Diplomarbeit

Realisierung hierarchischer Aktionsdekomposition im Planungswerkzeug CAPlan zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM)

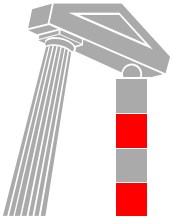
Christoph Keller

Juni 1996

Betreuung: Prof. Dr. Michael M. Richter
Dipl. Inform. Frank Weberskirch

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

Universität Kaiserslautern
Fachbereich Informatik



Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein umfangreiches Gebiet der Künstlichen Intelligenz beschäftigt sich mit dem Bereich Planung. Im wesentlichen gibt es zwei Planungsansätze, zum einen nicht-hierarchische und zum anderen hierarchische arbeitende Verfahren. Als Beispiel für einen nicht-hierarchischen Ansatz kann SNLP[1] genannt werden. Die nachfolgende Ausarbeitung ist auf dem zweiten Gebiet, der hierarchischen Planung, angesiedelt: Der Planungsassistent CAPLAN[2], der eigentlich auf dem nicht-hierarchischen Planungsansatz SNLP beruht, soll um die Möglichkeiten der hierarchischen Planung erweitert werden.

1.1 Der Planungsassistent CAPLAN

Der Planungsassistent CAPLAN [Weberskirch, 1994; Weberskirch, 1995] ist ein domänenunabhängiges Planungssystem. Hierunter versteht man ein System, in dem mittels einer geeigneten Beschreibungssprache, die Planungswelt (Domäne) mit ihren Zuständen, Objekten und Aktionen modelliert wird. In CAPLAN wird dabei der SNLP-Ansatz mit der Abhängigkeitsverwaltung REDUX kombiniert. REDUX stellt für die Planung und Konfiguration eine geeignete Struktur von Abhängigkeiten zur Verfügung.

Der SNLP-Ansatz ist einer der vielbeachtesten Planungsalgorithmen und auch fünf Jahre nach seiner Veröffentlichung in [McAllester und Rosenblitt, 1991] immer noch Gegenstand einer Vielzahl von Veröffentlichungen. Der SNLP-Algorithmus gehört zur Sparte der partiell ordnenden, least-commitment Planer, d.h. der Algorithmus operiert auf nur partiell geordneten Plänen, wodurch Planer dieser Art einige Freiheiten bei der Modifikation von Plänen haben.

Das CAPLAN-System ist als domänenunabhängiges System konzipiert, sein Hauptanwendungsbereich liegt jedoch in der computerintegrierten Fertigung (CIM). Mittels einer Domänenbeschreibung in diesem Anwendungsbereich werden dort für rotationssymetrische Drehteile die entsprechenden Arbeitspläne für die Fertigung dieser Teile auf CNC-Maschinen vom System erstellt. Dies stellt aber nur einen speziellen Anwendungsfall für CAPLAN dar. Die Möglichkeiten der Domänenmodellierung sind dabei gegenüber SNLP leicht erweitert. Das System ist so konzipiert, daß es im Dialog mit dem Benutzer die Plangenerierung vollziehen kann. Der Benutzer hat immer die Möglichkeit, in alle planungsrelevanten Entscheidungsprozesse sowohl führend als auch korrigierend einzugreifen. Dadurch können auch komplexe

[1] SNLP: Systematic Non Linear Planning
[2] CAPLAN: Computer Aided Plannning

Planungsaufgaben vom System besser mit Hilfe des Benutzers gelöst werden, da so das Benutzerwissen direkt in dem Planungsprozeß intergriert werden kann.

Die Steuerung des Planungsprozesses geschieht mit Hilfe von frei definierbaren Kontrollkomponenten, die im einfachsten Fall das SNLP-Verfahren realisieren. Neben vollautomatischen Kontrollstrategien, die im wesentlichen auf SNLP basieren, wird im Rahmen der fallbasierten Planung mit CAPlan/CbC [Muñoz-Avila *et al.*, 1995] auch Fallwissen, in Form gelöster Planungsaufgaben, zur Steuerung der Lösungssuche mitverwendet.

1.2 Hierarchisches Planen

In der hierarchischen Planung stehen zwei Fragestellungen im Vordergrund. Die erste Fragestellung beschäftigt sich mit der in der Planung allgemein üblichen Frage: „Wie erreiche ich ein gegebenes Ziel?“. Kann dies durch die Auswahl einer konkreten Aktion geschehen, oder ist es möglich, mit komplexen Aktionen Strategien zur Erfüllung eines Zieles zu definieren? Die zweite Frage, die diesen Ansatz vom nicht-hierarchischen Planen unterscheidet, ist: „Wie führe ich eine konkrete Aktion aus?“. Während man bei nicht-hierarchischen Planern bei Aktionen nur von primitiven Aktionen spricht, die in der Planungswelt direkt ausführbar sind, kommen beim hierarchischen Planen noch sogenannte komplexe Aktionen hinzu, deren direkte Ausführbarkeit aber nicht mehr verlangt wird. Stattdessen wird die konkrete Art der Ausführung einer komplexen Aktion im Laufe des Planungsprozesses bestimmt. Dabei werden unter direkt ausführbaren Aktionen solche verstanden, die direkte Zustandsänderungen in der Planungswelt bewirken.

Durch diese Aspekte ergibt sich, daß hierarchische Ansätze eine Reihe von Vorteilen gegenüber den nicht-hierarchischen Ansätzen haben.

- Unter dem Aspekt der Ausführung von Aktionen lassen sich wesentlich komplexere Zusammenhänge einfacher modellieren, d.h. es ist kann mehr planungsrelevantes Wissen in solche Aktionen eingebunden werden. Es ist z.B. möglich komplexe Aktionen, die aus einer Vielzahl von zueinander angeordneten Einzelaktionen bestehen, zu modellieren. Insbesondere die Anordnung der Einzelaktionen läßt sich mit nicht-hierarchischen Konzepten nicht oder nur sehr unzureichend modellieren.
- Zum anderen kann die Lösung des Planungsproblem auf verschiedenen abstrakteren Planungsstufen betrachtet werden, da mit der Definition von komplexen Aktionen erst die relevanten Aspekte der Problemspezifikation gelöst werden können. Die Schlüsselbegriffe sind hier Abstraktion und Konkretisierung der Lösungssuche.
- Vor dem Hintergrund der Verwendung im Planungsassistenten CAPlan ergibt sich daraus, daß insbesondere auch (teilweise) interaktives Planen für den Benutzer wesentlich vereinfacht werden kann, wenn ihm auch komplexe Aktionen zum Erreichen von Planungszielen zur Verfügung stehen.

Mit den genannten Vorteilen läßt sich erwarten, daß hierarchisch arbeitende Planungssysteme komplexe Problemstellungen effizienter lösen als nicht-hierarchische Planungssysteme.

Ein Problem bei älteren hierarchischen Planungsansätzen war, daß die verwendeten Konzepte nicht oder nur unzureichend formalisiert wurden. Es wurden meist nur syntaktische Kriterien betrachtet, eine Formalisierung der semantischen Aspekte der hierarchischen Planung wurde nicht vorgenommen. Somit konnten die Eigenschaften dieser Ansätze nicht immer bewiesen

werden. In der jüngeren Vergangenheit, wurde ein formaler Ansatz durch die Beschreibung des *Hierachical-Task-Networks* von [Erol *et al.*, 1994] vorgenommen. Mittels der dort definierten Konzepte zeigt Erol einige wichtige Eigenschaften des HTN-Ansatzes, wie Vollständigkeit, Korrektheit, Systematik und eine größere Ausdrucksfähigkeit gegenüber nicht-hierarchischen Verfahren. Eine große Klasse von hierarchischen Ansätzen läßt sich dabei mit den von Erol beschriebenen Konzepten erfassen.

1.3 Zielsetzung der Arbeit

Die Arbeit beschäftigt sich mit der Realisierung von hierachischer Aktionsdekomposition im Planungsassistenten CAPLAN [Weberskirch, 1994; Weberskirch, 1995]. Da CAPLAN auf dem partiell-ordnenden Planungsansatz SNLP basiert, war eine möglichst generische Erweiterung des Systems beabsichtigt, die kein komplett neues System entstehen läßt, sondern ein um den Aspekt Aktionsdekomposition erweitertes CAPLAN.

Die Zielsetzungen dieser Arbeit waren im einzelnen:

- Konzeptionelle Erweiterung der bestehenden Domänenmodellierungsmöglichkeiten des CAPLAN-Systems um sogenannte *Composite Actions*, d.h. komplexe, nicht-primitive Aktionen, die aus einem Netz von mehreren Einzelaktionen bestehen.
- Integration der komplexen Aktionen in den Verarbeitungsmechanismus des CAPLAN-Systems, insbesondere die Realisierung auf Basis des REDUX-Systems.
- Implementierung der Aktionsdekomposition im System.
- Diskussion des neuen Verfahrens gegenüber dem bisherigen Basisverfahren SNLP anhand verschiedener Domänen.

Die Einbindung der Konzepte in das CAPLAN-System sollte dabei unter Berücksichtigung der bereits realisierten SNLP-Konzepte geschehen. Daher wurde DPOCL von [Young *et al.*, 1994] zur Erweiterung des Systems gewählt, da dieser Ansatz SNLP um die Elemente von hierarchischer Aktionsdekomposition erweitert. Damit soll das System in die Lage versetzt werden, auch komplexere Probleme effizient zu lösen, um damit eine Verbesserung der Planung zu erreichen. Bei der Realisierung der neuen Konzepte auf der Basis des REDUX-Systems war auch hier eine generische, d.h. die bestehenden Konzepte möglichst nicht verändernde, Einbindung gewünscht.

1.4 Übersicht über die Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

- Im Kapitel 2 werden wesentliche Grundlagen für diese Arbeit dargestellt. Hierunter fallen der SNLP-Ansatz [McAllester und Rosenblitt, 1991], das CAPLAN-System [Weberskirch, 1995] sowie einige Konzepte von REDUX.
- Kapitel 3 nimmt eine Einführung in den Begriff des hierarchischen Planens vor und erläutert einige alternative Konzepte.

- In Kapitel 4 werden die Konzepte des im Rahmen dieser Arbeit näher untersuchten DPOCL-Algorithmus [Young *et al.*, 1994] zur Erweiterung von SNLP um hierarchische Aktionsdekomposition vorgestellt.
- Kapitel 5 beschreibt wichtige Aspekte bei der Realisierung der DPOCL-Konzepte auf der Basis von REDUX im CAPLAN-System.
- Kapitel 6 gibt einige Beobachtungen bezüglich der Leistungsfähigkeit der implementierten Erweiterungen wieder und faßt die Ergebnisse der Arbeit zusammen.

Das CAPLAN-System und die in dieser Arbeit vorgestellte Erweiterung wurde in der Programmiersprache Smalltalk, konkret VISUALWORKS 2, implementiert. Die wesentlichen Vorteile dieser Art der Implementierung mit einer objektorientierten Programmiersprache, wie beispielsweise Smalltalk, liegen in der übersichtlicheren Abbildung der Konzepte in der Programmstruktur. Zum Verständnis der Implementierung sind gründliche Kenntnisse der Sprache Voraussetzung. Empfohlene Literatur hierfür ist, z.B. [Goldberg und Robinson, 1983]. Weiterführende Informationen zum CAPlan-System, seine Implementierung und den SNLP-Ansatz finden sich in [Weberskirch, 1994; Weberskirch, 1995].

Kapitel 2

Grundlagen

Im folgenden Kapitel werden grundlegende Begriffe und Sachverhalte vorgestellt, die den Ausgangspunkt für die Ausarbeitung bilden. Dazu gehören der SNLP-Ansatzes und das auf SNLP und REDUX basierende CAPLAN-System, in das die Ergebnisse dieser Arbeit eingebunden wurden.

2.1 Der SNLP-Ansatz

Eine der wesentlichen Grundlagen der nachfolgenden Kapitel bildet der vieldiskutierte Ansatz zur *systematischen nicht-linearen Planung* SNLP [McAllester und Rosenblitt, 1991]. Der dort beschriebene Algorithmus ist domänenunabhängig, vollständig und korrekt. Darüber hinaus besitzt er die Eigenschaft, den Suchraum möglicher Pläne zu einem spezifizierten Problem systematisch zu durchsuchen, d.h. es ist garantiert, daß jeder mögliche Planungszustand höchstens einmal erzeugt wird.[1]

In den nachfolgenden Abschnitten werden die wesentlichen Konzepte des Ansatzes und der SNLP-Algorithmus erläutert.

2.1.1 Domänenbeschreibung und Problemspezifikation

Der SNLP-Ansatz ist domänenunabhängig. Eine Domänenbeschreibung ist daher auch Bestandteil einer Problembeschreibung und legt die Aktionen fest, die der Planer ausführen kann, um das gegebene Problem zu lösen. Desweiteren legt sie eine Beschreibungssprache für die Merkmale und Zustände der Planungswelt fest.

In der Domänenbeschreibung werden Aktionen durch STRIPS[2]-ähnliche Operatoren beschrieben. Diese Operatoren haben eine Reihe von Vorbedingungen, die zur Ausführungszeit der Aktion gelten müssen. Daneben werden die von den Aktionen bewirkten Veränderungen der Planungswelt durch eine Menge von Effekten beschrieben. Für die Beschreibung der Vorbedingungen und Effekte werden funktionsfreie Atomformeln verwendet. In der Regel hat ein solcher Operator daher noch eine Anzahl von Variablenbindungs-Constraints, welche die Bindung der in den Atomformeln verwendeten Variablen beschreiben.

[1] Die Systematik der Suche ist jedoch ein Umstand, der von vielen nicht als unumstrittener Vorteil gewertet wird. Beobachtungen zeigen sogar, daß die Einhaltung der Systematikeigenschaft u.U. erheblichen Mehraufwand bei der Planung verursacht. [Knoblock und Yang, 1995] zeigt einige Untersuchungen zu diesem Thema.

[2] Nähere Informationen zu STRIPS finden sich in [Fikes und Nilsson, 1971; Fikes *et al.*, 1972].

Die Aufgabenstellung bei der Planung ist es für ein gegebenes Problem eine Sequenz von Aktionen anzugeben. Diese Sequenz von Aktionen, in der angegeben Reihenfolge ausgeführt, überführt die Startsituation in die Zielsituation und löst damit das Planungsproblem. Die zu lösende Aufgabe (Zielsituation) wird bei SNLP durch eine Liste von Zielen repräsentiert. Diese Ziele lassen sich durch die Effekte der Aktionen, die in der Domänenbeschreibung festgelegt sind, erreichen. Eine Problembeschreibung umfaßt bei SNLP immer eine Beschreibung der initialen Situation sowie die zu erreichenden Ziele. Beides geschieht durch die Angabe einer Reihe von Atomformeln. Diese Atomformeln werden im weiteren auch als Prädikate bezeichnet.

2.1.2 Darstellung von Plänen

Der SNLP-Algorithmus erzeugt zu einer gegebenen Problembeschreibung durch eine systematische Suche im Lösungsraum einen Plan. Ein solcher Plan beschreibt eine Menge von auszuführenden Aktionen sowie deren Reihenfolge. Das Ziel bei der Erzeugung der Pläne ist es, einen Plan zu generieren, dessen Abfolge von Aktionen das gegeben Problem löst, d.h. die Startsituation in die Zielsituation überführt. Die Komponenten eines Plans sind bei SNLP wie folgt definiert.

Definition 2.1 (SNLP-Plan)
Sei $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c >$, *wobei*

- $\mathcal{S}$ *für eine Menge von Schritten,*
- $\mathcal{O}$ *für eine Menge von Ordnungen zwischen den Schritten aus* $\mathcal{S}$,
- $\mathcal{B}$ *für eine Menge von Variablenbindungs-Constraints über den freien Variablen der Elemente aus* $\mathcal{S}$ *und*
- $\mathcal{L}_c$ *für eine Menge von Causal Links zwischen den Schritten aus* $\mathcal{S}$

steht. Dann bezeichnet $\mathcal{P}$ *einen SNLP-Plan.*

Auf die in der Definition 2.1 beschriebenen Komponenten eines Planes, Schritte, Ordnungen und Causal Links wird in den anschließenden Abschnitten genauer eingegangen. Die im SNLP-Plan vorkommenden Variablenbindungs-Constraints ergeben sich aufgrund der Tatsache, daß die Schritte in SNLP-Plänen Domänenoperatoren (siehe Abschnitt 2.1.1) repräsentieren. Wie man weiter der Definition entnehmen kann, sind Pläne beim SNLP-Ansatz im allgemeinen nur partiell geordnet.

Planschritte

Ein Planschritt repräsentiert eine primitive Aktion in der Planungswelt. Die Menge der Aktionen wird durch die Operatoren der Planungsdomäne beschrieben. Jeder Planschritt ist dabei eindeutig einem Operator der Planungsdomäne zugeordnet. Dadurch sind dem Planschritt Vorbedingungen, Effekte und eine Menge von Constraints über den von ihm verwendeten Variablen zugeordnet. Es werden zwei Arten von Planschritten unterschieden:

Normale Planschritte: Diese repräsentieren die durchzuführenden Aktionen, um ein bestimmtes Ziel zu erreichen. Ihnen ist ein Domänenoperator zugeordnet und sie tragen einen eindeutigen Namen (z.B. $STEP_1$). Über den Domänenoperator werden Vorbedingungen, Effekte und Variablenbindungs-Constraints festgelegt.

Künstliche Planschritte: Man unterscheidet zwei künstliche Planschritte, START und FINISH. Sie dienen dem Zweck die Problemstellung, d.h. die initiale Situation sowie die Zielsituation, im Plan zu repräsentieren. Hierbei wird dem START-Schritt ein Operator zugewiesen, der keine Vorbedingungen, aber als Effekte die Prädikate, welche die initiale Situation beschreiben, hat. Ähnlich verhält es sich mit dem FINISH-Schritt. Auch er repräsentiert einen speziellen Operator, der als Vorbedingungen die zu erreichenden Ziele hat. Effekte bewirkt dieser Operator keine.

In jedem Plan $\mathcal{P}$ kommen immer die Schritte START und FINISH, sowie beliebig viele normale Planschritte vor.

Ordnungen

Ordnungen zwischen Planschritten sollen garantieren, daß bestimmte Bedingungen zur Ausführungszeit eines Schrittes wahr sind. Man unterscheidet zwei Arten von Ordnungen: normale Ordnungen zwischen Schritten und sogenannte *Protections*. Die Auswirkungen der beiden Ordnungen sind jeweils die gleichen: Beide ordnen zwei Schritte S und T zueinander an. Der Unterschied liegt lediglich im Grund ihrer Einführung in den Plan. Normale Ordnungen dienen in Kombination mit dem SNLP-spezifischen Konzept des Causal Links (vgl. Definition 2.2) dazu, Vorbedingungen von Schritten oder Planungsziele zu erreichen. Protections werden ausschließlich dazu genutzt, Interaktionen (siehe Abschnitt 2.1.3) aufzulösen. Die Notation einer Ordnung, die einen Planschritt S vor einem Planschritt T ordnet, ist $S \prec T$.

2.1.3 Interaktionen zwischen Schritten

Um die Gültigkeit einer Vorbedingung eines Planschrittes bei dessen Ausführung zu garantieren, werden beim SNLP-Ansatz zwei Konzepte verwandt. Einmal wird die Tatsache, daß ein Planschritt die Vorbedingung eines anderen Schrittes erfüllt, festgehalten. Dies geschieht in SNLP mittels des Konzepts des *Causal Links*. Durch ihn wird der Zweck festgehalten, warum ein Schritt in den aktuellen Plan eingeführt wurde.

Definition 2.2 (Causal Link)
Wenn S ein Planschritt mit dem Effekt p ist, der genutzt werden soll um die Vorbedingung p eines anderen Planschritts T zu erfüllen, so hält der Causal Link $S \xrightarrow{p} T$ diesen Zusammenhang fest.

Bei einem Causal Link handelt es sich um ein semantisches Konstrukt, nicht um eine reale Ordnung im Plan. Aber der Planungsalgorithmus sorgt dafür, daß für jeden Causal Link $S \xrightarrow{p} T$ dem Plan eine korrespondierende Ordnung $S \prec T$ hinzugefügt wird. Deshalb sind Causal Links auch immer mit einer entsprechenden Ordnung assoziiert, die die beiden Planschritte im Sinne des Causal Link anordnen.

Damit eine Vorbedingung eines Schritts bei dessen Ausführung auch tatsächlich gültig ist, reichen Causal Links alleine nicht aus. Sie beschreiben zwar die Beziehung, die zwischen zwei Schritten bezüglich der Erfüllung von Vorbedingungen gilt, treffen aber keine Aussage über andere Planschritte.

Betrachtet man Abbildung 2.1, so wird deutlich, daß hier der Planschritt V einen Effekt hat, der die Vorbedingung des Planschritts T ungültig machen kann oder sie ebenfalls garantieren könnte. Im SNLP-Sprachgebrauch spricht man in dieser Situation von einer *Interaktion*

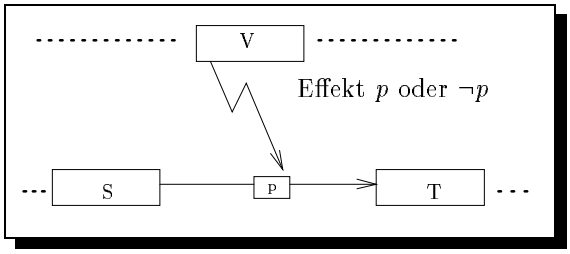


Abbildung 2.1: Beispiel für einen Threat

zwischen V und dem Causal Link $S \xrightarrow{p} T$. Dies wird mit dem zweiten zentralen Konzept im SNLP-Ansatz, den *Threats*, beschrieben.

Definition 2.3 (Threat)
Ein Threat zu einem Causal Link $S \xrightarrow{p} T$ ist ein anderer bezüglich S und T nicht angeordneter Schritt V, der p oder $\neg p$ als Effekt hat.

Definition 2.3 beschreibt eine Situation, in der ein Schritt, der parallel zu einem Causal Link liegt, einen Effekt besitzt, der die zu schützende Vorbedingung eventuell ungültig machen könnte. Gegen diese Interaktion muß der Causal Link geschützt werden. Ein Threat hält diese Tatsache der Interaktion zwischen einem Causal Link und einem parallel liegenden Schritt fest (vgl. Abbildung 2.1).

Aus Definition 2.3 wird ersichtlich, daß zwei Arten von Threats auftreten:

Positive Threats: Sie entstehen durch Planschritte, die den gleichen Effekt haben und somit auch zur Erfüllung der entsprechenden Vorbedingung herangezogen werden könnten. Nichtbeachtung dieser Threats führt nicht zu inkorrekten Plänen, verhindert aber die Systematik der Lösungssuche.

Negative Threats: Sie entstehen durch Schritte, die die Aussage des Causal Links negieren. Diese Schritte zerstören die Gültigkeit einer Aussage. Demzufolge führt eine Nichtbeachtung dieser Threats zu inkorrekten Plänen.

Dadurch, daß beim SNLP-Ansatz beide Arten von Threats aufgelöst werden, wird der Raum der möglichen Päne systematisch durchsucht. Das bedeutet, daß jeder Plan $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c >$ höchstens einmal erzeugt wird.

Auflösung von Threats

Der SNLP-Ansatz sieht vor, daß alle Threats, positive *und* negative, aufgelöst werden müssen. Dies kann zum einen durch das Einführen von Ordnungen, den im vorangegangen Abschnitt erwähnten Protections, geschehen, zum anderen können die verwendeten Variablen in den Prädikaten der Effekte so eingeschränkt werden, daß keine Bedrohung entsteht. Daher ergeben sich drei Strategien, um Threats aufzulösen:

Demotion: Hier wird der bedrohende Schritt *vor* der Quelle des Causal Links mittels einer Protection geordnet (siehe Abbildung 2.2).

Promotion: Der bedrohende Planschritt wird *nach* dem Zielschritt des Causal Links durch eine Protection angeordnet (siehe Abbildung 2.3).

Separation: Hier wird der Threat aufgelöst ohne das eine Ordnung in den Plan aufgenommen wird. Es wird eine Menge von Constraints eingeführt, so daß der Effekt des bedrohenden Schritts und der zu schützende Causal Link nicht identisch werden können. Hierbei muß jedoch darauf geachtet werden, daß die Systematik der Suche nicht zerstört wird.[3]

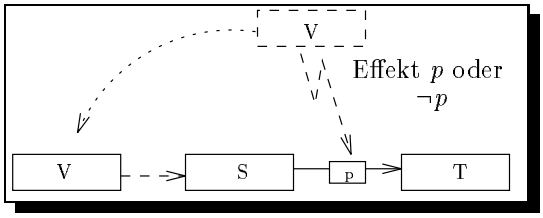


Abbildung 2.2: Auflösung von Threats durch Demotion

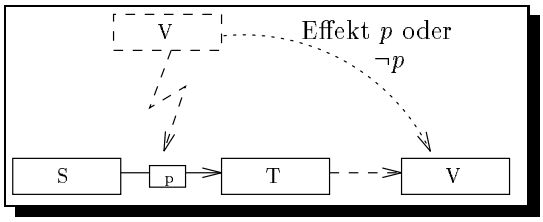


Abbildung 2.3: Auflösung von Threats durch Promotion

Durch die Anwendung dieser Strategien zur Auflösung von Interaktionen wird nun garantiert, daß jede Vorbedingung genau zur Ausführungszeit der betreffenden Aktion gültig ist.

2.1.4 Eigenschaften von SNLP-Plänen

Mit den aus den vorangegangenen Abschnitten erläuterten Konzepten lassen sich einige Eigenschaften von SNLP-Plänen beschreiben. Zunächst kann die Konsistenz eines Planes $\mathcal{P}$ definiert werden. Dabei begründet sich die Konsistenz eines Planes aus zwei Bereichen, den Ordnungen und den Variablenbindungen.

Definition 2.4 (Konsistenz von SNLP-Plänen)
Ein SNLP-Plan $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c >$ heißt konsistent, wenn gilt:

1. *Ordnungskonsistenz:*
   - *jeder Planschritt ist hinter $START$ angeordnet,*
   - *jeder Planschritt ist vor $FINISH$ angeordnet und*
   - *der Plan enthält keine Ordnungszykel.*
2. *Bindungskonsistenz:*

   Für jede Variable existiert mindestens eine Bindungsmöglichkeit, die konsistent mit allen Variablenbindungs-Constraints in $\mathcal{B}$ ist.

Das Kriterium der Ordnungskonsistenz ergibt sich zum Teil aufgrund der Definition eines Planungsproblems, das durch die Schritte $START$ und $FINISH$ in einem Plan repräsentiert wird. Da die Ordnungen der Planschritte die zeitliche Abfolge der Schritte beschreiben, ist hier die Zykelfreiheit ein wichtiges Kriterium für die Konsistenz eines Plans.

[3] Auflösung von Threats durch Separation ist nicht immer sinnvoll. [Peot und Smith, 1993] beschreibt eine Strategie die auf SNLP aufbaut und die nachweislich effizienter plant als SNLP in seiner ursprünglichen Version, aber gerade darauf beruht, Threats zu ignorieren, die durch Separation aufgelöst werden können.

Eine weitere Eigenschaft von SNLP-Plänen hält fest, wann ein Plan die Lösung eines gegebenen Problems ist. Bei SNLP spricht man dann von einem vollständigen Plan. Diese Eigenschaft läßt sich wie folgt notieren:

Definition 2.5 (Vollständigkeit von Plänen)
Ein (partiell geordneter) SNPL-Plan $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c >$ *heißt* vollständig, *wenn gilt:*

- *Für jede Vorbedingung* p *jedes* $S \in \mathcal{S}$ *existiert ein Causal Link der Form* $S_j \xrightarrow{p} S \in \mathcal{L}_c$.
- *Jeder Causal Link* $S_i \xrightarrow{p} S_j \in \mathcal{L}_c$ *ist gegen alle Threats geschützt.*

Bei der Vollständigkeitseigenschaft ist anzumerken, daß jeder vollständige Plan zwar korrekt[4], aber nicht jeder korrekte Plan im Sinne von SNLP vollständig ist. Dies ist darin begründet, daß die Causal Links gegen *alle* Threats geschützt sein müssen, insbesondere positive Threats. Dadurch werden evtl. Ordnungen zwischen Schritten eingeführt, die für die Korrektheit eines Planes nicht notwendig sind. Daher enthält ein SNLP-Plan in der Regel mehr Ordnungen, als eigentlich benötigt werden.

2.1.5 Der SNLP-Algorithmus

Nachdem in den vorangegangen Abschnitten die Grundbegriffe des SNLP-Ansatzes erläutert wurden, ist in Abbildung 2.4 der SNLP-Algorithmus dargestellt. Aufgerufen wird der Algorithmus mit dem Parameter $\mathcal{P}$, der jeweils den aktuellen Plan beschreibt. Die Komponenten des Plans ergeben sich wie in Definition 2.1 beschrieben.

Initialisiert wird der Plan vor dem ersten Aufruf mit den künstlichen Planschritten START und FINISH sowie der Ordnung $START \prec FINISH$. Damit ist dann das Planungsproblem für den Algorithmus initialisiert. Der Algorithmus versucht im weiteren sukzessiv alle Vorbedingungen der im Plan enthaltenen Schritte mittels Causal Links zu erfüllen. Sein Ziel ist ein konsistenter Plan für das gegebene Problem.

Der Algorithmus aus Abbildung 2.4 ist vollständig und korrekt, d.h. wenn es zu dem gegeben Problem ein Lösung gibt, so findet sie der Algorithmus auch. Ferner hat er die Eigenschaft, den Raum der möglichen Planungszustände systematisch zu durchsuchen, daher wird jeder Plan $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c >$ höchstens einmal erzeugt.

Der Algorithmus bricht ab, wenn ein vollständiger Plan gefunden wurde (vgl. Definition 2.5). Ist der gegenwärtige Plan noch nicht vollständig, so werden die folgenden Schritte durchgeführt. Es wird eine offene, d.h. noch nicht durch einen Causal Link garantierte Vorbedingung eines Schrittes aus dem Plan gesucht und diese dann durch einen Causal Link garantiert. Dabei wird entweder ein bereits vorhandener Planschritt verwendet oder ein neuer Planschritt in den Plan eingeführt. Die Bestandteile des Plans werden dann entsprechend aktualisiert. Danach werden die Interaktionen zwischen dem Causal Link und den restlichen Planschritten bestimmt und mittels der in Abschnitt 2.1.3 beschriebenen Strategien aufgelöst. Kann eine Interaktion nicht aufgelöst werden, oder steht kein Schritt zur Erfüllung einer Vorbedingung zur Verfügung, wird Backtracking durchgeführt.

Generell kann der Algorithmus zu einem gegeben Planungsproblem zu drei Ergebnissen führen:

[4] Mit korrekten Plänen ist hier ein konsistenter Plan gemeint, der ebenfalls das Planungsproblem löst.

SNLP($\mathcal{P}$)

1. **Termination:**
   Falls $\mathcal{P}$ vollständig ist, gebe $\mathcal{P}$ aus (Erfolg)
2. **Ziel Auswahl:**
   Wähle offene Vorbedingung p eines benutzten Schrittes $S_{used} \in \mathcal{S}$
3. **Operator Auswahl:**
   Wähle einen Schritt S_{source}, der p als Effekt hat (entweder ein neuer oder ein bereits in $\mathcal{S}$ vorhandener Schritt).
   Falls kein Schritt S_{source} gefunden wird, führe Backtracking durch.

   $\mathcal{S}' := \mathcal{S} \cup \{S_{source}\}$ $\quad$ $\mathcal{O}' := \mathcal{O} \cup \{S_{source} \prec S_{used}\}$

   $\mathcal{B}' := \mathcal{B} \cup \{newbindings\}$ $\quad$ $\mathcal{L}'_c := \mathcal{L}_c \cup \{S_{source} \xrightarrow{p} S_{used}\}$

   Backtrackingpunkte: Jede Möglichkeit einen Schritt S_{source} so zu wählen, daß ein konsistenter Plan $\mathcal{P}'$ entsteht.
4. **Bestimmen und Auflösen der Threats:**
   $\forall S_k \in \mathcal{S}$: *$S_k$ ist Threat für $S_i \xrightarrow{c} S_j \in \mathcal{L}_c$ und S_k parallel zu diesem Causal Link* do:
   Schütze den Causal Link durch Hinzufügen von Ordnungen oder Constraints zu $\mathcal{O}'$ bzw $\mathcal{B}'$.
   Gibt es keine Möglichkeit den Causal Links zu schützen, führe Backtracking durch.
   Backtrackingpunkte: Jede Möglichkeit, einen Causal Link konsistent zu schützen.
5. **Rekursiver Aufruf:**
   SNLP($\mathcal{P}'$) mit $\mathcal{P}' = < \mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}'_c >$

Abbildung 2.4: Der SNLP-Algorithmus

Erfolg: Der Algorithmus terminiert, da für alle offenen Vorbedingungen geschützte Causal Links erzeugt wurden und somit ein vollständiger Plan entstand. [McAllester und Rosenblitt, 1991] zeigt, daß wenn es eine Lösung zu dem gegebenen Problem gibt, der Algorithmus einen vollständigen Plan generiert.

Mißerfolg: Es existiert kein Backtracking-Punkt mehr, der bearbeitet werden könnte. Dies ist gleichbedeutend damit, daß es zu dem Problem keine Lösung gibt.

Nichttermination: Kann auftreten, wenn Teilzielrekursionen[5] entstehen, die vom Algorithmus aus Abbildung 2.4 nicht entdeckt werden. [McAllester und Rosenblitt, 1991] schlägt hierzu vor, ein iteratives Vertiefen und Suchtiefenbeschränkung zur Lösung des Problems zu verwenden. Eine andere Lösungsvariante wurde durch eine entsprechende Erweiterung des SNLP-Algorithmus in [Weberskirch, 1994] realisiert.

Eine wichtige Eigenschaft des Algorithmus ist es, daß die Zielauswahl, d.h. die Auswahl einer offenen Vorbedingung, keinen Backtracking-Punkt darstellt. Ein Verfahren welches die Zielauswahl als Backtracking-Punkt hat, kann das Kriterium der Systematik der Suche im Raum

[5] Eine Teilzielrekursion tritt dann auf, wenn ein Planschritt der eine Vorbedingung p garantieren soll, selbst wieder eine Vorbedingung p' herleitet, die durch Unifikation mit p identisch werden kann.

der möglichen Pläne nämlich nur in Einzelfällen erfüllen, da ein Teilziel unter Umständen mehrfach bearbeitet werden muß.

