Diplomarbeit

Effizienzsteigerung des partiell-ordnenden Planers CAPlan mit Hilfe von Operatorgraphen

Jochem Hüllen

Juli 1998

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

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

Universität Kaiserslautern
Fachbereich Informatik

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Das Gebiet der Planung ist ein klassischer Bereich in der Künstlichen Intelligenz, der bereits eine lange Tradition aufweist. Es gibt mehrere Ansätze, Planungsaufgaben zu lösen. Diese lassen sich durch viele unterschiedliche Aspekte klassifizieren. Dazu gehören Unterscheidungen wie *domänenabhängig - domänenunabhängig, Suche im Zustandsraum - Suche im Planraum, Planung mit totaler Ordnung - Planung mit partieller Ordnung, usw.* Der in *CAPlan* [Weberskirch, 1995] umgesetzte SNLP-Ansatz [McAllester und Rosenblitt, 1991] ist ein vielbeachteter Ansatz. *CAPlan* läßt sich bezüglich der oben erwähnten Eigenschaften als domänenunabhängiges Planungssystem mit Suche im Planraum und partiell geordneter Planrepräsentation einordnen. Wie auch in anderen Bereichen sind solche Ansätze oft, theoretisch betrachtet, korrekt und vollständig. Bei schwierigen Planungsaufgaben ergibt sich jedoch wegen der großen Komplexität der Planung an sich (NP-vollständig [Bylander, 1994]) häufig das Problem eines zu hohen Zeitaufwandes zur Lösungsfindung.

Diese Arbeit beschäftigt sich mit einer Möglichkeit zur Effizienzverbesserung, wobei das SNLP-basierte Planungssystem *CAPlan* verwendet wird. Dabei werden neue, zu lösende Probleme einer Vorverarbeitung unterzogen. Dort werden bestimmte Eigenschaften ermittelt, ohne jedoch das Problem zu lösen. Anschließend wird dem Planungssystem das neue Problem mit dem Zusatzwissen in Form der analysierten Eigenschaften übergeben. Das Planungssystem verwendet das Wissen, um effizienter eine Lösung zu finden.

In dieser Arbeit werden primär Eigenschaften ermittelt, die besagen, in welcher Reihenfolge die Teile eines Problems in allen möglichen Lösungen gelöst werden müssen. Dadurch kann der effektive Suchraum beim Planen eingeschränkt werden. Eigenschaften, die nicht für alle, sondern nur für einige Lösungen gelten, können auch den Suchraum einschränken. Sie schränken aber auch die Menge der Lösungen ein und werden in dieser Arbeit nicht untersucht.

1.1 Planung/CAPlan

CAPlan ist ein domänenunabhängiges Planungssystem. Darunter versteht man ein System, welches Planungsaufgaben in allen Planungswelten, den *Domänen*, die sich mit einer bestimmten Beschreibungssprache definieren lassen, lösen kann. In einer Domnänenbeschreibung werden u.a. die Möglichkeiten für Beschreibungen von *Situationen* und *Aktionen* festgelegt. Ein Planungsproblem in einer solchen Domäne besteht darin, daß eine Ausgangsituation durch mehrere elementare Aktionen in die gewünschte Zielsituation transformiert werden soll. Dabei wird die Zielsituation durch eine Menge von Teilzielen beschrieben. In einer Lösung für ein Planungsproblem, einem *Plan*, werden diese Teilziele durch einzelne, elementare Aktionen in einer bestimmten Reihenfolge erreicht. Diese Reihenfolge resultiert aus Abhängigkeiten zwischen den Aktionen und wird Zielordnung genannt. In diesem Zusammenhang sind folgende Fragestellungen von Interesse:

- Werden für ein Problem einzelne Teilziele bei allen Lösungen immer in der gleichen Reihenfolge gelöst?
- Können solche Zielordnungen mit angemessenem Aufwand berechnet werden?
- Kann ein Planungssystem mit dem Wissen solcher Zielordnungen effizienter planen?

Alle drei Fragen können mit 'Ja' beantwortet werden. In dieser Arbeit wird gezeigt, warum dies der Fall ist und im besonderen, wie man solche Zielordnungen berechnen kann.

1.2 Inhalt der Arbeit

Die oben erwähnten Zielordnungen stellen eine Art von Eigenschaften dar, die in der Vorverarbeitung eines Problems erkannt werden können. Diese Arbeit beschäftigt sich primär mit dem Erkennen solcher Zielordnungen. Von besonderem Interesse sind Zielordnungen, die für alle Lösungen eines Problems gelten, denn mit dem Wissen solcher notwendigen Zielordnungen kann das Planungssystem den Suchraum wirksam eingrenzen, ohne daß dabei Lösungen verloren gehen. Wie können notwendige Zielordnungen berechnet werden? Dazu müssen alle möglichen Lösungen eines Problems betrachtet werden! Also müssen alle Lösungen für ein Problem in der zugehörigen Domäne durch Ausnutzung von Gemeinsamkeiten in einer abstrakten Form so repräsentiert werden, daß Eigenschaften der Lösungen, die Zielordnungen zur Folge haben, erkannt werden können.

In dieser Arbeit werden die erstmals in [Smith und Peot, 1993] vorgestellten Operatorgraphen so erweitert, daß sie diesen Anforderungen in vielen Domänen entsprechen. Die einfachen Operatorgraphen halten die Abhängigkeiten zwischen den für ein Problem relevanten Prädikaten und Operatoren der Domäne grundsätzlich fest. Durch die Erweiterungen der Operatorgraphen ist zusätzliche Information vorhanden, welche Auskunft darüber gibt, auf welche Weise die Operatoren für die einzelnen Teil- und deren Unterziele relevant sind. Dadurch können Abhängigkeiten in Form von Ordnungen zwischen den Operatoren erkannt werden. Daraus können wiederum Zielordnungen abgeleitet werden. In dieser Arbeit werden nur notwendige Zielordnungen betrachtet. Andere Zielordnungen, die z.B. nur

für einige Lösungen gelten, aber nicht für alle, können den Suchraum auch einschränken, allerdings können dabei auch Lösungen verloren gehen. Schränkt man den Suchraum mit solchen Zielordnungen ein, so fokussiert man auf die Lösungen, die konsistent mit den angegeben Zielordnungen sind.

In dieser Arbeit werden mehrere Kriterien gegeben, die besagen, wann notwendige Zielordnungen gelten. Es wird bewiesen, daß diese Kriterien korrekt sind, das heißt, es werden nur Zielordnungen erkannt, die in allen Lösungen gelten müssen. Die Kriterien beziehen sich auf Eigenschaften der zugehörigen Operatorgraphen und sind operational formuliert. Sie haben also nicht nur den theoretischen Anspruch, daß damit notwendige Zielordnungen berechnet werden können, sondern es können daraus direkt effiziente Algorithmen abgeleitet werden. Alle Kriterien wurden implementiert und getestet. Die Tests zeigen, daß in vielen, in der Planungsliteratur erwähnten Domänen die meisten notwendigen Zielordnungen gefunden werden, daß es aber auch Domänen gibt, in denen kaum solche Ordnungen entdeckt werden, obwohl sie vorhanden sind. Weiter wird gezeigt, daß *CAPlan* mit den berechneten Ordnungen sowohl beim generativen wie auch beim fallbasierten Planen (CAPlan/CbC, siehe [Muñoz-Avila, 1998]) effizienter ist.

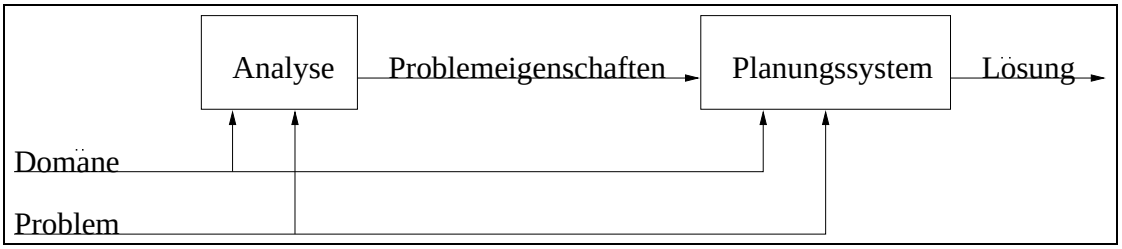

Abbildung 1.1: Schematischer Ablauf

Abbildung 1.1 veranschaulicht die oben beschriebene prinzipielle Vorgehensweise. Dem Planungssystem wird nicht wirklich neues, zusätzliches Wissen zur Verfügung gestellt. Vielmehr wird das Wissen in Form der Domänen- und Problembeschreibung so transformiert, daß es genutzt werden kann, um den Suchraum beim Planen einzuschränken. Die analysierten Problemeigenschaften, z.B. Teilzielordnungen, stellen also Wissen dar, welches so aufbereitet wurde, daß es direkt im Planungssystem zur Effizienzsteigerung verwendet werden kann.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

- Im 2. Kapitel werden die Grundlagen für diese Arbeit dargestellt. Neben den relevanten Aspekten von *CAPlan* und partiell-ordnender Planung werden die oben erwähnten Operatorgraphen und primär die für diese Arbeit notwendigen Erweiterungen davon erläutert.
- In Kapitel 3 werden die eigentlichen Analyseverfahren vorgestellt, die u.a. die Zielordnungen vorausberechnen.
- Kapitel 4 zeigt, wie die bei der Vorverarbeitung berechneten Eigenschaften in *CAPlan* bzw. der fallbasierten Erweiterung CAPlan/CbC genutzt werden, um effizienter zu planen.
- Im 5. Kapitel werden eine Reihe von empirischen Untersuchungen vorgestellt, welche die Effizienzsteigerungen in verschiedenen Domänen untersuchen.
- Schließlich werden in Kapitel 6 die Ergebnisse zusammengefaßt, bewertet und mit denen anderer Arbeiten verglichen.

Die Implementierung wurde in Smalltalk (VisualWorks2) erstellt und setzt auf die Implementation der Systeme *CAPlan* und *CAPlan/CbC* auf.

Kapitel 2

Grundlagen

In diesem Kapitel werden einige Begriffe als Grundlage für die folgenden Kapitel vorgestellt. Dabei werden die für diese Arbeit relevanten Teile von *CAPlan* und die Operatorgraphen erklärt.

2.1 CAPlan

CAPlan ist ein domänenunabhängiges, SNLP-basiertes Planungssystem. Eine vollständige Beschreibung findet sich in [Weberskirch, 1995]. Domänenunabhängige Systeme bedienen sich einer Domänenbeschreibungssprache, mit der die zu behandelnden Domänen definiert werden. In einer Domänenbeschreibung werden u.a. die Prädikate, die Zustände beschreiben, und die Operatoren, die Zustände verändern können, definiert.

Definition 2.1 (Domänenbeschreibung) *Eine Domänenbeschreibung D = (P, O, T) besteht aus drei Komponenten: P ist eine Menge von Prädikaten, O ist eine Menge von Operatoren und T ist die Beschreibung einer Typenhierarchie.*

Dabei sind Prädikate funktionsfreie Atomformeln und Operatoren werden in einer STRIPS-ähnlichen Form durch Vor- und Nachbedingungen notiert. Die Nachbedingungen eines Operators heißen auch Effekte. Operatoren können Variablen enthalten und sind somit Muster für die in Plänen vorkommenden Instanzen. In *CAPlan* können Operatoren mit Constraints über ihren Variablen definiert werden. Es sind sowohl Gleichheits-Constraints [Chapman, 1987] als auch Typ-Constraints [Weberskirch und Muñoz-Avila, 1997] erlaubt. Letztere stellen eine Erweiterung gegenüber dem Basisverfahren von *CAPlan*, SNLP [McAllester und Rosenblitt, 1991], dar. Dabei gibt es zwei Formen von Typ-Constraints:

- *IsOfType(typ, var)* besagt, daß die Variable *var* unbedingt den Typ *typ* haben muß.
- *IsNotOfType(typ, var)* besagt, daß die Variable *var* auf keinen Fall den Typ *typ* haben darf.

Nun kann ein Planungsproblem in einer bestimmten Domäne beschrieben werden, indem die Ausgangssituation, die gewünschte Zielsituation und alle vorkommenden Objekte mit ihren jeweiligen Typen angegeben werden.

Definition 2.2 (Planungsproblem) *Ein Planungsproblem $P = (S, Z, O)$ in einer Domäne D besteht aus drei Komponenten: S und Z sind Mengen von Literalen, die aus den Prädikaten von D gebildet werden, wobei S den Startzustand und Z den Zielzustand beschreibt. Die einzelnen Prädikate aus Z heißen* **Ziele** *von P. O ist eine Menge von Objekten mit zugehörigem Typ.*

Die Aufgabe eines Planungssystems ist es nun, für ein solches Problem eine Lösung in Form eines Planes zu finden.

Definition 2.3 (Lösung) *Eine Lösung $L = (M, Ord)$ zu $P = (S, Z, O)$ in D ist eine Menge M von Operatoranwendungen (Operatoren aus D samt Variablenbindungen) mit einer Ordnung Ord auf den Elementen von M, so daß ausgehend von S, bei Anwendung aller Operatoren von M, in der durch Ord definierten Reihenfolge, Z entsteht.*

Bei einer solchen Lösung müssen zum Zeitpunkt der Ausführung eines Operators seine Vorbedingungen gültig sein. Daraus ergeben sich Abhängigkeiten zwischen den Operatoren. Beim SNLP-Ansatz werden zur Buchführung dieser Abhängigkeiten Causal Links eingeführt. Diese sollen die Gültigkeit einer Vorbedingung eines Operators bei dessen Ausführung garantieren.

Definition 2.4 (Causal Link) *Wenn O_1 ein Operator mit einem Effekt p ist, der die Vorbedingung p eines Operators O_2 erfüllen soll, so hält ein Causal Link $O_1 \rightarrow p@O_2$ diesen Zusammenhang fest.*

Da mehrere in einem Plan zueinander parallel liegende Operatoren einen Effekt p oder $\neg p$ haben können, kann es zu Interaktionen kommen.

Definition 2.5 (Interaktion) *Eine Interaktion $O \leftrightarrow CL$ zwischen einem Operator O und einem Causal Link $CL = O_1 \rightarrow p@O_2$ besteht dann, wenn O einen Effekt p (positive Interaktion) oder $\neg p$ (negative Interaktion) hat und O nicht vor O_1 oder nach O_2 angeordnet ist.*

Positive Interaktionen: Sie beeinflussen nicht die Korrektheit des Planes, sondern Nichtbeachtung führt lediglich zum Verlust der Systematik der Lösungssuche. Daher können aus ihnen keine Zielordnungen resultieren, weshalb sie in dieser Arbeit nicht weiter betrachtet werden.

Negative Interaktionen: Hier wird die Gültigkeit benötigter Vorbedingungen zerstört. Somit können aus ihnen Ordnungen zwischen Operatoren und dadurch auch zwischen Zielen resultieren.

Aus den Ordnungen zwischen den Operatoren lassen sich Zielordnungen ableiten. Von besonderem Interesse sind aber nicht Ordnungen zwischen Zielen, die spezifisch für eine bestimmte Lösung des Problems sind, sondern vielmehr Ordnungen, die für alle Lösungen gelten und somit spezifisch für das Problem selber sind. Ordnungen, die die Lösungsmenge einschränken, werden hier nicht untersucht. Sie können allerdings auch eingesetzt werden, um den Suchraum einzuschränken oder um auf bestimmte Lösungen zu fokussieren.

Definition 2.6 (Notwendige Zielordnung) *Zwischen zwei Zielen Z_1 und Z_2 eines Problems P besteht eine notwendige Zielordnung, wenn für jede Lösung L von P gilt: In L werden wegen vorhandenen Ordnungen und Causal Links die Operatoren auf jeden Fall so ausgeführt, daß Z_1 vor Z_2 gelöst wird.* **Notation:** $Z_1 <_Z Z_2$

Es wird also von Ordnungen zwischen Operatoren auf Ordnungen zwischen Zielen geschlossen. Dabei ist in den einzelnen Lösungen die Ordnung der Operatoren, die die Teilziele letztendlich erfüllen (diese Operatoren nennt man **Establisher** des jeweiligen Ziels), von besonderem Interesse, denn daraus resultieren direkt Zielordnungen. Die Frage ist also, wie kann man zeigen, daß in jeder Lösung bestimmte Ordnungen zwischen bestimmten Operatoren und damit zwischen bestimmten Zielen gelten. Diese Frage wird in Kapitel 3 beantwortet.

2.2 Operatorgraphen

Für die in Kapitel 3 vorgestellten Analyseverfahren wird eine geeignete Abstraktionsebene für Probleme zusammen mit der jeweiligen Domäne bzw. für alle möglichen Lösungen von Problemen in einer Domäne benötigt. Diese sollte die Eigenschaften eines Problems so repräsentieren, daß sich die während der Vorverarbeitung aufkommenden Anfragen effizient beantworten lassen. Wie sich später herausstellen wird, sind die erweiterten Operatorgraphen eine geeignete Struktur. Operatorgraphen wurden erstmals in [Smith und Peot, 1993] vorgestellt, dort wurden mit ihnen Interaktionen analysiert. In [Smith und Peot, 1996] werden Operatorgraphen zur Erkennung von Zielrekursionen eingesetzt.

Definition 2.7 (Operatorgraph) *Ein Operatorgraph OG für ein Problem P in einer Domäne D ist ein gerichteter bipartiter Graph mit folgenden Eigenschaften:*

1. *Es gibt zwei Sorten von Knoten:* **Vorbedingungsknoten** *sind mit Prädikatnamen aus D beschriftet und* **Operatorknoten** *sind mit Operatornamen aus D beschriftet.*
2. *In OG gibt es einen Operatorknoten für den künstlichen Operator FINISH von P.*
3. *Gibt es einen Operatorknoten OK in OG, so gibt es für jede Vorbedingung des Operators von OK einen Vorbedingungsknoten VK mit einer gerichteten Vorgängerkante $VK \rightarrow OK$.*
4. *Gibt es einen Vorbedingunsknoten VK in OG, so gibt es für jeden Operator aus D, der einen Effekt besitzt, welcher mit dem Prädikat von VK unifizierbar ist, einen Operatorknoten OK mit einer gerichteten Vorgängerkante $OK \rightarrow VK$.*

5. *Für jeden Operator aus D gibt es höchstens einen Operatorknoten in OG, aber für jede Vorbedingung eines Operators gibt es einen eigenen Vorbedingungsknoten.*

Bemerkungen:

- Ein Operatorgraph gibt Auskunft über den Zusammenhang zwischen den Prädikaten und den Operatoren von D für P und über die Struktur von möglichen Lösungsplänen. Fragen wie *'Welche Operatoren sind für ein bestimmtes Ziel relevant?'* oder *'Welche Vorbedingungen können durch Interaktionen eines bestimmten Operators ihre Gültigkeit verlieren?'* können durch einfache Strukturanalyse des Graphen beantwortet werden. Daher sind Operatorgraphen in vielen Domänen für die in Kapitel 3 vorgestellten Analyseverfahren eine geeignete Abstraktionsebene für Probleme in der entsprechenden Domäne.
- Während jeder Operator aus D höchstens einmal in einem Operatorgraph vorkommt, kann ein Prädikat aus D mehrmals vorkommen, wenn es mehrmals als Vorbedingung von Operatoren vorhanden ist.
- Ein Operatorgraph kann Zykel enthalten. Ist dies der Fall, so wird D als **rekursiv** bezeichnet. Alle Knoten in Zykeln sind **rekursive** Knoten.
- Ein Operatorgraph besitzt eine implizite UND/ODER-Struktur: Die Vorgänger eines Operatorknotens sind UND-verbunden, da **alle** Vorbedingungen gelten müssen, um ihn auszuführen. Die Vorgänger eines Vorbedingungsknotens sind ODER-verbunden, da **ein** Operator genügt, um das entsprechende Prädikat zu erhalten.
- Nur in den Vorbedingunsknoten des FINISH-Operators sind Konstanten vorhanden. Sonst werden bei den Prädikaten die Variablen der zugehörigen Operatoren verwendet.
- Beim 4. Punkt der Definition werden die Typen bei der Unifikation zunächst nicht berücksichtigt. In Kapitel 2.3 werden Erweiterungen der Operatorgraphen vorgestellt, bei denen die Typen bzw. Typ-Constraints und die durch die Unifikation entstehenden lokalen Variablenbindungen berücksichtigt werden.

Da der Operatorgraph keine Lösung darstellt, sondern eine Abstraktion aller potentiellen Lösungspläne, gibt es keine konkreten Causal Links und damit auch keine Interaktionen nach Definition 2.5 in dieser Struktur. Allerdings werden in Operatorgraphen Interaktionen zwischen Operatoren und Prädikaten betrachtet, die wiederum für alle potentiellen Interaktionen in möglichen Lösungsplänen stehen.

Definition 2.8 (Operatorgraph mit Interaktionen) *Ein Operatorgraph OG mit Interaktionen genügt den Bedingungen aus Definition 2.7 und zusätzlich gibt es für jede negative Interaktion zwischen einem Operatorknoten OK und einem Vorbedingungsknoten VK aus OG eine gerichtete Interaktionskante $OK \rightarrow VK$. Zwischen OK und VK besteht in OG eine negative Interaktion, wenn OK einen Effekt besitzt, der die Negation des Prädikates von VK ist.*

Beispiel: In diesem Kapitel entstammen alle Beispiele einer künstlichen Domäne DOM_1 = $(P_{DOM_1}, O_{DOM_1}, T_{DOM_1})$. Diese Domäne hat keine semantische Bedeutung. Sie soll möglichst einfach sein und gleichzeitig alle auftretenden Effekte veranschaulichen. Natürlich werden bei den Tests in Kapitel 5 kompliziertere Domänen verwendet.

Für DOM_1 gilt:

P_{DOM_1} = {*p1(x), p21(x), p22(x), p3(x), Z1(x), Z2(x), Z3(x)*}
O_{DOM_1} = {*Op1(x), Op2(x), Op3(x)*}
T_{DOM_1} = {*typAB, typC*}

Operatoren werden in folgender Notation dargestellt: { *Menge der Vorbedingungen* } *Operatorname* { *Menge der Nachbedingungen* } *[Constraintmenge]*. Dabei werden die negierten Prädikate durch '-' gekennzeichnet.

Die Operatoren aus DOM_1 lassen sich dann so notieren:

{ *p1(x)* }	*Op1(x)*	{ *Z1(x)* }	*[ ]*
{ *p21(x), p22(x)* }	*Op2(x)*	{ *Z2(x), -Z1(x)*}	*[ ]*
{ *p3(x)* }	*Op3(x)*	{ *Z1(x), Z3(x), -Z2(x)* }	*[IsOfType(typC, x)]*

Abbildung 2.1 zeigt den Operatorgraph mit Interaktionen für das Problem $PRO_1 = (S_{PRO_1}$, $Z_{PRO_1}, O_{PRO_1})$ in DOM_1 mit:

S_{PRO_1} = {*p1(a), p1(b), p21(a), p22(a), p3(c)*}
Z_{PRO_1} = {*Z1(a), Z1(b), Z2(a), Z3(c)*}
O_{PRO_1} = {*(typAB: a, b), (typC: c)*}

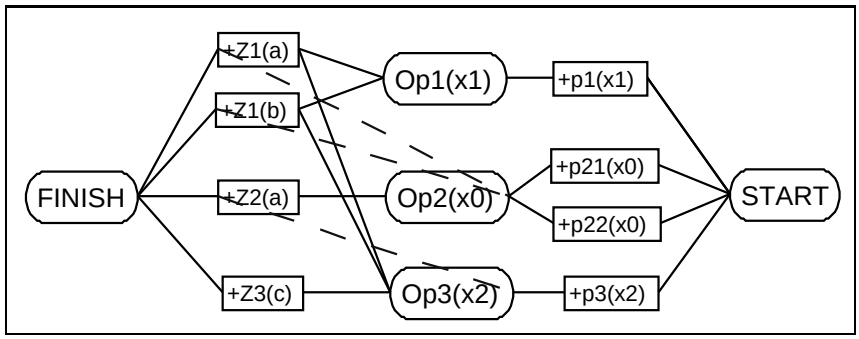

Abbildung 2.1: Operatorgraph für PRO_1

In allen Operatorgraphabbildungen in dieser Arbeit gelten folgende Konventionen: Vorbedingungsknoten sind eckig, Operatorknoten sind abgerundet, Vorbedingungskanten sind durchgezogen und Interaktionskanten sind gestrichelt. Alle auftretenden Variablen werden durchnummeriert, um sie im gesamten Operatorgraph zu unterscheiden, und entsprechen somit nicht immer den Namen in der Domänenbeschreibung.

In Abbildung 2.1 ist z.B. leicht erkennbar, daß das Ziel *Z1(a)* durch die Operatoren *Op1* und *Op3* gelöst und durch *Op2* evtl. wieder ungültig werden kann. Für den Operator *Op2*

ist zu erkennen, daß die beiden Vorbedingungen *p21(x0)* und *p22(x0)* für eine Ausführung gelten müssen und diese *Z2(a)* gültig und *Z1(a), Z1(b)* ungültig machen kann. Der Typ-Constraint von *Op3* spielt hier noch keine Rolle.

Nun werden noch einige Notationen für die Elemente von Operatorgraphen eingeführt. Sei dazu OG der zu einem Problem P gehörende Operatorgraph.

- Mit ***OK(OG)*** wird die Menge aller Operatorknoten und mit ***VK(OG)*** die Menge aller Vorbedingungsknoten von OG bezeichnet. Mit ***VE(OG)*** wird die Menge aller Vorgängerkanten und mit ***IE(OG)*** die Menge aller Interaktionskanten in OG bezeichnet.
- Alle Knoten werden mit ***K(OG)*** $= OK(OG) \cup VK(OG)$ und alle Kanten mit ***E(OG)*** $= VE(OG) \cup IE(OG)$ bezeichnet.
- Mit $\mathbf{VOR_{OG}(X)} = \{K \in K(OG) \mid \exists E \in VE(OG) : E = K \rightarrow X\}$ wird die Menge aller direkten Vorgänger von X in OG bezeichnet.

