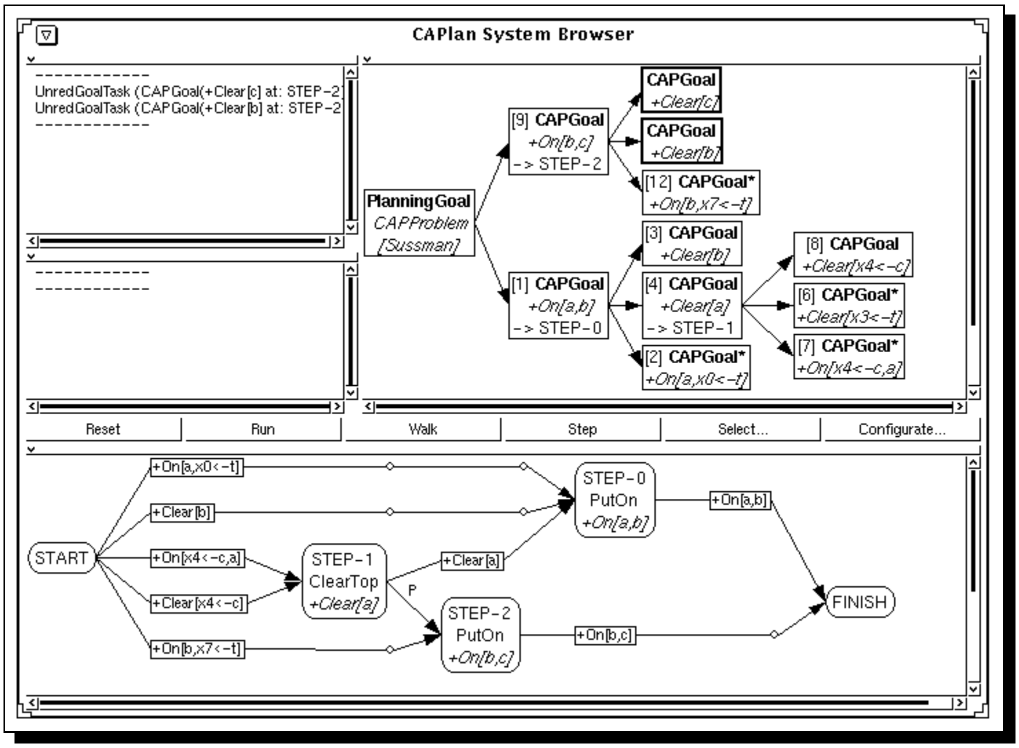

Abbildung A.7: Sussman-Anomalie – Interaktionsauflösung

A.2 Beispiel 2

Hier soll noch an einem zweiten Beispiel ein anderer Aspekt veranschaulicht werden. Abbildung A.9 zeigt die Problemstellung. Die Schwierigkeit hier ist, daß eines der zu erreichenden Teilziele der Zielsituation (`+On(a,b)`) bereits in der initialen Situation erfüllt ist, der Planer dieses Ziel jedoch nicht direkt phantomisieren kann, da er es zur Erreichung des zweiten Teilziels wieder zerstören muß. D.h. in dem Fall, daß `+On(a,b)` zunächst phantomisiert wird, kommt es zu Backtracking.

Abbildung A.10 zeigt den System Browser, nachdem das System für das Problem initialisiert wurde und nach einigen Inferenzschritten der kritische Fehler gemacht wurde, das Ziel `+On(a,b)` zu phantomisieren (Causal Link $START \overset{+On(a,b)}{\longrightarrow} FINISH$).

In Abbildung A.11 wurden nun einige weitere Inferenzschritte gemacht, die aber schließlich zu einer Zielblockade führen, welche im System Browser durch den *Goal-Block-Task* in der Agenda repräsentiert ist. Dessen Abarbeitung führt nun zu Backtracking.

In Abbildung A.12 ist das Problem schließlich gelöst, da wieder alle Tasks abgearbeitet wurden.