2.2 Das CAPLAN-System

Das CAPLAN-System ist ein domänenunabhängiges Aktionsplanungssystem, das auf dem SNLP-Ansatz basiert. Das Planungssystem ist dabei im Sinne eines „intelligenten Planungsassistenten“ [Weberskirch und Paulokat, 1995] konzipiert, d.h. es sucht sowohl selbständig nach einer Lösung, es kann aber auch jederzeit vom Benutzer in eine gewüschte Richtung gelenkt werden.

Eine wesentliche Komponente im Systemkern von CAPLAN bildet REDUX [Petrie, 1991]. Deshalb werden im Abschnitt 2.2.2 einige Konzepte von REDUX näher erläutert, soweit sie im Rahmen dieser Arbeit von Interesse sind. Zunächst erfolgt eine Übersicht über das CAPLAN-System. Eine detaillierte Beschreibung des CAPLAN-Systems findet sich in [Weberskirch, 1994; Weberskirch, 1995]).

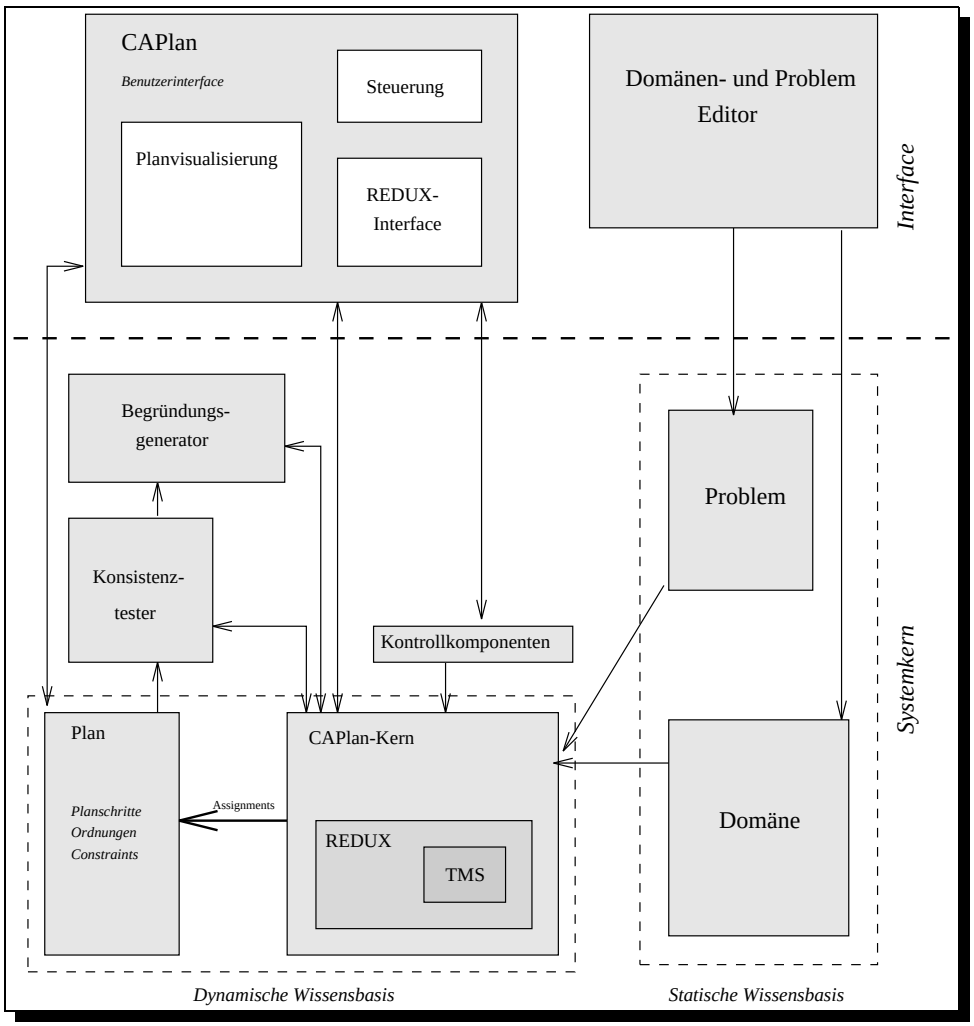


Abbildung 2.5: CAPLAN Systemübersicht

2.2.1 Systemübersicht

In Abbildung 2.5 ist das CAPLAN-System schematisch dargestellt. Es gliedert sich grob in zwei Komponenten, die Benutzerschnittstelle (Interface) und den eigentlichen Systemkern.

Das Interface realisiert die Schnittstelle zum Anwender. Hierüber werden die Benutzereingaben an das System weitergeleitet und der aktuelle Plan visualisiert (siehe Abbildung 2.6). Hier wird auch mittels des Domänen- und Problemeditors die Planungsdomäne und die zu lösenden Probleme definiert. Daneben gibt es noch eine Reihe von Schnittstellen, die es erlauben, systeminterne Entscheidungsprozesse nachzuvollziehen, wie beispielsweise das REDUX-Interface.

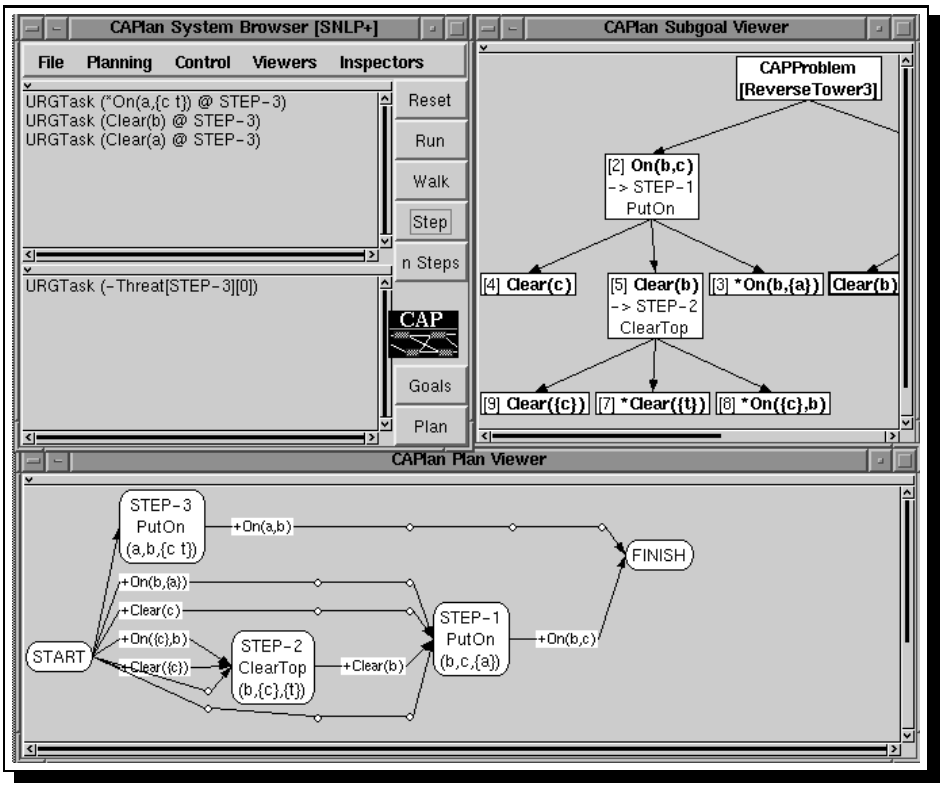


Abbildung 2.6: Die CAPLAN-Benutzerschnittstelle

Der Systemkern realisiert das eigentliche Planungssystem. Hier ist der in Abschnitt 2.1 beschriebene Algorithmus aufbauend auf REDUX lokalisiert. Die wichtigsten Komponenten des Systemkerns sind:

Wissensbasen: Im System finden sich zwei Wissensbasen, die statisches, d.h. während eines Problemlösevorgangs unverändertes, und dynamisches Wissen beinhalten. Letzteres entsteht im Laufe des eigentlichen Planungsprozesses und enthält insbesondere bei erfolgreicher Planung die Lösung des Planungsproblems.

- *Statische Wissensbasis:*
  In der statischen Wissenbasis sind die Informationen aus der Domänenbeschreibung enthalten. Sie umfassen die zulässigen Objekttypen, Prädikatssymbole sowie die möglichen Aktionen in der Planungswelt. Ein weiterer Bestandteil ist das zu

lösende Problem. Die statische Wissenbasis wird während des Problemlösevorgangs nicht verändert.

- *Dynamische Wissenbasis:*

  Die dynamische Wissenbasis besteht aus einer Erweiterung von REDUX um die SNLP-Konzepte, dem CAPLAN-Kern. Ein wichtiger Bestandteil dieser Wissenbasis ist der generierte Plan, welcher den aktuellen Zustand des Problemlösevorgangs beschreibt. Außerdem enthält die dynamische Wissenbasis eine Vielzahl von Abhängigkeiten zwischen Planungsentscheidungen, die sich im Laufe der Problemlösung ergeben.

Kontrollkomponenten: Die Kontrollkomponenten übernehmen die Steuerung des Problemlösevorgangs. Hier findet sich zum einen eine Komponente, die den Algorithmus aus Abschnitt 2.1.5 realisiert, zum anderen eine Komponente, die dem Benutzer die Kontrolle über den Problemlösevorgang ermöglicht. Hierbei kann der Benutzer an allen Stellen, an denen es mehrere Alternativen gibt, seine Entscheidung treffen. Auch das Backtracking-Verhalten des Planungssystems wird über eine spezielle Kontrollkomponente realisiert.

Konsistenztester und Begründungsgenerator: Der Konsistentester prüft, ob eine Menge von Zuweisungen, hier Ordnungen im Plan und Variablenbindungs-Constraints, mit den bis dahin gefundenen konsistent ist. Der Konsistenztester wird von der REDUX-Komponente im CAPLAN-Kern benötigt, da die Menge der möglichen Entscheidungen i.A. auch inkonsistente Entscheidungen enthält. Der Begründungsgenerator dient zur Generierung von Rückzugsentscheidungen, die von REDUX benötigt werden, um solche inkonsistente Entscheidungen zu markieren.

Die Basiskomponente des Systemkerns stellt REDUX dar. Dabei nutzen die im CAPLAN-Kern vorgenommenen Erweiterungen nicht alle Konzepte von REDUX voll aus, sondern vereinfachen manche Konzepte, die für die Realisierung des SNLP-Ansatzes nicht benötigt werden.

2.2.2 REDUX

CAPLAN basiert auf den Konzepten des REDUX-Systems [Petrie, 1991]. REDUX stellt für Planung und Konfiguration einige grundlegende und wichtige Mechanismen zur Verfügung. Das hier verwendete System geht auf eine Implementierung von [Ritzer, 1992] zurück, welche die Konzepte von REDUX auf der Basis eines *Justification-based Truth Maintanance Systems (JTMS)* (vgl. [Doyle, 1979]) realisierte.

Nachfolgend werden einige Konzepte, die für die weiteren Kapitel von Interesse sind, beschrieben. Dabei werden einige relevante Aspekte der Realisierung dieser Konzepte auf der TMS-Ebene erläutert. Hierbei wird ein grundsätzliches Verständnis von *Truth-Maintanance-Systemen*, speziell einem JTMS vorausgesetzt. Informationen über die grundsätzliche Funktion von TMS-Netzen finden sich z.B. in [Richter, 1992].

Grundbegriffe in REDUX

Die zentralen Begriffe in REDUX sind *Ziel* und *Operator*. Um ein Ziel zu erreichen, wird ein Operator angewandt. Dieser Operator kann nun neue (Unter-)Ziele herleiten und im Sinne von weiteren Auswirkungen, Zuweisungen (*Assignments*) treffen. Diese neuen Ziele und

Zuweisungen sind solange gültig wie der Operator, der sie einführte. Durch das Zerlegen von Zielen in neue Unterziele entsteht im Laufe des Problemlösevorgangs eine Teilzielhierarchie. In Abbildung 2.7 ist eine solche Teilzielhierarchie dargestellt. Hier wurde ein initiales

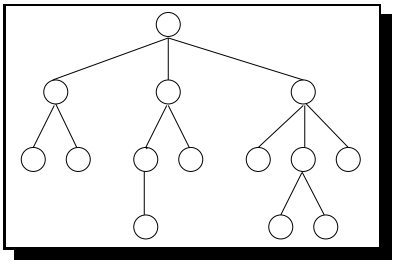


Abbildung 2.7: Beispiel eine Teilzeilhierarchie durch Operatoranwendungen

Ziel, welches die Wurzel des Baumes bildet, durch sukzessive Operatoranwendung in weitere Teilziele zerlegt. Die Zuweisungen der Operatoren, die während des Problemlösevorgangs angewandt werden, beschreiben dann die momentane Teillösung. Zuweisungen der Operatoren sind beispielsweise Planschritte und Ordnungen zwischen diesen.

Normalerweise kann ein Ziel durch mehrere Operatoren erfüllt werden. Hier spricht man von der Konfliktmenge der auf ein Ziel anwendbaren[6] Operatoren. Abbildung 2.8 verdeut-

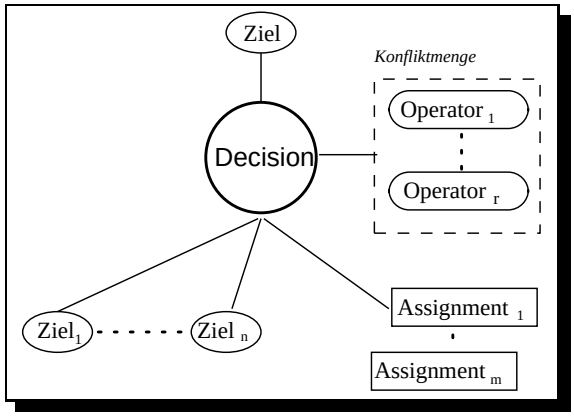


Abbildung 2.8: Operatorauswahl und Konfliktmenge

licht diesen Zusammenhang. Dort taucht noch ein weiterer wichtiger Begriff auf, der einer Entscheidung (*Decision*). Eine Decision repräsentiert die Entscheidung für einen bestimmten Operator aus der Konfliktmenge.

Alle Ziele auf die ein Operator angewandt wurde, werden als *reduziert* bezeichnet. Sind alle Ziele reduziert, so ist das Problem gelöst und die Zuweisungen (Assignments) der ausgewählten Operatoren beschreiben die Lösung des Problems. Die Suche nach einer Lösung ergibt sich als eine Folge von Operatoranwendungen in REDUX.

Im Normalfall wird es nicht gelingen, sofort die Lösung eines Problems zu finden. In REDUX werden deshalb Mechanismen bereitgestellt, um Entscheidungen die sich zu einem Zeit-

[6] Anwendbar im Sinne von REDUX heißt, daß der Operator geeignet ist, das Ziel zu erfüllen, wobei unberücksichtigt bleibt, ob seine Vorbedingungen (hier durch die hergeleiteten Unterziele repräsentiert) erfüllbar sind. In der Aktionsplanung ist ein Operator anwendbar, wenn seine Vorbedingungen erfüllt sind.

punkt als falsch erweisen, zurückzunehmen. Dies kann dadurch geschehen, daß Constraints über den momentan gültigen Entscheidungen verletzt werden oder das auf ein noch nicht reduziertes Ziel keine Operatoren mehr anwendbar sind. Das führt direkt zum Rückzug der jeweiligen Entscheidungen (Backtracking). Durch die Mechanismen in REDUX ist es möglich, jede einmal getroffene Entscheidung rückgängig zu machen, sofern die Notwendigkeit dazu erkannt wurde. Dies sind aber nicht die einzigen Fälle, in denen ein Entscheidung ungültig werden kann. REDUX stellt zwei weitere Mechanismen zur Verfügung, die zwar nicht zum Backtracking führen, aber verursachen, daß Teile eines Problems neu geplant werden müssen.

Für jede Entscheidung kann ihre Zulässigkeit (*admissibility*) definiert werden. Das bedeutet, daß eine Entscheidung auch durch andere (äußere) Umstände ungültig werden kann. Es sind in der Regel vom System nicht beeinflußbare Annahmen über die Umwelt, in der geplant wird. Ein zweiter Aspekt ist die Optimalität von Operatoren. Ein solches Kriterium setzt allerdings voraus, daß eine Ordnung auf den Operatoren der Konfliktmenge eines Ziels definiert ist, aufgrund der man dann einen optimalen Operator bestimmen könnte. Die im CAPLAN-Kern realisierten SNLP-Konzepte benötigen diese beiden Aspekte allerdings nicht, daher finden sie bei der Realisierung des SNLP-Algorithmus keine Verwendung.

REDUX unterscheidet zwei Fälle beim Rückzug von Entscheidungen. Einmal die Notwendigkeit zum Rückzug (*rejection*) und der eigentliche Rückzug (*retraction*). Beide führen zum Rückzug der getroffenen Entscheidung. Der Unterschied ist jedoch, daß eine nicht weiter bestehende Rückzugsnotwendigkeit nicht automatisch den Rückzug aufhebt. REDUX sieht hier vor, daß dies an den Problemlöser weitergeleitet wird, der dann die Möglichkeit hat, auf diese lokale Entscheidung zu reagieren und sie eventuell optimal zu lösen.

Auf den Abarbeitungsmechanismus von REDUX soll hier nicht weiter eingegangen werden. An dieser Stelle sei nur noch darauf verwiesen, das in REDUX zur Steuerung der Abarbeitung der Ziele eine Reihe von sogenannten Tasks erzeugt werden, deren Bearbeitung letztendlich die hier beschriebenen Konzepte realisiert. Die in dieser Arbeit relevanten Tasks sind der *Unreduced-Goal-Task*, der anzeigt, daß ein Ziel noch nicht reduziert wurde und der *Goal-Block-Task*, der signalisiert, daß ein Teilziel blockiert ist, d.h. für dieses Teilziel stehen im Moment keine anwendbaren Operatoren in der Konfliktmenge. Eine ausführliche Beschreibung der REDUX-Konzepte gibt [Petrie, 1991; Petrie, 1992].

Realisierung der REDUX-Konzepte auf der TMS-Ebene

Die in dem vorherigen Abschnitt vorgestellten Konzepte wurden auf der Basis eines JTMS implementiert. Die für diese Arbeit relevanten Aspekte dieser Implementierung werden nachfolgend näher erläutert.

Repräsentation von Zielen

Die Gültigkeit eines Ziels hängt in REDUX von verschiedenen Faktoren ab. Die TMS-Struktur, die diese Abhängigkeiten beschreibt, wird nun kurz vorgestellt.

In Abbildung 2.9 sind die Standardabhängigkeiten eines Ziels in REDUX dargestellt. Einen wichtigen TMS-Knoten in dieser Struktur bildet der *Valid-Goal*-Knoten. Dieser Knoten beschreibt die Gültigkeit des Ziels. Er ist deshalb so wichtig, da er auch in anderen Strukturen, wie zum Beispiel der Operatoranwendung wieder auftaucht, da die Anwendung eines Operators auch von der Gültigkeit des zu erfüllenden Ziels abhängt. Daneben hängt die Gültigkeit eines Teilziels natürlich auch von der Gültigkeit des Operators ab, der es herleitete.

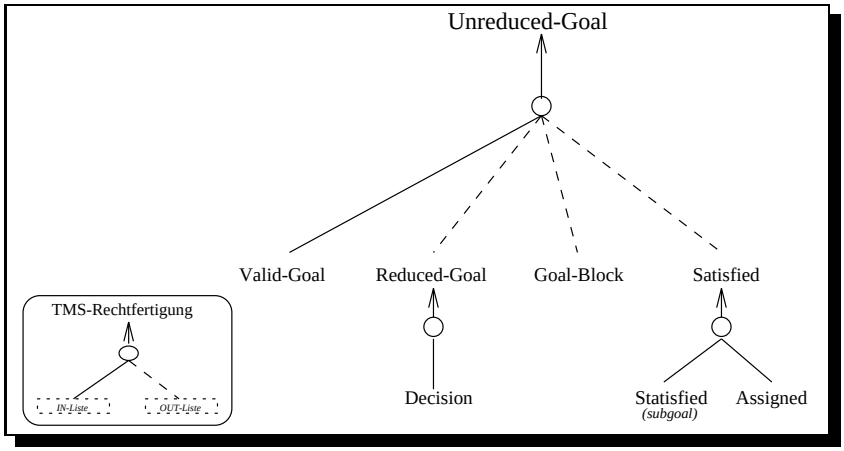


Abbildung 2.9: TMS-Struktur eines Zieles

Der *Unreduced-Goal*-Knoten zeigt an, ob ein Teilziel bereits reduziert ist oder nicht. Ein Teilziel gilt als nicht reduziert, wenn:

- das Ziel gültig (Valid-Goal) ist,
- es nicht bereits als erfüllt (Satisfied) gilt,
- es nicht bereits reduziert (Reduced) ist und
- es nicht blockiert (Goal-block) ist.

Dieser Zusammenhang ist durch die Rechtfertigung des *Unreduced-Goal*-Knoten ausgedrückt (vgl. Abbildung 2.9). Ein Wechsel des Labelings des *Unreduced-Goal*-Knotens löst im System einen *Unreduced-Goal-Task* aus, der anzeigt, daß dieses Ziel noch reduziert werden muß.

TMS-Struktur einer Operatoranwendung

Wird aus der Konfliktmenge eines Ziels ein Operator ausgewählt, so wird die in Abbildung 2.10 dargestellte Struktur für diesen Operator erzeugt. In ihr werden die verschiedenen Aspekte, die die Entscheidung beeinflussen, modelliert. Wichtig im Rahmen dieser Arbeit ist vor allem die Gültigkeit der Entscheidung, repräsentiert durch den *Decision*-Knoten.

Die Entscheidung, die zu diesem Operator führte, kann nur solange gültig sein wie die Entscheidung, die das entsprechende Ziel herleitete, auf das der Operator angewandt wurde, um es zu reduzieren. Dieser Umstand wird durch den *Valid-Goal*-Knoten in der Rechtfertigung für die *Decision* ausgedrückt.[7]

Der Punkt, an dem der übergeordnete Planer eingreift um eine Entscheidung zurückzunehmen, ist der *Rejected-Decision*-Knoten. Es ist bei der Realisierung der SNLP-Konzepte nicht notwendig zwischen dem Rückzug einer Entscheidung und der Notwendigkeit zum Rückzug zu unterscheiden, da bei dem SNLP-Algorithmus ein nicht mehr anwendbarer Operator immer zurückgenommen werden muß. Der *Rejected-Decision*-Knoten wird dann mit den entsprechenden Gründen, die zum Rückzug der Entscheidung führten, begründet. In diesem Zusammenhang sind dies Assignments aus Ordnungen oder Constraints.

[7]Diese Struktur gibt es erst seit der Version 3.0 in REDUX. In den vorherigen Versionen wurde hier als Begründung der *Valid-Parent-Decision*-Knoten verwendet (vgl. [Weberskirch, 1994]).

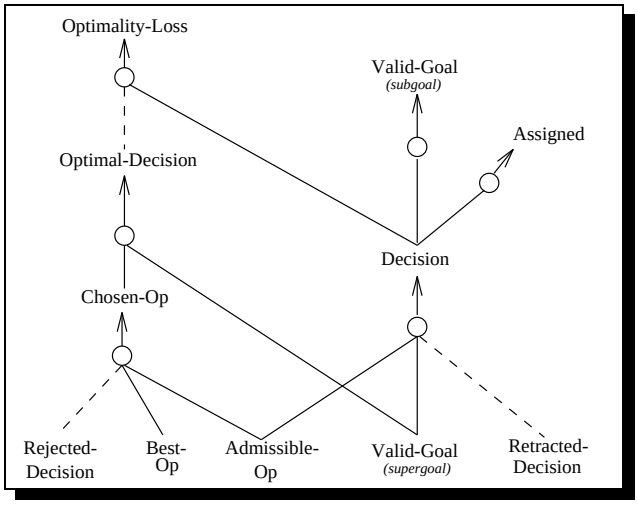


Abbildung 2.10: TMS-Struktur einer Operatoranwendung

TMS-Struktur zur Erkennung von Zielblockaden

Ein zentraler Punkt innerhalb eines Suchprozesses ist zu erkennen, wann man keine Alternativen mehr wählen kann und somit Backtracking durchführen muß. In REDUX wird dieser Umstand dadurch beschrieben, daß keine Operatoren mehr auf ein Ziel anwendbar sind. Die Struktur, die dies verwaltet, wird nun kurz beschrieben.

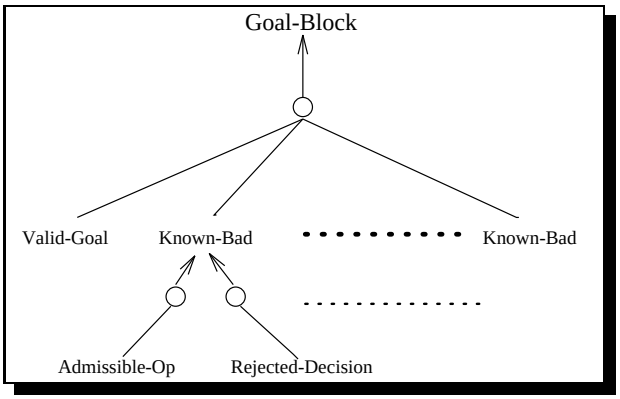


Abbildung 2.11: TMS-Struktur für Überwachung von Zielblockaden

In Abbildung 2.11 sieht man die Abhängigkeitsstruktur des *Goal-Block*-Knotens, der zur Erkennung von Zielblockaden dient. Diese Struktur wird dann erzeugt, wenn ein Ziel bearbeitet wird. Hierbei erhält jeder Operator in der Konfliktmenge einen *Known-Bad*-Knoten, der anzeigt, ob dieser Operator als ungeeignet erkannt wurde oder nicht. Sind alle Operatoren aus der Konfliktmenge zurückgewiesen, was bedeutet, daß alle *Known-Bad*-Knoten das Label IN tragen und das Ziel noch gültig ist, so wird eine *Goal-Block*-Task erzeugt. Durch die Abarbeitung dieser Task wird dann das Backtracking realisiert.

2.2.3 Domänen- und Problemspezifikation im CAPLAN-System

Da das CAPLAN-System ein domänenunabhängiges Planungssystem ist, ist es notwendig eine Domäne in einer geeigneten Bescheibungssprache zu modellieren. Das System bietet hier die Möglichkeit, in einem speziellen Editor die Domäne in einer STRIPS-ähnlichen Notation zu entwerfen. Die Domänenbeschreibung umfaßt dabei folgende Punkte:

Objekttypen: Sie definieren die Typen, die in der Planungsdomäne vorkommen und von denen in einem Planungsproblem einzelne Instanzen gebildet werden.

Prädikate: Dies sind Prädikatsymbole, die zum einen zur Beschreibung von Situationen in der Planungswelt dienen, zum anderen in Vorbedingungen und Effekten der Operatoren verwendet werden.

Operatoren: Sie stellen die in der Planungswelt durchführbaren Aktionen dar. Operatoren haben Vorbedingungen und Effekte, die durch die Prädikatsymbole ausgedrückt werden. Darüber hinaus werden bei ihnen Constraints[8] über die von ihnen verwandten Variablen spezifiziert.

Die vom Planer zu lösenden Probleme werden in einer ähnlichen Art definiert. Ein Planungsproblem wird in CAPLAN durch die folgenden Komponenten beschrieben:

Objekte: Sie repräsentieren die Gegenstände, die in dem Planungsproblem vorkommen.

Invarianten: Invarianten sind Aussagen, die während des gesamten Planungsprozesses Gültigkeit haben. Sie werden durch Prädikate ausgedrückt.

Startsituation: Die Startsituation wird durch eine Menge von Prädikaten, welche die initiale Situation des Planungsproblems darstellen, beschrieben.

Zielsituation: Die Zielsituation wird durch eine Menge von Prädikaten angegeben, die das zu lösende Problem charakterisieren.

Ordnungen: Zusätzlich kann eine Menge von Ordnungen spezifiziert werden, die beschreibt, in welcher Reihenfolge die Zielprädikate erreicht werden sollen.

Als Unterschied zu SNLP sind hier Ordnungen auf den Prädikaten der Zielsituation zu nennen, die es erlauben, den Planer Wissen über die Reihenfolge der Erfüllung von Zielen zu geben (vgl. [Weberskirch, 1995]).

2.2.4 Der CAPLAN-Kern

Im CAPLAN-Kern finden sich die in Abschnitt 2.1 beschriebenen Konzepte von SNLP als Erweiterung des REDUX-Systems. Die einzelnen Komponenten, die dies realisieren, werden nachfolgend kurz beschrieben, da sie den Ausgangspunkt für die Erweiterungen des Systems durch das in Kapitel 4 beschrieben Verfahren darstellen.

[8] Derzeit sind im CAPLAN-System die Gleichheits-Constraints Same und NotSame bzw. IsOfType und IsNotOfType als Bindungs-Constraints implementiert

Abbildungen der SNLP-Begriffswelt auf REDUX

Um den SNLP-Ansatz auf Basis von REDUX zu modellieren, ist es notwendig, folgende Konzepte auf Ziele zu übertragen:

1. Erzeugung eines Causal Links für eine offene Vorbedingung.
2. Die Auflösung von Threats.

Diese Abbildung ergibt sich aufgrund der Tatsache, daß Ziele in REDUX Backtracking-Punkte darstellen. Zur Realisierung der SNLP-Konzepte wird allerdings nicht die gesamte Funktionalität von REDUX benötigt, daher werden einige Konzepte gegenüber REDUX vereinfacht oder nicht genutzt (vgl. Abschnitt 2.2.2). Durch die Anwendung von Operatoren auf die Ziele ergibt sich, daß die Zuweisungen der Operatoren jeweils den aktuellen Planzustand beschreiben. Somit werden die SNLP-Konzepte wie Planschritte, Causal Links und Ordnungen zu Zuweisungen der REDUX-Operatoren.

Außerdem werden noch spezielle Ziele benötigt, welche die Start- und Zielsituation in das System einfügen. Generell gilt es verschiedene Typen von Zielen und Operatoren zur Realisierung von SNLP mittels REDUX zu identifizieren.

Ziele

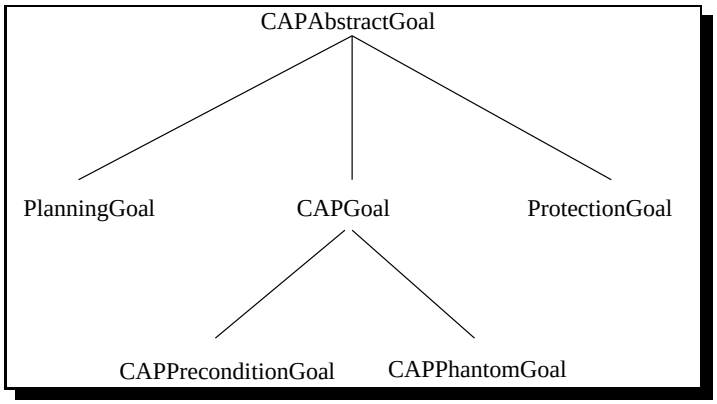


Abbildung 2.12: REDUX-Ziele zur Realisierung von SNLP

In Abbildung 2.12 sind die verschiedenen Erweiterungen von REDUX-Zielen dargestellt, die für den SNLP-Algorithmus benötigt werden. Hierbei hat das *CAPAbstractGoal* die Funktion gewisse Basisfunktionalitäten bereitzustellen. Die Aufgaben der einzelnen Ziele können wie folgt charakterisiert werden:

PlanningGoal: Dieses Ziel repräsentiert das Planungsziel und ist somit das initiale Ziel in der Hierarchie. Durch seine Bearbeitung werden die in der Problembeschreibung definierten Ziele für den Planer hergeleitet.

CAPGoal: Diese Ziele stehen für die offenen Vorbedingungen im Sinne von SNLP. Durch ihre Reduktion wird ein Causal Link eingeführt. Man unterscheidet zwei Arten solcher Ziele:

CAPPreconditionGoals: Sie repräsentieren die zu erfüllenden Vorbedingungen (Ziele) des SNLP-Ansatzes.

CAPPhantomGoals: Sie stellen eine Erweiterung der im SNLP-Ansatz beschriebenen Ziele dar. CAPPhantomGoals können nur durch einen bereits im Plan enthaltenen Schritt erfüllt werden.

ProtectionGoal: Sie repräsentieren die Threats im System. Die Anwendung eines Operators auf sie realisiert eine der Strategien aus Abschnitt 2.1.3.

Diese Ziele realisieren die notwendigen Backtracking-Punkte des SNLP-Algorithmus.

Operatoren

Zur Bearbeitung der einzelnen Ziele des vorangegangen Abschnitts sind Operatoren notwendig. Abbildung 2.13 gibt einen Überblick, über die im CAPLAN-Kern eingebundenen Operatoren. Hierbei stellt die Klasse *CAPAbstractOperator* wieder eine Basisfunktionalität für die einzelnen Operatortypen zur Verfügung.

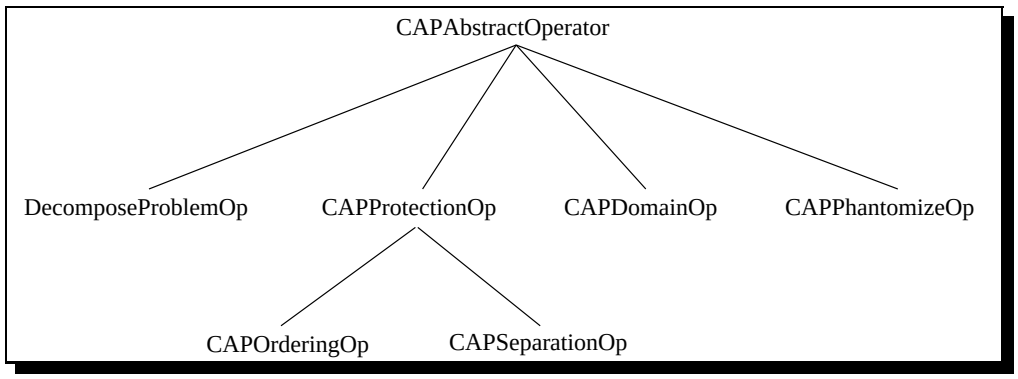


Abbildung 2.13: REDUX-Operatoren zur Realisierung von SNLP

Die Funktion der Operatoren ist im einzelnen:

DecomposeProblemOp: Dieser Operator ist ausschließlich auf Ziele der Art *Planning-Goal* anwendbar. Er leitet als neue Teilziele die zu erreichenden Ziele der Problembeschreibung her und hat als Zuweisungen die Schritte START und FINISH.

CAPDomainOp: Diese Operatoren repräsentieren die Domänenoperatoren im System. Anwendbar sind sie auf Ziele vom Typ *CAPGoal*. Sie haben als Zuweisungen den Planschritt, der sie im Plan repräsentiert sowie den Causal Link, der durch Einführung des Schrittes etabliert wird. Daneben werden noch die nötigen Variablenbindungs-Constraints des Domänenoperators zugewiesen. Als neue Unterzeile leitet der *CAPDomainOp* die entsprechenden Vorbedingungen aus der Operatordefinition her.