Eliminierung von Interaktionen

In Operatorgraphen können Interaktionskanten enthalten sein, für die es in keiner Lösung entsprechende Interaktionen gibt. In [Smith und Peot, 1993] werden Bedingungen angegeben, unter denen potentielle Interaktionen in Operatorgraphen nicht berücksichtigt werden müssen, da entsprechende Interaktionen in Lösungen nicht auftreten können. Dazu werden drei Theoreme angegeben, die solche Situationen beschreiben. Diese können dazu eingesetzt werden, Operatorgraphen weiter zu vereinfachen und auf die wesentlichen Interaktionen zu fokussieren. Bei der Implementierung der Operatorgraphen für diese Arbeit werden die Theoreme aus [Smith und Peot, 1993] angewendet. Dies beeinflußt zwar nicht die eigentlichen Ergebnisse, hat aber eine Effizienzsteigerung zur Folge. Allerdings ist die Formulierung von *Theorem 3* in [Smith und Peot, 1993] nicht korrekt, denn zwei Zweige sind nur dann disjunktiv, wenn **alle** nahesten gemeinsamen Vorgänger (es kann ja mehrere Wege Richtung FINISH geben) Vorbedingunsknoten sind. Eine genaue Betrachtung dieses Problems mit Beispiel befindet sich in Anhang A.1.

2.3 Erweiterungen von Operatorgraphen

Die oben vorgestellten Operatorgraphen entsprechen den schon in [Smith und Peot, 1993] erwähnten. Sie werden nun in zwei Stufen erweitert. Zuerst werden Typ-Constraints und Ordnungsrelevanzen berücksichtigt, dann werden Bindungen von Konstanten an Variablen betrachtet.

2.3.1 Typ-Constraints und Ordnungsrelevanzen

Sowohl Typ-Constraints als auch Ordnungsrelevanzen werden eingesetzt, um die Operatorgraphen zu kompaktifizieren und auf das wesentliche zu beschränken. Während Typ-

Constraints schon in der Domänenbeschreibung enthalten sind, stellen Ordnungsrelevanzen zusätzliches Wissen über die Domäne dar.

Definition 2.9 (Operatorgraph mit Typ-Constraints) *Ein Operatorgraph, der Typ-Constraints berücksichtigt, genügt der Definition 2.8, wobei aber die Typ-Constraints der Operatoren wie folgt berücksichtigt werden:*

- *Die Variablen in den Prädikaten der Vorbedingungsknoten übernehmen die zugehörigen Typ-Constraints der Operatoren, von denen sie die Variablen übernommen haben.*
- *Vorbedingungsknoten, die Konstanten enthalten (das sind genau die Ziele) übernehmen die Typen der Konstanten.*
- *Werden für einen Vorbedingungsknoten VK alle Operatoren gesucht, die einen Effekt besitzen, welcher sich mit dem Prädikat von VK unifizieren läßt, so werden dabei die Typ-Constraints von VK und die Typ-Constraints des jeweiligen Operators OK berücksichtigt. Nur wenn diese Constraint-Menge konsistent ist, wird OK und die Vorbedingungskante $VK \rightarrow OK$ in OG eingefügt.*
- *Interaktionskanten $OK \rightarrow VK$ werden nur dann eingefügt, wenn die Typ-Constraints von VK und OK konsistent sind.*

Beispiel: Abbildung 2.2 zeigt den zu Problem PRO_1 gehörenden Operatorgraph mit Typ-Constraints.

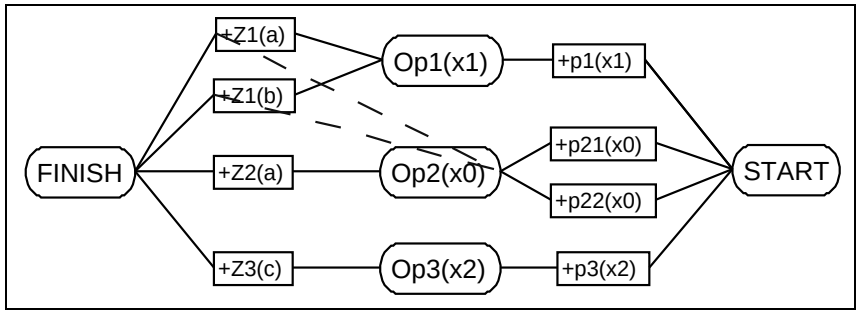

Abbildung 2.2: Operatorgraph mit Typ-Constraints

Anders als beim einfachen Operatorgraph in Abbildung 2.1 fehlen hier die Vorbedingungskanten *Op3* $\rightarrow$ *Z1(a)* und *Op3* $\rightarrow$ *Z1(b)*. Dies spiegelt die Tatsache wieder, daß in diesem Problem *Z1(a)* bzw. *Z1(b)* nicht mit *Op3* gelöst werden können, da *x2* von *Op3* wegen Typ-Constraints nicht an *a* bzw. *b* gebunden werden kann. Operatorgraphen sollen ja eine Abstraktion aller möglichen Lösungen sein. Diese Eigenschaft bleibt bei Berücksichtigung von Typ-Constraints erhalten, bzw. wird noch besser erreicht, da ja nur Kanten und evtl. auch Knoten weggelassen werden, die in Lösungen sowieso nicht auftreten können, da die Typ-Constraints dort auch immer berücksichtigt werden müssen. Knoten würden im obigen Beispiel wegfallen, wenn es z.B. bei PRO_1 das Ziel *Z3(c)* nicht gäbe. Weiterhin fehlt

die Interaktionskante *Op3* $\rightarrow$ *Z2(a)*, da ebenfalls *x2* in *Op3* wegen Typ-Constraints nicht an *a* gebunden werden kann. Auch hierbei gilt, daß diese potentielle Interaktion in keiner Lösung auftreten kann.

Im Gegensatz zu Typ-Constraints stellen Ordnungsrelevanzen zusätzliches Wissen über eine Domäne dar. In *CAPlan* können Prädikate in Domänenbeschreibungen als nicht-ordnungsrelevant markiert werden. Damit kann Metawissen über die Relevanz einzelner Prädikate bezüglich Zielordnungen in der betrachteten Domäne modelliert werden.

Definition 2.10 (Ordnungsrelevanter Operatorgraph) *Ein Ordnungsrelevanter Operatorgraph genügt der Definition 2.9, wobei die Expansion von Operatorknoten wie folgt abgeändert wird: Gibt es einen Operatorknoten OK in OG, so gibt es für jede Vorbedingung des Operators von OK, die ordnungsrelevant ist, einen Vorbedingungsknoten VK mit einer gerichteten Vorgängerkante* VK $\rightarrow$ OK.

Beispiel: Wenn zusätzliches Wissen über DOM_1 besteht, welches aussagt, daß *Z3(x)* als nicht-ordnungsrelevant einzustufen ist, dann sieht der Ordnungsrelevante Operatorgraph für PRO_1 wie in Abbildung 2.3 aus.

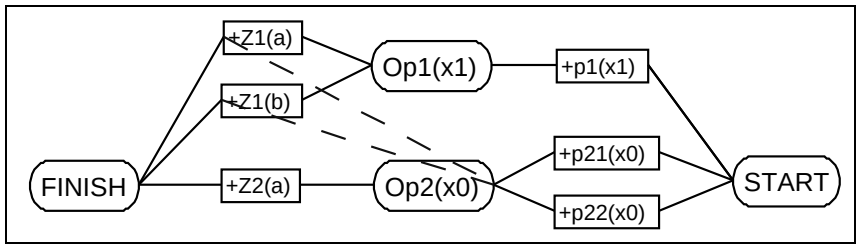

Abbildung 2.3: Ordnungsrelevanter Operatorgraph

Dabei verschwindet der Zweig mit *Op3* total. Dieser Operatorgraph fokussiert auf den ordnungsrelevanten Teil von PRO_1 in DOM_1. Dazu ist zu sagen, daß das zusätzliche Wissen in Form von Ordnungsrelevanzen, wenn es korrekt ist, die grundsätzlichen Ergebnisse der Analyseverfahren in Kapitel 3 nicht beeinflußt. Es dient lediglich der Übersichtlichkeit bei der Betrachtung der Operatorgraphen und einer Effizienzsteigerung bei der Analyse. Dies gilt, da, wenn das Wissen korrekt ist, nur Teile weggelassen werden, die für Zielordnungen irrelevant sind. Dagegen können die Analyseverfahren durch Berücksichtigung von Typ-Constraints echt mächtiger werden. Dies zeigt sich später in Kapitel 3. Im folgenden werden meistens ordnungsrelevante Operatorgraphen betrachtet, da es in dieser Arbeit primär um Zielordnungen geht, allerdings hat die Berücksichtigung von Ordnungsrelevanzen nur dann Auswirkungen, wenn überhaupt entsprechendes Wissen über die Domäne vorhanden ist. In Domänen mit vielen verschiedenen Typen, wie z.B. bei der Workpiece-Domäne, kann die Berücksichtigung von Typen und Typ-Constraints entscheidende Auswirkungen haben.

2.3.2 Variablenbindungen

Der wichtigste Teil der Analyse ist das Erkennen von notwendigen Zielordnungen. Diese können z.B. durch Interaktionen entstehen. Nun wird gezeigt, daß die oben definierten Operatorgraphen ein noch zu abstraktes Abbild von Problemen sind, um solche Ordnungen im allgemeinen zu garantieren.

Bei der Betrachtung des Ordnungsrelevanten Operatorgraphen von PRO_1 (siehe Abbildung 2.3) fällt auf, daß die beiden Ziele *Z1(a)* und *Z1(b)* gleichartig in die Struktur des Operatorgraphen eingebunden sind. Beide haben eine Vorbedingungskante zu *Op1* und eine Interaktionskante zu *Op2*. Nun gilt für alle Lösungen von PRO_1 und damit für das Problem selber aber folgendes: Irgendwann muß *Op2* für die Lösung von *Z2(a)* mit der Variablenbindung *(x0 = a)* ausgeführt werden, da es die einzige Möglichkeit ist, dieses Ziel zu lösen. Durch die negative Interaktion von *Op2* zu *Z1(a)* gilt *Z1(a)* nachher auf keinen Fall mehr, während *Z1(b)*, wenn es vorher schon galt, erhalten bleibt. Daraus folgt, daß *Z1(a)* auf jeden Fall erst nach *Z2(a)* gelöst werden kann, während *Z1(b)* auch vorher lösbar ist. Also gilt in PRO_1 die notwendige Zielordnung *Z2(a)* $<_Z$ *Z1(a)*, aber es gilt nicht *Z2(a)* $<_Z$ *Z1(b)*, obwohl, wie oben erwähnt, *Z1(a)* und *Z1(b)* bei einer Strukturanalyse gleiche Eigenschaften hätten. Um den entscheidenden Unterschied zu erkennen, fehlt offensichtlich die Information, an welche Konstanten die Variablen in den einzelnen Situationen gebunden werden können. Das Problem liegt darin, daß ein Operatorknoten für alle möglichen Anwendungen des entsprechenden Operators in allen möglichen Lösungen steht. Um mehr Informationen über Variablenbindungen bei den einzelnen Operatoranwendungen zu haben, werden nun die Operatorgraphen erweitert.

Definition 2.11 (Variablenbindung, Bindungsliste) *Die* **Variablenbindung** *VB einer Variablen V ist entweder eine Menge von Konstanten, welche Typen besitzen, die konsistent mit den evtl. vorhandenen Typ-Constraints bezüglich V sind, oder eine weitere Variable. Eine* **Bindungsliste** *BL ist eine Liste von Variablen, bei der jeder Variablen eine Variablenbindung zugeordnet wird.*
Notation: VB(V, BL) *bezeichnet die der Variablen V zugeordnete Variablenbindung in BL (also eine Konstantenmenge oder eine Variable).*

Wenn *a, b, c, d* Konstanten und *w* eine Variable sind, dann könnte eine Bindungsliste *BL* für die Variablen *x, y, z* z.B. so aussehen: *BL* = {*(x:* {*a, b, d*}*), (y:* {*w*}*), (z:* {*b, c*}*)*} und dann wäre *VB(x, BL) = (x:* {*a, b, d*}*)*.

Definition 2.12 (Operatorgraph mit Konstantenbeschriftung) *Ein Operatorgraph mit Konstantenbeschriftung genügt den Bedingungen aus Definition 2.10. Zusätzlich wird ein Teil der Kanten und Knoten wie folgt beschriftet:*

1. *Jede Interaktionskante IE = OK* $\rightarrow$ *VK wird mit einer Bindungsliste* BL_{IE} *beschriftet. Sie enthält für jede Variable V von OK eine Variablenbindung, die eine Konstante oder eine Variable enthält, an die V gebunden werden muß, damit der entsprechende Effekt von OK mit VK unifiziert werden kann. Ist VK ein Ziel, so ist die jeweilige Variablenbindung eine Menge mit der entsprechenden Konstante des Zieles, sonst ist die Zuordnung immer eine andere Variable.* **Notation: BL(IE) =** BL_{IE}

2. *Jeder Operatorknoten OK wird mit einer Konstantentabelle beschriftet.*
   Notation: KT(OK) *bezeichnet die zu OK gehörige Konstantentabelle.*

Während die Beschriftung der Interaktionskanten festhält, welche Bedingungen gelten müssen, damit die potentielle Interaktion eine tatsächliche ist, sollen die Konstantentabellen Auskunft darüber geben, an welche Konstanten die einzelnen Variablen der Operatoren gebunden werden können, wenn sie für die Lösung eines bestimmten Zieles ausgeführt werden. Die möglichen Bindungen müssen getrennt für alle Ziele betrachtet werden, damit z.B. bei der Frage: 'Gilt $Z_1 <_Z Z_2$?' bei einer bestimmten Interaktion erkannt werden kann, ob die nötigen Bindungen bei diesen beiden Zielen möglich oder auch zwingend sind. Die Begriffe *möglich* und *zwingend* werden in Definition 2.15 formal beschrieben und werden in den Kriterien in Kapitel 3 angewendet.

Definition 2.13 (Potentielle Variablenbindung) *Die potentielle Variablenbindung einer Variablen V eines Operatorknotens OK bezüglich eines Zieles Z* **PotVB(V, OK, Z)** *ist eine Variablenbindung für V mit der Menge aller Konstanten, an die V gebunden werden kann, wenn der Operator von OK für die Lösung von Z ausgeführt wird. Die potentielle Bindungsliste für einen Operatorknoten OK bezüglich eines Zieles Z* **PotBL(OK, Z)** *ist eine Bindungsliste, die für jede Variable V von OK PotVB(V, OK, Z) enthält.*

Bemerkung: Wenn ein Operator O für die Lösung eines Zieles Z ausgeführt wird, so heißt das, daß dieser Operator dabei einen Effekt besitzt, der entweder direkt Z selber ist, oder der eine Vorbedingung eines Operators ist, welcher wiederum für die Lösung von Z ausgeführt wird. Somit kann beim Problem PRO_1 (siehe dazu Abbildung 2.3) der Operator *Op2(x0)* für die Lösung des Zieles *Z2(a)* nur mit der Variablenbindung *(x0 = a)* ausgeführt werden, aber nicht mit der Variablenbindung *(x0 = b)*. Dagegen kann *Op1(x1)* gar nicht für die Lösung von *Z2(a)* ausgeführt werden.

Notation: Wird ein Operator O für die Lösung eines Zieles Z ausgeführt, so wird dies mit **Z|O** bezeichnet.

Definition 2.14 (Konstantentabelle) *Eine Konstantentabelle KT für einen Operatorknoten OK in einem Operatorgraphen OG ist eine Tabelle, die für jedes Ziel Z des FINISH-Knotens von OG die potentielle Bindungsliste BL_Z = PotBL(OK, Z) enthält.*
Notation: BL(KT, Z) *bezeichnet BL_Z.*

Beispiel: Die Konstantentabelle für *Op1* im Operatorgraph mit Konstantenbeschriftung für PRO_1 (siehe Abbildung 2.3) sieht wie folgt aus: *KT(Op1) = {(Z1(a): {(x1: {a})}), (Z1(b): {(x1: {b})})}* und die Variablenbindung von *x1* in *Op1* bezüglich *Z1(b)* sieht so aus: *VB(x1, BL(KT(Op1), Z1(b))) = (x1:{b})*. Die Beschriftung der Interaktionskante *IE = Op2(x0) → Z1(a)* sieht so aus: BL_{IE} = *{(x0: {a})}*.

Mit dem Operatorgraphen mit Konstantenbeschriftung kann z.B. für PRO_1 die Ordnung $Z2(a) <_Z Z1(a)$ berechnet werden. Dies wird in Kapitel 3 mit dem Zerstörungskriterium gezeigt. Bei der Formulierung der Kriterien werden weiterhin folgende Eigenschaften benutzt:

Definition 2.15 (Implikation, Teilmenge, Verknüpfung von Bindungslisten) *Für zwei Bindungslisten BL_1, BL_2 sind folgende Beziehungen definiert:*

Implikation: *BL_1 impliziert BL_2 ($BL_1 \Rightarrow BL_2$), wenn für alle Variablen V in BL_2 gilt: Es gibt in dem betrachteten Problem nur eine Konstante, die einen mit den evtl. zu V vorhandenen Typ-Constraints konsistenten Typ besitzt, oder es gilt:*
$(V \in BL_1) \wedge |VB(V, BL_1)| = 1 \wedge |VB(V, BL_2)| = 1 \wedge VB(V, BL_1) = VB(V, BL_2)$.

Teilmenge: *BL_1 ist Teilmenge von BL_2 ($BL_1 \subseteq BL_2$), wenn für alle Variblen V in BL_1 gilt: $(V \in BL_2) \wedge VB(V, BL_1) \subseteq VB(V, BL_2)$.*

Verknüpfung: *Die Verknüpfung von BL_1 und BL_2 ($BL_1 \odot BL_2$) ist eine Bindungsliste für alle Variablen von BL_2 mit folgenden Zuordnungen: Für jede Variable, deren Variablenbindung in BL_2 eine Menge von Konstanten ist, bleibt die Zuordnung erhalten. Für jede Variable, deren Zuordnung eine weitere Variable V_2 ist, wird als Zuordnung $VB(V_2, BL_1)$ übernommen.*

$BL_1 \Rightarrow BL_2$ bedeutet anschaulich, daß BL_1 eine Situation von Variablenbindungen beschreibt, in der auch alle Bindungen von BL_2 zwingend gelten. Beispiel: Wenn $BL_1 = \{(x: \{a, b\}), (y: \{c\})\}$ und $BL_2 = \{(y: \{c\})\}$, dann gilt $BL_1 \Rightarrow BL_2$, jedoch gilt nicht $BL_2 \Rightarrow BL_1$.

Dagegen besagt $BL_1 \subseteq BL_2$, daß BL_1 eine Situation von Variablenbindungen beschreibt, die in BL_2 auch möglich ist. Beispiel: Wenn $BL_1 = \{(x: \{a\})\}$ und $BL_2 = \{(x: \{a, b\})\}$, dann gilt $BL_1 \subseteq BL_2$, jedoch gilt weder $BL_1 \Rightarrow BL_2$ noch $BL_2 \Rightarrow BL_1$.

Bei der Verknüpfung, welche nicht kommutativ ist, wird die Transitivität der Gleichheit ausgenutzt. Beispiel: Wenn $BL_1 = \{(w: \{a\}), (z: \{c\})\}$ und $BL_2 = \{(x: \{a, b\}), (y: \{z\})\}$, dann gilt $BL_1 \odot BL_2 = \{(x: \{a, b\}), (y: \{c\})\} \neq BL_2 \odot BL_1 = BL_1$. In diesem Beispiel wird also aus $y = z$ und $z = c$ geschlossen, daß $y = c$ gilt.

Definition 2.16 (Negative Interaktion bezüglich Z) *Sei OG der zu einem Problem P gehörende Operatorgraph mit Konstantenbschriftung. Seien $OK \in OK(OG)$ und Z_1, Z Ziele in P. Gibt es in jedem Fall eine negative Interaktion von OK nach Z_1 in OG mit konsistenten Variablenbindungen für Z, so wird dies mit* **NegInt(OK, Z_1, Z, OG)** *bezeichnet, kann es eine solche Interaktion geben, so wird dies mit* **PotNegInt(OK, Z_1, Z, OG)** *bezeichnet. Dabei gilt: NegInt(OK, Z_1, Z, OG) gdw*

$$\exists IK \in IK(OG) : IK = OK \rightarrow Z_1 \quad \wedge \quad BL(KT(OK), Z) \Rightarrow BL(IE)$$

Und für potentielle Interaktionen gilt: PotNegInt(OK, Z_1, Z, OG) gdw

$$\exists IK \in IK(OG) : IK = OK \rightarrow Z_1 \quad \wedge \quad BL(IE) \subseteq BL(KT(OK), Z)$$

Bemerkung: Die Variablenbindungen in *BL(IE)* sind hier immer Mengen mit je einer Konstanten, da *IE* eine Interaktion zu einem Ziel ist. *PotNegInt(OK, Z_1, Z, OG)* bedeutet, daß nicht nur eine potentielle Interaktion bei irgendeiner Ausführung von *OK* nach Z_1

möglich ist, sondern daß bei Ausführung von *OK* für *Z* Variablenbindungen entstehen können, bei denen nachher Z_1 nicht mehr gilt. Dagegen gilt bei *NegInt(OK, Z_1, Z, OG)* nach Ausführung von *OK* für *Z* in jedem Fall Z_1 nicht mehr. Diese Eigenschaften werden für die Kriterien in Kapitel 3.2 benötigt.

In Kapitel 3.1 werden Operatorblockaden behandelt, welche sowohl selber ein Ergebnis der Analyse darstellen, als auch für die Berechnung der Zielordnungen benutzt werden.

Definition 2.17 (Operatorblockade) *Ein Operator O einer Domäne D ist für ein Ziel Z eines Problems P blockiert, wenn O bei der Lösung von P in D für das Lösen von Z nicht ausgeführt werden kann. Das heißt, daß O zwar evtl. wegen vorhandener Vorbedingungen anwendbar ist, allerdings nicht so, daß dabei ein für die Lösung von Z nützlicher Effekt entsteht. Ist dies der Fall, so wird das mit* **Block(O, Z)** *bezeichnet.*

Nun sind alle Notationen für das folgende Kapitel eingeführt. Es gibt aber für rekursive Domänen noch einige Besonderheiten.

2.4 Rekursive Domänen

In dieser Arbeit werden auch beliebige rekursive Domänen betrachtet. Dies hat sowohl bei der Berechnung von Operatorgraphen mit Konstantenbeschriftung, als auch bei den folgenden Analyseverfahren eine hohe Komplexität im theoretischen Sinne (Berechnungsaufwand) und im praktischen Sinne (Implementierungsaufwand) zur Folge. In rekursiven Domänen ist die Berechnung der potentiellen Variablenbindungen oft extrem aufwendig. Somit macht es Sinn, bei einer Implementierung der in dieser Arbeit vorgestellten Verfahren, die potentielle Variablenbindung für rekursive Teile der Domänen und damit für Zykel in Operatorgraphen, nach oben abzuschätzen. Später wird gezeigt, daß die Analyseverfahren dann zwar für diese Teile evtl. nicht mehr so mächtig aber immer noch korrekt sind.

2.4.1 Darstellung rekursiver Operatorgraphen

In diesem Kapitel wird eine rekursive Domäne $DOM_2 = (P_{DOM_2}, O_{DOM_2}, T_{DOM_2})$ betrachtet, für die gilt:

P_{DOM_2} = {*p(x), q1(x), q2(x), Z(x)*}
O_{DOM_2} = {*Op1(x, y), Op2(x), Op3(x, y)*}
T_{DOM_2} = {*typAB, typC*}

Die Operatoren sind wie folgt definiert:

{ *q1(x)* }	*Op1(x, y)*	{ *Z(y)* , *q2(y)* }	[*NotSame(x, y), IsNotOfType(typC, x)*]
{ *q2(x)* }	*Op2(x)*	{ *Z(x), q1(x)*}	[ ]
{ *p(x)* }	*Op3(x, y)*	{ *q1(y)* }	[*NotSame(x, y)*]

Nun wird ein Problem $PRO_2 = (S_{PRO_2}, Z_{PRO_2}, O_{PRO_2})$ in DOM_2 betrachtet, für das gilt:

$$\begin{aligned} S_{PRO_2} &= \{p(b)\} \\ Z_{PRO_2} &= \{Z(a)\} \\ O_{PRO_2} &= \{(typAB:\ a,\ b),\ (typC:\ c)\} \end{aligned}$$

Um Operatorgraphen mit Zykeln gut darzustellen, werden rekursive Kanten gepunktet dargestellt und mit einem Pfeil versehen, da sie in die entgegengesetzte Richtung aller anderen Kanten zeigen. Rekursive Vorbedingungskanten $OK \rightarrow VK$ sind Kanten, bei denen VK ein direkter oder indirekter Vorgänger von OK ist. Je nachdem, in welcher Reihenfolge ein rekursiver Operatorgraph aufgebaut wird, können die rekursiven Kanten eines Zykels an unterschiedlichen Stellen sitzen. Für PRO_2 ergeben sich zwei Möglichkeiten. Die rekursive Kante kann zwischen *Op2* und *q1* liegen oder zwischen *Op1* und *q2*. Abbildung 2.4 zeigt die erste Variante. Beide Varianten sind für die in Kapitel 3 folgenden Analyseverfahren äquivalent.

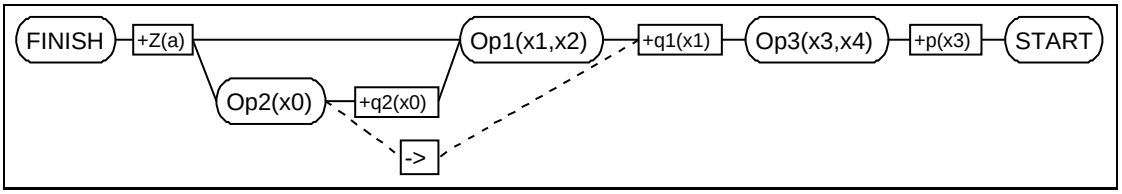

Abbildung 2.4: Operatorgraph für PRO_2

Zu beachten ist, daß die rekursive Kante nicht etwa bedeutet, daß *q1* eine Vorbedingung von *Op2* ist, sondern daß *q1* auch mit *Op2* gelöst werden kann.

2.4.2 Variablenbindungen in rekursiven Zweigen

Dieses Beispiel zeigt die Schwierigkeit der Berechnung der potentiellen Variablenbindungen für rekursive Zweige. Dazu betrachte man den minimalen Plan (dies ist die einzige Lösung mit vier Schritten und diese Tatsache ist nicht trivial) zu PRO_2 in Abbildung 2.5. Als Begründung dafür, daß dies die einzige Lösung mit vier Operatoranwendungen ist und daß es keine Lösung mit weniger Schritten gibt, ist folgende Deutung der Domäne möglich: Man betrachte die Kette *p(x3), Op3(x3, x4), q1(x1), Op1(x1, x2), q2(x0), Op2(x0)*. Für die Operatoren *Op3* und *Op1* gilt, daß x jeweils die Eingabevariable und y die Ausgabevariable darstellt (x und y sind die Namen aus der Definition der Domäne). Durch den *NotSame-Constraint* wechseln sie somit zwischen den Konstanten a und b (c ist wegen *Typ-Constraint* nicht möglich) hin und her. Für *Op2* gilt dies nicht. Um aus *p(b)* das Ziel *Z(a)* zu gewinnen, muß also eine ungerade Anzahl von Operatoranwendungen von *Op3* und *Op1* erfolgen. Demnach muß der Zykel mit *Op2* und *Op1* mindestens einmal durchlaufen werden.