Was nun noch gezeigt werden soll, gibt einen Eindruck davon, wie groß die Suchräume selbst bei relativ einfachen Problemstellungen werden können. Die Fehlentscheidung aus Abbildung A.10 muß vom Planer als solche erkannt werden. Dies geschieht dadurch, daß sie schließlich durch einen Backtracking-Schritt wieder zurückgezogen wird.

Abbildung A.13 zeigt den Suchraum, der insgesamt bis zur Lösung aus Abbildung A.12 durchsucht wurde. Knoten in diesem Baum repräsentieren wieder Ziele, die zur Abarbeitung ausgewählt wurden. Kanten stehen für Entscheidungen, einen speziellen Operator anzuwenden. Blätter in diesem Baum stehen für Zielblockaden, die zu Backtracking geführt haben, bis schließlich die Lösung gefunden wurde. Durchlaufen dieses Baumes (wie bereits in Kapitel 3.4.2 beschrieben) veranschaulicht den Suchprozeß. Was man hier gut sieht ist, daß der größte Teil dieses Baumes nur durchlaufen werden mußte, weil ziemlich früh eine falsche Entscheidung getroffen wurde.

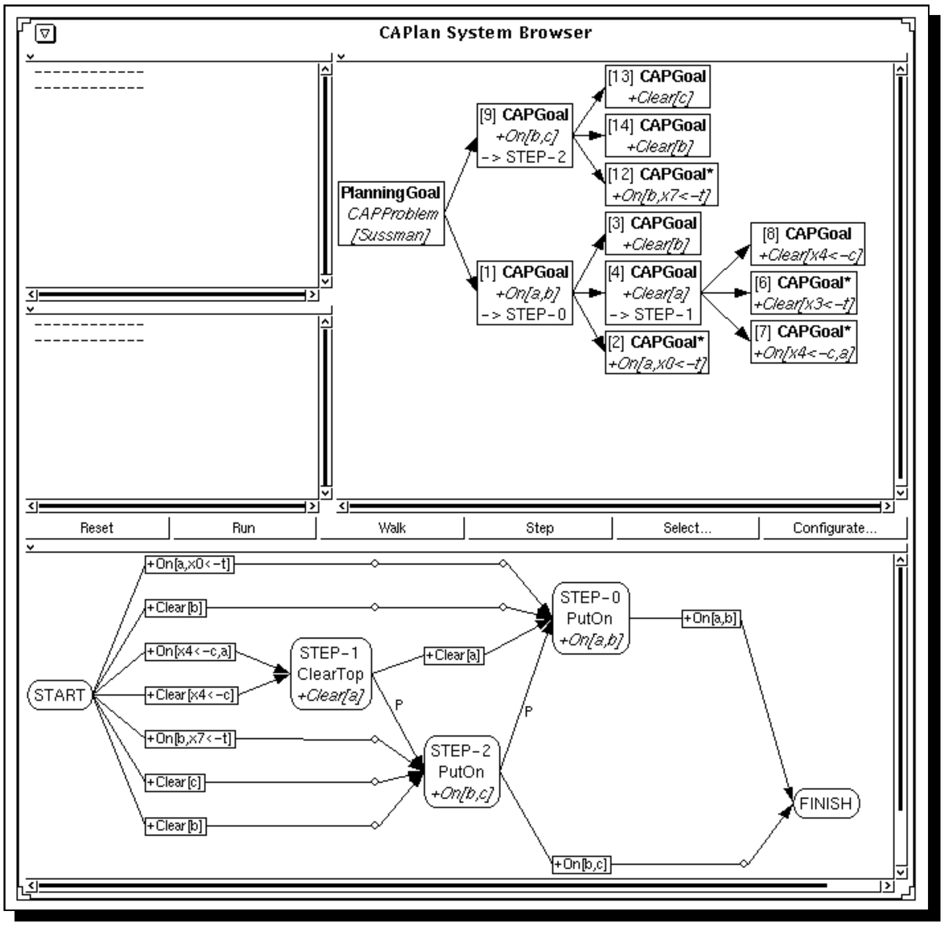

Abbildung A.8: Sussman-Anomalie – Lösung

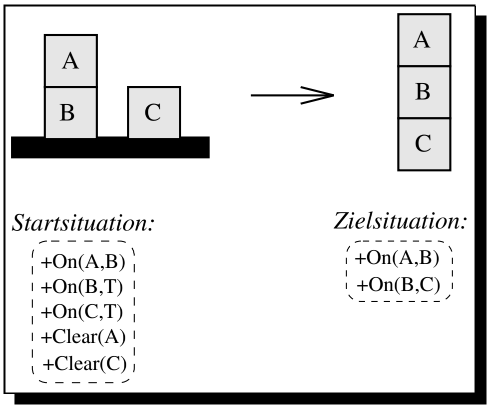

Abbildung A.9: Problemstellung von Beispiel 2

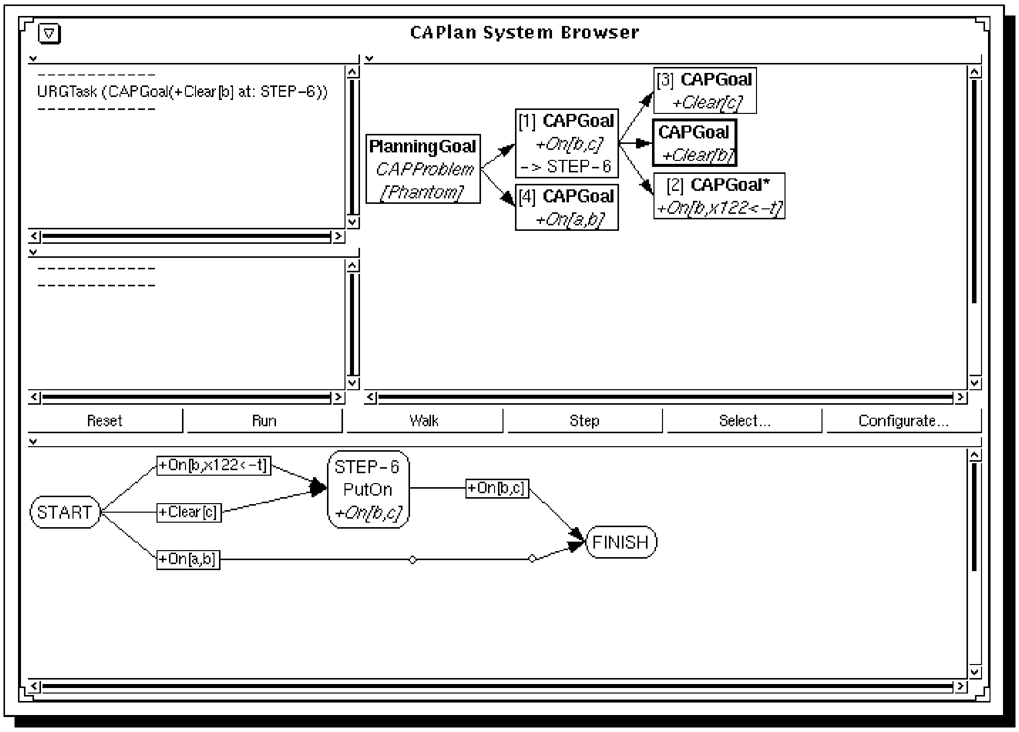

Abbildung A.10: Beispiel 2 – Fehlentscheidung

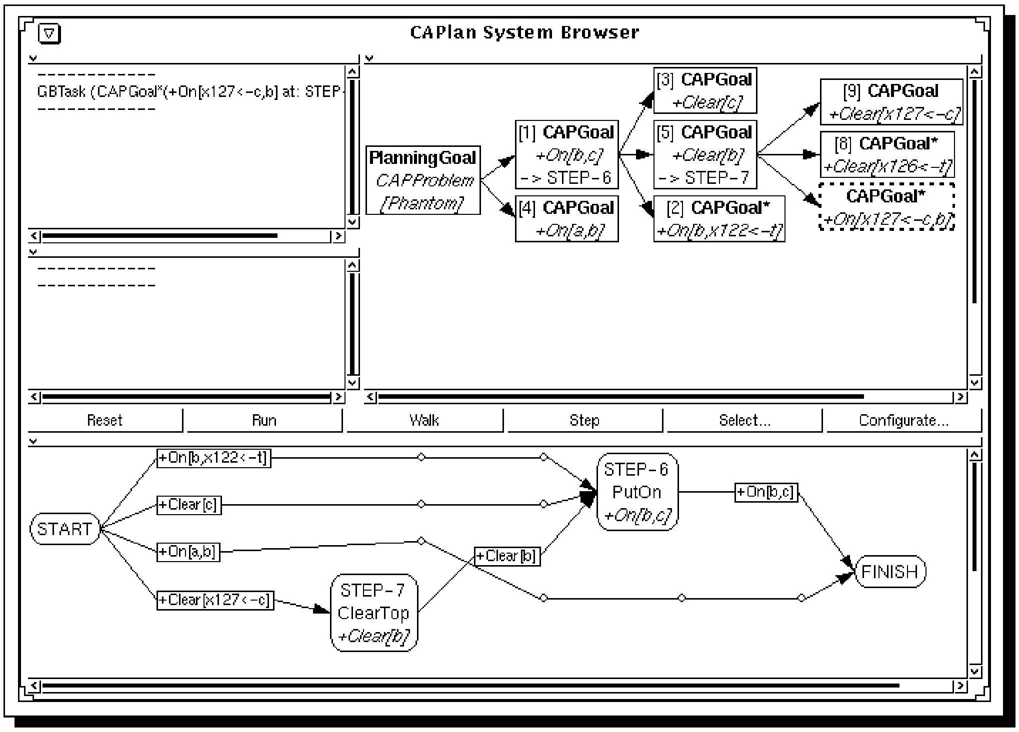

Abbildung A.11: Beispiel 2 – Erste Zielblockade

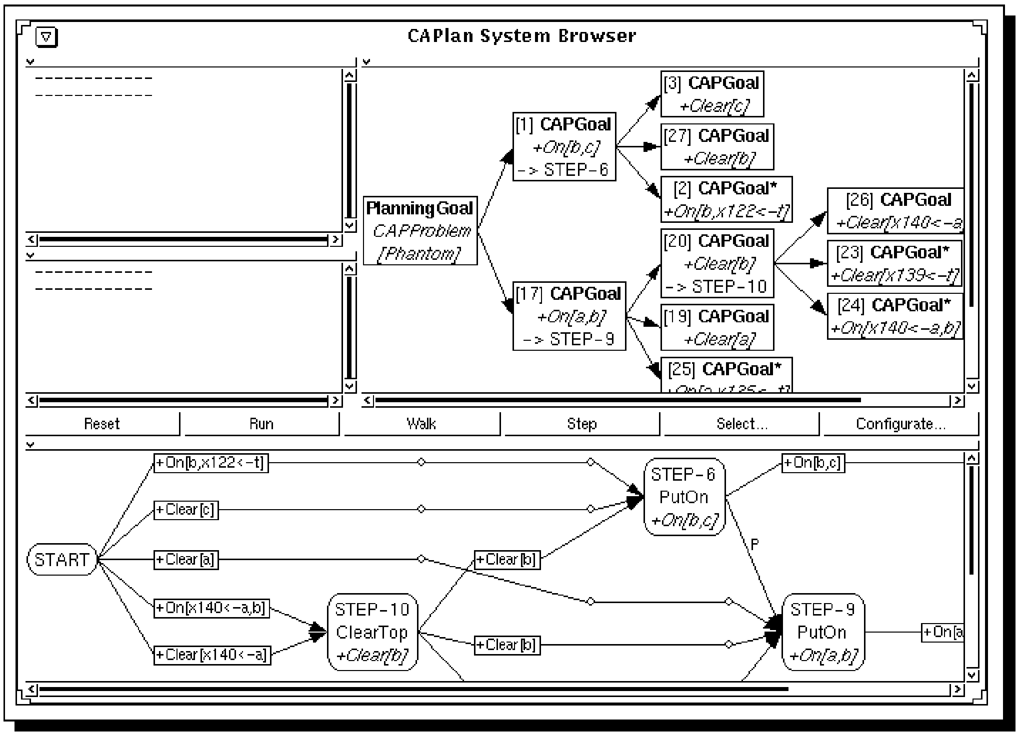

