Planen im Fluentkalkül mit binären Entscheidungsdiagrammen

Dissertation
zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht am 11.10.2004 von
Hans-Peter Störr
geboren am 3.4.1972 in Zwickau

Störr, Hans-Peter:
Planen im Fluentkalkül mit binären Entscheidungsdiagrammen / Hans-Peter Störr. –
Als Ms. gedr.. – Berlin : dissertation.de – Verlag im Internet GmbH, 2005
Zugl.: Dresden, Univ., Diss., 2005
ISBN 3-86624-062-7

Betreuer:	Prof. Dr. rer. nat. habil. Steffen Hölldobler
Gutachter:	Prof. Dr. rer. nat. Michael Thielscher
Promotionskommission:	Vorsitz: Prof. Dr.Ing. habil. Reiner G. Spallek Prof. Dr. rer. nat. Hermann Härtig (Ersatzmitglied) Dr. rer. nat. habil. Boris Flach
Tag der Einreichung:	11.10.2004
Tag der Verteidigung:	21.04.2005

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

dissertation.de - Verlag im Internet GmbH
Pestalozzistraße 9
10 625 Berlin

URL: http://www.dissertation.de

Geleitwort

Eines der wesentlichen Merkmale intelligenten Verhaltens ist die Fähigkeit zur Handlungsplanung. Es überrascht deshalb auch nicht, dass die Handlungsplanung von Anfang an ein zentrales Teilgebiet der Intellektik, d.h., der Künstlichen Intelligenz und der Kognitionswissenschaften, war und ist. Dabei stehen das Verstehen der menschlichen Fähigkeit zur Handlungsplanung wie auch die Erstellung maschineller Agenten, seien es autonome Robotor oder Softwareagenten, mit entsprechenden Fähigkeiten im Vordergrund.

Es zeigte sich sehr schnell, dass die Erstellung maschineller Agenten mit der Fähigkeit zur Handlungsplanung ein sehr anspruchsvolles und schwieriges Forschungsgebiet ist, da es u.a. die Fähigkeit zur Darstellung von Alltagswissen vorraussetzt und Schlüsse über Alltagswissen ermöglichen muss. In der Intellektik gibt es eine Reihe konkurrierender Ansätze, die sich mit diesen Fragestellungen beschäftigen, wobei die logik-basierten Ansätze u.a. den Vorteil einer formal definierten und deklarativen Semantik bieten. Seit Jahren gibt es mehrere miteinander konkurrierende logik-basierte Verfahren, darunter den Fluentkalkül.

Hans-Peter Störr beschäftigt sich in der vorliegenden Dissertation vor allem mit Fragen der Repräsentation von Wissen über ein komplexes dynamisches System im Fluentkalkül und der Berechnung von Handlungsfolgen, die auf dem repräsentierten Wissen beruhen. Aufbauend auf einer sehr schönen Einführung in Grundlagen, binäre Entscheidungsdiagramme und Planungsprobleme leistet die Arbeit im Kern zwei Beiträge: Die Entwicklung einer neuen, verbesserten Semantik für den Fluentkalkül sowie die Spezifikation und Implementierung eines auf binäre Entscheidungsdiagramme basierenden Planungsverfahren für ein aussagenlogisches Fragment des Fluentkalküls.

Die Arbeit vermittelt tiefe Einblicke in die Handlungsplanung und zeigt formal sehr sauber entwickelte und implementierte Lösungsmöglichkeiten für einzelne Aspekte auf.

Dresden, 11. November 2005 Steffen Hölldobler

Danksagung

Diese Arbeit ist zum größten Teil im Rahmen des Graduiertenkollegs „Spezifikation diskreter Prozesse und Prozesssysteme durch operationelle Modelle und Logiken“ entstanden. Ich möchte mich sowohl für die finanzielle Unterstützung durch ein Stipendium, als auch für die vielen Diskussionen und Anregungen, die ich in dessen Veranstaltungen bekommen habe, bedanken - ohne dies wäre diese Arbeit nicht möglich gewesen.

Dann möchte ich vor allem meinem Betreuer, Prof. Dr. Steffen Hölldobler, für die vielen Gespräche und Empfehlungen danken. Die gemeinsamen Publikationen haben mir Spaß gemacht und mich viel gelehrt. Die Vorträge und Diskussionen in und außerhalb der Seminare des Instituts für künstliche Intelligenz haben sich ebenfalls als sehr fruchtbar erwiesen.

Weiterhin möchte ich meinem Vater für seine Unterstützung danken, und meiner liebsten Eva-Maria für die vielen schönen Stunden, die mich motivierten. Ein herzliches Dankeschön auch an Ronald Friedrich, Matthias Winter, Claudia Heyroth und allen anderen, die mir beim Herausbügeln von Tippfehlern geholfen haben.

Man muss sich
ein bestimmtes Quantum Zeit gönnen
wo man nichts tut,
damit einem was einfaellt.

— J. Adler

Jemand,
der einen Berg versetzen will,
beginnt damit,
kleine Steine wegzutragen.

— Ein Chinesisches Sprichwort

Inhaltsverzeichnis

1. Einführung

Seit langem ist die Intelligenz des Menschen für viele Forscher und Philosophen ein faszinierendes Forschungsobjekt. Mit dem Aufkommen der Computertechnik erscheint nun zum ersten mal der Traum als realistisch, einige dieser typisch menschlichen Fähigkeiten nicht nur zu verstehen, sondern nachbauen oder gar übertreffen zu können. Es geht nicht mehr nur um Träume von Phantasten, Showobjekte wie eine Schach spielende Maschine zu erschaffen. Intelligente Maschinen bekommen auch in wachsendem Maße eine praktische Bedeutung. Einerseits entsteht durch die Computernetze ein virtueller Handlungsraum, in dem sich eine intelligente Maschine nutzbringend betätigen könnte, und andererseits macht der Bau von Robotern immer größere Fortschritte, so dass eine maschinelle Intelligenz nicht mehr losgelöst von der Realität wäre.

Aufgrund der unfassbaren Komplexität des menschlichen Hirns erfolgt die Forschung noch auf vielen, im Vergleich zur menschlichen Komplexität kleinen Teilgebieten. Dennoch besitzt die künstliche Intelligenz (KI) eine steigende Praxisrelevanz. Genauso wie ein Hammer für viele Anwendungen ein weit besseres Werkzeug als die Faust ist, obwohl die Hand den Hammer an Anwendungen und Vielseitigkeit bei weitem übertrifft, gestatten Schnelligkeit und Präzision dem Computer oft, das menschliche Vermögen in speziellen Anwendungen bei weitem zu übertreffen.

Eine der markantesten Fähigkeiten des Menschen, in der er die bisherigen Formen maschineller Intelligenz weit hinter sich lässt, ist die Möglichkeit eines zielstrebigen Handelns. Computer wurden erbaut, um extrem detaillierte Strukturen vorgegebener Aktionen und Handlungsanweisungen zu befolgen. Ein selbstständiges Finden von Wegen zur Verwirklichung von Zielen und Teilzielen ist aber auch heute noch vor allem eine Domäne des Menschen. Sein Wissen über seine Umwelt gestattet es ihm, Voraussagen über die Wirkung seiner Aktionen zu machen. Damit kann er Vermutungen über die möglichen Wege zum Erreichen eines Ziels anstellen, sowie eine Auswahl unter diesen Wegen treffen. Oft wird die Fähigkeit des zielstrebigen Handelns als eines der wichtigsten Merkmale der Intelligenz angesehen. Daher haben sich auch viele Forscher mit der maschinellen Umsetzung dieses Problems beschäftigt.

Wie sooft in der Forschung wurde hierbei zwar die Natur (in Form intelligenter Lebewesen) zum Vorbild genommen; aber die vorhandenen Hilfsmittel wurden neu kombiniert und erweitert, um auf kreative Weise das Vorbild nachzugestalten, ohne es zu kopieren. Damit wurden trotz begrenzter Mittel neue Eigenschaften ermöglicht. Eines der zentralen Hilfsmittel bei der Lösung von Problemstellungen aus der künstlichen Intelligenz ist die mathematische Logik. Diese ist ein Ergebnis der jahrtausendealten Bemühungen des Menschen zur Schärfung seines

Verstandes, um immer komplexere Probleme beherrschen zu können. Mit dem Aufkommen von Computern kam aber auch eine neue Motivation hinzu: Um den menschlichen Verstand nachbilden zu können, müssen Regeln des Verstandes in eine in sich abgeschlossene und dadurch maschinell beherrschbare Form gebracht werden. Die Anwendung der mathematischen Logik erscheint daher als ein geeignetes Mittel für die Charakterisierung der entsprechend dieser Verstandesregeln entwickelten komplexen Systeme und Strukturen in der Technik und zum Teil auch in der menschlichen Gemeinschaft.[1]

In der Vergangenheit wurden, beginnend mit [63], [39], [64], zahlreiche Ansätze zur Umsetzung des Problems zielgerichteten Handelns in diesem Rahmen entwickelt. Es stellen sich dabei zwei eng verknüpfte Grundfragen:

1. Wie ist das Wissen über ein komplexes dynamisches System und dessen Veränderung, sowie eine in diesem Zusammenhang zu lösende Problemstellung zu repräsentieren? (Der **repräsentationale Aspekt** des Problems.)

2. Wie ist es möglich, aus einer geeignet repräsentierten Problembeschreibung eine Handlungsanweisung (**Plan** genannt) zum Erreichen eines Zieles zu ermitteln? (Der **inferentielle Aspekt** des Problems.)

Im Laufe der Weiterentwicklung hat sich dabei eine gewisse Spaltung der Forschung gemäß der beiden Grundfragen entwickelt. Fokussiert auf vor allem die erste Grundfrage ist das Forschungsgebiet „Reasoning about Action and Change“ (engl. für Schließen über Aktion und Veränderung; seit neuerem auch oft als „Cognitive Robotics“, kognitive Robotik, benannt). Dieses beschäftigt sich – oft ohne große Beachtung des inferentiellen Aspekts – damit, ein dynamisches System und die darin geschehenden Veränderungen adäquat zu beschreiben. Darunter zählen sowohl die formale Darstellung von Vorgängen im System selbst (oft natürliche Aktionen genannt), als auch die von Veränderungen durch aktives Eingreifen eines Agenten in das System durch **Aktionen**. Auf der anderen Seite entwickelte sich das Gebiet des Planens, das sich vor allem dem inferentiellen Aspekt widmet. Dafür werden oft relativ starke Einschränkungen in der Ausdrucksstärke der Beschreibungssprache und Adäquatheit der Beschreibung des dynamischen Systems in Kauf genommen, um die Komplexität der Suche auf ein für praktische Probleme beherrschbares Maß zu begrenzen.

Für die Lösung des repräsentationalen Problems wurde eine Anzahl von Formalismen entwickelt wie z.B. der Situationskalkül und darauf basierende Ansätze [63, 64, 72], der Eventkalkül [57] oder der Fluentkalkül [44, 84]. Die vorliegende Arbeit leistet einen Beitrag zur Weiterentwicklung des Fluentkalkül. Der Fluentkalkül ist ein moderner Ansatz zur Beschreibung von dynamischen Systemen [86], der sich von anderen Ansätzen u.a. durch eine elegante Lösung

[1] Dem Autor ist allerdings unklar, inwieweit die Logik eine tatsächliche Basis menschlichen Alltagshandelns ist: Es scheint, dass man bei vielen praktisch relevanten Problemen eine Behauptung ebenso logisch und überzeugend nachweisen kann, wie ihr Gegenteil. Dies untergräbt aber keineswegs deren Bedeutung für die wissenschaftlich–technische Praxis.

des lange diskutierten sogenannten **Rahmenproblems** [64] unterscheidet [43], und zudem einige erweiterte Ausdrucksmöglichkeiten wie eine natürliche Repräsentation von Ressourcen gestattet.

Ein Kernpunkt des Fluentkalkül ist die Darstellung von Zuständen mit Hilfe eines zweistelligen Operators $\circ$. Dieser wird genutzt, um Fluenten, die Fakten über einen Zustand repräsentieren, zu einer kompletten Zustandsbeschreibung zu verknüpfen. So kann z.B. für die wohlbekannte „Blocksworld" der Zustand, in dem ein Block $\mathtt{A}$ auf einem Block $\mathtt{B}$ steht, der wiederum auf dem Tisch steht, durch folgenden Term repräsentiert werden:

$$\mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A}, \mathtt{B}) \circ \mathit{OnTable}(\mathtt{B}) \ .$$

Mit Hilfe dieser Darstellung können in Prädikatenlogik 1. Stufe unter Verwendung der Fluentkalkül-Gleichungstheorie AC1 (Assoziativität, Kommutativität und Einselement für den Operator $\circ$) dynamische Systeme axiomatisiert werden, so dass die Entwicklung eines solchen Systems logisch geschlussfolgert werden kann. Wenn z.B. in der Blocksworld in der Anfangssituation $\mathtt{S}_0$ die Aktion $\mathit{ToTable}(\mathtt{A})$ (Block $\mathtt{A}$ wird auf den Tisch gesetzt) ausgeführt wird, so ergibt sich ein Zustand (*State*), in dem beide Blöcke auf dem Tisch stehen. Dies spiegelt sich in einer logischen Schlussfolgerung wieder:

$$\mathcal{F}_{Blocksworld} \models \mathit{state}(\mathit{do}(\mathit{ToTable}(\mathtt{A}), \mathtt{S}_0)) = \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) \circ \mathit{OnTable}(\mathtt{A}) \circ \mathit{OnTable}(\mathtt{B}),$$

wobei $\mathcal{F}_{Blocksworld}$ alle zur Axiomatisierung der Blocksworld nötigen Axiome enthält. Eine ausführliche Beschreibung des Fluentkalküls ist in Kapitel 6 zu finden.

Als Inferenzmechanismus für den Fluentkalkül wurde SLDE- bzw. SLDENF-Resolution mit der Fluentkalkül-Gleichungstheorie AC1 vorgeschlagen [45, 80]. Dies ermöglicht eine effiziente Lösung des Rahmenproblems, sofern nicht Restriktionen der SLDENF-Resolution wie Beschränkung auf normale logische Programme und Beschränkungen bezüglich Anfragen mit nicht vollständig spezifizierten Zuständen, die bei Auftreten von Negation zu hängenden Zweigen (Floundering) führen können, verletzt werden. Insbesondere letzteres Problem wird durch die Kombination von logischer Programmierung mit einem Constraint Solver im neueren sogenannten *FLUent calculus eXecutor* **FLUX** behoben [88].

Obwohl der Fluentkalkül bereits mit Erfolg in einfachen realistischen Szenarien wie z.B. Kontrolle eines Roboters zur internen Belieferung in einem Büro eingesetzt wurde [88], so erbt er doch das den meisten Ansätzen zur Planung gemeinsame Problem der sogenannten Zustandsraumexplosion. Nehmen wir z.B. an, dass einem Agenten zu jeder Zeit 10 Aktionen zur Verfügung stehen. Die Ausführung genau einer Aktion kann daher unter Umständen zu verschiedenen 10 Zuständen führen. Die Ausführung zweier Aktionen kann aber bereits zu 100 Zuständen führen, die von dreien zu 1000 usw. Um festzustellen, mit welcher Aktionssequenz der Agent sein Ziel erreichen kann, muss aber u.U. die Mehrzahl dieser Zustände durchsucht werden. Je nach Komplexität des Problems kann dies sehr schnell zu einer ohne Weiteres nur schwer beherrschbaren Größe des Zustandsraumes führen. Da dessen Größe exponentiell mit

der Aktionen- bzw. Fluentenzahl ansteigt, ist dieses Problem nicht einfach durch Effizienzsteigerung der eingesetzten Implementation zu lösen. Eine in ähnlichen Szenarios viel benutzte Möglichkeit dieses Problem zu umgehen besteht darin, die Suche gezielt zu führen, und mit Heuristiken vor allem vielversprechende Aktionssequenzen zu berücksichtigen. In dieser Arbeit soll allerdings eine andere Methodik für den Kontext von Planungsproblemen im Fluentkalkül ausgearbeitet werden, die durch eine Änderung der Repräsentation ein paralleles Durchsuchen sehr großer Zustandsmengen ermöglicht.

Eine solche Herangehensweise an Planungsprobleme wurde in [17] eingeführt. Es handelt sich um die Nutzung von sogenannten BDD (engl. binary decision diagram, BDD) [13], die bereits bei vielen Anwendungen zu neuen Leistungsmerkmalen geführt haben. BDDs sind eine Repräsentation für boolesche Funktionen (siehe Kapitel 3), die sich in praktischen Problemen als oft sehr kompakt erwiesen hat: Sie ist oft exponentiell kompakter als andere Repräsentationen wie konjunktive oder disjunktive Normalform. Weiterhin ermöglicht sie eine effiziente Implementation von logischen Operationen über booleschen Funktionen [14]. Neben den ursprünglichen Einsatzmöglichkeiten, wie in der Konstruktion und Verifikation von digitalen elektronischen Schaltungen, hat der Einsatz von BDDs sich auch in Gebieten bewährt, die nicht direkt auf booleschen Funktionen beruhen. Zum Beispiel wurden BDDs in vielen Aufgaben bei computerunterstütztem Design und Verifikation zur Durchsuchung von Zustandsräumen von 10^{120} und mehr Zuständen eingesetzt, indem die Berechnung der Zustandsmengen in einer Kodierung als boolesche Funktion durchgeführt wird [16]. (In Kapitel 3 wird ein Eindruck von der Funktionsweise solcher Kodierungen über die charakteristische Funktion gegeben.) Mit dieser Verfahrensweise konnten Systeme untersucht werden, die außerhalb der Reichweite anderer Methoden lagen [15, 16]. Der Erfolg im Einsatz der BDDs beruht darauf, dass durch die symbolische Kodierung der Zustandsmengen die Zustände nicht einzeln untersucht werden müssen, sondern die Mengen „am Stück“ in ihrer Repräsentation als BDD verarbeitet werden. Die Größe dieser Repräsentation ist aber nicht abhängig von der Größe der Menge, sondern von der Struktur der Menge, und da die Operationen über der BDD-Darstellung im Aufwand meist proportional zu den Größen der beteiligten BDDs sind, kann dies sehr viel effizienter sein als bei anderen Darstellungsweisen.

In dieser Arbeit soll nun untersucht werden, ob und wie sich diese Verfahrensweise auf den Fluentkalkül in den Rahmen der Prädikatenlogik übertragen lässt. Hier handelt es sich ja nicht mehr um Zustandsräume, die dargestellt und durchsucht werden sollen, sondern um die Berechnung von logischen Konsequenzen aus der Axiomatisierung eines Planungsproblems. In dieser Arbeit wird nun die algorithmische Struktur zum Lösen einer Klasse von im Fluentkalkül formulierter Planungsprobleme auf der Basis von BDD-Algorithmen geschaffen. Die Grundidee dabei ist die schrittweise Berechnung von als BDD dargestellten Mengen von logischen Konsequenzen einer bestimmten Struktur aus der Axiomatisierung des Planungsproblems, bis eine Lösung des Problems gefunden wird oder die Unlösbarkeit des Problems nachgewiesen ist.

Dazu leistet die Arbeit drei grundsätzliche Beiträge. Erstens wird in Kapitel 6 eine neue Gleichungstheorie als Basis für den Fluentkalkül eingeführt, für die nachgewiesen wird, dass sie

(mit Anpassungen) bei Bedarf die alte Gleichungstheorie EUNA [84] ersetzen kann und einige Beschränkungen der alten Gleichungstheorie überwindet (siehe Abschnitt 6.4). Diese bildet die semantische Basis für die weiteren Betrachtungen in dieser Arbeit. (Sie fand zudem bereits Einsatz in anderen Arbeiten zum Fluentkalkül [86, 87].)

Zum zweiten führt Kapitel 7 eine formale Semantik im Fluentkalkül für ein praktisch bedeutsames Fragment der Beschreibungssprache PDDL (Planning Domain Definition Language, engl. für Planungsystembeschreibungssprache) ein. PDDL wurde zur vierten conference for Artificial Intelligent Planning Systems (AIPS) 1998 eingeführt, um einen Vergleich von verschiedenen Planungssystemen zu ermöglichen und sichert somit als de-facto Standardeingabesprache für eine Vergleichbarkeit der Effizienz der derzeitigen experimentellen Planungssysteme. Bisher existierte zwar eine Syntaxdefinition und informale Beschreibung der Semantik, aber keine formale Semantikdefinition. Diese formale Semantik bildet die Grundlage des Vergleichs einer Implementation der in dieser Arbeit entwickelten Algorithmen mit anderen Planungsprogrammen (Kapitel 9).

Zum dritten wird mit Kapitel 8 ein BDD-basierter Inferenzmechanismus geschaffen, der in einem Fragment[2] des Fluentkalküls formulierte Planungsprobleme lösen kann oder die Unlösbarkeit der Planungsprobleme nachweist. In Kapitel 9 werden die Implementation sowie Möglichkeiten zur Effizienzsteigerung diskutiert und experimentelle Ergebnisse der Implementation im Vergleich mit anderen Planungssystemen, die den Stand der Kunst darstellen, präsentiert. Kapitel 10 vergleicht den erreichten Stand mit Entwicklungen in verwandten Forschungsrichtungen und diskutiert Richtungen für mögliche zukünftige Weiterentwicklungen.

[2] Dieses Fragment umfasst das Fragment, in das die PDDL-Planungsprobleme übersetzt werden.

2. Grundlagen

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

In dieser Arbeit wird vorausgesetzt, das der Leser mit den Grundlagen der mathematischen Logik vertraut ist. Es wird weitgehend die Standardnotation und Terminologie der mathematischen Logik nach Lloyd [59] verwendet. Die benötigten Grundbegriffe sind im folgenden kurz dargestellt; für eine detaillierte Darstellung verweisen wir jedoch auf die Literatur.

2.1. Notationen und Konventionen

In Normalfall ist von einem Bezeichner auf dessen Typ zu schließen. In allgemeinen logischen Formeln gilt:

- Variablen werden mit $x, y, z \ldots$ bezeichnet.
- Konstanten werden mit großen Buchstaben $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ bezeichnet.
- Funktionen werden mit f, g, h bezeichnet.
- Prädikate und aussagenlogische Variablen werden mit p, q, r, s, t, u bezeichnet.
- Logische Formeln werden mit kleinen griechischen Buchstaben $\phi, \psi, \theta, \ldots$ bezeichnet.

Im Kontext von Planungsproblemen gelten die folgenden Regeln:

- Fluentnamen, Aktionsnamen und Bedingungsnamen werden kursiv gesetzt: $on, totabl$.
- Namen für Objekte werden mit Großbuchstaben $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ gesetzt.
- Namen für Prädikate werden kursiv gesetzt: $append, causes, doplan$.
- Variablen werden wie folgt bezeichnet:
  - a für Aktionsbezeichner (d.h. eine Variable der Sorte[1] *Action*),

[1] Siehe Abschnitt 2.3 auf Seite 10

- f für Fluenten (Sorte *Fluent*),
- o für Objekte (Sorte *Object*),
- s für Situationen (Sorte *Sit*),
- z für Zustände (Sorte *State*)

Diese werden zum Teil mit Indizes versehen.

Diese Benennungen haben in logischen Formeln Vorrang gegenüber den allgemeinen Konventionen.

Weiterhin werden

- Namen für Strukturen (Tupel von Mengen, Relationen usw.) in serifenloser Schrift gesetzt: $\mathsf{F}, \mathsf{G}, \mathsf{T}$.
- Mengen und Relationen mit kalligraphischen Buchstaben bezeichnet: $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$.

Definitionen von Symbolen werden mit $\underset{\text{Def}}{\equiv}$ gekennzeichnet. Zum Beispiel:

Definiendum $\underset{\text{Def}}{\equiv}$ **Definiens**

definiert den Begriff Definiendum zu Definiens.

2.2. Aussagenlogik

Sei $\mathcal{V}$ eine Menge aussagenlogischer Variablen. Eine aussagenlogische Formel ist eine Variable aus $\mathcal{V}$ oder ein Ausdruck der Form $\mathbf{0}$, $\mathbf{1}$, $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, $(\phi \leftrightarrow \psi)$, $(\phi \oplus \psi)$, wobei ϕ und ψ aussagenlogische Formeln sind.

Eine **Interpretation** ist eine Abbildung von $\mathcal{V}$ auf die Menge der Wahrheitswerte $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$, mit den Wahrheitswerten $\mathbf{0}$ für „falsch“ und $\mathbf{1}$ für „wahr“. Der **Wahrheitswert** einer Formel bezüglich einer Interpretation $\mathcal{I}$ ist induktiv definiert wie folgt:

- $\mathbf{1}$ ist wahr (d.h. hat den Wahrheitswert $\mathbf{1}$) und $\mathbf{0}$ ist falsch (d.h. hat den Wahrheitswert $\mathbf{0}$).
- Eine Variable $v \in \mathcal{V}$ ist wahr genau dann, wenn $\mathcal{I}(v) = \mathbf{1}$.
- (ϕ) ist wahr genau dann, wenn ϕ wahr ist.
- $(\neg\phi)$ ist wahr genau dann, wenn ϕ falsch ist.

- $(\phi \wedge \psi)$ ist wahr genau dann, wenn ϕ und ψ wahr sind.

Die restlichen Operatoren werden wie Abkürzungen behandelt:

$$\begin{aligned}
(\phi \vee \psi) &\underset{\text{Def}}{\equiv} \neg(\phi \wedge \psi) \\
(\phi \rightarrow \psi) &\underset{\text{Def}}{\equiv} (\neg\phi \vee \psi) \\
(\phi \leftrightarrow \psi) &\underset{\text{Def}}{\equiv} ((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)) \\
(\phi \oplus \psi) &\underset{\text{Def}}{\equiv} (\neg\phi \leftrightarrow \psi)
\end{aligned}$$

Eine Formel ist **erfüllbar**, wenn es eine Interpretation gibt, für die sie den Wahrheitswert $\mathbf{1}$ hat. Ansonsten ist sie unerfüllbar. Sie ist **allgemeingültig** (oder eine **Tautologie**), wenn sie für jede Interpretation den Wahrheitswert $\mathbf{1}$ hat.

Im Kontext aussagenlogischer Formeln bezeichnet $\phi\{p/\psi\}$ die Formel die entsteht, wenn man in der Formel ϕ alle Vorkommen der aussagenlogischen Variablen p simultan durch die Formel (ψ) ersetzt (**aussagenlogische Substitution**). Davon abgeleitet ergeben sich die folgenden Abkürzungen für aussagenlogische Formeln (auch **aussagenlogische Quantifizierung** genannt):

$$\begin{aligned}
(\exists\, x)\ \phi &\underset{\text{Def}}{\equiv} (\phi\{x/\mathbf{0}\}) \vee (\phi\{x/\mathbf{1}\}) \\
(\forall\, x)\ \phi &\underset{\text{Def}}{\equiv} (\phi\{x/\mathbf{0}\}) \wedge (\phi\{x/\mathbf{1}\})\ .
\end{aligned}$$

Analog zu den gleichnamigen Quantoren in Logik erster Stufe gelten hier die Äquivalenzen

$$\begin{aligned}
(\exists\, x)\ \phi &\leftrightarrow \neg\,(\forall\, x)\ \neg\,\phi \\
(\exists\, x)\ (\phi \vee \psi) &\leftrightarrow (\exists\, x)\ \phi \vee (\exists\, x)\ \psi \\
(\forall\, x)\ (\phi \wedge \psi) &\leftrightarrow (\forall\, x)\ \phi \wedge (\forall\, x)\ \psi\ .
\end{aligned}$$

Um Klammern zur Übersichtlichkeit weglassen zu können, haben die Operatoren die folgende Vorrangfolge (von stärkster zu geringster Bindungskraft): $\forall, \exists, \neg,\ \wedge\ ,\ \vee\ , \oplus,\ \rightarrow\ ,\ \leftrightarrow$. Weiterhin wird oft, sofern dies die Eindeutigkeit der Formeln nicht beeinträchtigt, $\wedge$ z.T. weggelassen sowie $\overline{\phi}$ anstatt von $\neg\,(\phi)$ geschrieben. Als Abkürzung wird $\mathrm{ite}(\phi, \psi, \theta)$ (von „if-then-else“, engl. für „wenn–dann–sonst“) wie folgt definiert:

$$\begin{aligned}
\mathrm{ite}(\phi, \psi, \theta) &\underset{\text{Def}}{\equiv} (\phi \wedge \psi) \vee (\neg\,\phi \wedge \theta) \\
&\equiv \quad \phi\psi \vee \overline{\phi}\theta\ .
\end{aligned}$$

Dabei wird ϕ als der Test, ψ als der Then-Teil und θ als der Else-Teil von $\mathrm{ite}(\phi, \psi, \theta)$ bezeichnet.

2.3. Ordnungssortierte Logik

In dieser Arbeit nutzen wir eine Variante der Prädikatenlogik erster Stufe, die es erlaubt, für Prädikate und quantisierte Variablen Sorten zu deklarieren. Diese ermöglicht sowohl eine übersichtlichere Darstellung der Sachverhalte, da unsinnige Terme (wie „1+Bratpfanne-7“) bereits syntaktisch ausgeschlossen werden, als auch u.U. eine effizientere Implementation des logischen Schließens, da der Suchraum der logischen Konsequenzen um Formeln mit unsinnigen Termen reduziert wird. Für unsere Präsentation modifizieren wir die Darstellung [73] etwas.

Eine ordnungssortierte Signatur ist ein Tupel $\Sigma = \left\langle \mathcal{S}, \preceq, (\mathcal{V}_s)_{s\in\mathcal{S}}, \mathcal{OP}, \mathcal{P} \right\rangle$, wobei folgendes gilt:

- S ist eine Menge von Sortensymbolen.
- $\preceq \subseteq S \times S$ ist eine partielle Ordnung.
- $(\mathcal{V}_s)_{s\in\mathcal{S}}$ eine Familie von abzählbar unendlichen disjunkten Mengen von Variablen (eine Menge pro Sorte).
- $\mathcal{OP}$ ist eine Menge von **Operationsdeklarationen** $f : s_1 s_2 \ldots s_n \rightarrow s$ mit $n \geq 0$, $s_1, \ldots, s_n, s \in \mathcal{S}$, wobei kein Funktionssymbol f in zwei verschiedenen Deklarationen vorkommen darf.
- $\mathcal{P}$ ist eine Menge von **Prädikatsdeklarationen** $p : s_1 s_2 \ldots s_m$ mit $m \geq 0$, $s_1, \ldots, s_m \in \mathcal{S}$, wobei kein Prädikatssymbol p in zwei verschiedenen Deklarationen vorkommen darf.
- Alle Variablen, Sortensymbole, Funktionssymbole und Prädikatssymbole sind paarweise verschieden.

Der Übersichtlichkeit halber wird im Text eine Signatur oft z.B. wie folgt geschrieben:

$$
\begin{array}{ll}
\textbf{SORT} & s_1,\ s_2,\ s_3 \preceq s_2,\ s_4 \preceq s_3, \\
\textbf{FUN} & f_1 \ : \rightarrow s_2, \\
 & f_2 \ : s_1 \times s_3 \rightarrow s_2, \\
 & f_3 \ : s_4 \rightarrow s_1, \\
\textbf{REL} & p_1 \ : s_1, \\
 & p_2 \ : s_2 \times s_3 \times s_4.
\end{array}
$$

Dies entspricht einer Signatur

$$
\begin{aligned}
\Sigma = \Big\langle & \{s_1, s_2, s_3, s_4\}, \\
& \{s_1 \preceq s_1, s_2 \preceq s_2, s_3 \preceq s_2, s_3 \preceq s_3, s_4 \preceq s_2, s_4 \preceq s_3, s_4 \preceq s_4\}^{\text{ref tra}}, \\
& \qquad (\mathcal{V}_s)_{s\in\mathcal{S}},\ \{f_1 : \rightarrow s_2,\ f_2 : s_1 s_3 \rightarrow s_2,\ f_3 : s_4 \rightarrow s_1\},\ \{p_1 : s_1,\ p_2 : s_2 s_3 s_4\} \Big\rangle\ ,
\end{aligned}
$$

wobei $\mathcal{R}^{\text{ref tra}}$ die transitive reflexive Hülle einer Relation $\mathcal{R}$ ist. Man beachte, dass die Relation $\preceq$ stets die transitive reflexive Hülle der im Text angegebenen Beziehungen ist.

Die Menge der **wohlsortierten Terme** $T_{\Sigma,s}$ der Sorte s bezüglich einer Signatur Σ wird rekursiv durch die folgenden Regeln erzeugt:

- Wenn $x \in \mathcal{V}_s$, dann ist $x \in T_{\Sigma,s}$.
- Wenn $t \in T_{\Sigma,s_1}$, $s_1 \neq s_2$ und $s_1 \preceq s_2$, dann ist $t \in T_{\Sigma,s_2}$.
- Wenn $f : s_1 s_2 \ldots s_n \rightarrow s \in \mathcal{O}P$ und $t_1 \in T_{\Sigma,s_1}$, $t_2 \in T_{\Sigma,s_2}$, $\ldots$, $t_n \in T_{\Sigma,s_n}$, dann ist $f(t_1, t_2, \ldots, t_n) \in T_{\Sigma,s}$. Falls $n = 0$ wird $f()$ zu f abgekürzt.

Ein **Grundterm** ist ein Term, der keine Variablen enthält.

Wohlsortierte Formeln bezüglich einer Signatur Σ werden rekursiv definiert:

- Wenn $p : s_1 s_2 \ldots s_n \in \mathcal{P}$ und $t_1 \in T_{\Sigma,s_1}$, $t_2 \in T_{\Sigma,s_2}$, $\ldots$, $t_n \in T_{\Sigma,s_n}$, dann ist $p(t_1, t_2, \ldots, t_n)$ eine wohlsortierte Formel. Eine solche Formel wird auch **Atom** genannt.
- Wenn ϕ und ψ wohlsortierte Formeln sind und x eine Variable aus $\mathcal{V}_s$ ist, dann sind $\mathbf{0}$, $\mathbf{1}$, $(\neg\phi)$, $(\phi \wedge \psi)$ und $(\forall\, x)\ \phi$ wohlsortierte Formeln.

Die übrigen Operatoren werden wieder als Abkürzungen eingeführt:

$$
\begin{aligned}
(\phi \vee \psi) &\underset{\text{Def}}{\equiv} \neg(\phi \wedge \psi) \\
(\phi \rightarrow \psi) &\underset{\text{Def}}{\equiv} (\neg\phi \vee \psi) \\
(\phi \leftrightarrow \psi) &\underset{\text{Def}}{\equiv} ((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)) \\
(\phi \oplus \psi) &\underset{\text{Def}}{\equiv} (\neg\phi \leftrightarrow \psi) \\
((\exists\, x)\ \phi) &\underset{\text{Def}}{\equiv} (\neg(\forall\, x)\ (\neg\phi))
\end{aligned}
$$

Abermals haben die Operatoren die folgende Vorrangfolge (von stärkster zu geringster Bindungskraft): $\forall, \exists, \neg, \wedge, \vee, \oplus, \rightarrow, \leftrightarrow$. Um die Lesbarkeit zu erhöhen und die Variablenmengen nicht schon von vorn herein festlegen zu müssen, wird auch oft $(\forall\, x : s)\ \phi$ oder $(\exists\, x : s)\ \phi$ geschrieben, um nebenbei auszudrücken, dass die Variable x eine Variable der Sorte s ist (d.h. $x \in \mathcal{V}_s$).

Bei Verwendung des Gleichheitssymbols wird für jede Sorte s ein Prädikat $=_s : ss$ zur Signatur hinzugefügt. Anstatt $=_s(t_1, t_2)$ schreibt man zur besseren Lesbarkeit $t_1 =_s t_2$ oder, falls s aus dem Kontext hervorgeht, einfach $t_1 = t_2$.

Betrachten wir nun eine beliebige fest vorgegebene Signatur $\Sigma = \langle \mathcal{S}, \preceq, (\mathcal{V}_s)_{s\in\mathcal{S}}, \mathcal{OP}, \mathcal{P} \rangle$. Im Folgenden werden wir nur wohlsortierte Formeln und Terme verwenden und lassen daher den Begriff „wohlsortiert“ der Kürze halber weg.

Ein **Vorkommen** einer Variablen x in einer Formel heißt **gebunden**, wenn es in einem Teilausdruck der Form $((\exists\ x)\ \ \phi)$ oder $((\forall\ x)\ \ \phi)$ liegt, ansonsten **frei**. Eine Formel heißt **abgeschlossen**, wenn es keine freien Vorkommen von Variablen enthält. $(\forall)\phi$ ist eine Abkürzung für die Allquantifizierung sämtlicher freier Variablen der Formel ϕ: wenn ϕ die freien Variablen $x_1, \ldots, x_k$ enthält, dann ist

$$(\forall)\phi \underset{\text{Def}}{\equiv} (\forall\, x_1)\ \ldots (\forall\, x_k)\ \phi\ .$$

$\mathrm{var}(\phi)$ bezeichnet die Menge der freien Variablen einer Formel ϕ; $\mathrm{var}(t)$ bezeichnet die Menge der Variablen, die in einem Term t vorkommen.

Eine (wohlsortierte) **Substitution** ist eine endliche Menge von Paaren $\{x_1/t_1, \ldots, x_n/t_n\}$, wobei für $i = 1, \ldots, n$ es eine Sorte $s_i \in \mathcal{S}$ gibt, so dass $x_i \in \mathcal{V}_s$, $t_i \in T_{\Sigma,s_i}$ und $x_i \neq t_i$. Die **Instanz** $t\sigma$ eines Terms t unter einer Substitution $\sigma = \{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\}$ ist der Term, der entsteht, wenn man alle Vorkommen der Variablen x_i in t simultan durch die jeweiligen Terme t_i ersetzt. Die Instanz $\phi\sigma$ einer Formel ϕ unter $\sigma = \{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\}$ ist die Formel, die entsteht, wenn man alle freien Vorkommen der Variablen x_i in ϕ simultan durch die jeweiligen Terme t_i ersetzt. Eine Menge von Termen $\{t_1, t_2, \ldots, t_n\}$ einer Sorte s heißt mit dem Unifikator σ **unifizierbar**, wenn σ eine Substitution ist, so dass $t_1\sigma = t_2\sigma = \cdots = t_n\sigma$.

Die **Domäne** $\mathrm{dom}(\sigma)$ einer Substitution σ ist

$$\mathrm{dom}(\sigma) \underset{\text{Def}}{\equiv} \{x \mid x/t \in \sigma\}\ .$$

Die **Einschränkung** $\sigma|_{\mathcal{V}}$ einer Substitution σ auf eine Variablenmenge $\mathcal{V}$ ist

$$\sigma|_{\mathcal{V}} = \{x/t \mid x \in \mathcal{V}\ \wedge\ x/t \in \sigma\}\ .$$

Zwei Substitutionen σ und θ können zu einer Substitution $(\sigma\theta)$ kombiniert werden, so dass für jeden Term t gilt, dass $(t\sigma)\theta = t(\sigma\theta)$: Die **Komposition von Substitutionen** $(\sigma\theta)$ von σ und θ ist definiert als

$$(\sigma\theta) \underset{\text{Def}}{\equiv} \{x/t\theta \mid x/t \in \sigma\ \wedge\ t\theta \neq x\} \cup \{y/s \mid y/s \in \theta\ \wedge\ y \notin \mathrm{dom}(\sigma)\}\ .$$

Eine **Interpretation** ist ein Tupel $\mathcal{I} = \langle (\mathcal{A}_s)_{s\in S}, \_^{\mathcal{I}} \rangle$ aus einer Familie von Mengen $(\mathcal{A}_s)_{s\in S}$ und einer Abbildung $\_^{\mathcal{I}}$ mit folgenden Eigenschaften:

- für alle $s \in \mathcal{S}$ ist $\mathcal{A}_s$ eine nichtleere Menge, genannt **Trägermenge** der Sorte s. Diese muss zu Σ passen, d.h es gilt für alle $s, s' \in \mathcal{S}$ dass $s \preceq s' \rightarrow \mathcal{A}_s \subseteq \mathcal{A}_{s'}$.

- für jede Operationsdeklaration $f : s_1 s_2 \dots s_n \to s \in \mathcal{OP}$ ist $f^{\mathcal{I}}$ eine Abbildung vom Typ $f^{\mathcal{I}} : \mathcal{A}_{s_1} \times \dots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$.
- für jede Prädikatsdeklaration $p : s_1 s_2 \dots s_n \in \mathcal{P}$ ist $p^{\mathcal{I}}$ eine Teilmenge von $\mathcal{A}_{s_1} \times \dots \times \mathcal{A}_{s_2}$.

Eine **Variablenbelegung** bezüglich einer Interpretation $\mathcal{I}$ ist eine Abbildung, die allen Variablen aus $(\mathcal{V}_s)_{s \in \mathcal{S}}$ ein Element aus der Trägermenge $\mathcal{A}_s$ der jeweiligen Sorte der Variable zuordnet. Die **Bedeutung** $\mathcal{I}_\xi(t)$ eines Terms t unter einer Interpretation $\mathcal{I}$ und einer Variablenbelegung ξ ist definiert als:

- Wenn $t = x$ mit $x \in \mathcal{V}_s$, dann ist $\mathcal{I}_\xi(t) = \xi(x)$.
- Wenn $t = f(t_1, t_2, \dots, t_n)$ mit $f : s_1 s_2 \dots s_n \to s \in \mathcal{OP}$, dann ist $\mathcal{I}_\xi(t) = f^{\mathcal{I}}(\mathcal{I}_\xi(t_1), \mathcal{I}_\xi(t_2), \dots, \mathcal{I}_\xi(t_n))$. (Falls $n = 0$ gilt entsprechend $\mathcal{I}_\xi(t) = f^{\mathcal{I}}$.)

Der **Wahrheitswert** einer Formel unter einer Interpretation $\mathcal{I}$ und einer Variablenbelegung ξ ist einer der Werte $\{\mathbf{0}, \mathbf{1}\}$ und wird wie folgt induktiv definiert:

- $\mathbf{0}$ hat den Wahrheitswert $\mathbf{0}$, $\mathbf{1}$ hat den Wahrheitswert $\mathbf{1}$.
- Der Wahrheitswert eines Atoms $p(t_1, t_2, \dots, t_n)$ ist $p^{\mathcal{I}}(\mathcal{I}_\xi(t_1), \mathcal{I}_\xi(t_2), \dots, \mathcal{I}_\xi(t_n))$.
- $(\neg\phi)$ hat den Wahrheitswert $\mathbf{1}$ genau dann, wenn ϕ den Wahrheitswert $\mathbf{0}$ hat.
- $(\phi \wedge \psi)$ hat den Wahrheitswert $\mathbf{1}$ genau dann, wenn ϕ und ψ den Wahrheitswert $\mathbf{1}$ haben.
- Für $x \in \mathcal{V}_s$ hat $(\forall\, x)\ \phi$ den Wahrheitswert $\mathbf{1}$ genau dann, wenn für alle $a \in A_s$ die Formel ϕ unter $\mathcal{I}$ und $(\xi \setminus \{x \mapsto \xi(x)\}) \cup \{x \mapsto a\}$ den Wahrheitswert $\mathbf{1}$ hat.

Für abgeschlossene Formeln hängt der Wahrheitswert der Formel nicht von der Variablenbelegung ab. Daher können wir vom Wahrheitswert einer abgeschlossen Formel unter einer Interpretation sprechen.

Eine Interpretation ist ein **Modell** für eine abgeschlossene Formel ϕ (schreibe: $\mathcal{I} \models \phi$), falls ϕ unter $\mathcal{I}$ den Wahrheitswert $\mathbf{1}$ hat. Eine abgeschlossene Formel ist **erfüllbar**, wenn es eine Interpretation gibt, für die sie den Wahrheitswert $\mathbf{1}$ hat; ansonsten ist sie unerfüllbar. Sie ist **allgemeingültig** (oder eine **Tautologie**), wenn sie für jede Interpretation den Wahrheitswert $\mathbf{1}$ hat.

Eine Formelmenge $\mathcal{G}$ **folgt** aus einer Formelmenge $\mathcal{F}$ (schreibe: $\mathcal{F} \models \mathcal{G}$), wenn jede Interpretation und Variablenbelegung, die jeder Formel aus $\mathcal{F}$ den Wahrheitswert $\mathbf{1}$ zuordnet, auch jeder Formel aus $\mathcal{G}$ den Wahrheitswert $\mathbf{1}$ zuordnet.

Die hier beschriebene mehrsortige Logik lässt sich bei Bedarf mit dem sogenannten Prozess der *Formelrelativierung* in die gewöhnliche unsortierte Prädikatenlogik erster Stufe abbilden, so dass die Erfüllbarkeit einer Formelmenge erhalten bleibt. Die Grundidee dieser Abbildung ist, jeder Sorte s ein Prädikat s in der unsortierten Logik zuzuordnen. Quantifizierte Unterformeln werden bei der Abbildung in den Formeln wie folgt ersetzt: $((\forall\, x : s)\ \phi)$ wird zu $((\forall\, x)\ s(x)\ \rightarrow\ \phi)$ und $((\exists\, x : s)\ \phi)$ wird zu $((\exists\, x)\ s(x)\ \wedge\ \phi)$, wenn s die Sorte der Variablen x ist (d.h. $x \in \mathcal{V}_s$). Weiterhin werden sog. Signaturaxiome zu der Formelmenge hinzugefügt: $((\exists\, x)\ s(x))$ für jede Sorte s, und $((\forall\, x)\ s_1(x)\ \rightarrow\ s_2(x))$ für jedes Paar von Sorten mit $s_1 \preceq s_2$. Für Details sei auf [73] verwiesen.

2.4. Gleichungstheorien und Unifikation

In Kapitel 6 wird die in dieser Arbeit konstruierte Axiomatisierung des Fluentkalküls mit der ursprünglichen Formalisierung, die auf einer unifikationsvollständigen Gleichungstheorie basiert, verglichen. Eine detaillierte Behandlung dieses Themas ist für das Verständnis der in dieser Arbeit behandelten Themen nicht nötig; wir beschränken uns daher auf die Einführung der zum Verständnis notwendigen Grundbegriffe, und verweisen für eine ausführlichere Diskussion z.B. auf [51, 74].

Im vorhergehenden Abschnitt haben wir das Gleichheitssymbol $=$ als normales Prädikatssymbol eingeführt, ohne ihm eine spezielle Semantik zuzuordnen. Um die mit der Gleichheit verbundene Intention zu formalisieren, muss diese in Axiome gefasst werden. Diese sind die im Folgenden präsentierten sogenannten **Standardgleichheitsaxiome**, hier für die mehrsortige Logik:

Für alle Sorten $s \in \mathcal{S}$ und jeweils ein Variablentripel $x, y, z \in \mathcal{V}_s$ ein Axiom

$$(\forall\, x)\ x =_s x\ , \qquad \text{(Reflexivität)}$$
$$(\forall\, x, y)\ (x =_s y\ \rightarrow\ y =_s x)\ , \qquad \text{(Symmetrie)}$$
$$(\forall\, x, y, z)\ (x =_s y\ \wedge\ y =_s z\ \rightarrow\ x =_s z)\ , \qquad \text{(Transitivität)}$$

sowie für jede Operationsdeklaration $f : s_1 s_2 \ldots s_n \rightarrow s \in \mathcal{OP}$, $i \in \{1, \ldots, n\}$ und je einen Satz verschiedener Variablen $x_1 \in \mathcal{V}_{s_1}, \ldots, x_n \in \mathcal{V}_{s_n}$ und $y \in \mathcal{V}_{s_i}$ ein Axiom

$$(\forall\, x_1, \ldots, x_n, y)\ \big(x_i =_{s_i} y \rightarrow\ f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) =_s f\big(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n\big)\ , \qquad \text{(Substitutivität I)}$$

und für jede Prädikatsdeklaration $p : s_1 s_2 \ldots s_n \in \mathcal{P}$, $i \in \{1, \ldots, n\}$ und je einen Satz

verschiedener Variablen $x_1 \in \mathcal{V}_{s_1}, \ldots, x_n \in \mathcal{V}_{s_n}$ und $y \in \mathcal{V}_{s_i}$ ein Axiom

$$(\forall\, x_1, \ldots, x_n, y)\ \big(x_i =_{s_i} y\ \rightarrow \\ [p(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)\ \leftrightarrow\ p(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)]\big)\ .$$

(Substitutivität II)

Oft ist es jedoch nötig weitere Gleichheiten zu definieren. Zum Beispiel wird in Kapitel 6 das bereits erwähnte Funktionssymbol $\circ$ definiert, das u.a. kommutativ und assoziativ ist. Solche Eigenschaften können durch eine Gleichungstheorie erfasst werden. Eine **Gleichungstheorie** E ist eine Menge von Formeln der Form $(\forall)\ t_1 =_s t_2$, wobei t_1 und t_2 Terme einer Sorte $s \in \mathcal{S}$ sind. Wenn für zwei Terme t_1 und t_2 Terme $E \models t_1 =_s t_2$ gilt, so schreibt man auch $t_1 =_E t_2$.

Das Berechnen von Schlussfolgerungen aus Formelmengen unter Verwendung von Gleichungstheorien ist oft sehr aufwändig, wenn man die Axiome Gleichungstheorie gleichberechtigt mit den anderen Formeln verwendet. Daher wird in der bereits erwähnten SLDENF-Resolution die Gleichungstheorie nicht explizit verwendet, sondern implizit über das Bilden von sogenannten E-Unifikatoren [3]. Ein **E-Unifikator** σ zweier Terme $t_1, t_2 \in T_{\Sigma,s}$ ist eine Substitution σ, so dass $t_1 =_E t_2$. Eine **vollständige Menge von E-Unifikatoren** $cU_E(t_1, t_2)$ für zwei Terme t_1 und t_2 ist eine Menge von E-Unifikatoren für t_1 und t_2, so dass es für jeden E-Unifikator ρ von t_1 und t_2 einen E-Unifikator $\sigma \in cU_E(t_1, t_2)$ gibt, der E-allgemeiner als ρ ist: es gibt eine Substitution θ, so dass $\rho|_{\mathrm{var}(s,t)} = (\sigma\theta)|_{\mathrm{var}(s,t)}$.

2.5. Multimengen

Eine Multimenge ist eine Erweiterung des Konzepts der klassischen Menge, so dass Elemente auch mehrfach vorkommen können. Die Häufigkeit des Vorkommens eines Elements ist dabei relevant. Wir kennzeichnen in dieser Arbeit Multimengensymbole und -operationen durch einen übergestellten Punkt. Formal lassen sich Multimengen wie folgt definieren:

Definition 1. *Eine **Multimenge** $\dot{\mathcal{X}}$ über einer Menge $\mathcal{D}$ ist eine Abbildung aus $\mathcal{D}$ in die Menge der natürlichen Zahlen $\mathbf{N}$. Sie ist **endlich**, wenn nur endlich viele Elemente aus $\mathcal{D}$ auf von Null verschiedene Zahlen abgebildet werden. Die Menge aller endlichen Multimengen über einer Menge $\mathcal{D}$ wird geschrieben als $\mathcal{M}_{fin}(\mathcal{X})$. In Analogie zu den klassischen Mengen schreibt man $e \in_k \dot{\mathcal{X}}$ (e ist k-faches Element der Multimenge), wenn $\dot{\mathcal{X}}$ das Element e auf k abbildet. Für $e \in_0 \dot{\mathcal{X}}$ wird auch $e \notin \dot{\mathcal{X}}$ geschrieben.*

Notiert werden endliche Multimengen durch Aufzählung ihrer Elemente in den Klammern $\dot{\{}$ und $\dot{\}}$. Jedes Element, dass k-fach in der Multimenge vorkommt, muss dabei genau k mal in der Aufzählung vorkommen. Die Reihenfolge der Elemente in der Aufzählung ist nicht relevant. Die leere Multimenge wird als $\dot{\emptyset}$ oder $\dot{\{\}}$ bezeichnet.

Für zwei Multimengen $\dot{\mathcal{X}}, \dot{\mathcal{Y}} \in \mathbf{N}^{\mathcal{D}}$ *sind die Vereinigung* $\dot{\mathcal{X}} \dot{\cup} \dot{\mathcal{Y}}$, *die Differenz* $\dot{\mathcal{X}} \dot{\setminus} \dot{\mathcal{Y}}$ *sowie die Teilmengenrelation* $\dot{\mathcal{X}} \dot{\subseteq} \dot{\mathcal{Y}}$ *wie folgt definiert:*

$$(\forall\, e \in \mathcal{D})\ e \in_m \dot{\mathcal{X}} \ \wedge\ e \in_n \dot{\mathcal{Y}} \ \rightarrow\ e \in_{m+n} (\dot{\mathcal{X}} \dot{\cup} \dot{\mathcal{Y}}).$$

$$(\forall\, e \in \mathcal{D})\ e \in_m \dot{\mathcal{X}} \ \wedge\ e \in_n \dot{\mathcal{Y}} \ \rightarrow\ e \in_{\min(m,n)} (\dot{\mathcal{X}} \dot{\cap} \dot{\mathcal{Y}}).$$

$$(\forall\, e \in \mathcal{D})\ e \in_m \dot{\mathcal{X}} \ \wedge\ e \in_n \dot{\mathcal{Y}} \ \rightarrow\ e \in_k (\dot{\mathcal{X}} \dot{\setminus} \dot{\mathcal{Y}}) \quad \textit{mit} \quad k = \begin{cases} m-n & \textit{für} \quad m > n \\ 0 & \textit{sonst.} \end{cases}$$

$$(\dot{\mathcal{X}} \dot{\subseteq} \dot{\mathcal{Y}}) \ \leftrightarrow\ (\forall\, e \in \mathcal{D})\ \big(e \in_m \dot{\mathcal{X}} \ \wedge\ e \in_n \dot{\mathcal{Y}} \ \rightarrow\ m \leq n\big).$$

Beispiel 2. *Für die Multimengen* $\dot{\mathcal{X}} = \dot{\{}a, a, b\dot{\}}$ *und* $\dot{\mathcal{Y}} = \dot{\{}b, a\dot{\}}$ *gelten die Beziehungen* $a \in_2 \dot{\mathcal{X}}$, $b \in_1 \dot{\mathcal{X}}$, $\dot{\mathcal{Y}} \dot{\subseteq} \dot{\mathcal{X}}$, $\dot{\mathcal{X}} \dot{\cup} \dot{\mathcal{Y}} = \dot{\{}a, a, a, b, b\dot{\}}$ *sowie* $\dot{\mathcal{X}} \dot{\setminus} \dot{\mathcal{Y}} = \dot{\{}a\dot{\}}$.

Klassische Mengen können nun als Spezialfall von Multimengen aufgefasst werden, in dem jedes Element maximal einmal in der Menge vorkommt. Dies ist allerdings keine Einbettung, da die Resultate der Mengenoperationen sich von den Resultaten der korrespondierenden Multimengenoperationen unterscheiden: $\{a\} \cup \{a\} = \{a\}$, aber $\dot{\{}a\dot{\}} \dot{\cup} \dot{\{}a\dot{\}} = \dot{\{}a, a\dot{\}}$. Unter bestimmten Bedingungen stimmen sie allerdings überein: So entspricht z.B. das Resultat der Vereinigung zweier Multimengen der Vereinigung der entsprechenden Mengen, falls die Multimengen disjunkt sind, d.h. $(\forall\, e \in \mathcal{D})\ e \notin \dot{\mathcal{X}} \ \vee\ e \notin \dot{\mathcal{Y}}$.

2.6. Weitere Definitionen

Die Menge aller Teilmengen einer Menge $\mathcal{A}$ wird als $\mathcal{P}(\mathcal{A})$ bezeichnet (Produktmenge).

Für zwei gleichlange Vektoren $\vec{x} = \langle x_1, \ldots, x_n \rangle$ und $\vec{y} = \langle y_1, \ldots, y_n \rangle$ bedeutet die Kurzschreibweise $\vec{x} = \vec{y}$ die komponentenweise Gleichheit: $x_1 = y_1 \ \wedge\ \ldots\ \wedge\ x_n = y_n$, weiterhin sei $f(\vec{x})$ eine Abkürzung für $f(x_1, \ldots, x_n)$.

Sei $\mathcal{F} \cup \{\phi\}$ eine Menge von prädikatenlogischen Formeln. In Anlehnung an die in der logischen Programmierung verwendeten Begriffe, definieren wir eine **Antwortsubstitution** als eine Substitution σ mit $\operatorname{dom}\sigma \subseteq \operatorname{var}\phi$. Eine **Anfrage** $\mathcal{F} \models^{?} \phi$ sei das Problem, welche Antwortsubstitutionen für ϕ **korrekt** sind, d.h. für welche Antwortsubstitutionen $\mathcal{F} \models (\forall)(\phi\sigma)$ gilt.

3. Binäre Entscheidungsdiagramme (BDDs)

In diesem Kapitel wird dem Leser ein Einblick in die Darstellung aussagenlogischer Formeln durch Binäre Entscheidungsdiagramme (engl. Binary Decision Diagrams, BDDs) gegeben, sowie exemplarisch grundlegende Operationen und Algorithmen beschrieben. Die Darstellung erfolgt weitgehend wie in [2]. Für eine tiefergehende Behandlung sowie die Beweise der zitierten Aussagen wird auf die Fachliteratur [76, 2, 13] verwiesen.

Obwohl Binäre Entscheidungsdiagramme (BDDs) recht alt sind [58, 1], wurden sie erst durch die Arbeit von Bryant [13] für viele Forscher interessant. Sie gestatten eine häufig sehr kompakte Darstellung von Formeln in Aussagenlogik und daher auch von beliebigen endlichen Mengen und Relationen in durch ihre charakteristische Funktion kodierter Form. Daher haben BDDs Verwendung bei vielen Aufgaben in Bereichen wie computerunterstütztes Design und Verifikation gefunden, in denen wie bei Planungsproblemen große Zustandsräume durchsucht werden müssen. Dabei wurden zum Teil Aufgaben gelöst, die zur Zeit noch außer Reichweite anderer Methoden liegen [15]. Dank der Kompaktheit von BDDs und der Effizienz der Algorithmen konnten einige Systeme mit mehr als 10^{120} Zuständen analysiert werden [16].

3.1. Die Grundidee

Die Idee von BDDs beruht auf der Shannon Zerlegung für aussagenlogische Formeln. Diese ermöglicht eine Darstellung von Formeln mit Hilfe des Operators ite [1] durch ihre **Kofaktoren** $\phi_p \underset{\text{Def}}{\equiv} \phi\{p/\mathbf{1}\}$ und $\phi_{\overline{p}} \underset{\text{Def}}{\equiv} \phi\{p/\mathbf{0}\}$ von ϕ bezüglich p und $\overline{p}$. Eine Formel lässt sich nun durch ihren Kofaktoren entsprechend der folgenden als **Shannon-Zerlegung**[2] bekannten Identität darstellen:

$$\phi = \mathrm{ite}(p, \phi_p, \phi_{\overline{p}}) \ . \tag{3.1}$$

Man beachte, dass die Kofaktoren $\phi_{\overline{p}}$ und ϕ_p die Variable p nicht mehr enthalten, so dass (falls p tatsächlich in ϕ enthalten ist) die Anzahl der Variablen in den Teilformeln echt kleiner

[1] Siehe Seite 9.

[2] Diese wurde allerdings von Boole eingeführt [11].

ist als in ϕ. Durch weitere Zerlegung der Kofaktoren mit Hilfe der Shannon-Zerlegung bis zum Erreichen der konstanten Kofaktoren $\mathbf{1}$ und $\mathbf{0}$ lässt sich eine Darstellung der Formel ausschließlich mit Hilfe des ite Operators erzielen. Zum Beispiel ist:

$$\begin{aligned}\phi &= ac \vee \overline{b}c \\ &= \mathrm{ite}(a, c \vee \overline{b}c, \overline{b}c) \\ &= \mathrm{ite}(a, \mathrm{ite}(b, c, c), \mathrm{ite}(b, \mathbf{0}, c)) \\ &= \mathrm{ite}(a, \mathrm{ite}(b, \mathrm{ite}(c, \mathbf{1}, \mathbf{0}), \mathrm{ite}(c, \mathbf{1}, \mathbf{0})), \mathrm{ite}(b, \mathrm{ite}(c, \mathbf{0}, \mathbf{0}), \mathrm{ite}(c, \mathbf{1}, \mathbf{0}))) \ .\end{aligned}$$

In einem derartigen Ausdruck (einer sogenannten **„if-then-else“-Normalform**) sind alle Tests aussagenlogische Variablen. Diese sind dabei der einzige Ort in der Formel, an denen Variablen vorkommen können. In einem Diagramm lässt sich dieser Term wie in Abbildung 3.1 darstellen.

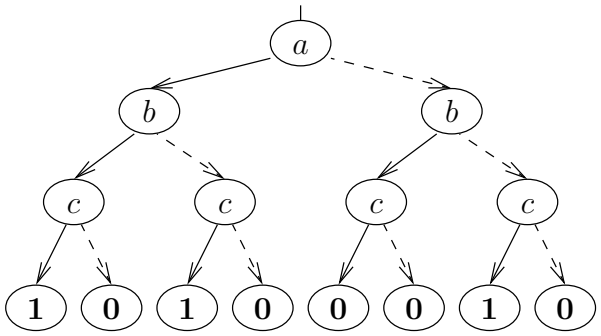


Abbildung 3.1.: Ein BDD für $ac \vee \overline{b}c$

Dabei repräsentiert jeder Knoten im Graphen ein Auftreten von ite in der Normalform. Die Markierung des Knotens entspricht der getesteten Variablen, die durchgezogene Kante dem Then -Teil und die gestrichelte Kante dem Else -Teil. Die mit $\mathbf{1}$ und $\mathbf{0}$ markierten Knoten stellen die entsprechenden Wahrheitswerte dar. Für eine gegebene Variablenbelegung kann man entscheiden, ob diese die Formel erfüllt, indem man den Graphen von der Wurzel aus durchläuft, und an jedem Knoten entsprechend den Wert der Variablen, die den Knoten markiert, entlang der durchgezogenen Kante (bei Wahrheitswert $\mathbf{1}$) oder der gestrichelten Kante (bei Wahrheitswert $\mathbf{0}$) weitergeht, bis man an einem der Blätter $\mathbf{1}$ oder $\mathbf{0}$ ankommt. Z.B. führt die Variablenbelegung $\{a/\mathbf{1}, b/\mathbf{0}, c/\mathbf{1}\}$ zu einem Knoten $\mathbf{1}$, d.h. diese Belegung erfüllt die Formel ϕ.

Definition 3 ([2]). *Ein **binäres Entscheidungsdiagramm** (engl. „Binary Decision Diagram“, **BDD**[3]) ist ein gerichteter azyklischer Graph mit einer Wurzel, der*

[3] In den anderen Kapiteln dieser Arbeit werden für BDDs noch weitere Eigenschaften gemäß Definition 4 (ROBDDs) Seite 20 gefordert, die aus didaktischen Gründen erst später eingeführt werden.