Für die potentiellen Variablenbindungen tritt bei diesem Problem folgender interessante Fall ein: Obwohl es nur das Ziel *Z(a)* gibt, wird *Op1* einmal mit der Variablenbindung *(x2 = a)* und einmal mit der Variablenbindung *(x2 = b)* ausgeführt. Diese Notwendigkeit

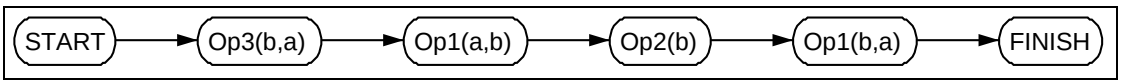

Abbildung 2.5: Plan für PRO_2

resultiert aus den Constraints der Operatoren. Dieses Beispiel läßt erahnen, daß rekursive Domänen formuliert werden können, in denen die Entscheidung, ob nun bestimmte Konstanten in einer potentiellen Variablenbindung einer bestimmten Variable enthalten sind, extrem kompliziert werden kann. Will man diese genau berechnen, so würde man das Problem in einer gewissen abstrakten Sicht schon lösen. Dies widerspricht nicht nur der Idee einer Vorverarbeitung, sondern kann auch sehr ineffizient sein. Wie oben erwähnt, ist daher eine Abschätzung sinnvoll. Eine einfache Abschätzung der potentiellen Variablenbindungen ist die Menge aller Konstanten des Problemes, welche Typen haben, die konsistent zu den Typ-Constraints der entsprechenden Variablen sind. Somit kann *x1* z.B. nur an *a* oder *b* gebunden werden.

Nun wird ein Verfahren beschrieben, welches die potentiellen Konstantenbelegungen in rekursiven Zweigen etwas besser abschätzt. Bei diesem implementierten Verfahren werden zwei Arten von Variablen in rekursiven Zweigen unterschieden, nämlich rekursive und nichtrekursive Variablen.

Definition 2.18 (rekursive/nicht-rekursive Variablen) *Eine Variable V eines Operatorknotens OK in OG heißt* **rekursiv**, *wenn OK in einem Zykel liegt und wenn V an unterschiedliche Konstanten gebunden werden kann, wenn der Zykel mehrmals durchlaufen wird. Alle anderen Variablen heißen* **nicht-rekursiv**.

Beispiel: Der Operator *DriveTruck* aus der erstmals in [Veloso, 1992] vorgestellten *Logistic-Transportation-Domäne* läßt sich so notieren:

{ *atTruck(truck, locFrom), locationAt(locFrom, city), sameCity(locFrom, locTo),*
truckInCity(truck, city), locationAt(locTo, city) }
DriveTruck(truck, locFrom, locTo, city)
{ *atTruck(truck, locTo), -atTruck(truck, locFrom)* }
[*IsOfType(Truck, truck), IsOfType(Location, locFrom, locTo), IsOfType(City, city),*
NotSame(locFrom, locTo)]

Das Prädikat *atTruck* kommt hier sowohl als Vorbedingung als auch als Nachbedinung vor. Dadurch entsteht in jedem Operatorgraph für Probleme aus der *LogisticTransportation-Domäne* ein kleiner Zykel mit *DriveTruck* und *atTruck*. Wird dieser Zykel mehrmals durchlaufen, so können die Variablen *locFrom* und *locTo* an unterschiedliche Konstanten gebunden werden und sind somit **rekursiv**. Die Variable *truck* kann zwar, wenn mehrere Objekte des Typs *Truck* in dem betrachteten Problem vorhanden sind, auch an unterschiedliche Konstanten gebunden werden, allerdings bleibt die Konstantenbindung bei mehrmaligem durchlaufen des Zykels immer die gleiche. Somit ist *truck* **nicht-rekursiv**.

Zusammenfassend werden nun die Berechnung und die Abschätzung von potentiellen Konstantenbelegungen beschrieben:

Abschätzung: Die Konstantenbelegungen einer Variablen V wird durch die Menge aller Konstanten des Problems, welche Typen haben, die konsistent zu den Typ-Constraints von V sind, abgeschätzt.

Berechnung: Die Konstantenbelegung einer Variablen V wird durch Konstantenpropagierung berechnet. Dabei werden die Konstanten des START- und des FINISH-Knotens über die Operatoren propagiert. Unter Berücksichtigung der UND-/ODER-Struktur des Operatorgraphen kann damit für nicht-rekursive Variablen $PotVB$ aus Definition 2.13 berechnet werden.

Damit lassen sich nun zwei Verfahren zur Bestimmung der potentiellen Konstantenbelegung beschreiben:

Einfaches Verfahren: Konstantenbelegungen von allen Variablen in Zykeln werden abgeschätzt, alle anderen werden berechnet.

Verbessertes Verfahren: Konstantenbelegungen von rekursiven Variablen werden abgeschätzt, alle anderen werden berechnet.

Im folgenden Kapitel werden nun einige Analyseverfahren vorgestellt, die auf Operatorgraphen mit Konstantenbeschriftung basieren.

Kapitel 3

Analyseverfahren mit Operatorgraphen

Die Analyseverfahren, die in diesem Kapitel vorgestellt werden, sind als Vorverarbeitung für ein neues, zu lösendes Problem zu betrachten, bei der einige Eigenschaften des Problems analysiert werden. Anschließend wird dem Planungssystem das Problem mit den analysierten Eigenschaften übergeben. Ziel dabei ist es, daß dieses durch die berechneten Zusatzinformationen effizienter planen kann. Wie die Analyseergebnisse im einzelnen genutzt werden, wird in Kapitel 4 beschrieben.

In dieser Arbeit werden zwei Arten von Eigenschaften untersucht, **Operatorblockaden** und **notwendige Zielordnungen**, wobei sich letztere besser berechnen lassen, wenn die Operatorblockaden bereits untersucht wurden. Im folgenden bezeichnet OG den Operatorgraphen mit Konstantenbeschriftung für das jeweils betrachtete Problem.

3.1 Operatorblockaden

Für Operatorblockaden wird ein Problem $PRO_3 = (S_{PRO_3}, Z_{PRO_3}, O_{PRO_3})$ in der Domäne DOM_1 betrachtet, dabei gilt:

S_{PRO_3} = {*p1(a), p21(a), p22(a), p3(c)*}
Z_{PRO_3} = {*Z1(a), Z1(b), Z3(c)*}
O_{PRO_3} = {*(typAB: a, b), (typC: c)*}

In einem Operatorgraph müssen nicht immer alle Operatoren der Domäne enthalten sein. Beim Problem PRO_3 z.B. gibt es keinen Operatorknoten für *Op2* im Operatorgraph (siehe dazu Abbildung 3.1). Diese Tatsache wird bereits in Definition 2.7 berücksichtigt und wird hier nicht als Operatorblockade bezeichnet. Hier geht es darum, daß ein Operator zwar in dem Operatorgraph enthalten ist, aber für bestimmte Ziele des Problems nicht mit Erfolg angewendet werden kann. Wenn man solche Blockaden in der Analyse vorausberechnet, kann diese Information an zwei Stellen ausgenutzt werden. Zum einen für die weitere Analyse, denn bei den Kriterien für Zielordnungen müssen Operatoren, die für die entsprechenden Ziele blockiert sind, nicht berücksichtigt werden. Das hat nicht nur eine Effizienzsteigerung

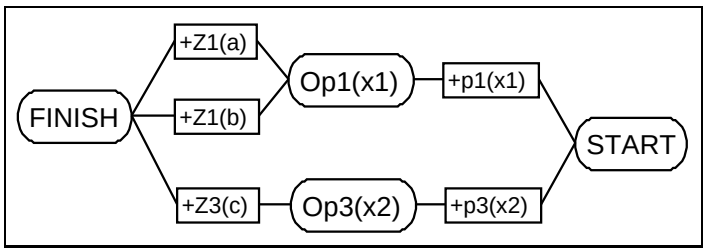

Abbildung 3.1: Operatorgraph für PRO_3

zur Folge, sondern teilweise können bestimmte Ordnungen nur so erkannt werden. Zum zweiten können Operatorblockaden bei der Planung selber berücksichtigt werden, wie es in Kapitel 4 näher beschrieben wird.

Es gibt zwei verschiedene Gründe, aus denen Operatorblockaden resultieren können, Variablenbindungen oder Interaktionen.

3.1.1 Operatorblockaden wegen Variablenbindungen

Wenn in einem Operatorgraph bei einem Problem P für eine Variable V eines Operatorknotens O die Menge der potentiellen Variablenbindungen bezüglich eines Ziels Z leer ist, so ist O für die Lösung von Z nicht erfolgreich anwendbar, denn in jeder Lösung von P muß V, wenn O für die Lösung von Z ausgeführt wird (siehe Bemerkung zu Definition 2.13), an eine Konstante gebunden werden. Wenn es eine Konstante gäbe, mit der Z gelöst werden kann, so wäre die potentielle Variablenbindung nicht leer.

Theorem 3.1 (Operatorblockade wegen Variablenbindung) *Sei O ein Operatorknoten in OG, V_O die Menge der Variablen von O und Z ein Ziel in P, dann gilt:*

$$\exists V \in V_O : VB(V, BL(KT(O), Z)) = \emptyset \qquad \Rightarrow \qquad Block(O, Z)$$

Beispiel: In S_{PRO_3} gibt es kein *p1(b)*, deshalb sieht die Konstantentabelle für *Op1* so aus: {*(Z1(a):{(x1:{a})}),(Z1(b):{(x1:{})})*}. Der Struktur des Operatorgraphen sieht man nicht an, daß *Op1* für die Lösung von *Z1(b)* nicht erfolgreich anwendbar ist. Mit Theorem 3.1 kann die Operatorblockade erkannt werden. Bei den in Kapitel 3.2 aufgeführten Kriterien muß, wenn man die Lösungsmöglichkeiten von *Z1(b)* betrachtet, *Op1* also nicht berücksichtigt werden. Bei diesem Beispiel ist *Z1(b)* sogar völlig unlösbar, siehe dazu Kapitel 3.3.

Bemerkung: Wenn für rekursive Zweige die potentielle Variablenbindung nach oben abgeschätzt wird, kann es passieren, daß eigentlich leere Konstantenmengen fälschlich Konstanten enthalten, aber nicht umgekehrt. Also können keine Fehler entstehen (Erkennung von Operatorblockaden, die keine sind), aber es kann passieren, daß Blockaden dadurch nicht erkannt werden.

3.1.2 Operatorblockaden wegen Interaktionen

Ein Operator kann für ein Ziel blockiert sein, wenn sich Interaktionen so überlagern, daß diese nicht alle gelöst werden können. In den erstmals in [Barrett und Weld, 1994] vorgestellten $\theta_k D^m S^n$ - Domänen (Definition in Anhang A.3.3) gibt es für $k \geq 2$ einfache Beispiele mit diesem Effekt. Abbildung 3.2 zeigt den Operatorgraph für ein einfaches Problem aus dieser Domäne mit $k = 2$ und $m = n = 1$. Dabei kann *G1* entweder durch *A1-a* oder *A1-b* gelöst werden. Wenn *G1* durch *A1-b* gelöst werden soll, kann *G-a* nicht mehr gelöst werden, denn dafür müßte auf jeden Fall *A-a* ausgeführt werden. Von *A-a* aus gibt es jedoch negative Interaktionen sowohl zu einer Vorbedingung, als auch zu dem benötigten Effekt von *A1-b*. Daher müßte *A-a* wegen der einen Interaktion hinter *A1-b* und wegen der anderen Interaktion vorher ausgeführt werden. Da dies nicht möglich ist, ist *A1-b* für *G1* blockiert. Somit muß *G1* mit *A1-a* gelöst werden.

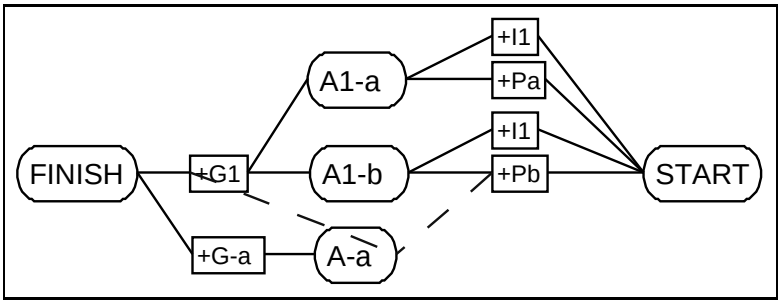

Abbildung 3.2: Operatorgraph für ein Problem in der $\theta_2 D^1 S^1$ - Domäne

Nach der Berechnung der Operatorblockaden können nun Zielordnungen besser erkannt werden, da blockierte Operatoren für die jeweiligen Ziele nicht berücksichtigt werden müssen. Damit reduziert sich nicht nur der für die Analyse relevante Teil des Operatorgraphen, sondern für diese Operatoren müssen dann bestimmte Eigenschaften in den jeweiligen Kriterien nicht nachgewiesen werden. Somit kann es sein, daß bestimmte Ordnungen sich nur unter Berücksichtigung der Blockierung vorhersagen lassen.

3.2 Zielordnungen

Zunächst werden drei grundsätzliche Kriterien vorgestellt (Mitlösung, Verhinderung, Zerstörung), mit denen sich notwendige Zielordnungen vorhersagen lassen. Dabei werden jeweils zwei Ziele Z_1 und Z_2 betrachtet und getestet, ob $Z_1 <_Z Z_2$ gilt. Später werden die Kriterien zu einem übergreifenden Gesamttest zusammengefaßt. Die Kriterien lassen sich mit Hilfe von Operatorgraphen effizient implementieren. Dies ist wichtig, denn schließlich soll der gewonnene Zeitvorteil beim Planen nicht durch eine sehr aufwendige Analyse egalisiert werden. Die Kriterien werden jeweils zuerst durch eine abstrakt formulierte Idee charakterisiert. Anschließend werden sie formal definiert. Diese Definitionen erscheinen vielleicht etwas zu kompliziert, denn die jeweiligen Ideen könnten auch formal abstrakter definiert werden, allerdings soll durch sie direkt gezeigt werden, wie sie sich mit Hilfe von Operatorgraphen

effizient implementieren lassen, ohne hier eine direkte Implementierungsbeschreibung zu geben. Was nutzt z.B. ein Kriterium, welches alle notwendigen Zielordnungen eines Problems bestimmt, indem der Schnitt aller Ordnungen aller möglichen Lösungen für das Problem berechnet wird? Also werden die Kriterien mit Hilfe der Begriffe der Operatorgraphen so formuliert, daß daraus direkt realisierbare Algorithmen abgeleitet werden können.

3.2.1 Mitlösung

Idee: Bei einem Problem P gilt $Z_1 <_Z Z_2$ wegen *Mitlösung*, wenn bei jeder Möglichkeit Z_2 zu lösen, Z_1 vorher zwingend mitgelöst werden muß und es kein Ziel $Z \neq Z_1$ in P gibt, bei dessen Lösung negative Interaktionen zu Z_1 entstehen können. Also wird bei der Lösung von Z_2 schon vorher Z_1 gelöst und die Gültigkeit von Z_1 kann nachher nicht mehr zerstört werden.

Abbildung 3.3 zeigt einen Operatorgraphen für ein einfaches Problem, bei dem Z1 $<_Z$ Z2 wegen *Mitlösung* gilt.

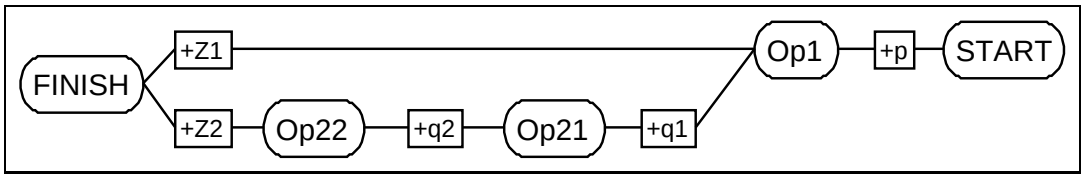

Abbildung 3.3: Beispiel für Mitlösung

Um *Z2* zu lösen, muß in jeder Lösung irgendwann *Op22* ausgeführt werden, da dies die einzige Möglichkeit ist. Dafür wird *q2* benötigt, weswegen vorher *Op21* ausgeführt werden muß. Dafür wird wiederum *q1* benötigt, weswegen vorher *Op1* ausgeführt werden muß. Also muß *Op1* für die Lösung von *Z2* ausgeführt werden. *Op1* löst aber auch direkt *Z1*. Da nach der Ausführung von *Op1* für die Lösung von *Z2* noch *Op21* und *Op22* ausgeführt werden müssen, wird *Z1* auf jeden Fall vor *Z2* gelöst.

Nun wird eine Eigenschaft *Mitlösung* definiert, mit der sich dann ein Theorem formulieren läßt, mit dem sich obige Idee formal ausdrücken läßt.

Definition 3.1 (Mitlösung) *Wenn Z_1, Z_2 Ziele in P sind, dann ist für alle $K \in VK(OG)$ folgende Eigenschaft definiert: $Mitlösung_{VK}(K, Z_1, Z_2)$ gdw*

$$\forall OK \in VOR_{OG}(K) : \neg Block(OK, Z_2) \Rightarrow Mitlösung_{OK}(K, OK, Z_1, Z_2) \quad (i)$$

Und für alle $OK \in VOR_{OG}(K)$ sei definiert: $Mitlösung_{OK}(K, OK, Z_1, Z_2)$ gdw

$$\begin{array}{lll} (\quad OK \in VOR_{OG}(Z_1) & \wedge & (ii) \\ \quad K \neq Z_2 & \wedge & (iii) \\ \quad BL(KT(OK), Z_2) \Rightarrow BL(KT(OK), Z_1) &) \quad \vee & (iv) \\ \quad \exists VK \in VOR_{OG}(OK) : Mitlösung_{VK}(VK, Z_1, Z_2) & & (v) \end{array}$$

Theorem 3.2 (Mitlösungskriterium) *Für $P = (S_P, Z_P, O_P)$ und $Z_1, Z_2 \in Z_P$ gilt:*

$$\begin{array}{lll} Mitlösung_{VK}(Z_2, Z_1, Z_2) & \wedge & (a) \\ \forall\, Z \in Z_P\,,\; Z \neq Z_1 \;\; \neg\exists\, OK \in OK(OG) : PotNegInt(OK, Z_1, Z, OG) & \Rightarrow & (b) \\ Z_1 <_Z Z_2 & & \end{array}$$

Bemerkungen: Die Definition von *Mitlösung* entspricht einer rekursiv formulierten Eigenschaft über der Struktur eines Operatorgraphen. Dabei werden alle Möglichkeiten, Z_2 zu lösen, untersucht. Die einzelnen Zeilen haben anschaulich formuliert folgende Bedeutungen:

(i) Für einen Vorbedingungsknoten K gilt $Mitlösung_{VK}$ für die Ziele Z_1 und Z_2, wenn für alle bezüglich Z_2 nicht blockierten Operatoren (denn die Lösung von Z_2 wird ja untersucht), die K lösen können, $Mitlösung_{OK}$ gilt.

Ein Operatorknoten OK, der den Effekt K lösen soll, hat die Eigenschaft $Mitlösung_{OK}$ für die Ziele Z_1 und Z_2 entweder direkt (durch die Zeilen (ii)-(iv)) oder indirekt über seine Vorbedingungsknoten (durch Zeile (v)). Bei obigem Beispiel haben die Operatorknoten *Op22* und *Op21* die Eigenschaft $Mitlösung_{OK}$ indirekt über *q2* bzw. *q1*, während *Op1* die Eigenschaft direkt besitzt.

(ii) Der Operator OK löst auch das Ziel Z_1.

(iii) OK wird nicht direkt für die Lösung von Z_2 ausgeführt, sondern ist über K nur ein indirekter Vorgänger von Z_2.

(iv) Wenn OK für die Lösung von Z_2 ausgeführt wird, so entstehen dabei Variablenbindungen, die auch Z_1 lösen.

(v) OK hat eine Vorbedingung, für die $Mitlösung_{VK}$ gilt. Da alle Vorbedingungen eines Operators gelöst werden müssen, um ihn auszuführen, reicht **eine** Vorbedingung aus, bei der $Mitlösung_{VK}$ gilt.

Das Theorem formuliert also dann genau oben aufgeführte Idee.

(a) Bei der Lösung von Z_2 wird Z_1 auf jeden Fall vorher mitgelöst.

(b) Es ist nicht möglich, daß Z_1 durch Operatoren, die für andere Ziele ausgeführt werden, wieder zerstört wird.

Beweis (von Theorem 3.2): Sei P ein Problem mit Zielen Z_1, Z_2 und es gelten die Voraussetzungen (a) und (b) von Theorem 3.2. Sei $PL = (M_{PL}, Ord_{PL})$ eine beliebige Lösung zu P. Nun wird die Lösung von Z_2 in PL betrachtet und dabei wird gezeigt, daß bei der Lösung von Z_2 schon vorher Z_1 gelöst werden muß und Z_1 nicht wieder zerstört werden kann und somit $Z_1 <_Z Z_2$ gilt.

Sei M_1 die Menge der Operatoren aus M_{PL}, die einen Effekt besitzen, der mit Z_2 unifizierbar ist. Da Z_2 in PL gelöst werden muß, wird mindestens einer der Operatoren aus M_1 für die Lösung von Z_2 in PL ausgeführt. Sei oBdA $O_1 \in M_1$ ein solcher Operator, dann gilt wegen (a) in der Voraussetzung und (i) in der Definition von *Mitlösung:* $Mitlösung_{OK}(Z_2, O_1, Z_1, Z_2)$. (I)

Da (iii) in (I) nicht gelten kann, muß (v) in der Definition von *Mitlösung* gelten. Es gibt also eine Vorbedingung V_2 von O_1, so daß $Mitlösung_{VK}(V_2, Z_1, Z_2)$ gilt. (II)

Sei nun M_2 die Menge der Operatoren aus M_{PL}, die einen Effekt besitzen, der mit V_2 unifizierbar ist. Da V_2 eine Vorbedingung eines Operators ist, der für die Lösung von Z_2

ausgeführt wird, muß auch V_2 dafür gelöst werden. Somit wird mindestens einer der Operatoren aus M_2 für die Lösung von Z_2 in PL ausgeführt. Sei oBdA $O_2 \in M_2$ ein solcher Operator, dann gilt wegen (II) $Mitlösung_{OK}(V_2, O_2, Z_1, Z_2)$. (III)

Nun sind zwei Fälle zu unterscheiden:
Fall 1: In (III) gelten (ii)-(iv) in der Definition von *Mitlösung*. Dann wird also ein Operator, nämlich O_2, für die Lösung von Z_2 ausgeführt, bei dem (ii)-(iv) gelten. (IV)

Fall 2: In (III) gilt (v) in der Definition von *Mitlösung*. Es gibt also eine Vorbedingung V_3 von O_2, so daß $Mitlösung_{VK}(V_3, Z_1, Z_2)$ gilt. (V)

Wegen (V) gibt es mit den gleichen Argumenten wie für (II) ein O_3 mit $Mitlösung_{OK}(V_3, O_3, Z_1, Z_2)$. Solange bei erneuter Fallunterscheidung Fall 2 auftritt, wird die Definition von *Mitlösung* rekursiv angewendet. Dabei entsteht folgende Kette: $Z_2 \rightarrow O_1 \rightarrow V_2 \rightarrow O_2 \rightarrow V_3 \rightarrow \ldots V_i \rightarrow O_i \rightarrow \ldots$ (VI)

Da PL endlich, ist obige Kette endlich und somit gibt es ein n, so daß in $Mitlösung_{OK}(V_n, O_n, Z_1, Z_2)$ (ii)-(iv) gelten. Wegen (IV)+(VI) wird in PL also auf jeden Fall ein Operator O_n für die Lösung von Z_2 ausgeführt, bei dem (ii)-(iv) gelten. (VII)

O_n löst wegen (ii)+(iv) in (VII) in PL auch Z_1. (VIII)

Wegen (*iii*) in (VII) gilt $n > 1$ und somit wird Z_1 echt vor Z_2 gelöst. (IX)

Wegen (*b*) in der Voraussetzung kann Z_1 durch andere Operatoren, die für andere Ziele ausgeführt werden, in PL nicht wieder ungültig werden. (X)

Also gilt wegen (VIII)-(X) in PL: $Z_1 <_Z Z_2$ qed.

Beispiel: In diesem Kapitel wird für jedes Kriterium eine eigene künstliche Domäne betrachtet. Das heißt nicht, daß diese Kriterien nur in stark konstruierten Fällen Ordnungen erkennen. In Kapitel 5 wird gezeigt, daß die Kriterien in den verschiedensten Domänen Erfolg haben. Mit ihnen werden Zielordnungen in einzelnen Teilen von komplizierten Domänen erkannt. Hier werden praktisch Domänen betrachtet, die nur aus diesen jeweiligen ordnungskritischen Teilen bestehen. Die künstlichen Domänen werden genau so konstruiert, daß sie möglichst einfach sind und dabei trotzdem alle wichtigen Effekte auftreten.

