

DIPLOMARBEIT

Codierungen paralleler Pläne im Kontext erfüllbarkeitsbasierter Handlungsplanung

verfasst von Martin Wehrle

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Lehrstuhl Grundlagen der Künstlichen Intelligenz
Prof. Dr. Bernhard Nebel

Abgabe

22. Mai 2006

Betreuer

PD Dr. Jussi Rintanen

Gutachter

Prof. Dr. Bernhard Nebel

PD Dr. Jussi Rintanen

Zusammenfassung

Die Erforschung automatisierter Handlungsplanung ist ein wichtiges Teilgebiet der Künstlichen Intelligenz.

Einer der führenden Ansätze zur algorithmischen Lösung von Planungsproblemen ist Planen durch aussagenlogische Erfüllbarkeitstests. Um eine möglichst hohe Effizienz zu erreichen ist es hierbei sinnvoll, die parallele Ausführung von Operatoren zuzulassen.

Dazu werden in dieser Arbeit aussagenlogische Codierungen vorgestellt, die zu einer neuen Klasse von parallelen Plänen für STRIPS-Operatoren führen. Die in einem Zustand s parallel ausgeführten Operatoren haben die Eigenschaft, dass mindestens eine Ordnung existiert, in der diese auch sequentiell in s ausgeführt werden können. Im Gegensatz zu früheren Arbeiten müssen aber noch nicht sämtliche Vorbedingungen bereits in s erfüllt sein, sondern dürfen auch noch von anderen, simultan ausgeführten Operatoren wahr gemacht werden.

In dieser neuen Semantik sind im Vergleich zu bisherigen Ansätzen mehr Operatoren parallel ausführbar. Es wird gezeigt, dass die darauf basierenden Codierungen in vielen interessanten Planungsproblemen zu Effizienzsteigerungen beim Planen führen.

Danksagungen

An dieser Stelle möchte ich mich bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

In besonderem Maße bedanke ich mich bei Herrn PD Dr. Jussi Rintanen für seine hervorragende Betreuung, viele gute Ideen sowie das Bereitstellen des Programmcodes seines SML-Planers und des Programmcodes der Algorithmen zur Plansuche. Herrn Prof. Dr. Bernhard Nebel danke ich für die Möglichkeit, diese Arbeit an seinem Lehrstuhl anzufertigen. Schließlich danke ich meinen Eltern, die mein Studium und damit insbesondere diese Diplomarbeit erst ermöglicht haben.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Freiburg, den 22. Mai 2006

Martin Wehrle

Inhaltsverzeichnis

1. Einleitung	7
1.1. Handlungsplanung	7
1.2. Verwandte Arbeiten	8
1.3. Überblick	10
2. Grundlagen und Definitionen	11
2.1. Notation	11
2.2. Transitionssysteme und Handlungsplanung	13
2.3. Parallele Pläne	15
2.3.1. Step-Semantik	15
2.3.2. 1-Linearisierungs-Semantik	16
2.4. Invarianten	17
2.5. Planen durch Erfüllbarkeitstests	18
2.5.1. Die Basiscodierung	19
2.5.2. Step-Semantik	20
2.5.3. 1-Linearisierungs-Semantik	21
3. Codierungen basierend auf Disabling-Enabling-Graphen	25
3.1. Der Disabling-Enabling-Graph	25
3.2. Umsetzung der Vorbedingungsaxiome	30
3.3. Umsetzungen der Parallelitätsaxiome	33
3.3.1. Allgemeine Codierung basierend auf exaktem Azyklizitätstest	33
3.3.2. Codierung basierend auf fester Reihenfolge der Operatoren	35
3.3.3. Codierung basierend auf Feedback-Vertices	37
3.4. Gesamtcodierungen	40
3.5. Eine weitere Codierung basierend auf fester Reihenfolge der Operatoren	41
3.6. Zusammenfassung	44
3.7. Erweiterungen und Verbesserungen	45
3.7.1. Heuristiken zur Anordnung der Operatoren	45
3.7.2. Erweiterung der Codierungen auf Operatoren mit komplementären Effekten	46
4. Experimente	48
4.1. Algorithmen zur Plansuche	48
4.2. Aufbau	51

4.2.1. Planungsdomänen	51
4.2.2. Implementierung der Heuristiken zur Anordnung der Operatoren	51
4.3. Ergebnisse	52
4.4. Fazit	58
5. Zusammenfassung und Ausblick	59
A. Die Planungsdomäne Boxes	61
B. Laufzeiten	62
B.1. Logistik	62
B.2. Blocks-World	62
B.3. Depots	63
B.4. Gripper	64
B.5. Freecell	64
B.6. Satellite	65
B.7. Rover	65
B.8. Boxes	65

1. Einleitung

1.1. Handlungsplanung

Der Begriff Handlungsplanung beschreibt die Aufgabe, durch situationsabhängige Entscheidungen über auszuführende Aktionen einen vorgegebenen Zielzustand zu erreichen. Probleme dieser Art ergeben sich in verschiedenen Gebieten der Künstlichen Intelligenz. Ein einfaches Beispiel ist etwa ein Roboter, der selbständig den Weg aus einem Labyrinth finden muss. Der Zielzustand kann durch die Ausführung der ihm zur Verfügung stehenden Aktionen (beispielsweise fahre-geradeaus, biege-rechts-ab, ...) erreicht werden. Auf einer abstrakten Ebene formalisiert man eine *Planungsaufgabe* typischerweise durch eine Menge von Zuständen, die durch Zustandsvariablen repräsentiert werden, einen Anfangs- und einen Zielzustand sowie eine Menge von Operatoren, die Aktionen repräsentieren und den aktuellen Zustand verändern können. Die Aufgabe ist es, durch eine geeignete Ausführung der Operatoren vom Ausgangs- in den Zielzustand zu gelangen. Hierzu ist man an allgemeinen *Algorithmen* interessiert, die unabhängig von einer speziellen Planungsdomäne sind und auf alle Probleme, die in einer allgemeinen Problembeschreibungssprache formalisierbar sind, anwendbar sein sollen. In diesem Zusammenhang spricht man von *domänen-unabhängigem Planen*.

Da die wirkliche Welt im Allgemeinen zu komplex ist, um komplett modelliert zu werden, können bei ihrer Beschreibung unterschiedliche Annahmen getroffen werden. Im einfachsten Fall wird sie *deterministisch* dargestellt. Dies bedeutet, dass nur ein Ausgangszustand existiert und die Anwendung eines Operators zu genau einem Nachfolgezustand führt. Ein *Plan*, d.h. eine Lösung des Planungsproblems, wird hier durch eine einfache Folge von Operatoren beschrieben, die vom Ausgangs- in den Zielzustand führt. Das Planexistenz-Problem ist jedoch bereits im deterministischen Fall PSPACE-vollständig [By194]. Weiterhin ist das deterministische Modell in manchen Fällen nicht adäquat, da in der realen Welt der Effekt einer Aktion im Voraus oftmals nicht exakt vorhergesagt werden kann. Wird die Umwelt *nichtdeterministisch* modelliert, so erlaubt man auch, dass die Ausführung eines Operators zu verschiedenen Nachfolgezuständen führen kann. Weiterhin kann die Möglichkeit bestehen, dass mehrere Ausgangszustände existieren und die Umwelt nur teilweise beobachtbar ist. Ein Plan besteht dann im Gegensatz zum deterministischen Fall zusätzlich aus *if-then*-Anweisungen, da es möglich sein muss, den nächsten Operator abhängig vom aktuell wahrgenommenen Zustand auszuwählen (ein Lösungsalgorithmus für diesen allgemeinen Fall, in dem die Umwelt nur teilweise beobachtbar ist, wird etwa von Bertoli et al. in [BCRT01] vorgestellt). Eine weitere Konsequenz für diesen allgemeineren Fall ist eine erheblich höhere Berechnungskomplexität zur Lösung des Planexistenz-Problems [Rin05].

In vielen Fällen genügt es, sich auf deterministisches Planen zu beschränken, da viele Ereignisse, die theoretisch eintreffen könnten, oftmals sehr unwahrscheinlich sind und deshalb ignoriert werden können. Hat beispielsweise ein Roboter die Aufgabe, den Ausweg aus einem Labyrinth

zu finden, dann muss nicht modelliert werden, dass sein Antrieb durch die theoretische, aber sehr unwahrscheinliche Möglichkeit eines Blitzschlags außer Betrieb gesetzt werden könnte. Weiterhin existieren aufgrund der geringeren Berechnungskomplexität viele für die meisten Planungsaufgaben effiziente Lösungsverfahren. Hierzu wird in der Arbeit von Blum und Furst zunächst ein *Planungsgraph* konstruiert, aus dessen Struktur hervorgeht, welche Zustände zu einem Zeitpunkt möglicherweise erreichbar und welche Operatoren ausführbar sind [BF95]. Hieraus kann anschließend durch Rückwärtssuche ein gültiger Plan mit einer maximalen Länge, die durch den Graph vorgegeben wird, extrahiert werden, falls ein solcher existiert. Eine andere Möglichkeit ist die heuristische Suche im Raum aller Zustände. Bonet und Geffner beschreiben hierzu eine Heuristik, um die Suche möglichst schnell zu einem Zielzustand zu leiten [BG01]. In einem weiteren Ansatz beschreiben Kautz und Selman, wie Planen auf aussagenlogische Erfüllbarkeitstests reduziert werden kann [KS92, KS96]. In diesem Kontext werden Planungsprobleme in aussagenlogische Formeln übersetzt, für die gilt, dass jede erfüllende Belegung einem Plan entspricht. Diesen Ansatz kombinieren sie in einer weiteren Arbeit mit dem Ansatz des Planens durch Planungsgraphen [KS99].

Darüberhinaus besteht eine strukturelle Ähnlichkeit zwischen Handlungsplanung und Modellprüfung (*Model Checking*), da sowohl in der Handlungsplanung als auch in der Modellprüfung nach einer Folge von Zuständen gesucht wird, für die bestimmte, vorgegebene Eigenschaften gelten. Dieser Zusammenhang ermöglicht es, Algorithmen aus der Handlungsplanung auch bei der Modellprüfung einzusetzen (im Kontext von erfüllbarkeitsbasierter Handlungsplanung beschreiben Biere et al. hierzu eine Möglichkeit, LTL-Modellprüfung auf aussagenlogische Erfüllbarkeit zu reduzieren [BCCZ99]).

Um die Effizienz beim Planen zu erhöhen, ist es in vielen Fällen sinnvoll, die parallele Ausführung von mehreren Operatoren pro Zeitpunkt zuzulassen. Um dies deutlich zu machen, betrachten wir n Operatoren, deren Vorbedingungen und Effekte auf unterschiedlichen Zustandsvariablen arbeiten und somit in jeder möglichen Anordnung ausgeführt werden können. Für n solcher Operatoren existieren damit $n!$ Pläne, die alle zum gleichen Zustand führen und somit in diesem Sinne äquivalent sind. Um zu zeigen, dass kein Plan der Länge n existiert, der aus diesen Operatoren besteht, müsste ein auf aussagenlogischer Erfüllbarkeit basierender Algorithmus folglich zeigen, dass keiner der $n!$ möglichen Pläne das Ziel erreicht, was mit wachsendem n im Allgemeinen sehr rechenaufwändig ist. Erlaubt man hingegen die simultane Ausführung von Operatoren, so müssen nicht mehr alle Zwischenzustände explizit repräsentiert werden, was in den meisten Fällen zu einer geringeren Formelgröße und damit effizienterem Planen führt.

In dieser Arbeit wird der Ansatz des Planens für deterministische Probleme durch aussagenlogische Erfüllbarkeitstests weiter verfolgt. Insbesondere wird ein Konzept vorgestellt, auf dessen Basis mehr Operatoren als bisher parallel ausgeführt werden können.

1.2. Verwandte Arbeiten

Planen durch aussagenlogische Erfüllbarkeitstests wurde erstmals von Kautz und Selman vorgeschlagen [KS92]. Die Idee ist, ein Planungsproblem in aussagenlogische Formeln $\Phi_1, \Phi_2, \Phi_3, \dots$ zu übersetzen, für die gilt, dass jede erfüllende Belegung von Φ_n einem gültigen Plan der Länge n entspricht. Um eine höhere Effizienz beim Auffinden von Plänen zu erreichen, ist es in vielen

Fällen sinnvoll, die parallele Ausführung von mehreren Operatoren pro Zeitpunkt zuzulassen. Kautz und Selman verlangen hierzu in [KS96], dass für parallel ausgeführte Operatoren auch deren sequentielle Ausführung in jeder Ordnung möglich sein muss und jedesmal der gleiche Zustand erreicht wird. Parallel ausgeführte Operatoren sind somit in diesem Sinne voneinander unabhängig. Dimopoulos, Nebel und Koehler zeigen, dass diese Bedingung noch gelockert werden kann, indem für simultan ausgeführte Operatoren nur noch die Existenz mindestens *einer* Ordnung gefordert wird, in der sie auch sequentiell ausgeführt werden können [DNK97]. Cayrol, Régnier und Vidal modifizieren mit dieser Idee den GraphPlan-Algorithmus [CRV01]. Im Kontext von erfüllbarkeitsbasierter Handlungsplanung wird sie erstmals von Rintanen, Heljanko und Niemelä umgesetzt [RHN04].

Im Folgenden werden die charakteristischen Eigenschaften von simultan ausgeführten Operatoren in parallelen Plänen verwandter Arbeiten mit einer allgemeinen Definition paralleler Pläne sowie mit dieser Arbeit verglichen.

In einer allgemeinen Definition fordert man für in einem Zustand s simultan ausgeführte Operatoren lediglich, dass mindestens eine Ordnung existiert, in der diese auch sequentiell ausgeführt werden können. Sie dürfen dabei inkonsistente Effekte haben und müssen nicht bereits alle in s ausführbar sein. Außerdem ist es möglich, dass Vorbedingungen von anderen Operatoren wieder falsifiziert werden. Diese Definition ist jedoch zu allgemein gehalten, um in kompakte aussagenlogische Codierungen umgesetzt werden zu können.

Aus diesem Grund werden in verwandten Arbeiten geeignete zusätzliche Bedingungen formuliert, die einerseits zwar die Anzahl der parallel ausführbaren Operatoren einschränken, andererseits aber kompakte und damit effiziente Codierungen zulassen. Die folgende Tabelle gibt hierzu einen kurzen Überblick.

	allg. Definition	[KS96]	[RHN04]	diese Arbeit
Effekte sind paarweise konsistent		×	×	×
Vorbedingungen sind bereits in s wahr		×	×	
Vorbedingungen können nicht falsifiziert werden		×		

Tabelle 1.1.: Eigenschaften simultan ausgeführter Operatoren in parallelen Plänen

In dieser Arbeit wird eine neue Klasse von parallelen Plänen vorgestellt. Es werden Codierungen in Aussagenlogik angegeben, die die parallele Ausführung von Operatoren in einem Zustand s erlauben, falls mindestens eine Ordnung existiert, in der diese auch sequentiell ausgeführt werden können. Im Unterschied zur Semantik in [RHN04] muss die Vorbedingung eines ausgeführten Operators in s aber noch nicht wahr sein, sondern darf auch durch andere, simultan ausgeführte Operatoren wahr gemacht werden. Es müssen also noch nicht alle Vorbedingungen der parallel ausgeführten Operatoren bereits in s erfüllt sein. Dies kann zu deutlich kürzeren Plänen und somit zu Effizienzsteigerungen beim Planen führen.

1.3. Überblick

In Abschnitt 2 werden grundlegende Konzepte und Definitionen, die im Rahmen dieser Arbeit benötigt werden, eingeführt. Anschließend wird in Abschnitt 3 ein Konzept für eine neue Klasse paralleler Pläne erarbeitet, das eine im Vergleich zu bisherigen Arbeiten höhere Anzahl von Operatoren parallel ausführbar macht. Dieses wird in aussagenlogische Codierungen umgesetzt, deren Effizienz in Abschnitt 4 untersucht wird. Die Schlussfolgerungen sowie ein kurzer Ausblick auf zukünftige Arbeiten werden abschließend in Abschnitt 5 formuliert.

Der Beitrag dieser Arbeit besteht in einer neuen Semantik für parallele Pläne, in der STRIPS-Operatoren auch dann parallel in einem Zustand s ausgeführt werden können, wenn noch nicht alle Vorbedingungen in s wahr sind. Sie haben hierbei konsistente Effekte und können die Vorbedingung anderer Operatoren falsifizieren. Hierzu wird zunächst das Konzept des *Disabling Graphen* aus [RHN04] auf den *Disabling-Enabling-Graph* erweitert, auf dessen Basis anschließend aussagenlogische Codierungen zur erfüllbarkeitsbasierten Handlungsplanung erarbeitet werden. Man erhält eine neue Klasse paralleler Pläne, in der mehr Operatoren als bisher simultan ausgeführt und damit kürzere Pläne erreicht werden können.

2. Grundlagen und Definitionen

In diesem Abschnitt werden die formalen Grundlagen und Definitionen beschrieben, die im Rahmen dieser Arbeit benötigt werden. Sie sind im Wesentlichen an die Arbeit von Rintanen [Rin05] angelehnt.

2.1. Notation

Für eine Menge X wird die Potenzmenge von X mit 2^X und die Mächtigkeit mit $|X|$ bezeichnet.

Definition 2.1 (Aussagenlogische Formel). Sei P eine Menge von Aussagenvariablen. Eine *aussagenlogische Formel* (im Folgenden meist kurz mit *Formel* bezeichnet) ist induktiv auf folgende Weise definiert.

- Die Symbole \top und \perp sind aussagenlogische Formeln.
- Für alle $a \in P$ ist a eine aussagenlogische Formel.
- Sind Φ und Ψ aussagenlogische Formeln, so auch $(\Phi \wedge \Psi)$.
- Sind Φ und Ψ aussagenlogische Formeln, so auch $(\Phi \vee \Psi)$.
- Ist Φ eine aussagenlogische Formel, so auch $\neg\Phi$.

Für aussagenlogische Formeln Φ und Ψ sei weiterhin $(\Phi \rightarrow \Psi)$ die abkürzende Schreibweise für $(\neg\Phi \vee \Psi)$ und $(\Phi \leftrightarrow \Psi)$ die Abkürzung für $((\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi))$.

Definition 2.2 (Belegung). Sei P eine Menge von Aussagenvariablen. Eine *Belegung von P* ist eine Funktion $v : P \rightarrow \{0, 1\}$. Für $a \in P$ schreiben wir $v \models a$ genau dann, wenn $v(a) = 1$ (ansonsten $v \not\models a$). Weiterhin ist zu einer gegebenen Belegung v auf den Aussagenvariablen der Wahrheitswert für alle Formeln induktiv auf die folgende Weise definiert. Seien Φ und Ψ Formeln. Dann gilt

- $v \models \top$ und $v \not\models \perp$.
- $v \models (\Phi \wedge \Psi)$ genau dann, wenn $v \models \Phi$ und $v \models \Psi$.
- $v \models (\Phi \vee \Psi)$ genau dann, wenn $v \models \Phi$ oder $v \models \Psi$.
- $v \models \neg\Phi$ genau dann, wenn $v \not\models \Phi$.

Eine Formel Φ ist *erfüllbar*, wenn eine Belegung v mit $v \models \Phi$ existiert. Sie heißt *unerfüllbar* genau dann, wenn sie nicht erfüllbar ist.

Definition 2.3. Sei P eine Menge von Aussagenvariablen und $a \in P$. Eine Formel der Form a oder $\neg a$ nennt man *Literal*. Für ein Literal m definieren wir das zu m komplementäre Literal \bar{m} durch $\bar{a} = \neg a$ und $\overline{\neg a} = a$ für alle $a \in P$. Die Menge aller Literale bezeichnen wir mit $\bar{P} = P \cup \{\neg a \mid a \in P\}$. Für eine Teilmenge $\Theta \subseteq \bar{P}$ und eine Belegung v von \bar{P} schreiben wir $v \models \Theta$, falls $v \models m$ für alle $m \in \Theta$. Für Literale $m_1, \dots, m_n \in \bar{P}$ sei $m_1 \wedge \dots \wedge m_n$ die abkürzende Schreibweise für $((\dots((m_1 \wedge m_2) \wedge m_3) \dots) \wedge m_n)$ (iterierte Konjunktion) und $m_1 \vee \dots \vee m_n$ die Abkürzung für $((\dots((m_1 \vee m_2) \vee m_3) \dots) \vee m_n)$ (iterierte Disjunktion). Die Konstante \perp sowie eine Disjunktion $m_1 \vee \dots \vee m_n$ für $n \geq 1$ bezeichnet man als *Klausel*.

Weiterhin benötigen wir den Begriff eines *Graphen*.

Definition 2.4 (Gerichteter Graph). Ein *gerichteter Graph* ist ein Tupel $G = \langle V, E \rangle$. Dabei ist

- V eine endliche Menge von Knoten, und
- $E \subseteq V \times V$ eine Menge von Kanten.

Zwischen zwei Knoten v und v' existiert ein *Pfad*, falls ein $k \geq 0$ und Knoten $v'_1, \dots, v'_k \in V$ mit $(v, v'_1) \in E, \dots, (v'_{k-1}, v'_k) \in E$ und $(v'_k, v') \in E$ existieren.

Eine *starke Zusammenhangskomponente* von G ist eine Teilmenge $V' \subseteq V$ mit der Eigenschaft, dass für alle $v, v' \in V'$ ein Pfad von v zu v' existiert und diese Eigenschaft für keine echte Obermenge von V' gilt. Wir bezeichnen eine starke Zusammenhangskomponente kurz mit SCC (*strongly connected component*).

Der Graph $G' = \langle V', E' \rangle$ heißt *der von V' induzierte Teilgraph von G* , falls $V' \subseteq V$ und $E' = E \cap (V' \times V')$.

Beispiel 2.1. Sei $G = \langle \{v_1, v_2, v_3, v_4\}, \{(v_1, v_2), (v_2, v_3), (v_3, v_1), (v_2, v_4)\} \rangle$ (Abbildung 2.1). Die Menge der Knoten $V' = \{v_1, v_2, v_3\}$ bildet ein SCC von G , da von jedem Knoten aus V' ein Pfad zu jedem anderen Knoten in V' existiert.

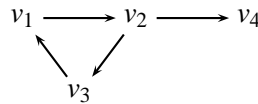


Abbildung 2.1.: Beispiel für einen gerichteten Graph

Der von der Menge $V'' = \{v_1, v_3, v_4\}$ induzierte Teilgraph von G ist $G' = \langle \{v_1, v_3, v_4\}, \{(v_3, v_1)\} \rangle$ (Abbildung 2.2).

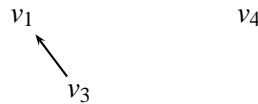


Abbildung 2.2.: Beispiel für einen induzierten Teilgraph

2.2. Transitionssysteme und Handlungsplanung

In diesem Abschnitt werden Planungsinstanzen basierend auf Transitionssystemen definiert, wie sie in dieser Arbeit verwendet werden.

Definition 2.5 (Transitionssystem). Ein *Transitionssystem* ist ein 4-Tupel $\mathcal{T} = \langle S, I, O, G \rangle$. Dabei ist

- S eine endliche Menge von Zuständen,
- $I \in S$ der Anfangszustand,
- O eine endliche Menge von Operatoren, wobei für alle $o \in O$ gilt, dass $o \subseteq S \times S$ eine partielle Funktion ist, und
- $G \subseteq S$ eine Menge von Zielzuständen.

Die partielle Funktion $o \subseteq S \times S$ ordnet den Zuständen einer Teilmenge $S' \subseteq S$ einen eindeutigen Nachfolgezustand zu.

Beispiel 2.2. Wir betrachten das Transitionssystem $\mathcal{T} = \langle S, I, O, G \rangle$ mit $S = \{s_1, s_2, s_3, s_4\}$, $I = s_1$, $O = \{o_1, o_2, o_3\}$ und $G = \{s_4\}$ (Abbildung 2.3). Sei $o_1 = \{(s_1, s_2), (s_3, s_4)\}$, $o_2 = \{(s_2, s_3)\}$ und $o_3 = \{(s_1, s_3)\}$. Wir gelangen vom Ausgangszustand in den Zielzustand, indem wir beispielsweise die Operatoren o_3 und o_1 nacheinander ausführen.

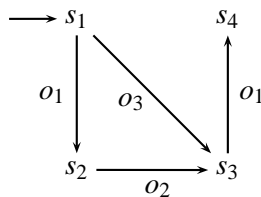


Abbildung 2.3.: Ein Transitionssystem

In dieser Arbeit identifizieren wir eine *Planungsinstanz* mit einem Transitionssystem, in dem Zustände durch Belegungen aussagenlogischer Variablen repräsentiert werden. Für eine Menge P von Aussagenvariablen ist ein Zustand s demnach eine Belegung der Variablen in P , d.h. $s : P \rightarrow \{0, 1\}$. Eine Formel Φ repräsentiert die Menge aller Zustände s mit $s \models \Phi$.

