ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG
Fakultät für Angewandte Wissenschaften
Institut für Informatik

Lehrstuhl für Grundlagen der Künstlichen Intelligenz
Prof. Dr. Bernhard Nebel

Optimales domänenspezifisches Planen in der Transport- & Routenplanungsfamilie

Diplomarbeit von
Thomas Keller

November 2008

Zusammenfassung

Die klassische, domänenunabhängige Handlungsplanung ist ein Gebiet, das heute von vielen Forschungsgruppen mit sehr unterschiedlichen Methoden untersucht wird. Trotz großer Fortschritte in den letzten Jahren mehren sich aber die Anzeichen, dass klassische Verfahren wie die heuristische Suche oder das Planen als Erfüllbarkeitsproblem bald an eine Grenze stoßen werden. Eine Möglichkeit, dennoch immer komplexere Planungsaufgaben zu lösen, könnte darin liegen, domänenspezifische Erkenntnisse in die Planung miteinfließen zu lassen. Diese Arbeit beschäftigt sich daher mit zwei dafür relevanten Themen: Zum einen werden Möglichkeiten formalisiert, mit deren Hilfe Teilprobleme erkannt und Planungssystemen zusätzliche Informationen über Domänen zugänglich gemacht werden können. Zum anderen werden zwei Planer für Domänen der Transport- und Routenplanungsfamilie vorgestellt, und dabei aufgezeigt, in welcher Form deren Analyse durchgeführt werden kann. Die Ergebnisse, die mit den implementierten Planern erzielt wurden, zeigen dabei deutlich auf, dass die Ausnutzung domänenspezifischen Wissens zu sehr viel leistungsfähigeren Planern führt.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben:
Ein ganz besonderer Dank geht dabei an Dipl.Inf. Robert Mattmüller, dessen hervorragende Betreuung dieser Arbeit, seine unermüdliche Bereitschaft zu Diskussionen sowie das Beisteuern vieler guter Ideen in dieser Form weder alltäglich ist noch erwartet werden kann.
Ebenfalls bedanken möchte ich mich bei Prof. Dr. Bernhard Nebel, der mir die Möglichkeit einräumte, meine Diplomarbeit an seinem Lehrstuhl anzufertigen. Auch dafür, dass er sich bereit erklärte, als Gutachter dieser Arbeit zur Verfügung zu stehen, möchte ich ihm ebenso wie dem Zweitgutachter, Prof. Dr. Wolfram Burgard meinen Dank aussprechen.
Letztlich möchte ich auch meinem Vater danken. Ohne deine Unterstützung, auf die ich mich immer verlassen konnte, wäre diese Arbeit nur schwer möglich gewesen.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, den 14. November 2008 Thomas Keller

Inhaltsverzeichnis

1 Einleitung

Einhergehend mit der Entwicklung leistungsstärkerer Computer bringt auch die Künstliche Intelligenz (KI) immer bessere Ergebnisse hervor. Dies galt lange auch für eines ihrer Teilgebiete, die Handlungsplanung, welche sich mit der „Aufgabe, eine Folge von Aktionen zu finden, die ein Ziel erreichen “beschäftigt [RN95]. Insbesondere an *domänenunabhängigen* Planungssystemen, also solchen, die alle durch eine *Beschreibungssprache* darstellbaren Probleme lösen können, wird intensiv geforscht. Die beiden populärsten und erfolgreichsten Techniken der letzten Jahre für diese Aufgabe sind die heuristische Suche [Bon01] sowie Planen als Erfüllbarkeitsproblem [KS92]. Obwohl immer komplexere *Domänen* zum Einsatz kommen, werden auch immer wieder Planer entwickelt, die diese Planungsaufgaben in angemessener Zeit lösen können. Dennoch gehören die Fortschritte im Hinblick auf die Leistungsfähigkeit verwendeter Algorithmen, die in den ersten Tagen automatisierter Planer erzielt wurden, der Vergangenheit an, und auch die Qualität der ermittelten Pläne, die schon immer unter der großen Anzahl teils grundverschiedener Aufgaben litt, konnte nicht in vergleichbarem Maße gesteigert werden.

Die Ermittlung *optimaler Pläne* ist, im Vergleich zu *suboptimaler Planung*, ungemein schwer. Leider ist diese für die Praxistauglichkeit automatisierter Planer in vielen Gebieten aber eine Grundvoraussetzung. Zwar werden auch in dieser Disziplin intensive Anstrengungen unternommen und Fortschritte erzielt, Grund zur Hoffnung, vergleichbar leistungsfähige, domänenunabhängige und optimale Planungssysteme in nächster Zeit zu entwickeln besteht aber nicht. Aus diesem Grund untersucht diese Arbeit, inwieweit *domänenspezifische* Erkenntnisse auf domänenunabhängiges, optimales Planen übertragen werden können. Dazu ist es notwendig, in Domänen auftretende *Subprobleme* zu erkennen, in *Domänenfamilien* zu beschreiben und diese dann gezielt zu untersuchen. Verbunden mit einer automatischen Kategorisierung beliebiger Domänen können dann Planungssysteme entwickelt werden, die das durch die Analyse von Domänenfamilien gewonnene Wissen dynamisch einsetzen.

Dazu wird in Kapitel 2 eine erweiterte Formalisierung von Grundlagen der Handlungsplanung gegeben, welche durch die auf dem *Internationalen Planungswettbewerb* verwendete Beschreibungssprache PDDL inspiriert ist. Insbesondere die Erweiterung um *Planungsaufgabenbeschränkungen* ist dabei ein interessanter Aspekt, der in Kapitel 3 näher untersucht und Auswirkungen auf die Generierung eines *relevanten Zustandsraumes* diskutiert werden. Nach einer Erweiterung des Grundlagenkapitels um numerische Variablen gibt Kapitel 5 eine Formalisierung von Domänenfamilien und stellt mit den *Familien der Transport- und Routenplanungprobleme* die beiden gängigsten vor. In den Kapiteln 6 und 7 werden schließlich zwei Mitglieder dieser Familien detailliert in Bezug auf optimales Planen untersucht, sowie die Ergebnisse der darauf aufbauenden Planer beschrieben. Abgeschlossen wird diese Arbeit mit einer kurzen Zusammenfassung.

2 Grundlagen der Handlungsplanung

Dieses Kapitel erläutert die zum Verständnis der Arbeit notwendigen Grundlagen und verdeutlicht diese anhand eines Beispiels. Der erste Abschnitt präsentiert eine Formalisierung der grundlegenden Konzepte. Diese ist angelehnt an die Form, in welcher Planungsaufgaben in dem im darauf folgenden Abschnitt vorgestellten *Internationalen Planungswettbewerb* gegeben sind, und unterscheidet sich dementsprechend stark von üblicherweise verwendeten Definitionen. Abgeschlossen wird dieses Kapitel mit einer Analyse der Komplexität der wichtigsten, für die Handlungsplanung relevanten Probleme.

2.1 Formalisierung

Ziel der Handlungsplanung ist es, für ein gegebenes Problem eine Abfolge von *Aktionen* oder *Operatoren* zu finden, durch deren Ausführung ein vorher bestimmtes Ziel erreicht wird. Um einander ähnliche *Planungsaufgaben* effizient stellen und bearbeiten zu können, wird dafür zuerst eine *Planungsdomäne* definiert, in welcher die allen Varianten gemeinen, grundlegenden Charakteristika der Welt schematisch beschrieben werden. In Planungsaufgaben können Eigenschaften von *Objekten* oder Beziehungen dieser zueinander dann durch Verwendung der *Prädikatenschemata* modelliert werden.

Definition 2.1 ***Prädikatenschema***
Sei Ω eine Menge von Objekten.
Ein ***Prädikatenschema*** *ist ein 2-Tupel $\rho = \langle \mathrm{na}, \mathrm{ar} \rangle$ mit den Komponenten:*

- na *ist der* ***Name*** *des Prädikatenschemas (über einem endlichen Alphabet Σ).*
- $\mathrm{ar} \in \mathbb{N}_0$ *ist die* ***Stelligkeit*** *des Prädikatenschemas.*

Äquivalent zur Angabe eines Prädikatenschemas durch $\rho = \langle \mathrm{na}, \mathrm{ar} \rangle$ sei die Angabe als $\mathrm{na}(\mathrm{par}_1, \ldots, \mathrm{par}_{\mathrm{ar}})$ *oder* $\rho(\mathrm{par}_1, \ldots, \mathrm{par}_{\mathrm{ar}})$.

Prädikatenschemata und Prädikate bilden die Grundlage vieler Definition dieser Arbeit und werden in unterschiedlichen Formen immer wieder verwendet. Daher werden im Folgenden Grammatiken für immer wiederkehrende Formeln in BNF [Knu64] sowie einige Definition von Mengen angegeben.

Definition 2.2 ***Grammatiken und Mengen***
Sei $\mathcal{P}$ eine Menge von Prädikatensymbolen $\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)$ und $\{\mathrm{par}_1, \ldots, \mathrm{par}_n\}$ eine Menge von Variablensymbolen sowie $n \geq k$, $n \geq j$ und $l \in \mathbb{N}$.
Eine ***Konjunktion*** *ϕ wird rekursiv gebildet durch die Grammatik:*

$$\phi := \rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \phi \wedge \phi$$

Eine ***aussagenlogische Formel*** *ϕ wird rekursiv gebildet durch die Grammatik:*

$$\phi := \rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \phi \wedge \phi \mid \phi \vee \phi$$

Eine ***prädikatenlogische Formel*** *ϕ wird rekursiv gebildet durch die Grammatik:*

$$\phi := \rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall\ \mathrm{par}_j \phi \mid \exists\ \mathrm{par}_j \phi \mid \exists^{=l}\ \mathrm{par}_j \phi$$

*Die **Symbolmenge** $\mathrm{Sy}(\phi)$ einer Konjunktion ϕ ist rekursiv über dem Aufbau von ϕ definiert durch:*

$$\mathrm{Sy}(\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)\}$$

$$\mathrm{Sy}(\neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \mathrm{Sy}(\phi)$$

$$\mathrm{Sy}(\phi \wedge \phi) := \mathrm{Sy}(\phi) \cup \mathrm{Sy}(\phi)$$

*Die **polarisierte Symbolmenge** $\mathrm{Sy}_{\pm}(\phi)$ einer Konjunktion ϕ ist rekursiv über dem Aufbau von ϕ definiert durch:*

$$\mathrm{Sy}_{\pm}(\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 1\rangle\}$$

$$\mathrm{Sy}_{\pm}(\neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 0\rangle\}$$

$$\mathrm{Sy}_{\pm}(\phi \wedge \phi) := \mathrm{Sy}_{\pm}(\phi) \cup \mathrm{Sy}_{\pm}(\phi)$$

*Die **Menge freier Variablen** $\mathrm{Fr}(\phi)$ einer prädikatenlogischen Formel ϕ ist rekursiv über dem Aufbau von ϕ definiert durch:*

$$\mathrm{Fr}(\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\mathrm{par}_1, \ldots, \mathrm{par}_k\}$$

$$\mathrm{Fr}(\neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\mathrm{par}_1, \ldots, \mathrm{par}_k\}$$

$$\mathrm{Fr}(\phi \wedge \phi) := \mathrm{Fr}(\phi) \cup \mathrm{Fr}(\phi)$$

$$\mathrm{Fr}(\phi \vee \phi) := \mathrm{Fr}(\phi) \cup \mathrm{Fr}(\phi)$$

$$\mathrm{Fr}(\forall\ \mathrm{par}_j\phi) := \mathrm{Fr}(\phi) \setminus \{\mathrm{par}_j\}$$

$$\mathrm{Fr}(\exists\ \mathrm{par}_j\phi) := \mathrm{Fr}(\phi) \setminus \{\mathrm{par}_j\}$$

$$\mathrm{Fr}(\exists^{=l}\ \mathrm{par}_j\phi) := \mathrm{Fr}(\phi) \setminus \{\mathrm{par}_j\}$$

*Eine **Effektkonjunktion** ist eine Konjunktion ϕ für die zusätzlich für alle $\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \in \mathcal{P}$ gilt, dass nicht:*

$$(\langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 0\rangle \in \mathrm{Sy}_{\pm}(\phi) \textit{ und } \langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 1\rangle \in \mathrm{Sy}_{\pm}(\phi))$$

*Eine **vollständige Konjunktion** ist eine Effektkonjunktion ϕ, für die zusätzlich für alle $\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \in \mathcal{P}$ gilt, dass:*

$$\langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 0\rangle \in \mathrm{Sy}_{\pm}(\phi) \textit{ oder } \langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 1\rangle \in \mathrm{Sy}_{\pm}(\phi)$$

*Eine **geschlossene prädikatenlogische Formel** ist eine prädikatenlogische Formel ϕ, für die zusätzlich gilt:*

$$\mathrm{Fr}(\phi) = \emptyset$$

Für alle gegebenen Symbole gelte die Semantik der Prädikatenlogik. Es sei allerdings darauf hingewiesen, dass in einigen Fällen die gegebene Grammatik von der üblicherweise in der Logik verwendeten abweicht oder zusätzliche Einschränkungen existieren. Außerdem werden nicht immer alle durch eine der gegebenen Grammatiken erstellten Formeln tatsächlich auch als Formel im Sinne der Logik genutzt werden.

Ein Beispiel dafür ist der *Effekt* eines *Operatorenschemas*, welcher abstrakt beschreibt, auf welche Weise durch die Domäne gegebene Welten verändert werden können. Die Voraussetzungen für diese Manipulation sind durch die *Vorbedingung* des Schemas gegeben.

Definition 2.3 ***Operatorenschema***
Sei $\mathcal{P}$ eine Menge von Prädikatenschemata.
Ein ***Operatorenschema*** *ist ein 4-Tupel $\sigma = \langle \text{na}, \text{par}, \text{pre}(\text{par}), \text{eff}(\text{par})\rangle$ mit den Komponenten:*

- na *ist der* ***Name*** *des Operatorenschemas (über einem endlichen Alphabet Σ).*
- $\text{par} = \{\text{par}_1, \ldots, \text{par}_n\}$ *ist die endliche Menge an* ***Parametern***.
- $\text{pre}(\text{par}) = \phi$ *ist die* ***Vorbedingung*** *des Operatorenschemas, eine aussagenlogische Formel ϕ mit Prädikatensymbolen $\rho \in \mathcal{P}$ und Variablensymbolen* $\text{par}_1, \ldots, \text{par}_n \in \text{par}$.
- $\text{eff}(\text{par}) = \phi$ *ist der* ***Effekt*** *des Operatorenschemas, eine Effektkonjunktion ϕ mit Prädikatensymbolen $\rho \in \mathcal{P}$ und Variablensymbolen* $\text{par}_1, \ldots, \text{par}_n \in \text{par}$.

Im Folgenden bezeichne

- $\text{pre}(\sigma)$ *die Vorbedingung eines Operators oder Operatorenschemas σ.*
- $\text{eff}(\sigma)$ *den Effekt eines Operators oder Operatorenschemas σ.*

Die beiden wichtigsten Werkzeuge zur Modellierung von *Planungsaufgaben* auf einer Domäne sind durch die Definitionen der Prädikaten- und Operatorenschemata beschrieben. Diese zeigen die Möglichkeiten auf, welche Eigenschaften auf Objekte einer Welt zutreffen können und wie mit diesen agiert werden kann. Üblicherweise ist die Definition einer Domäne mit diesen beiden Komponenten bereits komplett, und auch die im nächsten Abschnitt vorgestellte Beschreibungssprache PDDL liefert im Bereich *klassischen Planens* in frühen Standards keine weiteren Informationen. Tatsächlich ist es aber so, dass Planungsaufgaben immer auch eingeschränkt werden, und bestimmte Voraussetzungen, die nicht aus den Schemata hervorgehen, zusätzlich erfüllt sein müssen. Aus diesem Grund wird eine Domänenbeschreibung in dieser Arbeit zusätzlich eine Menge von *Planungsaufgabenbeschränkungen* enthalten, über welche weitere, allen Planungsaufgaben gemeine Informationen zugänglich gemacht werden. Diese werden auch Aufgaben von in späteren PDDL-Standards verwendete Konzepten wie die *Typisierung* von Parametern übernehmen und sind detailliert im folgenden Kapitel beschrieben. Allgemein wird mit ihrer Hilfe sichergestellt, dass alle auf der Domäne modellierbaren Probleme dem beabsichtigten Zweck entsprechen.

Definition 2.4 ***Planungsaufgabenbeschränkung***
Sei $\mathcal{P}$ eine Menge von Prädikatenschemata.
Eine ***Planungsaufgabenbeschränkung*** *ist ein 2-Tupel $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star\rangle$ mit den Komponenten:*

- $\mathcal{C}_0$ *ist die endliche Menge der* ***Initialzustandsbeschränkungen***.
- $\mathcal{C}_\star$ *ist die endliche Menge der* ***Zielzustandsbeschränkungen***.

Eine ***Zustandsbeschränkung*** *$c \in \mathcal{C}_0 \cup \mathcal{C}_\star$ ist eine geschlossene prädikatenlogische Formel mit Prädikatensymbolen $\rho \in \mathcal{P}$ und Variablensymbolen $\omega_1, \ldots, \omega_k$.*

Der letzte Bestandteil einer Domäne ist die *Kostenfunktion* der Operatorenschemata, welche ein Maß dafür angibt, welche Kosten durch die *Anwendung* einer Aktion entstehen. Auch diese wird nicht immer als Teil einer Domäne angesehen, da dadurch die Modellierung gleichartiger Operatoren mit von den beteiligten Objekten abhängigen Kosten nicht möglich ist. Diese Einschränkung wird aber in Kauf genommen, da in allen für diese Arbeit relevanten Domänen *Einheitskosten* vorliegen, die Kosten der Anwendung eines Operators also immer gleich eins betragen.

Definition 2.5 ***Planungsdomäne***
*Eine **Planungsdomäne** ist ein 4-Tupel $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \overline{w} \rangle$ mit den Komponenten:*

- $\mathcal{P}$ *ist die endliche Menge der **Prädikatenschemata**.*
- $\mathcal{O}$ *ist die endliche Menge der **Operatorenschemata** mit Prädikatenschemata aus $\mathcal{P}$.*
- $\mathcal{C}$ *ist die **Planungsaufgabenbeschränkung** mit Prädikatenschemata aus $\mathcal{P}$.*
- $\overline{w} : \mathcal{O} \to \mathbb{N}_0$ *ist die **Kostenfunktion** der Operatorenschemata.*
  *Entspricht diese einer konstanten Funktion mit $\overline{w}(\sigma) = 1$ für alle Operatorenschemata $\sigma \in \mathcal{O}$ wird von **Einheitskosten** gesprochen.*

Eine solche Domäne bildet die Grundlage einer Vielzahl ähnlicher Welten. Eine Bestimmte ergibt sich aber erst durch die zusätzliche Angabe eines Planungsproblems, welches insbesondere die für die Konkretisierung der Schemata benötigten Objekte liefert. Diese kann aber nicht völlig beliebig gegeben sein: Sowohl die *Initial-* als auch die *Zielzustandsbeschreibung*, deren Form in Definition 2.7 konkretisiert wird, müssen den in der Domäne gegebenen Planungsaufgabenbeschränkungen entsprechen.

Definition 2.6 ***Planungsaufgabe** oder **Planungsproblem***
*Eine **Planungsaufgabe** oder ein **Planungsproblem** ist ein 4-Tupel $\mathcal{T} = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star \rangle$ mit den Komponenten:*

- $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \overline{w} \rangle$ *mit $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ ist die **Domäne**, die der Planungsaufgabe zugrunde liegt.*
- $\Omega = \{\omega_1, \ldots, \omega_n\}$ *ist die endliche Menge der **Objekte**.*
- $\widehat{s}_0$ *ist die **Initialzustandsbeschreibung**, eine Zustandsbeschreibung (s.a. Def. 2.7) der Form, dass für alle $c_0 \in \mathcal{C}_0$ gilt: $\widehat{s}_0 \models c_0$.*
- $\widehat{s}_\star$ *ist die **Zielzustandsbeschreibung**, eine partielle Zustandsbeschreibung (s.a. Def. 2.7) der Form, dass für alle $c_\star \in \mathcal{C}_\star$ gilt: $\widehat{s}_\star \models c_\star$.*

Bei der angesprochenen Konkretisierung, im Folgenden *Instantiierung* genannt, werden die Parameter der Operatorenschemata sowie die durch die Stelligkeit induzierten Parameter der Prädikatenschemata durch Objekte ersetzt, wodurch die *Grundinstanzen* als Mengen der *Prädikate* und der *Operatoren* entstehen:

- Durch Einsetzen der Objekte $\omega \in \Omega$ in die Prädikatenschemata $\mathcal{P}$ der Domäne $\mathcal{D}$ entsteht die Menge der **Prädikate** P.
  Ein Prädikat $p \in P$, das durch Einsetzen von Objekten $\omega_1, \ldots, \omega_{\text{ar}}$ in das Prädikatenschema $\rho = \langle \text{na}, \text{ar} \rangle$ entsteht, sei auch dargestellt als $\text{na}(\omega_1, \ldots, \omega_{\text{ar}})$ oder $p(\omega_1, \ldots, \omega_{\text{ar}})$.
- Durch Einsetzen der Objekte $\omega \in \Omega$ in die Parameter der Operatorenschemata $\mathcal{O}$ der Domäne $\mathcal{D}$ sowie der zu den Objekten zugehörigen Prädikate $p \in P$ in die in Vorbedingung und Effekt vorkommenden Prädikatenschemata entsteht die Menge der **Operatoren** $\overline{O}$.
  Ein durch Einsetzen von $\omega_1, \ldots, \omega_{|\text{par}|}$ in das Operatorenschema $\sigma = \langle \text{na}, \text{par}, \text{pre}(\text{par}), \text{eff}(\text{par}) \rangle$ entstandener Operator $\overline{o} \in \overline{O}$ sei auch dargestellt als $\text{na}(\omega_1, \ldots, \omega_{|\text{par}|})$ oder $\overline{o}(\omega_1, \ldots, \omega_{|\text{par}|})$.

Erstellt werden durch diesen Vorgang immer alle möglichen Prädikate, es werden also alle Kombinationen der Objekte unter Beachtung der Reihenfolge in die Parameter der Prädikatenschemata eingesetzt. Eine triviale Instantiierung erstellt auch die Operatoren auf diese Art und Weise. Allerdings existieren auch Ansätze zur effizienteren Instantiierung, bei welchen nur der Teil aller möglichen Operatoren gebildet wird, der nicht offensichtlich niemals angewendet werden kann.
Das Grundgerüst einer derart instantiierten Welt bilden die Prädikate. Eine Belegung aller Prädikate einer Planungsaufgabe heißt *totale*, eine Belegung eines Teiles der Prädikate *partielle Zustandsbeschreibung*.

Definition 2.7 ***Zustandsbeschreibung***
Sei $\mathcal{T} = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star \rangle$ eine Planungsaufgabe, P die durch Instantiierung von $\mathcal{T}$ erhaltene Menge aller Prädikate p und $P' \subseteq P$.
*Eine **totale Zustandsbeschreibung** $\widehat{s}$ auf einer instantiierten Planungsaufgabe $\mathcal{T}$ ist eine vollständige Konjunktion mit allen Prädikatensymbolen $p \in P'$ und Variablensymbolen $\omega \in \Omega$, falls $P' = P$. Gilt $P' \neq P$, heißt $\widehat{s}$ **partielle Zustandsbeschreibung**.*
*Die Menge $\widehat{S}(\widehat{s})$ sei die Menge aller totalen Zustandsbeschreibungen $\widehat{s}_t$ auf $\mathcal{T}$, für die gilt $\widehat{s}_t \models \widehat{s}$ und heiße **Menge der von $\widehat{s}$ beschriebenen Zustände**.*
Ist $\widehat{s}$ total, gilt $\widehat{S}(\widehat{s}) = \{\widehat{s}\}$.
*$\widehat{S}(\top)$ heißt **Menge der Zustände**. Diese wird dargestellt durch $\widehat{S}$ und es gilt $\widehat{S} \cong 2^P$.*

Eine totale Zustandsbeschreibung gibt einen *Zustand* an, in welchem sich die Welt und darin vorkommende Objekte befinden können. Zu Beginn einer Planungsaufgabe ist dieser gegeben durch die *Initialzustandsbeschreibung*. Das gewünschte Ziel beschreibt dagegen die *Zielzustandsbeschreibung*, welche auch *partiell* sein kann und damit in jedem Zustand $\widehat{s}$ aus der *Menge der von $\widehat{s}_\star$ beschriebenen Zustände* $\widehat{S}(\widehat{s}_\star)$ als erreicht gilt.
Die Veränderung eines Zustandes ist durch die Anwendung eines Operators möglich: Gilt dessen Vorbedingung in einem Zustand, wird allen, im Effekt vorkommenden Prädikaten der entsprechende Wert zugewiesen.

Definition 2.8 ***Operatoranwendung***
Sei $\overline{o} = \langle \mathrm{na}, \{\omega_1, \ldots, \omega_n\}, \mathrm{pre}(\{\omega_1, \ldots, \omega_n\}), \mathrm{eff}(\{\omega_1, \ldots, \omega_n\}) \rangle$ ein Operator, $\widehat{s}$ eine totale Zustandsbeschreibung.
*Der Operator $\overline{o}$ ist **anwendbar** im Zustand $\widehat{s}$ wenn gilt:*

$$\widehat{s} \models \mathrm{pre}(\{\omega_1, \ldots, \omega_n\})$$

*Die dann mögliche **Operatoranwendung** von $\overline{o}$ in $\widehat{s}$ bewirkt einen Übergang von $\widehat{s}$ in den **Nachfolgerzustand** $\overline{o}(\widehat{s})$. In dieser ebenfalls totalen Zustandsbeschreibung behalten alle Prädikate, die im Effekt des Operators nicht enthalten sind, den in $\widehat{s}$ gegebenen Wert. Diese bilden die Menge der **Rahmenprädikate** $\mathrm{Sy}_{\pm}^{\mathrm{Frame}}$:*

$$\mathrm{Sy}_{\pm}^{\mathrm{Frame}} = \{\langle p, x \rangle \in \mathrm{Sy}_{\pm}(\widehat{s}) \mid p \notin \mathrm{Sy}(\mathrm{eff})\}$$

Alle Übrigen werden auf den im Effekt gegebenen Wert gesetzt, womit $\overline{o}(\widehat{s})$ gegeben ist als:

$$\overline{o}(\widehat{s}) = \bigwedge_{\langle p,1\rangle \in \mathrm{Sy}_{\pm}^{\mathrm{Frame}}} p \wedge \bigwedge_{\langle p,0\rangle \in \mathrm{Sy}_{\pm}^{\mathrm{Frame}}} \neg p \wedge \bigwedge_{\langle p,1\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff})} p \wedge \bigwedge_{\langle p,0\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff})} \neg p$$

*$\widehat{O}$ sei die Menge der **Operatoren**, die für alle $\overline{o} \in \overline{O}$ und alle $\widehat{s}$, in denen $\overline{o}$ anwendbar ist, einen Operator $\widehat{o} = \langle \widehat{s}, \overline{o}(\widehat{s}) \rangle$ enthält. Analog werden in der **Kostenfunktion** der Operatoren $\widehat{w}$ jedem so entstandenen Operator die Kosten aus $\overline{w}$ zugewiesen.*

Da die so definierte Menge der Operatoren $\widehat{O}$ nur eine andere Betrachtung von $\overline{O}$ darstellt, wurden bewusst unterschiedlich Symbole aber eine für beide identische Bezeichnung gewählt. Ermöglicht wird durch $\widehat{O}$ insbesondere eine elegante Definition des *vollständigen Zustandsraumes* eines Planungsproblems, welcher ebenfalls einen anderen Blickwinkel auf Komponenten darstellt, die in der Domäne oder der Planungsaufgabe bereits gegeben oder durch diese impliziert werden. Die Definition eines solchen Raumes eröffnet die Möglichkeit, diesen als Graph darzustellen, dessen Knotenmenge der Zustandsmenge entspricht und dessen Kantenmenge durch die Menge der Operatoren gegeben ist. Dieser Graph stellt ein wichtiges Werkzeug in der Handlungsplanung dar, das häufig zur Planfindung genutzt wird und *Zustandsübergangsgraph* heißt.

Definition 2.9 ***Vollständiger Zustandsraum*** *und* ***Zustandsübergangsgraph***
Sei $\mathcal{T} = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star \rangle$ *eine Planungsaufgabe auf der Domäne* $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \overline{w} \rangle$ *und P die durch Instanziierung erhaltene, endliche Menge der Prädikate.*
Der ***vollständige Zustandsraum*** *von* $\mathcal{T}$ *ist gegeben durch ein 5-Tupel* $\widehat{\mathcal{S}}(\mathcal{T}) = \langle \widehat{S}, \widehat{s}_0, \widehat{S}_\star, O, w \rangle$ *bestehend aus:*

- $\widehat{S}$ *ist die endliche* ***Zustandsmenge****.*
- $\widehat{s}_0 \in \widehat{S}$ *ist die* ***Initialzustandsbeschreibung****.*
- $\widehat{S}_\star \subseteq \widehat{S}$ *ist die Menge der* ***Zielzustände****, gegeben durch* $\widehat{S}(\widehat{s}_\star)$*.*
- $\widehat{O}$ *ist die endliche Menge der* ***Operatoren*** $\widehat{o} \in \widehat{S} \times \widehat{S}$*.*
- $\widehat{w} : \widehat{O} \to \mathbb{N}_0$ *ist die* ***Kostenfunktion*** *der Operatoren.*

Durch den vollständigen Zustandsraum wird ein gewichteter, gerichteter Graph mit Knotenmenge $\widehat{S}$ *und mit* $\widehat{w}$ *gewichteten Kanten* $\widehat{O}$ *induziert, der* ***Zustandsübergangsgraph*** $\widehat{G}(\mathcal{T}) = \langle \widehat{S}, \widehat{O}, \widehat{w} \rangle$*.*

Leider sind Zustände eines derart gegebenen vollständigen Zustandsraumes nur sehr schwer darzustellen, da jeder aus einer Belegung aller Prädikate besteht. Zusätzlich beinhaltet die Zustandsmenge alle theoretisch möglichen Zustände, ohne dass diese auch tatsächlich vom Initialzustand aus erreichbar sein müssen. Dabei lässt sich allein aufgrund der Domänenbeschreibung bereits einiges über die *Erreichbarkeit* von Zuständen sowie die *Aussagekraft* der Prädikatenschemata aussagen. Damit lässt sich die Menge der Prädikatenschemata für jede Planungsaufgabe derselben Domäne in zwei Teile aufteilen, nämlich in für Zustände *relevante* auf der einen sowie *konstante* auf der anderen Seite. Letztere werden in der *Menge der Grundelemente* repräsentiert, während erstere weiterhin zur Beschreibung von *erreichbaren* Zuständen genutzt werden. Diese werden dabei im *relevanten Zustandsraum* nicht weiter durch die Polarität aller instantiierten Prädikate angegeben, sonder effizienter durch generierte Mengen oder Funktionen beschrieben, ohne dass dadurch erreichbare Zustände nicht mehr unterschieden werden können. Kapitel 3 wird dazu eine Methode vorstellen, eine Beschreibung von diesen automatisch und nur auf Basis einer zu 2.5 analogen Domänenbeschreibung zu generieren.

Definition 2.10 ***Relevanter Zustandsraum***
Sei $\mathcal{T}$ *eine Planungsaufgabe auf der Domäne* $\mathcal{D}$ *mit vollständigem Zustandsraum* $\widehat{\mathcal{S}}(\mathcal{T})$*.*
Die ***Menge der Grundelemente*** *einer Planungsaufgabe* $\mathcal{T}$ *ist ein beliebigstellige Tupel und sei bezeichnet als* $\mathcal{B}(\mathcal{T})$*.*
Der ***relevante Zustandsraum*** *der Planungsaufgabe* $\mathcal{T}$ *ist gegeben durch ein 5-Tupel* $\mathcal{S}(\mathcal{T}) = \langle S, s_0, S_\star, O, w \rangle$ *mit den Komponenten:*

- S *ist die aus n-Tupeln bestehende* ***Menge relevanter Zustände****. Jede Komponente eines n-Tupels heißt* ***Zustandsbeschreibungsvariable****.*
- $s_0 \in S$ *ist der* ***Initialzustand****.*
- $S_\star \subseteq S$ *ist die Menge der* ***Zielzustände****..*
- O *ist die endliche Menge der* ***Operatoren*** $o : S \to S$*.*
- $w : O \to \mathbb{N}_0$ *ist die* ***Kostenfunktion*** *der Operatoren.*

Jeder Zustand $s \in S$ *impliziert mit der Menge der Grundelemente* $\mathcal{B}$ *den entsprechenden Zustand* $\widehat{s} \in \widehat{S}$*.*
Auch durch den relevanten Zustandsraum wird ein gewichteter, gerichteter Graph mit Knotenmenge S *und mit* w *gewichteten Kanten* O *induziert, der* ***Zustandsübergangsgraph*** $G(\mathcal{T}) = \langle S, O, w \rangle$*.*

Damit sind die grundlegendsten Konzepte definiert und seien im Folgenden am Beispiel der für diese Arbeit entworfenen SimpleRoute-Domäne verdeutlicht. Planungsaufgaben auf dieser Domäne sollen Welten modellieren, in welchen eine Person über miteinander verbundene Orte bewegt werden kann. Das Ziel besteht dabei immer daraus, zuerst einen beliebigen Gegenstand einzusammeln und dann den Zielort zu erreichen.

Beispiel Teil A ***Domäne***
Die **SimpleRoute-*Domäne*** $\mathcal{D}_{\mathrm{SR}} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *ist beschrieben durch:*

- $\mathcal{P} = \{\rho_1, \ldots, \rho_6\}$*:*
  - $\rho_1 = \langle \texttt{location}, 1 \rangle$
  - $\rho_2 = \langle \texttt{person}, 1 \rangle$
  - $\rho_3 = \langle \texttt{at}, 2 \rangle$
  - $\rho_4 = \langle \texttt{itemAt}, 1 \rangle$
  - $\rho_5 = \langle \texttt{pickedUp}, 1 \rangle$
  - $\rho_6 = \langle \texttt{connected}, 2 \rangle$
- $\mathcal{O} = \{\sigma_1, \sigma_2\}$*:*
  - $\sigma_1 = \langle \texttt{move}, \{p, l_1, l_2\}, \mathrm{pre}(p, l_1, l_2), \mathrm{eff}(p, l_1, l_2) \rangle$*, wobei*
    $\mathrm{pre}(p, l_1, l_2) = \texttt{connected}(l_1, l_2) \wedge \texttt{at}(p, l_1)$ *und*
    $\mathrm{eff}(p, l_1, l_2) = \texttt{at}(p, l_2) \wedge \neg \texttt{at}(p, l_1)$
  - $\sigma_2 = \langle \texttt{pickUp}, \{p, l\}, \mathrm{pre}(p, l), \mathrm{eff}(p, l) \rangle$*, wobei*
    $\mathrm{pre}(p, l) = \texttt{at}(p, l) \wedge \texttt{itemAt}(l)$ *und*
    $\mathrm{eff}(p, l) = \texttt{pickedUp}(p)$
- $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ *mit* $\mathcal{C}_0 = \{c_0^1, \ldots, c_0^9\}$ *und* $\mathcal{C}_\star = \{c_\star^1, c_\star^2\}$*:*
  - $c_0^1 = \forall\ \omega\ (\texttt{location}(\omega) \vee \texttt{person}(\omega))$
  - $c_0^2 = \forall\ \omega\ (\neg(\texttt{location}(\omega) \wedge \texttt{person}(\omega)))$
  - $c_0^3 = \forall\ p, l\ (\texttt{at}(p, l) \Rightarrow (\texttt{person}(p) \wedge \texttt{location}(l)))$
  - $c_0^4 = \forall\ l\ (\texttt{itemAt}(l) \Rightarrow \texttt{location}(l))$
  - $c_0^5 = \forall\ p\ (\texttt{pickedUp}(p) \Rightarrow \texttt{person}(p))$
  - $c_0^6 = \forall\ l_1, l_2\ (\texttt{connected}(l_1, l_2) \Rightarrow (\texttt{location}(l_1) \wedge \texttt{location}(l_2)))$
  - $c_0^7 = \exists^{=1}\ p\ \texttt{person}(p)$
  - $c_0^8 = \forall\ p\ (\texttt{person}(p) \rightarrow\ \exists^{=1}\ l\ (\texttt{location}(l) \wedge \texttt{at}(p, l)))$
  - $c_0^9 = \forall\ p\ (\neg \texttt{pickedUp}(p))$
  - $c_\star^1 = \forall\ p\ (\texttt{person}(p) \rightarrow\ \exists^{=1}\ l\ (\texttt{location}(l) \wedge \texttt{at}(p, l)))$
  - $c_\star^2 = \forall\ p\ (\texttt{person}(p) \Rightarrow\ \texttt{pickedUp}(p))$
- $\widehat{w}(\sigma_1) = \widehat{w}(\sigma_2) = 1$

Diese Beschreibung der Domäne ist sehr viel ausführlicher als notwendig und insbesondere das Prädikatenschema $\texttt{person}(p)$ dadurch eigentlich unnötig, dass in den Beschränkungen gefordert wird, dass nur ein Objekt existieren darf, für welches das Prädikat gilt. Es sollte aber ein möglichst einfaches Beispiel gefunden werden, das aber dennoch so viele vorgestellte Konzepte wie möglich beinhaltet, die dann damit verdeutlicht werden können.
Folgendermaßen sollten bei der Modellierung von Planungsaufgaben auf der SimpleRoute-Domäne die gegebenen Prädikatenschemata interpretiert werden, damit die instantiierten Operatoren den gewünschten, ebenfalls beschriebenen Effekt haben:

- `location`(l) gelte für ein Objekt l, falls es sich dabei um einen Ort handelt.
- `person`(p) gelte für ein Objekt p, falls es sich dabei um eine Person handelt.
- `at`(p, l) gelte, falls es sich dabei um den Ort l handelt, an dem sich die Person p befindet.
- `itemAt`(l) gelte für ein Objekt l, falls es sich dabei um einen Ort handelt, an dem sich ein Gegenstand befindet.
- `pickedUp`(p) gelte, sobald die Person p ein Gegenstand aufgesammelt hat.
- `connected`(l_1, l_2) gelte für zwei Orte l_1 und l_2, sofern l_1 mit l_2 verbunden ist.
- Durch `move`(p, l_1, l_2) ist es möglich, die Person p von l_1 zu l_2 zu bewegen, wenn l_1 der Ort ist, an welchem sie sich befindet, und l_1 mit l_2 verbunden ist.
- Durch `pickUp`(l) wird ein Gegenstand aufgesammelt, wenn sich die Person sowie ein Gegenstand bei l_1 befinden.

Beim Gebrauch des zweistelligen Prädikats `connected` ist bei der Formulierung von Planungsaufgaben besondere Vorsicht geboten: Dieses ist nicht symmetrisch definiert, es gilt also nur das erste in den Parametern vorkommende Objekt als mit dem zweiten verbunden, nicht aber umgekehrt. Natürlich existieren auch Möglichkeiten, die Symmetrie des Prädikates zu erzwingen oder zumindest Operatoren so zu verändern, dass die Asymmetrie keine Auswirkungen hat:

- Durch Ersetzen des Konjunktionsgliedes `connected`(l_1, l_2) in der Vorbedingung des Operatorenschemas `move`(p, l_1, l_2) durch (`connected`(l_1, l_2) $\vee$ `connected`(l_2, l_1)) hat die Asymmetrie keine Auswirkungen mehr.
- Durch die Erweiterung von $\mathcal{C}_0$ um $c_0 = \forall\ l_1, l_2$ `connected`(l_1, l_2) $\Leftrightarrow$ `connected`(l_2, l_1) ist das Prädikat symmetrisch.

In der SimpleRoute-Domäne ist die Asymmetrie aber gewünscht – wie im Übrigen auch, dass der Operator `pickUp`(p, l) den aufgesammelten Gegenstand nicht von seinem Ort entfernt, indem `itemAt`(l) ungültig gemacht wird. Es zeigt sich also, dass die Modellierung von Domänen nicht nur in der Wahl der Namen von Schemata nicht eindeutig ist, sondern zumeist viele Wege existieren, das gewünschte Ziel zu erreichen.

Beispiel Teil B ***Relevanter Zustandsraum der* SimpleRoute-*Domäne***
Der relevante Zustandsraum $S(\mathcal{T}_{\text{SR}})$ beliebiger Planungsaufgaben $\mathcal{T}_{\text{SR}}$ auf der SimpleRoute-*Domäne $\mathcal{D}_{\text{SR}}$ ist gegeben als:*

Grundelemente: *$\langle G, l_\star, L_i \rangle$, wobei $G = \langle L, E \rangle$ der **Wegenetzgraph** mit der Menge der **Orte** L und den **Straßen** E ist, $l_\star$ der **Zielort** der Person und $L_i \subseteq L$ die Menge der Orte, an welchen ein Gegenstand liegt.*

Zustände: *2-Tupel $\langle l, \mathrm{x} \rangle$, wobei $l \in L$ der Ort ist, an dem die Person sich befindet und $\mathrm{x} \in \{0, 1\}$ angibt, ob bereits ein Gegenstand aufgesammelt wurde.*

Startzustand: $\langle l_0, 0 \rangle$

Zielzustand: *Jeder Zustand $\langle l, 1 \rangle$ mit $l = l_\star$*

Operatoren: *Die Person kann sich von einem Ort l_1 zu einem anderen Ort l_2 **bewegen**, falls l_1 mit l_2 verbunden ist, sowie einen Gegenstand **aufnehmen**, falls sich die Person und ein solcher an demselben Ort befinden.*

Die Menge der Grundelemente besteht aus einem gerichteten Graphen, der als *Wegenetzgraph* bezeichnet wird und beschreibt, welche Orte miteinander verbunden sind, dem Zielort der Person sowie den Orten, an welchen Gegenstände vorhanden sind. Der Graph repräsentiert mit der Menge der Orte sowie den Straßen die beiden Prädikatenschemata `location`(l) und `connection`(l_1, l_2), während die Teilmenge der Orte L_i das Prädikat `itemAt`(l) beschreibt. Fehlend ist dagegen eine Menge, die das Prädikatenschema `person`(p) beinhaltet. Wie bereits angesprochen wurde, wird dieser Umstand im nächsten Kapitel beschrieben werden. Die Beschreibung von Zuständen zeigt dagegen deutlich, wie viel prägnanter diese im relevanten Zustandsraum im Vergleich zum vollständigen Zustandsraum ist. Während in letzterem alle Prädikate dafür benötigt werden, kommt ersterer damit aus, lediglich ein Objekt zur Repräsentierung von `at`(p, l) sowie einen Wert x anstelle von `pickedUp`(p) anzugeben. Dennoch können für alle Planungsaufgaben noch immer alle relevanten Zustände unterschieden werden – letztlich kommt es in der SimpleRoute-Domäne nur auf den Ort an, an dem sich die Person befindet sowie darauf, ob bereits ein Gegenstand aufgesammelt wurde oder nicht. Allein aufgrund dieser Informationen lässt sich die Belegung aller Prädikate beliebiger Planungsaufgaben in jedem beliebigen, erreichbaren Zustand implizieren.
Zusätzlich sei auf eine in dieser Arbeit verwendete Konvention hingewiesen: Trägt ein definiertes Bezeichnungssymbol, wie in diesem Fall l als der Ort, an welchem die Person sich befindet, den Index l_0 bzw. $l_\star$, so ist immer das entsprechende, im Initialzustand bzw. einem Endzustand geltende beschrieben. Dabei ist es auch unerheblich, ob es sich um ein Objekt wie in diesem Fall oder aber eine Menge oder Funktion handelt.

Die nur durch die Domänenbeschreibung gegebenen Konzepte sind damit alle an der SimpleRoute-Domäne verdeutlicht. Um auch die restlichen Definitionen zu verbildlichen ist zusätzlich die Angabe einer Planungsaufgabe nötig.

Beispiel Teil C ***Planungsaufgabe***
Eine **SimpleRoute-Planungsaufgabe** $\mathcal{T}^1_{\mathrm{SR}} = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star \rangle$ sei gegeben durch:

- $\mathcal{D} = \mathcal{D}_{\mathrm{SR}}$
- $\Omega = \{\text{home}, \text{uni}, \text{store}, \text{gym}, \text{robert}\}$
- $\widehat{s}_0 =$ `location`(home)$\wedge$`location`(uni)$\wedge$`location`(store)$\wedge$`location`(gym)$\wedge$`person`(robert)$\wedge$ `connected`(home, uni)$\wedge$connected(uni, home)$\wedge$connected(gym, uni)$\wedge$`connected`(uni, gym)$\wedge$ `connected`(home, store)$\wedge$`connected`(gym, store)$\wedge$`connected`(store, gym)$\wedge$`at`(robert,home)$\wedge$ `itemAt`(uni) $\wedge$ `itemAt`(store)
- $\widehat{s}_\star =$ `pickedUp`(robert) $\wedge$ `at`(robert,gym)

In diesem Fall existieren die vier Objekte home, uni, store und gym, für die alle `location` gilt, sowie das in der Initialzustandsbeschränkung auf genau ein Vorkommen beschränkte Objekt robert, für das `person` gilt. Wie in diesem Beispiel wird auch im Folgenden der Initialzustand so gegeben sein, dass alle geltenden Prädikate explizit aufgeführt sind und alle fehlenden dementsprechend nicht gelten, es wird also von der *Closed-World-Annahme* ausgegangen. Ein genauer Blick zeigt schnell, dass diese, wie auch die Zielzustandsbeschreibung, allen in $\mathcal{D}_{\mathrm{SR}}$ gegebenen Planungsaufgabenbeschränkungen entspricht.
Zwischen den Orten home und store ist die bereits angesprochene Asymmetrie von `connected`(l_1, l_2) eingebaut. Gut zu sehen ist dies auf der Abbildung des zu dieser Planungsaufgabe gehörenden Wegenetzgraphen in Teil D des Beispiels.

Beispiel Teil D ***Wegenetzgraph***

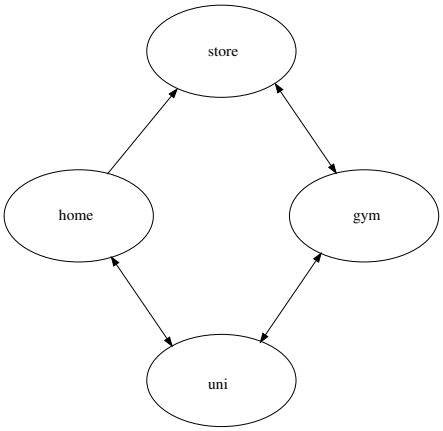

Abbildung 2.1: Wegenetzgraph von $\mathcal{T}^1_{\mathrm{SR}}$

Der Wegenetzgraph verdeutlicht gut, welche Wege den Teil der Aufgabe lösen, robert von home nach gym zu bewegen. Nicht ablesbar ist dagegen, dass zusätzlich einer der beiden Gegenstände bei store oder uni eingesammelt werden müssen. Der Zustandsübergangsgraph beinhaltet zusätzlich auch diese Informationen. Ein Blick auf diesen zeigt auch, warum viele Planungssysteme auf ihm aufbauen – in dieser Form fällt es leicht, die Aktionen abzulesen, durch die der gewünschte Zielzustand erreicht werden kann. Erwähnt sei noch, dass der Umstand, dass Operatoren im Folgenden durch einen Parameter weniger als in der Domäne beschrieben werden, im nächsten Kapitel erläutert wird.