Für das Mitlösungskriterium wird die Domäne $DOM_M = (P_{DOM_M}, O_{DOM_M}, T_{DOM_M})$ betrachtet, für die gilt:

P_{DOM_M} = {*p1(x),p2(x),p31(x),p32(x),p4(x),Z1a(x),Z1b(x),Z1c(x),Z1d(x),Z2(x)*}
O_{DOM_M} = {*Op1(x), Op2(x, y), Op3(x), Op4(x)*}
T_{DOM_M} = {*typAB*}

Für die Operatoren aus O_{DOM_M} gilt:

{ *p1(x)* }	*Op1(x)*	{ *Z1a(x) , Z2(x), -Z1c(x)* }	[ ]
{ *p2(x)* }	*Op2(x, y)*	{ *p1(x), Z1b(y), Z1c(x), Z1d(x)* }	[ ]
{ *p31(x), p32(x)* }	*Op3(x)*	{ *Z2(x)* }	[ ]
{ *p4(x)* }	*Op4(x)*	{ *p31(x), Z1a(x), Z1b(x), Z1c(x), Z1d(x)* }	[ ]

Abbildung 3.4 zeigt den Operatorgraph für das Problem $PRO_4 = (S_{PRO_4}, Z_{PRO_4}, O_{PRO_4})$ in DOM_M mit:

S_{PRO_4} = {*p2(a), p32(a), p4(a)*}
Z_{PRO_4} = {*Z1a(a), Z1b(a), Z1c(a), Z1d(a), Z2(a)*}
O_{PRO_4} = {*(typAB: a, b)*}

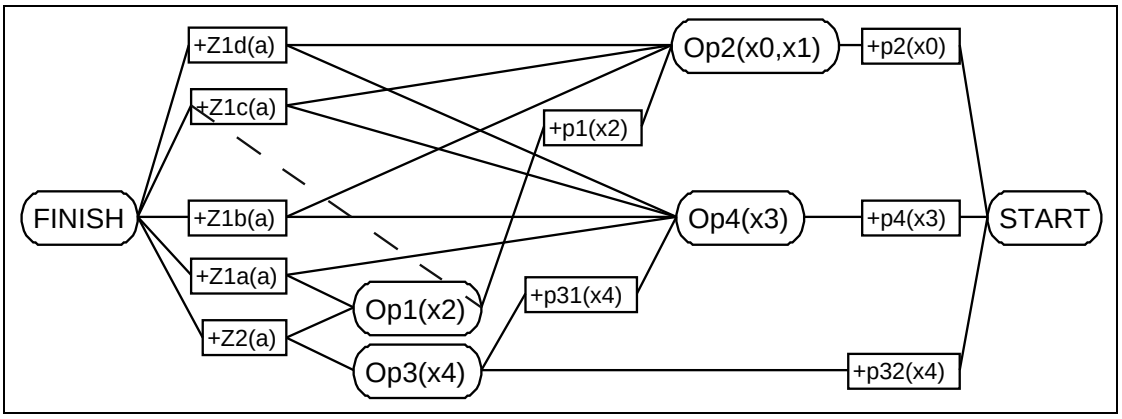

Abbildung 3.4: Operatorgraph für PRO_4

Für die Konstantenbeschriftung wird als Beispiel die Konstantentabelle von *Op2* gegeben:

KT(Op2) = { *(Z1d(a):* {*(x0:*{*a*}*)*, *(x1:*{*a,b*}*)*}*)*,
(Z1c(a): {*(x0:*{*a*}*)*, *(x1:*{*a,b*}*)*}*)*,
(Z1b(a): {*(x0:*{*a*}*)*, *(x1:*{*a*}*)*}*)*,
(Z1a(a): {*(x0:*{*a*}*)*, *(x1:*{*a,b*}*)*}*)*,
(Z2(a): {*(x0:*{*a*}*)*, *(x1:*{*a,b*}*)*} }

Dabei kann *x0* immer nur an *a* gebunden werden, da in S_{PRO_4} *p2(a)* aber nicht *p2(b)* vorkommt. Weiter ist zu beachten, daß *x1* für *Z1b(a)* nur an *a* gebunden werden kann, da dafür der Effekt *Z1b(y)* von *Op2(x, y)* ausgenutzt wird (*y* aus der Operatorbeschreibung entspricht *x1* im Operatorgraph).

Für PRO_4 gilt *Z1d(a)* $<_Z$ *Z2(a)* wegen *Mitlösung* und dies ist die einzige notwendige Zielordnung in PRO_4. Nun wird begründet, wie diese Ordnung durch das Mitlösungskriterium berechnet wird und wieso keine anderen Ordnungen, die ja falsch wären, erkannt werden.

Z1d(a) <$_Z$ Z2(a)! *Z2(a)* kann in PRO_4 in DOM_M auf zwei Weisen gelöst werden, mit *Op1* oder mit *Op3*. Wenn *Z2(a)* mit *Op1* gelöst werden soll, muß *p1(a)* gelöst werden. Dies geht nur mit *Op2*. Die Kette aus obigem Beweis wäre dann *Z2(a)* $\rightarrow$ *Op1(a)* $\rightarrow$ *p1(a)* $\rightarrow$ *Op2(a, x1)*, wobei *Op2* dann O_n entsprechen würde, denn es gilt $Mitlösung_{OK}$*(p1(a), Op2(a, x1), Z1d(a), Z2(a))* und (ii)-(iv) sind gültig, denn *Op2* löst dann auch *Z1d(a)*. Wenn *Z2(a)* mit *Op3* gelöst werden soll, ergibt sich mit analogen Argumenten die Kette *Z2(a)* $\rightarrow$ *Op3(a)* $\rightarrow$ *p31(a)* $\rightarrow$ *Op4(a)*, wobei zu beachten ist, daß *p32* nicht weiter berücksichtigt werden muß. Für *Op4* gilt dann das gleiche wie bei der ersten Möglichkeit für *Op2*. *Z1d(a)* wird also bei der Lösung von

Z2(a) auf jeden Fall vorher mitgelöst. Außerdem gibt es keine negative Interaktion zu *Z1d(a)*.

Z1c(a) <Z Z2(a)? Für *Z1c(a)* gilt prinzipiell das gleiche, wie für *Z1d(a)*, nur mit dem Unterschied, daß (b) in Theorem 3.2 nicht erfüllt ist. *Z1c(a)* wird bei der Lösung von *Z2(a)* zwar auf jeden Fall schon vorher mitgelöst, kann aber durch *Op1*, wenn dieser für *Z1a(a)* oder *Z2(a)* ausgeführt wird, wieder die Gültigkeit verlieren und muß somit anschließend nochmals gelöst werden.

Z1b(a) <Z Z2(a)? Für *Z1b(a)* gilt ebenfalls das gleiche, wie für *Z1d(a)*. Allerdings hat *Op2* nicht den Effekt *Z1b(x0)* sondern *Z1b(x1)*. Bei der Ausführung von *Op2* könnte *x1* auch an *b* gebunden sein. Es gilt *BL(KT(Op2), Z2(a)) = {(x0: {a}), (x1: {a, b})}* und außerdem *BL(KT(Op2), Z1b(a)) = {(x0: {a}), (x1: {a})}*, also ist (iv) aus Definition 3.1 für *Op2* nicht erfüllt.

Z1a(a) <Z Z2(a)? *Z1a(a)* wird zwar auf jeden Fall bei der Lösung von *Z2(a)* mitgelöst, allerdings nicht unbedingt vorher, denn mit *Op1* können beide Ziele gleichzeitig gelöst werden. Daher gilt (iii) für *Op1* nicht.

Alle anderen Kombinationen: Alle *Z1*(a)* können gleichzeitig durch *Op4* gelöst werden und somit kann zwischen ihnen keine Ordnung wegen *Mitlösung* bestehen. Und *Z2(a)* muß nicht vor einem *Z1*(a)* gelöst werden, da *Op4* alle *Z1*(a)* lösen kann, aber nicht *Z2(a)*.

Die Eigenschaft *Mitlösung* kann in einem Operatorgraph sehr effizient ermittelt werden. Es werden ausgehend von Z_2 die einzelnen Knoten untersucht. Dabei muß jeder Knoten höchstens einmal betrachtet werden.

Bemerkung zu rekursiven Domänen: Auch bei rekursiven Domänen muß jeder Knoten nur einmal betrachtet werden. Das liegt daran, daß dieser für alle evtl. vorkommenden Instanzen des jeweiligen Operators steht. Die Zykel müssen nicht mehrmals durchlaufen werden, da sich die Situation bei mehrmaligen Betrachtungen eines Knotens nicht ändert (im Gegensatz zur Planung). Denn während sich bei der Planung der Gesamtzustand durch Operatoranwendungen ändert, wird hier ein statisches Gebilde untersucht. Somit ist das oben formulierte Mitlösungskriterium auch für rekursive Domänen korrekt. Bei einer Implementierung muß allerdings eine Buchführung darüber stattfinden, welche Knoten schon betrachtet wurden, damit Zykel nicht unnötig mehrmals durchlaufen werden.

Werden für rekursive Zweige die potentiellen Variablenbindungen nach oben abgeschätzt, so hat dies an zwei Stellen Auswirkungen. Die eine Stelle befindet sich in Zeile (iv) der Definition von *Mitlösung*. Eine Abschätzung nach oben kann aber nur zur Folge haben, daß diese Bedingung nicht mehr gilt, aber nicht, daß dadurch diese Bedingung gilt. Also kann höchstens eine Zielordnung nicht erkannt werden, aber es können dadurch keine falschen Ordnungen berechnet werden. Die zweite Stelle ist in Zeile (b) des Mitlösungstheorems, dort wird die Eigenschaft *PotNegInt* benutzt. Betrachtet man die Definition 2.16, so wird nur *BL(KT(OK), Z)* nach oben abgeschätzt aber nicht *BL(IE)*. Dies kann nur zur Folge haben, daß dadurch Zeile (b) nicht gilt, aber nicht, daß (b) dadurch gilt. Also läßt sich

zusammenfassend sagen, daß Theorem 3.2 auch für rekursive Domänen, bei denen die potentiellen Variablenbindungen nach oben abgeschätzt werden, korrekt ist. Vollständig (in dem Sinne, daß alle notwendigen Zielordnungen erkannt werden) ist es ja auch für nicht rekursive Domänen nicht, denn es wird schließlich nur ein Aspekt betrachtet, aus dem Zielordnungen resultieren können.

3.2.2 Verhinderung

Idee: Bei einem Problem P gilt $Z_1 <_Z Z_2$ wegen *Verhinderung*, wenn bei jeder Möglichkeit Z_2 zu lösen, ein Zustand entsteht, in dem sich Z_1 nicht mehr lösen läßt. Also muß Z_1 vor Z_2 gelöst werden, da es nachher nicht mehr möglich ist.

Abbildung 3.5 zeigt einen Operatorgraph für ein einfaches Problem, bei dem $Z_1 <_Z Z_2$ wegen *Verhinderung* gilt.

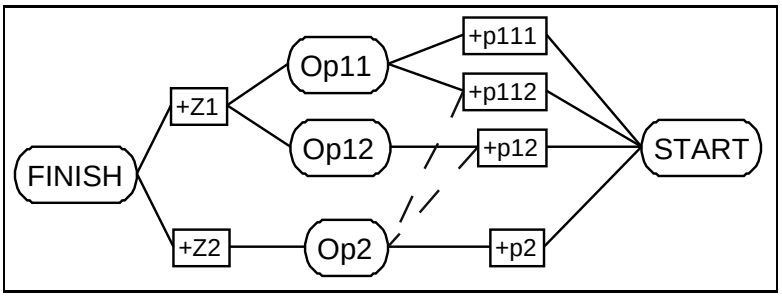

Abbildung 3.5: Beispiel für Verhinderung

Um *Z2* zu lösen, muß in jeder Lösung irgendwann *Op2* ausgeführt werden, da dies die einzige Möglichkeit ist, *Z2* zu lösen. Dabei treten negative Interaktionen zu *p112* und zu *p12* auf. Diese beiden Prädikate sind nur im Startzustand enthalten, können aber nicht durch andere Operatoren gültig werden. Also kann nun weder *Op11* noch *Op12* ausgeführt werden, da für beide Operatoren je eine Vorbedingung nicht mehr erfüllt werden kann. Deshalb läßt sich *Z1* nicht mehr lösen und muß somit vor *Z2* gelöst werden.

Obige Idee soll mit Operatorgraphen wie folgt umgesetzt werden: Es gibt einen lokalen Test für einen Operator *Op*. Dort wird getestet, ob sich nach Ausführung von *Op* für ein Ziel Z_2 ein anderes Ziel Z_1 noch lösen läßt. Dazu wird ein Operatorgraph OG so markiert, daß genau die Vorbedingungsknoten *VK ungültig* sind, zu denen *Op* negative Interaktionen besitzt, wenn *VK* für die Lösung von Z_1 benutzt wird. Dann wird in diesem Operatorgraph die *Pseudolösbarkeit* von Z_1 getestet. Z_1 ist in diesem markierten Operatorgraph nicht mehr *pseudolösbar*, wenn die Ausführung von *Op* für Z_2 einen Zustand zur Folge hat, in dem sich anschließend Z_1 nicht mehr lösen läßt. Global wird getestet, ob bei der Lösung von Z_2 auf jeden Fall ein solcher Operator *Op* ausgeführt wird, zu dem ein markierter Operatorgraph gehört, in dem Z_1 nicht mehr *pseudolösbar* ist, denn dann gilt $Z_1 <_Z Z_2$. Dies wird nun formal beschrieben.

Definition 3.2 (Markierter Operatorgraph) *Aus einem Operatorgraph OG entsteht ein markierter Operatorgraph* $\overline{OG}$*, indem alle Vorbedinungsknoten entweder als* **gültig** *oder als* **ungültig** *markiert werden. Wird ein Operator Op mit Operatorknoten OK für die Lösung eines Zieles* Z_2 *ausgeführt, dann ist der markierte Operatorgraph für ein Ziel* Z_1 *bezüglich der Ausführung von Op für* Z_2 *(Notation:* $\overline{OG}(Z_1, Z_2|OK)$*) wie folgt definiert: In* $\overline{OG}(Z_1, Z_2|OK)$ *sind alle Vorbedingungsknoten VK mit* $VK \in VOR_{OG}(OK_2)$ *(dabei ist* OK_2 *ein beliebiger Operatorknoten) als ungültig markiert, für die gilt:*

$$\exists IE \in IE(OG): \quad IE = OK \rightarrow VK \quad \wedge \quad (a)$$
$$BL(KT(OK_2), Z_1) \Rightarrow (BL(KT(OK), Z_2) \odot BL(IE)) \quad (b)$$

Ausßerdem wird Z_1 *als ungültig markiert, alle anderen Vorbedingungsknoten sind gültig.*

Beispiel: Wird der Operatorgraph *OG* von Abbildung 3.5 für das Ziel *Z1* bezüglich der Ausführung von *Op2* für *Z2* markiert (Bezeichnung dafür ist $\overline{OG}(Z1, Z2|Op2)$), so werden *Z1*, *p112* und *p12* als *ungültig* markiert und alle anderen Vorbedingungsknoten als *gültig*. Das Ziel *Z1* selber wird auf jeden Fall als *ungültig* markiert, denn für dieses soll ja die *Pseudolösbarkeit* erst hergeleitet werden und *p112*, *p12* werden als *ungültig* markiert, da es eine Interaktionskante von *Op2* nach *p112* bzw. *p12* gibt (siehe Zeile (a) in der Definition 3.2). Die Zeile (b) ist für dieses einfache Beispiel ohne Bedeutung, da es dort keine Variablen gibt. Sie soll sicherstellen, daß bei der Ausführung von *OK* (im Beispiel *Op2*) für Z_2 Variablenbindungen entstehen, die tatsächlich die von OK_2 (im Beispiel *Op11* bzw. *Op12*) für die Lösung von Z_1 benötigte Vorbedingung (im Beispiel *p112* bzw. *p12*) ungültig machen. Zeile (b) wird an Hand eines späteren Beispiels, bei dem auch Variablenbindungen auftreten, noch näher erläutert.

Mit Hilfe von markierten Operatorgraphen läßt sich nun ein Kriterium angeben, welches besagt, daß ein Ziel zumindest im Prinzip noch lösbar ist (pseudolösbar).

Definition 3.3 (Pseudolösbarkeit) *Seien* Z_1*,* Z_2 *Ziele von P,* $OK \in OK(OG)$ *und sei* $\overline{OG}(Z_1, Z_2|OK)$ *der bezüglich OK markierter Operatorgraph für* Z_1*, dann ist für alle* $K \in VK(\overline{OG}(Z_1, Z_2|OK))$ *folgende Eigenschaft definiert:* $Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), K)$ *gdw*

$$\begin{array}{lll} & K \text{ ist in } \overline{OG}(Z_1, Z_2|OK) \text{ als gültig markiert} & \vee \quad (i) \\ (& \exists OK_2 \in VOR_{\overline{OG}(Z_1,Z_2|OK)}(K): & \quad (ii) \\ & \neg Block(OK_2, Z_1) & \wedge \quad (iii) \\ & OK_2 \neq START & \wedge \quad (iv) \\ & \forall VK \in VOR_{\overline{OG}(Z_1,Z_2|OK)}(OK_2): Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), VK) &) \quad (v) \end{array}$$

Die Definition von *Pseudolösbar* entspricht, ähnlich der von *Mitlösung*, einer rekursiv formulierten Eigenschaft über der Struktur des markierten Operatorgraphen. Dabei werden alle Möglichkeiten, die Vorbedingung K zu lösen, untersucht. K ist *pseudolösbar* entweder direkt durch Zeile (i) oder indirekt über einen Operator OK_2 durch die Zeilen (ii)-(v). Dabei bedeuten die einzelnen Zeilen anschaulich:

(i) Wenn K als *gültig* markiert wurde, kann für diese Vorbedingung keine Verhinderung durch OK bestehen und somit ist K *pseudolösbar*.

(ii) Sonst muß es einen Operator OK_2 geben, der K erfüllen kann und für den folgendes gilt: (Zeilen (iii)-(v))

(iii) Da ja schließlich Z_1 gelöst werden soll, darf OK_2 nicht für Z_1 blockiert sein.

(iv) OK_2 darf nicht der START-Knoten sein, denn OK_2 soll ja nach der Ausführung von OK angewendet werden und das geht mit START natürlich nicht.

(v) OK_2 selber darf nicht durch die Ausführung von OK unanwendbar sein, da eine Vorbedingung nicht lösbar ist. Also müssen alle Vorbedingungen von OK_2 in $\overline{OG}(Z_1, Z_2|OK)$ *pseudolösbar* sein.

Beispiel: Wendet man die Definition von *Pseudolösbarkeit* für *Z1* in $\overline{OG}$*(Z1, Z2| Op2)* an, so gilt folgendes: *Z1* ist nicht als *gültig* markiert, also gilt (i) nicht. In Zeile (ii) kommen für OK_2 *Op11* und *Op12* in Frage. Für beide gelten Zeilen (iii) und (iv). Wegen (v) muß also die *Pseudolösbarkeit* von *p111* und *p112* bzw. die *Pseudolösbarkeit* von *p12* getestet werden. *p111* ist als *gültig* markiert und ist somit wegen (i) *pseudolösbar*, dagegen sind sowohl *p112* als auch *p12* nicht *pseudolösbar*, da sie beide als *ungültig* markiert sind und der einzige Operator, der sie erfüllen kann, START ist. Also ist *Z1* in $\overline{OG}$*(Z1, Z2|Op2)* nicht *pseudolösbar*.

Mit der *Pseudolösbarkeit* läßt sich die Eigenschaft *Verhinderung* nun wie folgt definieren.

Definition 3.4 (Verhinderung) *Wenn Z_1, Z_2 Ziele in P sind, dann ist für alle $K \in VK(OG)$ folgende Eigenschaft definiert: $Verhinderung_{VK}(K, Z_1, Z_2)$ gdw*

$$\forall\ OK \in VOR_{OG}(K) : \neg Block(OK, Z_2) \Rightarrow Verhinderung_{OK}(OK, Z_1, Z_2) \qquad (A)$$

Und für alle $OK \in VOR_{OG}(K)$ sei definiert: $Verhinderung_{OK}(OK, Z_1, Z_2)$ gdw

$$\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), Z_1) \qquad \vee \qquad (B)$$
$$\exists VK \in VOR_{\overline{OG}(Z_1,Z_2|OK)}(OK) : Verhinderung_{VK}(VK, Z_1, Z_2) \qquad (C)$$

Bemerkung: $Verhinderung_{VK}(K, Z_1, Z_2)$ in einem Problem P bedeutet anschaulich, daß, egal wie K in P gelöst wird (denn $Verhinderung_{OK}(OK, Z_1, Z_2)$ muß für alle möglichen Operatoren, die K lösen können, gelten - siehe Zeile (A)), immer ein Operator OK ausgeführt wird, so daß $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), Z_1)$ gilt (entweder direkt durch Zeile (B) oder indirekt durch (C)). Und dies wiederum bedeutet, daß nach Ausführung von OK für Z_2 das Ziel Z_1 nicht mehr gelöst werden kann.

Damit setzt folgendes Theorem direkt die oben beschriebene Idee der Verhinderung um.

Theorem 3.3 (Verhinderungskriterium) *Seien Z_1, Z_2 Ziele in P, dann gilt:*

$$Verhinderung_{VK}(Z_2, Z_1, Z_2) \qquad \Rightarrow \qquad Z_1 <_Z Z_2$$

Beispiel: Für das Verhinderungskriterium wird eine Domäne $DOM_{Va} = (P_{DOM_{Va}}, O_{DOM_{Va}}, T_{DOM_{Va}})$ betrachtet, für die gilt:

$P_{DOM_{Va}} = \{p1(x),\ p2(x),\ p3(x),\ p4(x),\ Z1(x),\ Z2(x)\}$
$O_{DOM_{Va}} = \{Op1(x),\ Op2(x),\ Op3(x),\ Op4(x,\ y)\}$
$T_{DOM_{Va}} = \{typAB\}$

Für die Operatoren aus $O_{DOM_{Va}}$ gilt:

{ p1(x) }	Op1(x)	{ Z1(x) }	[ ]
{ p2(x) }	Op2(x)	{ p1(x) }	[ ]
{ p3(x) }	Op3(x)	{ Z2(x) }	[ ]
{ p4(x) }	Op4(x, y)	{ p3(x), -p1(x), -p2(x) }	[ ]

Abbildung 3.6 zeigt den Operatorgraph OG für das Problem $PRO_6 = (S_{PRO_6}, Z_{PRO_6}, O_{PRO_6})$ in DOM_{Va} mit:

$S_{PRO_6} = \{p2(a),\ p4(a)\}$
$Z_{PRO_6} = \{Z1(a),\ Z2(a)\}$
$O_{PRO_6} = \{(typAB:\ a,\ b)\}$

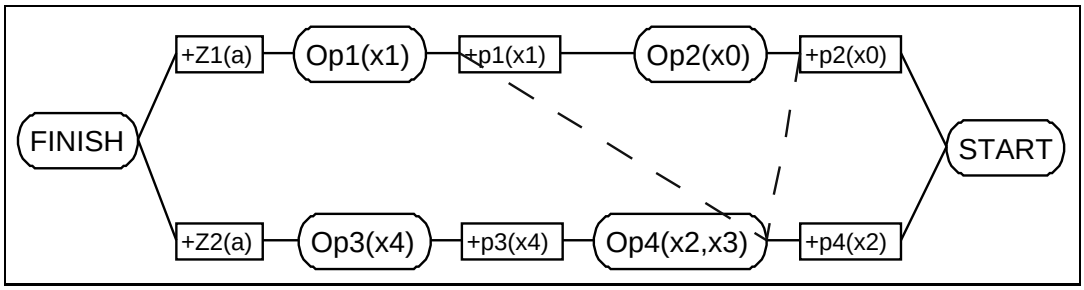

Abbildung 3.6: Operatorgraph für PRO_6 in DOM_{Va}

In PRO_6 gibt es die notwendige Zielordnung $Z1(a) <_Z Z2(a)$ wegen *Verhinderung*. Nun wird begründet, wie das Verhinderungskriterium diese erkennt. Es wird zunächst $Verhinderung_{VK}$ *(Z2(a),Z1(a),Z2(a))* getestet. Da es nur einen Operator, nämlich *Op3*, gibt, der *Z2(a)* lösen kann, wird $Verhinderung_{OK}$*(Op3, Z1(a), Z2(a))* getestet (siehe Zeile (A) in Definition 3.4). Wegen Zeile (B) wird dann *pseudolösbar*($\overline{OG}$*(Z1(a), Z2(a)|Op3), Z1(a))* getestet. Da *Op3* keine negativen Interaktionen besitzt, sind in $\overline{OG}$*(Z1(a), Z2(a)|Op3)* alle Vorbedingungsknoten, außer *Z1(a)* als *gültig* markiert. Somit ist Zeile (i) in Definition 3.3 nicht erfüllt, allerdings gelten die Zeilen (ii)-(v), denn *Op1* ist direkter Vorgänger von *Z1(a)* und *Op1* ist nicht bezüglich *Z1(a)* blockiert und *Op1* ist nicht START und die einzige Vorbedingung *p1* ist *pseudolösbar*, denn sie ist als *gültig* markiert. Also gilt *pseudolösbar*($\overline{OG}$*(Z1(a), Z2(a)|Op3), Z1(a))*, was die Tatsache wiederspiegelt, daß *Z1(a)* nach Ausführung von *Op3* noch lösbar ist.
Allerdings gibt es ja noch die offene Vorbedingung *p3(x4)*. Dies wird in Zeile (C) berücksichtigt. Es wird $Verhinderung_{VK}$*(p3(x4), Z1(a),Z2(a))* getestet. Mit analoger Begründung zu oben muß dann schließlich *pseudolösbar*($\overline{OG}$*(Z1(a), Z2(a)|Op4), Z1(a))* getestet werden. Nach Definition 3.2 sind *p1(x1)* und *p2(x0)* in $\overline{OG}$*(Z1(a), Z2(a)|Op4)* als *ungültig*

markiert. Deshalb gilt *pseudolösbar*($\overline{OG}$*(Z1(a), Z2(a)|Op4), Z1(a)* nicht, da der Test ausgehend von *Z1(a)* bis zum START-Knoten läuft und dann abbricht. Also ist *Z1(a)* nach Ausführung von *Op4* nicht mehr lösbar. Somit gilt Zeile (B) beim rekursiven Aufruf und damit $Verhinderung_{VK}$*(Z2(a), Z1(a), Z2(a))*.

Diskussion

Der in Anhang A.2 aufgeführte Beweis zum Verhinderungskriterium zeigt, daß dieses restriktiv genug ist, um keine falschen Ordnungen zu erkennen. Hier wird nun mit einigen Beispielen begründet, wieso die Eigenschaft *Verhinderung* so restriktiv sein muß, wie oben definiert, da sonst falsche Ordnungen erkannt würden. Dazu wird die Domäne DOM_{Va} bei den Operatoren *Op3* und *Op4* leicht variiert und für das gleiche Problem PRO_6 gezeigt, daß dann jeweils *Z1(a)* $<_Z$ *Z2(a)* nicht mehr gilt.