- *ein oder mehrere* ***Blätter*** *enthält, die mit* $\mathbf{1}$ *oder* $\mathbf{0}$ *markiert sind.*
- *einer Menge* $\mathcal{N}$ *von* ***inneren Knoten*** v*, die mit aussagenlogischen Variablen* $\mathrm{Var}(v)$ *markiert sind. Jeder innere Knoten hat genau zwei ausgehende Kanten, die* Then*– und die* Else*–Kante. Für einen Knoten* v *bezeichnen* $\mathrm{Then}(v)$ *und* $\mathrm{Else}(v)$ *die Knoten, zu denen diese Kanten führen. Diese Knoten werden auch als* Then*– bzw.* Else*– Nachfolger bezeichnet.*

Die Then– bzw. die Else–Kanten werden in Diagrammen als gestrichelte bzw. durchgezogene Kanten dargestellt.

Ein baumförmiges BDD wie in Abbildung 3.1 kann noch viel Redundanz in Form von isomorphen Teilbäumen enthalten. Wenn man in einem ersten Schritt eine Identifikation derartiger Teilbäume durchführt, wird noch eine zweite Form von Redundanz offensichtlich: Die Else und Then Kanten von einigen Knoten führen zum selben Knoten (Abbildung 3.2 links). Damit sind die beiden Kofaktoren ϕ_p und $\phi_{\overline{p}}$ in dem Knoten entsprechenden ite identisch, so dass wegen der Identität $\psi \equiv \mathrm{ite}(p, \psi, \psi)$ der entsprechende Knoten entfernt werden kann (Abbildung 3.2 rechts).

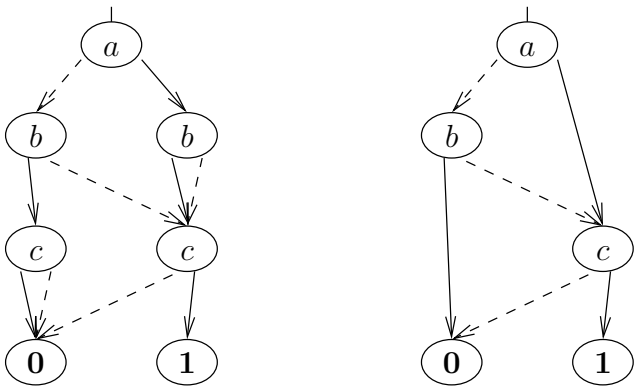


Abbildung 3.2.: BDDs für $ac \vee \overline{b}c$: Identifikation von isomorphen Teilbäumen (links), nachfolgende Elimination von redundanten Knoten (rechts)

Wenn man zusätzlich fordert, dass die Anordnung der Variablen in jedem Pfad des BDD einer vorgegebenen linearen Ordnung $<$ der aussagenlogischen Variablen entspricht (im gegebenen Beispiel wurde die Ordnung $a < b < c$ benutzt), hat diese Redundanzreduktion neben einer Reduktion der Repräsentationsgröße noch den wichtigen Nebeneffekt, dass die Darstellung der durch aussagenlogische Formel repräsentierten booleschen Funktion durch ein BDD sogar (bis auf Isomorphie) eindeutig ist. Dazu müssen wir zunächst die erwähnten Einschränkungen an die BDDs formal fassen und definieren, welche boolesche Funktion durch das BDD repräsentiert wird.

Definition 4. *Ein BDD ist* ***geordnet*** *(engl. ordered,* ***OBDD****), wenn auf allen Pfaden des BDD die Reihenfolge des Auftretens von Variablen einer gegebenen linearen Ordnung* $<$ *entspricht, d.h. für alle inneren Knoten* v *gilt* $v < \mathrm{Then}(v)$ *und* $v < \mathrm{Else}(v)$ *falls* $\mathrm{Then}(v) \notin \{\mathbf{1}, \mathbf{0}\}$ *bzw.* $\mathrm{Else}(v) \notin \{\mathbf{1}, \mathbf{0}\}$ *.*

Ein OBDD ist ***reduziert*** *(engl. reduced,* ***ROBDD****), wenn die drei folgenden Eigenschaften zutreffen:*

1. *Es gibt jeweils höchstens ein Blatt mit der Markierung* $\mathbf{1}$ *bzw.* $\mathbf{0}$ *.*
2. *Keine zwei unterschiedlichen inneren Knoten* u *und* v *haben die gleichen Markierungen und die gleichen* Else *– und* Then *–Nachfolger, d.h. für alle Knoten* u *und* v *des ROBDD gilt*
   $\mathrm{Var}(u) = \mathrm{Var}(v) \ \wedge \ \mathrm{Else}(u) = \mathrm{Else}(v) \ \wedge \ \mathrm{Then}(u) = \mathrm{Then}(v) \quad$ *gdw.* $\quad u = v$ *.*
3. *Kein innerer Knoten hat* v *identische* Else *– und* Then *–Nachfolger, d.h. für alle Knoten* v *des ROBDD gilt* $\mathrm{Else}(v) \neq \mathrm{Then}(v)$ *.*

Definition 5. *Ein BDD mit Wurzelknoten* v *repräsentiert die folgende aussagenlogische Formel* t^v *:*

$$t^v = \begin{cases} \mathbf{1} & \textit{wenn} \quad v = \mathbf{1} \\ \mathbf{0} & \textit{wenn} \quad v = \mathbf{0} \\ \mathrm{ite}(\mathrm{Var}(v), t^{\mathrm{Then}(v)}, t^{\mathrm{Else}(v)}) & \textit{sonst.} \end{cases}$$

Ein BDD mit der Variablenordnung $x_1 < x_2 < \cdots < x_n$ *und dem Wurzelknoten* v *repräsentiert die boolesche Funktion* f^v *, die* $(b_1, \ldots, b_n) \in \mathbb{B}^n$ *auf den Wahrheitswert von* $t^v\{x_1/b_1, x_2/b_2, \ldots, x_n/b_n\}$ *abbildet.*

Damit ergibt sich nun, wie Bryant nachwies, eine eindeutige Zuordnung:

Theorem 6 (Kanonische Repräsentation, [13])**.** *Für jede boolesche Funktion* $f : \mathbb{B}^n \to \mathbb{B}$ *existiert genau ein ROBDD mit Wurzelknoten* v *und mit einer Variablenordnung* $x_1, \ldots, x_n$ *, so dass das BDD die Funktion* $f^v = f(x_1, \ldots, x_n)$ *repräsentiert.*

Dies ist die theoretische Basis für viele effiziente Algorithmen, die eine große Vielfalt von logischen Operationen auf ROBDDs ausführen. Im Gegensatz zu allgemeinen aussagenlogischen Formeln oder deren konjunktiven und disjunktiven Normalformen ergibt sich zum Beispiel, dass sowohl ein Tautologietest als auch ein Unerfüllbarkeitstest in konstanter Zeit für ein ROBDD durchgeführt werden kann, da unabhängig von der Variablenordnung und der Anzahl der Variablen genau das BDD aus einem Blatt $\mathbf{1}$ bzw. $\mathbf{0}$ der konstant wahren bzw. konstant falschen Funktion entspricht. Bei geeigneter Darstellung der BDDs ist sogar der Äquivalenztest zweier BDDs in konstanter Zeit ausführbar. Logische Operationen, die auf BDDs arbeiten, sind Gegenstand des nächsten Abschnitts.

Man beachte, dass die Variablenordnung einen großen Einfluss auf die Größe des BDD hat. Die BDD-Darstellung der Formel $\phi = (p \oplus q) \vee (r \oplus s) \vee (t \oplus u)$ hat, abhängig von der Variablenordnung, zwischen 11 und 23 Knoten (Abbildung 3.3). Allgemein kann der Unterschied der Knotenzahl zwischen der günstigsten und der ungünstigsten Variablenordnung exponentiell sein: Z.B. hat die Darstellung von

$$\phi_n = \bigvee_{i=1\ldots n} (p_i \oplus q_i)$$

je nach Variablenordnung zwischen $3n + 2$ und $3 * 2^n - 1$ Knoten. Leider ist das Problem, die Variablenordnung mit geringster Knotenanzahl zu bestimmen, co-NP-vollständig [13], so dass man dabei oft auf heuristische Algorithmen, die eine gegebene Variablenordnung zu verbessern suchen, angewiesen ist.

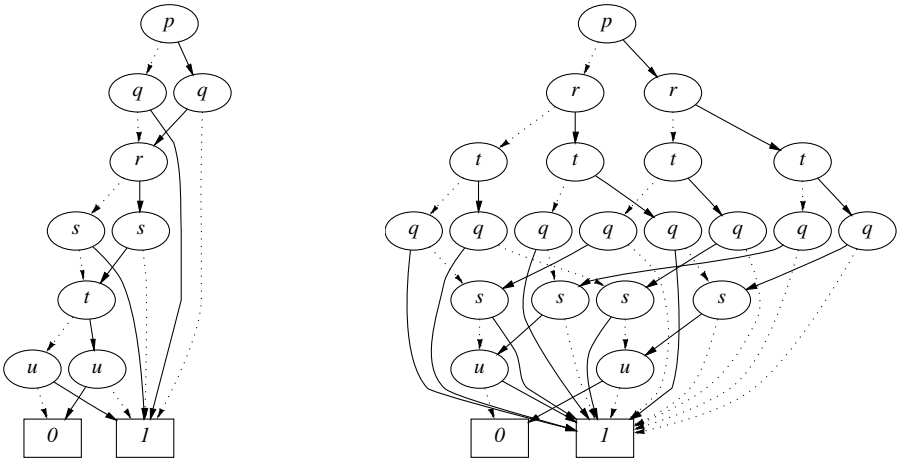


Abbildung 3.3.: BDDs für $\phi = (p \oplus q) \vee (r \oplus s) \vee (t \oplus u)$ mit Variablenordnungen $p < q < r < s < t < u$ (links) sowie $p < r < t < q < s < u$ (rechts).

Seit der originalen Idee wurden viele Verfeinerungen an der BDD-Struktur vorgenommen [14]. So wird ein einzelner Graph mit mehreren Wurzeln genutzt, um alle für eine Applikation benötigten BDDs gemeinsam darzustellen, und es wurden Markierungen der Kanten eingeführt, um Negation einfacher darzustellen. Dies ändert allerdings nichts an der grundlegenden Funktionalität. Weiterhin wurden Erweiterungen vorgestellt, die Funktionen $\mathbb{B}^n \to \mathcal{F}$ für beliebige endliche Mengen $\mathcal{F}$ oder $\mathbb{B}^n \to \mathbb{N}$ darstellen können. Für diese Arbeit können wir uns aber auf die genannte Form von BDDs beschränken.

3.2. Konstruktion und Manipulation von BDDs

Für die Darstellung von booleschen Funktionen ist es meist von großem Vorteil, geordnete und reduzierte BDDs zu verwenden. Daher haben eigentlich nur ROBDDs praktische Bedeutung.

Daher wird, wie bei vielen anderen Autoren auch, im Rest dieser Arbeit der Begriff BDD synonym zu ROBDD verwendet.

Im folgenden werden die Operationen von BDDs dargestellt, die in dieser Arbeit verwendet werden. Um dem Leser ein Gefühl für die Funktionsweise von BDDs zu geben, wird zunächst exemplarisch ein Algorithmus zur Konstruktion von BDDs dargestellt. Dieser berechnet die BDD-Darstellung der Konjunktion zweier boolescher Funktionen aus der BDD-Darstellung der beiden Operanden. Für eine tiefere Behandlung des Themas verweisen wir auf die Literatur [76, 2, 14].

Die Basis für logische Operationen in BDD-Repräsentation ist die bereits besprochene Shannon-Zerlegung (3.1) in Zusammenhang mit der Semantik der BDDs nach Definition 5. Dabei wird folgende Eigenschaft der Kofaktoren ausgenutzt:

$$\begin{aligned} (\phi \wedge \psi)_p &\equiv \phi_p \wedge \psi_p && \text{und} \\ (\phi \wedge \psi)_{\overline{p}} &\equiv \phi_{\overline{p}} \wedge \psi_{\overline{p}} \ , && \text{und damit} \\ \phi \wedge \psi &\equiv \mathrm{ite}(p, \phi_p \wedge \psi_p, \phi_{\overline{p}} \wedge \psi_{\overline{p}}) \ . && (3.2) \end{aligned}$$

Seien nun zwei Formeln ϕ und ψ in if-then-else Normalform gegeben. Nehmen wir weiterhin an, dass ϕ und ψ einer Variablenordnung folgen, d.h. bei jedem Teilterm $\mathrm{ite}(p, \tau_p, \tau_{\overline{p}})$ der Normalformen dürfen in den Termen τ_p und $\tau_{\overline{p}}$ nur Variablen vorkommen, die bezüglich dieser Ordnung größer als p sind. (Dies ist für die in Definition 5 auf Seite 20 gegebene Semantik von BDDs erfüllt.) Dann lassen sich aber die Kofaktoren wie folgt berechnen:

$$\begin{aligned} \mathrm{ite}(p, \phi_p, \phi_{\overline{p}})_q &= \mathrm{ite}(p, \phi_p, \phi_{\overline{p}}) \quad \text{für } q < p \ , \\ \mathrm{ite}(p, \phi_p, \phi_{\overline{p}})_{\overline{q}} &= \mathrm{ite}(p, \phi_p, \phi_{\overline{p}}) \quad \text{für } q < p \ , \\ \mathrm{ite}(p, \phi_p, \phi_{\overline{p}})_p &= \phi_p \ , \\ \mathrm{ite}(p, \phi_p, \phi_{\overline{p}})_{\overline{p}} &= \phi_{\overline{p}} \ . \end{aligned} \tag{3.3}$$

Die Einhaltung der Variablenordnung in ϕ und ψ garantiert dabei die Gültigkeit der zwei obersten Zeilen von (3.3): Wegen $q < p$ kann p nicht in ϕ_p und $\phi_{\overline{p}}$ vorkommen.

Weiterhin sind noch folgende Spezialfälle für die Konstruktion des Algorithmus relevant:

$$\begin{aligned} \phi \wedge \mathbf{1} &= \phi & \qquad \phi \wedge \mathbf{0} &= \mathbf{0} \\ \mathbf{1} \wedge \phi &= \phi & \qquad \mathbf{0} \wedge \phi &= \mathbf{0} \end{aligned} \tag{3.4}$$

Für zwei in if-then-else Normalform gegebene Operanden, die einer gemeinsamen Variablenordnung folgen, lässt sich nun durch rekursive Anwendung von (3.2) in Verbindung mit (3.3) und den Spezialfällen für $\mathrm{ite}(\_, \_, \_)$ für konstante Argumente eine if-then-else Normalform der Konjunktion der Operanden bestimmen. Dies ist die Grundidee der im folgenden beschriebenen Algorithmus zur Bestimmung der Konjunktion von BDDs.

Als einen ersten Schritt betrachten wir den rekursiven Algorithmus ANDITE, der zu zwei BDDs eine if-then-else Form der Konjunktion ihrer Semantiken nach Definition 5 berechnet.

Zu dieser kann dann leicht ein entsprechendes BDD angegeben werden, dass zwar geordnet, aber noch nicht reduziert ist. Wie eine automatische Reduktion in den Algorithmus einbezogen wird, diskutieren wir später. Der Leser ist eingeladen nachzuvollziehen, wie sich die einzelnen Fallunterscheidungen des Algorithmus aus (3.2), (3.3) und (3.4) ergeben.

Algorithmus 7 (**ANDITE**(t^u, t^v))**.** *Seien* u *und* v *die Wurzelknoten von geordneten BDDs mit der gleichen Variablenordnung* $<$*, und* t^u *und* t^v *die Semantik von* u *und* v *gemäß Definition 5.* ANDITE(t^u, t^v) *ergibt sich nun gemäß der folgenden Fallunterscheidung:*

$t^u = \mathbf{1}$ oder $t^v = \mathbf{1}$ *:*

$$\text{ANDITE}(\mathbf{1}, t^v) = t^v$$
$$\text{ANDITE}(t^u, \mathbf{1}) = t^u$$

$t^u = \mathbf{0}$ oder $t^v = \mathbf{0}$ *:*

$$\text{ANDITE}(\mathbf{0}, t^v) = \mathbf{0}$$
$$\text{ANDITE}(t^u, \mathbf{0}) = \mathbf{0}$$

$t^u = \text{ite}(\text{Var}(u), t^{\text{Then}(u)}, t^{\text{Else}(u)})$ und $t^v = \text{ite}(\text{Var}(v), t^{\text{Then}(v)}, t^{\text{Else}(v)})$
mit $\text{Var}(u) = \text{Var}(v)$ *:*

$$\text{ANDITE}(\text{ite}(\text{Var}(u), t^{\text{Then}(u)}, t^{\text{Else}(u)}), \text{ite}(\text{Var}(u), t^{\text{Then}(v)}, t^{\text{Else}(v)})) = \text{ite}(\text{Var}(u), \text{ANDITE}(t^{\text{Then}(u)}, t^{\text{Then}(v)}), \text{ANDITE}(t^{\text{Else}(u)}, t^{\text{Else}(v)}))$$

$t^u = \text{ite}(\text{Var}(u), t^{\text{Then}(u)}, t^{\text{Else}(u)})$ und $t^v = \text{ite}(\text{Var}(v), t^{\text{Then}(v)}, t^{\text{Else}(v)})$
mit $\text{Var}(u) < \text{Var}(v)$ *:*

$$\text{ANDITE}(\text{ite}(\text{Var}(u), t^{\text{Then}(u)}, t^{\text{Else}(u)}), t^v) = \text{ite}(\text{Var}(u), \text{ANDITE}(t^{\text{Then}(u)}, t^v), \text{ANDITE}(t^{\text{Else}(u)}, t^v))$$

$t^u = \text{ite}(\text{Var}(u), t^{\text{Then}(u)}, t^{\text{Else}(u)})$ und $t^v = \text{ite}(\text{Var}(v), t^{\text{Then}(v)}, t^{\text{Else}(v)})$
mit $\text{Var}(v) < \text{Var}(u)$ *:*

$$\text{ANDITE}(t^u, \text{ite}(\text{Var}(v), t^{\text{Then}(v)}, t^{\text{Else}(v)})) = \text{ite}(\text{Var}(v), \text{ANDITE}(t^u, t^{\text{Then}(v)}), \text{ANDITE}(t^u, t^{\text{Else}(v)}))\ .$$

Wie sich leicht durch Induktion über die Summe der Größen von t^u und t^v beweisen lässt, terminiert der Algorithmus und liefert eine if-then-else Normalform. Man beachte, dass die Umformungen im Algorithmus ANDITE bereits so gewählt sind, dass im Resultat die Variablen mit zunehmender Schachtelungstiefe der $\text{ite}(\_,\_,\_)$ größer bezüglich $<$ werden, sofern dies bei den Argumenten ebenfalls erfüllt ist. Dies ist aber bei der Semantik von BDDs gemäß Definition 5 automatisch der Fall, und ist eine Voraussetzung dafür, dass entsprechend dem Algorithmus leicht ein BDD konstruiert werden kann.

Beispiel 8. *Betrachten wir die BDDs für* $a \vee \neg b$ *und* $\neg b \wedge c$ *bezüglich der Variablenordnung* $a < b < c$ *(Abbildung 3.4).*

Die Semantik dieser BDDs ergibt sich wie folgt:

$$t^u = \mathrm{ite}(a, \mathbf{1}, \mathrm{ite}(b, \mathbf{0}, \mathbf{1}))$$
$$t^v = \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0})) \ .$$

Wenden wir nun den Algorithmus AndIte *auf diese BDDs an:*

$$\begin{aligned}
&\textsc{AndIte}(\mathrm{ite}(a, \mathbf{1}, \mathrm{ite}(b, \mathbf{0}, \mathbf{1})),\ \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0})))\\
&= \mathrm{ite}(a, \textsc{AndIte}(\mathbf{1}, \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0}))), \textsc{AndIte}(\mathrm{ite}(b, \mathbf{0}, \mathbf{1}), \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0}))))\\
&= \mathrm{ite}(a, \underline{\mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0}))}, \textsc{AndIte}(\mathrm{ite}(b, \mathbf{0}, \mathbf{1}), \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0}))))\\
&= \mathrm{ite}(a, \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0})), \underline{\mathrm{ite}(b, \textsc{AndIte}(\mathbf{0}, \mathbf{0}), \textsc{AndIte}(\mathbf{1}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0})))})\\
&= \mathrm{ite}(a, \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0})), \mathrm{ite}(b, \underline{\mathbf{0}}, \textsc{AndIte}(\mathbf{1}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0}))))\\
&= \mathrm{ite}(a, \mathrm{ite}(b, \mathbf{0}, \mathrm{ite}(c, \mathbf{1}, \mathbf{0})), \mathrm{ite}(b, \mathbf{0}, \underline{\mathrm{ite}(c, \mathbf{1}, \mathbf{0})})) \ .
\end{aligned}$$

(Zur besseren Übersicht sind die Stellen, an denen die Gleichungen des Algorithmus angewandt wurden, markiert.)

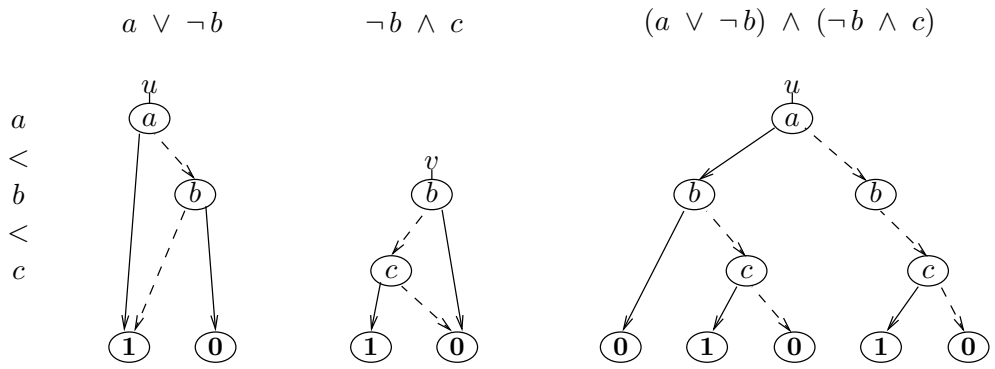


Abbildung 3.4.: Die BDDs aus Beispiel 8, sowie die (unreduzierte) BDD-Darstellung von Konjunktion berechnet mit Hilfe von Algorithmus AndIte .

Es ist nun zwar leicht möglich, ein nicht reduziertes BDD entsprechend dem Resultat des Algorithmus zu bestimmen; zur Konstruktion eines reduzierten BDDs ist jedoch noch etwas mehr Überlegung notwendig. Da ein unreduziertes BDD exponentiell größer sein kann, als ein reduziertes BDD,[4] wäre es ungeschickt, zuerst ein unreduziertes BDD zu erzeugen, und dieses dann zu reduzieren. Es ist also besser dafür zu sorgen, dass sofort im Algorithmus

[4] Man vergleiche dazu z.B. reduzierte und unreduzierte BDDs für $\bigwedge_{i=1\ldots n}(p_i \oplus q_i)$ mit Variablenordnung $p_1 < q_1 < p_2 < q_2 < \cdots < p_n < q_n$ mit linearer bzw. exponentieller Größe bezüglich n .

ein BDD generiert wird, das die Bedingungen von Definition 4 auf Seite 20 erfüllt. Dies geschieht, indem diese Bedingungen im Algorithmus zur Erzeugung neuer BDD-Knoten fest verankert werden. Wir beschreiben dafür nachfolgend einen Algorithmus $\mathrm{MK}(x, u, v)$, der einen BDD-Knoten mit der Semantik $\mathrm{ite}((, x, ,)t^u, t^v)$ zurückgibt. Bedingung 2 der Definition 4 wird erfüllt, indem vorhergehende Aufrufe von MK mit ihren Resultaten in einer Tabelle gespeichert werden, und bei identischen Aufrufen der gleiche Knoten zurückgegeben wird.[5] Bedingung 3 wird ebenfalls berücksichtigt:

Algorithmus 9 (MK(x, u, v)). *Sei x eine aussagenlogische Variable, sowie u und v Wurzelknoten von ROBDDs. Sei weiterhin $x < \mathrm{Var}(u)$ bzw. $x < \mathrm{Var}(v)$ falls u bzw. v innere Knoten sind.*

Wenn $u = v$ gib u zurück.

Wenn ein Aufruf $\mathrm{MK}(x, u, v)$ bereits in der Aufruftabelle enthalten ist, dann gib das dort eingetragene Resultat zurück.

Sonst erzeuge einen neuen Knoten w mit $\mathrm{Var}(w) = x$, $\mathrm{Then}(w) = u$ und $\mathrm{Else}(w) = v$, trage w als Resultat von $\mathrm{MK}(x, u, v)$ in die Aufruftabelle ein und gib w zurück.

Wenn nun neue BDD-Knoten ausschließlich mit Hilfe von MK konstruiert werden, dann sind die erzeugten BDDs damit automatisch reduziert und geordnet. MK erfüllt dazu noch eine weitere Bedingung: Wenn zwei BDDs gleiche Funktionen repräsentieren, dann gibt MK stets den gleichen Knoten zurück. Damit sind Tautologie-, Unerfüllbarkeits- und Äquivalenztests von BDDs in konstanter Zeit durchführbar.

Unter Verwendung von MK können wir nun einen ANDITE entsprechenden Algorithmus AND aufstellen, der die Konjunktion in BDD-Darstellung realisiert. Da MK so gestaltet ist, dass es bei mehrfachem Aufruf mit gleichen Parametern stets das gleiche Resultat liefert, gilt dies auch für AND, so dass aus Effizienzgründen hier ebenfalls eine Aufruftabelle verwendet wird. (Die hier verwendete Idee, die Parameter sowie Resultate bereits erfolgter Aufrufe des Algorithmus zu speichern, um für nachfolgende Aufrufe mit gleichen Parametern den Algorithmus nicht nocheinmal abarbeiten zu müssen, wird oft als **dynamische Programmierung** bezeichnet.)

Algorithmus 10 (AND(u, v)). *Seien u und v Wurzelknoten von ROBDDs. Wenn der Aufruf bereits $\mathrm{AND}(u, v)$ in der Aufruftabelle[6] eingetragen ist, dann gebe das dort eingetragene*

[5] In einer Implementation wird eine solche Tabelle üblicherweise als Hashtabelle realisiert.

[6] In einer Implementation wird hierfür üblicherweise ebenfalls eine Hashtabelle verwendet. Im Unterschied zu Algorithmus 9 (MK) ist die Aufruftabelle hier für die Funktion nicht unbedingt notwendig, sondern nur zur Effizienzsteigerung erforderlich. Damit können z.B. längere Zeit nicht verwendete Einträge entfernt, oder zur Speicherplatzersparnis periodisch gelöscht werden.

Resultat zurück. Sonst ermittle das Resultat wie folgt und trage es in die Aufruftabelle ein:

$$\textsc{And}(u,v) = \begin{cases} v & \textit{für } u = \mathbf{1}, \\ u & \textit{für } v = \mathbf{1}, \\ \mathbf{MK}(\mathrm{Var}(u),\ \textsc{And}(\mathrm{Then}(u),\mathrm{Then}(v)),\ \textsc{And}(\mathrm{Else}(u),\mathrm{Else}(v))) & \textit{für } \mathrm{Var}(u) = \mathrm{Var}(v), \\ \mathbf{MK}(\mathrm{Var}(u),\ \textsc{And}(\mathrm{Then}(u),v),\ \textsc{And}(\mathrm{Else}(u),v)) & \textit{für } \mathrm{Var}(u) < \mathrm{Var}(v), \\ \mathbf{MK}(\mathrm{Var}(v),\ \textsc{And}(u,\mathrm{Then}(v)),\ \textsc{And}(u,\mathrm{Else}(v))) & \textit{für } \mathrm{Var}(v) < \mathrm{Var}(u). \end{cases} \quad (3.5)$$

Man beachte, dass die Aufruftabelle für diesen Algorithmus eine starke Komplexitätsverringerung mit sich bringt. Ohne Verwendung der Aufruftabelle kann AND für jeden Knoten u ebensooft aufgerufen werden, wie dieser von der Wurzel aus erreichbar ist. Dies kann exponentiell oft in der Anzahl der Knoten des BDDs sein.[7] Wenn dagegen eine Aufruftabelle verwendet wird, wird der Algorithmus polynomiell in der Anzahl der BDD-Knoten: Jeder Algorithmusschritt (3.5) kann (zuzüglich der rekursiven Aufrufe) in konstanter Zeit ausgeführt werden; durch die Aufruftabelle wird der Algorithmusschritt (3.5) aber höchstens einmal für jede Kombination von Knoten u und v aus Knoten beider verarbeiteter BDDs ausgeführt, so dass die Komplexität des Gesamtalgorithmus maximal dem Produkt der Knotenanzahlen beider bearbeiteter BDDs entspricht.

Auf analoge Weise kann aus der Shannon-Zerlegung für viele weitere logische Operationen eine entsprechende BDD-Implementation abgeleitet werden. Insbesondere ist dies für die Negation und alle binären logischen Operationen, sowie für die aussagenlogische Quantifizierung möglich. Zudem wird im weiteren die Semantik einiger in dieser Arbeit verwendeter Operationen beschrieben, die z.T. nur im Kontext von BDDs sinnvoll sind.

Support(ϕ) liefert das sogenannte **support set** („Stützmenge“) des BDDs für ϕ, d.h. alle aussagenlogischen Variablen, die die Knoten markieren. Dies entspricht der Menge der aussagenlogischen Variablen, von denen der Wert der vom BDD dargestellten booleschen Funktionen abhängt.

AnySat(ϕ,ψ) liefert eine beliebige Variablenbelegung, für die das BDD (bzw. die dargestellte boolesche Funktion) erfüllt ist.

Simplify(δ,ϕ) versucht das BDD für ϕ zu vereinfachen. Dieser Algorithmus kann angewendet werden, wenn für eine Anwendung der Wahrheitswert einer booleschen Funktion

[7] Dies ist z.B. der Fall, wenn man die Konjunktion des BDDs für $\bigwedge_{i=1\ldots n}(p_i \oplus q_i)$ mit sich selbst berechnet (Variablenordnung $p_1 < q_1 < p_2 < q_2 < \cdots < p_n < q_n$).

$\neg\,\phi$	$O(\lvert\phi\rvert)$ [9]
$\phi\ \langle op\rangle\ \psi$	$O(\lvert\phi\rvert\lvert\psi\rvert)$
ϕ_p	$O(\lvert\phi\rvert)$
$(\exists\, p)\ \ \phi$	$O(\lvert\phi\rvert)$
$(\forall\, p)\ \ \phi$	$O(\lvert\phi\rvert)$
$\mathrm{Support}(\phi)$	$O(\lvert\phi\rvert)$
$\textsc{Simplify}(\phi,\psi)$	$O(\lvert\phi\rvert\lvert\psi\rvert)$
$\textsc{AnySat}(\phi,\psi)$	$O(\lvert\,\mathrm{Support}(\phi)\rvert)$

Tabelle 3.1.: Laufzeit für einige BDD-Operationen im schlechtesten Fall. Diese sind nur gültig, wenn dynamische Programmierung genutzt wird. $\langle op\rangle$ ist eine beliebige nichttriviale binäre logische Operation, $|\phi|$ und $|\psi|$ bezeichnen die Knotenanzahlen der BDDs für ϕ und ψ.

ϕ nur für Variablenbelegungen interessant ist, die innerhalb einer Domäne liegen, für die δ erfüllt ist. SIMPLIFY findet ein BDD mit oft weniger Knoten, das mit ϕ im Wahrheitswert immer übereinstimmt, wenn δ erfüllt ist:[8]

$$\textsc{Simplify}(\delta,\phi)\ \wedge\ \delta \quad\equiv\quad \phi\ \wedge\ \delta\ . \tag{3.6}$$

Die Komplexität der beschriebenen Operatoren ist in Tabelle 3.1 zusammengefasst. Man beachte dabei, dass die Komplexität der BDD-Algorithmen, obwohl polynomiell in der Anzahl der Knoten der beteiligten BDDs, dennoch exponentiell in der Anzahl der Variablen sein kann, da die maximale Größe der BDDs exponentiell in der Anzahl der Variablen ist.

Nach Theorem 6 auf Seite 20 besteht eine eineindeutige Zuordnung zwischen den durch aussagenlogischen Formeln dargestellten booleschen Funktionen und den BDDs, die diese repräsentieren. Da auch den logischen Operatoren entsprechende Operationen auf BDDs gegenüberstehen, werden wir im Folgenden, wo nicht anders angegeben, keine Unterscheidung zwischen aussagenlogischen Formeln und den BDDs, die sie repräsentieren, treffen.

[8] In der Literatur wird eine Anzahl von Operatoren definiert, die diese Eigenschaft erfüllen, wie z.B. die Operatoren CONSTRAIN und RESTRICT. Diese erfüllen aber noch zusätzliche Bedingungen.

[9] Die Komplexität der Negation kann bei Änderung der BDD-Darstellung durch Einführung von Kantenmarkierungen, die angeben, dass das BDD zu dem die Kante zeigt, als negiert zu betrachten ist, auf $O(1)$ reduziert werden. Da derartige Kantenmarkierungen generell die Kompaktheit der BDDs erhöhen, wird dieser Weg oft gegangen.

3.3. Anwendung von BDDs zur Darstellung von Mengen und Relationen

Eine Grundlage für viele Einsatzgebiete von BDDs ist die Möglichkeit, Mengen und Relationen sowie entsprechende Operationen mit Hilfe von BDDs und deren Operationen zu repräsentieren. In diesem Abschnitt werden wir die Grundidee dafür darstellen. Die im nächsten Kapitel beschriebene Methode für die Erreichbarkeitsanalyse in endlichen Systemen ist ein Beispiel, in der die oft kompakte Repräsentation von Mengen durch BDDs es ermöglicht, mit anderen Methoden noch nicht bearbeitbare Probleme zu lösen.

Betrachten wir zunächst die Darstellung von Mengen. Eine Grundidee, die dazu verwendet werden kann, ist die BDDs zur Darstellung ihrer charakteristischen Funktionen zu verwenden. Sei $\mathcal{V}$ eine Menge von aussagenlogischen Variablen, und $\mathbb{B}^{\mathcal{V}}$ die Menge von Variablenbelegungen von $\mathcal{V}$. Eine Menge von Variablenbelegungen von $\mathcal{V}$ kann nun durch ihre sog. **charakteristische Funktion** aus der Menge aller booleschen Funktionen über $\mathcal{V}$ repräsentiert werden. Diese ist für eine Variablenbelegung aus $\mathbb{B}^{\mathcal{V}}$ genau dann wahr, wenn diese zur repräsentierten Menge gehört, und kann als boolesche Funktion durch ein BDD dargestellt werden.

Um Mengen über einer beliebigen endlichen Grundmenge $\mathcal{A}$ darzustellen, ist nun eine Einbettung $\chi(\_)$ von $\mathcal{A}$ in $\mathbb{B}^{\mathcal{V}}$ nötig. (Dies entspricht einer binären Kodierung der Menge $\mathcal{A}$. Dazu muss natürlich $|\mathcal{A}| \leq |\mathbb{B}^{\mathcal{V}}| = 2^{|\mathcal{V}|}$ gelten.) Eine Menge $\mathcal{S}$ von Elementen von $\mathcal{A}$ wird dann durch ein BDD $\phi(\mathcal{S})$ dargestellt, so dass jedes $x \in \mathcal{A}$ genau dann in $\mathcal{S}$ enthalten ist, wenn $\chi(x) \models \phi(\mathcal{S})$. Wie kann nun eine solche Abbildung $\phi(\_)$, die Teilmengen von $\mathcal{A}$ auf BDDs abbildet, erzeugt werden?

Betrachten wir zunächst eine einelementige Menge $\{x\}$ mit $x \in \mathcal{A}$. Diese wird durch eine boolesche Funktion bzw. BDD dargestellt, dass nur für die Variablenbelegung $\chi(x)$ wahr ist. Ein solches BDD lässt sich erzeugen, indem man die Konjunktion der BDDs für die einzelnen Variablen, die in der Variablenbelegung $\chi(x)$ wahr sind, und der negierten BDDs, die in $\chi(x)$ falsch sind, bestimmt:

$$\phi(\{x\}) \underset{\text{Def}}{\equiv} \bigwedge_{v \in \mathcal{V} \,\wedge\, \chi(a) \models v} v \;\wedge \bigwedge_{v \in \mathcal{V} \,\wedge\, \chi(x) \not\models v} \overline{v}$$

Für eine Menge $\mathcal{S}$, die aus mehreren Elementen besteht, ist $\phi(\mathcal{S})$ dementsprechend wahr, wenn eins der $\phi(\{x\})$ für $x \in \mathcal{S}$ wahr ist, d.h. es ergibt sich die Disjunktion:

$$\phi(\mathcal{S}) \underset{\text{Def}}{\equiv} \bigvee_{x \in \mathcal{S}} \phi(\{x\}) \;.$$

Beispiel 11. *Sei $\mathcal{A}$ die Menge aller Teilmengen von $\{1,2,3\}$, d.h. $\mathcal{A} = \mathcal{P}(\{1,2,3\})$. Hier ist eine binäre Kodierung sehr einfach möglich: Man ordnet jedem Element i von $\{1,2,3\}$*

eine Variable x_i zu: $\mathcal{V} = \{x_1, x_2, x_3\}$, und ordnet jedem Element $x \in \mathcal{A}$ die Variablenbelegung v zu, die genau die Variablen x_i wahr macht, für die i in x enthalten ist.

$$\chi(x) = \{x_i \mapsto \mathbf{1} \mid i \in x, i \in \{1,2,3\}\} \cup \{x_i \mapsto \mathbf{0} \mid i \notin x, i \in \{1,2,3\}\}$$

Der Teilmenge $x = \{1,3\}$ von $\mathcal{A}$ wird dementsprechend die Variablenbelegung $\{x_1 \mapsto \mathbf{1}, x_2 \mapsto \mathbf{0}, x_3 \mapsto \mathbf{1}\}$ zugeordnet; der einelementigen Menge $\{x\}$ entspricht ein BDD $\phi(\{1,3\}) = x_1 \wedge \overline{x_2} \wedge x_3$, und einer Menge $\{\{1,2\},\{2,3\}\}$ entspricht ein BDD

$$\phi(\{\{1,2\},\{2,3\}\}) = (x_1 \wedge x_2 \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_2 \wedge x_3) \ .$$

Da bei einer Mengendarstellung über die charakteristische Funktion der Mengenschnitt der Konjunktion, die Mengenvereinigung der Disjunktion usw. entspricht, sind die grundlegenden Mengenoperationen in dieser Repräsentation einfach darstellbar:

$$\begin{array}{rcl} \phi(\mathcal{S} \cup \mathcal{T}) & \equiv & \phi(\mathcal{S}) \vee \phi(\mathcal{T}) \\ \phi(\mathcal{S} \cap \mathcal{T}) & \equiv & \phi(\mathcal{S}) \wedge \phi(\mathcal{T}) \\ \phi(\mathcal{S} \setminus \mathcal{T}) & \equiv & \phi(\mathcal{S}) \wedge \neg\phi(\mathcal{T}) \\ \mathcal{S} \subseteq \mathcal{T} & \text{gdw.} & \models \phi(\mathcal{S}) \rightarrow \phi(\mathcal{T}) \end{array} \tag{3.7}$$

Das Angenehme an einer BDD-Darstellung von Mengen ist nun, dass die Größe des BDDs, das die Menge repräsentiert, nicht abhängig von der Größe der Menge ist, sondern von ihrer Struktur. Angenommen wir erweitern die Kodierung in Beispiel 11 auf die Menge aller Teilmengen von $\{1,2,\ldots,500\}$. Der Menge $\mathcal{S}$ aller Teilmengen von $\{1,2,\ldots,500\}$, die die 2 enthalten, enthält nun 2^{499} Elemente; aber die Kodierung $\phi(\mathcal{S}) = x_2$ hat nur einen Knoten im BDD. Bei geeigneten Strukturen der Mengen lassen sich die Mengenoperationen oft leichter auf der Kodierung ausführen, als auf der eigentlichen Menge (was in späteren Kapiteln dieser Arbeit ausgenutzt wird.) Man beachte in diesem Zusammenhang, dass die Kodierung dabei eine sehr große Rolle spielt: Wenn die binäre Kodierung bei der o.g. Menge auf sehr ungeschickte (z.B. pseudo-zufällige) Weise geschieht, dann kann die BDD-Darstellung für die gleiche Menge bis in die Größenordnung von 2^{250} Knoten (der maximalen Größe eines BDDs mit 500 Variablen) kommen. (Wie schon erwähnt, kann auch die Anordnung der Variablen als Teil der Kodierung ähnlich großen Einfluss ausüben.)[10]

Natürlich ist die Kodierung nicht auf einfache endliche Mengen beschränkt, sondern es können auch endliche Relationen dargestellt und Operationen auf diesen ausgeführt werden. Wir beschränken unsere Diskussion auf binäre Relationen aus $\mathcal{A} \times \mathcal{A}$; die Methodik kann aber leicht auf mehrstellige Relationen oder Relationen unterschiedlicher Mengen übertragen werden.

[10] Umgekehrt ist es einfach zu zeigen, dass für jede Menge eine Kodierung existiert, für die die Größe des BDDs linear in der Anzahl der Variablen ist: Man zählt die Elemente der Menge auf, und ordnet ihnen der Reihe nach dem Dualsystem binäre Vektoren zu. Diese Art der Kodierung ist aber natürlich meist nutzlos, da sie die Menge als bekannt voraussetzt, in der Praxis aber die BDDs genutzt werden sollen, um die Menge(n) erst zu berechnen.

Wenn man nun anstatt eines Elements x von $\mathcal{A}$ ein Paar von Elementen $\langle x, y\rangle$ darstellen möchte, so bildet man x wie gehabt auf eine Variablenbelegung der Variablen $\mathcal{V}$ ab. Für y wird noch eine zweite dazu disjunkte Menge $\mathcal{V}' = \{v' \mid v \in \mathcal{V}\}$ von Variablen eingeführt, auf die y in gleicher Weise wie x abgebildet wird, nur dass nun die gestrichenen Variablen verwendet werden. Wir erweitern also $\chi(\_)$:

$$\chi(\langle x, y\rangle) = \chi(x) \cup \{v' \mapsto w \mid (v \mapsto w) \in \chi(y)\} \ . \tag{3.8}$$

Diese Abbildung ist eine Einbettung von $\mathcal{A} \times \mathcal{A}$ in $\mathbb{B}^{\mathcal{V} \cup \mathcal{V}'}$. Daher kann $\phi(\_)$ Relationen aus $\mathcal{A} \times \mathcal{A}$ in gleicher Weise wie Mengen aus $\mathcal{A}$ abbilden. Die Rechenregeln (3.7) sind ebenfalls für Relationen anwendbar.

Es gibt aber noch einige weitere Operationen auf Relationen, die oft von Bedeutung sind. Wie kann man z.B. für eine Menge $\mathcal{S} \subseteq \mathcal{A}$ und eine Relation $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ das Bild

$$\textsc{Img}(\mathcal{S}, \mathcal{R}) = \{y \mid x \in \mathcal{S} \ \wedge \ \langle x, y\rangle \in \mathcal{R}\} \tag{3.9}$$

bestimmen?[11] Gesucht sind also Elemente y entsprechend einer Variablenbelegung v' für $\mathcal{V}'$, für die eine Variablenbelegung v für $\mathcal{V}$ existiert, so dass $v \models \mathcal{S}$ und $v \cup v' \models \mathcal{R}$. Unter Verwendung der aussagenlogischen Quantifizierung stellt sich dies wie folgt dar:

$$\{v' \mapsto w \mid (v \mapsto w) \in \chi(y)\} \models (\exists\, \mathcal{V}) \ (\phi(S) \ \wedge \ \phi(R)) \ .$$

Unter Beachtung von (3.8) kann man nun durch eine Variablenumbenennung im BDD eine Kodierung der Menge aller y, die die obige Bedingung erfüllen, gewinnen:

$$\phi(\textsc{Img}(\mathcal{S}, \mathcal{R})) \ \equiv \ \big[(\exists\, \mathcal{V}) \ (\phi(S) \ \wedge \ \phi(R))\big]\{\mathcal{V}'/\mathcal{V}\} \ , \tag{3.10}$$

wobei für eine Formel ψ der Ausdruck $\psi\{\mathcal{V}'/\mathcal{V}\}$ die Formel (bzw. das BDD) darstellt, in der alle gestrichenen Variablen v' aus $\mathcal{V}'$ in die ungestrichenen Variablen v umbenannt worden sind.

Analog lässt sich ein Relationsprodukt $\mathcal{R}_1; \mathcal{R}_2 = \{\langle x, y\rangle \mid \langle x, z\rangle \in \mathcal{R}_1 \ \wedge \ \langle z, y\rangle \in \mathcal{R}_2\}$ unter Einführung einer dritten Variablenmenge $\mathcal{V}'' = \{v'' \mid v \in \mathcal{V}\}$ darstellen als:

$$\phi(\mathcal{R}_1; \mathcal{R}_2) \ \equiv \ (\exists\, \mathcal{V}'') \ \big[\phi(R_1)\{\mathcal{V}'/\mathcal{V}''\} \ \wedge \ \phi(R_2)\{\mathcal{V}/\mathcal{V}''\})\big] \ .$$

Beispiel (11 Fortsetzung)**.** *Sei $\mathcal{R}$ die Teilmengenrelation $\subseteq$ auf Teilmengen von $\{1, 2, 3\}$. Zur Variablenmenge $\mathcal{V} = \{x_1, x_2, x_3\}$ führen wir noch die Variablenmengen $\mathcal{V}' = \{x_1', x_2', x_3'\}$ und $\mathcal{V}'' = \{x_1'', x_2'', x_3''\}$ ein. Mit der gegebenen Kodierung ergibt sich nun*

$$\phi(R) \equiv (x_1 \ \rightarrow \ x_1') \ \wedge \ (x_2 \ \rightarrow \ x_2') \ \wedge \ (x_3 \ \rightarrow \ x_3') \ .$$

[11] Dies ist z.B. in der Systemverifikation oder beim Planen von Bedeutung: Wenn $\mathcal{S}$ eine Menge von Zuständen ist und $\mathcal{R}$ die Zustandsübergangsrelation, dann ist $\{y \mid x \in \mathcal{S} \ \wedge \ \langle x, y\rangle \in \mathcal{R}\}$ die Menge von Zuständen, die von $\mathcal{S}$ aus erreichbar sind.

Die Menge $\mathcal{S} = \{\{1,2\},\{1,3\}\}$ *wird kodiert als* $(x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge \overline{x_3})$. *Die Berechnung von* $\{y \mid x \in \mathcal{S} \wedge \langle x,y\rangle \in \mathcal{R}\}$ *ergibt nun*

$$(\exists\, \mathcal{V})\ (\phi(S) \wedge \phi(R))\{\mathcal{V}'/\mathcal{V}\} \equiv x_1 \wedge (x_2 \vee x_3)\ ,$$

was die Menge $\{\{1,2\},\{1,2,3\},\{1,3\}\}$ *repräsentiert.*

Für die Teilmengenrelation gilt $(\subseteq;\subseteq) = \subseteq$, *was sich entsprechend widerspiegelt in*

$$\phi(\mathcal{R}) \equiv (\exists\, \mathcal{V}'')\ \left[\phi(R)\{\mathcal{V}'/\mathcal{V}''\} \wedge \phi(R)\{\mathcal{V}/\mathcal{V}''\}\right]\ .$$

4. Erreichbarkeitsanalyse mit BDDs

Eine erfolgreiche Anwendungsmöglichkeit von BDDs ist die Erreichbarkeitsanalyse von Systemen mit endlicher Zustandsmenge. Wie der Leser im nächsten Kapitel sehen wird, besteht eine enge Verwandtschaft zwischen Erreichbarkeitsanalyse und Planen, die in dieser Arbeit für das Planen im Fluentkalkül ausgearbeitet wird.

Eine wichtige Aufgabe beim Design technischer Systeme ist die Überprüfung von deren Korrektheit in Bezug auf ihre Spezifikation. Bei der großen Komplexität der heutigen Systeme ist eine manuelle Überprüfung meist nur begrenzt möglich und fehleranfällig. Daher wird oft eine computerunterstützte Verifikation angewandt. Eine Möglichkeit dafür ist es, das technische System als endlichen Automaten oder „Finite State Machine„ (engl.) aufzufassen und zu prüfen, ob diese Eigenschaften, wie z.B. die Unerreichbarkeit bestimmter (z.B. gefährlicher) Zustände erfüllt. Da die Anzahl der Zustände dieser Finite State Machine exponentiell mit der Anzahl der Bestandteile des Systems wachsen kann, müssen u.U. sehr große Zustandsräume durchsucht werden. Dafür können BDDs oft gewinnbringend eingesetzt werden [15], wenn sie die Mengen erreichbarer Zustände kompakt darstellen können. In diesem Kapitel werden wir exemplarisch eine solche BDD-basierte Methode (eine symbolische Breitensuche unter Verwendung der Zustandsübergangsrelation[1] frei nach [76]) darstellen, die verwandt zu dem in Kapitel 8 entwickelten Planungsalgorithmus ist. Für eine ausführliche Behandlung dieses Themas verweisen wir auf Spezialliteratur wie [76].

Wir verwenden das folgende einfache Beispiel, um die Methode zu veranschaulichen.

Beispiel 12. *Eine Ventilsteuerung übernimmt das Nachfüllen eines Bassins mit Wasser. Wenn der Wasserstand im Bassin unter den niedrigstzulässigen Stand gesunken ist, muss die Steuerung dazu zuerst ein Kaltwasserventil und dann ein Warmwasserventil öffnen. Diese werden wieder geschlossen, wenn das Bassin aufgefüllt ist. Es ist zu überprüfen, ob für die elektronische Realisierung gilt, dass niemals das Warmwasserventil offen ist, wenn das Kaltwasserventil geschlossen ist, da dann der Inhalt des Bassins beschädigt werden könnte.*

Eine solches System lässt sich formal z.B. als sogenannte Finite State Machine beschreiben, an der dann die gewünschten Eigenschaften nachgewiesen werden können.

Definition 13. *Eine **Finite State Machine** ist ein 6-Tupel* $\langle I, S, O, \delta, \lambda, S^0 \rangle$*, wobei* I *das Eingabealphabet,* S *die endliche Menge der Zustände,* O *das Ausgabealphabet,* $\delta : S \times$

[1] Engl. transition relation.

$I \to S$ die Zustandsübergangsfunktion, $\lambda : S \times I \to O$ die Ausgabefunktion und $S^0 \subseteq S$ die Menge der Anfangszustände ist.

*Ein Zustand z' ist von einem Zustand z aus **erreichbar**, wenn eine Sequenz $z_0, z_1, \ldots, z_k \in S$ von Zuständen und eine (evtl. leere) Sequenz $i_0, i_1, \ldots, i_{k-1} \in I$ von Eingabedaten existiert, so dass $z_0 = z$, $z_{i+1} = \delta(z_i, i_i)$ für $i = 0, \ldots, k-1$ und $z_i = z'$. Ein Zustand ist einfach **erreichbar**, wenn er von irgendeinem Zustand aus S^0 erreichbar ist.*

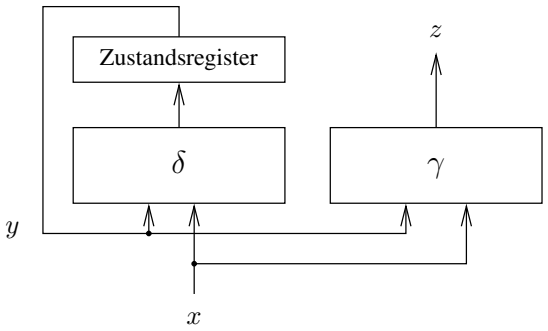


Abbildung 4.1.: Eine Finite State Machine.

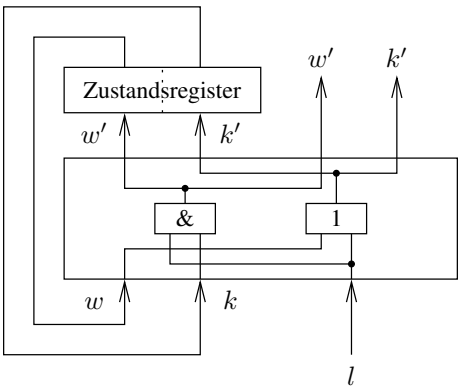


Abbildung 4.2.: Eine elektronische Realisierung von Beispiel 14. Der Eingang l wird aktiviert, wenn das Bassin leer ist, die Ausgänge w' und k' steuern das Warmwasser bzw. Kaltwasserventil an.

Betrachten wir dies an unserem Beispiel.

Beispiel 14 (Fortsetzung von 12)**.** *Die Ventilsteuerung für das Bassin sei wie in Abbildung 4.2 realisiert. Die Ausgabe des Systems wird durch ein Tupel $\langle w', k' \rangle \in \mathbb{B}^2$ mit $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$ gebildet, wobei w' und k' angeben, ob das Warmwasser bzw. Kaltwasserventil im nächsten Takt geöffnet ist. Ein entsprechendes Paar $\langle w, k \rangle$ bildet auch den Zustand des Systems - zum Beispiel stellt $\langle \mathbf{0}, \mathbf{1} \rangle$ den Zustand dar, in dem das Warmwasserventil geschlossen, aber das Kaltwasserventil geöffnet ist. Der Eingang l ist aktiviert, wenn das Bassin leer*

ist. Es gilt also: $w' = k \wedge l$ *und* $k' = w \vee l$*. Als Finite State Machine für dieses Beispiel ergibt sich:* $\mathbf{FSM}_{Basin} = \langle I, S, O, \delta, \lambda, S^{\mathbf{0}} \rangle$ *mit* $I = \mathbb{B}$, $S = O = \mathbb{B}^2$, $\delta = \lambda = \{\langle\langle w, k\rangle, l\rangle \mapsto \langle k \wedge l, w \vee l\rangle \mid w, k, l \in \mathbb{B}\}$ *und* $S^{\mathbf{0}} = \langle \mathbf{0}, \mathbf{0} \rangle$.

Wir sind nun an der Erreichbarkeit von Zuständen für alle möglichen Eingabedatensequenzen interessiert. Dazu charakterisiert die sog. Zustandsübergangsrelation für jeden Zustand der Finite State Machine, in welche anderen Zustände diese übergehen kann.

Definition 15. *Die* ***Zustandsübergangsrelation*** $\mathsf{T} : \mathcal{S} \times \mathcal{S}$ *einer Finite State Machine ist*

$$\mathsf{T} = \{\langle z_1, z_2\rangle \mid z_1 \in \mathcal{S}, (\exists\, i \in \mathcal{I})\ \ z_2 = \delta(z_1, i)\}\ .$$

Die Grundidee der Breitensuche ist es nun, schrittweise die Mengen $\mathcal{Z}_n$ der Zustände der Finite State Machine zu berechnen, die in n Schritten von einem Zustand aus S^0 aus erreichbar sind. Dieser Prozess kann abgebrochen werden, wenn keine neuen Zustände mehr hinzukommen und daher alle von S^0 erreichbaren Zustände berechnet worden sind, oder einer der gesuchten Zustände aus einer gegebenen Menge $\mathcal{G}$ gefunden worden ist. Der folgende Algorithmus liefert $\emptyset$, wenn keine Überlappung von $\mathcal{G}$ und der Menge aller erreichbaren Zustände existiert, oder eine Menge von erreichbaren Zuständen, die in $\mathcal{G}$ liegen.[2]

Algorithmus 16 (Breitensuche $(\mathsf{T}, S^0, \mathcal{G})$)**.** *Sei* T *die Zustandsübergangsrelation einer Finite State Machine,* S^0 *dessen Menge von Initialzuständen und* $\mathcal{G}$ *die Menge von Zuständen, deren Erreichbarkeit geprüft werden soll.*

$i := 0$
$\mathcal{Z}_0 := S^0$
$\mathcal{R} := \emptyset$
Wiederhole
    $i := i + 1$
    $\mathcal{Z}_i := \textsc{Img}(\mathsf{T}, \mathcal{Z}_{i-1})$
    $\mathcal{R} := \mathcal{R} \cup \mathcal{Z}_i$
so lange $\mathcal{G} \cap \mathcal{Z}_i = \emptyset$ *und* $\mathcal{Z}_i \not\subseteq \mathcal{R}$.

Das Resultat des Algorithmus ist $\mathcal{G} \cap \mathcal{R}$.

Dabei ist IMG wie in (3.9) auf Seite 30 definiert.

[2] Der Algorithmus liefert i.A. nur einen Teil der erreichbaren Zustände, die in $\mathcal{G}$ liegen, zurück. Dies reicht jedoch oft aus, z.B. um, wie in unserem Beispiel, die Korrektheit zu überprüfen. Um alle solchen Zustände zu liefern muss die Abbruchbedingung $\mathcal{Z}_i \not\subseteq \mathcal{R}$ lauten.

Beispiel 17 (Fortsetzung von 12)**.** *Für unser Beispiel ergibt sich eine Zustandsübergangsrelation*

$$\mathsf{T}_{Basin} = \left\{ \begin{array}{ll} \langle\langle \mathbf{0},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{0}\rangle\rangle, & \langle\langle \mathbf{0},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{1}\rangle\rangle \\ \langle\langle \mathbf{0},\mathbf{1}\rangle,\langle \mathbf{0},\mathbf{0}\rangle\rangle, & \langle\langle \mathbf{0},\mathbf{1}\rangle,\langle \mathbf{1},\mathbf{1}\rangle\rangle \\ \langle\langle \mathbf{1},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{1}\rangle\rangle, & \\ \langle\langle \mathbf{1},\mathbf{1}\rangle,\langle \mathbf{0},\mathbf{1}\rangle\rangle, & \langle\langle \mathbf{1},\mathbf{1}\rangle,\langle \mathbf{1},\mathbf{1}\rangle\rangle \end{array} \right\} .$$

Mit $\mathcal{G} = \{\langle \mathbf{1},\mathbf{0}\rangle\}$ *ergeben sich im Durchlauf des Algorithmus:*

i	0	1	2	3
$\mathcal{Z}_i$	$\{\langle \mathbf{0},\mathbf{0}\rangle\}$	$\{\langle \mathbf{0},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{1}\rangle\}$	$\{\langle \mathbf{0},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{1}\rangle,\langle \mathbf{1},\mathbf{1}\rangle\}$	$\{\langle \mathbf{0},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{1}\rangle,\langle \mathbf{1},\mathbf{1}\rangle\}$

Der Algorithmus terminiert bei $i = 3$*, da* $\mathcal{F}_3 \subseteq \{\langle \mathbf{0},\mathbf{0}\rangle,\langle \mathbf{0},\mathbf{1}\rangle,\langle \mathbf{1},\mathbf{1}\rangle\}$ *gilt – in diesem Schleifendurchlauf treten keine neuen Zustände mehr auf. Das Resultat ist* $\emptyset$*, d.h. es wird nie ein Zustand erreicht in dem das warme Wasser an ist, und das kalte Wasser aus ist. Unsere Beispielimplementierung hat also die zu überprüfende Eigenschaft.*

Eine Breitensuche wird als **symbolische Breitensuche** bezeichnet, wenn die auftretenden Zustandsmengen einer Breitensuche simultan verarbeitet werden – in unserem Beispiel werden wir die Mengen durch jeweils ein BDD darstellen und sämtliche Operationen des Algorithmus in BDD-Darstellung realisieren, um von der kompakten Darstellung dieser i.A. großen Mengen zu profitieren.

Seien nun $\mathcal{V}$ und $\mathcal{V}' = \{v' \mid v \in \mathcal{V}\}$ disjunkte Mengen von aussagenlogischen Variablen und $\chi(\_)$ eine umkehrbar eindeutige Abbildung von Menge der Zustände der untersuchten Finite State Machine auf $\mathbb{B}^{\mathcal{V}}$. Entsprechend Abschnitt 3.3 auf Seite 28 lässt sich nun eine Abbildung $\phi(\_)$ definieren, die Mengen und Relationen von Zuständen auf BDDs abbildet.

Algorithmus 18 (BDD-Breitensuche $(\mathtt{T}, \mathtt{S}_0, \mathtt{G})$)**.** *Seien* $\mathtt{T}, \mathtt{S}_0$ *und* $\mathtt{G}$ *BDDs, die die Zustandsübergangsrelation einer Finite State Machine, dessen Menge von Initialzuständen die Menge von Zuständen, deren Erreichbarkeit geprüft werden soll, repräsentieren:* $\mathtt{T} = \phi(\mathsf{T})$*,* $\mathtt{S}_0 = \phi(S^0)$ *und* $\mathtt{G} = \phi(\mathcal{G})$*.*

$i := 0$
$\mathtt{Z}_0 := \mathtt{S}^0$
$\mathtt{R} := \mathbf{0}$

Wiederhole

$i := i + 1$
$\mathtt{Z}_i := \textsc{Img}(\mathtt{T}, \mathtt{Z}_{i-1})$
$\mathtt{R} := \mathtt{R} \vee \mathtt{Z}_i$

so lange $\mathtt{G} \wedge \mathtt{Z}_i = \mathbf{0}$ *und* $\mathtt{Z}_i \not\subseteq \mathtt{R}$*.*

Das Resultat des Algorithmus ist $\mathtt{G} \wedge \mathtt{R}$*.*

Die Berechnung von IMG mit BDDs folgt dabei entsprechend (3.10) auf Seite 30.

Die Generierung der BDDs für $\mathtt{T}, \mathcal{S}^0$ und $\mathcal{G}$ ist abhängig von der Form, in der die Finite State Machine gegeben ist. Für unser Beispiel erledigen wir dies per Hand; im realen Einsatz der symbolischen Breitensuche für die Verifikation eines Systems wird natürlich die Finite State Machine mit Hilfe einer Beschreibungssprache gegeben sein, aus der automatisch eine solche Darstellung generiert werden kann.[3]

Schauen wir uns nun an, wie dies für unser Beispiel aussieht. Für die im Algorithmus auftretenden BDDs geben wir jeweils eine äquivalente aussagenlogische Formel an, da diese übersichtlicher als das entsprechende BDD ist.

Beispiel 19 (Fortsetzung von 12)**.** *Für die Abbildung auf BDDs benötigen wir zunächst eine Einbettung der Zustandsmenge in eine Menge von Variablenbelegungen. Es bietet sich an, für das Warmwasserventil und für das Kaltwasserventil eine Variable w bzw. k vorzusehen, die jeweils wahr sind, wenn das entsprechende Ventil offen ist:*

$$\chi(\langle w_s, k_s \rangle) = \{w \mapsto w_s, k \mapsto k_s\} \ .$$

Damit ergeben sich die BDDs für $\mathcal{S}^0$ und $\mathcal{G}$:

$$\begin{aligned} \mathtt{S}^0 &\equiv \overline{w} \ \wedge \ \overline{k} \ , \\ \mathtt{G} &\equiv w \ \wedge \ \overline{k} \ . \end{aligned}$$

Wegen $w' = k \ \wedge \ l$ und $k' = w \ \vee \ l$ ergibt sich eine Darstellung der Zustandsübergangsfunktion δ als

$$\boldsymbol{\delta} \equiv (w' \ \leftrightarrow \ k \ \wedge \ l) \ \wedge \ (k' \ \leftrightarrow \ w \ \vee \ l) \ .$$

Definition 15 „übersetzt“ in eine BDD-Darstellung lautet hier $\mathtt{T} \equiv (\exists \, l) \ \boldsymbol{\delta}$, oder:

$$\mathtt{T} \equiv ((w' \ \leftrightarrow \ k) \ \wedge \ k') \ \vee \ (\overline{w'} \ \wedge \ (k' \ \leftrightarrow \ w)) \ .$$

Im weiteren Ablauf des Algorithmus ergeben sich:

$$\begin{aligned} \mathtt{Z}_0 &\equiv \overline{w} \ \wedge \ \overline{k} \ , \\ \mathtt{Z}_1 &\equiv \overline{w} \ , \\ \mathtt{Z}_2 \equiv \mathtt{Z}_3 &\equiv \overline{w} \ \vee \ (w \ \wedge \ k), \end{aligned}$$

und das Resultat

$$\mathtt{G} \ \wedge \ \mathtt{R} \equiv \mathbf{0} \ .$$

[3] Dem in den späteren Kapiteln dieser Arbeit entwickelten Planungsalgorithmus liegt eine vergleichbare Idee zugrunde. Dort kann automatisch aus der Planungsdomänenbeschreibungssprache PDDL (Kap. 7) über den Fluentkalkül eine BDD-Darstellung erzeugt werden (Kap. 8, 9).