Beispiel Teil E ***Zustandsübergangsgraph***

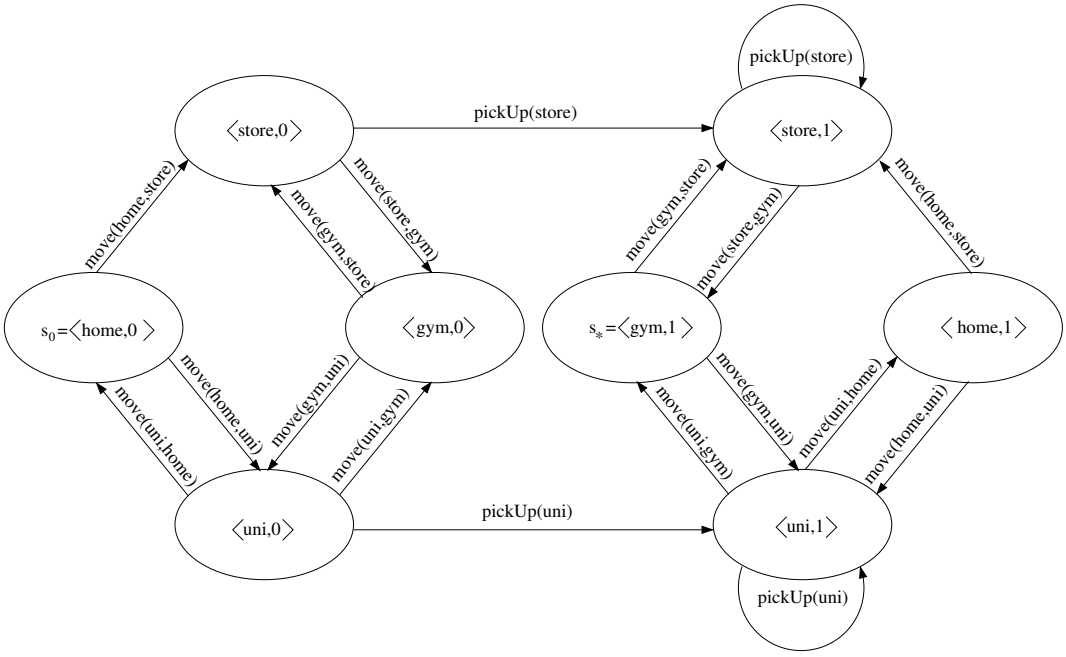

Abbildung 2.2: Zustandsübergangsgraph von $\mathcal{T}^1_{\mathrm{SR}}$

Damit sind die bislang vorgestellten Konzepte anhand des Beispiels der SimpleRoute-Domäne verdeutlicht, und die Grundlagen für einen tieferen Einblick in die Handlungsplanung geschaffen, was mit der Vorstellung des Internationalen Planungswettbewerbs im nächsten Abschnitt beginnen wird.

2.2 Internationaler Planungswettbewerb

Als standarddefinierend im Bereich des Planens gilt heute der Internationale Planungswettbewerb IPC, der erstmals im Jahre 1998 stattfand [HE05, HET+06]. Bei diesem alle zwei Jahre abgehaltenen Wettbewerb treten die modernsten Planungssysteme in verschiedenen Kategorien gegeneinander an. Traditionell wird dabei zwischen *klassischem Planen* sowie *Planen unter Unsicherheit* unterschieden, wobei bei ersterem Pläne aus deterministischen Aktionen in voll beobachtbaren Domänen gesucht werden, während letzteres auch nichtdeterministische Aktionen und nur teilweise oder gar nicht beobachtbare Domänen erlaubt. Ebenso wenig wie mit dem dritten, im Jahre 2008 neu eingeführten Teil des *Planens kombiniert mit Lernen* beschäftigt sich diese Arbeit mit Planen unter Unsicherheit. Alles Nachfolgende bezieht sich also auf das Teilgebiet des klassischen Planens. Die folgenden Aussagen treffen damit auf alle in dieser Arbeit betrachteten Domänen zu:

- Durch die volle Beobachtbarkeit ist dem Planer der aktuelle Zustand s zu jedem Zeitpunkt bekannt.
- Der aktuelle Zustand s kann sich ohne die Ausführung einer Aktion o durch den Planer nicht ändern.
- Da alle Aktionen deterministisch sind, kann der Nachfolgezustand $o(s)$, der durch Ausführung des Operators o im Zustand s erreicht wird, sicher vorhergesagt werden.

Diese Aussagen erlauben es einem Planer, durch entsprechende Berechnungen einen Pfad im Zustandsübergangsgraphen zu finden, der vom Initialzustand aus sicher in einen Endzustand führt, wenn ein solcher erreichbar ist. Die auf den Kanten eines solchen Pfades stehenden Operatoren $o \in O$ ergeben einen Plan π.

Definition 2.11 ***Plan***
Sei $\mathcal{D}$ eine Domäne und $\mathcal{T}$ eine dazugehörige Planungsaufgabe.
*Eine Folge von Aktionen $o_1, \ldots, o_n$ ist dann ein **Plan** für $\mathcal{T}$, wenn die Anwendung der Operatoren nacheinander, zuerst auf den Initialzustand s_0 und dann auf den jeweiligen Nachfolgerzustand $o_i(s)$ in einem Zielzustand $s_\star \in S_\star$ endet, also wenn gilt $o_n(o_{n-1}(\ldots o_1(s_o)\ldots)) \in S_\star$. Π bezeichne die Menge aller Pläne für $\mathcal{T}$.*
*Die **Kosten** oder **Güte** eines solchen Planes sind $m(\pi) = \sum_{i=1}^{n} w(o_i)$.*
*Ein Plan $\pi^\star$ heißt **optimal**, falls kein anderer Plan π für $\mathcal{T}$ existiert mit $m(\pi) < m(\pi^\star)$. Die Menge optimaler Pläne sei $\Pi^\star$.*
*Existiert für eine Planungsaufgabe $\mathcal{T}$ mindestens ein Plan π heißt diese **lösbar**.*

Definition 2.11 zieht einige einfache, für die Belange dieser Arbeit aber nicht unerhebliche Implikationen nach sich:

- Für eine Planungsaufgabe $\mathcal{T}$ existiert mindestens ein optimaler Plan $\pi^\star$ genau dann, wenn auch ein (beliebiger) Plan π existiert, genau dann, wenn $\mathcal{T}$ lösbar ist.
- Ist eine Planungsaufgabe $\mathcal{T}$ durch n optimale Pläne $\pi_i^\star \in \Pi^\star$ lösbar, gilt $m(\pi_1^\star) = m(\pi_2^\star) = \cdots = m(\pi_n^\star)$. Dieses allen optimalen Plänen gemeine Gütemaß, die **Minimalkosten**, sei im Folgenden dargestellt durch $m^\star$.

Zusammenfassend ist eine Planungsaufgabe also entweder gar nicht lösbar oder es existiert mindestens ein optimaler Plan. Diese Eigenschaft spiegelt sich auch in der weiteren Unterteilung des IPC in zwei Zweige

wieder: Im optimalen Zweig zählen nur Pläne mit *Minimalkosten* als eine Aufgabe erfüllend, während im suboptimalen beliebige Pläne akzeptiert werden. Gemein ist aber beiden Zweigen, dass alle gestellten Planungsaufgaben auch lösbar sind.
Diese Eigenschaften wirken sich auch auf die in dieser Arbeit entwickelten Planer aus: Da alle Planungsprobleme des IPC lösbar sind, existiert auch immer mindestens ein Plan mit Minimalkosten. Da sich diese Arbeit mit der Entwicklung *optimaler Planer* beschäftigt, ist damit die Existenz mindestens eines solchen Plans auf allen Aufgaben gesichert und das zugehörige Problem damit auch immer entscheidbar – es wurde also bei keinem entwickelten Planer darauf geachtet, inwieweit die verwendeten Algorithmen bei nicht lösbaren Planungsaufgaben terminieren. Des weiteren werden im Folgenden, sofern nicht anders angegeben, alle Aussagen auf lösbare Probleme bezogen sein.

Damit kann nun auch das Konzept von Plänen auf die in Teil C des Beispiels gegebene Planungsaufgabe angewandt werden.

Beispiel Teil F ***Plan***
Es existieren zwei optimale Pläne $\pi_1^\star$ *und* $\pi_2^\star$ *für die* SimpleRoute*-Planungsaufgabe* $\mathcal{T}_{\text{SR}}^1$ *auf* $\mathcal{D}_{\text{SR}}$*. Zusätzlich sind noch zwei suboptimale Pläne* π_3 *und* π_4 *angegeben:*

- $\pi_1^\star$: $move(home, uni), pickUp(uni), move(uni, gym)$ mit $m(\pi_1^\star) = 3$
- $\pi_2^\star$: $move(home, store), pickUp(store), move(store, gym)$ mit $m(\pi_2^\star) = 3$
- π_3 : $move(home, store), pickUp(store), pickUp(store), move(store, gym)$ mit $m(\pi_3) = 4$
- π_4 : $move(home, uni), move(uni, gym), move(gym, store), pickUp(store), move(store, gym)$ mit $m(\pi_4) = 5$

Die Minimalkosten von $\mathcal{T}_{\text{SR}}^1$ *betragen* $m^\star = 3$.

Dass $\pi_1^\star$ und $\pi_2^\star$ tatsächlich optimale Pläne sind, verdeutlicht ein Blick auf den Zustandsübergangsgraphen $G(\mathcal{T}_{\text{SR}}^1)$: Ein Weg von kürzerer Länge, was unter Einheitskosten dem Gütemaß entspricht, als $m^\star = 3$, vom Initialzustand $s_0 = \langle home, 0\rangle$ zum einzigen erreichbaren Zielzustand $s_\star = \langle gym, 1\rangle$ existiert nicht. Außerdem zeigt sich, dass $\pi_1^\star$ und $\pi_2^\star$ die einzigen beiden optimalen Pläne sind. Eine solche Einschränkung ist, wie bei den meisten Planungsaufgaben, für suboptimale Pläne nicht zu machen – es gibt unendlich viele, wenn nur ein einziger Zykel im Zustandsübergangsgraphen existiert, der vom Initialzustand aus erreichbar ist und von welchem aus wiederum ein Zielzustand erreicht werden kann.

Damit sind die Unterschiede der verschiedenen IPC-Wettbewerbe ausreichend diskutiert und es wird Zeit, sich der wichtigsten Gemeinsamkeit zuzuwenden: Es ist den Teilnehmern vor Beginn des Wettbewerbs nicht bekannt, auf welchen Domänen ihre Planer zum Einsatz kommen werden[1]. Damit muss jeder teilnehmende Planer in der Lage sein, alle lösbaren Planungsaufgaben aller möglichen Domänen und damit das *Allgemeine Planungsproblem* zu berechnen:

Definition 2.12 ***Allgemeines Planungsproblem***
Sei $\mathcal{D}^\infty$ *die* ***Klasse aller Planungsdomänen****.*
Das ***Allgemeine Planungsproblem*** Plan-$\mathcal{D}^\infty$ *ist gegeben durch:*

Eingabe: *Eine lösbare Planungsaufgabe* $\mathcal{T} = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star\rangle$ *auf einer beliebigen Domäne* $\mathcal{D} \in \mathcal{D}^\infty$.

Lösung: *Ein Plan* π *für* $\mathcal{T}$*. Das Problem, das* Plan-$\mathcal{D}^\infty$ *auf Lösungen* $\pi^\star$ *für* $\mathcal{T}$ *mit* $m(\pi^\star) = m^\star$ *beschränkt heißt* ***Allgemeines Planungsoptimierungsproblem*** OptPlan-$\mathcal{D}^\infty$.

[1] Beginnend mit IPC 6, meiner Meinung nach aber unabdingbare Voraussetzung.

Ein Algorithmus, der das Allgemeine Planungsproblem löst, und damit auch alle Aufgaben im suboptimalen Teil klassischer Planung des IPC, heißt *domänenunabhängiger Planer*. Berechnet ein solcher gar das A*allgemeine Planungsoptimierungsproblem*, ist der Planer zusätzlich optimal und somit sogar in der Lage, am optimalen Zweig klassischen Planens teilzunehmen. Diese Arbeit beschäftigt sich dagegen mit *domänenspezifischen Planern*, die nur Planungsaufgaben einer einzigen Domäne und damit das *domänenspezifische Planungsproblem* lösen.

Definition 2.13 ***Domänenspezifisches Planungsproblem***
Sei $\mathcal{D} \in \mathcal{D}^\infty$ eine Domäne.
*Das **Domänenspezifische Planungsproblem** PLAN-$\mathcal{D}$ der Domäne $\mathcal{D}$ ist gegeben durch:*

EINGABE: *Eine lösbare Planungsaufgabe $\mathcal{T} = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star \rangle$ auf $\mathcal{D}$.*

LÖSUNG: *Ein Plan π für $\mathcal{T}$. Das Problem, das PLAN-$\mathcal{D}$ auf Lösungen $\pi^\star$ für $\mathcal{T}$ mit $m(\pi^\star) = m^\star$ beschränkt heißt **Domänenspezifisches Planungsoptimierungproblem** OPTPLAN-$\mathcal{D}$ der Domäne $\mathcal{D}$.*

Damit kann nun auch eine Definition von Planern, inklusive derer Eigenschaften, angegeben werden:

Definition 2.14 ***Planer***
*Ein **domänenunabhängiger Planer** Ψ^∞ löst das allgemeine Planungsproblem PLAN-$\mathcal{D}^\infty$.*
*Ein **domänenspezifischer Planer** $\Psi^\mathcal{D}$ für eine Domäne $\mathcal{D}$ löst das domänenspezifische Planungsproblem PLAN-$\mathcal{D}$ der Domäne $\mathcal{D}$.*
*Ein Planer heißt zusätzlich **optimal** (Ψ_{opt}), wenn er das entsprechende Planungsoptimierungproblem löst.*

An dieser Stelle stellt sich die Frage, wozu domänenspezifische oder suboptimale Planer überhaupt benötigt werden – schließlich beinhaltet das Allgemeine Planungsoptimierungproblem alle anderen in 2.12 und 2.13 beschriebenen Probleme. Eine Antwort darauf lässt sich aus den in Tabelle 2.3 dargestellten Ergebnissen des IPC 5 (2006) herleiten:

Planer / Domäne	optimaler Zweig			suboptimaler Zweig	
	Maxplan	**SATPLAN**	**Mips-BDD**	**SGPlan.5**	**Downward**
TPP	15/30	24/30	8/30	30/30	30/30
OPENSTACKS	5/30	0/30	8/30	30/30	26/30
PATHWAYS	15/30	9/30	4/30	30/30	30/30
PIPESWORLD	8/50	16/50	7/50	30/50	23/50
ROVERS	18/40	14/40	7/40	40/40	39/40
STORAGE	18/30	15/30	14/30	30/30	18/30
TRUCKS	9/30	5/30	6/30	28/30	14/30
$\sum$	88/240	83/240	54/240	218/240	200/240

Abbildung 2.3: Ergebnisse des IPC 5 (in gelösten von gestellten Planungsaufgaben $\mathcal{T}$)

Zwar ist es tatsächlich so, dass alle teilnehmenden Systeme prinzipiell alle Planungsaufgaben ihres Zweiges lösen können, allerdings sind dafür benötigte Berechnungen bei der IPC zeitlich begrenzt. Dargestellt ist also jeweils die Anzahl von Aufgaben, die in der geforderten Zeit von 30 Minuten gelöst wurden. Des weiteren sind nur die in ihrem Zweig erfolgreichsten Planer aufgelistet. Dabei zeigen sich signifikante Unterschiede: Während der Gewinner des suboptimalen Teils, SGPlan 5 [HWHC06], mit 22 weniger als 10% der 240 Aufgaben nicht in der geforderten Zeit lösen konnte, fanden selbst die beiden erfolgreichsten optimalen Planer SATPLAN [HGS07] sowie dessen Variante Maxplan nur für etwas mehr als ein Drittel der Aufgaben einen Plan mit Minimalkosten. Außerdem konnte kein Teilnehmer des optimalen Zweiges alle

Planungsaufgaben einer einzigen Domäne lösen, womit keiner der Planer die für diese Arbeit entwickelten Planer gleichwertig ersetzen könnte.

Damit sei der Blick auf die unterschiedlichen Wettbewerbe des IPC abgeschlossen und auf den Aspekt gerichtet, welcher zu den einschneidendsten Veränderungen in der Handlungsplanung führte: Bis zur Einführung des IPC war es nicht möglich, die Leistungsfähigkeit verschiedener Planer zu vergleichen. Geändert wurde dies insbesondere durch die Definition eines einheitlichen Standards zur Beschreibung von Domänen und zugehörigen Planungsaufgaben in Form der Planning Domain Definition Language PDDL. Diese dem Wissensaustauschformat KIF [GF92] ähnliche Sprache wurde für die erste IPC von McDermott und anderen entworfen [MGH+98] und seitdem von Fox und Long [FL03, FL06], Edelkamp und Hoffmann [EH04a, EH04b] und schließlich von Gerevini und Long [GL05, GL06] bis zum gegenwärtigen Stand weiterentwickelt. Heute stellt sie den Standard zur Beschreibung von Planungsdomänen und zugehörigen Problemen dar und erlaubt es, dass verschiedene Planungssysteme einfach miteinander verglichen werden können. Anfangs wurde sie dabei stark vom STRIPS-Formalismus beeinflusst [FN71], der heute allerdings nur noch eine Teilmenge der durch PDDL beschreibbaren Probleme bildet.
Eine PDDL-Definition ist dabei analog zu den Definitionen 2.5 und 2.6 in eine Domänenbeschreibung auf der einen sowie Aufgabenbeschreibungen auf der anderen Seite unterteilt. Der PDDL-Code der Domäne beginnt mit ihrem Namen, gefolgt von einer schematischen Beschreibung der Prädikate und der Operatoren, kann jedoch je nach verwendetem PDDL-Typ zusätzliche Informationen wie Konstanten, Objekttypen oder von Einheitskosten abweichende Kostenfunktionen enthalten. Die in dieser Arbeit vorkommenden PDDL-Beispiele kommen aber ohne diese zusätzlichen Informationen aus, weshalb hier nicht näher darauf eingegangen werden soll. Die SimpleRoute-Domäne könnte in PDDL dementsprechend folgendermaßen aussehen:

Beispiel Teil G ***PDDL-Domänenbeschreibung***

```
(define (domain SimpleRoute)
(:predicates   (location ?loc)
               (at ?loc)
               (itemAt ?loc)
               (pickedUp)
               (connected ?loc1 ?loc2))
(:action move
:parameters    (?loc1 ?loc2)
:precondition  (and (location ?loc1) (location ?loc2)
               (connected ?loc1 ?loc2) (at ?loc1))
:effect        (and (not (at ?loc1)) (at ?loc2)))
(:action pickUp
:parameters    (?loc)
:precondition  (and (location ?loc) (at ?loc) (itemAt ?loc))
:effect        (pickedUp)))
```

Die einzige, in der PDDL-Domänenbeschreibung fehlende Komponente von Definition 2.5 sind die Beschränkungen, welche auf dem IPC in einer zusätzlichen, textuellen Domänenbeschreibung angegeben werden. Da die Domäne und die zugehörigen Aufgaben von derselben Personengruppe entworfen werden, entsprechen die Probleme natürlich immer den gewünschten Beschränkungen. Zusätzliche aus den Beschränkungen gewonnene Informationen, wie sie im nächsten Kapitel beschrieben werden, werden derzeit von Planern wie zum Beispiel Fast Downward [Hel06] durch eine spezielle Analyse jeder einzelnen Planungsaufgabe extrahiert oder sind, wie zum Beispiel Typenbeschreibungen, in fortschrittlicheren PDDL-Standards implementiert.
Auch die PDDL-Problembeschreibung beginnt mit einem Namen, gefolgt von demjenigen der zugehörigen

Domäne. Der Angabe vorhandener Objekte folgen schließlich Initial- und Zielzustand. Die im vorangehenden Abschnitt vorgestellte SimpleRoute-Planungsaufgabe wird durch den nachfolgenden PDDL-Code beschrieben:

Beispiel Teil H ***PDDL Problembeschreibung***

```
(define    (problem SimpleRoute-01)
(:domain   SimpleRoute)
(:objects  home uni store gym robert)
(:init     (location home)
           (location uni)
           (location store)
           (location gym)
           (person robert)
           (connected home uni)
           (connected uni home)
           (connected home store)
           (connected store gym)
           (connected gym store)
           (connected uni gym)
           (connected gym uni)
           (at robert home)
           (itemAt uni)
           (itemAt store))
(:goal     (and (at robert gym) (pickedUp robert))))
```

Auch die PDDL-Problembeschreibung ist weitgehend wie zu erwarten. Einzig erwähnenswert ist, dass wie bereits in der Aufgabendefinition in Teil C des Beispiels auch bei der PDDL Beschreibung des Initialzustandes von der Closed-World-Annahme ausgegangen wird.

Die Auswirkungen der Gründung des IPC sind somit dargestellt und PDDL als Werkzeug der Handlungsplanung hinreichend erläutert. Wie aufgezeigt ist optimales, domänenunabhängiges Planen zwar möglich, die Laufzeiten entsprechender Algorithmen genügen praktischen Anforderungen aber zumeist nicht. Daher befasst sich der nächste Abschnitt mit der Komplexität der vorgestellten Planungsprobleme.

2.3 Komplexität

Basis der in diesem Kapitel gegebenen Komplexitätsklassen ist immer die Größe der instantiierten Planungsaufgabe. Zudem werden in dieser Arbeit zusätzlich zu den gängigen Komplexitätsklassen P, NP und PSPACE [GJ79] die weniger bekannten *Approximationsklassen* verwendet. Um einen kurzen Überblick über deren Bedeutung zu erlangen, ist es zuerst notwendig, das *Leistungsverhältnis* eines Planes zu definieren.

Definition 2.15 ***Leistungsverhältnis***
Sei $\mathcal{D}$ eine Domäne und $\mathcal{T}$ eine lösbare Planungsaufgabe auf $\mathcal{D}$.
Für einen beliebigen Plan $\pi \in \Pi$ für $\mathcal{T}$ sei das ***Leistungsverhältnis*** *definiert als $\mathcal{L}(\pi) = \frac{m(\pi)}{m^\star}$, wobei $\frac{0}{0} = 1$ und $\frac{k}{0} = \infty$ für $k \neq 0$.*

Das Leistungsverhältnis eines Planes ist ein Maß für dessen Güte im Vergleich zu den Minimalkosten der Planungsaufgabe. Damit gibt sich die *Approximierbarkeit* einer Domäne wie folgt:

Definition 2.16 ***Approximierbarkeit***
Sei $\mathcal{D}$ eine Domäne, $\mathcal{T}_\mathcal{D}$ die Menge aller lösbaren Planungsaufgaben auf $\mathcal{D}$, $f : \mathbb{N}_0 \to \mathbb{R}^+$ eine Funktion und $c \geq 1$, $c \in \mathbb{R}^+$ eine Zahl.
*$\mathcal{D}$ heißt **f-approximierbar**, wenn ein deterministischer, polynomieller Planer Ψ existiert, für dessen Lösung π von* PLAN-$\mathcal{D}$ *für alle $\mathcal{T} \in \mathcal{T}_\mathcal{D}$ gilt: $\mathcal{L}(\pi) \leq f(|\mathcal{T}|)$, wobei $|\mathcal{T}|$ die Größe der instantiierten Planungsaufgabe bezeichne.*
*$\mathcal{D}$ heißt **c-approximierbar**, wenn $\mathcal{D}$ f-approximierbar ist für $f(n) = c$ für alle $n \in \mathbb{N}_0$.*
Ist $\mathcal{D}$ 1-approximierbar, so ist Ψ_{opt} ein optimaler Planer.

Eine Domäne ist also f-approximierbar, wenn ein deterministischer, polynomiell berechenbarer Algorithmus existiert, der für alle erfüllbaren Probleme Pläne ermittelt, deren Leistungsverhältnis nach oben durch eine Funktion f, die nur von der Eingabegröße abhängt, beschränkt ist. Damit können nun die von Ausiello und anderen beschriebenen [APMS+99] Approximationsklassen folgendermaßen definiert werden:

Definition 2.17 ***Approximationsklassen***
*Eine Domäne $\mathcal{D}$ liegt in einer der **Approximationsklassen*** PO, PTAS, APX, poly-APX, exp-APX, NPO, NPS *und* EXPO *wenn gilt:*

- *für* PO *(P-optimierbar):*
  alle $\mathcal{T}$ auf $\mathcal{D}$ sind durch einen deterministischen, polynomiell berechenbaren Algorithmus optimal lösbar.
- *für* PTAS *(polynomielles Approximations-Schema):*
  $\mathcal{D}$ ist c-approximierbar für alle reellen Zahlen $c > 1$.
- *für* APX *(approximierbar):*
  $\mathcal{D}$ ist c-approximierbar für eine reelle Zahl $c > 1$.
- *für* poly-APX *(poly-approximierbar):*
  $\mathcal{D}$ ist f-approximierbar für eine polynomielle Funktion f.
- *für* exp-APX *(exp-approximierbar):*
  $\mathcal{D}$ ist f-approximierbar für eine exponentielle Funktion f.
- *für* NPO *(NP-optimierbar):*
  alle $\mathcal{T}$ auf $\mathcal{D}$ sind durch einen nichtdeterministischen, polynomiell berechenbaren Algorithmus optimal lösbar.
- *für* NPS *(NP-lösbar):*
  alle $\mathcal{T}$ auf $\mathcal{D}$ sind durch einen nichtdeterministischen, polynomiell berechenbaren Algorithmus lösbar.
- *für* EXPO *(EXP-optimierbar):*
  alle $\mathcal{T}$ auf $\mathcal{D}$ sind durch einen exponentiell berechenbaren Algorithmus optimal lösbar.

Helmert erweitert diese Übersicht zwischen exp-APX und NPO um die Klasse PS [Hel08], in welcher Algorithmen liegen, die zwar polynomiell approximierbar sind, deren Plan aber so lang ist dass allein die Ausgabe mehr als polynomielle Zeit braucht. In dieser Arbeit wird PS lediglich als Sammelklasse aller polynomiell approximierbaren Klassen verwendet, was zu keinem Problem führt, da keine jemals bei der IPC verwendete Domäne im Zweig klassischen Planens der Klasse PS \ exp-APX angehört. Komplett verzichtet wird zudem auf eine Definition der Klasse FPTAS, der ebenfalls keine IPC-Domäne angehört. Die Beziehung der Approximationsklassen zueinander ist offensichtlich: Ist P ≠ NP, so gilt PO $\subsetneq$ PTAS $\subsetneq$ APX $\subsetneq$ poly-APX $\subsetneq$ NPS sowie PO $\subsetneq$ NPO $\subsetneq$ NPS. Unabhängig des Problems, ob P und NP identisch sind, gilt

zudem NPO $\subsetneq$ EXPO sowie NPS $\subsetneq$ NPO.

Komplexitätsuntersuchungen wie zum Beispiel von Helmert [Hel08] zeigen, dass für den IPC entwickelte Domänen über das gesamte Spektrum dieser Klassen verteilt liegen. Es existieren also sowohl solche, die in PO liegen, als auch solche, für die Pläne nur durch exponentielle Algorithmen ermittelt werden können und die somit der Komplexitätsklasse EXPO \ NPS angehören. Die Domänen, die in dieser Arbeit besondere Aufmerksamkeit finden, liegen zumeist in PS \ PTAS. Die Unterscheidung aller in diesem Bereich liegenden Klassen basiert lediglich auf der Güte des ermittelten Planes – mal ist dieser nur um einen konstanten Faktor zu lang (in APX \ PTAS), mal existiert gar keine Möglichkeit, die Länge zum optimalen Plan in Beziehung zu setzen (in PS \ exp-APX). Gemein ist aber allen, dass ein approximierender und damit suboptimaler Plan durch polynomielle Algorithmen generiert werden kann, sowie, dass der optimale Plan nicht auf diese Weise berechnet werden kann – dann läge die entsprechende Domäne schließlich in PO. Dasselbe trifft, für Laufzeiten schlechter als polynomiell, auf Domänen aus NPS \ NPO zu. Zusammenfassend kann also gesagt werden, dass domänenspezifisch große Unterschiede in der Komplexität der Probleme existieren sowie, dass suboptimale Pläne in für diese Arbeit relevanten Domänen auch komplexitätstheoretisch einfacher berechnet werden können als optimale.

Selbstverständlich lässt sich diese Aussage nicht auf domänenunabhängiges Planen übertragen, da sie lediglich besagt, dass für spezifische Domänen derart schnelle Algorithmen existieren, nicht jedoch, inwieweit diese auf beliebige Probleme übertragbar sind. Tatsächlich kann, falls P $\neq$ NP, ein derartiger Approximationsalgorithmus für domänenunabhängiges Planen nicht existieren: Wie beschrieben gibt es Domänen, für die bereits bewiesen wurde, dass sie in EXPO \ NPS liegen, und domänenunabhängiges Planen kann dementsprechend nicht einfacher sein. Für Domänen dieser Klasse existieren aber keine Algorithmen, die suboptimale Pläne komplexitätstheoretisch schneller ermitteln als optimale, was damit auch für domänenunabhängiges Planen im Allgemeinen gelten muss. Dass das Problem auch nicht schwerer ist, lässt sich durch einen Algorithmus beweisen, der es in exponentieller Zeit löst. Dafür sei auf die Breitensuche über dem Zustandsübergangsgraphen beliebiger Planungsprobleme hingewiesen, welche diesen Raum durchsucht, dessen Größe exponentiell in der Anzahl der Prädikate ist. Mit den Aussagen dieses Abschnitts kann also Satz 2.1 gefolgert werden.

Satz 2.1 ***Komplexität des Planens***
Domänenunabhängiges Planen benötigt exponentielle Zeit, also PLAN-$\mathcal{D}^\infty \in$ *EXPO \ NPS.*
Domänenunabhängiges, optimales Planen benötigt exponentielle Zeit, also OPTPLAN-$\mathcal{D}^\infty \in$ *EXPO \ NPS.*
Domänenabhängiges Planen benötigt Laufzeiten abhängig von der Domäne, kann im Einzelfall aber ebenfalls exponentielle Laufzeiten benötigen. Für eine beliebige Domäne $\mathcal{D}$ gilt also PLAN-$\mathcal{D} \in$ *EXPO sowie* OPTPLAN-$\mathcal{D} \in$ *EXPO.*

Auch wenn komplexitätstheoretisch keine Unterschiede bestehen, ist suboptimales Planen in der Praxis meistens schneller zu bewerkstelligen als das optimale Pendant (vgl. [Hel03]) – als Beispiel sei hier die *Tiefensuche* erwähnt, die häufig schneller einen Plan findet, sowie auf die *heuristische Suche* verwiesen, die, zur optimalen Planfindung eingesetzt, strengeren Beschränkungen unterliegt und damit zumeist auch komplexere Berechnungen erfordert.

Die Einführung in die Grundlagen der Handlungsplanung ist damit abgeschlossen. In diesem Kapitel wurden die grundlegendsten Konzepte definiert und um Planungsaufgabenbeschränkungen erweitert, der Internationale Planungswettbewerb sowie die dafür entwickelte Sprache PDDL vorgestellt und letztlich ein Überblick über die Komplexität verschiedener Probleme gegeben. Der nächste Abschnitt analysiert unterschiedliche Arten von Zustandsbeschränkungen und untersucht insbesondere deren Auswirkungen auf den relevanten Zustandsraum.

3 Zustandsbeschränkungen und Zustandsbeschreibungsvariablen

Im vorigen Abschnitt wurde der Begriff des relevanten Zustandsraumes definiert und an einem Beispiel vorgestellt, bislang ausgelassen wurde aber dessen Herleitung aus der Domänenbeschreibung. Ermöglicht wird dieser Vorgang insbesondere durch eine Analyse der Initialzustandsbeschränkungen, welche verschiedenen Kategorien zugeordnet werden können. Nach einer Klärung der zu diesem Kapitel führenden Motive werden verschieden Arten von Beschränkungen im zweiten Abschnitt beschrieben, gefolgt von einigen Erkenntnissen über Eigenschaften von Domänen, die deren Modellierung betreffen. Abgeschlossen wird das Kapitel dann mit der Vorstellung eines Algorithmus zur Generierung des relevanten Zustandsraumes von Domänen.

3.1 Motivation

Wenn in Veröffentlichungen der Handlungsplanung Domänen vorgestellt werden, wird häufig zum besseren Verständnis von Planungsaufgaben auf dieser Domäne ein Zustandsraum mit angegeben. Allerdings ist dieser dann normalerweise nicht wie in Definition 2.9 gegeben, sondern entspricht vielmehr dem, was in dieser Arbeit als relevanter Zustandsraum definiert wurde – Zustände werden also nicht durch die Polarität aller Prädikate angegeben, sondern Funktionen und Mengen definiert, die deren Belegung implizieren. Eine formale Herleitung dieser *Zustandsbeschreibungsvariablen* allein aus einer gegebenen Domäne, also ohne konkrete Planungsaufgaben zu analysieren, wird dabei aber nicht mitgeliefert.
Tatsächlich ist es sogar so, dass eine solche Herleitung aus üblichen, formalen Domänendefinitionen oder auch einer PDDL-Domänenbeschreibung gar nicht möglich ist. In beiden Fällen liegt das daran, dass wichtige Informationen über die Domäne aus einer zusätzlichen Beschreibung gewonnen werden müssen. Diese liegt dabei lediglich in natürlichsprachlicher Form vor, es existiert aber keine formalisierte Komponente, welche die entsprechenden Aussagen enthält.
Natürlich ist es allerdings so, dass alle in den Planungsaufgabenbeschränkungen enthaltenen Informationen für jede einzelne Planungsaufgabe einer Domäne extrahiert werden können, und zumindest Teile davon von einigen Planern wie zum Beispiel Helmerts Fast Downward auch extrahiert werden [Hel06]. Aus Sicht der für das Lösen einzelner Planungsaufgaben benötigten Rechenleistung, wie es auf dem IPC der Fall ist, ergeben sich durch die vorgeschlagene formale Erweiterung von Domänen auch keine Vorteile: Da für jede Aufgabe die Planer ohne Wissen über bereits geleistetes neu gestartet werden, wird die Analyse weiterhin für jede einzelne Aufgabe durchgeführt werden müssen. Zudem ist eine solche Analyse verglichen mit der Komplexität der meisten Probleme trivial. Vorteile in Bezug auf benötigte Rechenoperationen ergeben sich also erst bei Onlinesystemen, welchen im Allgemeinen aber keine Informationen über Domänen oder vergleichbare Konzepte vorliegen.
Es liegen aber dennoch gute Gründe für eine Erweiterung von Domänen um diese Informationen vor: Der offensichtlichste liegt dabei darin, dass diese Informationen Teil der Domäne und nicht jedes einzelnen Planungsproblems sind. Zu erwähnen, dass in der SimpleRoute-Domäne in den Initialzuständen aller Planungsaufgaben das Prädikat $\texttt{at}(p, l)$ für jede Person nur für genau einen Ort gelten darf, sollte nicht Aufgabe einer natürlichsprachlichen Beschreibung sein. Auch der momentan gängige Usus, diese Informationen aus einzelnen Planungsaufgaben zu gewinnen, erscheint eine Arbeit, die nicht für jede einzelne Planungsaufgabe einer Domäne mit identischen Ergebnissen wiederholt werden sollte – diese Informationen stellen wichtige Eigenschaften der Domäne dar, und sollten dementsprechend auch formal darin beschrieben sein. Ähnliches

gilt für die Ziele einer Domäne – auch diese haben in jeder Planungsaufgabe große Gemeinsamkeiten, und sollten Planern in maschinenlesbarer Form beschrieben werden. Dann kann auch eine andere Person als der Modellierer einer Domäne selbst problemlos beliebige Planungsaufgaben für eine Domäne entwerfen, ohne darauf hoffen zu müssen, den beabsichtigten Sinn korrekt aus einer Mischung von natürlichsprachlicher und formaler Domänenbeschreibung zu erkennen. Mehr noch ist dann sogar die maschinelle Modellierung von Planungsaufgaben auf Basis der Domänenbeschreibung möglich. Das wichtigste Argument für die Belange dieser Arbeit liegt aber darin, dass dann der relevante Zustandsraum beziehungsweise eine Beschreibung von diesem formal aus Domänen, ohne zusätzliche Angabe einer Planungsaufgabe, generiert werden kann.

Es sei abschließend noch erwähnt, dass für dieses Kapitel nicht nur kein Anspruch auf Vollständigkeit erhoben werden kann, sondern sogar eingestanden werden muss, dass das beschriebene Thema sicher nicht abschließend analysiert werden konnte. Teilweise werden Ideen angesprochen, ohne diese formal zu definieren, und teilweise sind Konzepte definiert, ohne diese vollständig zu beschreiben. Eine vollständige Definition hätte den Rahmen dieser Arbeit aber bei Weitem gesprengt.

3.2 Arten von Beschränkungen

Das Ziel dieses Kapitels besteht letztendlich darin, eine Methode zu formalisieren, mit deren Hilfe der relevante Zustandsraum beliebiger Planungsdomänen generiert werden kann. Dieser hat gegenüber dem vollständigen Zustandsraum einige Vorteile, wovon ein wichtiger eher pragmatischer Natur ist: Es ist einfach unmöglich, Zustände durch Aufzählung der Polarität aller existierenden Prädikate übersichtlich und leicht verständlich zu beschreiben, und selbst unter der Closed-World-Annahme sind es derer noch immer zu viele. Zusätzlich existieren aber tatsächlich auch Vorteile für den Planungsvorgang: Eine nicht-triviale Instantiierung wird erheblich erleichtert, da sinnvolle Operatoren und Prädikate einfacher als solche zu erkennen sind, wodurch Speicherplatz gespart werden kann – dies erscheint zwar auf den ersten Blick vernachlässigbar, da aber viele Planer unzählig viele mögliche Zustände explorieren und zum späteren Vergleich speichern, kann diese Ersparnis sehr schnell sehr wichtig werden. Zusätzlich sind häufig verwendete Techniken wie zum Beispiel Heuristiken auf Basis des relevanten Zustandsraumes oft sehr viel schneller zu berechnen.
Des Weiteren entpuppt sich ein vermeintlicher Nachteil schnell als Vorteil: Durch die prägnantere Darstellung von Zuständen, die weniger Informationen enthält, können natürlich weniger Zustände dargestellt und damit auch unterschieden werden. Diese fehlenden Informationen können aber mithilfe der Menge der Grundelemente in jedem erreichbaren Zustand reproduziert werden. Zudem würde auch der Mensch diese Prädikate nicht jedes Mal aufzählen – eine in der SimpleRoute-Domäne nicht in jedem Zustand gegebene Information ist die, dass das Objekt robert vom Typ `person`(p) ist, dass also `person`(*robert*) gilt sowie, dass `location`(*robert*) nicht gilt. Damit können Zustände, in welchen robert vom Typ Ort ist nicht mehr dargestellt werden, jedem Menschen ist aber sofort klar, dass solche Zustände keine Relevanz haben. Daher ist es auch für eine Maschine von Vorteil, derartige, offensichtlich niemals erreichbare Zustände gar nicht erst untersuchen zu können. Damit kann die Generierung effizienter Zustandsbeschreibungsvariablen sowie das Aufteilen der Informationen in konstante und veränderliche Prädikate mit einer, wenn auch trivialen, Erreichbarkeitsanalyse verglichen werden.
Nicht alle im Folgenden beschriebenen Methoden sind aber nur aufgrund der neu eingeführten Beschränkungen zu erreichen. Bereits die Analyse von Prädikaten- und Operatorenschemata führt zu erheblich kleineren relevanten Zustandsräumen. Daher seien zuerst einige, aus diesen Schemata gewonnene Eigenschaften beschrieben.

Definition 3.1 ***Eigenschaften von Prädikatenschemata***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *eine Domäne und* $\sigma(\text{par}_1, \ldots, \text{par}_k) \in \mathcal{O}$ *ein beliebiges Operatorenschema.*
Ein Prädikatenschema $\rho \in \mathcal{P}$ *ist* ***aussagekräftig****, wenn ein Operatorenschema* $\sigma \in \mathcal{O}$ *existiert mit* $\rho \in \text{Sy}(\text{eff}(\sigma))$.
Ein Prädikatenschema $\rho \in \mathcal{P}$ *ist* ***anzahlkonstant****, wenn es aussagekräftig ist und zusätzlich für alle Operatorenschema* $\sigma \in \mathcal{O}$ *gilt, dass die Anzahl der* $\langle \rho, 0 \rangle \in \text{Sy}_{\pm}(\text{eff}(\sigma))$ *gleich der Anzahl der* $\langle \rho, 1 \rangle \in \text{Sy}_{\pm}(\text{eff}(\sigma))$ *ist.*
Ein Prädikatenschema $\rho \in \mathcal{P}$ *ist* ***dekrementell****, wenn es aussagekräftig ist aber nicht anzahlkonstant, und zusätzlich für alle Operatorenschema* $\sigma \in \mathcal{O}$ *gilt, dass die Anzahl der* $\langle \rho, 0 \rangle \in \text{Sy}_{\pm}(\text{eff}(\sigma))$ *größer oder gleich der Anzahl der* $\langle \rho, 1 \rangle \in \text{Sy}_{\pm}(\text{eff}(\sigma))$ *ist.*
Ein Prädikatenschema $\rho \in \mathcal{P}$ *ist* ***konstant****, wenn es nicht aussagekräftig ist.*

All diese Eigenschaften beschreiben die Veränderungen, die ein Prädikat durch Operatoranwendungen vollziehen könnte. Aus *konstanten* Schemata gebildete Prädikate werden dabei niemals ihren anfangs gegebenen Wert ändern, diese gelten also entweder bereits im Initialzustand und dann auch in allen erreichbaren Zuständen, oder sie gelten nie. Einstellige Schemata dieser Art werden meistens dazu eingesetzt, *Objekttypen* zu deklarieren, wie es in der SimpleRoute-Domäne auf `location`(l) und `person`(p) zutrifft. Mehrstellige, konstante Prädikate beschreiben dagegen häufig äußere, *beschränkende Umstände* und sind sehr häufig Teil der Vorbedingung von Operatorenschemata. Ein solches ist in unserem Beispiel durch `connected`(l_1, l_2) gegeben. Gemein ist aber allen konstanten Prädikatenschemata, dass sie nichts zur Unterscheidung relevanter Zustände beitragen, da ihr Wert in allen erreichbaren Zuständen demjenigen des Initialzustandes entpricht.
Alle Prädikatenschemata, auf die eine der anderen beschriebenen Eigenschaften zutrifft, sind *aussagekräftig* in Bezug auf die Unterscheidung von Zuständen, sie werden also im relevanten Zustandsraum in der ein oder anderen Form erhalten bleiben. Sonderfälle von diesen sind *anzahlkonstante* Schemata, welche besonders häufig unter den zweistelligen Prädikaten zu finden sind und oft Eigenschaften beschreiben, die für ein in den ersten Parameter eingesetztes Objekt immer in Verbindung mit exakt einem anderem Objekt gilt. Diese Prädikate werden im sich momentan in der Entwicklung befindlichen, neuesten PDDL-Standard besonders häufig durch *objektwertige Funktionen* ersetzt werden. Im letzten Abschnitt dieses Kapitels wird beschrieben werden, wie dies auch mithilfe von Initialzustandsbeschränkungen für das Prädikat `at`(p, l) möglich ist.
Auch *dekrementelle* Prädikatenschemata sind immer aussagekräftig. Sie werden in Domänen vor allem dazu benötigt, um aufzuzeigen, dass ein Objekt verbraucht wurde – sie bezeichnen also meistens *knappe Ressourcen*, deren sinnvolle Verwendung einen wichtigen Bestandteil der Planung ausmacht. Ein Beispiel dafür hätte problemlos in die SimpleRoute-Domäne eingebaut werden können: Der Makel, dass `pickUp`(p, l) an einem Ort mit einem Gegenstand beliebig oft angewendet werden kann, hätte dadurch behoben werden können, dass der Gegenstand nach der Anwendung des Operators nicht mehr dort ist. Dann würde `itemAt`(l) für den entsprechenden Ort nicht mehr gelten, und das Prädikatenschema wäre dekrementell. Zwar würde dies, da nur eine Person erlaubt ist, keine Auswirkungen auf optimale Pläne haben, müsste dieselbe Aufgabe aber für mehrere Objekte des Typs `person`(p) gelöst werden, wäre die Planung mit knapper Ressource erheblich erschwert. Diese einfache Variante der SimpleRoute-Domäne, die entsteht, indem die Initialzustandsbeschränkung c_0^7 weggelassen wird, heiße im Folgenden MultiPersonRoute-Domäne.
Die Implikationen, die dekrementelle Prädikatenschemata auf den relevanten Zustandsraum haben, konnten leider nicht vollständig analysiert werden. Diese geben sogar Grund zu der Annahme, dass die Grammatik zur Bildung von Initialzustandsbeschränkungen nicht ausreichend ist: Beschreibt ein Prädikat eine knappe Ressource, existiert häufig ein anderes Prädikat, das diese Ressource nutzen muss, um das Ziel zu erreichen. In der eben beschriebenen Abwandlung der SimpleRoute-Domäne wären also Gegenstände die Ressource und Personen diejenigen, die diese nutzen. Dann wäre aber auch für alle Planungsaufgaben bekannt, dass die Anzahl von Gegenständen mindestens der Anzahl von Personen entsprechen muss, um die Aufgabe überhaupt lösen zu können. Mit der im vorigen Kapitel beschriebenen Grammatik ist eine solche Beschränkung aber nicht möglich, Planer könnten aber dennoch Vorteile aus dieser Information ziehen.

Allgemein stellt sich damit aber die Frage, an welcher Stelle eine solche Beschränkung tatsächlich eine Eigenschaft der Domäne ist, und ab wann sie lediglich die Lösbarkeit von Planungsaufgaben beschreibt. Ein Beispiel, dass eine solche Grenze gezogen werden muss, zeigt die folgende Aussage: Um eine lösbare Planungsaufgabe in der SimpleRoute-Domäne zu generieren, muss ein Weg zwischen dem Startort der Person, einem Ort mit einem Gegenstand und dem Zielort existieren. Diese Aussage beschreibt aber, obwohl von der Art vergleichbar mit der Aussage über knappen Ressourcen, bereits die Lösung einer Planungsaufgabe – diese liegt schließlich vor allem in dem Weg – weswegen eine derartige Beschränkung vermutlich bereits zu viel Information über den Lösungsweg beinhaltet, während der Mengenvergleich zumindest einen Grenzfall darstellt. Auch aus diesem Grund wurde auf eine Änderung der Grammatik für Beschränkungen verzichtet, Bedarf für weitere Überlegungen in diese Richtung sehe ich aber durchaus gegeben. Dennoch sei an dieser Stelle auf Planer wie TLPlan [BK00] oder TALPlanner [DK01] verwiesen, denen manuell vergleichbares Domänenwissen bergeben wird.

