



Diplomarbeit

Planen mit Präferenzen

Ein Ansatz zur Lösung partieller Erfüllbarkeitsprobleme

Robert Feldmann

Juni 2005

Betreuer: Prof. Dr. Gerhard Brewka

Institut für Informatik
Fakultät für Mathematik und Informatik
Universität Leipzig

Inhaltsverzeichnis

Einleitung	1
1 Automatisches Planen	3
1.1 Klassische Planungsprobleme	3
1.2 Partielle Erfüllbarkeitsprobleme	5
1.3 Repräsentationssprachen für Planungsprobleme	8
1.3.1 Historische Entwicklung	8
1.3.2 STRIPS	8
1.3.3 PDDL2.1	11
1.4 Spezielle Lösungsansätze in der Literatur	17
1.4.1 <i>Integer Programming</i>	17
1.4.2 Heuristische Planer – AltAlt ^{PS} und Sapa ^{PS}	19
1.4.3 Unterscheidung vom Ansatz dieser Arbeit	21
2 Wissensbasen und Präferenzen	23
2.1 Präferenzausdrücke auf gewichteten Wissensbasen	23
2.1.1 Gewichtete Wissensbasen	23
2.1.2 Basis-Präferenzausdrücke	25
2.1.3 Komplexe Präferenzausdrücke	26
2.2 Alternative Darstellung von Präferenzausdrücken	28
2.2.1 Die Sprache LPD*	29
2.2.2 Analyse der Sprachelemente	33
2.2.3 Die Sprache TLPD	37
2.3 Vergleich beider Ansätze	39
3 Linearisierung und numerische Präferenzbeschreibungen	41
3.1 Linearisierung einer Quasiordnung	41
3.2 Linearisierung von TLPD	43
4 Übersetzung in PDDL2.1	49
4.1 Lösung partieller Erfüllbarkeitsprobleme	49
4.2 Übersetzung in PDDL2.1	52

4.2.1	Erweiterung von PDDL2.1	52
4.2.2	Generelle Überlegungen	54
4.2.3	Erläuterung der einzelnen Übersetzungsschritte	55
4.2.4	Ein Beispiel	62
4.2.5	Implementation	71
Zusammenfassung		75
A BNF Spezifikation von PDDL2.1		77
A.1	Domänenbeschreibung	77
A.2	Listen	78
A.3	Aktionen	79
A.4	Faktenbeschreibung	80
A.5	PSP Beschreibung	81
B Auszüge aus dem Quellcode		83
B.1	Modifikation des Planers <i>Metric-FF</i>	83
B.2	Das Perl-Skript	86
Literaturverzeichnis		89
Danksagung		93

Abbildungsverzeichnis

1.1	Schematischer Aufbau eines konzeptionellen Modells für das Planen	4
1.2	Auszug aus der STRIPS-Domänenbeschreibung der Satelliten-Domäne	9
1.3	Auszug aus einer STRIPS-Faktenbeschreibung der Satelliten-Domäne	10
1.4	Auszug aus der PDDL2.1-Domänenbeschreibung der Satelliten-Domäne.	12
2.1	Meta-Relationen zwischen den strikten Präferenzrelationen	27
2.2	Anwendungsbeispiel der Präferenzbeschreibungssprache LPD.	28
3.1	Hasse-Diagramme von Quasiordnung und linearisierter Quasiordnung	43
4.1	Schematischer Anstieg der Güte q_C mit der Zeit	74