Dies repräsentiert die leere Zustandsmenge; es wurde also analog zu Beispiel 17 kein Zustand gefunden, der in $\mathcal{G}$ enthalten ist. Ebenfalls wurde damit die Korrektheit der Implementation bestätigt: Es wurde nachgewiesen, dass bei geschlossenem Kaltwasserventil das Warmwasserventil nie geöffnet ist.

Der aufmerksame Leser wird die Grundidee des hier präsentierten Algorithmus im Kapitel 8 wiederfinden. Doch wenden wir uns nun Planungsproblemen zu.

5. Planungsprobleme

Dieses Kapitel bietet eine kurze Einführung der Grundbegriffe von Planungsproblemen und umreißt die Art von Planungsproblemen, die in dieser Arbeit behandelt werden.

Ein Fernziel der künstlichen Intelligenz ist es, das intelligente Verhalten von Menschen nachzubilden bzw. intelligentes Handeln für ein künstlich geschaffenes System (hier **Agent** genannt) zu ermöglichen. In diesem Kontext ist ein **Planungsproblem** ein klar umrissenes Problem zielgerichteten Handelns: Unter für den Agenten überschaubaren Umständen soll der Agent eine Handlungsanleitung, einen **Plan**, zu konstruieren, der es verspricht, ein **Ziel** des Agenten zu erreichen. Es wurde eine Vielzahl von Wegen entwickelt, um diese Idee zu formalisieren. Zurückgehend auf [63, 64] liegen den meisten jedoch ähnliche Begriffswelten zugrunde.

Dabei geht man gewöhnlich verschiedene Vereinfachungen ein. Die vorwiegend genutzte Methode ist es, die Welt (bzw. den begrenzten Teil der Welt, der für das Problem relevant ist) als eine Folge von **Zuständen** zu modellieren. Ein Zustand wird dabei als „Schnappschuss" der Welt zu einem Zeitpunkt verstanden. Als Name für Zustände werden dabei meist **Situationen** eingeführt, die die Geschichte kodieren, die zum jeweiligen Zustand geführt hat. Jeder Übergang von einem Zustand zum nächsten entspricht einer **Aktion**. Diese Sicht auf Planungsprobleme liegt allen zustandsbasierten Planungssystemen (d.h. STRIPS [32] und seine Nachfolger), sowie formalen Modellen des Planens wie dem Situationskalkül und darauf basierenden Ansätzen, [63, 64, 72], dem Fluentkalkül [47] zugrunde, obwohl Erweiterungen existieren, in denen über diese Sichtweise hinausgegangen wird.

Zur Beschreibung eines Zustands dienen sog. **Fluenten**, die jeweils einen Fakt, der zur Unterscheidung der Zustände dienen kann, bezeichnen. Formal ist ein Fluent eine Funktion, die auf der Menge der Zustände definiert ist. Dabei kann es sich z.B. um **aussagenlogische Fluenten** handeln, wenn das Fluent in jeder Situation einen der Wahrheitswerte 1 (wahr) oder 0 (falsch) annimmt, oder um **Ressourcen**, deren Wert eine natürliche Zahl ist, oder sogenannte **funktionale Fluenten**, die als Wert eine Funktion haben (siehe Beispiel 20).

Der Modellierung eines Planungsproblems liegt nun jeweils eine Menge von Fluenten zugrunde. Jede Zuordnung von Werten zu allen diesen Fluenten charakterisiert nun einen **Zustand** im Modell der Welt. Man beachte, dass unterschiedliche Situationen im obigen Sinne den gleichen Zustand haben können, nämlich wenn die Unterschiede zwischen den Situationen sich nicht auf die Werte der Fluenten auswirken.

Beispiel 20. *In einem System „Schreibtisch“ seien 4 Fluenten definiert: ein Fluent „ stiftzahl “, das die Anzahl der Stifte angibt, ein Fluent „ licht “, das angibt, ob das Licht angeschaltet ist, ein Fluent „ schreibstift “, das angibt, welchen Stift der Schreibende in der Hand hält, und ein Fluent „ stiftfarben “, dessen Wert eine Funktion ist, die jedem Stift seine Farbe zuordnet. „ licht “ ist also ein aussagenlogisches Fluent, „ stiftzahl “ eine Ressource, und „ stiftfarben “ ein funktionales Fluent.*

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

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

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

Der Agent hat nun die Möglichkeit seine Umgebung (genannt **Planungsdomäne**) mit Hilfe von **Aktionen** zu verändern. Diese Aktionen sind **deterministisch**, wenn der Zustand in der Situation, die aus der Ausführung der Aktion resultiert, nur von dem Zustand vor Ausführung der Aktion abhängt, und eindeutig durch diesen und die ausgeführte Aktion bestimmt ist.

Im einfachsten Fall ist die Umgebung statisch, der Zustand ist dem Agenten vollständig bekannt und ändert sich nur durch die ausgeführten Aktionen des Agenten. Dies ist als das **klassische Planungsproblem** bekannt. In einer derartigen Umgebung ist es hinreichend, Pläne als Sequenzen von Aktionen aufzufassen. In diesem Fall ist die Suche nach einem Plan analog zu der im vorhergehenden Abschnitt diskutierten Erreichbarkeitsanalyse. Man fasst das Planungsproblem als eine Finite State Machine auf, deren Zustand der Zustand der Umgebung ist, und deren Eingabedaten die Aktionen des Agenten sind. Ein Plan existiert, wenn ein dem Planungsziel entsprechender Zustand vom Ausgangszustand erreicht werden kann. Dies eröffnet den Weg, Methoden aus der Erreichbarkeitsanalyse für Planen anzuwenden.

Wenn die Umgebung jedoch nichtdeterministisch ist, andere Agenten die Umgebung beeinflussen können oder diese natürlichen Veränderungen unterliegt, dann kann i.A. nicht mehr garantiert werden, dass eine bestimmte Sequenz von Aktionen zum Ziel führt, so dass als Pläne komplexere Strukturen, die z.B. Beobachtungen und Verzweigungen enthalten, genutzt werden müssen. Aber selbst hier ist es oft möglich das Planungsproblem zunächst in erster Näherung als klassisch zu betrachten, und dann z.B. während der Ausführung des Planes zu beobachten, ob der Plan den gewünschten Effekt erbringt (engl. *execution monitoring*), und gegebenenfalls neu zu planen (engl. *replanning*), oder einen Plan zu modifizieren (engl. *plan modification*). Dies ist jedoch nicht Gegenstand dieser Arbeit.

Plansuche kann in größeren Planungsdomänen schnell sehr komplex werden, da sowohl die Anzahl der Zustände als auch die Anzahl der während der Plansuche zu betrachtenden Pläne exponentiell mit der Anzahl der zur Domänenbeschreibung benutzten Fluenten bzw. der Anzahl der ausführbaren Aktionen steigen können. Damit ist Planen in klassischen Planungsdomänen nach wie vor eine berechnungstechnische Herausforderung. Zudem dienen Algorithmen für klassische Planungsdomänen oft als Grundlage für nichtklassisches Planen. Diese Arbeit schafft Grundlagen für das Lösen von im Fluentkalkül formulierten klassischen Planungsproblemen mit ausschließlich aussagenlogischen Fluenten unter Nutzung von Binären Entscheidungsdiagrammen und bildet so eine Grundlage für spätere Erweiterungen.

6. Fluentkalkül

Dieses Kapitel beinhaltet eine Darstellung der Grundlagen des Fluentkalküls. Insbesondere wird die Fluentkalkül-spezifische Darstellung von Zuständen in Planungsproblemen mit Hilfe des Funktionszeichens $\circ$ diskutiert. Für dies wird eine neue Axiomatisierung eingeführt, die mit der bisherigen Axiomatisierung verglichen wird.

6.1. Grundbegriffe des Fluentkalkül

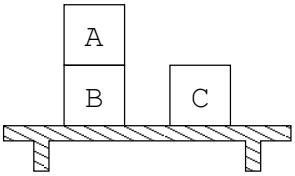
Eine wichtige Frage bei der logischen Modellierung von Planungsproblemen ist die Frage, wie der Zustand der Umgebung zu repräsentieren sei; genauer gesagt die Menge von Aussagen (Fluenten), die nötig ist, um ihn für die Zwecke der Modellierung zu charakterisieren. Um dies zu veranschaulichen, betrachten wir folgendes Beispiel.

Beispiel 21 (Welt der Blöcke)**.** *Gegeben sei eine Menge von Blöcken* $\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$ *und ein Tisch. Jeder Block kann entweder auf dem Tisch oder auf genau einem anderen Block stehen, wobei auf jedem Block höchstens ein anderer Block stehen kann. Weiterhin sei der Tisch groß genug, das alle Blöcke nebeneinander auf dem Tisch stehen können. Von der genauen Lage der Blöcke im Raum wird aber abstrahiert; von Interesse sind nur die Beziehungen, die sich in den folgenden Fluenten wiederspiegeln:*

On(x, y) ***:*** *ist wahr, wenn Block* x *auf Block* y *steht.*

Clear(x) ***:*** *ist wahr, wenn kein Block auf Block* x *steht.*

OnTable(x) ***:*** *ist wahr, wenn Block* x *auf dem Tisch steht.*



Zur Veränderung des Zustands steht eine Aktion $PutOn(x, y)$ *zur Verfügung, die Block* x *auf Block* y *setzt, sowie eine Aktion* $ToTable(x)$*, die Block* x *auf den Tisch stellt.*

In vielen Ansätzen wie dem Situationskalkül und darauf basierenden Ansätzen [63, 64, 72] oder dem Eventkalkül [57] werden Zustände repräsentiert, indem die Fluenten direkt als atomare Aussagen der Logik dargestellt werden: Im Situationskalkül wird z.B. der oben abgebildete Ausgangszustand durch eine Menge von Fakten $\{Holds(Clear(\mathtt{A}), \mathtt{S}_0)$, $\neg\, Holds(Clear(\mathtt{B}), \mathtt{S}_0)$, $Holds(Clear(\mathtt{C}), \mathtt{S}_0)$, $Holds(On(\mathtt{A}, \mathtt{B}), \mathtt{S}_0)$, $\neg\, Holds(On(\mathtt{A}, \mathtt{C}), \mathtt{S}_0)$, $\ldots, \neg\, Holds(OnTable(\mathtt{A}), \mathtt{S}_0)$, $Holds(OnTable(\mathtt{B}), \mathtt{S}_0)$,

$\mathit{Holds}(\mathit{OnTable}(\mathtt{B}), \mathtt{S}_0)\}$ dargestellt, wobei $\mathtt{S}_0$ eine Konstante ist, die die initiale Situation repräsentiert. Dies hat aber technische Nachteile [44, 84]. Im **Fluentkalkül** (abgekürzt **FC**) wird dagegen der **Zustand** reifiziert und als Objekt 1. Klasse behandelt, d.h. als einen Term.

Intuitiv kann ein Zustand z durch die Menge von Fluenten beschrieben werden, denen im Zustand z der Wahrheitswert $\mathbf{1}$ zugeordnet werden kann. Dabei wird jedes Fluent durch einen Term repräsentiert. Diese Intuition wird direkt im FC realisiert, indem diese Fluenten durch ein binäres Funktionszeichen $\circ$ zu einem Term verknüpft werden. Dem im obigen Beispiel abgebildeten Zustand entspricht also der Term $\mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C}) \circ \mathit{On}(\mathtt{A}, \mathtt{B}) \circ \mathit{OnTable}(\mathtt{B}) \circ \mathit{OnTable}(\mathtt{C})$. Man beachte, dass im Unterschied zur oben dargestellten Beschreibung des Zustands im Situationskalkül die Fluenten, die im beschriebenen Zustand falsch sind, nicht aufgeführt werden: Alle nicht aufgeführten Fluenten sind definitionsgemäß in diesem Zustand falsch. [1] Da gemäß unserer Intuition des Zustands die Reihenfolge in der Aufzählung der Fluenten irrelevant ist, werden Assoziativität und Kommutativität für $\circ$ gefordert. Zudem wird ein Symbol $\emptyset$ für den leeren Zustandsterm (der einen Zustand, in dem kein Fluent wahr ist, bezeichnet) eingeführt. Formal entspricht dies, in Zusammenhang mit den Standard-Gleichheitsaxiomen S. 2.4:[2] den folgenden Axiomen für Zustände:

$$\begin{aligned} (\forall\, z_1, z_2, z_3)\ \ (z_1 \circ z_2) \circ z_3 &= z_1 \circ (z_2 \circ z_3) \\ (\forall\, z_1, z_2)\ \ z_1 \circ z_2 &= z_2 \circ z_1 \\ (\forall\, z)\ \ z \circ \emptyset &= z \end{aligned} \tag{AC1}$$

Auf die Idempotenz $z = z \circ z$ des Funktionszeichens wird verzichtet. Dies hat einerseits technische Vorteile (darauf wird im Abschnitt 6.6 noch genauer eingegangen), andererseits ergeben sich dadurch erweiterte Ausdrucksmöglichkeiten (siehe nächster Abschnitt.) Der Zustand entspricht nun nicht mehr einfach einer Menge, da z.B. die Gleichheit $\mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A}, \mathtt{B})$ und $\mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{B}) \circ \mathit{Clear}(\mathtt{A})$ mit Hilfe von (AC1) nicht abgeleitet werden kann, obwohl sie beide aus der Menge $\{\mathit{Clear}(\mathtt{A}), \mathit{On}(\mathtt{B})\}$ von Fluenten zusammengesetzt sind. Zwei Terme sind bezüglich (AC1) nur dann gleich, wenn die Fluenten in beiden Termen gleich oft vorkommen. Dies entspricht dem Konzept von Multimengen (siehe Abschnitt 2.5 auf Seite 15), die jedes Element endlich oft enthalten können; die beiden genannten Terme entsprechen den unterschiedlichen Multimengen $\dot{\{}\mathit{Clear}(\mathtt{A}), \mathit{On}(\mathtt{A}, \mathtt{B})\dot{\}}$ und $\dot{\{}\mathit{Clear}(\mathtt{A}), \mathit{On}(\mathtt{B}), \mathit{Clear}(\mathtt{A})\dot{\}}$. Mengen werden dabei als spezielle Multimengen, in denen jedes Element nur einmal vorkommt, dargestellt.

Die Darstellung von Zuständen als Multimengen eröffnet nun eine elegante Möglichkeit zur

[1] Eine weitere Variante der Zustandsdarstellung, die in einigen Arbeiten ausgenutzt wird, ist die explizite Kennzeichnung von Fluenten, denen der Wahrheitswert $\mathbf{0}$ zugeordnet wird, durch überstreichen: Der angegebene Zustand stellt sich dann dar als $\mathit{Clear}(\mathtt{A}) \circ \overline{\mathit{Clear}(\mathtt{B})} \circ \mathit{Clear}(\mathtt{C}) \circ \ldots$. Dies bringt u.U. technische Vorteile bei der Darstellung von Aktionen mit negativen Effekten und / oder Vorbedingungen, sowie bei der Darstellung von unvollständiger Information über Zustände.

[2] Siehe z.B. [59].

Darstellung von Ressourcen: Wenn zum Beispiel modelliert werden soll, dass in einer Geldbörse eine bestimmte Anzahl von 1-Mark Stücken vorhanden ist, kann dies durch die Multiplizität eines entsprechenden Fluents m in der Multimenge bzw. im entsprechenden Zustandsterm ausgedrückt werden: Wenn fünf 1-Mark Stücke vorhanden sind entspricht dies einer Zustandsbeschreibung $m \circ m \circ m \circ m \circ m$ bzw. einer Multimenge $\dot{\{}m, m, m, m, m\dot{\}}$. Diese Form der Darstellung wird in einigen Arbeiten (z.B. [30, 48, 12, 10, 9]) ausgenutzt, bleibt aber in dieser Arbeit unberücksichtigt, da dem Autor kein geeigneter Weg zur Darstellung von Multimengen mit Hilfe von BDDs bekannt ist.

Diese Gleichungstheorie (AC1) ist zwar ausreichend die Gleichheit von Termen wie $\mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C}) \circ \mathit{On}(\mathtt{A}, \mathtt{B})$ und $\mathit{On}(\mathtt{A}, \mathtt{B}) \circ \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C})$ abzuleiten, aber es lässt sich nicht die Ungleichheit von Termen wie $\mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C})$ und $\mathit{On}(\mathtt{A}, \mathtt{B})$ schlussfolgern. So ist es z.B. wichtig $\mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A}, \mathtt{B}) \neq \mathit{Clear}(\mathtt{B}) \circ z$ für ein beliebiges z ableiten zu können, um zu modellieren, dass der Block `B` nicht *Clear* ist, wenn Block `A` darauf steht. Es bedarf also einer ergänzenden Axiomatisierung, die Thema der nächsten Abschnitte ist.

Doch zunächst führen wir einige allgemeine Symbole des Fluentkalkül ein. Es wird Logik mit Gleichheit benutzt. Der Modellierung liegt eine Signatur wie folgt zu Grunde:

Definition 22. *Eine Signatur wird* ***Fluentkalkül-Signatur*** *genannt, wenn es die folgende Struktur Σ_{FC} enthält:*

SORT	*Action*, *Sit*, *Fluent* $\preceq$ *State*,	
FUN	$\mathtt{S}_0$	$: \rightarrow$ *Sit*,
	do	: *Action* $\times$ *Sit* $\rightarrow$ *Sit*,
	state	: *Sit* $\rightarrow$ *State*,
	$\emptyset$	: *State*,
	$\circ$	: *State* $\times$ *State* $\rightarrow$ *State*.

Zu den erwähnten Konstanten und Funktionen kommen noch domänenspezifische Bestandteile hinzu.

Beispiel 23 (Welt der Blöcke, Fortsetzung)**.** *Zur Beschreibung der Welt der Blöcke eignet sich eine Fluentkalkül-Signatur mit folgenden Zusatzelementen:*

SORT	*Block*,	
FUN	$\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots$	: *Block*,
	Clear	: *Block* $\rightarrow$ *Fluent*,
	On	: *Block* $\times$ *Block* $\rightarrow$ *Fluent*,
	OnTable	: *Block* $\rightarrow$ *Fluent*,
	PutOn	: *Block* $\times$ *Block* $\rightarrow$ *Action*,
	ToTable	: *Block* $\rightarrow$ *Action*.

Objekte der Sorte *Fluent* denotieren die Fluenten einer Planungsdomäne (wie $\mathit{On}(\mathtt{A}, \mathtt{B}), \mathit{Clear}(\mathtt{C})$). Diese Sorte ist eine Untersorte der Sorte *State*, die alle Zustände

enthält, so dass ein einzelnes Fluent bereits die Beschreibung eines Zustands darstellt, in dem genau dieses einzelne Fluent wahr ist. Kombiniert werden diese wie beschrieben durch das Funktionszeichen $\circ$. $\mathit{state}(s)$ denotiert den Zustand, der in einer Situation s herrscht. $\mathtt{S}_0$ ist die initiale Situation, und $\mathit{do}(a, s)$ bezeichnet die Situation, die nach Ausführung einer Aktion (Sorte *Action*) in der Situation s auftritt. So ergibt sich für Beispiel 21 der Anfangszustand zu

$$\mathit{state}(\mathtt{S}_0) = \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C}) \circ \mathit{On}(\mathtt{A}, \mathtt{B}) \circ \mathit{OnTable}(\mathtt{B}) \circ \mathit{OnTable}(\mathtt{C}) \ .$$

Wenn im Anfangszustand die Aktion $\mathit{PutOn}(\mathtt{A}, \mathtt{C})$ ausgeführt wird, die Block A auf Block C setzt, dann ergibt sich ein Zustand

$$\mathit{state}(\mathit{do}(\mathit{PutOn}(\mathtt{A}, \mathtt{C}), \mathtt{S}_0)) = \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) \circ \mathit{On}(\mathtt{A}, \mathtt{C}) \circ \mathit{OnTable}(\mathtt{B}) \circ \mathit{OnTable}(\mathtt{C}) \ .$$

In Anlehnung an den Situationskalkül wird oft eine Symbol *Holds* verwendet. Im Fluentkalkül ist dieses allerdings ein Makro, anstatt wie im Situationskalkül ein Prädikat:

$$\mathit{Holds}(f, s) \underset{\text{Def}}{\equiv} (\exists\, z' : \mathit{State})\ \mathit{state}(s) = f \circ z' \qquad \text{(Holds)}$$

Ebenso wie im Situationskalkül gibt $\mathit{Holds}(f, s)$ an, ob ein Fluent f im Zustand der in der Situation s herrscht, wahr ist (bzw. hier: enthalten ist.)

Wenn wie in dieser Arbeit nur aussagenlogische Fluenten (die nur die Werte „wahr“ und „falsch“ annehmen können) verwendet werden, wird zusätzlich ein Axiom eingeführt, das festlegt, dass in Zuständen des Planungsproblems keine Fluenten mehr als einmal auftreten können. Dies ist insbesondere dann von Bedeutung, wenn der Anfangszustand nicht vollständig spezifiziert ist, so dass für Zustände Werte, in denen Fluenten mehrfach vorkommen, explizit ausgeschlossen werden müssen.

$$(\forall\, s : \mathit{State})\ \neg\, (\exists\, f : \mathit{Fluent}, z : \mathit{State})\ \mathit{state}(s) = f \circ f \circ z \qquad \text{(NonMult)}$$

Eine detailliertere Einführung in das Thema „Fluentkalkül“ gibt [84]. Auf welche Weise im Fluentkalkül die Wirkung von Aktionen unter Anwendung von *state*, *do* und $\mathtt{S}_0$ beschrieben wird, wird im Abschnitt 6.6 näher diskutiert.

6.2. Die Unifikationsvollständigkeit und ihre Beschränkungen

In vielen früheren Arbeiten wurde dem Fluentkalkül eine sogenannte (AC1)-unifikationsvollständige Theorie (AC1*) zugrundegelegt [45].[3] Eine solche (AC1)-unifikationsvollständige Gleichungstheorie hat zwei Eigenschaften: Erstens legt sie Terme,

[3] Eine genauere Diskussion dieses Begriffs erfolgt im Abschnitt 6.5.

die sich unter der Gleichungstheorie (AC1) nicht unifizieren lassen, als ungleich fest, und zweitens reflektiert sie für (AC1)-unifizierbare Terme die Unifikatoren, wie im folgenden diskutiert. Die Gleichungstheorie (AC1) wird mit (AC1*) in neueren Publikationen, die eine ordnungssortierte Logik verwenden, wie folgt zu EUNA (extended unique name assumption) zusammengefasst:

Definition 24 ([46], [84])**.** *Der Axiomsatz **EUNA** besteht aus folgenden Axiomen:*

1. *den Axiomen* (AC1)*,*
2. *für alle Paare von Termen t_1 und t_2 einer Sorte, mit Ausnahme von* State*, mit den Variablen $\vec{x}$:*
   a) *wenn t_1 und t_2 nicht (AC1)-unifizierbar sind, einem Axiom*
   $$\neg\,(\exists\,\vec{x})\;\; t_1 = t_2\;\;.$$
   b) *wenn t_1 und t_2 mit dem Unifikator $\theta = \{x_1/r_1, \ldots, x_n/r_n\}$ (AC1)-unifizierbar sind, einem Axiom*
   $$(\forall\,\vec{x})\;\; [t_1 = t_2\;\rightarrow\;(\exists\,\vec{y})\;\;\theta_{=}]\;\;.$$
   wobei $\theta_=$ die Formel $x_1 = r_1 \wedge \ldots \wedge x_n = r_n$ bezeichnet, und $\vec{y}$ die Variablen bezeichnet, die in $\theta_=$ aber nicht in $\vec{x}$ vorkommen,
3. *für alle Paare von Termen t_1 und t_2 der Sorte* State *mit den Variablen $\vec{x}$:*
   a) *wenn t_1 und t_2 nicht (AC1)-unifizierbar sind, einem Axiom*
   $$\neg\,(\exists\,\vec{x})\;\; t_1 = t_2\;\;.$$
   b) *wenn t_1 und t_2 mit der vollständigen Menge von Unifikatoren $cU_e(s,t)$ (AC1)-unifizierbar sind, einem Axiom*
   $$(\forall\,\vec{x})\;\; \left[t_1 = t_2\;\rightarrow\;\bigvee_{\theta\in cU_e(s,t)}(\exists\,\vec{y})\;\;\theta_{=}\right]\;\;.$$
   wobei $\theta_=$ für eine Substitution $\theta = \{x_1/r_1, \ldots, x_n/r_n\}$ die Formel $x_1 = r_1 \wedge \ldots \wedge x_n = r_n$ bezeichnet, und $\vec{y}$ die Variablen bezeichnet, die in $\theta_=$ aber nicht in $\vec{x}$ vorkommen.

Mit dieser Gleichungstheorie ist nun tatsächlich ableitbar, dass $\mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A},\mathtt{B}) = \mathit{Clear}(\mathtt{B}) \circ z$ nicht erfüllbar ist, da die linke und die rechte Seite der Gleichung unter (AC1) nicht unifizierbar sind, und daher nach 3a) ein Axiom $\neg\,(\exists\;z)\;\; \mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A},\mathtt{B}) \neq \mathit{Clear}(\mathtt{B}) \circ z$ in EUNA enthalten ist.

Wenn für zwei Terme mehrere Möglichkeiten der Unifikation bestehen, dann werden alle diese Möglichkeiten in ein Axiom kodiert. Z.B. für die Terme

$$\mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) \qquad \text{und} \qquad \mathit{Clear}(x) \circ z$$

bestehen zwei grundsätzlich verschiedene Möglichkeiten der Unifikation: Erstens $\{x/\mathtt{A}, z/\mathit{Clear}(\mathtt{B})\}$ und zweitens $\{x/\mathtt{B}, z/\mathit{Clear}(\mathtt{A})\}$. Diese zwei Unifikatoren bilden zusammen eine vollständige Menge von Unifikatoren, und damit enthält EUNA nach 3b) das Axiom

$$\begin{aligned}(\forall\, x, z)\ \ \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) = \mathit{Clear}(x) \circ z\ \rightarrow \\ [x = \mathtt{A}\ \wedge\ z = \mathit{Clear}(\mathtt{B})] \quad \vee \quad [x = \mathtt{B}\ \wedge\ z = \mathit{Clear}(\mathtt{A})]\ .\end{aligned} \tag{6.1}$$

Da die Menge aller Terme unendlich ist, enthält EUNA eine unendliche Menge von Axiomen. Damit ist es aber nicht möglich, diese alle explizit aufzuzählen, um sie in ein Programm zur Anwendung des Fluentkalküls zu verwenden. Dieses Problem kann auf zwei Arten gelöst werden:

- Da effiziente Algorithmen zur Bestimmung von vollständigen Mengen von Unifikatoren bezüglich (AC1) für bestimmte syntaktischen Einschränkungen der unifizerten Terme bekannt sind [82, 69], und die Verwendung des AC1-Funktionszeichens $\circ$ im Fluentkalkül diesen Einschränkungen genügt, ist die Definition 24 konstruktiv, so dass die im Prozess des logischen Schließens benötigten Axiome bei Bedarf generiert werden können. Von Nachteil bei dieser Verfahrensweise ist allerdings, dass logisches Schließen mit Gleichheit nur schwer effizient durchzuführen ist.

- Die Gleichungstheorie wird nur implizit als Basis für die Implementation verwendet. Dies geschieht z.B. bei der sogenannten SLDENF-Resolution [74], deren Anwendung für den Fluentkalkül in [45, 80] diskutiert wird, sowie bei FLUX [88], einer Kombination von logischer Programmierung mit einem Constraint Solver.[4]

Die (AC1)-Unifikationsvollständigkeit hat jedoch einige Beschränkungen. So ist es dabei schwierig oder unmöglich Ergänzungen der Gleichungstheorie durch einfache planungsdomänenspezifische Gleichungen oder Funktionen vorzunehmen. Versuchen wir z.B. unser Beispiel der Welt der Blöcke durch farbige Blöcke durch ein Funktionssymbol *Color*, das die Farbe eines Blocks angibt, zu erweitern: Z.B. $\mathit{Color}(\mathtt{A}) = \mathtt{Rot}$, $\mathit{Color}(\mathtt{B}) = \mathtt{Weiß}$ und $\mathit{Color}(\mathtt{C}) = \mathtt{Ungefärbt}$, wobei $\mathtt{Weiß}$ der „natürliche“ Zustand der Blöcke sei, d.h. $\mathtt{Ungefärbt} = \mathtt{Weiß}$. Wenn man diese Axiome ohne weitere Modifikationen zu EUNA hinzufügt, ergeben sich Widersprüche zu EUNA:

[4] Die später diskutierte Anwendung von BDDs auf das Schließen im Fluentkalkül folgt diesem Geist: die Gleichungstheorie bildet nur die semantische Basis für das Verfahren.

Beobachtung 25.
`Ungefärbt` $=$ `Weiß` *und EUNA sind inkonsistent.*
Color(`A`) $=$ `Rot` *und EUNA sind inkonsistent.*

Beweis. Die Terme `Ungefärbt` und `Weiß` sind nicht unifizierbar. Damit enthält EUNA nach Definition 24 Punkt 2a) ein Axiom $\neg$ `Ungefärbt` $=$ `Weiß` , was zu `Ungefärbt` $=$ `Weiß` widersprüchlich ist. Analog ergibt sich die andere Behauptung. ∎

Wenn man das Konzept der Unifikationsvollständigkeit nicht aufgeben möchte, dann ist der einzige Weg derartige Axiome zu integrieren, dass man diese in die Gleichungstheorie integriert, bezüglich derer die Unifikationsvollständigkeit besteht. Man benutzt also eine E-unifikationsvollständige Gleichungstheorie, wobei Gleichungstheorie E sowohl die Axiome (AC1), als auch die domänenspezifischen Gleichheitsaxiome enthält. Dies bringt jedoch erhebliche Probleme mit sich. Die Definition 24 setzt voraus, dass für jedes unifizierbare Paar von Termen eine höchstens endliche vollständige Menge von E-Unifikatoren existiert. Dies ist bei Hinzufügen neuer Gleichungen zu (AC1) nicht unbedingt gegeben. Damit müsste für jede neue Planungsdomäne mit domänenspezifischer Gleichungstheorie neu nachgewiesen werden, dass für jedes Paar von Termen eine endliche vollständige Menge von E-Unifikatoren existiert, da sonst die Definition der unifikationsvollständigen Gleichungstheorie nicht anwendbar ist. Zudem sind die bekannten Unifikationsalgorithmen nicht unbedingt für die ergänzte Gleichungstheorie anwendbar.

6.3. Eine neue Axiomatisierung

Die Beschränkungen einer (AC1)-unifikationsvollständigen Gleichungstheorie können aber überwunden werden, wenn man von der syntaktischen Betrachtungsweise auf eine semantische Betrachtungsweise wechselt. Wie bereits beschrieben, sollten zwei Terme der Sorte *State* genau dann gleich sein, wenn sie die gleichen Fluenten enthalten, d.h. wenn die Multimengen von Fluenten, die sie darstellen, identisch sind. Im folgenden wird nun eine Menge $\mathcal{F}_{mset}$ von Axiomen dargestellt, die das Konzept von endlichen Multimengen beschreiben, so dass diese Bedingung automatisch erfüllt wird.

Wie bereits erwähnt, ist das Funktionszeichen $\circ$ kommutativ, assoziativ, und hat die leere Zustandsbeschreibung als Einselement. Dies entspricht den folgenden Axiomen, die die Gleichungstheorie (AC1) darstellen: [5]

[5] Dem aufmerksamen Leser mag auffallen, das wir auf Seite 44 die gleichen Axiome ohne die Quantifizierung in der Sorte *State* eingeführt haben. Wir sind aber inzwischen in den Kontext einer ordnungssortierten Logik gewechselt, so dass sich diese Axiome nur noch auf die bereits eingeführte Sorte der Zustände, *State* , beziehen.

$$\begin{array}{rrcl} (\forall\, x,y,z : \mathit{State}) & (x \circ y) \circ z & = & x \circ (y \circ z) \\ (\forall\, x,y : \mathit{State}) & x \circ y & = & y \circ x \\ (\forall\, x : \mathit{State}) & x \circ \emptyset & = & x \end{array} \tag{AC1}$$

Um nun Multimengen von Fluenten und Terme der Sorte *State* vergleichen zu können, geben wir eine formale Zuordnung an, die einer Multimenge von Termen der Sorte *Fluent* einen Term der Sorte *State* zuordnet. Die Funktion [6] $\varrho : \mathcal{M}_{\text{fin}}(T_{\Sigma,\mathit{Fluent}}) \to T_{\Sigma,\mathit{State}}$ sei wie folgt definiert, wobei $\mathcal{M}_{\text{fin}}(A)$ für eine Menge A die Menge aller endlichen Multimengen über A bezeichnet:

$$\varrho(\dot\{\underbrace{f_1,\ldots,f_1}_{k_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{k_n \text{ mal}}\dot\}) = \emptyset \circ \underbrace{f_1 \circ \cdots \circ f_1}_{k_1 \text{ mal}} \circ \cdots \circ \underbrace{f_n \circ \cdots \circ f_n}_{k_n \text{ mal}} \ . \tag{6.2}$$

Z.B. ist $\varrho(\dot\{\mathit{Clear}(\mathtt{A}), \mathit{OnTable}(\mathtt{C}), \mathit{On}(\mathtt{A},\mathtt{B})\dot\}) = \emptyset \circ \mathit{Clear}(\mathtt{A}) \circ \mathit{OnTable}(\mathtt{C}) \circ \mathit{On}(\mathtt{A},\mathtt{B})$.

Was können wir nun über die Modelle der bereits dargestellten Axiome für $\circ$ sagen? Unsere Intention ist, dass in einem Modell $\mathcal{M}$ die Interpretation $\mathit{State}^{\mathcal{M}}$ der Sorte *State* die Menge von Multimengen über der Interpretation von $\mathit{Fluent}^{\mathcal{M}}$ der Sorte *Fluent* in einem noch genauer zu spezifizierenden Sinn darstellt. Um dies formal diskutieren zu können, übertragen wir die Funktion ϱ sinngemäß auf die Entsprechungen der Sorten im Modell. $\varrho^{\mathcal{M}} : \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}}) \to \mathit{State}^{\mathcal{M}}$ sei also wie folgt definiert:

$$\varrho^{\mathcal{M}}(\dot\{\underbrace{f_1,\ldots,f_1}_{k_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{k_n \text{ mal}}\dot\}) = \emptyset \circ^{\mathcal{M}} \underbrace{f_1 \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_1}_{k_1 \text{ mal}} \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} \underbrace{f_n \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_n}_{k_n \text{ mal}} \ . \tag{6.3}$$

Über Modelle der schon aufgestellten Axiome lässt sich bereits die folgende Aussage machen:

Lemma 26. *In jedem Modell $\mathcal{M}$ von* (AC1) *ist $\varrho^{\mathcal{M}}$ ein Homomorphismus von $\langle \mathcal{M}_{\mathit{fin}}(\mathit{Fluent}^{\mathcal{M}}), \dot\emptyset, \dot\cup \rangle$ nach $\langle \mathit{State}^{\mathcal{M}}, \emptyset^{\mathcal{M}}, \circ^{\mathcal{M}} \rangle$, d.h.*

$$(\forall\, \dot{\mathcal{Z}}_1, \dot{\mathcal{Z}}_2 \in \mathcal{M}_{\mathit{fin}}(\mathit{Fluent}^{\mathcal{M}}))\ \ \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_1 \dot\cup \dot{\mathcal{Z}}_2) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_1) \circ^{\mathcal{M}} \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_2)$$

Beweis. Seien $\dot{\mathcal{Z}}_1 = \dot\{\underbrace{f_1,\ldots,f_1}_{k_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{k_n \text{ mal}}\dot\}$ und $\dot{\mathcal{Z}}_2 = \dot\{\underbrace{f_1,\ldots,f_1}_{l_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{l_n \text{ mal}}\dot\}$.
ϱ ist nun ein Homomorphismus, da

$$\begin{aligned} &\varrho^{\mathcal{M}}(\dot\{\underbrace{f_1,\ldots,f_1}_{k_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{k_n \text{ mal}}\dot\} \dot\cup \dot\{\underbrace{f_1,\ldots,f_1}_{l_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{l_n \text{ mal}}\dot\}) \\ &= \varrho^{\mathcal{M}}(\dot\{\underbrace{f_1,\ldots,f_1}_{k_1+l_1 \text{ mal}},\ldots,\underbrace{f_n,\ldots,f_n}_{k_n+l_n \text{ mal}}\dot\}) \end{aligned}$$

[6] Formal gesehen ist die hier angegebene Zuordnung nicht eindeutig, aber da $\circ$ kommutativ und assoziativ ist, ist die Anordnung bedeutungslos, denn alle möglichen Anordnungen liegen in der gleichen Äquivalenzklasse bezüglich der Gleichungen (AC1). Wir gehen also im folgenden davon aus, dass ϱ jeder Multimenge einen festen Term zuordnet; welcher der vielen möglichen es ist, ist aber für unsere Betrachtungen nicht relevant.

nach der Definition der Multimengenvereinigung $\dot\cup$,

$$= \emptyset \circ^{\mathcal{M}} \underbrace{f_1 \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_1}_{k_1+l_1 \text{ mal}} \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} \underbrace{f_n \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_n}_{k_n+l_n \text{ mal}}$$

$$= \emptyset \circ^{\mathcal{M}} \underbrace{f_1 \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_1}_{k_1 \text{ mal}} \circ^{\mathcal{M}} \underbrace{f_1 \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_1}_{l_1 \text{ mal}} \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} \underbrace{f_n \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_n}_{k_n \text{ mal}} \circ^{\mathcal{M}} \underbrace{f_n \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_n}_{l_n \text{ mal}}$$

$$= \left(\emptyset \circ^{\mathcal{M}} \underbrace{f_1 \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_1}_{k_1 \text{ mal}} \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} \underbrace{f_n \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_n}_{k_n \text{ mal}} \right) \circ^{\mathcal{M}} \left(\emptyset \circ^{\mathcal{M}} \underbrace{f_1 \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_1}_{l_1 \text{ mal}} \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} \underbrace{f_n \circ^{\mathcal{M}} \ldots \circ^{\mathcal{M}} f_n}_{l_n \text{ mal}} \right)$$

wegen (AC1),

$$= \varrho^{\mathcal{M}}(\dot\{ \underbrace{f_1, \ldots, f_1}_{k_1 \text{ mal}}, \ldots, \underbrace{f_n, \ldots, f_n}_{k_n \text{ mal}} \dot\}) \circ^{\mathcal{M}} \varrho^{\mathcal{M}}(\dot\{ \underbrace{f_1, \ldots, f_1}_{l_1 \text{ mal}}, \ldots, \underbrace{f_n, \ldots, f_n}_{l_n \text{ mal}} \dot\})$$

entsprechend der Definition von $\varrho^{\mathcal{M}}$. ∎

Dies ist allerdings für unsere Zwecke noch nicht ausreichend, da dies auch Modelle gestattet, in denen z.B. $\mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) = \mathit{OnTable}(\mathtt{C}) \circ \mathit{On}(\mathtt{A}, \mathtt{B})$ gilt, selbst wenn $\mathit{Clear}(\mathtt{A})^{\mathcal{M}}, \mathit{Clear}(\mathtt{B})^{\mathcal{M}}, \mathit{OnTable}(\mathtt{C})^{\mathcal{M}}$ und $\mathit{On}(\mathtt{A}, \mathtt{B})^{\mathcal{M}}$ paarweise verschieden sind. Um dies auszuschließen, führen wir drei weitere Axiome ein.

- Erstens wird festgelegt, dass die Fluenten (d.h. Elemente aus einer Interpretation von *Fluent*) bezüglich $\circ$ irreduzibel, sowie verschieden von $\emptyset$ sind. Man beachte, dass $(\exists\, f : \mathit{Fluent})\ z = f$ genau dann wahr ist, wenn z in der Subsorte *Fluent* von *State* enthalten ist.

$$(\forall\, z : \mathit{State})\ \big[\big((\exists\, f : \mathit{Fluent})\ z = f\ \rightarrow\ z \neq \emptyset\ \wedge\ (\forall\, z', z'' : \mathit{State})\ (z = z' \circ z''\ \rightarrow\ z' = \emptyset\ \vee\ z'' = \emptyset)\big] \quad \text{(Irred)}$$

- Zweitens habe $\emptyset$ keine Teiler bezüglich $\circ$ (genauso wie $\dot\emptyset$ keine echten Teilmengen hat):

$$(\forall\, z, z' : \mathit{State})\ \big[\emptyset = z \circ z'\ \rightarrow\ z = \emptyset\big] \quad \text{(NullTeil)}$$

- Und drittens wird eine Eigenschaft der Multimengenalgebra $\langle \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}}), \dot\emptyset, \dot\cup \rangle$ für $\circ$ postuliert:

$$(\forall\, z_1, z_2, z_3, z_4)\ \Big[z_1 \circ z_2 = z_3 \circ z_4\ \rightarrow\ (\exists\, z_a, z_b, z_c, z_d)\ \left(\begin{array}{l} z_1 = z_a \circ z_b\ \wedge\ z_2 = z_c \circ z_d\ \wedge \\ z_3 = z_a \circ z_c\ \wedge\ z_4 = z_b \circ z_d \end{array} \right)\Big] \quad \text{(Levi)}$$

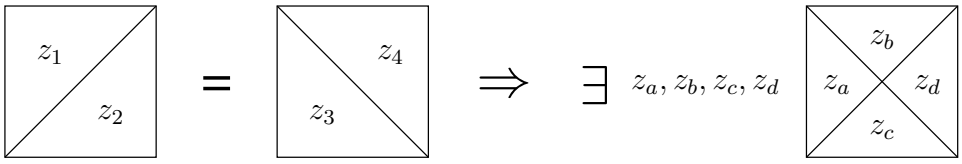


Abbildung 6.1.: Das Levi Axiom: Wenn ein Zustand (dargestellt durch ein Quadrat) sowohl in z_1 und z_2 als auch in z_3 und z_4 aufgeteilt werden kann, so kann er auch in z_a, z_b, z_c, z_d aufgeteilt werden, so dass gleiche Flächen bezüglich (AC1) gleiche Teilterme darstellen. Man beachte dabei, dass im Gegensatz zur geometrischen Veranschaulichung bei mehrfachem Vorkommen von Teiltermen in $z_1, \ldots, z_4$ die $z_a, \ldots, z_d$ nicht eindeutig bestimmt sind, wie man am Beispiel $(a \circ a) \circ (a \circ a) = a \circ (a \circ a \circ a)$ verifizieren kann.

Die letztere Eigenschaft wird für Spurmonoide als Levi's Lemma nachgewiesen [22]. Die Menge der endlichen Multimengen ist isomorph zu einem Spurmonoid, in dem alle Symbole unabhängig sind; wir „drehen" die Rolle dieser Eigenschaft „herum" und verwenden diese als Axiom, um Multimengen zu charakterisieren. Daher bezeichnen wir diese Eigenschaft als Levi Axiom. Abbildung 6.1 gibt eine graphische Veranschaulichung.

Nun können wir einige weitere Eigenschaften ableiten.

Lemma 27. *Sei* $\mathcal{M}$ *ein Modell für* (AC1),(Irred),(NullTeil) *und* (Levi). *Dann gilt für alle* $\dot{\mathcal{Z}} \in \mathcal{M}_{\mathit{fin}}(\mathit{Fluent}^{\mathcal{M}})$*:*

$$(\forall\, x, y \in \mathit{State}^{\mathcal{M}})\ \ (\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}) = x \circ^{\mathcal{M}} y\ \rightarrow \\ (\exists\, \dot{\mathcal{X}}, \dot{\mathcal{Y}} \in \mathcal{M}_{\mathit{fin}}(\mathit{Fluent}^{\mathcal{M}}))\ \ (\dot{\mathcal{X}} \dot{\cup} \dot{\mathcal{Y}} = \dot{\mathcal{Z}}\ \wedge\ \varrho(\dot{\mathcal{X}}) = x\ \wedge\ \varrho(\dot{\mathcal{Y}}) = y)). \quad (6.4)$$

Beweis. Der Beweis erfolgt durch wohlfundierte Induktion über $\mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$ bezüglich der Teilmengenrelation $\dot{\subset}$.

Angenommen (6.4) ist für alle $\dot{\mathcal{Z}}' \dot{\subset} \dot{\mathcal{Z}}$ erfüllt. Seien $x, y \in \mathit{State}^{\mathcal{M}}$ so dass $\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}) = x \circ^{\mathcal{M}} y$. Es werden nun zwei Fälle unterschieden:

$\dot{\mathcal{Z}} = \dot{\emptyset}$**:** Da $\mathcal{M}$ (NullTeil) erfüllt, gilt dann $x = y = \emptyset$. Da $\varrho^{\mathcal{M}}(\dot{\emptyset}) = \emptyset$ wird (6.4) durch $\dot{\mathcal{X}} = \dot{\mathcal{Y}} = \dot{\emptyset}$ erfüllt.

$\dot{\mathcal{Z}} \neq \dot{\emptyset}$**:** Sei $f \in \mathit{Fluent}^{\mathcal{M}}$ ein Fluent aus $\dot{\mathcal{Z}}$ sowie $\dot{\mathcal{Z}}' = \dot{\mathcal{Z}} \setminus \dot{\{}f\dot{\}}$. Es gilt also

$$\varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) \circ^{\mathcal{M}} \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}') = x \circ^{\mathcal{M}} y\ .$$

Wegen (Levi) gibt es nun $z_a, z_b, z_c, z_d \in \mathit{State}^{\mathcal{M}}$, so dass

$$f = z_a \circ^{\mathcal{M}} z_b \qquad x = z_a \circ^{\mathcal{M}} z_c$$
$$\dot{\mathcal{Z}}' = z_c \circ^{\mathcal{M}} z_d \qquad y = z_b \circ^{\mathcal{M}} z_d.$$

Wegen $\varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) = f$ werden nach (Irred) abermals 2 Fälle unterschieden:

$z_b = \emptyset$**,** und damit $z_a = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}})$. Da $\dot{\mathcal{Z}}' \dot{\subset} \dot{\mathcal{Z}}$ existieren wegen $\dot{\mathcal{Z}}' = z_c \circ^{\mathcal{M}} z_d$ entsprechend der Induktionsannahme $\dot{\mathcal{Z}}_c, \dot{\mathcal{Z}}_d \in \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$, so dass $\dot{\mathcal{Z}}_c \dot{\cup} \dot{\mathcal{Z}}_d = \dot{\mathcal{Z}}'$, $\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_c) = z_c$ und $\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_d) = z_d$. Also ist $x = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) \circ^{\mathcal{M}} \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_c) = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}} \dot{\cup} \dot{\mathcal{Z}}_c)$ und $y = \emptyset \circ^{\mathcal{M}} \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_d) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_d)$, und da auch $\dot{\{}f\dot{\}} \dot{\cup} \dot{\mathcal{Z}}_c \dot{\cup} \dot{\mathcal{Z}}_d = \dot{\{}f\dot{\}} \dot{\cup} \dot{\mathcal{Z}}' = \dot{\mathcal{Z}}$ gilt, ist (6.4) für z erfüllt.

$z_a = \emptyset$**.** Der Beweis dieses Falls erfolgt symmetrisch zum vorhergehenden Fall.

Per Induktionsschluss ergibt sich nun das Lemma. ∎

Nun können wir zeigen, dass $\mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$ durch $\varrho^{\mathcal{M}}$ in $\mathit{State}^{\mathcal{M}}$ eingebettet wird, d.h. nicht nur ein Homomorphismus, sondern auch injektiv ist, d.h. verschiedene Multimengen über $\mathit{Fluent}^{\mathcal{M}}$ durch verschiedene Elemente von $\mathit{State}^{\mathcal{M}}$ dargestellt werden.

Lemma 28. *Für jedes Modell von* (AC1)*,*(Irred) *und* (Levi) *gilt:*

$$(\forall\, \dot{\mathcal{Z}}, \dot{\mathcal{Z}}' \in \mathcal{M}_{\mathit{fin}}(\mathit{Fluent}^{\mathcal{M}})) \quad \left(\dot{\mathcal{Z}} \neq \dot{\mathcal{Z}}' \;\rightarrow\; \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}) \neq \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}')\right) \quad .$$

Beweis. Der Beweis erfolgt per Induktion über die Summe $s = |\dot{\mathcal{Z}}| + |\dot{\mathcal{Z}}'|$ der Kardinalitäten von $\dot{\mathcal{Z}}$ und $\dot{\mathcal{Z}}'$. Nehmen wir an, dass

$$\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}') \;\rightarrow\; \dot{\mathcal{Z}} = \dot{\mathcal{Z}}' \tag{i}$$

für $|\dot{\mathcal{Z}}| + |\dot{\mathcal{Z}}'| < s$ erfüllt ist. Ohne Beschränkung der Allgemeinheit sei $|\dot{\mathcal{Z}}| \leq |\dot{\mathcal{Z}}'|$. Wir unterscheiden 3 Fälle:

$\dot{\mathcal{Z}} = \dot{\emptyset}$ **und** $\dot{\mathcal{Z}}' = \dot{\emptyset}$**:** Damit ist (i) trivialerweise erfüllt.

$\dot{\mathcal{Z}} = \dot{\emptyset}$ **und** $\dot{\mathcal{Z}}' \neq \dot{\emptyset}$**:** Sei $f \in \dot{\mathcal{Z}}'$. Wenn $\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}')$ gelten würde, dann wäre also $\varrho^{\mathcal{M}}(\dot{\emptyset}) = \emptyset = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}} \dot{\cup} (\dot{\mathcal{Z}}' \dot{\setminus} \dot{\{}f\dot{\}})) = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) \circ \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \dot{\setminus} \dot{\{}f\dot{\}}) = f \circ \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \dot{\setminus} \dot{\{}f\dot{\}})$, und wegen (NullTeil) $f = \emptyset$, was (Irred) widerspricht. Daher ist (i) erfüllt.

$\dot{\mathcal{Z}} \neq \dot{\emptyset}$ **und** $\dot{\mathcal{Z}}' \neq \dot{\emptyset}$**:** Sei $f \in \dot{\mathcal{Z}}$ und $f' \in \dot{\mathcal{Z}}'$. Nehmen wir an, dass $\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}')$. Damit gilt also

$$\varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) \circ \varrho^{\mathcal{M}}(\dot{\mathcal{Z}} \dot{\setminus} \dot{\{}f\dot{\}}) = \varrho^{\mathcal{M}}(\dot{\{}f'\dot{\}}) \circ \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \dot{\setminus} \dot{\{}f'\dot{\}})$$

Wegen (Levi) gibt es $z_a, z_b, z_c, z_d \in \mathit{State}^{\mathcal{M}}$, so dass

$$\begin{aligned} \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) &= z_a \circ^{\mathcal{M}} z_b & \qquad \varrho^{\mathcal{M}}(\dot{\{}f'\dot{\}}) &= z_a \circ^{\mathcal{M}} z_c \\ \varrho^{\mathcal{M}}(\dot{\mathcal{Z}} \dot{\setminus} \dot{\{}f\dot{\}}) &= z_c \circ^{\mathcal{M}} z_d & \qquad \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \dot{\setminus} \dot{\{}f'\dot{\}}) &= z_b \circ^{\mathcal{M}} z_d \quad . \end{aligned} \tag{ii}$$

Wir unterscheiden nun unter Beachtung von (Irred) abermals 3 Fälle:

$\dot{\mathcal{Z}} = \dot{\{}f\dot{\}}$**:** Damit ist $\dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}} = \dot{\emptyset}$. Wegen (NullTeil) ist also $z_c = z_d = \emptyset$, daher ist $z_a = f'$ und folglich wegen (Irred) $f = f'$ und $z_b = \emptyset$ und somit $\varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f'\dot{\}}) = \emptyset$. Aus $f = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}})$ und $f' = \varrho^{\mathcal{M}}(\dot{\{}f'\dot{\}})$ folgt nach Induktionsvoraussetzung $\dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f'\dot{\}} = \dot{\emptyset}$ und damit $\dot{\mathcal{Z}}' = \dot{\{}f'\dot{\}} = \dot{\{}f\dot{\}} = \dot{\mathcal{Z}}$.

$\dot{\{}f\dot{\}} \dot{\subset} \dot{\mathcal{Z}} \wedge z_a = \emptyset$**:** Damit ist $z_b = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}})$, $z_c = \varrho^{\mathcal{M}}(\dot{\{}f'\dot{\}})$ und damit

$$\varrho^{\mathcal{M}}(\dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}}) = \varrho^{\mathcal{M}}(\dot{\{}f'\dot{\}}) \circ^{\mathcal{M}} z_d \qquad \text{und} \qquad \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f'\dot{\}}) = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) \circ^{\mathcal{M}} z_d \ .$$

Auf beide Gleichungen kann man nun Lemma 27 anwenden: Es gibt also $\dot{\mathcal{Z}}_f, \dot{\mathcal{Z}}_{f'}, \dot{\mathcal{Z}}_{d,1}, \dot{\mathcal{Z}}_{d,2} \in \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$, so dass

$$\begin{aligned} \dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}} &= \dot{\mathcal{Z}}_f \mathbin{\dot{\cup}} \dot{\mathcal{Z}}_{d,1} & \qquad \dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f'\dot{\}} &= \dot{\mathcal{Z}}_{f'} \mathbin{\dot{\cup}} \dot{\mathcal{Z}}_{d,2} \\ f &= \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_f) & f' &= \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_{f'}) \\ z_d &= \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_{d,1}) & z_d &= \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_{d,2}) \end{aligned}$$

Nach Induktionsvoraussetzung muss nun $\dot{\mathcal{Z}}_f = \dot{\{}f\dot{\}}$ und $\dot{\mathcal{Z}}_{f'} = \dot{\{}f'\dot{\}}$ gelten, womit $\dot{\mathcal{Z}}_{d,1} = \dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f, f'\dot{\}}$ und $\dot{\mathcal{Z}}_{d,2} = \dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f, f'\dot{\}}$. Da $|\dot{\mathcal{Z}}_{d,1}| + |\dot{\mathcal{Z}}_{d,2}| < |\dot{\mathcal{Z}}| + |\dot{\mathcal{Z}}'|$ und $z_d = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_{d,1}) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}_{d,1})$ kann abermals die Induktionsvoraussetzung angewandt werden, so dass $\dot{\mathcal{Z}}_{d,1} = \dot{\mathcal{Z}}_{d,2}$ folgt, und daher $\dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f, f'\dot{\}} = \dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f, f'\dot{\}}$, und somit $\dot{\mathcal{Z}} = \dot{\mathcal{Z}}'$.

$\dot{\{}f\dot{\}} \dot{\subset} \dot{\mathcal{Z}} \wedge z_b = \emptyset$**:** Damit ist $z_a = \varrho^{\mathcal{M}}(\dot{\{}f\dot{\}})$, woraus sich wegen (Irred) $z_c = \emptyset$ und $\varrho^{\mathcal{M}}(\dot{\{}f\dot{\}}) = \varrho^{\mathcal{M}}(\dot{\{}f'\dot{\}})$ ergibt, d.h. $f = f'$ nach Definition von $\varrho^{\mathcal{M}}$. Wegen (ii) resultiert also $z_d = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}}) = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}})$, und da $|\dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}}| + |\dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}}| < |\dot{\mathcal{Z}}| + |\dot{\mathcal{Z}}'|$ folgt nach Induktionsvoraussetzung $\dot{\mathcal{Z}} \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}} = \dot{\mathcal{Z}}' \mathbin{\dot{\setminus}} \dot{\{}f\dot{\}}$ und somit $\dot{\mathcal{Z}} = \dot{\mathcal{Z}}'$.

Damit ist auch in diesem Fall (i) erfüllt.

Per Induktionsschluss folgt also, dass (i) für alle $\dot{\mathcal{Z}}, \dot{\mathcal{Z}}' \in \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$ gilt. Dies ist äquivalent zum zu beweisenden Lemma. ∎

Die bisher präsentierten Axiome sind aber für das Formalisieren unserer Intuition der Zustände noch ungenügend, da sie Modelle zulassen, in denen Gleichungen wie $z \circ f = z$ erfüllbar sind, während es keine endliche Multimenge $\dot{\mathcal{Z}}$ von Fluenten gibt, so dass $\dot{\mathcal{Z}} \mathbin{\dot{\cup}} \dot{\{}f\dot{\}} = \dot{\mathcal{Z}}$.

Beobachtung 29. (AC1), (Irred), (NullTeil), (Levi) *und* $(\exists\, z : \mathit{State}, f : \mathit{Fluent})\ z = f \circ z$ *sind erfüllbar.*

Beweis. Ein Modell kann wie folgt konstruiert werden: Die Sorte *State* wird interpretiert als die Menge der natürlichen Zahlen $\mathbb{N}$ zuzüglich einem speziellen Element ω, die Sorte *Fluent* wird interpretiert als $\{1\}$, $\emptyset$ als 0 und $\circ$ durch folgende Funktion:

$$x \circ y \mapsto \begin{cases} x + y & \text{wenn } x \neq \omega \text{ und } y \neq \omega \\ \omega & \text{sonst.} \end{cases}$$

Es ist leicht zu sehen dass (AC1), (Irred) und (NullTeil) erfüllt sind.

(Levi) wird für $z_1 + z_2 = z_3 + z_4$ erfüllt durch

$$\begin{aligned} z_a &= \min(z_1, z_3) & z_c &= z_2 - z_d \\ z_b &= z_3 - z_a & z_d &= \min(z_2, z_4) \end{aligned}$$

falls $z_1, z_2, z_3, z_4 \neq \omega$. Nehmen dagegen an, dass $z_1 \circ z_2 = z_3 \circ z_4 = \omega$. Ohne Beschränkung der Allgemeinheit sei $z_1 = z_3 = \omega$. Wenn $z_2 \neq \omega$ und $z_3 \neq \omega$, dann wird (Levi) erfüllt durch

$$\begin{aligned} z_a &= \omega & z_c &= z_2 - z_d \\ z_b &= z_4 - z_d & z_d &= \min(z_2, z_4) \ . \end{aligned}$$

Wenn nur eins von z_2, z_4 gleich ω ist, ohne Beschränkung der Allgemeinheit z_2, dann wird (Levi) erfüllt durch

$$\begin{aligned} z_a &= \omega & z_c &= \omega \\ z_b &= z_4 & z_d &= 0 \ . \end{aligned}$$

und falls $z_1, z_2, z_3, z_4 = \omega$ wird (Levi) durch $z_a, z_b, z_c, z_d = \omega$ erfüllt.

$z = f \circ z$ ist erfüllbar durch $z = \omega$ und $f = 1$. ∎

Dieses Problem tritt dadurch auf, dass im angegebenen Modell *State* nicht isomorph zur Menge der endlichen Multimengen über *Fluent* ist; unter endlichen Multimengen ist $z = z \circ f$ nur für $f = \emptyset$ erfüllbar, was durch (Irred) ausgeschlossen wird. Es wird daher noch ein Induktionsaxiom eingeführt, das dieses Problem ausschließt. Dieses ist in Prädikatenlogik 2. Stufe formuliert: Die Quantifizierung $(\forall\, P : \langle \mathit{State} \rangle)$ erfolgt über alle Prädikate P über der Sorte *State*.

$$(\forall\, P : \langle \mathit{State} \rangle) \ \Big[P(\emptyset) \ \wedge \ (\forall\, f : \mathit{Fluent}, z : \mathit{State}) \ \ (P(z) \ \rightarrow \ P(z \circ f)) \\ \rightarrow \ (\forall\, z : \mathit{State}) \ \ P(z) \Big] \quad \text{(Ind)}$$

Theorem 30. *Für alle Modelle von* (AC1),(Irred), (Levi) *und* (Ind) *bezüglich einer ordnungssortierten Gleichheitslogik mit einer Fluentkalkülsignatur ist* $\varrho^{\mathcal{M}}$ *ein Isomorphismus von* $\langle \mathcal{M}_{\mathit{fin}}(\mathit{Fluent}^{\mathcal{M}}), \dot{\emptyset}, \dot{\cup} \rangle$ *und* $\langle \mathit{State}^{\mathcal{M}}, \emptyset^{\mathcal{M}}, \circ^{\mathcal{M}} \rangle$.

Beweis. Wegen Lemma 26 ist $\varrho^{\mathcal{M}}$ ein Homomorphismus; wegen Lemma 28 ist $\varrho^{\mathcal{M}}$ injektiv. Es ist also nur noch zu zeigen, dass $\varrho^{\mathcal{M}}$ auch surjektiv ist.

Betrachten wir nun eine einstellige Relation P über der Domäne von $\mathcal{M}$ bezüglich der Sorte *State*, so dass $z \in P$ genau dann, wenn es ein $\dot{\mathcal{Z}} \in \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$ gibt, für

das $z = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}})$. Es gilt nun $P(\emptyset^{\mathcal{M}})$, da $\varrho^{\mathcal{M}}(\dot{\emptyset}) = \emptyset^{\mathcal{M}}$, und auch $(\forall\ f : \mathit{Fluent}, z : \mathit{State})\ (P(z) \rightarrow P(z \circ f))$: Wenn es ein $\dot{\mathcal{Z}} \in \mathcal{M}_{\text{fin}}(\mathit{Fluent}^{\mathcal{M}})$ mit $z = \varrho^{\mathcal{M}}(\dot{\mathcal{Z}})$ gibt, dann gilt $\varrho^{\mathcal{M}}(\{\dot{f}\} \dot{\cup} \dot{\mathcal{Z}}) = z \circ^{\mathcal{M}} f$. Damit muss also P für alle $z \in \mathit{State}^{\mathcal{M}}$ erfüllt sein, d.h. $\varrho^{\mathcal{M}}$ ist surjektiv. ∎

Die bewiesenen Lemmata wären allerdings nutzlos, wenn der angegebene Satz von Axiomen widersprüchlich wäre. Daher weisen wir nach, dass dies nicht der Fall ist. Die im Beweis von Satz 31 angegebene Interpretation ist dabei die **„kanonische" Interpretation** der Axiome in dem Sinne, dass sie die Intuition hinter den Axiomen direkt widerspiegelt, und alle anderen Interpretationen wegen Theorem 30 isomorph dazu sind. Bei Betrachtungen von Modellen des Fluentkalküls mit dem Axiomensatz (AC1),(Irred), (NullTeil), (Levi) und (Ind) kann man sich also ohne Beschränkung der Allgemeinheit auf Modelle der im folgenden präsentierten Form einschränken.