Blickt man nun noch einmal auf konstante Prädikatenschemata zurück, stellt sich unweigerlich die Frage, welche Funktion diese überhaupt für eine Domäne haben. In unserem Beispiel kommen beide konstanten und einstelligen Prädikate zusätzlich auch in keiner Vorbedingung eines Operators vor, sie erscheinen also nutzlos. Ohne allzu sehr auf den folgenden Abschnitt vorgreifen zu wollen, handelt es sich bei diesen, sofern sie nicht tatsächlich nutzlos sind, um *Typenprädikate*, welche durch eine spezielle Initialzustandsbeschränkung einer *wohldefinierten* Domäne gegeben werden, der *Typenbeschreibung*. Diese hat in der SimpleRoute-Domäne die Form:

$$c_0^1 = \forall\ \omega\ (\texttt{location}(\omega) \vee \texttt{person}(\omega))$$

Mit dieser wird also sichergestellt, dass alle Objekte eine beliebigen Planungsaufgabe im Initialzustand einem der beiden Typen angehören. Eine Funktion ergibt sich für diese Prädikate aber erst durch die zusätzliche Angabe einer *Parametertypisierung*:

$$c_0^3 = \forall\ p, l\ (\texttt{at}(p, l) \Rightarrow (\texttt{person}(p) \wedge \texttt{location}(l)))$$

$$c_0^4 = \forall\ l\ (\texttt{itemAt}(l) \Rightarrow \texttt{location}(l))$$

$$c_0^5 = \forall\ p\ (\texttt{pickedUp}(p) \Rightarrow \texttt{person}(p))$$

$$c_0^6 = \forall\ l_1, l_2\ (\texttt{connected}(l_1, l_2) \Rightarrow (\texttt{location}(l_1) \wedge \texttt{location}(l_2)))$$

Auf diese Art wird allen Parametern von nicht in der Typenbeschreibung enthaltenen Prädikaten ein Typ zugewiesen. Im Beispiel wird also erwartet, dass das Prädikat $\texttt{at}(p, l)$ nur gelten kann, wenn es sich bei p um ein Objekt handelt, für das $\texttt{person}(p)$ gilt, und bei l um ein Objekt, für das $\texttt{location}(l)$ gilt. Damit werden zwei Ziele erreicht: Zum einen wird die Modellierung von Planungsaufgaben weiter vereinfacht, da die Bedeutung dieser *parametertypisierten* Prädikate durch die zusätzliche Information weiter hervorgehoben wird. Zum anderen werden aber auch Vorbedingungen von Operatoren auf das Wesentliche verkürzt – im Beispiel würde diese für das Operatorenschema $\texttt{move}(p, l_1, l_2)$ sonst folgendermaßen aussehen:

$$\text{pre}(p, l_1, l_2) = \texttt{connected}(l_1, l_2) \wedge \texttt{at}(p, l_1) \wedge \texttt{person}(p) \wedge \texttt{location}(l_1) \wedge \texttt{location}(l_2)$$

Die Aufgaben, die diese beiden Arten von Initialzustandsbeschränkungen erfüllen, sind in anderer Form auch in PDDL verfügbar. Ein Vorteil, Typen über Beschränkungen zu bestimmen wird aber im nächsten Abschnitt mit der Einführung numerischer Variablen vorgestellt. Es existiert aber auch ein Nachteil: Es muss bei der Modellierung von Operatoren darauf geachtet werden, dass Prädikate niemals für Objekte geltend gemacht werden, die diesen Parametertypisierungen widersprechen. Auch dies wird eine Eigenschaft wohldefinierter Domänen sein.

Zusammengefasst werden diese beiden Formen in der Kategorie der *Typisierungen*, der auch die *Typenexklusivitätsbeschreibungen* angehören:

$$c_0^2 = \forall\ \omega\ (\neg(\texttt{location}(\omega) \wedge \texttt{person}(\omega)))$$

Diese Art der Beschränkung beschreibt, ob zwei Objekttypen miteinander vereinbar sind. Häufig ist eine solche für alle Typenpaare gegeben, was dazu führt, dass die Menge der Objekte in paarweise disjunkte *Typenmengen* unterteilt werden kann. Selbstverständlich ist dies aber weder immer gegeben noch immer erwünscht.
Häufig existiert sogar eine Beschränkung die aussagt, dass für alle Objekte eines Typs auch ein anderer gelten muss. Eine solche Beschränkung heißt *Typenhierarchiebeschreibung* und ermöglicht es, weitere, komplexe Zusammenhänge zwischen Typenmengen zu beschreiben. Am ehesten vergleichbar ist dieses Konzept mit der Vererbung von Klassen in objektorientierten Programmiersprachen, deren Terminologie zur Beschreibung hierarchisch verbundener Typenmengen für diese Arbeit übernommen wird. Begriffe wie *Mehrfachvererbung* oder *mehrstufige Vererbung* werden also nicht explizit für aus Beschränkungen gewonnene Typenmengen definiert, sondern seien durch Anpassung entsprechender Konzepte der Vererbung in objektorientierten Programmiersprachen (s. z.B. [Bal98]) gegeben.
Auch parametertypisierte Prädikate können hierarchisch aufgebaut sein oder sich gegenseitig ausschließen. Vorstellbar wäre, übertragen auf die SimpleRoute-Domäne, zum Beispiel unterschiedliche Arten von Verbindungen zu erlauben, auf denen sich nur unterschiedliche *Subtypen* der `person`-Prädikates bewegen können oder auch unterschiedlichen Kosten dabei anfallen. Letzteres wurde für die Belange dieser Arbeit aber bereits ausgeschlossen und wird deshalb nicht weiter in Betracht gezogen werden.
Zusammenfassend ergeben sich die Beschränkungsformen, die sich mit dem Typ von Objekten befassen wie folgt:

Definition 3.2 ***Typisierungsbeschränkung***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *eine Domäne mit* $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$, $\rho_1^1(\text{par}), \ldots, \rho_n^1(\text{par}) \in \mathcal{P}$ *eine Menge einstelliger Prädikate und* $\rho(\text{par}_1, \ldots, \text{par}_k) \in \mathcal{P}$ *ein Prädikat.*
Eine Initialzustandsbeschränkung $c_0 \in \mathcal{C}^{Typ}$ *heiße* ***Typenbeschreibung*** *der Domäne* $\mathcal{D}$ *wenn sie darstellbar ist durch:*

$$c_0 = \forall\ \omega\ \bigvee_i \rho_i^1(\omega)$$

Einstellige Prädikate $\rho^1(\text{par})$*, die als Disjunktionsglied Teil der Typenbeschreibung sind heißen* ***Typenprädikat*** *von* $\mathcal{D}$*.* $\mathcal{P}^{Typ} \subseteq \mathcal{P}$ *sei die Menge aller Typenprädikate.*
Eine Initialzustandsbeschränkung $c_0 \in \mathcal{C}^{Typ}$ *heiße* ***Typenexklusivitätsbeschreibung*** *der Domäne* $\mathcal{D}$*, wenn sie mit* $\rho_1^1(\text{par}), \rho_2^1(\text{par}) \in \mathcal{P}^{Typ}$ *darstellbar ist durch:*

$$c_0 = \forall\ \omega\ (\neg(\rho_1^1(\omega_1) \wedge \rho_2^1(\omega_1)))$$

Eine Initialzustandsbeschränkung $c_0 \in \mathcal{C}^{Typ}$ *heiße* ***Typenhierarchiebeschreibung*** *der Domäne* $\mathcal{D}$*, wenn sie mit* $\rho_1^1(\text{par}), \rho_2^1(\text{par}) \in \mathcal{P}^{Typ}$ *darstellbar ist durch:*

$$c_0 = \forall\ \omega\ (\rho_1^1(\omega) \Rightarrow \rho_2^1(\omega))$$

Dann heißt ρ_1^1 ***Subtyp*** *von* ρ_2^1 *und* ρ_2^1 ***Supertyp*** *von* ρ_1^1*. Des Weiteren heiße ein* ρ ***Basistyp****, wenn es nicht Subtyp eines anderen* ρ' *ist.*

*Eine Initialzustandsbeschränkung $c_0 \in \mathcal{C}^{Typ}$ heiße **Parametertypisierung** der Domäne $\mathcal{D}$, wenn sie mit beliebigen, nicht notwendigerweise paarweise verschiedenen $\rho_i^1(\text{par}) \in \mathcal{P}^{Typ}$ und $\rho(\text{par}_1, \ldots, \text{par}_k) \in \mathcal{P} \setminus \mathcal{P}^{Typ}$ darstellbar ist durch:*

$$c_0 = \forall\ \omega_1, \ldots, \omega_k\ (\rho(\omega_1, \ldots, \omega_k) \Rightarrow (\bigwedge_{i=1}^{k} \rho_i^1(\omega_i)))$$

*Prädikate $\rho(\text{par}_1, \ldots, \text{par}_k) \in \mathcal{P} \setminus \mathcal{P}^{Typ}$, für die eine Parametertypisierung in den Initialzustandsbeschränkungen existiert, sowie alle nullstelligen Prädikate $\rho() \in \mathcal{P} \setminus \mathcal{P}^{Typ}$ heißen **parametertypisiert** mit den entsprechenden Typen. $\mathcal{P}^{Par} \subset \mathcal{P}$ sei die Menge alle parametertypisierten Prädikate. Aus der Definition der Parametertypisierung folgt für alle Domänen, dass $\mathcal{P}^{Typ} \cap \mathcal{P}^{Par} = \emptyset$.*
*Die Menge $\mathcal{C}^{Typ} \subseteq \mathcal{C}_0$, die alle Typenbeschreibungen, Typenexklusivitätsbeschreibungen, Typenhierarchiebeschreibung und Parametertypisierung enthält heiße Menge der **Typisierungsbeschränkungen**.*

Die zweite Kategorie, der eine Beschränkung angehören kann, heißt *Verbindungsbeschränkung*. Diese existiert in zwei Formen mit identischen Funktionalität, einmal für zweistellige und einmal für einstellige und ist in der SimpleRoute-Domäne durch c_0^8 repräsentiert:

$$c_0^8 = \forall\ p\ (\texttt{person}(p) \Rightarrow\ \exists^{=1}\ l\ (\texttt{location}(l) \wedge \texttt{at}(p, l)))$$

Eine zusätzliche Forderung besteht darin, dass das verbindungsbeschränkte Prädikat – in diesem Fall $\texttt{at}(p, l)$ – anzahlbeschränkt ist. Auf wohldefinierten Domänen, die im nächsten Abschnitt vorgestellt werden, folgt daraus, dass nur parametertypisierte Prädikate verbindungsbeschränkt sein können, da dort gefordert wird, dass Typenprädikate konstant sind. Dann ist es durch Beschränkungen dieser Art möglich, Zustandsbeschreibungsvariablen zu generieren, die in diesem Fall jedem Objekt, für das $\texttt{person}(p)$ gilt, genau ein Objekt, für das $\texttt{location}(l)$ gilt zuordnen. Allgemein haben Beschränkungen dieser Art also immer der Form, dass alle Objekte eines Typs mit einer bestimmten Anzahl von Objekten eines anderen (oder auch desselben) in Verbindung stehen. Da alle hier vorgestellten Domänen Verbindungsbeschränkungen nur für ein- oder zweistellige Prädikate benötigen, und der Nutzen im Hinblick auf die Herleitung des relevanten Zustandsraumes dann auch am größten ist, seien sie an dieser Stelle auch nur für diese definiert. Dennoch kann die Definition ohne großen Aufwand auch auf mehrstellige Prädikate ausgeweitet werden. Für einstellige Prädikate ist eine solche Beschränkung nur möglich, wenn nur ein Objekttyp in der Domäne existiert. Wäre das Prädikat $\texttt{person}(p)$ in der SimpleRoute-Domäne nicht enthalten, wäre Beschränkung c_0^8 funktional äquivalent dargestellt durch:

$$c_0 = \exists^{=1}\ l\ (\texttt{location}(l) \wedge \texttt{at}(l))$$

Die Kategorie der Verbindungbeschränkungen ist somit folgendermaßen definiert:

Definition 3.3 ***Verbindungsbeschränkung***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ eine Domäne mit $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$, $\rho_1(\text{par}), \rho_2(\text{par}) \in \mathcal{P}^{Typ}$ Typenprädikate und $\rho(\text{par}_1, \text{par}_2) \in \mathcal{P}^{Par}$ ein zweistelliges, mit $\rho_1(\text{par}_1)$ und $\rho_2(\text{par}_2)$ parametertypisiertes Prädikat.
*Eine Initialzustandsbeschränkung $c_0 \in \mathcal{C}_0$ heißt **Verbindungsbeschränkung** der Domäne $\mathcal{D}$, wenn sie mit einem anzahlkonstanten Prädikatenschema $\rho(\omega_1, \omega_2) \in \mathcal{P}^{Par}$ darstellbar ist durch:*

$$c_0 = \forall\ \omega_1\ (\rho_1(\omega_1) \Rightarrow \exists^{=l}\omega_2\ (\rho_2(\omega_2) \wedge \rho(\omega_1, \omega_2)))$$

Für einstellige, anzahlbeschränkte $\rho(\omega) \in \mathcal{P}^{Par}$ ist der analoge Fall:

$$c_0 = \exists^{=l}\omega\ (\rho_1(\omega) \wedge \rho(\omega))$$

*Die Menge $\mathcal{C}^{Rel} \subseteq \mathcal{C}_0 \setminus \mathcal{C}^{Typ}$, die alle Verbindungsbeschränkungen enthält heiße Menge der **Verbindungsbeschränkungen**.*

Die dritte Kategorie, der Initialzustandsbeschränkungen zugeordnet werden können, sind die *Spezialisierungen*. Diese sind für das in Kapitel 5 eingeführte System der Generierung von Spezialfällen von immenser Wichtigkeit und sind an dieser Stelle nur der Vollständigkeit halber sowie aufgrund dessen, dass die SimpleRoute-Domäne eine solche enthält, erwähnt. Auch wenn die Definition einer Kategorie mit nur einer zugehörigen Form sinnlos erscheinen mag soll diese bereits an dieser Stelle gegeben werden. In der SimpleRoute-Domäne existiert mit einer *Typelimination* eine Beschränkung dieser Form.

$$c_0^7 = \exists^{=1}\ p\ \texttt{person}(p)$$

Diese Art von Beschränkung existiert nur auf konstanten Prädikatenschemata und erscheint an dieser Stelle wenig sinnvoll. Dennoch wird ihre Existenz hier auch nicht gerechtfertigt werden, Typeliminationen spielen aber bei der Unterscheidung der Mitglieder von *Domänenfamilien* eine wichtige Rolle, und werden daher im entsprechenden Kapitel näher beschrieben.

Definition 3.4 ***Spezialisierungsbeschränkung***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ eine Domäne mit $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ und $\rho(\text{par}) \in \mathcal{P}^{Typ}$ ein Typenprädikat.
*Eine Initialzustandsbeschränkung $c_0 \in \mathcal{C}_0$ heißt **Typelimination** der Domäne $\mathcal{D}$, wenn sie für ein konstantes $\rho \in \mathcal{P}^{Typ}$ darstellbar ist durch:*

$$\exists^{=1}\ \omega\ \rho(\omega)$$

*Die Menge $\mathcal{C}^{Spe} \subseteq \mathcal{C}_0 \setminus (\mathcal{C}^{Typ} \cup \mathcal{C}^{Rel})$, die alle Typeliminationen enthält heißt Menge der **Spezialisierungsbeschränkungen**.*

Alle übrigen Initialzustandsbeschränkungen gehören der vierten und letzten Kategorie, den *domänenspezifischen Beschränkungen*, an und können dementsprechend jede beliebige, nicht beschriebene Form annehmen.

Definition 3.5 ***Domänenspezifische Beschränkung***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ eine Domäne mit $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$.
*Die Menge der **domänenspezifischen Beschränkungen** von $\mathcal{D}$ enthalte alle Beschränkungen $c_0 \in \mathcal{C}_0$, die nicht Teil der Typisierungsbeschränkung $\mathcal{C}^{Typ}$, Relationierungsbeschränkungen $\mathcal{C}^{Rel}$ oder Spezialisierungsbeschränkungen $\mathcal{C}^{Spe}$ sind, also $\mathcal{C}^{Dom} = \mathcal{C}_0 \setminus (\mathcal{C}^{Typ} \cup \mathcal{C}^{Rel} \cup \mathcal{C}^{Spe})$.*

Damit sind alle möglichen Formen von Initialzustandsbeschreibungen kategorisiert, womit nur noch ein Überblick über Zielzustandsbeschränkungen fehlt. Diese wurden dazu entwickelt, Planungsziele von Problemen der Domäne zu einzugrenzen, und Planern so eine maschinenlesbare Beschreibung des zu erreichenden Zieles zu geben. Dadurch werden aber tatsächlich nicht mehr alle Planungsaufgaben des IPC darstellbar: Häufig enthalten diese zusätzliche Objekte, von welchen ein Planer erkennen soll, dass sie zum Erreichen des Zieles unnötig sind. Dieser Vorgang ist aber so trivial, dass meiner Meinung nach fraglich ist, ob diese Einschränkung in der Modellierung von Planungsaufgaben tatsächlich einen Verlust darstellt. Ein Beispiel kann über die MultiPersonRoute-Domäne gegeben werden: Darin ist es möglich, beliebig viele Objekte des Typs $\texttt{person}(p)$ zu definieren. Wird nun in der Zielzustandsbeschreibung nur für einen Teil dieser Objekte gefordert, dass sie einen Zielort erreichen, ist ein Teil dieser Personen für das Erreichen des Zieles eigentlich unnötig. Dies ist durch die beiden Zielzustandsbeschränkungen aber nicht möglich:

$$c_\star^1 = \forall\ p\ (\texttt{person}(p) \Rightarrow\ \exists^{=1}\ l\ (\texttt{location}(l) \wedge \texttt{at}(p, l)))$$

$$c_\star^2 = \forall\ p\ (\texttt{person}(p) \Rightarrow\ \texttt{pickedUp}(p))$$

Diese fordern, dass alle Personen auch eine Aufgabe erfüllen. Dadurch kann ein Planer automatisch erkennen, worin eine Planungsaufgabe der Domäne besteht, es ist aber nicht weiter möglich, Personen anzugeben, die keine Funktion erfüllen.
Sonst besitzen Zielzustandsbeschränkungen häufig ein Pendant in den Initialzustandsbeschränkungen, wie es in der SimpleRoute-Domäne mit c_0^8 und c_0^9 der Fall ist. Auch Zielzustandsbeschreibungen können Komponenten der Menge der Grundelemente generieren, wie es im letzten Abschnitt dieses Kapitels beschrieben wird.

Damit sind alle für diese Arbeit wichtigen Formen von Beschränkungen kategorisiert. Insbesondere in der Kategorie der Spezialisierungen und vermutlich auch der Relationierungen können aber durchaus weitere, nicht beschriebene Formen existieren, die in Planungsalgorithmen verarbeitet werden können. Insbesondere decrementelle Prädikatenschemata bieten in dieser Hinsicht Raum für weitere Überlegungen.

3.3 Eigenschaften von Domänen

Bevor ein Algorithmus gegeben werden kann, der den relevanten Zustandsraum einer Domäne generiert, ist es notwendig, die Form, in welcher diese vorliegt zu beschränken. Dafür sei zunächst der Begriff der *minimalen* Domäne definiert, in welchen keine funktionslosen Schemata oder mehrfach beschriebenen Beschränkungen enthalten sein dürfen. Die Bezeichnung minimal ist dabei im Übrigen leicht irreführend: Es soll nicht sichergestellt werden, dass alle Schemata wirklich einen Nutzen haben, da dafür umfassendere Analysen notwendig werden, sondern lediglich einige Fälle ausgeschlossen werden, in welchen offensichtlich keine Funktionalität vorliegt. In Bezug auf Prädikatenschemata kann dabei festgestellt werden, dass sie immer entweder ein Typenprädikat sein müssen, oder aber parametertypisiert sind:

Bemerkung 3.1 ***Prädikatenschemata sind Typenprädikate oder parametertypisiert***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *eine Domäne. Dann existiert eine funktional äquivalente Domäne* $\mathcal{D}_\bullet = \langle \mathcal{P}_\bullet, \mathcal{O}_\bullet, \mathcal{C}_\bullet, \widehat{w}_\bullet \rangle$ *mit* $\mathcal{P}_\bullet^{Typ} \cup \mathcal{P}_\bullet^{Par} = \mathcal{P}_\bullet$.

Ein formaler Beweis für diese Bemerkung sei nicht gegeben, es sei lediglich erwähnt, dass alle nicht parametertypisierten Prädikatenschemata durch ein neu definiertes Typenprädikat `object`(ω), das zum Supertyp aller Basistypen gemacht wird, parametertypisiert werden können, ohne die Funktionalität zu verändern. Damit kann die Frage, wann ein Prädikatenschema funktionslos ist, für die beiden Arten von Prädikatenschemata getrennt beantwortet werden.
Ein Typenprädikat $\rho \in P^{Typ}$ ist sicher ohne Nutzen, wenn es konstant ist, keine Sub- oder Supertypen besitzt und zusätzlich keinen Parameter typisiert. Dann könnte es zwar noch immer in der Vorbedingung eines Operatorenschemas enthalten sein, in diesem Fall beschreibt es aber eine Eigenschaft von Objekten und sollte dementsprechend nicht als Typenprädikat gegeben sein.
Parametertypisierte Prädikate müssen dagegen in einer Vorbedingung oder in einem Effekt mindestens eines Operatorenschemas beinhaltet sein, damit eine Domäne ohne diese Prädikat nicht funktional äquivalent ist. Auch Zustandsbeschränkungen können verworfen werden, wenn es bereits durch alle anderen impliziert wird. Formal ergibt sich eine minimale Domäne, in welche jede beliebige Domäne transformiert werden kann wie folgt:

Definition 3.6 ***Minimale Domäne***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *eine Domäne mit* $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ *und* $\sigma(\mathrm{par}_1, \ldots, \mathrm{par}_k) \in \mathcal{O}$ *ein beliebiges Operatorenschema.*
$\mathcal{D}$ *ist* ***minimal****, wenn gilt:*

- *Alle Namen von Schemata sind eindeutig.*
- $\mathcal{P}^{Typ} \cup \mathcal{P}^{Par} = \mathcal{P}$.

- *Für alle $\rho \in P^{Typ}$ gilt mindestens eine der folgenden Eigenschaften:*
  - *ρ ist aussagekräftig.*
  - *Es existiert ein $\rho' \in \mathcal{P}^{Typ}$ und ρ' ist Subtyp von ρ.*
  - *Es existiert ein $\rho' \in \mathcal{P}^{Typ}$ und ρ' ist Supertyp von ρ.*
  - *Es existiert ein $\rho' \in \mathcal{P}^{Par}$ und ein Parameter von ρ' ist typisiert mit ρ.*
- *Für alle $\rho \in \mathcal{P}^{Par}$ gilt mindestens eine der folgenden Eigenschaften:*
  - *ρ ist aussagekräftig.*
  - *Es existiert ein $\sigma \in \mathcal{O}$ und $\rho \in \mathrm{Sy}(\mathrm{pre}(\sigma))$.*
- *Es existiert kein $\rho \in \mathcal{P}$, das Supertyp oder Subtyp von sich selbst ist.*
- *Für alle $\rho \in \mathcal{P}^{Typ}$, die Supertyp eines $\rho_1 \in \mathcal{P}^{Typ}$ sind, existiert auch ein $\rho_2 \in \mathcal{P}^{Typ}$, $\rho_1 \neq \rho_2$, dessen Supertyp ρ ist.*
- *Für alle $\sigma \in \mathcal{O}$ gilt $\mathrm{Sy}(\mathrm{pre}(\sigma)) \cap \mathcal{P}^{Typ} = \emptyset$.*
- *Für alle $\sigma \in \mathcal{O}$ gilt $\mathrm{Sy}(\mathrm{eff}(\sigma)) \neq \emptyset$.*
- *Für alle $c_0 \in \mathcal{C}_0$ gilt nicht $\mathcal{C}_0 \setminus \{c_0\} \models c_0$.*
- *Für alle $c_\star \in \mathcal{C}_\star$ gilt nicht $\mathcal{C}_\star \setminus \{c_\star\} \models c_\star$.*

Alle geforderten Eigenschaften sind entweder bereits beschrieben, sie sind trivial oder es sei im Fall der Einschränkungen der Typenhierarchie abermals auf [Bal98] verwiesen. Domänen dieser Form beinhalten zwar keine offensichtlich funktionslosen Komponenten, dennoch genügt eine solche noch nicht den Ansprüchen, die die im nächsten Abschnitt vorgestellte Methode benötigt. Zusätzlich sollen auch keine offensichtlichen Widersprüche in der Domänenbeschreibung enthalten sein, was durch die Definition *wohldefinierter* Domänen ausgeschlossen wird:

Definition 3.7 ***Wohldefinierte Domäne***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ eine Domäne mit $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ und $\sigma(\mathrm{par}_1, \ldots, \mathrm{par}_k) \in \mathcal{O}$ ein beliebiges Operatorenschema..
*$\mathcal{D}$ ist **wohldefiniert**, wenn sie minimal ist und zusätzlich gilt:*

- *Alle $\rho \in \mathcal{P}^{Typ}$ sind konstant.*
- *Für alle $\rho \in \mathcal{P}^{Typ}$ und alle $\sigma \in \mathcal{O}$ gilt $\rho \notin \mathrm{Sy}(\mathrm{pre}(\sigma))$.*
- *$\bigwedge_{c_0 \in \mathcal{C}_0} c_0$ ist erfüllbar.*
- *$\bigwedge_{c_\star \in \mathcal{C}_\star} c_\star$ ist erfüllbar.*
- *Alle $\rho_i \in \mathcal{P}^{Typ}$, die nicht Subtyp eines $\rho \in \mathcal{P}^{Typ}$ sind, sind paarweise typenexklusiv.*
- *Alle $\rho_i \in \mathcal{P}^{Typ}$, die Subtyp desselben $\rho \in \mathcal{P}^{Typ}$ sind, sind paarweise typenexklusiv.*
- *Alle $\sigma \in \mathcal{O}$ sind derart, dass ihr Effekt nur Prädikate gültig macht, für die sichergestellt ist, dass keine Parametertypisierung verletzt wird.*

In diese Form lassen sich nicht mehr beliebige Domänen funktional äquivalent transformieren, allerdings ist dies nur dann nicht möglich, wenn sich ein Widerspruch für Initial- oder Zielzustandsbeschreibung aus den Planungsaufgabenbeschränkungen ergibt und somit auch keine lösbaren Planungsaufgaben auf der Domäne modelliert werden können. Damit sind auch alle relevanten Domänen wohldefinierbar, auch wenn dann im Bereich der Typenhierarchie teilweise größere Umformungen notwendig werden.
Auch Domänen, in welchen Typenprädikate knappe Ressourcen beschreiben, müssen durch Einführung eines zusätzlichen Prädikats, das anzeigt, das die Ressource verbraucht wurde, erweitert werden. Die Forderung, dass Typenprädikate niemals Teil von Vorbedingungen sind, resultiert ebenfalls in manchen Fällen darin, dass eine Erweiterung zu einer funktional äquivalenten Domäne nötig wird: Durch die Forderung nach Parametertypisierung in minimalen Domänen sind diese nur dann notwendig, wenn ein Operator nur für einen Subtyp gelten soll, durch die Parametertypisierung aber nicht ausgeschlossen wird, dass auch auf einen anderer Subtyp desselben Supertyps der Operator angewendet nicht. Dann können die parametertypisierten Prädikate so erweitert werden, dass der Operator nur noch auf diese angewandt werden kann, oder alternativ ein neues Prädikat eingeführt werden, das lediglich kennzeichnet, ob der Operator mit einem bestimmten Objekt angewendet werden kann.

3.4 Generierung relevanter Zustandsräume

In diesem Abschnitt wird ein Algorithmus angegeben, der eine Beschreibung eines relevanten Zustandsraumes aller Planungsaufgaben einer Domäne liefert. Das Problem, das gelöst werden soll, kann also folgendermaßen beschrieben werden:

Definition 3.8 ***Generierung einer effizienten Zustandsbeschreibung***
Das Problem der ***Generierung einer effizienten Zustandsbeschreibung*** GenEff-$\mathcal{D}$ *ist gegeben durch:*

Eingabe: *Eine wohldefinierte Domäne $\mathcal{D}$.*

Lösung: *Eine polynomiell berechenbare Funktion, die für eine gegebene, instantiierte Planungsaufgabe $\mathcal{T}$ auf $\mathcal{D}$ effiziente Zustandsbeschreibungsvariablen sowie die Menge der Grundelemente generiert.*

Die beschriebene Funktion sei im Folgenden nicht formal definiert. Stattdessen werden vier Tupel erstellt, die Symbole enthalten, mit deren Hilfe der entsprechende relevante Zustandsraum offensichtlich generiert werden kann. Drei dieser Tupel haben dabei die grundsätzlich selbe Form, seien mit S^{Dsc}, S_0^{Dsc} und $S_\star^{Dsc}$ bezeichnet und werden Zustände, Initialzustand sowie Zielzustand repräsentieren. Die vierte ist eine Repräsentierung der Menge der Grundelemente und sei bezeichnet als $\mathcal{B}^{Dsc}$. Die durch GenEff-$\mathcal{D}$ gegebene Funktion füllt diese Komponenten dann für Planungsaufgaben anhand der darin enthaltenen Prädikate auf. Komponenten aller Tupel können grundsätzlich Mengen, Teilmengen oder Elemente von Mengen, Konstanten oder auch komplexere Konstrukte wie zum Beispiel Graphen oder Funktionen symbolisieren. Im Folgenden wird die Methode exemplarisch anhand der SimpleRoute-Domäne beschrieben und erst am Ende verallgemeinert.

Die Beschreibung von Initial- und Zielzustandsbeschreibung wird zu Beginn des Algorithmus nicht beachtet, es werden erst die beiden anderen Tupel initiiert. Die Repräsentation der Grundelemente beinhalte dabei zu Beginn den Namen aller Typenprädikate als je eine Komponente, und die Repräsentierung der Zustände den Rest:

$$\mathcal{B}^{Dsc} = \langle \texttt{location}, \texttt{person} \rangle$$

$$S^{Dsc} = \langle \texttt{at}, \texttt{itemAt}, \texttt{pickedUp}, \texttt{connected} \rangle$$

In der Beschreibung relevanter Zustandsräume wird, wie auch in der PDDL-Initialzustandsbeschreibung, von der Closed-World-Annahme ausgegangen, in welcher nur die Prädikate aufgezählt werden, die in diesem Zustand gelten. Dadurch wird es erleichtert, Zustände durch Mengen zu beschreiben: Gibt man für jedes nicht-nullstellige Prädikat die Teilmenge des n-fachen Kreuzprodukts der Objektmenge (symbolisch durch Ω repräsentiert) an, für die das Prädikat gilt, werden Zustände über Mengen beschrieben. Nullstellige Prädikate müssen dagegen weiter als Element aus $\{0, 1\}$ angegeben werden. Dies ist auch der nächste Schritt, wir erweitern jede Komponente um ein Symbol, das angibt ob ein Element ($\in$) oder eine Teilmenge ($\subseteq$) der darauf folgenden Menge (Ω oder $\{0, 1\}$) das Prädikat beschreibt.

$$\mathcal{B}^{Dsc} = \langle \texttt{location} \subseteq \Omega, \texttt{person} \subseteq \Omega \rangle$$

$$S^{Dsc} = \langle \texttt{at} \subseteq \Omega \times \Omega, \texttt{itemAt} \subseteq \Omega, \texttt{pickedUp} \subseteq \Omega, \texttt{connected} \subseteq \Omega \times \Omega \rangle$$

Auch der nächste Schritt ist trivial: Alle in S^{Dsc} enthaltenen, konstanten Prädikate werden aus S^{Dsc} gelöscht und in $\mathcal{B}^{Dsc}$ eingefügt. Diese müssen ebenfalls nicht für jeden erreichbaren Zustand mit angegeben werden und sind daher Elemente der Menge der Grundelemente.

$$\mathcal{B}^{Dsc} = \langle \texttt{location} \subseteq \Omega, \texttt{person} \subseteq \Omega, \texttt{itemAt} \subseteq \Omega, \texttt{connected} \subseteq \Omega \times \Omega \rangle$$

$$S^{Dsc} = \langle \texttt{at} \subseteq \Omega \times \Omega, \texttt{pickedUp} \subseteq \Omega \rangle$$

Nun sollen die gegebenen Objekttypen in eigenständige Mengen umgewandelt werden. Dafür wird jedem Typ ein Symbol zugeordnet, das die Menge repräsentiert, die alle Objekte enthält, für die das entsprechende Prädikate bereits in der Initialzustandsbeschreibung der Planungsaufgabe gilt. Zusätzlich werden eventuell vorhandene Hierarchiebeschreibungen durch entsprechende Teilmengenrelationen repräsentiert. In unserem Fall entstehen so die Mengen der Orte L und die der Personen P. Zusätzlich werden diese mithilfe der Parametertypisierungen statt der Ω in alle anderen Prädikate eingesetzt:

$$\mathcal{B}^{Dsc} = \langle \texttt{location}\ L, \texttt{person}\ P, \texttt{itemAt} \subseteq L, \texttt{connected} \subseteq L \times L \rangle$$

$$S^{Dsc} = \langle \texttt{at} \subseteq P \times L, \texttt{pickedUp} \subseteq P \rangle$$

An dieser Stelle ist bereits eine Beschreibung entstanden, mit deren Hilfe eine sehr viel kompaktere Zustandsbeschreibung möglich ist. Der nächste Schritt besteht darin, Informationen aus der Menge der Verbindungsbeschränkungen zu verwerten. Diese haben die Form, dass jedes Objekt einer Menge mit einer bestimmten Anzahl von Objekten einer anderen verbunden ist. Die zusätzliche Vorbedingung, dass das Verbindungsbeschränkte Prädikat anzahlkonstant ist, gilt im Falle der SimpleRoute-Domäne für das Prädikatenschema $\texttt{at}(p, l)$. In unserem Beispiel besagt diese Beschränkung, dass jede Person immer nur an genau einem Ort sein kann. Es kann also eine Funktion erstellt werden, die jeder Person diesen Ort zuweist, womit in unserem Fall $\subseteq P \times L$ durch $P \to L$ ersetzt wird. Auch diese erhalte einen Namen, in unserem Fall loc. Liegt dagegen eine Verbindungsbeschränkung eines einstelligen Prädikates vor, kann diese durch eines (bzw. l) Elemente der Menge beschrieben werden, welches einen Namen erhalte (x) und die Symbole $\subseteq \Omega$ dann durch $x \in \Omega$ ersetzt werden.

$$\mathcal{B}^{Dsc} = \langle \texttt{location}\ L, \texttt{person}\ P, \texttt{itemAt} \subseteq L, \texttt{connected} \subseteq L \times L \rangle$$

$$S^{Dsc} = \langle \texttt{at}\ \text{loc}{:}P \to L, \texttt{pickedUp} \subseteq P \rangle$$

Nun können auch komplexere Gebilde, die in einer der beiden Mengen enthalten sind, generiert und mit einem Namen versehen werden. So können im Allgemeinen immer zwei Komponenten M und $M \times M$ in einem Graphen zusammenfasst werden. Dabei können durchaus auch weitere Mengen in den Komponenten

enthalten sein und die Graphen so abhängig von anderen Objekten sein, was zum Beispiel durch zwei Komponenten $L \rightarrow M$ und $L \rightarrow (M \times M)$ gegeben ist und wodurch eine Graphenschar $G(l)$ für jedes Objekt $l \in L$ erstellt wird. Im Fall der SimpleRoute-Domäne wird so der Wegenetzgraph aus den Prädikatenschemata `location` und `connected` gebildet, der bezeichnet wird durch $G = \langle L, E \rangle$:

$$\mathcal{B}^{Dsc} = \langle G = \langle \texttt{location}\ L, \texttt{connected}\ E \rangle, \texttt{person}\ P, \texttt{itemAt} \subseteq L \rangle$$

$$S^{Dsc} = \langle \texttt{at}\ \text{loc:}P \rightarrow L, \texttt{pickedUp} \subseteq P \rangle$$

Zwei weitere spezielle Graphen, die auf diese Weise generiert werden können, seien an dieser Stelle aufgrund ihrer Relevanz für diese Arbeit vorgestellt. Der erste wird später in Tranportplanungsdomänen verwendet, um Liefergraphen zu beschreiben und heiße *Abhängigkeitsgraph*. Dieser wird impliziert durch zwei Funktionen $f : A \rightarrow B$, $g : A \rightarrow B$ und sei bezeichnet mit $D = \langle f, b \rangle$. Dieser existiert in einer gerichteten und einer ungerichteten Form, wobei ein gerichteter Graph erstellt wird, wenn g aus Zielzustandsbeschränkungen generiert wird. Definiert sei an dieser Stelle nur diese Form, das ungerichtete Pendant ist aber analog zu erstellen.

Definition 3.9 ***Abhängigkeitsgraph***
Seien A und B zwei endliche Mengen, sowie $f : A \rightarrow B$, $g : A \rightarrow B$ Funktionen, die jedem $a \in A$ ein $b \in B$ zuweisen. Durch f und g kann der ***Abhängigkeitsgraph*** *$G = V, E$ folgendermaßen generiert werden:*

- *Die Menge der Knoten V wird gebildet durch die Menge B, also $V = B$.*
- *Die Menge der Kanten E enthält für jedes $a \in A$ eine gerichtete Kante $e = \langle f(a), g(a) \rangle$, also $E = \{\langle f(a), g(a) \rangle | a \in A\}$.*

Wird im Folgenden ein Abhängigkeitsgraph mit $D = \langle f, g \rangle$ angegeben sei dadurch der so generierte Graph $G = \langle V, E \rangle$ beschrieben.

Der zweite Graph, durch welchen die spezielle Form des Wegenetzgraphen der Logistics-Domäne wiedergegeben wird, ist nicht so einfach aus einer Domänenbeschreibung zu generieren, insbesondere nicht ohne die im folgenden Kapitel vorgestellten numerischen Variablen. Daher sei er an dieser Stelle lediglich definiert und darauf hingewiesen, dass eine automatische Generierung durchaus möglich ist. Der Graph heiße *hierarchisch-vollständiger Graph* und kann theoretisch beliebig viele Hierarchiestufen enthalten, wird aber lediglich als zweistufiger hierarchisch-vollständiger Graph definiert:

Definition 3.10 ***Zweistufiger hierarchisch-vollständiger Graph***
Sei A eine endlich Menge und $B = \{B_1, \ldots, B_{|A|}\}$ eine Menge aus $|A|$ endlichen, paarweise disjunkten Mengen. Zudem gelte für jedes $a \in A$, dass ein $B_i \in B$ existiert mit $a \in B_i$, und für alle $B_j \neq B_i$, dass $a \notin B_j$. Dann kann durch A und B der ***zweistufige hierarchisch-vollständige Graph*** *$G = \langle V, E \rangle$ folgendermaßen generiert werden:*

- *Die Menge der Knoten V wird gebildet als $V = \bigcup_{i=1}^{|A|} B_i$.*
- *Die Menge der Kanten E wird folgendermaßen gebildet:*
  - *A bildet einen vollständigen* ***Subgraphen****, also für alle paarweise verschiedenen Knoten $a_1, a_2 \in A$ ist eine gerichtete Kante in beide Richtungen in E, also $\langle a_1, a_2 \rangle \in E$ und $\langle a_2, a_1 \rangle \in E$.*
  - *Jedes $B_i \in B$ bildet einen vollständigen Subgraphen, also für alle paarweise verschiedenen Knoten $b_{i1}, b_{i2} \in B_i$ ist eine gerichtete Kante in beide Richtungen in E, also $\langle b_{i1}, b_{i2} \rangle \in E$ und $\langle b_{i2}, b_{i1} \rangle \in E$.*

Wird im Folgenden ein zweistufig hierarchisch-vollständiger Graph mit $W = \langle A, B \rangle$ angegeben sei dadurch der so generierte Graph $G = \langle V, E \rangle$ beschrieben.

Es muss hier angemerkt werden, dass die automatische Generierung eines solchen Graphen aus einer Domänenbeschreibung sehr schwierig ist: Selbst wenn die beschriebenen Mengen A und B Teil der Zustandsbeschreibungsvariablen sind, macht diese Art der Verbindung nicht immer auch Sinn. Dafür sind zusätzlich Mengenrelationen nötig, welche die beschriebene, besondere Art der Verbindung repräsentieren. Dies soll an dieser Stelle aber nicht weiter vertieft werden, es sei lediglich darauf hingewiesen, dass dieser Graph durchaus automatisiert aus einer Domänenbeschreibung generiert werden kann, sofern dies für ein Planungssystem von Vorteil ist. Zur Verdeutlichung zweistufig hierarchisch-vollständiger Graphen sei noch ein Beispiel eines solchen angegeben. Die Knoten rechts oben bilden dabei die Menge A, die drei tiefer liegenden Knotenmengen zusammen mit dem verbundenen Knoten aus A jeweils ein B_i. Die vierte Menge aus B besteht dagegen nur aus dem mit "4-1" beschriebenen Knoten:

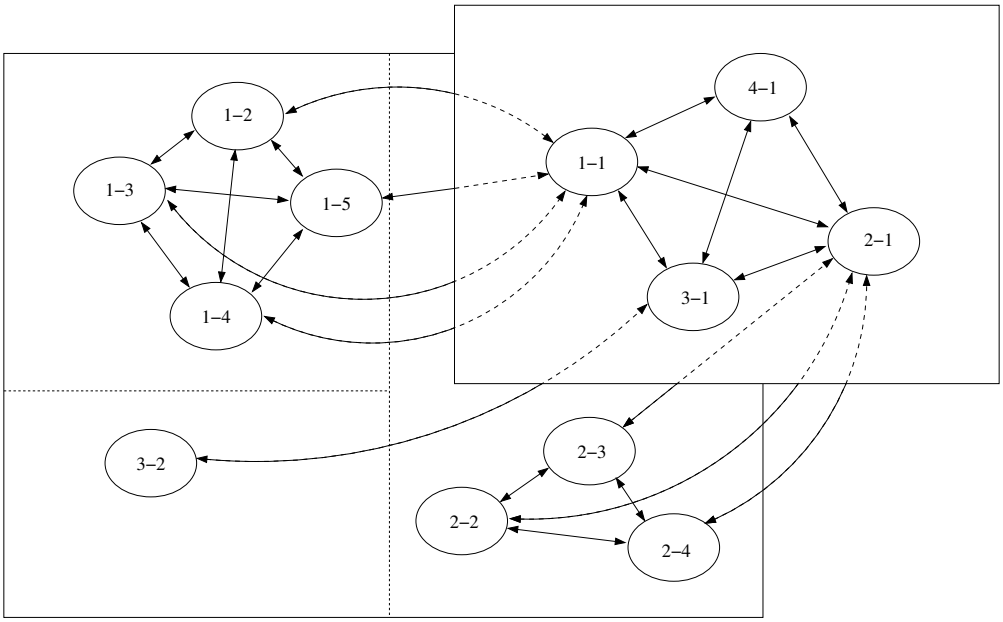

Abbildung 3.1: Beispiel eines zweistufig hierarchisch-vollständigen Graphen

Damit wurden zwei weitere Konstrukte vorgestellt, die aus der Domänenbeschreibung generiert werden können. Diese wurden aber nur aus dem Grund ausgewählt, da sie für die Belange dieser Arbeit benötigt werden. Allgemein kann eine Regel für jedes beliebige Konstrukt angegeben werden, was mit dieser Methode generiert wird hängt also sehr davon ab, auf welchen Konstrukten ein Planer gute Leistungen erbringen kann. Da die Graphentheorie ein intensiv untersuchtes Gebiet sowohl der Informatik als auch der Mathematik ist, wurden diese auch eingehender vorgestellt, es könnten aber zum Beispiel durchaus auch Matrizen generiert werden.

Damit sei nun der letzte Schritt des Algorithmus beschrieben, welcher auch die Zielzustandsbeschränkungen auswertet. In der SimpleRoute-Domäne ist die erste zugehörig zur Beschränkung c_0^8 und besagt, dass für jede Person ein besonderer Ort in der Zielzustandsbeschreibung existieren muss. Das bereits aus dem Prädikatenschema `at` gebildete Konstrukt wird mit einem speziell gekennzeichneten Symbol in die Menge der Grundelemente aufgenommen. Zielzustandsbeschränkungen generieren Komponenten dieser Menge, da sich diese während der Ausführung von Operatoren niemals ändern und somit konstant sind. Zusätzlich wird nun die Zielzustandsbeschreibung $S_\star^{Dsc}$ als Kopie der Zustandsbeschreibung S^{Dsc} generiert, und dann vermerkt, dass nur dann ein Zielzustand erreicht ist, wenn die analoge Komponente der Zustandsbeschreibung – in diesem Fall `at`, genau dieser konstanten Komponente entspricht. Auch $c_\star^2$ kann derart mit aufgenommen werden. Hier ist allerdings ein Unterschied zu bemerken: Während $C_\star^1$ eine variable Beschränkung

darstellt, da ein beliebiges Element aus L das Ziel ist, ist $C^2_\star$ nicht variabel, sondern in jeder Planungsaufgabe konstant – alle $p \in P$ müssen eine bestimmte Eigenschaft aufweisen. Dies ist immer dann der Fall, wenn eine Zielzustandsbeschränkung durch eines der Symbole 0, 1, $\emptyset$ oder eine beliebige, bereits erstellte Menge beschreibbar ist. Ist dies der Fall, muss keine zusätzliche Komponente in die Menge beschreibender Elemente aufgenommen werden, es genügt, dies in der Zielzustandsbeschreibung zu vermerken:

$$\mathcal{B}^{Dsc} = \langle G = \langle \texttt{location}\ L, \texttt{connected}\ E\rangle, \texttt{person}\ P, \texttt{itemAt} \subseteq L, \mathrm{loc}_\star{:}P \to L, \rangle$$

$$S^{Dsc} = \langle \texttt{at}\ \mathrm{loc}{:}P \to L, \texttt{pickedUp} \subseteq P\rangle$$

$$S^{Dsc}_\star = \langle \texttt{loc}_\star, P\rangle$$

Auch eine letzte domänenspezifische Beschränkung ist noch zu verarbeiten: Das Prädikat `pickedUp` gilt im Initialzustand für keine Person. All diese Beschränkungen werden nun genutzt, um auch die Initialzustandsbeschreibung zu generieren. Damit ergeben sich abschließend die folgenden vier beschreibenden Elemente:

$$\mathcal{B}^{Dsc} = \langle G = \langle \texttt{location}\ L, \texttt{connected}\ E\rangle, \texttt{person}\ P, \texttt{itemAt} \subseteq L, \mathrm{loc}_\star{:}P \to L, \rangle$$

$$S^{Dsc} = \langle \texttt{at}\ \mathrm{loc}{:}P \to L, \texttt{pickedUp} \subseteq P\rangle$$

$$S^{Dsc}_0 = \langle \mathrm{loc}_0, \emptyset\rangle$$

$$S^{Dsc}_\star = \langle \texttt{loc}_\star, P\rangle$$

An dieser Stelle muss aber auch auf ein Problem dieser automatischen Generierung hingewiesen werden: Zwar wurde mit der Definition wohldefinierter Domänen versucht, die Art, in welcher Domänen vorliegen können, zu beschränken, im Allgemeinen ist die Modellierung von Domäne der Handlungsplanung aber ein Gebiet, das nicht hinreichend untersucht wurde. Es existieren noch viele weitere Möglichkeiten, Zustandsbeschreibungen noch effizienter zu gestalten. Als Beispiel sei hier lediglich die Erstellung von Kostenfunktionen erwähnt, die notwendig werden, wenn mehrere Operatoren mit unterschiedliche Kosten denselben Zweck erfüllen. Aber auch sonst können funktional äquivalente Domänen durch Beschreibungssprachen wie PDDL auf zu vielen Wegen erreicht werden, als dass in dieser Arbeit ein vollständiger Algorithmus zur Generierung minimaler Zustandsräume aus beliebigen Beschreibungen präsentiert werden könnte. Die Generierung effizienter, wenngleich im Allgemeinen nicht minimaler Zustandsbeschreibungen ist mit der beschriebenen Methode aber dennoch möglich.