Basierend auf dieser Darstellung von Zuständen definieren wir *STRIPS-Operatoren*.

Definition 2.6 (STRIPS-Operator). Sei P eine Menge von Aussagenvariablen. Ein *STRIPS-Operator* ist ein Tupel $o = \langle p, e \rangle$, für das gilt:

- $p \subseteq \overline{P}$ ist eine Menge von Literalen (die *Vorbedingung* von o), und
- $e \subseteq \overline{P}$ ist eine Menge von Literalen (der *Effekt* von o).

Die leere Menge \emptyset als Vorbedingung identifizieren wir hierbei mit der Konstanten \top . Ein STRIPS-Operator $o = \langle p, e \rangle$ ist in einem Zustand s *anwendbar*, wenn $s \models p$ gilt. In diesem Fall bezeichnen wir den Nachfolgezustand mit $\text{app}_o(s)$. Wir erhalten $\text{app}_o(s)$, indem wir alle Literale aus e entsprechend setzen und alle anderen Literale unverändert beibehalten.

Beispiel 2.3. Sei $P = \{A, B, C\}$ eine Menge von Aussagenvariablen, $o = \langle \{A, B\}, \{\neg B, \neg C\} \rangle$ ein STRIPS-Operator und s ein Zustand mit $s \models A \wedge B \wedge C$. Dann gilt: o ist in s anwendbar, da $s \models A \wedge B$. Für den Nachfolgezustand $\text{app}_o(s)$ gilt $\text{app}_o(s) \models A \wedge \neg B \wedge \neg C$.

Ein STRIPS-Operator o *aktiviert* einen STRIPS-Operator o' , falls er ein Literal aus dessen Vorbedingung wahr macht.

Definition 2.7 (Aktivierung). Seien $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ STRIPS-Operatoren. Dann wird o' von o *aktiviert*, falls ein Literal m mit $m \in e$ und $m \in p'$ existiert.

Umgekehrt wird ein STRIPS-Operator o' von einem STRIPS-Operator o *deaktiviert*, falls o ein Literal aus der Vorbedingung von o' falsifiziert.

Definition 2.8 (Deaktivierung). Seien $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ STRIPS-Operatoren. Dann wird o' von o *deaktiviert*, falls ein Literal m mit $m \in e$ und $\bar{m} \in p'$ existiert.

Den Nachfolgezustand $\text{app}_{o_n}(\dots \text{app}_{o_1}(s) \dots)$ nach der Ausführung mehrerer Operatoren o_1, \dots, o_n in einem Zustand s bezeichnen wir kurz mit $\text{app}_{o_1; \dots; o_n}(s)$. Der Nachfolgezustand $\text{app}_{o_1; \dots; o_n}(s)$ ist nur dann definiert, wenn die Vorbedingung von $o_{i+1} = \langle p_{i+1}, e_{i+1} \rangle$ nach dem Ausführen von o_1, \dots, o_i erfüllt ist, d.h. $\text{app}_{o_1; \dots; o_i}(s) \models p_{i+1}$ für alle $i \in \{1, \dots, n-1\}$.

Basierend auf diesen Konzepten definieren wir den Begriff der Planungsinstanz.

Definition 2.9 (Planungsinstanz). Eine *Planungsinstanz* ist ein 4-Tupel $\pi = \langle P, I, O, G \rangle$. Dabei ist

- P eine endliche Menge von Aussagenvariablen,
- I der Anfangszustand,
- O eine endliche Menge von STRIPS-Operatoren, und
- G eine Formel, die die Zielzustände beschreibt.

Ein *sequentieller Plan der Länge n für π* ist eine endliche Folge von Operatoren $\sigma = o_1, \dots, o_n$ aus O mit der Eigenschaft, dass $\text{app}_{o_1; \dots; o_n}(I) \models G$.

Beispiel 2.4. Wir betrachten $\pi = \langle P, I, O, G \rangle$ mit $P = \{A, B\}$, $I \models A \wedge B$, $O = \{o_1, o_2, o_3\}$ und $G = \neg A \wedge \neg B$. Sei $o_1 = \langle \{A\}, \{\neg A\} \rangle$, $o_2 = \langle \{\neg A, B\}, \{A, \neg B\} \rangle$ und $o_3 = \langle \{A, B\}, \{\neg B\} \rangle$. Diese Planungsinstanz wird durch das Transitionssystem aus Beispiel 2.2 repräsentiert (wobei $s_1 \models A \wedge B$, $s_2 \models \neg A \wedge B$, $s_3 \models A \wedge \neg B$ und $s_4 \models \neg A \wedge \neg B$ gilt). Ein sequentieller Plan der Länge 2 ist die Folge o_3, o_1 , da $\text{app}_{o_3; o_1}(I) \models G$.

Im Hinblick auf die 1-Linearisierungs-Semantik (siehe Abschnitt 2.3.2) definieren wir abschließend, wann eine Menge von Operatoren linearisierbar heißt.

Definition 2.10 (Linearisierbarkeit). Sei $O = \{o_1, \dots, o_n\}$ eine Menge von Operatoren, s ein Zustand. O heißt *linearisierbar in s* , falls eine totale Ordnung $o_1 < \dots < o_n$ auf O existiert, sodass $\text{app}_{o_1; \dots; o_n}(s)$ definiert ist. Diese Ordnung wird als *Linearisierung* von O in s bezeichnet.

2.3. Parallele Pläne

Das im letzten Abschnitt eingeführte Konzept eines sequentiellen Plans erlaubt die Ausführung eines Operators pro Zeitpunkt. In vielen Fällen macht es jedoch Sinn, auch die gleichzeitige Anwendung mehrerer Operatoren in einem Zeitschritt zuzulassen. Im Folgenden werden hierzu analog zu [RHN05] zwei Konzepte eingeführt. In Abschnitt 2.3.1 definieren wir die *Step-Semantik*, die die parallele Ausführung von Operatoren erlaubt, wenn deren sequentielle Ausführung in jeder möglichen Reihenfolge definiert ist und zum gleichen Nachfolgezustand führt. Diese Forderung wird mit der *l-Linearisierungs-Semantik*, die wir in Abschnitt 2.3.2 definieren, abgeschwächt, indem nur noch die Existenz mindestens einer solchen Ordnung gefordert wird, für die dies gilt.

2.3.1. Step-Semantik

Mit der Step-Semantik wird für parallel ausgeführte Operatoren gefordert, dass auch deren sequentielle Ausführung in jeder möglichen Ordnung definiert ist und zum gleichen Nachfolgezustand führt.

Definition 2.11 (Step-Plan). Sei O eine Menge von Operatoren und I ein Zustand. Ein *Step-Plan der Länge l* für O und I ist eine Sequenz $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$ von Mengen von Operatoren, sodass eine Folge s_0, \dots, s_l von Zuständen (der *Ausführung* von T) existiert, für die gilt, dass

1. $s_0 = I$, und
2. für alle $i \in \{0, \dots, l-1\}$ und jede totale Ordnung $o_1 < \dots < o_n$ auf S_i der Zustand $\text{app}_{o_1; \dots; o_n}(s_i)$ definiert und gleich s_{i+1} ist.

Step-Pläne sind oftmals deutlich kürzer als sequentielle Pläne.

Beispiel 2.5. Sei $\pi = \langle P, I, O, G \rangle$ die folgende Planungsinstanz: $P = \{A, B, C\}$, $I \models A \wedge B \wedge C$, $O = \{o_1, o_2, o_3\}$ und $G = \neg A \wedge \neg B \wedge \neg C$. Sei $o_1 = \langle \top, \{\neg A\} \rangle$, $o_2 = \langle \top, \{\neg B\} \rangle$ und $o_3 = \langle \top, \{\neg C\} \rangle$. Darf pro Zeitschritt höchstens ein Operator ausgeführt werden, so erreicht man beispielsweise mit dem sequentiellen Plan $\langle \{o_1\}, \{o_2\}, \{o_3\} \rangle$ den Zielzustand in drei Schritten, wohingegen der Step-Plan $\langle \{o_1, o_2, o_3\} \rangle$ das Ziel in nur einem Schritt erreicht.

Im Folgenden bezeichnen wir für eine Menge S von STRIPS-Operatoren und einen Zustand s mit $\text{app}_S(s)$ den Zustand nach dem parallelen Ausführen aller Operatoren in S . Hierzu fordern wir zum einen, dass sämtliche Vorbedingungen in s erfüllt sind. Zum anderen müssen die Operatoren aus S konsistente Effekte haben, d.h. es dürfen keine komplementären Literale m und \bar{m} in $\bigcup_{o=\langle p, e \rangle \in S} e$ vorkommen.

Rintanen et al. zeigen in [RHN05], dass die folgenden Bedingungen notwendig und hinreichend für die Korrektheit eines Step-Plans sind.

Satz 2.12. Sei O eine Menge von STRIPS-Operatoren¹, I ein Zustand und $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$. Dann ist T ein Step-Plan für O und I genau dann, wenn es eine Folge von Zuständen s_0, \dots, s_l gibt, sodass

1. $s_0 = I$,
2. $s_{i+1} = \text{app}_{S_i}(s_i)$ für alle $i \in \{0, \dots, l-1\}$, und
3. kein $i \in \{0, \dots, l-1\}$ und keine Operatoren $\{o, o'\} \subseteq S_i$ mit $o \neq o'$ existieren, für die gilt, dass o' von o deaktiviert wird.

Beweis. siehe [RHN05]. □

Diese Bedingungen können in einfacher Weise in eine kompakte aussagenlogische Codierung umgesetzt werden (siehe Abschnitt 2.5).

2.3.2. 1-Linearisierungs-Semantik

Im Unterschied zur Step-Semantik fordert man mit der 1-Linearisierungs-Semantik lediglich die Existenz mindestens einer Ordnung, in der simultan ausgeführte Operatoren anwendbar sein müssen.

Definition 2.13 (1-Linearisierungs-Plan). Sei O eine Menge von Operatoren und I ein Zustand. Ein *1-Linearisierungs-Plan der Länge l* für O und I ist eine Sequenz $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$ von Mengen von Operatoren zusammen mit einer Folge s_0, \dots, s_l von Zuständen (der *Ausführung* von T), sodass

1. $s_0 = I$, und
2. für alle $i \in \{0, \dots, l-1\}$ eine totale Ordnung $o_1 < \dots < o_n$ auf S_i existiert, sodass der Zustand $\text{app}_{o_1; \dots; o_n}(s_i)$ definiert und gleich s_{i+1} ist.

Im Unterschied zur Step-Semantik muss in jedem Zustand s_i also nur eine Ordnung $<_i$ existieren, sodass die Operatoren aus S_i in dieser Ordnung ausgeführt werden können. Der Nachfolgezustand s_{i+1} ist hierbei allerdings nicht mehr allein durch die Operatoren in S_i eindeutig bestimmt, sondern hängt auch von $<_i$ ab. Daher werden die entsprechenden Zwischenzustände s_1, \dots, s_l in der Definition explizit vorgegeben.

Dieses Konzept führt in vielen Fällen zu deutlich kürzeren Plänen im Vergleich zur Step-Semantik.

Beispiel 2.6. Sei $\pi = \langle P, I, O, G \rangle$ die folgende Planungsinstanz: $P = \{A, B, C, D\}$, $I \models A \wedge B \wedge C \wedge \neg D$, $O = \{o_1, o_2, o_3\}$ und $G = \neg A \wedge \neg B \wedge C \wedge D$. Sei $o_1 = \langle \{A\}, \{D\} \rangle$, $o_2 = \langle \{B\}, \{\neg A, C\} \rangle$ und $o_3 = \langle \{C\}, \{\neg A, \neg B\} \rangle$. Der kürzeste Step-Plan ist $\langle \{o_1\}, \{o_2\}, \{o_3\} \rangle$, da zu keinem Zeitschritt mindestens zwei Operatoren existieren, die in allen möglichen Ordnungen auch sequentiell ausführbar sind. Der 1-Linearisierungs-Plan $\langle \{o_1, o_2, o_3\} \rangle$ erreicht den Zielzustand hingegen in nur einem Schritt, da $\text{app}_{o_1; o_2; o_3}(I)$ definiert ist.

¹Rintanen et al. zeigen diese Behauptung für allgemeine Operatoren.

Rintanen et al. zeigen in [RHN05], dass folgende Bedingungen hinreichend für die Korrektheit eines 1-Linearisierungs-Plans sind.

Satz 2.14. Sei O eine Menge von STRIPS-Operatoren², I ein Zustand, $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$ und s_0, \dots, s_l eine Folge von Zuständen, sodass

1. $s_0 = I$,
2. für alle $i \in \{0, \dots, l-1\}$ eine totale Ordnung $<$ auf S_i existiert, sodass für alle $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ mit $o < o'$ kein Literal m mit $m \in e$ und $\bar{m} \in p'$ existiert, und
3. $s_{i+1} = \text{app}_{S_i}(s_i)$ für alle $i \in \{0, \dots, l-1\}$.

Dann ist T ein 1-Linearisierungs-Plan für O und I .

Beweis. siehe [RHN05]. □

Diese Bedingungen können wiederum in Aussagenlogik umgesetzt werden (siehe Abschnitt 2.5). Sie sind hinreichend, jedoch nicht notwendig, da in 3. insbesondere gefordert wird, dass alle simultan ausgeführten Operatoren bereits in s_i anwendbar sein müssen.

2.4. Invarianten

Im Allgemeinen können in einer gegebenen Planungsinstanz $\pi = \langle P, I, O, G \rangle$ mit den Operatoren aus O nicht alle möglichen Zustände erreicht werden. Es entspricht also nicht jede Belegung der Zustandsvariablen aus P einem von I aus erreichbaren Zustand in der Welt.

Beispiel 2.7. Wir betrachten das Planungsproblem $\pi = \langle P, I, O, G \rangle$ in der Blocks-World-Domäne mit

- $P = \{\text{ontable}(A), \text{ontable}(B), \text{clear}(A), \text{clear}(B), \text{on}(A,B), \text{on}(B,A)\}$,
- $I \models \text{ontable}(A) \wedge \text{ontable}(B) \wedge \text{clear}(A) \wedge \text{clear}(B) \wedge \neg \text{on}(A,B) \wedge \neg \text{on}(B,A)$, und
- $O = \{o_1, o_2\}$ mit $o_1 = \{\{\text{clear}(A), \text{clear}(B)\}, \{\neg \text{clear}(B), \text{on}(A,B)\}\}$ und $o_2 = \{\{\text{clear}(A), \text{clear}(B)\}, \{\neg \text{clear}(A), \text{on}(B,A)\}\}$.

Zustände, die vom Ausgangszustand mit den beiden Operatoren o_1 und o_2 nie erreicht werden können, sind beispielsweise alle Zustände s mit der Eigenschaft $s \models \text{on}(A,B) \wedge \text{on}(B,A)$ oder s' mit der Eigenschaft $s' \models \text{clear}(A) \wedge \text{on}(B,A)$.

Um bei der Suche nach Plänen unnötige Sucharbeit zu vermeiden, ist es daher sinnvoll, solche Zustände möglichst schnell zu erkennen. Hierzu dienen *Invarianten*, d.h. Formeln, die in allen vom Ausgangszustand erreichbaren Zuständen gelten. In der folgenden Definition beschränken wir uns auf die effizient berechenbaren 2-Literal-Invarianten.

²Rintanen et al. zeigen diese Behauptung für allgemeine Operatoren.

Definition 2.15 (Invariante). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Die Menge der *erreichbaren* Zustände in π ist die kleinste Menge, die I enthält und unter $\text{app}_o(s)$, $o \in O$, abgeschlossen ist.

Eine *Invariante* in π ist eine Formel $\Phi = m_1 \vee m_2$ mit $m_1, m_2 \in \overline{P}$, für die gilt, dass $s \models \Phi$ für alle in π erreichbaren Zustände s .

Gilt also in einem Zustand s eine Invariante nicht, so ist dieser Zustand nicht vom Ausgangszustand I mit Operatoren aus O erreichbar. Wir erhalten somit eine hinreichende Bedingung für die Nichterreichbarkeit von Zuständen.³

Beispiel 2.8. Wir betrachten noch einmal Beispiel 2.7. In jedem Zustand, der von I aus erreichbar ist, gelten die Invarianten $\Phi_1 = (\neg \text{on}(A, B) \vee \neg \text{on}(B, A))$ und $\Phi_2 = (\neg \text{clear}(A) \vee \neg \text{on}(B, A))$.

Zur Berechnung werden von Rintanen in [Rin04b, Rin05] Algorithmen angegeben. Beschränkt man sich auf Invarianten gemäß Definition 2.15, so ist die Berechnung in polynomieller Zeit möglich.

Das Hinzufügen dieser Invarianten beim Planen durch Erfüllbarkeitstests (siehe folgenden Abschnitt 2.5) als weiteres Konjunktionsglied beschleunigt die Suche nach einer erfüllenden Belegung, da viele Zustände, die nicht erreichbar sind, früher erkannt werden.

2.5. Planen durch Erfüllbarkeitstests

Planen durch Erfüllbarkeitstests geht zurück auf einen Vorschlag von Kautz und Selman [KS92, KS96]. Hierzu übersetzt man eine gegebene Planungsinstanz $\pi = \langle P, I, O, G \rangle$ in aussagenlogische Formeln $\Phi_0, \Phi_1, \Phi_2, \dots$ mit der Eigenschaft, dass jede Belegung v mit $v \models \Phi_n$ einem Plan der Länge n für π entspricht.

Bemerkung 2.16. *Obwohl vermutet wird, dass $NP \neq PSPACE$ gilt, ist eine Reduktion des $PSPACE$ -vollständigen Planungs-Problems auf das NP -vollständige Erfüllbarkeitsproblem möglich, da ein kürzester Plan im schlechtesten Fall exponentielle Länge in der Größe der Planungsinstanz haben kann. Die Laufzeit, um das Planungsproblem in aussagenlogische Formeln zu übersetzen, ist dann folglich ebenfalls exponentiell, da eine Formel entsprechender Größe, die diesen kürzesten Plan repräsentiert, erzeugt werden muss.*

In diesem Kontext werden im Folgenden bereits existierende aussagenlogische Codierungen vorgestellt, die dieses Problem lösen. In Abschnitt 2.5.1 leiten wir die Basiscodierung her, die allen weiteren Codierungen zugrunde liegt. In Abschnitt 2.5.2 wird eine einfache Codierung zur Erzeugung von Step-Plänen beschrieben [KS96]. In Abschnitt 2.5.3 stellen wir danach eine kompakte Codierung von Rintanen et al. vor, mit der 1-Linearisierungs-Pläne gefunden werden [RHN05].

³Die exakte Berechnung der stärksten Invariante Ψ , die alle erreichbaren Zustände exakt charakterisiert und für die $\Psi \models \Phi$ für alle Invarianten Φ gilt, ist $PSPACE$ -schwer [Rin04b].

2.5.1. Die Basiscodierung

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Eine Formel Φ_n , die einen Plan der Länge $\leq n$ beschreibt, hat die allgemeine Struktur

$$\Phi_n = I^0 \wedge R(P^0, P^1) \wedge \dots \wedge R(P^{n-1}, P^n) \wedge G^n. \quad (2.1)$$

Die Konjunktionsglieder $R(P^t, P^{t+1})$ für alle $t \in \{0, \dots, n-1\}$ bezeichnen die Transitionsrelation, die vom Zustand zum Zeitpunkt t zum Nachfolgezustand zum Zeitpunkt $t+1$ führt. Hierbei bezeichnet $P^t = \{a_1^t, \dots, a_k^t\}$ für alle $t \in \{0, \dots, n\}$ die Menge aller Zustandsvariablen aus P , wobei jede Variable mit t ausgezeichnet ist. Die anschauliche Bedeutung von a_j^t ist, dass a_j zum Zeitpunkt t gilt.

Weiterhin bezeichnet I^0 die Konjunktion aller Literale aus \bar{P} , die im Anfangszustand I gelten und G^n die Konjunktion aller Literale, die im Zielzustand G gelten müssen. Eine erfüllende Belegung von Φ_n korrespondiert somit zu einem Plan der Länge n .

Im Folgenden werden aussagenlogische Umsetzungen der Transitionsrelation $R(P, P')$ vorgestellt, die einen vom Ausgangszustand I erreichbaren Zustand s in seinen Nachfolgezustand s' überführt (analog zu oben bezeichnet hierbei $P' = \{a_1', \dots, a_k'\}$ die entsprechend ausgezeichneten Variablen aus P). Sie bestehen stets aus der *Basiscodierung* (Definition 2.17) sowie aus von der Semantik abhängenden Konjunktionsgliedern (siehe nachfolgende Abschnitte 2.5.2 und 2.5.3).

Hierzu führen wir zunächst für alle $o \in O$ eine Aussagenvariable o ein, die genau dann wahr sein soll, wenn der entsprechende Operator angewendet wird. Die Menge dieser Hilfsvariablen für O bezeichnen wir mit $Var(O)$.

Die Basiscodierung, die allen Codierungen zu Grunde liegt, besteht nun aus den folgenden Konjunktionsgliedern.

Für alle $o \in O$, $o = \langle p, e \rangle$ sei

$$o \rightarrow e' \quad (2.2)$$

die Formel, die sicherstellt, dass bei Anwendung von Operator o dessen Effekte im nachfolgenden Zustand gelten (hierbei wird $e' \subset \bar{P}'$ mit der Konjunktion aller Literale aus e' identifiziert). Wir bezeichnen die Konjunktion der Formeln aus (2.2) mit *Effectaxioms* $^\pi$.

Weiterhin muss gewährleistet sein, dass sich der Wahrheitswert einer Variablen von einem Zustand s zum nachfolgenden Zustand s' nur ändern kann, wenn ein Operator mit einem entsprechenden Effekt ausgeführt wurde (*Frame Problem*). Wir definieren hierzu für alle Zustandsvariablen $a \in P$

$$(a \wedge \neg a') \rightarrow \bigvee \{o \mid o \in O, o = \langle p, e \rangle, \neg a \in e\} \quad (2.3)$$

und

$$(\neg a \wedge a') \rightarrow \bigvee \{o \mid o \in O, o = \langle p, e \rangle, a \in e\}, \quad (2.4)$$

wobei $\bigvee \emptyset$ mit \perp identifiziert wird. Wir bezeichnen die Konjunktion dieser Formeln aus (2.3) und (2.4) für alle $a \in P$ mit *Frameaxioms* $^\pi$.

Die Invarianten zu π (siehe Abschnitt 2.4) mit Zustandsvariablen aus P' werden mit *Invariantaxioms* $^\pi$ bezeichnet.

Zusammenfassend erhält man

Definition 2.17 (Basiscodierung). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Seien $Effectaxioms^\pi$, $Frameaxioms^\pi$ und $Invariantaxioms^\pi$ wie oben definiert. Dann ist

$$BaseEncoding^\pi = Effectaxioms^\pi \wedge Frameaxioms^\pi \wedge Invariantaxioms^\pi$$

die *Basiscodierung* zu π .

2.5.2. Step-Semantik

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Nach Satz 2.12 müssen alle in einem Zustand s simultan ausgeführten Operatoren auch bereits in s anwendbar sein. Dies wird durch die Formel

$$o \rightarrow p$$

für alle Operatoren $o \in O$ gewährleistet (wobei wir p mit der Konjunktion der entsprechenden Literale identifizieren). Es sei $Preconditionaxioms^\pi$ die Konjunktion dieser Formeln für alle $o \in O$.

Weiterhin muss nach Satz 2.12 noch garantiert werden, dass nie Operatoren $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ parallel ausgeführt werden, für die ein Literal m mit $m \in e$ und $\bar{m} \in p'$ existiert. Eine einfache Möglichkeit, dies sicherzustellen, ist, durch die Formel

$$\neg(o \wedge o')$$

explizit zu fordern, dass alle Operatoren o und o' , für die dies zutrifft, nicht simultan angewendet werden. Es sei $Parallelismaxioms^{step,\pi}$ die Konjunktion dieser Formeln für alle Paare o und o' mit dieser Eigenschaft. Man bemerkt, dass die Größe von $Parallelismaxioms^{step,\pi}$ quadratisch in der Anzahl der Operatoren wächst. Insgesamt erhält man

Definition 2.18 (Step-Codierung). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Dann ist

$$R^{step}(P, P') = BaseEncoding^\pi \wedge Preconditionaxioms^\pi \wedge Parallelismaxioms^{step,\pi}$$

die Gesamtcodierung zur Step-Semantik.