Satz 31. (AC1),(Irred), (NullTeil), (Levi) *und* (Ind) *sind widerspruchsfrei.*

Beweis. Wir weisen nach, dass eine Interpretation $\mathcal{M}$, die die Sorte *Fluent* als $\left\{\{\dot{a}\} \mid a \in A\right\}$ für eine beliebige Menge A interpretiert, sowie die Sorte *State* als $\mathcal{M}_{\text{fin}}(A)$, $\circ$ als $\dot{\cup}$ und $\emptyset$ als $\dot{\emptyset}$, ein Modell für die genannten Axiome ist.

Wie leicht zu sehen ist, sind (AC1), (NullTeil) und (Irred) erfüllt.

(Levi) wird durch

$$z_a = z_1 \dot{\cap} z_3 \qquad z_c = z_2 \dot{\setminus} z_d$$
$$z_b = z_1 \dot{\setminus} z_a \qquad z_d = z_2 \dot{\cap} z_4$$

erfüllt: Es ergibt sich sofort

$$z_a \dot{\cup} z_b = (z_1 \dot{\cap} z_3) \dot{\cup} \left(z_1 \dot{\setminus} (z_1 \dot{\cap} z_3)\right) = z_1 \qquad \text{sowie}$$
$$z_c \dot{\cup} z_d = \left(z_2 \dot{\setminus} (z_2 \dot{\cap} z_4)\right) \dot{\cup} (z_2 \dot{\cap} z_4) = z_2\ .$$

Um die restlichen beiden Teilgleichungen von (Levi) nachzuweisen, formen wir $z_3 \dot{\setminus} (z_1 \dot{\cap} z_3)$ zunächst um:

$$z_3 \dot{\setminus} (z_1 \dot{\cap} z_3) = \left(z_3 \dot{\cup} \underline{z_4}\right) \dot{\setminus} \left((z_1 \dot{\cap} z_3) \dot{\cup} \underline{z_4}\right)$$

(wegen $X \dot{\setminus} Y = (X \dot{\cup} Z) \dot{\setminus} (Y \dot{\cup} Z)$ mit $Z = z_4$)

$$= \left(z_3 \dot{\cup} z_4\right) \dot{\setminus} \left((z_1 \dot{\cup} \underline{z_4}) \dot{\cap} (z_3 \dot{\cup} \underline{z_4})\right)$$

(wegen $(X \dot\cap Y) \dot\cup Z = (X \dot\cup Z) \dot\cap (Y \dot\cup Z)$ mit $Z = z_4$)

$$= \big(\underline{z_1 \dot\cup z_2}\big) \dot\setminus \big((z_1 \dot\cup z_4) \dot\cap (\underline{z_1 \dot\cup z_2})\big)$$

(wegen $z_1 \dot\cup z_2 = z_3 \dot\cup z_4$)

$$= \big(\underline{z_2} \dot\cup \underline{z_1}\big) \dot\setminus \big((\underline{z_2} \dot\cup \underline{z_1}) \dot\cap (\underline{z_4} \dot\cup \underline{z_1})\big)$$

(mehrfache Anwendung von Kommutativität von $\dot\cup$)

$$= \big(z_2 \dot\cup z_1\big) \dot\setminus \big(\underline{(z_2 \dot\cap z_4) \dot\cup z_1}\big)$$

(wegen $(X \dot\cap Y) \dot\cup Z = (X \dot\cup Z) \dot\cap (Y \dot\cup Z)$ mit $Z = z_1$)

$$= z_2 \dot\setminus (z_2 \dot\cap z_4)$$

(wegen $X \dot\setminus Y = (X \dot\cup Z) \dot\setminus (Y \dot\cup Z)$ mit $Z = z_1$)

Zusammengefasst: $z_3 \dot\setminus (z_1 \dot\cap z_3) = z_2 \dot\setminus (z_2 \dot\cap z_4)$. Damit ergibt sich nun:

$$z_a \dot\cup z_c = (z_1 \dot\cap z_3) \dot\cup \Big(z_2 \dot\setminus (z_2 \dot\cap z_4)\Big) = (z_1 \dot\cap z_3) \dot\cup \Big(z_3 \dot\setminus (z_1 \dot\cap z_3)\Big) = z_3$$

und analog:

$$z_b \dot\cup z_d = \Big(z_1 \dot\setminus (z_1 \dot\cap z_3)\Big) \dot\cup (z_2 \dot\cap z_4) = \Big(z_4 \dot\setminus (z_2 \dot\cap z_4)\Big) \dot\cup (z_2 \dot\cap z_4) = z_4 \ .$$

Es bleibt zu beweisen, dass (Ind) erfüllt ist. Dies erfolgt durch wohlfundierte Induktion über $\mathit{State}^{\mathcal{M}} = \mathcal{M}_{\text{fin}}(A)$ bezüglich $\dot\subset$.

Betrachten wir ein einstelliges Prädikat P über *State*, das

$$P(\emptyset) \ \wedge \ (\forall\, f : \mathit{Fluent}, z : \mathit{State}) \ \ (P(z) \ \rightarrow \ P(z \circ f)) \tag{i}$$

erfüllt. Nehmen wir nun an, dass P für alle $z' \dot\subset z$ erfüllt ist. Es können nun 2 Fälle unterschieden werden:

$z = \dot\emptyset$**:** Damit ist $P(z)$ wegen (i) erfüllt.

$z \neq \dot\emptyset$**:** Es gibt also ein $f \in A$ und ein $z' \in \mathcal{M}_{\text{fin}}(A)$, so dass $z = \dot{\{}f\dot\} \dot\cup z'$. Da $z' \dot\subset z$ gilt $P(z')$ und wegen (i) gilt also auch $P(z' \circ \dot{\{}f\dot\})$, und da $z' \circ \dot{\{}f\dot\} = z$ gilt auch $P(z)$.

Per Induktionsschluss folgt also, dass $P(z)$ für alle z aus $\mathcal{M}_{\text{fin}}(A) = \mathit{State}^{\mathcal{M}}$ erfüllt ist, d.h. (Ind) ist erfüllt. ∎

Wir fassen nun die beschriebenen Axiome zu einem Axiomsatz zusammen, der das Fundament des Fragments des Fluentkalküls, das in dieser Arbeit betrachtet wird, bildet:

Definition 32.

$$\mathcal{F}_{mset} = \{\text{(AC1)}, \text{(Irred)}, \text{(NullTeil)}, \text{(Levi)}, \text{(Ind)}\} \ .$$

Zur Übersicht sind in Tabelle (6.1) auf Seite 71 noch einmal alle grundlegenden Fluentkalkül Axiome angegeben.

6.4. Anwendung der $\mathcal{F}_{mset}$ -Axiome

Im Gegensatz zu EUNA legt $\mathcal{F}_{mset}$ nur die Gleichheiten und Ungleichheiten von Zuständen, nicht aber die von Fluenten fest. Es ist also i.A. zusätzlich zu $\mathcal{F}_{mset}$ noch eine Menge von Gleichheitsaxiomen, die die Fluenten betreffen, erforderlich. Darin liegt auch die größere Flexibilität gegenüber EUNA begründet (vergleiche Beobachtung 25) – Gleichungen wie $\texttt{Ungefärbt} = \texttt{Weiß}$ und $\mathit{Color}(\texttt{A}) = \texttt{Rot}$ sind problemlos in die Domänenbeschreibung einfügbar. Um die Situation wie bei EUNA herbeizuführen, dass syntaktisch verschiedene Terme vom Typ *Fluent* sich aus den Axiomen als verschieden ableiten lassen, ist es nötig, Axiome für **Namenseindeutigkeit** (engl. **unique name assumption**, **UNA**) hinzuzunehmen. In unserem Beispiel der Welt der Blöcke (Seite 45) wird dies mit den Zusatzaxiomen

$$\text{UNA}_{\mathit{Blöcke}} = \text{UNA}[\texttt{A}, \texttt{B}, \texttt{C}, \ldots] \cup \text{UNA}[\mathit{Clear}, \mathit{On}, \mathit{OnTable}, \mathit{PutOn}]$$

erreicht, wobei die Abkürzung $\text{UNA}[f_1, \ldots, f_n]$ für eine Anzahl von Funktionssymbolen entsprechend [6] definiert ist als:

$$\begin{aligned}\text{UNA}[f_1, \ldots, f_n] = \{\ f_i(\vec{x}) \neq f_j(\vec{y}) \ \mid i \neq j,\ i, j = 1 \ldots n\} \cup \\ \{\ f_i(\vec{x}) = f_i(\vec{y}) \ \rightarrow \ \vec{x} = \vec{y} \ \mid i = 1 \ldots n\} \ , \quad (6.5)\end{aligned}$$

d.h.

$$\begin{aligned}\text{UNA}_{\mathit{Blöcke}} = \ & \{ \ \texttt{A} \neq \texttt{B}, \ \texttt{A} \neq \texttt{C}, \ \texttt{B} \neq \texttt{C}, \ \ldots \ \} \cup \\ & \{ \ \mathit{Clear}(x) \neq \mathit{On}(y_1, y_2), \ \mathit{Clear}(x) \neq \mathit{OnTable}(y), \ \ldots \ \} \cup \\ & \{ \ \mathit{Clear}(x) = \mathit{Clear}(y) \ \rightarrow \ x = y, \\ & \quad \mathit{On}(x_1, x_2) = \mathit{On}(y_1, y_2) \ \rightarrow \ x_1 = y_1 \ \wedge \ x_2 = y_2, \ \ldots \ \} \ .\end{aligned}$$

Damit ergeben sich zu EUNA (Abschnitt 6.2) analoge Resultate:

Beobachtung 33.

$$\mathcal{F}_{mset} \cup \mathit{UNA}_{Blöcke} \models \neg\,(\exists\, z)\;\; \mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A},\mathtt{B}) = \mathit{Clear}(\mathtt{B}) \circ z$$

Beweis. Nehmen wir an, dass in einem Modell von $\mathcal{F}_{mset} \cup \mathrm{UNA}_{Blöcke}$ für ein $z \in \mathit{State}$ gilt, dass

$$\mathit{Clear}(\mathtt{A}) \circ \mathit{On}(\mathtt{A},\mathtt{B}) = \mathit{Clear}(\mathtt{B}) \circ z \;\; . \tag{i}$$

Wenden wir darauf (Levi) an:

$$\mathit{Clear}(\mathtt{A}) = z_a \circ z_b \qquad \mathit{Clear}(\mathtt{B}) = z_a \circ z_c \tag{ii}$$

$$\mathit{On}(\mathtt{A},\mathtt{B}) = z_c \circ z_d \qquad z = z_b \circ z_d \;\; . \tag{iii}$$

Nach (Irred) kann man nun bezüglich (ii) zwei Fälle unterscheiden:

$z_a = \mathit{Clear}(\mathtt{A})$ **und** $z_b = \emptyset$**:** Damit müsste $\mathit{Clear}(\mathtt{B}) = \mathit{Clear}(\mathtt{A}) \circ z_c$ gelten. Nach (Irred) ist nun $\mathit{Clear}(\mathtt{A}) \neq \emptyset$, so dass $z_c = \emptyset$ und somit $\mathit{Clear}(\mathtt{B}) = \mathit{Clear}(\mathtt{A})$. Dies widerspricht $\mathrm{UNA}_{Blöcke}$.

$z_a = \emptyset$ **und** $z_b = \mathit{Clear}(\mathtt{A})$ **und** $z_c = \mathit{Clear}(\mathtt{B})$**:** Damit müsste $\mathit{On}(\mathtt{A},\mathtt{B}) = \mathit{Clear}(\mathtt{B}) \circ z_d$ gelten, woraus nach (Irred) $\mathit{On}(\mathtt{A},\mathtt{B}) = \mathit{Clear}(\mathtt{B})$ folgt, was ebenfalls $\mathrm{UNA}_{Blöcke}$ widerspricht.

■

Wie aus dem obigen Beispiel zu entnehmen ist, ist die direkte Anwendung der gegebenen Axiome ein recht weitschweifiger Prozess. Daher geben wir im Folgenden zwei Schlussregeln an, die dies z.T. stark vereinfachen können.

Satz 34 (Anullierungs-Regel)**.** *In allem Modellen von* (AC1)*,* (Irred)*,* (NullTeil) *und* (Levi) *gilt:*

$$(\forall\, z, z' : \mathit{State}, f : \mathit{Fluent})\;\; f \circ z = f \circ z' \;\rightarrow\; z = z' \;\; . \tag{Cancel}$$

Beweis. Nehmen wir an, dass $f \circ z = f \circ z'$. Wegen (Levi) finden wir z_a, z_b, z_c, z_d so dass

$$f = z_a \circ z_b \qquad z = z_c \circ z_d \tag{i}$$

$$f = z_a \circ z_c \qquad z' = z_b \circ z_d \;\; . \tag{ii}$$

Nach (Irred) können wir nun 2 Fälle unterscheiden:

$z_a = \emptyset$**:** (i), (ii) und (AC1) implizieren $z_b = z_c = f$, und somit $z = f \circ z_d$ und $z' = f \circ z_d$. Es folgt also $z = z'$.

$z_b = \emptyset$ **:** (i) und(AC1) ergeben $z_a = f$. Unter Anwendung von (Irred) ergibt sich $f \neq \emptyset$; damit ergibt abermaliges Anwenden von (Irred) auf $f = z_a \circ z_c$ dass $z_c = \emptyset$. Folglich gilt wegen (i) und (ii) $z = \emptyset \circ z_d = z'$.

■

Diese Regel erlaubt es, gleiche Fluenten auf beiden Seiten einer Gleichung zu streichen.

Satz 35 (Verteilungs-Regel)**.** *In allem Modellen von* (AC1)*,* (Irred)*,* (NullTeil) *und* (Levi) *gilt:*

$$(\forall\, f_1, f_2 : \mathit{Fluent}, z : \mathit{State})\ \ f_1 \neq f_2 \ \rightarrow\ \big[\mathit{Holds}(f_1, f_2 \circ z)\ \rightarrow\ \mathit{Holds}(f_1, z)\big]\ .\ \text{(Distrib)}$$

Beweis. Nehmen wir an, dass $f_1 \neq f_2$ und $\mathit{Holds}(f_1, f_2 \circ z)$ gelten. Nach der Definition von *Holds* finden wir also ein z' so dass $f_1 \circ z' = f_2 \circ z$, und wegen (Levi) finden wir z_a, z_b, z_c, z_d so dass

$$f_1 = z_a \circ z_b \qquad z' = z_c \circ z_d \qquad \text{(i)}$$
$$f_2 = z_a \circ z_c \qquad z = z_b \circ z_d\ . \qquad \text{(ii)}$$

Nach (Irred) können wir 2 Fälle unterscheiden:

$z_a = f_1 \wedge z_b = \emptyset$ **:** Wegen (ii) müsste dann $f_2 = f_1 \circ z_c$ gelten, was wegen $f_1 \neq f_2$ (Irred) widerspricht.

$z_a = \emptyset \wedge z_b = f_1$ **:** Damit folgt aus (ii) $z = f_1 \circ z_d$, und somit $\mathit{Holds}(f_1, z)$.

■

Die Verteilungs-Regel gestattet nun die unmittelbare Herleitung von Beobachtung 33 aus $\mathit{Clear}(\mathtt{B}) \neq \mathit{Clear}(\mathtt{A})$ und $\mathit{Clear}(\mathtt{B}) \neq \mathit{On}(\mathtt{A}, \mathtt{B})$ (beide Resultat von $\mathrm{UNA}_{Blöcke}$). Zur weiteren Illustration weisen wir nach, dass Gleichung (6.1) auf Seite 48, die Bestandteil von EUNA für die Welt der Blöcke ist, ebenfalls unter $\mathrm{UNA}_{Blöcke} \cup \mathcal{F}_{mset}$ gilt.

Beobachtung 36. *Es gilt:*

$$\mathit{UNA}_{Blöcke} \cup \mathcal{F}_{mset} \models (\forall\, x, z)\ \ \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) = \mathit{Clear}(x) \circ z\ \rightarrow$$
$$[x = \mathtt{A}\ \wedge\ z = \mathit{Clear}(\mathtt{B})] \quad \vee \quad [x = \mathtt{B}\ \wedge\ z = \mathit{Clear}(\mathtt{A})]\ .$$

Beweis. Betrachten wir ein Modell von $\mathrm{UNA}_{Blöcke} \cup \mathcal{F}_{mset}$ und eine Belegung für die Variablen x und z , so dass

$$\mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{B}) = \mathit{Clear}(x) \circ z\ . \qquad \text{(i)}$$

Wir können nun drei Fälle unterscheiden:

$x = \mathtt{A}$: Es gilt also $\mathit{Clear}(x) = \mathit{Clear}(\mathtt{A})$. Wegen (Cancel) ergibt sich $z = \mathit{Clear}(\mathtt{B})$; damit gilt die Beobachtung.

$x = \mathtt{B}$: Es gilt also $\mathit{Clear}(x) = \mathit{Clear}(\mathtt{B})$. Wegen (Cancel) und (AC1) ergibt sich $z = \mathit{Clear}(\mathtt{A})$; damit gilt die Beobachtung.

$x \neq \mathtt{A}$ und $x \neq \mathtt{B}$: Entsprechend $\mathrm{UNA}_{Blöcke}$ gilt $\mathit{Clear}(\mathtt{A}) \neq \mathit{Clear}(x)$ und $\mathit{Clear}(\mathtt{B}) \neq \mathit{Clear}(x)$. Zweifache Anwendung von (Distrib) ergibt nun $\mathit{Holds}(\mathit{Clear}(x), \emptyset)$, was wegen (Irred) (NullTeil) widerspricht.

■

6.5. Unifikationsvollständigkeit von $\mathcal{F}_{mset}$

In der praktischen Anwendung deduktiver Beschreibungen von Problemen ist es natürlich von großer Bedeutung, ein effizientes Inferenzverfahren anzuwenden, das insbesondere auch negative Aussagen erlaubt. Früheren Versionen des Fluentkalküls wurde daher, wie in Abschnitt 6.2 diskutiert, eine (AC1)–unifikationsvollständige Gleichungstheorie EUNA zugrundegelegt, so dass SLDENF-Resolution für die Inferenz verwendet werden kann.

Wie im vorigen Abschnitt in den Beobachtungen 33 und 36 anhand des Beispiels der Welt der Blöcke gezeigt, gelten unter bestimmten Voraussetzungen Gleichungen von EUNA ebenfalls unter $\mathcal{F}_{mset}$. Gibt es Bedingungen, unter denen die Gleichungstheorie $\mathcal{F}_{mset}$ ebenfalls unifikationsvollständig ist? Diese Frage soll in diesem Abschnitt behandelt werden.

Das Konzept der Unifikationsvollständigkeit wurde zuerst in [50] als Verallgemeinerung von Konzepten eingeführt, die von Clark zur theoretischen Begründung von SLDNF-Resolution verwendet wurden [21]. Ohne die dazu verwendeten Konzepte hier genau definieren zu wollen (dazu verweisen wir für den Fall der SLDENF-Resolution z.B. auf [45, 80]): Die für ein logisches Programm durch den Inferenzmechanismus der SLDENF-Resolution unter Zugrundelegung einer Gleichungstheorie E (z.B. (AC1)) berechneten Aussagen sind logische Konsequenzen aus der Vervollständigung [21] des logischen Programmes und einer bezüglich E unifikationsvollständigen Gleichungstheorie E^* (z.B. EUNA). Mit anderen Worten: Die unifikationsvollständige Theorie E^* bildet die semantische Basis für die Deduktion. Wenn wir also, unter noch zu bestimmenden Voraussetzungen, die Unifikationsvollständigkeit von $\mathcal{F}_{mset}$ bezüglich (AC1) nachweisen können, dann kann in diesem Fall SLDENF-Resolution mit Gleichungstheorie (AC1) als Deduktionsmechanismus herangezogen werden, um logische Konsequenzen aus z.B. einer geeigneten Fluentkalkül-Problembeschreibung unter Verwendung von $\mathcal{F}_{mset}$ zu berechnen.[7]

[7] Die in Kapitel 7 generierte Problembeschreibung erfüllt aber aus Gründen der einfachen, adäquaten Darstellung nicht die syntaktischen Bedingungen eines logischen Programms, und müsste daher angepasst werden. Da diese Arbeit sich auf die Verwendung von BDDs als Inferenzmethode konzentriert, ist dies aber nicht

Eine vollständige Behandlung dieses Themas sprengt den Rahmen dieser Arbeit, zumal in späteren Kapiteln ein anderer Inferenzmechanismus unter Nutzung von BDDs diskutiert werden soll. In diesem Kapitel soll daher nur eine kurze Exploration dieses Themas vorgenommen werden, um eine Verbindung zu der alten Axiomatisierung des Fluentkalküls (EUNA) herzustellen. Wir werden die Unifikationsvollständigkeit für ein praktisch bedeutsames Fragment des Fluentkalküls nachweisen, ohne den Anspruch auf Generalität zu erheben.

In [74] wird Unifikationsvollständigkeit wie folgt definiert:

Definition 37. *Sei E eine Gleichungstheorie. Eine konsistente Formelmenge E^* ist **unifikationsvollständig** bezüglich E, wenn sie aus den Axiomen von E, den Standard-Gleichheitsaxiomen, und einer Anzahl von Formeln mit = als einzigem Prädikat besteht, so dass für je zwei Terme s und t mit den Variablen $V = \mathrm{var}(s) \cup \mathrm{var}(t)$ und jede vollständige Menge von E-Unifikatoren $cU_e(s,t)$ gilt:*

$$E^* \models (\forall\, V) \quad \left(s = t \;\rightarrow \bigvee_{\theta \in cU_e(s,t)} (\exists\, V_\theta)\;\; \theta_= \right) \quad , \tag{6.6}$$

wobei $\theta_=$ für eine Substitution $\theta = \{v_1/t_1, v_2/t_2, \ldots, v_n/t_n\}$ die korrespondierende Formel $v_1 = t_1 \;\wedge\; v_2 = t_2 \;\wedge\; \ldots \;\wedge\; v_n = t_n$ bezeichnet, sowie $V_\theta = \mathrm{var}(\theta_=) \setminus V$.

Man beachte, dass für zwei nicht unifizierbare Terme $cU_e(s,t)$ leer ist, so dass die Disjunktion falsch wird, und damit $E^* \models (\forall\, V)\; s \neq t$ gilt. Für eine genauere Diskussion der verwendeten Begriffe siehe [45].

Wir betrachten im folgenden die eine Fluentkalkül-Signatur Σ mit folgenden Zusatzelementen:

SORT	*Object*, *Fluent* $\preceq$ *State*,	
FUN	$o_1, \ldots, o_k$	: *Object*,
	$\emptyset$	: *State*,
	$\circ$	: *State* $\times$ *State* $\rightarrow$ *State*,

sowie eine Menge $f_1, \ldots, f_l$ von Funktionssymbolen des Typs

$$f_i : \underbrace{\textit{Object} \times \cdots \times \textit{Object}}_{\mathrm{ar}(f_i)} \rightarrow \textit{Fluent} \;\; .$$

Fluenten entstehen also durch Kombination eines der Funktionssymbole $f_1, \ldots, f_l$ mit 0 oder mehreren Argumenten der Sorte *Object*, die nur die Symbole $o_1, \ldots, o_k$ enthält. Als Argumente von Fluenten sind also nur Konstanten erlaubt. Eine ähnliche Signatur wurde in Abschnitt 6.4 für die Welt der Blöcke verwendet und entsteht in Kapitel 7 bei Abbildung von

Gegenstand dieser Arbeit. Für die Darstellung von Planungsproblemen als logisches Programm verweisen wir auf die Literatur, z.B. [45, 80, 10, 9].

PDDL, wenn keine weiteren Objekttypen verwendet werden. Da sich die Gleichungstheorie $\mathcal{F}_{mset}$ nur mit den Sorten *Fluent* und *State* befasst, lassen wir die Sorten *Action* und *Sit* der Übersichtlichkeit halber aus unseren Betrachtungen aus; diese können bei Bedarf ergänzt werden.

$\mathcal{F}_{mset}$ für sich allein betrachtet ist noch keine unifikationsvollständige Gleichungstheorie: Z.B. sind die Terme o_1 und o_2 nicht unifizierbar, aber im Gegensatz zu den Forderungen von Definition 37 gilt nicht $\mathcal{F}_{mset} \models o_1 \neq o_2$. Daher fügen wir eine ergänzende Axiomatisierung der Gleichheit für die Sorten *Fluent* und *Object* ein:

$$\mathcal{F}_{uv} = \mathcal{F}_{mset} \cup \text{UNA}[o_1, \ldots, o_k] \cup \left\{ (\forall\, o : \textit{Object}) \bigvee_{i=1\ldots k} o = o_i \right\} \cup \text{UNA}[f_1, \ldots, f_l] \cup$$
$$\left\{ (\forall\, f : \textit{Fluent}) \bigvee_{i=1\ldots l} (\exists\, o_1, \ldots, o_{\text{ar}(f_i)} : \textit{Object})\ f = f_i(o_1, \ldots, o_{\text{ar}(f_i)}) \right\}$$

Zusätzlich zur Namenseindeutigkeit (vergleiche (6.5) auf Seite 58) wird hier noch die Abgeschlossenheit des Universums (engl. **domain closure**) verwendet, die hier garantiert, dass in allen Modellen von $\mathcal{F}_{mset}$ alle Elemente der Sorten *Object* und *Fluent* durch Grundterme darstellbar sind. Zusammen mit der Namenseindeutigkeit folgt, dass jedem Grundterm der Sorten *Object* und *Fluent* genau ein Element von $\textit{Object}^{\mathcal{M}}$ bzw. $\textit{Fluent}^{\mathcal{M}}$ zugeordnet ist und umgekehrt.

Satz 38. *$\mathcal{F}_{uv}$ ist unifikationsvollständig bezüglich* (AC1).

Beweis. Seien s und t zwei Terme der Signatur Σ mit den Variablen $v_1, \ldots, v_m$, $\mathcal{M}$ ein Modell von $\mathcal{F}_{uv}$ und $\mathcal{V}$ eine Variablenbelegung, so dass $\mathcal{M}, \mathcal{V} \models s = t$. Sei weiterhin $cU_e(s,t)$ eine vollständige Menge von (AC1)-Unifikatoren für s und t.

Für die Sorten *Object* und *Fluent* ist (6.6) erfüllt, da für diese Sorten die Clarksche Gleichungstheorie in $\mathcal{F}_{uv}$ enthalten ist. Da diese unifikationsvollständig bezüglich der leeren Gleichungstheorie ist, ist auch hier die Unifikationsvollständigkeit garantiert, da die Gleichungen von (AC1) sich nur auf Sorte *State* beziehen, und Fluenten in Σ keine Teilterme von Sorte *State* haben können.

Nehmen wir also an, dass s und t der Sorte *State* angehören. Wie bereits diskutiert, gibt es nun für die Werte $v_1^{\mathcal{V}}, \ldots, v_m^{\mathcal{V}}$ der Variablen $v_1, \ldots, v_m$ Grundterme $t_1, \ldots, t_m$, so dass $t_i^{\mathcal{M}} = v_i^{\mathcal{V}}$ für alle $i = 1 \ldots m$. Sei nun $\sigma = \{v_1/t_1, \ldots, v_m/t_m\}$. Aufgrund der Substitutivität der Gleichheit gilt nun

$$\mathcal{M}, \mathcal{V} \models s = s\sigma \qquad \text{und} \qquad \mathcal{M}, \mathcal{V} \models t = t\sigma\ ,$$

und somit $\mathcal{M}, \mathcal{V} \models s\sigma = t\sigma$ bzw., da σ eine Grundsubstitution ist,

$$\mathcal{M} \models s\sigma = t\sigma \ .$$

In Σ hat aber jeder Grundterm r der Sorte *State* eine Struktur $r_1 \circ r_2 \circ \ldots r_n$, $n \geq 0$, wobei die r_1 entweder Grundterme der Sorte *Fluent* oder $\emptyset$ sind. Wegen Theorem 30 sind zwei solche Grundterme aber genau dann gleich unter $\mathcal{M}$, wenn die entsprechenden Multimengen $\dot{\{}r_i \mid r_i \neq \emptyset\dot{\}}$ gleich sind. Damit sind diese Grundterme aber auch bezüglich (AC1) gleich, da sie, nachdem man durch Anwendung von (AC1) die Vorkommen von $\emptyset$ eliminiert hat, Permutationen voneinander sind, die man durch weitere Anwendung der Axiome (AC1) ineinander umformen kann. Folglich gilt

$$\text{(AC1)} \models s\sigma = t\sigma \ .$$

Mit anderen Worten: σ ist ein (AC1)-Unifikator von s und t. Da $cU_e(s,t)$ vollständig ist, gibt es aber einen Unifikator $\theta \in cU_e(s,t)$, der allgemeiner oder gleich zu σ unter (AC1) ist, d.h. wir finden eine Substitution ρ, so dass $\sigma =_{\text{(AC1)}} \theta\rho\,|_{\{v_1,\ldots,v_m\}}$.

Sei v_i eine beliebige der Variablen $v_1, \ldots, v_n$. Da σ die Variablenbelegung $\mathcal{V}$ charakterisiert, gilt $\mathcal{M}, \mathcal{V} \models v_i = (v_i\sigma)$ und damit $\mathcal{M}, \mathcal{V} \models v_i = (v_i\theta\rho)$. Dies impliziert die Gültigkeit der schwächeren Existenzaussage $\mathcal{M}, \mathcal{V} \models (\exists\ V_\theta)\ \ v_i = (v_i\theta)$ mit $V_\theta = \mathrm{var}(\theta_=) \setminus \{v_1, \ldots, v_n\}$, da V_θ die freien Variablen in $v_i\theta$ enthält.

Wenn man dies für alle Variablen $v_i \in \{v_1, \ldots, v_n\}$ zusammenfasst, ergibt sich

$$\mathcal{M}, \mathcal{V} \models (\exists\ V_\theta)\ \ [v_1 = (v_1\theta)\ \wedge\ \ldots\ \wedge\ v_n = (v_n\theta)]$$

und somit $\mathcal{M}, \mathcal{V} \models (\exists\ V_\theta)\ \ \theta_=$. Folglich gilt auch $\mathcal{M}, \mathcal{V} \models \bigvee_{\theta \in cU_e(s,t)} (\exists\ V_\theta)\ \ \theta_=$, woraus sich (6.6) ergibt. ∎

Unter geeigneten Bedingungen führt also auch die neue Axiomatisierung $\mathcal{F}_{mset}$ des Fluentkalküls zu Unifikationsvollständigkeit, so dass die Verwendung von SLDENF als Schlussverfahren möglich erscheint. Die genauere Charakterisierung der Bedingungen, unter denen dies möglich ist, ist allerdings Gegenstand zukünftiger Forschung. Für die Anwendung von SLDENF ist zudem noch eine weitere Umstellung der Axiomatisierung des Fluentkalküls erforderlich, da der Einsatz des Funktionssymbols *state* die Unifikationsvollständigkeit verletzt: Die rechte und linke Seite eines Axioms wie $\mathit{state}(\mathrm{S}_0) = \mathit{Clear}(\mathrm{A}) \circ \mathit{OnTable}(\mathrm{A})$ sind nicht unifizierbar, und damit wäre dieses Axiom bei Annahme von Unifikationsvollständigkeit unerfüllbar.

6.6. Zustandsübergangsaxiome (SUA)

Nun sind die Grundlagen gelegt, und wir können uns der Spezifikation der Wirkungen von Aktionen im Fluentkalkül widmen. Dabei geht es also darum, den Zustand $\mathit{state}(\mathit{do}(a, s))$ nach

der Ausführung einer Aktion a in Abhängigkeit vom Zustand $\mathit{state}(s)$ vor der Ausführung der Aktion zu charakterisieren. Die generelle Form der sogenannten Zustandsübergangsaxiome (engl. **State-Update Axiome**, kurz: SUA), die diese Aufgabe im Fluentkalkül erfüllen, ist für eine Aktion a:

$$(\forall\, s : \mathit{State})\ \ (\Delta(s)\ \rightarrow\ \Gamma[\mathit{state}(\mathit{do}(a, s)), \mathit{state}(s)])\ .$$

Hierbei ist $\Delta(s)$ eine von der Situation s bzw. dem Zustand $\mathit{state}(s)$ abhängige Bedingung (die **Vorbedingung**[8] des Zustandsübergangsaxioms), unter der die Formel $\Gamma[\mathit{state}(\mathit{do}(a, s)), \mathit{state}(s)]$ das Verhältnis von $\mathit{state}(\mathit{do}(a, s))$ und $\mathit{state}(s)$ ausdrückt. Es gibt oft mehrere Zustandsübergangsaxiome für eine Aktion: Z.B. könnte ein Zustandsübergangsaxiom ausdrücken, dass, wenn das Licht in Situation s an war (kodiert in $\Delta(s)$), das Licht nach Ausführen der Aktion „Betätige Lichtschalter“ (a) hinterher aus sein wird (Γ), und ein zweites Zustandsübergangsaxiom, dass das Licht hinterher aus ist, wenn es vorher an war.

In Abhängigkeit von der benötigten Ausdrucksstärke gibt es eine Anzahl von grundlegenden Formen von Zustandsübergangsaxiomen, so z.B. wenn nichtdeterministische Aktionen kodiert werden sollen, oder Aktionen mit indirekten Effekten (Ramifikation) [84]. In dieser Arbeit beschränken wir uns auf eine einfache Form, die voraussetzt, dass die Ausführbarkeit und die Effekte der Aktionen deterministisch von dem Zustand $\mathit{state}(s)$ abhängen, und von nichts anderem; zudem beschränken wir uns auf aussagenlogische Fluenten und direkte Effekte. Hier sieht die Form der Zustandsübergangsaxiome wie folgt aus:

$$(\forall)\ \Big[\bigwedge_{i=1\ldots k} \neg\, \mathit{Holds}(f_i^+, s) \bigwedge_{i=1\ldots l} \mathit{Holds}(f_i^-, s) \wedge \Delta_p(s)\ \rightarrow\ \mathit{state}(\mathit{do}(a, s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+\Big]\ , \quad (6.7)$$

wobei $\Delta_p(s)$ eine boolesche Kombination von Formeln der Form $\mathit{Holds}(f, s)$ ist, und $\vartheta^+ = f_1^+ \circ f_2^+ \circ \ldots \circ f_k^+$, $k \geq 0$ alle positiven Effekte der Aktion (d.h. die Fluenten, die durch die Aktion wahr gemacht werden), und $\vartheta^- = f_1^- \circ f_2^- \circ \ldots \circ f_l^-$, $l \geq 0$ alle negativen Effekte (d.h. die Fluenten, die durch die Aktion falsch gemacht werden) aufzählt. Die in der ersten Zeile von (6.7) aufgeführte Konjunktion sichert die Konsistenz der Zustandsübergangsaxiomen mit (NonMult). Dies wird anhand des in Abschnitt 6.7 gegebenen Beispiels in Beobachtung 40 auf Seite 69 diskutiert. Die linke Seite der Implikation wird im Folgenden oft zu $\Delta(s)$ zusammengefasst, obwohl wir stets, sofern nicht explizit anders festgelegt, die hier gegebene Form voraussetzen.

Betrachten wir dies am Beispiel der Aktion $\mathit{PutOn}(x, y)$ aus der Welt der Blöcke. Deren Beschreibung besteht aus zwei Zustandsübergangsaxiomen, deren erstes beschreibt was passiert,

[8] Man beachte, dass sich dies vom Begriff der Vorbedingung einer Aktion unterscheidet: Eine Aktion kann anwendbar sein, obwohl die Vorbedingung eines spezifischen Zustandsübergangsaxioms der Aktion nicht erfüllt ist – wenn die Vorbedingung eines anderen Zustandsübergangsaxioms erfüllt ist.

wenn der Block x auf einem anderen Block steht, und deren zweites für den Fall, dass der Block x auf dem Tisch steht, vorgesehen ist:

$$\begin{aligned}(\forall\, x, y, z : \textit{Block}, s : \textit{State})\ [&\textit{Holds}(\textit{Clear}(x), s)\ \wedge\ \textit{Holds}(\textit{Clear}(y), s)\ \wedge \\ &\textit{Holds}(\textit{On}(x, z), s)\ \wedge\ \neg\,\textit{Holds}(\textit{On}(x, y), s)\ \wedge\ \neg\,\textit{Holds}(\textit{Clear}(z), s)\ \rightarrow \\ &\textit{state}(\textit{do}(\textit{PutOn}(x, y), s)) \circ \textit{Clear}(y) \circ \textit{On}(x, z) = \textit{state}(s) \circ \textit{Clear}(z) \circ \textit{On}(x, y)]\end{aligned}$$

$$\begin{aligned}(\forall\, x, y : \textit{Block}, s : \textit{State})\ [&\textit{Holds}(\textit{Clear}(x), s)\ \wedge\ \textit{Holds}(\textit{Clear}(y), s)\ \wedge \\ &\textit{Holds}(\textit{OnTable}(x), s)\ \wedge\ \neg\,\textit{Holds}(\textit{On}(x, y), s)\ \rightarrow \\ &\textit{state}(\textit{do}(\textit{PutOn}(x, y), s)) \circ \textit{Clear}(y) \circ \textit{OnTable}(x) = \textit{state}(s) \circ \textit{On}(x, y)]\ .\end{aligned}$$

Betrachten wir die Gleichung

$$\textit{state}(\textit{do}(\textit{PutOn}(x, y), s)) \circ \textit{Clear}(y) \circ \textit{On}(x, z) = \textit{state}(s) \circ \textit{Clear}(z) \circ \textit{On}(x, y)\ ,$$

die den Kern des ersten Zustandsübergangsaxioms bildet. Die intuitive Interpretation einer derartigen Gleichung erschließt sich, wenn man Theorem 30 in Betracht zieht: Wenn man zu der Multimenge von Fluenten, die vor Ausführung der Aktion wahr sind ($\textit{state}(s)$), die Fluenten $\textit{Clear}(z)$ und $\textit{On}(x, y)$ hinzufügt, dann ergibt sich das Gleiche wie wenn man zur Multimenge von Fluenten, die nach der Ausführung der Aktion $\textit{PutOn}(x, y)$ wahr sind ($\textit{state}(\textit{do}(\textit{PutOn}(x, y), s))$), die Fluenten $\textit{Clear}(y)$ und $\textit{On}(x, z)$ hinzufügt. Setzen wir nun voraus, das $\textit{Clear}(y)$ und $\textit{On}(x, z)$ vor der Aktion erfüllt sind – dies ist durch die Vorbedingung des Zustandsübergangsaxioms garantiert. Dann ist dies nach den Regeln der Multimengenarithmetik das gleiche, wie denn man sagt, dass sich der Zustand nach Ausführung der Aktion ergibt, indem man aus dem Zustand vor der Aktionsausführung $\textit{Clear}(y)$ und $\textit{On}(x, z)$ entfernt und $\textit{Clear}(z)$ und $\textit{On}(x, y)$ hinzufügt. Dies ist genau die intendierte Semantik der Aktion. Man beachte, dass bei dieser Operation alle nicht erwähnten Fluenten (die sich z.B. auf andere Blöcke beziehen) automatisch in ihrem Wahrheitswert unverändert bleiben. Dies ist die Fluentkalkül-Idee zur Lösung des sogenannten **Rahmenproblems** [63, 64].

Man beachte, dass die beschriebene einfache Struktur der Zustandsübergangsaxiome nicht möglich wäre, wenn die Eigenschaft Idempotenz $(\forall\, z : \textit{State})\ \ z = z \circ z$ für $\circ$ gefordert würde, um der Sorte *State* eine Mengensemantik anstatt einer Multimengensemantik zu geben. So hätte z.B. eine Gleichung $\textit{state}(\textit{do}(a, s)) \circ \textit{On}(\mathtt{A}, \mathtt{B}) = \textit{On}(\mathtt{A}, \mathtt{B}) \circ \textit{OnTable}(\mathtt{B})$ nicht nur die Lösung $\textit{state}(\textit{do}(a, s)) = \textit{OnTable}(\mathtt{B})$, sondern auch $\textit{state}(\textit{do}(a, s)) = \textit{On}(\mathtt{A}, \mathtt{B}) \circ \textit{OnTable}(\mathtt{B})$, so dass $\textit{On}(\mathtt{A}, \mathtt{B})$ nicht mehr, wie beabsichtigt, zum negativen Effekt wird.

Eine Demonstration für die Anwendung der Zustandsübergangsaxiome zur Lösung eines Planungsproblems ist im folgenden Abschnitt zu finden.

6.7. Planen in der Welt der Blöcke

Als Abschluss dieses Kapitels betrachten wir ein Beispiel für ein Planungsproblem aus der Welt der Blöcke im Fluentkalkül. Der Ausgangszustand und der Zielzustand seien wie in Abbildung 6.2 angegeben. Im folgenden geben wir nun eine komplette Fluentkalkül-

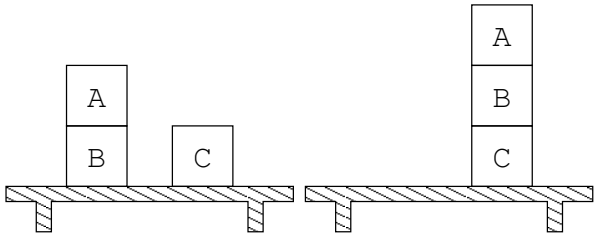


Abbildung 6.2.: Ein Planungsproblem in der Welt der Blöcke: Ausgangszustand (links) und Zielzustand (rechts).

Axiomatisierung dieses Planungsproblems an. Verwendet wird Signatur in Beispiel 21 auf Seite 43. Der Ausgangszustand ergibt sich zu:

$$\mathit{state}(\mathtt{S}_0) = \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C}) \circ \mathit{On}(\mathtt{A},\mathtt{B}) \circ \mathit{OnTable}(\mathtt{B}) \circ \mathit{OnTable}(\mathtt{C}) \ . \tag{6.8}$$

Die Aktionen werden durch die Zustandsübergangsaxiome

$$\begin{aligned}&(\forall\, x, y, z : \mathit{Block}, s : \mathit{State})\ [\mathit{Holds}(\mathit{Clear}(x), s) \ \wedge\ \mathit{Holds}(\mathit{Clear}(y), s) \wedge \\ &\mathit{Holds}(\mathit{On}(x, z), s) \ \wedge\ \neg\, \mathit{Holds}(\mathit{On}(x, y), s) \ \wedge\ \neg\, \mathit{Holds}(\mathit{Clear}(z), s) \ \rightarrow \\ &\quad \mathit{state}(\mathit{do}(\mathit{PutOn}(x, y), s)) \circ \mathit{Clear}(y) \circ \mathit{On}(x, z) = \mathit{state}(s) \circ \mathit{Clear}(z) \circ \mathit{On}(x, y)]\end{aligned} \tag{6.9}$$

$$\begin{aligned}&(\forall\, x, y : \mathit{Block}, s : \mathit{State})\ [\mathit{Holds}(\mathit{Clear}(x), s) \ \wedge\ \mathit{Holds}(\mathit{Clear}(y), s) \wedge \\ &\mathit{Holds}(\mathit{OnTable}(x), s) \ \wedge\ \neg\, \mathit{Holds}(\mathit{On}(x, y), s) \ \rightarrow \\ &\quad \mathit{state}(\mathit{do}(\mathit{PutOn}(x, y), s)) \circ \mathit{Clear}(y) \circ \mathit{OnTable}(x) = \mathit{state}(s) \circ \mathit{On}(x, y)]\end{aligned} \tag{6.10}$$

und

$$\begin{aligned}&(\forall\, x, y : \mathit{Block}, s : \mathit{State})\ [\mathit{Holds}(\mathit{Clear}(x), s) \ \wedge\ \neg\, \mathit{Holds}(\mathit{OnTable}(x), s) \\ &\ \wedge\ \mathit{Holds}(\mathit{On}(x, y), s) \ \wedge\ \neg\, \mathit{Holds}(\mathit{Clear}(y), s) \ \rightarrow \\ &\quad \mathit{state}(\mathit{do}(\mathit{ToTable}(x), s)) \circ \mathit{On}(x, y) = \mathit{state}(s) \circ \mathit{OnTable}(x) \circ \mathit{Clear}(y)\end{aligned} \tag{6.11}$$

beschrieben. Sei nun $\mathcal{F}_{Blocks}$ die Menge der grundlegenden Axiome (AC1), (Irred), (NullTeil), (Levi), (Ind) und (NonMult), sowie der Axiome (6.8), (6.9), (6.10) und (6.11). Die Behauptung, dass ein Plan existiert, stellt sich nun wie folgt dar:

$$\mathcal{F}_{Blocks} \models (\exists\, s : \mathit{State})\ [\mathit{Holds}(\mathit{On}(\mathtt{A},\mathtt{B}), s) \wedge \mathit{Holds}(\mathit{On}(\mathtt{B},\mathtt{C}), s) \wedge \mathit{Holds}(\mathit{OnTable}(\mathtt{C}), s)] \ .$$

(6.12)

Um nun das Planungsproblem zu lösen gilt es einen konstruktiven Beweis für diese Aussage zu finden, so dass in dem gefundenen Wert für s ein Plan kodiert ist, der das Problem löst.

Beobachtung 39. $s = do(PutOn(\mathtt{A}, \mathtt{B}), do(PutOn(\mathtt{B}, \mathtt{C}), do(ToTable(\mathtt{A}), \mathtt{S}_0)))$ *löst* (6.12) .

Beweis. Da für $s = \mathtt{S}_0$, $x = \mathtt{A}$ und $y = \mathtt{B}$ die Vorbedingung des Zustandsübergangsaxioms (6.11) erfüllt ist, ergibt sich aus (6.8) und (6.11) unter Anwendung der Substitutivität der Gleichheit

$$state(do(ToTable(BlockA), \mathtt{S}_0)) \circ On(\mathtt{A}, \mathtt{B}) =$$
$$Clear(\mathtt{A}) \circ Clear(\mathtt{C}) \circ On(\mathtt{A}, \mathtt{B}) \circ OnTable(\mathtt{B}) \circ OnTable(\mathtt{C}) \circ OnTable(\mathtt{A}) \circ Clear(\mathtt{B}) \ .$$

Anwendung der Kürzungsregel (Cancel) ergibt

$$state(do(ToTable(BlockA), \mathtt{S}_0)) =$$
$$Clear(\mathtt{A}) \circ Clear(\mathtt{C}) \circ OnTable(\mathtt{B}) \circ OnTable(\mathtt{C}) \circ OnTable(\mathtt{A}) \circ Clear(\mathtt{B}) \ .$$

Analog ergibt sich aus (6.10) mit $s = do(ToTable(\mathtt{A}), \mathtt{S}_0)$ und $x = \mathtt{B}$ und $y = \mathtt{C}$

$$state(do(PutOn(\mathtt{B}, \mathtt{C}), do(ToTable(\mathtt{A}), \mathtt{S}_0))) =$$
$$Clear(\mathtt{A}) \circ OnTable(\mathtt{C}) \circ OnTable(\mathtt{A}) \circ Clear(\mathtt{B}) \circ On(\mathtt{B}, \mathtt{C}) \ ,$$

sowie aus (6.10) mit $s = do(PutOn(\mathtt{B}, \mathtt{C}), do(ToTable(\mathtt{A}), \mathtt{S}_0)$ und $x = \mathtt{A}$ und $y = \mathtt{B}$

$$state(do(PutOn(\mathtt{A}, \mathtt{B}), do(PutOn(\mathtt{B}, \mathtt{C}), do(ToTable(\mathtt{A}), \mathtt{S}_0)))) =$$
$$Clear(\mathtt{A}) \circ OnTable(\mathtt{C}) \circ On(\mathtt{B}, \mathtt{C}) \circ On(\mathtt{A}, \mathtt{B}) \ .$$

Es zeigt sich also, dass das Problem (6.12) durch
$s = do(PutOn(\mathtt{A}, \mathtt{B}), do(PutOn(\mathtt{B}, \mathtt{C}), do(ToTable(\mathtt{A}), \mathtt{S}_0)))$ gelöst wird. ∎

Mit anderen Worten: Die Aktionssequenz $ToTable(\mathtt{A}), PutOn(\mathtt{B}, \mathtt{C}), PutOn(\mathtt{A}, \mathtt{B})$ löst unser Planungsproblem. Um Terme wie $do(PutOn(\mathtt{A}, \mathtt{B}), do(PutOn(\mathtt{B}, \mathtt{C}), do(ToTable(\mathtt{A}), \mathtt{S}_0)))$ übersichtlicher darzustellen, führen wir folgende abkürzende Schreibweise ein:

$$do([a_1, a_2, \ldots, a_{n-1}, a_n], \mathtt{S}_0) \underset{\text{Def}}{\equiv} do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, \mathtt{S}_0)) \ldots)) \ .$$

Das *Holds* -Makro wird oft auf formelartige Kombinationen von Fluenten ausgedehnt:

$$Holds(f, s) \underset{\text{Def}}{\equiv} (\exists\, z' : State)\ state(s) = f \circ z' \qquad \text{(Holds)}$$
$$Holds(\mathbf{1}, s) \underset{\text{Def}}{\equiv} \mathbf{1}$$
$$Holds(\neg\, \phi, s) \underset{\text{Def}}{\equiv} \neg\, Holds(\phi, s)$$
$$Holds(\phi\ \wedge\ \psi, s) \underset{\text{Def}}{\equiv} Holds(\phi)\ \wedge\ Holds(\psi, s)$$

(Dies ist entsprechend für $\mathbf{0}$ sowie die nicht explizit erwähnten binären logischen Operatoren anwendbar, da sich diese durch $\neg$ und $\wedge$ darstellen lassen.) Weiterhin wird *Holds* auch analog auf Zustände angewendet, indem $\mathit{state}(s)$ durch z eingesetzt wird:

$$\mathit{Holds}(f, z) \underset{\text{Def}}{\equiv} (\exists\, z' : \mathit{State})\ z = f \circ z'$$

$$\vdots$$

Es wird nun auch klar, warum die syntaktische Form (6.7) für die Zustandsübergangsaxiome gewählt wird. Angenommen man betrachtet Zustandsübergangsaxiome der allgemeineren Form

$$(\forall)\ \left[\Delta(s)\ \rightarrow\ \mathit{state}(\mathit{do}(a, s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+\right]\ . \tag{6.13}$$

Hier ist es nun möglich, dass Zustandsübergangsaxiome der in (6.7) gegebenen Form zu den grundlegenden Axiomen des Fluentkalküls inkonsistent sind:

Beobachtung 40. *Betrachten wir folgende Spezifikation der Aktion PutOn, die syntaktisch der Form* (6.13) *aber nicht der Form* (6.7) *genügt:*

$$\begin{aligned}(\forall\, x, y : \mathit{Block}, s : \mathit{State})\ [\mathbf{1}\ \rightarrow\ \\ \mathit{state}(\mathit{do}(\mathit{PutOn}(x, y), s)) \circ \mathit{Clear}(y) \circ \mathit{OnTable}(x) = \mathit{state}(s) \circ \mathit{On}(x, y)]\end{aligned}$$

Daraus lässt sich analog zum Beweis von Beobachtung 39 ableiten, dass

$$\begin{aligned}&\mathit{state}(\mathit{do}(\mathit{PutOn}(\mathtt{C}, \mathtt{A}), \mathit{do}(\mathit{PutOn}(\mathtt{C}, \mathtt{A}), \mathtt{S}_0))) \circ \\ &\mathit{Clear}(\mathtt{A}) \circ \mathit{OnTable}(\mathtt{C}) \circ \mathit{Clear}(\mathtt{A}) \circ \mathit{OnTable}(\mathtt{C}) = \\ &\quad \mathit{Clear}(\mathtt{A}) \circ \mathit{Clear}(\mathtt{C}) \circ \mathit{On}(\mathtt{A}, \mathtt{B}) \circ \mathit{OnTable}(\mathtt{B}) \circ \mathit{OnTable}(\mathtt{C}) \circ \mathit{On}(\mathtt{C}, \mathtt{A}) \circ \mathit{On}(\mathtt{C}, \mathtt{A})\end{aligned} \tag{6.14}$$

bzw., nach Anwenden von (Cancel)*,*

$$\begin{aligned}&\mathit{state}(\mathit{do}(\mathit{PutOn}(\mathtt{C}, \mathtt{A}), \mathit{do}(\mathit{PutOn}(\mathtt{C}, \mathtt{A}), \mathtt{S}_0))) \circ \mathit{Clear}(\mathtt{A}) \circ \mathit{OnTable}(\mathtt{C}) = \\ &\qquad \mathit{Clear}(\mathtt{C}) \circ \mathit{On}(\mathtt{A}, \mathtt{B}) \circ \mathit{OnTable}(\mathtt{B}) \circ \mathit{On}(\mathtt{C}, \mathtt{A}) \circ \mathit{On}(\mathtt{C}, \mathtt{A})\ .\end{aligned} \tag{6.15}$$

Dies ist aus zwei Gründen widersprüchlich zu $\mathcal{F}_{mset}$*,* (NonMult) *und der Namenseindeutigkeit:*

1. *Es folgt durch mehrmaliges Anwenden von* (Distrib)*, dass*

   $$(\exists\, z : \mathit{State})\ \mathit{state}(\mathit{do}(\mathit{PutOn}(\mathtt{C}, \mathtt{A}), \mathit{do}(\mathit{PutOn}(\mathtt{C}, \mathtt{A}), \mathtt{S}_0))) = z \circ \mathit{On}(\mathtt{C}, \mathtt{A}) \circ \mathit{On}(\mathtt{C}, \mathtt{A})\ ,$$

   was (NonMult) *widerspricht.*

2. *Es folgt durch mehrmaliges Anwenden von* (Distrib)*, dass*

$$(\exists\, z : \mathit{State})\ \ z \circ \mathit{OnTable}(\mathtt{C}) = \mathit{On}(\mathtt{C}, \mathtt{A})\ \ ,$$

was (Irred) *widerspricht.*

Wenn man dagegen von der Form (6.7) für die Zustandsübergangsaxiome ausgeht, können derartige Inkonsistenzen nicht auftreten.

Wenn man die Zustandsübergangsaxiome (6.9), (6.10), (6.11) im Vergleich mit der Problemspezifikation in Beispiel 21 auf Seite 43 betrachtet, fallen Redundanzen auf: So erscheint z.B. in (6.9) $\neg\, \mathit{Holds}(\mathit{On}(x, y), s)$ überflüssig, da $\mathit{Holds}(\mathit{Clear}(y), s)$ ebenfalls im Antezedent der Implikation steht, und laut Problembeschreibung ein Block genau dann *Clear* sein soll, wenn kein Block auf ihm steht. Tatsächlich lässt sich eine solche Redundanzen durch die Einführung eines Domänenaxioms, die dieses Verhältnis von *Clear* und *On* beschreiben, eliminieren [44]; der Nachweis der Widerspruchsfreiheit einer solchen Axiomatisierung ist aber domänenspezifisch und erfordert u.U. Induktion, so dass wir für die Zwecke dieser Arbeit die einfache syntaktische Bedingung (6.7) setzen, die Widerspruchsfreiheit garantiert.

Zum Abschluss des Kapitels stellt Tabelle 6.1 noch einmal alle grundlegenden Axiome und Symboldefinitionen des Fluentkalküls zusammen.

Fluentkalkül-Signatur ohne domänenspezifische Bestandteile:

$$
\begin{array}{ll}
\textbf{SORT} & \mathit{Action}, \mathit{Sit}, \mathit{Fluent} \preceq \mathit{State}, \\
\textbf{FUN} & \mathtt{S}_0 \quad : \mathit{Sit}, \\
& \mathit{do} \quad : \mathit{Action} \times \mathit{Sit} \to \mathit{Sit}, \\
& \mathit{state} \quad : \mathit{Sit} \to \mathit{State}, \\
& \emptyset \quad : \mathit{State}, \\
& \circ \quad : \mathit{State} \times \mathit{State} \to \mathit{State}.
\end{array}
$$

$$
\begin{array}{rrcl}
(\forall\, x, y, z : \mathit{State}) & (x \circ y) \circ z & = & x \circ (y \circ z) \\
(\forall\, x, y : \mathit{State}) & x \circ y & = & y \circ x \\
(\forall\, x : \mathit{State}) & x \circ \emptyset & = & x
\end{array}
\tag{AC1}
$$

$$
(\forall\, z, z', z'' : \mathit{State})\ \Big[\big((\exists\, f : \mathit{Fluent})\ z = f\ \to\ z \neq \emptyset\ \wedge\ (z = z' \circ z''\ \to\ z' = \emptyset\ \vee\ z'' = \emptyset)\big]
\tag{Irred}
$$

$$
(\forall\, z, z' : \mathit{State})\ \big[\emptyset = z \circ z'\ \to\ z = \emptyset\big]
\tag{NullTeil}
$$

$$
(\forall\, z_1, z_2, z_3, z_4)\ \Big[z_1 \circ z_2 = z_3 \circ z_4\ \to\ (\exists\, z_a, z_b, z_c, z_d)\ \big(\ z_1 = z_a \circ z_b\ \wedge\ z_2 = z_c \circ z_d\ \wedge\ z_3 = z_a \circ z_c\ \wedge\ z_4 = z_b \circ z_d\ \big)\Big]
\tag{Levi}
$$

$$
(\forall\, P : \langle \mathit{State} \rangle)\ \Big[P(\emptyset)\ \wedge\ (\forall\, f : \mathit{Fluent}, z : \mathit{State})\ (P(z)\ \to\ P(z \circ f))\ \to\ (\forall\, z : \mathit{State})\ P(z)\Big]
\tag{Ind}
$$

$$
(\forall\, s : \mathit{State})\ \neg\, (\exists\, f : \mathit{Fluent}, z : \mathit{State})\ \mathit{state}(s) = f \circ f \circ z
\tag{NonMult}
$$

$$
\begin{array}{rcl}
\mathit{Holds}(\phi, s) & \underset{\text{Def}}{\equiv} & \mathit{Holds}(\phi, \mathit{state}(s)) \\
\mathit{Holds}(f, z) & \underset{\text{Def}}{\equiv} & (\exists\, z' : \mathit{State})\ z = f \circ z' \\
\mathit{Holds}(\mathbf{1}, z) & \underset{\text{Def}}{\equiv} & \mathbf{1} \\
\mathit{Holds}(\neg\, \phi, z) & \underset{\text{Def}}{\equiv} & \neg\, \mathit{Holds}(\phi, z) \\
\mathit{Holds}(\phi\ \wedge\ \psi, z) & \underset{\text{Def}}{\equiv} & \mathit{Holds}(\phi, z)\ \wedge\ \mathit{Holds}(\psi, z)
\end{array}
\tag{Holds}
$$

$$
\mathit{do}([a_1, a_2, \ldots, a_{n-1}, a_n], \mathtt{S}_0) \underset{\text{Def}}{\equiv} \mathit{do}(a_n, \mathit{do}(a_{n-1}, \ldots, \mathit{do}(a_2, \mathit{do}(a_1, \mathtt{S}_0)) \ldots))
$$

Tabelle 6.1.: Die grundlegenden Axiome und Symboldefinitionen des Fluentkalküls

7. Eine Semantik von PDDL im Fluentkalkül

Zur Spezifikation von Planungsproblemen wird in diesem Buch ein Fragment der Sprache PDDL („Planning Domain Definition Language“) verwendet, die eingeführt wurde, um den Vergleich von modernen Planungsalgorithmen auf der AIPS Konferenz zu ermöglichen. In diesem Kapitel wird PDDL beschrieben, sowie eine Semantik von PDDL mit Hilfe des Fluentkalküls definiert.

Um das im weiteren Teil des Buches beschriebene Planungsprogramm BDDPLAN leichter mit anderen Planungsprogrammen vergleichen zu können, wurde die Planungsdomänenbeschreibungssprache PDDL als Eingabesprache genutzt. Damit kann auf die Sammlung [66] von Planungsproblemen für Testzwecke zurückgegriffen werden.

Für PDDL stand bisher nur eine informale Semantikbeschreibung [37, 67] zur Verfügung, sowie eine Referenzimplementation in LISP, die dazu dienen kann, Lösungen von Planungsproblemen zu überprüfen. Wir gehen in diesem Kapitel darüber hinaus und entwickeln eine formale Semantik für die PDDL.

7.1. Ein Beispiel

Um dem Leser die Semantik von PDDL nahezubringen, wird die Modellierung eines etwas abgewandelten Beispiels aus [70] mit Hilfe von PDDL beschrieben. Die Darstellung ist weitgehend aus dem PDDL Manual [37] übernommen.

Beispiel 41 („Aktenmappenwelt“)**.** *Ein Mann besitzt eine Aktenmappe, ein Buch und einen Scheck. Die Aktenmappe und das Buch befinden sich in seinem Haus und der Scheck befindet sich in der Mappe. Sein Ziel ist es, das Buch und die Mappe im Büro zu haben, den Scheck aber zu Hause zu lassen. Dazu kann er folgende Aktionen ausführen:*

1. *er kann etwas in seine Mappe legen*
2. *er kann etwas aus seiner Mappe herausnehmen*
3. *er kann seine Mappe nehmen und diese mit allen enthaltenen Gegenständen ins Büro oder zurück transportieren.*

Die erste Aufgabe bei der Modellierung dieses Beispiels ist es nun, die relevanten Fluenten zu identifizieren. Dies sind hier:

Fluent	Fluent ist wahr wenn:
am-Ort(`Mappe,Haus`*)*	die Mappe sich im Haus befindet,
am-Ort(`Mappe,Buero`*)*	die Mappe sich im Büro befindet,
am-Ort(`Scheck,Haus`*)*	der Scheck sich im Haus befindet,
am-Ort(`Scheck,Buero`*)*	der Scheck sich im Büro befindet,
am-Ort(`Buch,Haus`*)*	das Buch sich im Haus befindet,
am-Ort(`Buch,Buero`*)*	das Buch sich im Büro befindet,
in-Mappe(`Scheck`*)*	der Scheck sich in der Mappe befindet,
in-Mappe(`Buch`*)*	das Buch sich in der Mappe befindet.

Im Interesse einer kompakten Darstellung sind die Fluenten parametrisiert. Sie bestehen aus einem **Fluent-Kopf** (z.B. *am-Ort* und *in-Mappe*), der den Typ des Fluents charakterisiert, und einigen **Parametern**, die spezifizieren, auf welche Objekte (z.B. `Mappe, Scheck, Buch, Haus, Buero`) sich das Fluent bezieht. In PDDL sind Objekte in einer Typhierarchie angeordnet, bei der die Objektmengen der untergeordneten Typen Teilmengen der Objektmengen der übergeordneten Typen sind. Laut PDDL Manual steht an der Spitze der Typhierarchie stets implizit der Typ *object*, der alle anderen Typen umfasst. Für Beispiel 41 ergibt sich folgende Typhierarchie:

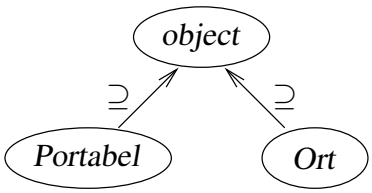


Abbildung 7.1.: Typhierarchie in Beispiel 41.