CAPPhantomizeOp: Mit diesem Operator wird der Umstand modelliert, daß auch ein bereits im Plan vorhandener Schritt als Quelle für einen Causal

Link in Frage kommt. Anwendbar ist dieser Operator demnach auch auf alle Ziele vom Typ *CAPGoal*. Der Unterschied zum *CAPDomainOp* ist, daß der *CAPPhantomizeOp* keine neuen Unterziele herleitet.

CAPProtectionOp: Der *CAPProtectionOp* ist unter anderem für die Auflösung der entstandenen Threats während des Planungsprozeß zuständig. Sie sind nur auf Ziele der Art *CAPProtectionGoal* anwendbar. Durch ihre Aufgabenstellung ergibt sich, daß diese Operatoren keine neuen Ziele herleiten. Im Detail lassen sich folgende Protection-Operatoren im System identifizieren:

CAPOrderingOp Er dient dazu, die im Abschnitt 2.1.3 beschriebenen Ordnungen, die durch Promotion bzw. Demotion entstehen, zu realisieren. Die Zuweisung, die dieser Operator macht, ist die entsprechende Ordnung.

CAPSeparationOp Er entspricht der in Abschnitt 2.1.3 beschriebene Strategie der Separation. Die Zuweisungen dieses Operators sind die Variablenbindungs-Constraints, die sicherstellen, daß der bedrohende Schritt nicht den Effekt des Causal Links bewirkt.

Diese Abbildung der SNLP-Begriffe in die REDUX-Definitionen alleine verwirklicht den SNLP-Ansatz noch nicht. Hierzu ist es notwendig, die Auswahl der einzelnen Ziele und Operatoren zu steuern, da der SNLP-Algorithmus aus Abschnitt 2.1.5 zum Beispiel vorschreibt, daß alle Threats aufgelöst werden müssen, bevor ein neuer Causal Link etabliert werden kann.

Eine Möglichkeit, den Algorithmus zu realisieren besteht darin, eine partielle Ordnung auf den hier beschiebenen Zielen und Operatoren zu definieren.[9] Diese Ordnung ergibt sich dann als folgende Abarbeitungsordnung der Ziele: ProtectionGoals vor CAPGoals. Das hat zur Folge, daß die auftretenden Threats immer zuerst vollständig bearbeitet werden.

Die beschriebene Ordnung der Ziele entspricht zwar dem SNLP-Algorithmus, dennoch wird im CAPLAN-System die Zielauswahl von der sogenannten Kontrollkomponente getroffen. Dies hat den Vorteil, daß man beliebige Strategien zur Lösung eines Planungsproblems anwenden kann. Die im System realisierte Kontrollkomponente SNLP z.B. realisiert den in Abschnitt 2.1.5 beschriebenen Algorithmus. Daneben gibt es noch eine Reihe weiterer Strategien, die in Einzelfällen wesentlich effizienter planen als SNLP (vgl. [Weberskirch, 1995]). In den Kontrollkomponenten existieren noch eine Reihe von Ordnungen auf den Operatoren, die aber mehr als Heuristiken für eine bessere Planungseffizienz zu verstehen sind, da sie mit dem eigentlichen SNLP-Algorithmus nichts zu tun haben.

[9] REDUX sieht diese Art der Steuerung der Lösungsuche schon vor.

Kapitel 3

Hierarchisches Planen

Ein Teilgebiet im Bereich Planen ist die hierarchische Planung. Sie hat ihren Ursprung in Systemen wie ABSTRIPS [Sacerdoti, 1973]. Der folgende Abschnitt soll einen Überblick über den *Hierarchical Task Network*[1]-Ansatz geben, wie er von [Erol *et al.*, 1994; Erol *et al.*, 1995] beschrieben wird und die Unterschiede zu nicht-hierarchischen Ansätzen aufzeigen. Im Anschluß daran werden einige konkretere Planungssysteme, die auf dem HTN-Ansatz aufbauen, beschrieben. Abschließend werden Alternativen zu dem HTN-Ansatz vorgestellt.

3.1 Der HTN-Ansatz

Eine Formalisierung des Begriffs „hierarchisches Planen“ wurde in [Erol *et al.*, 1994; Erol *et al.*, 1995] vorgenommen. Hierbei wurden für die verschiedenen Aspekte des Ansatzes theoretische Grundlagen geschaffen, die es erlauben, Eigenschaften des HTN-Ansatzes zu beweisen.

Zunächst wird der Begriff „hierarchisches Planen“ beschrieben und die wesentlichen Unterschiede zu nicht-hierarchischen Ansätzen aufgezeigt. Danach wird dann eine Einführung in den von Erol entwickelten Formalismus gegeben sowie ein allgemeiner HTN-Algoritmus vorgestellt.

3.1.1 Eingrenzung des Begriffs „hierarchisches Planen“

Ein wichtiger Aspekt, der von früheren Ansätzen nur unzureichend verwirklicht wurde, ist der Aspekt der Aktionsdekomposition. Hierunter versteht man die Zerlegung einer komplexen Handlung in eine Reihe einfacherer Aktionen, die dasselbe bewirken. Es gibt eine Reihe von Systemen die mit Dekompositionen arbeiten, wie zum Beispiel SIPE [Wilkins, 1988], NONLIN [Tate, 1977] oder sein Nachfolger O-Plan [Currie und Tate, 1991]. Sie alle arbeiten nach demselben Prinzip, der Zerlegung einer komplexen Aufgabe in einfachere Teilaufgaben. Ein Vorteil von komplexen Aktionen ist, daß man viele Dinge nicht oder nur schwer durch eine einzelne primitive Aktion ausdrücken kann, wie beispielsweise eine Rundreise durch eine Stadt, bei der Start- und Zielsituation teilweise gleich sind.

Unter dem Aspekt der komplexen Aktionen sind daher beim hierarchischen Planen zwei Fragenstellungen von besonderem Interesse:

[1] *Hierachical Task Network* wird im folgenden durch HTN abgekürzt.

1. *Wie kann ich ein gegebenes Ziel erreichen?*
2. *Wie führe ich die Aktion konkrekt aus, die ein Ziel erreicht?*

Um die beiden Aspekte etwas zu motivieren sei hier folgendes Beispiel gegeben.

Beispiel: Es soll ein Plan für eine Reise des 1 FCK von Kaiserlautern nach Meppen geplant werden. Um das Ziel Meppen zu erreichen, wird eine Aktion *Reise-nach(Ort)* gewählt. Die Aktion *Reise-nach* kann nun auf verschiedene Arten durchgeführt werden. Beispielsweise mit dem FCK-Bus, mit dem Zug oder mit dem Flugzeug. Jede dieser Möglichkeiten zieht eine Reihe von neuen Aufgaben nach sich. Entscheidet man sich dafür mit dem Flugzeug nach Meppen zu fliegen, so muß die Mannschaft erst zu einem Flughafen, dann müssen Flugtickets gekauft werden, man muß vom Zielflughafen nach Meppen in das Stadion selbst kommen. Diese Aktionen selbst können nun wieder zerlegt werden.

Die Fragestellung 1 wird in dem Beispiel dadurch ausgedrückt, daß die Aktion *Reise-nach(x)* gewählt wird, um einen Ort zu erreichen. Dabei ist hier noch nicht von Interesse wie dies konkret zu geschehen hat. Dieser Aspekt entspricht der zweiten Fragestellung. Wie im Beispiel angedeutet, kann eine Aktion *Reise-nach(x)* auf vielfältige Arten realisiert werden.

Ein weiterer wichtiger Aspekt des HTN-Ansatzes ist es, daß zu erreichende Ziele auch durch Situationen aus den bestehenden Aktionen erfüllt werden können, was der Phantomisierung eines Ziels entspricht. Daneben ist es auch möglich, Situationen, die aus der Zerlegung einer komplexen Aktion entstehen, für die Erfüllung einer anderen Aktion zu verwenden, also die Zerlegungen miteinander zu verzahnen. Um diese Überlegung zu illustrieren nehmen wir an, Meppen hätte einen eigenen Flughafen[2], dann könnten Teile der Aufgabe, das Meppener Stadion vom Zielflughafen zu erreichen, durch das Fliegen zum Zielflughafen schon erreicht werden. In diesem Fall würden dann Teile der Zerlegung einer komplexen Aktion *Erreiche-Stadion(Ort)* durch die Zerlegung der Aktion *Reise-nach(Ort)* zur Verfügung gestellt.

Man kann hierarchische Ansätze so charakterisieren, daß hier (Teil-)Pläne eingesetzt werden, um ein Ziel zu erreichen.

3.1.2 Unterschiede zu nicht-hierarchischen Ansätzen

Betrachtet man nicht-hierarchische Ansätze, wie zum Beispiel STRIPS oder SNLP, so läßt sich festhalten, daß sie teilweise ähnlich, wie hierarchische Ansätze, arbeiten,.

Bei STRIPS-artigen Ansätzen wird der Zustand der Welt durch atomare Formeln[3] dargestellt. Aktionen ihrerseits werden durch Operatoren, die den Zustand der Welt direkt verändern, definiert. Um nun ein Problem, das durch eine Reihe von solchen Formel charaketrisiert wird, zu lösen, versucht man eine (partiell) geordnete Sequenz von Aktionen zu finden, die diese Formeln gültig machen. Diese Aktionen sind im Sinne von HTN primitiv, da sich hier nicht die Frage stellt, wie man diese Aktion nun konkret realisieren muß. Der Operator repräsentiert die Aktion durch seine Vorbedingungen und Effekte. Bei der Lösungssuche handelt es sich um einen langen Suchprozeß, der versucht eine Sequenz von Operatoren zu finden, die die Startsituation in die Zielsituation überführt. Dabei gibt die Domänendefinition keinerlei Hinweise darauf, welche Kombination von Operatoren brauchbar sein könnte.

[2] Diese Annahme ist natürlich sehr theoretischer Natur für eine Stadt, die nur eine Zweitligamanschaft hat, wobei der Zusammenhang zwischen Zweitligafußball und Flughäfen Gegenstand weiterer Untersuchungen sein sollte.

[3] In der Aktionsplanung sind dies Prädikate.

Im Gegensatz hierzu ist bei hierarchischen Ansätzen, speziell beim HTN-Ansatz, neben der Suche nach einer geeigneten Aktion, die ein Ziel erfüllt, noch der Aspekt der Realisierung dieser Aktionen von Interesse. Es gibt dabei wieder Aktionen, die durch Vorbedingungen und Effekte beschrieben werden, allerdings werden diese Aktionen nicht zwangsläufig als primitive, d.h. die Welt direkt verändernde Aktionen, angesehen. Vielmehr ist es auch möglich, und bei hierarchischen Ansätzen gewünscht, daß eine nicht primitive Aktion konkretisiert wird, indem sie gemäß einer ebenfalls zum Domänenmodell gehörenden Dekompositionsregel in ein geordnetes Netz von weniger abstrakten Aktionen überführt wird. In diesem Punkt liegt der Hauptunterschied zwischen den nicht-hierarchischen und den hierarchischen Ansätzen.

Nimmt man die Fragestellungen aus Abschnitt 3.1.1 über die vom Planer durchzuführenden Schritte, so kann man sagen, daß Planer wie SNLP, die eine STRIPS-artige Operatorrepräsentation verwenden, nur nach dem Punkt 1 arbeitet. Dies bedeutet, daß sie ausschließlich versuchen eine geeignete Sequenz von primitiven Aktionen zu finden, die gewisse Ziele erfüllen. Ein Vertreter, der ausschließlich nach Punkt 2 vorgeht, ist NONLIN. Hier wird eine komplexe Aufgabe, das Planungsziel, jeweils durch fortschreitende Zerlegung in eine Reihe primitiver Aktionen zerlegt, die das initiale Problem lösen.

Vorteile der Aktionsdekomposition

Durch den hierarchischen Ansatz mittels Aktionsdekomposition ergeben sich eine Reihe von Vorteilen gegenüber nicht-hierarchischen Ansätzen. Die wichtigsten davon lassen sich mit den folgenden Punkten skizzieren:

1. Vereinfachung der Wissenmodellierung in Domänen.
2. Mögliche Beschränkung des Suchraums.
3. Planung und Ausführung können verzahnt werden.

Durch die komplexen Aktionen besteht die Möglichkeit umfangreiche Sachverhalte auszudrücken, die dann im Laufe des Planungsprozesses durch eine Folge von einfacheren Aktionen konkretisiert werden. Hierduch können bei der Modellierung von Domänen abstrakte Aktionen definiert werden, wie zum Beispiel *Reise-nach(Ort)* aus dem Beispiel aus Abschnitt 3.1.1. [Drummond, 1993] sieht hier einen großen Vorteil solcher Ansätze: Sie ermöglichen eine natürlichere Domänenmodellierung, da in komplexen Aktionen mehr Wissen über die Planungswelt kodiert werden kann.

Weil in komplexen Aktionen mehr Wissen kodiert ist, ergibt sich auch der zweite Aspekt. Die Reduzierung des Suchaufwandes bei der Lösungsfindung. Dies kann erreicht werden, indem abstrakte Aktionen jeweils nur die relevanten Aspekte der Planungswelt berücksichtigen. Daher ist möglich durch die Definition von solchen abstrakten Aktionen, erst die wichtigen Aspekte eines Problems anzugehen.[4] Hat der Planer erst eine passende komplexe Aktion gewählt, so kann er durch deren Verfeinerung direkt einige andere Ziele, die noch zu erreichen sind, erfüllen. Durch die Auswahl der passenden komplexen Aktion werden dann die Planungszustände, die nicht zu einer Lösung führen, von Anfang an ausgeschlossen, d.h. der Suchraum wird eingeschränkt. Ein Beispiel hierfür wäre eine Aktion *Baue-Haus*, welche die Schritte, die notwendig sind, um ein Haus zu bauen, schon vorgibt. Dies hängt aber sehr

[4] Dies ist eine ähnliche Strategie, die auch ein anderer Ansatz, ABTWEAK (vgl. Abschnitt 3.3.2), verfolgt, nämlich das Verbergen von Vorbedingungen (*precondition hiding*).

stark von der Domänenmodellierung ab, da sich der Suchraum natürlich entsprechend der Anzahl der definierten Aktionen auch wieder erweitert (vgl. Kapitel 6).

Der dritte Punkt beschreibt die Möglichkeit, daß der Planer Teile eines Planes komplett planen kann, bevor er sich den restlichen Problemstellungen zuwendet. Anschaulich bedeutet es, daß er erst eine komplexe Aktion, deren Ausführung sofort geschehen muß, so auflöst, daß sie nur noch aus primitiven Aktionen besteht, deren Abarbeitung dann sofort beginnen kann.[5] Die neuen Aktionen können dann wieder herangezogen werden, um andere Teile der Problemstellung zu lösen.

Die meisten hierarchischen Ansätze erfüllen die ersten beiden Punkte. Aber der dritte Punkt wird nur von Ansätzen erfüllt, die auf dem HTN-Ansatz basieren. Ein wichtiger Aspekt bei der Aktionsdekomposition ist noch, daß die einzelnen Zerlegungen von komplexen Aktionen ineinander verzahnt werden können, d.h. die konkreten Realisierungen von solchen Aktionen durchaus auch ineinandergreifend geschehen kann und soll.

3.1.3 Begriffe des HTN-Ansatz

Um die verschiedenen Aspekte der vorangegangen Abschnitte zu formalisieren, werden die Ziele, wie sie in STRIPS-artigen Planern verwendet werden, durch sogenannte *Tasks* ersetzt. Man unterscheidet drei Arten von Tasks:

Goal-Tasks: Sie stellen Ziele im Sinne von STRIPS, also Eigenschaften, die nach Möglichkeit gelten sollten, dar.

Beispiel: Beim Hausbau - das Vorhandensein einer Leiter.

Primitive-Tasks: Sie stellen Aktionen dar, die direkt in der Planungswelt ausgeführt werden können.

Beispiel: Einen Schalter betätigen, eine Leiter holen oder ein Klötzchen aufnehmen.

Compound-Tasks: Durch sie werden beabsichtigte Veränderungen ausgedrückt. Hierbei handelt es sich nicht um direkt ausführbare Aktionen, sondern um die Darstellung einer komplexen Handlung. Daher repräsentieren *Compound-Tasks* immer eine Ansammlung von *Goal-Tasks* und *Primitive-Tasks*.

Beispiel: Eine Leiter bauen. Dies würde bedeuten, daß eine Zeichnung angefertig wird, daß man Material besorgen und die Leiter montieren muß.

Compound-Tasks werden auch als *nicht-primitive* Tasks bezeichnet. Hierbei ist der Unterschied zwischen einem *Goal-Task* und einem *Compound-Task* wichtig. Eine *Goal-Task* beschreibt nur einen zu erreichenden Zustand, während ein *Compound-Task* stellvertretend für eine Ansammlung von Zuständen steht, die den gewünschten Endzustand charakterisieren. Wird bei einer Planung ein Ziel, wie beispielsweise *Eine Leiter benutzen* vorausgesetzt, so

[5] Dies ist ein Aspekt, der von vielen als Vorteil von hierarchischen Ansätzen, wie HTN, gesehen wird. Diese Eigenschaft ist aber nur bei Echtzeitsystemen interessant, bei denen auf gewisse Situationen sofort reagiert werden muß. Die Anzahl der Planer, die in Echtzeit einen Plan zu einem dynamischen Prozeß generieren, ist aber stark beschränkt.

kann es mittels eines *Goal-Task* beschrieben werden, jedoch können notwendige Nebenbedingungen wie Anstellwinkel oder Plazierung derselben, besser durch einen *Compound-Task* dargestellt werden.

Im Zusammenhang mit den *Compound-Tasks* wird nun das nächste, zentrale Konzept des HTN-Ansatzes deutlich, die sogenannten *Task-Netze*. Mit ihnen wird die kausale Struktur der einzelnen Tasks aus denen der *Compound-Task* besteht, festgehalten. Hierzu bedient man sich den schon in Abschnitt 2.1 vorgestellten Konzepte, wie Ordnungen und Causal Links.

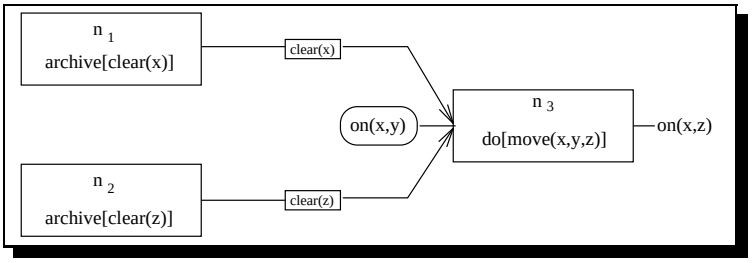


Abbildung 3.1: Beispiel eines Task-Netzes

In Abbildung 3.1 ist ein solches Task-Netz am Beispiel einer Aktion in der Blocksworld dargestellt. Das dort dargestellte Task-Netz enthält drei Tasks: Erst müssen die Blöcke x und z freigemacht werden, um dann x auf z zu stellen. Man sieht hier die unterschiedlichen Tasks: $achieve[\ldots]$ stellt einen Goal-Task dar und , $do[\ldots]$ repräsentiert einen primitiven Task. Die Symbole für Ordnungen und Causal Links sind analog denen in Abschnitt 2.1. Ein solches Task-Netzwerk könnte nun einen Compound-Task $perfom[makeon(x,z)]$ repräsentieren.

Neben den Goal- und Primitiven Tasks sieht man in Abbbidung 3.1 noch die Bedingung $on(x,y)$. Sie stellt eine Filterbedingung[6] dar, d.h. eine Einschränkung, die verlangt, daß x auf y steht, was sich aus dem Planungsprozeß ergeben sollte.

Die Bestandteile eines Task-Netzwerks sind in folgender Definition zusammengefaßt:

Definition 3.1 (Task-Netzwerk)
Ein Task-Netzwerk $t =< T, O, B, C >$ *besteht aus den Komponenten*

- T *einer endlichen Anzahl von Tupeln der Form* $(n_i : \alpha_i)$*, wobei* α_i *für einen Task steht und* n_i *ein eindeutiges Label für diesen Task darstellt.*
- O *einer Menge von Ordnungen und Causal Links zwischen den Tasks.*
- B *einer Menge von Variablenbindungs-Constaints über die freien Variablen der Tasks.*
- C *einer Menge von Filterbedingungen über zusätzliche Bedingungen in* t*.*

Man spricht von primitiven Task-Netzen, wenn nur primitive Tasks in einem solchen Netzwerk enthalten sind und ansonsten von einem komplexen Task-Netz. Die Komponenten eines Task-Netzwerks erinnern dabei an die Bestandteile eines Plans (vgl. Definition 2.1). Auch hier kommen Tasks, in Form von durchzuführenden Schritten, Ordnungen, Causal Links und Variablenbindungs-Constraints vor.

[6] Hierbei handelt es sich um Bedingungen, die während des Planungsprozesses erfüllt werden müssen. Dies kann beispielsweise dadurch geschehen, daß eine Bedingung durch vorhandene Schritte aus dem Plan erfüllt werden müssen. Bedingungen dieser Art unterliegen gewissen Einschränkungen. [Tate *et al.*, 1994] gibt hierzu einen Überblick.

Beispiel: Das Task-Netzwerk aus Abbildung 3.1 läßt sich dann mit der Definition 3.1 folgendermaßen notieren:

$$t = <\{(n_1 : achieve[clear(x)]), (n_2 : archieve[clear(z)]), (n_3 : do[move(x,y,z)])\},$$
$$\{(n_1 \prec n_3), (n_2 \prec n_3), n_1 \overset{clear(x)}{\longrightarrow} n_3, n_2 \overset{clear(z)}{\longrightarrow} n_3\},$$
$$\{x \neq y, x \neq z, y \neq z\},$$
$$\{(on(x,y), n_3)\} >$$

Ein weiteres Konzept innerhalb von HTN bilden die sogenannten *Planungsoperatoren*. Die Definition eines Planungsoperators weicht dabei wesentlich von der eines STRIPS-Operators ab. Ein Planungsoperator wird bei HTN durch einen primitiven Task mit einer Reihe von literalen Effekten beschrieben. Die Vorbedingungen, hier liegt der Unterschied zu STIPS-Operatoren, werden durch eine Reihe von Tasks, die komplex oder primitiv sind, repräsentiert. Planungsoperatoren dienen dazu die primitiven Tasks zu erfüllen.

Um einen nicht-primitiven Task zu erfüllen, werden sogenannte *Methoden* definiert. Die Methode gibt an, durch welches Task-Netz ein komplexer Task erreicht werden kann. Damit hält die Methode eine Verfeinerungsmöglichkeit für einen nicht-primitiven Task fest. Eine Methode wird durch $(\alpha : t)$ notiert, wobei α einen komplexen Task und t ein Task-Netz repräsentiert. Die Anwendung einer Methode hat zur Folge, daß der Task α durch ein Netzwerk von neuen Tasks aus t ersetzt wird. Die Semantik dieses Vorgehens ist eindeutig: um den Task α zu erreichen, müssen die Tasks aus dem Netzwerk t erreicht werden. Man spricht in diesem Zusammenhang davon, daß α durch t expandiert wird.[7]

HTN-Probleme und Domänenbeschreibungen

Auch der HTN-Ansatz ist i.A. domänenunabhängig und benötigt daher eine Domänenbeschreibung, die in der Planungswelt die gültigen Aktionen und Zustände beschreibt. Diese Beschreibung einer Domäne ergibt sich als ein Paar $\mathcal{D} = (Op, Me)$, wobei Op eine Menge von Planungsoperatoren, für jeden primitiven Task einen und Me eine Menge von Methoden, die jedem komplexen Task mit einem Task-Netzwerk verbinden, beschreibt. Dabei können mehrere Methoden für einen komplexen Task definiert sein.

Die im HTN-Ansatz zu lösenden Probleme haben die folgenden Struktur.

Definition 3.2 (HTN-Problem)
Ein Planungsproblem $\mathcal{P} = < d, I, \mathcal{D} >$ ist durch folgende Komponenten beschrieben:

- d *ein Task Netzwerk, das zu lösen ist.*
- I *eine Beschreibung der initialen Situation.*
- $\mathcal{D}$ *eine Domänenbeschreibung.*

Aus der Definition 3.2 ist ersichtlich, daß auch die Problembeschreibung durch Task-Netzwerken erfolgt, dabei also schon Teilpläne definiert werden, die zu lösen sind, d.h. das Problem wird durch einen initialen Plan dargestellt. Dadurch ergibt sich natürlich auch, daß es hiermit erleichtert wird, komplexe Problemstellungen auszudrücken, da schon in

[7] Formal gibt es beim HTN-Ansatz, wie er von [Erol *et al.*, 1994] beschreiben wird, noch eine spezielle Methode, die es erlaubt einen Task, der aus dem Planungszustand heraus bereits erfüllt ist, als expandiert zu kennzeichnen. Diese Methode hat sonst keine Auswirkungen auf den Plan, und kann auf jeden Task angewandt werden. Man nennt diesen Prozeß auch Phantomisierung.

der Problembeschreibung Ordnungsbeziehungen zwischen Einzelaufgaben spezifiziert werden können.[8]

3.1.4 Der HTN-Algorithmus

In den vorangegangen Abschnitten wurden die grundsätzlichen Konzepte des HTN-Ansatzes beschrieben. Der HTN-Algorithmus aus Abbildung 3.2 zeigt die grobe Vorgehensweise mit der ein gegebenes HTN-Problem gelöst wird.

HTN(P)

1. Wenn P nur noch primitive Tasks enthält, löse Konflikte in P auf und gebe P als Ergebnis aus.
   Sind die Konflikte in P nicht auflösbar, Abbruch und Ausgabe „keine Lösung".
2. Wähle nicht primitve Task t aus P
3. Wähle Expansion e für t
4. P' entsteht durch Ersetzen von t durch die gefunde Expansion e in P.
5. Benutze die *critics* um Interaktionen zwischen den Tasks in P' zu bestimmen und löse sie auf.
6. Rekursiver Aufruf: HTN(P')

Abbildung 3.2: Der HTN-Algorithmus im Überblick

Aufgerufen wird er mit einem Planungsproblem P. Der Algorithmus versucht nun einen Plan, bestehend aus einer Sequenz von primitiven Tasks zu erzeugen, die dann das Problem lösen. Dies geschieht, in dem ein nicht-primitiver Task sowie eine Expansion für ihn gewählt wird. Die Anwendung der Expansion liefert einen verfeinerten Plan, der rekursiv so lange expandiert wird, bis entweder nur noch primitive Tasks im Plan vorhanden sind und damit, bis auf bestehende Konflikte, die Lösung gefunden wurde, oder ein unauflösbarer Konflikt erreicht ist, der dann zum Backtracking führt.

Interaktionen zwischen Tasks

Durch die Expansion eines komplexen Tasks entsteht eine Reihe von neuen Tasks. Diese Tasks können bestehende Tasks, welche die Zustände in der Planungswelt zu einem Zeitpunkt erfüllen, ungültig machen.

Um solche Interaktionen zwischen den Tasks zu bestimmen, wird beim HTN-Ansatz eine sogenannte *critics*-Funktion verwendet, die dazu dient, Konflikte zwischen Tasks zu bestimmen und aufzulösen (siehe Abbildung 3.2 Schritt 5). Zur Auflösung der Interaktionen werden Strategien verwandt, die analog zu den Strategien, die schon in Abschnitt 2.1.3 beschrieben wurden, sind.

[8] Im CAPLAN-System ist etwas ähnliches implementiert. Hier kann auf den Zielen einer Problembeschreibung eine Ordnung definiert werden in der die Ziele zu erreichen sind (vgl. [Weberskirch, 1995]).

3.2 Beispiele einiger Realisierungen des HTN-Ansatzes

Dieser Abschnitt beschreibt an Beispielen die Realisierung des HTN-Ansatzes in konkreten Planungssystemen. Vorgestellt werden im einzelnen NONLIN [Tate, 1977], UMCP [Erol *et al.*, 1994] und DPOCL [Young *et al.*, 1994]. Die Ansätze verdeutlichen die unterschiedliche Anwendung der Konzepte, d.h. die typischen Arbeitsweisen und Definitionen, des HTN-Ansatzes. NONLIN wurde auch deshalb in diese Auswahl aufgenommen, um zu verdeutlichen, daß die Konzepte des HTN-Ansatzes schon sehr früh entwickelt und angewandt wurden.

3.2.1 NONLIN

Einer der ersten hierarchischen Planer war NONLIN [Tate, 1977]. NONLIN baut dabei auf NOAH auf und war der erste Planer, der eine Backtracking-Strategie bei der Lösungssuche verwendete. Die Problemlösung wurde bei NONLIN dabei so vorgenommen, daß ein initiales Ziel, das Planungsproblem, durch fortschreitende Anwendung von Domänenoperatoren in immer primitivere Teilprobleme zerlegt wurde. Diese Teilprobleme werden in NONLIN auch als zu erfüllende Tasks bezeichnet, wobei eine ähnliche Unterscheidung der Tasks erfolgt wie bereits in Abschnitt 3.1.3 beschrieben. Die bei der Erfüllung der Tasks entstehenden Interaktionen zwischen den einzelnen Teilzielen und den Operatoren werden analog den in Abschnitt 2.1.3 beschriebenen Verfahren, also durch Einführung von Variablenbindungs-Constraints oder Ordnungen, aufgelöst. Kann ein Konflikt nicht aufgelöst werden, so führt dies zum Backtracking, ebenso wenn eine Task nicht erfüllt werden kann. Insgesamt stellt NONLIN den ersten hierarchisch arbeitenden Planer dar, der eine Backtracking-Strategie zur Lösungsfindung einsetzte, und bildet damit die Grundlage für zahlreiche weitere Ansätze, wie beispielsweise den HTN-Ansatz.

3.2.2 UMCP

Der *Universal Method-Composition Planner* (UMCP) [Erol *et al.*, 1994] ist eine direkte Umsetzung des HTN-Algorithmus, wie er in Abschnitt 3.1.4 beschrieben wurde. Es handelt sich hierbei um einen domänenunabhängigen Planer, der vollständig und korrekt ist. Daneben hat er auch die Eigenschaft der systematischen Suche im Raum der möglichen Lösungen.

Zum Verständnis des Algorithmus sollen einige Notationen erläutert werden:

- $comp(P)$ entscheidet, ob das gefundene Task-Netz d schon ein Lösung ist. Hierbei ist ein Kriterium einer Lösung ein total geordnetes Task-Netz.
- $reduce(d, n, m)$ expandiert die gewählte Task mit dem Label n in dem aktuellen Task-Netzwerk d mit der Methode m.
- τ ist die *Critics*-Funktion[9], die anhand einer Reihe von Kritieren das Task-Netzwerk d im Zusammenhang mit der initialen Situation I und der Planungsdomäne D überprüft. Dabei werden die durch die Lösung der Konflikte neu entstehenden Task-Netz d' in einer Liste Γ gesichert.

[9] An τ werden einige spezielle Anforderungen gestellt, so u.a. daß sie vollständig ist, d.h. τ schließt keine möglichen Lösungen aus.

UMCP(P)

1. Wenn d primitiv dann
   wenn $comp(P) \neq \emptyset$ gebe Element von $comp(P)$ aus. Sonst: Abbruch Fehler.
2. Wähle nicht primitive Task $t := (n : \alpha)$ aus d
3. Wähle Methode m für α
4. $d := reduce(d, n, m)$.
5. $\Gamma := \tau(d, I, D)$.
6. Wähle zufällig ein Element $d \in \Gamma$.
7. Rekursiver Aufruf: UMCP(P)

Abbildung 3.3: Der UMCP-Algorithmus im Überblick

Der Algorithmus arbeitet dabei nach dem in Abschnitt 3.1.4 beschriebenen Verfahren. Aufgerufen wird der Algorithmus mit einem Planungsproblem $P =< d, I, D >$, wie in Definition 3.2 beschrieben. Zur Lösung eines solchen Problems werden drei Strategien verwandt: Reduktion einer komplexen Task, Einführung von Ordnungen oder Variablenbindungs-Constraints und domänenspezifische Kriterien[10]. Die Backtracking-Punkte ergeben sich als die Auswahl eines neuen Task-Netzwerks aus Γ. Eine Lösung ist dann gefunden, wenn das Task-Netzwerk d nur noch aus primitiven, total geordneten Tasks besteht.

3.2.3 DPOCL

Der von [Young *et al.*, 1994] vorgestellte *Decomposition and Partial Order Causal Link Planner* (DPOCL)[11] verbindet den SNLP-Ansatz mit dem HTN-Ansatz. Hierbei wurde fast der komplette Ablauf der HTN-Planung in einen SNLP ähnlichen Algorithmus eingepaßt. Der Ansatz ist domänenunabhängig, vollständig und korrekt. Da er in Kapitel 4 noch ausführlicher behandelt wird, sollen an dieser Stelle nur einige kurze Anmerkungen gemacht werden.

Young unterscheidet zwischen komplexen und primitiven Aktionen, die in der Planungswelt durchzuführen sind. Daneben gibt es die in der SNLP-Definition beschriebenen Ziele, die zu erreichen sind. Die Ziele werden durch literale Prädikatssymbole dargestellt. Die Aktionen werden in der Domänenbeschreibung durch STRIPS ähnliche Operatoren beschrieben. Den Unterschied zwischen primitiven und komplexen Aktionen beschreibt Young durch die Möglichkeit der Dekomposition mittels sogenannter *Dekompositionschemata*. Diese Schemata entsprechen den in Abschnitt 3.1.3 beschriebenen *Task-Netzen*. Die Schemata sind eindeutig auf eine Aktion anwendbar, was den Methoden aus selbigen Abschnitt entspricht. Young macht hierbei keinerlei Einschränkungen bezüglich der Vollständigkeit oder Konfliktfreiheit eines solchen Schemas,[12] sondern erlaubt auch partielle und unvollständige, ja sogar überlade-

[10] In UMCP ist es vorgesehen, daß domänenspezifische Verfeinerungsstrategien in die *critics*-Funktion τ integriert werden können.

[11] Der Name DPOCL (Decompositional POCL) kommt folgendermaßen zustande: POCL ist als konkrete Implementierung des SNLP-Algorithmus aus [Barrett und Weld, 1994a] bekannt geworden. [Young *et al.*, 1994] basiert auf diesem Algorithmus, erweitert ihn jedoch um die Dekomposition und komplexe Aktionen.