In Abbildung 3.7 ist der Operatorgraph für PRO_6 in DOM_{Vb} dargestellt. Dabei unterscheidet sich DOM_{Vb} von DOM_{Va} nur durch die Definition von *Op4*:

{ *p4(x)* } *Op4(x, y)* { *p3(x), -p2(x)* } [ ]

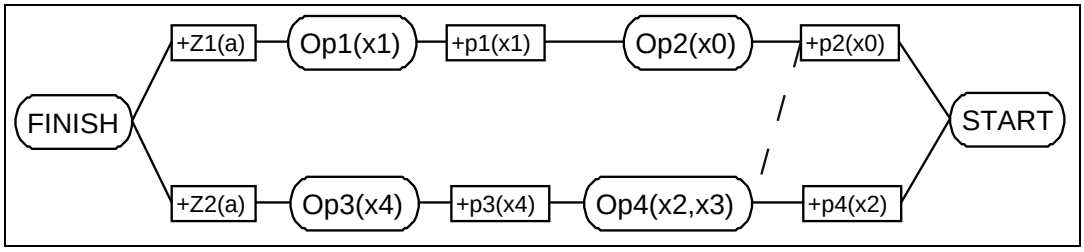

Abbildung 3.7: Operatorgraph für PRO_6 in DOM_{Vb}

Es fehlt die Interaktion *Op4* $\rightarrow$ *p1*. Zwar wird bei der Lösung von *Z2(a)* auf jeden Fall *p2(a)* ungültig und *p2(a)* wird für die Lösung von *Z1(a)* auf jeden Fall benötigt, allerdings ist folgender Lösungsplan möglich: *[Op2(a)* $\rightarrow$ *Op4(a)* $\rightarrow$ *Op3(a)* $\rightarrow$ *Op1(a)]*, wobei *Z2(a)* **vor** *Z1(a)* gelöst wird. Für den Fall, daß die Interaktion zu *p1* vorhanden ist und die zu *p2* weggelassen wird, gilt ähnliches. In beiden Fällen ist *Z1(a)* sowohl nach der Ausführung von *Op3* als auch nach der Ausführung von *Op4* noch *pseudolösbar*. Es reicht also nicht, daß die Gültigkeit von benötigten Vorbedingungen zerstört werden. Deshalb ist die Definition von *pseudolösbar* wesentlich restriktiver.

Nun wird gezeigt, daß es nicht ausreicht, wenn *Z1(a)* nach der gesamten Lösung von *Z2(a)* nicht mehr *pseudolösbar* ist, sondern, daß diese Eigenschaft von einem Operator ausgehen muß. Dazu wird PRO_6 in DOM_{Vc} in Abbildung 3.8 betrachtet. Dabei sind in DOM_{Vc} die Operatoren *Op3* und *Op4* wie folgt definiert:

{ *p3(x)* } *Op3(x)* { *Z2(x) -p2(x)* } [ ]
{ *p4(x)* } *Op4(x, y)* { *p3(x), -p1(x)* } [ ]

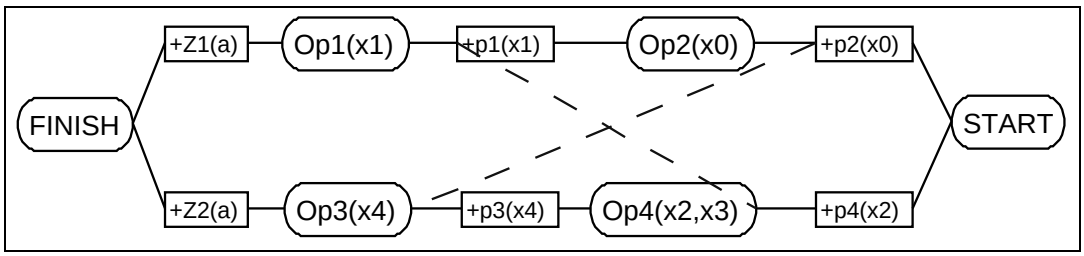

Abbildung 3.8: Operatorgraph für PRO_6 in DOM_{Vc}

Bei der Lösung von *Z2(a)* werden sowohl *p1(a)* als auch *p2(a)* ungültig und somit ist *Z1(a)* nicht mehr *pseudolösbar*. Allerdings ist folgender Lösungsplan möglich: *[Op4(a) → Op2(a) → Op3(a) → Op1(a)]*, wobei *Z2(a)* wieder **vor** *Z1(a)* gelöst wird. Dabei wird zuerst die Lösung von *Z2(a)* vorbereitet und danach die von *Z1(a)*. Dann erst wird die Lösung von *Z2(a)* abgeschlossen und dann die von *Z1(a)*. Deshalb wird bei der *Verhinderung* nicht getestet, ob bei jeder Lösung von Z_2 Operatoren ausgeführt werden, so daß nachher Z_1 nicht mehr *pseudolösbar* ist, sondern ob bei jeder Lösung von Z_2 **ein** Operator *Op* ausgeführt werden muß, so daß durch die Ausführung von *Op* Z_1 nicht mehr *pseudolösbar* ist.

Im letzten Beispiel wird nun Zeile (b) aus Definition 3.2 motiviert. Dazu wird PRO_6 in DOM_{Vd} betrachtet, wobei *Op4* in DOM_{Vd} folgendermaßen definiert ist:

{ p4(x) } Op4(x, y) { p3(x), -p1(x), -p2(y) } [ ]

Der Operatorgraph dazu sieht aus wie bei DOM_{Va} in Abbildung 3.6. Der entscheidende Unterschied liegt darin, daß *x3* bei der Ausführung von *Op4(x2, x3)* auch an *b* gebunden werden kann. Da *Op4* den Effekt *-p2(y)* (*y* entspricht *x3*) und nicht *-p2(x)* besitzt, kann *p2(a)* dabei gültig bleiben. Dies wird durch Zeile (b) in Definition 3.2 erkannt. Im markierten Operatorgraph $\overline{OG}$*(Z1(a), Z2(a)|Op4(x2, x3))* wird *p2(x0)* **nicht** als *ungültig* markiert, denn in Zeile (b) gilt: $BL(KT(OK_2), Z_1) = BL(KT(Op2(x0)), Z1(a)) = \{(x0 : \{a\})\}$ und $BL(KT(OK), Z_2) = BL(KT(Op4(x2, x3)), Z2(a)) = \{(x2: \{a\}), (x3: \{a, b\})\}$ und $BL(IE) = \{(x0: \{x3\})\}$. Also gilt $(BL(KT(OK), Z_2) \odot BL(IE)) = \{(x0: \{a, b\})\}$ und somit gilt Zeile (b) nicht. Deshalb wird *p2(x0)* als *gültig* markiert und somit ist *Z1(a)* in $\overline{OG}$*(Z1(a), Z2(a)|Op4(x2, x3))* *pseudolösbar* und demnach gilt *Verhinderung(Z2(a), Z1(a), Z2(a))* nicht.

3.2.3 Zerstörung

Idee: Bei einem Problem P gilt $Z_1 <_Z Z_2$ wegen *Zerstörung*, wenn bei jeder Möglichkeit Z_1 zu lösen, negative Interaktionen zu Z_2 entstehen. Also muß Z_1 vor Z_2 gelöst werden, da Z_2 sonst wieder ungültig würde und nochmals gelöst werden müßte.

Abbildung 3.9 zeigt einen Operatorgraph für ein einfaches Problem, bei dem $Z_1 <_Z Z_2$ wegen *Zerstörung* gilt.

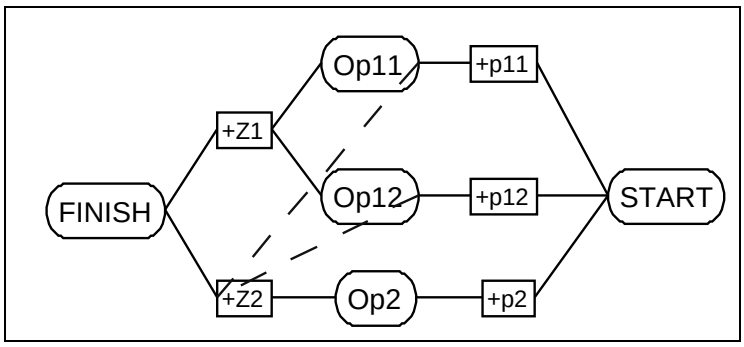

Abbildung 3.9: Beispiel für Zerstörung

Um *Z1* zu lösen, muß in jeder Lösung entweder *Op11* oder *Op12* ausgeführt werden. Beide Operatoren haben eine negative Interaktion zu *Z2*. Deshalb muß danach auf jeden Fall noch *Op2* ausgeführt werden und somit wird *Z2* immer nach *Z1* gelöst.

Definition 3.5 (Zerstörung) *Wenn Z_1, Z_2 Ziele in* P *sind, dann ist folgende Eigenschaft definiert: $Zerstörung(Z_1, Z_2)$ gdw*

$$\forall OK \in VOR_{OG}(Z_1) : \neg Block(OK, Z_1) \Rightarrow NegInt(OK, Z_2, Z_1, OG)$$

Theorem 3.4 (Zerstörungkriterium) *Seien Z_1, Z_2 Ziele in P, dann gilt:*

$$Zerstörung(Z_1, Z_2) \quad \Rightarrow \quad Z_1 <_Z Z_2$$

Das Zerstörungskriterium ist in dieser ersten Fassung sehr einfach, denn es werden nur Interaktionen von direkten Vorgängern von Zielen zu anderen Zielen betrachtet. Darum wird auf einen Beweis verzichtet. Will man auch Interaktionen von indirekten Vorgängern betrachten, so wird dies deutlich komplizierter. Daß indirekte Vorgänger nicht so einfach behandelt werden können wie direkte, zeigt folgendes Beispiel in Abbildung 3.10.

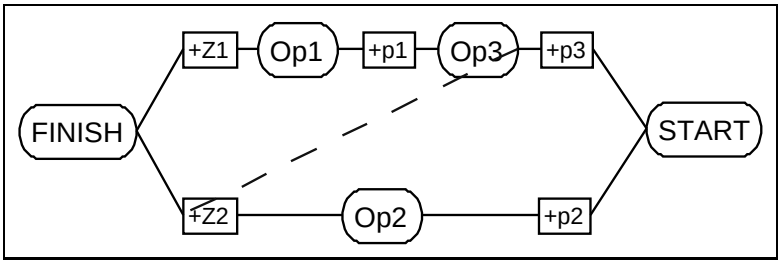

Abbildung 3.10: Interaktion von indirektem Vorgänger von Z_1

In diesem Problem wird zwar die Gültigkeit von *Z2* auch auf jeden Fall zerstört, wenn *Z1* gelöst wird, allerdings nicht direkt bei der Lösung von *Z1*. Daher ist folgender Lösungsplan möglich, bei dem *Z2* **vor** *Z1* gelöst wird: *[Op3 → Op2 → Op1]*. Deshalb ist Theorem

3.4 nur für direkte Vorgänger formuliert. Allerdings können Interaktionen von indirekten Vorgängern von Zielen auch Zielordnungen zur Folge haben, wie das Beispiel in Abbildung 3.11 verdeutlicht.

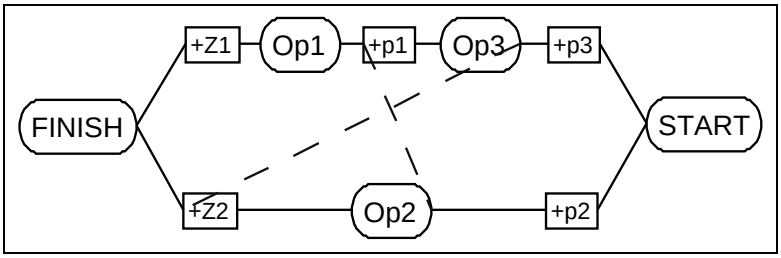

Abbildung 3.11: Kombinierte Interaktionen

Für dieses Problem gilt *Z1* $<_Z$ *Z2*, da die Interaktion von *Op2* nach *p1* sicherstellt, daß nach der Lösung von *Z2* für die Lösung von *Z1* unbedingt noch *Op3* ausgeführt werden muß. Nun könnte man das Zerstörungskriterium so erweitern, daß auch solche Fälle erkannt werden. Eine andere Möglichkeit ist eine Kombination aus Verhinderungs- und Zerstörungskriterium.

3.2.4 Kombiniationen der Kriterien

Die Betrachtung des Beispiels in Abbildung 3.11 legt eine Kombination der Kriterien nahe. Zunächst werden die Ideen von *Verhinderung* und *Zerstörung* kombiniert:

Kombination V&Z: Bei einem Problem P gilt $Z_1 <_Z Z_2$ wegen *Verhinderung&Zerstörung*, wenn bei jeder Möglichkeit Z_2 zu lösen, ein Zustand entsteht, in dem sich Z_1 nur noch so lösen läßt, daß dabei die Gültigkeit von Z_2 zerstört wird. Wenn also Z_2 vor Z_1 gelöst werden soll, kann Z_1 danach nicht mehr gelöst werden oder Z_2 wird wieder ungültig und muß somit nochmals gelöst werden. Daher muß Z_2 immer nach Z_1 gelöst werden.

Die kombinierten Kriterien sind implementiert und getestet. Trotzdem wird hier auf eine formale Definition, die eine Implementierungsmöglichkeit darstellt, verzichtet, da diese eine Kombination der bereits vorgestellten Definitionen ist und sehr kompliziert würde.

Während weder das Verhinderungs- noch das Zerstörungskriterium die notwendige Zielordnung für das in Abbildung 3.11 dargestellte Problem finden, hat das Kombinationskriterium Erfolg, denn es gilt: Bei jeder Möglichkeit *Z2* zu lösen (es gibt nur die Möglichkeit mit *Op2*), entsteht dabei ein Zustand (es gilt dann *p1* nicht mehr), in dem sich *Z1* nur noch so lösen läßt (nämlich mit *Op1* und *Op3*), daß dabei *Z2* zerstört wird (durch *Op3*).

Die Kombination der beiden Kriterien erkennt nicht nur alle notwendigen Zielordnungen, die von den beiden Kriterien entdeckt würden, sondern noch weitere darüber hinaus. Wird ein Gesamtkriterium, welches die Ideen von *Verhinderung*, *Zerstörung* und *Mitlösung* vereint, noch mächtiger? Folgendes Beispiel in Abbildung 3.12 zeigt, daß diese Frage mit 'Ja' beantwortet werden kann.

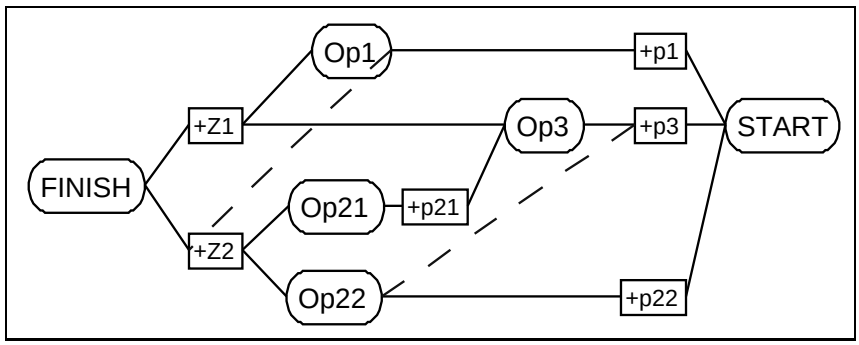

Abbildung 3.12: Beispiel für Gesamtkombination

Damit die Betrachtung nicht zu kompliziert wird, wurde hier auf Variablen in den Prädikaten und Operatoren verzichtet, so daß die Domäne und das Problem nicht explizit definiert werden müssen, da alle relevanten Informationen dem Operatorgraph entnommen werden können. In diesem Problem gilt $Z_1 <_Z Z_2$. Diese Ordnung wird von keinem der oben vorgestellten Kriterien, wenn diese jeweils einzeln getestet werden, erkannt, jedoch wird die Ordnung von folgendem Gesamtkriterium entdeckt.

Kombination M&V&Z: Bei einem Problem P gilt $Z_1 <_Z Z_2$ wegen *Mitlösung&Verhinderung&Zerstörung*, wenn bei jeder Möglichkeit Z_2 zu lösen dabei entweder Z_1 schon vorher mitgelöst wird und Z_1 bei der Lösung eines anderen Zieles nicht wieder ungültig werden kann, oder dabei ein Zustand entsteht, in dem sich Z_1 nur noch so lösen läßt, daß dadurch Z_2 ungültig wird. Also muß Z_1 immer vor Z_2 gelöst werden.

Dieses Gesamtkriterium wird nun an obigem Beispiel durchgetestet: In jeder Lösung muß *Z2* gelöst werden. Dafür gibt es zwei Möglichkeiten, mit *Op21* oder mit *Op22*. Wenn *Z2* mit *Op21* gelöst wird, muß vorher *p21* gelöst werden, was nur mit *Op3* möglich ist. Durch *Op3* wird aber *Z1* schon vor *Z2* gelöst und *Z1* kann bei der Lösung von *Z2* nicht wieder ungültig werden. Wenn *Z2* mit *Op22* gelöst wird, gilt *p3* danach nicht mehr. Also kann *Z1* nicht mehr mit *Op3* gelöst werden, sondern nur noch mit *Op1*. Dabei wird aber *Z2* wieder ungültig. Also muß *Z1* immer vor *Z2* gelöst werden.

Dagegen findet keines der einzelnen Kriterien diese Ordnung:

- *Mitlösung* findet keine Ordnung, denn es gibt eine Möglichkeit (nämlich mit *Op22*) *Z2* zu lösen, ohne daß dabei *Z1* mitgelöst wird.
- Auch *Verhinderung* findet keine Ordnung, da es eine Möglichkeit (nämlich mit *Op21* und *Op3*) gibt, *Z2* zu lösen, so daß nachher *Z1* mit *Op1* noch lösbar ist.
- Schließlich findet *Zerstörung* keine Ordnung, da es eine Möglichkeit (nämlich mit *Op3*) gibt, *Z1* so zu lösen, ohne daß dabei *Z2* ungültig wird.

Offensichtlich gelten zwar für einzelne Teile des Operatorgraphen und damit für bestimmte Lösungsmöglichkeiten der Ziele die entsprechenden Voraussetzungen der jeweiligen Kriterien, jedoch gibt es immer eine Möglichkeit, für die nicht gezeigt werden kann, daß

$Z_1 <_Z Z_2$ gilt. Nur wenn alle Kriterien simultan getestet werden, ist es möglich, die notwendige Zielordnung zu berechnen. Die einzelnen Kriterien ergänzen sich dabei, indem sie jeweils einzelne Teile des Operatorgraphen abdecken. Das Gesamtkriterium kann somit für die verschiedenen Lösungsmöglichkeiten der Ziele jeweils unterschiedliche Gründe für das Vorhandensein einer Ordnung heranziehen.

3.3 Unlösbarkeit

Ein Planungssystem kann ein Problem nur dann lösen, wenn es überhaupt eine Lösung gibt. Gibt es für ein Problem keine Lösung, so sollte das System sinngemäß mit 'nicht lösbar' antworten. Wenn bei der vorverarbeitenden Analyse eines zu lösenden Problems schon erkannt wird, daß dieses nicht lösbar ist, so ist dies wesentlich effizienter, als wenn das Planungssystem den gesamten Suchraum durchlaufen muß, um zu der gleichen Feststellung zu kommen. Ein nicht vollständiges System kann zu unlösbaren Problemen gar keine sichere Aussage machen. Bei den hier vorgestellten Analyseverfahren sind für die Unlösbarkeit zwei Gründe möglich: Es gibt ein Ziel, für das es nur blockierte Operatoren gibt oder es werden inkonsistente Zielordnungen erkannt.

3.3.1 Unlösbarkeit durch Operatorblockaden

Nach der Berechnung der Operatorblockaden ist mit einem einfachen Test festzustellen, ob ein Problem $P = (S_P, Z_P, O_P)$ wegen Operatorblockaden unlösbar ist.

$$\exists\, Z \in Z_P \forall\, OK \in VOR_{OG}(Z) : Block(OK, Z) \quad \Rightarrow \quad P \textit{ ist unlösbar}$$

Dies ist leicht einsichtig, denn ist ein Operator OK für Z wegen Variablenbindungen blockiert, so kann OK nicht so angewendet werden, daß dadurch Z gelöst wird. Und ist OK wegen Interaktionen blockiert, so kann, wenn OK für die Lösung von Z ausgeführt wird, ein anderes Ziel aus P nicht gelöst werden. Sind also alle Operatoren für ein Ziel blockiert, kann mindestens ein Ziel in P nicht gelöst werden.

Der Struktur des Operatorgraphen sieht man die Unlösbarkeit des zugehörigen Problems häufig nicht an. Nur durch die Analyse von blockierten Operatoren ist diese zu erkennen. So ist z.B. PRO_3 (siehe Abbildung 3.1) unlösbar, obwohl die Struktur allein nicht darauf schließen läßt.

3.3.2 Unlösbarkeit durch inkonsistente Ordnungen

Wird für ein Problem P eine Menge Ord von notwendigen Zielordnungen berechnet, so ist P unlösbar, wenn Ord inkonsistent ist. Dabei ist eine Menge von Ordnungen inkonsistent, wenn sie einen Zykel enthält. Der Test, ob eine Menge von Ordnungen einen Zykel enthält, ist einfach zu realisieren. Abbildung 3.13 zeigt einen Operatorgraph für ein wegen inkonsistenten notwendigen Zielordnungen unlösbares Problem.

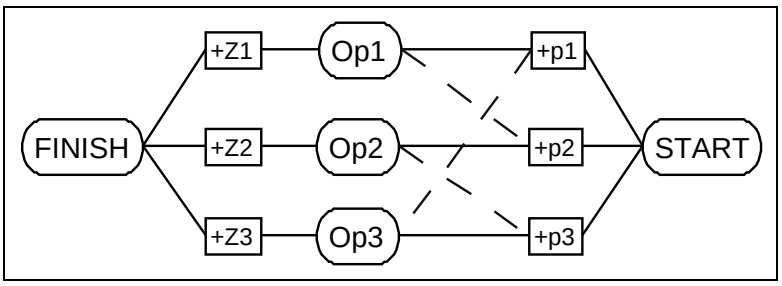

Abbildung 3.13: Zyklische Zielordnungen

Hier gibt es die drei notwendigen Zielordungen $Z3 <_Z Z2$, $Z2 <_Z Z1$, $Z1 <_Z Z3$ (welche auch von den Kriterien berechnet werden). Die drei Ordnungen stellen einen Zykel dar, dahalb ist dieses Problem nicht lösbar.

Mit diesen beiden einfachen Tests ist es möglich, die Unlösbarkeit vorzeitig zu erkennen und somit dem Planungssystem unnötige Arbeit zu ersparen. Diese Form der Analyse kann auch dazu eingesetzt werden, um Fehler in Domänen- oder Problembeschreibungen zu entdecken.

3.4 Gesamtanalyse

Soll ein Problem $P = (S_P, Z_P, O_P)$ analysiert werden, so müssen folgende Schritte ausgeführt werden:

1. Zuerst muß der zugehörige Operatorgraph mit Konstantenbeschriftung erstellt werden. Steht Metawissen über die Domäne bezüglich Ordnungsrelevanzen zur Verfügung, so kann dies dabei verwendet werden.

2. Nun können die Operatorblockaden wegen Variablenbindungen und Interaktionen berechnet werden.

3. Danach kann bereits der Test auf Unlösbarkeit des Problems wegen Operatorblockaden erfolgen und bei Unlösbarkeit die Analyse abgebrochen werden.

4. Dann werden notwendige Zielordnungen berechnet, indem für jedes ungeordnete Paar $Z_1, Z_2 \in Z_P$ das Gesamtkriterium *Kombination M&V&Z* getestet wird. Anschließend wird die transitive Hülle der berechneten Zielordnungen gebildet.

5. Schließlich wird das Problem auf Unlösbarkeit wegen inkonsistenter Ordnungen getestet.

Die Berechnung der notwendigen Zielordnungen im 4. Punkt kann noch effizienter erfolgen: Wenn die Eigenschaft der Unlösbarkeit unwichtig ist bzw. vorausgesetzt werden kann, daß alle zu lösenden Probleme auch lösbar sind, so ist es nicht nötig, nach erfolgreicher Berechnung von $Z_1 <_Z Z_2$ noch $Z_2 <_Z Z_1$ zu testen. Außerdem kann die Transitivität der Ordnungsbeziehung ausgenutzt werden. Wurde also bereits $Z_1 <_Z Z_2$ und $Z_2 <_Z Z_3$

berechnet, so ist es nicht mehr notwendig, $Z_1 <_Z Z_3$ zu testen. Bei dieser Vorgehensweise ist eine besondere Reihenfolge der zu testenden Paare sinnvoll, damit möglichst oft solche Ketten ausgenutzt werden können. Wurde also $Z_1 <_Z Z_2$ berechnet, so sollten die folgenden Tests nicht $Z_1 <_Z Z_i$ oder $Z_j <_Z Z_2$ lauten, sondern $Z_i <_Z Z_1$ und $Z_2 <_Z Z_j$.

Ist dabei das Problem als unlösbar eingestuft worden, so ist der Problemlösevorgang nach der Analyse bereits abgeschlossen. Sonst werden die ermittelten Eigenschaften (Operatorblockaden und Zielordnungen) an das Planungssystem weitergeleitet.

Im folgenden Kapitel wird gezeigt, wie diese analysierten Eigenschaften während des Planungsvorgangs genutzt werden.

Kapitel 4

Verarbeitung der Analyseergebnisse in CAPlan

Wie schon erwähnt, stellen die analysierten Eigenschaften aufbereitetes Wissen über die Domäne und das Problem dar. Dieses kann direkt dazu eingesetzt werden, den Planungsvorgang dahingehend zu steuern, daß der Suchraum eingegrenzt wird. Dabei dürfen allerdings keine Lösungen verloren gehen, deshalb werden auch nur **notwendige** Zielordnungen betrachtet. Würden auch Zielordnungen berücksichtigt, die nur in vielen aber nicht unbedingt in allen Lösungen gelten, so könnten genau die Lösungen, in denen die Ordnungen nicht gelten, verloren gehen. Es ist auch vorstellbar, daß nicht notwendige Zielordnungen betrachtet werden, z.B. um auf bestimmte Lösungen zu fokussieren, dies wird hier jedoch nicht behandelt.

Es gibt mehrere verschiedene Möglichkeiten, die Problemeigenschaften, wie Operatorblockaden und Zielordnungen, zu nutzen. Hier werden die wichtigsten getrennt für das generative und das fallbasierte Planen vorgestellt.