Die Codierung $R^{step}(P, P')$ stellt also die Transitionsrelation dar, um einen Zustand s , der durch Zustandsvariablen aus P repräsentiert wird, in einen Zustand s' (entsprechend repräsentiert durch Zustandsvariablen aus P') zu überführen. Nach Satz 2.12 entspricht dann eine erfüllende Belegung von

$$\Phi_n^{step} = I^0 \wedge R^{step}(P^0, P^1) \wedge \dots \wedge R^{step}(P^{n-1}, P^n) \wedge G^n$$

einem Step-Plan der Länge n , wobei $R^{step}(P^t, P^{t+1})$ für alle $t \in \{0, \dots, n-1\}$ die Transitionsrelation beschreibt, um vom Zustand s^t zum Zeitpunkt t (repräsentiert durch Zustandsvariablen aus P^t) zum entsprechenden Nachfolgezustand s^{t+1} zu gelangen.

2.5.3. 1-Linearisierungs-Semantik

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. In diesem Abschnitt wird eine Codierung zur 1-Linearisierungs-Semantik aus [RHN05] basierend auf Satz 2.14 vorgestellt.

Zunächst müssen auch hier alle Vorbedingungen der simultan ausgeführten Operatoren in einem Zustand s wahr sein. Wir erhalten

$$o \rightarrow p$$

für alle Operatoren $o \in O$. Die Konjunktion dieser Formeln für alle Operatoren bezeichnen wir wiederum mit *Preconditionaxioms* ^{π} . Weiterhin muss nach Satz 2.14 für eine Menge $O' \subseteq O$ von simultan ausgeführten Operatoren eine totale Ordnung $<$ auf O' existieren, sodass für alle $o, o' \in O'$, $o = \langle p, e \rangle$, $o' = \langle p', e' \rangle$ mit $o < o'$ gilt, dass kein Literal $m \in \bar{P}$ mit $m \in e$ und $\bar{m} \in p'$ existiert.

Hierzu müssen zunächst alle Teilmengen $\{o_1, \dots, o_n\} = O' \subseteq O$ identifiziert werden, die nicht notwendigerweise linearisierbar sind, d.h. für die es einen erreichbaren Zustand s gibt, in dem zwar $\text{app}_{O'}(s)$ definiert ist, aber keine Ordnung $o_1 < \dots < o_n$ auf O' existiert, sodass auch $\text{app}_{o_1, \dots, o_n}(s)$ definiert ist. Hierzu dient der im Folgenden vorgestellte *Disabling Graph*.

Definition 2.19 (Disabling Graph). ⁴ Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Der *Disabling Graph* zu π ist der gerichtete Graph $G = \langle O, E \rangle$, sodass für alle Operatoren $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ mit $o \neq o'$ gilt: $(o', o) \in E$ genau dann, wenn

1. o' von o deaktiviert wird,
2. kein Literal m mit $\{m, \bar{m}\} \subseteq (p \cup p')$ oder $\{m, \bar{m}\} \subseteq (e \cup e')$ existiert, und
3. keine Invariante $(m_1 \vee m_2)$ mit entweder $\bar{m}_1 \in p$ und $\bar{m}_2 \in p'$ oder $\bar{m}_1 \in e$ und $\bar{m}_2 \in e'$ existiert.

Der wesentliche Teil der Definition steckt hierbei im ersten Punkt. Er besagt, dass eine Kante von o' zu o existieren muss, falls o' von o deaktiviert wird. Die Punkte 2. und 3. dienen dazu, Kanten einzusparen, da die entsprechenden Operatoren aufgrund der anderen Konjunktionsglieder *Preconditionaxioms* ^{π} , *Effectaxioms* ^{π} und *Invariantaxioms* ^{π} ohnehin nicht simultan ausgeführt werden können und somit im Hinblick auf die Linearisierbarkeit von simultan ausgeführten Operatoren unkritisch sind.

Beispiel 2.9. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $I \models A \wedge \neg B \wedge \neg C$, $O = \{o_1, o_2, o_3\}$ mit $o_1 = \langle \{A\}, \{B\} \rangle$, $o_2 = \langle \{\neg B\}, \{C\} \rangle$ und $o_3 = \langle \{\neg C\}, \{\neg A\} \rangle$. Dann existiert ein Zykel im Disabling Graph (Abbildung 2.4): Es existiert eine Menge $O' = \{o_1, o_2, o_3\} \subseteq O$ und ein Zustand s (mit $s \models A \wedge \neg B \wedge \neg C$), für die $\text{app}_{O'}(s)$ definiert ist und es keine Ordnung gibt, in der o_1, o_2 und o_3 sequentiell ausführbar sind. Es sind also zusätzliche Bedingungen in der Codierung nötig, die verhindern, dass diese Operatoren simultan ausgeführt werden.

⁴Rintanen et al. geben in [RHN04] eine allgemeinere Definition an. Im Rahmen dieser Arbeit genügt eine Vereinfachung, da wir nur STRIPS-Operatoren betrachten.

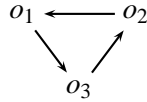


Abbildung 2.4.: Beispiel für einen zyklischen Disabling Graph

Beispiel 2.10. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $I \models A \wedge \neg B \wedge \neg C$, $O = \{o_1, o_2, o_3\}$ mit $o_1 = \langle \{A\}, \{B\} \rangle$, $o_2 = \langle \{\neg A, \neg B\}, \{C\} \rangle$ und $o_3 = \langle \{\neg C\}, \{\neg A\} \rangle$. Dann existiert kein Zykel im Disabling Graph (Abbildung 2.5). Zwischen o_1 und o_2 ist im Gegensatz zu Beispiel 2.9 keine Kante, da sowohl A als auch $\neg A$ in den Vorbedingungen enthalten ist. Da die Vorbedingungen aller zu einem Zeitpunkt t simultan ausgeführten Operatoren auch bereits in t erfüllt sein müssen, können o_1 und o_2 ohnehin nie gemeinsam ausgeführt werden und sind daher im Hinblick auf die Linearisierbarkeit unkritisch. Es werden also keine zusätzlichen Bedingungen benötigt.

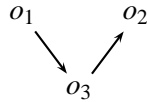


Abbildung 2.5.: Beispiel für einen azyklischen Disabling Graph

Sämtliche Mengen O' von Operatoren, die zwar im Hinblick auf die restlichen Konjunktionsglieder parallel ausführbar, aber in einem erreichbaren Zustand nicht linearisierbar sind, sind in einem SCC des Disabling Graphen enthalten. Um die Linearisierbarkeit stets zu gewährleisten muss also die Azyklizität des von O' induzierten Teilgraphen garantiert werden. Hierzu werden im Folgenden für alle SCCs entsprechende Bedingungen formuliert.

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling Graph zu π . Um die Azyklizität von G zu gewährleisten, weisen wir allen Operatoren der SCCs des Disabling Graphen zu π im Voraus eine feste Ordnung $<$ zu und erlauben deren Ausführung nur bezüglich $<$. Dies schränkt zwar die Anzahl der simultan ausführbaren Operatoren ein, lässt aber eine Codierung linearer Größe in der Problem Instanz zu.

Sei $G_i = \langle V_i, E_i \rangle$ ein SCC in G mit $V_i = \{o_1, \dots, o_n\}$ und der Ordnung $o_1 < \dots < o_n$. Wir fordern, dass Operatoren $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ nicht gemeinsam ausgeführt werden dürfen, falls $o < o'$ und $m \in e$ und $\bar{m} \in p'$ für ein Literal m gilt. Hierzu verwenden wir die *chain*-Codierung, die von Rintanen et al. eingeführt wurde [RHN05].

Definition 2.20 (Chain-Codierung). Sei $O = \{o_1, \dots, o_n\}$ eine Menge von STRIPS-Operatoren und $X, Y \subseteq O$. Sei $o_1 < \dots < o_n$ eine totale Ordnung auf O . Dann ist

$$\begin{aligned} \text{chain}(o_1, \dots, o_n; X; Y) = & \bigwedge \{o_i \rightarrow a^j \mid i < j, o_i \in X, o_j \in Y, \{o_{i+1}, \dots, o_{j-1}\} \cap Y = \emptyset\} \\ & \cup \{a^i \rightarrow a^j \mid i < j, \{o_i, o_j\} \subseteq Y, \{o_{i+1}, \dots, o_{j-1}\} \cap Y = \emptyset\} \\ & \cup \{a^j \rightarrow \neg o_j \mid o_j \in Y\}. \end{aligned}$$

Die Hilfsvariable a^j für einen Operator $o_j \in Y$ ist hierbei wahr, falls bereits ein Operator $o_i \in X$ mit $o_i < o_j$ ausgeführt wird. Mit dieser *chain*-Codierung ist also gewährleistet, dass bei Anwendung eines Operators $o \in X$ kein Operator $o' \in Y$ mit $o < o'$ angewendet wird.

Seien $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs von G mit $|V_i| \geq 2$ für $i = 1, \dots, k$. In unserem Fall muss garantiert sein, dass für Operatoren o und o' aus einem SCC von G mit $o < o'$ und o deaktiviert o' gilt, dass sie nicht simultan angewendet werden. Hierzu definieren wir für alle $G_i = \langle V_i, E_i \rangle$ mit $V_i = \{o_1^i, \dots, o_{n_i}^i\}$ und alle Literale $m \in \bar{P}$ die folgenden Operatormengen. Es seien

- $E_m^i = \{o \mid o \in V_i, o = \langle p, e \rangle, \bar{m} \in e\}$ die Menge aller Operatoren aus V_i , die m falsifizieren, und
- $R_m^i = \{o \mid o \in V_i, o = \langle p, e \rangle, m \in p\}$ die Menge aller Operatoren aus V_i , für die m in der Vorbedingung enthalten ist.

Die Formel $chain(o_1^i, \dots, o_{n_i}^i; E_m^i, R_m^i)$ garantiert dann die Linearisierbarkeit von simultan ausgeführten Operatoren aus V_i .

Die Gesamtcodierung ergibt sich als Konjunktion dieser Formeln.

Definition 2.21 (1-Linearisierungs-Codierung). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling Graph zu π mit den SCCs $G_1 = \langle V_1, E_1 \rangle, \dots, G_n = \langle V_n, E_n \rangle$ mit $V_i = \{o_1^i, \dots, o_{n_i}^i\}$ und der Ordnung $o_1^i <_i \dots <_i o_{n_i}^i$ für alle $1 \leq i \leq n$. Seien ferner die Mengen E_m^i und R_m^i für alle $1 \leq i \leq n$ und alle Literale $m \in \bar{P}$ wie oben definiert. Dann ist

$$R^{lin}(P, P') = BaseEncoding^\pi \wedge Preconditionaxioms^\pi \wedge \bigwedge_{i=1}^n chain(o_1^i, \dots, o_{n_i}^i; E_m^i, R_m^i).$$

Die Hilfsvariablen in $chain$ sind hierbei für jedes Literal m eindeutig zu wählen, d.h. für jeden Operator o_j und Literal m wird eine Hilfsvariable $a^{j,m}$ eingeführt.

Die Codierung $R^{lin}(P, P')$ stellt eine Transitionsrelation dar, um einen Zustand s , der durch Zustandsvariablen aus P repräsentiert wird, in den Nachfolgeszustand s' (entsprechend repräsentiert durch Zustandsvariablen aus P') zu überführen. Nach Satz 2.14 entspricht dann eine erfüllende Belegung von

$$\Phi_n^{lin} = I^0 \wedge R^{lin}(P^0, P^1) \wedge \dots \wedge R^{lin}(P^{n-1}, P^n) \wedge G^n$$

einem 1-Linearisierungs-Plan der Länge n , wobei $R^{lin}(P^t, P^{t+1})$ für alle $t \in \{0, \dots, n-1\}$ die Transitionsrelation beschreibt, um vom Zustand s^t zum Zeitpunkt t (repräsentiert durch Zustandsvariablen aus P^t) zum entsprechenden Nachfolgeszustand s^{t+1} zu gelangen.

Zusammenfassend stellt man fest, dass bisherige Umsetzungen der 1-Linearisierungs-Semantik im Kontext von erfüllbarkeitsbasierter Handlungsplanung auf Satz 2.14 basieren, wobei insbesondere gefordert wird, dass für Operatoren $O = \{o_1, \dots, o_n\}$, die in einem Zustand s simultan ausgeführt werden, $app_O(s)$ definiert ist. Dies bedeutet, dass stets die Vorbedingung aller simultan ausgeführten Operatoren bereits in s erfüllt sein müssen.

In dieser Arbeit wird auf diese Forderung verzichtet: Die Vorbedingung eines Operators o muss in einem Zustand s noch nicht erfüllt sein, wenn garantiert ist, dass o von anderen, simultan ausgeführten Operatoren entsprechend aktiviert wird. Da kompakte Codierungen für allgemeine Operatoren wohl nicht existieren, beschränken wir uns hierbei auf STRIPS-Operatoren.

Wir geben aussagenlogische Codierungen an, sodass Operatoren o_1, \dots, o_n zu einem Zeitpunkt t in der Ordnung $o_1 < \dots < o_n$ ausgeführt werden dürfen, wenn die Vorbedingung von o_i nach dem Ausführen von o_1, \dots, o_{i-1} erfüllt ist. Dies bedeutet, dass o_i auch erst von anderen Operatoren aktiviert werden darf. In vielen Fällen sind so mehr Operatoren als bisher gemeinsam ausführbar.

Beispiel 2.11. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz mit $P = \{A_i | i = 1, \dots, n\}$, $I \models \bigwedge_{i=1}^n \neg A_i$, $G = \bigwedge_{i=1}^n A_i$ und $O = \{o_i | i = 1, \dots, n\}$ mit $o_1 = \langle \top, \{A_1\} \rangle$ und $o_i = \langle \{A_{i-1}\}, \{A_i\} \rangle$, $i = 2, \dots, n$. In der bisherigen Semantik hat der kürzeste 1-Linearisierungs-Plan die Länge n , da zu jedem Zeitpunkt nur ein Operator ausgeführt werden kann. Dürfen hingegen Operatoren noch von anderen, simultan ausgeführten Operatoren aktiviert werden, so erreicht man das Ziel mit dem Plan $\langle \{o_1, \dots, o_n\} \rangle$ in nur einem Schritt.

Im folgenden Abschnitt wird hierzu das Konzept des Disabling Graphen entsprechend erweitert und auf dessen Basis aussagenlogische Codierungen hergeleitet. In den Abschnitten 3.2 und 3.3 geben wir Codierungen für *Preconditionaxioms* und *Parallelismaxioms* an, die dann in Abschnitt 3.4 zu Gesamtcodierungen zusammengefasst werden.

3. Codierungen basierend auf Disabling-Enabling-Graphen

In diesem Abschnitt werden aussagenlogische Codierungen hergeleitet, um 1-Linearisierungspläne durch aussagenlogische Erfüllbarkeitstests effizient zu finden. Sie beruhen alle auf einer Erweiterung des Disabling Graphen, der mit *Disabling-Enabling-Graph* bezeichnet wird. In Abschnitt 3.1 werden zunächst verschiedene Arten von Konflikten für STRIPS-Operatoren identifiziert, auf deren Basis der Disabling-Enabling-Graph danach formal definiert wird. Weiterhin werden hinreichende Bedingungen an eine Menge O von STRIPS-Operatoren hergeleitet, die die Linearisierbarkeit von O garantieren. Diese Bedingungen werden dann in den Abschnitten 3.2 und 3.3 in Aussagenlogik umgesetzt und anschließend in Abschnitt 3.4 zu Gesamtcodierungen zusammengefasst.

3.1. Der Disabling-Enabling-Graph

In diesem Abschnitt wird das Konzept des *Disabling-Enabling-Graphen* eingeführt, der den *Disabling-Graphen* von Rintanen et al. aus [RHN04] erweitert. Er stellt die Grundlage zu kleinen und somit effizienten Codierungen für unser Problem dar. Ziel ist es, für einen gegebenen Zustand s und eine Menge von STRIPS-Operatoren $O = \{o_1, \dots, o_n\}$ Eigenschaften zu identifizieren, sodass eine totale Ordnung $<$ mit $o_1 < \dots < o_n$ auf O existiert und $\text{app}_{o_1, \dots, o_n}(s)$ definiert ist.

Hierzu werden für Operatoren o und o' zunächst notwendige Bedingungen erarbeitet, um in unserer Semantik simultan (im Sinne von linearisierbar) ausgeführt werden zu können. Es wird sich zeigen, dass o und o' hierfür *verträgliche Effekte* und *verträgliche Vorbedingungen* haben müssen.

Definition 3.1 (Verträgliche Effekte). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $o, o' \in O$, $o = \langle p, e \rangle$, $o' = \langle p', e' \rangle$ STRIPS-Operatoren.

- Existiert ein Literal m mit $m \in e$ und $\bar{m} \in e'$, so haben o und o' *komplementäre Effekte*.
- Sei $(m_1 \vee m_2)$ eine Invariante in π . Wenn $\bar{m}_1 \in e$ und $\bar{m}_2 \in e'$, dann haben o und o' *mit einer Invarianten unverträgliche Effekte*.

Die Operatoren o und o' haben *unverträgliche Effekte*, falls sie komplementäre Effekte oder mit einer Invarianten unverträgliche Effekte haben. Sie haben *verträgliche Effekte* genau dann, wenn sie keine unverträglichen Effekte haben.

Beispiel 3.1. Wir betrachten die folgende Instanz der Logistik-Domäne. Sei

- $P = \{\text{truck-at-pos1}, \text{truck-at-pos2}, \text{truck-at-pos3}\}$,
- $I \models \text{truck-at-pos1} \wedge \neg\text{truck-at-pos2} \wedge \neg\text{truck-at-pos3}$,
- $O = \{\text{drive-truck-pos2}, \text{drive-truck-pos3}\}$, wobei
 $\text{drive-truck-pos2} = \langle \{\text{truck-at-pos1}\}, \{\neg\text{truck-at-pos1}, \text{truck-at-pos2}\} \rangle$ und
 $\text{drive-truck-pos3} = \langle \{\text{truck-at-pos1}\}, \{\neg\text{truck-at-pos1}, \text{truck-at-pos3}\} \rangle$

Aufgrund der Invariante $\Phi = (\neg\text{truck-at-pos2} \vee \neg\text{truck-at-pos3})$ haben die beiden Operatoren unverträgliche Effekte.

Im Folgenden beschreiben wir analog zur Verträglichkeit von Effekten, wann Operatoren verträgliche Vorbedingungen haben. Da die in dieser Arbeit betrachtete Semantik erlaubt, dass Operatoren auch noch von anderen, simultan ausgeführten Operatoren aktiviert werden können, werden die Bedingungen weniger streng als bei Effekten formuliert.

Analog zu Definition 3.1 formulieren wir zunächst die Unverträglichkeit zweier Vorbedingungen aufgrund komplementärer Literale und Invarianten.

Definition 3.2 (Verträgliche Vorbedingungen bezüglich einer Ordnung $<$). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $o, o' \in O$, $o = \langle p, e \rangle$, $o' = \langle p', e' \rangle$ STRIPS-Operatoren.

- Wenn für ein Literal m mit $m \in p$ und $\bar{m} \in p'$ kein Operator $o'' = \langle p'', e'' \rangle \in O$ existiert, für den gilt, dass
 1. $\bar{m} \in e''$,
 2. o' nicht von o'' deaktiviert wird, und
 3. o und o'' sowie o'' und o' verträgliche Effekte haben,

dann haben o und o' mit einem Literal unverträgliche Vorbedingungen bezüglich der Ordnung $o < o'$.

- Sei $(m_1 \vee m_2)$ eine Invariante in π . Wenn $\bar{m}_1 \in p$ und $\bar{m}_2 \in p'$ und kein Operator $o'' = \langle p'', e'' \rangle \in O$ existiert, für den gilt, dass
 1. $\bar{m}_2 \in e''$,
 2. o' nicht von o'' deaktiviert wird, und
 3. o und o'' sowie o' und o'' verträgliche Effekte haben,

dann haben o und o' mit einer Invarianten unverträgliche Vorbedingungen bezüglich der Ordnung $o < o'$.

Die Operatoren o und o' haben *unverträgliche Vorbedingungen bezüglich einer Ordnung $<$* , falls sie mit einem Literal oder einer Invarianten unverträgliche Vorbedingungen bezüglich $<$ haben. Sie haben *verträgliche Vorbedingungen bezüglich einer Ordnung $<$* genau dann, wenn sie keine unverträglichen Vorbedingungen bezüglich $<$ haben.

Haben zwei Operatoren o und o' unverträgliche Vorbedingungen bezüglich der Ordnung $o < o'$, so bedeutet dies anschaulich, dass sie in keinem erreichbaren Zustand in dieser Ordnung ausgeführt werden können.

Haben sie verträgliche Vorbedingungen bezüglich der Ordnung $o < o'$, so kann ein erreichbarer Zustand existieren, in dem sie in dieser Ordnung ausführbar sind. Hierbei können komplementäre Literale oder verletzte Invarianten in den Vorbedingungen noch durch einen Operator o'' aufgelöst werden, damit eine Ausführung $o < o'' < o'$ möglich ist (insbesondere darf natürlich $o'' = o$ gelten).

Beispiel 3.2. Wir betrachten das folgende Planungsproblem $\pi = \langle P, I, O, G \rangle$ aus der Logistik-Domäne:

- $P = \{\text{truck-at-pos1}, \text{truck-at-pos2}, \text{object-at-pos2}, \text{object-in-truck}\},$
- $I \models \text{truck-at-pos1} \wedge \neg \text{truck-at-pos2} \wedge \text{object-at-pos2} \wedge \neg \text{object-in-truck},$
- $O = \{o_1, o_2\}$, wobei $o_1 = \langle \{\text{truck-at-pos1}\}, \{\neg \text{truck-at-pos1}, \text{truck-at-pos2}\} \rangle$, und $o_2 = \langle \{\text{truck-at-pos2}, \text{object-at-pos2}\}, \{\neg \text{object-at-pos2}, \text{object-in-truck}\} \rangle$

Es gilt die Invariante $(\neg \text{truck-at-pos1} \vee \neg \text{truck-at-pos2})$. Trotzdem haben die Operatoren o_1 und o_2 verträgliche Vorbedingungen bezüglich der Ordnung $o_1 < o_2$, da ein Operator $o'' = \langle p'', e'' \rangle (= o_1)$ existiert, für den gilt, dass

- $\text{truck-at-pos2} \in e''$,
- o_2 wird nicht von o'' deaktiviert, und
- alle Operatoren haben verträgliche Effekte.

Somit ist beispielsweise $\text{app}_{o_1, o_2}(I)$ definiert.

Beispiel 3.3. Wir betrachten die drei Operatoren $o_1 = \langle \{A\}, \{B\} \rangle$, $o_2 = \langle \{\neg A\}, \{C\} \rangle$, $o' = \langle \top, \{A\} \rangle$. Die Operatoren o_1 und o_2 haben unverträgliche Vorbedingungen bezüglich der Ordnung $o_1 < o_2$. Sie haben verträgliche Vorbedingungen bezüglich der Ordnung $o_2 < o_1$.

Abschließend ergänzen wir

Definition 3.3 (Verträgliche Vorbedingungen). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $o, o' \in O$, $o = \langle p, e \rangle$, $o' = \langle p', e' \rangle$ STRIPS-Operatoren. Die Operatoren o und o' haben *unverträgliche Vorbedingungen*, falls sie sowohl unverträgliche Vorbedingungen bezüglich der Ordnung $o < o'$ als auch bezüglich der Ordnung $o' < o$ besitzen. Sie haben *verträgliche Vorbedingungen* genau dann, wenn sie keine unverträglichen Vorbedingungen haben.

Anschaulich haben also zwei Operatoren unverträgliche Vorbedingungen, falls sie in keinem erreichbaren Zustand simultan ausgeführt werden können.

Auf der Basis dieser Definitionen wird im Folgenden der *Disabling-Enabling-Graph* für eine Planungsinstanz $\pi = \langle P, I, O, G \rangle$ eingeführt. Analog zum Disabling Graph ist das Ziel die Identifikation von nicht notwendigerweise linearisierbaren Teilmengen $\{o_1, \dots, o_n\} = O' \subseteq O$, für die wir neben der Basiscodierung zusätzliche Bedingungen angeben müssen. Solche Mengen können folgendermaßen charakterisiert werden. Es existiert ein erreichbarer Zustand s , sodass

1. keine Ordnung $<$ mit $o_1 < \dots < o_n$ existiert, sodass $\text{app}_{o_1, \dots, o_n}(s)$ definiert ist, und
2. alle Operatoren $o, o' \in O'$ verträgliche Effekte gemäß Definition 3.1 und verträgliche Vorbedingungen gemäß Definition 3.3 haben.