Algorithmus: ***Generierung einer effizienten Zustandsbeschreibung***

INPUT: *Eine wohldefinierte Domäne* $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w}\rangle$ *mit* $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star\rangle$

OUTPUT: $\mathcal{B}^{Dsc}$,S^{Dsc},S^{Dsc}_0,$S^{Dsc}_\star$

ALGORITHMUS:
create $\mathcal{B}^{Dsc}$ *containing all* $\rho \in \mathcal{P}^{Typ}$
create S^{Dsc} *containing all* $\rho \in \mathcal{P}^{Par}$
expand each component containing $\rho = \langle \mathrm{na}, \mathrm{ar}\rangle$ *in* S^{Dsc} *and* $\mathcal{B}^{Dsc}$ *by*
  $\subseteq \Omega^{ar}$ *if* $ar \geq 1$
  $\in \{0, 1\}$ *otherwise*
remove all constant ρ *from* S^{Dsc} *and add them to* $\mathcal{B}^{Dsc}$
for each basic type $\rho \in \mathcal{P}^{Typ}$ *create a symbol* Ω_ρ
for each $\rho \in \mathcal{P}^{Typ}$ *that is subtype of* $\rho' \in \mathcal{P}^{Typ}$ *create a symbol* $\Omega_\rho \subset \Omega_{\rho'}$
for each $\rho \in \mathcal{P}^{Typ}$ *in* $\mathcal{B}^{Dsc}$ *replace all* $\subseteq \Omega$ *by*

the symbol created for that type

for each $\rho \in \mathcal{P}^{Par}$ *in* S^{Dsc} *or* $\mathcal{B}^{Dsc}$ *replace each* Ω *by*
the symbol created for the type of the parameter

for each relation constraint $c \in \mathcal{C}^{Rel}$ *constraining* Ω_i *with*
exactly 1 object in Ω_j *replace* $\subseteq \Omega_i \times \Omega_j$ *with* $\Omega_i \rightarrow \Omega_j$

for each relation constraint $c \in \mathcal{C}^{Rel}$ *constraining* Ω_i *with*
exactly 1 object replace $\subseteq \Omega_i$ *with* $\omega \in \Omega_i$

createComplexStructures$(S^{Dsc}, \mathcal{B}^{Dsc})$

includeDomainSpecificConstraints$(S^{Dsc}, \mathcal{B}^{Dsc}, S_0^{Dsc}, S_\star^{Dsc})$

return $\mathcal{B}^{Dsc}, S^{Dsc}, S_0^{Dsc}, S_\star^{Dsc}$

Die beiden abschließenden Funktionen wurden aus unterschiedlichen Gründen nicht näher beschrieben: Komplexe Konstrukte können, wie beschrieben, in vielen Formen vorliegen, und dementsprechend je nachdem, was generiert werden soll, variieren. Die aus den Zielzustandsbeschränkungen sowie domänenspezifischen Beschränkungen generierten Zustandsbeschreibungsvariablen können dagegen von sehr unterschiedlicher Form sein, und dementsprechend wären noch einmal sehr viele Unterscheidungen nötig gewesen, ohne dass dadurch ein großer Erkenntnisgewinn entstanden wäre. Zusätzlich wurde darauf verzichtet, Verbindungsbeschränkungen mit $l \neq 1$ im Pseudeo-Code zu beachten.

Damit ist ein Algorithmus zur Generierung relevanter Zustandsräume angegeben, dennoch ist der hergeleitete relevante Zustandsraum der SimpleRoute-Domäne nicht identisch mit dem in Teil E des Beispiels gegebenen. Der Grund liegt darin, dass in dieser eine Typelimination existiert, welche eigentlich als erstes hätte verarbeitet werden sollen. Dann wäre aber das Beispiel weniger aussagekräftig gewesen. Zudem wäre eigentlich Kapitel 5 der geeignetere Ort, dieses Verfahren zu präsentieren, da darin auch der Zweck dieser Art von Beschränkung erläutert wird. Um aber denselben relevanten Zustandsraum wie im vorigen Kapitel für das Beispiel bereits an dieser Stelle zu generieren sei das Verfahren an dieser Stelle präsentiert. Das Prädikateneliminierungsproblem kann folgendermaßen angegeben werden:

Definition 3.11 ***Problem der Prädikateneliminierung***
Das Problem der ***Prädikateneliminierung*** PredEl-$\mathcal{D}$ *ist gegeben durch:*

Eingabe: *Eine wohldefinierte Domäne* $\mathcal{D}$.

Lösung: *Eine funktional äquivalente, wohldefinierte Domäne* $\mathcal{D}'$ *ohne Typeliminationsbeschränkungen.*

Tatsächlich sollte der dieses Problem lösende Algorithmus Teil des vorher beschriebenen zur Generierung einer effizienten Zustandsbeschreibung sein, da einige Probleme – außer durch Beibehaltung der Beschränkung – nur durch Erstellung bestimmter Symbole im relevanten Zustandsraum lösbar sind: Wird zum Beispiel ein Typenprädikat eliminiert, das Subtyp eines anderen ist, und es existiert nur genau ein weiterer Subtyp desselben, könnten beide – sofern keine, noch eine Hierarchieebene weiter unten stehenden Typen existieren – im Supertyp aufgehen. Dennoch können dann weiterhin Prädikate existieren, für welche die Unterscheidung der beiden ehemals vorhandenen Subtypen wichtig ist. Die einfachste Lösung hierfür ist die Repräsentierung des eliminierten Prädikats als konstantes Symbol im relevanten Zustandsraum, das in die entsprechenden Parameter eingesetzt werden kann. Letztlich ist dieses Problem aber zu speziell, als dass es an dieser Stelle ausführlich diskutiert oder der gegebene Lösungsvorschlag formalisiert werden müsste und wird somit im Folgenden keine Beachtung mehr finden.

Algorithmus: ***Prädikateneliminierung***

INPUT: *Eine wohldefinierte Domäne* $\mathcal{D}' = \langle \mathcal{P}', \mathcal{O}', \mathcal{C}', \widehat{w}' \rangle$ *mit* $\mathcal{C}' = \langle \mathcal{C}'_0, \mathcal{C}'_\star \rangle$

OUTPUT: $\mathcal{D}$

ALGORITHMUS:
create $\mathcal{D} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *as a copy of* $\mathcal{D}'$
for each elimination constraint eliminating $\rho \in \mathcal{P}^{Typ}$ *do*
  remove ρ *from* $\mathcal{P}$
  for each $\rho_1 \in \mathcal{P}^{Par}$ *do*
    decrease ar *by* x, *where* x *is the number of parameters with type* ρ
    update all $\sigma \in \mathcal{O}$ *by deleting all according parameters in* ρ_1
  for each $\sigma \in \mathcal{O}$ *do*
    delete all par $\in$ Par *not contained in any* ρ
    $\mathcal{D}$ = *updateConstraints*$(\mathcal{D}, \rho)$
return $\mathcal{D}$

Der Algorithmus ist leicht zu erklären: Das zu eliminierende Prädikatenschema wird aus $\mathcal{P}$, allen Prädiaktenschemata, in denen es einen Parameter typisiert sowie allen Operatoren, in denen es vorkommt gelöscht. Auch die verwendete Methode `updateConstraints` ist größtenteils trivial, aus allen Typisierungen wird das entsprechende Prädikat einfach gelöscht, die gegebene Spezialisierungsbeschränkung wird ersatzlos gestrichen und zweistellige Verbindungsbeschränkungen werden zu dem in Definition 3.3 gegebenen, einstelligen Pendant.

Wird erst diese Methode auf die Domänenbeschreibung der SIMPLEROUTE-Domäne angewendet und dann der Algorithmus zur Generierung einer effizienten Zustandsbeschreibung, entsteht das bereits im vorigen Kapitel beschriebene Ergebnis. Damit wurde aufgezeigt, wie mithilfe von Initialzustandsbeschränkungen der relevante Zustandsraum generiert werden kann, ohne dass die zusätzliche Angabe einer konkreten Planungsaufgabe notwendig ist. Zusätzlich wurden in diesem Kapitel verschiedene Eigenschaften von Prädikatenschemata definiert und der Versuch unternommen, die Modellierungsmöglichkeiten von Domänen einzugrenzen, ohne ihre Funktionsvielfalt zu beschränken. Diese werden im nächsten Kapitel mit der Einführung numerischer Variablen sogar noch erweitert.

4 Erweiterung des Grundkonzepts

Dieses Kapitel bietet einen kurzen Überblick über eine Erweiterung des in Kapitel 2 vorgestellten Grundkonzepts der Handlungsplanung, das *Numerische Planen*.

Diese Erweiterung führt als zusätzliches Werkzeug numerische Variablen ein, deren einziger Unterschied zu Prädikaten darin besteht, dass sie beliebige Werte aus der Menge natürlicher Zahlen sowie unendlich (∞) oder undefiniert ($\perp$) annehmen können. Damit wird die Menge modellierbarer Domänen und Planungsaufgaben erweitert, aber auch viele der bereits mit dem Grundkonzept modellierbaren Domänen können mit numerischen Variablen leichter verständlich angegeben werden. Auch diese existieren sowohl in schematischer als auch instantiierter Form und sind definiert als:

Definition 4.1 ***Numerisches Variablenschema***
Ein ***numerisches Variablenschema*** *ist ein 2-Tupel* $\psi = \langle \mathrm{na}, \mathrm{ar} \rangle$ *mit den Komponenten:*

- na *ist der* ***Name*** *des numerischen Variablenschemas (über einem endlichen Alphabet* Σ*).*
- $\mathrm{ar} \in \mathbb{N}_0$ *ist die* ***Stelligkeit*** *des numerischen Variablenschemas.*

Äquivalent zur Angabe eines numerischen Variablenschemas durch $\psi = \langle \mathrm{na}, \mathrm{ar} \rangle$ *seien* $\mathrm{na}(\mathrm{par}_1, \ldots, \mathrm{par}_{\mathrm{ar}})$ *und* $\psi(\mathrm{par}_1, \ldots, \mathrm{par}_{\mathrm{ar}})$.

Durch die Einführung numerischer Variablenschemata und Variablen müssen die Grammatiken und rekursiven Mengenaufbaubeschreibungen aus Definition 2.2 entsprechend erweitert werden.

Definition 4.2 ***Grammatiken und Mengen***
Sei $\mathcal{P}$ *eine Menge von Prädikatensymbolen* $\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \in \mathcal{P}$, $\{\mathrm{par}_1, \ldots, \mathrm{par}_n\}$ *eine Menge von Variablensymbolen,* $\mathcal{V}$ *eine Menge numerischer Variablensymbole* $\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m)$ *sowie* $n \geq k$, $n \geq j$, $n \geq m$, $x, x' \in \mathbb{N}_0 \cup \{\infty, \perp\}$ *und* $l \in \mathbb{N}$.
Ein ***zahlwertiger Ausdruck*** # *wird rekursiv gebildet durch die Grammatik:*

$$\# := \psi(\mathrm{par}_1, \ldots, \mathrm{par}_m) \mid x \mid (\# \bullet \#)$$

$$\bullet := + \mid - \mid *$$

Im Folgenden sei # das Symbol für ein zahlwertigen Ausdruck.
Ein ***Boolescher Ausdruck*** $\Diamond$ *wird rekursiv gebildet durch die Grammatik:*

$$\Diamond := (\# \circ \#)$$

$$\circ := < \mid > \mid = \mid \leq \mid \geq \mid \neq$$

Im Folgenden sei $\Diamond$ *das Symbol für einen Booleschen Ausdruck.*
Eine ***numerische Zuweisungskonjunktion*** ϕ *wird rekursiv gebildet durch die Grammatik:*

$$\phi := \varphi \mid \psi(\mathrm{par}_1, \ldots, \mathrm{par}_m) \rightarrow \# \mid \phi \wedge \phi$$

$$\varphi := \rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)$$

*Eine **numerische aussagenlogische Formel** ϕ wird rekursiv gebildet durch die Grammatik:*

$$\phi := \rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \Diamond \circ \Diamond \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

*Eine **numerische prädikatenlogische Formel** ϕ wird rekursiv gebildet durch die Grammatik:*

$$\phi := \rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k) \mid \Diamond\circ\Diamond \mid \phi\wedge\phi \mid \phi\vee\phi \mid \forall\ \mathrm{par}_j\phi \mid \exists\ \mathrm{par}_j\phi \mid \exists^{=l}\ \mathrm{par}_j\phi$$

*Die **Symbolmenge** $\mathrm{Sy}(\phi)$ einer numerische Zuweisungskonjunktion ϕ ist rekursiv über dem Aufbau von ϕ definiert durch:*

$$\mathrm{Sy}(\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m) \rightarrow \#) := \{\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m)\}$$

$$\mathrm{Sy}(\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)\}$$

$$\mathrm{Sy}(\neg\varphi) := \mathrm{Sy}(\varphi)$$

$$\mathrm{Sy}(\phi \wedge \phi) := \mathrm{Sy}(\phi) \cup \mathrm{Sy}(\phi)$$

*Die **polarisierte Symbolmenge** $\mathrm{Sy}_{\pm}(\phi)$ einer numerische Zuweisungskonjunktion ϕ ist rekursiv über dem Aufbau von ϕ definiert durch:*

$$\mathrm{Sy}_{\pm}(\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m) \rightarrow \#) := \{\langle\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m), \#\rangle\}$$

$$\mathrm{Sy}_{\pm}(\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 1\rangle\}$$

$$\mathrm{Sy}_{\pm}(\neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\langle\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k), 0\rangle\}$$

$$\mathrm{Sy}_{\pm}(\phi \wedge \phi) := \mathrm{Sy}_{\pm}(\phi) \cup \mathrm{Sy}_{\pm}(\phi)$$

*Die **Menge freier Variablen** $\mathrm{Fr}(\phi)$ einer numerischen prädikatenlogischen Formel ϕ ist rekursiv über dem Aufbau von ϕ definiert durch:*

$$\mathrm{Fr}(\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\mathrm{par}_1, \ldots, \mathrm{par}_k\}$$

$$\mathrm{Fr}(\neg\rho(\mathrm{par}_1, \ldots, \mathrm{par}_k)) := \{\mathrm{par}_1, \ldots, \mathrm{par}_k\}$$

$$\mathrm{Fr}(\# \circ \#) := \mathrm{Fr} \cup \mathrm{Fr}$$

$$\mathrm{Fr}(\phi \wedge \phi) := \mathrm{Fr}(\phi) \cup \mathrm{Fr}(\phi)$$

$$\mathrm{Fr}(\phi \vee \phi) := \mathrm{Fr}(\phi) \cup \mathrm{Fr}(\phi)$$

$$\mathrm{Fr}(\forall\ \mathrm{par}_j\phi) := \mathrm{Fr}(\phi) \setminus \{\mathrm{par}_j\}$$

$$\mathrm{Fr}(\exists\ \mathrm{par}_j\phi) := \mathrm{Fr}(\phi) \setminus \{\mathrm{par}_j\}$$

$$\mathrm{Fr}(\exists^{=l}\ \mathrm{par}_j\phi) := \mathrm{Fr}(\phi) \setminus \{\mathrm{par}_j\}$$

$$\mathrm{Fr}(x) := \emptyset$$

$$\mathrm{Fr}(\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m)) := \{\mathrm{par}_1, \ldots, \mathrm{par}_m\}$$

$$\mathrm{Fr}(\# \bullet \#) := \mathrm{Fr}(\#) \cup \mathrm{Fr}(\#)$$

*Eine **numerische Effektkonjunktion** ist eine numerische Zuweisungskonjunktion ϕ, für die zusätzlich für alle $\upsilon \in \mathcal{P} \cup \mathcal{V}$ gilt, dass nicht:*

$$\langle\upsilon, x\rangle \in \mathrm{Sy}_{\pm}(\phi) \textit{ und } \langle\upsilon, x'\rangle \in \mathrm{Sy}_{\pm}(\phi) \textit{ für } x \neq x'$$

*Eine **vollständige numerische Konjunktion** ist eine numerische Zuweisungskonjunktion ϕ, für die zusätzlich für alle $\upsilon \in \mathcal{P} \cup \mathcal{V}$ gilt, dass:*

$$\text{es existiert genau ein } x \text{ mit } \langle \upsilon, x \rangle \in \mathrm{Sy}_{\pm}(\phi)$$

*Eine **geschlossene numerische prädikatenlogische Formel** ist eine numerische prädikatenlogische Formel ϕ, für die zusätzlich gilt:*

$$\mathrm{Fr}(\phi) = \emptyset$$

Die Semantik aller bereits im vorigen Kapitel verwendeten Symbole bleibt bestehen. Die verwendeten Rechen-, Vergleichs- und Zuweisungssymbole sind auf den natürlichen Zahlen selbsterklärend. Zusätzlich gelten die folgenden Regeln:

- Rechenoperatoren (+ , - , *):
  - Für $x, y \in \mathbb{N}_0$ und $x < y$ gilt $x - y = 0$.
  - Für $x = \bot$, $y \in \mathbb{N}_0 \cup \{\infty, \bot\}$ und $\bullet \in \{+, -, *\}$ gilt $x \bullet y = \bot$ und $y \bullet x = \bot$.
  - Für $x = \infty$ und $y \in \mathbb{N}_0 \cup \{\infty\}$ gilt $x + y = \infty$, $y + x = \infty$, $x * y = \infty$ und $y * x = \infty$.
  - Für $x = \infty$, $y \in \mathbb{N}_0$ und $z = \infty$ gilt $x - y = \infty$, $x - z = \infty$ und $y - x = 0$.
- Vergleichsoperatoren ($<$, $>$, $=$, $\leq$, $\geq$, $\neq$):
  - Für $x = \bot$ und $y \in \mathbb{N}_0 \cup \{\infty, \bot\}$ gilt keiner der Vergleichsoperatoren.
  - Für $x = \infty$ und $y = \infty$ gilt nur $x = y$.
  - Für $x = \infty$ und $y \in \mathbb{N}_0$ gelten $x > y$, $x \geq y$, $y < x$, $y \leq x$, $x \neq y$ und $y \neq x$.
- Zuweisungsoperator ($\rightarrow$):
  - Sei Val eine Funktion, die jeder numerischen Variable $\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m)$ einen Wert $\mathrm{Val}(\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m)) \in \mathbb{N}_0 \cap \{\infty, \bot\}$ zuweist. Val kann auf zahlwertige Ausdrücke erweitert werden, indem Literale auf sich selbst abgebildet und arithmetische Operationssymbole durch die entsprechenden Rechenoperationen interpretiert werden.
  - Der Zuweisungsoperator $\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m) \rightarrow \#$ setzt die numerische Variable $\psi(\mathrm{par}_1, \ldots, \mathrm{par}_m)$ auf den Wert Val(#).

Es soll nicht unerwähnt bleiben, dass in der Literatur numerische Variablen teilweise reellwertig definiert werden, und zusätzlich auch das Rechensymbol $\div$ erlaubt wird. Für die in dieser Arbeit betrachteten Domänen ergibt eine solche Definition aber keine Vorteile, dafür verlieren einige komplexitätstheoretische Erkenntnisse, die am Ende dieses Abschnitts beschrieben sind, ihre Gültigkeit. Aus diesen Gründen wird auf reellwertige numerische Variablen verzichtet.
Natürlich müssen nicht nur die Grammatiken und Mengen erweitert werden, sondern auch die meisten Definitionen von Kapitel 2. Dabei werden allerdings häufig nur Formeln und Mengen aus 2.2 durch die entsprechenden numerischen Formeln und Mengen aus 4.2 ersetzt. Exemplarisch sei dies anhand der Definition *numerischer Operatorenschemata* aufgezeigt, wird im Folgenden dann aber nur noch erwähnt werden. Der Name solcher nicht explizit neu definierten Konzepte sei, wie bei Def. 4.3, derjenige, der diesen im Kapitel über die Grundlagen der Handlungsplanung gegebenen wurde, um den Zusatz “numerisch” erweitert.

Definition 4.3 ***Numerisches Operatorenschema***
Sei $\mathcal{P}$ eine Menge von Prädikatenschemata und $\mathcal{V}$ eine Menge von numerischen Variablenschemata. Ein ***Numerisches Operatorenschema*** *ist ein 4-Tupel $\sigma = \langle \text{na}, \text{par}, \text{pre(par)}, \text{eff(par)} \rangle$ mit den Komponenten:*

- na *ist der* ***Name*** *des Operatorenschemas (über einem endlichen Alphabet Σ).*
- $\text{par} = \{\text{par}_1, \ldots, \text{par}_n\}$ *ist die endliche Menge an* ***Parametern****.*
- $\text{pre(par)} = \phi$ *ist die* ***Vorbedingung*** *des Operatorenschemas, eine numerische aussagenlogische Formel ϕ mit Prädikatensymbolen $\rho \in \mathcal{P}$, Variablensymbolen $\text{par}_1, \ldots, \text{par}_n \in \text{par}$ und numerischen Variablensymbolen $\psi \in \mathcal{V}$.*
- $\text{eff(par)} = \phi$ *ist der* ***Effekt*** *des Operatorenschemas, eine numerische Effektkonjunktion ϕ mit Prädikatensymbolen $\rho \in \mathcal{P}$, Variablensymbolen $\text{par}_1, \ldots, \text{par}_n \in \text{par}$ und numerischen Variablensymbolen $\psi \in \mathcal{V}$.*

Im Folgenden bezeichne

- $\text{pre}(\sigma)$ *die Vorbedingung eines numerischen Operators oder Operatorenschemas σ.*
- $\text{eff}(\sigma)$ *den Effekt eines numerischen Operators oder Operatorenschemas σ.*

Auf die selbe Weise werden auch die Planungsbeschränkungen angepasst, während die *numerische Planungsdomäne* um eine zusätzliche Komponente, der Menge numerischer Variablenschemata, zu einem 5-Tupel erweitert wird.

Definition 4.4 ***Numerische Planungsdomäne***
Eine ***Numerische Planungsdomäne*** *ist ein 5-Tupel $\mathcal{D} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \overline{w} \rangle$ mit den Komponenten:*

- $\mathcal{P}$ *ist die endliche Menge der* ***Prädikatenschemata****.*
- $\mathcal{V}$ *ist die endliche Menge der* ***Numerischen Variablenschemata****.*
- $\mathcal{O}$ *ist die endliche Menge der* ***Operatorenschemata****.*
- $\mathcal{C}$ *ist die* ***Planungsaufgabenbeschränkung****.*
- $\overline{w} : \mathcal{O} \to \mathbb{N}_0$ *ist die* ***Kostenfunktion*** *der Operatorenschemata.*
  Entspricht diese einer konstanten Funktion mit $\widehat{w}(\sigma) = 1$ für alle Operatorenschemata $\sigma \in \mathcal{O}$ wird von ***Einheitskosten*** *gesprochen.*

Die Definition von Planungsaufgaben bleibt, abgesehen davon, dass ihre Komponenten numerisch sind, unverändert. Bei der Instantiierung muss dagegen ein zusätzlicher Arbeitsschritt bewerkstelligt werden:

- Die Menge der **Prädikate** P entsteht analog.
- Durch Einsetzen der Objekte $\omega \in \Omega$ in die numerischen Variablenschemata $\mathcal{V}$ der Domäne $\mathcal{D}$ entsteht die Menge der **numerischen Variablen** V.
  Eine numerische Variable $v \in V$, die durch Einsetzen von Objekten $\omega_1, \ldots, \omega_{\text{ar}}$ in das numerische Variablenschema $\psi = \langle \text{na}, \text{ar} \rangle$ entsteht, sei auch dargestellt als $\text{na}(\omega_1, \ldots, \omega_{\text{ar}})$ oder $v(\omega_1, \ldots, \omega_{\text{ar}})$.
- Durch Einsetzen der Objekte $\omega \in \Omega$ in die Parameter der Operatorenschemata $\mathcal{O}$ der Domäne $\mathcal{D}$ sowie der zu den Objekten zugehörigen Prädikate $p \in P$ und numerischen Variablen $v \in V$ in die in Vorbedingung und Effekt vorkommenden Prädikatenschemata und numerischen Variablenschemata entsteht die Menge der **Operatoren** $\overline{O}$.

Damit ergibt sich auch eine erweiterte Definition der Zustandsbeschreibung, in welcher neben der Polarität der Prädikate nun zusätzlich auch eine Belegung numerischer Variablen enthalten sein kann.

Definition 4.5 ***Numerische Zustandsbeschreibung***
Sei $\Im = \langle \mathcal{D}, \Omega, \widehat{s}_0, \widehat{s}_\star \rangle$ *eine Planungsaufgabe auf der numerischen Domäne* $\mathcal{D}$*, P die durch Instantiierung von* $\Im$ *erhaltene Menge aller Prädikate p, V die durch Instantiierung* $\Im$ *erhaltene Menge aller numerischen Variablen v sowie* $P' \subseteq P$ *und* $V' \subseteq V$*.*
Eine ***totale Zustandsbeschreibung*** $\widehat{s}$ *auf einer instantiierten Planungsaufgabe* $\Im$ *ist eine vollständige numerische Konjunktion mit allen Prädikatensymbolen* $p \in P'$*, numerischen Variablensymbolen* $v \in V'$ *und Variablensymbolen* $\omega \in \Omega$*, falls* $P' = P$ *und* $V' = V$*. Gilt* $P' \neq P$ *oder* $V' \neq V$*, heißt* $\widehat{s}$ ***partielle Zustandsbeschreibung****.*
Die Menge $\widehat{S}(\widehat{s})$ *sei die Menge aller totalen Zustandsbeschreibungen* $\widehat{s}_t$ *auf* $\Im$*, für die gilt* $\widehat{s}_t \models \widehat{s}$ *und heiße* ***Menge der von*** $\widehat{s}$ ***beschriebenen Zustände****.*
Ist $\widehat{s}$ *total, gilt* $\widehat{S}(\widehat{s}) = \{\widehat{s}\}$*.*
$\widehat{S}(\top)$ *heißt* ***Menge der Zustände****. Diese wird dargestellt durch* $\widehat{S}$ *und es gilt* $\widehat{S} \cong 2^P \times (\mathbb{N}_0 \cap \{\infty, \bot\})^V$*.*

Einhergehend mit dieser Neudefinition ist eine nicht unwichtige Veränderungen: Die Menge der von einer partiellen Zustandsbeschreibung beschriebenen Zustände und damit auch die Menge der Zustände ist im Allgemeinen nicht länger endlich. Dieser Umstand sei an dieser Stelle lediglich erwähnt, näher darauf eingegangen wird erst am Ende dieses Abschnittes.
Keine großen Überraschungen, aber dennoch so viele Änderungen, dass eine Angabe der Definition nötig ist, entstehen bei der *Anwendung numerischer Operatoren*:

Definition 4.6 ***Anwendung numerischer Operatoren***
Sei $\overline{o} = \langle \mathrm{na}, \{\omega_1, \ldots, \omega_n\}, \mathrm{pre}(\{\omega_1, \ldots, \omega_n\}), \mathrm{eff}(\{\omega_1, \ldots, \omega_n\}) \rangle$ *ein numerischer Operator und* $\widehat{s}$ *eine totale numerische Zustandsbeschreibung.* $\overline{o}$ *ist* ***anwendbar*** *in* $\widehat{s}$ *wenn gilt:*

$$\widehat{s} \models \mathrm{pre}(\{\omega_1, \ldots, \omega_n\})$$

Die dann mögliche ***Operatoranwendung*** *von* $\overline{o}$ *in* $\widehat{s}$ *bewirkt einen Übergang von* $\widehat{s}$ *in den* ***Nachfolgerzustand*** $\overline{o}(\widehat{s})$*. In dieser ebenfalls totalen Zustandsbeschreibung behalten alle Prädikate und numerischen Variablen, die im Effekt des Operators nicht enthalten sind, den in* $\widehat{s}$ *gegebenen Wert. Diese bilden die Menge der* ***Rahmenvariablen****:*

$$\mathrm{Sy}_{\pm}^{\mathrm{Frame}} = \{\langle p, x \rangle \in \mathrm{Sy}_{\pm}(\widehat{s}) \mid p \notin \mathrm{Sy}(\mathrm{eff})\}$$

Alle Übrigen werden auf den im Effekt gegebenen Wert gesetzt, womit $\overline{o}(\widehat{s})$ *gegeben ist als:*

$$\overline{o}(\widehat{s}) = \bigwedge_{\langle p,1\rangle \in \mathrm{Sy}_{\pm}^{\mathrm{Frame}} \cap P} p \wedge \bigwedge_{\langle p,0\rangle \in \mathrm{Sy}_{\pm}^{\mathrm{Frame}} \cap P} \neg p \wedge \bigwedge_{\langle p,1\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff}) \cap P} p \wedge \bigwedge_{\langle p,0\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff}) \cap P} \neg p \wedge \bigwedge_{\langle v,\#\rangle \in \mathrm{Sy}_{\pm}^{\mathrm{Frame}} \cap V} v \rightarrow \mathrm{Val}(\#) \wedge \bigwedge_{\langle v,\#\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff}) \cap V} v \rightarrow \mathrm{Val}(\#)$$

$\widehat{O}$ *sei die Menge der* ***Operatoren****, die für alle* $\overline{o} \in \overline{O}$ *und alle* $\widehat{s}$*, in denen* $\overline{o}$ *anwendbar ist, einen Operator* $\widehat{o} = \langle \widehat{s}, \widehat{o}(\widehat{s}) \rangle$ *enthält. Analog werden in der* ***Kostenfunktion*** *der Operatoren* $\widehat{w}$ *jedem so entstandenen Operator die Kosten aus* $\overline{w}$ *zugewiesen.*

Damit verbleiben lediglich noch die beiden Definitionen der Zustandsräume. Die vollständige Variante wird lediglich dahingehend angepasst, dass die Zustandsmenge nicht länger endlich ist. Da eine numerische Variable jeden Wert in den natürlichen Zahlen annehmen kann führt schon eine solche Variable dazu, dass nur durch diese bereits unendlich viele unterschiedliche Zustände beschrieben werden können. Auf die Definition des relevanten Zustandsraumes hat die Einführung numerischer Variablen dagegen keine Auswirkungen: Zwar ist die Menge relevanter Zustände in nicht-numerischen Domänen auch immer endlich, es ist in der

Definition aber nicht als Voraussetzung enthalten. Zudem wurde im vorangehenden Kapitel bereits beschrieben, wie Funktionen als Zustandsbeschreibungsvariable aus Beschränkungen und Schemata gebildet werden können. Eine solche, wenn auch zahlwertige, ist natürlich immer dazu geeignet, eine numerische Variable im relevanten Zustandsraum zu repräsentieren.

Die im vorigen Kapitel beschrieben Analyse unterschiedlicher Typen von Beschränkungen und deren Auswirkungen auf die Generierung des relevanten Zustandsraumes sollen an dieser Stelle weder wiederholt noch vollständig auf numerische Variablen angepasst werden, da die Definitionen sich relativ einfach anpassen lassen, wenn die Zuweisung $\perp$ als Pendant zu einem nicht geltenden Prädikatenschema angesehen wird. An dieser Stelle seien lediglich einige Gedanken über zusätzliche Möglichkeiten angedeutet.
Ein Beispiel dazu, was Beschränkungen mit numerischen Variablen ermöglichen, ist, die Anzahl unterschiedlicher Objekttypen je nach Planungsaufgabe einer Domäne variabel zu gestalten:

$$c = \forall\ \omega\ (\rho_1(\omega) \vee \rho_2(\omega) \vee (\texttt{varObjType}(\omega) \neq \perp))$$

Auf diese Weise sind sogar Möglichkeiten geboten, die mit der Methode, durch welche Typen in PDDL beschrieben werden, nicht existieren. Ob daraus tatsächlich Vorteile bei der Planung resultieren könnten, wird in dieser Arbeit nicht geklärt werden. Möglichkeiten bei der Modellierung existieren aber durchaus, die sich auch auf die Leistungsfähigkeit von Planern auswirken könnten. Als Beispiel sei die MultiPersonRoute-Domäne folgendermaßen geändert: Jede Person bewege sich in ihrem eigenen, von allen anderen getrennten Wegenetz und habe zur Aufgabe, den Weg zu einem Zielort finden müssen. Diese, an die später vorgestellte Logistics-Domäne angelehnte Variante, könnte derart modelliert werden, dass Typen durch zwei numerische Variablen – eine für Personen und eine für Orte – ausgedrückt werden. Hat eine Person denselben Wert in der numerischen Personentypvariable wie ein Ort in der Ortstypvariable, befinden sich beide in der selben Stadt. Dafür müssen aber variabel viele solcher Typenvariablen möglich sein, da die Anzahl von Planungsaufgabe zu Planungsaufgabe verschieden sein kann. Dies kann mithilfe numerischer Variablen in der Typenbeschreibung folgendermaßen gelöst werden:

$$c = \forall\ \omega\ ((\texttt{locationType}(\omega) \neq \perp) \vee (\texttt{personType}(\omega) \neq \perp))$$

Wird nun zusätzlich Typenexklusivität gefordert, was auf numerischen Variablen gelöst wird, indem genau eine der Variablen für alle Objekte undefiniert sein muss, kann die beschriebene Domäne wie gewünscht mit einer variablen Anzahl von Typen modelliert werden, ohne dass es dadurch unerheblich wird, welchen Typ ein bestimmtes Objekte hat. Vorteile ergeben sich für Planungssysteme dadurch, dass auf diese Weise ein noch größerer Teil der Erreichbarkeitsanalyse bereits aus der Domäne hervorgeht, da Personen p einen Ort l nur besuchen können, wenn $\texttt{locationType}(l) = \texttt{personType}(p)$ gilt. Allerdings muss auch hier kritisch hinterfragt werden, inwieweit eine möglicherweise daraus resultierende verbesserte Leistung von Planern akzeptiert werden kann, wenn letztlich lediglich eine informationsreichere Modellierung von Domänen dafür verantwortlich ist. Ohne eine Antwort zu dieser Frage zu geben sei abermals auf Planer wie TLPlan [BK00] oder TALPlanner [DK01] verwiesen.

Mit diesem Beispiel sei die Betrachtung numerischer Variablen in Bezug auf Zustandsbeschränkungen bereits abgeschlossen. Es sollte lediglich ein Bild der Möglichkeiten angedeutet werden, die daraus resultieren können. Stattdessen sei der Blick zum Abschluss noch auf die veränderte Komplexität numerischer Planungsprobleme gerichtet. Wie bereits beschrieben, ist der Zustandsraum nicht länger endlich, was dazu führt, dass die Komplexitätsbetrachtungen aus Abschnitt 2.3 auch nicht mehr gültig sind: Die dort beschriebenen Aussagen sind nur deshalb möglich, da der Zustandsraum nicht-numerischer Domänen exponentiell in der Anzahl der Prädikate groß ist und dementsprechend in exponentieller Zeit durchsucht werden kann. Da die Größe des Zustandsraumes in numerischen Domänen aber unendlich ist, kann keine Obergrenze für die Planungs- und Planungsoptimierungprobleme der Definitionen 2.12 und 2.13 gegeben werden, diese sind im Allgemeinen also *unentscheidbar* [Hel02].

Satz 4.1 ***Komplexität des Planens auf numerischen Domänen***
PLAN-$\mathcal{D}^\infty$ *und* OPTPLAN-$\mathcal{D}^\infty$ *sind* ***unentscheidbar*** *auf numerischen Domänen.*
Domänenabhängiges Planen benötigt auch auf numerischen Domänen Laufzeiten abhängig von der Domäne und kann ebenfalls unentscheidbar sein. Für eine beliebige numerische Domäne $\mathcal{D}$ *gilt also, dass* PLAN-$\mathcal{D}$ *und* OPTPLAN-$\mathcal{D}$ ***unentscheidbar*** *sind.*

Dennoch kann eine Gruppe numerischer Domänen ausgemacht werden, für welche dieselben komplexitätstheoretischen Schranken gelten, wie sie im vorigen Kapitel gegeben wurden. Dies gilt genau dann, wenn es möglich ist, eine solche Domäne gleichwertig ohne numerische Variablen zu definieren, also wenn alle numerischen Variablenschemata *prädikatenreduzierbar* sind.

Definition 4.7 ***Prädikatenreduzierbarkeit***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \overline{w} \rangle$ *eine numerische Domäne,* $\psi \in \mathcal{V}$ *ein numerisches Variablenschema und bezeichne* $V(\mathcal{T}, \psi)$ *die Menge aller numerischen Variablen, die durch Instanziierung mit Objekten* $\omega_1, \ldots, \omega_n \in \Omega$ *aus* ψ *hervorgehen.* $\psi = \langle \mathrm{na}, \mathrm{ar} \rangle$ *ist genau dann* ***prädikatenreduzierbar****, wenn in allen Planungsaufgaben* $\mathcal{T}$ *auf* $\mathcal{D}$ *für alle* $v \in V(\mathcal{T}, \psi)$ *eine endliche Menge* $M \subseteq \mathbb{N}_0 \cup \{\bot, \infty\}$ *existiert, so dass* $v : \Omega^{\mathrm{ar}} \to M$ *gilt.*
Sind alle $\psi \in \mathcal{V}$ *prädikatenreduzierbar heißt* $\mathcal{D}$ ***prädikatenreduzierbare Domäne****.*

Ein numerisches Variablenschema ist also genau dann prädikatenreduzierbar, wenn es auf allen Planungsaufgaben in einen endlichen Werteraum abgebildet wird. Dann ist es möglich, alle in diesem Raum vorkommenden natürlichen Zahlen (sowie ∞ und $\bot$ wenn nötig) als Objekt zu definieren und dann die numerische Variable durch ein Menge von Prädikaten zu ersetzen, deren Parameter aus denselben Objekten und jeweils einem zusätzlichen Zahlenobjekt bestehen. Gesetzt wird in jedem Zustand dann genau das eine Prädikat, dessen letzter Parameter das dem numerischen Variablenwert entsprechende Zahlenobjekt ist. Diese Umformung ist also gewissermaßen genau das Gegenteil der Methode, die bei der Erstellung mehrwertiger Zustandsbeschreibungsvariablen angewandt wird. Zusätzlich werden über den Zahlenobjekten noch dreistellige Prädikate benötigt, die die verwendeten Rechenoperationen repräsentieren – z.B. $add(1, 2, 3)$ als gültiges oder $\neg multiply(1, 2, 4)$ als nicht gültiges Prädikat für $\{1, 2, 3, 4\} \subseteq \Omega$. Auch die Operatoren werden entsprechend angepasst: Die Parameter werden um die benötigten Zahlenobjekte erweitert, und die entstandenen Prädikate in Vorbedingungen und Effekte eingesetzt.
Sind diese Umformungen für alle numerischen Variablen und damit auch für alle Vorkommen in Operatoren einer Domäne möglich, kann also eine funktional äquivalente, nicht-numerische Domäne dazu angegeben werden. Diese ist im Vergleich zu üblichen instantiierten Planungsaufgaben zwar erheblich größer, aber nicht um einen über-exponentiellen Faktor, weswegen Planungsprobleme auf *prädikatenreduzierbaren Domänen* denselben Komplexitätsklassen angehören.

Satz 4.2 ***Komplexität des Planens auf prädikatenreduzierbaren numerischen Domänen***
Sei $\mathcal{D}^\infty_{\mathrm{pr}}$ *die Klasse aller prädikatenreduzierbaren numerischen Domänen und* $\mathcal{D}_{\mathrm{pr}} \in \mathcal{D}^\infty_{\mathrm{pr}}$.
Dann gilt PLAN-$\mathcal{D}^\infty_{\mathrm{pr}} \in$ *EXPO* $\setminus$ *NPS,* OPTPLAN-$\mathcal{D}^\infty_{\mathrm{pr}} \in$ *EXPO* $\setminus$ *NPS,* PLAN-$\mathcal{D}_{\mathrm{pr}} \in$ *EXPO sowie* OPTPLAN-$\mathcal{D}_{\mathrm{pr}} \in$ *EXPO.*

Damit wurde ein kurzer Überblick über numerische Planungsdomänen gegeben und die Auswirkungen dieser Erweiterung diskutiert. Somit ist es möglich, Domänen kompakter und damit meistens auch intuitiver darzustellen. Dies kann zwar zu unentscheidbaren Problemstellungen führen, im Fall von prädikatenreduzierbaren numerischen Domänen behalten die Komplexitätsbetrachtungen des Grundlagenkapitels aber ihre Gültigkeit.

5 Transport- und Routenplanungsprobleme

In diesem Kapitel wird die Familie der Transport- und Routenplanungsprobleme vorgestellt. Der erste Teil beschreibt dabei formal die Eigenschaften von Domänenfamilien, einer Menge von Domänen, die Varianten desselben *Subproblems* beinhalten. In den beiden darauf folgenden Abschnitten werden die gewonnen Erkenntnisse auf Transport- und Routenplanungsprobleme angewandt und deren Eigenschaften beschrieben.

5.1 Domänenfamilien

Nicht nur die Komplexität der bislang für die IPC verwendeten Welten weist eine enorme Bandbreite auf, auch deren Thematik könnte unterschiedlicher kaum sein. So mussten Planer bereits die Instrumente von Satelliten bedienen (SATELLITE), die Energieversorgung aufrecht erhalten (PSR), Kartenspiele spielen (FREECELL) oder auch einen Filmeabend organisieren (MOVIE). Bei allen Unterschieden zwischen den Domänen lässt sich aber auch ein häufig wiederkehrendes Thema ausmachen, nämlich das der *Transport- und Routenplanungsprobleme*.
Ähnlich der Zusammenfassung gleichartiger Planungsaufgaben in Domänen lassen sich auch letztere in sogenannten *Domänenfamilien* gruppieren. Der Unterschied besteht dabei darin, dass Planungsprobleme durch Domänen derart beschränkt werden, dass in diesen zwar auf einen Teil der durch die Domäne gegebenen Möglichkeiten verzichtet werden kann, niemals aber zusätzliche Prädikate oder Operatoren hinzugefügt werden können. Eine Domäne ist also die Beschreibung dessen, was in zugehörigen Planungsaufgaben maximal möglich ist. Eine *Familienbeschreibung* definiert dagegen, welche Prädikate, numerische Variablen und Operatoren vorhanden sein müssen, beschränkt zusätzliche Funktionalitäten aber nicht, und entspricht dementsprechend einer Beschreibung dessen, was minimal in allen *Mitgliedern* erlaubt sein muss. Die einzige Ausnahme hiervon bildet die *Effekteinschränkung*, welche sicherstellt, dass kein Operatorenschema dem beabsichtigten Sinn widerspricht.

Definition 5.1 ***Effekteinschränkung***
Eine ***Effekteinschränkung*** *ist ein 3-Tupel* $\mathcal{O}^{\mathcal{C}_-} = \langle \mathcal{P}_{\text{const}}, \mathcal{V}_{\text{const}}, \mathcal{C}_{\text{eff}} \rangle$ *mit den Komponenten:*

- $\mathcal{P}_{\text{const}}$ *ist die Menge* ***unveränderbarer Prädikatenschemata****, eine Menge von Prädikatenschemata.*
- $\mathcal{V}_{\text{const}}$ *ist die Menge* ***unveränderbarer numerischer Variablenschemata****, eine Menge numerischer Variablenschemata.*
- $\mathcal{C}_{\text{eff}}$ *ist die Menge der* ***Effektbeschränkungen****, eine Menge geschlossener numerischer prädikatenlogischer Formeln.*

Ein Operatorenschema widerspricht dann nicht einer Effekteinschränkung, wenn es *effektkonform* mit dieser ist. Dafür müssen drei Voraussetzungen erfüllt sein: Alle in den beiden Mengen unveränderbarer Schemata enthaltenen Prädikate beziehungsweise numerische Variablen dürfen in keiner Form Teil des Effektes des Operatorenschemas sein, und zusätzlich darf der Effekt auch nicht gegen eine der gegebenen Effektbeschränkungen verstoßen. Diese können beliebige numerische prädikatenlogische Formeln sein, und so zum Beispiel Anzahlkonstanz bestimmter Prädikatenschemata garantieren.

Definition 5.2 ***Effektkonformität von Operatorenschemata mit Effekteinschränkungen***
Sei $\sigma = \langle \text{na}, \{\text{par}_1, \ldots, \text{par}_k\}, \text{pre}(\{\text{par}_1, \ldots, \text{par}_k\}), \text{eff}(\{\text{par}_1, \ldots, \text{par}_k\}) \rangle$ *ein Operatorenschema und* $\mathcal{O}^{\mathcal{C}_-} = \langle \mathcal{P}_{\text{const}}, \mathcal{V}_{\text{const}}, \mathcal{C}_{\text{eff}} \rangle$ *eine Effekteinschränkung.* σ *ist* ***effektkonform*** *mit* $\mathcal{O}^{\mathcal{C}_-}$*, also* $\mathcal{O}^{\mathcal{C}_-} \models_{ec} \sigma$*, wenn gilt:*

- *Für alle* $\rho \in \mathcal{P}_{\text{const}}$ *gilt* $\rho \notin \text{Sy}(\text{eff}(\sigma))$.
- *Für alle* $\psi \in \mathcal{V}_{\text{const}}$ *gilt* $\psi \notin \text{Sy}(\text{eff}(\sigma))$.
- *Für alle* $c \in \mathcal{C}_{\text{eff}}$ *gilt* $c \models \text{eff}(\sigma)$.

Sonst ähnelt der Aufbau einer Familienbeschreibung stark dem einer Domäne selbst: Die verbliebenen vier Komponenten, die *Anforderungen an die Planungsaufgabenbeschränkungen* sowie diejenigen *an die Prädikaten-, numerischen Variablen- und Operatorenschemata* sind in der selben Form auch Teil einer Domäne. Lediglich eine Art der Planungsaufgabenbeschränkungen, die Typenbeschreibung, hat eine leicht veränderte Form: Da diese in Domänenbeschreibungen fordert, dass alle Objekte aller zugehörigen Planungsaufgaben einem der gegebenen Typen angehören müssen, kann sie so nicht Teil einer Familienbeschreibung sein – schließlich sollen Domänenfamilien ein allen Mitgliedern gemeines Subproblem beschreiben, zusätzliche Funktionalitäten aber nicht unterbinden. Da durch die Typenbeschreibung aber auch Parametertypisierungen ermöglichen soll ein analoges Konstrukt auch Teil von Dömanenfamilien haben:

Definition 5.3 ***Typenbeschreibung in Domänenfamilien***
Sei $\mathcal{P}$ *eine Menge von Prädikatenschemata und* $\rho^1 \in \mathcal{P}$ *einstellig.*
Eine ***Typenbeschreibung*** *einer Domänenfamilie ist eine Planungsaufgabenbeschränkung c der Form:*

$$c = \bigwedge_{\rho^1} \exists\, \omega\, \rho^1(\omega)$$

Alle weiteren Planungsaufgabenbeschränkungen können in Domänenfamilien weitgehend beliebig sein. Üblicherweise sind aber keine Spezialisierungen enthalten, da deren wichtigste Funktion die Generierung von Spezialfällen einer Domänefamilie ist und dementsprechend als Teil von Domänenfamilien wenig Sinn ergeben. Dennoch soll die Möglichkeit, auch Spezialisierungen zu verwenden nicht ausgeschlossen werden, da schließlich auch Unterfamilien von Domänenfamilien definierbar sein sollen, in welchen einzelne Komponenten beschränkt werden.