4.1 Verarbeitung beim generativen Planen

4.1.1 Operatorblockaden

Es wird ein Planungsvorgang für ein Problem P betrachtet, welches u.a. ein Ziel Z_1 enthält. Während des Planungsvorgangs müssen immer wieder bestimmte Arten von Entscheidungen getroffen werden. Dazu gehört auch die Auswahl eines Operators für ein bestimmtes (Unter-)Ziel Z. Dabei entscheidet das Planungssystem, welcher Operator gewählt wird, um die Gültigkeit von Z zu garantieren. Dafür kommen alle Operatoren in Frage, welche einen Effekt besitzen, der mit Z unifizierbar ist. Diese Operatoren bilden die Konfliktmenge für die Entscheidung. Mit dem Wissen von Operatorblockaden können diese Konfliktmengen verkleinert werden. Dies ist mit folgendem einfachen Test möglich:

> Wenn ein Operator für die Lösung von Z gesucht wird und Z ist gleich Z_1 oder Z ist ein Unterziel von Z_1 und es gilt *Block(Op, Z_1)*, dann kann *Op* aus der Konfliktmenge für Z entfernt werden.

Durch diese Konfliktmengenverkleinerung gehen keine Lösungen verloren, denn *Block(Op, Z_1)* bedeutet ja, daß *Op* nicht erfolgreich für die Lösung von Z_1 ausgeführt werden kann. Die Auswahl dieses Operators würde also irgendwann garantiert Backtracking zur Folge haben. Demnach kann mit dem Wissen von Operatorblockaden der Suchraum durch einfache Tests wirksam eingeschränkt werden, ohne die Vollständigkeit zu beeinträchtigen.

4.1.2 Teilzielordnungen

Für die Nutzbarkeit von Wissen über Teilzielordnungen in *CAPlan* werden zwei Möglichkeiten gegeben (siehe dazu [Weberskirch, 1998]): Ordnen der Teilziele in der Agenda und Einfügen von Ordnungen zwischen Planschritten.

Ordnung in der Agenda

Bei diesem Verfahren werden die Teilziele eines Problems zu Beginn des Planungsvorgangs so in die Agenda eingeordnet, daß folgende Bedingung erfüllt ist:

> Gibt es eine Ordnung $Z_1 <_Z Z_2$, so steht Z_1 vor Z_2 in der Agenda.

Werden die Ziele nicht zufällig aus der Agenda ausgewählt, sondern werden sie von vorne nach hinten abgearbeitet, so werden bei dieser Vorsortierung der Teilziele eines Problems diejenigen Ziele zuerst bearbeitet, welche bezüglich der berechneten Zielordnungen in der Lösung zuerst erfüllt werden. Diese Vorgehensweise schränkt zwar nicht den Suchraum ein, aber es gibt weniger Möglichkeiten, Fehler zu machen und somit werden im Mittel weniger Entscheidungen zurückgezogen (siehe dazu [Weberskirch, 1998]). Hier wird also das Wissen von Zielordnungen ausgenutzt, um statistisch bessere Entscheidungen zu treffen. Bei Domänen, die ohne Ordnungen nicht trivial serialisierbar sind, aber *modulo* der berechneten Ordnungen trivial serialisierbar sind, kann der Suchraum durch die Ordnung in der Agenda garantiert eingeschränkt werden. Dies wird im folgenden Kapitel genauer erläutert.

Einfügen von Planschrittordnungen

Ein Lösungsplan besteht aus einzelnen Schritten, welche partiell geordnet sind. Die Ordnungen zwischen den Schritten resultieren aus Entscheidungen, die während des Planungsvorgangs getroffen werden. So kann z.B. die Auflösung einer Interaktion eine Ordnung zwischen Planschritten zur Folge haben. Ist das Wissen von Teilzielordnungen vorhanden, so können während des Planungsvorgangs zusätzliche Ordnungen zwischen Planschritten eingefügt werden:

Ist in einer partiellen Lösung PL ein Operator O_1 der Establisher von einem Teilziel Z_1 (das heißt, daß O_1 ausgewählt wurde, um die Gültigkeit von Z_1 zu garantieren) und O_2 ist der Establisher von Z_2 und es gilt $Z_1 <_Z Z_2$, dann kann die Planschrittordnung $O_1 < O_2$ eingefügt werden.

Wird PL zu einer Lösung $\widetilde{PL}$ vervollständigt, so muß in $\widetilde{PL}$ $O_1 < O_2$ gelten, denn O_1, O_2 erfüllen ja direkt Z_1, Z_2 und somit würde sonst nicht $Z_1 <_Z Z_2$ gelten. Also muß diese Ordnung auf jeden Fall irgendwann während der Planung eingefügt werden. Es ist sinnvoll, dies möglichst früh zu tun, denn dadurch kann der Suchraum eingeschränkt werden, da bei der Vervollständigung von PL bei weiteren Entscheidungen das Vorhandensein solcher Ordnungen nützlich sein kann.

Dazu wird folgendes Beispiel gegeben. Abbildung 4.1 zeigt einen partiellen Plan für ein Problem mit den Zielen Z_1, Z_2. Weiterhin ist die Teilzielordnung $Z_1 <_Z Z_2$ bekannt. Daher wurde eine zusätzliche Planschrittordnung zwischen *OpC* und *OpD* eingefügt. Für die Lösung der Vorbedingung q von *OpD* wurde *OpB* ausgewählt und für die Vorbedingung p von *OpC* wurde *OpA* ausgewählt, wodurch eine Interaktion *IK* von *OpA* nach *q@OpD* entstand.

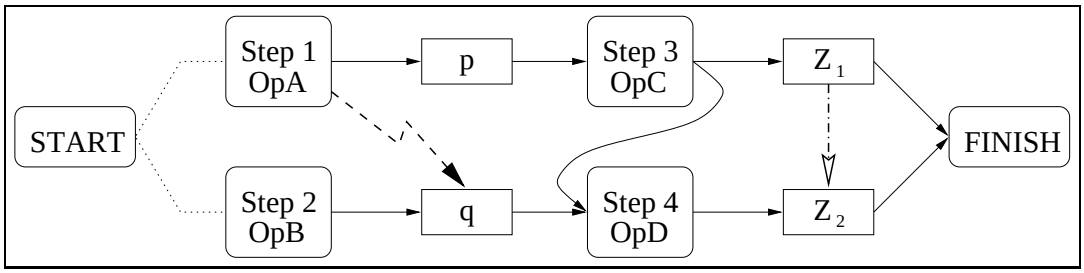

Abbildung 4.1: Einfügen von Ordnungen zwischen Planschritten

Da es keine Variablen gibt, ist *Separation* für *IK* nicht möglich. Also gibt es nun für den Planer zwei Möglichkeiten *IK* aufzulösen: *Promotion* oder *Demotion*. Dabei kann mit *Promotion* der partielle Plan nicht zu einer Lösung vervollständigt werden, da dadurch die Ordnung $OpD < OpA$ entstehen würde. Da außerdem $OpA < OpC$ gilt, läßt sich $OpD < OpC$ folgern. Dies kann nicht gelten, da $Z_1 <_Z Z_2$ gilt. Ohne die zusätzliche Ordnung $OpC < OpD$ würde dies jedoch nicht erkannt und somit müßte bei der Wahl von *Promotion* zu einem späteren Zeitpunkt des Planungsvorgangs Backtracking erfolgen.

Mit der zusätzlichen Ordnung $OpC < OpD$ wird bei der Wahl von *Promotion* allerdings direkt ein inkonsistenter Planschrittzyklus $OpA < OpC < OpD < OpA$ erkannt. Damit wird direkt die richtige Wahl, nämlich *Demotion*, getroffen und dadurch Backtracking zu einem späteren Zeitpunkt vermieden und somit der Suchraum wirksam eingeschränkt.

4.2 Verarbeitung beim fallbasierten Planen

Beim fallbasierten Planen werden Lösungen von ähnlichen Problem benutzt, um neue Probleme zu lösen. Das System besitzt dazu eine Fallbasis, in der eine Menge von Fällen (Probleme mit Lösung) enthalten ist. Soll nun ein neues Problem P gelöst werden, so müssen drei Schritte durchgeführt werden:

1. Ein oder mehrere zu P ähnliche Probleme müssen in der Fallbasis gefunden werden.
2. Die Lösungen der ähnlichen Probleme müssen auf P übertragen werden.
3. Der durch die Lösungsübertragung entstandene partielle Plan muß evtl. vervollständigt werden.

Für den 3. Punkt ergeben sich prinzipiell die gleichen Möglichkeiten, wie beim generativen Planen. Besonders effektvoll lassen sich Teilzielordnungen beim 1. Punkt verwenden. Dabei werden die Fälle u.a. durch ihre Teilzielordnungen charakterisiert und somit die Fallauswahl unterstützt. Dies wird im nächsten Abschnitt beschrieben.

4.2.1 Teilzielindizierte Fallbasis

Um für ein neues Problem P ähnliche Fälle zu finden, muß ein Ähnlichkeitsmaß für Probleme definiert werden. Dazu wird ein Problem durch seinen *Footprint* charakterisiert. Dieser enthält charakteristische Merkmale des Problems. Zielordnungen sind auch charakteristische Merkmale, denn oft entscheiden gerade die Interaktionen zwischen den Teilzielen, ob sich Lösungen wirklich übertragen lassen und damit, ob Probleme wirklich ähnlich sind. Also müssen ähnliche Probleme auch ähnliche Zielordnungen aufweisen. Um eine effiziente Fallauswahl zu ermöglichen, werden die Fälle bezüglich ihres *Footprint* indiziert. Sind nun Zielordnungen ein Teil des *Footprint*, so werden die Fälle u.a. bezüglich ihrer Zielordnungen in die Fallbasis einsortiert (siehe dazu auch [Hüllen, 1995]).
Für die Fallauswahl können nun mit den im letzten Kapitel vorgestellten Verfahren Zielordnungen des neuen, zu lösenden Problems berechnet werden. Dadurch wird die Fallauswahl nicht nur effizienter, da Probleme mit völlig anderen Zielordnungen erst gar nicht betrachtet werden müssen, sondern das Ergebnis wird auch besser. Das bedeutet, daß die ausgewählten Fälle ähnlicher zu P sind, denn die Ähnlichkeit der Zielordnungen wurde ja mitberücksichtig. Dadurch werden die Punkte 2. und 3. effizienter. Dieses hier prinzipiell vorgestellte Verfahren wird in [Muñoz-Avila, 1998] ausführlich behandelt.

Die vorgestellten Verfahren für das generative Planen wurden in *CAPlan* implementiert. Im folgenden Kapitel wird untersucht, inwieweit Teilzielordnungen mit den vorgestellten Analyseverfahren gefunden werden können und welche Effizienzsteigerungen sich damit in *CAPlan* erzielen lassen.

Kapitel 5

Testergebnisse

In diesem Kapitel werden zuerst die qualitativen Ergebnisse zusammengefaßt. Es wird also gezeigt, inwieweit die Analyseverfahren Teilzielordnungen in den verschiedenen Domänen berechnen können. Anschließend wird die Effizienz der Analyse und die aus der Analyse resultierende Effizienzsteigerung des Planungssystems betrachtet. Dabei werden Probleme mit und ohne Vorverarbeitung gelöst und die Problemlösezeiten untereinander sowie mit der Analysezeit verglichen. Alle verwendeten Domänen finden sich in der Planungsliteratur und werden im Anhang A.3 definiert. In diesem Kapitel werden die Domänen lediglich durch Operatorgraphabbildungen oder allgemeine Aussagen über bestimmte Eigenschaften charakterisiert.

5.1 Berechnung von Teilzielordnungen

Oft gibt es in den einzelnen Domänen *Klassen* von Ordnungen. Das sind bestimmte Mengen von Ordnungen zwischen Gruppen von Zielen, die eine bestimmte Eigenschaft der Domäne wiederspiegeln. Diese Klassen werden nun für die verschiedenen Domänen vorgestellt und es wird beschrieben, ob und wie diese durch die Analyse vorausberechnet werden können.

5.1.1 D^mS^n-Domänen

Diese Menge von künstlichen Domänen wurde erstmals in [Barrett und Weld, 1994] vorgestellt. Sie besitzen keinen direkten semantischen Hintergrund, stellen aber typische, für Planungssysteme schwierige Situationen dar. Dabei gibt es eine Reihe von Zielen ($G1 \ldots Gi$), die jeweils über n Stufen erreicht werden können. Das heißt, für jedes Ziel Gj gibt es genau n Operatoren ($Aj1..Ajn$), die der Reihe nach ausgeführt werden müssen.

Außerdem gibt m den Grad der Interaktionen an: D^0 bedeutet keine Interaktionen, D^1 bedeutet Interaktionen zum nächst kleineren Ziel und D^m bedeutet Interaktionen zu allen Zielen mit kleinerem Index.

Abbildung 5.1 zeigt einen Operatorgraph für ein Problem mit $i = 4$ in der D^1S^2-Domäne. In dieser Domäne gibt es auch noch Interaktionen von den *Aj2-Operatoren* zu den *Ik-Vorbedingungen*, diese wurden wegen der Übersichtlichkeit in dieser Abbildung nicht dargestellt. Sie haben zur Folge, daß alle Operatoren der 2. Stufe erst nach allen Operatoren der 1. Stufe ausgeführt werden können.

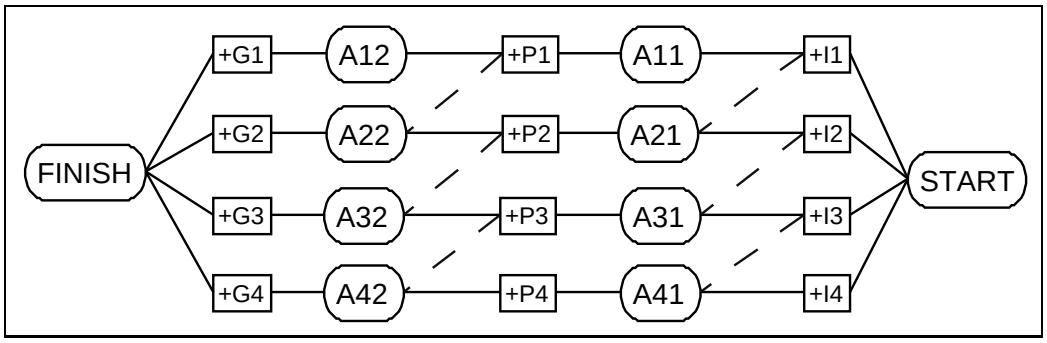

Abbildung 5.1: Problem mit vier Zielen aus der D^1S^2-Domäne

Für $m > 0$ sind in diesen Domänen alle Teilziele linear geordnet, es gilt dann: $G1 <_Z G2 <_Z \ldots <_Z Gi$. Diese Ordnungen resultieren aus den Interaktionen zu den Zielen mit kleinerem Index und werden durch das Verhinderungskriterium erkannt. Für D^m kann sogar die komplette transitive Hülle der oben aufgeführten Ordnungskette analysiert werden. Dadurch ändert sich jedoch das Gesamtergebnis nicht, da die transitive Hülle bei der Gesamtanalyse in jedem Fall berechnet wird. Diese Domänen stellen somit einen Idealfall dar: Alle Ziele sind notwendig linear geordnet und diese Ordnungen können alle analysiert werden.

5.1.2 $\theta_k D^m S^n$-Domänen

Diese Varianten der $D^m S^n$-Domänen wurden ebenfalls in [Barrett und Weld, 1994] vorgestellt. Die neu hinzukommende Schwierigkeit ist, daß die Ziele hier jeweils mit k verschiedenen Operatoren gelöst werden können, statt mit einem. Ein spezielles Ziel $G\text{-}a$ kann allerdings nur so gelöst werden, daß nur eine dieser k Varianten wirklich möglich ist. Abbildung 5.2 zeigt einen Operatorgraph für ein Problem mit $i = m = n = 1$ und $k = 2$.

Diese Domäne zeigt eine sehr typische Situation für viele Domänen mit realen Anwendungen auf: Bestimmte Ziele lassen sich prinzipiell auf verschiedene Weisen lösen, jedoch ist wegen Interaktionen zu anderen Zielen bei einem konkreten Problem letztendlich doch nur ein Teil dieser Varianten möglich. Auch in diesen Domänen sind die Ziele linear geordnet: $G\text{-}a <_Z G1 <_Z G2 <_Z \ldots <_Z Gi$. Wieder können alle Ordnungen durch die in dieser Arbeit vorgestellten Kriterien berechnet werden. Bei den Tests zur Effizienzsteigerung zeigt sich, daß den Ordnungen $G\text{-}a <_Z Gj,\ j=1..i$ besondere Bedeutung zukommt (siehe dazu Abschnitt 5.2.2).

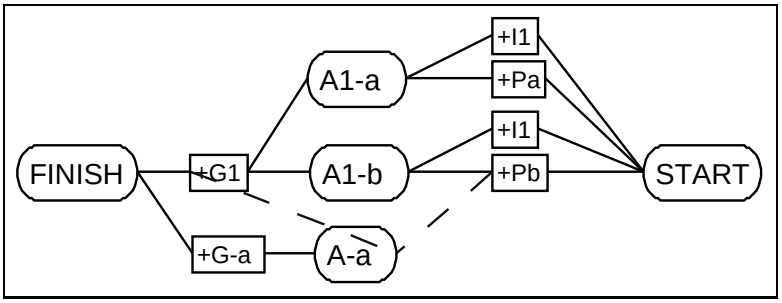

Abbildung 5.2: Operatorgraph für ein Problem in der $\theta_2 D^1 S^1$ - Domäne

5.1.3 Workpiece-Domäne

In der Workpiece-Domäne sind Objekte und Aktionen modelliert, welche es erlauben, für rotationssymmetrische Drehteile die entsprechenden Arbeitspläne zur Fertigung solcher Teile auf CNC-Drehmaschinen zu erstellen. Dafür gibt es eine Vielzahl primitiver Aktionen, deren Anwendung die verschiedenen zu fertigenden *Aspekte* wie Hinterschneidungen, Bohrungen oder Gewinde bei den Drehteilen realisieren. Um solche Aspekte zu fertigen, muß das Werkstück entsprechend aufgespannt und mit einem passenden Werkzeug bearbeitet werden.

Die Tests haben ergeben, daß in dieser Domäne zwei *Klassen* von Ordnungen erkannt werden: Ordnungen durch die Eigenschaft $Aspekt_B$ ist Unterbereich von $Aspekt_A$ und Ordnungen durch Interaktionen, die sich auf das Aufspannen des Drehteils beziehen.

Ordnungen durch Unterbereiche

Die zu fertigenden Werkstücke besitzen mehrere Aspekte. Häufig treten dabei Situationen auf, in denen ein Aspekt B ein Unterbereich eines anderen Aspektes A ist. So kann z.B. in einem Bereich eine Hinterschneidung erfolgen (Aspekt A) und außerdem soll die Oberfläche in einem Teil dieses Bereiches behandelt werden (Aspekt B). In diesem Fall muß die Fertigung von A **vor** der Fertigung von B erfolgen, da A eine Vorbedingung für die Fertigung von B ist. Da die Aspekte Teilziele des Problems darstellen, gilt also $A <_Z B$. Alle notwendigen Zielordnungen dieser *Klasse* werden in der *Workpiece-Domäne* von dem Mitlösungskriterium erkannt (es wird zwar das kombinierte Gesamtkriterium getestet, aber der Teil von *Mitlösung* ist hier entscheidend).

Ordnungen durch Interaktionen beim Aufspannen

Es gibt eine Menge Restriktionen bezüglich des Aufspannens des Werkstücks. So kann z.B. nicht an einem Bereich aufgespannt werden, in dem bereits ein Gewinde gefertigt wurde. Wenn nun bei einem Problem ein Aspekt A vorliegt, der nur durch Aufspannen an einem Bereich gefertigt werden kann, in dem ein Gewinde (Aspekt B) gefertigt werden soll, so muß die Fertigung von A **vor** der Fertigung von B erfolgen, da es nachher nicht mehr möglich ist. Ordnungen dieser Art können durch das Verhinderungskriterium berechnet werden.

5.1.4 Blocksworld-Domäne

Für diese bekannte Domäne gibt es mehrere Varianten mit unterschiedlich vielen Operatoren. Obwohl diese Domäne recht einfach erscheint, bergen alle Varianten erhebliche Schwierigkeiten für die Analyse von Teilzielordnungen. Obwohl in vielen Problemen notwendige Zielordnungen vorkommen, können die in dieser Arbeit vorgestellten Kriterien keine berechnen. Die Gründe dafür werden nun mit dem Beispiel PRO_A dargestellt. Abbildung 5.3 zeigt die Problemstellung von PRO_A (in der *Blocksworld-Domäne* lassen sich Probleme mit solchen Bildern wesentlich besser veranschaulichen als durch eine Auflistung von Prädikaten).

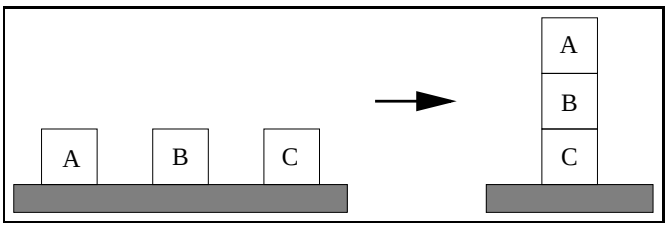

Abbildung 5.3: Blocksworld-Problem PRO_A

Abbildung 5.4 zeigt den Operatorgraph für PRO_A. Dabei wurden die zahlreichen rekursiven Kanten nicht dargestellt, damit der Graph übersichtlich bleibt. Hier wird eine Version der Domäne mit zwei Operatoren betrachtet. Für die bekanntere Version mit vier Operatoren (eine Definition dafür findet sich z.B. in [Russell und Norvig, 1995]) gilt jedoch prinzipiell das gleiche.

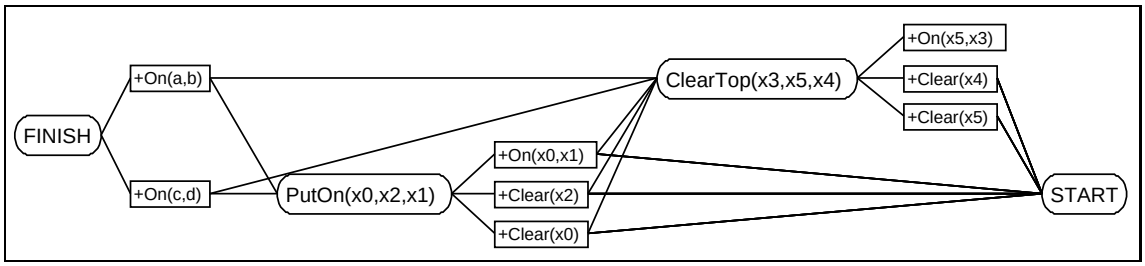

Abbildung 5.4: Operatorgraph für PRO_A

Für dieses Beispiel gilt natürlich *On(b, c)* $<_Z$ *On(a, b)*. Die beiden Ziele *On(a, b)* und *On(b, c)* haben identische Eigenschaften bezüglich der Struktur des Operatorgraphen. Sie lassen sich beide durch die Operatoren *ClearTop* und *PutOn* erfüllen und unterscheiden sich nur durch ihre Konstanten. Somit ist es kaum möglich, durch eine Strukturanalyse des Operatorgraphen die notwendige Zielordnung *On(b, c)* $<_Z$ *On(a, b)* in PRO_A zu erkennen.

Es gilt sogar, daß alle Probleme in Blocksworld Operatorgraphen besitzen, die dem in Abbildung 5.4 bezüglich der Topologie sehr ähnlich sind. Das kommt daher, daß es nur zwei Arten von Zielen gibt (*On(x, y)* und *Clear(z)*) und nur zwei Operatoren (*PutOn* und *ClearTop*). Außerdem können beide Operatoren beide Arten von Zielen lösen. Daher sind die

Operatorgraphen für diese Domäne bezüglich der Eigenschaften der Zielordnungen nicht sehr aussagekräftig. Dies bedeutet, daß weder das Prinzip der vorverarbeitenden Analyse, noch die grundsätzlichen Ideen der vorgestellten Kriterien für diese Domäne ungeeignet sind. Vielmehr ist die Wahl der Operatorgraphen als Berechnungsgrundlage ungünstig. Für eine solch hochgradig rekursive Domäne (es gibt sehr wenig unterschiedliche Operatoren, aber eine hohe Anzahl von Operatoranwendungen eines Domänenoperators in Lösungen) müßten die Zykel in den Operatorgraphen aufgelöst werden, also mehrere Operatoranwendungen eines Operators dargestellt werden, damit Abhängigkeiten zwischen den einzelnen Operatoranwendungen besser erkennbar sind. Dies kommt aber dem Lösen des Problems sehr nahe und ist somit als Vorverarbeitung weniger geeignet.

5.1.5 Zusammenfassung der qualitativen Ergebnisse

In den untersuchten künstlichen Domänen sind die Teilziele durch notwendige Ordnungen linear geordnet. Dabei können alle Ordnungen berechnet werden. Diese Domänen stellen einen Extremfall dar, der in den meisten Domänen mit realen Anwendungen nicht vorhanden ist.

Dagegen sind die Teilziele in der *Workpiece-Domäne* und in der *Blocksworld-Domäne* durch notwendige Ordnungen partiell geordnet. Während in der *Workpiece-Domäne* der größte Teil dieser Ordnungen analysiert werden kann, gelingt dies in der *Blocksworld-Domäne* nicht, da die Operatorgraphen keine geeignete Berechnungsgrundlage für diese Domäne darstellen.

5.2 Effizienzsteigerung durch Vorverarbeitung

In den einzelnen Domänen wurde jeweils eine Menge von Problemen ohne Vorverarbeitung gelöst und im Vergleich dazu wurden sie analysiert und dann mit dem analysierten Wissen gelöst. Dabei wurden die Berechnungszeiten für die jeweiligen Problemlösungen und für die Analyse verglichen. Bei der Problemlösung wurde eine zielgerichtete Auswahl für das als nächstes zu bearbeitende (Unter-)Ziel verwendet. Das heißt, daß die Agenda als Stack realisiert wird. Das nächste zu bearbeitende Ziel wird von oben vom Stack genommen und die evtl. neu entstehenden Unterziele werden oben auf den Stack gelegt. Diese zielgerichtete Auswahl ist für SNLP-Planer oft erfolgreicher, als eine zufällige Auswahl der Ziele. Damit die Reihenfolge der Ziele eines Problems in seiner Definitionsbeschreibung nicht die Ergebnisse beeinflußt, wurden die Ziele zu Beginn der Planung in zufälliger Reihenfolge eingefügt. So konnte die zielgerichtete Auswahl genutzt werden, ohne daß durch die Definition des Problems implizit Wissen über Zielreihenfolgen entsteht. Die Probleme wurden zwar nicht mit allen möglichen Permutationen gelöst, aber es wurden die Zeiten für eine große Anzahl an Lösungen mit zufälliger Zielsortierung gemittelt. Bei der Problemlösung mit Analyse wurden die zufällig sortierten Teilziele durch evtl. vorhandene Ordnungen natürlich neu sortiert, siehe dazu Kapitel 4.1.2.