Die wesentliche Aussage, nämlich dass O' in einem erreichbaren Zustand nicht linearisierbar ist, steckt hierbei im ersten Punkt. Die Forderung aus Punkt zwei dient dazu, nur Operatoren zu betrachten, die in unserer Semantik grundsätzlich parallel ausführbar sind. Da Operatoren $o = \langle p, e \rangle$ und $o' = \langle p', e' \rangle$ auch dann noch verträgliche Vorbedingungen haben können, wenn für ein Literal m gilt, dass $\{m, \bar{m}\} \subseteq (p \cup p')$ (siehe Beispiel 3.3), ist es möglich, dass eine Menge O' diesen Bedingungen genügt, obwohl sie keinen SCC im Disabling Graph im Sinne von Definition 2.19 formt.

Dies führt zur folgenden Definition als Erweiterung des Disabling Graphen.

Definition 3.4 (Disabling-Enabling-Graph). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Der *Disabling-Enabling-Graph* zu π ist der gerichtete Graph $G = \langle O, E \rangle$ mit den folgenden Eigenschaften: Für alle $o, o' \in O$ ist $(o, o') \in E$ genau dann, wenn $o \neq o'$ und

1. o und o' verträgliche Vorbedingungen und verträgliche Effekte haben, und
2. entweder o von o' deaktiviert oder o' von o aktiviert wird.

Der Disabling-Enabling-Graph kann in polynomieller Zeit berechnet werden. Es ergeben sich folgende Unterschiede zum Disabling Graph:

- Es werden zusätzliche Kanten aufgenommen: Zum einen werden Kanten für Paare von Operatoren aufgenommen, die sich aktivieren. Zum anderen können nach Definition 3.3 auch Operatoren mit komplementären Literalen in den Vorbedingungen noch verträgliche Vorbedingungen haben.
- Die Menge der Kanten im Disabling Graph ist somit eine Teilmenge der Kanten im Disabling-Enabling-Graph. Eine logische Konsequenz hieraus ist, dass die Operatoren aus einem SCC im Disabling Graph eine Teilmenge eines SCCs im Disabling-Enabling-Graph sind.

Sämtliche Teilmengen $O' \subseteq O$, deren Operatoren verträgliche Vorbedingungen und Effekte haben und für die es einen Zustand gibt, in dem keine Linearisierung von O' existiert, sind in den SCCs des Disabling-Enabling-Graphen enthalten. Zusätzliche Bedingungen, die garantieren, dass eine Linearisierung existiert, müssen folglich nur für solche Mengen von Operatoren angegeben werden, die einen SCC im Graphen bilden.

Zunächst vergleichen wir die beiden Graphen an einem Beispiel.

Beispiel 3.4. Sei $P = \langle P, I, O, G \rangle$ eine Planungsinstanz mit $P = \{A, B, C, D\}$, $I \models A \wedge B \wedge C \wedge D$ und $O = \{o_1, o_2, o_3\}$ mit $o_1 = \langle \{A\}, \{\neg B\} \rangle$, $o_2 = \langle \top, \{\neg A, \neg C\} \rangle$ und $o_3 = \langle \{B, \neg C\}, \{D\} \rangle$.

Die Operatoren haben paarweise verträgliche Vorbedingungen und Effekte. Weiterhin gilt: o_2 deaktiviert o_1 , o_2 aktiviert o_3 und o_1 deaktiviert o_3 . Im Folgenden vergleichen wir den Disabling-Enabling-Graph zu π mit dem entsprechenden Disabling Graph.

Im Disabling-Enabling-Graph (Abbildung 3.1) entsteht ein Zykel: Es existiert ein erreichbarer Zustand s ($s \models A \wedge B \wedge C \wedge D$), in dem die drei Operatoren in keiner Ordnung sequentiell ausführbar sind.

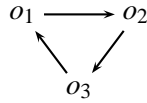


Abbildung 3.1.: Beispiel für einen zyklischen Disabling-Enabling-Graph

Im Disabling Graph (Abbildung 3.2) entsteht hingegen kein Zykel: In allen Zuständen s , in denen die Vorbedingung aller Operatoren erfüllt ist ($s \models A \wedge B \wedge \neg C$), existiert eine Ordnung, in der o_1, o_2 und o_3 sequentiell ausführbar sind ($\text{app}_{o_3; o_1; o_2}(s)$ ist dann definiert).

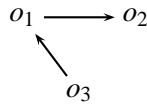


Abbildung 3.2.: Beispiel für einen azyklischen Disabling Graph

Mit Hilfe des Disabling-Enabling-Graphen können nun für einen gegebenen Zustand s hinreichende Bedingungen an eine Menge $O = \{o_1, \dots, o_n\}$ von STRIPS-Operatoren formuliert werden, sodass eine totale Ordnung $o_1 < \dots < o_n$ auf O existiert und $\text{app}_{o_1; \dots; o_n}(s)$ definiert ist. Dies wird im folgenden Satz gezeigt. Er stellt die Grundlage zu den in den folgenden Abschnitten angegebenen aussagenlogischen Codierungen dar.

Satz 3.5. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, s ein Zustand. Sei $\{o_1, \dots, o_n\} = O' \subseteq O$ eine Menge von Operatoren mit paarweise verträglichen Vorbedingungen und paarweise verträglichen Effekten. Sei $G = \langle V, E \rangle$ der Disabling-Enabling-Graph zu π . Es gelte:

1. Der von O' induzierte Teilgraph von G ist zykliefrei.
2. Für alle $\langle p, e \rangle = o \in O'$ und alle $m \in p$: Falls $s \models \bar{m}$, so existiert ein Operator $\langle p', e' \rangle = o' \in O'$, $o' \neq o$, mit $m \in e'$.

Dann gilt: Es existiert eine totale Ordnung $o_1 < \dots < o_n$ auf O' , sodass $\text{app}_{o_1; \dots; o_n}(s)$ definiert ist.

Beweis. Da der von O' induzierte Teilgraph $G' = \langle V', E' \rangle$ von G zykliefrei ist, existiert eine totale Ordnung $<$ auf O' mit der Eigenschaft, dass für alle $o, o' \in O'$ mit $(o, o') \in E'$ auch $o < o'$ gilt. Für diese Ordnung $o_1 < \dots < o_n$ auf O' gilt: $\text{app}_{o_1; \dots; o_n}(s)$ ist definiert, d.h. die Vorbedingung von o_i ist wahr nach der Ausführung von o_1, \dots, o_{i-1} .

Der Beweis erfolgt per vollständiger Induktion über i , $1 \leq i \leq n$.

Induktionsanfang $i = 1$: Betrachte $o_1 = \langle p_1, e_1 \rangle$. Es gilt: $s \models p_1$, denn angenommen, es existiert ein Literal $m \in p_1$ mit $s \models \bar{m}$. Dann existiert nach Voraussetzung ein Operator $o' = \langle p', e' \rangle \in O'$ mit $m \in e'$. Nach Voraussetzung haben o_1 und o' verträgliche Vorbedingungen und Effekte, daher wäre dann $(o', o_1) \in E'$ im Widerspruch zur Wahl der Ordnung $<$ (o_1 wäre dann nicht der erste Operator bzgl. $<$). Somit ist $\text{app}_{o_1}(s)$ definiert.

Induktionsschritt $i \rightarrow i + 1$: Sei $s_i = \text{app}_{o_1, \dots, o_i}(s)$ definiert. Wir zeigen: $\text{app}_{o_{i+1}}(s_i)$ ist definiert. Betrachte $o_{i+1} = \langle p_{i+1}, e_{i+1} \rangle$ und $m \in p_{i+1}$ beliebig. Wir unterscheiden zwei Fälle.

1. $s \models m$. Annahme, es existiert ein Operator $\langle p, e \rangle = o \in \{o_1, \dots, o_i\}$ mit $\bar{m} \in e$. Da o_{i+1} und o nach Voraussetzung verträgliche Vorbedingungen und Effekte haben, wäre dann $(o_{i+1}, o) \in E'$ im Widerspruch zur Wahl von $<$ (o_{i+1} hätte vor o ausgeführt werden müssen). Somit kann ein solcher Operator nicht existieren und es gilt $s_i \models m$.
2. $s \models \bar{m}$. Nach Voraussetzung existiert ein Operator $o' = \langle p', e' \rangle \in O'$ mit $m \in e'$. Weiterhin haben o_{i+1} und o' nach Voraussetzung verträgliche Vorbedingungen und Effekte. Also ist $(o', o_{i+1}) \in E'$ und es gilt nach Konstruktion von $<$, dass $o' \in \{o_1, \dots, o_i\}$. Da alle Operatoren aus O' insbesondere konsistente Effekte haben, folgt: $s_i \models m$.

In beiden Fällen gilt also $s_i \models m$. Da $m \in p_{i+1}$ beliebig gewählt war, gilt auch $s_i \models p_{i+1}$ und somit ist $\text{app}_{o_{i+1}}(s_i)$ definiert. \square

Für einen gegebenen Zustand s_0 und eine Menge O von STRIPS-Operatoren folgt dann für $T = \langle S_0, \dots, S_{l-1} \rangle \in (2^O)^l$: Sind die Voraussetzungen von Satz 3.5 für alle $S_i = \{o_1^i, \dots, o_{n_i}^i\}$ erfüllt und ist somit $s_{i+1} = \text{app}_{o_1^i, \dots, o_{n_i}^i}(s_i)$ definiert für alle $i \in \{0, \dots, l-1\}$, so ist T ein 1-Linearisierungs-Plan der Länge l . In den folgenden Abschnitten werden nun Codierungen in Aussagenlogik angegeben, die garantieren, dass die Voraussetzungen von Satz 3.5 erfüllt sind. Dies bedeutet zum einen, dass für jeden Operator und für jedes Literal aus dessen Vorbedingung, das noch nicht wahr ist, ein anderer Operator ausgeführt werden muss, der es wahr macht (Codierung in Abschnitt 3.2). Zum anderen muss der zugehörige Disabling-Enabling-Graph zyklfrei sein (Codierungen in Abschnitt 3.3). Diese Codierungen werden dann in Abschnitt 3.4 zu Gesamtcodierungen zusammengefasst.

3.2. Umsetzung der Vorbedingungsaxiome

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. In diesem Abschnitt wird eine Codierungen angegeben, die für jeden Operator $o \in O$ sicherstellt, dass jedes Literal aus seiner Vorbedingung entweder bereits wahr ist oder durch einen anderen Operator wahr gemacht wird.

Hierzu definieren wir für jeden Operator $o = \langle p, e \rangle$ und alle Literale $m \in p$ die Menge der Operatoren, die in unserer Semantik simultan mit o angewandt werden können und m wahrmachen. Diese Menge bezeichnen wir mit $\text{enablingOperators}^\pi(o; m)$.

Definition 3.6 (Aktivierungsoperatoren). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und $\langle p, e \rangle = o \in O$ ein STRIPS-Operator. Für alle $m \in p$ sei $\text{enablingOperators}^\pi(o; m) \subseteq O$ die Menge mit der Eigenschaft, dass $o' = \langle p', e' \rangle \in \text{enablingOperators}^\pi(o; m)$ genau dann, wenn $o \neq o'$ und

1. $m \in e'$,
2. o' und o verträgliche Vorbedingungen bezüglich der Ordnung $o' < o$ haben,
3. o und o' verträgliche Effekte haben, und
4. o nicht von o' deaktiviert wird.

Die Menge *enablingOperators* enthält also Operatoren, die simultan mit o ausgeführt werden und o aktivieren können. Für alle $o \in O$ sei dann

$$\Phi_o = (o \rightarrow \bigwedge_{m \in p} (\bar{m} \rightarrow \bigvee \text{enablingOperators}^\pi(o; m))),$$

wobei $\bigvee \emptyset$ mit \perp identifiziert wird.

Definition 3.7. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Dann ist

$$\text{Preconditionaxioms}_1^\pi = \bigwedge_{o \in O} \Phi_o$$

die Konjunktion der Formeln Φ_o für alle Operatoren $o \in O$.

Φ_o ist bereits linear in der Anzahl der Operatoren, was zu einem quadratisch großen Konjunktionsglied führen kann.

Lemma 3.8. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $P = \{a_1, \dots, a_k\}$ und $O = \{o_1, \dots, o_n\}$. Die Größe von $\text{Preconditionaxioms}_1^\pi$ ist in $O(n^2 \cdot k)$.

Beweis. Die Größe von Φ_o für ein $o \in O$ ist in $O(n \cdot k)$, somit ist die Größe von $\text{Preconditionaxioms}_1^\pi$ in $n \cdot O(n \cdot k) = O(n^2 \cdot k)$. \square

Es hat sich dennoch gezeigt, dass diese Codierung in der Praxis effizient ist. Dies liegt zum einen daran, dass keine zusätzlichen Hilfsvariablen benötigt werden. Zum anderen werden viele Operatoren durch die Definition von *enablingOperators* aufgrund unverträglicher Vorbedingungen oder Effekte bereits ausgeschlossen.

Bemerkung 3.9. Mittels zusätzlicher Hilfsvariablen ist es auch möglich, eine lineare Codierung für unseren Zweck herzuleiten. Diese ist aber in der Praxis deutlich weniger effizient. Daher wird darauf verzichtet, sie hier anzugeben.

Seien s und s' erreichbare Zustände, die durch Zustandsvariablen aus P bzw. P' repräsentiert werden. Mit der Definition von *enablingOperators* und $\text{Preconditionaxioms}_1^\pi$ zeigen wir im Folgenden, dass für alle aussagenlogischen Belegungen v von $P \cup P' \cup \text{Var}(O)$ mit

$$v \models \text{Preconditionaxioms}_1^\pi \wedge \text{Effectaxioms}^\pi \wedge \text{Invariantaxioms}^\pi$$

gilt, dass die der Belegung v entsprechenden simultan ausgeführten Operatoren, die von s in s' führen, stets verträgliche Vorbedingungen und verträgliche Effekten haben (Effectaxioms^π und $\text{Invariantaxioms}^\pi$ sind hierbei gemäß der Basiscodierung für die Zustandsvariablen aus P bzw. P' definiert). Dadurch wird gerechtfertigt, dass wir zwischen Operatoren mit unverträglichen Vorbedingungen oder unverträglichen Effekten keine Kante im Disabling-Enabling-Graph benötigen, da sie um Hinblick auf die Linearisierbarkeit simultan ausgeführter Operatoren unkritisch sind.

Satz 3.10. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, s ein von I erreichbarer Zustand und $o_1, o_2 \in O$. Sei ferner v eine Belegung von $P \cup P' \cup \text{Var}(O)$ mit $v(a) = s(a)$ für alle $a \in P$ und $v \models \text{Effectaxioms}^\pi$, $v \models \text{Invariantaxioms}^\pi$ sowie $v \models o_1 \wedge o_2$. Dann haben o_1 und o_2 verträgliche Effekte.

Beweis. Um zu zeigen, dass o_1 und o_2 verträgliche Effekte haben, muss ausgeschlossen werden, dass sie komplementäre oder mit einer Invarianten unverträgliche Effekte haben.

1. Annahme, es existiert ein Literal m mit $m \in e_1$ und $\bar{m} \in e_2$. Wegen $v \models \text{Effectaxioms}^\pi$ gilt dann aber im Nachfolgezustand $v \models m'$ und $v \models \bar{m}'$, Widerspruch.
2. Annahme, es existiert eine Invariante $(m_1 \vee m_2)$ in π mit $\bar{m}_1 \in e_1$ und $\bar{m}_2 \in e_2$. Widerspruch zu $v \models \text{Invariantaxioms}^\pi$ (die Invariante wäre im Nachfolgezustand verletzt).

In allen anderen Fällen als 1. und 2. ist die Verträglichkeit der Effekte unmittelbar klar. Insgesamt folgt also mit 1. und 2. die Behauptung. \square

Eine analoge Aussage gilt für die Vorbedingungen von Operatoren.

Satz 3.11. *Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, s ein von I erreichbarer Zustand und $o_1, o_2 \in O$. Sei ferner v eine Belegung von $P \cup P' \cup \text{Var}(O)$ mit $v(a) = s(a)$ für alle $a \in P$, $v \models \text{Preconditionaxioms}_1^\pi$, $v \models \text{Effectaxioms}^\pi$, $v \models \text{Invariantaxioms}^\pi$ und $v \models o_1 \wedge o_2$. Dann haben o_1 und o_2 verträgliche Vorbedingungen.*

Beweis. Um zu zeigen, dass $o_1 = \langle p_1, e_1 \rangle$ und $o_2 = \langle p_2, e_2 \rangle$ verträgliche Vorbedingungen haben, muss ausgeschlossen werden, dass sie bezüglich einer Ordnung unverträgliche Vorbedingungen mit einem Literal oder einer Invarianten haben.

1. Annahme, es existiert ein Literal m mit $m \in p_1$ und $\bar{m} \in p_2$.
 - a) $s \models m$. Wegen $v \models o_2$ und $v \models (o_2 \rightarrow \bigwedge_{\bar{m} \in p_2} (m \rightarrow \bigvee \text{enablingOperators}^\pi(o_2; \bar{m})))$ folgt: es existiert ein Operator $o'' \in O$, $o'' = \langle p'', e'' \rangle$, mit $v \models o''$, $\bar{m} \in e''$, o_2 und o'' haben verträgliche Effekte und o_2 wird nicht von o'' deaktiviert (nach Definition von $\text{enablingOperators}^\pi(o_2; m)$). Aus $v \models o_1 \wedge o''$ folgt aus Satz 3.10, dass o_1 und o'' verträgliche Effekte haben.
Aus der Existenz von o'' folgt: o_1 und o_2 haben verträgliche Vorbedingungen bezüglich der Ordnung $o_1 < o_2$ und somit verträgliche Vorbedingungen gemäß Definition 3.3.
 - b) $s \not\models m$. Analog zeigt man, dass o_1 und o_2 verträgliche Vorbedingungen bezüglich der Ordnung $o_2 < o_1$ haben.
2. Annahme, es existiert eine Invariante $(m_1 \vee m_2)$ in π mit $\bar{m}_1 \in p_1$ und $\bar{m}_2 \in p_2$.
 - a) $s \models \bar{m}_1 \wedge m_2$. Wegen $v \models o_2$ und $\bar{m}_2 \in p_2$ existiert wegen $v \models (o_2 \rightarrow \bigwedge_{\bar{m}_2 \in p_2} (\bar{m}_1 \rightarrow \bigvee \text{enablingOperators}^\pi(o_2; m)))$ ein Operator $o'' = \langle p'', e'' \rangle$ mit $v \models o''$, $\bar{m}_2 \in e''$, o_2 und o'' haben verträgliche Effekte und o_2 wird nicht von o'' deaktiviert. Wegen $v \models o_1 \wedge o''$ folgt aus Satz 3.10, dass o_1 und o'' verträgliche Effekte haben.
Somit gilt: o_1 und o_2 haben verträgliche Vorbedingungen bezüglich der Ordnung $o_1 < o_2$ und somit verträgliche Vorbedingungen gemäß Definition 3.3.
 - b) $s \models m_1 \wedge \bar{m}_2$. Analog zeigt man: o_1 und o_2 haben verträgliche Vorbedingungen bezüglich der Ordnung $o_2 < o_1$.

- c) Die Fälle $s \models \overline{m_1} \wedge \overline{m_2}$ (im Widerspruch zur Voraussetzung wäre die Invariante in s verletzt und s somit nicht erreichbar) bzw. $s \models m_1 \wedge m_2$ (o_1 und o_2 könnten nicht parallel ausgeführt werden, da simultan auch Operatoren mit $\overline{m_1}, \overline{m_2}$ sowie entweder m_1 oder m_2 im Effekt ausgeführt werden müssten) sind nicht möglich.

In allen anderen Fällen als 1. und 2. ist die Verträglichkeit der Vorbedingungen unmittelbar klar. Insgesamt folgt also mit 1. und 2. die Behauptung. \square

Da also in keinem erreichbaren Zustand Operatoren mit unverträglichen Vorbedingungen oder unverträglichen Effekten simultan ausgeführt werden, sind diese im Hinblick auf eine Linearisierung von simultan ausgeführter Operatoren unkritisch und wir benötigen keine Kante im Disabling-Enabling-Graph.

3.3. Umsetzungen der Parallelitätsaxiome

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. In diesem Abschnitt werden drei Codierungen angegeben, die die Azyklizität des Disabling-Enabling-Graphen garantieren. Da Zykel nur in den SCCs mit mindestens zwei Operatoren entstehen können, werden die Bedingungen zur Gewährleistung der Azyklizität für diese Komponenten spezifiziert.

Die Codierung in Abschnitt 3.3.1 beschreibt einen exakten Azyklizitätstest, in Abschnitt 3.3.2 wird den Operatoren eine feste Ausführungsreihenfolge zugewiesen. Die Codierung in Abschnitt 3.3.3 stellt einen Kompromiss zwischen den beiden ersten dar.

3.3.1. Allgemeine Codierung basierend auf exaktem Azyklizitätstest

In diesem Abschnitt beschreiben wir einen exakten Azyklizitätstest im Disabling-Enabling-Graph. Die entsprechende Codierung ist analog zur Codierung von Rintanen et al. [RHN04] für den reinen Disabling Graph.

Sei $G_k = \langle V_k, E_k \rangle$ ein SCC des Disabling-Enabling-Graphen mit den Operatoren o_1, \dots, o_n ($n \geq 2$). Für jedes Paar von Operatoren o_i und o_j führen wir eine Hilfsvariable $d_{i,j}$ ein, die genau dann wahr sein soll, wenn ein Pfad von o_i nach o_j im Graphen existiert und alle Operatoren auf diesem Pfad ausgeführt werden. Es muss also garantiert werden, dass $d_{i,j}$ wahr ist, falls Operatoren $o_i, o'_1, \dots, o'_k, o_j$ ausgeführt werden, für die gilt, dass eine Kante von jedem Operator zum unmittelbaren Nachfolger existiert.

Hierzu müssen für alle Paare von Operatoren $o_i, o_j \in V_k$ mit $(o_i, o_j) \in E_k$ die folgenden Bedingungen gelten.

1. Da ein Pfad von o_i zu o_j existiert, muss $d_{i,j}$ wahr sein, falls beide Operatoren ausgeführt werden.

$$(o_i \wedge o_j) \rightarrow d_{i,j}$$

2. Weiterhin gilt: Existiert ein Pfad von o_j zu einem Operator o_k , dann existiert wegen $(o_i, o_j) \in E_k$ auch ein Pfad von o_i zu o_k . Daher definieren wir für alle k mit $j \neq k \neq i$

die Formel

$$(o_i \wedge d_{j,k}) \rightarrow d_{i,k}.$$

3. Abschließend muss verhindert werden, dass Operatoren ausgeführt werden, die einen Zykel im Graph vervollständigen.

$$\neg(o_i \wedge d_{j,i})$$

Sei $\Phi_{o_i; o_j}$ die Konjunktion der obigen Formeln für das Operatorpaar (o_i, o_j) .

Zusammenfassend erhält man

Definition 3.12. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Seien $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs im Disabling-Enabling-Graph zu π mit $|V_i| \geq 2$ für alle $1 \leq i \leq k$. Dann ist

$$\text{Parallelismaxioms}_{1, G_i}^\pi = \bigwedge_{(o, o') \in V_i \times V_i} \Phi_{o; o'}$$

die Konjunktion dieser Formeln für alle SCCs G_i , $1 \leq i \leq k$. Insgesamt erhalten wir

$$\text{Parallelismaxioms}_1^\pi = \bigwedge_{i=1}^k \text{Parallelismaxioms}_{1, G_i}^\pi$$

als Formel zur Verhinderung von Zykeln im gesamten Graphen. Die Menge der zusätzlich benötigten Hilfsvariablen $d_{i,j}$ bezeichnen wir hierbei mit Aux_1^π .

Im Folgenden wird die Korrektheit dieser Codierung gezeigt.

Lemma 3.13. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und G der Disabling-Enabling-Graph zu π . Sei ν eine Belegung von $\text{Var}(O) \cup \text{Aux}_1^\pi$ mit $\nu \models \text{Parallelismaxioms}_1^\pi$. Sei $O' = \{o \mid \nu \models o\}$ die Menge der ausgeführten Operatoren. Dann gilt: Der von O' induzierte Teilgraph von G ist zyklfrei.

Beweis. Annahme, G ist nicht zyklfrei, d.h. es existieren ein SCC $G_i = \langle V_i, E_i \rangle$ und Operatoren $o_1^i, \dots, o_n^i \in V_i \cap O'$ mit $(o_1^i, o_2^i), \dots, (o_{n-1}^i, o_n^i), (o_n^i, o_1^i) \in E_i$.

Es gilt für alle $1 \leq j \leq n-1$ wegen $(o_j^i, o_{j+1}^i) \in E_i$ und $\nu \models o_j^i \wedge o_{j+1}^i$: $\nu \models d_{j, j+1}$. Somit gilt auch $\nu \models d_{1, n}$, Widerspruch zu $\nu \models \neg(o_n^i \wedge d_{1, n})$. \square

Die Größe von $\text{Parallelismaxioms}_1^\pi$ wächst kubisch in der Anzahl der Operatoren.