Die Objekte `Mappe, Scheck` und `Buch` sind dabei vom Typ *Portabel*, und `Haus` und `Buero` sind vom Typ *Ort*. Der Sinn dieser Typisierung der Objekte besteht darin zu sichern, dass für die Parameter eines Fluents nur sinnvolle Objekte verwendet werden können. So sollte in unserem Aktenmappen-Beispiel *am-Ort* als ersten Parameter ein Objekt vom Typ *Portabel* und als zweiten ein Objekt vom Typ *Ort* haben, und *in-Mappe* einen Parameter vom Typ *Portabel*. Damit wird vermieden, dass ein Fluent wie *in-Mappe(Haus)* als „Ballast“ in der Modellierung auftaucht.

Eine PDDL–Beschreibung eines Planungsproblems besteht aus zwei Teilen: der PDDL–Domäne, die eine Klasse von Planungsproblen deklariert, und dem PDDL–Problem, das

ein Planungsproblem konkret spezifiziert. Für das Aktenmappen-Problem kann die PDDL–Domäne beispielsweise so aussehen:

```
(define (domain Aktenmappenwelt)
  (:types Portabel Ort)
  (:constants Mappe - Portabel)
  (:predicates (am-Ort ?x - Portabel ?l - Ort)
               (in-Mappe ?x - Portabel))
```

Die Syntax von PDDL orientiert sich stark an LISP, einer sehr verbreitete Programmiersprache in der künstlichen Intelligenz. Terme wie $f(a, b, c)$ werden als eine geklammerte Liste von Funktor und Argumenten dargestellt: `(f a b c)`. Parameter und Variablen werden in PDDL durch ein vorangestelltes `?` und Kommentare durch ein vorangestelltes Semikolon gekennzeichnet. [1]

Hier werden zunächst die Objekttypen *Portabel* und *Ort*, sowie ein Objekt `Mappe` des Typs *Portabel*, das in allen Planungsproblemen dieser Klasse vorkommt, deklariert, und zu zwei Prädikaten *am-Ort* und *in-Mappe*, die die o.g. Fluenten darstellen, die Argumenttypen angegeben: bei *am-Ort* ein Argument vom Typ *Portabel* und ein Argument vom Typ *Ort*, und bei *in-Mappe* ein Argument vom Typ *Portabel*.[2]

Als nächster Schritt ist es nun nötig die Aktionen zu beschreiben. Dazu gehören zwei Bestandteile: Erstens ist es nötig anzugeben, wann eine Aktion ausführbar ist (die **Vorbedingung**, engl. precondition, der Aktion), und zweitens ist es nötig, den **Effekt** einer Aktion, d.h. die Veränderungen des Zustands, die die Aktion hervorruft, zu beschreiben. Dazu geht PDDL wie viele andere Planungssysteme von der Grundidee aus, dass Werte von Fluenten sich nur dann verändern, wenn dies explizit in den Effekten der Aktion angegeben wird. Dies wird als STRIPS-Annahme (engl. STRIPS-assumption, nach dem STRIPS-Planungssystem [32]) bezeichnet. Die Aktion *Herausnehmen* eines Objektes aus der Mappe kann z.B. so beschrieben werden:

```
  (:action Herausnehmen
     :parameters (?x - Portabel)
     :precondition (in-Mappe ?x)
     :effect (not (in-Mappe ?x)))
```

[1] PDDL–Beschreibungen sind syntaktisch korrekte LISP-Terme. Es wurde von den PDDL–Autoren ein in LISP programmierter Syntaxchecker und ein LISP-Programm zum Überprüfen von Lösungen für Planungsprobleme bereitgestellt. Die Kennzeichnung von Variablen durch `?` wird in LISP allerdings normalerweise nicht getroffen.

[2] In der angegebenen Form lässt die Typisierung allerdings noch das Fluent *in-Mappe*(`Mappe`) zu, das in Bezug auf das originale Problem nicht sinnvoll ist. Um die Domänenbeschreibung nicht unnötig kompliziert zu gestalten, wird hier ein Kompromiss eingegangen, und darauf verzichtet durch eine Einführung weiterer Typen dies auch noch auszuschließen.

Ebenso wie Fluenten sind die Aktionen parametrisiert, um die Beschreibungen von Gruppen gleichartiger Aktionen zusammenfassen zu können. Die Aktion *Herausnehmen* ist mit einem Parameter `?x` vom Typ *Portabel* versehen, der das herausgenommene Objekt angibt. Als Vorbedingung der Aktion wird *in-Mappe(?x)* angegeben, d.h. dass sich Objekt `?x` in der Mappe befindet, und als Effekt, dass *in-Mappe(?x)* falsch ist (`not` drückt die Negation aus). Da über die anderen Fluenten des Problems nichts gesagt wird, bleiben deren Werte durch die Ausführung der Aktion unverändert.

Ergänzt wird die PDDL–Domäne durch die Beschreibungen der zwei anderen Aktionen:

```
  (:action Hineinlegen
     :parameters (?x - Portabel ?l - Ort)
     :precondition (and (am-Ort ?x ?l) (am-Ort Mappe ?l)
                        (not (in-Mappe ?x)))
     :effect (in-Mappe ?x))



  (:action Transport
     :parameters (?m ?l - Ort)
     :precondition (and (am-Ort Mappe ?m) (not (= ?m ?l)))
     :effect (and (am-Ort Mappe ?l) (not (am-Ort Mappe ?m))
                  (forall (?z - Portabel)
                       (when (in-Mappe ?z)
                             (and (am-Ort ?z ?l)
                                  (not (am-Ort ?z ?m)))))))

) ; Ende der domain Aktenmappenwelt
```

Hierbei ist zu erkennen, dass die Vorbedingungen und Effekte von Aktionen komplexe logische Formeln beinhalten können. Im Detail wird dies später diskutiert. An dieser Stelle seien nur zwei Bemerkungen dazu gemacht. Es ist in PDDL möglich über Objekte eines Typs zu quantifizieren, d.h. zu sagen, dass ein Effekt für alle (`forall`) Objekte eines Typs eintreten muss, oder als Vorbedingung anzugeben, dass ein Objekt eines Typs existieren muss (`exists`), für das eine Bedingung gilt. Weiterhin ist es möglich mit Hilfe von `(when` ϕ ψ `)` anzugeben, dass ein Effekt ψ genau dann eintritt, wenn die Bedingung ϕ in dem Zustand, in dem die Aktion ausgeführt wird, erfüllt ist. Diese zwei Elemente werden in der Aktion *Transport* dazu benutzt auszudrücken, dass, wenn die Mappe von Ort `?m` zu `?l` bewegt wird, sich alle Objekte `?z` mitbewegen, die sich in der Mappe befinden.

Um ein vollständiges Planungsproblem zu erhalten, muss zur Beschreibung der Planungsdomäne noch eine Aufzählung der weiteren für das Problem relevanten Objekte, sowie der

Ausgangszustand und das zu erreichende Ziel angegeben werden. Dies erfolgt innerhalb des PDDL–Problems:

```
(define (problem Transportiere)
    (:domain Aktenmappenwelt)
    (:objects Buch Scheck - Portabel  Haus Buero - Ort)
    (:init (am-Ort Mappe Haus) (am-Ort Buch Haus)
           (am-Ort Scheck Haus) (in-Mappe Scheck))
    (:goal (and (am-Ort Mappe Buero) (am-Ort Buch Buero)
                (am-Ort Scheck Haus)))
)
```

Hier werden die Objekte `Buch` und `Scheck` als vom Typ *Portabel*, und `Haus` und `Buero` als vom Typ *Ort* deklariert. In der mit `:init` gekennzeichneten Liste werden alle Fluenten aufgezählt, die im Ausgangszustand wahr sind. Alle nicht explizit erwähnten Fluenten werden als falsch vorausgesetzt. Das mit `:goal` markierte Ziel kann dagegen wieder eine komplexe logische Formel sein, die zur Lösung des Planungsproblems erfüllt werden muss.

Damit ist Beispiel 41 komplett in PDDL spezifiziert. In den folgenden Abschnitten wird nun die Syntax von PDDL formal definiert und ein Übersetzungsschema angegeben, so dass ein in PDDL spezifiziertes Planungsproblem mit Hilfe des Fluentkalküls ausgedrückt wird. Dies bildet, zusammen mit dem in Kapitel 8 beschriebenen Planungsalgorithmus, die semantische Basis für das Planungsprogramm BDDPLAN.

7.2. Syntax von PDDL

Im folgenden Abschnitt wird die Syntax von PDDL nach [37][3]in einer erweiterten Backus-Naur Form (EBNF) mit folgenden Konventionen dargestellt:

- Syntaktische Elemente werden durch die Klammerung mit $\langle\ \rangle$ gekennzeichnet (z.B. $\langle$ *Domäne* $\rangle$, $\langle$ *Ziel* $\rangle$).
- Jede Regel wird in der Form $\langle$ *syntaktisches Element* $\rangle$ ::= `Expansion` geschrieben.
- Runde Klammern haben keine spezielle Bedeutung in der EBNF.
- Durch * werden Elemente markiert, die mehrfach auftreten können, aber auch weggelassen werden können, und durch $^+$ werden Elemente markiert, die mindestens einmal auftreten müssen, aber auch mehrfach auftreten können.

[3] Man beachte, dass die Namen der syntaktischen Konstrukte für die Wiedergabe in dieser Arbeit übersetzt wurden.

- Einige syntaktische Elemente sind parametrisiert. So enthält z.B. die EBNF-Definition

  ⟨ *Liste(x)* ⟩ ::= *x* *

  den Parameter x, der bei Verwendung von ⟨ *Liste* ⟩ durch andere syntaktische Elemente instanziiert werden kann: ⟨ *Liste(Name)* ⟩ entspricht also ⟨ *Name* ⟩ *.

Im Folgenden wird nur der Teil der Sprache PDDL dargestellt, der sich direkt mit Hilfe von Aussagenlogik darstellen lässt. Dies entspricht dem `:adl`-Fragment im PDDL–Manual [37]. Weiterhin werden der Übersicht halber einige syntaktische Konstrukte weggelassen, die lediglich „syntaktischen Zucker“ darstellen, d.h. nur einer angenehmeren Notation dienen, und keinen Beitrag zur Ausdrucksstärke bieten.[4]

Die Einführung der Syntax erfolgt „Bottom-Up“, d.h. bevor die Struktur eines vollständigen PDDL–Textes beschrieben wird, werden zunächst einige Konstruktionen eingeführt, die an mehreren Stellen verwendet werden. Als Vorarbeit für die nachfolgende Übersetzung in den Fluentkalkül definieren wir im Laufe dieser Beschreibung bereits einige Transformationen ($\mathfrak{L}(\_)$, $\mathfrak{E}(\_)$, $\mathfrak{P}(\_)$), die den Listen und Formeln der PDDL–Darstellungen eine mathematische oder logische Darstellung im Rahmen einer geeigneten Fluentkalkül-Signatur zuordnen.

7.2.1. Grundlagen

Wie schon erwähnt, basiert der PDDL–Syntax auf der Syntax der Programmiersprache LISP. Namen von Domänen, Fluenten, Objekten usw., die dem syntaktischen Element ⟨ *Name* ⟩ entsprechen, sind Zeichenketten, die mit einem Buchstaben beginnen, und Buchstaben, Ziffern, „-“ und „_“ enthalten können. Dabei wird Groß- und Kleinschreibung ignoriert. Davon abgeleitet sind die syntaktischen Elemente

⟨ *Fluentname* ⟩ ::= ⟨ *Name* ⟩
⟨ *Variable* ⟩ ::= ? ⟨ *Name* ⟩

Ein in PDDL häufig verwendetes Konstrukt ist die getypte Liste, die eine Sequenz von Objekten oder Variablen deklariert und ihnen Typen zuordnet. Dies sieht in EBNF wie folgt aus:

⟨ *getypte Liste (x)* ⟩ ::= *x* *
⟨ *getypte Liste (x)* ⟩ ::= *x* + - ⟨ *Typ* ⟩ ⟨ *getypte Liste(x)* ⟩
⟨ *Typ* ⟩ ::= ⟨ *Name* ⟩
⟨ *Typ* ⟩ ::= (either ⟨ *Typ* ⟩ +)

[4] Die weggelassenen Konstrukte sind: ⟨ *extension-def* ⟩, ⟨ *initsit def* ⟩, Domänen-Addenda und syntaktische Konstrukte, die nur von diesen verwendet werden.

In einer solchen Liste wird Typangaben ein „-“ Zeichen vorangestellt; jedem Element der Liste wird der erste Typ zugeordnet, der ihm folgt, oder der Typ *object*, wenn dem Element kein anderer Typ folgt. Um dies formal zu fassen wird eine rekursiv definierte Transformation $\mathfrak{L}(\_)$ definiert, die einer solchen getypten Liste eine Sequenz der Elemente mit ihren Typen zuordnet:

$$\begin{aligned}\mathfrak{L}(\mathtt{x_1} \ldots \mathtt{x_n} - \mathtt{t}|\mathtt{R}) &= (x_1 : t \quad x_2 : t \ \ldots \ (x_n : t)).\mathfrak{L}(\mathtt{R}) \\ \mathfrak{L}(\mathtt{x_1} \ldots \mathtt{x_n}) &= (x_1 : \textit{object} \quad x_2 : \textit{object} \ \ldots \ x_n : \textit{object}) \\ \mathfrak{L}() &= () \ .\end{aligned}$$

Ein Typ der Form `(either` $\langle$ *Typ* $\rangle^+$ `)` stellt die Vereinigung der genannten Typen dar, und wird als zusätzlicher übergeordneter Typ für alle aufgezählten Typen in die Hierarchie eingefügt.[5]

Hierbei stellt $x_1 \ldots x_n - t|R$ eine Sequenz $x_1 \ldots x_n - t$ dar, auf die eine Sequenz R folgt. $()$ stellt eine leere Sequenz dar. Das Funktionszeichen $.$ ist der Konkatenationsoperator von Sequenzen. Somit ergibt sich z.B.

$$\begin{aligned}&\mathfrak{L}(\mathtt{Buch\ Scheck\ -\ Portabel\ Haus\ Buero} - \mathtt{Ort}) \\ &\quad = (\mathtt{Buch} : \textit{Portabel}\ \mathtt{Scheck} : \textit{Portabel}).\mathfrak{L}(\mathtt{Haus\ Buero} - \mathtt{Ort}) \\ &\qquad = (\mathtt{Buch} : \textit{Portabel} \quad \mathtt{Scheck} : \textit{Portabel} \quad \mathtt{Haus} : \textit{Ort} \quad \mathtt{Buero} : \textit{Ort}) \ .\end{aligned}$$

7.2.2. Ziel- und Effektformeln

In PDDL gibt es zwei Arten von logischen Formeln. Die eine dient zum Darstellen von Vorbedingungen (im folgenden eine **Zielformel** genannt), und die andere zur Darstellung von Effekten (**Effektformel** genannt). Als Vorbedingungen sind im Wesentlichen beliebige funktorlose prädikatenlogische Formeln zugelassen, als Effekte nur konjunktive Formeln (mit einer Erweiterung durch den Operator `when`; siehe unten.) Als erster Schritt zu einer logischen Darstellung der Semantik nehmen wir nun eine Übersetzung von Ziel- und Effektformeln in logische Formeln vor. Dazu werden Transformationen $\mathfrak{P}(\_)$ und $\mathfrak{E}(\_)$ definiert, die diese Aufgabe übernehmen.

Zunächst geben wir nun die Syntax von Zielformeln in EBNF und die Definition der Transformation $\mathfrak{P}(\_)$:

```
⟨ Atom ⟩              ::= ( ⟨ Fluentname ⟩  ⟨ Term ⟩ * )
⟨ Term ⟩              ::= ⟨ Name ⟩
⟨ Term ⟩              ::= ⟨ Variable ⟩
⟨ Zielformel ⟩        ::= ⟨ Atom ⟩
```

[5] Siehe Seite 84.

```
⟨ Zielformel ⟩          ::= (and ⟨ Zielformel ⟩ * )
⟨ Zielformel ⟩          ::= (or ⟨ Zielformel ⟩ * )
⟨ Zielformel ⟩          ::= (not ⟨ Zielformel ⟩ )
⟨ Zielformel ⟩          ::= (imply ⟨ Zielformel ⟩ ⟨ Zielformel ⟩ )
⟨ Zielformel ⟩          ::= (exists ( ⟨ getypte Liste(Variable) ⟩ * )
                                ⟨ Zielformel ⟩ )
⟨ Zielformel ⟩          ::= (forall ( ⟨ getypte Liste(Variable) ⟩ * )
                                ⟨ Zielformel ⟩ )
```

$$\mathfrak{P}\big(\langle \mathit{Fluentname}\rangle\langle \mathit{Term}\rangle_1 \ldots \langle \mathit{Term}\rangle_n\big) = \langle \mathit{Fluentname}\rangle\,(\langle \mathit{Term}\rangle_1, \ldots, \langle \mathit{Term}\rangle_n)$$

$$\mathfrak{P}\big(= \langle \mathit{Term}\rangle_1\langle \mathit{Term}\rangle_2\big) = (\langle \mathit{Term}\rangle_1 = \langle \mathit{Term}\rangle_2)$$

$$\mathfrak{P}\big(\texttt{not}\ \langle \mathit{Zielformel}\rangle\big) = \neg\,\mathfrak{P}\big(\langle \mathit{Zielformel}\rangle\big)$$

$$\mathfrak{P}\big(\texttt{and}\ \langle \mathit{Zielformel}\rangle_1 \ldots \langle \mathit{Zielformel}\rangle_n\big) = \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle_1\big) \wedge \ldots \wedge \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle_n\big)$$

$$\mathfrak{P}\big(\texttt{or}\ \langle \mathit{Zielformel}\rangle_1 \ldots \langle \mathit{Zielformel}\rangle_n\big) = \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle_1\big) \vee \ldots \vee \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle_n\big)$$

$$\mathfrak{P}\big(\texttt{imply}\ \langle \mathit{Zielformel}\rangle_1\langle \mathit{Zielformel}\rangle_2\big) = \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle_1\big) \rightarrow \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle_n\big)$$

$$\mathfrak{P}\big(\texttt{exists}\ (\langle \mathit{getypte\ Liste(Variable)}\rangle^*)\langle \mathit{Zielformel}\rangle\big) = (\exists\ \mathfrak{L}\big(\langle \mathit{getypte\ Liste(Variable)}\rangle^*\big))\ \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle\big)$$

$$\mathfrak{P}\big(\texttt{forall}\ (\langle \mathit{getypte\ Liste(Variable)}\rangle^*)\langle \mathit{Zielformel}\rangle\big) = (\forall\ \mathfrak{L}\big(\langle \mathit{getypte\ Liste(Variable)}\rangle^*\big))\ \mathfrak{P}\big(\langle \mathit{Zielformel}\rangle\big)$$

Damit ergibt sich z.B.:

$$\mathfrak{P}\big(\texttt{forall (?z - Portabel) (imply (in-Mappe ?z) (= ?x Buch))}\big) = (\forall\, ?z : \textit{Portabel})\ (\textit{in-Mappe}(?z) \rightarrow (?x = \texttt{Buch})).$$

Die Übersetzung von Effektformeln gestaltet sich etwas komplizierter. In PDDL sind nur konjunktive Effekte zugelassen, d.h. alle Aktionen funktionieren deterministisch. Diese Effekte können aber bedingt sein. Effektformeln folgen folgender Syntax:

```
⟨ Literal ⟩             ::= ⟨ Atom ⟩
```

```
⟨ Literal ⟩            ::= (not ⟨ Atom ⟩)

⟨ Effektformel ⟩       ::= ⟨ Literal ⟩
⟨ Effektformel ⟩       ::= (and ⟨ Effektformel ⟩ *)
⟨ Effektformel ⟩       ::= (forall (⟨ Variable ⟩ *) ⟨ Effektformel ⟩)
⟨ Effektformel ⟩       ::= (when ⟨ Zielformel ⟩ ⟨ Effektformel ⟩)
```

Dabei entsprechen `not`, `and` und `forall` den entsprechenden logischen Funktoren wie bei Zielformeln. `when` dient dagegen zur Beschreibung von bedingten Effekten: die in einem Konstrukt (`when` ⟨ *Zielformel* ⟩ ⟨ *Effektformel* ⟩) in der ⟨ *Effektformel* ⟩ angegebenen Effekte treten nur auf, wenn die ⟨ *Zielformel* ⟩ erfüllt ist. Wir führen nun eine Transformation $\mathfrak{E}(\_)$ ein, die einer ⟨ *Effektformel* ⟩ die Menge aller Effekte (im Sinne eines Fluents oder eines negierten Fluents) zusammen mit den Bedingungen, unter denen sie auftreten, zuordnet. Dabei wird jeweils ein Effekt φ_e und die Bedingung φ_p, unter der er auftritt, zu einem Paar $\langle \varphi_p, \varphi_e \rangle$ zusammengefasst. Es gilt z.B.

$$\mathfrak{E}\left(\begin{array}{l}\texttt{when (in-Mappe ?z) (and (am-Ort ?z ?l)}\\ \qquad\qquad\qquad\qquad\texttt{(not (am-Ort ?z ?m)))}\end{array}\right)$$
$$= \{\langle \textit{in-Mappe}(?z), \textit{am-Ort}(?z, ?l)\rangle,\ \langle \textit{in-Mappe}(?z), \neg\, \textit{am-Ort}(?z, ?m)\rangle\}\ .$$

$\mathfrak{E}(\_)$ wird nun wie folgt definiert:

$$\mathfrak{E}\big(\langle \textit{Fluentname}\rangle\langle \textit{Term}\rangle_1 \ldots \langle \textit{Term}\rangle_n\big) = \langle \mathbf{1}, \langle \textit{Fluentname}\rangle\,(\langle \textit{Term}\rangle_1, \ldots, \langle \textit{Term}\rangle_n)\rangle$$
$$\mathfrak{E}\big(\texttt{not}\ \langle \textit{Atom}\rangle\big) = \langle \mathbf{1}, \neg\,\varphi\rangle \qquad \text{für} \quad \mathfrak{E}\big(\langle \textit{Atom}\rangle\big) = \langle \mathbf{1}, \varphi\rangle$$
$$\mathfrak{E}\big(\texttt{and}\ \langle \textit{Effektformel}\rangle_1 \ldots \langle \textit{Effektformel}\rangle_n\big) = \bigcup_{i=1\ldots n} \mathfrak{E}\big(\langle \textit{Effektformel}\rangle_i\big)$$

Beim Vorkommen von `when` wird die Bedingung in der ⟨ *Zielformel* ⟩ zu den in der ⟨ *Effektformel* ⟩ schon definierten Bedingungen hinzugefügt. Damit ist auch eine Schachtelung von `when` möglich.

$$\mathfrak{E}\big(\texttt{when}\ \langle \textit{Zielformel}\rangle\langle \textit{Effektformel}\rangle\big)$$
$$= \big\{\langle \mathfrak{P}\big(\langle \textit{Zielformel}\rangle\big)\ \wedge\ \varphi_p,\ \varphi_e\rangle \mid \langle \varphi_p, \varphi_e\rangle \in \mathfrak{E}\big(\langle \textit{Effektformel}\rangle\big)\big\}$$

Für `forall` werden die in ⟨ *getypte Liste(Variable)* ⟩ * quantifizierten PDDL–Variablen v_i durch alle Kombinationen von Objekten der entsprechenden Sorten ersetzt, und die so instanziierten Effekte gesammelt. Die Objekte einer Sorte s_i in einer PDDL-Domäne werden durch die Terme t_i aus der Menge T_{s_i} von Termen der jeweiligen Sorte s_i dargestellt: Jeder Term bezeichnet genau ein Objekt. Diese Terme ergeben sich gemäß der Signatur Σ für das Planungsproblem, die im Abschnitt 7.3 auf Seite 84 definiert wird. (Hier ist ein

Vorgriff notwendig, da die Abbildung $\mathfrak{E}()$ bereits im laufenden Abschnitt definiert wurde, um einen besseren Textfluss zu gewährleisten.) [6]

$$\mathfrak{E}\big(\texttt{forall}\,(\langle \textit{getypte Liste(Variable)}\rangle^*)\langle \textit{Effektformel}\rangle\big) = \bigcup_{o_1\in T_{s_1},\ldots,o_n\in T_{s_n}} \mathfrak{E}\big(\langle \textit{Effektformel}\rangle\{\mathtt{v_1}/\mathtt{o_1},\ldots,\mathtt{v_n}/\mathtt{o_n}\}\big)$$

$$\text{für } \mathfrak{L}\big(\langle \textit{getypte Liste(Variable)}\rangle^*\big) = (v_1 : s_1 \quad \ldots \quad v_n : s_n)\ .$$

wobei $\langle\, \textit{Effektformel} \,\rangle\, \{v_1/o_1, \ldots, v_n/o_n\}$ die textuelle Ersetzung von v_1 durch o_1, ..., v_n durch o_n in $\langle\, \textit{Effektformel} \,\rangle$ darstellt.

Damit sind die benötigten Hilfskonstruktionen angegeben. Wir geben nun die Syntax der restlichen Konstruktionen von PDDL an.

7.2.3. PDDL–Domänen

Die Spezifikation eines Planungsproblems in PDDL erfolgt in zwei Teilen. Erstens der ⟨ ***PDDL-Domäne*** ⟩, die die verwendeten Typen, Objekte, Fluenten, Aktionen usw. beschreibt. Die ⟨ *PDDL-Domäne* ⟩ kann von mehreren Planungsproblemen verwendet werden. Ergänzt wird sie durch ein ⟨ *PDDL-Problem* ⟩, das Ausgangszustand und Ziel des Planungsproblems beschreibt. Die Syntax einer ⟨ *PDDL-Domäne* ⟩ ist wie folgt:

```
⟨ PDDL-Domäne ⟩        ::= (define (domain ⟨ Name ⟩)
                               [⟨ Typenliste ⟩]
                               [⟨ Konstantenliste ⟩]
                               [⟨ Fluentliste ⟩]
                               [⟨ Unveränderliche ⟩]
                               ⟨ Aktionsschema ⟩ * )
```

Als erstes kann in der ⟨ *PDDL-Domäne* ⟩ eine Hierarchie von Typen von Objekten (d.h. Argumenten von Fluenten) gegeben werden. Wie schon erwähnt, ist *object* implizit der Typ an der Spitze der Hierarchie.

```
⟨ Typenliste ⟩         ::= (:types ⟨ getypte Liste (Name) ⟩)
```

[6] Die Sorten, die den Objekttypen zugeordnet sind, enthalten ausschließlich Konstantensymbole, so dass die Anzahl dieser Terme endlich ist.

Sodann können Objekte deklariert werden, die allen Problemen einer ⟨ *PDDL-Domäne* ⟩ gemeinsam sind. Dabei werden die Objekte mit ihren zugehörigen Typen angegeben.

```
⟨ Konstantenliste ⟩    ::= (:constants ⟨ getypte Liste (Name) ⟩)
```

Als nächstes werden die Fluenten mit ihren Argumenttypen angegeben. Die an dieser Stelle verwendeten Variablen sind nur Platzhalter – die Namen sind nicht relevant.

```
⟨ Fluentliste ⟩        ::= (:predicates ⟨ Fluentkopf ⟩ +)
⟨ Fluentkopf ⟩         ::= (⟨ Fluentname ⟩
                            ⟨ getypte Liste (Variable) ⟩)
```

Es ist möglich, die Wahrheitswerte einzelner Fluenten anzugeben, die nicht von Aktionen verändert werden können. Dazu dient `:timeless`:

```
⟨ Unveränderliche ⟩    ::= (:timeless ⟨ Grundliteral ⟩ +)
⟨ Grundliteral ⟩       ::= ⟨ Grundatom ⟩
⟨ Grundliteral ⟩       ::= (not ⟨ Grundatom ⟩)
⟨ Grundatom ⟩          ::= (⟨ Fluentname ⟩ ⟨ Name ⟩ *)
```

Aufbauend auf den bereits erfolgten Deklarationen können nun Aktionen definiert werden. Dazu werden für jede Aktion drei Angaben benötigt: erstens die Typen der Parameter der Aktion (`:parameters`), zweitens die Vorbedingung der Aktion (`:precondition`), und drittens der Effekt der Aktion (`:effect`). `:vars` wird auf Seite 91 diskutiert.

```
⟨ Aktionsschema ⟩      ::= (:action ⟨ Aktionsname ⟩
                               :parameters
                                   ( ⟨ getypte Liste (Variable) ⟩ )
                               [:vars
                                   ( ⟨ getypte Liste(Variable) ⟩ )]
                               [:precondition ⟨ Zielformel ⟩]
                               [:effect ⟨ Effektformel ⟩])
⟨ Aktionsname ⟩        ::= ⟨ Name ⟩
```

In einer Aktion dürfen keine freien Variablen auftreten, d.h. alle Variablen, die in der ⟨ *Zielformel* ⟩ nach `:precondition` bzw. in der ⟨ *Effektformel* ⟩ nach `:effect` auftreten, müssen entweder in den Variablenlisten nach `:parameters` und `:vars` auftreten, oder in der Variablenliste eines `exists` oder `forall`, das dieses Auftreten der Variablen umfasst.

7.2.4. PDDL–Probleme

Ein ⟨ ***PDDL-Problem*** ⟩ gehört zu einer bestimmten ⟨ *PDDL-Domäne* ⟩, die in der Deklaration angegeben wird. Weiterhin können zu der ⟨ *PDDL-Domäne* ⟩-Deklaration zusätzliche Objekte eingeführt, sowie Ausgangszustand und Ziel angegeben werden.

```
⟨ PDDL-Problem ⟩        ::= (define (problem ⟨ Name ⟩)
                                 (:domain ⟨ Name ⟩)
                                     [ ⟨ Objektliste ⟩ ]
                                     [ ⟨ Ausgangszustand ⟩ ]
                                 ⟨ Ziel ⟩ +
⟨ Objektliste ⟩         ::= (:objects ⟨ getypte Liste (Name) ⟩)
⟨ Ausgangszustand ⟩     ::= (:init ⟨ Literal ⟩ +)
⟨ Ziel ⟩                ::= (:goal ⟨ Zielformel ⟩) )
```

Im Folgenden wird die Beschreibung eines Planungsproblems bestehend aus einer ⟨ *PDDL-Domäne* ⟩ und einem passenden ⟨ *PDDL-Problem* ⟩ als **PDDL–Text** bezeichnet.

7.3. Semantik von PDDL

Wir beschreiben nun, wie ein PDDL–Text in eine Fluentkalkül-Beschreibung der Planungsdomäne umzuwandeln ist. Zunächst konstruieren wir eine passende Fluentkalkül–Signatur Σ, die zusätzlich zu den Fluentkalkül-Grundelementen aus Σ_{FC} [7] die folgenden Sorten $\mathcal{S}$, Untersortenbeziehungen $\preceq$ und Funktionen $\mathcal{OP}$ enthält:

$\mathcal{S}$ sei die Menge aller in der ⟨ *Typenliste* ⟩ des PDDL–Texts vorkommenden Typennamen zuzüglich der Sorte *object*, bzw. nur *object*, falls keine ⟨ *Typenliste* ⟩ im Text vorkommt. Weiterhin seien für alle Vorkommen von `(either` $\mathtt{Typ}_1$ $\mathtt{Typ}_2$ `...`$\mathtt{Typ}_n$`)` im PDDL–Text jeweils ein Typ *either(*Typ_1*,*Typ_2*,...,*Typ_n*)* enthalten.

$\preceq$ ist die reflexive transitive Hülle von

- den Subtyprelationen, die sich aus der ⟨ *Typenliste* ⟩ ergeben: $t_1 \preceq t_2$ für $t_1 : t_2 \in \mathfrak{L}\big(\langle$ *Typenliste* $\rangle\big)$.
- $t_1, \ldots, t_n \preceq$ *either(*t_1*,...,*t_n*)* für alle Vorkommen von *either(*t_1*,*t_2*,...,*t_n*)* in $\mathcal{S}$.

[7] Für die Definition von Σ_{FC} siehe Seite 45.

- $t \preceq \mathit{object}$ für alle $t \in \mathcal{S}$.

$\mathcal{OP}$ ist eine Sammlung von Funktionssymbolen, die

- für jeden Fluentkopf
  (⟨ `Fluentname` ⟩ ⟨ `getypte Liste (Variable)` ⟩)
  in der ⟨ `Fluentliste-Deklaration` ⟩ ein Funktionssymbol

  $$\langle \texttt{Fluentname} \rangle : s_1 \dots s_n \to \mathit{Fluent}$$

  mit $\big(v_1 : s_1 \dots v_n : s_n\big) = \mathfrak{L}\big(\langle$ `getypte Liste(Variable)` $\rangle\big)$ enthält. Diese stellen die Fluenten dar.
- für jedes Paar $o_s : s$ von Objekt o_s und Sorte s aus $\mathfrak{L}\big(\langle$ `getypte Liste(Name)` $\rangle_1\big)$ und $\mathfrak{L}\big(\langle$ `getypte Liste(Name)` $\rangle_2\big)$, wobei `(:constants` ⟨ `getypte Liste(Name)` ⟩ 1 `)` und `(:objects` ⟨ `getypte Liste(Name)` ⟩ 2 `)` die ⟨ `Konstantenliste` ⟩ der PDDL–Domäne bzw. die ⟨ `Objektliste` ⟩ des PDDL–Problems sind, ein Konstantensymbol

  $$o_s : \to s$$

  enthält.

Beispiel (41, Fortsetzung)**.** *Für die Aktenmappenwelt ergibt sich eine Signatur Σ, die außer den allgemeinen Fluentkalkül-Elementen der Signatur Σ_{FC} die zusätzlichen Elemente*

SORT	*Portabel* $\preceq$ *object*, *Ort* $\preceq$ *object*	
FUN	*am-Ort*	: *Portabel* $\times$ *Ort* $\to$ *Fluent*,
	in-Mappe	: *Portabel* $\to$ *Fluent*,
	`Mappe`	: $\to$ *Portabel*,
	`Buch`	: $\to$ *Portabel*,
	`Scheck`	: $\to$ *Portabel*,
	`Haus`	: $\to$ *Ort*,
	`Buero`	: $\to$ *Ort*

enthält.

Zur Axiomatisierung der Gleichheit müssen neben den grundlegenden Fluentkalkül-Axiomen $\mathcal{F}_{mset}$ die Gleichheitsaxiome für die Fluenten, sowie die in Termen der Sorte *Fluent* auftretenden Sorten festgelegt werden. Die Idee von PDDL ist, dass jeder wohlsortierte Term ein anderes Fluent darstellt. Dem wird mit dem Einführen von Namenseindeutigkeit entsprochen:[8]

$$\mathrm{UNA}[\mathit{Func}_{\mathit{Fluent}}] \cup \bigcup_{s \in \mathcal{S}} \mathrm{UNA}[\mathit{Func}_s] \tag{7.1}$$

[8] Siehe (6.5) auf Seite 58.

wobei Func_s die Menge aller Funktions- und Konstantensymbole der Sorte s in $\mathcal{OP}$ ist. Weiterhin nutzt PDDL die **Annahme einer abgeschlossenen Welt** (engl. **closed world assumption**), d.h. im Planungsproblem existieren nur genau die Objekte, die tatsächlich im Text spezifiziert sind. Dies entspricht der **Abgeschlossenheit der Domäne**:

$$\text{CLOSED}[\mathit{Fluent}; \mathit{Func}_{\mathit{Fluent}}] \cup \bigcup_{s \in \mathcal{S}} \text{CLOSED}[s; \mathit{Func}_s] \ , \tag{7.2}$$

wobei die Abkürzung $\text{CLOSED}[s; f_1, \ldots, f_n]$ für eine Sorte s und eine Anzahl von Funktionssymbolen $f_1, \ldots, f_n$ der Sorte s wie folgt definiert ist:

$$\begin{aligned}&\text{CLOSED}[s; f_1, \ldots, f_n] = \\ &\left\{ (\forall\, x : s) \bigvee_{\substack{f \in \{f_1, \ldots, f_n\} \wedge \\ (f : s_1 \ldots s_n \to s) \in \mathcal{OP}}} (\exists\, x_1 : s_1, \ldots, x_n : s_n)\ x = f(x_1, \ldots, x_n) \right\} \ .\end{aligned} \tag{7.3}$$

Beispiel (41, Fortsetzung)**.** *Für das Aktenmappenbeispiel ergeben sich aus der Namenseindeutigkeit die Axiome*

$$\begin{aligned}&\big\{ (\forall\, x_1, y_1 : \mathit{Portabel}, x_2 : \mathit{Ort})\ \ \mathit{am\text{-}Ort}(x_1, x_2) \neq \mathit{in\text{-}Mappe}(y_1), \\ &(\forall\, x_1, y_1 : \mathit{Portabel}, x_2, y_2 : \mathit{Ort})\ \ \big[\mathit{am\text{-}Ort}(x_1, x_2) = \mathit{am\text{-}Ort}(y_1, y_2) \\ &\qquad\qquad \to\ (x_1 = y_1\ \wedge\ x_2 = y_2)\big], \\ &(\forall\, x, y : \mathit{Portabel})\ \ [\mathit{in\text{-}Mappe}(x) = \mathit{in\text{-}Mappe}(y)\ \to\ x = y]\big\} \cup \\ &\mathit{UNA}[\texttt{Buch}, \texttt{Scheck}, \texttt{Mappe}] \cup \mathit{UNA}[\texttt{Haus}, \texttt{Buero}] \\ &\qquad\qquad \cup\, \mathit{UNA}[\texttt{Buch}, \texttt{Scheck}, \texttt{Haus}, \texttt{Buero}, \texttt{Mappe}] \ .\end{aligned}$$

Einige der generierten Gleichungen sind redundant, da z.B. `Buch` *und* `Scheck` *sowohl der Sorte* Portabel *als auch der Sorte* object *angehören, und daher die Ungleichung* $\texttt{Buch} \neq \texttt{Scheck}$ *mehrfach durch UNA erzeugt wird.*

Aus der Abgeschlossenheit der Domäne ergeben sich die Axiome

$$\begin{aligned}&\Big\{ (\forall\, f : \mathit{Fluent})\ \ \big[(\exists\, x_1 : \mathit{Portabel}, x_2 : \mathit{Ort})\ \ f = \mathit{am\text{-}Ort}(x_1, x_2) \vee \\ &(\exists\, x : \mathit{Portabel})\ \ f = \mathit{in\text{-}Mappe}(x)\big], \\ &(\forall\, x : \mathit{Portabel})\ \ [x = \texttt{Buch}\ \vee\ x = \texttt{Scheck}\ \vee\ x = \texttt{Mappe}], \\ &(\forall\, x : \mathit{Ort})\ \ [x = \texttt{Buero}\ \vee\ x = \texttt{Haus}], \\ &(\forall\, x : \mathit{object})\ \ [x = \texttt{Buch}\ \vee\ x = \texttt{Scheck}\ \vee\ x = \texttt{Buero}\ \vee\ x = \texttt{Haus}\ \vee \\ &\qquad\qquad x = \texttt{Mappe}]\Big\} \ .\end{aligned}$$

Auf dieser Grundlage können wir uns nun der Übersetzung der einzelnen Bestandteile des im PDDL-Text dargestellten Planungsproblems widmen. Betrachten wir zunächst die Aktionen. Die Kodierung einer Aktionsbeschreibung als Zustandsübergangsaxiom

$$\Delta(s) \;\rightarrow\; \mathit{state}(\mathit{do}(a(\vec{v}, s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+$$

erfordert einigen Vorbedacht. Durch ϑ^- und ϑ^+ werden in einem Zustandsübergangsaxiom genau die Unterschiede des aktuellen und des Nachfolgezustands in Form der Fluenten in ϑ^-, deren Wert von $\mathbf{1}$ nach $\mathbf{0}$ wechselt, und der Fluenten in ϑ^+, deren Wahrheitswert von $\mathbf{0}$ nach $\mathbf{1}$ wechselt, angegeben. Die PDDL–Aktionsbeschreibung folgt zwar ebenfalls der STRIPS-Idee, dass alle Fluenten, über die keine Angaben gemacht werden, ihren Wahrheitswert nicht verändern, aber für die Fluenten, die die Aktion verändern kann, werden explizite Angaben über den Wahrheitswert im Nachfolgezustand gemacht, anstatt nur die Unterschiede zum vorhergehenden Zustand zu benennen. So wird z.B. in PDDL eine Aktion "Licht an", deren Effekt einfach darin besteht, dass es hell ist, z.B. so kodiert:

```
(:action Licht-an
    :effect (hell)
)
```

Im Fluentkalkül sind dafür aber zwei Zustandsübergangsaxiome nötig, da es erstens möglich ist, dass die Aktion den Zustand unverändert lässt, falls es bereits *hell* ist:

$$\mathit{Holds}(\mathit{hell}, s) \;\rightarrow\; \mathit{state}(\mathit{do}(\mathit{licht\text{-}an}, s)) = \mathit{state}(s) \;\;.$$

und zweitens, dass das Fluent *hell* von $\mathbf{0}$ nach $\mathbf{1}$ wechselt:

$$\neg\, \mathit{Holds}(\mathit{hell}, s) \;\rightarrow\; \mathit{state}(\mathit{do}(\mathit{licht\text{-}an}, s)) = \mathit{state}(s) \circ \mathit{hell} \;\;.$$

Dementsprechend ist also für jede Teilmenge der Fluenten, deren Zustand durch die Effektbeschreibung der Aktion spezifiziert wird, ein Zustandsübergangsaxiom nötig. Dieses beschreibt jeweils die Ausführung der Aktion, bei der diese Teilmene von Fluenten sich verändert, während die restlichen Fluenten bereits vorher erfüllt waren. Dabei können allerdings eine große Anzahl der Zustandsübergangsaxiome durch Wechselwirkung mit den Vorbedingungen der Aktion zu Tautologien werden, die weggelassen werden können. Weiterhin ist zu beachten, dass die Effekte der Aktion bedingt sein können, so dass diese Verfahrensweise für die verschiedenen Varianten der bedingten Effekte durchgeführt werden muss. Es ergibt sich also der im Folgenden beschriebene Algorithmus, der sinngemäß dem in [85] beschriebenen entspricht.

Sei ein Aktionsschema

```
(:action ⟨Aktionsname⟩
      :parameters (⟨getypte Liste(Variable)⟩)
     :precondition ⟨Zielformel⟩ :effect ⟨Effektformel⟩ )
```

aus dem PDDL-Text gegeben und sei

$$a = \langle \texttt{Aktionsname} \rangle$$

der Name der Aktion,

$$(x_1 : s_1 \ \ldots \ x_n : s_n) = \mathfrak{L}\big(\langle \texttt{getypte Liste(Variable)} \rangle\big)$$

die Sequenz der Argumentvariablen,

$$\pi = \mathfrak{P}\big(\langle \texttt{Zielformel} \rangle\big)$$

die Vorbedingung der Aktion,

$$\mathcal{E} = \mathfrak{E}\big(\langle \texttt{Effektformel} \rangle\big)$$

die Menge der Effekte mit ihren Vorbedingungen.

Wenn die Vorbedingung in der PDDL–Aktionsbeschreibung nicht spezifiziert ist, wird an Stelle von $\mathfrak{P}\big(\langle\, \texttt{Zielformel} \,\rangle\big)$ der Wert $\mathbf{1}$ verwendet; bei weggelassenem Effekt wird die leere Menge an Stelle von $\mathfrak{E}\big(\langle\, \texttt{Effektformel} \,\rangle\big)$ eingesetzt. Ein Tupel $\langle a, (x_1 : s_1 \ \ldots \ x_n : s_n), \pi, \mathcal{E} \rangle$ wird im Folgenden **Aktionsbeschreibung** genannt.

Beispiel (41 Fortsetzung)**.** *In der Aktenmappenwelt ergeben sich für die einzelnen Aktionen die Aktionsbeschreibungen*[9]

$$\Big\langle \mathit{Herausnehmen},\ (?x : \mathit{Portabel}),\ \mathit{in\text{-}Mappe}(?x),\ \big\{\big\langle \mathbf{1}, \overline{\mathit{in\text{-}Mappe}(?x)}\,\big\rangle\big\}\Big\rangle,$$

$$\Big\langle \mathit{Hineinlegen},\ (?x : \mathit{Portabel} \quad ?l : \mathit{Ort}),$$

$$\mathit{am\text{-}Ort}(?x, ?l)\ \wedge\ \mathit{am\text{-}Ort}(\texttt{Mappe}, ?l)\ \wedge\ \overline{\mathit{in\text{-}Mappe}(?x)},\ \big\{\big\langle \mathbf{1}, \mathit{in\text{-}Mappe}(?x)\big\rangle\big\}\Big\rangle,$$

$$\Big\langle \mathit{Transport},\ (?m : \mathit{Ort} \quad ?l : \mathit{Ort}),\ \mathit{am\text{-}Ort}(\texttt{Mappe}, ?m) \wedge ?m \neq ?l,$$

$$\big\{\big\langle \mathbf{1}, \mathit{am\text{-}Ort}(\texttt{Mappe}, ?l)\big\rangle,\ \big\langle \mathbf{1}, \overline{\mathit{am\text{-}Ort}(\texttt{Mappe}, ?m)}\,\big\rangle,$$
$$\big\langle \mathit{in\text{-}Mappe}(\texttt{Buch}), \mathit{am\text{-}Ort}(\texttt{Buch}, ?l)\big\rangle,$$
$$\big\langle \mathit{in\text{-}Mappe}(\texttt{Buch}), \overline{\mathit{am\text{-}Ort}(\texttt{Buch}, ?m)}\,\big\rangle$$
$$\big\langle \mathit{in\text{-}Mappe}(\texttt{Scheck}), \mathit{am\text{-}Ort}(\texttt{Scheck}, ?l)\big\rangle,$$
$$\big\langle \mathit{in\text{-}Mappe}(\texttt{Buch}), \overline{\mathit{am\text{-}Ort}(\texttt{Scheck}, ?m)}\,\big\rangle$$
$$\big\langle \mathit{in\text{-}Mappe}(\texttt{Mappe}), \mathit{am\text{-}Ort}(\texttt{Mappe}, ?l)\big\rangle,$$
$$\big\langle \mathit{in\text{-}Mappe}(\texttt{Buch}), \overline{\mathit{am\text{-}Ort}(\texttt{Mappe}, ?m)}\,\big\rangle \qquad \big\}\ \Big\rangle\ .$$

[9] Vergleiche Seite 75.

Algorithmus 42. *Für eine Aktionsbeschreibung* $\langle a, (x_1 : s_1 \ \ldots \ x_n : s_n), \pi, \mathcal{E} \rangle$ *werden die Zustandsübergangsaxiome wie folgt konstruiert:*

Sei $V = \{\varphi_p \mid \langle \varphi_p, \varphi_e \rangle \in \mathcal{E}\}$ *die Menge aller Vorbedingungen für die bedingten Effekte. Für alle Teilmengen* $V_p \subseteq V$ *:*

> *Sei* $E = \{\varphi_e \mid \langle \varphi_p, \varphi_e \rangle \in \mathcal{E} \ \wedge \ \varphi_p \in V_p\}$ *die Menge aller Effekte, die Bedingungen aus* V_p *haben. Füge das folgende Zustandsübergangsaxiome für alle Teilmengen* $E_c \subseteq E$ *hinzu, falls die linke Seite der Implikation mit* $\mathcal{F}_{mset}$ *konsistent ist:*
>
> $$\Big[\mathit{Holds}(\pi, s) \ \wedge \bigwedge_{\varphi_p \in V_p} \mathit{Holds}(\varphi_p, s) \ \wedge \bigwedge_{\varphi'_p \in V \setminus V_p} \neg\, \mathit{Holds}(\varphi'_p, s)$$
> $$\wedge \bigwedge_{f \in E_c} \neg\, \mathit{Holds}(f, s) \ \wedge \bigwedge_{f' \in E \setminus E_c} \mathit{Holds}(f', s)\Big]$$
> $$\rightarrow \ \mathit{state}(\mathit{do}(a(x_1, \ldots, x_n), s)) \circ \bigcirc_{f \in E_c^-} f = \mathit{state}(s) \circ \bigcirc_{f' \in E_c^+} f'$$

Hierbei ist $\bigcirc_{f \in \{f_1, \ldots, f_n\}}$ eine Kurzform für $\emptyset \circ f_1 \circ \ldots \circ f_n$. Die Konsistenzbedingung eliminiert dabei Tautologien aus den Zustandsübergangsaxiomen.

Beispiel (Anwendung des Algorithmus 42)**.** *Betrachten wir die Aktion Transport der Aktenmappenwelt:*

```
(:action Transport
   :parameters (?m ?l - Ort)
   :precondition (and (am-Ort Mappe ?m) (not (= ?m ?l)))
   :effect (and (am-Ort Mappe ?l) (not (am-Ort Mappe ?m))
                (forall (?z - Portabel)
                     (when (in-Mappe ?z)
                           (and (am-Ort ?z ?l)
                                (not (am-Ort ?z ?m)))))))
)
```

mit der dazugehörigen Aktionsbeschreibung $\langle a, (x_1 : s_1 \ \ldots \ x_n : s_n), \pi, \mathcal{E} \rangle$ *:*

$$
\begin{aligned}
& a = \textit{Transport}, \\
& (x_1 : s_1 \ \ldots \ x_n : s_n) = (?m : \textit{Ort} \quad ?l : \textit{Ort}), \\
& \pi = \textit{am-Ort}(\texttt{Mappe}, ?m) \wedge ?m \neq ?l, \\
& \mathcal{E} = \left\{ \begin{array}{l}
\big\langle \mathbf{1}, \textit{am-Ort}(\texttt{Mappe}, ?l) \big\rangle, \ \big\langle \mathbf{1}, \overline{\textit{am-Ort}(\texttt{Mappe}, ?m)} \big\rangle, \\
\big\langle \textit{in-Mappe}(\texttt{Buch}), \textit{am-Ort}(\texttt{Buch}, ?l) \big\rangle, \\
\big\langle \textit{in-Mappe}(\texttt{Buch}), \overline{\textit{am-Ort}(\texttt{Buch}, ?m)} \big\rangle, \\
\big\langle \textit{in-Mappe}(\texttt{Scheck}), \textit{am-Ort}(\texttt{Scheck}, ?l) \big\rangle, \\
\big\langle \textit{in-Mappe}(\texttt{Buch}), \overline{\textit{am-Ort}(\texttt{Scheck}, ?m)} \big\rangle, \\
\big\langle \textit{in-Mappe}(\texttt{Mappe}), \textit{am-Ort}(\texttt{Mappe}, ?l) \big\rangle, \\
\big\langle \textit{in-Mappe}(\texttt{Buch}), \overline{\textit{am-Ort}(\texttt{Mappe}, ?m)} \big\rangle
\end{array} \right\}
\end{aligned}
$$

Damit ergibt sich die Menge V *zu*

$$V = \{\mathbf{1}, \textit{in-Mappe}(\texttt{Scheck}), \textit{in-Mappe}(\texttt{Buch}), \textit{in-Mappe}(\texttt{Mappe})\} \ .$$

Von dieser gibt es 16 Teilmengen V_p*. Für die Mengen, die* $\mathbf{1}$ *nicht enthalten, ist*

$$\bigwedge_{\varphi_p \in V_p} \textit{Holds}(\varphi_p, s) \ \wedge \bigwedge_{\varphi'_p \in V \setminus V_p} \neg \textit{Holds}(\varphi'_p, s)$$

nicht erfüllbar, da die Konjunktion $\neg \mathbf{1}$ *enthält, so dass die daraus konstruierten Zustandsübergangsaxiome Tautologien wären; es sind also nur die anderen 8 davon zu bearbeiten. Betrachten wir als Beispiel die Menge*

$$V_p = \{\mathbf{1}, \textit{in-Mappe}(\texttt{Scheck})\} \ .$$

Damit wird

$$E = \big\{\textit{am-Ort}(\texttt{Mappe}, ?m), \neg\, \textit{am-Ort}(\texttt{Mappe}, ?l), \textit{am-Ort}(\texttt{Scheck}, ?m), \neg\, \textit{am-Ort}(\texttt{Scheck}, ?l)\big\} \ .$$

Sei E_c *zum Beispiel*

$$E_c = \{\textit{am-Ort}(\texttt{Mappe}, ?m), \neg\, \textit{am-Ort}(\texttt{Mappe}, ?l)\} \ .$$

Es wird das folgende Axiom hinzugefügt:

$$
\begin{aligned}
& \textit{am-Ort}(\texttt{Mappe}, ?m) \wedge ?m \neq ?l \wedge \\
& \textit{Holds}(\mathbf{1}, s) \ \wedge \ \textit{Holds}(\textit{in-Mappe}(\texttt{Scheck}), s) \wedge \\
& \neg\, \textit{Holds}(\textit{in-Mappe}(\texttt{Buch}), s) \ \wedge \ \neg\, \textit{Holds}(\textit{in-Mappe}(\texttt{Mappe}), s) \wedge \\
& \textit{Holds}(\textit{am-Ort}(\texttt{Mappe}, ?m), s) \ \wedge \ \textit{Holds}(\neg\, \textit{am-Ort}(\texttt{Mappe}, ?l), s) \wedge \\
& \neg\, \textit{Holds}(\textit{am-Ort}(\texttt{Scheck}, ?m), s) \ \wedge \ \neg\, \textit{Holds}(\neg\, \textit{am-Ort}(\texttt{Scheck}, ?l), s) \ \rightarrow \\
& \textit{state}(\textit{do}(\textit{Transport}(?m, ?l), s)) \circ \textit{am-Ort}(\texttt{Mappe}, ?l) \\
& \qquad\qquad = \textit{state}(s) \circ \textit{am-Ort}(\texttt{Mappe}, ?m) \ .
\end{aligned}
$$

Insgesamt ergeben sich 125 Zustandsübergangsaxiome für diese Aktion. Viele davon haben identische Gleichungen auf der rechten Seite, so dass diese bei Bedarf zu 64 Zustandsübergangsaxiomen zusammengefasst werden können.[10]

Eine Besonderheit ist die `:vars` Konstruktion in einem Aktionsschema des PDDL–Texts. In Planungsproblemen kommt es häufig vor, dass man, ein bestimmtes Objekt identifizieren muss, das nicht direkt in den Parametern der Aktion erscheint, um die Effekte einer Aktion zu ermitteln. Zum Beispiel hat in der Welt der Blöcke (Beispiel 21 auf Seite 43) die Aktion *PutOn*(x, y) , die den Block x auf den Block y stellt, den (Neben-)Effekt, dass der Block z , auf dem der Block x vor Ausführung der Aktion steht, frei wird. Der Effekt der Aktion ist also u.a. *Clear*(z) für den Block z , der *On*(x, z) erfüllt. Um diese Art von Bedingungen elegant spezifizieren zu können wurde `:vars` eingeführt. Das Argument von `:vars` ist eine Liste von Variablen (siehe Seite 83), die sich in der Vorbedingung der Aktion existentiell quantifiziert verhalten. Dabei wäre es aber ein Fehler in der Planungsdomänenspezifikation, wenn es mehr als eine Instanziierung dieser Variablen gibt, so dass die Vorbedingung erfüllt ist.[11] Die so gefundene Instanziierung dieser Variablen wird dann in die Effekte der Aktion eingesetzt. Eine PDDL–Beschreibung der Aktion *PutOn* könnte also z.B. so aussehen:

```
(:action puton
         :parameters (?x - block ?y - block)
         :vars (?z - block)
         :precondition (and (on ?x ?z) (clear ?x) (clear ?y))
         :effect (and (clear ?z) (on ?x ?y) (not (clear ?y))))
```

Durch die Vorbedingung wird `?z` zu dem Block instanziiert, für den `(clear ?x ?z)` gilt, und dieser Wert wird dann in die Effekte eingesetzt.

In unserer[12] Übersetzung wird nun eine Aktion mit `:vars` zunächst umgeformt. Aus

```
(:action ⟨Aktionsname⟩
         :parameters ( ⟨getypte Liste (Variable)⟩1 )
         :vars ( ⟨getypte Liste(Variable)⟩2 )
         :precondition ⟨Zielformel⟩
         :effect ⟨Effektformel⟩)
```

wird ein neues Aktionsschema erzeugt:

[10] Es sind sparsamere Möglichkeiten zur Spezifikation einer solchen Aktion im Fluentkalkül mit Hilfe von Ramifikation [83] bekannt, aber dies ist nicht Gegenstand dieser Arbeit.

[11] d.h. dies ist durch die Spezifikation der Planungsdomäne zu gewährleisten; das Planungsprogramm kann diese Eigenschaft einfach voraussetzen.

[12] Die Interpretation des `:vars`-Konstrukts ist in [37] nicht ganz klar; die hier angegebene Interpretation entstand in privater Email-Konversation mit Drew McDermott und ist konsistent mit dem in [37] auf Seite 8 gegebenen Beispiel für `:vars`.

```
(:action ⟨Aktionsname⟩
         :parameters ( ⟨getypte Liste (Variable)⟩_1 )
         :precondition (exists ( ⟨getypte Liste(Variable)⟩_2 )
                               ⟨Zielformel⟩)
         :effect (forall ( ⟨getypte Liste(Variable)⟩_2 )
                           (when ⟨Zielformel⟩) ⟨Effektformel⟩))   .
```

Unter der Bedingung, dass es höchstens eine Instanziierung der Variablen in ⟨ `getypte Liste(Variable)` ⟩$_2$ gibt, so dass die Vorbedingung ⟨ `Zielformel` ⟩ der Aktion erfüllt wird, ist dies äquivalent zur oben beschriebenen Interpretation von `:vars` – die neue generierte Vorbedingung ist genau dann erfüllt, wenn eine solche Belegung existiert, und die Kombination von `forall` und `when` im neu generierten Effekt selektiert diese Belegung für die Effekte der Aktion. Das so umgeformte Aktionsschema kann mit dem bereits angegebenen Algorithmus übersetzt werden.

Komplettiert wird die Übersetzung der PDDL-Planungsdomäne durch die Übersetzung des Initialzustands und des Ziels. Sei dieser spezifiziert als ⟨ `Ausgangszustand` ⟩: `(:init` ⟨ `Literal` ⟩$_1$ `...` ⟨ `Literal` ⟩$_n$ `)`. Dann ergibt sich eine Formel

$$\mathit{state}(\mathrm{S}_0) = \bigcirc \left\{ f \mid f \in T_{\mathit{Fluent}} \wedge \langle \mathbf{1}, f \rangle \in \mathfrak{E}\big(\texttt{(and } \langle \texttt{Literal} \rangle_1 \ldots \langle \texttt{Literal} \rangle_{\texttt{n}} \texttt{)}\big) \right\} \tag{7.4}$$

die dem Initialzustand die Verknüpfung aller im ⟨ `Ausgangszustand` ⟩ als wahr spezifizierten Fluenten mit $\circ$ zuweist. Das Ziel des Planungsproblems drückt sich durch das zu lösende Folgerungsproblem aus:

$$\mathcal{F}_{PDDL} \models (\exists\, s : \mathit{State})\ \mathit{Holds}(\mathfrak{E}\big(\langle \texttt{Zielformel} \rangle\big), s)\ .$$

Dabei ist $\mathcal{F}_{PDDL}$ eine Formelmenge, die aus $\mathcal{F}_{mset}$, (7.1), (7.2), (7.4), sowie den von Algorithmus 42 generierten Zustandsübergangsaxiomen besteht.

Beispiel (41, Fortsetzung)**.** *In der Aktenmappenwelt sind der Ausgangszustand und das Ziel spezifiziert als:*

```
(:init (am-Ort Mappe Haus) (am-Ort Buch Haus)
       (am-Ort Scheck Haus) (in-Mappe Scheck))
(:goal (and (am-Ort Mappe Buero) (am-Ort Buch Buero)
            (am-Ort Scheck Haus)))
```

Damit ergibt sich

$$\mathit{state}(\mathrm{S}_0) = \mathit{am\text{-}Ort}(\texttt{Mappe}, \texttt{Haus}) \circ \mathit{am\text{-}Ort}(\texttt{Buch}, \texttt{Haus}) \circ \mathit{am\text{-}Ort}(\texttt{Scheck}, \texttt{Haus}) \circ \mathit{in\text{-}Mappe}(\texttt{Scheck})$$

für den Ausgangszustand, und das Planungsproblem stellt sich als das Folgerungsproblem

$$\mathcal{F}_{PDDL} \models (\exists\, s : \textit{State}) \begin{pmatrix} \textit{Holds}(\textit{am-Ort}(\texttt{Mappe}, \texttt{Buero}), s) \wedge \\ \textit{Holds}(\textit{am-Ort}(\texttt{Mappe}, \texttt{Buero}), s) \wedge \\ \textit{Holds}(\textit{am-Ort}(\texttt{Scheck}, \texttt{Haus}), s) \end{pmatrix}$$

dar, wobei $\mathcal{F}_{PDDL}$ alle aus der PDDL-Beschreibung generierten Axiome enthält. Um einen Plan zu finden ist natürlich ein konstruktiver Beweis für dieses Problem erforderlich, d.h. ein Beweis unter Angabe eines s, dass

$$\mathcal{F}_{PDDL} \models \begin{pmatrix} \textit{Holds}(\textit{am-Ort}(\texttt{Mappe}, \texttt{Buero}), s) \wedge \\ \textit{Holds}(\textit{am-Ort}(\texttt{Mappe}, \texttt{Buero}), s) \wedge \\ \textit{Holds}(\textit{am-Ort}(\texttt{Scheck}, \texttt{Haus}), s) \end{pmatrix}$$

erfüllt. Dies ist z.B. für

$$s = \textit{do}([\textit{Herausnehmen}(\texttt{Scheck}), \textit{Hineinlegen}(\texttt{Buch}, \texttt{Haus}), \textit{Transport}(\texttt{Haus}, \texttt{Buero})], \texttt{S}_0)$$

erfüllt.[13]

[13] Zum Vergleich ist der gesamte PDDL-Quelltext des Aktenmappen-Beispiels noch einmal im Anhang A.1 auf Seite 145 zu finden.

8. BDD-unterstütztes Planen im Fluentkalkül

In diesem Kapitel wird die Möglichkeit diskutiert, deduktives Planen für ein deterministisches aussagenlogisches Planungsproblem mit vollständig spezifiziertem Initialzustand mit Hilfe von BDDs zu unterstützen, und ein entsprechender Algorithmus entwickelt.

Betrachten wir ein Planungsproblem wie die bisher betrachteten Beispiele, bei denen ein bestimmter Ausgangszustand gegeben ist, und eine Aktionssequenz gesucht wird, nach deren Ausführung ein Zustand aus einer Menge von Zielzuständen erreicht wird. Eine mögliche Grundidee für den Planungsvorgang ist wie folgt: Man generiert logische Konsequenzen aus der Axiomatisierung der Domäne, die aussagen, dass ein bestimmter Zustand nach Ausführung einer Aktionssequenz erreicht wird, so lange bis eine solche Konsequenz gefunden wird, für die der erreichte Zustand ein Zielzustand ist. Es ist nun zu überprüfen, wie dies bei Benutzung unseres Formalismus aussehen kann.