[12] Es sind natürlich nicht beliebige Schema für eine Aktion zugelassen, sondern sie müssen zu einer komplexen Aktion „passen“. Siehe hierzu auch Abschnitt 4.2.2.

ne[13] Schemata. Die Schemata selbst bestehen wieder aus primitiven und komplexen Schritten sowie Ordnungen und Causal Links zwischen ihnen. Da die von den Schritten verursachten Effekte durch Prädikate, die Variablen enthalten können, beschrieben werden, ist eine Menge von Variablenbindungs-Constraints ebenfalls Bestandteil eines solchen Schemas.

Man kann folgende Zuordnung der HTN-Konzepte auf DPOCL vornehmen:

- Goal-Tasks sind in DPCOL die zu erreichenden Vorbedingungen und Ziele.
- Primitive Tasks werden durch primitive Aktionen dargestellt.
- Compound-Tasks entsprechen den komplexen Aktionen.
- Task Netze werden in DPCOL durch die Dekompositionsschemata definiert.

Planungsprobleme und Domänenbeschreibung entsprechen dann, mit den entsprechenden Erweiterungen für die Dekompositionschemata, denen des SNLP-Ansatzes aus Abschnitt 2.1. Dadurch ergibt sich, daß die Beschreibungsmöglichkeiten für Planungsprobleme, wie sie der HTN-Ansatz bietet, von DPOCL nicht voll ausgeschöpft werden.

Der DPOCL-Planungsablauf

Der DPOCL-Algorithmus versucht zu einem gegebenen Planungsproblem eine Folge von primitiven Schritten anzugeben, die das Problem lösen. In dem in Abbildung 3.4 dargestellten Algorithmus erkennt man die Anlehnung an den SNLP-Algorithmus (vgl. Abbildung 2.4) deutlich. In jedem Inferenzschritt wird entweder ein normaler SNLP-Planungsschritt

DPOCL(P)

1. Wenn P vollständig , Stop Lösung.
   Alle Konflikte nicht lösbar, Stop keine Lösung.
2. Wähle aus:
   - Führe SNLP-Planungschritt aus.
   - Zerlege eine komplexe Aktion (falls vorhanden).
3. Bestimme auftretende Konflikte.
4. Löse auftretene Konflikte. Sei P' der neue Plan.
5. Rekusiver Aufruf: DPOCL(P')

Abbildung 3.4: Der DPOCL-Algorithmus im Überblick

ausgeführt, d.h. eine Vorbedingung erfüllt, oder es wird eine komplexe Aktion durch die Einführung eines Dekompositionsschemas zerlegt. Danach werden aller Interaktionen, die durch die neuen Planschritte mit den Causal Links entstehen, bestimmt und mit den SNLP-Strategien aufgelöst (vgl. Abschnitt 2.1.3). Weitere Details zu den Konzepten des DPOCL-Ansatzes finden sich in Kapitel 4.

[13] Gemeint sind hier Schemata die Aktionen enthalten, die eigentlich nichts dazu beitragen, die von der komplexen Aktion intendierten Effekte zu bewirken. Young spricht hier von Vorschlägen, die, wenn sie vom Planer nicht benutzt werden, bei der Lösung wieder entfernt werden.

3.3 Alternativen zum HTN-Ansatz

Einige Eigenschaften des HTN-Ansatzes wie Abstraktion und Strukturierung des Suchraumes leisten auch andere Ansätze, die sich in das Gebiet des hierarchischen Planens einordenen lassen. Hier sollen nun zwei Ansätze kurz erläutert werden, die zum Teil Konzepte aus dem HTN-Ansatz beinhalten, zum Teil andere Wege beschreiten. Beim ersten Ansatz handelt es sich um UCPOP+PARSE [Barrett und Weld, 1994b], eine „hierarchische Erweiterung" von UCPOP [Penberthy und Weld, 1992], und beim zweiten um ABTWEAK [Yang und Tenenberg, 1990], eine um Abstraktion erweiterte Variante des Planers TWEAK [Chapman, 1987].

3.3.1 UCPOP+PARSE

Der von [Barrett und Weld, 1994b] vorgeschlagene UCPOP+PARSE-Algorithmus, der auf UCPOP [Penberthy und Weld, 1992] aufbaut, fällt hier eigentlich etwas aus dem Rahmen, obwohl auch er mit hierarchischer Aktionsdekomposition arbeitet. Der Algorithmus ist domänenunabhängig, vollständig und korrekt. UCPOP kann als Erweiterterung des in Abschnitt 2.1 vorgestellten SNLP-Ansatz gesehen werden.

UCPOP+PARSE(*Problem*)

1. Initialisiere: *Plan* := UCPOP-Plan(*Problem*).
   Schemata := Parse(*Plan*).
2. Termination: Wenn *Plan Problem* löst, fertig
3. Lösungen := UCPOP-Planer(*Plan*).
4. *Schemata* := Parse(Lösungen).
   Wenn *Schemata* = $\emptyset$ dann keine Lösung.
5. Lösungen := Alle Lösungen, die sich parsen lassen.
   Plan := eine der möglichen Lösungen.
6. Gehe zu 2.

Abbildung 3.5: Der UCPOP+PARSE-Algorithmus im Überblick

UCPOP+PARSE unterscheidet sich deshalb von den hier vorgestellten Ansätzen, da er als einziger eine Bottom-Up-Strategie bei der Plangenerierung verfolgt. Hierbei wird ein Problem daduch zu lösen versucht, daß eine Folge von primitiven Schritten gesucht wird die das Problem löst, wie es UCPOP normalerweise macht. Aber nach jedem Planungsschritt wird versucht, die gefundenen Aktionen in komplexere Aktionen zu parsen. Es wird also ganz normal nicht-hierarchisch geplant, aber nur solche Pläne werden akzeptiert, die sich mittels der in der Domänenbeschreibung gegebenen Dekompositionsschemata parsen lassen (vgl. Abbildung 3.5). Dadurch verliert der Ansatz die Möglichkeit Planung und Ausführung zu verzahnen, weil es aufgrund der Bottom-Up Strategie nicht möglich ist, mit der Zerlegung einer abstrakteren Aktion zu warten. Diese Vorgehensweise widerspricht eigentlich dem, was man intuitiv von einem Ansatz erwartet, der mit hierarchischer Aktionsdekomposition arbeitet: nämlich die fortschreitende Konkretisierung einer Aktion und nicht deren Abstraktion.

Der Vorteil dieses Ansatzes ergibt sich aus der Kombination von UCPOP und der hierarchi-

schen Aktionsdekomposition. UCPOP verfügt über eine erweiterte Beschreibungsmöglichkeit der Aktionen mit allquantifizierten und bedingten Effekten. Durch die Aktionsdekomposition wird eine Reduktion Reduktion des Suchraums erzielt.

3.3.2 ABTWEAK

Abstracting TWEAK [Yang und Tenenberg, 1990], ist eine Erweiterung von TWEAK [Chapman, 1987]. Auch hierbei handelt es sich um ein domänenunabhängiges Planungssystem.

Dieser Ansatz stellt allerdings eine Alternative zur hierarchischen Aktionsdekomposition dar, das Verbergen von Vorbedingungen (*precondition hiding*). Die Idee hinter diesem Ansatz ist nicht neu. Sie wurde erstmals in ABSTRIPS [Sacerdoti, 1973] entwickelt und reflektiert damit das Verständnis, daß in den Anfängen der hierarchischen Planung von solchen System vorherrschte.

Hierzu wird bei ABTWEAK folgende Strategie verwendet: Das zu lösende Problem wird auf verschiedenen Abstraktionsebenen betrachtet. Dazu wird jeder zu erfüllenden Vorbedingung eines Operators mittels einer Funktion ein ganzzahliger Abstraktionslevel zugewiesen. Bei der Planung auf einer Abstraktionsebene werden nun alle Vorbedingungen die einen kleineren Abstraktionslevel haben, zu erfüllen versucht. Dabei wird auf jeder Abstraktionsebene nach dem TWEAK Algorithmus geplant. Dies ist im großen und ganzen auch die Vorgehensweise von ABSTRIPS. Eine solcher Ansatz wird heute mehr in den Bereich *Planen mit Abstraktion* eingeordnet als unter den hierarchischen Ansätzen. Dennoch beschreibt er einen wesentlichen Aspekt, den auch die hierarchischen Ansätze beinhalten, die Abstraktion von unwichtigen Details bei der Problemlösung.

Kapitel 4

Der DPOCL-Ansatz

Die Grundlage der hierarchischen Aktionsdekomposition, die im Planungssytem CAPLAN realisiert wurde, bildet der sogenannte DPOCL[1]-Ansatz aus [Young *et al.*, 1994]. DPOCL erweitert SNLP aus Abschnitt 2.1 um Aktionsdekomposition und umfaßt deshalb einige der grundlegenden Konzepte aus diesem Abschnitt. Daher wurde er ausgewählt, um im Planungssytem CAPLAN einen hierarchischen Planungsansatz zu realisieren.

Der hier beschriebene DPOCL-Ansatz ist vollständig und korrekt. Da er auf SNLP basiert, hat auch er die Eigenschaft der systematischen Suche im Raum der möglichen Pläne (vgl. Abschnitt 2.1.5). Im Anschluß werden die grundlegenden Begriffe und Konzepte von DPOCL vorgestellt.

4.1 Planungsprobleme und Domänenbeschreibung

Ein Planungsproblem wird in DPOCL durch die Angabe der Start- und Zielsituation in Form von literalen Atomformeln (Prädikaten) beschrieben. Diese Ziele können mittels der in der Domänenbeschreibung festgelegten Aktionen erreicht werden. Ziel des Ansatzes ist es, eine Sequenz von primitiven Aktionen, die im folgenden als Planschritte bezeichnet werden, zu erzeugen, welche die initiale Situation in die Zielsituation überführt.

Der DPOCL-Ansatz ist wie der SNLP-Ansatz domänenunabhängig. Die Domänenbeschreibung umfaßt bei DPOCL neben den Objekt-Typen und die mit ihnen definierten Aktionen in der Planungswelt noch die Angaben, auf welche Art bestimmte Aktionen durchzuführen sind (vgl. Abschnitt 4.2). Wie bei SNLP ist daher eine Domänenbescheibung Bestandteil eines Planungsproblems (siehe Abschnitt 2.1.1). Im folgenden Abschnitt wird die Darstellung von komplexen Aktionen in einer DPOCL-Domänenbeschreibung näher beschrieben.

4.2 Darstellung von Aktionen

Ein wesentlicher Bestandteil einer Domänenbeschreibung sind die mit den Objekten der Planungswelt durchführbaren Aktionen. In hierarchischen Ansätzen kommt noch hinzu, daß für manche Aktionen zusätzlich spezifiziert werden muß, auf welche Art sie konkret auszuführen sind. In DPOCL wird dies durch eine Erweiterung der Aktionsbeschreibung von SNLP erreicht.

[1] DPOCL steht hier für *Decompositional Partial-Order Causal Link Planner*, siehe auch Abschnitt 3.2.3.

4.2.1 Aktions- und Dekompositionsschemata

Die Beschreibung einer Aktion zerfällt bei DPOCL in zwei Teile. Der erste Teil umfaßt die „klassische“ Definition einer Aktion im Sinne eines STRIPS-Operators (vgl. Definition 4.1).

Definition 4.1 (Aktionsschema)
Ein Aktionsschema $\mathcal{A}$ ist ein Tupel $< A, V, P, E, B >$, wobei

- A *ein Aktionstyp,*
- V *eine Menge von freien Variablen,*
- P *eine Menge von literalen Vorbedingungen,*
- E *eine Menge von literalen Effekten und*
- B *eine Menge von Variablenbindungs-Constraints über die Variablen in V ist.*

Die Definition beschreibt eine in der Planungswelt durchführbare Aktion. Der Aktionstyp ist hierbei als eindeutiger Name der Aktion zu verstehen. Bei der Definition dieser Schemata sind für die Effekte und Vorbedingungen nur funktionsfreie Atomformeln zugelassen. Die Variablenbindungs-Constraints leiten sich durch die in den Atomformeln verwandten Variablen ab. Definition 4.1 deckt sich mit der Beschreibung eines SNLP-Operators (vgl. Abschnitt 2.1.1). Da Aktionen in einem DPOCL-Plan (vgl. Abschnitt 4.3) ebenfalls durch Planschritte repräsentiert werden, werden die Begriffe Aktion (bzw. Aktionsschema) und Planschritt im weiteren synonym verwendet.

Der zweite Aspekt bei der Definition von Aktionen beinhaltet die Spezifikation der möglichen Dekompositionsschemata für die Aktionen. Sie beschreiben, wie eine definierte Aktion explizit erreicht werden kann.

Definition 4.2 (Dekompositionsschema)
Ein Dekompositionsschema ist ein Tupel $\mathcal{D} =< A, S, O, B, L_c >$, wobei

- A *ein Aktionstyp,*
- S *eine Menge von Schritten (Aktionsschemata),*
- O *eine Menge von Ordnungen,*
- B *eine Menge von Variablenbindungsconstraints über die freien Variablen in S und*
- L_c *eine Menge von Causal Links zwischen den Schritten aus S ist.*

Die Definition beschreibt den Aufbau eines solchen Dekompositionsschemas[2]. Der Aktionstyp gibt an, auf welches Aktionsschema das Dekompositionsschema anwendbar ist. Durch die Anwendung eines Dekompositionsschemas auf ein Aktionsschema wird dieses in eine Reihe von Schritten zerlegt. Dabei ist es zulässig, daß in einem Dekompositionsschema wiederum komplexe Aktionen vorkommen.

Vergleicht man die Bestandteile eines solchen Dekompositionschemas mit den Komponenten eines SNLP-Plans (vgl. Definition 2.1), so erkennt man, daß es sich bei den Schemata um Planfragmente handelt, die die entsprechende komplexe Aktion ersetzen. Dabei werden keinerlei Einschränkungen bezüglich der Vollständigkeit oder Konfliktfreiheit der Fragmente gemacht.

Da die Schemata aus Definition 4.2 Planfragmenten entsprechen, sind ihre Bestandteile Elemente aus der SNLP-Begriffswelt (vgl. Abschnitt 2.1.2), wie Causal Links, Schritte und Ordnungen. Die Schritte, die in der Definition eines Schemas festgelegt werden, sind nur als

[2] Die hier beschriebenen Dekompositionsschemata entsprechen den *Task-Netzwerken* aus Abschnitt 3.1.3.

Platzhalter für die Planschritte zu sehen, die bei der Dekomposition einer komplexen Aktion mit dem Schema in einem DPOCL-Plan entstehen.

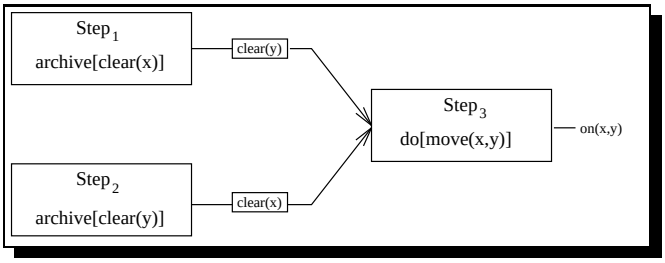


Abbildung 4.1: Beispiel eines Dekompositionsschemas

In Abbildung 4.1 ist ein Dekompositionsschema am Beispiel einiger Aktionen aus der Blocksworld dargestellt. Dabei sind die mit den Causal Links assoziierten Ordnungen nicht explizit eingetragen. Ein solches Dekompositionsschema könnte z.B. auf eine Aktion $MakeOn(x, y)$ anwendbar sein.

Aufbauend auf die Definitionen der Aktions- und Dekompositionsschemata unterscheidet man zwei Arten von Aktionen:

Komplexe Aktionen: Für sie sind eine oder mehrere Dekompositionsschemata definiert. Komplexe Aktionen entsprechen den *Compound-Tasks* aus Abschnitt 3.1.3. Sie werden erst durch Anwendung eines Dekompositionsschemas zu direkt in der Planungswelt zu ausführbaren Aktionen.

Primitive Aktionen: Für sie sind keine Dekompositionsschemata definiert, d.h. dies sind Schritte, die in der Planungswelt direkt ausführbar sind.

Planschritte werden dann, je nach Aktion, die sie repräsentieren, ebenfalls als komplexe oder primitive Schritte bezeichnet.

4.2.2 Einschränkungen bei der Definition der Dekompositionsschemata

Obwohl die Dekompositionsschemata aus Definition 4.2 beliebig sein können ist es wenig sinnvoll, hier willkürliche Planfragmente zuzulassen. Es werden daher einige Einschränkungen getroffen, wann ein Schema auf eine komplexe Aktion anwendbar ist. [Yang, 1990] zum Beispiel fordert, daß die Schemata konfliktfreie, fast vollständige Teilpläne sind und jeder Effekt bzw. Vorbedingung der komplexen Aktione im Schema von mindestens einem Schritt bedingt wird. [Young *et al.*, 1994] verlangt für DPOCL weniger restriktive Einschränkungen bezüglich der Definition eines Dekompositionsschemas:

- Jede Vorbedingung der zu zerlegenden Aktion wird von mindestens einem Schritt in dem Dekompositionsschema ebenfalls benötigt.
- Jeder Effekt der komplexen Aktion wird von mindestens einem Schritt in dem Schema ebenfalls bewirkt.
- Jedes Schema enthält einen künstlichen Schritt s_s, dessen Effekte die Vorbedingungen der übergeordeneten komplexen Aktion sind.

- Jedes Schema enthält einen künstlichen Schritt s_f, dessen Vorbedingungen die von der übergeordneten Aktion beabsichtigen Effekte sind.

- Für die künstlichen Schitte s_s und s_f gilt: Kein Schritt aus dem Dekompositionsschema ist vor s_s angeordnet und kein Schritt hinter s_f. Weiterhin läßt sich für jeden Effekt von s_s ein Pfad aus Causal Links zu einer Vorbedingung aus s_f angeben.

Die Idee der künstlichen Planschritte s_s und s_f ist nicht neu: SIPE [Wilkins, 1983] verwendet eine ähnliche Konstruktion unter dem Namen *Split-* und *Join*-Knoten. In DPOCL haben diese künstlichen Planschritte die folgenden Funktionen: Der künstliche Planschritt s_s zeigt an, daß ein Planschritt, der eine Vorbedingung der komplexen Aktion erfüllt, auch die entsprechenden Vorbedingungen der Schritte im Schema erfüllt. Dies bedeutet implizit, daß s_s die entsprechenden Vorbedingungen der Schritte garantiert, bis sie tatsächlich bei der komplexen Aktion erfüllt sind. Die Forderung nach einem Pfad von Causal Links von jedem Effekt von s_s zu einer Vorbedingung von s_f stellt hierbei sicher, daß alle Schritte, die eine Vorbedingung von der komplexen Aktion benötigen auch tatsächlich etwas zu den Effekten dieser beitragen.[3]

Einige der Vorbedingungen der Planschritte im Schema können, wie aus Definition 4.2 ersichtlich, schon durch Causal Links garantiert sein. Es kann aber noch Vorbedingungen geben, die noch nicht durch Causal Links garantiert sind. Diese Vorbedingungen können dann beliebig, entweder durch Einführung neuer Schritte, oder durch bereits im Plan vorhandene Schritte, erfüllt werden.

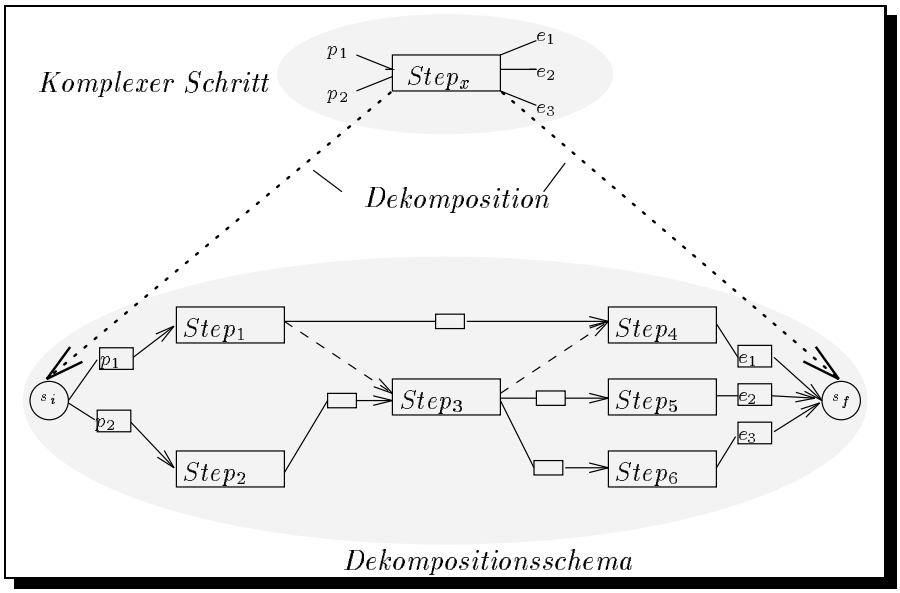


Abbildung 4.2: Beispiel einer komplexen Aktion

Abbildung 4.2 verdeutlicht den Zusammenhang dieser Einschränkungen bezüglich der Definition eines Dekompositionsschemas und einer komplexen Aktion.

[3] Daneben können in einem Dekompositionsschema auch Schritte vorkommen, die keinen der von s_s bewirkten Effekte benötigen. Young spricht hier von Vorschlägen, die ein Planer benutzen kann oder nicht.

Konsistenzeigenschaften von Dekompositionsschemata

Mit den Einschränkungen aus Abschnitt 4.2.2 kann nun folgende Konsistenzeigenschaft bezüglich der Ordnungen innerhalb eines Dekompositionsschemas definiert werden:

Definition 4.3 (Ordnungskonsistenz von Dekompositonsschemata)
Ein Dekompositionsschema $\mathcal{D} =< A_\mathcal{D}, S_\mathcal{D}, O_\mathcal{D}, B_\mathcal{D}, L_{c_\mathcal{D}} >$ *heißt ordnungskonsistent, wenn gilt:*

- *Kein Schritt* $S_i \in S_\mathcal{D}$ *ist vor* s_s *angeordnet.*
- *Kein Schritt* $S_i \in S_\mathcal{D}$ *ist hinter* s_f *angeordnet.*
- *Das Schema enthält keine Zykel von Schritten bezüglich ihrer Ordnung.*

Die Definition der Ordnungskonsistenz ist an die Konsistenzeigenschaften von SNLP-Plänen (vgl. Abschnitt 2.1.4) angelehnt, da jedes Dekompositionsschema ein kleines Planfragment repräsentiert. Auch hier beschreiben die Ordnungen zwischen den Schritten deren zeitliche Abfolge. Daher ist die Zykelfreiheit der Dekompositionsschemata ein wichtiges Konsistenzkriterium. Die künstlichen Dekompositionschritte s_s und s_f garantieren die entsprechenden Vorbedingungen und Effekte der zerlegten komplexen Aktion, weshalb eine Anordnung der Schemaschritte vor bzw. hinter diesen Schritten nicht zulässig ist.

Die Konsistenzeigenschaft bezüglich der Variablenbindungs-Constraints wird an dieser Stelle nicht definiert, da die Konsistenzeigenschaft eines DPOCL-Plans (vgl. Definition 4.6) diesen Zusammenhang mit umfaßt.

4.3 Plandarstellung bei DPOCL

DPOCL sucht zu einem gegebenen Planungsproblem systematisch einen Plan mit einer (partiell) geordneten Folge von primitiven Aktionen, die das Problem lösen (vgl. Abschnitt 4.1). Dieser Plan enthält folgende Komponenten:

Definition 4.4 (DPOCL-Plan)
Ein DPOCL-Plan $\mathcal{P}$ *ist ein Tupel* $< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c, \mathcal{L}_d >$, *wobei*

- $\mathcal{S}$ *eine Menge von Schritten,*
- $\mathcal{O}$ *eine Menge von Ordnungen über den Elementen aus* $\mathcal{S}$,
- $\mathcal{B}$ *eine Menge von Variablenbindungs-Constraints über den freien Variablen in* $\mathcal{S}$,
- $\mathcal{L}_c$ *eine Menge von Causal Links zwischen den Elementen aus* $\mathcal{S}$ *und*
- $\mathcal{L}_d$ *eine Menge von Dekompositionslinks über den Elementen aus* $\mathcal{S}$

ist.

Da der DPOCL-Ansatz auf SNLP basiert, finden sich ein einem DPOCL-Plan ähnliche Komponenten wie in SNLP-Plänen (vgl. Abschnitt 2.1.2). Wie bei SNLP besteht ein DPOCL-Plan aus Schritten und einer auf diesen Schritten definierten (partiellen) Ordnung. Die Schritte repräsentieren die in Abschnitt 4.2 beschriebenen Aktionsschemata. Dabei stehen die im Plan verwendeten Schritte als Instanzen für das jeweilige Aktionsschema. Man unterscheidet bei DPOCL insgesamt drei Arten von Planschritten:

Primitive Schritte: Sind Aktionen, die in der Planungswelt direkt ausführbar sind. Ein Beispiel hierfür wäre eine Aktion *PutOn(a,b)* aus der Blocksworld, die zwei Klötzchen aufeinander stellt.

Komplexe Schritte: Hierbei handelt es sich um Aktionen, die nicht direkt ausführbar sind. Bei diesen Aktionen muß noch festgelegt werden, wie sie konkret auszuführen sind. Wie z.B. bei einer Aktion *Baue-Haus*.

Künstliche Schritte: Dies sind zum einen die schon in Abschnitt 2.1.2 beschriebenen Schritte $START$ und $FINISH$, die auch hier das Planungsproblem repräsentieren. Daneben finden sich noch die künstlichen Dekompositionsschritte s_s und s_f.

Ein Planschritt im Plan wird nun je nach Aktion, die er repräsentiert, komplex oder primitiv genannt. Man sieht auch in der Definition 4.4 die enge Anlehnung an SNLP (vgl. Definition 2.1). Als neuen Bestandteil des DPOCL-Plans findet sich hier die Dekompositionslinks $\mathcal{L}_d$, welche die Zerlegung der komplexen Aktionen festhalten.

4.3.1 Kausalität in DPOCL-Plänen

Für eine Lösung eines Planungsproblems (d.h. einen linearisierten Plan) muß gelten, daß die Vorbedingungen der einzelnen Schritte zur Ausführungszeit der Planschritte gelten. Um diese Bedingung zu erfüllen, werden Vorbedingungen von Schritten wie bei SNLP durch Causal Links (siehe Definiton 2.2) garantiert (vgl. Abschnitt 2.1.3).

Ein ähnliches Konzept existiert für die Beziehung zwischen einem komplexen Schritt und einem Dekompositionsschema. Im Mittelpunkt dieses Konzepts steht, wie auch beim Causal Link, die Darstellung eines kausalen Zusammenhangs. Hier wird allerdings die Zerlegung eines komplexen Schrittes mittels eines Dekompositionsschemas festgehalten.

Definition 4.5 (Dekompositionslink)
Sei S eine komplexe Aktion, $\mathcal{D} =< A_\mathcal{D}, S_\mathcal{D}, O_\mathcal{D}, B_\mathcal{D}, L_{c_\mathcal{D}} >$ ein Dekompositionsschema mit passendem Aktionstyp $A_\mathcal{D}$. Dann hält $S \overset{\mathcal{D}}{\rightsquigarrow} S_\mathcal{D}$ die Zerlegung von S mit $\mathcal{D}$ fest.

Definition 4.5 hält die Tatsache fest, daß die Planschritte eines Schemas Teil der Zerlegung einer komplexen Aktion sind (siehe auch Abbildung 4.2). Auf der anderen Seite zeigt ein solcher Dekompositionslink an, daß eine komplexe Aktion nun in eine Reihe primitiver (konkreterer) Schritte zerlegt wurde.

Interaktionen zwischen Schritten

Durch die Verwendung des Konzepts der Causal Links, müssen noch die Interaktionen zwischen Schritten betrachtet werden (vgl. Abschnitt 2.1.3). Diese Interaktionen werden wie beim SNLP-Ansatz als *Threats* bezeichnet. Ihre Definition deckt sich mit der aus Definition 2.3.

Die Auflösung von Interaktionen erfolgt durch eine der in Abschnitt 2.1.3 beschriebenen Techniken: *Demotion*, *Promotion* oder *Separation*. Die zur Auflösung von Interaktionen eingeführten Ordnungen werden wie bei SNLP als *Protections* bezeichnet.

4.3.2 Eigenschaften von DPOCL-Plänen

Analog zu Definition 2.4 für SNLP-Pläne, kann man für DPOCL-Pläne im Bezug auf Ordnungen und Variablenbindungs-Constraints die nachfolgenden Konsistenzeigenschaften definieren:

Definition 4.6 (Konsistenz von DPOCL-Plänen)
Ein DPOCL-Plan $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c, \mathcal{L}_d >$ heißt konsistent, wenn gilt:

1. *Ordnungskonsistenz:*
   - *kein Schritt vor $START$ angeordnet ist,*
   - *kein Schritt hinter $FINISH$ angeordnet ist,*
   - *der Plan bezüglich der Ordnungen zykelfrei ist und*
   - *jedes Dekompositionsschema $\mathcal{D}$, das in einem Dekompositionslink in $\mathcal{L}_D$ vorkommt, ordnungskonsistent ist.*
2. *Bindungskonsistenz:*

   Wenn die Menge der Variablenbindungs-Constraints konsistent ist, d.h. für jede Variable mindestens eine Bindungsmöglichkeit in $\mathcal{B}$ existiert, die konsistent mit allen anderen Constraints aus $\mathcal{B}$ ist.

Die Konsistenzeigenschaften von DPOCL-Plänen ergeben sich analog denen eines SNLP-Plans (vgl. Definition 2.4). Die Schritte $START$ und $FINISH$ repräsentieren das zu lösende Problem. Daher ist eine Anordnung der Planschritte vor bzw. hinter diesen künstlichen Schritten nicht möglich. Die Zykelfreiheit ist wiederum eine Konsequenz, die sich aus der zeitlichen Anordnung der Planschritte ergibt. Neu im Vergleich zur Konsistenz der SNLP-Pläne ist nur der Punkt, in dem die Ordnungskonsistenz der im Plan verwendeten Dekompositionsschemata gefordert wird. Die Forderung ist notwendig, da die Schemata bei ihrer Einführung in den Plan keine Ordnungsbeziehungen zu den anderen Planschritten erhalten, sondern diese Beziehungen nur implizit durch die künstlichen Schemaschritte s_s und s_f ausgedrückt wird. Die Bindungskonsistenz begründet sich analog zu der von SNLP-Plänen (vgl. Abschnitt 2.1.4).

Ziel des DPOCL-Algorithmus ist die Erzeugung eines Planes, der ein gegebenes Planungsproblem löst. An diese Lösung werden einige Anforderungen gestellt, die unter der Vollständigkeitseigenschaft eines Planes zusammengefaßt werden .

Definition 4.7 (Vollständigkkeit von DPOCL-Plänen)
Ein DPOCL-Plan $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c, \mathcal{L}_d >$ heißt vollständig, wenn gilt:

1. *Für alle Vorbedingungen p der im Plan benutzen Schritte S_j existiert ein Causal Link der Form $S_i \xrightarrow{p} S_j \in \mathcal{L}_c$.*
2. *Alle Causal Links aus $\mathcal{L}_c$ sind gegen alle Threats geschützt.*
3. *Für alle komplexen Schritte $S_i \in \mathcal{S}$ existiert ein Dekompostionslink der Form $S_i \overset{D}{\rightsquigarrow} S_D \in \mathcal{L}_d$*

Auch hier zeigt ein Vergleich mit Definition 2.5, daß sich kaum Veränderungen gegenüber dem SNLP-Ansatz ergeben. Wie bei vollständigen SNLP-Plänen müssen alle Vorbedingungen der Planschritte durch geschützte Causal Links garantiert sein. Allerdings gilt ein DPOCL-Plan

erst dann als vollständig, wenn alle komplexen Aktionen, die in einem DPOCL-Plan vorkommen, durch Dekomposition zerlegt sind. Das läßt sich intuitiv auch erwarten, da komplexe Aktionen ihrer Bedeutung nach, nicht direkt in der Planungswelt ausführbar sind. [Young *et al.*, 1994] zeigt, daß durch die Erweiterung des SNLP-Ansatzes zu DPOCL, weder die Vollständigkeit, noch die Systematik der Suche im Raum der möglichen Pläne verloren geht.

4.3.3 Unterschiede von SNLP- und DPOCL-Plänen

In den vorangegangenen Abschnitten wurden die wesentlichen Begriffe des DPOCL-Ansatzes beschrieben. Abbildung 4.3 zeigt, die in einem DPOCL-Plan verwendeten Symbole. Die zu SNLP neu hinzugekommenen Symbole sind unterlegt dargestellt.

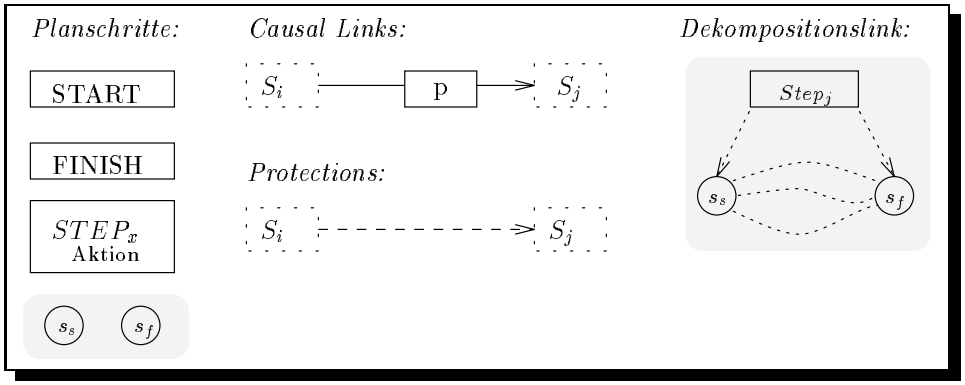


Abbildung 4.3: Zusammenfassung der DPOCL-Planunssymbole

Vergleicht man die SNLP- und DPOCL-Pläne (siehe Definitionen 2.1 und 4.4), so kann festgehalten werden, daß DPOCL hier nur eine Erweiterung vornimmt: Es wird eine zusätzliche Komponente in den Plan aufgenommen, die es erlaubt, die Zerlegung von komplexen Aktionen zu erfassen. Mit dieser neuen Komponente werden dann die Konsistenz- und Vollständigkeitseigenschaften eines Plans so erweitert, daß die hierarchischen Aspekte reflektiert werden. Dies bedeutet, daß die einzige Veränderung gegenüber SNLP-Plänen, die Einbindung der Dekompositionen darstellt.