Tabellenverzeichnis

2.1	Definition der Semantik der Sprache LPD*	29
3.1	Die Definition der Funktionen val und maxval	44

Einleitung

Viele Probleme, denen wir im täglichen Lebens begegnen, erweisen sich als Planungsprobleme. Angenommen Herr K. möchte Karten für einen Kinoabend kaufen. Ein Plan bestünde darin, zum Kino zu fahren, die Karten zu bezahlen und zurück nach Hause zu fahren. Wenn Herr K. jedoch kein Bargeld mehr übrig hat, könnte er stattdessen folgenden Ablauf von Aktionen wählen: zur Bank fahren, Geld abheben, von der Bank zum Kino fahren, die Karten bezahlen, zurück nach Hause fahren. Ein Planungsproblem zu lösen bedeutet, abhängig von der gegebenen Ausgangssituation eine oder mehrere Aktionsfolgen zu finden, die bei Ausführung zur Erfüllung gewünschter Ziele führen. Bestimmte Problemstellungen lassen sich durch Nachdenken bzw. reines Ausprobieren lösen. Mit steigender Komplexität des Planungsproblems wird dies jedoch immer schwieriger und zeitaufwändiger¹, da der Suchraum aller möglichen Aktionsfolgen zu groß wird. Als Beispiel sei die Verwaltung eines Warenlagers einer Spedition erwähnt. Mögliche Aktionen sind hier folgender Art: LKW X zur Verladerrampe von Lager L fahren, LKW X mit Gut Y beladen, LKW X von L zu Zielort Z fahren, LKW X von Gut Y entladen. Ziele können darin bestehen, diverse Güter an bestimmten Orten zu deponieren und dabei gewisse Nebenbedingungen einzuhalten. Falls hinreichend viele Objekte (Güter, LKW, Orte usw.) beteiligt sind, empfiehlt es sich, den Planungsprozess zu automatisieren. Dazu wird die Beschreibung von Aktionen, Zuständen und Zielen formalisiert und durch syntaktische Konstrukte einer Repräsentationssprache für Planungsprobleme ausgedrückt. Anschließend kann ein automatisches Planungssystem mit der Suche nach einer Lösung des Problems beginnen. In Abschnitt 1.1 werden wir obige Begriffe eines Planungsproblems bzw. einer Lösung desselben im Rahmen des klassischen Planens präzisieren. Dabei wird ein in [1] vorgestelltes konzeptuelles Modell als Ausgangspunkt verwendet. Etwas später, in Abschnitt 1.3, werden Repräsentationssprachen für Planungsprobleme vorgestellt. Insbesondere wird auf die Sprachen STRIPS [2] und PDDL2.1 [3] eingegangen. Spezifische Planungsprobleme werden oft in Problemklassen, auch Planungsdomänen genannt, eingeteilt. Dies geschieht meist mit dem Ziel, verschiedene Planungssysteme hinsichtlich ihrer Leistungsfähigkeit zu testen und zu vergleichen. Obiges Speditionsbeispiel stellt eine Variation der sogenannten Depotdomäne dar.

Einen ebenfalls wichtigen Einfluss auf alltägliche Entscheidungen, die wir treffen, besitzen Präferenzen. Üblicherweise entscheiden wir so, dass vorgegebene Präferenzen bestmöglich erfüllt werden. Dies trifft insbesondere dann zu, wenn wir zwischen mehreren, sich einander ausschließenden, Alternativen wählen müssen. Bei der Behandlung von Präferenzen durch ein automatisches System sind zwei Teilaspekte zu berücksichtigen. Zum einen müssen Präferenzen eindeutig definiert werden, d.h. dass eine Präferenzbeschreibung benötigt wird. Zum anderen muss das automatische System dieses zusätzliche Wissen geeignet

¹Diese Arbeit verwendet, so gut es geht, die neue Rechtschreibung.

einsetzen. Die Behandlung von Präferenzen ist in vielen Gebieten der künstlichen Intelligenz von Bedeutung, z.B. in der logischen Programmierung [4] oder dem Schlussfolgern mit Defaults [5, 6]. Auch im Planungskontext werden Präferenzen (meist implizit) betrachtet. Hier sind jedoch in aller Regel Aktionsfolgen Gegenstand einer Bewertung und nicht die Zielmengen. Der Hauptgrund liegt darin, dass normale Planungsprobleme eine Lösung nach dem Alles-oder-Nichts Prinzip anstreben: Das Problem gilt genau dann als gelöst, wenn alle geforderten Ziele erreicht sind. So wird häufig versucht, ein Problem mit einer möglichst kurzen Zahl von Aktionen oder bezüglich einer anderen geeigneten Kostenfunktion über den Aktionen zu lösen. *Partiellen Erfüllbarkeitsproblemen* begegnet man immer dann, wenn eine Menge von Zielen gegeben ist, welche sich nicht alle gleichzeitig erfüllen lassen. Dies mag daran liegen, dass sich einzelne Ziele rein logisch ausschließen oder dass das gegebene Planungsproblem dies per se verhindert. So läßt es sich unter geeigneten Ausgangssituationen bei Problemen in der Depotdomäne nicht erreichen, dass sich ein spezielles Gut an zwei verschiedenen Orten zugleich befindet. Ein partielles Erfüllbarkeitsproblem läge beispielsweise vor, wenn Herr K. neben Kinokarten noch Theaterkarten und ein Geschenk erstehen möchte, seine zur Verfügung stehende Zeit es jedoch lediglich erlaubt, zwei dieser drei Vorhaben umzusetzen. Falls das Geschenk eine höhere Präferenz besitzt als Kino- oder Theaterkarten, welche ihrerseits Herrn K. gleich wichtig sind, so existieren genau zwei Lösungen des partiellen Erfüllbarkeitsproblems. Herr K. wird das Geschenk und entweder Kino- oder Theaterkarten kaufen. Das Augenmerk liegt in partiellen Erfüllbarkeitsproblemen darauf, eine Aktionsfolge zu finden, die eine möglichst optimale Menge von Zielen erfüllt. Optimalität bedeutet hier Maximalität bezüglich einer Präferenzrelation über der Menge aller Zielmengen, die durch eine Präferenzbeschreibung spezifiziert wird. Partielle Erfüllbarkeitsprobleme werden in Abschnitt 1.2 besprochen.