Im Fluentkalkül stellt sich nun die Aussage, dass ein Zustand z_1 nach Ausführung der Aktionsfolge $a_1, \ldots, a_n$ erreicht wird, als $z_n = \mathit{state}(\mathit{do}([a_1, \ldots, a_n], \mathrm{S}_0))$ dar. Eine Aussage dieser Form bezeichnen wir im Folgenden als eine **Zustandsaussage**. Die Ermittlung aller Zustandsaussagen, die sich aus der Planungsdomänenaxiomatisierung ergeben, kann nun schrittweise geschehen: Die Spezifikation des Initialzustands ergibt eine Aussage der Form $z_0 = \mathit{state}(\mathrm{S}_0)$, und die Zustandsübergangsaxiome stellen die Verbindung von Aussagen der Form $z_n = \mathit{state}(\mathit{do}([a_1, \ldots, a_n], \mathrm{S}_0))$ und $z_{n+1} = \mathit{state}(\mathit{do}([a_1, \ldots, a_n, a_{n+1}], \mathrm{S}_0))$ her. Es werden also schrittweise die Mengen von Zuständen bestimmt, die nach der Ausführung von n Aktionen erreicht werden können (Abb. 8.1).

In welcher Form lässt sich nun dieser Prozess durch BDDs unterstützen? Dazu ist zunächst zu überlegen, was man mit BDDs überhaupt darstellen kann.

Wie in Kapitel 3 beschrieben, eignen sich BDDs zur Darstellung von einer endlichen Grundmenge, wie z.B. der Menge der Fluenten. Da in dem von uns betrachteten Fragment von PDDL nur aussagenlogische Fluenten auftauchen, lassen sich die Zustände als solche Teilmengen der Menge aller Fluenten darstellen. Offenbar ist es aber schlecht möglich, die unendliche Menge der Situationen (d.h. in unserem Fall Aktionssequenzen) darzustellen. Daher bietet sich eine Einschränkung an: Man bestimmt nur die Zustände, die erreicht werden können (d.h. alle z , für die sich eine Aussage $z = \mathit{state}(s)$ in der jeweiligen Menge $\mathcal{Z}_i$ befindet), bis ein Zustand z_n gefunden ist, der das Planungsziel erfüllt, und rekonstruiert dann rückwärts den Weg

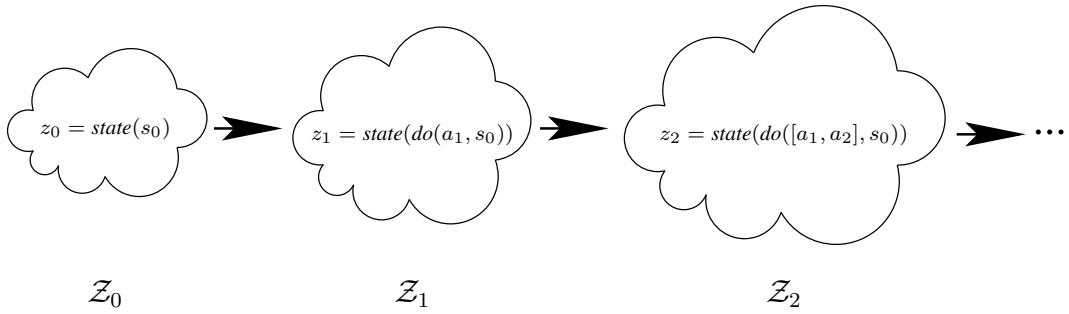


Abbildung 8.1.: Der Planungsalgorithmus: BDD-basierte, schrittweise Bestimmung der Mengen $\mathcal{Z}_i$, die die Mengen von Zuständen z_i darstellen, die nach i Aktionen vom Ausgangszustand erreicht werden können.

$z_0, z_1, \ldots, z_n$, auf dem dieser Zustand erreicht werden kann, und ermittelt zum Schluss als Lösung des Planungsproblems eine Aktionssequenz, die dieser Zustandssequenz entspricht. Diese Verfahrensweise ist analog zum Algorithmus 18 auf Seite 36 aus Kapitel 4.

8.1. Grundidee

Wir treffen zunächst die folgenden Voraussetzungen, die – wie in diesem Kapitel gezeigt werden wird – hinreichend sind, um den Planungsprozess für ein Fluentkalkül–Planungsproblem mit Hilfe von BDDs zu realisieren. Die beschriebene Axiomatisierung von PDDL-Texten entspricht automatisch den getroffenen Annahmen.

Annahme 43. *Wir nehmen an, dass $\mathcal{F}_{mset}, \mathcal{F}_{un}, \mathcal{F}_{sua}, \Phi_I, \Phi_G$ eine widerspruchsfreie Axiomatisierung eines Planungsproblems mit endlicher Fluentenanzahl ist, wobei gilt:*

- *$\mathcal{F}_{mset}$ ist die Axiomatisierung von $\circ$ aus Kap. 6.*
- *$\mathcal{F}_{un}$ ist eine Menge von Axiomen, die festlegt, dass die Sorte* Fluent *genau der Menge T_{Fluent} der Grundterme der Sorte* Fluent *entspricht. D.h., syntaktisch unterschiedliche Terme müssen auch semantisch unterschiedlich sein und die Sorte* Fluent *erschöpft sich mit diesen Termen. Dies kann mit der „Unique Names Assumption“ (vergleiche Abschnitt 6.4 auf Seite 58) und der „Closed World Assumption“ (vergleiche* (7.3) *auf Seite 86) erreicht werden.*
- *$\mathcal{F}_{sua}$ ist eine Menge von Zustandsübergangsaxiomen in der Form entsprechend* (6.7) *auf Seite 65.*
- *Φ_I ist ein Axiom der Form $state(\mathrm{S}_0) = t$, wobei t ein Grundterm Sorte* Fluent *ist.*

- $\Phi_G(z)$ *ist eine boolesche Kombination von Aussagen der Form* $\mathit{Holds}(f, z)$*, die das Ziel des Planungsproblems charakterisiert.*

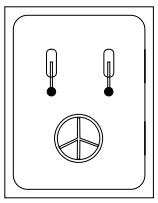
Das Ziel des Planungsproblems ist dabei als eine Formel $\Phi_G(z)$ mit einer freien Variablen z vom Typ *State* dargestellt, die die Eigenschaft kodiert, die ein Zustand erfüllen muss, um das Planungsproblem zu lösen. In der Fluentkalkül–Beschreibung stellt sich dann das Planungsproblem als das folgende Folgerungsproblem dar:

$$\mathcal{F} \models (\exists\, s : \mathit{Sit}, z : \mathit{State})\ \ [z = \mathit{state}(s)\ \wedge\ \Phi_G(z)]\ \ , \tag{8.1}$$

wobei $\mathcal{F} = \mathcal{F}_{mset} \cup \mathcal{F}_{un} \cup \mathcal{F}_{sua} \cup \{\Phi_I\}$. Ein konstruktiver Beweis dieser Aussage liefert, als Antwortsubstitution für s kodiert, einen Plan und in z den Zustand, der nach Ausführung des Plans besteht.

Beispiel 44. *Ein (absurd einfacher) Safe habe zwei Hebel, die beide nach oben bewegt werden müssen, damit der Safe geöffnet werden kann. Der Zustand des Systems sei durch folgende Fluenten charakterisiert:*

L : der linke Hebel ist oben,
R : der rechte Hebel ist oben,
O : die Tür ist offen.



Es stehen die Aktionen ToggleL und ToggleR zum Umschalten der beiden Hebel zur Verfügung, sowie die Aktion Open zum Öffnen des Safes, die natürlich nur dann funktioniert, wenn die beiden Hebel oben sind.

Der Kodierung im Fluentkalkül liegt eine Signatur zugrunde, die neben den Elementen von Σ_{FC} *(Seite 45) die folgenden Elemente enthält:*

SORT	*object*,	
FUN	L, R, O	: *Fluent*,
	$\mathit{ToggleL}, \mathit{ToggleR}, \mathit{Open}$	: *Action*.

Es lässt sich nun folgender Axiomsatz konstruieren, der der Annahme 43 entspricht:

$$\begin{aligned}
\mathcal{F}_{un} = \{\ & L \neq R,\ L \neq O,\ R \neq O, \\
& (\forall\, f : \mathit{Fluent})\ [f = L\ \vee\ f = R\ \vee\ f = O] \qquad \},
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}_{sua} = \{\ & (\forall\, s : \mathit{Sit})\ [\mathit{Holds}(L, s)\ \rightarrow\ \mathit{state}(\mathit{do}(\mathit{ToggleL}, s)) \circ L = \mathit{state}(s)], \\
& (\forall\, s : \mathit{Sit})\ [\neg\, \mathit{Holds}(L, s)\ \rightarrow\ \mathit{state}(\mathit{do}(\mathit{ToggleL}, s)) = \mathit{state}(s) \circ L], \\
& (\forall\, s : \mathit{Sit})\ [\mathit{Holds}(R, s)\ \rightarrow\ \mathit{state}(\mathit{do}(\mathit{ToggleL}, s)) \circ R = \mathit{state}(s)], \\
& (\forall\, s : \mathit{Sit})\ [\neg\, \mathit{Holds}(R, s)\ \rightarrow\ \mathit{state}(\mathit{do}(\mathit{ToggleL}, s)) = \mathit{state}(s) \circ R], \\
& (\forall\, s : \mathit{Sit})\ [\mathit{Holds}(L, s)\ \wedge\ \mathit{Holds}(R, s)\ \wedge\ \neg\, \mathit{Holds}(O, s) \\
& \qquad\qquad \rightarrow\ \mathit{state}(\mathit{do}(\mathit{Open}, s)) = \mathit{state}(s) \circ O] \qquad \},
\end{aligned}$$

$$\Phi_I = \ \mathit{state}(\mathtt{S}_0) = \emptyset,$$

$$\Phi_G(z) = \ \mathit{Holds}(O, z).$$

Diesen Axiomsatz erhält man (nach einigen äquivalenten Umformungen zur Übersichtlichkeit) durch Anwenden des in Kapitel 7 beschriebenen Algorithmus aus dem folgenden PDDL–Text:

```
(define (domain Safe)
  (:predicates (L) (R) (O))
  (:action ToggleL
           :parameters ()
           :effect (and
                    (when (L) (not (L)))
                    (when (not (L)) (L))))
  (:action ToggleR
           :parameters ()
           :effect (and
                    (when (R) (not (R)))
                    (when (not (R)) (R))))
  (:action Open
           :precondition (and (L) (R))
           :effect (O))
)

(define (problem Safe-Offen)
  (:domain Safe)
  (:init )
  (:goal (O))
)
```

Wie der Leser leicht nachvollziehen kann, wird (8.1) für dieses Planungsproblem z.B. durch $s = do([ToggleL, ToggleR, Open], \mathrm{S}_0)$ mit $z = L \circ R \circ O$ gelöst. Mit anderen Worten: Eine korrekte Antwortsubstitution σ für die Anfrage

$$\mathcal{F} \models^{?} z = state(s) \ \wedge \ \Phi_G(z) \ , \tag{8.2}$$

ist

$$\sigma = \{s/do([ToggleL, ToggleR, Open], \mathrm{S}_0), z/L \circ R \circ O\} \ . \tag{8.3}$$

Im Folgenden arbeiten wir nun die Idee näher aus, ein Planungsproblem dadurch zu lösen, dass man korrekte Antwortsubstitutionen bzw. Mengen von korrekten Antwortsubstitutionen σ für Anfragen $\mathcal{F} \models \phi\sigma$ mit einer Fluentkalkül-Formel ϕ mit Hilfe von BDDs kodiert bzw. in BDD-Form berechnet. Wie bereits in Kapitel 3 besprochen, eignen sich BDDs zur Darstellung von Teilmengen von endlichen Mengen sowie zur Berechnung von Operationen darüber. Die Antwortsubstitution (8.3) besteht nun aus zwei Bestandteilen:

- der Bindung $s/do([\mathit{ToggleL}, \mathit{ToggleR}, \mathit{Open}], \mathtt{S}_0)$. Diese ist in einer unendlichen Grundmenge möglicher Substitutionen von s durch Terme der Sorte *Sit* enthalten, da hier eine beliebig lange Aktionssequenz kodiert wird.[1]

- der Bindung $z/L \circ R \circ O$. Diese ist für ein aussagenlogisches Planungsproblem in einer nur endlichen Grundmenge von Substitutionen der Variable z durch Terme vom Typ *State* enthalten, da der substituierte Term aus der endlichen Menge der Fluenten jedes Fluent wegen (NonMult) höchstens einmal enthalten kann.[2]

Wenn wir nun versuchen diese Bestandteile gemäß der in Kapitel 3 beschriebenen Verfahren zu kodieren, dann ist der erste Bestandteil aufgrund der unendlichen Grundmenge möglicher Bindungen problematisch; der zweite lässt sich jedoch problemlos kodieren. Eine kleine Umformung des Problems ermöglicht es jedoch, diesen Bestandteil zu vermeiden: Anstatt der Anfrage (8.2) verwendet man die äquivalente Anfrage

$$\mathcal{F} \stackrel{?}{\models} (\exists\, s : \mathit{State})\ \ z = \mathit{state}(s)\ \wedge\ \Phi_G(z)\ . \tag{8.4}$$

so dass nur noch die freie Variable z vorkommt, und damit die Antwortsubstitutionen aus einer endlichen Grundmenge stammen.

Wir haben nun eine Darstellung des Planungsproblems gefunden, die die Frage nach der Lösbarkeit des Problems auf die Frage, ob die Menge der korrekten Antwortsubstitutionen für (8.4) nicht leer ist, reduziert, was sich äquivalent mit Hilfe von BDDs ausdrücken lässt. Es ist aber noch ein Weg zu finden, wie diese Menge auf geeignete Weise berechnet werden kann. Eine Antwort darauf findet man in der Struktur der Situationen: Jede Situation entspricht einer Aktionssequenz der Länge 0 oder der Länge 1 oder der Länge 2 usw. Die Menge $\mathcal{Z}$ der Antwortsubstitutionen σ für

$$\mathcal{F} \models [(\exists\, s : \mathit{State})\ \ z = \mathit{state}(s)]\sigma$$

lässt sich also in die Mengen von Antwortsubstitutionen $\mathcal{Z}_i$ für alle $i \in \mathbb{N}$ unterteilen, die Situationen s, die eine Aktionssequenz der Länge i kodieren, entsprechen. Diese Mengen kodieren gemäß der Semantik von *State* die Mengen der Zustände, die in i Schritten erreichbar sind. Tatsächlich ist es zur Entscheidung der Lösbarkeit ausreichend, maximal 2^n Schritte zu berücksichtigen, wenn n die Anzahl von Fluenten im Planungsproblem ist, da es maximal 2^n erreichbare Zustände im Problem gibt, und bei Ausführung des kürzesten Plans

[1] Wenn man im Kontext von deterministischen Planungsproblemen mit endlicher Menge von Fluenten nur nach den kürzesten Plänen zur Lösung des Planungsproblems sucht, ist die Menge der Aktionssequenzen, die zu betrachten sind, eigentlich endlich: Wie später diskutiert, kann der kürzeste Plan höchstens eine Länge von $2^n - 1$ haben, wenn das Planungsproblem n Fluenten hat. Die Menge der Aktionssequenzen ist hier aber trotzdem mit $(m^{2^n} - 1)/(m - 1)$ extrem groß.

[2] Eigentlich kann jeder Term – zusätzlich zu den Fluenten – $\emptyset$ beliebig oft enthalten; diese Terme sind jedoch äquivalent bezüglich $\mathcal{F}_{mset}$, so dass wir uns hier problemlos auf maximal einmaliges Vorkommen von $\emptyset$ beschränken können.

jeder Zustand höchstens einmal durchlaufen wird. (Wenn bei Ausführung eines Plans ein Zustand zweimal durchlaufen würde, dann könnte die Aktionssequenz, die zu dieser Schleife im Zustandsraum führt, aus dem Plan herausgenommen werden, so dass ein kürzerer Plan gefunden wird.) Es sind also im ungünstigsten Fall die 2^n Mengen $\mathcal{Z}_0$, $\mathcal{Z}_1$, $\mathcal{Z}_2$, ... $\mathcal{Z}_{2^n}$ zu berechnen, um die Existenz eines Planes zu entscheiden. Wie sich herausstellen wird, folgen diese Mengen einer Rekursionsgleichung, die sich zu einer Rekursion, die eine BDD-basierte Darstellung dieser Mengen berechnet, transformieren lässt, so dass sich das Planungsproblem BDD-basiert lösen lässt. (Die Extraktion eines Planes aus den Mengen wird in Abschnitt 8.3 diskutiert.)

Im nächsten Abschnitt wird nun dieser Vorgang im Detail formal dargestellt und dessen Korrektheit nachgewiesen. Die Übersetzung der Mengen $\mathcal{Z}_i$ geschieht in mehreren Schritten: (siehe Abbildung 8.2)

1. Für die Mengen $\mathcal{Z}_i$ werden Darstellungen als Mengen von Antwortsubstitutionen gleichungslogischer Formeln ζ_i gegeben. In diese gehen die Zustandsübergangsaxiome mit ein, so dass sämtliche Vorkommen des Funktionssymbols *state* sowie von Termen der Sorte *Sit* eliminiert sind. Diese Formeln lassen sich mit Hilfe einer Rekursionsgleichung berechnen.

2. Die Struktur der Formeln ζ_i gestattet die Konstruktion einer Abbildung $\mathfrak{B}()$, die diese in aussagenlogische Formeln $\mathtt{Z}_i$ übersetzt, die abermals die Antwortsubstitutionen für die ζ_i kodieren. Durch die Anwendung von $\mathfrak{B}()$ auf die Rekursionsgleichung für die Mengen $\mathcal{Z}_i$ kann eine Rekursionsgleichung zur Berechnung der ζ_i konstruiert werden.

3. Da aussagenlogische Formeln als BDD dargestellt werden können und die Rekursionsgleichung ebenfalls mit Hilfe von BDDs berechenbar ist, kann letztendlich eine BDD-Darstellung für die $\mathcal{Z}_i$, und somit $\mathcal{Z}$, generiert und damit die Lösbarkeit des Planungsproblem entschieden werden.

8.2. Abbildung des Fluentkalküls auf Aussagenlogik

Der Inhalt dieses Abschnitts ist ein schrittweiser, konstruktiver Beweis für das folgende Theorem 45 auf der nächsten Seite. Anstatt die dazu benötigten Konstruktionen vorher aus dem Zusammenhang gerissen einzuführen, werden diese im Laufe des Beweises an der Stelle, an der sie benötigt werden, eingeführt, und einige Lemmata darüber nachgewiesen. Der Beweis dieses Theorems bildet also den Rahmen für den gesamten Abschnitt.

Im nachfolgenden Abschnitt 8.3 wird dann, basierend auf den hier eingeführten Abbildungen, die Extraktion des Planes aus den generierten aussagenlogischen Formeln / BDDs behandelt.

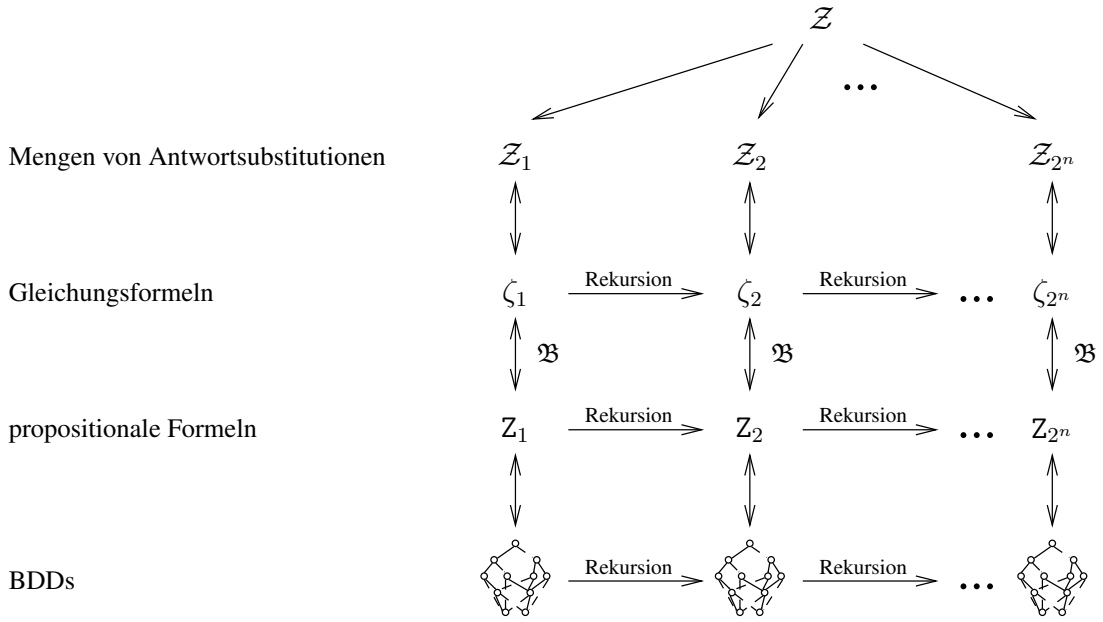


Abbildung 8.2.: Schritte zur Übersetzung eines Fluentkalkül–Planungsproblems in eine BDD-basierte Darstellung. Dabei ist n die Anzahl der Fluenten.

Theorem 45. *Ein Fluentkalkül-Planungsproblem*

$$\mathcal{F} \models (\exists\, s : \mathit{Sit}, z : \mathit{State})\ \ [z = \mathit{state}(s)\ \wedge\ \Phi_G(z)]\,, \tag{8.5}$$

entsprechend Annahme 43 (Seite 96) kann auf ein aussagenlogisches Erfüllbarkeitsproblem abgebildet werden, das genau dann erfüllbar ist, wenn (8.5) *erfüllbar (und damit das Planungsproblem lösbar) ist.*

Beweis

Der Beweis dieses Theorems folgt den im letzten Abschnitt diskutierten Schritten. Formen wir zunächst (8.5) um, so dass wir auf eine äquivalente – mit Hilfe von Mengen von Antwortsubstitutionen formulierte – Aussage kommen.

(8.5) ist erfüllt, wenn es eine Antwortsubstitution σ gibt, so dass

$$\mathcal{F} \models \big[(\exists s)\ z = \mathit{state}(s)\ \wedge\ \Phi_G(z)\big]\sigma. \tag{8.6}$$

Anders formuliert: (8.5) gilt genau dann, wenn die folgende Menge der Antwortsubstitutionen nicht leer ist:

$$\big\{\sigma \mid \mathcal{F} \models \big[(\exists\, s : \mathit{State})\ \ z = \mathit{state}(s)\ \wedge\ \Phi_G(z)\big]\sigma\big\} \neq \emptyset\ . \tag{8.7}$$

Da die Konjunktion auf Mengendurchschnitt abgebildet werden kann, ist dies äquivalent zu

$$\mathcal{Z} \cap \mathcal{G} \neq \emptyset \tag{8.8}$$

mit

$$\mathcal{Z} = \{\sigma \mid \mathcal{F} \models [(\exists s)\ z\sigma = \mathit{state}(s)]\sigma\} \tag{8.9}$$

und

$$\mathcal{G} = \{\sigma \mid \mathcal{F} \models [\Phi_G(z)]\sigma\}\ . \tag{8.10}$$

Für jede Substitution, die ein Folgerungsproblem wie (8.6) erfüllt, gibt es eigentlich unendlich viele weitere äquivalente Substitutionen, die dieses erfüllen: Nämlich alle Substitutionen $\{z/t'\}$, für die $\mathcal{F}_{mset} \models t = t'$ gilt, d.h. bei denen t und t' den gleichen Zustand darstellen. Für unser Beispiel 44 ist $\{z/L \circ R \circ O\}$ Bestandteil von $\mathcal{Z}$, aber ebenso $\{z/\emptyset \circ R \circ O \circ L\}$ und auch $\{z/O \circ \emptyset \circ \emptyset \circ R \circ \emptyset \circ L \circ \emptyset\}$. Bevor wir mit dem Beweis von Theorem 45 fortfahren, führen wir daher zunächst eine Beschränkung auf genau eine Substitution aus jeder Äquivalenzklasse ein. Wir nennen diese die *Grundzustandssubstitution*:

Definition 46. *Sei $\sqsubseteq$ eine beliebige, aber für das Planungsproblem fest vorgegebene lineare Ordnung über den Grundtermen der Sorte Fluent.*[3] *Ein **Grundzustandsterm** ist ein Term der Form $\emptyset \circ f_1 \circ f_2 \circ \ldots \circ f_n$, wobei die f_i Grundterme von Sorte Fluent sind und $f_i \sqsubseteq f_j$ für $i < j$. Ein Grundzustandsterm ist **aussagenlogisch**, wenn jeder Grundterm höchstens einmal darin vorkommt. Eine **Grundzustandsbindung** ist eine Bindung, die eine Variable der Sorte State an einen Grundzustandsterm bindet. Eine **Grundzustandssubstitution** ist eine Substitution, die nur aus Grundzustandsbindungen besteht.*

Die lineare Ordnung bestimmt eindeutig die Anordnung der Fluenten im Term, so dass wegen Theorem 30 auf Seite 55 zwei Grundzustandsterme genau dann bezüglich $\mathcal{F}_{mset}$ äquivalent sind, wenn sie gleich sind. Zudem gibt es für jede Substitution, die Variablen der Sorte *State* durch Grundterme ersetzt, eine bezüglich $\mathcal{F}_{mset}$ äquivalente Grundzustandssubstitution. Wir können uns also ohne Beschränkung der Allgemeinheit auf Antwortsubstitutionen, die Grundzustandssubstitutionen sind, beschränken, wenn in den Formeln nur freie Variablen der Sorte *State* vorkommen. Wegen (NonMult) können zudem nur aussagenlogische Grundzustandssubstitutionen in $\mathcal{Z}$ enthalten sein.

Zurück zum Beweis von Theorem 45. Sei n die Anzahl von Grundtermen der Sorte *Fluent*. Nach Annahme 43 und Axiom (NonMult) gibt es höchstens 2^n unterschiedliche erreichbare Zustände. Da die Vorbedingungen und Effekte der Aktionen nur vom aktuellen Zustand abhängen, kann für jeden dieser Zustände die Länge der kürzesten Aktionssequenz, die ihn

[3] Welche Ordnung hierfür verwendet wird, ist für das Problem irrelevant. Zum Beispiel kann die in Abschnitt 9.2.1 auf Seite 120 behandelte Sortenordnung verwendet werden.

erreicht, höchstens $2^n - 1$ sein (so dass jeder Zustand einmal besucht wird). Damit können wir $\mathcal{Z}$ (Gleichung (8.9)) zerlegen:

$$\mathcal{Z} = \bigcup_{i=0\ldots 2^n} \mathcal{Z}_i \ , \tag{8.11}$$

wobei $\mathcal{Z}_i$ die Menge der unter Ausführen von i Aktionen erreichbaren Zustände charakterisiert:

$$\mathcal{Z}_i = \left\{\sigma \mid \mathcal{F} \models \left[(\exists\, a_1, \ldots, a_i : \mathit{Action})\ \ z = \mathit{state}(\mathit{do}([a_1, \ldots, a_i], \mathtt{S}_0))\right]\sigma\right\} \tag{8.12}$$

für $i = 0 \ldots 2^n$. $\mathcal{Z}_0$ lässt sich nun unmittelbar gemäß der Annahme 43 zu

$$\mathcal{Z}_0 = \{\{z/t\}\} \qquad \text{für } \Phi_I = \ \mathit{state}(\mathtt{S}_0) = t \tag{8.13}$$

bestimmen. (Eventuell muss t als Grundzustandsterm umgeschrieben werden, indem die darin vorhandenen Fluenten gemäß $\sqsubseteq$ sortiert werden. Der so entstandene Term ist nach Theorem 30 äquivalent zum Ursprungsterm.) Wie kann man aber $\mathcal{Z}_{i+1}$ aus $\mathcal{Z}_i$ bestimmen? Die Beziehung zwischen dem Zustand vor und nach Ausführung einer Aktion wird nun durch die Zustandsübergangsaxiome bestimmt. Schauen wir uns diese also näher an.

Man beachte, dass, nach Expansion des Makro *Holds* , in einem Zustandsübergangsaxiom die Vorbedingung $\Delta(s)$ tatsächlich von $\mathit{state}(s)$ abhängt, da dieser Term die einzigen Vorkommen von Termen der Sorte *Sit* enthält. Wir schreiben also $\Delta(\mathit{state}(s))$ anstatt von $\Delta(s)$. Sei nun $\Delta(z)$ die Formel Δ , in der jedes Vorkommen von $\mathit{state}(s)$ durch die Variable z ersetzt wurde.

Für jedes Zustandsübergangsaxiom $\phi(a)$ der Form

$$(\forall)\ [\Delta(\mathit{state}(s)) \rightarrow \mathit{state}(\mathit{do}(a, s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+]$$

definieren wir eine Relation

$$\mathtt{T}_{\phi(a)}(z, z') \stackrel{def}{=} [\Delta(z) \ \wedge \ z' \circ \vartheta^- = z \circ \vartheta^+] \ . \tag{8.14}$$

Der Zweck dieser Formel ist zu charakterisieren, wann das Zustandsübergangsaxiom $\phi(a)$ angibt, das durch Ausführung der Aktion a in Zustand z der Zustand z' entsteht: die Vorbedingung $\Delta(s)$ des Zustandsübergangsaxioms muss dabei erfüllt sein, damit das Zustandsübergangsaxiom anwendbar ist, und die Zustandsgleichung muss für z und z' erfüllt sein. Um alle Möglichkeiten für einen Zustandsübergang bei Ausführung einer Aktion zu erfassen, fassen wir dies für alle Zustandsübergangsaxiome aus $\mathcal{F}_{sua}$ zusammen:

$$\mathtt{T}(z, z') \stackrel{def}{=} \bigvee_{\phi(a) \in \mathcal{F}_{su}} \mathtt{T}_{\phi(a)}(z, z'). \tag{8.15}$$

Im Beispiel:

Beispiel (44 Fortsetzung)**.** *Für das Zustandsübergangsaxiom*

$$\phi(\mathit{ToggleL}) = \big[\mathit{Holds}(L,s) \;\rightarrow\; \mathit{state}(\mathit{do}(\mathit{ToggleL},s)) \circ L = \mathit{state}(s)\big]$$

ergibt sich[4]

$$\mathtt{T}_{\phi(\mathit{ToggleL})}(z,z') = \big[\mathit{Holds}(L,z) \;\wedge\; z' \circ L = z\big] \;.$$

Entsprechend ist

$$\begin{aligned}\mathtt{T}(z,z') = \quad & [\mathit{Holds}(L,z) \;\wedge\; z' \circ L = z\,] \quad \vee \quad [\neg\mathit{Holds}(L,z) \;\wedge\; z' = z \circ L\,] \\ \vee \quad & [\mathit{Holds}(R,z) \;\wedge\; z' \circ R = z\,] \quad \vee \quad [\neg\mathit{Holds}(R,z) \;\wedge\; z' = z \circ R\,] \\ \vee \quad & [\mathit{Holds}(L,z) \;\wedge\; \mathit{Holds}(R,z) \;\wedge\; \neg\mathit{Holds}(O,z) \;\wedge\; z' = z \circ O\,] \;.\end{aligned}$$

Bevor wir mit dem Beweis von Theorem 45 fortfahren, beweisen wir ein Lemma, das zeigt, dass sich mit Hilfe von $\mathtt{T}(z,z')$ die Zustände z' charakterisieren lassen, die von einem Zustand z aus als erreichbar nachgewiesen werden können.[5] Dies wird es uns ermöglichen, eine Rekursionsformel für die Bestimmung der $\mathcal{Z}_i$ aufzustellen.

Lemma 47. *Sei s eine Situation, a eine Aktion und seien t und t' zwei Grundzustandsterme so dass $\mathcal{F} \models \mathit{state}(s) = t$. Dann gilt $\mathcal{F} \models \mathit{state}(\mathit{do}(a,s)) = t'$ genau dann, wenn $\mathcal{F} \models \mathtt{T}_{\phi(a)}(t,t')$ für ein $\phi(a) \in \mathcal{F}_{sua}$.*

Beweis.
„⇒“: Nehmen wir an, dass

$$\mathcal{F} \models \mathit{state}(\mathit{do}(a,s)) = t' \tag{i}$$

für eine Situation s und eine Aktion a gelten, wobei t und t' zwei Grundzustandsterme sind. Dann finden wir ein Zustandsübergangsaxiom

$$\Delta(\mathit{state}(s)) \rightarrow \mathit{state}(\mathit{do}(a,s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+ \tag{ii}$$

aus $\mathcal{F}_{sua}$, das dies erklärt, d.h.

$$\mathcal{F} \models \Delta(\mathit{state}(s)) \qquad \text{und} \qquad \mathcal{F} \models \mathit{state}(\mathit{do}(a,s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+ \;. \tag{iii}$$

Um die Vorkommen von $\mathit{state}(s)$ bzw. $\mathit{state}(\mathit{do}(a,s))$ in (iii) durch t bzw. t' zu ersetzen, erhalten wir unter Nutzung der Voraussetzung (i)

$$\mathcal{F} \models \Delta(t) \;\wedge\; t' \circ \vartheta^- = t \circ \vartheta^+ \;, \tag{iv}$$

so dass nach Definition von $\mathtt{T}$ gilt: $\mathcal{F} \models \mathtt{T}_{\phi(a)}(t,t')$.

[4] Man beachte die unterschiedlichen Definitionen von $\mathit{Holds}(f,s)$ und $\mathit{Holds}(f,z)$ Seite 68.
[5] Dies entspricht sinngemäß der Zustandsübergangsrelation Mengen mit endlicher Zustandsmenge.

„$\Leftarrow$“: Seien t, t' zwei Grundzustandsterme mit

$$\mathcal{F} \models \mathit{state}(s) = t. \tag{v}$$

Nehmen wir nun an, dass $\mathcal{F} \models \mathtt{T}_{\phi(a)}(t, t')$ für ein Zustandsübergangsaxiom $\phi(a)$:

$$\Delta(\mathit{state}(s)) \rightarrow \mathit{state}(\mathit{do}(a, s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+ \tag{vi}$$

aus $\mathcal{F}_{sua}$ gilt, so dass entsprechend der Definition von $\mathtt{T}$

$$\mathcal{F} \models \Delta(t) \wedge t' \circ \vartheta^- = t \circ \vartheta^+ \ . \tag{vii}$$

Wegen (v) und (vii) ist die Vorbedingung von (vi) erfüllt, so dass

$$\mathcal{F} \models \mathit{state}(\mathit{do}(a, s)) \circ \vartheta^- = \mathit{state}(s) \circ \vartheta^+$$

und somit wegen (v) und (vii)

$$\mathcal{F} \models \mathit{state}(\mathit{do}(a, s)) \circ \vartheta^- = t' \circ \vartheta^- \ . \tag{viii}$$

Durch mehrfaches Anwenden von (Distrib) erhält man

$$\mathcal{F} \models \mathit{state}(\mathit{do}(a, s)) = t' \ . \tag{ix}$$

■

Zurück zu Theorem 45. Es ergibt sich nun ein Zusammenhang zwischen $\mathcal{Z}_i$ und $\mathcal{Z}_{i+1}$:

- $\{z/t'\}$ ist wegen (8.12) auf Seite 103 in Z_{i+1} genau dann, wenn
- es $a_1, \ldots, a_{i+1} \in \mathit{Action}$ gibt, so dass $\mathcal{F} \models \mathit{state}(\mathit{do}([a_1 \ldots a_{i+1}], \mathtt{S}_0)) = t'$;
  dies gilt genau dann, wenn
- es $a_1, \ldots, a_{i+1} \in \mathit{Action}$ und einen Grundzustandsterm t gibt, so dass $\mathcal{F} \models \mathit{state}(\mathit{do}([a_1 \ldots a_i], \mathtt{S}_0)) = t$ und $\mathcal{F} \models \mathtt{T}_{\phi(a_{i+1})}(t, t')$ für ein Zustandsübergangsaxiom $\phi(a_{i+1})$ für Aktion a_{i+1} gilt;
  dies gilt genau dann, wenn
- es $a_1, \ldots, a_{i+1} \in \mathit{Action}$ und einen Grundzustandsterm t gibt, so dass $\{z/t\} \in Z_i$ und $\mathcal{F} \models \mathtt{T}(t, t')$ (entsprechend (8.15) auf Seite 103).

Anders formuliert:

$$\mathcal{Z}_{i+1} = \{\sigma \mid \mathcal{F} \models \mathtt{T}(z\hat{\sigma}, z\sigma), \hat{\sigma} \in \mathcal{Z}_i\}, \quad i \geq 0. \tag{8.16}$$

(8.13) und (8.16) erlauben es uns nun, gleichungslogische Formeln ohne Bezugnahme auf die Sorte *Sit* zu konstruieren, die die $\mathcal{Z}_i$ charakterisieren:

$$\mathcal{Z}_i = \{\sigma \mid \mathcal{F} \models \zeta_i(z\sigma)\} \tag{8.17}$$

mit

$$\zeta_0(z) = z = t \text{ für } \Phi_I = \mathit{state}(\mathtt{S}_0) = t \ . \tag{8.18}$$

$$\zeta_{i+1}(z) = (\exists\, \hat{z})\ [\zeta_i(\hat{z}) \ \wedge\ \mathtt{T}(\hat{z}, z) \ \wedge\ \mathit{IsSet}(\hat{z})] \ . \tag{8.19}$$

IsSet ist ein aus technischen Gründen eingeführtes Makro (siehe Beweis von dem später folgenden Satz 49), das ausdrückt, dass ein Zustand aussagenlogisch ist (d.h. einer Menge, engl. *set*, entspricht):

$$\mathit{IsSet}(z) \underset{\text{Def}}{\equiv} \neg\,(\exists\, f : \mathit{Fluent}, z' : \mathit{State})\ z = f \circ f \circ z' \ . \tag{8.20}$$

Da in allen ζ_i wegen (NonMult) nur aussagenlogische Substitutionen enthalten sind, und daher entsprechend (8.17) $\mathcal{F} \models \zeta_i(\hat{z})$ nur für aussagenlogische $\hat{z}$ erfüllt ist, ist der Term $\mathit{IsSet}(\hat{z})$ redundant. Er hat also keinen Beitrag zur Semantik - seine Einführung dient nur der Vereinfachung der Beweisführung.

Der nächste Schritt in unserem Beweis von Theorem 45 (vergleiche Schritt 2 auf Seite 100) ist nun die aussagenlogische Kodierung der Fluentkalkül–Formeln ζ_i. Genauer gesagt geht es nach wie vor darum, die Menge von auf aussagenlogische Grundzustandssubstitutionen beschränkten Antwortsubstitutionen solcher Fluentkalkül–Formeln zu charakterisieren, wobei wir uns gemäß der Struktur der ζ_i auf eine Klasse von Formeln, die nur Variablen vom Typ *State* enthalten, beschränken können.

Im Folgenden wird eine Abbildung $\mathfrak{B}_S()$ von Grundzustandssubstitutionen auf Belegungen aussagenlogischer Variablen angegeben, und eine Abbildung $\mathfrak{B}()$ einer die ζ_i enthaltenten Klasse von Fluentkalkül–Formeln auf aussagenlogische Formeln konstruiert, so dass eine aussagenlogische Grundzustandssubstitution genau dann in der Menge der Antwortsubstitutionen $\{\sigma \mid \mathcal{F} \models \phi\sigma\}$ einer Fluentkalkül–Formel ϕ ist, wenn die Variablenbelegung $\mathfrak{B}_S(\sigma)$ ein Modell der aussagenlogischen Formel $\mathfrak{B}(\phi)$ ist.

$$\mathcal{F} \models \phi\sigma$$
gdw.
$$\mathfrak{B}_S(\sigma) \models \mathfrak{B}(\phi)$$

Bei der Kodierung der Substitutionen folgen wir der Idee, die in Kapitel 3 gegeben wurde. Entsprechend Theorem 30 wird eine aussagenlogische Grundzustandssubstitution $\sigma = \{z/t\}$ mit $t = \emptyset \circ f_1 \circ f_2 \circ \ldots \circ f_n$ eineindeutig durch die Menge $\{f_1, f_2, \ldots, f_n\}$ charakterisiert. Diese kann nun z.B. dargestellt werden, indem jedem Fluent eine aussagenlogische Variable zugeordnet wird, die in der Variablenbelegung $\mathfrak{B}_S(\{z/t\})$ wahr ist, wenn dieses Fluent in $\{f_1, f_2, \ldots, f_n\}$ bzw. in t vorkommt, und sonst falsch ist. Die Menge aller Grundzustandssubstitutionen $\{z/t\}$ entspricht dann genau der Produktmenge der Menge T_{Fluent} von

Grundtermen der Sorte *Fluent*, und kann entsprechend durch eine Menge solcher Variablenbelegungen dargestellt werden – oder durch eine aussagenlogische Formel, die für diese Variablenbelegungen wahr ist.

In Teilformeln der ζ_i kommen aber z.T. mehrere freie Variablen vor, so dass jede Bindung der Substitution separat kodiert werden muss: In Verallgemeinerung der obigen Idee werden zur Abbildung einer Substitution mit mehreren Bindungen aussagenlogische Variablen z_f für jede Kombination aus einer Variable z in der Domäne der Substitution und einem Fluent $f \in T_{Fluent}$ eingeführt, deren Wahrheitswert angibt, ob f in der Bindung für z vorkommt.

Am Beispiel der Substitution $\sigma = \{z/\emptyset \circ L, \hat{z}/\emptyset \circ L \circ R\}$ entsprechend dem Safe–Beispiel 44 stellt sich dies so dar, dass für jede der 2 Variablen z und $\hat{z}$ der Domäne jeweils 3 aussagenlogische Variablen z_L, z_R und z_O bzw. $\hat{z}_L$, $\hat{z}_R$ und $\hat{z}_O$ genutzt werden, und sich die Variablenbelegung $\mathfrak{B}_S(\sigma)$ wie folgt ergibt:

$$\begin{array}{rl} \sigma: & \{z/\emptyset \circ \quad L \qquad\qquad\qquad\qquad\qquad , \ \hat{z}/\emptyset \circ \quad L \circ \qquad R \qquad\qquad\quad \} \\ \mathfrak{B}_S(\sigma): & \qquad\qquad z_L \mapsto \mathbf{1} \quad z_R \mapsto \mathbf{0} \quad z_O \mapsto \mathbf{0} \qquad\qquad \hat{z}_L \mapsto \mathbf{1} \quad \hat{z}_R \mapsto \mathbf{1} \quad \hat{z}_O \mapsto \mathbf{0} \end{array}$$

Stellen wir dies nun formal dar:

Abbildung $\mathfrak{B}_S()$ von Grundzustandssubstitutionen:
Sei z/t eine Bindung einer Grundzustandssubstitution. Dann ist $\mathfrak{B}_S(z/t)$ eine aussagenlogische Variablenbelegung v der Variablen $\{z_f \mid f \in T_{Fluent}\}$, wobei $v(z_f) = \top$ genau dann, wenn f in t vorkommt.

Sei σ eine Grundzustandssubstitution. Dann ist

$$\mathfrak{B}_S(\sigma) = \bigcup_{z/t \in \sigma} \mathfrak{B}_S(z/t) \ . \tag{8.21}$$

Im Folgenden bezeichnen wir für eine Variable z von Sorte *State* den Vektor der aussagenlogischen Variablen $\{z_f \mid f \in T_{Fluent}\}$ geordnet durch $\sqsubseteq$ (vergleiche Definition 46) als $\vec{z}_{Fluent}$.

Die Abbildung $\mathfrak{B}()$ der Fluentkalkül-Formeln wird in mehreren Schritten definiert. Um die Gleichheit von Termen unter einer Substitution zunächst fluentweise fassen zu können, wird eine Hilfsabbildung $\mathfrak{B}_f()$, $f \in T_{Fluent}$ definiert, die Grundzustandstermen t eine aussagenlogische Formel $\mathfrak{B}_f(t)$ zuordnet, die unter einer Variablenbelegung $\mathfrak{B}_S(\sigma)$ dann wahr ist, wenn $t\sigma$ das Fluent f ungerade oft enthält. Unter der Beschränkung, dass jedes Fluent höchstens einmal in einem Term vorkommt, ist dies äquivalent zu einem Test, ob das Fluent vorkommt.

Abbildung $\mathfrak{B}_f()$ von *State*-Termen:

Für jedes $f \in T_{Fluent}$ sei[6]

$$\begin{aligned} \mathfrak{B}_f(\emptyset) &= \bot & \qquad \mathfrak{B}_f(z) &= z_f \\ \mathfrak{B}_f(f') &= \begin{cases} \mathbf{1} & \text{wenn } f = f' \\ \mathbf{0} & \text{wenn } f \neq f' \end{cases} & \qquad \mathfrak{B}_f(t_1 \circ t_2) &= \mathfrak{B}_f(t_1) \oplus \mathfrak{B}_f(t_2) \end{aligned} \tag{8.22}$$

Beweisen wir zunächst eine Hilfsaussage.

Lemma 48. *Sei t ein Term der Sorte State, dessen freie Variablen nur von der Sorte State sind, und σ eine Grundzustandssubstitution, für die $t\sigma$ grundinstanziiert ist. Für jedes $f \in T_{Fluent}$ gilt nun $\mathfrak{B}_S(\sigma) \models \mathfrak{B}_f(t)$ genau dann, wenn f ungerade oft in $t\sigma$ vorkommt.*

Beweis. Sei $f \in T_{Fluent}$. Der Beweis erfolgt via Induktion über die Struktur von t:

- t **ist** $\emptyset$**:** Wegen (8.22) ist $\mathfrak{B}_f(t) = \bot$; $\mathfrak{B}_S(\sigma) \models \mathfrak{B}_f(t)$ gilt also nicht. Da f nicht ungerade oft in $\emptyset\sigma = \emptyset$ vorkommt, ist die Behauptung erfüllt.
- t **ist ein Fluent** f'**:** Entsprechend (8.22) gilt $\mathfrak{B}_S(\sigma) \models \mathfrak{B}_f(t)$ genau dann, wenn $f = f'$. f ist aber auch genau dann ungerade oft in t enthalten, wenn $f = f'$.
- t **ist eine Variable** z**:** Da σ nach Voraussetzung alle freien Variablen grundinstanziiert, ist $z \in \mathrm{dom}(\sigma)$. Nach Definition belegt $\mathfrak{B}_S(\sigma)$ die Variable z_f nun mit $\mathbf{1}$, wenn f in $z\sigma$ vorkommt (also genau einmal und damit ungerade oft vorkommt, da es sich um eine Grundzustandssubstitution handelt). $\mathfrak{B}_S(\sigma)$ belegt z_f mit $\mathbf{0}$, wenn f in $z\sigma$ nicht vorkommt (also nicht ungerade oft vorkommt). Es folgt die Behauptung.
- t **ist ein Term** $t_1 \circ t_2$**:** f kommt genau dann ungerade oft in t vor, wenn es in genau einem von t_1 und t_2 ungerade oft vorkommt. Nach Induktionsvoraussetzung entspricht dies $\mathfrak{B}_S(\sigma) \models \mathfrak{B}_f(t_1) \oplus \mathfrak{B}_f(t_2)$. Wegen (8.22) folgt die Behauptung.

■

Im Folgenden werden nun aufeinander aufbauend die Abbildungen $\mathfrak{B}_G$, $\mathfrak{B}_Z$ und $\mathfrak{B}_I$ für mehrere Klassen von Fluentkalkül–Formeln definiert, die zusammen die Abbildung $\mathfrak{B}$ bilden. Später werden dann die Indizes weggelassen, wenn aus dem Kontext klar ist, um welche der Teilabbildungen es sich handelt.

Unter Verwendung der Abbildung $\mathfrak{B}_f()$ kann nun eine aussagenlogische Formel konstruiert werden, die eine Formel $z = t$ mit einem Grundzustandsterm t darstellt. Die Idee ist hier in

[6] $\phi \oplus \psi$ stellt das exklusive Oder $\phi \oplus \psi \underset{\text{Def}}{\equiv} \neg(\phi \leftrightarrow \psi)$ dar.

die Formel $\mathfrak{B}_I(z = t)$ die Aussage zu kodieren, dass alle Fluenten $f \in T_{Fluent}$ genau dann in z vorkommen, wenn sie auch in t vorkommen.

Abbildung $\mathfrak{B}_I()$ für den Anfangszustand:
Der Anfangszustand wird gemäß (8.18) durch eine Formel $z = t$ mit einem Grundzustandsterm t dargestellt:

$$\mathfrak{B}_I(z = t) = \bigwedge_{f \in T_{Fluent}} (z_f \leftrightarrow \mathfrak{B}_f(t)) \ . \tag{8.23}$$

Im Safe–Beispiel ist $\mathfrak{B}_I(z = \emptyset \circ L \circ O)$ die Formel $(z_L \leftrightarrow \mathbf{1}) \wedge (z_R \leftrightarrow \mathbf{0}) \wedge (z_O \leftrightarrow \mathbf{1})$, was äquivalent zu $z_L \wedge \neg z_R \wedge z_O$ ist.

Abbildung $\mathfrak{B}_G()$ für Zielformeln:
Entsprechend Annahme 43 ist die Formel $\Phi_G(z)$, die das Ziel eines Planungsproblems charakterisiert, eine boolesche Kombination von Formeln der Form $\mathit{Holds}(f, z)$. Gleiches gilt wegen der für diese Arbeit vorausgesetzten Form der Zustandsübergangsaxiome für die Formeln $\Delta(z)$, die bei der Definition (8.14) von $\mathtt{T}$ entstehen. Entsprechend der besprochenen Intuition der z_f entspricht der Aussage $\mathit{Holds}(f, z)$ einfach die aussagenlogische Formel z_f. Wir definieren also:

$$\begin{aligned} \mathfrak{B}_G(\mathit{Holds}(f, z)) &= z_f, & \mathfrak{B}_G(\neg G) &= \neg\, \mathfrak{B}_G(G), \\ & & \mathfrak{B}_G(G \wedge H) &= \mathfrak{B}_G(G) \wedge \mathfrak{B}_G(H) \end{aligned} \tag{8.24}$$

für alle boolesche Kombinationen B und G von Formeln der Form $\mathit{Holds}(f, z)$. (Die anderen binären logischen Operationen lassen sich durch $\neg$ und $\wedge$ darstellen und lassen sich daher analog abbilden.)

Für die Abbildungen von Formeln $\mathtt{T}$ (siehe (8.14) und (8.15) auf Seite 103) folgen wir abermals der Struktur der Formeln. Die in (8.14) vorkommende Formel $z' \circ \vartheta^- = z \circ \vartheta^+$ wird dabei analog zu $\mathfrak{B}_I()$ fluentweise behandelt.

Abbildung $\mathfrak{B}_T()$ für $\mathtt{T}$:
Seien $\mathtt{T}_{\phi(a)}(z, z')$ bzw. $\mathtt{T}(z, z')$ definiert wie in (8.14) und (8.15). Dann ist:

$$\begin{aligned} \mathfrak{B}_T(\mathtt{T}_{\phi(a)}(z, z')) &= \mathfrak{B}_G(\Delta(z)) \wedge \bigwedge_{f \in T_{Fluent}} \left(\mathfrak{B}_f(z' \circ \vartheta^-) \leftrightarrow \mathfrak{B}_f(z \circ \vartheta^+)\right) \ , \\ \mathfrak{B}_T(\mathtt{T}(z, z')) &= \bigvee_{\phi(a) \in \mathcal{F}_{sua}} \mathfrak{B}_T(\mathtt{T}_{\phi(a)}(z, z')) \end{aligned} \tag{8.25}$$

Im Safe–Beispiel ergibt sich $\mathtt{T}_\phi(\mathit{ToggleL})$ für das Zustandsübergangsaxiom

$$\phi(\mathit{ToggleL}) = \left[\mathit{Holds}(L, s) \rightarrow \mathit{state}(\mathit{do}(\mathit{ToggleL}, s)) \circ L = \mathit{state}(s)\right]$$

(siehe Seite 104) mit $\mathtt{T}_{\phi(\mathit{ToggleL})}(z, z') = \big[\mathit{Holds}(L, z) \ \wedge \ z' \circ L = z\big]$ zu

$$\mathfrak{B}_T(\mathtt{T}_{\phi(\mathit{ToggleL})}(z, z')) = \big[z_L \ \wedge \ (\neg\, z'_L \ \leftrightarrow \ z_L) \ \wedge \ (z'_R \ \leftrightarrow \ z_R) \ \wedge \ (z'_O \ \leftrightarrow \ z_O)\big] \ .$$

Abbildung $\mathfrak{B}_Z()$ für ζ–Formeln:
Damit sind nun die Grundlagen definiert, um Formeln wie die ζ_i (8.18) und (8.19) von Seite 106 umzusetzen. Wir nennen im Folgenden eine Formel eine **ζ–Formel**, wenn sie entweder die Struktur $z = t$ mit einem Grundzustandsterm t hat, oder die Form $(\exists\ z : \mathit{State})\ \ [F(z) \ \wedge \ \mathtt{T}(z, z') \ \wedge \ \mathit{IsSet}(z)]$ hat, wobei $F(z)$ eine ζ–Formel ist und $\mathtt{T}(z, z')$ wie definiert in (8.14) und (8.15). Für jede ζ–Formel sei nun:

$$\begin{aligned} \mathfrak{B}_Z(z = t) &= \bigwedge_{f \in T_{Fl}} \big(z_f \ \leftrightarrow \ \mathfrak{B}_f\big(t\big)\big) \\ \mathfrak{B}_Z((\exists\, z : \mathit{State})\ \ [F(z) \ \wedge \ \mathtt{T}(z, z') \ \wedge \ \mathit{IsSet}(z)]) & \\ = (\exists\, \vec{z}_{\mathit{Fluent}})\ \ [\mathfrak{B}_Z(F(z)) \ \wedge \ \mathfrak{B}_T(\mathtt{T}(z, z'))] \ , & \end{aligned} \tag{8.26}$$

für alle ζ–Formeln F, wobei

$$\begin{aligned} (\exists\, z_{f_1} z_{f_2} \ldots z_{f_n})\ F \ &= \ (\exists\, z_{f_1}) \ \ldots (\exists\, z_{f_m})\ F \text{ mit} \\ (\exists\, z_f)\ F \ &= \ F\{z_f/\top\} \ \vee \ F\{z_f/\bot\} \end{aligned}$$

die sogenannte aussagenlogische Quantifikation darstellt. Die Abkürzungen $F\{z_f/\top\}$ bzw. $F\{z_f/\bot\}$ stellen dabei die Formel dar, die entsteht, wenn man in einer aussagenlogischen Formel F alle Vorkommen von z_f durch $\top$ bzw. $\bot$ ersetzt wurden.

Nun können wir nachweisen, dass die Abbildung $\mathfrak{B}\big()$ tatsächlich eine Formel liefert, die die Antwortsubstitutionen der abgebildeten Formel charakterisiert.

Satz 49. *Sei F entweder $\Phi_I(z)$, $\Phi_G(z)$ oder eine ζ–Formel, und σ eine aussagenlogische Grundzustandssubstitution, deren Domäne genau die freien Variablen von F sind. Es gilt $\mathcal{F} \models F\sigma$ genau dann, wenn $\mathfrak{B}(\sigma) \models \mathfrak{B}(F)$.*

Beweis. Wir beweisen statt dessen die äquivalente Aussage, dass für jedes Modell $\mathcal{M}$ von $\mathcal{F}$ gilt:

$$\mathcal{M} \models F\sigma \ \text{ genau dann, wenn } \ \mathfrak{B}_S\big(\sigma\big) \models \mathfrak{B}\big(F\big) \ . \tag{i}$$

Entsprechend Theorem 30 können wir uns im Beweis auf Modelle beschränken, in denen alle Elemente von $\mathit{State}^{\mathcal{M}}$ unterschiedliche Multimengen mit genau einem Element sind, und *State* die Menge aller endlichen Multimengen der Elemente dieser Einer-Multimengen.[7]

Zum Beweis von (i) betrachten wir die verschiedenen Teilabbildungen $\mathfrak{B}_G()$, $\mathfrak{B}_I()$, $\mathfrak{B}_T()$ und $\mathfrak{B}_Z()$ von $\mathfrak{B}\big()$ separat.

[7] Wenn wir *State* als Multimengen interpretieren, sind wegen $\mathit{Fluent} \preceq \mathit{State}$, d.h. $\mathit{Fluent}^{\mathcal{M}} \subseteq \mathit{State}^{\mathcal{M}}$, die Elemente von $\mathit{Fluent}^{\mathcal{M}}$ ebenfalls Multimengen. Wir benutzen also Einer-Multimengen für die Elemente der Sorte *Fluent*. Vergleiche auch den Beweis von Satz 31 auf Seite 56.

Abbildung von Zielformeln ($\mathfrak{B}_G()$): Entsprechend (8.24) erfolgt eine Induktion über die Struktur der abgebildeten Formel F. Wir unterscheiden 3 Fälle:

$F = \textbf{\textit{Holds}}(f, z)$ **:** (i) gilt wegen (8.21).

$F = \neg G$ **:** Wegen der Induktionsannahme

$$\mathcal{M} \models G\sigma \text{ gdw. } \mathfrak{B}_S(\sigma) \models \mathfrak{B}(G)$$

ergibt sich (i).

$F = G \wedge H$ **:** Wegen der Induktionsannahmen

$$\mathcal{M} \models G\sigma \text{ gdw. } \mathfrak{B}_S(\sigma) \models \mathfrak{B}(G)$$

und

$$\mathcal{M} \models H\sigma \text{ gdw. } \mathfrak{B}_S(\sigma) \models \mathfrak{B}(H)$$

ergibt sich (i).

Anfangszustand ($\mathfrak{B}_I()$): Da jedes Fluent höchstens einmal in $z\sigma$ sowie t vorkommt, folgt aus (8.23) und (8.21), dass $\mathfrak{B}_S(\sigma) \models \mathfrak{B}(z = t)$ genau dann gilt, wenn jedes Fluent in $z\sigma$ gleich oft vorkommt wie in t; dies gilt genau dann, wenn $\mathcal{M} \models z\sigma = t$. Also folgt (i).

$\mathtt{T}$–Formeln ($\mathfrak{B}_T()$): Wir betrachten die 2 Fälle aus (8.25) auf Seite 109:

$\mathtt{T}_{\phi(a)}(z, z')$ **:** Entsprechend (8.14) ist

$$\mathtt{T}_{\phi(a)}(z, z') = [\Delta(z) \wedge z' \circ \vartheta^- = z \circ \vartheta^+]$$

für irgendein $\Delta(z)$, ϑ^- und ϑ^+. Wegen der Festlegungen über die Struktur der Zustandsübergangsaxiome (Abschnitt 6.6 auf Seite 64) entspricht $\Delta(z)$ der Syntax einer Zielformel. Wir haben also weiter oben bereits bewiesen, dass

$$\mathcal{M} \models \Delta(z)\sigma \text{ gdw. } \mathfrak{B}_S(\sigma) \models \mathfrak{B}(\Delta(z)). \tag{ii}$$

Es können nun 2 Fälle unterschieden werden:

$\mathcal{F} \not\models \Delta(z)\sigma$ **:** Wegen (ii) ergeben sich

$$\mathcal{F} \not\models \Delta(z) \wedge z' \circ \vartheta^- = z \circ \vartheta^+ \ ,$$

sowie

$$\mathfrak{B}_S(\sigma) \not\models \mathfrak{B}_G(\Delta(z)) \wedge \bigwedge_{f \in \Sigma_{Fl}} \left(\mathfrak{B}_f(z' \circ \vartheta^-) \leftrightarrow \mathfrak{B}_f(z \circ \vartheta^+)\right)$$

für diesen Fall, und gemäß (8.25) folgt (i).

$\mathcal{F} \models \Delta(z)\sigma$ **:** Wegen (8.21), (8.22) und Lemma 48 ist leicht zu sehen, dass für jeden Grundzustandsterm t und jede Grundzustandssubstitution σ mit grundinstanziiertem $t\sigma$ die Formel $\mathfrak{B}_S(\sigma) \models \mathfrak{B}_f(t)$ genau dann gilt, wenn f ungerade oft in $t\sigma$ enthalten ist. Daher schließen wir, dass

$$\mathfrak{B}_S(\sigma) \models \bigwedge_{f \in \Sigma_{Fl}} \big(\mathfrak{B}_f(z' \circ \vartheta^-) \leftrightarrow \mathfrak{B}_f(z \circ \vartheta^+)\big) \tag{iii}$$

genau dann gilt, wenn sich für jedes Fluent f die Anzahl von Vorkommen von f in $(z' \circ \vartheta^-)\sigma$ und $(z \circ \vartheta^+)\sigma$ um eine gerade Zahl unterscheiden. Wir weisen nun nach, dass es nicht möglich ist, dass sich die Anzahl von Vorkommen auf beiden Seiten um mehr als 1 unterscheidet, so dass (iii) zu

$$\mathcal{F} \models (z' \circ \vartheta^-)\sigma = (z \circ \vartheta^+)\sigma \ .$$

äquivalent ist, was die Behauptung (i) wegen (ii) und (8.25) beweist.
Sei $f \in \mathit{Fluent}$. Wir unterscheiden 2 Fälle:

$f \in \vartheta^+$ **oder** $f \in \vartheta^-$ **:** Da σ eine Grundzustandssubstitution ist, kann f höchstens einmal in z' vorkommen, und daher 0, 1 oder 2 mal in $(z' \circ \vartheta^-)\sigma$. Wegen unserer Fallannahme $\mathcal{F} \models \Delta(z)\sigma$ kommt es aber genau einmal in $(z \circ \vartheta^+)\sigma$ vor: Da wir annehmen, dass $\phi(a)$ ein Zustandsübergangsaxiom wie (6.7) ist, bedeutet dies, dass $z\sigma$ keines der Fluenten aus ϑ^+ enthält, und $z\sigma$ alle Fluenten aus ϑ^- enthält. Folglich ist die Differenz der Anzahlen von Vorkommen höchstens 1.

$f \notin \vartheta^+$ **und** $f \notin \vartheta^-$ **:** Da σ eine Grundzustandssubstitution ist, kommt f höchstens einmal in $z\sigma$ und $z'\sigma$ vor, und daher auch höchstens einmal in $(z \circ \vartheta^+)\sigma$ und $(z' \circ \vartheta^-)\sigma$, so dass die Differenz der Anzahlen von Vorkommen höchstens 1 ist.

Wie bereits diskutiert, ist damit der Induktionsschritt bewiesen.

$\mathtt{T}(z, z')$ **:** Aus der Induktionsannahme

$$\mathcal{M} \models \mathtt{T}_{\phi(a)}(z, z')\sigma \text{ gdw. } \mathfrak{B}_S(\sigma) \models \mathfrak{B}\big(\mathtt{T}_{\phi(a)}(z, z')\big)$$

für alle $\phi(a) \in \mathcal{F}_{sua}$ folgt sofort die Behauptung (i).

ζ **–Formeln (** $\mathfrak{B}_Z()$ **):** Es erfolgt eine Induktion entsprechend der rekursiven Definition (8.26).

$F = (z = t)$ **:** (Induktionsanfang) Dies entspricht dem Anfangszustand weiter oben im Beweis.