4.4 Der DPOCL-Algorithmus

In der Abbildung 4.4 ist der DPOCL-Algorithmus dargestellt. Vor dem ersten Aufruf wird der DPOCL-Plan $\mathcal{P}$ mit den Schritten $START$ und $FINISH$ sowie der Ordnung $START \prec FINISH$ initialisiert. Damit ist dann, wie bei SNLP, daß Planungsproblem für den Algorithmus repräsentiert. Aufgerufen wird der Algorithmus dann rekursiv mit dem Plan $\mathcal{P}$. Die Beschreibung der Planungsdomäne wird mit $DOM =< \Lambda, \Sigma >$ notiert, wobei Λ die Menge der zur Verfügung stehenden Aktionsschemata (Operatoren) und Σ die Menge der möglichen Dekompositionsschemata beschreibt.

Zum besseren Verständnis des Algorithmus werden einige Notationen erläutert:

- *constr*(S_{source}, e, p) liefert eine Menge von Variablenbindungs-Constraints, die sicherstellen, daß die Prädikate e und p eine gleiche Variablenbindung erhalten.

DPOCL($\mathcal{P}$)

1. **Termination:**
   Falls $\mathcal{P}$ vollständig ist, dann fertig (Erfolg)
2. **Planverfeinerung:** Wähle nichtdeterministisch eine Möglichkeit:
   (a) Vorbedingungen erfüllen:
      i. Ziel-Auswahl: Wähle Vorbedingung p eines Schrittes S_{dest} für die kein Causal Link in $\mathcal{L}_c$ existiert.
      ii. Operator-Auswahl: Sei S_{source} ein Schritt mit Effekt e (entweder ein bereits in $\mathcal{S}$ vorhandener oder ein durch Anwendung eines Operators aus Λ entstehender). Falls kein solcher Schritt S_{source} existiert, führe Backtracking durch.

      $\mathcal{S}' := \mathcal{S} \cup \{S_{source}\}$ $\quad$ $\mathcal{O}' := \mathcal{O} \cup \{S_{source} \prec S_{dest}\}$

      $\mathcal{L}'_c := \mathcal{L}_c \cup \{S_{source} \xrightarrow{p} S_{dest}\}$ $\quad$ $\mathcal{B}' := \mathcal{B} \cup \{constr(S_{source}, e, p)\}$ $\quad$ $\mathcal{L}'_d := \mathcal{L}_d$

      Backtracking-Punkt: Jede Möglichkeit einen Schritt S_{source} so zu wählen, daß ein konsistenter Plan $\mathcal{P}' =< \mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}'_c, \mathcal{L}'_d >$ entsteht.

   (b) Komplexe Schritte zerlegen:
      i. Aktions-Auswahl: Wähle zufällig einen Schritt $S_{complex} \in \mathcal{S}$ ohne Dekompositionlink in $\mathcal{L}_d$ aus.
      ii. Dekompositions-Auswahl:
      $D :=$ Wähle-Schema($S_{complex}$)
      Falls kein solches Schema D existiert, führe Backtracking durch.
      Expandiere(D)

      $\mathcal{S}' := \mathcal{S} \cup S_D$ $\quad$ $\mathcal{O}' := \mathcal{O} \cup O_D$

      $\mathcal{L}'_c := \mathcal{L}_c \cup L_{c_D}$ $\quad$ $\mathcal{B}' := \mathcal{B} \cup B_D$ $\quad$ $\mathcal{L}'_d := \mathcal{L}_d \cup \{S_{complex} \overset{D}{\rightsquigarrow} S_D\}$

      Backtracking-Punkt: Jede Möglichkeit ein Schema D so zu wählen, daß ein konsistenter Plan $\mathcal{P}' =< \mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}'_c, \mathcal{L}'_d >$ entsteht.
3. **Auflösung der Threats:** Für jeden Schritt S_{threat} der einen Causal Link $S_i \xrightarrow{p} S_j \in \mathcal{L}'_c$ bedroht: Schütze den Causal Link durch Hinzunahme von Ordnungen oder Constraints zu $\mathcal{O}'$ bzw. $\mathcal{B}'$.
   Backtracking-Punkt: Jede Möglichkeit einen Causal Link so zu schützen, daß ein konsistenter Plan $\mathcal{P}' =< \mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}'_c, \mathcal{L}'_d >$ entsteht.
4. **Rekursiver Aufruf:** DPOCL(P').

Abbildung 4.4: Der DPOCL-Algorithmus

- *Wähle-Schema($S_{complex}$)* bedeutet, daß ein Schema $D =< A_D, S_D, B_D, O_D, L_{c_D} > \in \Sigma$ mit passendem Aktionstyp A_D zu $S_{complex}$ ausgewählt wird.
- *Expandiere(D)* hat zur Folge, daß die Schritte in S_D durch aktuell in $\mathcal{S}$ enthaltene Schritte oder durch Einführung eines neuen Operators aus Λ entstehenden Schritts, ersetzt werden und die Referenzen in B_D, O_D, L_{c_D} entsprechend geändert werden.

Der Algorithmus aus Abbildung 4.4 besteht im wesentlichen aus zwei Teilen. Entweder wird ein SNLP-Planungschritt (vgl. Abschnitt 2.1.5) ausgeführt oder eine komplexe Aktion wird

mittels eines Dekompositionsschemas zerlegt. Die Bestimmung der Interaktionen und deren Auflösung verläuft analog zu dem in Abschnitt 2.1.5 beschriebenen SNLP-Algorithmus. Vergleicht man die beiden Algorithmen, so stellt man fest, daß sich hier nur das Abbruchkriterium, die Vollständigkeitseigenschaft des Plans erweitert hat und ein zusätzlicher Planungschritt in den ursprünglichen SNLP-Algorithmus eingebaut wurde. Daher bleiben auch die Möglichkeiten der Termination des Algorithmus gleich denen aus Abschnitt 2.1.5. Die dort vorgestellten Verbesserungen der Terminationseigenschaften können in den DPOCL-Algorithmus mit übernommen werden, da sie nur den SNLP-Planungsschritt betreffen.

4.4.1 Vergleich von DPOCL und SNLP

Beim Vergleich der Grundkonzepte (vgl. Abschnitt 4.3.3) stellt man eine große Übereinstimmung zwischen dem SNLP- und DPOCL-Ansatz fest. DPOCL erweitert die SNLP-Konzepte lediglich um den Punkt der Dekomposition von Aktionen, läßt aber alle übrigen Konzepte wie Causal Links und Threats unverändert. Daraus ergeben sich auch fast identische Konsistenz- und Vollständigkeitskriterien, was unter anderem auch in der Zielsetzung der beiden Ansätze deutlich wird. Beide erzeugen einen Plan, der eine Sequenz von primitiven Planschritten enthält, die ein Planungsproblem lösen.

Dieses Vorgehen spiegelt sich auch im DPOCL-Algorithmus aus Abbildung 4.4 wieder. Hier findet sich der SNLP-Algorithmus aus Abbildung 2.4 erweitert um die Zerlegung von komplexen Aktionen.

Insgesamt stellt der DPOCL-Ansatz eine einfache Verbindung zwischen den HTN-Konzepten (vgl. Abschnitt 3.1.3) und SNLP her. Dabei sind die notwendigen Erweiterungen der SNLP-Konzepte nur minimal (vgl. Abschnitt 4.3.3). Die Definition der Dekompositionsschemata ist bezüglich Konfliktfreiheit und Vollständigkeit der Planfragmente nicht eingeschränkt, so daß bei der Modellierung einer Planungsdomäne eine sehr große Freiheit besteht. Durch diese enge Anlehnung an SNLP, ergibt sich, daß eine Erweiterung des Planungsablaufs eines SNLP-basierten Planers um Aktionsdekomposition relativ einfach gelingt.

4.5 Erweiterungen des DPOCL-Algorithmus

Eine Veränderung des Algorithmus von Young wurde bezüglich der Interaktionsauflösung vorgenommen. Neben den normalen SNLP-Strategien zur Auflösung von Interaktionen kann eine Interaktion, die auf einem höhren Planungslevel unauflösbar scheint, durch Dekomposition auflösbar werden. Das hat zur Folge, daß bei nicht auflösbaren Interaktionen nicht sofort Backtracking durchgeführt werden sollte, da durch weitere Verfeinerung der Aktionen durchaus wieder ein konsistenter Plan entstehen kann. Bei Young führen diese unauflösbaren Konflikte zum Backtracking.[4] In [Yang, 1990] wird ein Kriterium beschrieben, das es erlaubt bei solchen Konflikten zu entscheiden, ob der Konflikt durch Dekomposition aufgelöst werden kann oder nicht.

[4] Durch das Vorgehen von Young verliert man interessanterweise nicht die Vollständigkeitseigenschaften des Algorithmus, was man sich leicht klarmachen kann, da der DPOCL-Algorithmus im Extremfall als SNLP-Algorithmus, d.h. völlig ohne Aktionsdekomposition, arbeitet.

4.5.1 Erweiterung der Interaktionsauflösung

Unauflösbare Konflikte, die auf abstrakteren Planungsebenen auftauchen, können durch Dekomposition der Aktionen durchaus lösbar sein, da es evtl. möglich ist, die Schemata durch Protections so zu ordnen, daß die Schemata ineinander verzahnt werden. Nimmt man dieses Vorgehen, also das Auflösen von Threats durch Dekomposition von Aktionen, als vierte Strategie zu den in Abschnitt 2.1.3 beschriebenen hinzu, so verliert man zwar die Systematikeigenschaft des Algorithmus, kommt aber einer möglichen Lösung eventuell früher näher. In dem von Young vorgestellten Algorithmus wird dieses Vorgehen nicht eingebunden, da hier die Systematikeigenschaft von SNLP erhalten bleiben sollte. Das hat zur Folge, daß Teilpläne, die auch zu einer Lösung führen, möglicherweise zu früh verworfen werden. Man kann jedoch festhalten, daß durch das frührere Backtracking der Algorithmus nicht unvollständig wird, aber das eine mögliche Lösung u.U. zu früh verworfen wird.

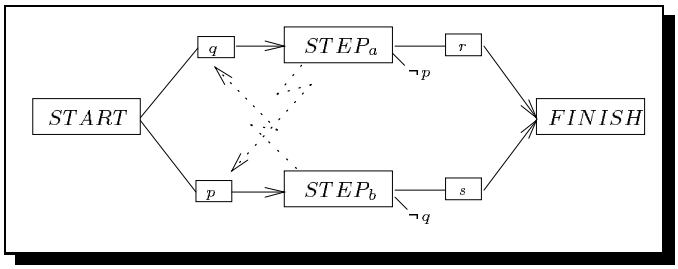


Abbildung 4.5: Beispiel eines unauflösbaren Konflikts

Um dies zu vermeiden, muß also versucht werden, einen potentiell unauflösbaren Konflikt durch eine Verfeinerung der beteiligten Aktionen zu beseitigen. In Abbildung 4.5 ist eine solche Situation dargestellt. Aus dem Beispiel wird ersichtlich, daß es nicht möglich ist $STEP_a$ hinter $STEP_b$ anzuordnen, da gleichzeitig $STEP_b$ hinter $STEP_a$ angeordnet werden müßte, um die Konflikte aufzulösen. Konflikte dieser Art werden auch als *double-cross Konflikte* bezeichnet.

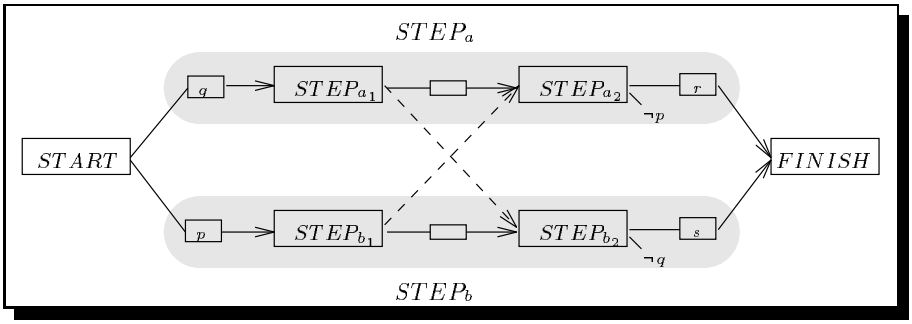


Abbildung 4.6: Auflösung des Konflikts aus Abbildung 4.5 durch Aktionsdekomposition

Nimmt man dagegen an, die beiden Aktionen könnten durch die in Abbildung 4.6 dargestellten Dekompositionen verfeinert werden, so läßt sich der Konflikt, wie in der Abbildung angedeutet, auflösen. Das Problem ist zu erkennen, ob man sich in einer Sitution befindet, in der ein Konflikt durch Dekomposition der Schritte aufgelöst werden kann. Ein solches Entscheidungskriterium könnte verhindern, daß man weiter in eine Richtung plant, die keine Lösung erwarten läßt.

Um das im folgenden beschriebene Kriterium zu motivieren, sei nochmals auf Abbildung 4.5 verwiesen. Tritt auf einer abstrakteren Ebene des Planungsprozesses ein unlösbarer Konflikt auf, so deutet er schon an, daß in diesem Teil des Suchraums ein Problem auftreten könnte, d.h. es ist hier zumindest unwahrscheinlich eine Lösung zu finden, unabhängig davon, ob eine weitere Zerlegung der Aktionen durchgeführt wird. Andererseits muß auch gelten, daß wenn doch eine Lösung des Konflikts existiert, dann ist die gefundene Situation auf der abstrakteren Planungsebene ebenso korrekt, wie die verfeinerte Situation. Ziel ist es also, anhand der zur Auswahl stehenden Verfeinerungsvarianten entscheiden zu können, ob ein Konflikt auf einer konkreteren Planungsebene auflösbar wird, um dann gegebenfalls früher Backtracking durchführen zu können. [Yang, 1990] macht hierzu einige Vorschläge.

4.5.2 Abwärtslinearisierbarkeit von Dekompositionsschemata

Die Auflösung von Konflikten geschieht entweder durch Einführung von Constraints, die verhindern, daß ein Konflikt auftreten kann, oder durch die Anordnung der Schritte durch Protections. Im letzteren Fall spricht [Yang, 1990] auch von der Linearisierung eines Plans. Ziel dabei ist immer, einen konsistenen Plan zu erhalten. Betrachtet man eine (Teil-)Menge der Causal Links in einem Plan, die hier mit $\mathcal{L}_{c_\Delta}$ bezeichnet wird, so kann damit folgende Eigenschaft definiert werden:

Definition 4.8 ($\mathcal{L}_{c_\Delta}$-konsistent, $\mathcal{L}_{c_\Delta}$-linearisierbar)
Sei $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}_c, \mathcal{L}_d >$ ein Plan und $\mathcal{L}_{c_\Delta} \subseteq \mathcal{L}_c$ eine (Teil-)Menge von Causal Links in $\mathcal{P}$. Dann heißt $\mathcal{P}$ $\mathcal{L}_{c_\Delta}$-linearisierbar genau dann, wenn es eine Anordnung L der Schritte in $\mathcal{P}$ gibt, so daß alle Causal Links geschützt sind.

Die Linearisierung L wird dann auch $\mathcal{L}_{c_\Delta}$-konsistent genannt.

Um die Definition etwas näher zu erläutern, sei auf das Beispiel aus Abbildung 4.7 verwiesen.

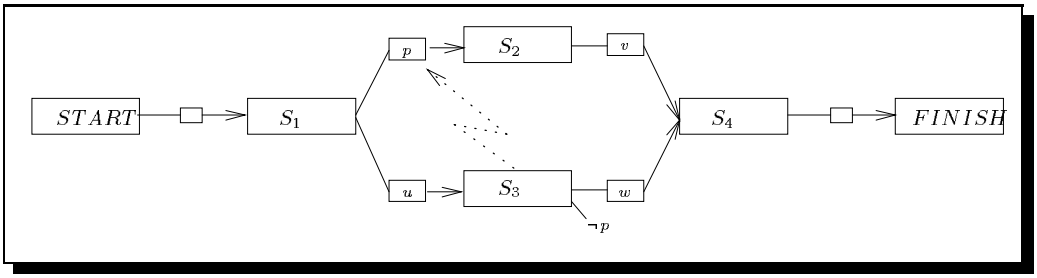


Abbildung 4.7: Teilplan mit Causal Links und Konflikt.

In dem Teilplan aus Abbildung 4.7 ist ein Konflikt zwischen den Planschritt S_3 und dem Causal Link $S_1 \xrightarrow{p} S_2$ dargestellt. Betrachtet man die Causal Links der Menge $\mathcal{L}_{c_\Delta} = \{ S_1 \xrightarrow{p} S_2,\ S_2 \xrightarrow{v} S_4,\ S_1 \xrightarrow{u} S_3,\ S_3 \xrightarrow{w} S_4 \}$, dann wird deutlich, daß $\mathcal{P}$ in diesem Fall $\mathcal{L}_{c_\Delta}$-linearisierbar ist. Eine mögliche Anordnung der Schritte, die den Konflikt auflöst ist in Abbildung 4.8 dargestellt. Die dort dargestellte Linearisierung der Planschritte ist $L = (S_1 \prec S_2 \prec S_3 \prec S_4)$. Es sei noch angemerkt, daß es meist mehrere Möglichkeiten für die Linearisierung einer Menge $\mathcal{L}_{c_\Delta}$ gibt. Dies ergibt sich aus den verschiedenen Strategien zur Auflösung von Interaktionen durch die Einführung von Protections (vgl. Abschnitt 2.1.3).

Das Beispiel aus Abbildung 4.5 ist dagegen nicht $\mathcal{L}_{c_\Delta}$-linearisierbar. Betrachtet man $\mathcal{L}_{c_\Delta} = \{ START \xrightarrow{q} STEP_a,\ START \xrightarrow{p} STEP_b \}$, so läßt sich keine Linearisierung der Plan-

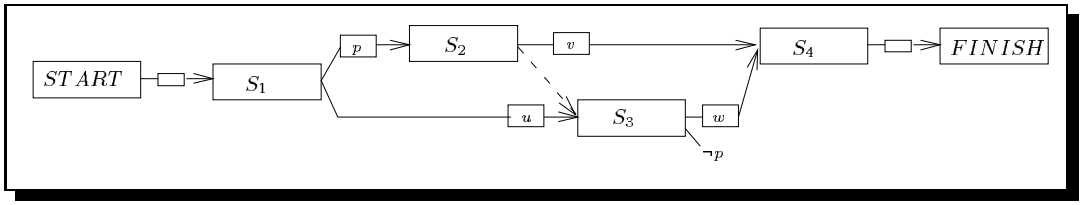


Abbildung 4.8: Linearisierung eines Teilplans.

schritte finden, die alle Konflikte auflöst.

Betrachtet man den Planungsablauf des hierarchischen Planens, dann kann dieses Kriterium auch auf einen Plan $\mathcal{P}'$, der durch die Anwendung einer Sequenz $D_1, \ldots, D_n \in \Sigma$ von Dekompositionsschemata aus einem Plan $\mathcal{P}$ entsteht, erweitern. In diesem Fall spricht man dann von der Abwärtslinearisierbarkeit von Plänen.

Das Kriterium der Abwärtslinearisierbarkeit

Ziel ist es, anhand der Dekompositionsschemata zu entscheiden, ob ein Konflikt auf einer abstrakteren Ebene, durch Verfeinerung auflösbar wird. Dazu müssen die Konflikte auch auf der konkreteren Planungsebene unauflösbar bleiben, d.h. sich über die verschiedenen Planungsebenen erhalten.

Definition 4.9 (Abwärtslinearisierbarkeit)
Sei Σ eine Menge von Dekompositionsschemata, $\mathcal{P}$ ein DPOCL-Plan und $\mathcal{P}_D$ der Plan der aus $\mathcal{P}$ durch Anwendung von $D_1, \ldots, D_n \in \Sigma$ entsteht. Σ heißt **nicht abwärtslinearisierbar** *wenn folgende Bedingung gilt:*

Wenn $\mathcal{P}$ nicht $\mathcal{L}_{c_\Delta}$-linearisierbar ist, dann ist auch $\mathcal{P}_D$ nicht $\mathcal{L}_{c_\Delta}$-linearisierbar.

Ein Konflikt auf der abstrakteren Planungsebene bleibt auch durch Dekomposition der Aktionen unauflösbar, d.h. man kann ebenfalls keine Linearisierung angeben, die den Konflikt auf der konkreteren Planunsebene löst. Es ergibt sich die Fragestellung, wie eine Menge Σ von Dekompositionsschemata beschaffen sein muß, um das Kriterium aus Definition 4.9 zu erfüllen.

4.5.3 Syntaktische Beschränkungen der Schemata

Auflösung von Konflikten durch Dekomposition beinhaltet, daß eine Anordnung der verfeinerten Aktionen derart stattfindet (siehe auf Abbildung 4.6), daß die bedrohten Vorbedingungen durch Verzahnung der Schemata geschützt werden.

[Yang, 1990] zeigt, daß folgende Bedingung genügt, um festzustellen, ob eine Menge Σ von Dekompositionsschemata das Kriterium aus Definition 4.9 erfüllt und damit bei auftretenden Konflikten entscheiden zu können, ob eine Dekomposition der beteiligten Aktionen zur Auflösung des Konflikts führt.

Definition 4.10 (*main subaction* Einschränkung)
Sei $A \in \Lambda$ eine komplexe Aktion und $D \in \Sigma$ eine auf A anwendbare Dekomposition. Dann existiert in D eine primitive Aktion A_{main} mit:

1. *A_{main} bedingt alle Effekte von A und keine andere primitive Aktion in D bedingt einen Effekt von A.*

2. *A_{main} benötigt alle Vorbedingungen, die auch A benötigt und keine andere Aktion in D kann diese Vorbedingungen etablieren.*

Man kann sich die Einschränkung aus Definition 4.10 so vorstellen, daß jedes Dekompositionsschema eine Hauptaktion A_{main} enthält, die alle Effekte, die auch die komplexe Aktion bewirkt, einführt und die dafür sorgt, daß die von der komplexen Aktion bedingten Vorbedingungen bis zu dieser Hauptaktion im Schema benötigt werden. [Yang, 1990]) zeigt, daß alle Schemata, die die in der Definition 4.10 beschriebene Eigenschaft haben, auch die Definition 4.9 erfüllen. Plant ein System mit einer Domäne, die nur solche Schemata enthält, so kann bei einem Konflikt auf der abstrakten Ebene schon gefolgert werden, daß dieser Konflikt auch auf einer konkreteren Planungsebene nicht auflösbar sein wird.

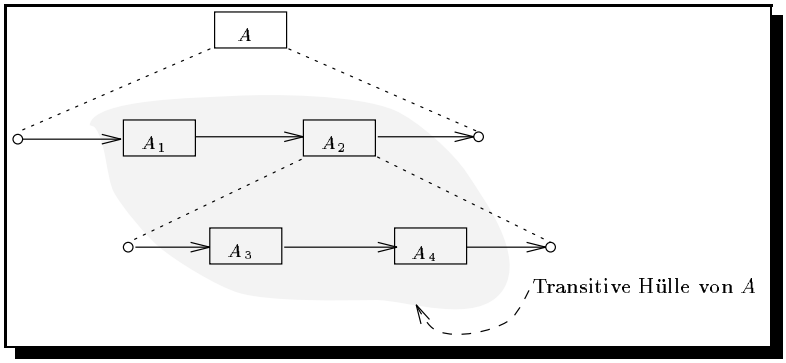


Abbildung 4.9: Transitive Hülle einer komplexen Aktion

Enthält das Dekompositonschema nicht nur primitive Aktionen, so muß die transitive Hülle der komplexen Aktion betrachtet werden. Die transitive Hülle einer solchen Aktion erhält man, in dem man alle komplexen Schritte und deren Dekompositionen betrachtet, die in einem Dekompositionsschema zu der komplexen Aktion vorkommen. Das folgende Beispiel illustriert diese Vorgehensweise.

Beispiel: Sei A ein komplexe Aktion, wie in Abbildung 4.9 dargestellt, mit einer Dekomposition in die Schritte A_1 und A_2 und A_2 eine komplexe Aktion mit der Zerlegung in die Schritte A_3 und A_4. Die transitive Hülle von A ergibt sich dann als $TC(A) = \{A_1, A_2, A_3, A_4\}$.

[Yang, 1990] zeigt, daß bei komplexen Aktionen, auf die nicht-primitive Dekompostionsschemata anwendbar sind, das Kriterium aus Definition 4.9 dadurch erfüllt wird, daß die transitive Hülle dieser Aktion der Beschränkung aus Definition 4.10 genügt. Dann kann auch für diese Schemata bei einem Konflikt entsprechend früher Backtracking durchgeführt werden.

Anwendung des Kriteriums auf die Domänenbeschreibung

Domänen, die sich leicht mit dem Kriterium aus Definition 4.10 modellieren lassen, sind in der Regel solche, die um komplexe Aktionen durchzuführen, immer mehrere Vorbereitungs- und Nachbearbeitungsschritte benötigen. Die Vorbereitungsschritte dienen zur Schaffung von zusätzlichen Voraussetzungen für die Durchführung der Hauptaktion, während die Nachbearbeitungsschritte zusätzliche Nebeneffekte der Hauptaktion unterstützen. Domänen auf die

dies zutrifft, sind solche, deren komplexe Aktionen sich in immer primitivere Teilaktionen zerlegen lassen.

Durch die Restriktion auf die *main-subaction* in einem Schema ist es ausgeschlossen, daß die Vorbedingungen und Effekte der Hauptaktion von anderen Schritten des Schemas beeinflußt werden. Dadurch schränkt sich die Ausdrucksfähigkeit der Dekompositionschemata und Aktionen ein. Andererseits erlaubt das Kriterium eine Domänenbeschreibung vor dem Planungsbeginn effizient zu überprüfen und kann so helfen, falsche Lösungsansätze frühzeitig als solche zu erkennen.

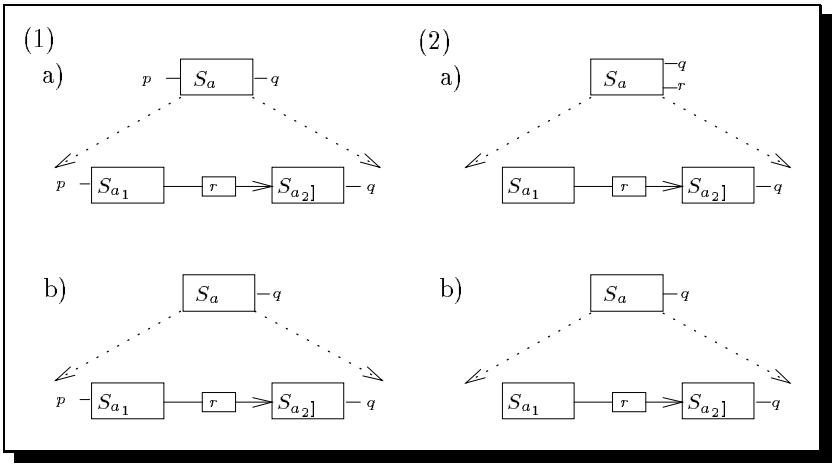


Abbildung 4.10: Beispiele für die Anwendung von Definition 4.10 auf Aktionen

In der Abbildung 4.10 sind einige Beispiele dargestellt, wie eine Aktionsbeschreibung geändert werden kann, damit die Dekompositionsschemata das Kriterium aus Definition 4.10 erfüllen. In Abbildung 4.10 (1) wird das Kriterium dadurch erfüllt, daß die Vorbedingung p der komplexen Aktion S_a entfernt wird, da sie keine Vorbedingung der *main-subaction* S_{a_2} ist. Analog verhält es sich im Beispiel 2. Hier wird der Effekt r der komplexen Aktion entfernt, um so sicherzustellen, daß S_{a_2} *main-subaction* wird (vgl. Abbildung 4.10 (2b)).

Durch dieses Vorgehen bei der Domänenbeschreibung wird ein weiteres Problem deutlich. Durch das Entfernen von Vorbedingungen und Effekten bei den komplexen Aktionen kann es vorkommen, daß Konflikte u.U. erst später erkannt, weil sie wegen der entfernten Effekte auf der abstrakten Ebenen nicht auftreten können. Betrachtet man das Beispiel aus Abbildung 4.5, so tritt hier kein unauflösbarer Konflikt auf, wenn man den Effekt $\neg q$ von $Step_b$ entfernen würde. Ein weiterer Nachteil ist, daß man durch die Einschränkungen bei der Definition der Dekompositionschemata einige Modellierungsfreiheiten bei der Domänendefiniton gegenüber DPOCL verliert.

4.5.4 Änderungen im DPOCL-Algorithmus

Die Vorschläge von Yang erfordern einige kleinere Veränderungen an dem Algorithmus aus Abbildung 4.4. Eine Änderung ergibt sich aus der Tatsache, daß komplexe Aktionen bei [Yang, 1990] komplett durch die Dekompositionsschemata ersetzt werden. Aus diesem Grund müssen alle Referenzen in dem aktuellen Plan $\mathcal{P}$, die auf den komplexen Schritt $S_{complex}$ deuten, durch die entsprechenden Schritte aus dem Dekompositionsschema ersetzt werden. Im einzelnen werden folgende Planungsschritte im Algorithmus geändert.

DPOCL($\mathcal{P}$)

1. **Termination:** ...
2. **Planverfeinerung:** ...
   (a) Vorbedingungen erfüllen: ...
   (b) Komplexe Schritte zerlegen:
      i. Aktions-Auswahl: ...
      ii. Dekompositions-Auswahl: ...
      Expandiere(D)
      Ersetze alle Referenzen aus $\mathcal{L}'_c$ die $S_{complex}$ enthalten durch die entprechenden Schritte aus S_D.
      ...
3. **Auflösung der Threats:** ...
   Schütze ...
   Ist dies nicht möglich, teste auf unauflösbaren Konflikt und Σ erfüllt Definition 4.10 dann führe Backtracking durch, ansonsten forciere die Dekomposition von S_{threat}, S_i und S_j .
   Backtracking-Punkt: ...
4. **Rekursiver Aufruf:** ...

Abbildung 4.11: Änderungen im DPOCL-Algorithmus

Abbildung 4.11 zeigt die Änderungen am DPOCL-Algorithmus aus Abbildung 4.4 für die erweiterte Interaktionsauflösung nach [Yang, 1990]. Die Veränderungen, die durch die Strategie von Yang notwendig sind, haben keine Auswirkung auf die Vollständigkeit und Korrektheit des Algorithmus. Allerdings geht die Systematik der Suche verloren. Der Verlust dieser Eigenschaft alleine stellt aber noch keinen keinen gravierenden Nachteil dar (vgl. Kapitel 6).

Kapitel 5

Realisierung der DPOCL-Konzepte im CAPLAN-System

In diesem Kapitel werden einige wichtige Aspekte bei der Realisierung des DPOCL-Ansatzes und seiner Erweiterungen aus Kapitel 4 im Planungsassistenten CAPLAN beschrieben. Der im CAPLAN-System implementierte SNLP-Planer stellt dabei schon einen Teil der Konzepte für die Einbindung des DPOCL-Ansatzes in das System zur Verfügung. Die Erweiterungen, die sich aus den DPOCL-Konzepten ergeben, lassen sich dabei gut mit vorhandenen SNLP-Konzepten modellieren.

5.1 Erweiterungen des Systems für DPOCL

Das in Kapitel 4 vorgestellte Verfahren untergliedert sich in zwei Hauptbestandteile, erstens das SNLP-Verfahren und als zweites einen hierarchischen Planungschritt. Die Konzepte des DPOCL-Ansatzes decken sich zum großen Teil mit denen des in Abschnitt 2.1 beschriebenen SNLP-Verfahrens. Daher können die in CAPLAN eingebundenen Konzepte, die den SNLP-Planer realisieren, für die Erweiterungen, die durch den DPOCL-Ansatz erforderlich sind, verwendet werden. Hierbei sind nur wenige Änderungen an den bestehenden Implementierungen notwendig. Die vom CAPLAN-System realisierten Erweiterungen des SNLP-Ansatzes (vgl. Abschnitt 2.2.4) können in die Erweiterungen durch DPOCL übernommen werden, da sie den hierarchischen Teil des DPOCL-Ansatzes nicht betreffen und für den SNLP-Planungsschritt eine Verbesserung darstellen.

Als notwendige Erweiterungen des CAPLAN-Systems zur Realisierung der neuen Konzepte lassen sich folgende Punkte identifizieren:

- Unterscheidung komplexer und primitiver Aktionen in der Domänenbeschreibung,
- Erweiterung der Domänenbeschreibung um Dekompositionsschemata,
- Erweiterung der Plandarstellung um verfeinerte Aktionen und ihre Ordnungen,
- Erweiterung der Interaktionsbehandlung um die Auflösung von Threats durch Verfeinerung.

Anhand dieser Aufstellung wird deutlich, daß die Erweiterungen lediglich den hierarchischen Teil des Ansatzes betreffen. Dies ist möglich, da die implementierten Konzepte des SNLP-

Ansatzes vollständig für die Einbindung von DPOCL genutzt werden können. Die Veränderungen betreffen lediglich die Unterscheidung von komplexen und primitiven Aktionen und die Integration der Dekompositionsschemata. Für das CAPLAN-Interface ist eine erweiterte Plandarstellung notwendig, um die neuen Konzepte in die Visualisierung einzubinden. Durch die in Abschnitt 4.5 beschriebene Veränderung des DPOCL-Ansatzes um die erweiterte Interaktionbehandlung, mußte auch eine Änderung im REDUX-Kern und im Verarbeitungsmechanismus des Systems vorgenommen werden. Die beschriebenen Änderungen realisieren dann den hierarchischen Planungsansatz im CAPLAN-System als generische Erweiterung der bestehenden Implementierung.

5.1.1 Eingrenzung der Erweiterungen im System

In Anlehnung an die schematische Übersicht aus Abbildung 2.5 gibt Abbildung 5.1 einen Überblick über die am CAPLAN-System vorgenommen Änderungen zur Realisierung der hierarchischen Aktionsdekomposition. Hierbei ergibt sich aus dem DPOCL-Ansatz, daß die vorgenommenen Erweiterungen das System nur partiell verändern bzw. erweitern.