Definition 5.4 ***Familienbeschreibung***
Eine ***Familienbeschreibung*** *ist ein 5-Tupel* $\widehat{\mathcal{D}}^{\text{Fam}} = \langle \mathcal{P}^{\mathcal{C}}, \mathcal{V}^{\mathcal{C}}, \mathcal{O}^{\mathcal{C}_+}, \mathcal{O}^{\mathcal{C}_-}, \mathcal{C}^{\mathcal{C}} \rangle$ *mit den Komponenten:*

- $\mathcal{P}^{\mathcal{C}}$ *ist die endliche Menge der* ***Anforderungen an die Prädikatenschemata****, einer Menge von Prädikatenschemata.*
- $\mathcal{V}^{\mathcal{C}}$ *ist die endliche Menge der* ***Anforderungen an die numerischen Variablenschemata****, einer Menge numerischer Variablenschemata.*
- $\mathcal{O}^{\mathcal{C}_+}$ *ist die endliche Menge der* ***Anforderungen an die Operatorenschemata****, einer Menge von Operatorenschemata.*
- $\mathcal{O}^{\mathcal{C}_-} = \langle \mathcal{P}_{\text{const}}, \mathcal{V}_{\text{const}}, \mathcal{C}_{\text{eff}} \rangle$ *ist die* ***Effektbeschränkung****, für die gilt:* $\mathcal{P}_{\text{const}} \subseteq \mathcal{P}^{\mathcal{C}}$, $\mathcal{V}_{\text{const}} \subseteq \mathcal{V}^{\mathcal{C}}$ *und* $\mathcal{C}_{\text{eff}}$ *ist eine Menge geschlossener numerischer prädikatenlogischer Formeln mit Prädikatensymbolen aus* $\mathcal{P}^{\mathcal{C}} \setminus \mathcal{P}_{\text{const}}$*, numerischen Variablensymbolen aus* $\mathcal{V}^{\mathcal{C}} \setminus \mathcal{V}_{\text{const}}$ *und Variablensymbolen* $\text{par}_1, \ldots, \text{par}_k$.
- $\mathcal{C}^{\mathcal{C}} = \langle \mathcal{C}_0^{\mathcal{C}}, \mathcal{C}_\star^{\mathcal{C}} \rangle$ *ist die endliche Menge der* ***Anforderungen an die Planungsaufgabenbeschränkungen****, einer Menge von Planungsaufgabenbeschränkungen.*

Für die Mitgliedschaft einer Domäne in einer Familie ist es nötig, dass die ersten beiden Komponenten der Familienbeschreibung Teilmenge der entsprechenden Komponenten der Domäne sind. Mit den Anforderungen an die Operatorenschemata verhält es sich ähnlich, es wird aber eine größere Freiheit für die Modellierung von Domänen gegeben: Für jedes Schema muss in allen Familienmitgliedern mindestens ein Operator

gegeben sein, dessen Vorbedingung die Vorbedingung der Anforderung und dessen Effekt den Effekt der Anforderung enthält. Der Operator der Domäne darf also auch zusätzliche Bedingungen zur Anwendung und Effekte haben. Außerdem kann auch ein einziger Operator alle Anforderungen an die Operatorenschemata erfüllen, solange diese sich nicht widersprechen. Eyerich und andere beschreiben dieses Konzept formal als *abstrakte Subsumierung* von Operatoren [EBN08]. Für die Belange dieser Arbeit ist aber die davon abgeleitete Definition der *Verallgemeinerung von Operatorenschemata* ausreichend:

Definition 5.5 ***Verallgemeinerung von Operatorenschemata***
Seien σ_1 und σ_2 Operatorenschemata über einer Menge von Prädikatenschemata $\mathcal{P}$ und einer Menge numerischer Variablenschemata $\mathcal{V}$ mit $\sigma_i = \langle \mathrm{na}_i, \mathrm{Par}_i, \mathrm{pre}_i(\mathrm{Par}_i), \mathrm{eff}_i(\mathrm{Par}_i)\rangle$.
*o_1 ist eine **Verallgemeinerung des Operatorenschemas** o_2, also $o_1 \subseteq_{as} o_2$ wenn gilt:*

- *Für alle $\rho \in \mathcal{P} \cup \mathcal{V}$ mit $\langle \rho, x\rangle \in \mathrm{Sy}_{\pm}(\mathrm{pre}(\sigma_1))$ gilt $\langle \rho, x\rangle \in \mathrm{Sy}_{\pm}(\mathrm{pre}(\sigma_2))$.*
- *Für alle $\rho \in \mathcal{P} \cup \mathcal{V}$ mit $\langle \rho, x\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff}(\sigma_1))$ gilt $\langle \rho, x\rangle \in \mathrm{Sy}_{\pm}(\mathrm{eff}(\sigma_2))$.*

*Gilt $o_1 \subseteq_{as} o_2$, heißt o_2 **abgeleitet** von o_1.*

Zusammenfassend ergibt sich eine Domänenfamilie aus einer Domänenfamilienbeschreibung also wie folgt:

Definition 5.6 ***Domänenfamilie***
*Sei $\widehat{\mathcal{D}}^{\text{Fam}} = \langle \mathcal{P}^{\mathcal{C}}, \mathcal{O}^{\mathcal{C}_+}, \mathcal{O}^{\mathcal{C}_-}, \mathcal{C}^{\mathcal{C}}\rangle$ mit $\mathcal{O}^{\mathcal{C}_-} = \langle \mathcal{P}_{\text{const}}, \mathcal{V}_{\text{const}}, \mathcal{C}_{\text{eff}}\rangle$ die Familienbeschreibung der **Domänenfamilie** $\mathcal{D}^{\text{Fam}}$, einer Menge von Domänen, und $\mathcal{D} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \overline{w}\rangle$ eine Domäne. $\mathcal{D}$ ist **Mitglied** der Domänenfamilie $\mathcal{D}^{\text{Fam}}$, also $\mathcal{D} \in \mathcal{D}^{\text{Fam}}$, wenn gilt:*

- $\mathcal{P}^{\mathcal{C}} \subseteq \mathcal{P}$
- $\mathcal{V}^{\mathcal{C}} \subseteq \mathcal{V}$
- *Für alle $\sigma_+ \in \mathcal{O}^{\mathcal{C}_+}$ existiert ein Operatorenschema $\sigma \in \mathcal{O}$ mit $\sigma_+ \subseteq_{as} \sigma$.*
- *Für alle $\sigma \in \mathcal{O}$ gilt $\mathcal{O}^{\mathcal{C}_-} \models_{ec} \sigma$*
- $\mathcal{C}_0^{\mathcal{C}} \subseteq \mathcal{C}_0$
- $\mathcal{C}_\star^{\mathcal{C}} \subseteq \mathcal{C}_\star$

Damit sind Domänenfamilien formal definiert, und es stellt sich die Frage nach dem Sinn und Zweck einer solchen Beschreibung. Zur Beantwortung dieser Frage sei noch einmal wiederholt, dass Familien ein allen Mitgliedern gemeines Subproblem beinhalten und im Allgemeinen über einem solchen definiert werden. Alle Domänen, die dieses Problem beinhalten, können dann übersichtlich und kompakt als Spezialfall einer Familie definiert werden, ohne dass die dafür benötigten Schemata noch einmal angeführt werden müssen.
Eine weitere Möglichkeit zur Verwendung ergibt sich folgendermaßen: Angenommen es liegt nun ein Menge intensiv untersuchter Familienbeschreibungen vor, und für jede ergeben sich gewisse Techniken, mit deren Hilfe dieses Subproblem besonders effizient gelöst werden kann, oder auch einfach nur etwas bessere Pläne generiert werden. Dann könnte ein domänenunabhängiger Planer die Mitgliedschaft einer beliebigen Domäne in bekannten Familien überprüfen, und den Planungsvorgang gegebenenfalls entsprechend anpassen. Zwar ist dieser Weg, domänenspezifisches Wissen in domänenunabhängige Planung einfließen zu lassen ein Weg, der bislang meines Wissens von keinem Planer verfolgt wird, es zeigt sich aber in den letzten Jahren immer deutlicher, dass nicht mit der Entdeckung einer "Weltformel" der domänenunabhängigen Handlungsplanung gerechnet werden kann (vgl. [HR08]). Eine formale, maschinenverständliche Kategorisierung von häufig auftretenden Teilproblemen und die dadurch ermöglichte Übertragung domänenspezifischer Erkenntnisse erscheint mir ein Weg, von dem es lohnenswert ist, zumindest zu untersuchen, ob sich

nicht Fortschritte in der Handlungsplanung daraus ergeben könnten.

Aber auch für die Studie von Domänenfamilien selbst kann die Beschreibung hilfreich sein: Häufig ergeben sich diese besonders effizienten Techniken vor allem für vereinfachte Spezialfälle, in denen Teile der Funktionalität eingeschränkt werden. Mithilfe der in Kapitel 3 vorgestellten Spezialisierungsbeschränkungen ist es, durch Anwendung der ebenfalls dort beschriebenen Typeliminierung sehr einfach, beliebige Spezialfälle einer einmal formalisierten Domänenfamilie zu generieren. Natürlich müssen diese trotzdem weiterhin untersucht werden, aber es ergibt sich noch eine weitere Anwendungsmöglichkeit von Domänenfamilienbeschreibungen, die für beliebige Planer nützlich sein kann: Wenn versucht wird, herauszufinden, welche Stärken und Schwächen ein gegebener Planer hat, ist es denkbar, sich mithilfe der in dieser Arbeit beschriebenen Methoden automatisiert zuerst Spezialfälle von Domänenfamilien und zu diesen dann – ebenfalls automatisch – Planungsaufgaben generieren und durch das Planungssystem bearbeiten zu lassen. Dadurch könnten sich Planungsaufgaben herauskristallisieren, in welchen die von dem Planungssystem verwendeten Methoden versagen und entsprechend gegengesteuert werden.

Notwendig dafür ist zuerst die Definition der *Basisdomäne einer Domänenfamilie*, die sich direkt aus der Domänenfamilienbeschreibung ergibt:

Definition 5.7 ***Basisdomäne einer Domänenfamilie***
$\widehat{\mathcal{D}}^{\text{Fam}} = \langle \mathcal{P}^{\mathcal{C}}, \mathcal{V}^{\mathcal{C}}, \mathcal{O}^{\mathcal{C}_+}, \mathcal{O}^{\mathcal{C}_-}, \mathcal{C}^{\mathcal{C}} \rangle$ *eine Domänenfamilienbeschreibung.*
Die Domäne $\mathcal{D} = \langle \mathcal{P}^{\mathcal{C}}, \mathcal{V}^{\mathcal{C}}, \mathcal{O}^{\mathcal{C}_+}, \mathcal{C}^{\mathcal{C}}, \overline{w}$ *mit* $\overline{w}(\sigma) = 1$ *für alle* $\sigma \in \mathcal{O}^{\mathcal{C}_+}$ *heiße* ***Basisdomäne der Domänenfamilie*** $\mathcal{D}^{\text{Fam}}$.
Der relevante Zustandsraum der Domäne $\mathcal{D}$ *heiße* ***relevanter Zustandsraum der Domänenfamilie*** $\mathcal{D}^{\text{Fam}}$ *und sei bezeichnet durch* $S(\mathcal{D}^{\text{Fam}})$.

Verwendet wird von dieser lediglich die Menge der Grundelemente sowie die Zustandsbeschreibung des relevanten Zustandsraumes. Leider kann an dieser Stelle keine vollständige Analyse angegeben werden, wie mit deren Hilfe Spezialfälle konstruiert werden, einige Thesen seien im Folgenden aber formuliert. Dazu sei abermals darauf verwiesen, dass für die Belange dieser Arbeit Einheitskosten vorausgesetzt wurden, und dementsprechend Spezialfälle, welche durch Operatoren unterschiedlicher Kosten entstehen nicht beachtet werden. Allgemein existieren zwei unterschiedlich Arten von Spezialfällen: Solche, die die Basisdomäne vereinfachen, und solche, welche diese komplexer machen. Zuerst seien einige Möglichkeiten der Vereinfachung diskutiert, die alle auch über Beschränkungen der Basisdomäne dargestellt und dementsprechend einfach generiert werden können.

Eine erste Art, Spezialfälle zu konstruieren, ist die Beschränkung konstanter Mengen in $\mathcal{B}$. Dies ist genau die Art, auf welche aus der MultyPersonRoute- die SimpleRoute-Domäne erstellt wurde. Für genau diesen Fall wurde auch der in Kapitel 3 gegebene Algorithmus zur Prädikatenelimination gegeben, mit dessen Hilfe unnötige Parameter aus Operatoren und anderen Schemata gelöscht werden. Diese Form der Spezialisierung existiert analog auch für numerische Variablenschemata:

Definition 5.8 ***Elimination numerischer Variablen***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \overline{w}$ *eine Domäne und* $c_0 \in \mathcal{C}_0$ *eine Initialzustandsbeschränkung.*
c heiße ***Elimination der numerischen Variable*** $\psi \in \mathcal{V}$ *der Domäne* $\mathcal{D}$ *wenn sie darstellbar ist durch:*

$$c_0 = \forall\, \omega_1, \ldots, \omega_k\ ((\psi(\omega_1, \ldots, \omega_k) = \infty) \vee (\psi(\omega_1, \ldots, \omega_k) = \bot))$$

Auch Graphen können in eine bestimmte Form gezwungen werden. Ein häufig verwendetes Konzept ist dabei die Forderung nach vollständigen Graphen:

Definition 5.9 ***Forderung nach Graphenvollständigkeit***
Sei $\mathcal{D} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \overline{w}$ *eine Domäne und* $c_0 \in \mathcal{C}_0$ *eine Initialzustandsbeschränkung. Des Weiteren beinhalte*

die Menge $\mathcal{B}$ des relevanten Zustandsraumes einen Graphen $G = \langle V, E \rangle$, E sei entstanden durch das zweistellige Prädikatenschema $\rho(\text{par}_1, \text{par}_2)$ und $\rho^1 \in \mathcal{P}^{Typ}$ parametertypisiere par_1 und par_2.
*c heiße **Vollständigkeitsforderung** von G, wenn sie darstellbar ist durch:*

$$c_0 = \forall\ \omega_1, \omega_2\ ((\rho^1(\omega_1) \wedge \rho^2(\omega_1)) \Rightarrow \rho(\omega_1, \omega_2))$$

Natürlich können Graphen auch anders vereinfacht werden, wie zum Beispiel durch die Forderung, zweifach hierarchisch-vollständig zu sein. Dies zu formalisieren ist aber nicht das Ziel dieses Abschnittes, es sollte lediglich die grundsätzliche Idee vermittelt werden. Zudem wäre in diesem Fall eine andere Form der Spezialisierung zusätzlich erforderlich, die die Basisdomäne erschwert: Liegt ein solcher Graph vor, muss sowohl ein Typenprädikat der Basisdomäne Supertyp mehrerer Prädikatenschemata sein, und zusätzlich muss mindestens ein Operatorenschema mehrfach abgeleitet werden. Dies sind auch die beide am häufigsten verwendeten Ansätze, komplexere Spezialfälle einer Domäne zu generieren, die aber meist nur im Zusammenspiel mit weiteren Veränderungen auch zu sinnvoll abgeleiteten Domänen führen.

Aus diesem Grund sei der Versuch, über Beschränkungen Spezialisierungen von Domänenfamilien automatisch zu generieren bereits abgeschlossen. Es zeigt sich, dass für eine allgemeingültige Methode, funktionierende, spezialisierte Domänen aus einer Familienbeschreibung zu generieren noch viele Fragen unbeantwortet bleiben. Dennoch kann bereits jetzt gesagt werden, dass Familienbeschreibungen auch zur manuellen Erstellung von Spezialfällen hilfreich sind, da über den relevanten Zustandsraum der Basisdomäne alle wichtigen Konzepte eines Subproblemes leicht ersichtlich werden. Zudem helfen Domänenfamilienbeschreibungen, Domänen übersichtlicher darzustellen, ohne dass diese dadurch funktional verändert werden. Genauer zu sehen sein wird dies in den beiden Kapiteln über die Miconic-10-Strips- und die Logistics-Domäne, in welchen beide als Spezialfall der Transportplanungsfamilie kompakt beschrieben werden.

5.2 Routenplanungsfamilie

Das Ziel von Planungsaufgaben der Routenplanungsfamilie $\mathcal{D}^{\text{Route}}$ ist dem Namen leicht zu entnehmen – jede Mitglied enthält das Subproblem, eine Route zu planen. Notwendig ist dazu eine Menge von Fahrzeugen sowie miteinander verbundene Orten, über welche die Fahrzeuge sich bewegen können. Die folgenden vier Eigenschaften gilt es also, in einer Domänenfamilienbeschreibung zu modellieren:

- Es existiert eine endliche Menge von **Orten**, die über **Straßen** miteinander verbunden sind.
- Es existiert eine endliche Menge von **Fahrzeugen**, die über die Straßen von Ort zu Ort bewegt werden können.
- Beide Mengen sind disjunkt, kein Objekt ist also gleichzeitig ein Fahrzeug und ein Ort.
- Planungsaufgaben bestehen darin, eine Menge von **Zielorten** in beliebiger Reihenfolge mit einem Fahrzeug anzufahren.

Ebenso wenig wie die Modellierung einer Domäne selbst ist auch die Modellierung von Domänenfamilienbeschreibungen eindeutig.

Definition 5.10 ***Familie der Routenplanungsprobleme***
Die ***Familie der Routenplanungsprobleme*** $\mathcal{D}^{\text{ROUTE}}$ *ist gegeben durch die Familienbeschreibung* $\widehat{\mathcal{D}}^{\text{ROUTE}} = \langle \mathcal{P}^{\mathcal{C}}_{\text{ROUTE}}, \mathcal{V}^{\mathcal{C}}_{\text{ROUTE}}, \mathcal{O}^{\mathcal{C}_+}_{\text{ROUTE}}, \mathcal{O}^{\mathcal{C}_-}_{\text{ROUTE}}, \mathcal{C}^{\mathcal{C}}_{\text{ROUTE}} \rangle$ *mit den Komponenten:*

- $\mathcal{P}^{\mathcal{C}}_{\text{ROUTE}} = \{\rho_1, \ldots, \rho_6\}$*:*
  - $\rho_1 = \langle \texttt{location}, 1 \rangle$
  - $\rho_2 = \langle \texttt{mobile}, 1 \rangle$
  - $\rho_3 = \langle \texttt{connected}, 2 \rangle$
  - $\rho_4 = \langle \texttt{visited}, 1 \rangle$
  - $\rho_5 = \langle \texttt{at}, 2 \rangle$
  - $\rho_6 = \langle \texttt{targetlocation}, 1 \rangle$
- $\mathcal{V}^{\mathcal{C}}_{\text{ROUTE}} = \{\langle \texttt{Fuel}, 1 \rangle\}$
- $\mathcal{O}^{\mathcal{C}_+}_{\text{ROUTE}} = \{\sigma_1, \sigma_2\}$*:*
  - $\sigma_1 = \langle \texttt{move}, \{m, l_1, l_2\}, \text{pre}(m, l_1, l_2), \text{eff}(m, l_1, l_2) \rangle$*, wobei*
    $\text{pre}(m, l_1, l_2) = \texttt{connected}(l_1, l_2) \wedge \texttt{at}(m, l_1) \wedge \texttt{Fuel}(m) \geq 1$ *und*
    $\text{eff}(m, l_1, l_2) = \texttt{at}(m, l_2) \wedge \neg\texttt{at}(m, l_1) \wedge \texttt{Fuel}(m) \rightarrow \texttt{Fuel}(m) - 1$
  - $\sigma_2 = \langle \texttt{markVisited}, \{m, l\}, \text{pre}(m, l), \text{eff}(m, l) \rangle$*, wobei*
    $\text{pre}(m, l) = \texttt{at}(m, l) \wedge \texttt{targetlocation}(l)$ *und*
    $\text{eff}(m, l) = \texttt{visited}(l)$
- $\mathcal{O}^{\mathcal{C}_-}_{\text{ROUTE}} = \langle \mathcal{P}_{\text{const}}, \mathcal{V}_{\text{const}}, \mathcal{C}_{\text{eff}} \rangle$ *mit* $\mathcal{C}_{\text{eff}} = \{c^1_{\text{eff}}\}$*:*
  - $\mathcal{P}_{\text{const}} = \{\texttt{location}, \texttt{mobile}, \texttt{connected}, \texttt{targetlocation}\}$
  - $\mathcal{V}_{\text{const}} = \emptyset$
  - $c^1_{\text{eff}} := \forall\, m(\exists^{=n}\, l_1\ (\texttt{at}(m, l_1) \Rightarrow \exists^{=n}\, l_2\ \neg\texttt{at}(m, l_2)))$
- $\mathcal{C}^{\mathcal{C}}_{\text{ROUTE}} = \langle \mathcal{C}^{\text{ROUTE}}_0, \mathcal{C}^{\text{ROUTE}}_\star \rangle$ *mit* $\mathcal{C}^{\text{ROUTE}}_0 = \{c^1_0, \ldots, c^9_0\}$ *und* $\mathcal{C}^{\text{ROUTE}}_\star = \{c_\star\}$*:*
  - $c^1_0 = (\exists\, m\ \texttt{mobile}(m)) \wedge (\exists\, l\ \texttt{location}(l))$
  - $c^2_0 = \forall\, \omega\ (\neg(\texttt{mobile}(\omega) \wedge \texttt{location}(\omega))$
  - $c^3_0 = \forall\, l_1, l_2\ (\texttt{connected}(l_1, l_2) \Rightarrow (\texttt{location}(l_1) \wedge \texttt{location}(l_2)))$
  - $c^4_0 = \forall\, l\ (\texttt{visited}(l) \Rightarrow \texttt{targetlocation}(l)$
  - $c^5_0 = \forall\, m, l\ (\texttt{at}(m, l) \Rightarrow (\texttt{mobile}(m) \wedge \texttt{location}(l)))$
  - $c^6_0 = \forall\, l\ (\texttt{targetlocation}(l) \Rightarrow \texttt{location}(l))$
  - $c^7_0 = \forall\, m\ (\texttt{Fuel}(m) \neq \bot \Rightarrow \texttt{mobile}(m))$
  - $c^8_0 = \forall\, m\ (\texttt{mobile}(m) \Rightarrow (\exists^{=1} l\ (\texttt{location}(l) \wedge \texttt{at}(m, l))))$
  - $c^9_0 = \forall\, l\ (\texttt{targetlocation}(l) \Rightarrow \neg\texttt{visited}(l))$
  - $c_\star = \forall\, l\ (\texttt{targetlocation}(l) \Rightarrow \texttt{visited}(l))$

Eine Eigenschaft in dieser Domäne ist dabei auch in der Literatur nicht eindeutig gegeben: Es existieren Definitionen, in welchen die `Fuel`-Variable wie in diesem Fall für Fahrzeuge definiert ist, aber auch solche, in welchen sie über den Orten gegeben ist. Beide Möglichkeiten wurden bereits in IPC Domänen (die aber den Transportplanungsproblemen zugehören) verwendet, und sie war zeitweise auch in dieser Definition in beiden Varianten beinhaltet. Da der Nutzen aber in keiner Relation zur erhöhten Komplexität der Beschreibung liegt wurde letztendlich hier wie auch bei der Definition von Transportplanungsproblemen darauf verzichtet und der meiner Meinung nach intuitivere Weg mit Fahrzeugen bevorzugt. Zudem implementiert keine der beiden Routenplanungsdomänen, welche einschließlich des IPC 5 jemals Teil des Internationalen Planungswettbewerbs gewesen sind, das Treibstoffproblem.

- ROVERS (IPC 3 & 5): In dieser Domäne wird die Arbeit von Rovern auf einem Planeten simuliert. Dabei müssen diese zwischen Orten bewegt, Analysen von Gestein und Boden durchgeführt sowie mit einer Kamera Bilder bestimmter Gegenden aufgenommen werden. ($\in$ poly-APX $\setminus$ PTAS)
- AIRPORT (IPC 4): In dieser Domäne wird der Bodenverkehr eines Flughafens simuliert. Landende und startende Flugzeuge müssen durch ein Wegenetz geleitet und Sicherheitsabstände dabei eingehalten werden. ($\in$ EXPO $\setminus$ NPS)

Die in Klammern gegebenen Komplexitätsklassen stammen, wie auch im nächsten Abschnitt, alle aus Helmerts Untersuchungen dieser Familien in [Hel08]. Domänen wie ROVERS sind im Übrigen dafür verantwortlich, dass das Bewegen eines Fahrzeuges auf einen Ort nicht in allen Mitgliedern ausreicht, um den Ort als besucht zu markieren: Häufig sind zusätzliche Aufgaben gegeben, die dort erst erfüllt werden müssen, die dann mit der Routenplanung selbst aber nichts zu tun haben. Natürlich kann ein Mitglied der Domänenfamilie aber auch einfach beide Operatorenschemata in einem ableiten, wodurch ein Ort direkt bei Betreten eines Fahrzeuges als besucht markiert wird. Wird nun der in Kapitel 3 gegebene Algorithmus zur Generierung einer effizienten Zustandsbeschreibung auf die durch diese Familienbeschreibung implizierte Basisdomäne angewandt, entsteht der folgende relevante Zustandsraum:

Definition 5.11 ***Relevanter Zustandsraum der Routenplanungsfamilie***
*Der **relevante Zustandsraum der Routenplanungsfamilie** $S(\mathcal{D}^{\text{ROUTE}})$ ist gegeben als:*

GRUNDELEMENTE: $\langle W, M, L^{\oplus}\rangle$*, wobei* $W = \langle L, E\rangle$ *der Wegenetzgraph mit der Menge der Orte* L *und den Straßen* E*,* M *die Menge der Fahrzeuge und* $L^{\oplus} \subseteq L$ *die Menge der Zielorte ist.*

ZUSTÄNDE: *3-Tupel* $\langle \text{loc}, \text{fuel}, L^{\otimes}\rangle$*, wobei* $\text{loc} : M \to L$ *die Standortfunktion,* $\text{fuel} : M \to \mathbb{N}_0 \cup \{\infty, \bot\}$ *die Treibstofffunktion und* $L^{\otimes} \subseteq L^{\oplus}$ *die Menge der besuchten Zielorte ist.*

STARTZUSTAND: $\langle \text{loc}_0, \text{fuel}_0, \emptyset\rangle$

ZIELZUSTAND: *Jeder Zustand* $\langle \text{loc}, \text{fuel}, L^{\otimes}\rangle$ *mit* $L^{\otimes} = L^{\oplus}$.

OPERATOREN: *Ein Fahrzeug kann sich von einem Ort zu einem anderen **bewegen**, falls eine Straße zwischen diesen existiert und einen Zielort als besucht **markieren**, falls es sich an diesem befindet.*

Es sei an dieser Stelle angemerkt, dass die Beschreibung der Operatoren für einen relevanten Zustandsraum von Domänenfamilien nicht sehr aussagekräftig ist, da diese lediglich Verallgemeinerungen von Operatoren konkreter Familienmitglieder sind, welche durch zusätzliche Vorbedingungen und Effekte viele weitere Beschränkungen und Auswirkungen haben können. Dennoch geben sie ein grundsätzliches Bild des durch die Familienbeschreibung gegebenen Subproblems und sind deswegen bewusst mit angegeben.

Eine Analyse der gegebenen Grundelemente sowie der Zustandsbeschreibungsvariablen gibt ein genaueres Bild über mögliche Spezialfälle des Routenplanungsproblems. In diesem Fall existieren die folgenden, mit Einheitskosten modellierbaren Spezialfälle:

- Der Wegenetzgraph kann beliebig oder komplett sein (oder eine andere, besondere Form annehmen).
- Die Menge der Fahrzeuge kann auf genau ein Fahrzeug, beliebig viele Fahrzeuge eines Typs oder beliebig viele Fahrzeuge beliebig vieler Typen beschränkt werden.
- Die Treibstofffunktion kann gar nicht, für alle Fahrzeuge identisch, oder für verschiedene Fahrzeugtypen unterschiedlich definiert sein.

Es ist bei dieser Auflistung durchaus beabsichtigt, dass die Knotenmenge des Wegenetzgraphen nicht beschränkt werden kann, um dennoch eine Spezialisierung der Routenplanungsfamilie zu erhalten. Allgemeine kann vermutet werden, dass das Subproblem von Domänenfamilien zwar durch Beschränkung der Verbindungen enthaltener Graphen erhalten bleibt, eine Anzahlbeschränkung der Knotenmenge aber das Subproblem zunichte machen.
Damit kann auch die SimpleRoute-Domäne auf ihre Zugehörigkeit zu dieser Familie überprüft werden: Bei dieser handelt es sich um eine Routenplanungsdomäne, deren Wegenetzgraph beliebig ist, die auf genau ein Fahrzeuge beschränkt ist und deren Treibstofffunktion nicht definiert ist (also ∞ für alle Fahrzeuge). Dennoch gibt es einen Punkt, der SimpleRoute nicht zu einer Routenplanungsdomäne macht: Es wird gefordert, dass zuerst ein Gegenstand abgeholt wird, und dann ein Zielort erreicht wird. Routenplanungsprobleme sind aber so gegeben, dass Zielorte in beliebiger Reihenfolge angefahren werden können. Daher handelt es sich bei der SimpleRoute-Domäne eher um ein Transportplanungsproblem.

5.3 Transportplanung

Auch diese Familie ist durch ihren Namen bereits äußerst gut beschrieben. Planungsaufgabe haben zur Aufgaben, mit Fahrzeugen Pakete an Orten abzuholen und an andere Orte zu bringen. Übertragen auf die SimpleRoute-Domäne muss der Gegenstand also abgeholt werden und dann an den Zielort gebracht werden – dies wurde natürlichsprachlich zwar anders beschrieben, ist in Bezug auf die Funktionalität aber äquivalent. Die Familie der Transportplanungsprobleme kann folgendermaßen definiert werden:

Definition 5.12 ***Familie der Transportplanungsprobleme***
*Die **Familie der Transportplanungsprobleme** $\mathcal{D}^{\text{Trans}}$ ist gegeben durch die Familienbeschreibung $\widehat{\mathcal{D}}^{\text{Trans}} = \langle \mathcal{P}^{\mathcal{C}}_{\text{Trans}}, \mathcal{V}^{\mathcal{C}}_{\text{Trans}}, \mathcal{O}^{\mathcal{C}_+}_{\text{Trans}}, \mathcal{O}^{\mathcal{C}_-}_{\text{Trans}}, \mathcal{C}^{\mathcal{C}}_{\text{Trans}} \rangle$ mit den Komponenten:*

- $\mathcal{P}^{\mathcal{C}}_{\text{Trans}} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$:
  - $\rho_1 = \langle \texttt{portLoc}, 1 \rangle$
  - $\rho_2 = \langle \texttt{location}, 1 \rangle$
  - $\rho_3 = \langle \texttt{mobile}, 1 \rangle$
  - $\rho_4 = \langle \texttt{portable}, 1 \rangle$
  - $\rho_5 = \langle \texttt{connected}, 2 \rangle$
  - $\rho_6 = \langle \texttt{mobileAt}, 2 \rangle$
  - $\rho_7 = \langle \texttt{portableAt}, 2 \rangle$
  - $\rho_8 = \langle \texttt{portableDestination}, 2 \rangle$
- $\mathcal{V}^{\mathcal{C}}_{\text{Trans}} = \{\langle \psi_1, \psi_2, 1 \rangle\}$:
  - $\psi_1 = \langle Fuel, 1 \rangle$
  - $\psi_2 = \langle Capacity, 1 \rangle$
- $\mathcal{O}^{\mathcal{C}_+}_{\text{Trans}} = \{\sigma^1_{\text{Trans}}, \sigma^2_{\text{Trans}}, \sigma^3_{\text{Trans}}\}$:
  - $\sigma^1_{\text{Trans}} = \langle \texttt{move}, \{m, l_1, l_2\}, \text{pre}(m, l_1, l_2), \text{eff}(m, l_1, l_2) \rangle$, *wobei*
    $\text{pre}(m, l_1, l_2) = \texttt{connected}(l_1, l_2) \wedge \texttt{mobileAt}(m, l_1) \wedge \texttt{Fuel}(m) \geq 1$ *und*
    $\text{eff}(m, l_1, l_2) = \texttt{mobileAt}(m, l_2) \wedge \neg\texttt{mobileAt}(m, l_1) \wedge \texttt{Fuel}(m) \rightarrow \texttt{Fuel}(m) - 1$
  - $\sigma^2_{\text{Trans}} = \langle \texttt{pickUp}, \{m, p, l\}, \text{pre}(m, p, l), \text{eff}(m, p, l) \rangle$, *wobei*
    $\text{pre}(m, p, l) = \texttt{mobileAt}(m, l) \wedge \texttt{portableAt}(p, l) \wedge \texttt{Capacity}(m) \geq 1$ *und*
    $\text{eff}(m, p, l) = \texttt{portableAt}(p, m) \wedge \neg\texttt{portableAt}(p, l) \wedge \texttt{Capacity}(m) \rightarrow \texttt{Capacity}(m) - 1$
  - $\sigma^3_{\text{Trans}} = \langle \texttt{drop}, \{m, p, l\}, \text{pre}(m, p, l), \text{eff}(m, p, l) \rangle$, *wobei*
    $\text{pre}(m, p, l) = \texttt{mobileAt}(m, l) \wedge \texttt{portableAt}(p, m)$ *und*
    $\text{eff}(m, p, l) = \texttt{portableAt}(p, l) \wedge \neg\texttt{portableAt}(p, m) \wedge \texttt{Capacity}(m) \rightarrow \texttt{Capacity}(m) + 1$

- $\mathcal{O}^{\mathcal{C}_-}_{\text{TRANS}} = \langle \mathcal{P}_{\text{const}}, \mathcal{V}_{\text{const}}, \mathcal{C}_{\text{eff}} \rangle$ *mit* $\mathcal{C}_{\text{eff}} = \{c^1_{\text{eff}}\}$*:*
  - $\mathcal{P}_{\text{const}} = \{\mathit{portLoc}, \mathit{location}, \mathit{mobile}, \mathit{portable}, \mathit{connected}, \mathit{portableGoal}\}$
  - $\mathcal{V}_{\text{const}} = \emptyset$
  - $c^1_{\text{eff}} := \forall\, m(\exists^{=n}\, l_1\ \mathtt{at}(m, l_1) \Rightarrow \exists^{=n}\, l_2\ \neg \mathtt{at}(m, l_2)$
- $\mathcal{C}^{\mathcal{C}}_{\text{TRANS}} = \langle \mathcal{C}^{\text{TRANS}}_0, \mathcal{C}^{\text{TRANS}}_\star \rangle$ *mit* $\mathcal{C}^{\text{TRANS}}_0 = \{c^1_0, \ldots, c^1_0 4\}$ *und* $\mathcal{C}^{\text{TRANS}}_\star = \{c_\star\}$*:*
  - $c^1_0 = (\exists\, \omega\ \mathit{portLoc}(\omega)) \wedge (\exists\, m\ \mathit{mobile}(m)) \wedge (\exists\, l\ \mathit{location}(l)) \wedge (\exists\, p\ \mathit{portable}(p))$
  - $c^2_0 = \forall\, l\ \mathit{location}(l) \Rightarrow \mathit{portLoc}(l)$
  - $c^3_0 = \forall\, m\ \mathit{mobile}(m) \Rightarrow \mathit{portLoc}(m)$
  - $c^4_0 = \forall\, \omega\ (\neg(\mathit{location}(\omega) \wedge \mathit{mobile}(\omega))$
  - $c^5_0 = \forall\, \omega\ (\neg(\mathit{location}(\omega) \wedge \mathit{portable}(\omega))$
  - $c^6_0 = \forall\, \omega\ (\neg(\mathit{mobile}(\omega) \wedge \mathit{portable}(\omega))$
  - $c^7_0 = \forall\, l_1, l_2\ (\mathit{connected}(l_1, l_2) \Rightarrow (\mathit{location}(l_1) \wedge \mathit{location}(l_2)))$
  - $c^8_0 = \forall\, m, l\ (\mathit{mobileAt}(m, l) \Rightarrow (\mathit{mobile}(m) \wedge \mathit{location}(l)))$
  - $c^9_0 = \forall\, p, \omega\ (\mathit{portableAt}(p, \omega) \Rightarrow (\mathit{portable}(p) \wedge \mathit{portLoc}(\omega)))$
  - $c^9_0 = \forall\, p, \omega\ (\mathit{portableGoal}(p, \omega) \Rightarrow (\mathit{portable}(p) \wedge \mathit{portLoc}(\omega)))$
  - $c^{10}_0 = \forall\, m\ (\mathit{Fuel}(m) \neq \bot \Rightarrow \mathit{mobile}(m))$
  - $c^{11}_0 = \forall\, m\ (\mathit{Capacity}(m) \neq \bot \Rightarrow \mathit{mobile}(m))$
  - $c^{12}_0 = \forall\, m\ (\mathit{mobile}(m) \Rightarrow (\exists^{=1} l\ \mathit{mobileAt}(m, l)))$
  - $c^{13}_0 = \forall\, p\ (\mathit{portable}(p) \Rightarrow (\exists^{=1} \omega\ \mathit{portableAt}(m, l)))$
  - $c^{14}_0 = \forall\, p\ (\mathit{portable}(p) \Rightarrow (\exists^{=1} \omega\ \mathit{portableDestination}(m, l)))$
  - $c^{15}_0 = \forall p\ \mathit{portable}(p) \Rightarrow (\neg\ \exists\, l\ (\mathit{portableAt}(p, l) \wedge \mathit{portableDestination}(p, l)))$
  - $c^{16}_0 = \forall p\ \mathit{portable}(p) \Rightarrow (\neg\ \exists\, m\ (\mathit{portableAt}(p, m) \wedge \mathit{mobile}(m)))$
  - $c_\star = \forall p\ \mathit{portable}(p) \Rightarrow (\exists^{=1}\ l\ (\mathit{portableAt}(p, l) \wedge \mathit{portableDestination}(p, l)))$

Man sieht, dass $\mathcal{D}^{\text{ROUTE}}$ und $\mathcal{D}^{\text{TRANS}}$ große gemeinsame Teile in ihrer Beschreibung aufweisen, und tatsächlich kann das Routenplanungsproblem auch als Subproblem des Transportplanungsproblems angesehen werden. Der relevante Zustandsraum der Transportplanungsfamilie ist folgendermaßen gegeben:

Definition 5.13 ***Relevanter Zustandsraum der Transportplanungsfamilie***
Der ***relevante Zustandsraum der Transportplanungsfamilie*** $S(\mathcal{D}^{\text{TRANS}})$ *ist gegeben als:*

GRUNDELEMENTE: $\langle W, M, P, L^\star \rangle$*, wobei* $W = \langle L, E \rangle$ *der Wegenetzgraph ist mit der Menge der Orte* L *und den Straßen* E*,* M *die Menge der Fahrzeuge,* P *die Menge der Pakete, und* $D = \langle \text{loc}_0, \text{loc}_\star \rangle$ *der* ***Liefergraph*** *ist mit der* ***Startortfunktion*** $\text{loc}_0 : P \to L$ *und der* ***Zielortfunktion*** $\text{loc}_\star : P \to L$*.*

ZUSTÄNDE: *4-Tupel* $\langle \text{mAt}, \text{pAt}, \text{fuel}, \text{capcity} \rangle$*, wobei* $\text{mAt} : M \to L$ *die Standortfunktion der Fahrzeuge,* $\text{pAt} : P \to M \cup L$ *die Ortsfunktion der Pakete,* $\text{fuel} : M \to \mathbb{N}_0 \cup \{\infty, \bot\}$ *die Treibstofffunktion und* $\text{capcity} : M \to \mathbb{N}_0 \cup \{\infty, \bot\}$ *die Kapazitätsfunktion ist.*

STARTZUSTAND: $\langle \text{mAt}_0, \text{pAt}, \text{fuel}_0, \text{capacity}_0 \rangle$ *mit* $\text{pAt}(p) = \text{loc}_0(p)$ *für alle* $p \in P$

ZIELZUSTAND: *Jeder Zustand* $\langle \text{mAt}, \text{pAt}, \text{fuel}, \text{capacity} \rangle$ *mit* $\text{pAt}(p) = \text{loc}_\star(p)$*.*

OPERATOREN: *Ein Fahrzeug kann sich von einem Ort zu einem anderen* ***bewegen****, falls eine Straße zwischen diesen existiert, ein Paket* ***einladen****, wenn diese sich am selben Ort befinden und ein Paket* ***ausladen****, wenn es dieses geladen hat.*

Es wird darauf verzichtet, die einzelnen Bestandteile an dieser Stelle vorzustellen, da diese in den beiden folgenden Kapiteln bereits untersucht werden. Daher seien nur noch mögliche Spezialfälle der Familie angegeben:

- Der Wegenetzgraph kann beliebig oder vollständig sein (oder eine andere, besondere Form annehmen).
- Die Menge der Fahrzeuge kann auf genau ein Fahrzeug, beliebig viele Fahrzeuge eines Typs oder beliebig viele Fahrzeuge beliebig vieler Typen beschränkt werden.
- Die Treibstofffunktion kann gar nicht, für alle Fahrzeuge identisch, oder für verschiedene Fahrzeugtypen unterschiedlich definiert sein.
- Die Kapazitätsfunktion kann gar nicht, für alle Fahrzeuge identisch, oder für verschiedene Fahrzeugtypen unterschiedlich definiert sein.

Eine Beschränkung des Liefergraphen ist, wie auch des Wegenetzgraphen, nicht möglich, wenn das Subproblem erhalten bleiben soll. Die folgenden Transportplanungsdomänen waren, einschließlich des IPC 5, bereits Teil eines Planungswettbewerbs:

- GRID (IPC 1): In dieser Domäne existiert ein **Roboter**, der sich innerhalb einer **Gitternetzwelt** bewegt und dabei **Objekte** aufsammelt und zu gegebenen **Feldern** transportiert. Manche solcher Felder sind unpassierbar und können nur durch **Schlüssel** geöffnet werden. ($\in$ poly-APX \ APX)
- MYSTERY (IPC 1): Objekte dieser Welt sind vom Typ **Ort**, **Fahrzeug** oder **Ladung**. Jeder Ort stellt eine gegebene Menge an **Treibstoff** bereit, und Fahrzeuge können sich nur von einem Knoten wegbewegen, wenn dort noch solcher vorhanden ist. Dabei wird eine Einheit des Treibstoffs verbraucht. Das Ziel besteht darin, Ladung an ihren Zielort zu bewegen. ($\in$ NPO \ PS)
- MYSTERYPRIME (IPC 1): Diese Domäne entspricht der MYSTERY-Domäne, es kann aber zusätzlich Treibstoff zwischen Knoten ausgetauscht werden. ($\in$ NPO \ PS)
- LOGISTICS (IPC 1 & 2): Diese Welt besteht aus **Städten**, die wiederum aus **Orten** bestehen. **Flugzeuge** sowie **Lastwagen** transportieren **Pakete** von einem Ort zum anderen, wobei sich Flugzeuge lediglich zwischen **Flughäfen** bewegen können und Trucks lediglich zwischen Orten innerhalb einer Stadt. (s. Kap. 7)
- MICONIC-10 (IPC 2): Aufgaben in dieser Domäne bestehen daraus, **Passagiere** mit einem **Fahrstuhl** zwischen verschiedenen **Stockwerken** zu befördern. Es existieren drei Varianten dieser Domäne, inklusive einer, in der zusätzliche Beschränkungen wie VIP-Service bestehen. (s. Kap. 6)
- DEPOTS (IPC 3): Diese Domäne besteht aus **Orten** und **Lastwagen** auf der einen sowie **Blöcken** und **Paletten** auf der anderen Seite. Die Lastwagen bringen die Blöcke an ihren Zielort, wo mehrarmige **Kräne** diese in vorgegebener Reihenfolge auf die Paletten stapeln. ($\in$ APX \ PTAS)
- DRIVERLOG (IPC 3): Diese Probleme bestehen daraus, mit **Lastwagen Pakete** an bestimmte **Orte** zu befördern. Zusätzlich benötigt jeder Lastwagen einen **Fahrer**, ohne den dieser nicht bewegt werden kann. ($\in$ poly-APX \ APX)
- ZENOTRAVEL (IPC 3): Probleme dieser Domäne bestehen daraus, **Passagiere** mithilfe von **Flugzeugen** zu bestimmten **Orten** zu transportieren. Abhängig von der **Geschwindigkeit** des Flugzeuges wird dabei mehr oder weniger **Treibstoff** verbraucht.($\in$ APX \ PTAS)

- TRUCKS (IPC 5): Probleme dieser Domäne bestehen daraus, **Pakete** zu den richtigen **Orten** zu transportieren. Erschwert wird dies dadurch, dass das Abladen der Pakete nur möglich ist, wenn ausreichend **Platz** dazu vorhanden ist. Zusätzlich ist es möglich, **Deadlines** für Pakete zu formulieren.

Es zeigt sich, dass auch innerhalb einer Familie sehr unterschiedliche Aufgaben bewältigt werden müssen, und trotzdem in allen das gemeinsame Subproblem enthalten ist. Die beiden folgenden Kapitel werden zwei dieser Domänen, MICONIC-10 und LOGISTICS näher in Bezug auf optimale Planung untersuchen und einen implementierten Planer beschreiben.

6 Miconic-10-Strips

Im ersten Abschnitt dieses Kapitels wird die Miconic-10-Strips-Domäne vorgestellt und formal beschrieben. Darauf folgt die Beschreibung zweier Algorithmen zur Ermittlung von suboptimalen bzw. optimalen Plänen für Planungsaufgaben der Domäne. Zudem werden deren Eigenschaften detailliert beschrieben und so grundlegende Erkenntnisse für die Transportplanung gewonnen. Abgeschlossen wird dieses Kapitel mit einer kurzen Vorstellung der Ergebnisse, die mit dem implementierten, optimalen Planer erzielt wurden.

6.1 Formalisierung

Die Miconic-10-Strips-Domäne wurde erstmals durch Köhler und Schuster beschrieben [KS00] und erhielt ihren Namen von einem Aufzugssystem der Schweizer Firma Schindler. Im Jahr 2000 war sie als Mitglied der Miconic-10 genannten Domänenfamilie, der auch Miconic-10-SimpleADL und Miconic-10-FullADL angehören, Teil des IPC 2. Diese Arbeit beschäftigt sich dabei lediglich mit Miconic-10-Strips, da die Unterschiede zu den anderen beiden Domänen im Hinblick auf Transportplanungsprobleme vernachlässigbar sind. Planungsaufgaben haben zum Ziel, Passagiere mithilfe eines Fahrstuhls von ihrem Startstockwerk aus zu einem Zielstockwerk zu transportieren. Dabei steht immer lediglich ein einziges Fahrzeug unbegrenzter Kapazität und ohne Treibstoffverbrauch zur Verfügung, das Aufzug genannt wird.

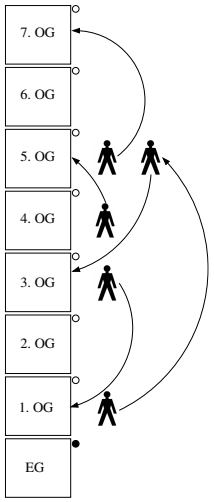

Abbildung 6.1: Bildliche Darstellung einer Miconic-10-Strips-Planungsaufgabe

In diesem Beispiel existieren mit dem Erdgeschoss und sieben Obergeschossen acht Stockwerke. Der Aufzug, verdeutlicht durch einen schwarzen Kreis neben dem Stockwerk, ist im gegebenen Zustand im Erdgeschoss, und hat die Aufgabe, fünf Passagiere von der Etage, neben der sie abgebildet sind, an das durch einen Pfeil gekennzeichnete Zielstockwerk zu transportieren. Damit beinhaltet dieses einfache Bild bereits alle Informationen, die eine Miconic-10-Strips-Planungsaufgabe definieren, es fehlt lediglich der Wegenetzgraph. Nun würde man intuitiv erwarten, dass jedes Stockwerk mit den direkt darüber und darunter liegenden verbunden ist. Um dies zu klären sei erst ein Blick auf die Domänendefinition geworfen, die zum größten Teil direkt aus der PDDL-Beschreibung des IPC 2 generiert wurde. Lediglich die angegebenen Be-

schränkungen wurden mithilfe der zusätzlichen Beschreibung in natürlicher Sprache sowie einer Analyse aller Planungsaufgaben entworfen.