Satz 3.14. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und $O = \{o_1, \dots, o_n\}$. Dann hat die Codierung $\text{Parallelismaxioms}_1^\pi$ die Größenordnung $O(n^3)$ in der Anzahl n der Operatoren.

Beweis. Für jedes Paar von Operatoren (o, o') ist die Größe von $\Phi_{o; o'}$ linear in n , die Anzahl der Paare wächst quadratisch in n . Hieraus folgt die Behauptung. \square

Insgesamt erweist sich diese Codierung aufgrund ihrer Größe und der quadratischen Anzahl zusätzlicher Hilfsvariablen als nicht praktikabel.

3.3.2. Codierung basierend auf fester Reihenfolge der Operatoren

In diesem Abschnitt wird einen Azyklizitätstest linearer Größe in der Anzahl der Operatoren angegeben. Im Gegensatz zum vorigen Abschnitt werden die Bedingungen an simultan ausgeführte Operatoren strenger formuliert.

Die Idee ist, allen Operatoren in den SCCs im Disabling-Enabling-Graphen im Voraus eine feste Ordnung $<$ zuzuweisen und ihre Ausführung nur bezüglich dieser Reihenfolge zuzulassen [RHN04]. Dies schränkt zwar die Anzahl der simultan ausführbaren Operatoren ein, lässt aber eine kompakte Codierung zu.

Seien o_1, \dots, o_n aus einem SCC mit der Ordnung $o_1 < \dots < o_n$. Wir garantieren die Azyklizität, indem wir fordern, dass für alle Operatoren o_i das folgende gilt.

1. Wird ein Operator $o_j \neq o_i$ von o_i deaktiviert und gilt $o_i < o_j$, so dürfen nicht beide simultan ausgeführt werden (“spätere Operatoren dürfen nicht von früheren deaktiviert werden”).
2. Wird ein Operator $o_j \neq o_i$ von o_i aktiviert und gilt $o_j < o_i$, so dürfen nicht beide simultan ausgeführt werden (“frühere Operatoren dürfen nicht von späteren aktiviert werden”).

Es wird sich zeigen, dass diese zwei Bedingungen hinreichend für die Zykelfreiheit im zugehörigen Disabling-Enabling-Graph sind.

Zur Umsetzung dieser Ideen verwenden wir die *chain*-Codierung aus Definition 2.20. Sei hierzu $G_i = \langle V_i, E_i \rangle$ ein SCC des Disabling-Enabling-Graphen mit den Operatoren $o_1^i, \dots, o_n^i \in V_i$ und der Ordnung $o_1^i < \dots < o_n^i$.

Operatoren o und o' aus V_i mit $o < o'$, für die gilt, dass o' von o deaktiviert wird, dürfen nicht simultan angewendet werden. Für alle Literale $m \in \bar{P}$ sei hierzu $E_m^{1,i} = \{o = \langle p, e \rangle \mid o \in V_i, \bar{m} \in e\}$ die Menge der Operatoren, die m falsifizieren und $R_m^i = \{o = \langle p, e \rangle \mid o \in V_i, m \in p\}$ die Menge der Operatoren, für die m in der Vorbedingung enthalten ist. Dann leistet die Formel

$$\text{chain}(o_1^i, \dots, o_n^i; E_m^{1,i}; R_m^i)$$

das Gewünschte.

Weiterhin muss garantiert werden, dass Operatoren o und o' aus V_i mit $o < o'$, für die gilt, dass o von o' aktiviert wird, nicht simultan angewendet werden. Für alle Literale $m \in \bar{P}$ sei hierzu $E_m^{2,i} = \{o = \langle p, e \rangle \mid o \in V_i, m \in e\}$ die Menge der Operatoren, die m wahr machen. Dann leistet die Formel

$$\text{chain}(o_n^i, \dots, o_1^i; E_m^{2,i}; R_m^i)$$

das Gewünschte.

Zusammenfassend erhält man

Definition 3.15. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling-Enabling-Graph zu π . Seien $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs von G mit $|V_i| \geq 2$ für alle $1 \leq i \leq k$. Sei $V_i = \{o_1^i, \dots, o_{n_i}^i\}$ und $o_1^i < \dots < o_{n_i}^i$ eine totale Ordnung auf V_i für alle $1 \leq i \leq k$. Seien ferner die Mengen $E_m^{1,i}, E_m^{2,i}$ und R_m^i für alle $1 \leq i \leq k$ und alle Literale $m \in \bar{P}$ wie oben definiert. Dann ist

$$\text{Parallelismaxioms}_{2,G_i}^\pi = \bigwedge_{m \in \bar{P}} (\text{chain}(o_1^i, \dots, o_{n_i}^i; E_m^{1,i}; R_m^i) \wedge \text{chain}(o_{n_i}^i, \dots, o_1^i; E_m^{2,i}; R_m^i))$$

für alle SCCs G_i . Insgesamt erhalten wir

$$\text{Parallelismaxioms}_2^\pi = \bigwedge_{i=1}^k \text{Parallelismaxioms}_{2,G_i}^\pi.$$

Die Menge der zusätzlich benötigten Hilfsvariablen aus den *chain*-Codierungen bezeichnen wir hierbei mit Aux_2^π .

Satz 3.16. *Sei $\pi = \langle P, I, O, G \rangle$. Die Größe von $\text{Parallelismaxioms}_2^\pi$ ist linear in der Anzahl der Operatoren und Aussagenvariablen.*

Beweis. Seien o_1, \dots, o_n die Operatoren aus einem SCC G im Disabling-Enabling-Graph zu π und $P = \{a_1, \dots, a_k\}$. Wir untersuchen die Mächtigkeit der drei Konjunktionsglieder der Formel $\text{chain}(o_1^i, \dots, o_n^i; E_m^1; R_m)$ für ein Literal m . Hierzu seien die G entsprechenden Mengen E_m^1 und R_m wie oben definiert.

1. $|\{o_i \rightarrow a^{j,m} \mid i < j, o_i \in E_m^1, o_j \in R_m, \{o_{i+1}, \dots, o_{j-1}\} \cap R_m = \emptyset\}| = O(|E_m^1|) = O(n)$ (nur für jeden Operator aus E_m^1 existiert eine Klausel).
2. $|\{a^{i,m} \rightarrow a^{j,m} \mid i < j, \{o_i, o_j\} \subseteq R_m, \{o_{i+1}, \dots, o_{j-1}\} \cap R_m = \emptyset\}| = O(|R_m|) = O(n)$ (für je zwei verschiedene Klauseln existieren Operatoren $o, o' \in R_m$ mit $o \neq o'$).
3. $|\{a^{j,m} \rightarrow \neg o_j \mid o_j \in R_m\}| = O(|R_m|) = O(n)$ (jedem Operator aus R_m ist genau eine Klausel zugeordnet und umgekehrt, d.h. die Menge der Klauseln kann bijektiv auf R_m abgebildet werden).

Insgesamt ist die Größe von *chain* für ein Literal m somit in $O(n)$. Die Größe der Formel $\text{Parallelismaxioms}_2^\pi$ ist somit in $O(n \cdot k)$. \square

Die Anzahl der zusätzlichen Hilfsvariablen ist ebenfalls linear in der Anzahl der Operatoren und der Aussagenvariablen.

Lemma 3.17. *Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und G der Disabling-Enabling-Graph zu π . Sei v eine Belegung von $\text{Var}(O) \cup \text{Aux}_2^\pi$ mit $v \models \text{Parallelismaxioms}_2^\pi$. Sei $O' = \{o \mid v \models o\}$ die Menge der ausgeführten Operatoren. Dann gilt: Der von O' induzierte Teilgraph von G ist zyklfrei.*

Beweis. Sei $G' = \langle V', E' \rangle$ ein beliebiger SCC von G mit gemäß Definition 3.15 vorgegebener Ordnung $<$ auf V' . Seien $o = \langle p, e \rangle, o' = \langle p', e' \rangle \in V'$ mit $o < o'$ und $(o', o) \in E'$. Dann unterscheiden wir zwei mögliche Fälle.

1. Es existiert ein Literal m mit $m \in e$ und $\bar{m} \in p'$, d.h. o deaktiviert o' .
Wegen $v \models \text{chain}(o_1, \dots, o_n; E_m^1; R_m)$ für alle Literale m gilt dann aber $v \models o \rightarrow \neg o'$.
2. Es existiert ein Literal m mit $m \in e'$ und $m \in p$, d.h. o' aktiviert o .
Wegen $v \models \text{chain}(o_n, \dots, o_1; E_m^2; R_m)$ für alle Literale m gilt dann aber $v \models o' \rightarrow \neg o$.

In beiden Fällen werden also wegen $v \models \text{Parallelismaxioms}_2^\pi$ nicht beide Operatoren simultan ausgeführt. Die Annahme eines Zyklus in dem von O' induzierten Teilgraph von G führt somit sofort zu einem Widerspruch: Annahme, es existieren ein SCC $G' = \langle V', E' \rangle$ und Operatoren $o_1, \dots, o_n \in V' \cap O'$ mit $(o_1, o_2) \in E', \dots, (o_{n-1}, o_n) \in E'$ und $(o_n, o_1) \in E'$. Dann muss aber für mindestens zwei dieser Operatoren, o.E. für o_1 und o_n , gelten, dass $o_1 < o_n$ und $(o_n, o_1) \in E'$. Aufgrund der obigen Argumentation können dann aber nicht beide simultan ausgeführt werden, d.h. $\{o_1, o_n\} \not\subseteq O'$. \square

3.3.3. Codierung basierend auf Feedback-Vertices

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Im Folgenden wird ein Azyklizitätstest angegeben, der in vielen Fällen mehr Parallelität als $\text{Parallelismaxioms}_2^\pi$ zulässt. Er beruht auf *Feedback-Vertices* und wurde in ähnlicher Weise von Khomenko et al. angegeben [KKKV05]. Zunächst benötigen wir hierfür das Konzept des *Feedback-Vertex*, das analog zu [KKKV05] definiert wird.

Definition 3.18 (Feedback-Vertex). Sei $G = \langle V, E \rangle$ ein gerichteter Graph und $<$ eine totale Ordnung auf V . Ein Knoten $v \in V$ heißt *Feedback-Vertex bezüglich $<$* , falls ein Knoten $v' \in V$ mit $v < v'$ und $(v', v) \in E$ existiert.

Beispiel 3.5. Sei $G = \langle V, E \rangle$ ein gerichteter Graph mit $V = \{v_1, v_2, v_3\}$ und $E = \{(v_1, v_2), (v_2, v_3)\}$ (Abbildung 3.3). Bezüglich der Ordnung $v_3 < v_2 < v_1$ sind v_3 und v_2 Feedback-Vertices, bezüglich $v_3 < v_1 < v_2$ ist v_3 Feedback-Vertex. Bezüglich der Ordnung $v_1 < v_2 < v_3$ gibt es keine Feedback-Vertices.

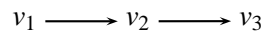


Abbildung 3.3.: Ein azyklischer gerichteter Graph

Es gilt: Sind v_1, \dots, v_n die Feedback-Vertices bezüglich einer Ordnung $<$ von V , dann ist der von $V \setminus \{v_1, \dots, v_n\}$ induzierte Teilgraph zykliefrei (im Fall eines Zyklus muss bezüglich jeder Ordnung noch mindestens ein Feedback-Vertex existieren).

Beispiel 3.6. Sei $G = \langle V, E \rangle$ ein gerichteter Graph mit $V = \{v_1, v_2, v_3\}$ und $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$ (Abbildung 3.4). Dann existiert bezüglich jeder Ordnung $<$ auf V mindestens ein Feedback-Vertex bezüglich $<$. Wird beispielsweise die Ordnung $v_1 < v_2 < v_3$ gewählt, so ist v_1 Feedback-Vertex bezüglich $<$, da für v_3 gilt: $v_1 < v_3$ und $(v_3, v_1) \in E$.

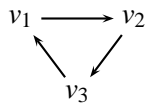


Abbildung 3.4.: Ein zyklischer gerichteter Graph

Im Folgenden sprechen wir der Einheit halber nur von einem Feedback-Vertex und lassen den Zusatz, welche Ordnung gemeint ist, weg, wenn dies klar ist bzw. aus dem Kontext hervorgeht.

Sei $G_k = \langle V_k, E_k \rangle$ ein SCC des Disabling-Enabling-Graphen mit den Operatoren o_1, \dots, o_n . Für einen exakten Azyklizitätstest benötigt man quadratisch viele Hilfsvariablen (vgl. Abschnitt 3.3.1). Die folgende Codierung kommt mit linear vielen Hilfsvariablen aus. Wir wählen hierzu zunächst eine feste Ordnung $<$ auf V_k . Im Unterschied zum vorigen Abschnitt ist dies jedoch keine Restriktion bezüglich der Ausführungsreihenfolge (siehe Beispiel 3.7), sondern eine Maßnahme zur Identifikation von Feedback-Vertices. Um die Existenz einer Linearisierung einer Menge $O' \subseteq V_k$ basierend auf $<$ zu garantieren, müssen für die entsprechenden Feedback-Vertices in V_k zusätzliche Bedingungen formuliert werden.

Hierzu führen wir für alle $o_i \in V_k$ eine Hilfsvariable d_i ein, die genau dann wahr sein soll, wenn ein Pfad von einem Feedback-Vertex zu o_i existiert und alle Operatoren auf diesem Pfad ausgeführt werden. Wir fordern, dass ein Operator, der einen potentiellen Zykel vervollständigt, von keinem Feedback-Vertex aus erreichbar sein darf. Sei $G_k = \langle V_k, E_k \rangle$ ein SCC mit $o_1, \dots, o_n \in V_k$ und $<$ eine totale Ordnung auf V_k .

Für alle $o_i, o_j \in V_k$ gilt: Wenn ein Pfad von einem Feedback-Vertex zu o_i existiert und weiterhin $(o_i, o_j) \in E_k$ ist, dann existiert auch ein Pfad von einem Feedback-Vertex zu o_j .

$$\Phi_{E_k} = \bigwedge_{\substack{i,j=1 \\ (o_i,o_j) \in E_k}}^n (d_i \wedge o_j) \rightarrow d_j$$

Weiterhin darf kein Operator, der einen potentiellen Zykel im Graph vervollständigt, von einem Feedback-Vertex erreichbar sein. Für jeden Feedback-Vertex o_i definieren wir

$$\Psi_{E_k}^{o_i} = (o_i \rightarrow (d_i \wedge \bigwedge_{(o_j,o_i) \in E_k} \neg d_j)).$$

Um zu verhindern, dass von einem Feedback-Vertex o ein Pfad zu sich selbst existiert, wurde also eine strengere Bedingung formuliert und gefordert, dass o von *keinem* Feedback-Vertex aus erreichbar ist. So benötigt man im Unterschied zur Codierung *Parallelismaxioms* $^{\pi}_1$ aus Abschnitt 3.3.1 nur linear viele Hilfsvariablen, im Unterschied zur Codierung *Parallelismaxioms* $^{\pi}_2$ aus Abschnitt 3.3.2 ist die vorgegebene Ordnung keine Einschränkung bezüglich der Ausführungsreihenfolge der Operatoren. Dies führt zu einer in der Praxis effizienten Codierung, die mehr Parallelität als *Parallelismaxioms* $^{\pi}_2$ zulässt.

Beispiel 3.7. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz mit $O = \{o_1, o_2, o_3\}$. Sei G der Disabling-Enabling-Graph zu π (Abbildung 3.5). Sei $o_1 < o_2 < o_3$ eine Ordnung auf O . Mit der Codierung

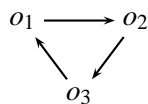


Abbildung 3.5.: Beispiel für einen zyklischen Disabling-Enabling-Graph

Parallelismaxioms $^{\pi}_2$ aus Abschnitt 3.3.2 können o_1 und o_3 bezüglich dieser Ordnung nicht zeitgleich ausgeführt werden. Mit der Codierung aus diesem Abschnitt können o_1 und o_3 simultan ausgeführt werden, wenn nicht noch gleichzeitig o_2 ausgeführt wird.

Zusammenfassend erhält man

Definition 3.19. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der zugehörige Disabling-Enabling-Graph. Seien $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs in G mit $|V_i| \geq 2$ und einer totalen Ordnung $<_i$ auf V_i für alle $1 \leq i \leq k$. Dann ist

$$\text{Parallelismaxioms}_{3,G_i}^\pi = \Phi_{E_i} \wedge \bigwedge_{\substack{j=1 \\ o_j \text{ Feedback-Vertex} \\ \text{bzgl. } <_i}}^{|V_i|} \Psi_{E_i}^{o_j}$$

für alle SCCs $G_i = \langle V_i, E_i \rangle$. Insgesamt ist dann

$$\text{Parallelismaxioms}_3^\pi = \bigwedge_{i=1}^k \text{Parallelismaxioms}_{3,G_i}^\pi.$$

Die Menge der zusätzlichen Hilfsvariablen d_i sei hierbei mit Aux_3^π bezeichnet.

So ist gewährleistet, dass der von simultan ausgeführten Operatoren induzierte Teilgraph von G zyklfrei ist. Dies wird im folgenden Lemma gezeigt.

Lemma 3.20. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und G der Disabling-Enabling-Graph zu π . Sei v eine Belegung von $\text{Var}(O) \cup \text{Aux}_3^\pi$ mit $v \models \text{Parallelismaxioms}_3^\pi$. Sei $O' = \{o \mid v \models o\}$ die Menge der ausgeführten Operatoren. Dann gilt: Der von O' induzierte Teilgraph G' von G ist zyklfrei.

Beweis. Annahme, G' ist nicht zyklfrei, d.h. es existiert ein SCC $G_i = \langle V_i, E_i \rangle$ und Operatoren $o_1^i, \dots, o_n^i \in V_i \cap O'$ mit der gemäß Definition 3.19 vorgegebenen Ordnung $o_1^i < \dots < o_n^i$ und $(o_1^i, o_2^i), \dots, (o_{n-1}^i, o_n^i), (o_n^i, o_1^i) \in E_i$. Dann existiert ein $k \in \{2, \dots, n\}$ mit $(o_k^i, o_1^i) \in E_i$ (sonst formen o_1^i, \dots, o_n^i keinen Zykel). Somit ist o_1^i Feedback-Vertex bezüglich $o_1^i < \dots < o_n^i$ und es gilt wegen $v \models (o_1^i \rightarrow d_1)$: $v \models d_1$. Wegen $v \models \Phi_{E_i}$ gilt auch $v \models \bigwedge_{j=2}^n d_j$. Wegen $v \models \Psi_{E_i}^{o_1^i}$ gilt aber auch $v \models \neg d_k$, Widerspruch. Somit kann kein Zykel existieren. \square

Satz 3.21. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz mit $O = \{o_1, \dots, o_n\}$, $G = \langle V, E \rangle$ der Disabling-Enabling-Graph zu π und $e = |E|$. Dann gilt: Die Größe von $\text{Parallelismaxioms}_3^\pi$ ist in $O(n+e)$.

Beweis. Im schlechtesten Fall existiert nur ein SCC $G_1 = \langle O, E \rangle = G$ mit $O = \{o_1, \dots, o_n\}$. Für jeden Operator o_i bezeichne im Folgenden $e_i = \{(o_j, o_i) \mid o_j \in O, (o_j, o_i) \in E\}$ die Menge der eingehenden Kanten von o_i . Die Größe von $\bigwedge_{i=1}^n \Psi_E^{o_i}$ ist demnach $\sum_{i=1}^n (O(1) + O(|e_i|))$. Wegen $e_i \cap e_j = \emptyset$ für $i \neq j$ gilt $\sum_{i=1}^n O(|e_i|) = O(e)$. Somit ist die Größe von $\bigwedge_{i=1}^n \Psi_E^{o_i}$ aus $O(n+e)$. Die Größe von Φ_E ist $O(e)$. Somit folgt die Behauptung. \square

Insgesamt erhält man eine Codierung, die im schlechtesten Fall (wenn der Disabling-Enabling-Graph viele Kanten enthält) quadratische Größe in der Anzahl der Operatoren erreichen kann. In der Praxis ist sie jedoch in vielen Planungsproblemen aufgrund ihrer geringen Anzahl an Hilfsvariablen und der im Vergleich zu den bisherigen Codierungen größeren Anzahl an parallel ausführbarer Operatoren sehr effizient.

3.4. Gesamtcodierungen

In diesem Abschnitt werden die Gesamtcodierungen angegeben, die wir später in Abschnitt 4 mit der 1-Linearisierungs-Codierung von Rintanen et al. vergleichen. Sie beruhen auf der Azyklizität des Disabling-Enabling-Graphen und bestehen (neben der Basiscodierung) aus der Kombination der Implementierung von *Preconditionaxioms*^π aus Abschnitt 3.2 sowie der drei verschiedenen Implementierungen von *Parallelismaxioms*^π aus Abschnitt 3.3.

Definition 3.22 (Gesamtcodierungen). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, seien die Codierungen *BaseEncoding*^π gemäß Definition 2.17, *Preconditionaxioms*₁^π gemäß Definition 3.7 und *Parallelismaxioms*_i^π, $i = 1, 2, 3$, gemäß den Definitionen 3.12, 3.15 bzw. 3.19 gegeben. Dann sind

$$R_i(P, P') = \text{BaseEncoding}^\pi \wedge \text{Preconditionaxioms}_1^\pi \wedge \text{Parallelismaxioms}_i^\pi, i = 1, 2, 3$$

die Gesamtcodierungen.

Satz 3.23 (Korrektheit). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling-Enabling-Graph zu π . Sei s ein Zustand und ν eine aussagenlogische Belegung von $P \cup P' \cup \text{Var}(O) \cup \text{Aux}_i^\pi$ mit $\nu(a) = s(a)$ für alle $a \in P$ und $\nu \models R_i(P, P')$, $i = 1, 2, 3$. Dann existieren Operatoren $\{o'_1, \dots, o'_{n'}\} = O' \subseteq O$ und eine totale Ordnung $o'_1 < \dots < o'_{n'}$ auf O' , sodass $\text{app}_{o'_1; \dots; o'_{n'}}(s)$ definiert ist.

Beweis. Wir zeigen die Behauptung exemplarisch für $R_1(P, P')$ (der Beweis zu $R_2(P, P')$ und $R_3(P, P')$ ist analog). Sei $O' = \{o \mid \nu \models o\}$ die Menge der ausgeführten Operatoren. Wir zeigen, dass die Vorbedingungen von Satz 3.5 für O' erfüllt sind.

Wegen $\nu \models \text{Preconditionaxioms}_1^\pi \wedge \text{Invariantaxioms}^\pi \wedge \text{Effectaxioms}^\pi$ haben alle Operatoren in O' nach Satz 3.11 paarweise verträgliche Vorbedingungen. Wegen $\nu \models \text{Invariantaxioms}^\pi \wedge \text{Effectaxioms}^\pi$ haben alle Operatoren in O' nach Satz 3.10 paarweise verträgliche Effekte. Weiterhin gilt wegen $\nu \models \text{Preconditionaxioms}_1^\pi$ für alle $o \in O'$, $o = \langle p, e \rangle$: Falls $s \not\models m$ für ein $m \in p$, dann existiert ein simultan ausgeführter Operator $o' \in O'$, $o' = \langle p', e' \rangle$, mit $m \in e'$. Schließlich gilt aufgrund Lemma 3.13: Der von O' induzierte Teilgraph von G ist zyklfrei. Insgesamt folgt somit mit Satz 3.5 die Behauptung. \square

Eine erfüllende Belegung von

$$\Phi_i^n = I^0 \wedge R_i(P^0, P^1) \wedge \dots \wedge R_i(P^{n-1}, P^n) \wedge G^n, i = 1, 2, 3$$

entspricht dann einem 1-Linearisierungs-Plan der Länge n . Hierbei sind I^0 und G^n die Konjunktion der entsprechend mit 0 bzw. n ausgezeichneten Literale, die im Anfangs- bzw. Zielzustand gelten. Die Formel $R_i(P^t, P^{t+1})$ für $i = 1, 2, 3$ und für alle $t \in \{0, \dots, n-1\}$ ist die entsprechende Transitionsrelation, die vom Zustand zum Zeitpunkt t zum Nachfolgezustand zum Zeitpunkt $t+1$ führt, wobei die Zustandsvariablen entsprechend mit t bzw. $t+1$ ausgezeichnet sind.

3.5. Eine weitere Codierung basierend auf fester Reihenfolge der Operatoren

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. In diesem Abschnitt wird eine Verbesserung der Codierung $R_2(P, P')$ vorgestellt, die auf einer festen Reihenfolge der Operatoren in den SCCs des Disabling-Enabling-Graphen zu π beruht. In $R_2(P, P')$ wird ein Azyklizitätstest verwendet, der mit zwei *chain*-Codierungen Zykel verhindert.

Nun ist die Zykelfreiheit zwar eine hinreichende, jedoch keine notwendige Bedingung, um für eine Menge von Operatoren und einen Zustand s zu garantieren, dass eine Linearisierung von O in s existiert.