In dieser Arbeit wird ein Algorithmus zur Lösung von partiellen Erfüllbarkeitsproblemen vorgestellt (Abschnitt 4.1). Dieser Ansatz unterscheidet sich in folgenden Punkten von den bereits existierenden Planungssystemen für partielle Erfüllbarkeitsprobleme. Erstens stellen diese Planer Spezialsysteme dar, die ad-hoc zur Lösung von partiellen Erfüllbarkeitsproblemen konzipiert wurden. Sie können somit nicht von der schnellen Weiterentwicklung klassischer Planungssysteme direkt profitieren, sondern müssen Verbesserungen stets neu implementieren. In dieser Arbeit wird ein Ansatz vorgestellt, der direkt auf klassischen Planern aufbaut. Zweitens sind Präferenzen zwischen Zielmengen in den existierenden Planungssystemen oft durch einfache Nutzen-Kosten Abschätzungen definiert. Die Güte einer Aktionssequenz besteht in der Differenz aus Summe der Nutzen aller erreichten Ziele und der Summe der Kosten aller in der Sequenz enthaltenen Aktionen. In der vorliegenden Arbeit werden Präferenzbeschreibungen zunächst allgemein auf einer abstrakten Ebene untersucht (in den Kapiteln 2 und 3) und anschließend auf den Planungskontext angepasst (vgl. Abschnitt 4.2). Als Ergebnis wird eine Präferenzbeschreibungssprache als Erweiterung der Repräsentationssprache PDDL2.1 vorgeschlagen. Eine Vorstellung existierender Ansätze und ein Vergleich dieser Ansätze mit dem Algorithmus dieser Arbeit findet sich in Abschnitt 1.4.

Der vorgestellte Lösungsalgorithmus wurde schließlich in Form eines C-Programmes und eines Perl-Skriptes implementiert. Dies zeigt die prinzipielle Leistungsfähigkeit des Lösungsansatzes. Die Ausführungen dazu finden sich in Abschnitt 4.2. Teile des Quellcodes sind im Anhang angegeben. Dieser Arbeit liegt zudem eine CD mit dem vollständigen Quellcode der Implementation bei.

Kapitel 1

Automatisches Planen

1.1 Klassische Planungsprobleme

Nach [7] wird mit *Planen* die Aufgabe beschrieben, eine Sequenz von Aktionen zu finden, um ein Ziel zu erreichen. Diese Definition lässt sich mit der Einführung eines konzeptuellen Modells des Planens präzisieren. Wir werden dieses Modell kurz vorstellen und anschließend zur Formulierung des klassischen Planungsproblems heranziehen. Nach [1] besteht ein derartiges Modell aus drei Komponenten (siehe Abbildung 1.1):

- einem Transitionssystem, welches durch seine Übergangsfunktion $\tilde{\gamma} : S \times A \times E \rightarrow \mathcal{P}(S)$ beschrieben wird. Hierbei bezeichnet S eine Menge von Zuständen des Systems, A eine Menge von Aktionen und E eine Menge von Ereignissen.
- einer Steuereinheit (Kontroller), welche in Abhängigkeit vom gegebenen Plan und ihrem Wissen über den Zustand des Transitionssystems eine Aktion $a \in A$ aussucht und diese an das Transitionssystem weitergibt.
- einem Planer, der aus einer Beschreibung des Transitionssystems (Domänenbeschreibung) und einer Beschreibung der Probleminstanz (Ausgangszustand und Ziele) einen Plan erzeugt.