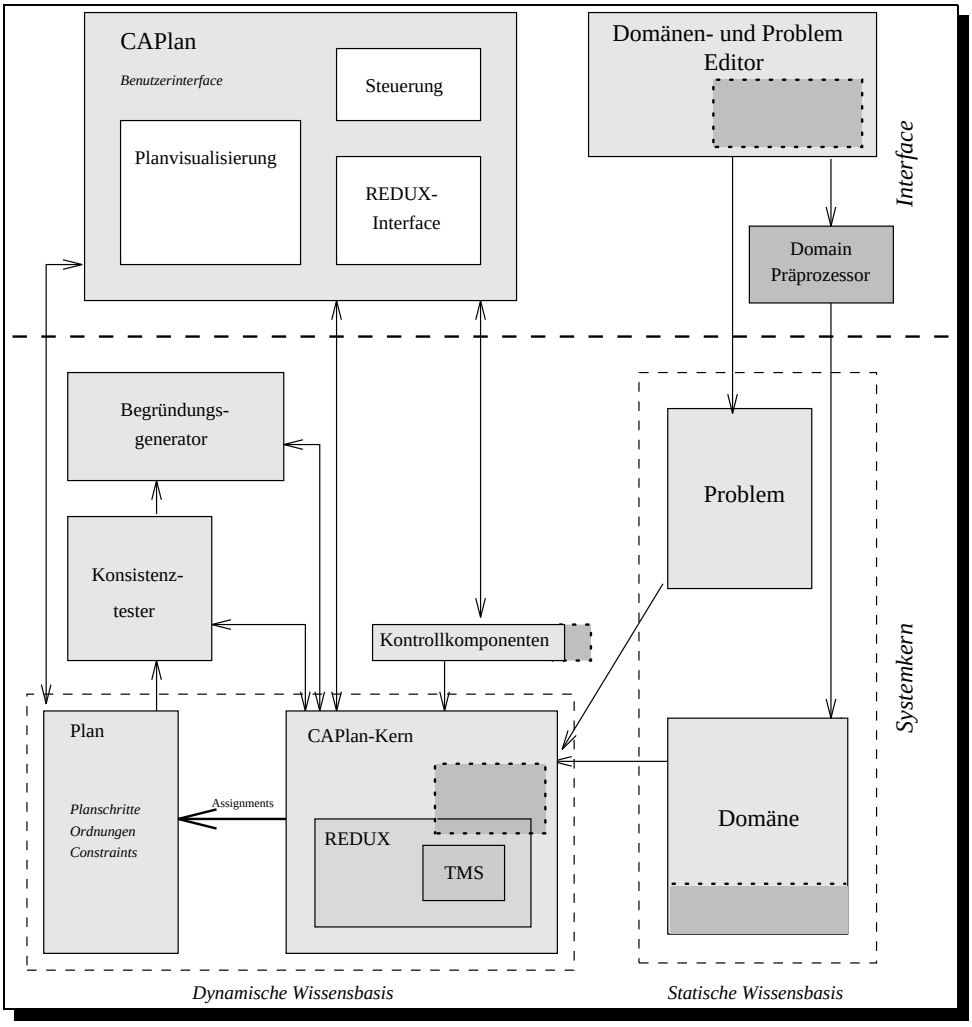


Abbildung 5.1: Das erweiterte CAPLAN-System

Im Detail lassen sich folgende Komponenten identifizieren, an denen Änderungen vorgenommen wurden:

Domänen-/Problemeditor:	Erweiterung für die Definition von Dekompositionsschemata und die Unterscheidung komplexen und primitiven Aktionen (Operatoren).
Domänenpräprozessor:	Er überprüft die Dekompositionsschemata der Domänenbeschreibung auf die Erfüllung des in Definiton 4.10 formulierten Kriteriums der *main-subaction*-Einschränkung.
Statische Wissensbasis:	Erweiterung der vom Planer benutzten Planungsdomäne um komplexe Operatoren und Dekompositionsschemata.
CAPLAN-**Kern:**	Einbindung der DPOCL-Konzepte basierend auf REDUX sowie eine Änderung der Abhängigkeitsstrukturen im REDUX-Kern für die erweiterte Interaktionsbehandlung.
Kontollkomponenten:	Realisierung des DPOCL-Algorithmus und Erweiterung der Backtracking-Steuerung bei der erweiterten Interaktionsbehandlung.
CAPLAN-**Benutzerinterface:**	Anpassung der Plandarstellung für hierarchische Aktionsdekomposition.

Aus der Aufstellung ist ersichtlich, daß die Änderung im System nur die DPOCL-spezifische Konzepte betreffen. Grundsätzliche Komponenten des Systems, wie Konsistenztester und Begründungsgenerator bleiben unverändert. Dies war auch gewünscht, bzw. eine wesentliche Vorraussetzung bei der Erweiterung des CAPLAN-Systems.

5.1.2 Erweiterung der Domänenbeschreibung

Das CAPLAN-System ist ein domänenunabhängiger Planer und benötigt daher eine Beschreibung der Planungswelt, in der ein Problem zu lösen ist (vgl. Abschnitt 2.2.3). Die im CAPLAN-System vorhandenen Möglichkeiten der Domänenbeschreibung enthalten dabei schon die grundsätzlichen Bestandteile, die auch von dem DPOCL-Ansatz benötigt werden. Die beiden notwendigen Erweiterungen, die vorgenommen werden müssen, betreffen einerseits die Definition von komplexen und primitiven Aktionen, andererseits die Bescheibung der Dekompositionsschemata (siehe Abschnitt 4.2).

Planschritte repräsentieren in SNLP einen Domänenoperator. Vergleicht man nun die Komponenten eines solchen Domänenoperators des CAPLAN-Systems mit der Bestandteilen eines Aktionsschemas aus Definition 4.1, so enthält der Domänenoperator schon alle Komponenten aus dieser Definition. Primitive und komplexe Planschritte werden bei DPOCL nur dadurch unterschieden, daß Dekompositionsschemata auf sie anwendbar sind oder nicht. Dieser Tatsache wurde Rechnung getragen, indem die Operatorspezifikation in der CAPLAN-Domänenbeschreibung um die Angabe der auf ihn anwendbaren Dekompositionsschemata erweitert wurde. Komplexe und primitive Operatoren werden dann, wie im DPOCL-Ansatz, durch die Möglichkeit der Dekomposition unterschieden.[1] Die Anwendbarkeitskriterien eines Dekom-

[1] Dieses Vorgehen vereinfacht dann auch die Modellierung einer Domäne, da primitive Operatoren noch nachträglich durch die Spezifikation von anwendbaren Dekompositionsschemata zu komplexen Operatoren werden können.

positionsschemas auf einen Domänenoperator ergeben sich anhand der in Abschnitt 4.2.2 beschriebenen Einschränkungen.

Die Dekompositionsschemata sind neu in der Domänenbeschreibung. Ihre Definition beinhaltet die folgenden Komponenten:

- Eine Menge von Schritten, dargestellt durch Domänenoperatoren.
- Eine Menge von Ordnungen zwischen den Schritten.
- Eine Menge von Caual Links zwischen den Schritten.
- Eine Menge von Anwendbarkeitsbedingungen (*phantom preconditions*).
- Eine Menge zusätzlicher Variablenbindungs-Constraints über die in den Schritten verwandten Variablen.

Die aufgeführten Bestandteile ergeben sich aus der Definition eines Dekompositionsschemas von Young (vgl. Definition 4.2). Die Auswahl der Operatoren als Planschritte ist dabei beliebig, d.h. es können sowohl komplexe als auch primitive Operatoren als Schritte in einem Schema auftreten. Es gilt dabei festzuhalten, daß es bei der Definition der Schemata nicht möglich ist ordnungsinkonsistente Schemata anzulegen.

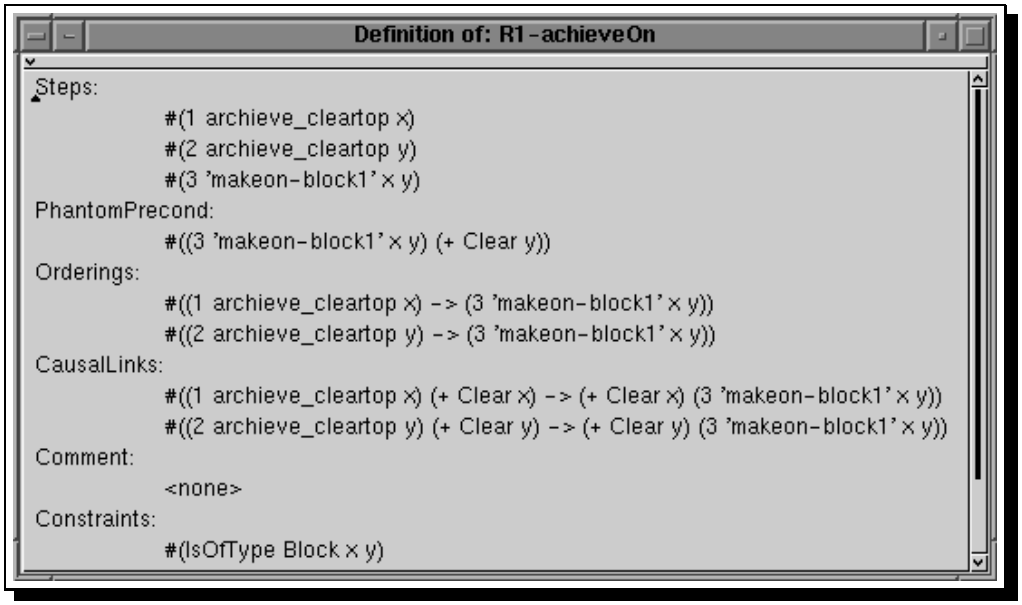


Abbildung 5.2: Beispiel eines Dekompositionsschemas der Domänenbeschreibung

Eine Erweiterung des DPOCL-Ansatzes stellt die Definiton der Anwendbarkeitsbedingungen dar. Im CAPLAN-System werden *phantom preconditions* im Sinne von Filterbedingungen behandelt. Hierbei liegt der Unterschied zu normalen Vorbedingungen darin, wie diese *phantom preconditions* erfüllt werden. Vorbedingungen dieser Art dürfen ausschließlich durch Schritte, die bereits im Plan enthalten sind, erfüllt werden. Ist dies nicht möglich, so muß das angewandte Schema wieder zurückgezogen werden. Dabei ist die Definition solcher Bedingungen unabhängig von der in der Operatordefinition. Weiterhin können die in den Schritten verwandten Variablen durch die Angabe zusätzlicher Variablenbindungs-Constraints eingschränkt werden. Diese Constraints schränken dann die Bindungsmöglichkeiten zusätzlich zu

den durch die Operatordefinitionen der Schritte vorgegebenen Variablenbindungs-Constraints ein. In Abbildung 5.2 ist eine solche Definition an einem Beispiel in der erweiterten CAPLAN-Domänenbeschreibung für das Dekompositionsschema aus Abbildung 4.1 dargestellt. Wird ein Dekompositionsschema einem Operator als mögliche Verfeinerung zugewiesen, so wird mittels des Domänenpräprozessors überprüft, ob das Dekompositionsschema das *main-subaction*-Kriterium aus Definition 4.10 erfüllt.

Damit realisieren die beschriebenen Erweiterungen der Domänenbeschreibung die vom DPOCL-Ansatz benötigten Anforderungen an die Definitionsmöglichkeiten einer Planungsdomäne.

Der Domänenpräprozessor

Der Domänenpräprozessor hat die Aufgabe, die den Domänenoperatoren zugewiesenen Dekompositionsschemata auf das Kriterium der *main-subaction* hin zu überprüfen (siehe Abschnitt 4.5.3). Jedes Schema, das dieses Kriterium erfüllt, wird entsprechend markiert.

Bei der Planung kann dann entschieden werden, welche Verfeinerungsvarianten einen potentiell unauflösbaren Konflikt auf einer konkreteren Planungsebene nicht auflösen können. Schemata, die entsprechend das Kriterium erfüllen, können dann frühzeitig ausgeschlossen werden, da sie garantiert nicht zu einer Lösung führen. Schemata die das Kriterium nicht erfüllen, müssen erst angewandt werden um festzustellen, ob sie den Konflikt auflösen.

5.1.3 Ordnungen in hierarchischen Plänen

Ordnungen zwischen Schritten in DPOCL-Plänen bestehen, im Gegensatz zu SNLP, immer über verschiedene Abstraktionsebenen hinweg. In der erweiterten Version des DPOCL-Ansatzes werden die komplexen Schritte bei der Zerlegung durch ein Dekompositionsschema aus dem Plan entfernt und die Referenzen der Causal Links auf die entsprechenden Schritte in dem Dekompositionsschema abgebildet. In der hier implementierten Variante ist dies nicht der Fall, da sich dieses Vorgehen nicht mit den vorhandenen REDUX-Konzepten ausdrücken läßt. Es wurde daher eine Ordnungsbeziehung zwischen einem komplexen Schritt und

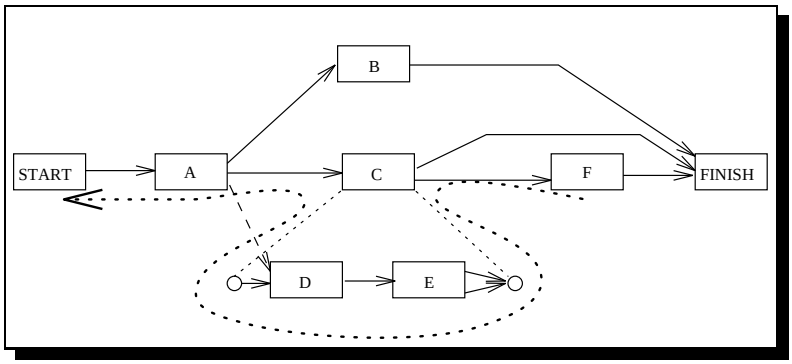


Abbildung 5.3: Beispiel eines Suchpfades in einem geordneten Plan

dem Dekompositonsschema implementiert, die ein entsprechendes Verhalten in Teilen realisiert. Beim Durchlaufen eines (partiell) geordneten Plans werden die Vorgängerschritte bzw. Nachfolgerschritte eines zerlegten komplexen Schrittes durch die künstlichen Schemaschritte s_f bzw. s_s erweitert. Dadurch werden die evtl. noch nicht mit den anderen Planschritten

angeordneten Schemaschritte mit durchlaufen.[2]

Beispiel: Abbildung 5.3 zeigt den Suchweg durch einen Plan mit komplexen Aktionen. Die Vorgänger des Schrittes F sind hier die Schritte C, E, D, A und $START$. Ebenso lassen sich die Nachfolger von A als C, E, D, F, und $FINISH$ identifizieren.

Ordnungen zwischen komplexen Schritten, die bei der Erweiterung von Yang bei der Zerlegung einer komplexen Aktion zerstört werden (vgl. Abschnitt 4.5.1), sind in der realisierten Version des Algorithmus nicht zugelassen, da sich dieses Verhalten mit den vorhanden REDUX-Konzepten nicht modellieren läßt (vgl. Abschnitt 5.3.1).

5.2 Abbildung der DPOCL-Konzepte auf REDUX

Die grundsätzliche Struktur des DPOCL-Ansatzes war durch den realisierten SNLP-Planer im CAPLAN-Kern schon gelegt. So finden die Abbildungen von Zielen für die Vorbedingungen, Planungsziele und Threats ebenso wie die dafür implementierten Operatoren Verwendung (siehe Abschnitt 2.2.4). Neue Strukturen sind ledigich für die Darstellung der komplexen Aktionen sowie für die Zerlegung derselben notwendig. Die zur Übertragung dieser Konzepte auf REDUX notwendigen Ziele und Operatoren werden anschließend beschrieben.

5.2.1 Realisierung in REDUX

Alle Backtracking-Punkte des Algorithmus aus Abbildung 4.4 müssen in REDUX durch Ziele repräsentiert werden, damit sie im REDUX-Abarbeitungsmechanismus als solche genutzt werden können. Unter diesen Aspekt fallen die Erzeugung von Causal Links, die Behandlung von Interaktionen und das Zerlegen komplexer Aktionen. Die im CAPLAN-Kern realisierten SNLP-Konzepte decken schon einen Teil der DPOCL-Konzepte ab (siehe Abschnitt 2.2.4).

Betrachtet man die nicht im CAPLAN-Kern repräsentierten Konzepte des DPOCL-Algorithmus aus Abbildung 4.4, so wird ersichtlich, daß lediglich der Schritt (2b) nicht durch entsprechende Ziele und Operatoren abgebildet ist. Es ergibt sich somit folgende Erweiterungen des CAPLAN-Kerns:

CAPDecomposeGoal: Dieses Ziel stellt die Notwendigkeit, einen komplexen Schritt zu verfeinern, dar. Es hat einen Parameter *step*, der den zu verfeinernden Schritt enthält.

CAPDecomposeOp: Dieser Operator ist ausschließlich auf Ziele vom Typ *CAPDecomposeGoal* im Sinne von REDUX anwendbar und realisiert den im Abbildung 4.4 Schritt (2b) der Dekomposition einer komplexen Aktion. Ein *CAPDecomposeOp* repräsentiert ein Dekompositionsschema aus der Domänenbeschreibung. Die Anwendung eines solchen Operators hat die folgenden Auswirkungen:

- Es werden die Planschritte, Ordnungen, Causal Links und Constraints des Dekompositionsschemas als Zuweisungen des Operators in den Plan eingeführt.

[2] Diese Vorgehensweise ist natürlich zu restriktiv, läßt sich aber aus implementierungstechnischen Gründen nicht anderes im CAPLAN-System realisieren.

- Neue Unterziele dieses Operators sind entsprechend die offenen Vorbedingungen der Schritte im Schema sowie *CAPDecomposeGoals* für die im Schema vorkommenden komplexen Schritte.

Die Anwendung eines *CAPDecomposeOp* entspricht daher im Algorithmus aus Abbildung 4.4 Schritt 2b, der Dekomposition einer komplexen Aktion.

CAPDomainOp: Der *CAPDomainOp* repräsentiert einen Domänenoperator und wird auf ein *CAPGoal* angewandt, um eine entsprechende Vorbedingung zu erfüllen. Er wurde erweitert, da bei DPOCL sowohl komplexe als auch primitive Domänenoperatoren vorkommen. Repräsentiert ein Domänenoperator eine komplexe Aktion, so leitet er zusätzlich als neues Unterziel ein *CAPDecomposeGoal* her. Diese Ziel signalisiert, daß es sich hier um eine komplexe Aktion handelt, die noch weiter zerlegt werden muß.

Abbildung 5.4 faßt die im erweiterten CAPLAN-Kern realisierten Ziele und die auf sie anwendbaren Operatoren noch einmal zusammen. Dabei sind die durch den SNLP-Planer implementierten Ziele und Operatoren mitaufgeführt. Die gegenüber SNLP neu hinzugekommenen Aspekte im CAPLAN-Kern sind in der Aufstellung hervorgehoben.

Operator	**anwendbar auf**	**Zuweisungen**	**neue Ziele**
DecomposeProblemOp	PlanningGoal	START FINISH	CAPPrecondGoal
CAPDomainOp	CAPPrecondGoal	Schritt Causal Link Constraints	CAPPrecondGoal CAPPhantomGoal **CAPDecomposeGoal**
CAPPhantomOp	CAPPrecondGoal CAPPhantomGoal	Causal Link Constraints	keine
CAPDecomposeOp	**CAPDecomposeGoal**	**Schritte** **Causal Links** **Ordnungen** **Constraints**	**CAPPrecondGoal** **CAPPhantomGoal** **CAPDecomposeGoal**
CAPProtectionOp	CAPProtectioGoal	Ordnungen Constraints	keine

Abbildung 5.4: Übersicht über die Ziele und Operatoren im CAPLAN-Kern

5.2.2 Auswahl der Dekompositionsschemata

Die Auswahl eines Dekompositionsschemas wird in REDUX dadurch realisiert, daß ein Operator aus der Konfliktmenge eines *CAPDecomposeGoals* ausgewählt wird. Wie in [Webers-

kirch, 1994] beschrieben, wurde der Mechanismus für inkonsistente Operatoren dahingehend vereinfacht, daß aus der Konfliktmenge eines Ziels nur konsistente Operatoren, d.h. Operatoren deren Zuweisung mit dem Plan konsistent sind, ausgewählt werden können. Wie in Abschnitt 5.1.2 vorgestellt, ist es nicht möglich ordnungsinkonsistente Schemata mit dem zur Verfügung gestellten Domäneneditor zu definieren. So ist bei der Aufnahme eines *CAPDecompseOp* in die Konfliktmenge eines *CAPDecomposeGoals* lediglich zu prüfen, ob die Menge der Variablenbindungs-Constraints konsistent mit der des Plans ist. Ist dies nicht der Fall wird der Operator direkt vom Konsistenztester und Begründungsgenerator mit der Rückzugsbegründung zurückgewiesen. Auch an dieser Stelle bleibt die Vorgehensweise des CAPlan-Systems unverändert. Der Rückzug von Dekompositionsschemata erfolgt dann nur noch, wenn festgestellt wird, daß eines seiner Unterziele nicht erfüllbar ist oder ein unauflösbarer Konflikt mit anderen Planschritten entsteht. Dies sind genau die Anforderungen des DPOCL-Algorithmus.

5.3 Änderungen in Redux

Eine Vorgabe bei der Einbindung der DPOCL-Konzepte in CAPlan war, die Redux-Konzepte so wenig wie möglich zu verändern. Bei der Abbildung der Konzepte für komplexe Aktionen und ihre Dekompositionsschemata war eine Änderung auch nicht notwendig. So paßt sich die Zerlegung von komplexen Aktionen durch ein Dekompositonsschema durch die Realisierung mittels des *CAPDecomposeGoals* in den normalen Redux-Mechanismus ein. Die Realisierung des SNLP-Planers konnte dabei genutzt werden, den SNLP-Planungsschritt aus dem DPOCL-Algoritmus zu realisieren. Dadurch konnten alle Komponenten des CAPlan-Systems, wie Konsistenztester und Begründungsgenerator, für die Erweiterungen genutzt werden. Eine Veränderung mußte lediglich für die Behandlung von Interaktionen vorgenommen werden.

5.3.1 Erweiterte Interaktionsbehandlung

Bei der Erzeugung und Abarbeitung von Zielen zum Schützen der Causal Links, den *ProtectionGoals*, mußte gegenüber SNLP eine Erweiterung vorgenommen werden. Dafür lagen im wesentlichen zwei Gründe vor:

1. Durch die Erweiterung des Algorithmus nach den Vorschlägen von Yang dürfen Ordnungen zwischen komplexen Operatoren nur temporär sein. Solche Abhängigkeiten lässen sich in Redux aber nicht modellieren, da eine Zuweisung, in diesem Fall eine Ordnung, immer nur von einem Operator abhängt und ihre Gültigkeit von der Gültigkeit des Operators ableitet. In der von Redux verwalteten Teilzielstruktur (vgl. Abschnitt 2.2.2) ist es nicht möglich, einen Operator einfach als ungültig zu kennzeichnen, da so die von dem Operator abhängige Unterzielstruktur mit ungültig werden würde. Daher müssen Interaktionen zwischen komplexen Schritten anders gehandhabt werden.
2. Um eine Verzahnung der Schritte aus den Dekompositionsschemata zu ermöglichen, ist es notwendig, die Auflösung von Interaktionen erst auf der primitiven Planungsebene vorzunehmen. Hier tritt auch wieder das Problem auf, daß Causal Links als Zuweisungen nicht von mehreren Operatoren abhängen können.

Da es aus den genannten Gründen nicht möglich ist, Causal Links bei der Verfeinerung von komplexen Aktionen direkt auf die Schritte des Dekompositionsschemas abzubilden, mußte

eine Lösung gefunden werden, die ein äquivalentes Verhalten auf anderem Wege realisiert. Dazu konnte der implementierte Mechanismus zur Bestimmung von Interaktionen zwischen Schritten und Causal Links im CAPLAN-System auf folgende Weise genutzt werden: Vorbedingungen von komplexen Aktionen sind in dem Dekompositionsschema mit Causal Links von den künstlichen Schemaschritten s_s zu dem entsprechenden Schritt im Schema vertreten. Dabei stehen diese Causal Links stellvertretend für jene Causal Links, die entsprechende Vorbedingungen des komplexen Schrittes garantiert. Eine Interaktion mit einem solchen Causal Link steht also stellvertretend für die Interaktion zwischen den Schritten auf der primitiven Planungsebene.

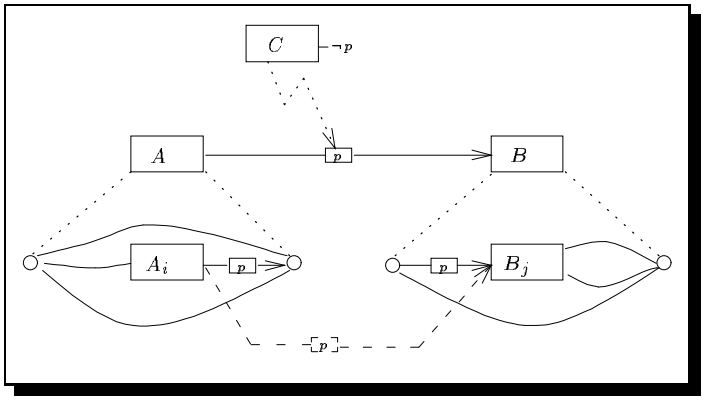


Abbildung 5.5: Interaktionen zwischen komplexen Schritten

In Abbildung 5.5 ist eine Interaktion zwischen einem Schritt C und einem Causal Link $A \xrightarrow{p} B$ dargestellt. Wird dieser Threat nach den SNLP-Strategien aufgelöst, so müßte eine Linearisierung $C \prec A \prec B$ oder $A \prec B \prec C$ entstehen. Eine solche Anordnung ist jedoch für komplexe Schritte nicht wünschenswert, da damit komplette Dekompositionsschemata gegeneinander angeordnet werden. Betrachtet man die Situation aus Abbildung 5.5 genauer, so stellt man fest, daß der Causal Links $A \xrightarrow{p} B$ stellvertretend für den Causal Link $A_i \xrightarrow{p} B_j$ steht. Die korrekte Anordnung der Schritte wäre also $C \prec A_i \prec B_j$ bzw. $A_i \prec B_j \prec C$.

Aus diesen Überlegungen heraus ergibt sich, daß eine Interaktion zwischen komplexen Schritten so lange zurückgestellt werden muß, bis die beteiligten Schritte auf der primitiven Planungsebene bestimmt werden können um dann die entsprechenden Ordnungen einzuführen.

Erweiterung der TMS-Struktur für Protection-Goals

Um die Anordnung der Schritte auf der primitiven Planebene zu gewährleisten, wurde die TMS-Rechtfertigung des *Valid-Goal*-Knotens bei *Protection-Goals* auf folgende Weise erweitert (siehe Abbildung 5.6): Die Notwendigkeit, einen Threat zu bearbeiten, tritt erst dann auf, wenn

1. Die Vorbedingung des übergeordneten komplexen Schrittes erfüllt und nicht blockiert ist.
2. Der Zielschritt des bedrohten Causal Links durch Dekompositionsschemata auf die primitive Planebene zerlegt und nicht blockiert ist.
3. Der Quellschritt des Causal Links durch Dekompositionsschemata auf die primitive Planebene zerlegt und nicht blockiert ist.

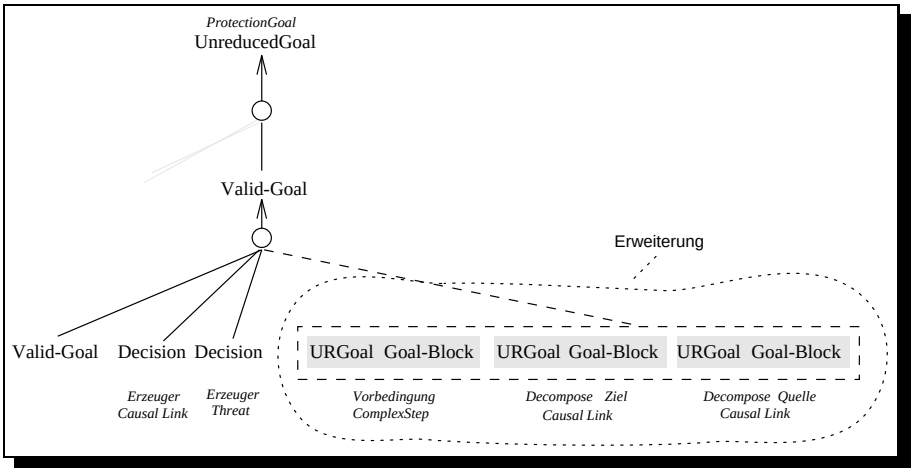


Abbildung 5.6: Erweiterte TMS-Struktur für die Gültigkeit eines ProtectionGoals

Die aufgezählten Fälle werden auf der TMS-Ebene dadurch ausgedrückt, daß jeweils der *Unreduced-Goal-* und der *Goal-Block*-Knoten der genannten Ziele in der Out-Liste des *Valid-Goal*-Knotens des *Protection-Goals* aufgenommen werden (vgl. Abbildung 5.6). Der Aufbau der Out-Liste kann sich je nach aufgetretener Interaktion änderen. Ist zum Beispiel der Zielschritt eines Causal Links ein primitiver Schritt, so kann für diesen Schritt natürlich kein *Unreduced-Goal*-Knoten des zugehörigen *Decompose-Goals* bestimmt werden.

Die Struktur der OUT-Liste ändert sich durch fortschreitende Verfeinerung der betroffenen Schritte, da die entsprechenden Knoten für weitere komplexe Schemaschritte aufgenommen werden müssen. Insgesamt handelt es sich hierbei um eine *dynamische* Abhängigkeitserweiterung in REDUX.[3]

5.4 Realisierung des Algorithmus

Die Realisierung des Algorithmus aus Abbildung 4.4 geschieht nun mittels der Reduktion der in Abschnitt 5.2 beschriebenen Ziele. Dabei bestimmt die Reihenfolge, in der die Ziele zur Reduktion gewählt werden, die Abfolge der Schritte im Algorithmus. Hierbei bildet die Auswahl und Abarbeitung der Task, die von REDUX für jedes offene Ziel erzeugt wird, die Ausführung eines Schrittes im Algorithmus. Durch die Teilung des Algorithmus in einen SNLP-Schritt und einen hierarchischen Schritt konnte die im CAPLAN-System implementierte SNLP-Kontrollkomponente für die Realisierug des Algorithmus herangezogen werden. Die jeweils aktive Kontrollkomponente bestimmt den nächsten zu bearbeitenden Task.

5.4.1 Abarbeitung eines Tasks

REDUX erzeugt für nicht reduzierte Ziele eine *Unreduced-Goal-Task*. Bei Zielen, für die kein Operator mehr anwendbar ist, wird eine *Goal-Block-Task* vom System erzeugt. Die Realisierung des Algorithmus geschieht durch die Bearbeitung dieser Tasks. Dabei stellt die Abarbeitung einer *Unreduced-Goal-Task* für die in Abschnitt 5.2.1 beschriebenen Ziele die

[3] REDUX sieht solche dynamischen Abhängigkeitsstrukturen eigentlich nicht vor. Ein einmal angelegtes Abhängigkeitsnetz bleibt normalerweise in seinen Bestandteilen konstant.

Realisierung der im Algorithmus beschriebenen Planungsschritte dar, während mit der Bearbeitung einer *Goal-Block-Task* das Backtracking realisiert wird. Daher ist die Reihenfolge der Auswahl dieser Tasks an dieser Stelle von Bedeutung. Diesen Auswahlmechanismus leistet die Kontrollkomponente.

Im einzelnen geschieht bei der Bearbeitung dieser Tasks folgendes:

Berabeitung eines *Unreduced-Goal-Task*

Durch die Bearbeitung eines solchen Tasks wird das zugehörige Ziel reduziert. Für offene Vorbedingungen und Protection-Goals werden die im CAPLAN-System implementierten Mechanismen für den SNLP-Planer verwendet. Im Überblick werden bei der Bearbeitung einer solchen Task immer folgende Schritte durchgeführt:

1. Berechnung der Konfliktmenge
2. Konsistenztest der Konfliktmenge
3. Auswahl eines Operators
4. Anwendung des Operators
5. Bestimmung neuer Threats, wobei gleichzeitig überprüft wird ob, bestehende Threats jetzt inaktiv sind.

Nur bei Punkt 5 wurde der Mechanismus in CAPLAN dahingehend erweitert, daß zusätzlich überprüft wird, ob ein unauflösbarer Konflikt entstanden ist, bei dem entschieden werden kann, ob gegebenenfalls schon Backtracking eingeleitet werden muß.

Bearbeitung eines *Goal-Block-Task*

Dieser Task wird von REDUX immer dann erzeugt, wenn auf ein Ziel keine Operatoren mehr anwendbar sind. Da im CAPLAN-Kern die Operatoren vor ihrer Anwendung auf Konsistenz gestestet werden, ist es der einzige Task, der neben einem *Unreduced-Goal-Task* erzeugt wird. Er signalisiert die Notwendigkeit Backtracking durchzuführen. Bei der Bearbeitung einer solchen Task müssen zunächst die betroffenen Entscheidungen bestimmt werden. Bei normalen Zielen bleibt der Mechanismus unverändert. Durch das Auftreten eines unauflösbaren Konflikts und der Zurückweisung von Dekompositionsschemata kann es jedoch vorkommen, daß eine Menge abhängiger Entscheidungen zurückgezogen werden muß.

5.4.2 Die neue Kontrollkomponente DPOCL

Die Kontrollkomponente setzt den Algorithmus aus Abbildung 4.4 durch die Steuerung der Auswahl der von REDUX erzeugten Tasks um. Hierbei konnte die implementierte Kontrollkomponente SNLP als Ausgangspunkt für die neue Kontrollkomponente DPOCL genutzt werden. Das Konzept der austauschbaren Kontrollkomponenten, wie es in CAPLAN realisiert ist erleichterte dabei die Einbindung der neuen Kontrollstrategie.

Wie in [Weberskirch, 1995] dargestellt, verwirklicht eine Kontrollkomponente die Hauptauswahlpunkte im Algorithmus. Der SNLP- und DPOCL-Algorithmus unterscheiden sich dabei

nur an einer Stelle: Bei DPOCL wird zusätzlich die Zerlegung einer komplexen Aktion ausgeführt. Prinzipiell arbeiten die beiden Algorithmen gleich, d.h. wie bei SNLP werden immer zuerst die auftretenden Interaktionen behandelt und erst dann wieder eine offene Vorbedingung erfüllt bzw. eine komplexe Aktion zerlegt. Da die Zerlegung einer Aktion durch das *CAPDecomposeGoal* angezeigt wird, war zur Umwandlung einer SNLP-Kontrollkomponente zu einer DPOCL-Kontollkomponente eine Erweiterung der Zielauswahl für Ziele dieser Art notwendig. Sie werden gemäß dem Algorithmus gleichwertig mit den Zielen vom Typ CAP-Goal (vgl. Abschnitt 2.2.4) bei der Auswahl behandelt, was letztlich den Algorithmus realisiert.[4] Daneben wurde in die DPOCL-Kontrollkomponente noch eine Heuristik eingebunden, welche komplexe Domänenoperatoren primitiven vorzieht, da erwartet werden kann, daß komplexe Operatoren einen Aspekt der Problembeschreibung besser lösen als primitive dies könnten.