Abbildung A.12: Beispiel 2 – Lösung

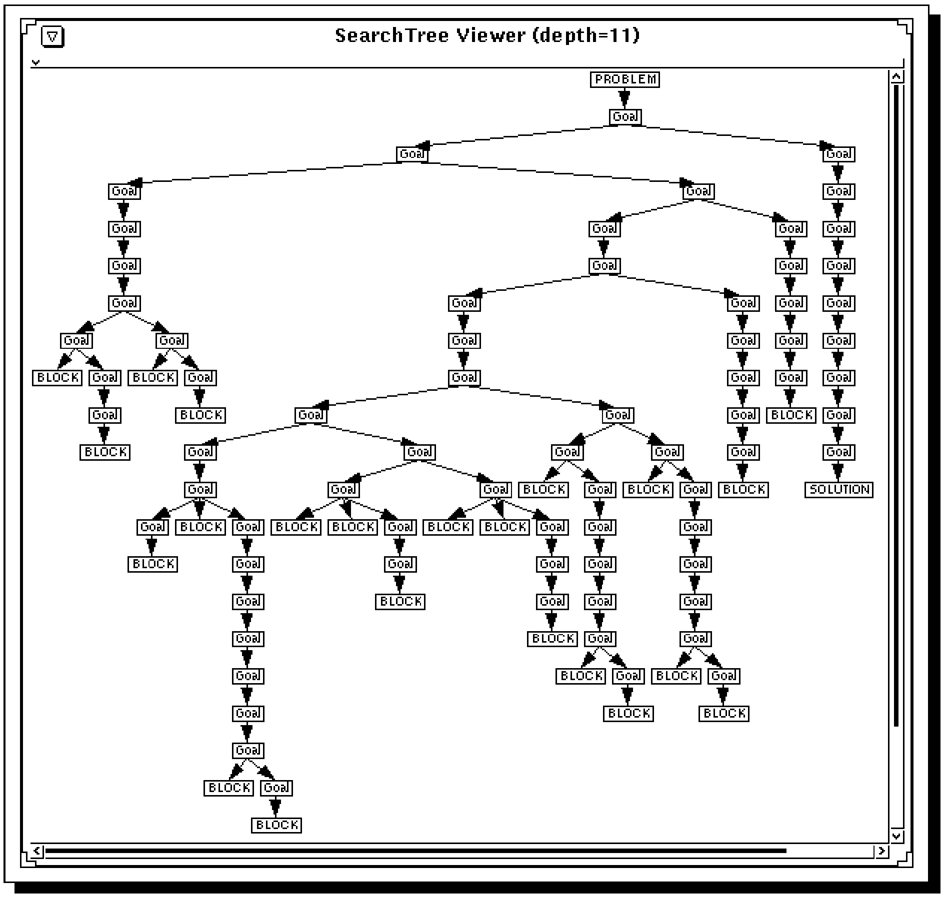

Abbildung A.13: Beispiel 2 – Visualisiserung des Suchraumes

Anhang B

Danksagung

An dieser Stelle möchte ich allen denen danken, die mir während der Implementations- und Dokumentationsphase dieser Arbeit mit Rat und Tat zur Seite gestanden haben, Prof. M. M. Richter, Jürgen Paulokat und allen Mitarbeitern und Hiwis der Arbeitsgruppe Expertensysteme.