Definition 6.1 MICONIC-10-STRIPS-*Domäne*
Die **MICONIC-10-STRIPS**-*Domäne* $\mathcal{D}_{\text{M10}} = \langle \mathcal{P}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *ist (analog zum IPC 2) gegeben durch:*

- $\mathcal{P} = \{\rho_1, \ldots, \rho_8\}$*:*
  - $\rho_1 = \langle \mathit{floor}, 1 \rangle$
  - $\rho_2 = \langle \mathit{passenger}, 1 \rangle$
  - $\rho_3 = \langle \mathit{origin}, 2 \rangle$
  - $\rho_4 = \langle \mathit{destin}, 2 \rangle$
  - $\rho_5 = \langle \mathit{above}, 2 \rangle$
  - $\rho_6 = \langle \mathit{boarded}, 1 \rangle$
  - $\rho_7 = \langle \mathit{served}, 1 \rangle$
  - $\rho_8 = \langle \mathit{lift\text{-}at}, 1 \rangle$
- $\mathcal{O} = \{\sigma_1, \ldots, \sigma_4\}$*:*
  - $\sigma_1 = \langle \mathit{board}, \{f, p\}, \text{pre}(f, p), \text{eff}(f, p) \rangle$*, wobei*
    $\text{pre}(f, p) = \mathit{lift\text{-}at}(f) \wedge \mathit{origin}(p, f)$ *und*
    $\text{eff}(f, p) = \mathit{boarded}(p)$
  - $\sigma_2 = \langle \mathit{depart}, \{f, p\}, \text{pre}(f, p), \text{eff}(f, p) \rangle$*, wobei*
    $\text{pre}(f, p) = \mathit{lift\text{-}at}(f) \wedge \mathit{destin}(p) \wedge \mathit{boarded}(p)$ *und*
    $\text{eff}(f, p) = \neg \mathit{boarded}(p) \wedge \mathit{served}(p)$
  - $\sigma_3 = \langle \mathit{up}, \{f_1, f_2\}, \text{pre}(f_1, f_2), \text{eff}(f_1, f_2) \rangle$*, wobei*
    $\text{pre}(f_1, f_2) = \mathit{lift\text{-}at}(f_1) \wedge \mathit{above}(f_1, f_2)$ *und*
    $\text{eff}(f_1, f_2) = \neg \mathit{lift\text{-}at}(f_1) \wedge \mathit{lift\text{-}at}(f_2)$
  - $\sigma_4 = \langle \mathit{down}, \{f_1, f_2\}, \text{pre}(f_1, f_2), \text{eff}(f_1, f_2) \rangle$*, wobei*
    $\text{pre}(f_1, f_2) = \mathit{lift\text{-}at}(f_1) \wedge \mathit{above}(f_2, f_1)$ *und*
    $\text{eff}(f_1, f_2) = \neg \mathit{lift\text{-}at}(f_1) \wedge \mathit{lift\text{-}at}(f_2)$
- $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ *mit* $\mathcal{C}_0 = \{c_0^1, \ldots, c_0^{13}\}$ *und* $\mathcal{C}_\star = \{c_\star\}$*:*
  - $c_0^1 = \forall\, \omega\ (\mathit{floor}(\omega) \vee \mathit{passenger}(\omega))$
  - $c_0^2 = \forall\, \omega\ \neg(\mathit{floor}(\omega) \wedge \mathit{passenger}(\omega))$
  - $c_0^3 = \forall\, p, f\ (\mathit{origin}(p, f) \Rightarrow (\mathit{passenger}(p) \wedge \mathit{floor}(f)))$
  - $c_0^4 = \forall\, p, f\ (\mathit{destin}(p, f) \Rightarrow (\mathit{passenger}(p) \wedge \mathit{floor}(f)))$
  - $c_0^5 = \forall\, f_1, f_2\ (\mathit{above}(f_1, f_2) \Rightarrow (\mathit{floor}(f_1) \wedge \mathit{floor}(f_2)))$
  - $c_0^6 = \forall\, p\ (\mathit{boarded}(p) \Rightarrow \mathit{passenger}(p))$
  - $c_0^7 = \forall\, p\ (\mathit{served}(p) \Rightarrow \mathit{passenger}(p))$
  - $c_0^8 = \forall\, f\ (\mathit{lift\text{-}at}(f) \Rightarrow \mathit{floor}(f))$
  - $c_0^9 = \forall\, p\ (\mathit{passenger}(p) \Rightarrow (\exists^{=1}\, f\ \mathit{origin}(p, f)))$
  - $c_0^{10} = \forall\, p\ (\mathit{passenger}(p) \Rightarrow (\exists^{=1}\, f\ \mathit{destin}(p, f)))$
  - $c_0^{11} = \exists^{=1}\, f\ (\mathit{floor}(f) \wedge \mathit{lift\text{-}at}(f))$
  - $c_0^{12} = \forall\, f_1, f_2\ ((\mathit{floor}f_1 \wedge \mathit{floor}f_2) \Rightarrow (\mathit{above}(f_1, f_2) \vee \mathit{above}(f_2, f_1)))$
  - $c_0^{13} = \forall\, f_1, f_2\ ((\mathit{floor}f_1 \wedge \mathit{floor}f_2) \Rightarrow (\neg(\mathit{above}(f_1, f_2) \wedge \mathit{above}(f_2, f_1))))$
  - $c_0^{14} = \forall\, p\ (\mathit{passenger}(p) \Rightarrow\ \neg \mathit{served}(p))$
  - $c_\star = \forall\, p\ (\mathit{passenger}(p) \Rightarrow\ \mathit{served}(p))$

Es sei erwähnt, dass die Initialzustandsbeschränkung c_0^{14} sowie die Zielzustandsbeschränkung $c_\star$ einige Planungsaufgaben verletzen. Eine Eliminierung aller bereits abgelieferten sowie nicht im Zielzustand erwähnten Passagiere vor Beginn der eigentlichen Planung ist aber trivial, weswegen beide Beschränkungen zur Verdeutlichung der Ziele mit aufgenommen wurden.
Eine Überprüfung der Mitgliedschaft dieser Domäne in der Transportplanungsfamilie wird lediglich durch das Fehlen von Fahrzeugen etwas erschwert, was aber durch die Beschränkung dieses Objekttyps auf ein Vorkommen erklärt wird. Aus demselben Grund sind auch keine numerischen Variablenschemata enthalten: Auch die numerischen Variablen der Transportplanungsfamilie, welche die Kapazität des Fahrstuhls sowie den Treibstoffverbrauch beschreiben werden eliminiert. Dennoch liegt offensichtlich ein Transportproblem vor, und damit muss auch ein zweistelliges Prädikat, das Orte (bzw. Stockwerke) verbindet, gegeben sein, welches in dieser Domäne als $\texttt{above}(f_1, f_2)$ enthalten ist. Besonders interessant sind dabei die beiden Initialzustandsbeschränkungen c_0^{11} und c_0^{12}. Es ist nicht wie vermutet, dass ein Stockwerk lediglich mit den beiden umliegenden verbunden ist, sondern alle Etagen stehen in genau einer Richtung über das Prädikat $\texttt{above}(l_1, l_2)$ zueinander in Beziehung. Da zusätzlich mit $\texttt{up}(f_1, f_2)$ und $\texttt{down}(f_1, f_2)$ zwei Operatoren mit demselben Effekt existieren, deren Vorbedingung sich nur in der Richtung der $\texttt{above}(f_1, f_2)$-Beziehung unterscheidet, kann der Wegenetzgraph aller Miconic-10-Strips-Planungsaufgaben als komplett angesehen werden – für alle beliebigen, paarweise verschiedenen Etagen existiert ein Operator, mit welchem der Fahrstuhl von einem zu einem anderen Stockwerk bewegt werden kann. Wird die Domäne als Spezialfall der Transportplanungsfamilie definiert, werden beide Operatoren zu einem vereinigt werden, da die Übersetzung von darauf generierten Plänen trivial ist: Jeder derartige Bewegungsoperator wird einfach durch $\texttt{up}(f_1, f_2)$ ersetzt, wenn $\texttt{above}(f_1, f_2)$ gilt, und durch $\texttt{down}(l_1, l_2)$ sonst. Damit muss die Richtung der Verbindung nicht weiter beachtet werden, und Wegenetzgraph sowie Liefergraph derselben Beispielsaufgabe ergeben sich folgendermaßen:

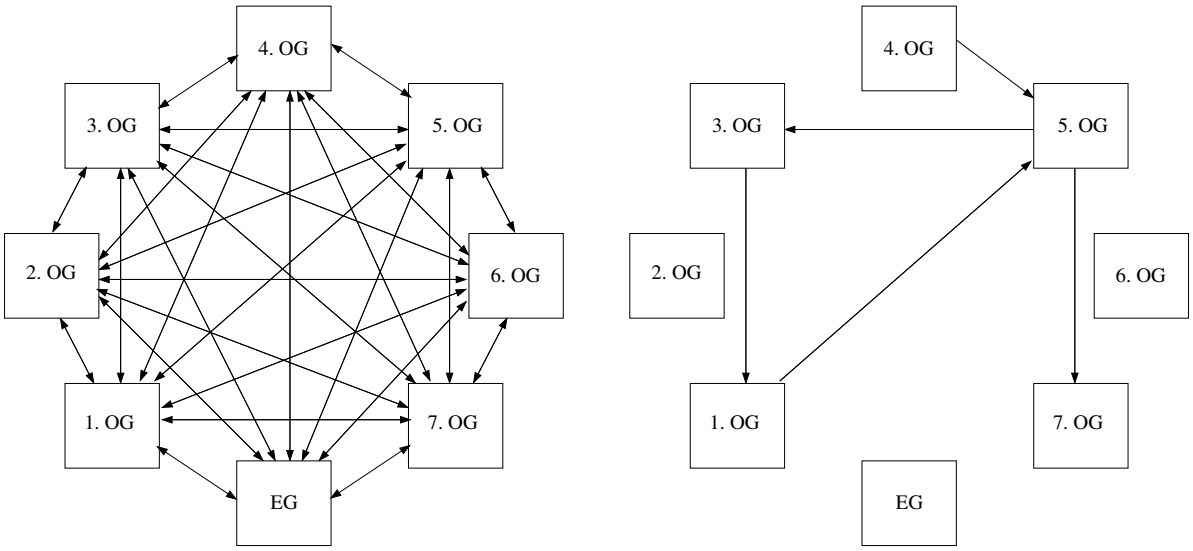

Abbildung 6.2: Wegenetzgraph und Liefergraph einer Miconic-10-Strips-Planungsaufgabe

Damit sind alle für Transportplanungsaufgaben im vorigen Kapitel definierten Konzepte auch für die Miconics-10-Strips-Domäne gegeben, und diese dementsprechend auch beschreibbar als Spezialfall dieser Familie.

Definition 6.2 MICONIC-10-STRIPS-*Domäne als Spezialfall der Transportplanungsfamilie*
Die **MICONIC-10-STRIPS**-*Domäne* $\mathcal{D}_{\text{M10}} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *ergibt sich als Spezialfall der Transportplanungsfamilie* $\mathcal{D}^{\text{TRANS}}$ *folgendermaßen:*

- $\mathcal{P} = \mathcal{P}^{\mathcal{C}}_{\text{TRANS}}$
- $\mathcal{V} = \mathcal{V}^{\mathcal{C}}_{\text{TRANS}}$
- $\mathcal{O} = \mathcal{O}^{\mathcal{C}_+}_{\text{TRANS}}$
- $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ *mit* $\mathcal{C}_0 = \mathcal{C}_0^{\text{TRANS}} \cup \{c_0^1, \ldots, c_0^5\}$ *und* $\mathcal{C}_\star = \mathcal{C}_\star^{\text{TRANS}}$*:*
  - $c_0^1 = \forall\, \omega\ (\texttt{portLoc}(\omega) \vee \texttt{mobile}(\omega) \vee \texttt{location}(\omega) \vee \texttt{portable}(\omega))$
  - $c_0^2 = \exists^{=1}\, m\ \texttt{mobile}(m)$
  - $c_0^3 = \forall\, l_1, l_2\ ((\texttt{location}(l_1) \wedge \texttt{location}(l_2)) \Rightarrow \texttt{connected}(l_1, l_2))$
  - $c_0^4 = \forall\, m\ (\texttt{mobile}(m) \Rightarrow \texttt{Fuel}(m) = \infty)$
  - $c_0^5 = \forall\, m\ (\texttt{mobile}(m) \Rightarrow \texttt{Capacity}(m) = \infty)$
- $\widehat{w}(\sigma_1) = \widehat{w}(\sigma_2) = 1$

Wie man sieht, ist die Darstellung dieser Domäne als Spezialfall einer bereits definierten Familie sehr viel effizienter – es sind lediglich die Typenbeschreibung und vier zusätzliche Beschränkungen nötig, um die gesamte Domäne zu definieren. Zudem zeigt sich, dass MICONICS-10-STRIPS nicht nur Mitglied dieser Familie ist, sondern zudem keine zusätzlichen Funktionalitäten implementiert – es sind keine zusätzlichen Schemata in der Domänenbeschreibung enthalten. Wird auf diese die in Kapitel 3 beschriebene Typeliminierung der beiden numerischen Variablen sowie des Prädikatenschemas `moblie`(m) angewendet, gefolgt vom Algorithmus zur Generierung effizienter Zustandsbeschreibungen entsteht der folgende relevante Zustandsraum. Die dafür notwendigen Schritte werden an dieser Stelle nicht wiederholt werden, es wird im Folgenden aber davon ausgegangen, dass die Domäne in der Form vorliegt, die sie nach Anwendung der Methoden hat, Prädikaten- und Operatorenschemata werden also ohne die Angabe der Parameter dargestellt, die eliminierte Variablen referenzieren. Zudem werden die in der Familenbeschreibung `pickUp` und `drop` benannten Operatoren wie in der IPC 2 Domänenbeschreibung mit `board` bzw. `depart` bezeichnet.

Definition 6.3 ***Relevanter Zustandsraum der* MICONIC-10-STRIPS-*Domäne***
Der relevante Zustandsraum $S(\mathcal{T}_{\text{M10}})$ *beliebiger Planungsaufgaben* $\mathcal{T}_{\text{M10}}$ *auf der* MICONIC-10-STRIPS-*Domäne* $\mathcal{D}_{\text{M10}}$ *ist gegeben als:*

GRUNDELEMENTE: $\langle W, P, D \rangle$, *wobei* $W = \langle F, F^2 \rangle$ *der vollständige* ***Wegenetzgraph*** *mit der Menge der* ***Stockwerke*** F, P *die Menge der* ***Passagiere****, und* $D = \langle \text{loc}_0, \text{loc}_\star \rangle$ *der* ***Liefergraph*** *ist mit der* ***Startstockwerksfunktion*** $\text{loc}_0 : P \to F$ *und der* ***Zielstockwerksfunktion*** $\text{loc}_\star : P \to F$.

ZUSTÄNDE: *2-Tupel* $\langle f^e, \text{at} \rangle$, *wobei* $f^e \in F$ *das Stockwerk ist, an dem der Fahrstuhl sich befindet, und* $\texttt{at} : P \to F \cup \{E\}$ *die* ***Ortsfunktion*** *ist, die angibt in welchem Stockwerk sich ein Passagier befindet, bzw.* E*, falls der Passagier im Fahrstuhl ist.*

STARTZUSTAND: $\langle f_0^e, \text{at} \rangle$ *mit* $\text{at}(p) = \text{loc}_0(p)$ *für alle* $p \in P$

ZIELZUSTAND: *Jeder Zustand* $\langle f^e, \text{at} \rangle$ *mit* $\text{at}(p) = \text{loc}_\star(p)$ *für alle* $p \in P$

OPERATOREN: *Der Fahrstuhl kann sich zwischen Stockwerken beliebig* ***bewegen****, eine Person kann* ***einsteigen****, falls sie sich im selben Stockwerk wie der Fahrstuhl befindet, und in Stockwerk* f ***aussteigen****, falls sie im Fahrstuhl ist und der Fahrstuhl in* f.

6.2 Planer

In diesem Abschnitt werden zwei Planer für die Miconic-10-Strips-Domäne vorgestellt: Begonnen wird mit einem Algorithmus, der Lösungen approximiert, also Plan-$\mathcal{D}_{\text{M10}}$ löst, bevor ein Planer angegeben wird, welcher Lösungen für OptPlan-$\mathcal{D}_{\text{M10}}$ liefert. In beiden Fällen wird die Domäne als Spezialfall der Transportplanungsfamilie betrachtet werden, also lediglich ein Operator `move`(f_1, f_2) verwendet werden.

Zusätzlich zu den im vorigen Abschnitt bereits bezeichneten Mengen und Funktionen seien die folgenden beiden Teilmengen der Menge der Stockwerke für die Belange dieses Abschnittes definiert. Außerdem werden die Kosten von Plänen in zwei Teile aufgeteilt:

- $F^b \subseteq F$ sei die Menge der Stockwerke, die Startstockwerk mindestens eines Passagiers ist.
- $F^d \subseteq F$ sei die Menge der Stockwerke, die Zielstockwerk mindestens eines Passagiers ist.
- $\#^{b,d}(\pi)$ sei die Anzahl der `board`- und `depart`-Aktionen im Plan π.
- $\#^{m}(\pi)$ sei die Anzahl der `move`-Aktionen im Plan π.

Bevor die beiden Planer vorgestellt werden sei erst der Teil von Plänen dieser Domäne besprochen, dessen optimale Lösung trivial zu generieren ist: Die Anzahl der `board`- und `depart`-Aktionen $\#^{b,d}(\pi)$ eines Planes π. In der Beschreibung von Transportplanungsproblemen wird als Zieldefinition angegeben, dass alle transportierbaren Objekte, in diesem Fall also Passagiere, Teil der Zielzustandsbeschreibung sein müssen und nicht bereits im Initialzustand ihr Ziel erreicht haben dürfen. Zusätzlich sei angenommen, dass kein Passagier im Zielzustand im Fahrstuhl verbleiben soll – diese Beschränkung macht die folgenden Beschreibungen prägnanter, ohne dass die Komplexität der Planungsaufgabe verändert wird. Unter diesen Voraussetzungen und der zusätzlichen Eigenschaft der Domäne, dass der Fahrstuhl unbegrenzte Kapazität aufweist, wird jeder Passagier in optimalen Plänen genau einmal in den Aufzug ein- und genau einmal wieder aussteigen, und zudem wird jeder an einem Stockwerk wartende Passagier, der sein Ziel noch nicht erreicht hat, einsteigen, sobald der Fahrstuhl an dieser Etage Halt macht und aussteigen, sobald das Zielstockwerk erreicht ist. Damit ist bereits eine triviale, optimale Strategie für die `board` und `depart`-Operatoren gefunden, die aus diesem Grund im Folgenden nicht weiter beachtet werden. Für die Anzahl dieser Aktionen in einem mit dieser Strategie generierten Plan π gilt also:

$$\#^{b,d}(\pi) = 2|P|$$

Unter Anwendung dieser Strategie für `board` und `depart`-Aktionen sei nun ein trivialer Algorithmus für die Bewegungsoperatoren formuliert. Dabei sollen zuerst alle wartenden Passagiere in beliebiger Reihenfolge abgeholt und dann, ebenfalls in beliebiger Reihenfolge, bei ihrem Zielstockwerk abgeliefert werden. Dass dieser Methode einen Plan generiert, ist dabei leicht ersichtlich.

Algorithmus: ***Algorithmus für* Plan-$\mathcal{D}_{\text{M10}}$**

Input: *Eine lösbare, instantiierte Planungsaufgabe* $\mathcal{T}_{\text{M10}}$

Output: *Ein Plan* π *für* $\mathcal{T}_{\text{M10}}$

Algorithmus:
```
create F^b and F^d
remove f_0^e from F^b
for each p with loc_0(p) = f_0^e do
    π.add(board(p, f_0^e))
set f = f_0^e
```

```
while F^b ≠ ∅ do
   f' = F^b.first
   remove f' from F^b
   π.add(move(f, f'))
   set f = f'
   for each p with loc_0(p) = f do
     π.add(board(p, f))
if f ∈ F^d do
   remove f from F^d
   for each p with loc_⋆(p) = f do
     π.add(depart(p, f))
while F^d ≠ ∅ do
   f' = F^d.first
   remove f' from F^b
   π.add(move(f, f'))
   f = f'
   for each p with loc_⋆(p) = f do
     π.add(depart(p, f))
return π
```

Die Anzahl der benötigten `move`-Aktionen $\#^m(\pi)$ in mit diesem Algorithmus generierten Plänen kann einfach berechnet werden werden. Dabei sei angenommen, dass der Aufzug im Initialzustand nicht an einem Stockwerk startet, an welchem auch ein Passagier wartet, was auf die im Folgenden beschrieben Analyse der Approximationseigenschaften der Methode keine Auswirkungen hat.

$$\#^m(\pi) = |F^b| + |F^d|$$

Nun sei überlegt, ob die Anzahl der Bewegungsoperatoren in einem optimalen Plan abgeschätzt werden kann. Eine offensichtlich gültige Abschätzung kann dadurch generiert werden, dass jedes Stockwerk, das entweder Start- oder Zielstockwerk eines Passagiers ist, auch in optimalen Plänen mindestens einmal angefahren werden muss.

$$\#^m(\pi^\star) \geq |F^b \cap F^d|$$

Sind die beiden Mengen F^b und F^d disjunkt, ist der beschriebene Planer offensichtlich bereits optimal, da dann gilt:

$$|F^b \cap F^d| = |F^b| + |F^d|$$

Damit sollte noch der Worst-Case im Vergleich zu optimaler Planung betrachtet werden. Dieser tritt dann ein, wenn eine der beiden Mengen eine Teilmenge der anderen ist. Es sei nun angenommen, dass $F^d \subseteq F^b$ gilt, wodurch die Allgemeinheit dieser Analyse nicht beeinträchtigt wird – sie ist auf dem selben Weg auch für $F^b \subseteq F^d$ durchführbar. In jedem Fall gilt für die Länge eines durch den beschriebenen Approximationsalgorithmus generierten Planes im Vergleich zu einem optimalen Plan für beliebige MICONIC-10-STRIPS-Planungsaufgaben:

$$\frac{m(\pi)}{m(\pi^\star)} \leq \frac{\#^{b,d}(\pi)+\#^m(\pi)}{\#^{b,d}(\pi^\star)+\#^m(\pi^\star)} = \frac{2|P|+|F^b|+|F^d|}{2|P|+|F^b \cap F^d|}$$

Nun sei noch noch eine letzte Überlegung getätigt: zum einen ist, sowohl in durch den Approximationsalgorithmus generierten als auch in optimalen Plänen, der letzte angewendete Operator immer eine `depart`-Aktion, und zwischen zwei `move`-Aktionen muss eine der beiden anderen Aktionen liegen, da die Bewegung

durch die Vollständigkeit des Wegenetzgraphen sonst unnötig wäre. Damit können in einem Plan, der durch diesen Algorithmus generiert wird, höchstens genauso viele Bewegungsaktionen wie Ein- und Ausstiegsaktionen enthalten sein, also

$$\#^m(\pi) \leq \#^{b,d}(\pi)$$

Damit folgt für das Verhältnis von durch diesen Algorithmus generierten Plänen zum möglichen Optimalfall im Worst-Case:

$$\frac{m(\pi)}{m(\pi^\star)} \leq \frac{2|P|+|F^b|+|F^d|}{2|P|+|F^b \cap F^d|} \leq \frac{2|F^b|+2|F^b|}{2|F^b|+|F^b|} = \frac{4}{3}$$

Es werden mit diesem Algorithmus also Pläne generiert, deren Länge höchsten dem $\frac{4}{3}$-fachen der Minimalkosten entsprechen, und der Algorithmus ist somit $\frac{4}{3}$-approximierend und Miconic-10-Strips in APX. Helmert zeigt sogar, dass die Domäne $\frac{7}{6}$-approximierbar ist und zudem nicht in PTAS liegt, wenn P ≠ NP gilt [Hel08].

Satz 6.1 ***Komplexität suboptimalen Planens auf der* Miconic-10-Strips-*Domäne***
Miconic-10-Strips *ist $\frac{7}{6}$-approximierbar, liegt aber nicht in PTAS, sofern P ≠ NP. Es gilt also* Plan-$\mathcal{D}_{\text{M10}} \in$ *APX \ PTAS.*

Suboptimales Planen auf der Miconic-10-Strips-Domäne ist also, ohne dass sehr viel größere Pläne als durch optimales Planen generiert werden, in polynomieller Zeit möglich. Nun sei ein Algorithmus vorgestellt, der optimale Pläne generiert. Die bislang beschriebenen Erkenntnisse über `board`- und `depart`-Aktionen gelten weiterhin und müssen nicht erneut untersucht werden. Im Folgenden wird also lediglich eine optimale Folge von `move`-Operatoren gesucht, die durch einen *Pfad* repräsentiert werden können. Die Auswirkungen, die sich durch den Startknoten des Fahrstuhls ergeben, werden dabei vorerst vernachlässigt, es wird also davon ausgegangen, dass sich dieses an einem beliebigen Knoten ohne ein- und ausgehende Kanten im Liefergraphen befindet.

Definition 6.4 ***Pfad***
Sei $D = \langle \text{loc}_0, \text{loc}_\star \rangle$ ein Abhängigkeitsgraph mit $\text{loc}_0 : P \to L$ und $\text{loc}_\star : P \to L$ auf den Mengen L und P. Ein ***Pfad*** *π im Abhängigkeitsgraphen D ist eine Folge von $l_1, \ldots, l_n$, $l_i \in L$ und sei beschrieben durch:*

$$\pi = l_1 \to l_2 \cdots \to l_n$$

Es gelte $l \in \pi$, wenn l im Pfad π enthalten ist, also wenn $\pi = l_1 \to \cdots \to l \to \cdots \to l_n$.
Zudem gelte $l_i <_\pi l_j$, wenn l_i vor l_j in π enthalten ist, also wenn $\pi l_1 \to \cdots \to l_i \to \cdots \to l_j \to \cdots \to l_n$.

Ein Pfad ist, bezogen auf die Miconics-10-Strips-Domäne, also eine beliebige Abfolge von Stockwerken. Beim Symbol $l_1 <_\pi l_2$, das angibt, ob ein Stockwerk l_1 vor einem anderen l_2 im Plan enthalten ist, ist Vorsicht geboten: Es ist durchaus möglich, dass sowohl $l_1 <_\pi l_2$ als auch $l_2 <_\pi l_1$ gelten, wenn eines der Stockwerke mehrmals im Plan enthalten ist. Nun ist ein Pfad in Bezug auf Planen aber nicht besonders aussagekräftig – jede beliebige Abfolge von Knoten stellt einen solchen dar. Gesucht werden aber Pfade, durch die auch ein Plan generiert werden kann. Ein solcher heiße *planerfüllend*:

Definition 6.5 ***Planerfüllender Pfad***
Sei $D = \langle \text{loc}_0, \text{loc}_\star \rangle$ ein Abhängigkeitsgraph mit $\text{loc}_0 : P \to L$ und $\text{loc}_\star : P \to L$ auf den Mengen L und P, $G = \langle V, E \rangle$ der durch D implizierte Graph sowie ein π ein Pfad.
π ist ***planerfüllend****, wenn gilt:*

Für alle $e \in E$, $e = \langle l_1, l_2 \rangle$ gilt: $l_1 <_\pi l_2$.

Diese Definition ergibt sich direkt aus dem, was ein Liefergraph impliziert: Jede Person ist darin durch eine Kante zwischen ihrem Startstockwerk und ihrem Zielstockwerk repräsentiert. Wird in einem Pfad an einer Stelle das Startstockwerk besucht, kann die Person eingeladen werden, und da diese Forderung beschreibt, dass später auch das Zielstockwerk besucht wird, kann die Person auch abgeliefert werden. Ist dies für alle Personen gegeben, kann der Pfad einfach in einen MICONICS-10-STRIPS-Plan transformiert werden: Für jedes l_i im Pfad π wird ein Operator `move`(l_i, l_{i+1}) in den Plan eingefügt, sowie zusätzlich die am jedem Stockwerk nötigen `board`- und `depart`-Aktionen.

Die Grundidee des im Folgenden beschriebenen Algorithmus liegt darin, einen Knoten nach dem anderen zu ermitteln, der sicher als nächstes besucht werden kann, ohne dass dadurch ein nicht-optimaler Plan entsteht. Das wichtigste Werkzeug dafür ist der Liefergraph einer Planungsaufgabe, für welchen ein Weg gefunden werden muss, wie in diesem vermerkt wird, dass ein Knoten bereits besucht wurde. Dazu sei zuerst die Forderung, die ein Pfad erfüllen muss, um planerfüllend zu sein, in drei Teile geteilt:

- Für alle $e = \langle l_1, l_2 \rangle$ gilt $l_1 \in \pi$.
- Für alle $e = \langle l_1, l_2 \rangle$, $e \in E$ gilt: $l_1 <_\pi l_2$.
- Für alle $e = \langle l_1, l_2 \rangle$ gilt $l_1 \in \pi$.

Es werden also zwei zusätzliche Forderungen generiert, dass jeder Knoten, der Teil einer Kante ist, auch besucht werden muss, was durch die ursprüngliche, weiterhin enthaltene Forderung bereits impliziert wird. Dennoch können Fälle auftreten, in welchen diese Unterscheidung eine Rolle spielt. Nun soll untersucht werden, welche Änderungen in einem Liefergraphen durch den Besuch eine Knotens entstehen. Dafür sei erst eine Planungsaufgabe angegeben, die der Approximationsalgorithmus bereits optimal lösen kann, in welcher also kein Stockwerk gleichzeitig Start und Ziel von Passagieren ist.

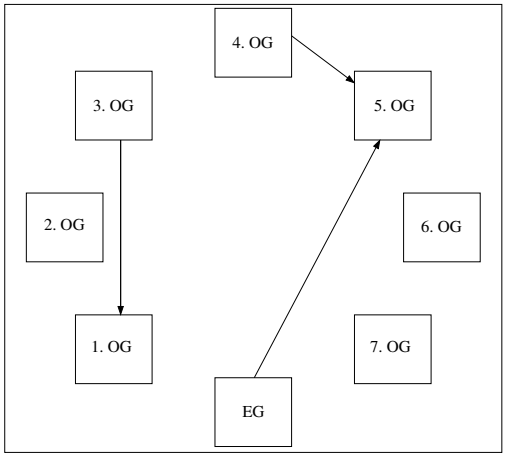

Abbildung 6.3: Liefergraph ohne Stockwerk, das gleichzeitig Start und Ziel eines Passagiers ist

Der Approximationsalgorithmus wird in diesem Beispiel zuerst die Knoten EG, 3.OG sowie 4.OG in beliebiger Reihenfolge besuchen, und dann das fünfte sowie das erste OG, abermals in beliebiger Reihenfolge. Es sei angenommen, dass der folgende Plan generiert werde:

$$\text{EG} \rightarrow \text{3.OG} \rightarrow \text{4.OG} \rightarrow \text{5.OG} \rightarrow \text{1.OG}$$

Nun muss zuerst festgestellt werden, dass bisher abgebildete Liefergraphen meist gar nicht deren Definition entsprechen: Nach Definition 3.9, in welcher mit Abhängigkeitsgraphen die Grundlage von Liefergraphen

gegeben werden, besteht ein Liefergraph immer nur aus den Knoten, die auch eine ein- oder ausgehende Kante haben. Im Folgenden seien Liefergraphen in Darstellungen dann als minimaler Liefergraph bezeichnet, auch wenn diese eigentlich den normalen Liefergraphen darstellen – bisherige Abweichungen in Abbildungen sind rein ästhetischer Natur. Damit gilt für jeden im Liefergraphen vorhandenen Knoten, dass er irgendwann im Pfad enthalten sein muss. Jede Kante beschreibt dagegen die Abhängigkeit von Knoten untereinander: Der Ursprungsknoten muss vor dem Zielknoten besucht werden. Da jede Anwendung eines Operators einen Teil dieser Bedingungen erfüllen kann, kann auch der Liefergraph immer wieder angepasst werden.

Diese Bilder zeigen, wie sich der Liefergraph nach Anwendung jedes Operators aus dem Beispiel ändert.

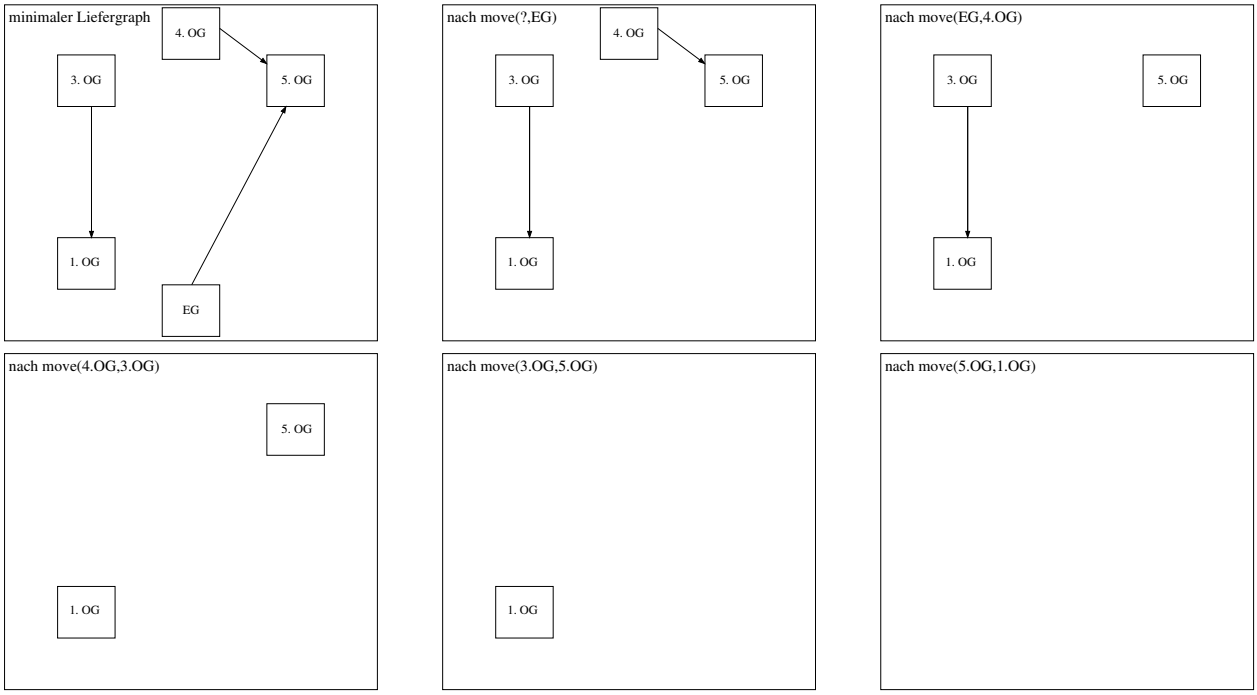

Abbildung 6.4: Auswirkung der Ausführung eines Planes auf den Liefergraphen

Es ist auch zu sehen, weswegen die Minimalität wichtig ist: Wäre diese nicht gegeben, und zum Beispiel das zweite OG beinhaltet, wäre dieses nicht von den beiden Knoten im Bild links unten zu unterscheiden, weswegen dann eine zusätzliche Markierung eingeführt werden müsste. Nun soll eine Regel gefunden werden, wie ein solcher Liefergraph nach Anwendung eines Operators angepasst werden kann. Für dieses Beispiel würde die Regel genügen, jeden besuchten Knoten und alle Kanten nach Besuch des Fahrstuhls zu löschen. Es muss allerdings beachtet werden, dass bislang nur sehr eingeschränkte Formen von Liefergraphen betrachtet wurden, da jeder Knoten immer nur entweder ein- oder ausgehende Kanten besitzen durfte. Daher sei ein weiteres Beispiel gegeben, welches zwar Knoten mit ein- und ausgehenden Kanten enthält, aber noch immer durch die Forderung beschränkt sei, dass keine Zykel enthalten sind.

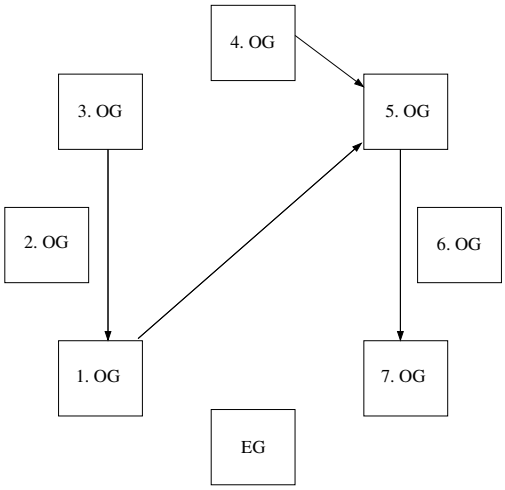

Abbildung 6.5: Zykelfreier Liefergraph einer MICONIC-10-STRIPS-Planungsaufgabe

Angenommen, dieses Beispiel wird mit dem Approximationsalgorithmus gelöst und der folgende Pfad generiert:

$$4.\text{OG} \rightarrow 1.\text{OG} \rightarrow 3.\text{OG} \rightarrow 1.\text{OG} \rightarrow 5.\text{OG} \rightarrow 7.\text{OG}$$

Dann beschreiben die folgenden Abbildungen die Veränderung des Liefergraphen bei Anwendung der ersten beiden Operatoren des Planes:

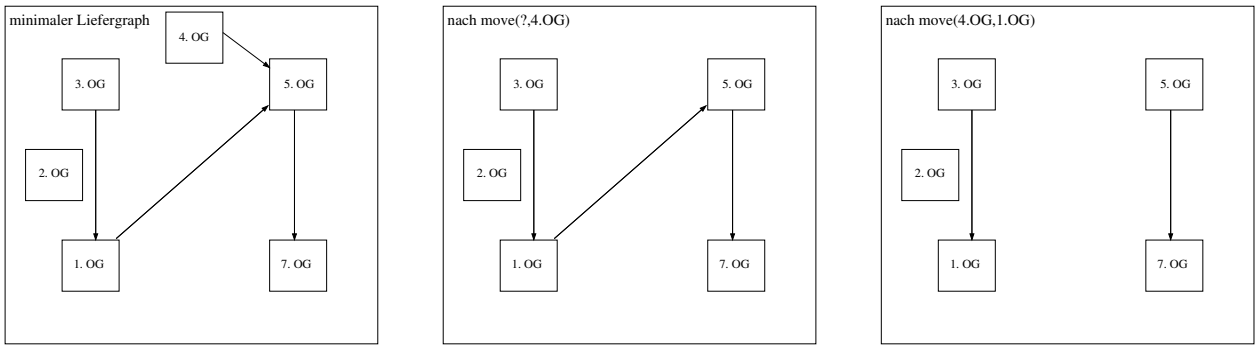

Abbildung 6.6: Auswirkung der Ausführung eines Planes auf den Liefergraphen

Wie man sieht, wird das 1. OG nach Anwendung des Operators `move(4.OG,1.OG)` nicht gelöscht. Der Grund dafür liegt darin, dass die Forderung, dass das 1. OG nach dem 3.OG besucht wird, noch nicht erfüllt ist. Daher muss dieses Stockwerk zu einem späteren Zeitpunkt in einem planerfüllenden Graphen noch einmal besucht werden – wie es im angegebenen Plan auch der Fall ist. Insgesamt können daraus die folgenden Regeln abgeleitet werden, was das *Markieren* eines Knotens, dass dieser bereits besucht wurde, bewirkt:

- Lösche alle Kanten aus E, deren Ursprungsknoten l ist.
- Wenn keine Kanten auf l zeigt, lösche l aus V.

Dieses Markieren wird im Algorithmus also immer dann benötigt, wenn ein weiterer, sicher optimaler Bewegungsoperator an den Pfad angehängt wird. Dann drückt der durch markieren des entsprechenden Knotens entstandene Liefergraph genau das Problem aus, das es noch zu lösen gilt. Dies sei durch die Methode

`markVisited(`D`,`l`)` für den Knoten l im Liefergraphen D implementiert.

Daraus kann bereits ein Schritt eines optimalen Algorithmus hergeleitet werden: Es wurde bereits beschrieben, dass jeder Knoten eines Liefergraphs mindestens einmal besucht werden muss. Solange also Knoten gefunden werden können, die durch Anhängen an den Pfad und das dann nötige Markieren selbst gelöscht werden, wird durch den entsprechenden Operator die Optimalität des entstehenden Pfades sicher nicht verletzt – es wurde ein Operator ausgeführt, aber es ist auch ein Knoten weniger im Liefergraphen. Ein Knoten, der durch Markieren aus dem Liefergraphen gelöscht wird, heiße *Exportknoten*.

- Ein Knoten l heiße **Exportknoten** des Liefergraphen $D = \langle \mathrm{loc}_0, \mathrm{loc}_\star \rangle$, wenn für alle p gilt $\mathrm{loc}_\star(p) \neq l$. M^e sei die Menge aller Exportknoten.

Da durch das Markieren nicht nur möglicherweise Knoten, sondern meistens auch Kanten gelöscht werden, kann es sein, dass neue, vor dem Markieren nicht zu den Exportknoten gehörende Knoten entstehen. Eine Schleife im Algorithmus, welche, solange es möglich ist, Exportknoten ermittelt, an den Plan anhängt und markiert, wird also sicher keine Teilpfade generieren, die die Optimalität verletzen. Daher sei eine Methode gegeben, die Exportknoten eines Liefergraphen D liefert:

`getExNodes(`D`,`M^e`)`

Wendet man die beschrieben Schleife auf die beiden Beispiele an, kann sogar der ganze, optimale Plan generiert werden: Jeder Markiervorgang macht weitere Knoten zu Exportknoten, bis die Liefergraphen leer sind. Dennoch stellt dies noch keinen vollständigen Algorithmus zur optimalen Pfadfindung dar: Ist in einem Liefergraphen ein Zykel enthalten, wird in der wiederkehrenden Anwendung dieser Methode irgendwann kein Exportknoten mehr gefunden, ohne dass der Graph leer ist. Dennoch ist so bereits ein Algorithmus zur optimalen Pfadfindung in zykelfreien Graphen gegeben.

Algorithmus: ***Algorithmus für optimale Pfadplanung in zykelfreien Liefergraphen***

INPUT: *Ein zykelfreier Liefergraph* $D = \langle \mathrm{loc}_0, \mathrm{loc}_\star \rangle$ *und ein Startknoten* v.

OUTPUT: *Ein minimaler Pfad* $\pi = l_1, \ldots, l_n$

ALGORITHMUS:
```
markVisited(D,v)
while D is not empty do
   getExNodes(D,M^e)
   for each m ∈ M^e do
     π.add(m)
     markVisited(D,m)
return π
```

Bislang ungeklärt ist lediglich die erste Zeile dieses Pfadplaners, in welcher der Startknoten als besucht markiert wird. Dass dies so korrekt ist, ist aber schnell ersichtlich, da der Besuch eines Knoten durch den beschriebenen Markiervorgang so definiert wurde, dass unerheblich ist, was vorher im Pfad geschehen ist. Damit ist der einzige Fall, in welchem der Knoten durch Markieren auch gelöscht wird der, dass er selbst ein Exportknoten ist. Daraus folgt eine andere, interessante Eigenschaft: Liegt der Startknoten in der Mitte eines Liefergraphen, hat also sowohl eingehende als auch ausgehende (oder auch nur eingehende) Kanten, wird – zumindest in zykelfreien Liefergraphen – ein Plan derselben Länge generiert, wie wenn der Startknoten außerhalb des Graphen liegt. Es werden also nicht zwingend kürzere, optimale Pläne erzeugt, wenn am Startstockwerk des Fahrstuhls bereits ein Passagier wartet.

Damit bleibt lediglich, den Algorithmus so zu erweitern, dass auch Graphen mit Zykeln geplant werden können. Dafür sei das Konzept der starken Zusammenhangskomponenten (SCC) kurz erläutert, eine detailliertere Beschreibung findet sich zum Beispiel in [CLRS01]. Ein SCC ist die maximal große Teilmenge der Knotenmenge des Liefergraphen, in welchen jeder Knoten von allen anderen Knoten der Teilmenge aus erreichbar ist. Wird ein Graph in all seine SCCs zerlegt, und zwischen den entstandenen SCCs Kanten eingeführt, die vor der Zerteilung zwischen Knoten eines SCCs und Knoten eines anderen waren, entsteht der SCC-Liefergraph.

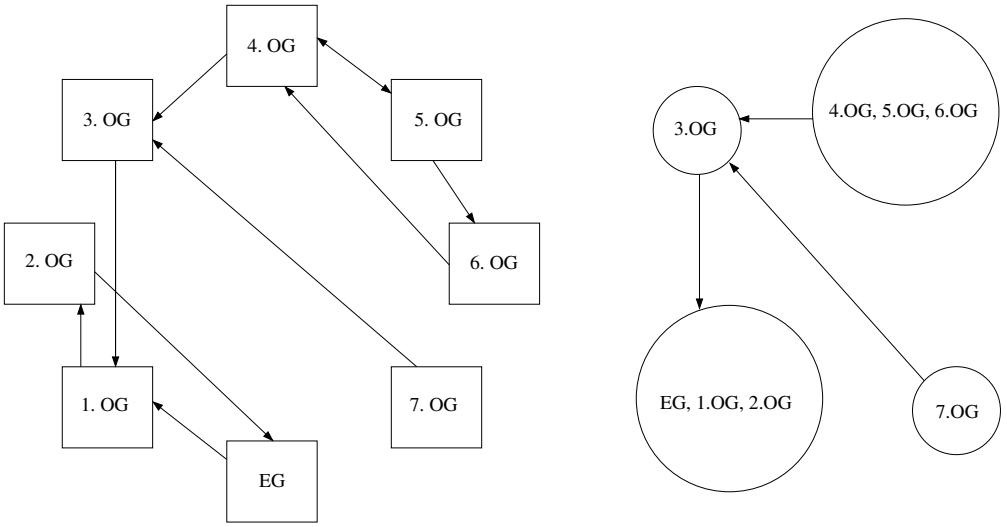

Abbildung 6.7: Liefergraph (links) und zugehöriger SCC-Liefergraph (rechts)

Eine wichtige Eigenschaft eines derartigen SCC-Liefergraphen ist es, dass er azyklisch sein muss – denn wäre zwischen verschiedenen SCCs ein Zykel, könnten die in einem SCC enthaltenen Knoten von allen Knoten der anderen SCCs im Zykel erreicht werden, und damit müssten alle beteiligten Knoten Teil desselben SCCs sein. Wichtig für die Belange dieser Arbeit ist zudem, dass ein SCC-Liefergraph ein Liefergraph ist, der selbst Liefergraphen als Knoten enthält – im Minimalfall besteht dieser zwar nur aus einem Knoten, aber auch dieser kann als Liefergraph angesehen werden. Die folgende Methode generiere den SCC-Liefergraphen:

`createSCCDelGraph(`D`,`D^{SCC}`)`

Da dieser Liefergraph azyklisch ist, kann der bereits beschriebene Algorithmus für zykelfreie Liefergraphen auch auf den SCC-Liefergraphen angewendet werden. Soll dabei ein einzelner Knoten zum Plan hinzugefügt werden, ist keine Anpassung nötig. Es bleibt also nur zu klären, was genau getan werden muss, wenn ein Zykel zum Plan hinzugefügt werden soll. In Zykeln besteht dabei ein Besonderheit: Durch den zyklischen Aufbau muss mindestens ein Knoten doppelt angefahren werden. Das Problem, das hier vorliegt, ist es also, möglichst wenige Knoten zu bestimmen, ohne die der Graph wieder azyklisch wird – denn dann muss jeder übrige Knoten nur noch einmal angefahren werden. Dieses Problem ist in der Informatik als Problem der Generierung eines *Minimum Feedback Vertex Sets* (MFVS) bekannt:

Definition 6.6 ***Generierung eines Minimum Feedback Vertex Sets***
Das Problem der ***Generierung eines Minimum Feedback Vertex Sets*** GENMFVS *ist gegeben durch:*

EINGABE: *Ein Graph* $G = \langle V, E \rangle$.