Beispiel 3.8. Sei $\pi = \langle P, I, O, G \rangle$ das folgende Planungsproblem: $P = \{A, B, C, D\}$, $O = \{o_1, o_2, o_3, o_4\}$ mit $o_1 = \langle \top, \{A\} \rangle$, $o_2 = \langle \{A\}, \{B\} \rangle$, $o_3 = \langle \{B\}, \{C\} \rangle$ und $o_4 = \langle \{C\}, \{A, D\} \rangle$, $I = \neg A \wedge \neg B \wedge \neg C \wedge \neg D$ und $G = A \wedge B \wedge C \wedge D$.

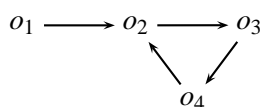


Abbildung 3.6.: Beispiel für einen zyklischen Disabling-Enabling-Graph

Trotz Zykel im Disabling-Enabling-Graphen existiert eine Linearisierung $o_1 < o_2 < o_3 < o_4$ von O im Anfangszustand I . Der Zielzustand wird von dem 1-Linearisierungs-Plan $\langle \{o_1, o_2, o_3, o_4\} \rangle$ in einem Schritt erreicht.

Die Idee ist nun, dass Operatoren o und o' in der Ordnung $o < o'$ auch dann simultan ausgeführt werden dürfen, wenn o von o' aktiviert wird (dies wird in $R_2(P, P')$ durch die zweite *chain*-Codierung verhindert). Im Gegenzug verlangen wir, dass nur solche Operatoren als Aktivierungsoperatoren in $enablingOperators(o; m)$ für einen Operator o und ein Literal m verwendet werden dürfen, die in der (ohnehin vorgegebenen) Ordnung vor o stehen. Im Vergleich zu $R_2(P, P')$ wird also auf die zweite *chain*-Codierung verzichtet und dafür strengere Bedingungen an die Menge $enablingOperators$ gestellt.

Im Folgenden werden wir diese Ideen umsetzen und die Codierung formal definieren.

Definition 3.24. Sei O eine Menge von STRIPS-Operatoren, $<$ eine totale Ordnung auf O und $o \in O$. Dann ist

$$Succ_{<}^O(o) = \{o' \mid o < o'\}$$

die Menge der Operatoren, die nach o in der gegebenen Ordnung $<$ stehen.

Definition 3.25. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling-Enabling-Graph zu π , $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs von G und seien $<_i$ totale Ordnungen auf V_i für alle $i \in \{1, \dots, k\}$. Für alle G_i und alle $o \in V_i$ mit $o = \langle p, e \rangle$ sei dann

$$Preconditionaxioms_2^{G_i} = \bigwedge_{o \in V_i} (o \rightarrow \bigwedge_{m \in p} (\bar{m} \rightarrow \bigvee_{enablingOperators^\pi(o; m) \setminus Succ_{<_i}^{V_i}(o)})).$$

Sei $G_i = \langle V_i, E_i \rangle$ ein SCC von G mit $V_i = \{o_1^i, \dots, o_{n_i}^i\}$. Analog zu $R_2(P, P')$ muss weiterhin gelten, dass Operatoren $o, o' \in V_i$ mit $o <_i o'$, für die gilt, dass o' von o deaktiviert wird, nicht

simultan ausgeführt werden dürfen. Hierzu seien $E_m^{1,i}$ bzw. R_m^i analog zu Definition 3.15 die Menge der Operatoren aus V_i , die m falsifizieren bzw. für die m in der Vorbedingung enthalten ist. Wie in $R_2(P, P')$ leistet dann die *chain*-Codierung

$$\text{chain}(o_1^i, \dots, o_{n_i}^i; E_m^{1,i}; R_m^i)$$

nach Definition 2.20 das Gewünschte. Es sei

$$\text{Parallelismaxioms}_4^{G_i} = \bigwedge_{m \in \bar{P}} \text{chain}(o_1^i, \dots, o_{n_i}^i; E_m^{1,i}; R_m^i)$$

für alle SCCs G_i , $1 \leq i \leq k$. Dann ist zusammen mit der Basiscodierung BaseEncoding^π

$$R_4^\pi(P, P') = \text{BaseEncoding}^\pi \wedge \bigwedge_{i=1}^k (\text{Preconditionaxioms}_2^{G_i} \wedge \text{Parallelismaxioms}_4^{G_i})$$

die Gesamtcodierung. Die Menge der zusätzlich benötigten Hilfsvariablen aus der *chain*-Codierung bezeichnen wir mit Aux_4^π .

Im Unterschied zu den bisherigen Codierungen können mit $R_4(P, P')$ teilweise auch Operatoren, die einen Zykel im Disabling-Enabling-Graphen formen, simultan ausgeführt werden.

Beispiel 3.9. Sei $O = \{o_1, o_2\}$, $o_1 = \langle \{A\}, \{B\} \rangle$ und $o_2 = \langle \{B\}, \{A\} \rangle$. Sei $o_1 < o_2$ eine Ordnung auf O . Dann ist $\text{enablingOperators}(o_1; A) = \{o_2\}$ und $\text{enablingOperators}(o_2; B) = \{o_1\}$. Wegen $\text{enablingOperators}(o_1; A) \setminus \text{Succ}_<^O(o_1) = \emptyset$ kann o_2 nicht mehr verwendet werden, um o_1 zu aktivieren. Trotz Zykel im Disabling-Enabling-Graph können somit (im Unterschied zu $R_2(P, P')$) in Zuständen s mit $s \models A$ beide Operatoren simultan ausgeführt werden.

$R_4(P, P')$ kann höchstens so groß wie $R_2(P, P')$ werden. Die Korrektheit wird im Folgenden gezeigt. Hierzu macht man sich zunächst klar, dass auch für unsere neue Definition für die Vorbedingungsaxiome gilt, dass nie Operatoren mit unverträglichen Vorbedingungen oder Effekten simultan angewendet werden. Um diese Aussage später referenzieren zu können, wird sie im folgenden Lemma kurz zusammengefasst.

Lemma 3.26. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, s ein von I erreichbarer Zustand und $o_1, o_2 \in O$. Sei ferner ν eine Belegung von $P \cup P' \cup \text{Var}(O)$ mit $\nu(a) = s(a)$ für alle $a \in P$ und $\nu \models \text{Preconditionaxioms}_2^\pi$, $\nu \models \text{Effectaxioms}^\pi$, $\nu \models \text{Invariantaxioms}^\pi$ sowie $\nu \models o_1 \wedge o_2$. Dann haben o_1 und o_2 verträgliche Vorbedingungen und verträgliche Effekte.

Beweis. analog zum Beweis von Satz 3.10 und 3.11 □

Um den Korrektheitsbeweis von $R_4(P, P')$ übersichtlich zu halten, zeigen wir für eine Planungsinstanz $\pi = \langle P, I, O, G \rangle$ im folgenden Lemma schließlich noch die Existenz einer totalen Ordnung auf O , die im sich anschließenden Satz benötigt wird.

Lemma 3.27. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling-Enabling-Graph zu π . Seien $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs von G mit den totalen Ordnungen $<_1, \dots, <_k$ auf V_1, \dots, V_k . Dann existiert eine totale Ordnung $<$ auf O , sodass für $o, o' \in O$ mit $(o, o') \in E$ gilt:

1. Wenn $o, o' \in V_i$ für ein $i \in \{1, \dots, k\}$, dann ist $o < o'$ genau dann, wenn $o <_i o'$.
2. Wenn $o \in V_i$ und $o' \in V_j$ für $i \neq j$, dann ist $o < o'$.

Beweis. Zunächst stellt man fest, dass die SCCs einen azyklischen Graphen formen, d.h. es existiert eine topologische Sortierung $<_{top}$ auf $\{V_1, \dots, V_k\}$, o.E. $V_1 <_{top} \dots <_{top} V_k$, mit der Eigenschaft, dass für alle $i \in \{2, \dots, k\}$ kein Pfad von einem Knoten in V_i zu einem Knoten in V_1, \dots, V_{i-1} existiert. Weiterhin existiert nach Voraussetzung eine totale Ordnung $<_i$ auf $V_i = \{o_1^i, \dots, o_{n_i}^i\}$ für alle $i \in \{1, \dots, k\}$. Dann wird $<$ durch

$$o_1^1 <_1 \dots <_1 o_{n_1}^1 <_{top} \dots <_{top} o_1^k <_k \dots <_k o_{n_k}^k.$$

konstruiert. Für diese Ordnung gilt die Behauptung. Seien hierzu $o, o' \in O$ mit $(o, o') \in E$.

1. Seien $o, o' \in V_i$ für ein $i \in \{1, \dots, k\}$. Dann gilt nach Konstruktion sofort $o <_i o' \Leftrightarrow o < o'$.
2. Seien $o \in V_i$ und $o' \in V_j$ für $i \neq j$. Dann existiert wegen $(o, o') \in E$ ein Pfad von einem Knoten in V_i zu einem Knoten in V_j , d.h. es gilt $V_i <_{top} V_j$ und somit nach Konstruktion von $<$ auch $o < o'$.

Hieraus ergibt sich die Behauptung. □

Satz 3.28 (Korrektheit). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, $G = \langle V, E \rangle$ der Disabling-Enabling-Graph zu π . Sei s ein Zustand und ν eine aussagenlogische Belegung von $P \cup P' \cup \text{Var}(O) \cup \text{Aux}_4^\pi$ mit $\nu(a) = s(a)$ für alle $a \in P$ und $\nu \models R_4(P, P')$. Dann existieren Operatoren $\{o_1, \dots, o_n\} = O' \subseteq O$ und eine totale Ordnung $<$ auf O' , $o_1 < \dots < o_n$, sodass $\text{app}_{o_1, \dots, o_n}(s)$ definiert ist.

Beweis. Seien $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs zu G mit den gemäß Definition 3.25 vorgegebenen totalen Ordnungen $<_1, \dots, <_k$ auf V_1, \dots, V_k . Nach Lemma 3.27 existiert eine totale Ordnung auf O , sodass für Operatoren $o, o' \in O$ mit $(o, o') \in E$ gilt:

1. Befinden sich o und o' im gleichen SCC $G_i = \langle V_i, E_i \rangle$, so gilt $o < o'$ genau dann, wenn $o <_i o'$.
2. Befinden sich o und o' in verschiedenen SCCs, so ist $o < o'$.

Wir betrachten die von dieser Ordnung auf O' induzierte totale Ordnung $o_1 < \dots < o_n$, für die diese Eigenschaften weiterhin gelten, und zeigen, dass $\text{app}_{o_1, \dots, o_n}(s)$ definiert ist.

Der Beweis erfolgt per vollständiger Induktion über j , $1 \leq j \leq n$.

Induktionsanfang $j = 1$: Wir betrachten den ersten ausgeführten Operator $o_1 = \langle p_1, e_1 \rangle$ und den zugehörigen SCC, o.E. $G_1 = \langle V_1, E_1 \rangle$, mit $o_1 \in V_1$. Es gilt $s \models p_1$, denn angenommen, es existiert ein Literal $m \in p_1$ mit $s \not\models m$. Dann existiert wegen $\nu \models \text{Preconditionaxioms}_2^{G_1}$ ein Operator $o' \in \text{enablingOperators}(o_1; m) \setminus \text{Succ}_{<_1}^{V_1}(o_1)$ mit $\nu \models o'$. Dieser wäre nach Konstruktion von $<$ vor o_1 ausgeführt worden, denn da o_1 von o' aktiviert wird und o_1 und o' wegen Lemma 3.26 verträgliche Vorbedingungen und verträgliche Effekte haben, existiert eine Kante im Disabling-Enabling-Graph von o' zu o_1 . Nach Konstruktion von $<$ gilt dann mit Lemma 3.27

aber in jedem Fall $o' < o_1$. Hieraus ergibt sich ein Widerspruch zur Wahl von o_1 als ersten ausgeführten Operator. Somit ist $\text{app}_{o_1}(s)$ definiert.

Induktionsschritt $j \rightarrow j+1$: Sei $s_j = \text{app}_{o_1, \dots, o_j}(s)$ definiert. Wir zeigen: $\text{app}_{o_{j+1}}(s_j)$ ist definiert. Sei $o_{j+1} = \langle p_{j+1}, e_{j+1} \rangle$ und $m \in p_{j+1}$ beliebig. Dann unterscheidet man zwei Fälle.

1. $s \models m$. Annahme, es existiert ein Operator $o = \langle p, e \rangle$ mit $\bar{m} \in e$ und $o \in \{o_1, \dots, o_j\}$. Wegen Lemma 3.26 haben o und o_{j+1} verträgliche Vorbedingungen und verträgliche Effekte und somit ist $(o_{j+1}, o) \in E$. Weiterhin gilt, dass sich o und o_{j+1} im gleichen SCC befinden, denn angenommen, $o_{j+1} \in V_i$ und $o \in V_j$ für SCCs $G_i = \langle G_i, V_i \rangle$ und $G_j = \langle G_j, V_j \rangle$ mit $i \neq j$, dann wäre wegen $(o_{j+1}, o) \in E$ nach Lemma 3.27 auch $o_{j+1} < o$. Also ist $\{o, o_{j+1}\} \subseteq V_i$ für ein SCC $G_i = \langle V_i, E_i \rangle$. Dann gilt wegen $v \models \text{Parallelismaxioms}_4^{G_i}: v \models (o \rightarrow \neg o_{j+1})$, Widerspruch zur Voraussetzung $v \models o_{j+1}$. Daher kann ein solcher Operator $o = \langle p, e \rangle$ mit $\bar{m} \in e$ und $o \in \{o_1, \dots, o_j\}$ nicht existieren und es gilt $s_j \models m$.
2. $s \models \bar{m}$. Sei $G_i = \langle V_i, E_i \rangle$ der SCC mit $o_{j+1} \in V_i$. Wegen $v \models o_{j+1}$ gilt $v \models \text{enablingOperators}(o_{j+1}; m) \setminus \text{Succ}_{<_i}^{V_i}(o_{j+1})$, d.h. es existiert ein Operator o' mit $v \models o'$, der nach Konstruktion von $<$ vor o_{j+1} ausgeführt wird. Da nur Operatoren mit verträglichen, also insbesondere mit konsistenten Effekten simultan ausgeführt werden, gilt daher $s_j \models m$.

In beiden Fällen gilt somit $s_j \models m$. Da m beliebig gewählt war, gilt auch $s_j \models p_{j+1}$ und somit ist $\text{app}_{o_{j+1}}(s_j)$ definiert. \square

Diese Codierung hat sich in der Praxis in den meisten Planungsproblemen als sehr effizient herausgestellt.

3.6. Zusammenfassung

Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz. Wir haben vier aussagenlogische Codierungen hergeleitet, um 1-Linearisierungs-Pläne zu finden. In diesem Abschnitt werden diese Codierungen mit ihren charakteristischen Eigenschaften noch einmal kurz zusammenfasst.

Sei $O = \{o_1, \dots, o_n\}$ und $P = \{a_1, \dots, a_k\}$. Die Bedingung, dass für einen Zustand s und Operator o jedes Literal aus dessen Vorbedingung, das in s noch nicht wahr ist, durch einen anderen ausgeführten Operator wahr gemacht werden muss, wird bei den Codierungen $R_1(P, P')$, $R_2(P, P')$ und $R_3(P, P')$ durch die Formel $\text{Preconditionaxioms}_1^\pi$ erfüllt. Im Unterschied hierzu wird dies bei $R_4(P, P')$ durch die Formel $\text{Preconditionaxioms}_2^\pi$ gewährleistet, indem eine strengere Bedingung an die Aktivierungsoperatoren in der Menge enablingOperators formuliert wird.

Weiterhin basieren die Codierungen $R_1(P, P')$, $R_2(P, P')$ und $R_3(P, P')$ auf der Azyklichkeit im Disabling-Enabling-Graphen. Die Codierung $R_4(P, P')$ lässt in einem gewissen Maße auch Zyklen zu, wenn trotzdem die Existenz einer Linearisierung von simultan ausgeführten Operatoren garantiert werden kann. Die folgende Tabelle gibt einen Überblick über die Größe des entsprechenden Konjunktionsglieds Parallelismaxioms der Gesamtcodierung (e ist hierbei die Anzahl der Kanten im Disabling-Enabling-Graph und Auxiliaries die Menge der zusätzlich benötigten Hilfsvariablen).

Codierung	$\ Parallelismaxioms\ $	$ Auxiliaries $
$R_1(P, P')$	$O(n^3)$	$O(n^2)$
$R_2(P, P')$	$O(n \cdot k)$	$O(n \cdot k)$
$R_3(P, P')$	$O(n + e)$	$O(n)$
$R_4(P, P')$	$O(n \cdot k)$	$O(n \cdot k)$

Tabelle 3.1.: Vergleich der Implementierungen von *Parallelismaxioms*

In $R_1(P, P')$ verwenden wir den exakten Azyklizitätstest. Die Größe ist kubisch in der Anzahl der Operatoren, darüberhinaus benötigt man quadratisch viele zusätzliche Hilfsvariablen. Daher ist diese Codierung nicht praktikabel.

In $R_2(P, P')$ und $R_4(P, P')$ wird den Operatoren aus den SCCs des Disabling-Enabling-Graphen bereits im Voraus eine feste Reihenfolge zugewiesen und gefordert, dass sie nur bezüglich dieser Ordnung ausgeführt werden dürfen. Dies schränkt zwar die Anzahl der simultan ausführbaren Operatoren ein, lässt aber eine lineare Codierung zu. Es hat sich gezeigt, dass diese Einschränkung in der Praxis nicht sehr gravierend ist.

In $R_3(P, P')$ wird die Azyklizität durch die Betrachtung von Feedback-Vertices garantiert. Obwohl in vielen Fällen mehr Parallelität als bei $R_2(P, P')$ und $R_4(P, P')$ möglich ist, benötigt man nur linear viele zusätzliche Hilfsvariablen in der Anzahl der Operatoren.

3.7. Erweiterungen und Verbesserungen

In diesem Abschnitt geben wir einen kleinen Ausblick auf mögliche Erweiterungen und Verbesserungen der bisher beschriebenen Codierungen. In Abschnitt 3.7.1 werden zwei einfache Heuristiken zur Anordnung der Operatoren in den SCCs des Disabling-Enabling-Graphen beschrieben. Anschließend wird in Abschnitt 3.7.2 eine Möglichkeit aufgezeigt, um auch Operatoren mit nichtkonsistenten Effekten simultan ausführen zu können.

3.7.1. Heuristiken zur Anordnung der Operatoren

Für die Codierungen $R_2(P, P')$ und $R_4(P, P')$ haben wir eine feste, aber beliebige Reihenfolge der Operatoren der SCCs des Disabling-Enabling-Graphen vorgegeben, in denen die Operatoren ausgeführt werden können. Eine naheliegende Idee ist nun, dies durch eine möglichst "sinnvolle" Anordnung zu verbessern und somit mehr Parallelität zuzulassen sowie die Anzahl der Klauseln zur Codierung zu reduzieren.

Eine einfache Möglichkeit besteht darin, dass Operatoren, die viele andere Operatoren deaktivieren, möglichst weit hinten angeordnet werden. Umgekehrt sollen Operatoren, die viele andere aktivieren, möglichst weit vorne stehen. Hierzu definieren wir eine Funktion $h : O \rightarrow \mathbb{Z}$, die jedem Operator $o \in O$ eine ganze Zahl zuweist, anhand der dann die Anordnung definiert wird.

Definition 3.29 (Differenz-Heuristik). Sei O eine Menge von STRIPS-Operatoren.

Für alle $o \in O$ ist

$$index_e(o) = |\{o' \in O \mid o \neq o' \text{ und } o \text{ aktiviert } o'\}|$$

die Anzahl der Operatoren, die durch o aktiviert werden und

$$index_a(o) = |\{o' \in O \mid o \neq o' \text{ und } o \text{ deaktiviert } o'\}|$$

die Anzahl der Operatoren, die durch o deaktiviert werden.

Die Funktion $h : O \rightarrow \mathbb{Z}$ sei dann definiert als

$$h(o) = index_a(o) - index_e(o).$$

Operatoren aus einem SCC werden bezüglich h so angeordnet, dass für je zwei Operatoren o und o' gilt:

$$o < o' \Rightarrow h(o) \leq h(o')$$

Operatoren werden also umso weiter vorne plziert, je mehr andere Operatoren sie aktivieren (und umso weiter hinten, je mehr andere Operatoren sie deaktivieren).

Im Folgenden beschreiben wir abschließend für die Codierung $R_3(P, P')$ noch eine andere einfache Heuristik, um Operatoren so anzuordnen, dass die Anzahl der Feedback-Vertices möglichst gering ist (das Problem des Auffindens einer *Minimal Feedback Vertex Set* ist bereits NP-schwer [GJ79]).

Definition 3.30 (Feedback-Heuristik). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und die Funktion h gemäß Definition 3.29 gegeben. Sei $G = \langle V, E \rangle$ ein SCC des Disabling-Enabling-Graphen zu π . Die Anordnung der Operatoren in V wird induktiv auf folgende Weise konstruiert.

1. Starte mit einem Operator o mit $h(o) \leq h(o')$ für alle $o' \in V$.
2. Sind n Operatoren o_1, \dots, o_n angeordnet ($n \geq 1$), dann wähle als nächsten Operator ein $o' \in V \setminus \{o_1, \dots, o_n\}$ mit $(o, o') \in E$ für ein $o \in \{o_1, \dots, o_n\}$.

Es hat sich gezeigt, dass diese Heuristik in der Praxis schnelle Ergebnisse beim Planen liefert.

3.7.2. Erweiterung der Codierungen auf Operatoren mit komplementären Effekten

Sämtliche bisher in dieser Arbeit beschriebenen Codierungen haben die Eigenschaft, dass nur Operatoren mit verträglichen, also insbesondere konsistenten Effekten simultan ausgeführt werden können. Deshalb ist in einigen Planungsproblemen (z.B. Blocks-World) nur wenig Parallelität möglich. Im Folgenden geben wir eine einfache Idee an, sodass auch Operatoren mit komplementären Effekten parallel ausgeführt werden können.

Definition 3.31 (Nachfolger). Sei $O = \{o_1, \dots, o_n\}$ eine Menge von STRIPS-Operatoren, $<$ eine totale Ordnung auf O mit $o_1 < \dots < o_n$. Sei $o = \langle p, e \rangle \in O$ und m ein Literal. Dann ist die Menge $Successors_{<}^O(o; m) \subset O$ definiert durch

$$Successors_{<}^O(o; m) = \{o^* = \langle p^*, e^* \rangle \mid o^* \in O, o < o^*, m \in p^* \text{ und } \bar{m} \in e^*\},$$

d.h. es sind alle Operatoren enthalten, die in der vorgegebenen Reihenfolge hinter o stehen und m in der Vorbedingung sowie das komplementäre Literal \bar{m} im Effekt haben.

Beispiel 3.10. Sei $O = \{o_1, o_2\}$ eine Menge von Operatoren mit der Ordnung $o_1 < o_2$. Sei $o_1 = \langle \{A\}, \{B\} \rangle$ und $o_2 = \langle \{B\}, \{\neg B\} \rangle$. Dann ist $Successors_{<}^O(o_1; B) = \{o_2\}$ und $Successors_{<}^O(o_2; \neg B) = \emptyset$.

Die Idee für eine Codierung ist nun, dass für bestimmte Operatoren ein Effektliteral m nicht notwendigerweise im nachfolgenden Zustand gilt, sondern alternativ ein anderer Operator (aus der Nachfolger-Menge) ausgeführt werden wird, der \bar{m} als Effekt besitzt.

Definition 3.32 (Effektaxiome mit komplementären Literalen). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz, G der Disabling-Enabling-Graph zu π und $G_1 = \langle V_1, E_1 \rangle, \dots, G_k = \langle V_k, E_k \rangle$ die SCCs von G mit $|V_i| \geq 2$ und den Ordnungen $<_i$ auf V_i für alle $i \in \{1, \dots, k\}$. Für alle SCCs G_i sei

$$Effectaxioms_2^{G_i} = \bigwedge_{\substack{o \in V_i \\ o = \langle p, e \rangle}} (o \rightarrow \bigwedge_{m \in e} (m' \vee \bigvee Successors_{<_i}^{V_i}(o; m))).$$

Hierbei bezeichnet m' das Literal m im nachfolgenden Zustand. Insgesamt erhält man

$$Effectaxioms_2^\pi = \bigwedge_{i=1}^k Effectaxioms_2^{G_i}$$

als Codierung der Effektaxiome, die auch komplementäre Literale von simultan ausgeführten Operatoren zulassen.

Im Folgenden zeigen wir die Korrektheit dieser Codierung, die einen Zustand s in einen Nachfolgezustand s' überführt.