$F = (\exists\, z : \mathit{State})\ [G(z) \wedge \mathtt{T}(z, z') \wedge \mathit{IsSet}(z)]$ **:** Als Induktionsvoraussetzung nehmen wir an, dass

$$\mathcal{M} \models G(z)\sigma \text{ genau dann, wenn } \mathfrak{B}_S(\sigma) \models \mathfrak{B}\big(G(z)\big) \ .$$

Wir formen nun $\mathcal{M} \models F\sigma$ äquivalent um:

- $\mathcal{M} \models (\exists\, z : \mathit{State})\ [G(z)\ \wedge\ \mathtt{T}(z, z')\ \wedge\ \mathit{IsSet}(z)]\sigma$ gilt nach Annahme 43 auf Seite 96 (T_{Fluent} ist isomorph zu $\mathit{Fluent}^{\mathcal{M}}$) genau dann, wenn
- es einen Grundterm t gibt, so dass $\mathcal{M} \models [G(t)\ \wedge\ \mathtt{T}(t, z')\ \wedge\ \mathit{IsSet}(t)]\sigma$. Nach Definition von *IsSet* ist $\mathcal{M} \models \mathit{IsSet}(t)$ genau dann erfüllt, wenn t ein aussagenlogischer Grundterm ist; daher gilt unsere Aussage genau dann, wenn
- es einen aussagenlogischen Grundterm t gibt, so dass $\mathcal{M} \models [G(t)\ \wedge\ \mathtt{T}(t, z')]\sigma$, bzw. $\mathcal{M} \models [G(t)]\sigma$ und $\mathcal{M} \models [\mathtt{T}(t, z')]\sigma$. Dies gilt genau dann, wenn
- es eine aussagenlogische Grundsubstitution $\sigma \cup \{z/t\}$ gibt, so dass $\mathcal{M} \models [G(z)](\sigma \cup \{z/t\})$ und $\mathcal{M} \models [\mathtt{T}(z, z')](\sigma \cup \{z/t\})$. Nach Induktionsvoraussetzung sowie dem obigen Nachweis der Gültigkeit von Satz 49 für Formeln der Form $\mathtt{T}(z, z')$ gilt dies genau dann, wenn
- es eine aussagenlogische Grundsubstitution $\sigma \cup \{z/t\}$ gibt, so dass $\mathfrak{B}_S\big(\sigma \cup \{z/t\}\big) \models \mathfrak{B}\big(G(z)\big)$ und $\mathfrak{B}_S\big(\sigma \cup \{z/t\}\big) \models \mathfrak{B}\big(\mathtt{T}(z, z')\big)$, bzw.

  $$\mathfrak{B}_S\big(\sigma \cup \{z/t\}\big) \models \mathfrak{B}_F(G(z))\ \wedge\ \mathfrak{B}_T(\mathtt{T}(z, z'))\ .$$

  Nach der Definition (8.21) von $\mathfrak{B}_S\big()$ gilt dies genau dann, wenn
- es eine Variablenbelegung v für die Variablen $\{z_f \mid f \in T_{\mathit{Fluent}}\}$ gibt, so dass

  $$v \cup \mathfrak{B}_S\big(\sigma\big) \models \mathfrak{B}_F(G(z))\ \wedge\ \mathfrak{B}_T(\mathtt{T}(z, z'))\ .$$

  Dies wiederum entspricht gemäß (8.26)
- $\mathfrak{B}_S\big(\sigma\big) \models \mathfrak{B}\big((\exists\, (z_f)_{f \in T_{\mathit{Fluent}}})\ [\mathfrak{B}_Z(F(z))\ \wedge\ \mathfrak{B}_T(\mathtt{T}(z, z'))]\big)$.

Damit gilt also $\mathcal{M} \models F\sigma$ genau dann, wenn $\mathfrak{B}_S\big(\sigma\big) \models \mathfrak{B}\big(F\big)$.

■

Nun können wir den letzten Schritt im Beweis des Theorems 45 gehen. Aus Satz 49 ergibt sich nun unter Beachtung von (8.9), (8.11) auf Seite 102f. und (8.17) auf Seite 106 als Konsequenz, dass die Übersetzungen

$$\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}] = \mathfrak{B}\big(\zeta_i(z)\big), \quad i \geq 0 \tag{8.27}$$

zusammen die Antwortsubstitutionen charakterisieren, die die erreichbaren Zustände im Planungsproblem darstellen. (Wir bezeichnen eine aussagenlogische Formel F , die sich auf Variablen aus dem Vektor von aussagenlogischen Variablen $\vec{v}$ bezieht, als $F[\vec{v}]$.)

$$\begin{aligned} \mathcal{Z}_i &= \big\{\sigma \mid \mathcal{F} \models \big[(\exists\, a_1, \ldots, a_i : \mathit{Action})\ z = \mathit{state}(\mathit{do}([a_1, \ldots, a_i], \mathtt{S}_0))\big]\big\} \\ &= \big\{\sigma \mid \mathfrak{B}_S\big(\sigma\big) \models \mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]\big\}, \quad i \geq 0 \end{aligned} \tag{8.28}$$

Man beachte, dass die Definition (8.26) von $\mathfrak{B}_Z$ entsprechend (8.18) und (8.19) eine direkte Rekursionsgleichung für die Ermittlung der $\mathtt{Z}_i$ liefert:

$$\begin{aligned} \mathtt{Z}_0[\vec{z}_{\mathit{Fluent}}] &= \mathfrak{B}_Z(z = t) \text{ für } \Phi_I = \mathit{state}(\mathtt{S}_0) = t \ . \\ \mathtt{Z}_{i+1}[\vec{z'}_{\mathit{Fluent}}] &= (\exists\, \vec{z}_{\mathit{Fluent}}) \ [\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}] \ \wedge \ \mathfrak{B}_T(\mathtt{T}(z, z'))] \ . \end{aligned} \tag{8.29}$$

Der Übergang von $\mathtt{Z}_{i+1}[\vec{z'}_{\mathit{Fluent}}]$ zu $\mathtt{Z}_{i+1}[\vec{z}_{\mathit{Fluent}}]$ entspricht dabei einem Umbenennen der gestrichenen in die entsprechenden ungestrichenen Variablen.

Analog zu den $\mathcal{Z}_i$ lässt sich $\mathcal{G}$ ((8.10) auf Seite 102) nach Satz 49 durch eine aussagenlogische Formel darstellen:

$$\mathcal{G} \quad = \quad \{\sigma \mid \mathcal{F} \models [\Phi_I(z)]\sigma\} \quad = \quad \big\{\sigma \mid \mathfrak{B}_S\big(\sigma\big) \models \mathfrak{B}\big(\mathtt{G}[\vec{z}_{\mathit{Fluent}}]\big)\big\} \tag{8.30}$$

mit

$$\mathtt{G}[\vec{z}_{\mathit{Fluent}}] = \mathfrak{B}\big(\Phi_I(z)\big) \ . \tag{8.31}$$

Das Folgerungsproblem (8.5) auf Seite 101, dass das Fluentkalkül-Planungsproblem darstellt, ist nun wegen (8.8) sowie (8.11) genau dann lösbar, wenn $(\bigcup_{i=0\ldots 2^n} \mathcal{Z}_i) \cap \mathcal{G} \neq \emptyset$, und damit nach (8.28) und (8.30) genau dann, wenn

$$\left\{\sigma \mid \mathfrak{B}_S\big(\sigma\big) \models \big(\bigvee_{i=0\ldots 2^n} \mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]\big) \ \wedge \ \mathtt{G}[\vec{z}_{\mathit{Fluent}}]\right\} \neq \emptyset \ . \tag{8.32}$$

Dies gilt, da sich Mengendurchschnitt bzw. Mengenvereinigung auf Konjunktion bzw. Disjunktion abbilden lassen. Da sich die $\mathtt{Z}_i$ über eine Rekursionsgleichung (8.29) berechnen lassen, ist es aus praktischen Gründen für den Entscheidungsprozess eines Planungsproblems günstiger, (8.32) umzuformen, so dass die Frage nach der Erfüllbarkeit von (8.5) sich auf die Frage nach der Erfüllbarkeit von

$$\bigvee_{i=0\ldots 2^n} (\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}] \ \wedge \ \mathtt{G}[\vec{z}_{\mathit{Fluent}}]) \tag{8.33}$$

reduziert. (Man beachte dazu, dass $\mathfrak{B}_S\big()$ injektiv ist, d.h. einerseits entspricht jeder Antwortsubstitution für (8.5) eine Variablenbelegung, die (8.33) erfüllt, und andererseits entspricht jeder Variablenbelegung, die (8.33) erfüllt, eine Antwortsubstitution.) Die Form (8.33) hat den Vorteil, dass die Disjunktion bereits als wahr nachgewiesen werden kann, wenn $\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}] \ \wedge \ \mathtt{G}[\vec{z}_{\mathit{Fluent}}]$ für irgendein i wahr ist, so dass nicht unbedingt alle $\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]$ berechnet werden müssen. Zudem kann sich bei der Berechnung von (8.29) ergeben, dass ab einem bestimmten i die Folge der $\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]$ periodisch oder konstant wird. In solch einem Falle kann das Planungsproblem ebenfalls entschieden werden, ohne explizit alle 2^n Formeln $\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]$ zu berechnen.

Damit ist Theorem 45 bewiesen. ∎

Das folgende Korollar ist eine direkte Konsequenz von Theorem 45 und der Entscheidbarkeit von Aussagenlogik.

Korollar 50. *Das Folgerungsproblem* (8.5) *ist entscheidbar.*

Wie in Kapitel 3 besprochen, sind viele logische Operationen in BDD-Darstellung für die logischen Formeln berechenbar. Insbesondere sind die für die Ermittlung der BDD-Darstellung von (8.33) nötigen Operationen durch BDD-Algorithmen darstellbar, so dass wir aus dem Beweis von Theorem 45 bereits einen BDD basierten Mechanismus für die Entscheidung des Planungsproblems gewonnen haben.

8.3. Planextraktion

In der praktischen Anwendung ist es normalerweise nicht nur wichtig zu entscheiden, ob ein Plan, der das Problem löst, existiert, sondern man möchte, dass der Planungsalgorithmus in der Lage ist, solche Pläne zu konstruieren. Wie in diesem Abschnitt beschrieben wird, ist es möglich, die in Abschnitt 8.2 beschriebene Entscheidungsmethode für Planungsprobleme so zu erweitern, dass eine Aktionssequenz ausgegeben wird, die das Problem löst. Der im Folgenden beschriebene Algorithmus hat dabei die angenehme Eigenschaft, stets einen kürzesten Plan zu liefern.

Die Grundidee zur Planextraktion ist die folgende. In der Konstruktion der zum Planungsproblem erfüllbarkeitsäquivalenten aussagenlogischen Formel (8.33) werden die Formeln $\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]$ verwendet. Diese beschreiben Mengen $\mathcal{Z}_i$ (vergleiche (8.12) auf Seite 103) von Antwortsubstitutionen, die die Zustände, die vom Anfangszustand durch Ausführung von i Aktionen erreichbar sind, charakterisieren. Aus diesem „Nebenprodukt“ des Algorithmus lässt sich nun ein Plan konstruieren, indem man

1. die erste Menge $\mathcal{Z}_n$ bestimmt, in der eine Substitution $\{z/t_n\}$ enthalten ist, die einen Zustand t_n kodiert, der das Ziel des Planungsproblems erfüllt,

2. von dieser Substitution ausgehend rückwärts eine Sequenz $\{z/t_n\}, \{z/t_{n-1}\}, \ldots, \{z/t_0\}$ von Substitutionen bestimmt, so dass jeweils $\{z/t_i\} \in \mathcal{Z}_i$ gilt und t_i ein von t_{i-1} durch Ausführen einer Aktion erreichbarer Zustand ist, und

3. eine Sequenz $a_1, \ldots, a_n$ von Aktionen bestimmt, bei denen jeweils Aktion a_i Zustand t_i in t_{i+1} überführt.

Diese Idee ist im nachfolgend beschriebenen Algorithmus realisiert. Da unser Ziel eine Abbildung auf BDDs ist, arbeitet der Algorithmus nicht mit den Substitutionen $\{z/t_i\}$ und den Mengen von Substitutionen $\mathcal{Z}_i$, sondern mit den entsprechenden aussagenlogischen Variablenbelegungen $\mathfrak{B}\big(\{z/t_i\}\big)$ sowie den aussagenlogischen Formeln $\mathtt{Z}_i$ wie beschrieben im vorangegangenen Abschnitt.

Algorithmus 51. *Seien* $\mathtt{Z}_i[\vec{z}_{Fluent}]$ *für* $0 \leq i \leq 2^n$ *die Formeln generiert für die Planungsdomäne durch Gleichung* (8.29)*, sowie* $\mathtt{G}[\vec{z}_{Fluent}]$ *generiert durch Gleichung* (8.31)*. Wenn nun* (8.33) *unerfüllbar ist, dann ist das Planungsproblem unlösbar; sonst ergibt sich eine Lösung wie folgt:*

Sei n *die kleinste Zahl in* $0 \ldots 2^n$*, so dass*

$$\mathtt{Z}_n[\vec{z}_{Fluent}] \ \wedge \ \mathtt{G}[\vec{z}_{Fluent}] \tag{8.34}$$

ist. Berechne eine Sequenz von Grundzustandssubstitutionen $\{z/t_n\}, \{z/t_{n-1}\}, \ldots, \{z/t_0\}$*, so dass:*

$$\mathfrak{B}_S\big(\{z/t_n\}\big) \models \mathtt{Z}_n[\vec{z}_{Fluent}] \ \wedge \ \mathtt{G}[\vec{z}_{Fluent}] \tag{8.35}$$

$$\mathfrak{B}_S\big(\{z/t_{i-1}\}\big) \cup \mathfrak{B}_S\big(\{z'/t_i\}\big) \models \mathfrak{B}\big(\mathtt{T}(z,z')\big) \qquad \textit{für } 1 \leq i \leq n \ , \tag{8.36}$$

sowie eine Sequenz von Aktionen $a_1, \ldots, a_n$*, so dass es jeweils ein Zustandsübergangsaxiom* $\phi(a_i)$ *gibt mit:*

(8.37)

$$\mathfrak{B}_S\big(\{z/t_{i-1}\}\big) \cup \mathfrak{B}_S\big(z'/t_i\big) \models \mathfrak{B}\big(\mathtt{T}_{\phi(a_i)}(z,z')\big) \quad \textit{für } 1 \leq i \leq n. \tag{8.38}$$

Die Aktionssequenz $a_1, \ldots, a_n$ *ist das Resultat des Algorithmus.*

Man beachte, dass es ausreichend ist, die Formeln $\mathtt{Z}_i$ bis zu n auszurechnen, falls das Planungsproblem lösbar ist, oder bis die Sequenz der $\mathtt{Z}_i$ (bezüglich aussagenlogischer Äquivalenz) periodisch wird, da $\mathtt{Z}_i$ eindeutig von $\mathtt{Z}_{i-1}$ bestimmt wird. Die Berechnung der Substitutionen $\{z/t_i\}$ ist nach (8.21) äquivalent zu der Berechnung der entsprechenden aussagenlogischen Variablenbelegungen $\mathfrak{B}_S\big(\{z/t_i\}\big)$.

Theorem 52. *Algorithmus 51 ist korrekt und vollständig und liefert, sofern es einen Plan gibt, einen kürzesten Plan.*

Beweis. Wir beweisen zuerst die Korrektheit des Algorithmus.

Wie wir im Beweis von Theorem 45 gezeigt haben, ist (8.33) unerfüllbar, falls das Planungsproblem unlösbar ist. Damit liefert der Algorithmus in diesem Fall die korrekte Antwort.

Nehmen wir nun an, dass der Algorithmus eine Sequenz von Substitutionen $\{z/t_n\}, \{z/t_{n-1}\}, \ldots, \{z/t_0\}$ und eine Sequenz von Aktionen $a_0, a_1, \ldots, a_n$ liefert, so dass (8.35), (8.36) und (8.38) erfüllt sind.

Da wir einen vollständig spezifizierten Anfangszustand annehmen, enthält $\mathcal{Z}_0$ nur ein einzelnes Element, und daher gilt nach (8.12)

$$\mathcal{F}_{\mathtt{S}_0} \models \mathit{state}(\mathtt{S}_0) = t_0 \ . \tag{i}$$

Wegen (8.36) und Satz 49 für $i = 1, \ldots, n$

$$\mathcal{F} \models \mathtt{T}(t_{i-1}, t_i) \ ,$$

und wegen Lemma 47

$$\mathcal{F} \models t_{i-1} = \mathit{state}(\mathit{do}(a_i, t_i))$$

für $i = 1, \ldots, n$, und damit ergibt sich aus der Substitutivität der Gleichheit und (i)

$$\mathcal{F} \models t_n = \mathit{state}([a_1, \ldots, a_n]\mathtt{S}_0) \ . \tag{ii}$$

Aus (8.35) ergibt sich wegen Satz 49

$$\mathcal{F} \models \Phi_G(t_n) \ ,$$

und mit (ii) schließen wir, dass $s = \mathit{do}([a_1, \ldots, a_n], \mathtt{S}_0)$ in der Tat eine Lösung für unser Planungsproblem (8.5) bildet.

Als zweites weisen wir nach, dass der Algorithmus vollständig ist und einen kürzesten Plan liefert. Nehmen wir an, dass unser Planungsproblem lösbar ist und einen kürzesten Plan der Länge l hat. Wir werden nun beweisen, dass jeder Schritt des Algorithmus ausführbar ist, d.h. n, $\{z/t_n\}, \{z/t_{n-1}\}, \ldots, \{z/t_0\}$ und $a_1 \ldots a_n$ existieren, sowie dass $n = l$.

Wenn eine Aktionssequenz $a_1 \ldots a_n$ ein Plan ist, dann gibt es einen Grundzustandsterm t_n, so dass

$$\mathcal{F} \models \ t_n = \mathit{state}(\mathit{do}([a_1, \ldots, a_n], \mathtt{S}_0)) \ \wedge \ \Phi_G(t_n) \ .$$

Wegen Satz 49 sowie (8.28) und (8.31) ist dies äquivalent zu

$$\mathfrak{B}_S\big(\{z/t_n\}\big) \models \mathtt{Z}_n[\vec{z}_{\mathit{Fluent}}] \ \wedge \ \mathtt{G}[\vec{z}_{\mathit{Fluent}}] \ .$$

Da l die Länge des kürzesten Plans ist, ergibt sich also, dass (8.34) für $n = l$ nicht erfüllbar ist, für $n < l$ aber nicht. Daher findet der Algorithmus $n = l$.

Entsprechend ergibt sich, dass es einen Grundzustandsterm t_n gibt, so dass (8.35) erfüllt ist.

Als nächstes weisen wir per Induktion nach, dass der Algorithmus, beginnend mit $\{z/t_n\}$, eine Sequenz von Grundzustandssubstitutionen findet. Nehmen wir an, dass der Algorithmus bereits die Grundzustandssubstitutionen $\{z/t_n\}, \{z/t_{n-1}\}, \ldots, \{z/t_k\}$ für ein k mit $n \geq k > 1$ bestimmt hat. Wegen (8.28) gibt es eine Sequenz $\hat{a}_1, \ldots, \hat{a}_k$ von Aktionen, so dass

$$\mathcal{F} \models t_k = \mathit{state}(\mathit{do}([\hat{a}_1, \ldots, \hat{a}_k], \mathtt{S}_0)) \ ,$$

bzw.

$$\mathcal{F} \models t_k = \mathit{state}(\mathit{do}(ah_k, \mathit{do}([\hat{a}_1, \ldots, \hat{a}_{k-1}], \mathtt{S}_0))) \ . \tag{iii}$$

Sei nun t_{k-1} ein Grundzustandsterm, so dass

$$\mathcal{F} \models t_{k-1} = \mathit{state}(\mathit{do}([\hat{a}_1, \ldots, \hat{a}_{k-1}], \mathtt{S}_0)) \ .$$

Lemma 47 auf Seite 104 sagt uns nun, dass $\mathcal{F} \models \mathtt{T}(t_{k-1}, t_k)$ bzw. $\mathcal{F} \models [\mathtt{T}(z, z')]\{z/t_{k-1}, z'/t_k\}$. Wegen Satz 49 ist dies äquivalent zu (8.36) für $i = k$, d.h. es existiert eine Grundzustandssubstitution $\{z/t_{k-1}\}$, die (8.36) für $i = k$ erfüllt. Per Induktionsschluss ergibt sich also, dass der Algorithmus eine Sequenz $\{z/t_n\}, \{z/t_{n-1}\}, \ldots, \{z/t_0\}$ findet, die (8.35) und (8.36) erfüllt.

Wegen der Definition (8.15) (Seite 103) von $\mathtt{T}$ folgt aber sofort für jedes i mit $0 \leq i < n$ aus (8.36), dass es ein Zustandsübergangsaxiom $\phi(a_i)$ gibt, dass (8.38) erfüllt. Damit findet der Algorithmus ebenfalls eine Sequenz $a_1, \ldots, a_n$ von Aktionen, so dass (8.38) erfüllt ist. Damit ist der Algorithmus vollständig und liefert wegen $n = l$ einen kürzesten Plan. ∎

Mit anderen Worten, der Algorithmus weist stets entweder nach, dass es keinen Plan gibt, der das Planungsproblem löst, oder er liefert einen kürzesten Plan, der das Problem löst.

9. Die Implementation und Methoden zur Effizienzsteigerung

Wir beschreiben in diesem Kapitel die Implementation des beschriebenen Planungsalgorithmus und diskutieren Methoden zur Verbesserung der Effizienz.

9.1. Die Implementation

Um den beschriebenen Inferenzmechanismus zum Lösen von Planungsproblemen mit anderen existierenden Planungsalgorithmen vergleichen zu können, wurde dieser in einer prototypischen Implementation BDDPLAN realisiert. Als Eingabesprache dient PDDL gemäß der in Kapitel 7 entwickelten Fluentkalkül-Semantik.[1] Die Operationen zur Konstruktion der in Algorithmus 51 verwendeten aussagenlogischen Formeln durch die entsprechenden BDD-Operationen ersetzt, so dass aussagenlogische Formeln ausschließlich in BDD-Form repräsentiert werden. Die Implementation erfolgte in `C++` unter Verwendung der BDD Bibliothek CALBDD [71] unter Verwendung der GNU [38] Tools `g++`, `flex` und `bison` und ist unter [78] erhältlich.

9.2. Variablenordnung im BDD

Wie in Kapitel 3 erwähnt, hat die Anordnung der aussagenlogischen Variablen im BDD einen großen Einfluss auf die Größe von BDDs, und damit auf die Effizienz von Algorithmen, die BDDs verwenden. In unseren Kodierungen $\mathfrak{B}_S()$ und $\mathfrak{B}()$ werden für jedes Fluent f, das im Planungsproblem auftritt, aussagenlogische Variablen z_f, z'_f eingeführt, deren Anordnung festgelegt werden muss.

Eine grundlegende Richtlinie zur Minimierung der Größe von BDDs ist es, den „Informationstransport“ zu minimieren. Dies bedeutet, Variablen, die eng miteinander verknüpft sind,

[1] Da durch Algorithmus 42 auf Seite 89 unter ungünstigen Umständen eine größere Anzahl von Zustandsübergangsaxiomen pro Aktion entstehen (siehe das dem Algorithmus folgende Beispiel), die dann aber gemäß (8.15) auf Seite 103 wieder kombiniert werden, werden die Zustandsübergangsaxiome nicht explizit berechnet, sondern die Algorithmen miteinander kombiniert.

nahe beieinander anzuordnen. Ein direktes Resultat ist, dass die Variablen z_f und z'_f für alle Fluenten f zusammen zu gruppieren sind, da sie gemäß 8.25 auf Seite 109 in den Kodierungen $\mathfrak{B}\big(\mathtt{T}_{\phi(a)}(z, z')\big)$ aller Zustandsübergangsaxiome direkt miteinander verknüpft sind. Offen ist nun noch die Frage, wie diese zweielementigen Gruppen untereinander anzuordnen sind.

Dies kann nach allgemeinen Richtlinien geschehen, wie in der unten entwickelten Sortenordnung, oder anhand einer Analyse der konkreten Funktionen (in unserem Falle der Aktionen), die betrachtet werden: D.h. es wäre eine algorithmische Idee zu entwickeln, die die Variablen, die in den Vorbedingungen und Effekten der jeweiligen Aktion auftreten, nahe beieinander anzuordnen. Wir betrachten an dieser Stelle nur die erste Variante, die weniger aufwendig erscheint.

9.2.1. Die „Sortenordnung“

Diese Ordnung arbeitet ohne Betrachtung der vorliegenden Aktionen. In Betracht gezogen werden nur die Fluenten mit ihren Argumenten. Die Grundidee ist hierbei die Annahme, dass Fluenten mit gleichen Argumenten sich beeinflussen. Betrachten wir z.B. die sog. GRIPPER-Planungsdomäne [66]:

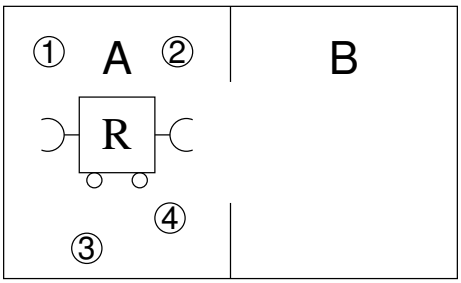


Abbildung 9.1.: Der Ausgangszustands im GRIPPER-Problem.

Beispiel 53.
Ein Roboter mit zwei Greifern `GripperA` *und* `GripperB` *(Sorte* Gripper*), in denen er jeweils einen Ball tragen kann, kann sich zwischen zwei Räumen* `RoomB` *und* `RoomB` *(Sorte* Room*) hin- und herbewegen (Abbildung 9.1). Wenn ein Greifer des Roboters frei ist, dann kann er mit diesem Greifer einen der Bälle* `Ball1,Ball2,...` *(Sorte* Ball*) greifen, die sich im gleichen Raum wie er befinden, oder falls er einen Ball in einem der Greifer trägt, kann er diesen in dem Raum ablegen.*

Es sind also folgende Fluenten und Aktionen zum Modellieren dieser Planungsdomäne nötig:

AtRobby :	*Room* → *Fluent*	*Position des Roboters*
At :	*Ball* × *Room* → *Fluent*	*Position der Bälle*
Carry :	*Ball* × *Gripper* → *Fluent*	*Ball im Greifer*
Free :	*Gripper* → *Fluent*	*Greifer frei*
Pickup :	*Ball* × *Room* × *Gripper* → *Action*	*Nehme Ball in Zimmer auf*
Drop :	*Ball* × *Room* × *Gripper* → *Action*	*Lege Ball in Zimmer ab*
Move :	*Room* × *Room* → *Fluent*	*Bewege Roboter von Raum zu Raum*

Die vollständige PDDL-Spezifikation des Problems ist in Anhang A.3 auf Seite 147 zu finden.

In der GRIPPER-Domäne zum Beispiel gibt es das Fluent $At(b, r)$, das für einen Ball b charakterisiert, ob er sich in Raum r befindet, und das Fluent $Carry(b, g)$, das charakterisiert, ob der Roboter Ball b in Greifer g transportiert. Wenn man nun zwei Instanzen $At(\texttt{Ball3}, \texttt{RoomB})$ und $Carry(\texttt{Ball3}, \texttt{GripperB})$ betrachtet, deren Argument `Ball3` identisch ist, so beeinflussen sich diese Fluenten über die Aktion $Pickup(\texttt{Ball3}, \texttt{RoomB}, \texttt{GripperB})$ gegenseitig. Die Instanzen $At(\texttt{Ball3}, \texttt{RoomB})$ und $Carry(\texttt{Ball7}, \texttt{GripperB})$ tun dies dagegen nicht, da sie sich auf verschiedene Bälle beziehen, ebenso wie $At(\texttt{Ball3}, \texttt{RoomB})$ und $At(\texttt{Ball1}, \texttt{RoomB})$. Es erscheint daher erfolgversprechend, die Fluenten nach ihren Argumenten anzuordnen.

Offen bleibt hierbei die Frage, nach welchem Argument man die Fluenten anordnet. Ein wichtiger Parameter ist hierbei die Anzahl der Objekte in der Sorte des entsprechenden Arguments. Wenn die Fluenten nach den Argumenten der Sorten mit geringerer Objektzahl sortiert werden, dann ergeben sich weniger Gruppen mit größerer Anzahl von Fluenten, wenn dagegen nach Argumenten der Sorten mit größerer Objektzahl sortiert wird, ergeben sich mehr Gruppen mit geringerer Anzahl von Fluenten, d.h. es wird mehr von der vorhandenen Information benutzt. Es ist also zu vermuten, dass nach Fluenten mit größerer Objektzahl sortiert werden sollte. Dies konnte experimentell bestätigt werden.

Formal funktioniert die Sortenordnung wie folgt: Jedem Fluent $f(o_1, o_2, \ldots, o_n)$ wird eine Sortierschlüssel

$$\kappa(f(o_1, o_2, \ldots, o_n)) = o_{p(1)}, o_{p(2)}, \ldots, o_{p(n)}, f$$

zugeordnet, wobei p eine Permutation von $1, \ldots, n$ ist, so dass

$$\left|Sort(o_{p(i)})\right| > \left|Sort(o_{p(i)})\right| \;\Rightarrow\; p(i) < p(j) \qquad \text{für alle } i, j \in [1, \ldots, n] \;.$$

Dem Fluent $Carry(\texttt{Ball3}, \texttt{GripperB})$ wird z.B. über die Permutation $2, 1$ der Sortierschlüssel $\texttt{Ball3}, \texttt{GripperB}, Carry$ zugeordnet, da die Sorte *Ball* mehr Elemente als die Sorte *Gripper* hat. Die Fluenten werden dann nach der lexikografischen Ordnung der Sortierschlüssel sortiert. Damit würde das Fluent $Carry(\texttt{Ball3}, \texttt{GripperB})$ vor dem Fluent

Variable für das Fluent	–	Sortierschlüssel
AtRobby(RoomB)	–	RoomB, *AtRobby*
AtRobby(RoomB)	–	RoomB, *AtRobby*
At(Ball1, RoomB)	–	Ball1, RoomB, *At*
At(Ball1, RoomB)	–	Ball1, RoomB, *At*
Carry(Ball1, GripperA)	–	Ball1, GripperA, *Carry*
Carry(Ball1, GripperB)	–	Ball1, GripperB, *Carry*
At(Ball2, RoomB)	–	Ball2, RoomB, *At*
At(Ball2, RoomB)	–	Ball2, RoomB, *At*
Carry(Ball2, GripperA)	–	Ball2, GripperA, *Carry*
Carry(Ball2, GripperB)	–	Ball2, GripperB, *Carry*
⋮	–	⋮
Free(GripperA)	–	GripperA, *Free*
Free(GripperB)	–	GripperB, *Free*

Tabelle 9.1.: Die Fluenten und ihr Sortierschlüssel im GRIPPER-Beispiel

Problem	GRIPPER (20 Balls)	BLOCKSWORLD (8 Blocks)	GET–PAID
lexikografische Ordnung	217409	206995	25633
Sortenordnung	3087	23373	38367

Tabelle 9.2.: BDD-Größen für die Zustandsübergangsrelation im Vergleich von Sortenordnung und lexikografischen Ordnung der Variablen. Die Probleme sind aus der Planungsdomänenbibliothek [66] entnommen.

At(Ball3, RoomB) (Symbolkette Ball3, RoomB, *At*) stehen. Für das GRIPPER-Beispiel erhält man damit eine Ordnung der Variablen wie in Tabelle 9.1.

Wie Tabelle 9.2 zeigt, hat die Sortenordnung für einige Beispiele einen recht dramatischen Effekt. Für andere tritt dagegen keine Verbesserung oder z.T. eine leichte Verschlechterung auf. Dies ist vor allem der Fall, wenn keine Sorten mit vielen Fluenten auftreten, so dass die Idee, die der Sortenordnung zugrunde liegt, nicht angemessen ist.

9.3. Disjunktive Partitionierung von T

Da die maximale Größe von BDDs exponentiell in der Anzahl der aussagenlogischen Variablen ist, kann die BDD-Darstellung von $\mathfrak{B}\big(\mathtt{T}(z, z')\big)$ viel größer als die BDDs, die die $\mathtt{Z}_i$

darstellen, werden, da sie doppelt so viele Variablen enthält. Eine Standardtechnik im Modelchecking zur Verringerung eines ähnlichen Problems ist nun die Partitionierung der Zustandsübergangsrelation. In unserem Beispiel spielt $\mathfrak{B}\big(\mathtt{T}(z,z')\big)$ die Rolle einer Zustandsübergangsrelation. Nach Gleichung (8.15) auf Seite 103 ist deren disjunktive Zerlegung leicht möglich.

Sei $\mathcal{F}_{su,1} \cup \mathcal{F}_{su,2} \cup \ldots \mathcal{F}_{su,k} = \mathcal{F}_{su}$ eine Partitionierung von $\mathcal{F}_{su}$ und sei

$$\mathtt{T}_j(z,z') \stackrel{def}{=} \bigvee_{\phi(a)\in\mathcal{F}_{su,i}} \mathtt{T}_{\phi(a)}(z,z') \quad \text{für } i = 1\ldots k.$$

Damit ist

$$\mathfrak{B}\big(\mathtt{T}(z,z')\big) \equiv \bigvee_{i=1\ldots k} \mathfrak{B}\big(\mathtt{T}_j(z,z')\big) \ .$$

Die zweite Gleichung in (8.29) auf Seite 114 wird also modifiziert zu

$$\begin{aligned}
&\mathtt{Z}_{i+1}[\vec{z'}_{\mathit{Fluent}}] \\
&\qquad = (\exists\, \vec{z}_{\mathit{Fluent}})\ [\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}] \ \wedge \bigvee_{i=1\ldots k} \mathfrak{B}\big(\mathtt{T}_j(z,z')\big)] \\
&\qquad = \bigvee_{i=1\ldots k} (\exists\, \vec{z}_{\mathit{Fluent}})\ [\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}] \ \wedge\ \mathfrak{B}\big(\mathtt{T}_j(z,z')\big)] \ . \qquad (9.1)
\end{aligned}$$

Die Umformung dieser Gleichung hat den Effekt, dass die aussagenlogische Quantifizierung (vgl. Seite 9) so zeitig wie möglich erfolgt. Dies verringert oft die BDD-Größe, da die Quantifizierung Variablen entfernt und damit die maximale BDD-Größe vermindert. Abbildung 9.2 veranschaulicht diese Methode.

In der Implementation BDDPLAN erfolgt die Partitionierung adaptiv: Zuerst werden die BDDs $\mathfrak{B}\big(\mathtt{T}_{\phi(a)}\big)$ für alle Aktionen konstruiert, und dann diese zu immer größeren Gruppen kombiniert, bis die Größe des BDDs, das die jeweilige Gruppe repräsentierent, einen Parameter „Partitionierungsschwelle“ überscheitet.

Die positive Wirkung der Partitionierung erklärt sich daraus, dass jede der Aktionen nur einen Teil der Fluenten beeinflusst, und daher, wenn eine Partition $\mathtt{T}_j(z,z')$ nur wenige Aktionen enthält, auch nur ein Teil der Fluenten beeinflusst wird. Die maximale Größe des BDDs für $\mathfrak{B}\big(\mathtt{T}_j(z,z')\big)$ ist nun exponentiell in der Anzahl der beeinflussten Variablen, so dass die Gesamtgröße aller BDDs für die Partitionen mit größerer Anzahl der Partitionen oft kleiner ist. Dies ist natürlich nur eine sehr grobe Schätzung. In praktischen Versuchen (Abbildung 9.3) ergab sich bei den meisten Problemen, in denen das BDD für $\mathfrak{B}\big(\mathtt{T}(z,z')\big)$ groß ist, eine z.T. starke Reduktion in der Gesamtgröße der BDDs durch die Partitionierung. Bei den Rechenzeiten zur Lösung der Planungsprobleme (Abbildung 9.4) hat die Partitionierung dagegen oft einen ungünstigen Effekt, da die Resultate der Verknüpfung der einzelnen Teile $\mathfrak{B}\big(\mathtt{T}_j(z,z')\big)$ mit $\mathtt{Z}_i$ gemäß (9.1) wieder zusammengesetzt werden müssen. Allerdings ist selbst in solchen Fällen eine Partitionierung oft nützlich, da die Speicherresourcen begrenzt sind.

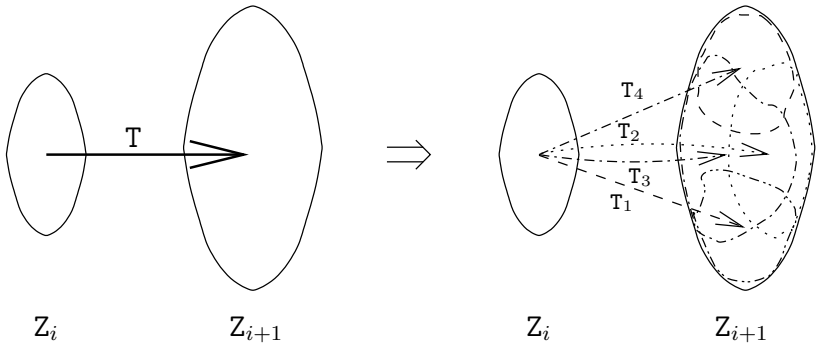


Abbildung 9.2.: Das Grundprinzip der disjunktiven Partitionierung von $\mathtt{T}(z, z')$: Die Relation $\mathtt{T}$, die $\mathtt{Z}_i$ und $\mathtt{Z}_{i+1}$ verknüpft, wird in mehrere Teilrelationen $\mathtt{T}_1, \ldots, \mathtt{T}_k$ zerlegt, deren Bilder separat berechnet werden. Die Resultate werden dann zusammengefügt.

9.4. Suchfrontreduktion

Weiterhin gibt es eine Technik genannt **search frontier reduction** (Suchfrontreduktion), die sich sinngemäß auf unseren Algorithmus übertragen lässt. Betrachten wir den Nachweis der Korrektheit und Vollständigkeit des Algorithmus 51 auf Seite 116 (Theorem 52). Dabei erkennt man, dass es für die Eigenschaft von Algorithmus 51, stets entweder einen kürzesten Plan zu liefern, oder nachzuweisen, dass das Planungsproblem unlösbar ist, ausreicht, dass die Mengen $\mathcal{Z}_i$ folgende Kriterien erfüllen:

(i).. Sie sollen alle korrekten Antwortsubstitutionen enthalten, die allen Zuständen entsprechen, die in i Schritten vom Ausgangszustand erreicht werden können, aber nicht in weniger Schritten. Zustände, die bereits vorher erreicht worden sind, können – brauchen aber nicht – berücksichtigt zu sein.

(ii).. Sie dürfen keine Antwortsubstitutionen enthalten die Zuständen entsprechen, die nicht in höchstens i Schritten vom Ausgangszustand erreicht werden können. Zustände, die bereits vorher erreicht worden sind können also durchaus hinzugenommen werden.

Die Mengen $\mathcal{Z}_i$ können also für die Funktionsfähigkeit des Algorithmus innerhalb dieser Grenzen frei gewählt werden. Da die Größe des BDDs, das eine Menge repräsentiert, von deren Struktur abhängt, liegt es nun nahe, jeweils Mengen zu nutzen, deren BDD-Repräsentation möglichst klein ist, um so den Speicherverbrauch des Algorithmus zu reduzieren. Zur Unterscheidung von der in Kapitel 8 diskutierten Grundform des Algorithmus 51 modifizieren wir den Algorithmus so, dass eine Folge von Mengen $\widehat{\mathcal{Z}}_i$ repräsentiert durch aussagenlogische Formeln $\widehat{\mathtt{Z}}_i$ verwendet wird, deren Berechnung im folgenden diskutiert wird. Seien $\mathcal{Z}_i$ ent-

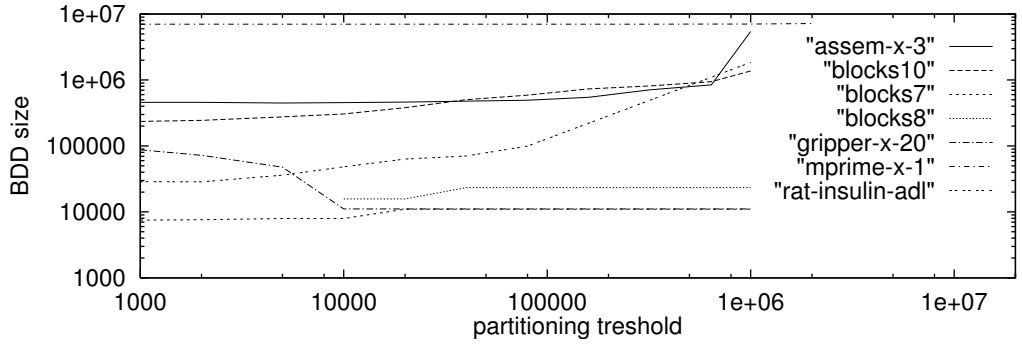


Abbildung 9.3.: Die Summe der Größen der BDDs, die die Zustandsübergangsrelation repräsentieren, in Abhängigkeit von der Partitionierungsschwelle – einem Parameter, der die Maximalgröße der BDDs für die einzelnen Partitionen der Zustandsübergangsrelation festlegt. Die Laufzeit wird relativ zur Laufzeit ohne Partitionierung dargestellt.

sprechend (8.12) auf Seite 103 in der Grundform des Algorithmus

$$\mathcal{Z}_i = \left\{\sigma \mid \mathcal{F} \models \left[(\exists\, a_1, \ldots, a_i : \mathit{Action})\ \ z = \mathit{state}(\mathit{do}([a_1, \ldots, a_i], \mathtt{S}_0))\right]\right\}\ .$$

Damit die Bedingungen (i). und (ii). erfüllt sind, muss also für alle $0 \leq i \leq n$ gelten:

$$\mathcal{Z}_i \setminus \bigcup_{l=0\ldots(i-1)} \mathcal{Z}_l \quad\subseteq\quad \widehat{\mathcal{Z}}_i \quad\subseteq\quad \mathcal{Z}_i \cup \bigcup_{l=0\ldots(i-1)} \mathcal{Z}_l\ . \tag{9.2}$$

Wie lässt sich diese Bedingung nun geeignet für den Algorithmus ausnutzen? Da die beschriebene Grundidee nur dann von Nutzen ist, wenn auf die Berechnung der $\mathcal{Z}_i$ zugunsten der $\widehat{\mathcal{Z}}_i$ mit kleinerer BDD-Darstellung verzichtet werden kann, benötigen wir eine Rekursionsgleichung für die $\widehat{\mathcal{Z}}_i$. Dafür gibt es mehrere Möglichkeiten, die (9.2) erfüllen, wie sich leicht per Induktion zeigen lässt. Sei

$$\widetilde{\mathcal{Z}}_{i+1} = \left\{\sigma \mid \sigma' \in \widehat{\mathcal{Z}}_i \text{ und } \mathcal{F} \models \mathtt{T}(z\sigma, z\sigma')\right\}$$

die Menge der koorrekten Antwortsubstitutionen entsprechend Zuständen, die von $\widehat{\mathcal{Z}}_i$ erreichbar sind.

Wir betrachten nun folgende Varianten von Rekursionsgleichungen:

$$\widetilde{\mathcal{Z}}_{i+1} \setminus \bigcup_{l=0\ldots i} \widehat{\mathcal{Z}}_l \quad\subseteq\quad \widehat{\mathcal{Z}}_{i+1} \quad\subseteq\quad \widetilde{\mathcal{Z}}_{i+1} \cup \bigcup_{l=0\ldots i} \widehat{\mathcal{Z}}_l \qquad \text{(reduceall')}$$

$$\widetilde{\mathcal{Z}}_{i+1} \setminus \widehat{\mathcal{Z}}_i \quad\subseteq\quad \widehat{\mathcal{Z}}_{i+1} \quad\subseteq\quad \widetilde{\mathcal{Z}}_{i+1} \cup \widehat{\mathcal{Z}}_i \qquad \text{(reduce')}$$

$$\widetilde{\mathcal{Z}}_{i+1} \setminus (\widehat{\mathcal{Z}}_i \cup \widehat{\mathcal{Z}}_{i-1}) \quad\subseteq\quad \widehat{\mathcal{Z}}_{i+1} \quad\subseteq\quad \widetilde{\mathcal{Z}}_{i+1} \cup (\widehat{\mathcal{Z}}_i \cup \widehat{\mathcal{Z}}_{i-1}) \qquad \text{(reduce2')}$$

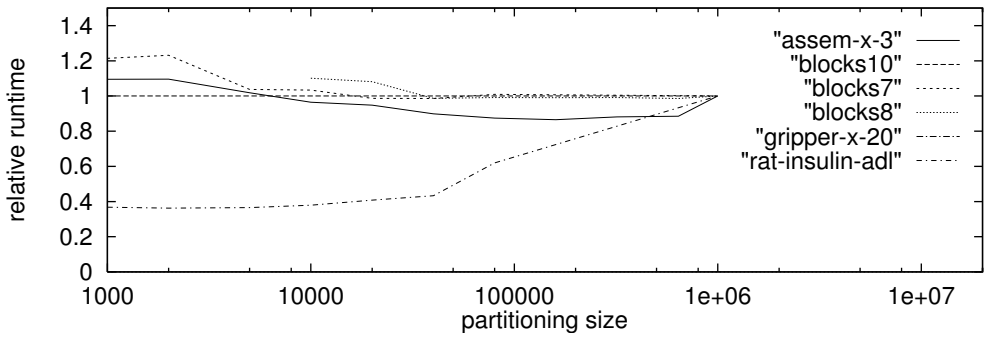


Abbildung 9.4.: Die Laufzeit des Planungsalgorithmus bei verschiedenen Problemen in Abhängigkeit von der Partitionierungsschwelle. Die Laufzeit wird relativ zur Laufzeit ohne Partitionierung dargestellt.

(reduceall') lässt den größtmöglichen Spielraum für die Wahl der Menge $\widehat{\mathcal{Z}}_{i+1}$ in jedem Schritt, und gestattet damit die potentiell größte Reduktion in der Größe des BDDs. Dafür ist aber auch die Berechnung am aufwändigsten. (reduce2') und (reduce') erlauben geringeren Spielraum und damit geringere Reduktionen, aber bilden einen Kompromiss im Berechnungsaufwand.

Nun fehlt noch die Umsetzung auf BDDs. Nehmen wir an wir haben zwei Mengen $\mathcal{A}$ und $\mathcal{B}$, die durch die charakteristischen Funktionen $\mathtt{A}$ und $\mathtt{B}$ repräsentiert werden, und suchen eine Menge $\mathcal{C}$, die durch eine charakteristische Funktion $\mathtt{C}$ repräsentiert wird, so dass $\mathcal{A} \setminus \mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{A} \cup \mathcal{B}$ gilt und die BDD-Darstellung von $\mathtt{C}$ möglichst klein ist. Die Bedingung $\mathcal{A} \setminus \mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{A} \cup \mathcal{B}$ stellt sich auf der Ebene der charakteristischen Funktionen wie folgt dar:

$$\models [(\mathtt{A} \wedge \neg \mathtt{B}) \rightarrow \mathtt{C}] \wedge [\mathtt{C} \rightarrow (\mathtt{A} \vee \mathtt{B})] \ .$$

Wenn man dies umformt,[2] erhält man:

$$\models \mathtt{C} \wedge \neg \mathtt{B} \leftrightarrow \mathtt{A} \wedge \neg \mathtt{B} \ .$$

Wenn man dies mit (3.6) auf Seite 27 vergleicht, erkennt man, das dies für

$$\mathtt{C} \equiv \textsc{Simplify}(\mathtt{A}, \neg \mathtt{B}) \tag{9.3}$$

erfüllt ist. (Vergleiche die Semantik von SIMPLIFY: Es wird eine Formel $\mathtt{C}$ mit möglichst wenig Knoten in der BDD-Darstellung bestimmt, die mit $\mathtt{A}$ übereinstimmt, wenn $\mathtt{B}$ nicht erfüllt ist, d.h. die alle Elemente aus $\mathcal{A} \cap \mathcal{B}$ enthält, und keine aus $\overline{\mathcal{B}} \setminus \mathcal{A}$. Die Formel kann dabei beliebig sein, wenn $\mathtt{B}$ erfüllt ist.)

[2] $[A\overline{B} \rightarrow C] \wedge [C \rightarrow (A \vee B)] \equiv \overline{[A\overline{B} \vee C]} \wedge [\overline{C} \vee A \vee B] \equiv [\overline{A} \vee B \vee C] \wedge [\overline{C} \vee A \vee B] \equiv \overline{AC} \vee B \vee CA$, sowie $[C\overline{B} \leftrightarrow A\overline{B}] \equiv B \vee [C \leftrightarrow A] \equiv B \vee \overline{CA} \vee CA$.

Dementsprechend können (reduceall'), (reduce') bzw. (reduce2') erfüllt werden, indem

$$\widehat{\mathtt{Z}}_{i+1} \equiv \textsc{Simplify}\left(\widetilde{\mathtt{Z}}_{i+1},\ \neg \bigvee_{l=0\ldots i} \widehat{\mathtt{Z}}_l\right) \qquad \text{(reduceall)}$$

$$\widehat{\mathtt{Z}}_{i+1} \equiv \textsc{Simplify}\left(\widetilde{\mathtt{Z}}_{i+1},\ \neg \widehat{\mathtt{Z}}_i\right) \qquad \text{(reduce)}$$

$$\widehat{\mathtt{Z}}_{i+1} \equiv \textsc{Simplify}\left(\widetilde{\mathtt{Z}}_{i+1},\ \neg \left(\widehat{\mathtt{Z}}_i \ \vee\ \widehat{\mathtt{Z}}_{i-1}\right)\right) \qquad \text{(reducetwo)}$$

genutzt wird. Der Algorithmus arbeitet also wie folgt: (8.29) auf Seite 114 wird durch

$$\begin{aligned} \widehat{\mathtt{Z}}_0[\vec{z}_{\mathit{Fluent}}] &= \mathfrak{B}_Z(z = t) \ \text{ für } \Phi_I = \ \mathit{state}(\mathtt{S}_0) = t \ , \\ \widetilde{\mathtt{Z}}_{i+1}[\vec{z'}_{\mathit{Fluent}}] &= (\exists\ \vec{z}_{\mathit{Fluent}})\ \left[\widehat{\mathtt{Z}}_i[\vec{z}_{\mathit{Fluent}}]\ \wedge\ \mathfrak{B}_T(\mathtt{T}(z, z'))\right] \ . \end{aligned} \tag{9.4}$$

und eine von (reduce), (reducetwo) und (reduceall) ersetzt. Algorithmus 51 auf Seite 116 arbeitet dann mit den $\widehat{\mathtt{Z}}_i$ anstatt mit den $\mathtt{Z}_i$. In Experimenten führte Suchfrontreduktion z.T. zu begrenzten Verbesserungen in Speicherverbrauch und Berechnungszeit (bis etwa 40%).

9.5. Vergleich mit anderen Planungsprogrammen

Um den Algorithmus experimentell mit anderen Planungsystemen vergleichen zu können, wurden die besprochenen Algorithmen zur Übersetzung der Semantik von PDDL aus Kapitel 7, sowie der Planungsalgorithmus aus Kapitel 8 mit den Optimierungen, die in diesem Kapitel besprochen wurden, implementiert. Die Implementation folgt weitgehend der Struktur der Konstruktionen, die in dieser Arbeit besprochen wurden. Beginnend mit einer PDDL- oder Fluentkalkülspezifikation des Planungsproblems, wird für jede Aktion die BDD-Repräsentation $\mathfrak{B}\left(\mathtt{T}_{\phi(a)}(z, z')\right)$ bestimmt, und deren Disjunktion gebildet ((8.15) auf Seite 103). Alternativ wird hier die in diesem Kapitel besprochene disjunktive Partitionierung eingesetzt, wobei die Aktionen nach Heuristiken, wie der diskutierten Sortenordnung, in die Partitionen sortiert werden. Die BDDs für die Formeln $\mathtt{Z}_i(z)$ werden iterativ durch Anwendung von (8.29) auf Seite 114 bzw. (9.1) auf Seite 123 berechnet, bis $\mathtt{Z}_i[\vec{z}_{\mathit{Fluent}}]\ \wedge\ \mathtt{G}[\vec{z}_{\mathit{Fluent}}]$ erfüllbar ist, oder die Folge der $\mathtt{Z}_i$ periodisch wird. [3] Entsprechend Algorithmus 51 auf Seite 116 kann dann ein Plan konstruiert werden, der das Planungsproblem löst. Dabei ist die in Kapitel 3 besprochene Operation ANYSAT nützlich, die für die im Algorithmus mehrfach auftretende Auswahl eines beliebigen Zustands aus einer (durch ein BDD repräsentierten) Menge von Zuständen eingesetzt werden kann.

[3] Durch Einführung einer zusätzlichen „No-Op"-Aktion, die nichts verändert, kann man erreichen, dass die Folge nur eine Periode 1 haben kann, d.h. nur konstant werden kann.

9.6. Einige experimentelle Resultate

Der implementierte Algorithmus BDDPLAN wurde auf viele Planungsprobleme der PDDL-Planungsproblembibliothek [66], sowie die auf den Planungswettkämpfen [65, 4, 61] angewendet. (BDDPLAN nahm am Planungsproblemwettkampf 2000 teil.) Im Folgenden sind die Resultate für einige Domänen dargestellt, für die BDDPLAN gut einsetzbar ist.

Ein Beispiel, das sich nur schwer mit Algorithmen wie Tiefensuche oder Breitensuche behandeln lässt, ist das sog. „Türme von Hanoi“-Problem. Dabei geht es um das bekannte Problem, n Scheiben, die geordnet nach fallender Größe auf einem Stab gesteckt sind, auf einen anderen Stab zu bringen. Jeder Zug besteht dabei aus dem Umstecken der obersten Scheibe eines der Stäbe auf einen anderen Stab, wobei eine Scheibe nicht über eine größere Scheibe gesteckt werden darf. Für n Scheiben hat der kürzeste Plan eine Länge von $2^n - 1$ und der Zustandsraum ist 3^n . Der PDDL-Quelltext ist in Abschnitt A.2 auf Seite 146 zu finden.

Für n Scheiben hat der kürzeste Plan eine Länge von $2^n - 1$, der Zustandsraum ist 3^n . Algorithmen, die den Planraum oder den Zustandsraum durchsuchen, gelangen daher mit wachsendem n schnell an ihre Grenzen. Die BDD-Darstellung der Mengen $\mathcal{Z}_i$ hat aber selbst bei $n = 15$ in unserer Implementierung weniger als 2000 BDD-Knoten. Daher lässt sich dieses Problem gut lösen. Die Laufzeiten sind in Abbildung 9.5 dargestellt.

Ein anderes Beispiel, entnommen aus dem AIPS-Planungsprogrammwettkampf [65], ist die Gripper-Domäne (Siehe Abbildung 9.6 und Abschnitt A.3 auf Seite 147). Wie Abbildung 9.7 zeigt, bereitete dieses Problem fast allen teilnehmenden Planungsprogrammen große Schwierigkeiten. Nur einer der Planer gelangte schnell zum Ziel. Dies geschah aber dadurch, dass er den zweiten Greifer ignorierte, und daher keine optimalen Pläne lieferte. Mit einer BDD-Darstellung ist jedoch ein vollständiges Durchsuchen des Zustandsraums möglich.

Abbildungen 9.8 und 9.9 zeigen die Resultate zweier weiterer Planungsdomänen aus dem AIPS-Planungsprogrammwettkampf 2000, in denen BDDPLAN gute Resultate lieferte, und seinen Vorteil, stets einen kürzesten Plan zu liefern, ausspielen konnte [4].

Bei anderen Planungsdomänen scheinen aber andere Planungsalgorithmen, wie z.B. eine heuristische Tiefensuche, angemessener, da bei zu vielen vorkommenden Fluenten (und entsprechend vielen aussagenlogischen Variablen) die BDDs zu groß werden, und die in dieser Arbeit präsentierte Vorgehensweise nur schlecht zum Ziel führt. Nach Meinung des Autors schöpfen aber die hier vorgestellten und implementierten Optimierungen das Potential von BDDs bei weitem nicht aus. Dies bleibt Gegenstand weiterer Forschung. Zum Beispiel realisiert [27] eine heuristische Suche im Zustandsraum mit einer BDD-Darstellung der auftretenden Zustandsmengen.

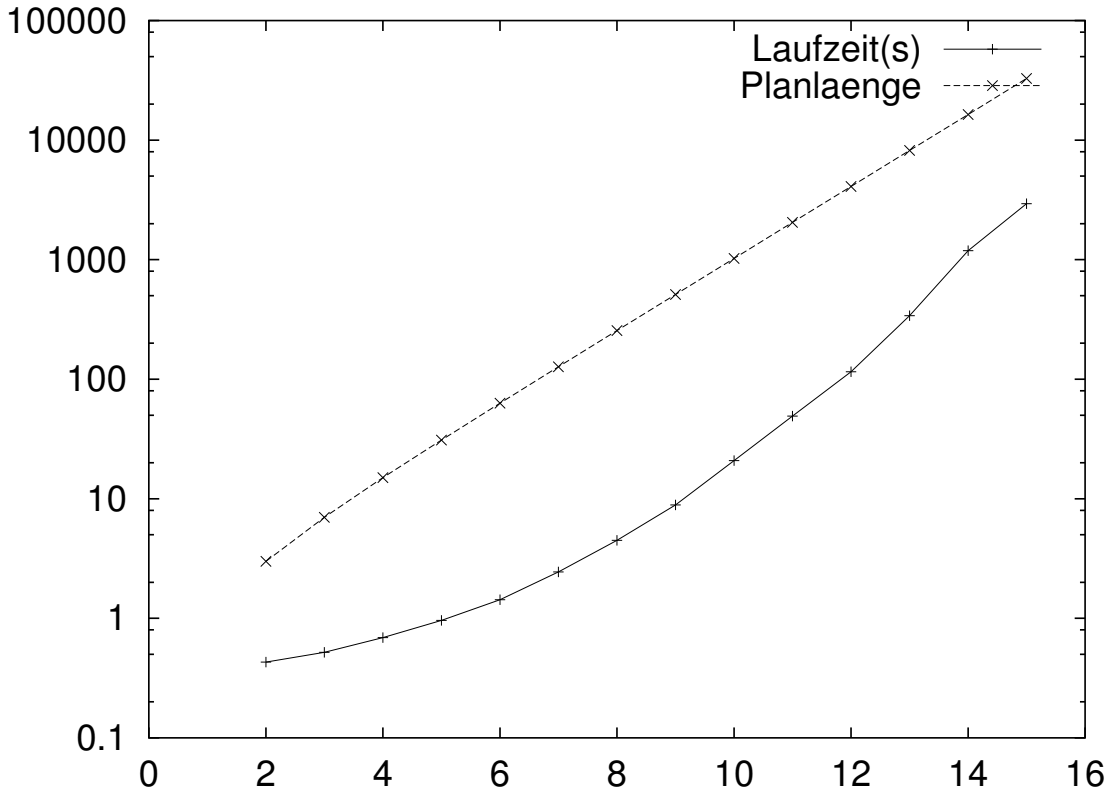


Abbildung 9.5.: Die Laufzeit und Planlängen für das bekannte „Türme von Hanoi“-Problem.

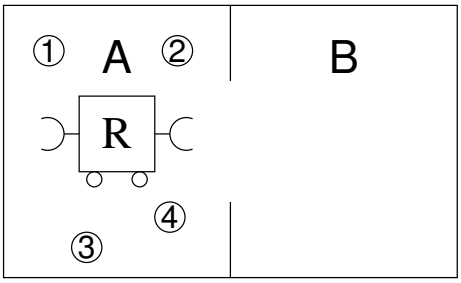


Abbildung 9.6.: Das Gripper-Problem: Ein Roboter mit 2 Greifarmen muss n Bälle von Raum A in Raum B schaffen. Zur Verfügung stehen die Aktionen Greifen/ Loslassen eines Balls im gleichen Raum mit Greifer L / R, sowie Wechsel des Raums.

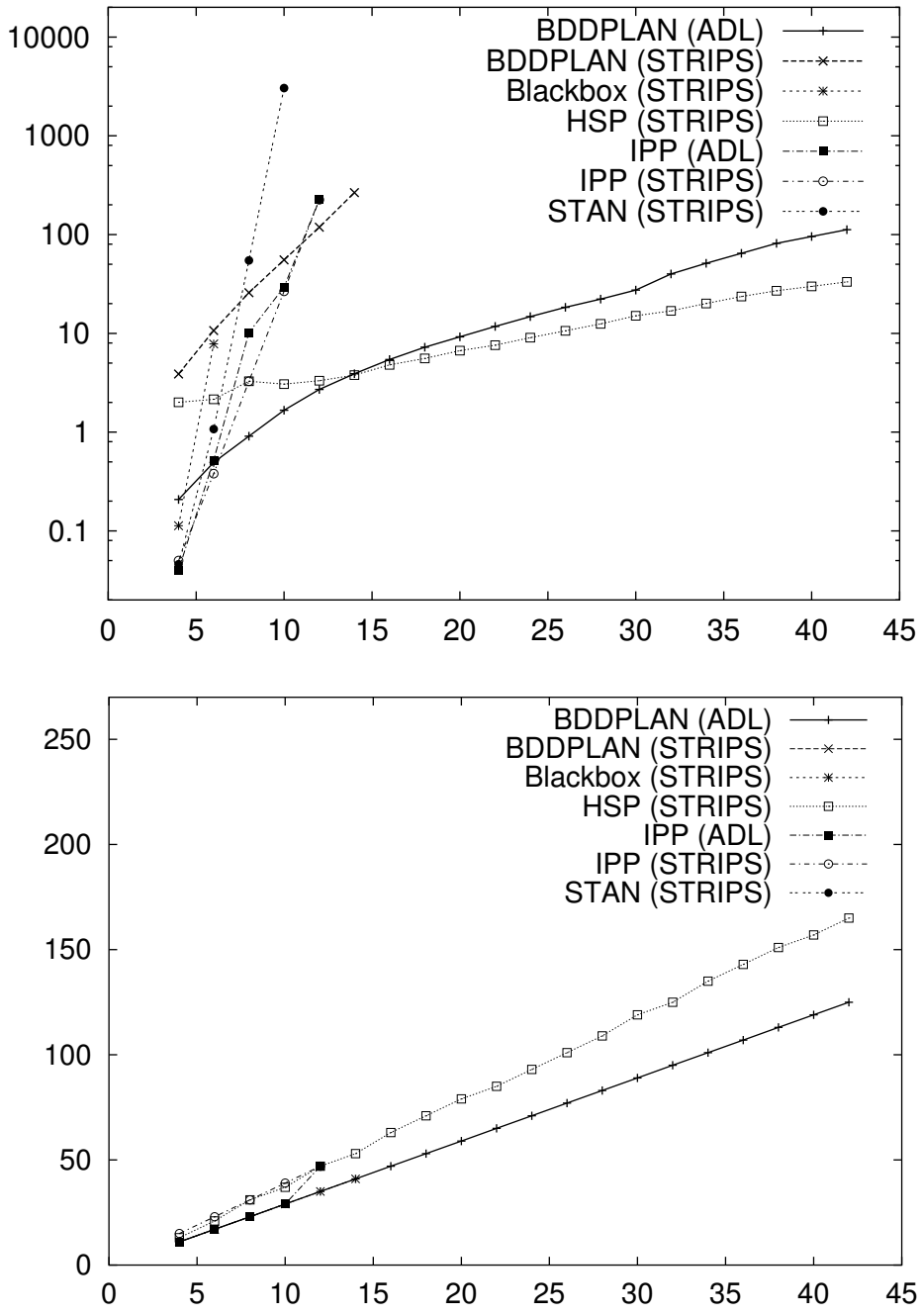


Abbildung 9.7.: Die Laufzeiten verschiedener Planungsprogramme auf den „Gripper“-Domänen aus dem AIPS98 Planungsprogrammwettkampf [65] über der Anzahl der Bälle (siehe Abschnitt A.3 auf Seite 147). Planer, die mit `(ADL)` markiert sind, arbeiten auf der im Anhang gegebenen Variante der Domäne, während die mit `(STRIPS)` markierten auf einer STRIPS-Version arbeiten, d.h. in einer PDDL-Version geringerer Ausdruckskraft.