5.2.1 $D^m S^n$-Domänen

Abbildung 5.5 zeigt die Ergebnisse für die D^1S^2-Domäne. Dabei ist zu beachten, daß in dieser und in allen folgenden Abbildungen die Berechnungszeit für die Problemlösung mit Analyse der gesamten Problemlösungszeit entspricht. Also ist dies die Summe der Zeiten von Analyse und Problemlösung mit analysiertem Wissen. Abbildung 5.5 läßt folgendes erkennen:

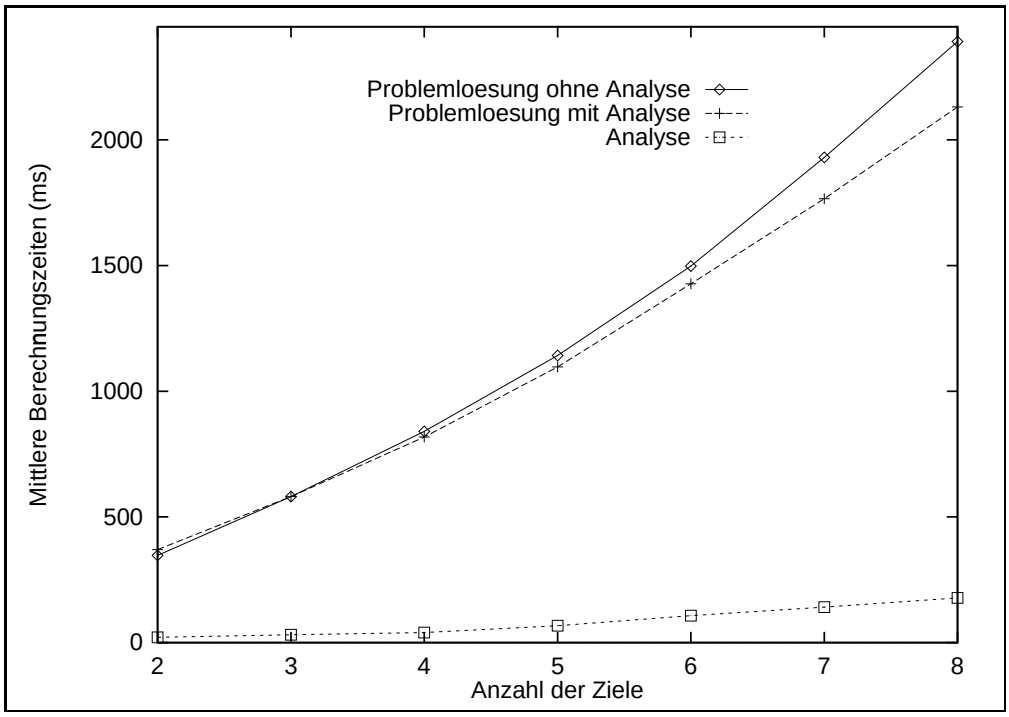

Abbildung 5.5: Ergebnisse für die D^1S^2-Domäne

- Die Berechnungszeiten der Analyse sind für alle Problemgrößen wesentlich geringer als die der Problemlösung. Das heißt, daß die Idee der Vorverarbeitung, welche wesentlich weniger aufwändig als die eigentliche Lösung sein sollte, verwirklicht wurde.
- Die Berechnungszeiten für die Problemlösung mit Analyse steigen weniger stark als die ohne Analyse, so daß für nicht trivial kleine Probleme ein Vorteil entsteht, der mit steigender Problemgröße wächst.
- Diese Vorteile sind aber auch für größere Probleme relativ gering. Dies ist um so bemerkenswerter, da in dieser Domäne die Analyse sehr erfolgreich ist (es werden Ordnungen analysiert, so daß alle Teilziele linear geordnet sind).

Wie läßt sich der letzte Punkt erklären? In [Barrett und Weld, 1994] wurde gezeigt, daß die $D^m S^n$-Domänen für SNLP-Planer trivial serialisierbar sind. Dies bedeutet, daß sich in einer solchen Domäne jeder partielle Plan PL für ein Problem P, welcher einen Teil der

Ziele von P löst, so zu einem Lösungsplan für P vervollständigen läßt, daß dabei keine Entscheidungen von PL zurückgenommen werden müssen. Dadurch sind Probleme in den D^mS^n-Domänen für SNLP-Planer nicht sehr schwierig, da die Reihenfolge der Teilziele nicht von entscheidender Bedeutung ist. Es ist immer schwierig, ein System für Probleme zu verbessern, für die es schon vorher sehr gut geeignet ist. Auf diesem Hintergrund ist die geringe Verbesserung durch die Analyse nicht mehr verwunderlich. In den später folgenden $\theta_k D^mS^n$-Domänen sieht dies ganz anders aus. Zuvor werden jedoch noch die Ergebnisse der D^mS^2-Domäne betrachtet, siehe dazu Abbildung 5.6.

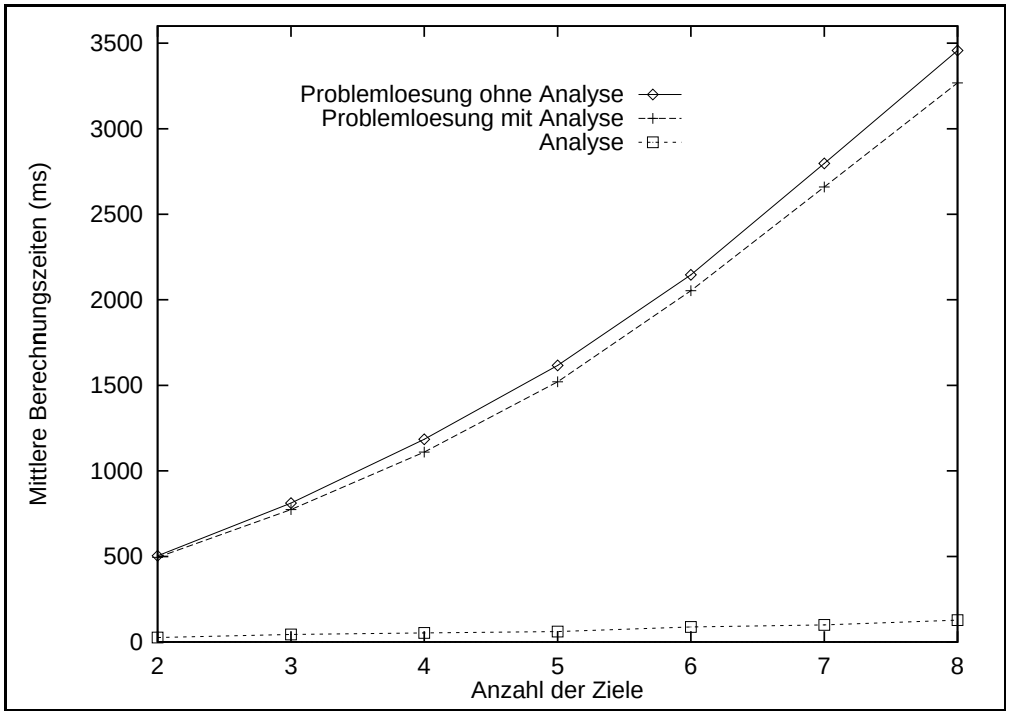

Abbildung 5.6: Ergebnisse für die D^mS^2-Domäne

In dieser Domäne gibt es wesentlich mehr Interaktionen, nämlich zu allen Zielen mit kleinerem Index. Wie schon in Abschnitt 5.1.1 beschrieben, ändert sich das Gesamtergebnis der Analyse nicht, da in jedem Fall die gesamte transitive Hülle der Ordnungskette berechnet wird. Der informierte Planer (Planen mit Analysewissen) erhält also die gleiche Information über Zielordnungen, wie bei der D^1S^2-Domäne, dagegen hat der uninformierte Planer (Planen ohne Analysewissen) mehr indirekte Informationen über die Zielordnungen als in der D^1S^2-Domäne, da es mehr Interaktionen gibt. Somit ist der Vorteil der Analyse sogar noch etwas geringer, was sich mit den Ergebnissen von [Barrett und Weld, 1994] deckt. Insgesamt sind die Problemlösezeiten etwas höher, da die Berücksichtigung der Vielzahl der Interaktionen bei der Planung aufwendig ist.

5.2.2 $\theta_k D^m S^n$-Domänen

Diese Domänen sind für SNLP-Planer nicht trivial serialisierbar und somit wesentlich schwieriger. Um diese Problematik zu verstehen, wird nochmals Abbildung 5.2 betrachtet. Wird für die Lösung von *G1* der Operator *A1-b* gewählt, so läßt sich dieser partielle Plan nicht mehr ohne Rückzug dieser Entscheidung zu einem Lösungsplan vervollständigen, denn die notwendige Auswahl von *A-a* für *G-a* hat zur Folge, daß *G1* nur noch mit *A1-a* gelöst werden kann.

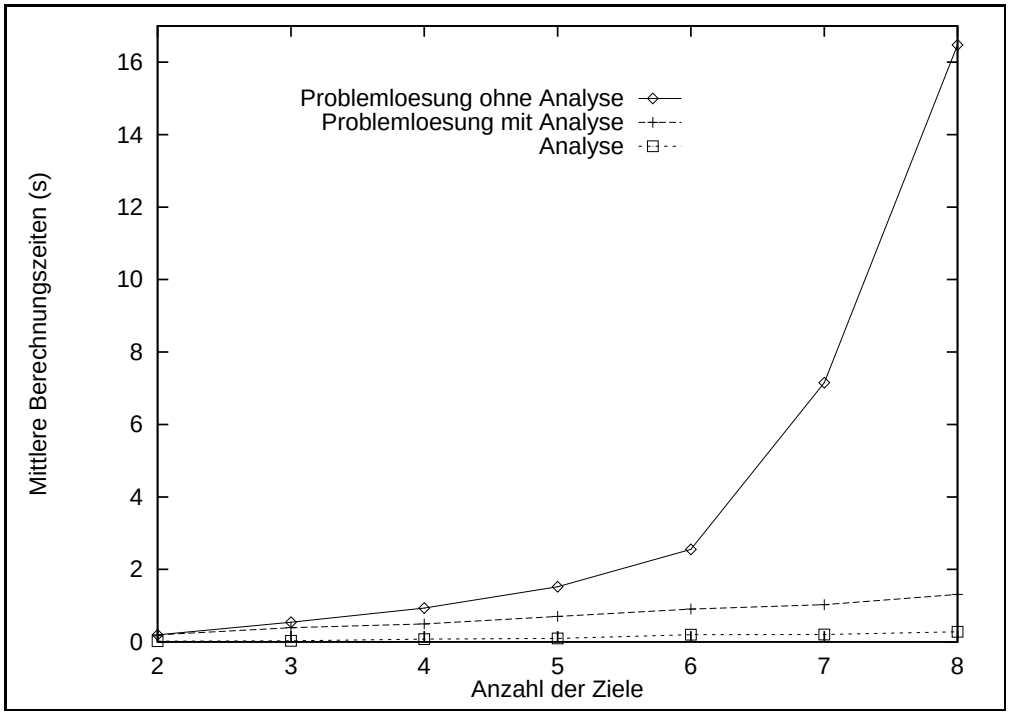

Abbildung 5.7: Ergebnisse für die $\theta_2 D^1 S^1$-Domäne

Abbildung 5.7 zeigt die Ergebnisse für Probleme aus der $\theta_2 D^1 S^1$-Domäne, dabei ist folgendes zu erkennen:

- Wie bei den vorherigen Tests ist die Berechnungszeit für die Analyse sehr gering gegenüber der Problemlösezeit. Im Vergleich zur Problemlösezeit ohne Analyse ist sie für größere Probleme sogar vernachlässigbar.
- Die Berechnungszeiten für die Problemlösung ohne Analyse zeigen, bezogen auf die Problemgröße, exponentielles Verhalten.
- Dagegen verhalten sich die Berechnungszeiten mit Analyse fast linear. Daraus resultiert für größere Probleme ein extremer Vorteil.

Dieses auffällige Ergebnis läßt sich erklären: In den $\theta_k D^m S^n$-Domänen gibt es u.a. notwendige Zielordnungen der Form: *G-a* $<_Z$ *Gj, j=1..i* und diese können auch analysiert

werden (siehe Abschnitt 5.1.2). Diese Ordnungen werden nun *OrdG-a* genannt. Außerdem gilt, daß die Domänen *modulo OrdG-a* trivial serialisierbar sind. Das bedeutet, daß, wenn die Teilziele in der durch *OrdG-a* angegebenen Reihenfolge gelöst werden, alle möglichen partiellen Pläne *PL*, die einen Teil der Ziele lösen, sich zu einem Lösungsplan für das Problem vervollständigen lassen, ohne daß dabei Entscheidungen von *PL* zurückgezogen werden müssen. Dies läßt sich so begründen: In den $\theta_k D^m S^n$-Domänen muß wegen den Ordnungen *OrdG-a* das Ziel *G-a* vor den Zielen *Gj* gelöst werden. Für die Lösung von *G-a* muß *A-a* gewählt werden. Nun kann für ein *Gj* der Operator *Aj-b* nicht mehr gewählt werden, da dabei sofort unlösbare Interaktionen entstehen. Also leiten hier die Ordnungen das Planungssystem direkt auf den richtigen Weg.

Probleme aus den $\theta_k D^m S^n$-Domänen gehören also mit dem Wissen der Ordnungen *OrdG-a* in eine echt leichtere Problemklasse als ohne dieses Wissen. Daher ist hier die Anwendung der vorverarbeitenden Analyse besonders effektiv.

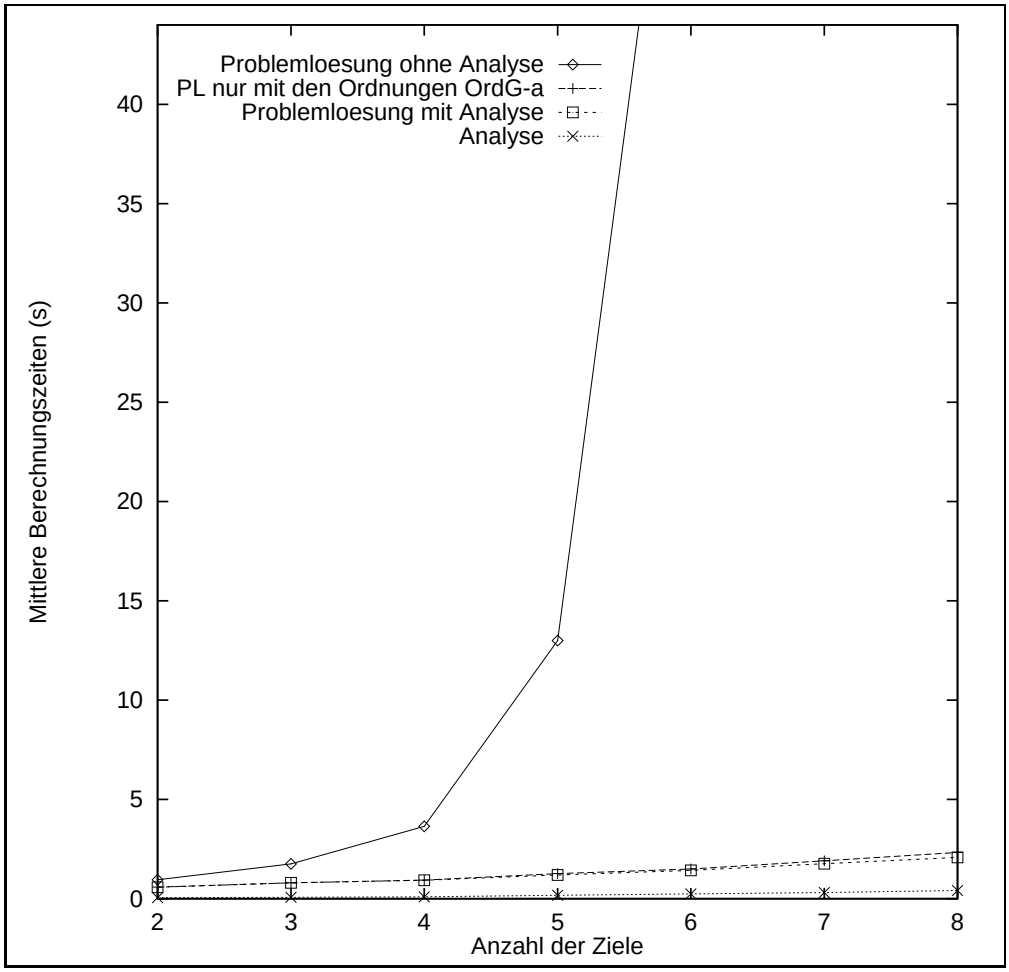

Abbildung 5.8: Ergebnisse für die $\theta_3 D^1 S^1$-Domäne

Abbildung 5.8 zeigt die Ergebnisse für die $\theta_3 D^1 S^1$-Domäne. In dieser Domäne gibt es drei

Möglichkeiten, die Ziele *Gj* zu lösen und nur mit einer ist ein Lösungsplan möglich. Daher ist die Wahrscheinlichkeit, daß das Planungssystem ohne das Wissen von *OrdG-a* eine falsche Möglichkeit wählt, noch größer. Dementsprechend fallen die Ergebnisse noch extremer aus:

- Die Werte für die Problemlösung ohne Analyse steigen mit wachsender Problemgröße so stark, daß sie ab sechs Zielen nicht mehr dargestellt wurden. Bei sechs Zielen wurden durchschnittlich 64s benötigt, bei sieben Zielen 267s und bei acht Zielen 970s.

- Es werden zwar alle nötigen Zielordnungen analysiert, aber wenn dem Planungssystem nur die Ordnungen *OrdG-a* gegeben werden, so wird fast die gleiche Effizienzsteigerung erreicht. Dies zeigen die beiden mittleren Kurven, die fast identisch sind. Entscheidend ist also nur die Teilmenge *OrdG-a* aller notwendigen Zielordnungen. Mit Hinzunahme der Ordnungen $G1 <_Z G2 <_Z G3 \ldots$ wird die Berechnung nur unwesentlich effizienter. Diese Tatsache entspricht genau den Ergebnissen der $D^m S^n$-Domänen.

- Mit steigendem k werden die $\theta_k D^m S^n$-Domänen für SNLP-Planer immer schwieriger. Für das Problemlösen mit Analyse bleiben die Domänen auch bei wachsendem k unproblematisch, was der nahezu lineare Anstieg der Berechnungszeit wiederspiegelt.

5.2.3 Workpiece-Domäne

In [Muñoz-Avila und Weberskirch, 1996] wurde bewiesen, daß diese Domäne ohne Zielordnungen nicht trivial serialisierbar ist. Weiterhin wird ein Menge von Zielordnungen für Probleme beschrieben, mit denen die Probleme *modulo* dieser Ordnungen trivial serialisierbar sind. Diese Ergebnisse lassen vermuten, daß die Analyse von Zielordnungen in der *Workpiece-Domäne* eine große Effizienzsteigerung zur Folge hat. In der Tat zeigen die Untersuchungen in dieser Arbeit, daß die Effizienz durch die Vorverarbeitung deutlich gesteigert werden konnte, allerdings nicht so extrem, wie in den $\theta_k D^m S^n$-Domänen. Dies hat zwei Ursachen:

1. Es können zwar, wie in Kapitel 5.1.3 beschrieben, viele notwendige Zielordnungen berechnet werden, allerdings werden nicht alle in [Muñoz-Avila und Weberskirch, 1996] beschriebenen Ordnungen analysiert, so daß die Probleme nicht zwingend *modulo* der berechneten Zielordnungen trivial serialisierbar sind.

2. Viele Operatoren der *Workpiece-Domäne* sind sehr kompliziert und besitzen eine große Anzahl von *Constraints*. Daher ist die Problemlösung in dieser Domäne, selbst wenn nicht viel *Backtracking* nötig ist, oft sehr aufwendig.

Da für Probleme aus der *Workpiece-Domäne* unterschiedlich viele Ordnungen analysiert werden können und da sich die Schwierigkeit der Probleme nicht so einfach wie in den künstlichen Domänen durch die Anzahl der Ziele charakterisieren läßt, können die Ergebnisse nicht in vergleichbaren Kurven dargestellt werden. Daher werden hier die wichtigsten Durchschnittswerte für die getestete Problemmenge genannt:

- Die durchschnittliche Analysezeit betrug 2,6% der Problemlösezeit. Damit beträgt der Aufwand für die Vorverarbeitung nur ca. 1/40 der gesamten Problemlösung.
- Für einige Probleme konnte die Problemlösung deutlich, für andere gar nicht beschleunigt werden, so daß für letztere die gesamte Problemlösung mit Analyse sogar etwas länger dauerte, als ohne.
- Die Problemlösezeit mit Analyse betrugen zwischen 17% und 102% und im Mittel 65% der Problemlösezeit ohne Analyse. Damit konnte die Berechnungszeit durchschnittlich um immerhin ca. 1/3 gesenkt werden.

Die Ergebnisse fallen sehr unterschiedlich für die verschiedenen Probleme aus, so daß man zusammenfassend sagen kann, daß die Analyse im Durchschnitt zwar eine deutliche Effizienzverbesserung zur Folge hat, daß einzelne Probleme aber schwierig bleiben.

5.2.4 Zusammenfassung der quantitativen Ergebnisse

In allen Domänen, in denen notwendige Zielordnungen analysiert werden konnten, wurde die Effizienz von *CAPlan* durch die Vorverarbeitung der Probleme verbessert. Allerdings ist der Grad der Verbesserung sehr unterschiedlich. In Domänen, in denen die Reihenfolge, in der die Teilziele gelöst werden, nicht das Hauptproblem ist (wie in den D^mS^n-Domänen, sind die Effizienzgewinne nicht so deutlich, wie in Domänen, in denen die Reihenfolge der Teilziele von entscheidender Bedeutung ist (wie in den $\theta_k D^m S^n$-Domänen). Diese Domänen werden durch die Analyse von Teilzielordnungen für SNLP-Planer in eine einfachere Klasse gebracht.

Kapitel 6

Zusammenfassung

In den vorangegangenen Kapiteln wurden Analyseverfahren vorgestellt, die bestimmte Eigenschaften von zu lösenden Problemen, primär notwendige Zielordnungen, berechnen. Es wurde gezeigt, daß die Analyseergebnisse, die die hier vorgestellten Kriterien produzieren, korrekt sind. Die Ergebnisse dieser Vorverarbeitung werden als zusätzliches Wissen bei der Lösung des entsprechenden Problems verwendet. Dies wurde für das generative Planen sowie für das fallbasierte Planen vorgestellt. Anschließend wurde getestet, in wie weit notwendige Zielordnungen in den verschiedenen Domänen gefunden wurden und welche Effizienzsteigerungen sich bei *CAPlan* damit erzielen lassen.

6.1 Bewertung der Ergebnisse

Der Erfolg der in dieser Arbeit vorgestellten Verfahren hängt stark von der betrachteten Domäne ab. Für viele Domänen ließen sich alle oder die meisten notwendigen Zielordnungen berechnen. Allerdings gibt es Domänen, in denen die vorgestellten Kriterien überhaupt keinen Erfolg haben. Die Ursache dafür ist erkennbar: Wenn die verschiedenen Teilziele durch unterschiedliche Operatoren erfüllt werden können, wie z.B. bei der Workpiece-Domäne oder bei den erstmals in [Barrett und Weld, 1994] vorgestellten künstlichen Domänen, so lassen sich alle oder die meisten vorhandenen Ordnungen berechnen. Wenn aber die Teilziele größtenteils durch einen oder wenige Operatoren erfüllt werden, wie z.B. bei der Blocksworld-Domäne, so werden kaum Ordnungen erkannt. Dies liegt hauptsächlich daran, daß die Operatorgraphen für solche Domänen nicht die geeignete Darstellung aller potentiellen Lösungen sind, da sie über Zielordnungen nicht viel aussagen.

In den Domänen, in denen Zielordnungen berechnet werden konnten, wurden unterschiedliche Effizienzsteigerungen erreicht. Der Grad der Effizienzsteigerung hängt davon ab, wieviel *einfacher* die Domäne mit den Ordnungen wird. Es gibt Domänen, die auch ohne Ordnungen *einfach* sind und es gibt Domänen, die auch mit Ordnungen *schwierig* sind. In beiden Fällen ist der Vorteil der Analyse gering. Wenn allerdings die Domäne durch Ordnungen einfacher wird (wie z.B. bei den $\theta_k D^m S^n$-Domänen), so ist der Effizienzgewinn groß.

Der Idealfall für die vorgestellten Verfahren ist somit die Situation bei den $\theta_k D^m S^n$-Domänen. Die Probleme sind ohne Ordnungen nicht trivial serialisierbar, also *schwierig*, und sie sind *modulo* aller notwendigen Zielordnungen trivial serialisierbar, also relativ *einfach*. Außerdem lassen sich alle notwendigen Zielordnungen berechnen.

Die Testergebnisse haben weiterhin gezeigt, daß, wenn die Ziele nur partiell geordnet sind oder nicht alle sondern nur ein Teil der notwendigen Zielordnungen berechnet werden (wie bei der *Workpiece-Domäne)*, trotzdem eine deutliche Effizienzsteigerung möglich ist. Somit hat also eine Abweichung vom Idealfall **nicht** zur Folge, daß der Nutzen übermäßig stark abnimmt, was sehr wichtig für die Anwendbarkeit der Verfahren ist.

Insgesamt sind folgende Voraussetzungen nötig, damit die vorgestellten Verfahren Erfolg haben:

1. Es müssen Abhängigkeiten zwischen den Zielen der Probleme in der betrachteten Domäne vorhanden sein, die Zielordnungen zur Folge haben. Dies ist für schwierige Domänen gerade typisch.
2. Die Domäne muß eine Form haben, in der sich solche Ordnungen mit Operatorgraphen berechnen lassen. Viele Domänen mit realen Anwendungen besitzen eine Vielzahl von Operatoren, also ist die problematische Situation in der *Blocksworld-Domäne* nicht so typisch.
3. Das Problemlösen in der Domäne muß durch Ordnungen *einfacher* werden. Dies gilt zumindest für viele Domänen.