LÖSUNG: *Ein minimal große Teilmenge* $V^{mfvs} \subseteq V$*, so dass* $G' = \langle V', E' \rangle$ *azyklisch, mit* $V' = V \setminus V^{mfvs}$ *und* E' *den Kanten aus* E*, die von einem Knoten* $v \in V'$ *zu einem Knoten* $v' \in V'$ *führen.*

Das MFVS-Problem, also das zu Definition 6.6 zugehörige Entscheidungsproblem, ob für einen gegebenen Graphen G und eine Zahl $n \in \mathbb{N}$ ein MFVS der Größe n existiert, ist ein bewiesenermaßen NP-hartes Problem und war bereits Objekt zahlreicher Studien (z.B. [GJ79]). Dementsprechend existieren viele unterschiedliche Algorithmen zu dessen Lösung. Eine mögliche Implementierung ist die Brute-Force-Methode, die für immer größerer Teilmengen der Knotenmenge überprüft, ob der durch Löschen dieser Knoten entstandene Graph azyklisch ist, wodurch im Worst-Case alle Elemente der Potenzmenge untersucht werden müssen. Intelligentere Algorithmen wie zum Beispiel von Smith und Walford identifizieren dagegen Knoten, die besonders wahrscheinlich Teil des MFVS sind und sind so im Average-Case deutlich schneller [SW75]. Dennoch ist es, sofern P ≠ NP, nicht möglich, einen Algorithmus anzugeben, der ein MFVS im Worst-Case schneller als in exponentieller Zeit generiert.

Damit ist beschrieben, wie ein MFVS generiert werden kann, und es bleibt zu klären, inwieweit die bisher beschrieben Methode zur Anpassung des Liefergraphen verändert werden muss. Dieser Vorgang kann etwas erleichtert werden: Anstatt MFVS-Knoten mit der beschriebenen Methode zu markieren, und dadurch den zweiten Besuch erneut planen zu müssen, können diese inklusive aller ein- und ausgehenden Kanten komplett gelöscht (also nicht nur markiert) werden, und nach Abarbeitung aller anderen SCC-Knoten ein weiteres Mal an den Pfad angehängt werden. Dafür seien die folgenden Methoden gegeben:

$MFVS$ = `getMFVS(SCC)`

$MFVS$ = `getMFVS(SCC,`V`)`

`deleteMFVSNode(`$MFVS$`,`l`)`

Die erste und die dritte Methode sind selbsterklärend, die Erweiterung der zweiten um einen zusätzlichen Parameter V dagegen nicht. Diese Methode stellt bereits einen Vorgriff auf die im folgenden Kapitel beschrieben Logistics-Domäne dar. Da das MFVS-Problem aber bislang in diesem Teil dieser Arbeit erläutert wird, soll auch diese Erweiterung hier nicht fehlen: V ist ein Menge von Knoten, die bevorzugt Teil des MFVS sein sollen. Dies ist möglich, da die Größe des MFVS zwar eindeutig ist, nicht aber die Knoten selbst. Da die Logistics-Domäne mehrere Fahrzeuge beinhaltet, kann es sein, dass es von Vorteil ist, wenn ein MFVS generiert wird, das einen bestimmten Knoten, nämlich den Startknoten eines der Fahrzeuge, enthält:

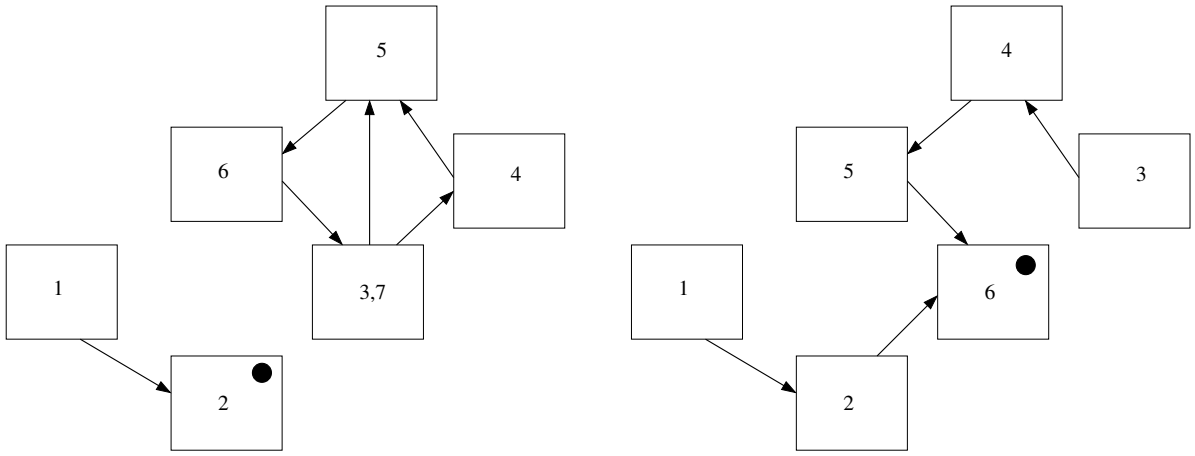

Abbildung 6.8: Kürzere Pfade durch Startknoten im MFVS

Die Beschriftungen der Knoten in diesem Miconics-10-Strips-Beispiel bezeichnen in diesen Darstellungen nicht deren Namen, sondern die Position im optimalen Pfad. Man sieht, dass im rechten Liefergraphen nur

sechs Schritte benötigt werden, während im linken sieben erforderlich sind. Dabei stellen beide, abgesehen von der unterschiedlichen Startposition, dieselbe Planungsaufgabe nach Markieren des Startknotens dar. Daraus ist auch direkt ersichtlich, dass Pfade kürzer werden, wenn der Startknoten Teil des MFVS ist. Die beschriebene Methode wird für die MICONICS-10-STRIPS-Domäne aber keine Relevanz haben, da nicht zwischen verschiedenen Startknoten gewählt werden muss. Wird aber in dem im nächsten Kapitel vorgestellten LOGISTICS-Planer eine MFVS-Berechnung für einen Liefergraphen benötigt, der die Form der rechten Abbildung hat, ist es wichtig, dass nicht das MFVS geliefert wird, das den mit 4 beschrifteten Knoten liefert, sondern den mit 6 beschrifteten.

Dies führt gewissermaßen direkt dazu, welche Methode letztlich zur MFVS-Generierung implementiert wurde. Da in Algorithmen wie dem von Smith und Walford die zusätzliche Forderung, dass ein bestimmter Knoten bevorzugt Teil des Resultates sein soll, nicht implementiert ist, und eine Anpassung dadurch erschwert wird, dass diese Algorithmen bereits selbst Knoten nach bestimmten Kriterien bevorzugt untersuchen, wurde die beschriebene, triviale Methode mithilfe einer Methode zur effizienten Nummerierung von Teilmengen einer Menge [LHS00] implementiert. Eine mögliche Alternative wäre es gewesen, einen effizienten Algorithmus zu wählen, und dann, sofern das Ergebnis keinen bevorzugten Knoten enthält, alle möglichen MFVS gleicher Größe, die einen solchen beinhalten, daraufhin zu überprüfen, ob sie azyklisch sind. Dann wären aber wieder Teile der gewonnenen Leistungssteigerung verloren gegangen, weswegen der beschriebene Weg gewählt wurde. Zudem zeigen die im nächsten Abschnitt vorgestellten Ergebnisse auch, dass die gewählte Implementierung für MICONICS-10-STRIPS-Planungsaufgaben mehr als hinreichend ist.

Damit ist die Beschreibung des Algorithmus zur optimalen Pfadfindung in beliebigen Graphen abgeschlossen. Mit den gewonnenen Erkenntnissen ist es möglich, beliebige MICONICS-10-STRIPS-Planungsaufgaben zu lösen:

Algorithmus: ***Algorithmus für optimale Pfadplanung in Liefergraphen***

INPUT: *Ein Liefergraph* $D = \langle \mathrm{loc}_0, \mathrm{loc}_\star \rangle$ *und ein Startstockwerk* v.

OUTPUT: *Ein minimaler Pfad* $\pi = l_1, \ldots, l_n$.

ALGORITHMUS:

```
markVisited(D,v)
createSCCDelGraph(D,D^SCC)
while D^SCC is not empty do
   getExNodes(D,SCC^e)
   for each scc ∈ SCC^e do
     if scc.size == 1
       π.add(scc)
     else
       MFVS = getMFVS(scc)
       for each l ∈ MFVS do
         π.add(l)
         deleteMFVSNode(scc,l)
       while scc is not empty do
         getExNodes(scc,scc^e)
         for each l ∈ scc^e do
           π.add(l)
```

```
            markVisited(scc,l)
        for each l ∈ MFVS do
          π.add(l)
      markVisited(D^SCC,scc)
return π
```

Der Algorithmus beginnt damit, den Startknoten zu markieren und den SCC-Liefergraphen zu bilden. Dann beginnt die Hauptschleife des Algorithmus, deren Funktion der des Algorithmus für optimale Pfadplanung in zykelfreien Graphen entspricht, es wird also in jedem Schleifendurchlauf eine Menge von Exportknoten (die in diesem Fall Export-SCCs sind) generiert. Ist ein solcher Exportknoten selbst nur ein einzelner Knoten, kann er in den Pfad aufgenommen. Besteht er aber aus mehreren Knoten, handelt es sich dabei um einen Zykel. Von diesem wird erst das MFVS ermittelt, diese Knoten an den Plan gehängt und dann aus dem SCC gelöscht. Der Rest des SCC wird dann in einer zweiten, analogen Schleife geplant. Dann werden die Knoten des MFVS ein zweites mal dem Plan hinzugefügt. In der letzten Zeile der Schleife wird das SCC dann in beiden Fällen als besucht markiert.

Auf die Angabe eines Algorithmus, der mithilfe der optimalen Pfadplanungsmethode Miconics-10-Strips-Planungsaufgaben löst sei verzichtet. Aber es kann noch eine genaue Angabe über die Länge optimaler Pläne gegeben werden:

$$m^\star = \#^{b,d}(\pi^\star) + \#^m(\pi^\star) = 2|P| + |L^d \cup L^e| + |MFVS|$$

Diese Formel ergibt sich daraus, dass jeder relevante Knoten, also auch diejenigen, die das MFVS bilden, einmal angefahren wird, was $|L^d \cup L^e|$ entspricht. Die Knoten des MFVS werden dann ein zweites mal addiert. Auch die Komplexität optimalen Planens auf der Miconic-10-Strips-Domäne kann abschließend beschrieben werden.

Satz 6.2 ***Komplexität optimalen Planens auf der* Miconic-10-Strips-*Domäne***
Optimales Planen auf der Miconic-10-Strips-*Domäne ist, sofern $P \neq NP$, NP-vollständig.*

Damit wurden ein optimaler sowie ein suboptimaler Planer zur Lösung von Planungsaufgaben auf der Miconic-10-Strips-Domäne vorgestellt. Es zeigte sich, wie viel einfacher suboptimales, domänenspezifisches Planen sein kann: Während der suboptimale Planer Pläne, deren Länge höchstens dem $\frac{7}{6}$-fachen der Minimalkosten entspricht, in polynomieller Zeit generiert, muss der optimale Planer mit dem Minimum Feedback Vertex Set-Problem ein NP-vollständiges lösen. Inwieweit die Leistung des Planers darunter leidet, wird im nächsten Abschnitt diskutiert werden.

6.3 Ergebnisse

Die Präsentation der Ergebnisse an dieser Stelle wird aus zwei Gründen ungewöhnlich kurz ausfallen. Zum einen fehlt leider repräsentatives Vergleichsmaterial vom IPC 2, was daran liegt, dass dieser im Jahr 2000 noch nicht in einen optimalen und einen suboptimalen Zweig aufgeteilt war, und so lediglich Vergleichsdaten suboptimaler Planer vorhanden sind. Die Planungsaufgaben wurden aber von mehreren der angetretenen Planer verhältnismäßig gut gelöst: Die Rechenzeiten lagen auf einem Pentium 3 450MHz-Prozessor mit 256 MB RAM zwischen weniger als einer Sekunde für die einfachsten und mehreren Minuten für etwas komplexere Probleme – es wurden aber wie beschrieben lediglich suboptimale Pläne gefordert.
Der zweite Grund liegt darin, dass die Laufzeiten des implementierten optimalen Planers so gering sind, dass eine genaue Angabe keinen großen Erkenntnisgewinn mit sich bringt. Alle Probleme wurden auf einem Computer mit Intel Pentium 4 1,7 GHz-Prozessor mit 512 MB RAM in Zeiten im Hunderdstelsekundenbereich gelöst. Damit kann festgehalten werden, dass die Laufzeiten des optimalen Planers denen der

suboptimalen Planer auf dem IPC 2 wie erwartet weit überlegen sind.

Abgeschlossen sei dieses Kapitel mit der Feststellung, dass trotz komplexer Probleme wie MFVS durch die Aufteilung einer Planungsaufgabe in kleine Komponenten und die Verwendung aller möglichen Domäneninformationen optimales Planen mit sehr guten Laufzeiten möglich ist. Im nächsten Kapitel wird daher beschrieben werden, wie mit einem vergleichbaren Ansatz auch für die LOGISTICS-Domäne ein optimaler Planer entwickelt wurde.

7 LOGISTICS

Der Aufbau dieses Kapitels entspricht dem von Kapitel 6, es wird also erst die Logistics-Domäne formalisiert, dann ein suboptimaler Planer kurz skizziert sowie ein optimaler Planer beschrieben. Der letzte Abschnitt beschreibt dann die Ergebnisse des implementierten Algorithmus.

7.1 Formalisierung

Während vor einigen Jahren Blocksworld noch die meist beschriebene Handlungplanungsdomäne war, kann dies mittlerweile am ehesten über die Logistics-Domäne gesagt werden, welche wie Miconics-10-Strips ein Spezialfall der Transportplanungsfamilie ist. Im Gegensatz zu dieser handelt es sich bei Logistics aber nicht um eine strikte Vereinfachung, da in dieser mit Lastwagen und Flugzeugen zwei Typen von Fahrzeugen gegeben sind, die sich auf unterschiedlichen Teilen des Wegenetzgraphen bewegen können. Zudem können beide Fahrzeugarten jeweils in beliebiger Anzahl existieren. Lastwagen sind dabei auf die Fortbewegung innerhalb einer Stadt beschränkt, während Flugzeuge sich nur zwischen den Flughäfen der Städte bewegen können, von denen je einer in jeder Stadt existiert. Analog zur Miconics-10-Strips-Domäne ist dagegen, dass alle Fahrzeuge unbegrenzte Kapazität besitzen, keinen Treibstoff verbrauchen und das Ziel wie in jeder Transportplanungsdomäne darin besteht, eine Menge von Paketen an einen Zielort zu transportieren.
Es wird an dieser Stelle darauf verzichtet, eine Beschreibung der Logistics-Domäne analog zur PDDL-Beschreibung einer der beiden Planungswettbewerbe anzugeben. Stattdessen sei sie direkt als Spezialfall der Transportplanungsfamilie definiert:

Definition 7.1 Logistics-*Domäne als Spezialfall der Transportplanungsfamilie*
Die **Logistics**-*Domäne* $\mathcal{D}_{\text{Log}} = \langle \mathcal{P}, \mathcal{V}, \mathcal{O}, \mathcal{C}, \widehat{w} \rangle$ *ergibt sich als Spezialfall der Transportplanungsfamilie* $\mathcal{D}^{\text{Trans}}$ *folgendermaßen:*

- $\mathcal{P} = \mathcal{P}^{\mathcal{C}}_{\text{Trans}} \cup \{\rho_1, \ldots, \rho_3\}$*:*
  - $\rho_1 = \langle \texttt{truck}, 1 \rangle$
  - $\rho_2 = \langle \texttt{airplane}, 1 \rangle$
  - $\rho_3 = \langle \texttt{isAirport}, 1 \rangle$
- $\mathcal{V} = \mathcal{V}^{\mathcal{C}}_{\text{Trans}} \cup \{\psi_1\}$*:*
  - $\psi_1 = \langle \texttt{City}, 1 \rangle$
- $\mathcal{O} = \{\sigma^1, \sigma^2, \sigma^2_{\text{Trans}}, \sigma^3_{\text{Trans}}\}$
  - $\sigma^1 = \langle \texttt{moveTruck}, \{t, l_1, l_2\}, \text{pre}(t, l_1, l_2), \text{eff}(t, l_1, l_2) \rangle$*, wobei*
    $\text{pre}(t, l_1, l_2) = \texttt{connected}(l_1, l_2) \wedge \texttt{mobileAt}(t, l_1) \wedge \texttt{Fuel}(t) \geq 1 \wedge \texttt{City}(l_1) = \texttt{City}(l_2)$ *und*
    $\text{eff}(t, l_1, l_2) = \texttt{mobileAt}(t, l_2) \wedge \neg\texttt{mobileAt}(t, l_1)$
  - $\sigma^2 = \langle \texttt{moveAirplane}, \{a, l_1, l_2\}, \text{pre}(a, l_1, l_2), \text{eff}(a, l_1, l_2) \rangle$*, wobei*
    $\text{pre}(t, l_1, l_2) = \texttt{connected}(l_1, l_2) \wedge \texttt{mobileAt}(a, l_1) \wedge \texttt{Fuel}(a) \geq 1 \wedge \texttt{isAirport}(l_2)$ *und*
    $\text{eff}(t, l_1, l_2) = \texttt{mobileAt}(a, l_2) \wedge \neg\texttt{mobileAt}(a, l_1)$
- $\mathcal{C} = \langle \mathcal{C}_0, \mathcal{C}_\star \rangle$ *mit* $\mathcal{C}_0 = \mathcal{C}_0^{\text{Trans}} \cup \{c_0^1, \ldots, c_0^{12}\}$ *und* $\mathcal{C}_\star = \mathcal{C}_\star^{\text{Trans}}$*:*

- $c_0^1 = \forall\ \omega\ (\mathtt{portLoc}(\omega) \vee \mathtt{mobile}(\omega) \vee \mathtt{location}(\omega) \vee \mathtt{portable}(\omega) \vee \mathtt{truck}(\omega) \vee \mathtt{airplane}(\omega))$
- $c_0^2 = \forall\ t\ (\mathtt{truck}(t) \Rightarrow \mathtt{mobile}(t))$
- $c_0^2 = \forall\ a\ (\mathtt{airplane}(a) \Rightarrow \mathtt{mobile}(a))$
- $c_0^3 = \forall\ \omega\ (\neg(\mathtt{truck}(\omega) \wedge \mathtt{airplane}(\omega)))$
- $c_0^4 = \forall\ l\ (\mathtt{isAirport}(l) \Rightarrow \mathtt{location}(l))$
- $c_0^5 = \forall\ l\ (\mathtt{City}(l) \neq \bot \Rightarrow \mathtt{location}(l))$
- $c_0^6 = \forall\ l_1, l_2\ ((l_1 \neq l_2) \wedge (\mathtt{location}(l_1) \wedge \mathtt{location}(l_2) \wedge (\mathtt{City}(l_1) = \mathtt{City}(l_1))) \Rightarrow \mathtt{connected}(l_1, l_2))$
- $c_0^7 = \forall\ l_1, l_2\ ((l_1 \neq l_2) \wedge (\mathtt{location}(l_1) \wedge \mathtt{location}(l_2) \wedge \mathtt{isAirport}(l_1) \wedge \mathtt{isAirport}(l_2)) \Rightarrow \mathtt{connected}(l_1, l_2))$
- $c_0^8 = \forall\ l_1\ (\mathtt{location}(l_1) \Rightarrow (\exists^{=1}\ l_2\ (\mathtt{isAirport}(l_2) \wedge (\mathtt{City}(l_1) = \mathtt{City}(l_2)))))$
- $c_0^9 = \forall\ m\ (\mathtt{mobile}(m) \Rightarrow \mathtt{Fuel}(m) = \infty)$
- $c_0^{10} = \forall\ m\ (\mathtt{mobile}(m) \Rightarrow \mathtt{Capacity}(m) = \infty)$
- $c_0^{11} = \exists\ a\ \mathtt{airplane}(a)$
- $c_0^{12} = \forall\ l_1\ (\mathtt{location}(l_1) \Rightarrow (\exists\ l_2, t\ (\mathtt{location}(l_2) \wedge \mathtt{truck}(t) \wedge \mathtt{mobileAt}(t, l_2) \wedge (\mathtt{City}(l_1) = \mathtt{City}(l_2)))))$

- $\widehat{w}(\sigma_1) = \widehat{w}(\sigma_2) = 1$

Da Teile dieser Definition aus der Beschreibung der Transportplanungsfamilie geerbt werden, seien einige Zusammenhänge etwas ausführlicher beschrieben. So gibt die folgende Abbildung erst einmal einen Überblick über den hierarchischen Zusammenhang der existierenden Objekttypen:

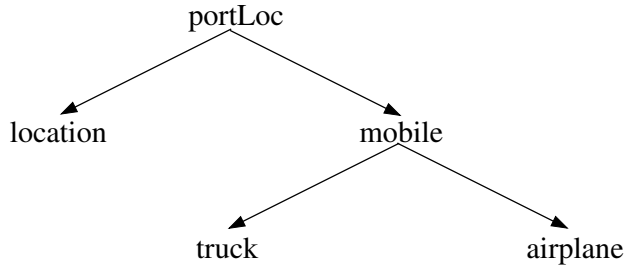

Abbildung 7.1: Typenhierarchie der Logistics-Domäne

Ein Blick auf die Typenbeschreibung c_0^1 zeigt, dass der einzige, nicht in dieser Abbildung enthaltene Objekttyp `portable` ist – dieser besitzt keine Sub- oder Supertypen, und muss daher nicht bildlich dargestellt werden. Die beiden anderen, neu definierten und einstelligen Prädikaten- bzw. numerischen Variablenschemata `isAirport`(l) und `City`(l) sind in der Typenbeschreibung dagegen nicht enthalten, und stellen dadurch keine Objekttypen dar. Dennoch sind diese als einstellige Prädikate leicht damit zu verwechseln, im Fall der Logistics-Domäne beschreiben sie aber lediglich Eigenschaften von Orten. Es hätte an dieser Stelle auch eine andere Modellierung gewählt werden können, dann wäre es aber nötig geworden, das Prädikatenschema `connected`(l_1, l_2) doppelt zu definieren, da nach Kapitel 3 in wohldefinierten Domänen keine Objekttypen in Operatorenvorbedingungen enthalten sein dürfen. In dieser Modellierung wurde die Anwendbarkeit der beiden Bewegungsoperatoren aber durch ebendiese Prädikate unterschieden, wodurch dieselbe Verbundenheitsrelation beibehalten werden kann. Mit dieser Definition werden also die beiden Schemata, zusammen mit den Beschränkungen c_0^6 bis c_0^8, dafür genutzt, einen zweistufigen, hierarchisch-vollständigen Wegenetzgraphen zu modellieren. Dieser wurde in Kapitel 3 definiert und ein Beispielsgraph dazu angegeben. Übertragen auf die Logistics-Domäne entspricht die folgende Abbildung demselben Beispiel:

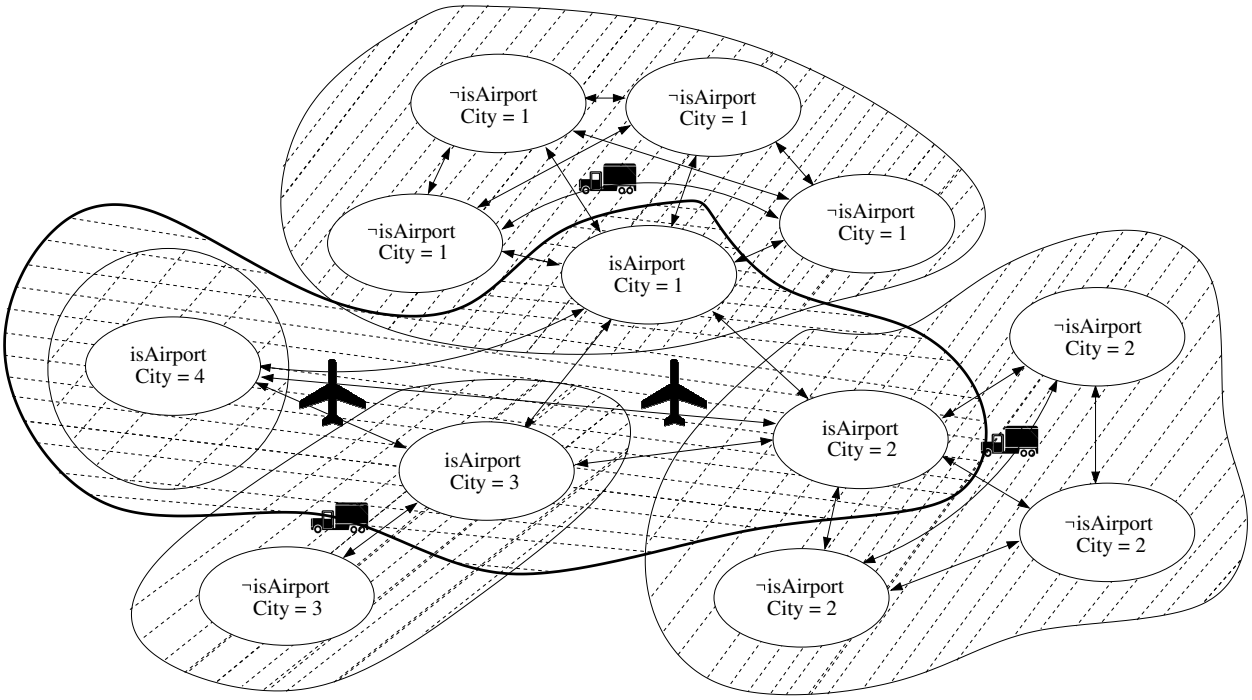

Abbildung 7.2: Exemplarischer Wegenetzgraph einer LOGISTICS-Planungsaufgabe

Der horizontal gestreifte, dick umrahmte Bereich, in welchem die Flughäfen liegen, bildet den vollständigen Graphen erster Hierarchiestufe, den *Flugnetzgraphen*, welcher nur von Flugzeugen benutzt werden kann. Die vier etwas dünner umrahmten, schräg gestreiften Zonen sind die *Stadtnetzgraphen* zweiter Hierarchiestufe. In diesen können sich die Lastwagen bewegen, allerdings nur innerhalb der Stadt, in der sie sich im Initialzustand befinden. Beide seien im Folgenden, sofern sie unabhängig vom Rest des Wegenetzgraphen betrachtet werden, als Subwegenetzgraphen bezeichnet. Der relevante Zustandsraum der LOGISTICS-Domäne ergibt sich wie folgt:

Definition 7.2 ***Relevanter Zustandsraum der* LOGISTICS*-Domäne***
Der relevante Zustandsraum $S(\mathcal{T}_{\text{LOG}})$ *beliebiger Planungsaufgaben* $\mathcal{T}_{\text{LOG}}$ *auf der* LOGISTICS*-Domäne* $\mathcal{D}_{\text{LOG}}$ *ist gegeben als:*

GRUNDELEMENTE: $\langle W, P, F, T, D\rangle$, *wobei* $W = \langle W^A, W^C\rangle$ *der zweistufige hierarchisch-vollständige Wegenetzgraph ist, bestehend aus dem* ***Flugnetzgraphen*** *erster Hierarchiestufe* $W^A = \langle A, A^2\rangle$ *mit der Menge der* ***Flughäfen*** A *sowie der Menge der* ***Stadtnetzgraphen*** *zweiter Hierarchiestufe* $W^C = \{W_1^C \dots W_{|A|}^C\}$, $W_i^C = \langle L_i, L_i^2\rangle$ *mit der* ***Stadt*** L_i [1]. *Jedes* $l \in \bigcup_i L_i$ *heiße* ***Ort****. Des Weiteren ist* P *die Menge der* ***Pakete****,* F *die Menge der* ***Flugzeuge****,* T *die Menge der* ***Lastwagen****, und* $D = \langle \text{loc}_0, \text{loc}_\star\rangle$ *der* ***Liefergraph*** *mit der* ***Startortfunktion*** $\text{loc}_0 : P \to \bigcup_i L_i$ *und der* ***Zielortfunktion*** $\text{loc}_\star : P \to \bigcup_i L_i$.

ZUSTÄNDE: *2-Tupel* $\langle \text{mAt}, \text{pAt}\rangle$, *wobei* $\text{mAt} : F \cup T \to \bigcup_i L_i$ *die* ***Standortfunktion*** *der Fahrzeuge ist und* $\text{pAt} : P \to \bigcup_i L_i \cup F \cup T$ *die* ***Paketortfunktion*** *ist, die angibt, an welchem Ort* $l \in \bigcup_i L_i$ *oder in welchem Fahrzeug* $m \in F \cup T$ *die Pakete liegen.*

STARTZUSTAND: $\langle \text{mAt}_0, \text{pAt}\rangle$ *mit* $\text{pAt}(p) = \text{loc}_0(p)$ *für alle* $p \in P$

ZIELZUSTAND: *Jeder Zustand* $\langle \text{mAt}, \text{pAt}\rangle$ *mit* $\text{pAt}(p) = \text{loc}_\star(p)$ *für alle* $p \in P$

[1] A^2 und L_i^2 seien so definiert, dass kein Kanten existieren, die denselben Knoten als Ursprung und Ziel haben

OPERATOREN: *Jedes Flugzeug kann von einem Flughafen zu einem anderen* ***fliegen*** *und jeder Lastwagen kann von einem Ort zu einem anderen* ***fahren****, falls beide in derselben Stadt liegen. Flugzeuge und Lastwagen können ein Paket* ***einladen****, wenn sie sich am selben Ort wie das Paket befinden, und ein Paket* ***ausladen****, falls sie das Paket geladen haben.*

7.2 Planer

Ein approximierenden Algorithmus wird detailliert von Helmert in [Hel08] beschrieben, an dieser Stelle sei lediglich die Grundidee wiedergegeben. Er teilt die Planung dazu in drei Teile auf, nämlich in die *Einsammelphase*, die *Flugphase* sowie die *Austeilphase*. In der ersten Phase wird in jeder Stadt ein beliebiger Truck gewählt (der nach Beschränkung c_0^{12} existieren muss), welcher alle Pakete einsammelt und diese an den Flughafen transportiert. Dort werden alle Pakete, deren Ziel in einer anderen Stadt liegt, ausgeladen. In der Flugphase wird dann zufällig ein Flugzeug bestimmt (das nach c_0^{11} existieren muss), das alle Flughäfen mit Paketen für andere Städte anfliegt und diese an die Flughäfen der Zielstädte transportiert. In der Austeilphase fahren schließlich abermals beliebig gewählte Lastwagen die Pakete an ihren Zielort aus. Helmert zeigt, dass dieser Algorithmus $\frac{4}{3}$-approximierend ist, was auch sein Ergebnis für die Komplexität suboptimalen Planens auf der LOGISTICS-Domäne darstellt.

Satz 7.1 ***Komplexität suboptimalen Planens auf der* LOGISTICS*-Domäne***
LOGISTICS *ist* $\frac{4}{3}$*-approximierbar, liegt aber nicht in PTAS, sofern* $P \neq NP$*. Es gilt also* PLAN-$\mathcal{D}_{\text{LOG}} \in APX \setminus PTAS$.

Bevor nun eine Methode beschrieben wird, mit welcher optimale Pläne für beliebige, lösbare Planungsaufgaben auf der LOGISTICS-Domäne generiert werden können, seien einige, nicht bereits in Definition 7.2 gegebene Mengen und Eigenschaften definiert.

- Sei $L = \bigcup_i L_i$ die Menge der **Orte**.
- Sei $A(L_i)$ der **Flughafen der Stadt** L_i.
- Sei $A(l_i)$ der Flughafen, der in derselben Stadt wie l_i liegt. Dieser heiße auch **Flughafen des Ortes** l_i.
- Ein Paket $p \in P$ heiße **lokal**, wenn Start- und Zielort in derselben Stadt liegen.
- Ein Pakte $p \in P$ heiße **global**, wenn es nicht lokal ist.

Im vorigen Kapitel wurde bereits ein optimaler Planer für ein Mitglied der Transportplanungsdomäne beschrieben. Daher sei mit einer Analyse von Gemeinsamkeiten und Unterschieden der beiden Domänen begonnen. Ein erster Unterschied kann in der in der Form des Wegenetzgraphen festgestellt werden, betrachtet man aber jeden Subwegenetzgraphen separat, entsteht für jeden dasselbe Problem. Es muss also untersucht werden, inwieweit die hierarchische Verbundenheit dieser Graphen Anpassungen im optimalen MICONICS-10-STRIPS-Planer erfordert, und inwieweit dieser zumindest als Teil eines Algorithmus verwendet werden kann. Die Existenz eines hierarchischen Graphen ist offensichtlich eng verbunden mit der Existenz mehrerer Fahrzeugtypen in der LOGISTICS-Domäne, und tatsächlich kann festgestellt werden, dass eine separate Betrachtung der einzelnen Subwegenetzgraphen zu Planungsaufgaben führt, in welchen nur ein Fahrzeugtyp existiert. Damit muss diese Eigenschaft der LOGISTICS-Domäne nicht weiter betrachtet werden – für jedes beliebige Paar von Orten gibt es nur einen Fahrzeugtypen, der sich auf dieser Strecke bewegen kann. Dennoch ist dieser Teil der Domäne noch nicht analog zu MICONICS-10-STRIPS, da in jedem Subwegenetzgraphen noch immer mehrere Fahrzeuge (desselben Typs) zum Transport zur Verfügung stehen.

Durch die beschriebenen Unterschiede wird es notwendig, auch die `pickUp`- und `drop`-Aktionen erneut zu untersuchen. Dafür sei der Logistics-*Liefergraph* D^{Log} der Logistics-Domäne definiert. Dieser kann direkt aus dem Liefergraphen generiert werden, indem Kanten, die *globale* Pakete repräsentieren, gelöscht und folgendermaßen ersetzt werden:

- Wenn $\text{loc}_0(p) \notin A$ enthalte D^{Log} eine Kante $\langle \text{loc}_0(p), A(\text{loc}_0(p)) \rangle$.
- D enthalte eine Kante $\langle A(\text{loc}_0(p)), A(\text{loc}_\star(p)) \rangle$.
- Wenn $\text{loc}_\star(p) \notin A$ enthalte D^{Log} eine Kante $\langle A(\text{loc}_\star(p)), \text{loc}_\star(p) \rangle$.

Kanten, die den Lieferweg *lokaler* Pakete beschreiben, bleiben im Logistics-Liefergraphen unverändert erhalten. Die folgende Abbildung zeigt, wie aus dem Liefergraphen der Logistics-Liefergraph erstellt wird:

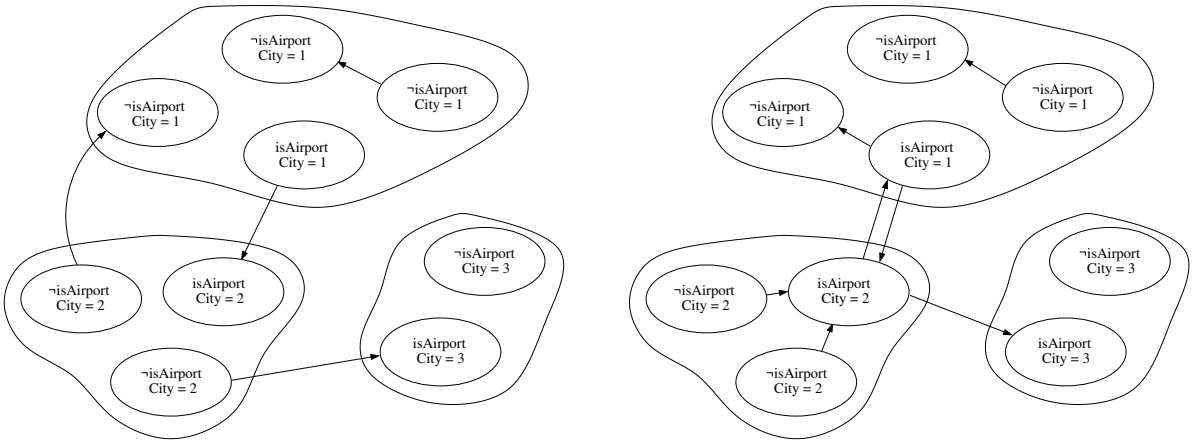

Abbildung 7.3: Liefergraph D (links) und zugehöriger Logistics-Liefergraph D^{Log} (rechts)

Durch den folgenden Methodenaufruf werde dieser Graph für eine beliebige Logistcs-Planungsaufgabe $\mathcal{T}$ generiert:

$$D^{Log} = \texttt{createLogDelGraph}(\mathcal{T})$$

Auch im Logistics-Liefergraph sei jeder separat betrachtete, auf eine Stadt oder das Luftwegnetz beschränkte Graph als *Subliefergraph* bezeichnet. Der Logistics-Liefergraph sagt bereits einiges über sicher notwendige `pickup`- und `drop`-Aktionen aus: Jede der bis zu drei Kanten, die den Weg eines Paketes in diesem Graphen repräsentieren, beginnt an einem Knoten, an dem eine `pickup`-Aktion nötig ist, und endet an einem, an welchem eine `drop`-Aktion ausgeführt werden muss. Für lokale Pakete werden also weiterhin dieselben beiden Aktionen wie in Miconics-10-Strips benötigt. Globale Pakete müssen dagegen, je nachdem, ob sie bereits im Initialzustand an einem Flughafen liegen oder an einen solchen transportiert werden sollen, bis zu dreimal ein- und wieder ausgeladen werden. Insgesamt sind also die folgenden `pickUp`- und `drop`-Aktionen sicher in jedem Plan für eine Logistics-Planungsaufgabe enthalten:

- Jedes lokale $p \in P$ muss an seinem Startort von einem Lastwagen eingeladen werden.
- Jedes lokale $p \in P$ muss an seinem Zielort von einem Lastwagen ausgeladen werden.
- Jedes globale $p \in P$ muss an seinem Startort von einem Lastwagen eingeladen werden, wenn der Startort nicht selbst ein Flughafen ist.
- Jedes globale $p \in P$ muss am Flughafen des Startortes von einem Lastwagen ausgeladen werden, wenn der Startort nicht selbst ein Flughafen ist.

- Jedes globale $p \in P$ muss am Flughafen des Startortes von einem Flugzeug ausgeladen werden.
- Jedes globale $p \in P$ muss am Flughafen des Zielortes von einem Flugzeug ausgeladen werden.
- Jedes globale $p \in P$ muss am Flughafen des Zielortes von einem Lastwagen eingeladen werden, wenn der Zielort nicht selbst ein Flughafen ist.
- Jedes globale $p \in P$ muss am Zielort von einem Lastwagen ausgeladen werden, wenn der Zielort nicht selbst ein Flughafen ist.

Damit sind sicher notwendige Ladeoperationen beschrieben. Bezogen auf die separate Betrachtung der Subliefergraphen ergibt sich dabei bislang kein Unterschied zur MICONIC-10-STRIPS-Domäne. Daher sei im Folgenden jeder Knoten mit einer eingehenden Kante als *Ziel* des Paketes im entsprechenden Subliefergraphen bezeichnet, sowie jeder Knoten mit ausgehender Kante als *Startort*. In Bezug auf Liefergraphen, die aus mehr als einem Subliefergraphen bestehen, sei dagegen der Begriff *Zwischenziel* verwendet.
Damit bleibt noch zu klären, ob möglicherweise zusätzliche Ladeaktionen in optimalen Plänen notwendig sein können. Dabei kommen grundsätzlich nur drei Fälle innerhalb eines Subliefergraphs oder an deren Schnittstellen, den Flughäfen, in Frage:

- Dasselbe Fahrzeug lädt ein Paket aus und irgendwann später an demselben Ort wieder ein.
- Eine Fahrzeug eines Typs lädt ein Paket aus und ein Fahrzeug desselben Typs lädt es an demselben Ort wieder ein.
- Ein Fahrzeug eines Typs lädt ein Paket aus und ein Fahrzeug eines anderen Typs lädt es an demselben Ort wieder ein.

Die ersten beiden beschreiben Aktionen können nur innerhalb eines Subliefergraphen entstehen, während der letzte Punkt die einzige Möglichkeit an den Schnittstellen darstellt. Diese kann auch am einfachsten ausgeschlossen werden: Die maximal zwei sicher notwendigen `pickup`- und `drop`-Aktionen wurden bereits beschrieben. Zusätzliche können nur geschehen, wenn ein Flugzeug ein Paket an einer Stadt auslädt, das nicht die Zielstadt ist. Da dies offensichtlich niemals Teil optimaler Pläne sein kann, ist auch die zweite Möglichkeit, dass sich ein Paket in einer dritten Stadt befindet und in ein Flugzeug umgeladen werden muss sicher nicht möglich.
Auch die erste beschriebene Aktion kann aufgrund der unbegrenzten Kapazität nicht Teil optimaler Pläne sein, da dadurch zwei Operatoren mehr angewendet werden, um wieder denselben Zustand – in Bezug auf dieses Paket und das Fahrzeug – herzustellen. Damit bleibt die mittlere in der Auflistung beschriebene Aktion. Auch diese kann schnell verworfen werden: Da alle Subwegenetzgraphen vollständig sind, kann jedes Fahrzeug jeden Knoten in diesem mit genau einer Bewegungsaktion erreichen. Der einzige mögliche Vorteil, der durch das Umladen eines Paketes entstehen kann, ist wenn ein anderes Fahrzeug, aus welchen Gründen auch immer, sowieso schon an das Ziel des Paketes fahren muss. Dann wird diese eine Bewegungsaktion zwar eingespart, es kommen aber je eine `pickup`- und `drop`-Aktion hinzu, wodurch insgesamt mehr Operatoranwendungen nötig werden.

Damit kann auch für die LOGISTICS-Domäne gesagt werden, dass die optimale Planung der Ein- und Ausladevorgänge trivial ist. Noch nicht geklärt ist allerdings, welches Fahrzeug am besten geeignet ist, bestimmte Pakete zu transportieren. Dafür sei zuerst ein für die folgenden Erläuterungen wichtiges Konzept beschrieben, die Zusammenhangskomponenten (CC) [CLRS01]. Diese entsprechen exakt den im vorigen Kapitel beschriebenen SCCs, wenn jede Kante als in beide Richtungen zeigend betrachtet wird. Jedes CC eines LOGISTICS-Liefergraphen stellt dabei selbst wieder einen LOGISTICS-Liefergraphen dar. Dabei ist zu beachten, dass diese Art der Aufteilung nichts mit den Stadtwegenetzen oder dem Luftwegenetz gemein hat. Durch die

Generierung verbundener Komponenten kann jede Stadt Teil beliebig vieler CCs sein, und auch das Luftwegenetz kann über mehrere CCs verteilt sein. Es ist aber unmöglich, dass derselbe Knoten Teil mehrerer CCs ist – dann wären diese schließlich über diesen Knoten miteinander verbunden.
Das gilt insbesondere auch für den Flughafen jeder Stadt, weshalb in allen Städten zwar mehrere `lokale CCs` existieren können, deren Knoten alle in dieser Stadt liegen müssen, sie aber Teil von höchstens einem *globalen CC* sein können, denen Knoten beliebig vieler Städte angehören können (und Knoten von mindestens zwei Städten angehören müssen). Dementsprechend können globale CCs auch weiter in einen Flughafenstadtteil für jede enthaltene Stadt sowie genau einen Luftwegteil unterteilt werden. Um diese verschiedenen Konzepte übersichtlich darzustellen seien sie noch einmal in einer Liste angeführt:

- Eine Zusammenhangskomponente (CC) ist ein maximal großer Teil des Logistics-Liefergraphs, in welchem jeder Knoten von jedem Knoten aus erreichbar ist, wenn alle Kanten als in beide Richtungen zeigend betrachtet werden. Der Logistics-Liefergraph lässt sich so vollständig in disjunkte CCs aufteilen.
- Ein CC ist lokal, wenn nur Knoten derselben Stadt darin enthalten sind.
- Ein CC ist global, wenn es nicht lokal ist.
- Ein globales CC kann weiter aufgeteilt werden in je einen Flughafenstadtteil für jede Stadt, der mindestens ein Knoten des CCs angehört, sowie einen Luftwegteil, der alle Flughäfen des CCs angehören.
- Zusätzlich seien lokale CCs sowie der optional einzig existierende Flughafenstadtteil jeder Stadt als **Stadtteil** einer Stadt bezeichnet.

Es sei an dieser Stelle darauf hingewiesen, dass alle diese Komponente von der Form her Liefergraphen bzw. Logistics-Liefergraphen darstellen, obwohl dies durch ihren Namen nicht mehr sofort ersichtlich ist. Zudem ist zwar sicher, dass der Flughafenstadtteil einer Stadt, sofern er existiert, den Flughafen dieser Stadt auch beinhaltet, es kann, wenn die Stadt Teil keines globalen CCs ist, aber durchaus auch möglich sein, dass der Flughafen der Stadt in einem lokalen CC liegt. In diesem Fall hat er aber auch keine besondere Funktion und kann wie jeder andere Knoten betrachtet werden.
Bereits jetzt sei ein Ziel dieses Algorithmus beschrieben: Alle CCs, global wie lokal, sollen unabhängig voneinander und nacheinander geplant werden. Dass der optimale Pfad jedes CC unabhängig von allen anderen CCs sind ist offensichtlich – alle Pakete haben ihren Start- und Zielort immer in demselben CC, und da sich keine Pfade zweier CCs kreuzen können, ist auch kein schnellerer Weg durch zwischenzeitliches Wechseln in ein anderes CC möglich. Nicht unabhängig ist aber die Wahl des oder der Fahrzeuge, die sich auf diesem Pfad bewegen. Dabei wurde bereits im vorigen Kapitel beschrieben, dass die optimale Pfadlänge um eins kürzer ist, wenn sich der Fahrzeug zu Beginn an einem Knoten befindet, der einen Exportknoten im Liefergraphen darstellt, oder wenn es auf einem Knoten des MFVS startet. Diese Erkenntnis kann auch auf separat betrachtete Subliefergraphen übertragen werden. Damit muss vermieden werden, dass zuerst ein Stadtteil oder Luftwegteil geplant wird, in welchem dieser Fall eintreten könnte, und dann erst diejenigen, in denen es nicht sein kann.
Dieses Problem lässt sich aber einfach lösen: Die CCs werden so angeordnet, dass zuerst lokale CCs geplant werden, die mindestens einen Ort mit einem Lastwagen beinhalten. Es ist zwar nicht sicher, dass in all diesen Stadtteilen dann auch Bewegungskosten eingespart werden, es ist aber sicher, dass dabei kein Lastwagen von einem solchen, Bewegungskosten sparenden Knoten abgezogen wird, ohne das dort liegende Paket einzuladen. Werden darauf die globalen CCs geplant, die mindestens einen Flughafen beinhalten, der Startort eines Flugzeuges ist, kann derselbe Fall auch für Flugzeuge nicht eintreten. Darauf folgend müssen die globalen CCs geplant werden, in denen kein Flugzeug enthalten ist. Dadurch werden auch die letzten, möglicherweise noch auf Bewegungskosten sparenden Knoten startenden Lastwagen optimal geplant, wodurch zuletzt alle lokalen CCs ohne Lastwagen geplant werden können. Daher teile die folgende Methode den Logistics-Liefergraphen in fünf unterschiedliche Arten von CCs ein:

$$\texttt{createCCs}(D^{Log}, CC^{l+}, CC^{g+}, CC^{g-}, CC^{l-}, CC^{m})$$

Diese fünf Mengen können folgendermaßen beschrieben werden:

- CC^{l+} sei die Menge aller lokalen CCs, die mindestens einen Ort beinhalten, auf welchem sich ein Lastwagen im Initialzustand befindet.
- CC^{l-} sei die Menge aller lokalen CCs, die keine Ort beinhalten, auf welchem sich ein Lastwagen im Initialzustand befindet.
- CC^{g+} sei die Menge aller globalen CCs, die mindestens einen Flughafen beinhalten, auf welchem sich ein Flugzeug im Initialzustand befindet.
- CC^{g-} sei die Menge aller globalen CCs, die keinen Flughafen beinhalten, auf welchem sich ein Flugzeug im Initialzustand befindet.
- CC^{m} sei die Menge aller (lokalen) CCs, die nur einen einzigen Knoten beinhalten, auf dem sich zusätzlich ein Fahrzeug befindet.