Inbesondere geht mein Dank auch an Barbara Dellen, Christiane Gresse, Holger Hoch, Vera Kettnaker, Hector Munioz, Reinhard Präger, Jürgen Rahmel, Werner Schirp, Holger Wache, Aldo von Wangenheim, Thomas Wirz und Max Wolf.

Literaturverzeichnis

[BV92] J. Blythe und M. Veloso. An Analysis of Search Techniques for a Totally-Ordered Nonlinear Planner. Aus: *Proceedings of 1st International Conference on AI Planning Systems*, Seite: 13–19, 1992.

[BW93] A. Barrett und D.S. Weld. Partial-Order Planning: Evaluating Possible Efficiency Gains. Technischer Bericht, Dep. of Computer Science and Engineering, University of Washington, Seattle, WA 98195, 1993.

[Cha87] D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–337, 1987.

[Doy79] J. Doyle. A Truth Maintenance System. *Artificial Intelligence*, 1979.

[FHN72] R.E. Fikes, P.E. Hart und N.J. Nilsson. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, 3:251–288, 1972.

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

[Geo87] M.P. Georgeff. Planning. *Annual Review of Computer Science*, 2:359–400, 1987.

[GR83] A. Goldberg und D. Robinson. *Smalltalk 80 - The Language and Its Implementation.* Addison-Wesley, 1983.

[Kam92] S. Kambhampati. Characterizing Multi-Contributor Causal Structures for Planning. Aus: *Proceedings of 1st International Conference on AI Planning Systems*, Seite: 116–125, 1992.

[Kor87] R.E. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, 33:65–88, 1987.

[Lif87] V. Lifschitz. On the Semantics of STRIPS. Aus: *Reasoning about Actions and Plans: Proceedings of 1986 Workshop*, Timberline, Oregon, 1987.

[MR91] D. McAllester und D. Rosenblitt. Systematic Nonlinear Planning. Aus: *Proceedings of AAAI-91*, Seite: 634–639, 1991.

[Pet91] Ch. Petrie. *Planning and Replanning with Reason Maintenance.* Dissertation, University of Texas/Austin, 1991.

[Pet92] Ch. Petrie. Constrained Decision Revision. *Proceedings of AAAI-92*, Seite: 393–400, 1992.

[Ric92] M.M. Richter. *Prinzipien der künstlichen Intelligenz.* 2. Auflage. B.G. Teubner, Stuttgart, 1992.

[Rit92] H. Ritzer. *Konzeption und Implementierung einer TMS-basierten Komponente zur Verwaltung von Abhängigkeiten bei der Konfiguration und Planung.* Diplomarbeit, Universität Kaiserslautern, 1992.

[Sac75] E. Sacerdoti. The Nonlinear Nature of Plans. Aus: *Proceedings of IJCAI-75*, Seite: 206–214, 1975.

[Sus75] G. Sussman. *A Computer Model of Skill Acquisition.* American Elsevier, New York, 1975.

[Tat77] A. Tate. Generating Project Networks. Aus: *IJCAI-77*, Seite: 888–893, 1977.

[Vel92] M.M. Veloso. *Learning by Analogical Reasoning in General Problem Solving.* Dissertation, Carnegie Mellon University, Pittsburgh, PA, 1992.

[Wal77] R. Waldinger. Archieving Several Goals Simultaneously. *Machine Intelligence*, 8, 1977.

[Wil84] D.E. Wilkins. Domain-independent Planning: Representation and Plan Generation. *Artificial Intelligence*, 22:269–301, 1984.

[Wil88] D. E. Wilkins. *Practical Planning - Extending the classical AI Planning Paradigm.* Morgan Kaufmann, 1988.