Im Fall von Backtracking wurde hier Erweiterung des chronologischen Backtrackings realisiert, da bei einem unauflösbaren Konflikt auf einer höheren Planungsebene unter Umständen mehrere Entscheidungen zurückgezogen werden müssen.

[4] Daneben wurden, aufbauend auf dieser Kontollkomponente eine Reihe weiterer Auswahlstrategien implementiert, die die Zielauswahl steuern, um eine bessere Effizienz des Planungsprozesses zu erreichen (vgl. Kapitel 6).

Kapitel 6

Diskussion des DPOCL-Algorithmus

In den vorangegangen Kapiteln wurde die Erweiterung des CAPLAN-System um hierarchische Aktionsdekomposition mittels des DPOCL-Ansatzes beschrieben. Um eine Einschätzung des neuen Verfahrens zu bekommen, wurden eine Reihe von Tests mit verschiedenen Planungsdomänen und Problemen durchgeführt. Hierbei sollte bewußt ein Vergleich mit dem bisherigen nicht-hierarchischen Basisverfahren SNLP angestellt werden, da der DPOCL-Ansatz auf diesem basiert. Daneben waren auch die Auswirkungen des Verfahrens bei der interaktiven Plangenerierung von Interesse.

6.1 Analyse des DPOCL-Ansatzes

Um einen Eindurck von dem in Kapitel 4 beschriebenen Verfahren und seiner Erweiterung zu bekommen, wurde das Basisverfahren DPOCL und einige Varianten davon an verschiedenen Planungsdomänen und Problemen getestet. Die Betrachteten Domänen waren zum einen zwei Varianten der Blocksworld-Domänen und die komplexe Workpiece-Domäne zur Fertigung von rotationssymmetrischen Drehteilen. Die dabei gewonnen Erkenntnisse sollen die Vor- und Nachteile des DPOCL-Ansatzes aufzeigen.

6.1.1 Eingesetzte Kontrollstrategien

Der in Kapitel 4 beschriebene Algorithmus diente als Basis für eine Reihe von Varianten, die im CAPLAN-System mit Hilfe verschiedenener Kontrollkomponenten implementiert wurden. Dabei standen zwei Strategien im Vordergrund: Zum einen sollte der reine DPOCL-Ansatz aus Abschnitt 4.4 getestet werden, zum anderen sollten die Erweiterungen, die sich durch die Vorschläge von Yang ergeben (vgl. Abschnitt 4.5) auf ihren eventuellen Effizienzgewinn dagegengestellt werden. Da sich die beiden Verfahren nur durch die Interaktionsauflösung unterscheiden, wurden noch eine Reihe von Varianten des DPOCL-Verfahrens als Kontrollkomponenten in CAPLAN implementiert, die sich in der Reihenfolge, in der die zu bearbeitenden Tasks von REDUX ausgewählt werden, unterscheiden. Dies resultiert aus der Tatsache, daß im DPOCL-Ansatz keine Vorschriften bzgl. Zielerfüllung und Dekomposition von Schritten bestehen, es müssen lediglich die Threats nach jedem Planungschritt bestimmt und aufgelöst werden. Im Detail wurden die folgenden Kontrollstrategien verwendet:

DPOCL: Das DPOCL-Basisverfahren

DPOCL-E: Eine DPOCL-Variante, die erst alle offenen Vorbedingungen auf einer Planungsebene zu erfüllen versucht (Breitensuche).

DPOCL-D: Eine weitere Variante des DPOCL-Algorithmus, die versucht erst alle komplexen Aktionen vollständig zu bearbeiten (Tiefensuche).

SNLP+: Eine Kontrollkomponente, die das in Abschnitt 2.1 beschriebene Basisverfahren SNLP mit einigen Verbesserungen realisiert.

DSep: Eine Variante des SNLP-Verfahrens, welches die Auflösung von Interaktionen mittels Separation so lange zurückstellt, bis keine anderen Ziele mehr zu erfüllen sind. Diese Kontrollstrategie zeigt im Mittel ein sehr effizientes Planungsverhalten.

Alle Varianten wurden sowohl mit der von Young beschriebenen Interaktionsbehandlung, als auch mit der Erweiterten von Yang getestet. Eine weitere Variante der Tests war die Aufhebung der Systematikeigenschaft des DPOCL-Ansatz durch die Nichtbeachtung von positiven Interaktionen. Die ausgewählten SNLP-Kontrollkomponenten *SNLP+* und *DSep* dienten dazu, Vergleichsergebnisse für den SNLP-Planer zu erhalten.

6.1.2 Verwendete Beispieldomänen und Probleme

Zum Test in Abschnitt 6.1.1 beschriebenen Kontrollstrategien wurden verschiedene Planungsdomänen verwendet. Um hier einen Vergleich mit dem bisherigen Basisverfahren SNLP anstellen zu können, wurden einige Domänen, die für SNLP entwickelt wurden, um komplexe Aktionen erweitert. Im einzelnen wurden folgende Domänen für die Testdurchläufe verwandt:

Blocksworld: Hierbei handelt es sich um eine Blocksworld-Domäne, die aus zwei primitiven Aktionen, *Puton* und *Cleartop* besteht. Dies war ursprünglich eine für den SNLP-Planer entwickelte Domäne, die um einige komplexe Aktionen erweitert wurde, die spezielle Teilprobleme lösen, wie beispielsweise die Bildung von Türmen aus mehreren Blöcken.

Yang-Blocksworld: In [Yang, 1990] wurde eine Planungsdomäne aus der Blocksworld angegeben, bei der die Abwärtslinearisierbarkeit (vgl. Definition 4.9) der Dekompositionsschemata gegeben ist.

Workpiece-Domäne: Diese Planungsdomäne dient zur Arbeitsplanerstellung bei der Fertigung von rotationssysmmetrischen Drehteilen. Sie wurde u.a. deshalb ausgewählt, da sie eine Vielzahl primitiver Aktionen enthält und so die Erwartung groß war, durch komplexe Aktionen bessere Plaungsergebnisse zu erhalten. Ein weiterer Aspekt war auch, daß sich diese Planungsdomäne aufgrund ihrer Struktur anbot, sie hierarchisch zu strukturieren.

Die aufgeführten Domänen sind in Anhang A ausführlicher beschrieben. Durch die Verwendung der Blocksworld-Planungsdomäne (vgl. Anhang A.1.1), die schon für den SNLP-Planer erstellt wurde, konnte ein direkter Vergleich mit SNLP durchgeführt werden, da es für diese

Domäne auch entsprechende Planungsprobleme gab, die auch für den DPOCL-Ansatz verwendet werden konnten (vgl. Abschnitt 4.1). Bei der in [Yang, 1990] beschriebenen Domäne fiel allerdings auf, daß sie unvollständig war. Die Domäne enthielt lediglich Operatoren, die es erlauben, Klötzchen von einem zu einem anderen Klötzchen zu bewegen, oder sie auf den Tisch zu stellen. Es fehlte ein Operator, um sie vom Tisch wieder aufnehmen zu können. Damit war eine sinnvolle Problemlösung für Probleme aus der Blocksworld nicht möglich. Die hier verwendete Testdomäne korrigiert diesen Fehler (vgl. Anhang A.1.2). Sie wurde gewählt, da sie das Kriterium der Abwärtslinearisierbarkeit (vgl. Abschnitt 4.5.2) erfüllt.

Bei der Workpiece-Planungsdomäne war vor allem der Aspekt der verbesserten Domänenbeschreibung interessant. Hier wurde unter dem Gesichtspunkt eines als Planungsassistenten arbeitenden Systems erwartet, daß durch die Definition von komplexen Aktionen die interaktive Plangenerierung erleichtet wird (vgl. Anhang A.1.3).

Betrachtete Problemstellungen

Die beschriebenen Planungsdomänen wurden an verschiedenen Problemstellungen getestet. Die Auswahl der Probleme reicht dabei von sehr einfach zu lösenden bis zu sehr aufwendig lösbaren Problemen. Abschnitt A.2.1 gibt eine detaillierte Übersicht über die zum Test herangezogenen Beispielprobleme. Die Probleme sind nur exemplarisch zu sehen und wurden gewählt um die Schwächen und Stärken des DPOCL-Verfahrens darzustellen.

6.1.3 Ergebnisse der Tests

Der DPOCL-Ansatz wurde auf verschiedene Kriterien hin untersucht. Wichtige Aspekte dabei waren das Verhalten als autonomes Planungssystem und der Vorteil bei der interaktiven Plangenerierung. Eine ausführliche Übersicht der mit den Kontrollstrategien aus Abschnitt 6.1.1 durchgeführten Tests gibt Abschnitt A.2.

Die Tests zeigen einen gravierenden Nachteil des Verfahrens: Durch die Erweiterung der Domänenbeschreibung kommt es zu einer drastischen Vergrößerung des Suchraums bei der Plangenerierung, so daß ein autonomes Planen des Systems schon bei einfachen Domänen und Problemen fast aussichtslos wird. Insgesamt ergibt sich bei der autonomen Planung kein Vorteil gegenüber SNLP. Das ist u.a. darauf zurückzuführen, daß die Operatorbeschreibung keinerlei Hinweise gibt, in welcher Situation die Anwendung des Operators sinnvoll ist. Insbesondere bei der systematischen Suche im Planungsraum werden selbst einfache Probleme in der Blocksworld nicht mehr akzeptabel gelöst. Dieses Verhalten ist auch bei komplizierteren Problemen zu beobachten. Die eingesetzte Kontrollkomponente spielt dabei nur eine untergeordnete Rolle, daher kann auch keine Aussage darüber gemacht werden, welche Strategie zu bevorzugen ist: Erfüllen von Vorbedingungen oder Zerlegung komplexer Aktionen. Gibt man jedoch einige komplexe Aktionen vor, so ist die Strategie mit der Tiefensuche im Mittel etwas besser als die anderen. Dies ist aber nur eine subjektive Einschätzung, da dieses Verhalten auch sehr stark von der Planungsdomäne abhängt.

Die in [Yang, 1990] beschriebene erweiterte Interaktionsbehandlung brachte dagegen keinen signifikanten Effizienzgewinn. Dieses Verhalten ist auch auf die Implementierung zurückzuführen (vgl. Abschnitt 5.3.1), da durch die Verzögerung der Auflösung von Interaktionen mögliche Backtracking-Punkte erst spät erreicht werden. Aber auch im Zusammenhang mit der Modellierung der Workpiece-Domäne wurde die Beobachtung gemacht, daß die syntaktischen Einschränkungen, die sich aus Definition 4.10 ergeben, die Ausdrucksfähigkeit der

Schemata sehr einschränken. Damit ergibt sich insgesamt eine sehr unnatürliche Domänenbeschreibung, was den Hauptgrund darstellte, die Workpiece-Domäne nicht nach dem von Yang beschriebenen Kriterien zu erstellen.

Die Systematik bei der Suche hat dagegen einen entscheidenden Einfluß auf die Planungseffizienz. Es zeigt sich, daß durch die Nichtbeachtung der positiven Interaktionen eine nicht unerhebliche Verbesserung in der Plangenerierung erfolgt. Dies ist eine Eigenschaft, die auch schon bei dem SNLP-Ansatz zu beobachten ist.

Betrachtet man den DPOCL-Ansatz unter dem Aspekt des als Planungsassistenten fungierenden Systems, so erscheint er in einem anderen Licht. Die Tests zeigen, gibt man einige komplexe Operatoren vor, die für die Problemlösung sinnvoll sind, so kann das System dann anschließend als autonomer Planer eine Lösung recht effizient finden. Dies zeigte sich insbesondere bei der Workpiece-Domäne. Auch in der Blocksworld-Domäne wird deutlich, daß durch die Vorauswahl entsprechender Aktionen eine Lösung recht schnell und ohne großen Backtracking-Aufwand gefunden wird.

Einfluß der Domänenmodellierung

Eine der Beobachtung mit den verwendteten Domänen war, daß unvollständig definierte Domänenoperatoren, d.h. Operatoren bei denen nicht alle Vorbedingungen und Effekte modelliert waren, dazu führen, daß der DPOCL-Algorithmus keine Lösung findet. Dies ist darin begründet, daß durch die fehlenden Prädikate nicht alle Interaktionen entstehen, die eigentlich notwendig wären, um eine korrekte Lösung zu finden. Hier unterscheidet sich der DPOCL-Ansatz nicht von SNLP.

Eine weitere Beobachtung betrifft die Anzahl der auf eine offene Vorbedingung anwendbaren Operatoren. Da bei DPOCL immer eine Reihe von primitiven Operatoren vorhanden sein muß, um die Vorbedingungen erfüllen zu können, und Dekompositionschemata wieder aus diesen Schritten bestehen, haben komplexe Operatoren immer auch solche Effekte, die es erlauben eine entsprechende Vorbedingung zu erfüllen. Daher ergeben sich meist eine Anzahl von Wahlmöglichkeiten wie eine Vorbedingung erfüllt werden kann.

Auf der anderen Seite ist mit der verbesserten Domänenbeschreibung die Möglichkeit vorhanden zu jedem Problem einen passenden Operator zu definieren, der die schwierigen Entscheidungsstellen des Planungsproblems effizient löst. So kann jedes Planungsproblem, daß in der Planungsdomäne lösbar ist, durch entsprechende Definition und Auswahl eines Domänenoperators, mit einem linearen Aufwand berechnet werden.

6.2 Bewertung des DPOCL-Ansatz

Die in Abschnitt 6.1 durchgeführten Tests haben eine Schwäche des DPOCL-Verfahrens deutlich gezeigt: Die Anzahl der Operatoren, die verwendet werden können, um ein Ziel zu erfüllen ist sehr entscheidend für die Effizienz des Planungsprozesses. Trotz des im allgemeinen ineffizienten Verhaltens des Ansatzes lassen sich aber auch einige positive Aspekte sehen.

6.2.1 Vorteile des Verfahrens

Ein wesentlicher Vorteil des DPOCL-Ansatzes liegt in der erweiterten Domänendefinitionsmöglichkeit. Dies ist besonders unter dem Aspekt eines als Planungsassistent ausgelegten

Systems, wie CAPLAN es ist, eine wichtige Eigenschaft. Primitive Aktionen haben meist den Nachteil, daß Laien in der Regel mit der sinnvollen Anwendung solcher Aktionen Probleme haben oder das zur Lösung eines Teilproblems immer mehere primitive Aktionen in einer bestimmten Reihenfolge gekoppelt werden müssen. Durch die Möglichkeit, komplexe Aktionen zu definieren, die eine Reihe von primitiven Schritten in einer sinnvollen Weise in den Problemlöseprozeß einführen, fällt es dem Benutzer leichter eine sinnvolle Auswahl von (komplexen) Planschritten zu treffen, die dann vom System zu Ende geplant werden können. Dabei sind besonders die in Abschnitt 4.2.2 beschriebenen Freiheiten bei der Definition von Dekompositionsschemata von Nutzen.

Ein weiterer Vorteil des DPOCL-Ansatzes ist es, daß er sich ohne weiteres in einen bestehenden SNLP-Planer integrieren läßt und es weiterhin möglich ist den SNLP-Planer alleine zu nutzen.

6.2.2 Nachteile des Verfahrens

Der Vorteil der erweiterten Domänendefinition ist für einen Planungssystem unter dem Gesichtspunkt einer autonomen Plangenerierung ein gravierender Nachteil. Wie in Abschnitt 6.1.3 beschrieben, ist ein autonomens Planen fast nicht sinnvoll, da in der Regel zu viele Alternativen ausprobiert werden müssen, bis ein sinnvoller Operator ausgewählt ist.

Ein weiterer Nachteil des von Young beschriebenen Verfahrens liegt darin, daß er sich mit der Annährung an SNLP etwas von dem HTN-Ansatz (vgl. Kapitel 3.1) entfernt. Vor allem die Problemspezifikation von DPOCL, die der von SNLP entspricht und keine Ordnungsbeziehungen zwischen den einzelnen Zielen kennt ist hier zu nennen.[1] Auch werden in einer Problemspezifikation von DPOCL nur eine Art der Tasks verwandt (vgl. Abschnitt 3.2.3). Das zeigen auch die Tests mit den Domänen. Werden einige komplexe Planschritte vorgegeben, so entsteht ein HTN-Problem (vgl. Definition 3.2). Die Lösung eines solchen Problems ist mit dem DPOCL-Ansatz aber in der Regel sehr effizient. Daneben nutzen die Aktionschemata des DPOCL-Ansatzes nicht alle Konzepte der HTN-Planungsoperatoren, da die Vorbedingungen von Aktionsschemata nur durch Prädikatssymbole beschrieben werden können und nicht, wie bei HTN durch beliebige Tasks.

6.2.3 Bewertung von DPOCL und Vergleich mit SNLP

Durch seinen Aufbau auf den SNLP-Ansatz erhält der DPOCL-Ansatz alle Probleme und Vorteile, die auch der SNLP-Ansatz hat. Dabei ist die einzige positive Erweiterung, die besseren Domänenbeschreibungsmöglichkeit. Ansonsten kann kein signifikanter Unterschied zum Verhalten des SNLP-basierten Planers festgestellt werden, außer daß die erweiterten Wahlmöglichkeiten bei der Zielerfüllung zu mehr Backtracking führen.

Durch die Vielzahl der anwendbaren Domänenoperatoren ist es notwendig, bei der Operatorauswahl geeignete Kriterien zu besitzen, wann ein Domänenoperator einem anderen vorgezogen wird, um eine Vorbedingung zu erfüllen. Auch bei der Auswahl der zur Verfügung stehenden Verfeinerungsvarianten einer komplexen Aktion sind gute Heuristiken gefordert. [Yang, 1990] schlägt vor, bei der Auswahl von Dekompositionsschemata die Anzahl der von ihm ausgelösten Interaktionen zu einem solchen Kriterium zu machen, und erst solche Schemata zu wählen, die möglichst wenige Interaktionen in dem aktuellen Plan auslösen. Hier müssen geeignete Heuristiken gefunden werden, die eine effiziente Auswahl gewährleisten.

[1] CAPLAN kompensiert diesen Nachteil mit seiner erweiterten Problembeschreibung (vgl. Abschnitt 2.2.3).

Gerade hier wird der Ruf nach besseren Verfahren, zur Kontrolle des Planungsvorgangs noch lauter als gewöhnlich bei SNLP. Eine Interessante offene Frage bleibt, ob sich durch den Einsatz von Lernverfahren vielleicht leistungsfähigere Kontrollkomponenten entwickeln lassen.

6.3 Die Rolle von REDUX bei der Implementierung

Der Aufbau auf REDUX brachte eine Reihe von Vor- und Nachteilen bei der Realisierung des DPOCL-Ansatzes mit sich. Zu den Vorteilen zählt die durch REDUX bereitgestellten Konzepte für die Planung, die es einfach gestatten, die vielfältigen Abhängigkeiten, die während eines Planungsprozesses entstehen, gut zu modellieren. Daneben zeigt sich, daß manche Zusammenhänge, die für das hierarchische Planen notwendig sind, sich nicht mit den vorhandenen Konzepten ausdrücken lassen.

6.3.1 Vorteile von REDUX

Ein wesentlicher Vorteil von REDUX ist die Verwaltung einer abhängigen Teilzielhierarchie. Sie ist unabdingbar, um die bei unauflösbaren Konfliken abhängigen Ziele wieder zurückziehen zu können. Daneben erlauben es die Mechanismen zum Rückzug von Entscheidungen eine natürliche und einfache Abbildung der planungsrelevanten Aspekte des DPOCL-Ansatzes auf REDUX.

6.3.2 Einschränkungen in REDUX bei der Realisierung

In Abschnitt 5.3.1 wurde schon ein Problem angesprochen, das im Zusammenhang mit der Realisierung des DPOCL-Ansatzes mit REDUX auftritt. Einige Konzepte, wie die Abbildung von Causal Links auf die neuen Schritte des Dekompositionsschemas bei der Zerlegung einer komplexen Aktion, lassen sich mit REDUX nicht modellieren. Das betrifft vor allem die Abhängigkeitstrukturen von Zielen und Zuweisungen. Daher führt es insgesamt zu einer zu restriktiven Implementierung der DPOCL-Konzepte. Für die Realisierung des DPOCL-Ansatzes ist es notwendig, daß die Gültigkeit von Zielen und Zuweisungen auch von mehreren Operatoren bedingt werden kann. Dadurch kann bei der Dekomposition von komplexen Aktionen die Ordnungsstruktur der Aktion auf das Schema übertragen werden, und die Anwendung des Dekompositionsoperators sorgt dafür, daß der komplexe Schritt aus dem Plan herausfällt, so lange diese Zerlegung existiert.

Vorschläge zur möglichen Erweiterung von REDUX

Um die Abhängigkeiten in REDUX modellieren zu können, muß die Begründungstruktur, insbesondere für Zuweisungen, erweitert werden. Dazu kann die in Abbildung 6.1 dargestellte Struktur dienen.

Abbildung 6.1 stellt eine mögliche Erweiterung für die Abhängigkeitstruktur von Zuweisungen eines Operators dar (vgl. Abbildung 2.10). Zuweisungen eines komplexen Schrittes müssen bei dessen Zerlegung durch ein Dekompositionsschema aus dem Plan verschwinden, d.h. solange ungültig werden, wie die Zerlegung existiert. Dieser Fall wird in der Abbildung dadurch ausgedrückt, daß die Gültigkeit einer Zuweisung von dem Operator abhängt, der den komplexen Schritt einführte sowie von der Reduzierung des komplexen Schrittes, der durch die Anwendung des Operators in den Plan aufgenommen wurde. Wird der Schritt durch Anwendung

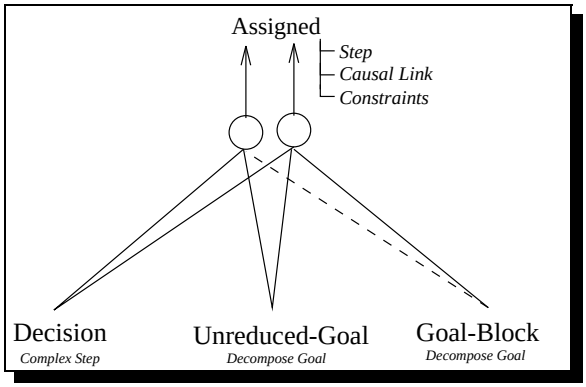


Abbildung 6.1: Erweiterung der Abhängigkeiten in REDUX

eines Dekompositionsschemas zerlegt, so wechselt das Label des *Unreduced-Goal-Knotens* des zugehörigen *Decompose-Goals* auf OUT. Damit werden die Zuweisungen des Domänenoperators automatisch ungültig, so lange der Schritt zerlegt bleibt. Der *Goal-Block-Knoten* in der Abbildung stellt sicher, daß bei einer Blockierung der Zerlegung die Zuweisung wieder gültig werden.

Eine weitere offene Frage bleibt allerdings, ob und wie die Abhängigkeitsstruktur eines Zieles, d.h. einer offenen Vorbedingung eines komplexen Schrittes, erweitert werden muß. Diese Vorbedingungen kommen auch wieder in der Zerlegung vor und dürfen daher nicht nocheimal in die Teilzielhierarchie aufgenommen werden, da sie evtl. schon durch einen Schritt erfüllt sind. Ein weiters Problem stellen die *Protections* dar. Auch hier muß eine geeignete Struktur gefunden werden, welche die Gültigkeit von der Zerlegung der komplexen Schritte abhängig macht.

Ein Nachteil dieser Art der Realisierung ist ein Anwachsen der TMS-Struktur, da einige neue Abhängigkeiten mitzuverwalten sind. Da die Größe der verwalteten TMS-Struktur einen Einfluß auf die Effizienz des Planungsprozesses hat, muß hier abgewogen werden, was mehr Vorteile bringt, eine bessere Modellierung der hierarchischen Konzepte oder eine effizientere Verwaltung der Abhängigkeitsstruktur.

6.4 Zusammenfassung der Ergebnisse der Arbeit

Der hier untersuchte DPOCL-Ansatz weist in der praktischen Anwendung einige Nachteile auf. Diese sind unter anderem darin begründet, daß durch die Erweiterung der Domänenbeschreibung der potentielle Suchraum des Planers drastisch erweitert wird, so daß ein autonomes Planen eines Systems, nur sehr eingeschränkt sinnvoll ist. Hier fehlen Kriterien, die eine sinnvolle Auswahl der Domänenoperatoren steuern. Eine Möglichkeit hierzu ist der Einsatz von Lernverfahren, um eine Situation zu identifizieren, in der die Anwendung eines komplexen Operators sinnvoll ist. Eine ähnliche Feststellung läßt sich auch für die Anwendung von Dekompositionsschemata auf komplexe Aktionen treffen. Hier sind Heuristiken gefragt, die für die Lösungsfindung gute Entscheidungshilfen liefern. Eine weitere Möglichkeit den Suchraum zu beschränken ist, analog zu dem in Abschnitt 3.3.2 beschriebenen Verfahren, ein Abstraktionsniveau auf den Operatoren einzuführen. Eine Kontrollkomponente kann dann bei der Zielerfüllung immer solche Operatoren bevorzugen die dieselbe Abstraktionstufe wie das Ziel haben. So wären dann komplexe Aktionen beispielsweise nur auf die initialen Ziele

einer Problembeschreibung anwendbar. Eine solche Strukturierung der Domänenoperatoren dürfte zu einer entscheidenden Verbesserung des Planungsverhaltens führen.

Ein weiterer Nachteil des Ansatzes stellt das Abrücken von HTN-Konzepten, für eine Annäherung an SNLP dar. Hierdurch geht die Möglichkeit einer besseren Problembeschreibung verloren. Außerdem werden die Konzepte der *Tasks* im Sinne von HTN (vgl. Abschnitt 3.1.3) nicht voll ausgenutzt, was dadurch zum Ausdruck kommt, daß Begriffe wie zu erreichendes Ziel nur durch Prädikate, aber nicht durch zu erfüllende Aufgaben, ausgedrückt werden können.

Die Vorteile des DPOCL-Ansatzes liegen darin, daß ein bestehendes SNLP-System leicht durch die entsprechenden Konzepte der Dekomposition, zu einem hierarchisch arbeitenden System erweitert werden kann. Ein weiterer nicht unerheblicher Vorteil bietet die verbesserte Domänenbeschreibung. Hierdurch lassen sich viele Sachverhalte einfacher und natürlicher in Planungsdomänen einbringen. Gerade bei kompexen Sachverhalten lassen sich so effizient Operatoren finden, die das gewünschte Verhalten in der Planungswelt realisieren. Daneben sind die Erweiterung der Domänenbeschreibung ein gutes Hilfsmittel für die interaktive Plangenerierung (vgl. [Weberskirch und Paulokat, 1995]) in einem System wie CAPLAN. Es wird hierdurch dem Benutzer wesentlich erleichtert eine sinnvolle Vorauswahl von Aktionen zu treffen, die das System dann selbstständig zu der Lösung eines (komplexen) Planungsproblems führt.

Anhang A

Verwendete Domänen und Testergebnisse

Die Domänen, die für die Testdurchläufe im Planungssystem CAPLAN verwandt wurden, reflektieren die verschiedenen Aspekte, die von dem DPOCL-Ansatz erwartet werden. Zum einen eine Verbesserung der Modellierungsmöglichkeiten und zum anderen ein Effizienzgewinn bei der Plangenerierung. Daher wurden bestehende Planungsdomänen des System entsprechend erweitert bzw. neu erstellt. Konkret wurden zwei Blocksworld-Domänen und die Workpiece-Planungsdomäne des System verwandt. Im folgenden sind die einzelnen Planungsdomänen ausführlicher beschrieben. Abschließend finden sich einige Ergebnisse der Testdurchläufe mit dem Planungssystem CAPLAN.

A.1 Verwendete Planungsdomänen

Um die Tests mit dem implementierten DPOCL-Verfahren durchzuführen wurden zwei Varianten einer Blocksworld-Domäne verwandt. Einerseits wurde eine für den SNLP-Planer entwickelte Domäne um einige komplexe Aktionen erweitert, als zweites wurde die in [Yang, 1990] beschriebene Domäne korrigiert.

Weiterhin wurde die für den SNLP-Planer entwickelte Workpiece-Domäne mit den Konzepten der Aktionsdekomposition neu modelliert.

A.1.1 Erweiterte Blocksworld

In der Planungsdomäne gibt es die Objekte *Block* und *Table*, die für die Klötzchen und den Tisch stehen. Der Tisch hat dabei eine unbegrenzte Ausdehnung, was bedeutet, das beliebig viele Klötzchen auf ihm stehen können. Die in der Planungswelt gültigen Zustände werden mit den Prädikaten *on(x,y)* und *clear(x)* beschrieben. Die Gültigkeit einer Aussage wird durch die Symbole + für gültig und − für ungültig ausgedrückt.

- **Primitive Aktionen:**

ClearTop(x,y,z)	
Constraints:	$IsOfType(Block, x, y), x \neq z, x \neq y, y \neq z$
Vorbedingungen:	$+On(x, y), +Clear(z), +Clear(y)$
Effekte:	$-On(y, x), +On(y, z), -Clear(z), +Clear(x)$

PutOn(x,y,z)

Constraints:	$IsOfType(Block, x), x \neq y, x \neq z, y \neq z$
Vorbedingungen:	$+On(x, z), +Clear(x) + Clear(y)$
Effekte:	$-On(x, z), +Clear(z), +On(x, y), -Clear(y)$

- **Komplexe Aktionen:**

ClearAndStack(x,y,z)

Constraints:	$IsOfType(Block, x, y, z), x \neq y, x \neq z, y \neq z$
Vorbedingungen:	$+On(x, y), +Clear(y), +Clear(z)$
Effekte:	$+On(x, z)$
Dekompositionen:	R1-MCS

StackTwo(x,y,z)

Constraints:	$IsOfType(Block, x, y, z), x \neq y, x \neq z, y \neq z$
Vorbedingungen:	$+Clear(y), +CLear(z)$
Effekte:	$+On(y, z), +On(x, y)$
Dekompositionen:	R2-S2

Swap(x,y)

Contraints:	$IsOfType(Block, x, y), x \neq y$
Vorbedingungen:	$+On(x, y), +Clear(x)$
Effekte:	$+On(y, x)$
Dekompositionen:	R3-Swap

- **Dekompositionsschemata:**

R1-MCS

Schritte:	$ClearTop(x, y, b), PutOn(x, z, b)$
Ordnungen:	$ClearTop(x, y, b) \prec PutOn(x, z, b)$
Causal Links:	$ClearTop(x, y, b) \stackrel{clear(x)}{\longrightarrow} PutOn(x, z, b)$

R2-S2

Schritte:	$PutOn(y, z, b), PutOn(x, y, a)$
Ordnungen:	$PutOn(y, z, b) \prec PutOn(x, y, a)$

R3-Swap

Schritte:	$ClearTop(y, x, a), PutOn(y, x, b)$
Ordnungen:	$ClearTop(y, x, a) \prec PutOn(y.x, b)$
Causal Links:	$ClearTop(y, x, a) \stackrel{+Clear(y)}{\longrightarrow} PutOn(y, x, b)$

Die primitiven Aktionen dienen dazu, Klötzchen zu stapeln oder sie frei zu machen. Die komplexen Aktionen sind, ganz im Sinne von DPOCL, so definiert, daß sie immer wieder auftauchende Teilsituationen lösen, wie beispielsweise die Aktion *Clear and Stack*, die zwei Klötzchen stapelt, obwohl eines der Klözchen nicht frei ist.

A.1.2 Korrekturen an der Domäne von Yang

Die in [Yang, 1990] beschriebe Domäne aus der Blocksworld war unvollständig, da ein Operator fehlte, der es erlaubt, Klötzchen, die einmal auf den Tisch gestellt wurden, von diesem wieder auf andere Klötzchen zu stellen. Durch diesen Umstand war eine sinnvolle Problemlösung mit dieser Domäne nicht möglich. Der primitive Operator *put-block-from-table* wurde daher der Domänenbeschreibung hinzugefügt. Weiterhin wurde ein neues Dekompositionschema

mit diesem neuen Planschritt definiert. Dieses Schema ist ist auf alle komplexen Schritte anwendbar, auf die auch das Schema *R5-makeon-block1* anwendbar ist.

put-block-from-table(x,y)	
Constraints:	$IsOfType(Block, x, y)$
	$x \neq y$
Vorbedingungen:	$+Clear(x), +Clear(y)$
	$+Ontable(x)$
Effekte:	$-Ontable(x), -Clear(y)$
	$+On(x, y)$

R5a-makeon-block1	
Schritte:	*put-block-from-table(x,y)*

Eine vollständige Beschreibung der Domäne findet sich in [Yang, 1990].

A.1.3 Die Workpiece-Domäne

In der Workpiece-Domäne sind die notwendigen Objekte und Aktionen modelliert, die es erlauben für rotationssymmetrische Drehteile die entsprechenden Arbeitspläne zur Fertigung solcher Teile auf CNC-Drehmaschinen zu erstellen. Die Planungsdomäne zeichnet sich durch eine Vielzahl primitiver Aktionen aus, deren Anwendung die verschiedenen zu fertigenden Aspekte bei den Drehteilen realisiert. Man unterscheidet verschiedenene Features die bei einem Drehteil zu Fertigen sind. Als Beispiel hierfür können Hinterschneidungen oder Gewinde genannt werden. Betrachtet man die Fertigung eines solchen Features genauer, so wird deutlich, daß hierzu immer dieselbe Sequenz von Aktionen ausgeführt werden muß: Das Drehteil muß aufgespannt, ein Werkzeug zur Berabeitung in die Maschine eingelegt und das entsprechende Feature gefertigt werden. Dabei unterliegen die einzelnen Aktionen gewissen Beschränkungen. So darf eine Einspannung des Drehteils beispielsweise nicht auf der zu fertigenden Seite erfolgen, da sonst die zu bearbeitende Fläche verdeckt wird. Ebenso darf auf ein einmal gefertigtes Gewinde nicht mehr aufgespannt werden, da dadurch das Gewinde beschädigt wird.

Mit dieser Überlegung, das alle zu fertigenden Features aus einer Sequenz von drei primitiven Schritten bestehen, ergibt sich, daß die Workpiece-Domäne ideale Vorraussetzungen mitbringt, um sie mit den Konzepten der hierarchischen Aktionsdekomposition zu modellieren. Im vorliegenden Fall wurde daher eine komplexe Aktion für die Fertigung einer Hinterschneidung definiert.