Satz 3.33 (Korrektheit). Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und s ein Zustand. Sei ν eine Belegung von $P \cup P' \cup Var(O)$ mit $\nu \models Effectaxioms_2^\pi$ und $O' = \{o \mid \nu \models o\} = \{o_1, \dots, o_n\}$ die Menge der ausgeführten Operatoren. Sei $<$ die nach Lemma 3.27 existierende Ordnung auf O' mit $o_1 < \dots < o_n$ und sei $s' = app_{o_1, \dots, o_n}(s)$ definiert. Dann gilt für alle $m \in \bar{P}$: Wenn $s' \models m$, dann $\nu \models m'$.

Beweis. Sei $m \in \bar{P}$ und gelte $s' \models m$. Sei $O^m = \{o = \langle p, e \rangle \in O' \mid \{m, \bar{m}\} \cap e \neq \emptyset\} = \{o_1^m, \dots, o_{n_m}^m\}$ die Menge der ausgeführten Operatoren, die m oder \bar{m} im Effekt haben. Sei $<_m$ die von $<$ induzierte Ordnung auf O^m mit $o_1^m <_m \dots <_m o_{n_m}^m$.

Betrachte $o_{n_m}^m = \langle p_n, e_n \rangle$. Es gilt $Successors_{<}^{O'}(o_{n_m}^m; m) = Successors_{<}^{O'}(o_{n_m}^m; \bar{m}) = \emptyset$ (da $o_{n_m}^m$ der bezüglich $<$ letzte ausgeführte Operator ist, der m oder \bar{m} im Effekt hat). Wäre $\bar{m} \in e_n$, dann würde $s' \models \bar{m}$ gelten (im Widerspruch zur Voraussetzung). Somit gilt $m \in e_n$. Sei $G_i = \langle V_i, E_i \rangle$ der SCC des Disabling-Enabling-Graphen zu π mit $o_{n_m}^m \in V_i$. Dann gilt auch $Successors_{<_i}^{V_i}(o_{n_m}^m; m) \cap O' = \emptyset$ und somit wegen $\nu \models (o_{n_m}^m \rightarrow m' \vee \bigvee Successors_{<_i}^{V_i}(o_{n_m}^m; m))$ auch $\nu \models m'$ und damit die Behauptung. \square

Wird das Konjunktionsglied $Effectaxioms_2^\pi$ in unserer Basiscodierung aus Definition 2.17 durch $Effectaxioms_2^\pi$ ersetzt, erhält man somit nochmals eine andere, neue Klasse von parallelen Plänen.

4. Experimente

In diesem Abschnitt wird die Effizienz unserer Codierungen untersucht. Hierzu werden die von Rintanen vorgestellten Auswertungsstrategien verwendet [Rin04a, Rin05].

4.1. Algorithmen zur Plansuche

Seien $\Phi_0, \Phi_1, \Phi_2, \dots$ Codierungen für Planlänge $0, 1, 2, \dots$. Eine einfache Auswertungsstrategie ist, die Erfüllbarkeit dieser Formeln nacheinander zu prüfen und bei der ersten erfüllbaren zu stoppen (Algorithmus S).

```
1: procedure AlgorithmusS()
2:    $i := 0$ ;
3:   repeat
4:     untersuche Erfüllbarkeit von  $\Phi_i$ ;
5:     if  $\Phi_i$  ist erfüllbar then terminate;
6:      $i := i + 1$ ;
7:   until  $1 = 0$ 
```

Abbildung 4.1.: Algorithmus S

Wird nur die Ausführung maximal eines Operators pro Zeitschritt erlaubt, so findet man mit dieser Vorgehensweise einen Plan mit minimaler Anzahl an Operatoren. Es hat sich allerdings gezeigt, dass oft nur einige (wenige) Formeln¹, etwa die im Hinblick auf die Planlänge letzte unerfüllbare, schwierig zu lösen sind, wohingegen viele nachfolgende, erfüllbare Formeln viel einfacher gelöst werden können. Mit dem sequentiellen Algorithmus S müssen diese schwierigen Formeln aber komplett ausgewertet werden, was ihn in vielen Fällen ineffizient machen kann. Sucht man Pläne mit minimaler Länge, so scheint jedoch im Allgemeinen keine bessere Auswertungsstrategie zu existieren. Wird beispielsweise eine Schrittlänge größer als eins gewählt, um nach einer erfüllbaren Formel zu suchen, und anschließend die erste erfüllbare mittels binärer Suche ermittelt, so müssen nach wie vor unerfüllbare Formeln ausgewertet werden. Die Laufzeit der Auswertung unerfüllbarer Formeln wächst jedoch im Allgemeinen exponentiell in ihrer Größe, was dazu führt, dass mit dieser Strategie keine Effizienzsteigerungen im Vergleich zur sequentiellen Auswertung zu erwarten sind.

Sucht man hingegen nicht notwendigerweise Pläne minimaler Länge, so gibt es effizientere Verfahren, die von Rintanen in [Rin04a, Rin05] vorgestellt wurden. Die zugrundeliegende Idee ist hierbei, durch parallele Auswertung mehrerer Formeln durch mehrere Prozesse schneller eine erfüllbare Formel zu finden und damit die potentiell schwieriger zu lösenden unerfüllbaren nicht komplett auswerten zu müssen. Wir veranschaulichen diesen Sachverhalt an folgendem Beispiel.

¹Formeln im Bereich der *Phasenübergänge*

Beispiel 4.1. Sei $\pi = \langle P, I, O, G \rangle$ eine Planungsinstanz und $\Phi_1^\pi, \dots, \Phi_7^\pi$ Formeln zur Beschreibung von möglichen Plänen der Länge 1 bis 7. Typischerweise steigen die Laufzeiten für unerfüllbare Formeln mit wachsender Planlänge exponentiell an, wohingegen die ersten erfüllbaren oft einfacher zu lösen sind. In unserem Fall sei beispielhaft Φ_6 die erste erfüllbare Formel.

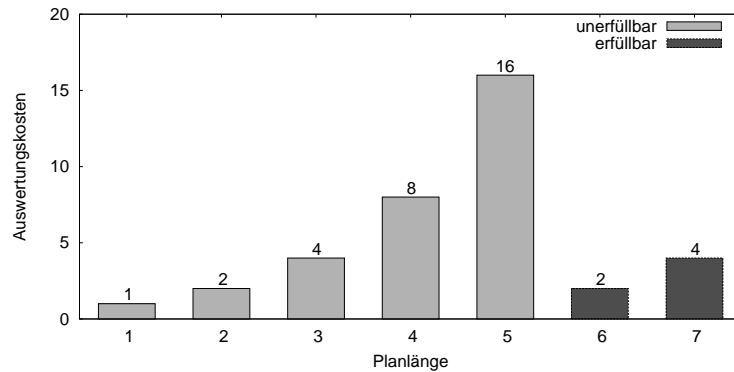


Abbildung 4.2.: Beispielhafte Laufzeiten für Formeln der Planlängen 1 bis 7

Bei einer sequentiellen Auswertung mit Algorithmus S benötigt man in diesem Fall $1s + 2s + 4s + 8s + 16s + 2s = 33s$, um diese zu lösen und damit einen Plan zu finden. Werden hingegen vier parallele Prozesse gestartet und übernimmt ein beendeter Prozess die nächste bisher nicht ausgewertete Formel, so hat Prozess Nummer zwei nach der Auswertung von Φ_2 und Φ_6 bereits nach nur $4s$ die erste erfüllbare Formel gelöst. Die Gesamtzeit der Rechenzeit für alle Prozesse liegt mit $4 \cdot 4s = 16s$ deutlich unter der Zeit von $33s$, die Algorithmus S benötigt.

Im Folgenden beschreiben wir hierzu zwei entsprechende Algorithmen zur Plansuche. Der erste Algorithmus, im Folgenden mit Algorithmus A bezeichnet (Abbildung 4.3), basiert direkt auf der Idee der gleichzeitigen Auswertung von n Formeln durch n Prozesse (im Spezialfall $n = 1$ erhält man Algorithmus S). Stellt sich eine Formel als unerfüllbar heraus, so wird diese verworfen und die nächste noch nicht ausgewertete Formel für die nächstgrößere Planlänge zu den restlichen hinzugenommen. Hierbei ist eine "gute" Wahl für n oft schwierig, da sie im Wesentlichen durch die Anzahl der schwierig zu lösenden unerfüllbaren Formeln sowie deren Laufzeit bestimmt wird. Es hat sich gezeigt, dass eine kleine Änderung von n bereits große Auswirkungen auf die Laufzeit haben kann (beispielsweise benötigt man mit der Codierung $R_4(P, P')$ bei Gripper-5 mit $n = 2$ im Vergleich zu $n = 4$ mehr als die 15-fache Zeit).

Dieses Problem wird mit Algorithmus B (Abbildung 4.4) auf die folgende Weise gelöst. Anstatt mehrere Prozesse zu starten, werden von nur einem Prozess die Formeln nacheinander für eine gewisse Zeit ausgewertet. Die Rechenzeit für eine Formel Φ_k hängt hierbei von ihrem Index k ab. Wurde Φ_k für t Sekunden ausgewertet, so wird die nächste Formel Φ_{k+1} nur noch für $\gamma \cdot t$ Sekunden ausgewertet, wobei $\gamma \in (0, 1)$ ein Parameter des Algorithmus ist. Dies bedeutet anschaulich, dass Formeln mit zunehmender Länge entsprechend weniger Rechenzeit zur Verfügung gestellt wird. Eine gute Wahl für γ ist hierbei weniger kritisch als für n bei Algorithmus A .

```

1: procedure AlgorithmusA( $n$ )
2:  $P = \{\Phi_0, \dots, \Phi_{n-1}\}$ ;
3:  $uneval := n$ ;
4: repeat
5:    $P' := P$ ;
6:   for each  $\Phi \in P'$  do
7:     setze die Auswertung von  $\Phi$  für  $\epsilon$  Sekunden fort;
8:     if  $\Phi$  ist erfüllbar then goto finish;
9:     if  $\Phi$  ist unerfüllbar then
10:       $P := P \cup \{\Phi_{uneval}\} \setminus \{\Phi\}$ ;
11:       $uneval := uneval + 1$ ;
12:     end if
13:   end do
14: until  $1 = 0$ 
15: finish;

```

Abbildung 4.3.: Algorithmus A

```

1: procedure AlgorithmusB( $\gamma$ )
2:  $t := 0$ ;
3: for each  $i \geq 0$  do  $done[i] := false$ ;
4: for each  $i \geq 0$  do  $time[i] := 0$ ;
5: repeat
6:    $t := t + \delta$ ;
7:   for each  $i \geq 0$  mit  $done[i] := false$  do
8:     if  $time[i] + n\epsilon \leq t\gamma^i$  für maximales  $n \geq 1$  then
9:       fahre mit der Auswertung von  $\Phi_i$  für  $n\epsilon$  Sekunden fort;
10:      if  $\Phi_i$  ist erfüllbar then goto finish;
11:       $time[i] := time[i] + n\epsilon$ ;
12:      if  $\Phi_i$  ist unerfüllbar then  $done[i] := true$ ; end if
13:    end if
14:  end do
15: until  $1 = 0$ 
16: finish;

```

Abbildung 4.4.: Algorithmus B

4.2. Aufbau

Die Implementierung der Codierungen erfolgt in der Programmiersprache SML [Pau91] und erweitert den SML-Planer von Rintanen. Die Erfüllbarkeitstests führen wir auf einem Rechner mit 3.6 GHz Intel Xeon Prozessor und dem SAT-Solver SIEGE (Version 4) von Lawrence Ryan [Rya04] durch. Da die Laufzeiten von SIEGE aufgrund der Randomisierung variieren können, wird jeweils die durchschnittliche Laufzeit nach 60 Durchläufen ermittelt (wobei die erwartete Laufzeit auf ∞ gesetzt wurde, wenn die Laufzeit von SIEGE bei mindestens einem Durchlauf über einer Stunde lag). Mit diesen Erwartungswerten simulieren wir anschließend die im vorigen Abschnitt vorgestellten Algorithmen A und B für verschiedene $n \in \{1, 2, 4, 8, 16\}$ und $\gamma \in (0, 1)$.

4.2.1. Planungsdomänen

Die Codierungen werden anhand von STRIPS-Planungsdomänen der Planungswettbewerbe (*AI Planning Systems Competition*) der Jahre 1998, 2000 und 2002 getestet. Es handelt sich hierbei meist um vereinfachte Probleme, wie sie in ähnlicher Form in der wirklichen Welt (beispielsweise als Transportprobleme) auftreten können.

In der *Logistik*-Domäne (seit 1998) müssen Objekte mittels Lastwagen und Flugzeugen an einen gegebenen Zielort transportiert werden. In der *Blocks-World*-Domäne (2000) besteht die Aufgabe darin, Objekte (*Blöcke*) in einer vorgegebenen Ordnung aufeinander zu stapeln. Die Aufgaben dieser beiden Domänen werden in *Depots* (2002) kombiniert. Hier müssen Kisten zunächst transportiert und anschließend gestapelt werden. In *Gripper* (1998) muss ein Roboter Bälle von einem Raum zu einem anderen transportieren, *Freecell* (2002) beschreibt das gleichnamige Kartenspiel. In der *Satellite*-Domäne (2002) müssen im Rahmen einer Weltraumumgebung Beobachtungsaufgaben mit unterschiedlich ausgestatteten Satelliten gelöst werden, bei *Rover* (2002) müssen auf der Oberfläche eines Planeten Proben genommen und deren Ergebnisse übermittelt werden.

Weiterhin entwickeln wir exemplarisch eine neue Planungsdomäne *Boxes*, um die Möglichkeiten unserer Codierungen zu demonstrieren. In dieser Domäne existiert ein Lastenkrane, der je drei Objekte A, B und C in dieser Reihenfolge in Schachteln verpacken muss. Der leere Kran kann jeweils mit dem Operator `takeSet` einen Satz der drei Objekte hochheben und diese daraufhin einzeln mit den Operatoren `putA(b)`, `putB(b)` und `putC(b)` in eine Schachtel b herunterlassen. Hierbei ist es von Bedeutung, dass sich zum Schluss Objekt A unten, Objekt B mittig und Objekt C oben in der Schachtel befindet. Die Reihenfolge, in der die Objekte verpackt werden, spielt also eine wichtige Rolle. Eine komplette Beschreibung dieser Planungsdomäne in PDDL (*Planning Domain Definition Language*, [McD98]) findet sich in Anhang A.

4.2.2. Implementierung der Heuristiken zur Anordnung der Operatoren

Sowohl für die Codierung $R_3(P, P')$ als auch $R_4(P, P')$ wurde mit verschiedenen Heuristiken zur Anordnung der Operatoren in den SCCs des Disabling-Enabling-Graphen experimentiert. Hierbei hat sich herausgestellt, dass nicht notwendigerweise die Anordnung, die die kleinsten Formeln erzeugt, auch zum schnellsten Planer führt. Weiterhin hat sich gezeigt, dass sich mit unserer Differenz-Heuristik nach Definition 3.29 aus Symmetriegründen meist nur wenige ver-

schiedene Indexwerte der Operatoren ergeben und sie daher nur in wenigen Fällen zu einer Effizienzsteigerung führt.

In der Codierung $R_3(P, P')$ wird zur Anordnung der Operatoren in den SCCs die Greedy-Heuristik aus Definition 3.30 verwendet, um die Anzahl der Feedback-Vertices möglichst gering zu halten. Für die Codierung $R_4(P, P')$ ändern wir die ursprüngliche (vom SML-Planer vorgegebene) Reihenfolge der Operatoren in den SCCs nur geringfügig: Es werden höchstens Operatoren, die ohnehin von keinem anderen Operator aktiviert werden, nach vorne platziert. Es hat sich gezeigt, dass dies in vielen Fällen sehr effizient ist.

4.3. Ergebnisse

Im Folgenden vergleichen wir die Laufzeiten der Codierungen $R_3(P, P')$ und $R_4(P, P')$ mit der bisherigen Codierung $R^{lin}(P, P')$ aus [RHN04] mit dem sequentiellen Algorithmus S sowie den neuen Algorithmen A und B (die gesamten Testergebnisse finden sich in Anhang B). Um eine konservative Abschätzung zu erhalten, wählen wir als Parameter für die neuen Algorithmen $n = 2$ für Algorithmus A (wobei n aus $\{2, 4, 8, 16\}$ gewählt wird, da Algorithmus S bereits Algorithmus $A(1)$ entspricht) und $\gamma = 0.5$ für Algorithmus B . Für diese gilt, dass sie bei der Mehrzahl der getesteten Instanzen im Vergleich zu anderen Parametern die schnellsten Laufzeiten für die bisher existierende Codierung $R^{lin}(P, P')$ erzielen (vgl. Anhang B). In den Tabellen 4.1, 4.2 und 4.3 werden hierzu die ermittelten Laufzeiten zum Auffinden eines Plans mit den Algorithmen S , $A(2)$ und $B(0.5)$ verglichen (die Angaben beziehen sich hierbei auf die reine Laufzeit des SAT-Solvers). Tabelle 4.4 zeigt die entsprechenden Längen der gefundenen Pläne, Tabelle 4.5 vergleicht die Anzahl und Größe der SCCs im zugehörigen Disabling-Enabling-Graphen. Ein \times kennzeichnet hierbei Instanzen, für die keine obere Schranke der Laufzeit angegeben werden kann, da SIEGE bei mindestens einem Durchlauf der entsprechenden Formeln mehr als 60 Minuten benötigte.

Man stellt zunächst fest, dass die Größe der SCCs im Disabling-Enabling-Graph im Vergleich zum reinen Disabling Graph meist anwächst, was eine direkte Folge der verallgemeinerten Definition ist. In vielen Fällen führt dies jedoch dazu, dass eine erfüllende Belegung der entsprechenden Formel aufgrund der zusätzlichen Konjunktionsglieder schneller gefunden werden kann.

Bei Betrachtung der Laufzeiten erkennt man, dass mit der konventionellen sequentiellen Auswertung mit Algorithmus S die Ergebnisse recht unterschiedlich sind (Tabelle 4.1). Während in den Domänen *Logistik* und *Freecell* keine grundsätzlichen Effizienzsteigerungen erkennbar sind, wird bei *Depots*, *Gripper* und *Boxes* das Ziel meist schneller erreicht. *Blocks-World* liefert vergleichbare Ergebnisse.

Bei der Auswertung mit den Algorithmen $A(2)$ und $B(0.5)$ erhält man in vielen Fällen eine deutlichere Effizienzsteigerung (Tabelle 4.2 und 4.3). Obwohl in fast allen getesteten Planungsdomänen (außer *Boxes*) die gefundenen Pläne mit den Codierungen $R_3(P, P')$ und $R_4(P, P')$ aufgrund inkonsistenter Effekte der Operatoren nicht oder nur unwesentlich kürzer als mit $R^{lin}(P, P')$ sind, erhält man oftmals schnellere Ergebnisse, da die Formeln aufgrund der zusätzlichen Konjunktionsglieder für den SAT-Solver einfacher zu lösen sind. Beim Vergleich mit Algorithmus $A(2)$ erhält man in den Domänen *Logistik*, *Depots*, *Gripper*, *Satellite* und *Boxes* überwiegend schnellere Laufzeiten mit einem Speedup von teilweise über 100%. Ähnlich sieht es beim Ver-

gleich mit Algorithmus $B(0.5)$ aus. In der Domäne *Boxes* erhält man darüber hinaus im Wesentlichen nur noch halb so lange Pläne wie bisher. Da die Operatoren `putB` bzw. `putC` erst von `putA` bzw. `putB` aktiviert werden, existiert kein Zustand, in dem mindestens zwei verschiedene Vorbedingungen von Operatoren gleichzeitig erfüllt sind. Mit $R^{lin}(P, P')$ erhält man folglich Pläne, die pro Zeitpunkt nur einen Operator enthalten. Mit den neuen Codierungen $R_3(P, P')$ und $R_4(P, P')$ können hingegen die drei `put`-Operatoren simultan ausgeführt werden. Hieraus resultiert eine Effizienzsteigerung von einem Faktor der Größenordnung 10^3 .

	$R^{lin}(P, P')$	$R_3(P, P')$	$R_4(P, P')$
Logistik 22	1.4	1.9	1.5
Logistik 23	3.3	2.7	5.9
Logistik 24	1.8	2.5	1.9
Blocks 18	7.1	7.2	7.1
Blocks 20	10.9	10.9	11.2
Blocks 22	140.4	135.8	140.3
Depots 9	8.1	7.3	8.8
Depots 12	111.6	27.1	62.9
Depots 18	2.7	3.0	1.0
Gripper 3	1.0	0.4	0.8
Gripper 4	38.5	6.7	60.7
Gripper 5	×	389.9	×
Freecell 3	0.4	0.3	0.2
Freecell 5	48.8	102.3	57.2
Satellite 10	1.3	0.3	0.6
Satellite 16	6.5	16.3	8.2
Rover 19	0.9	0.9	0.5
Rover 20	2.7	3.9	1.9
Boxes 8	×	1.2	1.6
Boxes 10	×	27.9	37.9

Tabelle 4.1.: Laufzeiten in s von Algorithmus S

	$R^{lin}(P, P')$	$R_3(P, P')$	$R_4(P, P')$
Logistik 22	2.3	1.9	1.7
Logistik 23	3.9	2.1	2.8
Logistik 24	2.8	1.2	2.5
Blocks 18	6.2	6.2	5.7
Blocks 20	12.5	12.5	12.8
Blocks 22	131.6	124.6	130.5
Depots 9	12.9	10.4	13.2
Depots 12	222.9	53.8	125.5
Depots 18	5.1	5.7	1.9
Gripper 3	0.8	0.4	0.8
Gripper 4	11.5	5.8	14.6
Gripper 5	934.3	193.3	921.3
Freecell 3	0.2	0.4	0.2
Freecell 5	77.8	89.3	92.5
Satellite 10	1.8	0.4	0.4
Satellite 16	5.6	10.7	5.0
Rover 19	0.6	1.1	0.6
Rover 20	2.3	4.9	2.3
Boxes 8	2247	1.2	1.2
Boxes 10	×	25.8	30.4

Tabelle 4.2.: Laufzeiten in s von Algorithmus $A(2)$

	$R^{lin}(P, P')$	$R_3(P, P')$	$R_4(P, P')$
Logistik 22	2.6	1.8	2.4
Logistik 23	6.0	2.2	2.1
Logistik 24	3.1	1.5	1.8
Blocks 18	6.3	7.0	5.3
Blocks 20	13.4	13.5	13.7
Blocks 22	139.9	130.9	140.7
Depots 9	13.3	11.2	14.6
Depots 12	223.0	53.8	125.5
Depots 18	5.2	5.8	1.9
Gripper 3	0.7	0.2	0.2
Gripper 4	5.7	2.9	1.4
Gripper 5	74.0	25.2	12.2
Freecell 3	0.2	0.5	0.3
Freecell 5	85.6	142.9	103.0
Satellite 10	1.6	0.5	0.5
Satellite 16	9.3	21.2	6.8
Rover 19	0.8	1.3	0.7
Rover 20	3.4	7.2	3.0
Boxes 8	954	0.7	1.2
Boxes 10	×	10.9	9.1

Tabelle 4.3.: Laufzeiten in s von Algorithmus $B(0.5)$

	$R^{lin}(P, P')$	$R_3(P, P')$	$R_4(P, P')$
Logistik 22	9 / 9 / 9	8 / 9 / 10	9 / 10 / 10
Logistik 23	9 / 10 / 9	8 / 9 / 10	9 / 10 / 11
Logistik 24	9 / 9 / 9	8 / 9 / 10	9 / 10 / 11
Blocks 18	58 / 58 / 58	58 / 58 / 58	58 / 58 / 58
Blocks 20	60 / 60 / 60	60 / 60 / 60	60 / 60 / 60
Blocks 22	72 / 72 / 72	72 / 72 / 72	72 / 72 / 72
Depots 9	20 / 20 / 20	20 / 20 / 20	20 / 20 / 20
Depots 12	20 / 20 / 20	20 / 20 / 20	20 / 20 / 20
Depots 18	12 / 12 / 12	12 / 12 / 12	12 / 12 / 12
Gripper 3	8 / 8 / 11	8 / 8 / 9	8 / 9 / 9
Gripper 4	10 / 10 / 10	10 / 10 / 14	10 / 10 / 10
Gripper 5	12 / 12 / 13	12 / 12 / 12	12 / 12 / 13
Freecell 3	8 / 8 / 9	7 / 7 / 7	7 / 7 / 7
Freecell 5	13 / 13 / 13	12 / 12 / 12	12 / 12 / 12
Satellite 10	5 / 5 / 9	4 / 4 / 4	5 / 5 / 6
Satellite 16	4 / 4 / 4	4 / 4 / 4	4 / 5 / 6
Rover 19	8 / 8 / 8	6 / 6 / 6	7 / 7 / 7
Rover 20	8 / 8 / 8	6 / 7 / 6	7 / 7 / 7
Boxes 8	× / 32 / 35	16 / 16 / 17	16 / 16 / 18
Boxes 10	× / × / ×	20 / 20 / 20	20 / 20 / 20

Tabelle 4.4.: Planlängen mit Algorithmus $S / A(2) / B(0.5)$

	Disabling Graph	Disabling-Enabling-Graph
Logistik 22	1664 × 1	8 × 98, 2 × 440
Blocks 18	648 × 1	648 × 1
Depots 9	1812 × 1	1620 × 1, 2 × 96
Gripper 3	66 × 1	1 × 66
Freecell 3-4	37 × 1, 1 × 1107	22 × 1, 1 × 1122
Satellite 16	7194 × 1	46 × 1, 2 × 626, 1 × 676, 2 × 703, 1 × 727, 1 × 728, 2 × 778, 1 × 803
Rover 19	8 × 3, 1502 × 1, 16 × 21, 9 × 12, 2 × 24, 17 × 2, 7 × 6, 2 × 9, 2 × 15, 2 × 18, 24 × 14, 3 × 10, 11 × 8, 9 × 4, 3 × 16, 16 × 7, 2 × 5	1 × 629, 1 × 479, 1 × 484, 1 × 587, 1 × 452, 1 × 203, 4 × 1
Boxes 8	1 × 1, 3 × 8	1 × 1, 3 × 8

Tabelle 4.5.: Anzahl und Größe der SCCs: $n \times m$ entspricht n SCCs mit m Operatoren

4.4. Fazit

Zusammenfassend erhält man mit den Codierungen $R_3(P, P')$ und $R_4(P, P')$ bei Auswertung mit den neuen Algorithmen A und B im Vergleich zu der bisherigen 1-Linearisierungs-Codierung $R^{lin}(P, P')$ in vielen Fällen aus zwei Gründen deutliche Effizienzsteigerungen beim Planen. Zum einen wird aufgrund der größeren SCCs im Disabling-Enabling-Graph die Formel zum Auffinden eines Plans durch zusätzliche Konjunktionsglieder oftmals leichter lösbar, obwohl die Planlängen nicht signifikant kürzer sind. Insbesondere mit $R_4(P, P')$ erhält man bei der Auswertung mit Algorithmus $B(0.5)$ Effizienzsteigerungen von teilweise über 100%. Abbildung 4.5 veranschaulicht diesen Sachverhalt noch einmal anhand der jeweils größten getesteten Instanz der verschiedenen Planungsdomänen (aufgrund der großen Laufzeitunterschiede ist die Zeit-Achse logarithmisch skaliert): Man erhält meist deutliche Effizienzsteigerungen in den Domänen *Logistik*, *Depots* und *Gripper* sowie leichte Effizienzsteigerungen in den Domänen *Satellite* und *Rover*. In der *Blocks-World*-Domäne erhält man vergleichbare Ergebnisse, einzig in der *Freecell*-Domäne ist $R_4(P, P')$ weniger effizient als $R^{lin}(P, P')$.