Die genaue Spezifikation von Plänen, Aktion, Ereignissen, Zielen und Zuständen hängt vom jeweils verwendeten Planungskontext ab. Wir werden uns nun mit dem *klassischen Planungsproblem* und einer dafür geeigneten Beschreibungssprache beschäftigen.

Im klassischen Planen werden folgende Einschränkungen des konzeptuellen Modells vereinbart:

- Der Zustandsraum S ist endlich.
- Das Wissen über das Transitionssystem ist vollständig. Insbesondere kennt die Steuereinheit den aktuellen Zustand.
- Das Transitionssystem ist deterministisch, d.h. der Wert der Übergangsfunktion ist eine einelementige Menge oder die leere Menge.

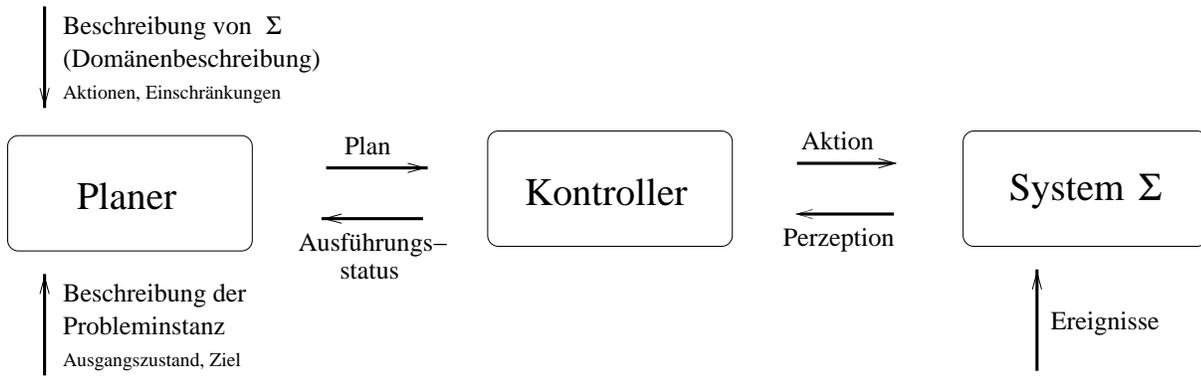


Abbildung 1.1: Schematischer Aufbau eines konzeptionellen Modells für das Planen

- Die Menge der Ereignisse ist leer, d.h. Zustandsveränderungen werden nur durch die Wirkung von Aktionen verursacht.
- Ziele werden durch Zustandsmengen S_G angegeben.
- Eine Lösung für ein Planungsproblem besteht aus einer linear geordneten, endlichen Sequenz von Aktionen.

Überdies besitzen Aktionen im Kontext des klassischen Planens keine zeitliche Ausdehnung. Dieser Forderung wird bereits implizit durch die Verwendung eines Transitionssystems entsprochen. Die Zeit wird in einem derartigen System implizit in Form diskreter Zeitschritte durch Zustandsübergänge modelliert. Um *durative*, d.h. zeitlich ausgedehnte, Aktionen zu beschreiben, ist das obige konzeptionelle Modell entsprechend anzupassen. Da das System deterministisch ist, wird im Weiteren folgende Notation der Übergangsfunktion verwendet. Eine Aktion a heißt anwendbar im Zustand s bzw. ein Tupel (s, a) heißt anwendbar, wenn $\tilde{\gamma}(s, a) \neq \emptyset$. Es wird nun üblicherweise folgende modifizierte Übergangsfunktion γ verwendet:

$$\gamma : S \times A \rightarrow S,$$

die auf allen anwendbaren Paaren (s, a) definiert ist als

$$\gamma(s, a) = s', \text{ falls } \tilde{\gamma}(s, a) = \{s'\}.$$

Ein klassisches Planungsproblem Γ und eine Lösung desselben werden nun wie folgt formal definiert.

Definition 1 (Klassisches Planungsproblem). Ein 5-Tupel $\Gamma = (S, A, \gamma, s_I, S_G)$ bestehend aus einer endlichen¹ Menge von Zuständen S , einer endlichen Menge von Aktionen A , einer Übergangsfunktion $\gamma : S \times A \rightarrow S$, einem Anfangszustand $s_I \in S$ und einer Beschreibung der Zielzustände $S_G \subseteq S$ heißt klassisches Planungsproblem.