Wenn für eine Domäne nicht bekannt ist, ob sie die oben genannten Voraussetzungen erfüllt, ist es trotzdem sinnvoll, die Vorverarbeitung anzuwenden, denn eine Analyse mit wenig Ergebnissen hat kaum negativen Einfluß auf die gesamte Problemlösezeit, da der Aufwand für den Aufbau und die Analyse der Operatorgraphen sehr gering ist.

6.2 Vergleich mit anderen Arbeiten

Natürlich gibt es eine große Anzahl von Möglichkeiten, den Planungsvorgang zu beschleunigen. Die meisten Ansätze versuchen den Suchraum einzuschränken. In der Regel wird zusätzliches Wissen genutzt, um die Suche zu steuern. Dabei sind folgende Aspekte zu unterscheiden:

- Welche Art von Wissen wird verwendet (allgemeines Wissen über Suche/Planung, Domänenwissen, problemspezifisches Wissen, ...)?
- Welche Form hat das Wissen bzw. wie wird es verarbeitet (direkt kodiert in der Kontrollstrategie, Regeln, Heuristiken, ...)?
- Woher kommt das Wissen (Eingabe durch Experten, Lernen von Beispielen, Analyse von Domänen oder Problemen, ...)?

Nun werden die Verfahren dieser Arbeit mit *STATIC*, einem Domänenanalyseverfahren mit *PSGs* von *Oren Etzioni* verglichen, da es einige Gemeinsamkeiten gibt. Bei einer Einordnung bezüglich der oben genannten Aspekte werden aber auch entscheidende Unterschiede deutlich.

Hier wird von *STATIC* nur die grundsätzliche Vorgehensweise vorgestellt, eine genaue Beschreibung findet sich in [Etzioni, 1991; Etzioni, 1993]. *STATIC* aquiriert Kontrollwissen in Form von domänenspezifischen Regeln durch Analyse der jeweiligen Domäne. Dafür werden *PSGs* (problem space graphs) für alle möglichen Ziele in einer Domäne gebildet. Diese werden analysiert, wobei Kontrollregeln für das zustandsbasierte Planungssystem *PRODIGY* (eine Beschreibung dafür findet sich in [Blythe *et al.*, 1992; Fink und Veloso, 1995]) abgeleitet werden. Mit diesen Kontrollregeln wird *PRODIGY* beschleunigt.

Gemeinsamkeiten

- Ziel der Verfahren ist jeweils, daß bei der Planung zusätzliches Wissen verwendet wird, welches vorher automatisch aquiriert wird.
- Es werden jeweils Eigenschaften und Abhängigkeiten von Operatoren der Domäne untersucht. Der *semantische Inhalt* des daraus gefolgerten Wissens ist ähnlich (z.B. werden jeweils Teilzielordnungen betrachtet).
- Die verwendeten Strukturen, die als Analysegrundlage dienen, haben einige Gemeinsamkeiten, denn wie bei den Operatorgraphen gibt es bei den *PSGs* auch zwei Arten von Knoten, für Prädikate und Operatoren. Die Graphen weisen eine analoge UND/ODER-Struktur auf.
- Schließlich hängt die Qualität der Ergebnisse ebenfalls stark von der betrachteten Domäne ab. Auch die Tatsache, daß rekursive Domänen meistens erheblich schwieriger sind, ist gemeinsam.

Unterschiede

- Das Wissen in *STATIC* ist spezifisch für eine ganze Domäne und nicht für ein Problem, wie in dieser Arbeit. Der Vorteil besteht darin, daß die Analyse nur einmal für eine Domäne erfolgen muß und nicht für jedes zu lösende Problem. Aber dafür können bei einer Analyse des Problems sowohl Informationen über die Domäne, als auch über das Problem selber genutzt werden. Somit kann auf für das Problem spezifische Eigenschaften fokussiert werden, während bei *STATIC* von einer gleichverteilten Menge von Problemen ausgegangen werden muß. Außerdem haben die Tests ergeben, daß der Zeitaufwand der Problemanalyse im Vergleich zur Problemlösung in der Regel vernachlässigbar ist, während die Analyse einer Domäne für alle möglichen Probleme aufwendiger ist.
- In *STATIC* wird das Wissen durch Kontrollregeln für zustandsbasierte Planer dargestellt. Dies ist ein entscheidender Unterschied, denn in den produzierten Kontrollregeln gibt es Bedingungen für den aktuellen Zustand, in dem der Planer sich gerade

befindet. Daher läßt sich das durch *STATIC* gewonnene Wissen nicht für SNLP-basierte Planer wie *CAPlan* nutzen, da es dort keine Weltzustände, sondern nur Planzustände gibt. Dagegen können Informationen über Zielordnungen oder Operatorblockaden, wie sie in dieser Arbeit berechnet werden, von allen Planertypen auf verschiedene Weise genutzt werden (Kapitel 4 zeigt einige Möglichkeiten auf).

- Der Ansatz mit *PSGs* benötigt *Axiome* über gültige Zustände in der Domäne. Diese spielen eine entscheidende Rolle bei der Termination der *PSG*-Expansion. Solche *Axiome* können zwar für manche Domänen automatisch berechnet werden, allerdings haben Untersuchungen gezeigt, daß es oft sehr schwierig oder unmöglich ist, solche *Axiome* für eine Domäne zu bestimmen (Beispiel dafür ist die Workpiece-Domäne von *CAPlan* [Muñoz-Avila und Weberskirch, 1996]). Fehlen solche *Axiome*, so werden die *PSGs* einzelner Prädikate sehr groß und es wird extrem schwierig, aus ihnen noch verwertbare Informationen zu extrahieren.

Zusammenfassend läßt sich sagen, daß trotz inhaltlicher Gemeinsamkeiten entscheidende Unterschiede in der Gesamtkonzeption bestehen, so z.B. die Analyse einer Domäne im Gegensatz zur Analyse eines Problems sowie die Beschränkung von *STATIC* auf zustandsbasierte Planung.

6.3 Ausblicke

Die Analyseverfahren könnten dahingehend erweitert werden, daß die Ergebnisse für die oben erwähnten *problematischen Domänen* besser werden. Dazu müsste eine andere Berechnungsgrundlage als die Operatorgraphen gewählt werden.

In dieser Arbeit wurden nur notwendige Zielordnungen betrachtet. Eine andere Vorgehensweise läßt sich so beschreiben:

> Für ein Problem P werden möglichst viele Teilzielordnungen *Ord* so bestimmt, daß garantiert werden kann, daß es für P noch mindestens eine *gute* (z.B. minimale) Lösung gibt, bei der die Teilziele in der Ordnung *Ord* gelöst werden.

Bei diesem Verfahren wird natürlich der Lösungsraum stark eingeschränkt und auf ganz bestimmte Lösungen fokussiert. Aber auch so läßt sich der Suchraum einschränken und die Effizienz der Planung steigern. Dies haben bereits durchgeführte Tests belegt.

Operatorgraphen basieren auf STRIPS-Repräsentation. HTN-Planung stellt im Gegensatz dazu eine wesentlich mächtigere Domänenbeschreibungssprache zur Verfügung. Analysen wie die hier vorgestellten sind für HTN-Systeme bisher nicht bekannt, aber es ist denkbar, daß die Idee der Vorverarbeitung dafür ebenfalls anwendbar ist.

Anhang A

A.1 Eliminierung von Interaktionen

In [Smith und Peot, 1993] werden drei Theoreme formuliert, die besagen, wann potentielle Interaktionen in Operatorgraphen nicht berücksichtigt werden müssen, da entsprechende Interaktionen in Lösungen nicht auftreten können. Dazu werden drei Theoreme angegeben, die solche Situationen beschreiben. Das Theorem 3 lautet wie folgt:

> *Wenn eine Interaktion IK = OK → VK zwischen disjunktiven Zweigen eines Operatorgraphen liegt und es von OK nur einen Weg nach FINISH gibt, so muß IK nicht berücksichtigt werden. Dabei sind die Zweige von OK und VK disjunktiv, wenn der naheste gemeinsame Vorgänger ein Vorbedingungsknoten ist.*

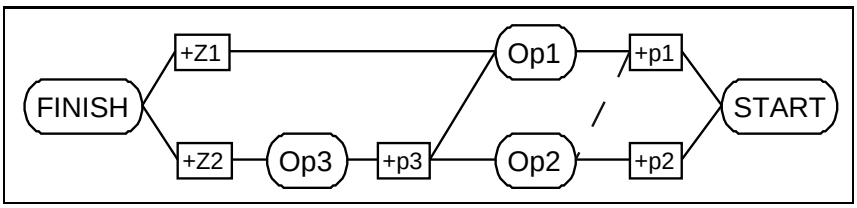

Abbildung A.1: Beispiel für Theorem 3

Abbildung A.1 zeigt einen Operatorgraph, in dem die Interaktion *IK = Op2 → p1* wegen Theorem 3 eliminiert werden würde, denn es gilt: Der naheste gemeinsame Vorgänger von *Op2* und *p1* ist *p3* und *p3* ist ein Vorbedingungsknoten. Außerdem gibt es nur einen Weg von *Op2* zu FINISH. Allerdings hat die potentielle Interaktion *IK* in diesem Beispiel entscheidende Bedeutung. Sie ist zwar für die Lösung von *Z2* unerheblich (und genau dies besagt obige Formulierung von Theorem 3, denn dort wird nur der gemeinsame Vorgänger *p3* betrachtet). Aber sie ist für die Lösung von *Z1* wichtig, denn *Z1* ist nach der Ausführung von *Op2* nicht mehr lösbar. Somit gilt in diesem Problem $Z1 <_Z Z2$ und diese Teilzielordnung wird von dem Verhinderungskriterium erkannt.

Ohne die Interaktion *IK* gilt dies nicht. Um die gesamte Bedeutung von *IK* zu erkennen, müssen **alle** nahesten gemeinsamen Vorgänger betrachtet werden. Von *Op1* aus führen zwei

Wege nach FINISH und ein weiterer nahester gemeinsamer Vorgänger von *Op2* und *p1* ist somit FINISH und dies ist kein Vorbedingungsknoten. Demnach ist folgende Formulierung von Theorem 3 korrekt:

> *Wenn eine Interaktion $IK = OK \rightarrow VK$ zwischen disjunktiven Zweigen eines Operatorgraphen liegt und es von OK nur einen Weg nach FINISH gibt, so muß IK nicht berücksichtigt werden. Dabei sind die Zweige von OK und VK disjunktiv, wenn* **alle** *nahesten gemeinsamen Vorgänger Vorbedingungsknoten sind.*

A.2 Beweis des Verhinderungskriteriums

Der Beweis des Verhinderungskriteriums ist prinzipiell ähnlich zu dem des Mitlösungskriteriums. Dort wurde gezeigt, daß zur Lösung von Z_2 immer ein Operator ausgeführt wird, so daß Z_1 bereits vorher mitgelöst wird. Hier wird gezeigt, daß zur Lösung von Z_2 immer ein Operator ausgeführt wird, so daß sich Z_1 nicht mehr lösen läßt. Dazu wird zuerst folgendes Lemma bewiesen:

Lemma A.1 *Seien Z_1, Z_2 Ziele in P, dann gilt:*

OK wird in einem partiellen Plan PL für die Lösung von Z_2 ausgeführt.	$\wedge$
$\neg Pseudolösbar(\overline{OG}(Z_1, Z_2 \mid OK), Z_1)$	$\Rightarrow$
PL läßt sich nicht zu einem Lösungsplan für P vervollständigen, in dem Z_1 **nicht vor** *Z_2 gelöst wird.*	

Beweis (von Lemma A.1): Sei PL ein partieller Plan, in dem eine Anwendung von OK für die Lösung von Z_2 vorkommt und es gelte $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), Z_1)$. (I)

Nun wird gezeigt, daß sich PL nicht so zu einer Lösung $\widetilde{PL}$ von P vervollständigen läßt, daß dabei Z_1 nicht vor Z_2 gelöst wird.

Sei M_1 die Menge der Operatoren, die Z_1 lösen können. OK wird für die Lösung von Z_2 ausgeführt, also wird Z_2 erst nach der Ausführung von OK gültig. Soll Z_1 nicht vor Z_2 gelöst werden, so muß in $\widetilde{PL}$ ein Operator aus M_1 vorkommen, der hinter OK angeordnet wird. Sei oBdA O_1 ein solcher Operator. Aus (I) und Definition 3.3 folgt aber für O_1:

$Block(O_1, Z_1)$	$\vee$	(iii)
$O_1 = \text{START}$	$\vee$	(iv)
$\exists VK \in VOR_{\overline{OG}(Z_1, Z_2 \mid OK)}(O_1) : \neg Pseudolösbar(\overline{OG}(Z_1, Z_2 \mid OK), VK)$		(v)

Da O_1 für die Lösung von Z_1 ausgeführt wird, kann (iii) nicht gelten und da O_1 nach OK ausgeführt werden soll, kann (iv) nicht gelten, also gilt (v). Es gibt also eine Vorbedingung V_2 von O_1, für die gilt: $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), V_2)$. (II)

Wegen (II) gilt: V_2 ist in $\overline{OG}(Z_1, Z_2|OK)$ als ungültig markiert. Da $V_2 \neq Z_1$ folgt aus Definition 3.2, daß es eine Interaktion IK von OK nach V_2 gibt (wegen Zeile (a)) und diese läßt sich nicht durch *Separation* auflösen (wegen Zeile (b)). IK läßt sich auch nicht durch *Promotion* auflösen, denn O_1 soll ja nach OK ausgeführt werden. Somit ist nur *Demotion* möglich und damit muß V_2 auch nach OK gelöst werden. (III)

Sei M_2 die Menge der Operatoren, die V_2 lösen können. Sei oBdA O_2 ein solcher Operator, dann wird O_2 wegen (III) nach OK ausgeführt. Außerdem gilt wegen (II) für O_2:

$Block(O_2, Z_1)$ $\vee$ (iii)
$O_2 = \text{START}$ $\vee$ (iv)
$\exists VK \in VOR_{\overline{OG}(Z_1,Z_2|OK)}(O_2) : \neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), VK)$ (v)

Mit den Argumenten von oben gilt wiederum (v). Es gibt also eine Vorbedingung V_3 von O_2, für die gilt: $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), V_3)$. (IV)

Mit den Argumenten von oben entsteht wegen (IV) eine Interaktion von OK nach V_3, welche wiederum nur durch *Demotion* aufgelöst werden kann. Also muß ein weiterer Operator O_3 nach OK ausgeführt werden. Durch rekursive Anwendung der Definitionen entsteht dabei eine unendliche Kette $Z_1 \rightarrow O_1 \rightarrow V_2 \rightarrow O_2 \rightarrow V_3 \rightarrow \ldots$, denn es gibt keine Abbruchmöglichkeit. Somit läßt sich PL nicht zu einem Lösungsplan $\widetilde{PL}$ vervollständigen, in dem Z_1 nicht vor Z_2 gelöst wird. qed.

Nun wird gezeigt, daß, wenn *Verhinderung*(Z_2, Z_1, Z_2) gilt, in jeder Lösung ein Operator OK für die Lösung von Z_2 ausgeführt wird, so daß $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), Z_1)$ gilt. Somit wird Z_1 in dieser Lösung wegen obigem Lemma vor Z_2 gelöst.

Beweis (von Theorem 3.3): Sei P ein Problem mit Zielen Z_1, Z_2 und es gelte $Verhinderung_{VK}(Z_2, Z_1, Z_2)$. (I)

Nun wird gezeigt, daß in jeder Lösung PL von P ein Operator OK für die Lösung von Z_2 ausgeführt wird, so daß $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), Z_1)$ gilt.

Sei M_1 die Menge aller Operatoren, die einen Effekt besitzen, der mit Z_2 unifizierbar ist. Da Z_2 in PL gelöst werden muß, wird mindestens einer der Operatoren aus M_1 für die Lösung von Z_2 in PL ausgeführt. Sei oBdA $O_1 \in M_1$ ein solcher Operator, dann gilt wegen (I) und (A) in Definition 3.4 $Verhinderung_{OK}(O_1, Z_1, Z_2)$. (II)

Nun sind zwei Fälle zu unterscheiden:
Fall 1: In (II) gilt Zeile (B), dann wird also ein Operator, nämlich O_1, für die Lösung von Z_2 ausgeführt, so daß $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|O_1), Z_1)$ gilt. (III)

Fall 2: In (II) gilt Zeile (C), dann gibt es eine Vorbedingung V_2 von O_1, für die gilt: $Verhinderung_{VK}(V_2, Z_1, Z_2)$. (IV)

Sei M_2 die Menge aller Operatoren, die einen Effekt besitzen, der mit V_2 unifizierbar ist. Da V_2 eine Vorbedingung eines Operators ist, der für die Lösung von Z_2 ausgeführt wird, muß in PL mindestens ein Operator aus M_2 für die Lösung von Z_2 ausgeführt werden. Sei oBdA O_2 ein solcher Operator. Wegen (IV) gilt dann $Verhinderung_{OK}(O_2, Z_1, Z_2)$. (V)

Nun erfolgt wie oben eine analoge Fallunterscheidung für O_2. Solange dabei Fall 2 auftritt, wird die Definition von *Verhinderung* rekursiv angewendet und es entsteht dabei folgende Kette: $Z_2 \rightarrow O_1 \rightarrow V_2 \rightarrow O_2 \rightarrow \ldots V_i \rightarrow O_i \rightarrow \ldots$ (VI)

Da PL endlich, ist obige Kette endlich und somit gibt es ein n, so daß bei der Fallunterscheidung für O_n Fall 1 auftritt. Wegen (VI) wird dann also ein Operator, nämlich O_n, für die Lösung von Z_2 ausgeführt, so daß $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|O_n), Z_1)$ gilt. (VII)

Wegen (III) und (VII) wird also in jeder Lösung PL für P mindestens ein Operator OK ausgeführt, so daß $\neg Pseudolösbar(\overline{OG}(Z_1, Z_2|OK), Z_1)$ gilt. (VIII)

Wegen (VIII) und Lemma A.1 wird damit Z_1 in jeder Lösung von P **vor** Z_2 gelöst und somit gilt $Z_1 <_Z Z_2$. qed.

A.3 Domänendefinitionen

In diesem Abschnitt werden die in dieser Arbeit vorkommenden Domänen in der in Kapitel 2.2 beschriebenen Notation aufgelistet.

A.3.1 D^1S^2-Domäne

$DOM_{D^1S^2} = (P, O, T)$ mit:

$$\begin{array}{lcl} P & = & \{G_1, G_2, G_3, \ldots, I_1, I_2, I_3, \ldots, P_1, P_2, P_3, \ldots\} \\ O & = & \{A_{11}, A_{12}, A_{21}, A_{22}, A_{31}, A_{32}, \ldots\} \\ T & = & \{\ \} \end{array}$$

und für die Operatoren aus O gilt:

$$\begin{array}{llll} \{\ I_i\ \} & A_{i1} & \{P_i, & \text{-}I_{i-1}\ \} \quad [\,] \\ \{\ P_i\ \} & A_{i2} & \{G_i, & \text{-}I_j\ |\forall j, \quad \text{-}P_{i-1}\ \} \quad [\,] \end{array}$$

A.3.2 D^mS^2-Domäne

$DOM_{D^mS^2} = (P, O, T)$ mit:

$$\begin{array}{lcl} P & = & \{G_1, G_2, G_3, \ldots, I_1, I_2, I_3, \ldots, P_1, P_2, P_3, \ldots\} \\ O & = & \{A_{11}, A_{12}, A_{21}, A_{22}, A_{31}, A_{32}, \ldots\} \\ T & = & \{\ \} \end{array}$$

und für die Operatoren aus O gilt:

$$\begin{array}{llll} \{\ I_i\ \} & A_{i1} & \{P_i, & \text{-}I_j\ |i-m \leq j < i\ \} \quad [\,] \\ \{\ P_i\ \} & A_{i2} & \{G_i, & \text{-}I_j\ |\forall j, \quad \text{-}P_j\ |i-m \leq j < i\ \} \quad [\,] \end{array}$$

A.3.3 $\theta_2 D^1S^1$-Domäne

$DOM_{\theta_2 D^1S^1} = (P, O, T)$ mit:

$$\begin{array}{lcl} P & = & \{G\text{-}a, G_1, G_2, G_3, \ldots, I_1, I_2, I_3, \ldots, Pa, Pb\ \} \\ O & = & \{A\text{-}a, A_{1a}, A_{1b}, A_{2a}, A_{2b}, A_{3a}, A_{3b}, \ldots\} \\ T & = & \{\ \} \end{array}$$

und für die Operatoren aus O gilt:

$$\begin{array}{llll} \{\ \} & A\text{-}a & \{G\text{-}a, & \text{-}P_b, \ \text{-}G_j\ |\forall j\ \} \quad [\,] \\ \{\ I_i, P_a\ \} & A_{ia} & \{G_i, & \text{-}I_{i-1}\ \} \quad [\,] \\ \{\ I_i, P_b\ \} & A_{ib} & \{G_i, & \text{-}I_{i-1}\ \} \quad [\,] \end{array}$$

A.3.4 $\theta_3 D^1 S^1$-Domäne

$DOM_{\theta_3 D^1 S^1} = (P, O, T)$ mit:

$P = \{G\text{-}a,\ G_1,\ G_2,\ G_3,\ \ldots, I_1,\ I_2,\ I_3,\ \ldots,\ Pa,\ Pb,\ Pc\ \}$
$O = \{A\text{-}a,\ A_{1a},\ A_{1b},\ A_{1c},\ A_{2a},\ A_{2b},\ A_{2c},\ A_{3a},\ A_{3b},\ A_{3c},\ \ldots\}$
$T = \{\ \}$

und für die Operatoren aus O gilt:

$\{\ \}$	$A\text{-}a$	$\{G\text{-}a,\ -P_b, -P_c,\ -G_j\ \vert\forall j\ \}$	$[\ ]$
$\{\ I_i,\ P_a\ \}$	A_{ia}	$\{G_i,\ -I_{i-1}\ \}$	$[\ ]$
$\{\ I_i,\ P_b\ \}$	A_{ib}	$\{G_i,\ -I_{i-1}\ \}$	$[\ ]$
$\{\ I_i,\ P_c\ \}$	A_{ic}	$\{G_i,\ -I_{i-1}\ \}$	$[\ ]$

A.3.5 Workpiece- und Blocksworld-Domäne

Die komplette Domänendefinition der *Workpiece-Domäne* ist sehr umfangreich und wird deswegen hier nicht dargestellt. Eine Definition dieser Domäne befindet sich in [Muñoz-Avila und Weberskirch, 1996].

Für die *Blocksworld-Domäne* wird hier die Variante mit zwei Operatoren notiert, Varianten mit drei oder vier Operatoren finden sich in [Russell und Norvig, 1995].

$DOM_{BW} = (P, O, T)$ mit:

$P = \{\ Clear(x),\ On(x,y)\ \}$
$O = \{\ ClearTop(x,y,z),\ PutOn(x,y,z)\ \}$
$T = \{\ Block,\ Table\ \}$

und für die Operatoren aus O gilt:

$\{\ Clear(y)\ \}$ $\quad ClearTop(x,y,z) \quad$ $\{\ Clear(x),\ \text{-}Clear(z),\ On(y,z),\ \text{-}On(y,x)\ \}$
$[\ IsOfType(Block,\ x,y),\ NotSame(x,y),\ NotSame(x,z),\ NotSame(y,z)\ ]$
$\{\ Clear(x),\ Clear(y)\ \}$ $\quad PutOn(x,y,z) \quad$ $\{\ On(x,y),\ \text{-}On(x,z),\ Clear(z),\ \text{-}Clear(y)\ \}$
$[\ IsOfType(Block,\ x),\ NotSame(x,y),\ NotSame(x,z),\ NotSame(y,z)\ ]$

Literaturverzeichnis

BARRETT, A. UND WELD, D.S. 1994. Partial-Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*, **67**(1), 71–112.

BLYTHE, J.; ETZIONI, O.; GIL, Y.; JOSEPH, R.; PEREZ, A.; REILLY, S.; VELOSO, M. UND WANG, X. 1992. *PRODIGY4.0: The manual and tutorial.* Technical Report CMU-CS-92-150. School of Computer Science, Carnegie Mellon University.

BYLANDER, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, **69**, 165–204.

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

ETZIONI, O. 1991. STATIC: A problem space compiler for PRODIGY. *Seiten 533–540 in: Proceedings of AAAI-91.*

ETZIONI, O. 1993. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, **62**, 255–301.

FINK, E. UND VELOSO, M. 1995. Formalizing the PRODIGY Planning Algorithm. *In: Proceedings of the 3rd European Workshop on Planning (EWSP-95).*

HÜLLEN, JOCHEM. 1995. *Aufbau einer domänenunabhängigen Fallbasis und Fallauswahl mit Zielabhängigkeiten in CAPlan/CbC.* Projektarbeit, Universität Kaiserslautern.

MCALLESTER, D. UND ROSENBLITT, D. 1991. Systematic Nonlinear Planning. *Seiten 634–639 in: Proceedings of AAAI-91.*

MUÑOZ-AVILA, H. 1998. *Integrating Twofold Case Retrieval and Complete Decision Replay in CAPlan/CbC.* Dissertation, University of Kaiserslautern.

MUÑOZ-AVILA, H. UND WEBERSKIRCH, F. 1996. *A Specification of the Domain of Process Planning: Properties, Problems and Solutions.* Bericht LSA-96-10E. Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

RUSSELL, S. UND NORVIG, P. 1995. *Artificial Intelligence - A Modern Approach.* Prentice-Hall International.

SMITH, D. UND PEOT, M. 1993. Postponing threats in partial-order planning. *Seiten 500–506 in: Proceedings of AAAI-93.*

SMITH, D. UND PEOT, M. 1996. Suspending Recursion in Causal-link Planning. *Seiten 182–190 in: Proceedings of the 3rd International Conference on AI Planning Systems (AIPS-96).*

VELOSO, M.M. 1992. *Learning by Analogical Reasoning in General Problem Solving.* Dissertation, Carnegie Mellon University, Pittsburgh, PA.

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. 1998. *Extending Problem Specifications for Plan-Space Planners.* Bericht LSA-98-02E. Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

WEBERSKIRCH, F. UND MUÑOZ-AVILA, H. 1997. Advantages of Types in Partial-Order Planning. *In:* MÜLLER, M.; SCHUMANN, O. UND SCHUMANN, S. (Hrsg.), *Beiträge zum 11. Workshop 'Planen und Konfigurieren' (PuK-97).* FORWISS-REPORT, Nr. FR–1997–001. FORWISS.