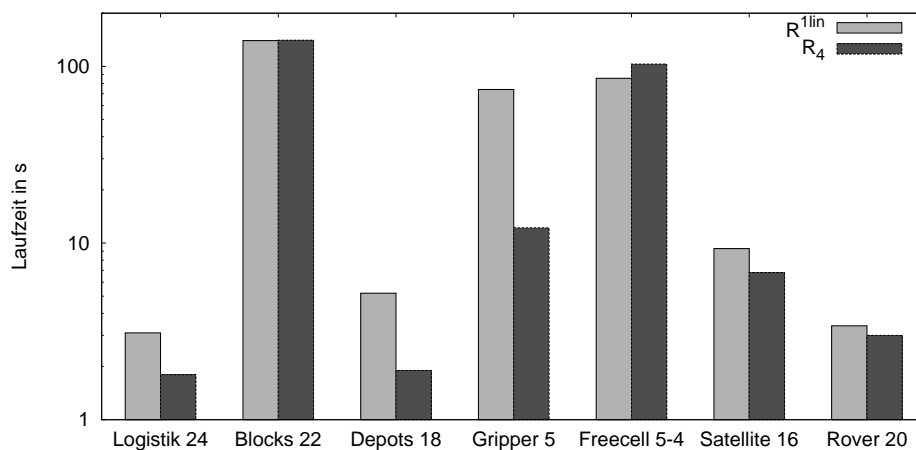


Abbildung 4.5.: Laufzeiten von $R^{lin}(P, P')$ und $R_4(P, P')$ mit Algorithmus $B(0.5)$

In Planungsdomänen, die die Tatsache explizit ausnutzen, dass Operatoren auch dann simultan in einem Zustand s ausgeführt werden dürfen, obwohl noch nicht alle Vorbedingungen in s bereits erfüllt sind, erhält man durch die kürzeren Pläne darüberhinaus eine noch größere Effizienzsteigerung. Dies haben wir exemplarisch an der Domäne *Boxes* gesehen, in der man annähernd eine Beschleunigung um den Faktor 10^3 erhält.

5. Zusammenfassung und Ausblick

In dieser Arbeit wurde eine neue Klasse von parallelen Plänen sowie vier korrespondierende aussagenlogische Codierungen zur erfüllbarkeitsbasierten Handlungsplanung vorgestellt. Ein zentrales Konzept hierzu ist der Disabling-Enabling-Graph, der eine Erweiterung des Disabling Graphen aus [RHN04] darstellt. Im Gegensatz zum reinen Disabling Graph enthält der Disabling-Enabling-Graph zusätzliche Kanten, die der Tatsache Rechnung tragen, dass mehr Operatoren als bisher parallel ausführbar sind. Um die Korrektheit der Codierungen zu garantieren, musste im Wesentlichen die Azyklizität dieses Graphen gesichert werden.

Unsere erste Codierung $R_1(P, P')$ lässt hinsichtlich der betrachteten Semantik maximal viele parallele Operatoren zu, da sie auf einem exakten Azyklizitätstest beruht. Hierzu werden allerdings quadratisch viele zusätzliche Hilfsvariablen benötigt, ihre Gesamtgröße ist kubisch in der Anzahl der Operatoren. In der Praxis ist sie deshalb außer in sehr kleinen Probleminstanzen mit wenigen Operatoren wohl nicht effizient.

Um die Größe der Codierung zu reduzieren, wird in $R_2(P, P')$ statt des exakten Azyklizitätstests eine hinreichende Bedingung formuliert, die die Ausführung von Operatoren nur bezüglich einer fest vorgegebenen Reihenfolge zulässt. Dies schränkt zwar die Anzahl der parallel ausführbaren Operatoren ein, lässt aber im Gegenzug eine lineare Codierung zu. Es hat sich gezeigt, dass diese Einschränkung der Ausführung auf eine festen Reihenfolge in der Praxis nicht sehr gravierend ist.

Die dritte Codierung $R_3(P, P')$, die ebenfalls auf der Azyklizität im Disabling-Enabling-Graph beruht, stellt einen Kompromiss zwischen den beiden ersten dar. Es werden hierzu Operatoren identifiziert, ohne die der Graph garantiert zyklfrei ist (*Feedback-Vertices*), und anschließend gefordert, dass kein Operator, der einen möglichen Zykel vervollständigt, von solch einem Feedback-Vertex aus erreichbar sein darf. Ihre Größe ist zwar linear in der Anzahl der Kanten im Graph und damit im schlechtesten Fall quadratisch in der Anzahl der Operatoren, dafür ist aber mehr Parallelität als mit $R_2(P, P')$ möglich. Weiterhin benötigt man nur linear viele Hilfsvariablen in der Anzahl der Operatoren.

Unsere vierte Codierung $R_4(P, P')$ stellt eine naheliegende Verbesserung von $R_2(P, P')$ dar. Zwar wird wieder eine feste Ausführungsreihenfolge gewisser Operatoren vorgegeben, aber es werden in gewissem Maße auch Zykel im Graph zugelassen, wenn immer noch garantiert ist, dass die simultan ausgeführten Operatoren linearisierbar sind. Dies führt zu einer in der Praxis kleineren und damit effizienteren Codierung.

Als Ergebnisse der Experimente erhält man mit den neuen Codierungen in vielen bekannten Planungsdomänen, so z.B. in der Logistik- oder Gripper-Domäne, teilweise Effizienzsteigerungen von über 100% zum Lösen der entsprechenden Formeln und damit zum Finden eines Plans. In der Planungsdomäne *Boxes*, die explizit ausnutzt, dass Operatoren auch noch von simultan ausgeführten anderen Operatoren aktiviert werden dürfen, erhält man darüberhinaus eine Effizienzsteigerung von einem Faktor der Größenordnung 10^3 .

In einem kurzen Ausblick wurde ferner die Möglichkeit skizziert, um auch Operatoren mit komplementären Effekten zuzulassen. Dies führt zu einer anderen, ebenfalls neuen Klasse paralleler Pläne, deren Effizienz jedoch nicht getestet wurde.

Für zukünftige Arbeiten stellt sich die Frage, ob durch noch bessere Heuristiken zur Anordnung der Operatoren weitere Effizienzsteigerungen beim Planen möglich sind. Wie bereits in Abschnitt 4 beschrieben, führt nicht notwendigerweise die Anordnung, die die kleinsten Formeln erzeugt, auch notwendigerweise zum schnellsten Planer. In diesem Zusammenhang könnten weitere Kriterien, die zu effizienten Heuristiken führen, erarbeitet werden.

Ferner verbleibt die Frage, ob noch weitere interessante Klassen von effizienten parallelen Plänen existieren. Hierzu wurde in dieser Arbeit bereits eine kurze Möglichkeit skizziert, mit der auch Operatoren mit komplementären Effekten simultan ausgeführt werden können. Es verbleibt aber die Frage nach ihrer Effizienz sowie nach weiteren möglichen Klassen.

Abschließend könnten die Codierungen auf Operatoren mit bedingten Effekten erweitert werden. Eine effiziente Erweiterung auf beliebige Operatoren scheint aufgrund möglicher disjunktiver Vorbedingungen jedoch nicht möglich zu sein.

A. Die Planungsdomäne *Boxes*

```
(define (domain BOXES)
  (:requirements :strips :typing)
  (:types box)

  (:predicates
    (Ain ?b - box)
    (Bin ?b - box)
    (Cin ?b - box)
    (empty ?b - box)
    (full ?b - box)
    (holdingA)
    (holdingB)
    (holdingC)
    (hoistempty)
  )

  (:action TAKESET
    :precondition (hoistempty)
    :effect (and (not (hoistempty)) (holdingA) (holdingB) (holdingC))
  )

  (:action PUTA
    :parameters (?b - box)
    :precondition (and (empty ?b) (not (Ain ?b)) (holdingA))
    :effect (and (not (empty ?b)) (not (holdingA)) (Ain ?b))
  )

  (:action PUTB
    :parameters (?b - box)
    :precondition (and (not (Bin ?b)) (Ain ?b) (holdingB))
    :effect (and (not (holdingB)) (Bin ?b))
  )

  (:action PUTC
    :parameters (?b - box)
    :precondition (and (not (Cin ?b)) (Bin ?b) (holdingC))
    :effect (and (not (holdingC)) (Cin ?b) (full ?b) (hoistempty))
  ))
```

B. Laufzeiten

Im Folgenden werden die Laufzeiten der Algorithmen A und B für verschiedene Parameter n und γ aufgezeigt. Hierbei bedeutet \times eine fehlende obere Schranke für die Laufzeit, da bei mindestens einem Durchlauf des SAT-Solvers mehr als 60 Minuten benötigt wurden.

B.1. Logistik

Logistik 22

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	1.4	2.3	3.1	5.7	6.8	2.6	4.8	8.3	11.5
$R_2(P, P')$	4.7	5.0	1.1	1.7	3.2	3.1	2.0	2.5	4.0
$R_3(P, P')$	1.9	1.9	0.8	1.4	2.7	1.8	1.7	2.1	3.3
$R_4(P, P')$	1.5	1.7	1.2	1.3	2.3	2.4	1.8	2.2	3.2

Logistik 23

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	3.3	3.9	3.4	5.9	4.4	6.0	6.4	6.2	7.5
$R_2(P, P')$	15.6	8.3	1.6	2.2	3.0	4.9	3.0	3.7	5.2
$R_3(P, P')$	2.7	2.1	1.0	2.0	2.9	2.2	2.0	2.8	4.7
$R_4(P, P')$	5.9	2.8	0.8	1.2	2.1	2.1	1.8	1.9	2.9

Logistik 24

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	1.8	2.8	4.9	5.2	5.2	3.1	5.9	10.1	10.1
$R_2(P, P')$	3.1	1.4	1.8	1.4	2.7	2.5	3.3	3.1	4.1
$R_3(P, P')$	2.5	1.2	0.7	1.3	2.4	1.5	1.3	1.8	3.0
$R_4(P, P')$	1.9	2.5	0.8	1.0	1.8	1.8	1.6	2.0	2.7

B.2. Blocks-World

Blocks 18

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	7.1	6.2	5.1	4.5	5.3	6.3	5.8	5.8	7.3
$R_2(P, P')$	7.1	5.9	5.5	4.5	3.9	6.6	6.0	5.0	5.4
$R_3(P, P')$	7.2	6.2	5.9	4.6	3.9	7.0	6.0	5.0	5.4
$R_4(P, P')$	7.1	5.7	4.2	4.3	4.1	5.3	4.9	5.1	5.6

Blocks 20

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	10.9	12.5	15.7	20.2	17.0	13.4	18.3	21.1	18.2
$R_2(P, P')$	11.0	12.6	15.8	21.2	18.2	13.6	18.6	25.9	24.5
$R_3(P, P')$	10.9	12.5	16.0	19.7	17.6	13.5	18.6	25.0	24.6
$R_4(P, P')$	11.2	12.8	15.8	17.6	17.6	13.7	18.6	21.2	18.1

Blocks 22

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	140.4	131.6	122.8	123.4	86.7	139.9	152.0	165.5	167.9
$R_2(P, P')$	137.6	127.6	117.5	120.7	86.5	136.7	148.5	162.8	166.8
$R_3(P, P')$	135.8	124.6	112.5	109.9	93.5	130.9	138.9	171.6	181.3
$R_4(P, P')$	140.3	130.5	121.8	127.3	91.6	140.7	155.0	171.7	176.1

B.3. Depots

Depots 9

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	8.1	12.9	22.1	42.3	84.4	13.3	23.9	44.9	87.1
$R_2(P, P')$	11.1	18.3	32.3	60.4	120.5	18.7	33.7	63.8	124.0
$R_3(P, P')$	7.3	10.4	17.0	31.7	63.1	11.2	19.1	34.8	66.3
$R_4(P, P')$	8.8	13.2	24.0	46.7	93.1	14.6	26.2	49.4	95.9

Depots 12

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	111.6	222.9	445.6	891.0	1781.9	223.0	445.7	891.1	1782.0
$R_2(P, P')$	101.8	203.3	406.2	812.3	1624.5	203.3	406.3	812.4	1624.6
$R_3(P, P')$	27.1	53.8	107.2	214.3	428.4	53.8	107.4	214.4	428.5
$R_4(P, P')$	62.9	125.5	250.6	501.1	1002.1	125.5	250.7	501.2	1002.2

Depots 18

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	2.7	5.1	9.9	19.7	39.2	5.2	10.0	19.8	39.4
$R_2(P, P')$	6.5	12.8	25.4	50.7	101.2	12.9	25.5	50.8	101.4
$R_3(P, P')$	3.0	5.7	11.2	22.4	44.6	5.8	11.3	22.5	44.8
$R_4(P, P')$	1.0	1.9	3.5	6.9	13.7	1.9	3.6	7.0	13.9

B.4. Gripper

Gripper 3

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	1.0	0.8	1.2	0.2	0.3	0.7	0.4	0.3	0.4
$R_2(P, P')$	1.4	0.3	0.1	0.2	0.2	0.2	0.2	0.2	0.3
$R_3(P, P')$	0.4	0.4	0.1	0.2	0.2	0.2	0.1	0.2	0.2
$R_4(P, P')$	0.8	0.8	0.1	0.2	0.2	0.2	0.2	0.2	0.3

Gripper 4

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	38.5	11.5	2.8	1.1	0.4	5.7	1.8	0.7	0.7
$R_2(P, P')$	55.2	11.5	0.7	0.2	0.3	0.5	0.3	0.3	0.5
$R_3(P, P')$	6.7	5.8	2.1	1.5	0.3	2.9	0.8	0.5	0.6
$R_4(P, P')$	60.7	14.6	0.9	0.2	0.2	1.4	0.4	0.3	0.3

Gripper 5

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	×	934.3	51.2	11.3	7.0	74.0	20.8	14.3	14.0
$R_2(P, P')$	×	720.1	80.4	16.9	8.1	79.4	37.0	19.9	17.5
$R_3(P, P')$	389.9	193.3	25.7	9.2	6.8	25.2	21.4	13.5	13.5
$R_4(P, P')$	×	921.3	60.9	1.6	1.5	12.2	3.8	2.8	2.8

B.5. Freecell

Freecell 3

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	0.4	0.2	0.2	0.2	0.3	0.2	0.2	0.3	0.4
$R_2(P, P')$	0.2	0.3	0.5	0.5	0.8	0.4	0.6	0.7	1.1
$R_3(P, P')$	0.3	0.4	0.6	1.1	2.1	0.5	0.8	1.4	2.6
$R_4(P, P')$	0.2	0.2	0.2	0.3	0.4	0.3	0.3	0.3	0.5

Freecell 5

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	48.8	77.8	122.9	174.1	347.4	85.6	159.4	306.9	461.4
$R_2(P, P')$	60.0	104.7	107.8	155.7	306.6	111.4	193.1	241.0	386.4
$R_3(P, P')$	102.3	89.3	119.7	181.7	325.9	142.9	223.1	319.4	486.2
$R_4(P, P')$	57.2	92.5	125.9	131.9	257.6	103.0	194.6	257.6	390.6

B.6. Satellite

Satellite 10

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	1.3	1.8	3.1	0.3	0.4	1.6	0.6	0.5	0.6
$R_2(P, P')$	0.7	0.3	0.6	1.0	2.0	0.6	0.7	1.2	2.3
$R_3(P, P')$	0.3	0.4	0.8	1.5	3.0	0.5	0.9	1.7	3.2
$R_4(P, P')$	0.6	0.4	0.3	0.4	0.9	0.5	0.4	0.6	1.0

Satellite 16

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	6.5	5.6	11.0	6.6	3.4	9.3	11.1	6.5	7.5
$R_2(P, P')$	12.1	4.7	5.9	9.8	19.5	8.3	10.7	14.9	24.1
$R_3(P, P')$	16.3	10.7	10.4	17.1	31.2	21.2	23.4	30.6	47.4
$R_4(P, P')$	8.2	5.0	3.3	6.3	12.6	6.8	6.1	8.8	15.0

B.7. Rover

Rover 19

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	0.9	0.6	0.6	1.2	2.3	0.8	0.9	1.5	2.7
$R_2(P, P')$	1.5	2.1	3.4	6.5	12.9	2.4	4.3	8.0	14.4
$R_3(P, P')$	0.9	1.1	1.8	3.6	7.2	1.3	2.2	4.0	7.6
$R_4(P, P')$	0.5	0.6	0.8	1.5	3.1	0.7	1.1	1.9	3.4

Rover 20

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	2.7	2.3	2.3	3.9	7.7	3.4	4.0	5.6	9.3
$R_2(P, P')$	4.2	6.2	9.3	15.0	29.7	7.1	12.7	23.2	37.5
$R_3(P, P')$	3.9	4.9	6.7	13.0	25.9	7.2	10.8	17.5	30.1
$R_4(P, P')$	1.9	2.3	2.6	4.4	8.6	3.0	4.7	6.5	10.5

B.8. Boxes

Boxes 8

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	×	2247.2	1305.2	624.1	70.4	954.1	164.7	44.1	26.7
$R_2(P, P')$	1.9	1.4	1.2	0.6	0.8	1.5	1.0	0.9	1.1
$R_3(P, P')$	1.2	1.2	0.6	0.3	0.5	0.7	0.5	0.4	0.6
$R_4(P, P')$	1.6	1.2	1.0	0.4	0.5	1.2	0.6	0.6	0.7

Boxes 10

Codierung	Algorithmus A mit n					Algorithmus B mit γ			
	1	2	4	8	16	0.5000	0.7500	0.8750	0.9375
$R^{lin}(P, P')$	×	×	×	×	2419.3	×	2309.4	332.7	124.4
$R_2(P, P')$	28.4	25.4	11.9	2.9	3.5	8.4	4.9	4.5	5.6
$R_3(P, P')$	27.9	25.8	11.5	3.9	3.4	10.9	5.8	3.4	2.4
$R_4(P, P')$	37.9	30.4	14.4	2.8	3.2	9.1	5.0	4.6	5.4

Abbildungsverzeichnis

2.1. Beispiel für einen gerichteten Graph	12
2.2. Beispiel für einen induzierten Teilgraph	12
2.3. Ein Transitionssystem	13
2.4. Beispiel für einen zyklischen Disabling Graph	22
2.5. Beispiel für einen azyklischen Disabling Graph	22
3.1. Beispiel für einen zyklischen Disabling-Enabling-Graph	29
3.2. Beispiel für einen azyklischen Disabling Graph	29
3.3. Ein azyklischer gerichteter Graph	37
3.4. Ein zyklischer gerichteter Graph	37
3.5. Beispiel für einen zyklischen Disabling-Enabling-Graph	38
3.6. Beispiel für einen zyklischen Disabling-Enabling-Graph	41
4.1. Algorithmus S	48
4.2. Beispielhafte Laufzeiten für Formeln der Planlängen 1 bis 7	49
4.3. Algorithmus A	50
4.4. Algorithmus B	50
4.5. Laufzeiten von $R^{lin}(P, P')$ und $R_4(P, P')$ mit Algorithmus $B(0.5)$	58

Tabellenverzeichnis

1.1. Eigenschaften simultan ausgeführter Operatoren in parallelen Plänen	9
3.1. Vergleich der Implementierungen von <i>Parallelismaxioms</i>	45
4.1. Laufzeiten in s von Algorithmus S	54
4.2. Laufzeiten in s von Algorithmus $A(2)$	55
4.3. Laufzeiten in s von Algorithmus $B(0.5)$	56
4.4. Planlängen mit Algorithmus $S / A(2) / B(0.5)$	57
4.5. Anzahl und Größe der SCCs: $n \times m$ entspricht n SCCs mit m Operatoren	57

Literaturverzeichnis

- [BCCZ99] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. *Lecture Notes in Computer Science*, 1579:193–207, 1999.
- [BCRT01] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, pages 473–478, 2001.
- [BF95] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.
- [BG01] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.
- [Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.
- [CRV01] Michel Cayrol, Pierre Régnier, and Vincent Vidal. Least commitment in graphplan. *Artificial Intelligence*, 130:85–118, 2001.
- [DNK97] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In *ECP*, pages 169–181, 1997.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.
- [KKKV05] Victor Khomenko, Alex Kondratyev, Maciej Koutny, and Walter Vogler. Merged processes – a new condensed representation of petri net behaviour. In *CONCUR*, pages 338–352, 2005.
- [KS92] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.
- [KS96] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In Howard Shrobe and Ted Senator, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California, 1996. AAAI Press.

- [KS99] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. Computer Science Department, University of Maryland, 1999.
- [McD98] Drew McDermott. PDDL — The Planning Domain Definition Language. 1998.
- [McD00] Drew McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, Summer 2000.
- [NGT04] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory and Practice*. San Francisco, CA : Morgan Kaufmann Publishers, 2004.
- [Pau91] Laurence C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.
- [RHN04] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Parallel encodings of classical planning as satisfiability. In J. J. Alferes and J. Leite, editors, *Logics in Artificial Intelligence: 9th European Conference, JELIA*, number 3229 in Lecture Notes in Computer Science, pages 307–319, 2004.
- [RHN05] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: Parallel plans and algorithms for plan search. Technical Report 216, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2005.
- [Rin04a] Jussi Rintanen. Evaluation strategies for planning as satisfiability. In R. Lopez de Mantaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI 2004*, pages 682–687. IOS Press, 2004.
- [Rin04b] Jussi Rintanen. Introduction to automated planning. Vorlesungsmanuskript, 2004.
- [Rin05] Jussi Rintanen. Automated Planning: Algorithms and Complexity. Habilitationsschrift, Juli 2005.
- [RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence – A Modern Approach*. Series in Artificial Intelligence. Prentice Hall, 2. edition, 2003.
- [Rya04] Lawrence Ryan. Efficient Algorithms for Clause-Learning SAT Solvers. Diplomarbeit, Simon Fraser University, 2004.
- [Sch00] Uwe Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 5. edition, 2000.