Durch diese Aufteilung wird nicht mehr der gesamte Logistics-Liefergraph in disjunkte Mengen unterteilt. Es fallen aber nur solche Knoten weg, die keine ein- oder ausgehende Kante haben und auf welchen zusätzlich kein Fahrzeug steht. Diese Knoten werden durch die Vollständigkeit der Subwegenetzgraphen in einem optimalen Plan aber sicher niemals angefahren werden, und können daher eliminiert werden. Damit ist eine Reihenfolge beschrieben, in welcher alle CCs unabhängig von allen anderen nacheinander geplant werden können, ohne dass jemals ein Fahrzeug aus einem noch nicht geplanten CC abgezogen wird. Damit bleibt der letzte in Bezug auf die Fahrzeugauswahl noch zu klärende Fall, inwieweit weitere Bewegungskosten eingespart werden können, wenn mehrere Fahrzeuge in einem Subliefergraphen beginnen, wofür zwei Beispiele angegeben seien.

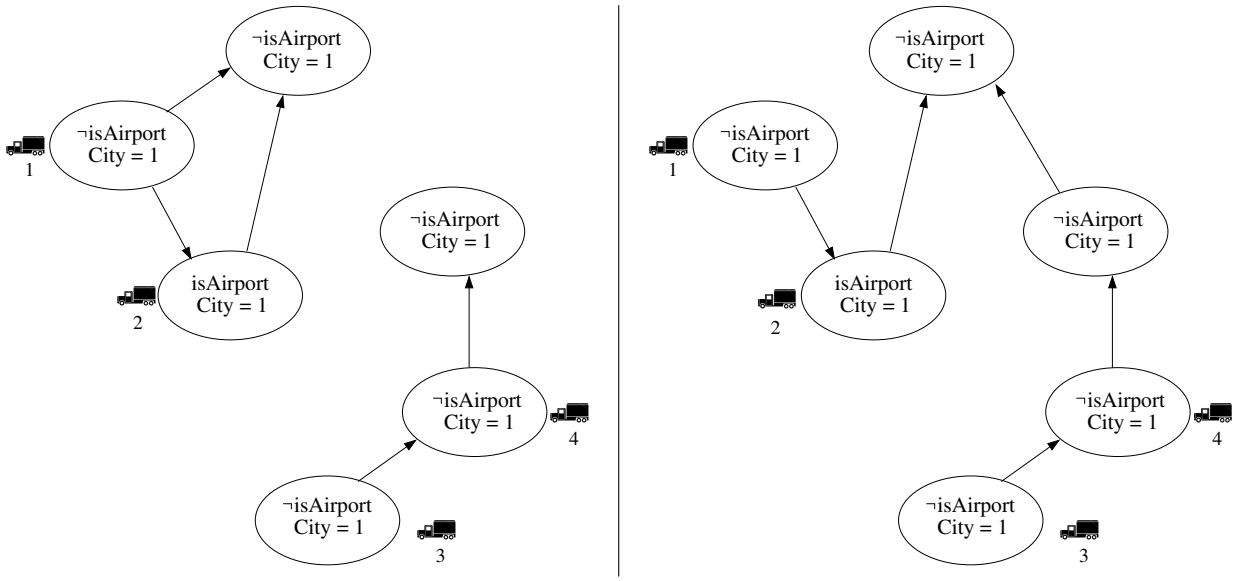

Abbildung 7.4: Subliefergraphen mit zwei (links) bzw. einem Stadtteil

In der Abbildung links sind im rechten Stadtteil zwei Lastwagen enthalten. Es zeigt sich schnell, dass kein Vorteil erzielt werden kann, wenn beide Fahrzeuge ihre Pakete aufladen – es ist unerheblich, ob Lastwagen 3 den Stadtteil alleine bedient, oder ob beide das Paket an ihrem Startort einladen und zum Zielort transportieren. Interessanter ist aber der linke Stadtteil desselben Beispiels: Hier entstehen sogar Mehrkosten, wenn

beide Lastwagen das Paket an ihrem Startort einladen, da dann beide an den einzigen Ort ohne Fahrzeug fahren müssen, aber kein Bewegungsoperator eingespart wird. In diesem Fall ist es also sogar besser, nur ein Fahrzeug zur Planausführung zu verwenden. Interessant ist auch das zweite Beispiel: Intuitiv wäre zu vermuten, dass, wenn mehrere Fahrzeuge an einem Exportknoten beginnen, dass dann auch jedes einen Bewegungsoperator spart. Tatsächlich zeigt sich aber, dass auch in diesem Fall keine Bewegungskosten gespart werden können. Damit kann die Analyse der optimalen Fahrzeugwahl abgeschlossen werden: Es existiert kein Fall, in welchem es am kostengünstigsten ist, mit mehreren Fahrzeugen einen Stadt- oder Luftwegteil zu planen, aber es ist möglich, dass dadurch Mehrkosten entstehen. Daher wird im Folgenden immer mit exakt einem Fahrzeug für alle Stadt- und Luftwegteile geplant werden, unabhängig davon, ob mit mehreren Fahrzeugen auch ein Plan derselben Länge generiert werden könnte.

Damit können lokale CCs, da nur noch mit einem einzigen Fahrzeug in einem vollständigen Wegenetzgraphen geplant werden kann, mit dem Planer des vorigen Kapitels berechnet werden. Der Miconics-10-Strips-Planer sei daher durch die folgende Methode aufgerufen:

$$\pi = \texttt{solveM10}(CC, V, \mathcal{T})$$

Wie man sieht sind zwei Teile der Methode leicht verändert: Zum einen wird ein zusätzlicher Parameter in Form einer Logistics-Planungsaufgabe übergeben. Dies stellt lediglich eine formale Änderung dar, die es ermöglichen soll, dass der Planer einen Logistics-Plan und keinen Miconics-10-Strips-Plan zurück gibt, was offensichtlich in polynomieller Zeit transformiert werden kann. Die zweite Änderung besteht in der übergebenen Menge von Knoten V, was im vorigen Kapitel bereits diskutiert wurde.

Damit bleibt noch die Planung globaler CCs zu beschreiben. Es sei an dieser Stelle angemerkt, dass damit die Reihenfolge der Beschreibung nicht mehr mit der Reihenfolge, in welcher die CCs später im Algorithmus abgearbeitet werden übereinstimmt, da lokale CCs ohne Lastwagen bereits beschrieben, im Algorithmus aber erst als letztes geplant werden. Da nun keine Verwechslungsgefahr mehr besteht, seien Flughafenstadtteile im Folgenden als Liefergraph einer Stadt bezeichnet, und der Luftwegteil als Luftliefergraph. Die Bezeichnung Liefergraph einer Stadt ist daher eindeutig, da nun, da die Planung lokaler CCs beschrieben wurde, jede Stadt nur in einem noch zu planenden CC enthalten sein kann.
Sicher wird der Planer eine Methode benötigen, die ein globales CC cc^g in eine Menge von Liefergraphen von Städten CC^c und den Luftliefergraphen cc^a aufteilt. Städte, die nur aus dem Flughafen bestehen, seien darin nicht enthalten, da in diesen keine Bewegungen von Lastwagen notwendig sind. Im Luftliefergraphen sei der entsprechende Flughafen aber selbstverständlich repräsentiert.

$$\texttt{globalCC2SubGraphs}(cc^g, cc^a, CC^c)$$

Betrachtet man jeden Bestandteil des CCs separat, kann der optimale Plan offensichtlich mit dem Miconics-10-Strips-Planer generiert werden. Leider ist dies im Allgemeinen aber nicht möglich, da alle Pakete, die erst in eine Stadt geliefert werden müssen, sich zu Beginn der Planung noch nicht in der jeweiligen Stadt befinden. Es wird also Städte geben müssen, in welchen der Liefergraph in mindestens zwei Teile geteilt werden muss. Es gibt aber auch Städte, die unabhängig vom gewählten Pfad des Flugzeugs immer optimal geplant werden können, nämlich Exportstädte. Eine Stadt ist dabei eine Exportstadt, wenn ihr Flughafen im Luftliefergraphen ein Exportknoten ist. Damit können auch hier zumindest Teile des Miconics-10-Strips-Planers angewendet werden: Das Flugzeug kann zu Beginn der Planung sicher immer eine Exportstadt anfliegen, welche dann optimal geplant wird, dann alle Pakete einladen (die, da die Stadt geplant wurde, am Flughafen sein müssen) und den Flughafen der Stadt dann im Luftliefergraphen markieren. Damit ist auch eine Methode beschrieben, mit deren Hilfe globale, zykelfreie CCs optimal geplant werden können. Dies sei durch die folgende Methode gegeben, die lediglich den simplen Algorithmus zur Generierung optimaler Pfade in zykelfreien Graphen des vorigen Kapitels auf den Luftliefergraphen anwendet, und den

entstehenden Pfad mit dem Flugzeug abfliegt, wobei an jedem Flughafen nach Abladen der Pakete die entsprechende Stadt mithilfe des MICONICS-10-STRIPS-Planers geplant wird:

$$\pi \texttt{ = planCycleFreeGlobalCC(}cc^g\texttt{,}V\texttt{,}\mathcal{T}\texttt{)}$$

Damit sind alle möglichen Zusammenhangskomponenten optimal gelöst, lediglich globale CCs, die einen Zykel enthalten, sind noch zu beschreiben. Leider ist die Generierung optimaler Pläne für diese nicht trivial, und es muss eine weitere Zwischenschritt analog zum MICONIC-10-STRIPS-Planer eingelegt werden. Dieser betrifft den Luftliefergraphen, der im Fall, dass ein Zykel darin enthalten ist, leicht transformiert werden muss: Anstatt für jeden Flughafen einen Knoten zu enthalten, enthalte der *SCC-Luftliefergraph* cc^{scc} für jede im Luftliefergraphen enthaltene, starke Zusammenhangskomponente einen Knoten, der alle ein- und ausgehenden Kanten der darin aufgehenden Flughäfen behalte:

$$\texttt{airDelGraph2SCCGraph(}cc^a\texttt{,}scc^a\texttt{)}$$

Abermals können Teile dieses Graphen mit den üblichen Methoden geplant werden, nämlich alle SCCs, die nur aus einem Flughafen bestehen. Auch die jetzt vorgestellte Methode wird dementsprechend auf einer Schleife aufbauen, die nach und nach Exportstädte an den Plan anhängt. Ein Unterschied zeigt sich erst, wenn keine Exportstadt mehr existiert. Dann wäre die Strategie des MICONICS-10-STRIPS-Planers, eine MFVS für das SCC zu generieren, die MFVS-Knoten an den Plan anzufügen, zu löschen und dann ein weiteres Mal an den Plan anzufügen.
Im Fall eines zweistufig hierarchisch-vollständigen Wegenetzgraphen sind dann aber noch immer einige Fragen ungeklärt. In der MICONICS-10-STRIPS-Domäne sind alle Personen bereits an dem Stockwerk, das durch den MFVS-Knoten repräsentiert wird. Im Fall von LOGISTICS sind die Pakete aber noch gar nicht an diese Stelle geliefert, sondern unter Umständen über die Stadt verteilt. Bislang wurde aber nicht geklärt, wie genau der Liefergraph der Stadt in zwei Teile geteilt wird – schließlich können noch gar nicht alle Pakete an Bord des Flugzeuges sein. Eine Lösung wäre also, eine Aufteilung analog zu Helmerts Approximationsalgorithmus zu wählen, und beim ersten Besuch alle Pakete einzusammeln und mit dem Austeilen zu warten, bis das Flugzeug die Stadt das zweite Mal besucht. Dann könnte es aber sein, dass ein Teil der Pakete an Bord des Flugzeuges an Orte gebracht werden müssen, an denen auch ein Paket abgeholt werden muss. Also wäre es offensichtlich sinnvoller, diese gleich mit auszuteilen. Dann entsteht aber das zusätzliche Problem, dass auch Städte, die Teil des MFVS sind, miteinander verbunden sein können, weswegen es wichtig wird, in welcher Reihenfolge die MFVS-Knoten an die beiden Stellen im Plan eingefügt werden.
Leider kann dieser Ansatz nicht funktionieren. Der Grund dafür ist ein einfacher: Die wichtigste Annahme eines Ansatzes, der auf dem Minimum Feedback Vertex Set basiert, ist die, dass alle Knoten in einem optimalen Plan höchstens zweimal angefahren werden müssen. Dies gilt für LOGISTICS in allen Städten genauso wie für alle Planungsaufgaben der MICONICS-10-STRIPS-Domäne, es gilt aber nicht für den Graphen erster Hierarchiestufe in einem zweistufig hierarchisch-vollständigen Graphen, von welchem der Luftliefergraph eines globalen CCs ein Teil ist. Ein Beispiel dazu ist durch den folgenden Liefergraphen gegeben:

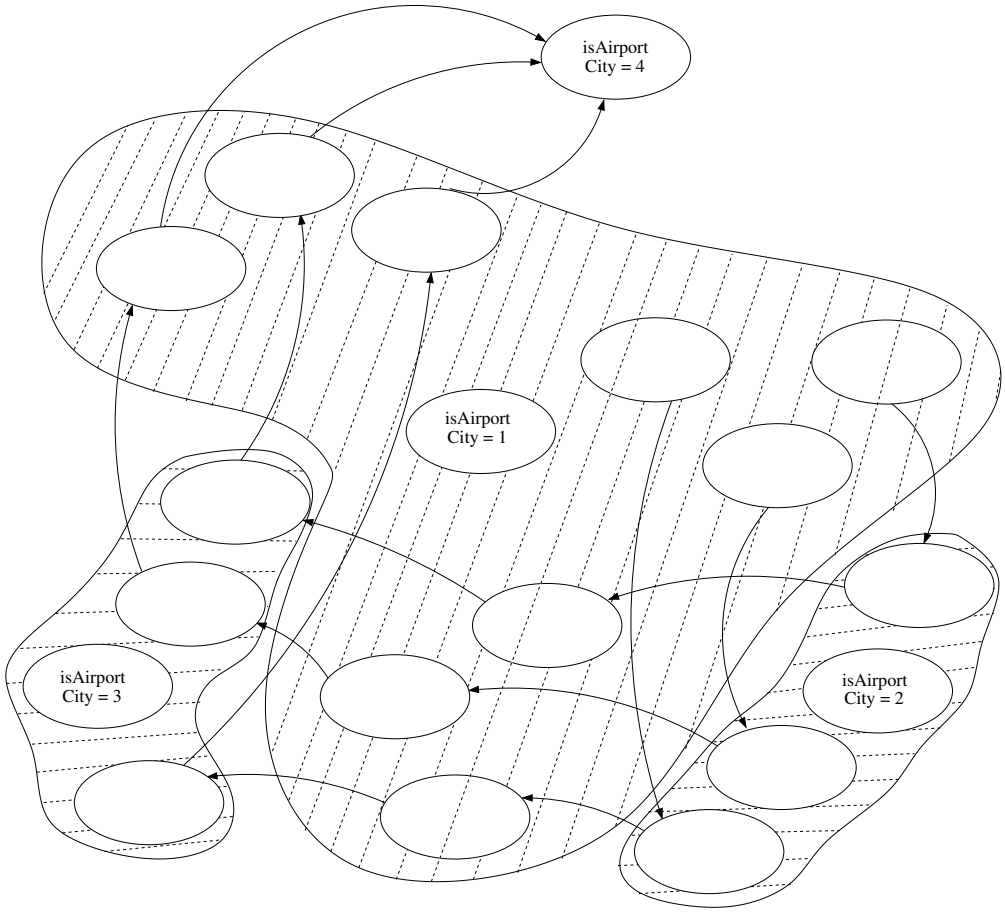

Abbildung 7.5: Liefergraph einer Logistics-Planungsaufgabe

In diesem Bild wurde für eine übersichtlichere Darstellung darauf verzichtet, jeden Knoten zu beschriften. Zudem sind keine Lastwagen und Flugzeuge eingezeichnet, es sei für jede Stadt sowie den Luftnetzgraphen aber angenommen, dass sich je ein Fahrzeug auf einem nicht abgebildeten, zusätzlichen Knoten befinde. Eine Möglichkeit, mit dem MFVS-Ansatz einen Plan für diesen Graphen zu generieren, ist die Folgende:

- Das Flugzeug fliegt den Weg → FH1 → FH2 → FH3 → FH1 → FH4 (5)
- In Stadt 1 (S1) werden alle Pakete eingesammelt und zu Flughafen 1 (FH1) gebracht (10)
- In S2 und S3 werden alle Pakete vom FH abgeholt, ausgeteilt und alle für andere Städte bestimmten eingesammelt und zum FH gebracht ($2 * 5 = 10$)
- Dann werden alle Pakete in S1 wieder ausgeteilt (6)
- In S4 müssen die Pakete nur abgelegt werden (0)

Die Gesamtbewegungskosten betragen in diesem Plan 31. Es existiert nur eine andere Möglichkeit, diese Logistics-Planungsaufgabe mit dem MFVS-Ansatz zu lösen:

- Das Flugzeug fliegt den Weg → FH1 → FH2 → FH3 → FH1 → FH4 (5)
- In S1 werden alle Pakete für S2 und S3 eingesammelt und zu FH1 gebracht (7)
- In S2 und S3 werden alle Pakete vom FH abgeholt, ausgeteilt und alle für andere Städte bestimmten eingesammelt und zum FH gebracht ($2 * 5 = 10$)

- Dann werden alle Pakete in S1 wieder ausgeteilt, und gleichzeitig die Pakete für S4 eingesammelt (7)
- In S4 müssen die Pakete nur abgelegt werden (0)

Dass dies die beiden einzige Möglichkeiten sind kann folgendermaßen gezeigt werden: Das MFVS des zu dieser Planungsaufgabe gehörenden Luftliefergraphen enthält mit Stadt 1 nur einen Knoten. Damit ist die Flugroute des Flugzeugs wie in beiden beschriebenen Plänen festgelegt. Dadurch müssen alle Pakete für die Städte 2 und 3 bereits beim ersten Besuch des Flugzeugs eingesammelt werden, da beide Städte nach dem zweiten Besuch nicht noch einmal angeflogen werden. Es bleibt also lediglich die Möglichkeit, die für Stadt 4 bestimmten Pakete schon beim ersten oder erst beim zweiten Besuch einzusammeln. Dies entspricht exakt den beiden gegebenen Plänen. Die Gesamtkosten des zweiten Planes betragen im Übrigen 29. Ein Plan der diese Planungsaufgabe mit 28 Bewegungaktionen und damit weniger als in beiden bislang vorgestellten löst, ist der Folgende:

- Das Flugzeug fliegt den Weg → FH1 → FH2 → FH1 → FH3 → FH1 → FH4 (6)
- In S1 werden nur die Pakete für S2 eingesammelt (4)
- In S2 werden alle Pakete vom FH abgeholt, ausgeteilt und alle für andere Städte bestimmten eingesammelt und zum FH gebracht (5)
- Dann werden in S1 die Pakete für S3 eingesammelt, und gleichzeitig die aus S2 ausgeteilt (4)
- In S3 werden alle Pakete vom FH abgeholt, ausgeteilt und alle für andere Städte bestimmten eingesammelt und zum FH gebracht (5)
- Dann werden in S1 die Pakete für S4 eingesammelt, und gleichzeitig die aus S3 ausgeteilt (4)
- In S4 müssen die Pakete nur abgelegt werden (0)

Damit konnte die Annahme widerlegt werden, dass jeder Knoten auch in hierarchisch-vollständigen Graphen höchstens zweimal angefahren werden muss. Tatsächlich kann dieser Graph sogar so erweitert werden, dass der Flughafen der mittleren Stadt beliebig oft angeflogen werden muss, indem einfach weitere Städte wie die Städte 2 und 3 mit analogen Verbindungen zu zusätzlichen Knoten in Stadt 1 eingeführt werden.

Damit kann nicht länger eine eindeutige Strategie angegeben werden, wie in solchen Fällen ein optimaler Plan generiert wird, weswegen eine Suche über dem Zustandsraum für Zykel im Luftliefergraphen mit einem A*-Planer implementiert wurde. Dabei sei angemerkt, dass auch Zykel existieren, die auch ohne eine solche Suche geplant werden können: Besteht ein solcher nur aus Städten, in welchen in jeder Stadt entweder alle ankommenden Pakete den Flughafen zum Ziel haben oder aber alle ausgehenden auf dem Flughafen starten, kann der Liefergraph der Stadt optimal in zwei Phasen geplant werden. Es gibt noch einige weitere Fälle, in welchen in der implementierten Version des Planers darauf verzichtet wird, eine Zustandsraumsuche zu starten. Auf eine ausführliche Beschreibung dieser Spezialfälle wird aber verzichtet.
Auch die Funktionsweise eines A*-Planers sei nicht detailliert vorgestellt, sondern auf [RN95] verwiesen. Ein kurzer Überblick sei aber dennoch gegeben: A* ist eine Methode, mit der versucht wird, gute Pfade in einem Suchraum zu einem Zielknoten mithilfe einer Heuristik zu erkennen und so nicht den gesamten Raum durchsuchen zu müssen. Mithilfe von Heuristiken wird dabei ein Maß dafür berechnet, wie weit entfernt das Ziel von einem beliebigen Knoten noch ist. Viele auf dem IPC erfolgreiche, domänenunabhängige Planer bauen vollständig auf einer solchen Suchstrategie auf. Erwähnt seien an dieser Stelle Hoffmanns Fast-Forward [HN01] sowie Helmerts Fast-Downward [Hel06]. Der Unterschied, der zwischen optimaler und suboptimaler Planung in Bezug auf heuristische Suche besteht, liegt insbesondere in der Wahl möglicher Heuristiken: Während suboptimale Planer beliebige Heuristiken verwenden können, muss die Heuristik

eines optimalen Planers *zulässig* sein, was dann der Fall ist, wenn die Kosten zum Ziel niemals überschätzt werden.
Dies ist in Bezug auf Routenplanung eine große Einschränkung: Es existieren zwar viele Algorithmen, mit deren Hilfe ein MFVS abgeschätzt werden kann, allerdings ist allen gemein, dass diese die Größe und daraus resultierend auch die Planlänge in der Logistics-Domäne überschätzen. Daher muss in der gewählten Heuristik entweder komplett darauf verzichtet werden, eine sinnvolle Abschätzung des MFVS zu generieren, oder dieses muss in jedem Schritt korrekt berechnet werden. Da die Leistungsfähigkeit des Miconics-10-Strips-Planers sich als sehr gut herausstellte, wurde eine Heuristik implementiert, die das MFVS wenn nötig korrekt berechnet. Die gewählte Heuristik entspricht genau der bereits beschriebenen separaten Betrachtung der Liefergraphen aller Städte und des Luftliefergraphen, es wird also für jede Stadt angenommen, sie könnte so geplant werden, dass bereits alle ankommenden Pakete am Flughafen sind wenn die Pakete, die die Stadt verlassen abgeholt werden müssen. Die Zulässigkeit dieser Heuristik sollte dabei offensichtlich sein: Da für jede Stadt und auch den Luftliefergraphen der Optimalfall angenommen wird, und diese unabhängig von allen anderen Liefergraphen geplant werden, können Kosten niemals überschätzt werden. Für die Beispielsplanungaufgabe würden also die folgenden Heuristikwerte für den Initialzustand berechnet:

- Für den Luftliefergraphen: 5
- Für S1: 10
- Für S2 und S3: je 5
- Für S4: 0

Es ergibt sich also eine insgesamt eine Abschätzung der Bewegungsoperatoren von 25, was für einen Spezialfall wie das Beispiel sehr nahe am Optimum liegt. Die Heuristik wird also für jeden Zustand folgendermaßen berechnet:

Algorithmus: ***Heuristik des A*-Moduls des* Logistics-*Planers***

Input: *Ein globales CC cc sowie eine* Logistics-*Planungsaufgabe* $\mathcal{T}$

Output: *Ein zulässige Abschätzung der Bewegungskosten h.*

Algorithmus:
```
globalCC2SubGraphs(cc,cc^a,CC^c)
V = getStartPos(cc^a,T)
h = solveMic10(cc^a,V,T).size()
for each cc_i^c ∈ CC^c do
    V = getStartPos(cc_i^c,T)
    h+ = solveMic10(cc^a,v).size()
return h
```

Damit bleibt lediglich die Frage, welche Informationen dem A*-Planer übergeben werden. In der Implementierung wurde darauf geachtet, diesen nur mit relevanten Operatoren und Objekten zu instantiieren.

- Der A*-Planer wird immer nur für SCCs des Luftliefergraphen gestartet. Damit wird jeder Zykel separat geplant, was die Planung erheblich beschleunigt.
- Es wird darauf geachtet, so wenige Fahrzeuge wie möglich zu instantiieren. Der einzige Fall, in welchem für einen Stadt (den Luftliefergraphen) ein Lastwagen (Flugzeug) nicht eindeutig bestimmt werden kann, tritt dann ein, wenn mehrere Lastwagen (Flugzeuge) in einem Zykel des Liefergraphs der Stadt (des Luftliefergraphs) starten.

- Alle pickUp- und drop-Aktionen sind als Axiome implementiert. Diese entsprechen Zustandsübergängen, die zu bestimmten Bedingungen eintreten und dann ohne Kosten automatisch ausgeführt werden. Dadurch wird der Planer nicht zusätzlich belastet, da er keine Ladeoperatoren planen muss..

- Zusätzlich werden nur diejenigen Bewegungsoperatoren instantiiert, die auch angewendet werden können. Insbesondere werden dadurch Teile der einzelnen Liefergraphen vorgeplant, also sicher in einem optimalen Plan enthaltene Planfragmente als einzige Option übergeben.

Der letzte Punkt dieser Aufzählung kann durch das folgende Beispiel verdeutlicht werden: Angenommen, im Liefergraphen einer Stadt existiert eine Aneinanderreihung von Orten, also eine Kette, die mit einem Exportknoten beginnt, gefolgt von einer Menge von Knoten, die nur eine eingehende Kante und auch nur eine ausgehende Kante haben. Für diese Knoten ist sicher, dass die Einschränkung, diese nur in der beschriebenen Reihenfolge abzufahren, niemals zu nicht-optimalen Plänen führen kann. Daher wurde für jeden der Knoten nur ein Bewegungsoperator instantiiert, der auf den Nachfolgerknoten führt. Lediglich der Startknoten dieser Kette kann von jedem beliebigen Knoten der Stadt erreicht werden. Damit sind die wichtigsten Eigenschaften des verwendeten A*-Planers beschrieben, der durch die folgende Methode gestartet werde und den ermittelten, optimalen Plan zurückgebe:

$$\pi = \texttt{planWithA*}(scc, V, \mathcal{T})$$

Damit kann auch der gesamte Algorithmus, der für beliebige Logistics-Planungsaufgaben einen optimalen Plan liefert, angegeben werden.

Algorithmus: ***Optimale Plangenerierung für* Logistics-*Planungsaufgaben***

Input: *Eine instantiierte* Logistics-*Planungsaufgabe* $\mathcal{T}$.

Output: *Ein optimaler Plan* $\pi^\star$.

Algorithmus:

```
01:  D^Log = createLogDelGraph(T)
02:  createCCs(D^Log,CC^l+,CC^g+,CC^g−,CC^l−,CC^m)
03:  for each cc ∈ CC^l+ do
04:    V = getStartPos(cc,T)
05:    π.add(solveM10(cc,V,T))
06:  for each cc ∈ CC^g+ do
07:    globalCC2SubGraphs(cc^g,cc^a,CC^c)
08:    airDelGraph2SCCGraph(cc^a,scc^a)
09:    v^a = solveStartPos(scc^a,T)
10:    p^a = getPos(v^a,T)
11:    while scc^a is not empty do
12:      SCC^ex = getExportNodes(scc^a)
13:      for each scc^ex ∈ SCC^ex do
14:        if scc^ex is a city-cycle
15:          π.add(planWithA*(scc^ex,p^a,T))
16:        else
17:          π.add(move(v^a, p^a, scc^ex)
18:          π.add(solveM10(scc^ex,p^a,T))
19:        deleteSCC(scc^a,sccex)
```

```
20:  v^a = getRandomPlane(T)
21:  p^a = getPos(v^a,T)
22:  for each cc ∈ CC^g- do
23:    globalCC2SubGraphs(cc^g,cc^a,CC^c)
24:    airDelGraph2SCCGraph(cc^a,scc^a)
25:    while scc^a is not empty do
26:      SCC^ex = getExportNodes(scc^a)
27:      for each scc^ex ∈ SCC^ex do
28:        if scc^ex.size() ≠ 1
28:          π.add(planWithA*(scc^ex,p^a,T))
30:        else
31:          π.add(move(v^a, p^a, scc^ex)
32:          π.add(solveM10(scc^ex,p^a,T))
33:        deleteSCC(scc^a,sccex)
34:  for each cc ∈ CC^l- do
35:    v = getRandomTruck(cc,T)
36:    π.add(solveM10(cc,v,T))
37:  return π
```

In dieser Form ist der Algorithmus im Vergleich zur tatsächlichen Implementierung erstens noch stark vereinfacht, es sind also einige, in diesem Kapitel bereits angesprochene Verfeinerung nicht enthalten, und zweitens ist eine Stelle so auch nicht ganz korrekt: Tatsächlich ist die in Zeile `09` gegebene Methode `solveStartPos` an dieser Stelle noch nicht zu berechnen. Da das Fahrzeugproblem aber ausreichend diskutiert wurde, und eine genau Angabe sehr viele Fallunterscheidungen zur Folge gehabt hätte, wurde darauf verzichtet wurde, diese Ungenauigkeit zu beheben. Sonst kann der Planer zusammengefasst folgendermaßen beschrieben werden:

- Die Zeile `01` und `02` erstellen den beschriebenen Logistics-Liefergraphen sowie die Zusammenhangskomponenten.
- In den Zeilen `03`–`05` werden die lokalen CCs, die einen Lastwagen beinhalten mithilfe des Miconics-10-Strips-Planers wie beschrieben geplant. Dabei sei an dieser Stelle angemerkt, dass die Funktion `add()`, die auf einen Plan angewendet wird, immer auch die im vorigen Kapitel beschriebenen Markier- bzw. Löschmethoden beinhalte und so der Liefergraph immer auf dem aktuellen Stand ist.
- In den Zeilen `06`–`19` wird die Planung globaler CCs mit enthaltenem Flugzeug beschrieben:
  - Für jedes dieser CCs wird in Zeile `07` der Liefergraph in Städte und den Luftliefergraphen aufgeteilt.
  - In Zeile `08` wird der Luftliefergraph dann in SCCs aufgeteilt.
  - Die Zeilen `11`–`19` planen dann die einzelnen SCCs: Es werden nach und nach SCCs ermittelt, die keine eingehenden Kanten haben (`12`). Bestehen diese aus mehreren Knoten (`14`), sind sie ein Zykel aus Städten, der durch den A*-Planer gelöst werden muss (`15`). Sind sie ein einzelner Knoten (`16`), kann das Flugzeug an diesen fliegen (`17`) und die Stadt wird durch den Miconics-10-Strips-Planer geplant (`18`). Damit ist das SCC abgearbeitet und kann gelöscht werden (`19`).
- In den Zeilen `20` und `21` wird ein zufälliges Flugzeug gewählt und dessen Position bestimmt.
- In den Zeilen `22`–`33` wird die Planung globaler CCs ohne enthaltenes Flugzeug analog zur Planung globaler CCs mit Flugzeug beschrieben. Der einzige Unterschied besteht in dem zufällig gewählten Flugzeug.

- Die Zeilen 34–36 beschreiben die Planung lokaler CCs ohne Lastwagen. Auch hier besteht der einzige Unterschied zu lokalen CCs mit Lastwagen darin, dass ein zufälliger, sich in der Stadt befindende, gewählt wird.

Damit wurde ein Algorithmus vorgestellt, der optimale Pläne für beliebige Logistics-Planungsaufgaben liefert. Wie durch das gemeinsame Subproblem erwartet werden konnte, sind große Teile von Planungsaufgaben mit derselben Methode lösbar, die im vorigen Kapitel für die Miconics-10-Strips-Domäne entwickelt wurde. Dennoch existiert durch die besondere Form des Wegenetzgraphen eine Art von Teilproblem, das nicht auf dieselbe Art geplant werden kann, nämlich Zykel im Graphen erster Hierarchiestufe des zweistufig hierarchisch-vollständigen Liefergraphen. In diesem Fall konnte keine Strategie, die auf eine Zustandsraumsuche verzichtet, gefunden werden, weswegen der Planer um ein A*-Planungsmodul mit einer domänenspezifischen, zulässigen Heuristik erweitert wurde. Damit können im nächsten Abschnitt Ergebnisse der Implementierung beschrieben und mit denen von verschiedenen domänenunabhängigen Planern verglichen werden.

7.3 Ergebnisse

Der beschrieben Planer wurde, mit einigen zusätzlichen Verfeinerungen, implementiert und an den `Logistics`-Planungsaufgaben des IPC 2 getestet. Auch Helmert et. al. vergleichen in [HHH07] den darin beschriebenen LFPA-Planer mit anderen domänenunabhängigen, optimalen Planern, weswegen die dort gegebenen Ergebnisse zum Vergleich des domänenspezifischen Planers herangezogen werden können. Allerdings wurden von den Autoren des LFPA-Planers nicht alle mir vorliegenden Planungsaufgaben gelöst.

Planer / Planungsaufgabe	Logistics-Planer	LFPA	PDB
4-0	0,005	0,10	3,21
4-1	0,005	0,09	5,48
5-0	0,017	0,87	16,75
5-1	0,008	0,88	4,58
6-0	0,044	3,65	16,48
6-1	0,005	3,85	3,16
7-0	0,126	24,56	91,12
7-1	0,081	26,84	156,28
8-0	0,027	37,09	79,49
8-1	0,049	40,77	160,02
9-0	0,043	55,47	127,52
9-1	0,004	53,42	92,85
10-0	0,051	117,46	497,51
10-1	0,054	129,97	405,24
11-0	0,036	129,82	377,28
11-1	0,233	284,91	
12-0	0,181	185,90	545,76
12-1	0,067	221,51	
13-0			
13-1	0,085		

Planer / Planungsaufgabe	Logistics-Planer	LFPA	PDB
14-0	0,051		
14-1	0,121		
15-0			
15-1	0,100		

Abbildung 7.6: Laufzeit (in Sek.) verschiedener Planer auf den Logistics-Planungsaufgaben des IPC 2

Wie zu erwarten ist der domänenspezifische, für diese Arbeit entwickelte Logistics-Planer den domänenunabhängigen Planern in allen Planungsaufgaben überlegen. Während einfache Planungsaufgaben noch in vergleichbar schneller Zeit gelöst werden, wird der Unterschied bei komplexeren Aufgaben signifikant. So wird die Planungsaufgabe 11-0, für welche der Logistics-Planer die längste Zeit unter allen Planungsaufgaben benötigt, von diesem noch immer in etwas mehr als 0,2 Sekunden optimal geplant, während der LFPA-Planer beinahe drei Minuten benötigt und der PDB-Planer überhaupt keinen Plan in der eingeräumten Zeit liefert. Demnach kann auch festgestellt werden, dass alle Planungsaufgaben in weit unter einer Sekunde gelöst wurden – von den beiden Fällen abgesehen, in welchen auch der domänenspezifische Planer keine Lösung generieren konnte.
Leider können die Gründe, warum der Logistics-Planer bei diesen beiden Aufgaben versagte, nicht restlos aufgeklärt werden. Sicher ist, dass es sich bei diesen beiden Instanzen um diejenigen handelt, die die größten Städtezykel beinhalten: In beiden Planungsaufgaben existiert ein solcher aus fünf Städten bestehender Zykel. Da aber die Planungsaufgaben 11-1, 12-0, 12-1 und 14-1 Zykel aus vier Städten enthalten, kann der Unterschied zwischen einem in wenigen Zehntelsekunden generierten Plan auf einem Zykel aus vier Städten und dem vollständigen Fehlen einer Lösung für die Aufgaben 13-0 und 15-0 nicht auf diese Eigenschaft zurückgeführt werden. Es gibt aber noch einen weiteren, vermutlich verantwortlichen Grund: Die beiden nicht gelösten Planungsaufgaben sind die einzigen, in welchen der A*-Planer mit mehr als einem Flugzeug instantiiert werden muss. Zwar ist die Existenz mehrerer Fahrzeuge eine nicht zu unterschätzende Erschwernis, es bleiben aber dennoch Zweifel, ob dies tatsächlich der Grund für den immensen Leistungsunterschied ausmacht.
Eine Möglichkeit, die aus Zeitmangel leider nicht mehr implementiert werden konnte, wäre eine Modifikation des Planers derart gewesen, dass für jedes Flugzeug getrennt ein optimaler Plan berechnet wird, und letztendlich der kürzere verwendet wird. Ich bin überzeugt, dass dann auch diese beiden Planungsaufgaben in vergleichbar guter Laufzeit gelöst worden wären.

Abschließend kann also festgestellt werden, dass domänenspezifische Planer domänenunabhängigen wie erwartet weit überlegen sind. Sowohl in der Miconics-10-Strips-Domäne als auch in der Logistics-Domäne konnten hervorragende Ergebnisse mit Laufzeiten von weit unter einer Sekunde erreicht werden, und auch die komplexesten Planungsaufgaben wurden in beiden Fällen weitgehend gelöst. Festzuhalten ist insbesondere, dass nicht nur domänenspezifische Strategien, die keine Zustandsraumsuche benötigen, geeignet sind, Planungsaufgaben derselben Domäne in sehr kurzer Zeit zu lösen, sondern auch ein herkömmlicher A*-Planer, der eine domänenspezifische Heuristik verwendet, Ergebnisse in weit unter einer Sekunde liefern kann. Es scheint also tatsächlich lohnenswert, heuristische, domänenunabhängige Planer zu entwickeln, die domänenspezifische Erkenntnisse in der Berechnung der Heuristik einbeziehen.

8 Zusammenfassung und Ausblick

Das Ziel dieser Diplomarbeit war es, domänenspezifische, optimale Planer für Domänen der Transport- und Routenplanungsfamilie zu entwickeln. Für die beiden Domänen Miconics-10-Strips sowie Logistics wurde je ein solcher beschrieben und mit, nicht nur im Vergleich zu anderen Planungssystemen, hervorragenden Ergebnissen implementiert. Die Laufzeiten des Miconics-10-Strips-Planers, die für alle Planungaufgaben weniger als eine Zehntelsekunde betrugen, verdeutlichen dies eindrucksvoll. Die genaue Studie dieser Domäne führte zur Entwicklung eines universellen Pfadplanungsalgorithmus für vollständige Wegenetzgraphen, der aufgrund der geforderten Optimalität von Plänen aber ohne eine Komponente, die das NP-vollständige Minimum Feedback Vertex Set berechnet, nicht auskommen kann. So wurde bereits bei einer vergleichsweise einfachen Domäne wie Miconics-10-Strips deutlich, wie groß die Unterschiede zwischen suboptimaler und optimaler Planung sind: Suboptimale Pläne mit maximal $\frac{7}{6}$-facher Länge des optimalen Planes können im Gegensatz dazu polynomiell durch einen Approximationsalgorithmus berechnet werden.

Auf Grundlage dieser Erkenntnisse wurde dann versucht, einen Algorithmus für die Logistics-Domäne zu entwickeln. Dabei zeigte sich aber ein unerwartet komplexes Teilproblem: Es konnte gezeigt werden, dass die Verwendung der in der Miconic-10-Strips-Domäne entwickelten Methode für vollständige Wegenetzgraphen, die Pfadplanung durch einen MFVS-basierten Ansatz zu lösen, bereits in der leicht veränderten Form zweistufig hierarchisch-vollständiger Wegenetzgraphen nicht länger möglich ist. Es zeigte sich nämlich, dass es nicht optimal sein muss, dass jeder Knoten einer Logistics-Planungsaufgabe höchstens zweimal von einem Fahrzeug besucht wird, was eine Eigenschaft ist, die ein tieferes Verständnis für optimale Pfadplanung sowohl mit mehreren Fahrzeugtypen als auch auf hierarchisch verbundenen Wegenetzgraphen ermöglicht. Dadurch wurde es in der Logistics-Domäne nötig, Teile von Planungsaufgaben wie in der domänenunabhängigen Planung durch eine Zustandsraumsuche zu lösen. Dafür wurde eine zulässige, domänenspezifische Heuristik entworfen. Die resultierenden Laufzeiten zeigen deutlich, dass auch die Zustandsraumsuche erheblich von der Verwendung domänenspezifischen Wissens profitiert.

Zusätzlich wurde in dieser Arbeit großer Wert darauf gelegt, eine der auf dem Internationalen Planungswettbewerb verwendeten Beschreibungssprache PDDL nahe Formalisierung zu entwickeln. Dabei wurde insbesondere festgestellt, dass eine große Menge von Informationen, die in dieser Arbeit durch Planungsaufgabenbeschränkungen repräsentiert werden, Planungssystemen durch PDDL nicht zugänglich gemacht werden. Ausgehend von einem Modell, in welchem diese Beschränkungen existieren, wurden daraufhin Methoden entwickelt, effiziente Zustandsbeschreibungen inklusive objektwertiger Funktionen und verschiedenen Arten von Graphen alleine aus einer Domänenbeschreibung zu generieren, ohne dass dafür eine konkrete Planungsaufgabe angegeben werden muss. Auf Basis dieser Definitionen wurden auch Domänenfamilien formalisiert, welchen all diejenigen Domänen angehören, die dasselbe Subproblem enthalten. Es wurde zudem beschrieben, wie aus einer Domänenfamilienbeschreibung mithilfe von Initialzustandsbeschränkungen Spezialfälle dieses gemeinsamen Subproblems konstruiert werden können. Letzlich wurden auch Möglichkeiten skizziert, inwiefern beliebige Domänen automatisch einer Domänenfamilie zugeordnet werden können.

Damit könnte diese Arbeit durchaus als Grundlage für die Entwicklung domänenunabhängiger Planer verwendet werden. Eine Möglichkeit wäre ein Planungssystem, welches je nach gegebener Domäne domänenspezifisches Wissen ausnutzt um die verwendete Heuristik dynamisch anzupassen. Eine Voraussetzung dafür

scheint aber zu sein, vorhandene Beschreibungssprachen so zu erweitern, dass Planungssystemen nicht länger wichtige Informationen vorenthalten werden. Soll die Leistungsfähigkeit automatischer Planer weiter voranschreiten, wird es zwingend nötig sein, neue Wege einzuschlagen. Sowohl spezifischere, informationsreichere Domänenbeschreibungen als auch die Verwendung domänenspezifischen Wissens können die Grundlage für weiterer Fortschritte sein.

Literaturverzeichnis

[APMS+99] Ausiello, Giorgio, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi und V. Kann: *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Secaucus, NJ, USA, 1999.

[Bal98] Balzert, Helmut: *Lehrbuch der Software-Technik, Software Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1998.

[BK00] Bacchus, Fahiem und Froduald Kabanza: *Using Temporal Logics to Express Search Control Knowledge for Planning*. 116(1–2):123–191, 2000.

[Bon01] Bonet, Blai: *Planning as heuristic search*. Artificial Intelligence, 129:5–33, 2001.

[CLRS01] Cormen, Thomas, Charles Leiserson, Ronald Rivest und Clifford Stein: *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[DK01] Doherty, Patrick und Jonas Kvarnström: *TALplanner: A Temporal Logic Based Planner*. AI Magazine, (22), 2001.

[EBN08] Eyerich, Patrick, Michael Brenner und Bernhard Nebel: *On the Complexity of Planning Operator Subsumption*. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning*, S. 518–527, 2008.

[EH04a] Edelkamp, Stefan und Jörg Hoffmann: *PDDL2.2: The Language for the Classical Part of IPC-4*. IPC-4 Booklet, S. 2–6, 2004.

[EH04b] Edelkamp, Stefan und Jörg Hoffmann: *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition*. Tech. Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2004.

[FL03] Fox, Maria und Derek Long: *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. Journal of Artificial Intelligence Research, 20:61–124, 2003.

[FL06] Fox, Maria und Derek Long: *Modelling Mixed Discrete-Continuous Domains for Planning*. Journal of Artificial Intelligence Research, 27:235–297, 2006.

[FN71] Fikes, Richard und Nils J. Nilsson: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence, 2(3/4):189–208, 1971.

[GF92] Genesereth, Michael und Richard Fikes: *Knowledge Interchange Format*. Tech. Report Logic-92-1, Stanford Logic Group, 1992.

[GJ79] Garey, Michael und David Johnson: *Computer and Intractability – A Guide to the Theory of NP-Cpleteness*. Freeman, 1979.

[GL05] Gerevini, Alfonso und Derek Long: *Plan Constraints and Preferences in PDDL3*. Tech. Report R. T. 2005-08-47, Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia, 2005.

[GL06] Gerevini, Alfonso und Derek Long: *Preferences and Soft Constraints in PDDL3*. Proceedings of the ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning, S. 46–54, 2006.

[HE05] Hoffmann, Jörg und Stefan Edelkamp: *The Deterministic Part of IPC-4: An Overview*. Journal of Artificial Intelligence Research, 24:519–579, 2005.

[Hel02] Helmert, Malte: *Decidability and Undecidability Results for Planning with Numerical State Variables*. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling*, S. 302–312, 2002.

[Hel03] Helmert, Malte: *Complexity Results for Standard Benchmark Domains in Planning*. 143(2):219–262, 2003.

[Hel06] Helmert, Malte: *The Fast Downward Planning System*. Journal of Artificial Intelligence Research, 26:191–246, 2006.

[Hel08] Helmert, Malte: *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*. Springer, Berlin, Germany, 2008.

[HET+06] Hoffmann, Jörg, Stefan Edelkamp, Sylvie Thiébaux, Roman Englert, Frederico dos Santos Liporace und Sebastian Trüg: *Engineering Benchmarks for Planning: The Domains Used in the Deterministic Part of IPC-4*. Journal of Artificial Intelligence Research, 26:453–541, 2006.

[HGS07] Hoffmann, Jörg, Carla Gomes und Bart Selman: *Structure and Problem Hardness: Goal Asymmetry and DPLL Proofs in SAT-based Planning*. lmcs, 3(1:6), 2007.

[HHH07] Helmert, Malte, Patrik Haslum und Jörg Hoffmann: *Flexible Abstraction Heuristics for Optimal Sequential Planning*. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, S. 176–183, 2007.

[HN01] Hoffmann, Jörg und Bernhard Nebel: *The FF Planning System: Fast Plan Generation Through Heuristic Search*. Journal of Artificial Intelligence Research, 14:253–302, 2001.

[HR08] Helmert, Malte und Gabriele Röger: *How Good is Almost Perfect?* In *AAAI*, S. 944–949, 2008.

[HWHC06] Hsu, Chih-Wei, Benjamin W. Wah, Ruoyun Huang und Yixin Chen: *New Features in SGPlan for Handling Preferences and Constraints in PDDL3.0*. 2006.

[Knu64] Knuth, Donald E.: *Backus Normal Form vs. Backus Naur form*. Communications of the ACM, 7(12):735–736, 1964.

[KS92] Kautz, Henry und Bart Selman: *Planning as satisfiability*. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, S. 359–363, 1992.

[KS00] Koehler, Jana und Kilian Schuster: *Elevator Control as a Planning Problem*. In *Artificial Intelligence Planning Systems*, S. 331–338, 2000.

[LHS00] Loughry, J., J. I. Van Hemert und L. Schoofs: *Efficiently Enumerating the Subsets of a Set*, 2000.

[MGH+98] McDermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld und David Wilkins: *PDDL – The Planning Domain Definition Language*. Tech. Report CVC TR-98-003, Yale Center for Computational Vision and Control, 1998.

[RN95] Russell, Stuart und Peter Norvig: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[SW75] Smith, George und Robert Walford: *The Identification of a Minimal Feedback Vertex Set of a Directed Graph*. IEEE Transactions on Circuits and Systems, 22(1):9–14, 1975.