A.2 Verschiedene Testergebnisse

Im folgenden sind einige Testergebnisse des implementierten DPOCL-Verfahrens mit verschiedenen Domänen aufgeführt. Zum Vergleich werden sie, wo dies möglich ist, dem SNLP-Verfahren gegenübergestellt. Konkret werden einige Ergebnisse mit der Blocksworld-Domäne und der Workpiece-Domäne vorgestellt. Bei der Durchführung der Tests waren mehrere Askpekte interessant. Einerseits sollte das autonome Verhalten des System beobachtet werden, andererseits war auch von Interesse wie sich das Planungsverhalten ändert, wenn interaktiv, d.h. unter Einbeziehung des Benutzers geplant wird. Daher wurden die Problemstellungen einmal völlig selbstständig vom System gelöst, in anderen Fällen wurde eine Hilfestellung in Form einiger komplexer Schritte gegeben.

A.2.1 Betrachtete Problemstellungen

In den beschriebenen Blocksworld-Domänen wurden drei verschiedenen Problemstellungen formuliert. Abbildung A.1 zeigt die definierten Probleme im Überblick.

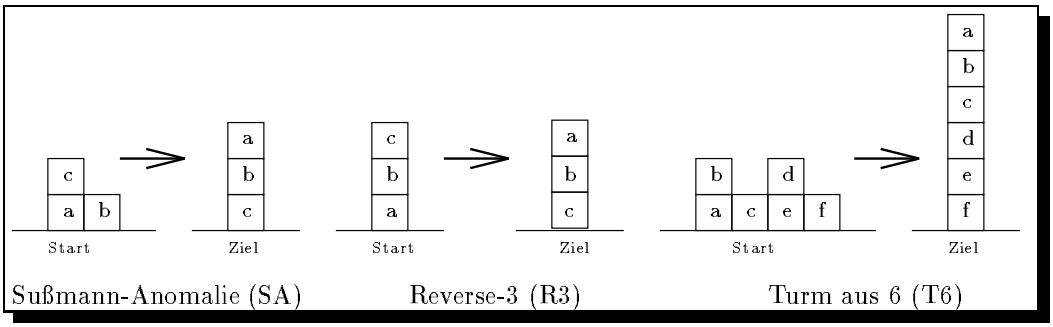


Abbildung A.1: Beispielprobleme aus der Blocksworld

Die Lösung der Probleme reicht von sehr einfach bis zu sehr aufwendig. Die Probleme wurden unter dem Aspekt ausgewählt, daß hier der SNLP-Planer eine entsprechend lange Kette von Inferenzschritten durchführen muß, um zu einer Lösung zu gelangen. Als einfach zu lösendes Problem wurde das Problem, daß auch unter der Bezeichnung Sußmann-Anomalie bekannt ist, gewählt. Ein etwas aufwendigeres Problem stellt das Umdrehen eines Turms aus drei Klötzchen dar. Schließlich wurde noch ein sehr komplexes Problem ausgewählt, in dem ein Turm aus sechs Klotzchen zu bauen ist. Hierbei sind die Klötzchen zum Teil schon in der richtigen Reihenfolge gestapelt, was zu einer Vielzahl von Interaktionen bei der Plangenerierung führt.

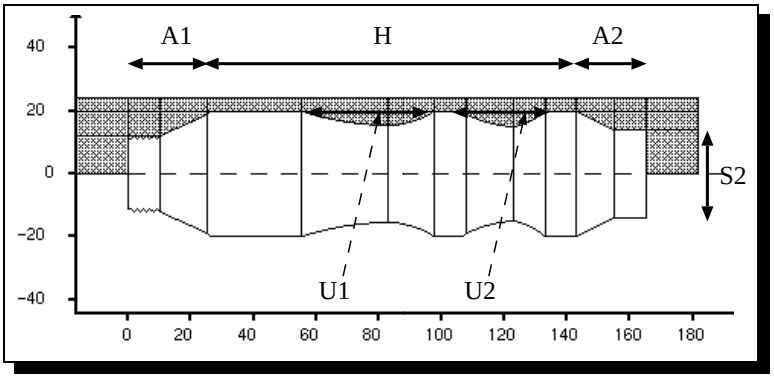


Abbildung A.2: Beispiel eines Drehteils

In der Workpiece-Planungsdomäne wurde eine Problembeschreibung für ein komplett zu fertigendes Drehteil mit verschiedenen gewählt, da es bei diesem Problem oft sehr lange dauert, bis der SNLP-Planer alle auftretenden Konflikte auflöst. Durch die Vorgabe der entsprechenden komplexen Aktionen wird hier erwartet, däs der Planer früher eine Lösung findet. Die zu fertigenden Hinterschneidungen sind in Abbildung A.2 mit $U1$ und $U2$ markiert.

A.2.2 Ergebnisse mit den Blocksworld-Domänen

Die in Abbildung A.1 dargestellten Probleme wurden in den beiden Blocksworld Planungsdomänen getestet. Alle Testdurchläufe hatten ein Zeitlimit von 15 Minuten. Zum Verständnis der Notationen in den Tabellen werden die folgenden Abkürzungen erläutert:

IS: Anzahl der benötigten Inferenzschritte.

VD/D: Anzahl der Entscheidungen, gültige und Entscheidungen insgesamt.

S/US: Anzahl der gültigen Schritte im Plan, die primitiven und die Anzahl der Schritte insgesamt.

SA: Das Planungsproblem Sußmann-Anomalie.

R3: Das Problem einen Dreierturm umzudrehen (Reverse-3).

T6: Das Planungsproblem einen Turm aus 6 Klötzchen zu bauen.

Eine Beschreibung der verwendeten Kontrollstrategien ist in Abschnitt 6.1.1 zu finden. Bei Planungsproblemen bei denen innerhalb des vorgegeben Zeitlimits keine Lösung gefunden wurde, sind die Eintragungen hervorgehoben.

Blocksworld									
	Problem								
	SA			R3			T6		
Strategie	IS	VD/D	S/US	IS	VD/D	S/US	IS	VD/D	S/US
SNLP+	16	16/16	3/3	*401*	*43/121*	*10/10*	*429*	*69/131*	*12/12*
DSep	14	14/14	3/3	32	13/20	3/3	*485*	*73/136*	*14/14*
Yang-Domäne									
	Problem								
	SA			R3			T6		
Strategie	IS	VD/D	S/US	IS	VD/D	S/US	IS	VD/D	S/US
SNLP+	13	13/13	3/3	33	14/22	4/4	*141*	*61/152*	*14/14*
DSep	13	13/13	3/3	38	14/24	4/4	*239*	*38/110*	*13/13*

Tabelle A.1: SNLP-Planungsergebnisse mit den primititven Blocksworld-Domänen

Tabelle A.1 zeigt einige Ergebnisse des SNLP-Basisplaners des CAPlan-Systems. Als Planungsdomänen wurden die entsprechenden primitiven Varianten der in den Abschnitten A.1.1 und A.1.2 beschriebenen Planungsdomänen verwendet. Alle Testdurchläufe erfolgten mit nicht-systematischer Interaktionsbehandlung, um so eine Verbesserung der SNLP-Planung zu erhalten. Aus den Ergebnissen ist ersichtlich, daß die Kontrollstrategie *DSep* schon eine wesentliche Verbesserung des SNLP-Algorithmus darstellt. Während mit der SNLP-Strategie schon das zweite Problem nicht innerhalb des vorgegebenen Zeitlimits gelöst wurde, fand die andere Kontrollkomponente recht schnell die Lösung. Lediglich bei dem aufwendigsten Planungsproblem konnte keine der Strategien innerhalb der Zeitgrenze eine Lösung finden.

In Tabelle A.2 sind einige Ergebnisse aus Planungsdurchläufen mit der erweiterten Blocksworld dargestellt. Hier wurde dem Planer keine Hilfestellung gegeben, er sollte das Problem selbstständig lösen. Da die Planungsdomäne nicht das in Definition 4.10 beschriebene Kriterium erfüllt, wurde nur die normale Interaktionbehandlung in den Planungsdurchläufen

Nicht-systematisches Vorgehen									
	Problem								
	SA			R3			T6		
Strategie	IS	VD/D	S/US	IS	VD/D	S/US	*IS*	*VD/D*	*S/US*
DPOCL	16	16/16	3/6	14	14/14	3/6	*1158*	*48/101*	*13/26*
DPOCL-D	16	16/16	3/6	14	14/14	3/6	*838*	*102/194*	*16/31*
DPOCL-B	16	16/16	3/6	14	14/14	3/6	*285*	*89/137*	*10/21*
Systematisches Vorgehen									
	Problem								
	SA			R3			T6		
Strategie	IS	VD/D	S/US	IS	VD/D	S/US	*IS*	*VD/D*	*S/US*
DPOCL	*593*	*46/152*	*7/15*	23	16/19	3/6	*1066*	*55/192*	*18/39*
DPOCL-D	*560*	*50/159*	*11/15*	21	16/18	3/6	*285*	*104/167*	*18/51*

Tabelle A.2: DPOCL-Planungsergebnisse mit der erweiterten Blocksworld-Domänen

verwendet. Um die Auswirkungen von systematischer und nicht-systematischer Interaktionsbehandlung zu erfassen, wurde entsprechend unterschieden. Aus den Ergebnissen erkennt man, daß sich bei der nicht-systematischen Interaktionsbehandlung eine leichte Verbesserung gegenüber SNLP ergibt. Mit dem hierarchischen Planungsansatz konnte der Planer auch das zweite Problem selbstständig lösen. Allerdings konnte auch hier innerhalb der Zeitbegrenzung keine Lösung für die komplexe Problemstellung gefunden werden. Im Falle der systematischen Interaktionsbehandlung tritt sogar eine Verschlechterung in der Planungseffizienz auf. Aus den Ergebnissen kann entnommen werden, daß es dem Planer nicht möglich war, innerhalb der Zeitgrenze das einfachste Problem, die Sußmann-Anomalie zu lösen. Es läßt sich festghalten, daß hier das Problem des DPOCL-Ansatzes deutlich zum tragen kommt, nämlich die Vielzahl der potentiell anwendbaren Operatoren auf ein offenes Ziel. Ohne eine sinnvolle Steuerung der Auswahl von Operatoren, führt das systematische Ausprobieren selten zu sinnvollen Ergebnissen.

Normale Interaktionsauflösung									
	Problem								
	SA			R3			T6		
Strategie	*IS*	*VD/D*	*S/US*	*IS*	*VD/D*	*S/US*	*IS*	*VD/D*	*S/US*
DPOCL	*33*	*26/28*	*17/38*	*31*	*25/25*	*16/33*	*27*	*26/26*	*8/33*
Erweiterte Interaktionsauflösung									
	Problem								
	SA			R3			T6		
Strategie	*IS*	*VD/D*	*S/US*	*IS*	*VD/D*	*S/US*	*IS*	*VD/D*	*S/US*
DPOCL	*319*	*161/241*	*45/54*	*350*	*154/230*	*12/53*	*309*	*156/253*	*12/53*

Tabelle A.3: DPOCL-Planungsergebnisse mit der Yang-Domäne

Die Tabelle A.3 gibt einen Überblick über die, mit der in Abschnitt A.1.2 beschriebenen Planungsdomäne, durchgeführten Tests. Da diese Domäne das Kriterium aus Definition 4.10 erfüllt, konnten die Auswirkungen der erweiterten Interaktionsbehandlung auf die Planungseffizienz überprüft werden. Aus den Ergebnissen der Planungsdurchläufe wird deutlich, daß sich durch diese Vorgehensweise keine Vorteile ergeben. Vielmehr ist erkennbar, daß durch

die Einschränkungen bei der Definitionsmöglichkeit der Planungsdomäne eine deutliche Verschlechterung der Planungsergebnisse eintritt, da es dem Planer in keinem Fall gelang, innerhalb der Zeitgrenze eine Lösung der gestellten Probleme zu finden. Auch durch die erweiterte Interaktionsauflösung ändert sich das Gesamtergebnis nicht. Zwar werden deutlich mehr Inferenzschritte ausgeführt, aber eine Lösung wird auch hier in keinem Fall gefunden.

Erweiterte Blocksworld									
Nicht-Systematisches Vorgehen									
	Problem								
	T6 - 2			T6 - 3			T6 - 4		
Strategie	IS	VD/D	S/US	IS	VD/D	S/US	IS	VD/D	S/US
DPOCL	1032	46/98	18/42	310	31/46	9/15	15	28/28	9/15
DPOCL-D	1102	45/107	19/39	285	30/46	9/15	13	27/27	9/15

Tabelle A.4: DPOCL-Planungsergebnisse mit Vorgabe von komplexen Schritten

Die Testdurchläufe als autonomes Plaungssystem haben gezeigt, daß durch den DPOCL-Ansatz nur in Teilen eine Verbesserung eintritt. Daher wurde noch untersucht inwiefern durch interaktive Planung eine Verbesserung erreicht werden kann. In Tabelle A.4 sind einige Ergebnisse mit der erweiterten Blocksworld dargestellt. Aus den Ergebnissen wird deutlich, daß sich die Planung durch die Vorgabe einiger komplexer Aktionen fortlaufend verbessert. Dies war auch zu erwarten, da es mit der erweiterten Beschreibungsmöglcihkeit in der Domänendefinition immer möglich ist, sich zu einem gegeben Planungsproblem den geeigneten komplexen Operator zu definieren. Daher ist es fast immer möglich, eine drastische Verbesserung bei der Lösungssuche zu erhalten. Im vorliegenden Fall wurde davon jedoch kein gebrauch gemacht, sondern es wurden lediglich komplexe Schritte aus der in Abschnitt A.1.1 beschriebenen Domänenbeschreibung verwendet.

Einfluß der Kontrollstrategien

Ein signifikanter Unterschied der gewählten DPOCL-Kontrollkomponenten bei der Lösungsfindung konnte nicht beobachtet werden. Daher kann auch keine Aussage darüber gemacht werden, welcher Art der Zielbearbeitung der Vorzug gegeben werden sollte, der Zerlegung komplexer Aktionen oder der Erfüllung von Vorbedingungen. Was allen Strategien fehlt sind effiziente Heuristiken, welche die Anwendung der komplexen Domänenoperatoren steuern. Hier wird deutlich, das durch die interaktive Planung, bei der sinnvolle Aktionen im Plan vorgegeben werden. eine deutliche Verbesserung eintritt. Daher muß das Ziel für eine Kontrollkomponente die Integration solcher Heuristiken sein.

A.2.3 Ergebnisse mit der Workpiece-Domäne

Die Versuche mit den Blocksworld-Domänen zeigten, daß der DPOCL-Ansatz ungeeignet scheint vom System eine autonome Problemlösung vornehmen zu lassen. Daher war bei der Workpiece-Domäne vor allem der interaktive Aspekt der Plangenerierung interessant. Betrachtet wurde ein Problem, bei dem auf einem Werkstück zwei Hinterschneidungen zu fertigen sind.

Aus Tabelle A.5 wird ersichtlich, wie sich das Planungsverhalten ändert, wenn komplexe Schritte vom Benutzer vorgegeben werden. Benötigt SNLP noch sehr viele Inferenzschritte,

	Problem			
Strategie	IS	VD/D	S/US	Vorgabe Schritte
SNLP+	128	23/58	5/5	0
DSep	112	23/58	5/5	0
DPOCL	52	27/39	5/10	1
DPOCL	14	18/18	5/8	2

Tabelle A.5: Planungsergebnisse in der Workpiece-Domäne

so erreicht man duch die Vorgabe von nur einer komplexen Aktionen eine deutliche Verbesserung bei der Lösungsuche. Gibt man bei dem betrachteten Problem zwei Planschritte vor, hat die Berechnung der Lösung sogar nur noch linearen Aufwand, da alle schwierigen Entscheidungspunkte schon gelöst sind. Ein solcher Verhalten läßt sich aber nur durch sinnvolle Integration des Benutzers in den Planungsprozeß erreichen, da der DPOCL-Ansatz als selbstständiger Planer keine solchen Ergebnisse erreicht.

Anhang B

Beispiel für einen Planungsablauf mit Aktionsdekomposition

Als Beispiel für einen Planungsablauf mit hierarchischer Aktionsdekomposition wird ein einfaches Beispiel aus der Blocksworld in der erweiterten Planungsdomäne (vgl. Anhang A.1.1) mit komplexen Aktionen gelöst. Die verwendeten Abbildungen bilden Screenshots der Ausgaben des Planungssystems CAPlan.

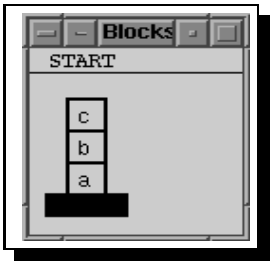


Abbildung B.1: Startsituation

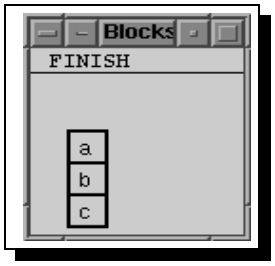


Abbildung B.2: Zielsituation

Das zu lösende Planungsproblem ist in den Abbildungen B.1 und B.2 einer graphischen Veranschaulichung dargestellt. Die zugehörige Problembeschreibung im CAPlan-System wird durch folgende Prädikate beschrieben:

Startsituation: $\{+On(a,t),+On(b,a),On(c,b),+Clear(c)\}$

Zielsituation: $\{+On(a,b),+On(b,c)\}$

Die Abbildungen B.3 und B.4 zeigen die initiale Situation im Plan und in der Redux-Teilzielhierarchie nachdem das Planungsproblem an CAPlan übergeben wurde. Wie in Abbildung B.4 dargestellt, sind nun die zu erreichenden Ziele der Problemstellung für das Planungssystem hergeleitet.

Zur Lösung der gestellten Aufgabe wird der komplexe Schritt $Swap(b,a)$ in den Plan aufgenommen und durch Anwendung des zugeordneten Dekompositionsschemas zerlegt. Das entspricht der Anwendung des komplexen Domänenoperators $Swap(x,y)$ auf das Teilzeil $+On(a,b)$ sowie die Anwendung des Dekompositionsoperators *R3-Swap* auf das hergeleitete *Decompose-Goal*. Abbildung B.5 zeigt den so entstandenen Plan. Die Teilzielstruktur mit den Entscheidungen, die zu diesem Plan führten, sind in der Abbildung B.6 festgehalten.

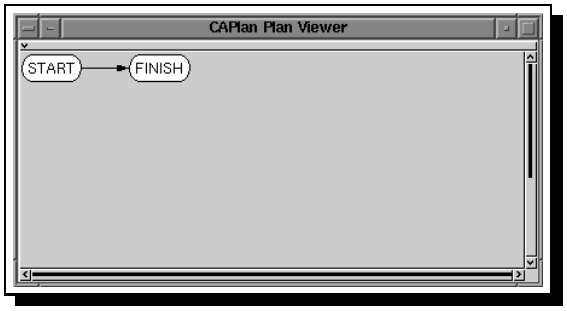


Abbildung B.3: Initialer Plan des Problems

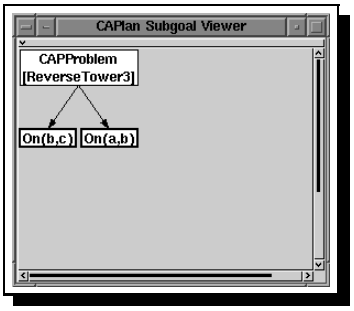


Abbildung B.4: Initiale Teilzielstruktur

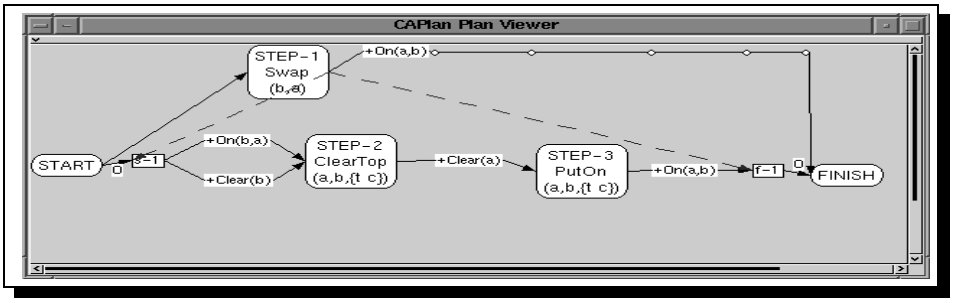


Abbildung B.5: Einführung eines komplexen Schrittes in eine Plan

Aus Abbildung B.8 wird ersichtlich, daß der Schemaschritt *ClearTop* genutzt wurde, daß noch offene Ziel $+On(b, c)$ zu erfüllen. Nach diesem Inferenschritt ist das Planungsproblem grundsätzlich gelöst. Es ist lediglich noch erforderlich die Vorbedingung $+Clear(b)$ des Planschritts *Step-3* zu erfüllen. Eine sinnvolle Entscheidung, um die Vorbedingung zu garantieren, ist die Aufnahme des primitiven Schritts *Step-4* in den Plan. Er repräsentiert den Domänenoperator *ClearTop* und dient dem Zweck, daß Klötzchen c von b auf den Tisch zu stellen.

Durch die Aufnahme dieses primitiven Planschritts in den Plan entsteht ein Konflikt mit

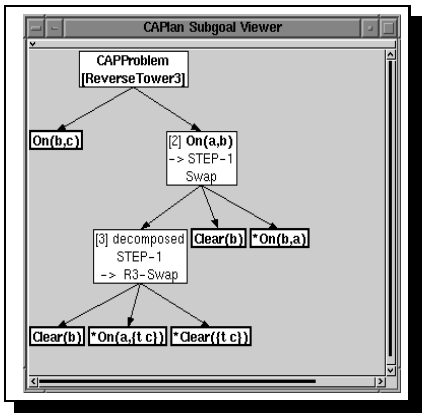


Abbildung B.6: Teilzielstruktur nach der Zerlegung einer komplexen Aktion

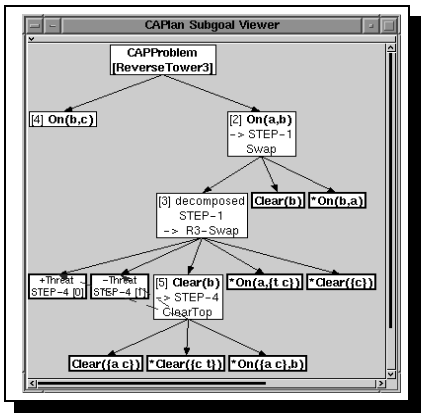


Abbildung B.7: Teilzielstruktur für den Konflikt im Plan

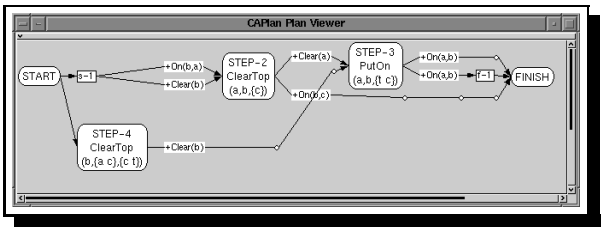


Abbildung B.8: Plan mit Konflikt

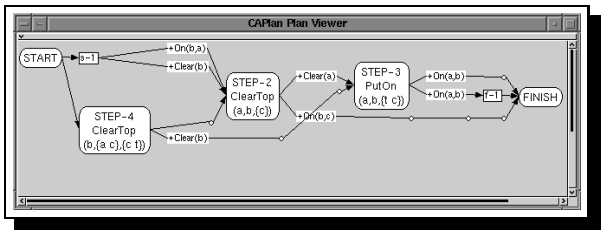


Abbildung B.9: Auflösung des Konflikts durch Protections

den Schritten, die in dem Dekompositionsschema *R3-Swap* enthalten sind. Abbildung B.7 zeigt die zugehörige Teilzielstruktur des erzeugten Planes mit den entsprechenden Zielen, welche die Konflikte repräsentieren. In dem Beispiel lassen sich folgende Interaktionen zwischen den Planschritten identifizieren: Einerseits bedroht *Step-2* mit seinem Effekt $+Clear(a)$ den Causal Link **Step-4** $\stackrel{+Clear(b)}{\longrightarrow}$ **Step-3**, andereseits tritt eine Interaktion zwischen *Step-4* und einer Vorbedingung der komplexen Aktion, hier symbolisiert durch den Causal Link **s-1** $\stackrel{+Clear(b)}{\longrightarrow}$ **Step-2** auf. In beiden Fällen handelt es sich um eine positive Interaktion. Um den ersten Konflikt aufzulösen wird eine Protection in den Plan eingeführt, die Step-4 vor Step-2 anordnet. Abbildung B.9 zeigt den daraus resultierenden Plan.

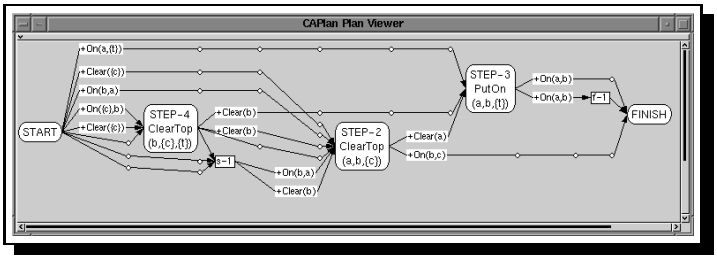


Abbildung B.10: Plan zur Lösung des Problems

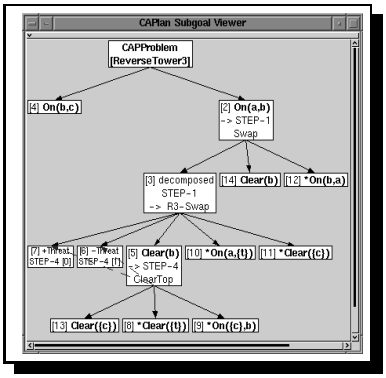


Abbildung B.11: Teilzielstruktur der Lösung des Problems

Die gefundene Sequenz der Schritte ist dazu geeignet das Planungsproblem vollständig zu lösen. Es folgen daher noch einige Inferenzschritte, welche die noch offenen Ziele aus dem Plan heraus erfüllen. Abbildungen B.11 und B.10 zeigen den Plan und die Teilzielstruktur, welche die tatsächlich gefundene Lösung des Planungsproblems beschreiben.

Im Zusammenhang mit einer autonomen Planung mit dem DPOCL-Verfahren sind in den Abbildung B.12 und B.13 zwei Suchbäume dargestellt. Der Planer versuchte dort ein sehr einfaches Planungsproblem, daß Umdrehen zweier Klötzchen, selbständig zu lösen. Dabei wurde einerseits die primitive Planungsdomäne Blocksworld und andereseits die erweiterte Blocksworld verwandt. In dem in Abbildung B.13 dargestellten Suchbaum wird das Problem des DPOCL-Ansatzes deutlich: Ohne eine sinnvolle Steuerung der Operatorauswahl müssen sehr viele Entscheidungen wieder zurückgenommen werden, da sie nicht geeignet sind das Problem zu lösen. Daher wird der ein sehr großer Teil des Suchraums unnötig durchlaufen,

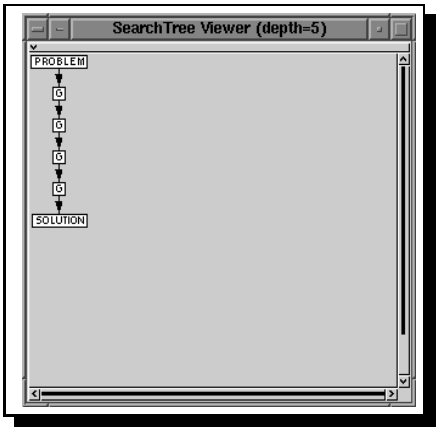


Abbildung B.12: Suchbaum für die Blockworld

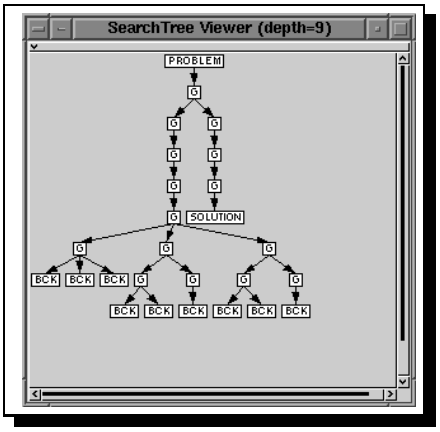


Abbildung B.13: Suchbaum für die erweiterte Blocksworld

bis endlich ein sinnvoller Operator angwandt wird. Das Verfahren produziert daher bei komplexeren Problemstellungen sehr große Suchbäume, was das sehr ineffiziente Verhalten bei der selbstständigen Planung erklärt.

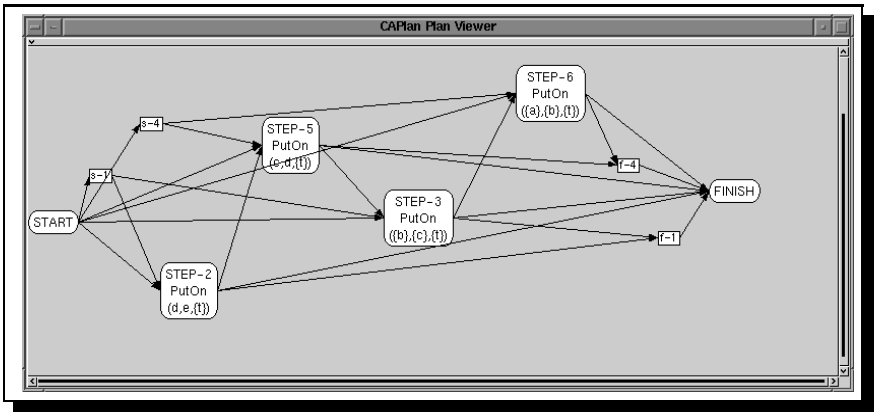


Abbildung B.14: Beispiel für die Verzahnung von Dekompositionsschemata

Ein Vorteil des DPOCL-Ansatzes besteht darin, daß die Zerlegungen von komlexen Aktionen ineinandergreifend Probleme lösen können. In Abbildung B.14 ist ein solcher Fall dargestellt. Hier bestand das Planungsproblem darin einen Turm aus vier Klötzchen zu bauen, wobei alle Klötzchen am Anfang frei auf dem Tisch stehen. Zur Lösung des Planungsproblems wurden zwei Domänenoperatoren *StackTwo* verwandt, deren Dekompositionschritte durch Verzahnung in die Zielsituation führen.

Abbildungsverzeichnis

Literaturverzeichnis

Barrett, A. und Weld, D.S. 1994a. Partial-Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, **67**(1), 71–112.

Barrett, A. und Weld, D.S. 1994b. Task-Dekomposition via Plan Parsing. *Seiten 1117–1122 in: Proceedings of AAAI-94.*

Chapman, D. 1987. Planning for Conjunctive Goals. *Artificial Intelligence*, **32**, 333–377.

Currie, K. und Tate, A. 1991. O-Plan: The Open Planning Architecture. *Artificial Intelligence*, **52**(1), 49–86.

Doyle, J. 1979. A Truth Maintenance System. *Artificial Intelligence*, **12**(3), 231–272.

Drummond, M. 1993. On Precondition achievement and the computational economics of automatic planning. *In: Proceedings of the 2nd European Workshop on Planning (EWSP-93).*

Erol, K., Hendler, J. und Nau, D.S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. *Seiten 249–254 in: Proceedings of the 2nd International Conference on AI Planning Systems (AIPS-94).*

Erol, K., Hendler, J., Nau, D.S. und Tsuneto, R. 1995. A Critical Look at Critics in HTN Planning. *In: Proceedings of IJCAI-95.*

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

Fikes, R.E., Hart, P.E. und Nilsson, N.J. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence*, **3**(4), 251–288.

Goldberg, A. und Robinson, D. 1983. *Smalltalk 80 - The Language and Its Implementation.* 2. Auflage. Stuttgart: Addison-Wesley.

Knoblock, C.A und Yang, Q. 1995. Relating the Performance of Partial-Order Planning Algorithms to Domain Features. *SIGART Bulletin*, **6**(1).

McAllester, D. und Rosenblitt, D. 1991. Systematic Nonlinear Planning. *Seiten 634–639 in: Proceedings of AAAI-91.*

Muñoz-Avila, H., Paulokat, J. und Wess, S. 1995. Controlling non-linear hierarchical planning by case replay. *In:* Keane, M., Halton, J.P. und Manago, M. (Hrsg.), *Advances in Case-Based Reasoning. Selected Papers of the 2nd European Workshop (EWCBR-94).* Lecture Notes in Artificial Intelligence, Nr. 984. Springer.

PENBERTHY, J.S. UND WELD, D.S. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. *In: Proceedings of KR-93.*

PEOT, M. UND SMITH, D. 1993. Threat-removal strategies for partial-order planning. *Seiten 492–499 in: Proceedings of AAAI-93.*

PETRIE, CH. 1991. *Planning and Replanning with Reason Maintenance.* Dissertation, University of Texas at Austin, CS Dept.

PETRIE, CH. 1992. Constrained Decision Revision. *Seiten 393–400 in: Proceedings of AAAI-92.*

RICHTER, M.M. 1992. *Prinzipien der künstlichen Intelligenz.* 2. Auflage. Stuttgart: B.G. Teubner.

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

SACERDOTI, E. 1973. Planning in a Hierarchy of Abstraction Spaces. *Seiten 412–422 in: Proceedings of IJCAI-73.*

TATE, A. 1977. *Project Planning Using a Hierarchical Non-linear Planner.* Research Report No. 25. Dept. of Artificial Intelligence, Edinburgh University.

TATE, A., DRABBLE, B. UND DALTON, J. 1994. The Use of Condition Types to Restrict Search in an AI Planner. *Seiten 1129–1134 in: Proceedings of AAAI-94.*

WEBERSKIRCH, F. 1994. *Realisierung eines nichtlinearen Planungssystems zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM).* Diplomarbeit, Universität Kaiserslautern.

WEBERSKIRCH, F. 1995. *Combining SNLP-like Planning and Dependency-Maintenance.* Bericht LSA-95-10E. Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

WEBERSKIRCH, F. UND PAULOKAT, J. 1995. CAPlan - ein SNLP-basierter Planungsassistent. *In:* BIUNDO, S. UND TANK, W. (Hrsg.), *Beiträge zum 9. Workshop 'Planen und Konfigurieren' (PuK-95).* DFKI.

WILKINS, D.E. 1983. Representation in a Domain-Independent Planner. *In: Proceedings of IJCAI-83.*

WILKINS, D.E. 1988. *Practical Planning - Extending the classical AI Planning Paradigm.* Morgan Kaufmann.

YANG, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, **6**, 12–24.

YANG, Q. UND TENENBERG, J.D. 1990. ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner. *Seiten 204–209 in: Proceedings of AAAI-90.*

YOUNG, R.M., POLLACK, M.E. UND MOORE, J.D. 1994. Decomposition and Causality in Partial-Order Planning. *Seiten 188–193 in: Proceedings of the 2nd International Conference on AI Planning Systems (AIPS-94).*