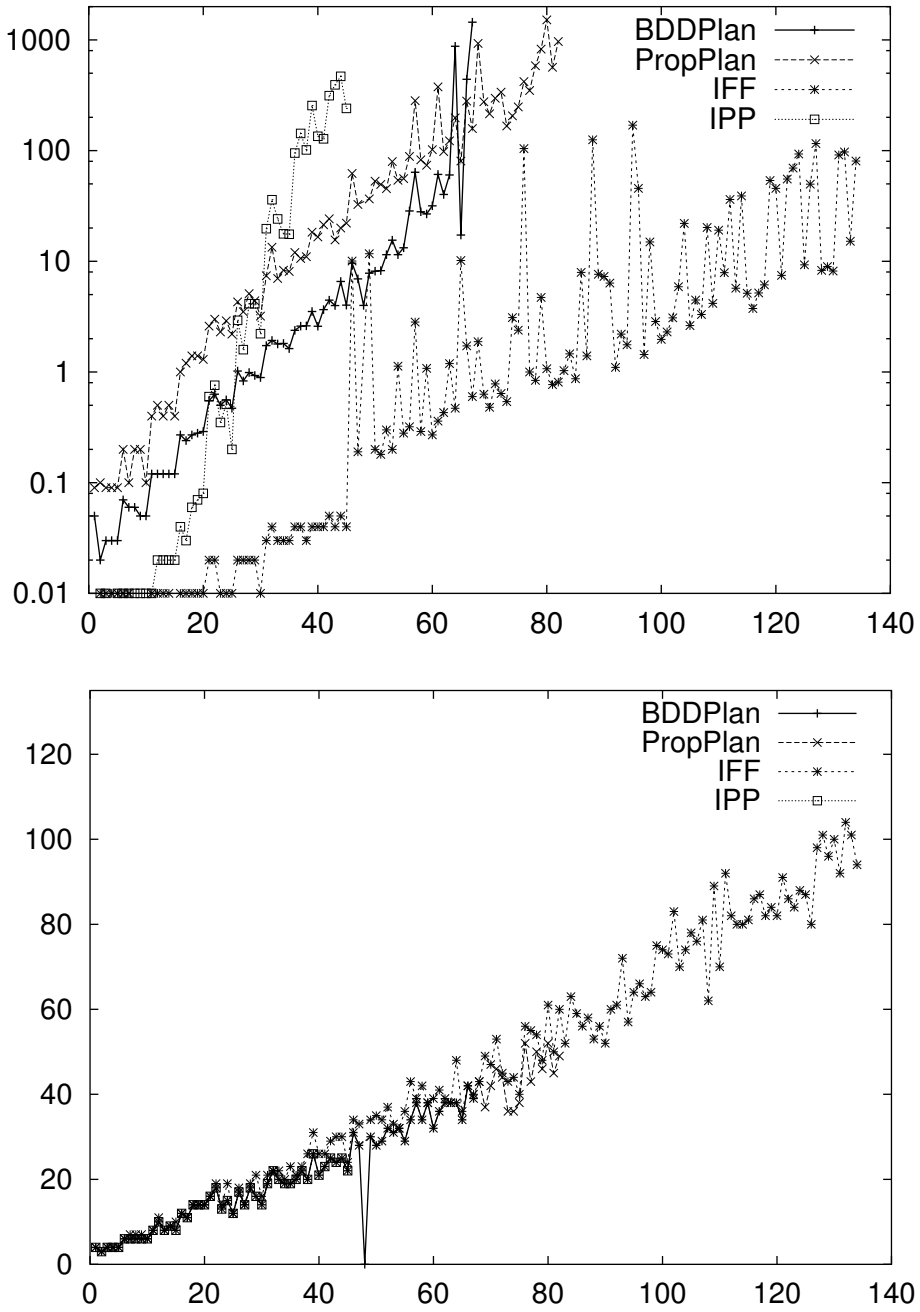


Abbildung 9.8.: Die Laufzeit in s (oben) und die Länge (unten) der generierten Pläne für die Probleme der ADL-Miconic Domäne im AIPS Planungsprogrammwettbewerb 2000. Die X-Achse zeigt die Nummer des Planungsproblems, die annähernd nach wachsender Komplexität geordnet sind. BDDPLAN liegt abermals im Mittelfeld. (Im Wettkampf schied BDDPLAN bei diesem Problem leider aufgrund eines kleinen Programmfehlers, der in einigen Fällen unkorrekte Pläne verursachte, aus. Die angegebenen Zeiten wurden nach Korrektur des Fehlers ermittelt.)

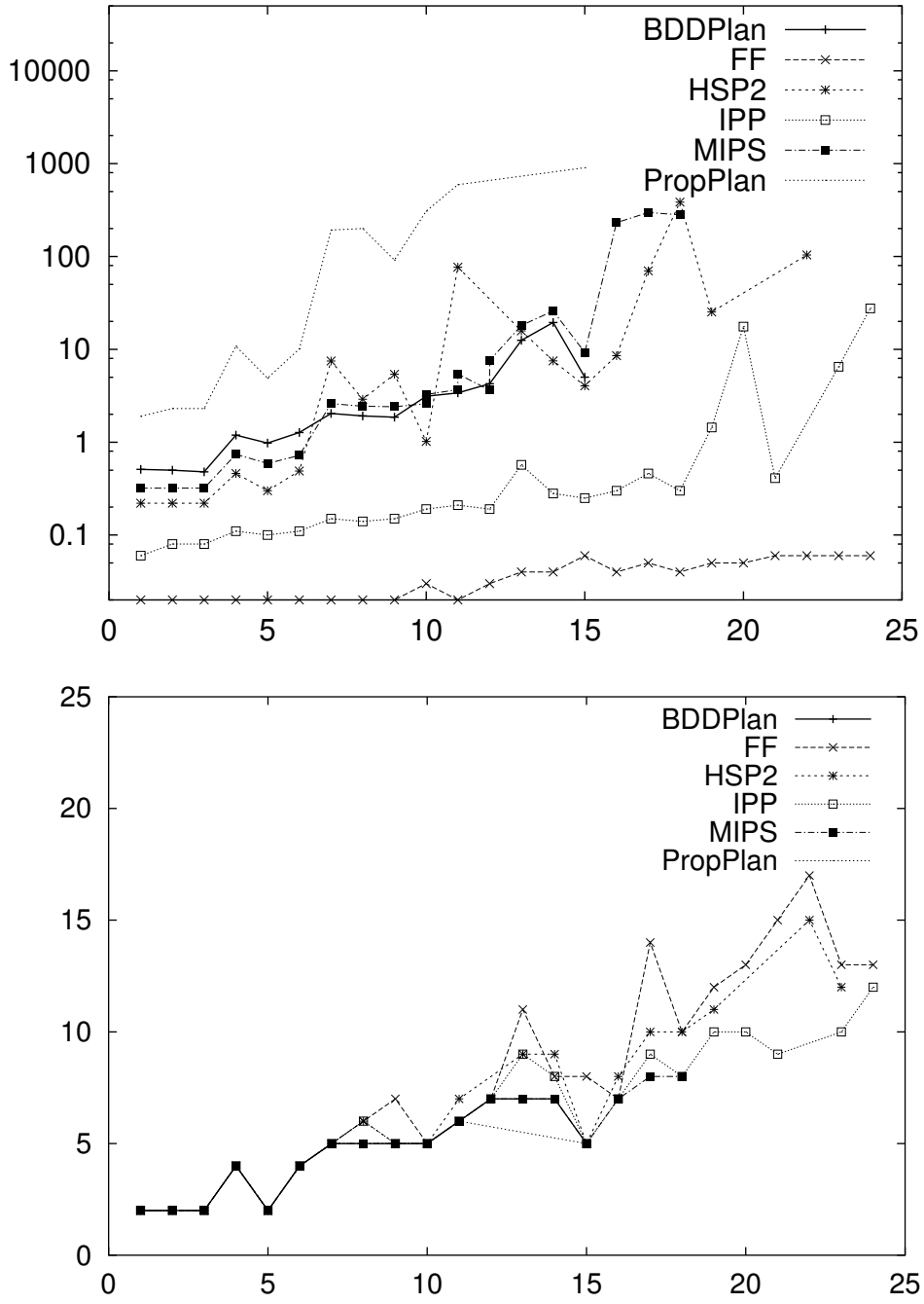


Abbildung 9.9.: Laufzeit in s (oben) und die Länge (unten) der generierten Pläne für die Probleme der ADL-Schedule Domäne im AIPS Planungsprogrammwettbewerb 2000 in Abhängigkeit von der Nummer des Problems. Für die Probleme, die BDDPLAN unter den gegebenen Zeit- / Speicherbeschränkungen lösen kann, liegt die benötigte Zeit im Mittelfeld, und die Pläne sind stets die kürzestmöglichen.

10. Zusammenfassung

In diesem Kapitel fassen wir unsere Resultate zusammen, vergleichen unsere Verfahrensweise mit anderen Techniken und identifizieren Berührungspunkte, offene Probleme und Möglichkeiten der zukünftigen Weiterentwicklung.

10.1. Ergebnisse

Im Folgenden geben wir eine Übersicht über die in dieser Arbeit erzielten Ergebnisse.

10.1.1. Eine Gleichungstheorie für den Fluentkalkül

In Kapitel 6 wird eine neue Gleichungstheorie konstruiert, die als Basis für den Fluentkalkül dienen kann. Diese bildet eine Alternative zu der (AC1)-unifikationsvollständigen Theorie (AC1*), definiert in [45], und der in Abschnitt 10.2.1 diskutierten verwandten Gleichungstheorie für FLUX in [90]. (AC1*) legt fest, dass unterschiedliche grundinstanziierte Terme unterschiedliche Fluenten denotieren. Dies behindert oft natürliche Formulierungen von Planungsproblemen, da z.B. funktionale Fluenten nur beschränkt verwendet werden können: die Aussage $\mathit{Color}(\mathtt{A}) = \mathtt{Rot}$, die Farbe (*Color*) des Blocks `A` ist rot (`Rot`), widerspricht (AC1*), da die Terme $\mathit{Color}(\mathtt{A})$ und `Rot` nicht unifizierbar sind. Der neu definierte Axiomensatz $\mathcal{F}_{mset}$ umgeht dieses Problem, indem nicht das syntaktische Kriterium der Unifizierbarkeit zur Basis der Gleichheit und Ungleichheit gemacht wird, sondern den Zuständen eines Planungsproblems eine axiomatische Spezifikation der Multimengensemantik zugrundegelegt wird: Ein Zustand wird als eine Multimenge von Fluenten spezifiziert. Die Gleichheit und Ungleichheit der Fluenten kann unabhängig davon beliebig spezifiziert werden.

In der Arbeit werden nachgewiesen, dass

- der Axiomensatz $\mathcal{F}_{mset}$ konsistent ist (Satz 31 auf Seite 56),
- für Modelle von $\mathcal{F}_{mset}$ die Sorte *State* der Zustände genau den Multimengen über der Sorte *Fluent* der Fluenten entsprechen (Theorem 30 auf Seite 55).

In Abschnitt 6.5 wurde eine Fluentkalkül-Signatur betrachtet, in der Fluenten entweder Konstanten oder Funktionen über einer Menge Objekten sind. Mit einer solchen Signatur lassen

sich viele praktisch bedeutsame Probleme beschreiben. Es wurde nachgewiesen, dass man durch Hinzunahme von Axiomen für die Namenseindeutigkeit und die Abgeschlossenheit des Universums für die Gleichungstheorie Unifikationsvollständigkeit analog zu (AC1*) erreichen kann (Satz 38 auf Seite 63). Damit ist es möglich, neben der später diskutierten auf Binären Entscheidungsdiagrammen basierenden Inferenzmethode auch die SLDENF-Resolution für mit Hilfe von $\mathcal{F}_{mset}$ entsprechend spezifizierte Planungsprobleme anzuwenden. Auszüge zu dieser Thematik wurden bereits in [79] publiziert.

10.1.2. Eine Semantik für PDDL

In Kapitel 7 wurde für das sogenannte ADL-Fragment der Planungsdomänenbeschreibungssprache PDDL eine Semantik im Fluentkalkül gegeben. Für diese existierte nur eine informale Semantikbeschreibung [37, 67]. Der Algorithmus zur Übersetzung von PDDL in den Fluentkalkül ermöglicht es, direkte Vergleiche mit anderen Planungsalgorithmen anzustellen, die ebenfalls in PDDL spezifizierte Probleme bearbeiten können [5]. So sind z.B. eine Menge von PDDL-Planungsproblemen zugänglich, die auf den Planungswettkämpfen auf den AIPS (Artificial Intelligence Planning and Scheduling Systems)-Konferenzen verwendet wurden, sowie eine Planungsdomänenbibliothek [66].

10.1.3. BDD-basiertes Planen im Fluentkalkül

Kapitel 8 legt eine Grundlage für das Lösen von Planungsproblemen im Fluentkalkül mit Hilfe von Binären Entscheidungsdiagrammen (BDDs). Wir beschränken uns auf deterministische Planungsprobleme mit endlicher Fluentenzahl und vollständig spezifizierten Ausgangszustand, die in einer Form gemäß Annahme 43 auf Seite 96 denotiert werden. Die diskutierte Übersetzung von PDDL in den Fluentkalkül genügt dieser Form. Nun wird eine Abbildung des Planungsproblems auf eine Formel der Aussagenlogik geschaffen, so dass die gebildete Formel genau dann erfüllbar ist, wenn das Planungsproblem lösbar ist (Theorem 45 auf Seite 101). Auf dieser Basis konstruiert Algorithmus 51 auf Seite 116 einen Plan für das Planungsproblem. Da dieser auf Grundlage von aussagenlogischen Formeln arbeitet, kann er mit Hilfe von BDDs implementiert werden.

Der hier beschriebene Algorithmus funktioniert ähnlich zu Modelchecking Algorithmen [15], indem er eine symbolische Breitensuche durchführt. Es wird eine Serie von als BDD dargestellten aussagenlogischen Formeln generiert, die jeweils eine Menge logischer Konsequenzen der Fluentkalkül-Beschreibung der Planungsdomäne darstellen. In der n-ten Formel sind jeweils die logischen Konsequenzen kodiert, die die mit n Aktionen erreichbaren Zustände darstellen. Der Planungsalgorithmus ist deterministisch: es wird kein Backtracking benötigt. Wie in Theorem 52 auf Seite 116 nachgewiesen, ist der Algorithmus korrekt und liefert stets einen kürzesten Plan oder weist nach, dass es keinen Plan für das gegebene Planungsproblem gibt. Im Gegensatz zu vielen anderen Planungsalgorithmen ist der Algorithmus nicht auf

Pläne polynomieller Länge beschränkt – so wurde beispielsweise das bekannte „Türme von Hanoi“-Problem für Planlängen bis zu 32767 Schritten gelöst. Auf der anderen Seite kann jeder Schritt bei ungünstiger Problemstruktur exponentiell viel Platz benötigen, da die maximale Größe der auftretenden BDDs für n Fluenten in der Größenordnung von $O(4^n)$ sein kann. Erste Resultate wurden bereits in [49] diskutiert.

10.1.4. Die Implementation und Methoden der Effizienzsteigerung

In Kapitel 9 werden die Implementierung BDDPLAN [78] beschrieben und angewandte Methoden zur Effizienzsteigerung diskutiert. Einen großen Einfluss auf die Größe der BDD-Darstellung einer aussagenlogischen Formel hat die dafür benutzte Ordnung der Variablen. Es wird die “Sortenordnung” entwickelt, die der Grundidee folgt, Variablen, die sich auf die gleichen Objekte im Planungsproblem beziehen, nahe beieinander anzuordnen. Im Experiment erzielt dies gute Resultate. Weiterhin werden zwei Optimierungstechniken für Modelchecking mit BDDs (die sogenannte disjunktive Partitionierung der Zustandsübergangsrelation und die Suchfrontreduktion) sinngemäß in den Algorithmus integriert und experimentell deren Effizienz geprüft. Zum Schluss des Kapitels wird BDDPLAN experimentell mit anderen Planungsprogrammen verglichen. Ein Überblick der damaligen Version von BDDPLAN wurde in [77] gegeben.

10.2. Berührungspunkte zu anderen Arbeiten

10.2.1. Fluentkalkül

Eine Grundidee des Fluentkalküls ist es, für Roboter oder andere künstliche Agenten, die sich in einer virtuellen oder realen Umgebung zurechtfinden sollen, eine logikbasierte Beschreibungssprache für diese Umgebung zur Verfügung zu stellen. Auf deren Basis kann der Agent das Verhalten der Umgebung vorhersehen, und mit Hilfe logischer Schlüsse die ihm vorgegebenen Ziele verwirklichen. Dementsprechend wurden und werden zahlreiche Erweiterungen des Fluentkalkül geschaffen, die diesen zu einer der umfangreichsten Beschreibungssprachen seiner Art macht. Um nur einige Erweiterungen zu der in Kapitel 6 beschriebenen Version des Fluentkalkül zu nennen:

- Für die Spezifikation von Aktionen ist es oft ungeschickt alle möglichen Effekte als direkten Effekt der Aktion zu spezifizieren (wenn man einen Tisch umwirft, fallen alle Objekte darauf herunter). Besser ist es aus dem direkten Effekt (der Tisch fällt um) automatisch *indirekte Effekte* abzuleiten (alle Objekte auf dem Tisch fallen herunter) und somit explizit kausale Beziehungen abzubilden [86]. Dies wird als *Ramifikation*

bezeichnet. Weiterhin werden in der genannten Arbeit *parallele Aktionen* behandelt (nur wenn man beide Enden eines Tisches gleichzeitig hochhebt, bleiben die Objekte drauf).

- In einer dem Agent nicht vollständig bekannten Umgebung ist er darauf angewiesen *Beobachtungen* anzustellen und entsprechend seinem erworbenen *Wissen* zu handeln. Das modellierte Wissen über den Zustand der Umgebung kann z.B. als zusätzliche Vorbedingung von Aktionen eingesetzt werden [87, 91] („Gehe nur dann vorwärts, wenn Du weißt, dass die Tür offen ist.“)
- Die Umgebung des Agenten kann *nichtdeterministisch* sein. Dementsprechend ist es nötig, Unsicherheiten in die Modellierung der Umgebung einzubeziehen [86].
- Besondere Beachtung verdienen Aktionen, deren Effekte nicht spontan eintreten, sondern über einen Zeitraum *stetige Veränderungen* bewirken (Ortsveränderung des Agenten bei der Fortbewegung), sowie die Ereignisse, die unabhängig vom Agenten eintreten (*„natürliche Ereignisse“*) [86, 62].
- Wenn ein Agent einen Plan ausführt müssen beobachtete Änderungen in der Umgebung oder neu zu realisierende Ziele (neue Aufträge) in Betracht gezogen werden. Dies geschieht durch *Execution Monitoring* (engl. für Beobachtung der Planausführung) und *Replanning* (Umplanen) bei der Planausführung eines Roboters [31].

Ein Einbeziehen einiger der genannten Erweiterungen des Umgebungsmodells in einen BDD-basierten Planungsprozess könnte z.B. eine wertvolle Unterstützung für die Steuerung des in der letztgenannten Arbeit beschriebenen Roboters sein.

Neben Arbeiten, die auf der hier entwickelten Fluentkalkül-Gleichungstheorie basieren (z.B. [49, 79, 77, 87]), wird neueren Arbeiten zum Fluentkalkül von der Intention der Interpretation der Sorte *State* der Zustände als Multimenge von Fluenten (Sorte *Fluent*) abgewichen, und statt dessen eine Interpretation als Menge von Fluenten angestrebt. Es wurde ein System FLUX (für engl. FLUent eXecutor) geschaffen, das als logikbasierte höhere Programmiersprache für Agenten dienen kann [91]. Weitere Arbeiten zu diesem Thema sind z.B. [89, 31, 62]. FLUX ist mit Hilfe von sog. Constraint Logic Programming realisiert; die Semantik basiert aber auf dem Fluentkalkül [90]. Hier wird nun eine Gleichungstheorie verwendet, für die sich nachweisen lässt, dass die Zustände aus *State* Mengen über *Fluent* entsprechen:

$$
\begin{array}{rll}
(\forall\, x, y, z : \mathit{State}) & (x \circ y) \circ z = x \circ (y \circ z) & \text{(Assoziativität)} \\
(\forall\, x, y : \mathit{State}) & x \circ y = y \circ x & \text{(Kommutativität)} \\
(\forall\, x : \mathit{State}) & x \circ x = x & \text{(Idempotenz)} \\
(\forall\, x : \mathit{State}) & x \circ \emptyset = x & \text{(Einselement)}
\end{array}
\tag{10.1}
$$

$$
\begin{array}{rll}
(\forall\, f : \mathit{Fluent}) & \neg\, \mathit{Holds}(f, \emptyset) & \text{(Leere Menge)} \\
(\forall\, f_1, f_2 : \mathit{Fluent}) & \Big[\mathit{Holds}(f_1, f_2) \;\rightarrow\; f_1 = f_2\Big] & \text{(Irreduzibilität)}
\end{array}
$$

$$(\forall\ f : \mathit{Fluent}, z_1, x_2 : \mathit{State})$$
$$\Big[\mathit{Holds}(f, z_1 \circ z_2)\ \rightarrow\ \mathit{Holds}(f, z_1)\ \vee\ \mathit{Holds}(f, z_2)\Big] \quad \text{(Dekomposition)}$$

$$(\forall\ z_1, x_2 : \mathit{State})$$
$$\Big[(\forall\ f : \mathit{Fluent})\ \big[\mathit{Holds}(f, z_1)\ \leftrightarrow\ \mathit{Holds}(f, z_2)\big]\ \rightarrow\ z_1 = z_2\Big] \quad \text{(Zustandsgleichheit)}$$

$$(\forall\ P : \langle \mathit{Fluent} \rangle)\ (\exists\ z : \mathit{State})\ (\forall\ f : \mathit{Fluent})\ \big[\mathit{Holds}(f, z)\ \leftrightarrow\ P(z)\big]$$
(Zustandsexistenz)

Da sich dadurch die Sorte *State* auf natürliche Weise mit Hilfe von BDDs darstellen lässt, könnte dies eine Erweiterung der in dieser Arbeit diskutierten Methode auf FLUX-basierte Arbeiten begünstigen.

10.2.2. Planen mit BDDs

In den letzten Jahren gab es eine rapide Entwicklung von BDD-basierten Techniken zum Lösen von Planungsproblemen. Im Folgenden diskutieren wir eine Reihe von Techniken, die Ergänzungen oder Alternativen zu den in dieser Arbeit verwendeten Methoden darstellen können.

Optimierung der Abbildung des Planungsproblems auf BDDs

In Kapitel 8 wurde für die Abbildung der Sorte *State* auf aussagenlogische Variablenbelegungen eine eins-zu-eins Abbildung der Fluenten auf die verwendeten aussagenlogischen Variablen benutzt. In vielen praktisch auftretenden Planungsproblemen treten aber nicht alle Kombinationen von Fluenten auf: Manche Fluenten sind unveränderlich, manche schließen sich gegenseitig aus usw. Es ist daher meist möglich, alle tatsächlich auftretenden Zustände mit weniger aussagenlogischen Variablen als Fluenten zu kodieren. Da die maximale Größe von BDDs von der Variablenzahl abhängt, bringt dies unter Umständen erhebliche Effizienzgewinne. Domänenanalysetechniken [60] können hier helfen, solche Möglichkeiten zu erkennen. Z.B. definiert [33] Algorithmen, die Gruppen von Fluenten identifizieren können, von denen stets nur ein Fluent wahr ist (wie z.B. $\mathit{At}(\texttt{Ball1}, \texttt{RoomB})$ und $\mathit{At}(\texttt{Ball1}, \texttt{RoomB})$ im Gripper-Problem Seite 120). Der Zustand einer Gruppe von n solchen Fluenten kann nun binär kodiert werden, so dass statt n Variablen nur $\lceil \log_2(n) \rceil$ Variablen benötigt werden. [68] diskutiert vor dem Hintergrund des Planungsprogramms GRAPHPLAN heuristische Methoden, die Relevanz von Aktionen und Fluenten für ein Planungsproblem zu entscheiden.

[56] behandelt eine Anzahl von weiteren Optimierungen bei der Übersetzung von PDDL Aktionsbeschreibungen in aussagenlogische Formeln, die im Planungsprogramm IPP genutzt

werden. Vor allem betrifft dies die Konsequenzen der sog. „inertia predicates“, d.h. Fluenten, die nicht in den Effekten auftauchen bzw. nur positiv / nur negativ in den Effekten auftauchen. Diese Fluenten können bei der Kodierung z.T. weggelassen werden. Dies führt bei der für IPP nötigen Grundinstantiierung der Aktionen zur Verringerung der Menge der Instanzen (da oft schon im Voraus ausgeschlossen werden kann, dass die Vorbedingungen von einigen Instantiierungen erfüllt werden können)[1] sowie zu einer Reduktion der bei der Expansion von `forall` und `exists` Quantoren auftretenden Teilformeln. Weiterhin wird die Umformung von einstelligen „inertia predicates“ zu Typen betrachtet, was insbesondere bei STRIPS-Problembeschreibungen ohne explizite Typangaben von Vorteil ist. Die beschriebenen Verfahren führen dort zum Teil zu Effizienzsteigerungen der Planungsprozeduren um Größenordnungen.

Die hier diskutierten Optimierungen wurden im BDD-basierten „intelligenten Modelchecking- und Planungssystem“ MIPS [28] mit großem Erfolg angewandt. Erstens werden – wie in [56] – konstante und Einwegprädikate unterschieden (dort: „inertia predicates“), die sich während der Planausführung nicht bzw. nur in eine Richtung ändern können. Die konstanten Prädikate können daher aus der Problembeschreibung eliminiert werden. Dies gilt auch für Einwegprädikate, falls diese im Anfangszustand des Planungs problems bereits ihren „Endzustand“ erreicht haben. Zweitens wird ein Algorithmus beschrieben, der Fluenten identifiziert, die die bereits erwähnten Gruppen bilden, und daher ein kompaktes Kodieren in Form eines binären Variablenvektors zulassen. Es ist sogar ein „Mischen“ verschiedener Prädikate möglich: Z.B. ist das Prädikat $\mathit{Free}(\mathit{Gripper})$ von dem Prädikat $\mathit{Carry}(\mathit{Ball}, \mathit{Gripper})$ eindeutig bestimmt und kann daher bei der Kodierung ganz eliminiert werden [25].

Heuristische Suche

Die Integration von Heuristiken ist für die „klassische“ Tiefen- oder Breitensuche sehr erfolgreich, ja geradezu notwendig, um gute Ergebnisse zu erzielen. Eine Reihe von Arbeiten bemühen sich, Heuristiken auch in die symbolische Breitensuche mit Hilfe von BDDs zu integrieren, d.h. eine symbolische heuristische Suche zu realisieren, die große Zustandsmengen in kompakter BDD-Darstellung durchsuchen kann. Dies hat zwar den Nachteil, dass die typische Eigenschaft der Breitensuche, stets einen kürzesten Plan zu liefern, oft geopfert wird; aber andererseits wird der Anwendungsbereich erweitert, wenn die Heuristik besser zum Ziel führt.

Einen ersten Schritt in diese Richtung ging [27] mit einer BDD-Realisierung des klassischen A* Suchalgorithmus, der die Suche in einem Suchbaum durch eine Prioritätswarteschlange der als nächstes zu expandierenden Zustände steuert. In ihrer Implementation BDDA* wird die Menge der Zustände als charakteristische Funktion über Tupeln aus den Zuständen und

[1] Bei BDDPLAN ist eine solche Instanziierung nicht nötig. Das Weglassen unnötiger Fluenten führt automatisch zu analogen Einsparungen. In der Implementation von BDDPLAN wurde eine Elimination unveränderlicher Fluenten vorgesehen, die hier nicht diskutiert wurde.

deren Bewertung aussagenlogisch kodiert durch ein BDD dargestellt. Die Zustände höchster Priorität (d.h höchster Bewertung) werden dann simultan expandiert. Der Algorithmus wird anhand des Sokoban Puzzles getestet, wobei das Resultat in diesem Fall allerdings noch nicht besser als eine normale Breitensuche mit BDDs ist.

[53] verbessert dies mit dem Algorithmus SETA* unter stärkerer Nutzung von Partitionierung. Erstens wird die Prioritätswarteschlange nicht wie bei BDDA* in ein einziges BDD kodiert, sondern die Menge der zu expandierenden Zustände für jede einzelne Priorität in einem separaten BDD gespeichert. Zweitens wird die Zustandsübergangsrelation partitioniert (vergleiche Abschnitt 9.3 auf Seite 122), wobei die in den einzelnen Partitionen versammelten Aktionen die gleiche Veränderung des Werts der Heuristikfunktion hervorrufen müssen. Durch diese Verfahrensweise ergibt sich z.T. eine Effizienzverbesserung um Größenordnungen gegenüber BDDA*.

[23] diskutiert die Verwendung von BDD-basierten Pattern-Datenbanken als Suchheuristik sowohl für die klassische als auch für symbolische heuristische Suche. Ein solches Pattern gibt jeweils für eine Menge von Zuständen eine untere Grenze für die Länge einer Aktionssequenz, die vom betrachteten Zustand zum Ziel führt, an. Eine solche untere Grenze wird durch Abstraktion von Fluenten erzeugt: Es wird ein vereinfachtes Planungsproblem gelöst, das jeweils nur ein Teil der im Problem auftretenden Fluenten berücksichtigt. Die heuristische Bewertung der Zustände wird aus den Planlängen aus mehreren solchen Abstraktionen mit unterschiedlichen berücksichtigten Fluenten zusammengesetzt. Dies führt in der Blocksworld zu deutlichen Ersparnissen. Außerdem wird eine Möglichkeit beschrieben, eine klassische A* Suche durch BDD-Techniken zu unterstützen, indem die zur Heuristikberechnung eingesetzten Pattern-Datenbanken mit Hilfe von BDDs kodiert werden. Sie können dadurch (bezüglich der erfassten Zustandsmenge) viel größer sein, als Datenbanken unter expliziter Aufführung der erfassten Zustände.

Das Planungssystem MIPS [24] integriert die im vorhergehenden Abschnitt beschriebene umfangreiche Vorverarbeitung aus Elimination von konstanter Fluenten und von trivialen und nachweislich nie ausführbaren Aktionen, sowie dem Zusammenfassen von Atomen in Gruppen, von denen jeweils nur 1 Fluent wahr ist, mit mehreren alternativ einsetzbaren Suchmethoden. Im Normalfall wird die heuristische Suche mit A* benutzt, aber es steht auch eine bidirektionale BDD-basierte Breitensuche sowie heuristische BDD-Suche mit BDDA* zur Verfügung. Die heuristischen Suchalgorithmen können optional mit symbolischen Patterndatenbanken als Heuristik unterstützt werden.

Erweiterung des Begriffes der Planungsdomäne

Für die maschinelle Bearbeitung von Planungsproblemen werden eine ganze Menge von Vereinfachungen gegenüber der realen Welt getroffen. Neben der Verbesserung der Suchalgorithmen zur Planfindung ist eine wichtige Entwicklungsrichtung von Planungssystemen, diese Vereinfachungen zu reduzieren. Die in Abschnitt 10.2.1 diskutierten Erweiterungen der Aus-

drucksmächtigkeit des Fluentkalküls auf gleichzeitige Aktionen, stetige Veränderungen und Nichtdeterminismus zielen in diese Richtung. Ebenso ist dies an der Weiterentwicklung der Planungsdomänendefinitionssprache PDDL zu beobachten, die sich bemüht, eine Übermenge der Fähigkeiten der an den Planungswettkämpfen teilnehmenden Planungssysteme darzustellen. So gibt es in PDDL2.1 [35] und PDDL+ [34] Domänenaxiome, Sicherheitsbedingungen, numerische Fluenten, Aktionsexpansionen zur Verfeinerung hierarchischer Pläne, parallele Aktionen, Aktionen mit Zeitdauern und stetigen Veränderungen, eine beschränkte Einführung exogener Ereignisse [26] sowie Planmetriken für die Suche optimaler Pläne. Sogar PDDL-Formulierungen Markovscher Entscheidungsprozesse sind im Gespräch [94] (vergleiche auch [41, 75] für den Fluentkalkül). Einige dieser Erweiterungen werden auch von BDD-unterstützten Planern wie MIPS [24] unterstützt.

Es gibt aber auch Erweiterungen, die den Begriff eines Plans verändern. Insbesondere bei nichtdeterministischen Umgebungen ist oft ein Plan im Sinne einer linearen Sequenz von Aktionen nicht mehr erfolgreich, da er dem Agenten keine Möglichkeit lässt, auf Fehlschläge / unerwartete Ausgänge von Aktionen zu reagieren. Neben der für den Rahmen des Fluentkalküls bereits angedeuteten Möglichkeit, Pläne als Programme mit abwechselnden Beobachtungen und Aktionen aufzufassen [91], oder durch eine Beobachtung der Planausführung neue Planungsvorgänge auszulösen [31], gibt es aber noch weitere Möglichkeiten, den Begriff eines Plans anders zu fassen. Einige davon, die den Einsatz von Zustands-Aktionstabellen erfordern, werden z.T. erst durch die Kompaktheit BDDs als Repräsentation für diese u.U. sehr großen Tabellen gangbar.

[20, 8] diskutieren das Konzept der sogenannten „conformant plans“ (englisch für konforme Pläne), das einen ersten Schritt in diese Richtung darstellt. Dies sind Pläne, die einen Agenten in einer nichtdeterministischen Planungsdomäne trotz unvollständiger Information über den Ausgangszustand unabhängig vom Ausgangspunkt seiner Aktionen zum Ziel führen, d.h. ohne dass er während der Planausführung Informationen aus seiner Umgebung sammeln muss. Das Grundprinzip dieser Arbeit ist es, die Menge von Zuständen, die durch Ausführung einer Aktionssequenz unabhängig vom Ausgang der Aktionen zum Erreichen des Zieles führen, rekursiv für alle Aktionssequenzen der Länge $1, 2, 3, \ldots$ zu berechnen, bis die Menge der Anfangszustände in einer solchen Zustandsmenge enthalten ist. In diesem Fall ist die zugehörige Aktionssequenz der gesuchte Plan.

Die Funktion, die die Menge dieser Zustände in Abhängigkeit von der Aktionssequenz beschreibt, wird dabei durch ein BDD kodiert, um auch Probleme beherrschbar zu machen, für die diese Funktion nicht mehr tabulierbar ist. Die Rekursion wird ebenfalls mit Hilfe von BDDs implementiert. Bei der Kodierung der beschriebenen Plan - Zustandsmengenrelation werden aussagenlogische Variablen für jedes Fluent eingeführt. Weiterhin wird für jeden Schritt im Plan eine Anzahl von Variablen hinzugefügt, die die Aktion im entsprechenden Schritt kodieren. Damit ist die Anzahl der Variablen relativ groß und steigt mit jedem Schritt (was den Planer zu einem für deterministische Planungsprobleme ungeeigneten Verfahren macht). Wie die Autoren experimentell nachweisen, schneidet der Planer durch seine BDD-Realisierung im Vergleich zu aktuellen Planungssystemen für „conformant plans“ gut

ab, und übertrifft diese bei einer Reihe von Problemen. [7] arbeitet dieses Konzept mit dem Planer HCSP unter Einsatz von symbolischer heuristischer Suche weiter aus.

Das Konzept der konformen Pläne ist aber relativ eng gefasst – solche Pläne existieren für viele Probleme nicht. [54, 52] gehen dafür von der Idee der universellen Pläne („universal plans“) aus. Diese bilden eine Tabelle, welche Aktion abhängig vom Zustand auszuführen ist, so dass der Agent unabhängig von den möglichen Ausgängen der nichtdeterministischen Aktionen das vorgegebene Ziel erreicht. Diese Zustand-Aktionstafel wird in BDD-Kodierung berechnet. [52] ergänzt dies durch eine heuristische Suche, die die Heuristik analog zu SETA* behandelt.

[19] erweitert das Konzept der universellen Pläne durch zyklische Pläne („cyclic plans“), die Wiederholungen einer fehlschlagenden Aktionsequenz zulassen („man schlage das Ei auf eine Kante, bis es zerbricht“). Weiterhin werden schwache Pläne („weak plans“), die vielleicht (d.h. bei geeignetem Ausgang der nichtdeterministischen Aktionen) zum Ziel führen, und starke Pläne, die unabhängig vom Ausgang der Aktionen immer zum Ziel führen, unterschieden. Das BDD-basierte System MBP ist in der Lage, alle Kombinationen von starken/schwachen und zyklischen/nicht-zyklischen Plänen zu generieren. MBP ist weiterhin in der Lage, solche Pläne auch unter nur eingeschränkter Beobachtbarkeit der Fluenten zu berechnen [7]. Es basiert auf einer Übersetzung des Planungsproblems in einen endlichen Automaten und Anwendung von Techniken aus dem Modelchecker NUSMV.

[55] diskutiert sogenannte n-fehlertolerante universelle Pläne: Zusätzlich zu den “normalen” deterministischen Effekten einer Aktion werden andere, unwahrscheinliche, Ausgänge der Aktionen zugelassen, die als „Fehlschlag“ bezeichnet werden und die Aktion damit nichtdeterministisch machen. Ein universeller Plan ist n-fehlertolerant, wenn er in der Lage ist, trotz max. n solcher Fehlschläge das Ziel zu erreichen. Die Plansuche erfolgt BDD-basiert unter zusätzlicher Integration eines „Fehlerzählers“ in den Zustand, der bei jedem „Fehlschlag“ um eins erhöht wird.

In eine andere Richtung erweitert [92] das System MBP. In praktischen Applikationen ist es oft nötig, Ziele wie „versuche Ziel A zu erreichen, wenn das möglich ist“, „wenn Ziel A nicht erreichbar ist, dann versuche Ziel B“ oder „erreiche Ziel A, und dann Ziel B, während Bedingung C aufrechterhalten wird“ zu spezifizieren. Dies geht aber über die in dieser Arbeit und in den meisten der erwähnten Arbeiten verwendete Definition eines Ziels als Bedingung, die der aktuellen Zustand erfüllen muss, hinaus, und unterscheidet sich in der Semantik auch von modalen Logiken wie LTL und CTL, die zum Teil zur Definition solcher *zeitlich ausgedehnter Ziele* verwendet wurden. [92] definiert eine Sprache zur Darstellung dieser Art von Zielen und präsentiert Planungsalgorithmen, die Pläne für solche Ziele berechnen können. Dabei wird ein Automat konstruiert, der terminiert, wenn das Ziel vollständig realisiert ist. Dieser Automat wird mit dem Automaten, der aus der Planungsdomäne konstruiert wird, zur Plansuche kombiniert. Diese Technik ist so allgemein, dass sie sich vermutlich auch entsprechend mit anderen Planungsalgorithmen kombinieren lässt.

10.3. Offene Probleme

Für die Erweiterung der Anwendbarkeit der in dieser Arbeit konstruierten Verfahrensweise gibt es einige naheliegende Ansatzpunkte.

1. Wie lässt sich das Fragment des Fluentkalkül erweitern, das mit Hilfe von BDDs behandelt werden kann? In die hier genutzte Methode lassen sich sicherlich weitere Elemente, wie indirekte Effekte von Aktionen, Schritt für Schritt integrieren. Es ist aber auch vorstellbar, dass eine allgemeinere Methode existiert, mit der eine großere Klasse von Formeln einer Fluentkalkül-Signatur automatisch zu BDDs übersetzt werden kann, so dass Korrektheit und Vollständigkeit bei der Modellsuche erhalten bleiben. Während die Fluentkalkül-Implementation FLUX in der Lage ist, die Effekte von Hunderten von Aktionen in der Sekunde zu berechnen [81], bereitet Planen oft aufgrund der exponentiellen Explosion des Suchraums Schwierigkeiten. Eine Kombination mit den in dieser Arbeit ausgearbeiteten Techniken könnte hier vielleicht Abhilfe schaffen, da BDDs oft geeignet sind, exponentielle Mengen von Schlussfolgerungen kompakt darzustellen und zu bearbeiten.

2. Wie bereits diskutiert, bieten die Weiterentwicklungen PDDL2.1 [35] und PDDL+ [34] zahlreiche zusätzliche Ausdrucksmöglichkeiten. Wie lassen sich diese auf den Fluentkalkül abbilden und damit der vorhandene Reichtum an PDDL-Planungsproblemen (z.B. in der Planungsdomänenbibliothek [66] sowie aus den Planungswettkämpfen [65, 4, 61]) für Experimente nutzbar machen?

3. Für die Behandlung von Nichtdeterminismus wurden im vorhergehenden Abschnitt eine Anzahl von Ansätzen diskutiert. Wie lassen sich die Konzepte von konformen, universellen und zyklischen Plänen usw. auf den Fluentkalkül übertragen und wie lässt sich dies geeignet auf BDDs abbilden?

4. Bei zunehmender Komplexität von Planungsproblemen werden die BDDs oft so groß, dass sie mit der gegenwärtigen Technik nicht mehr in den Speicher passen. Wie lässt sich diese Grenze, ausgehend von den bereits diskutierten Ansätzen wie z.B. in [24], umgehen?

5. Studien wie [93] und [40] weisen nach, dass BDD-Techniken sich fundamental von Techniken wie die Davis-Putnam Prozedur bzw. Resolution unterscheiden: Es gibt Problemklassen, auf denen nachweislich mit BDDs besser gearbeitet werden kann, als mit den anderen Prozeduren, und umgekehrt. Ein ähnliches Bild ergibt sich für Planungsprobleme: Es gibt Probleme, die sich mit BDDs gut behandeln lassen, aber mit anderen Planungstechniken wie heuristischer Suche [42] oder stochastischer lokaler Suche für Planen als Erfüllbarkeitsproblem [36] schwierig sind, und umgekehrt. Doch darüber, für welche Probleme BDDs besonders gut geeignet sind, ist nur wenig bekannt. Verschiedene Planungsprogramme setzen daher auf eine heuristische Suche und schalten

auf BDDs um, wenn diese Methode nicht in angemessener Zeit zum Ziel führt [18, 24]. Doch welche Möglichkeiten gibt es für eine weitergehende Integration, wie z.B. in [23]?

A. PDDL-Quelltexte einiger verwendeter Beispiele

A.1. Das Aktenmappen-Beispiel

```
(define (domain Aktenmappen-Welt)
  (:types Objekt Ort)
  (:constants Mappe - Objekt)
  (:predicates (am-Ort ?x - Objekt ?l - Ort)
               (in-Mappe ?x - Objekt))

  (:action Herausnehmen
     :parameters (?x - Objekt)
     :precondition (in-Mappe ?x)
     :effect (not (in-Mappe ?x)))

  (:action Hineinlegen
     :parameters (?x - Objekt ?l - Ort)
     :precondition (and (am-Ort ?x ?l) (am-Ort Mappe ?l)
                                       (not (in-Mappe ?x)))
     :effect (in-Mappe ?x))

  (:action Transport
     :parameters (?m ?l - Ort)
     :precondition (and (am-Ort Mappe ?m) (not (= ?m ?l)))
     :effect (and (am-Ort Mappe ?l) (not (am-Ort Mappe ?m))
                  (forall (?z - Objekt)
                       (when (in-Mappe ?z)
                             (and (am-Ort ?z ?l)
                                  (not (am-Ort ?z ?m)))))))
)

(define (problem Transportiere)
    (:domain Aktenmappen-Welt)
```

```
    (:objects Buch Scheck - Objekt  Haus Buero - Ort)
    (:init (am-Ort Mappe Haus) (am-Ort Buch Haus)
           (am-Ort Scheck Haus) (in-Mappe Scheck))
    (:goal (and (am-Ort Mappe Buero) (am-Ort Buch Buero)
                (am-Ort Scheck Haus)))
)
```

A.2. Das “Türme von Hanoi“-Problem

Dabei geht es um das bekannte Problem n Scheiben, die geordnet nach fallender Größe auf einem Stab gesteckt sind, auf einen anderen Stab zu bringen. Jeder Zug besteht dabei aus dem Umstecken der obersten Scheibe eines der Stäbe auf einen anderen Stab, wobei eine Scheibe nicht über eine größere Scheibe gesteckt werden darf. Für n Scheiben hat der kürzeste Plan eine Länge von $2^n - 1$, der Zustandsraum ist 3^n. Das Problem ist für 4 Scheiben als PDDL-Quelltext dargestellt.

```
(define (domain towers-of-hanoi)
  (:requirements :typing :adl)
  (:types disk peg)
  (:constants pega pegb pegc - peg)
  (:predicates (on ?d - disk ?p - peg)
               (larger ?d1 ?d2 - disk)) ; d1 is larger than d2
  (:action move
       :parameters (?d - disk ?p1 ?p2 - peg)
       :precondition
           (and (on ?d ?p1) (not (= ?p1 ?p2))
                (not (exists (?d1 - disk)
                             (and (on ?d1 ?p1) (larger ?d ?d1))))
                (not (exists (?d1 - disk)
                             (and (on ?d1 ?p2) (larger ?d ?d1)))))
       :effect (and (not (on ?d ?p1)) (on ?d ?p2))))


(define (problem hanoi-4)
  (:domain towers-of-hanoi)
  (:objects d1 d2 d3 d4 - disk)
  (:init (on d1 pega) (on d2 pega) (on d3 pega) (on d4 pega)
         (larger d2 d1)
         (larger d3 d1) (larger d3 d2)
         (larger d4 d1) (larger d4 d2) (larger d4 d3)
         )
```

```
  (:goal (and (on d1 pegc) (on d2 pegc) (on d3 pegc)
         (on d4 pegc))))
```

A.3. Das GRIPPER-Problem

Dieses Beispiel (siehe Figur 9.6 auf Seite 129) ist aus der AIPS-competition 1998 [65] entnommen. Die Anzahl der Bälle ist variabel. Für $2n$ Bälle hat der kürzeste Plan eine Länge von $6n - 1$; dabei ist ein Zustandsraum von ca. $n^2 2^{n-1}$ Zuständen zu durchsuchen.

```
(define (domain gripper-typed)
   (:requirements :typing)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robby ?r - room)
                (at ?b - ball ?r - room)
                (free ?g - gripper)
                (carry ?o - ball ?g - gripper))

   (:action move
       :parameters  (?from ?to - room)
       :precondition (at-robby ?from)
       :effect (and  (at-robby ?to)
                     (not (at-robby ?from))))



   (:action pick
       :parameters (?obj - ball ?room - room
                                ?gripper - gripper)
       :precondition  (and  (at ?obj ?room) (at-robby ?room)
                            (free ?gripper))
       :effect (and (carry ?obj ?gripper)
                    (not (at ?obj ?room))
                    (not (free ?gripper))))


   (:action drop
       :parameters  (?obj - ball ?room - room
                                  ?gripper - gripper)
       :precondition  (and  (carry ?obj ?gripper)
                            (at-robby ?room))
```

```
       :effect (and (at ?obj ?room)
                    (free ?gripper)
                    (not (carry ?obj ?gripper)))))

(define (problem gripper-x-1)
   (:domain gripper-typed)
   (:objects rooma roomb - room
             ball4 ball3 ball2 ball1 - ball)
   (:init (at-robby rooma)
          (free left)
          (free right)
          (at ball4 rooma)
          (at ball3 rooma)
          (at ball2 rooma)
          (at ball1 rooma))
   (:goal (and (at ball4 roomb)
               (at ball3 roomb)
               (at ball2 roomb)
               (at ball1 roomb))))
```

Literaturverzeichnis

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 6(C-27):509–516, June 1978. 17

[2] Henrik Reif Andersen. An introduction to binary decision diagrams. Lecture notes, `http://www.itu.dk/people/hra/notes-index.html`, October 1997. 17, 18, 22

[3] F. Baader and J. Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, chapter 2, pages 41–126. Oxford University Press, 1994. 15

[4] Fahiem Bacchus. Aips-00 planning competition. `http://www.cs.toronto.edu/aips2000/`, 2000. 128, 142

[5] Fahiem Bacchus. The AIPS '00 planning competition. *AI Magazine*, 22(3):47–56, 2001. 134

[6] A. B. Baker. A simple solution to the Yale shooting problem. In *Proceedings of the International Conference on Knowledge Representation and Reasoning*, pages 11–20, 1989. 58

[7] Piergiorgio Bertoli, Alessandro Cimatti, Marco Pistore, Marco Rovieri, and Paolo Traverso. MBP: a model based planner. In *IJCAI'01 Workshop on Planning under Uncertainty*, 2001. 141

[8] Piergiorgio Bertoli, Alessandro Cimatti, and Marco Roveri. Heuristic search + symbolic model checking = efficient conformant planning. In *Proceedings of the IJCAI*, pages 467–472, 2001. 140

[9] S.-E. Bornscheuer and H. Lehmann. On the combination of partial action descriptions. In G. Antoniou and J. Slaney, editors, *Advanced Topics in Artificial Intelligence*, volume 1502 of *LNAI*. Proceedings of the 11th Australian Joint Conference on Artificial Intelligence (AI'98), Springer–Verlag, 1998. 45, 62

[10] S.-E. Bornscheuer and Helko Lehmann. Concurrent productions, consumptions, and occupations. In M.-A. Williams, editor, *JCAI'97 Workshop on Nonmonotonic Reasoning, Action and Change*, pages 52–64, 1997. 45, 62

[11] F. M. Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer, Boston, 1990. 17

[12] S. Brüning, S. Hölldobler, J. Schneeberger, U. Sigmund, and M. Thielscher. Disjunction in resource–oriented deductive planning. In D. Miller, editor, *Proceedings of the International Logic Programming Symposium*, page 670, 1993. 45

[13] Randal E. Bryant. Graph–based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8(C-35):677–691, 1986. 4, 17, 20, 21

[14] Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992. 4, 21, 22

[15] J. Burch, E. Clarke, K. McMillan, and D. Dill. Symbolic model checking: 10^{120} states and beyond. *Information and Computation*, 98(2):142–170, 1992. 4, 17, 33, 134

[16] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 13(4):401–424, April 1994. 4, 17

[17] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for AR. In S. Steel and R. Alami, editors, *Proceedings of the Fourth European Conference on Planning (ECP97)*, number 1348 in Lecture Notes in Artificial Intelligence, pages 130–142, Toulouse, France, Sept. 1997. Springer-Verlag. 4

[18] Alessandro Cimatti, Enrico Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Integrating BDD-based and SAT-based symbolic model checking. In *Proceedings of the Frontiers of Combining Systems (FROCOS)*, number 2309 in LNAI, Santa Margherita, Italy, 2002. Springer. 143

[19] Alessandro Cimatti, Marco Pistore, Marco Rovieri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence, special issue on planning with uncertainty and incomplete information*, 147(1-2):35–84, 2003. 141

[20] Alessandro Cimatti and Marco Rovieri. Conformant planning via model checking. In *Proceedings of the Fifth European Conference on Planning (ECP99)*, number 1809 in LNAI, pages 21–34, Durham, UK, september 1999. 140

[21] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum, New York, NY, 1978. 61

[22] V. Diekert and G. Rozenberg, editors. *The book of traces*. World Scientific, Singapore etc., 1995. 52

[23] Stefan Edelkamp. Symbolic pattern databases in heuristic search planning. In *The International Conference on AI Planning and Scheduling (AIPS)*, Toulouse, 2002. 139, 143

[24] Stefan Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research, Special Issue on the 3rd International Planning Competition*, 20:195–238, 2003. 139, 140, 142, 143

[25] Stefan Edelkamp and Malte Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP'99*, LNAI, pages 135–147, Durham, 1999. Springer. `http://www.informatik.uni-freiburg.de/~edelkamp/.index.eng.html`. 138

[26] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Fakultät Informatik, Freiburg, 2003. 140

[27] Stefan Edelkamp and Frank Reffel. OBDDs in heuristic search. In *KI'98*, LNAI, pages 81–92. Springer, Bremen, 1998. `http://www.informatik.uni-freiburg.de/~edelkamp/.index.eng.html`. 128, 138

[28] Stefan Edelkamp and Frank Reffel. Deterministic state space planning with BDDs. In *ECP'99*, pages 381–382. Springer, 1999. Extended Version published as [29]. `http://www.informatik.uni-freiburg.de/~edelkamp/.index.eng.html`. 138

[29] Stefan Edelkamp and Frank Reffel. Deterministic state space planning with BDDs. Technical Report 102, University Freiburg, Computer Science Institute, 1999. `http://www.informatik.uni-freiburg.de/tr/1999/Report120/.index.html`. 151

[30] Kerstin Eder. A resource oriented deductive approach towards hierarchical planning. Master's thesis, TU Dresden, 1995. 45

[31] Matthias Fichtner, Axel Großmann, and Michael Thielscher. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2–4):371–392, 2003. 136, 140

[32] Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971. 39, 75

[33] Maria Fox and David Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998. 137

[34] Maria Fox and Derek Long. PDDL+ level 5: An extension to PDDL2.1 for modelling planning domains with continuous time-dependent effects. Technical report, University of Durham, 2001. Verfügbar unter `http://planning.cis.strath.ac.uk/competition/`. 140, 142

[35] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research, Special Issue on the 3rd International Planning Competition*, 20:61–124, 2003. 140, 142

[36] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research, Special Issue on the 3rd International Planning Competition*, 20:239–290, 2003. 142

[37] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. The planning domain definition language. Technical report, Verfügbar unter `http://www.cs.yale.edu/homes/dvm/`, 1998. 73, 77, 78, 91, 134

[38] Gnu's not unix. `http://www.gnu.org/home.de.html`. 119

[39] C. Green. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 219–239. Morgan Kaufmann, 1969. 2

[40] Jan Friso Groote and Hans Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *Perspectives of System Informatics, 4th International Andrei Ershov Memorial Conference (PSI 2001)*, number 2244 in LNCS, pages 33–38, Novosibirsk, Russia, 2001. Springer. 142

[41] Axel Großmann, Steffen Hölldobler, and Olga Skvortsova. Symbolic dynamic programming within the fluent calculus. In Naohiro Ishii, editor, *Proceedings of the IASTED International Conference on Artificial and Computational Intelligence*, pages 378–383, Tokyo, Japan, September 25-27 2002. ACTA Press. ISBN: 0-88986-358-X. 140

[42] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 142

[43] S. Hölldobler. On deductive planning and the frame problem. In A. Voronkov, editor, *Proceedings of the Conference on Logic Programming and Automated Reasoning*, pages 13–29. Springer, LNCS, 1992. 3

[44] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8(3):225–244, 1990. A short version appeared in the Proceedings of the German Workshop on Artificial Intelligence, Informatik Fachberichte *216*, pages 63-73, 1989. 2, 44, 70

[45] S. Hölldobler and M. Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:99–133, 1995. 3, 46, 48, 61, 62, 133

[46] S. Hölldobler and M. Thielscher. Objects, specificity, logic, and change. In L. Dreschler-Fischer and S. Pribbenow, editors, *KI–95 Activities: Workshops, Posters, Demos*, pages 37–42, Bonn, 1995. Gesellschaft für Informatik e.V. 47

[47] Steffen Hölldobler and Josef Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8(3):225–244, 1990. 39

[48] Steffen Hölldobler and Josef Schneeberger. Constraint equational logic programming and resource-based partial order planning. Technical Report WV-96-08, TU Dresden, Institut KI, Fachgebiet Wissensverarbeitung, 1996. 45

[49] Steffen Hölldobler and Hans-Peter Störr. Solving the entailment problem in the fluent calculus using binary decision diagrams. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic (CL)*, number 1861 in LNCS, pages 747–760. Springer, 2000. 135, 136

[50] J. Jaffar, J-L. Lassez, and M. J. Maher. A theory of complete logic programs with equality. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 175–184. ICOT, 1984. 61

[51] Joxan Jaffar, Jean-Louis Lassez, and Michael J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 1(3):211–223, 1984. 14

[52] Rune Jensen, Manuela Veloso, and Randy Bryant. Guided symbolic universal planning. In *Proceedings of ICAPS'03*, pages 123–132, 2003. 141

[53] Rune M. Jensen, Randal E. Bryant, and Manuela M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAI'02)*, pages 668–673, 2002. 139

[54] Rune M. Jensen and Manuela M. Veloso. OBDD-based universal planning: Specifying and solving planning problems for synchronized agents in non-deterministic domains. In M.J. Wooldrige and M. Veloso, editors, *Artificial Intelligence Today*, pages 213–248. Springer, 1999. 141

[55] Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling ICAPS-04*, 2004. 141

[56] Jana Koehler and Jörg Hoffmann. Handling of inertia in a planning system. Technical Report 122, University Freiburg, 1999. `http://www.informatik.uni-freiburg.de/~koehler/papiere/planning.html`. 137, 138

[57] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 1(4):67–95, 1986. 2, 43

[58] C. Y. Lee. Binary decision programs. *Bell System Technical Journal*, 4(38):985–999, July 1959. 17

[59] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987. 7, 44

[60] Derek Long. Automatic analysis of planning domains. In *Tenth Workshop on Automated Reasoning*, University of Liverpool, GB, 2003. 137

[61] Derek Long, Maria Fox, David E Smith, Drew McDermott, Fahiem Bacchus, and Hector Geffner. The 2002 international planning competition. `http://planning.cis.strath.ac.uk/competition/`, 2002. 128, 142

[62] Yves Martin. The concurrent, continuous FLUX. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003. 136

[63] J. McCarthy. Situations and actions and causal laws. Stanford Artificial Intelligence Project: Memo 2, 1963. 2, 39, 43, 66

[64] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463 – 502. Edinburgh University Press, 1969. 2, 3, 39, 43, 66

[65] Drew McDermott. Aips98 planning competition results. `ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`, 1998. 128, 130, 142, 147, 158

[66] Drew McDermott. Planning problem repository. `ftp://ftp.cs.yale.edu/pub/mcdermott/domains/`, 1999. 73, 120, 122, 128, 134, 142, 161

[67] Drew McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000. 73, 134

[68] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *ECP-97*, pages 338–350, 1997. `http://www.informatik.uni-freiburg.de/~koehler/papiere/planning.html`. 137

[69] Leszek Pacholski and Andreas Podelski. Set constraints: a pearl in research on constraints. In G. Smolka, editor, *Proceedings of the International Conference on Constraint Programming (CP)*, volume 1330 of *LNCS*, pages 549–561. Springer, 1997. 48

[70] E. P. D. Pednault. Synthesizing plans that contaion actions with context–dependent effects. *Computational Intelligence, special issue on planning*, 4(4):356–372, nov. 1988. 73

[71] Rajeev Ranjan et al. Cal BDD package, 1999. `http://www-cad.eecs.berkeley.edu/Respep/Research/bdd/cal_bdd/`. 119

[72] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991. 2, 39, 43

[73] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. Number 395 in LNAI. Springer Verlag, Berlin, 1989. 10, 14

[74] John C. Shepherdson. SLDENF - resolution with equality. *Journal of Automated Reasoning*, 8:297–306, 1992. 14, 48, 62

[75] Olga Skvortsova. Towards automated symbolic dynamic programming. Master's thesis, TU Dresden, April 2003. 140

[76] Fabio Somenzi. Binary decision diagrams. Working Material for the International Summer School Markoberdort in Calculational System Design. Nato Science Comitee & Institut für Informatik TU München, July 1998. 17, 22, 33

[77] Hans-Peter Störr. Planning in the fluent calculus using binary decision diagrams. *AI Magazine*, 22(3):103–106, 2001. 135, 136

[78] Hans-Peter Störr. Bddplan, 2003. `http://www.stoerr.net/bddplan.html`. 119, 135

[79] Hans-Peter Störr and Michael Thielscher. A new equational foundation for the fluent calculus. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic (CL)*, number 1861 in LNCS, pages 733–745. Springer, 2000. 134, 136

[80] M. Thielscher. *Automatisiertes Schließen über Kausalbeziehungen mit SLDENF–Resolution*, volume 76 of *DISKI*. infix, 1995. 3, 48, 61, 62

[81] M. Thielscher. Pushing the envelope: Programming reasoning agents. In Chitta Baral and Sheila McIlraith, editors, *AAAI Workshop Technical Report WS-02-05: Cognitive Robotics*, volume Technical Report WS-02-05. AAAI Press, 2002. 142

[82] Michael Thielscher. AC1-Unifikation in der linearen logischen Programmierung. Master's thesis, Technische Hochschule Darmstadt, Fachgebiet Intellektik, Fachbereich Informatik, 1992. 48

[83] Michael Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1997. 91

[84] Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998. URL: `http://www.ep.liu.se/ea/cis/1998/014/`. 2, 5, 44, 46, 47, 65

[85] Michael Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299, 1999. 87

[86] Michael Thielscher. Modeling actions with ramifications in nondeterministic, concurrent, and continuous domains—and a case study. In H. Kautz and B. Porter, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, pages 497–502, Austin, TX, July 2000. MIT Press. 2, 5, 135, 136

[87] Michael Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 109–120, Breckenridge, CO, April 2000. Morgan Kaufmann. 5, 136

[88] Michael Thielscher. Programming of reasoning and planning agents with FLUX. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the International Conference on Principles of Knowlege Representation and Reasoning*, Toulouse, France, April 2002. Morgan Kaufmann. 3, 48

[89] Michael Thielscher. Programming of reasoning and planning agents with FLUX. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 435–446, Toulouse, France, April 2002. Morgan Kaufmann. 136

[90] Michael Thielscher. Reasoning about actions with CHRs and finite domain constraints. In *Proceedings of the International Conference on Logic Programming (ICLP)*, Copenhagen, Danmark, 2002. 133, 136

[91] Michael Thielscher. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 2004. 136, 140

[92] P. Traverso U. Dal Lago, M. Pistore. Planning with a language for extended goals. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence, Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 447–454, Edmonton, Alberta, Canada, 2002. AAAI Press. 141

[93] T. E. Uribe and M. E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In J. P. Jouannaud, editor, *1st International Conference on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 34–49. Springer-Verlag Inc, September 1994. 142

[94] Håkan L. S. Younes. Extending PDDL to model stochastic decision processes. In *Proceedings of the ICAPS-03 Workshop on PDDL*, pages 95–103, Trento, Italy, 2003. 140

Abbildungsverzeichnis

Tabellenverzeichnis

Index

Row, row, row your boat,

gently down the stream.

Merrily, merrily, merrily, merrily –

life is but a dream.