Ein Zustand s' heißt erreichbar von Zustand s , falls $s' = s$ oder eine linear geordnete, endliche Sequenz von Aktionen $\langle a_1, \dots, a_n \rangle$ existiert, so dass gilt:

- (s_0, a_1) ist anwendbar, mit $s_0 = s$,

¹Soweit nicht anders erwähnt werden alle folgenden Mengen als endlich angenommen. Oft gelten die entsprechenden Sätze aber auch für den Fall abzählbar unendlicher Mengen.

- für alle $i \in \{1, \dots, n\}$ ist (s_{i-1}, a_i) anwendbar und $s_i = \gamma(s_{i-1}, a_i)$,
- $s' = s_n$.

Man kann die Übergangsfunktion erweitern zu einer Funktion $\gamma^* : S \times A^* \rightarrow S$, die einem Paar, bestehend aus einem Zustand und einer linear geordneten, endlichen Aktionssequenz, in kanonischer Weise wieder einen Zustand zuordnet. Sei nämlich $A_n = \langle a_1, \dots, a_n \rangle$ eine derartige nichtleere Sequenz, dann ist γ^* wie folgt rekursiv definiert:

$$\gamma^*(s, A_1) = \gamma(s, a_1), \quad \gamma^*(s, A_n) = \gamma(\gamma^*(s, A_{n-1}), a_n).$$

Definition 2 (Lösung des klassischen Planungsproblems). Sei ein Planungsproblem Γ gegeben. Es heie lösbar, falls ein Zustand $s_G \in S_G$ existiert, der vom Ausgangszustand s_I erreichbar ist. Eine zugehörige linear geordnete, endliche Sequenz von Aktionen $\langle a_1, \dots, a_n \rangle$ heißt Lösung des Planungsproblems oder kurz Plan².

Die Menge aller vom Ausgangszustand s_I erreichbaren Zustände definiert den Zustandsraum des Planungsproblems. Dieser kann als Graph aufgefasst werden, mit den erreichbaren Zuständen als Knoten und den Aktionen als Kanten. Das Auffinden eines Planes kann daher durch eine Pfadsuche in diesem Graphen erfolgen. Standardalgorithmen stellen Tiefensuche, Breitensuche und iterative Tiefensuche dar. Damit mag das klassische Planen trivial erscheinen, denn diese Art von Graphsuche ist gründlich erforscht. Allerdings kann gezeigt werden [8], dass selbst bei einfachen Planungsproblemen die Anzahl der im Zustandsraum enthaltenen Zustände größer sein kann als die Menge aller Atome im sichtbaren Universum. Dieser Ansatz erscheint daher für umfangreichere Planungsprobleme hoffnungslos schwierig. Ein menschlicher Planer würde sich der Lösung eines Planungsproblems auch nicht mit einer *brute force* Graphsuche nähern. Stattdessen würde er Wissen über das Problem verwenden, somit bestimmte Aktionsfolgen bevorzugt betrachten und andere Aktionsfolgen aus der Betrachtung ausschließen. Um eine derartige informierte Suche automatisch durchzuführen, stattet man oft den Planer mit einer *Heuristik* aus, die den minimalen Abstand des aktuellen Zustandes zu einem Zielzustand schätzt. Die Güte dieser Schätzung bestimmt wesentlich den Aufwand für die Suche nach einem Plan. Die genaue Definition eines Abstandes von Zuständen hängt von der Planungsdomäne ab. Oft wird darunter die minimale Länge der linear geordneten, endlichen Sequenz von Aktionen $\langle a_1, \dots, a_n \rangle$ verstanden, die den einen Zustand in den anderen überführt³.

1.2 Partielle Erfüllbarkeitsprobleme

In Abschnitt 1.3.2 wird das klassische STRIPS Planen vorgestellt werden. Etwas vorgehend besteht das Planungsproblem darin, eine Folge von Aktionen zu finden, so dass ein gegebener Ausgangszustand s_I in einen Zielzustand überführt wird, wobei die Menge der Zielzustände durch eine Menge von Formeln S_G repräsentiert wird. Ein Zustand s ist ein Zielzustand, falls alle Formeln aus S_G in s erfüllt sind, d.h. falls $s \subseteq S_G$ gilt. Der Erfolg einer Plansuche wird somit nur dadurch bestimmt, ob eine derartige Aktionsfolge

²Man beachte, dass die Lösung im Allgemeinen nicht eindeutig ist.

³Die Abstände sind im Allgemeinen nicht symmetrisch bzgl. Vertauschung dieser Zustände.

gefunden werden kann oder nicht. Oftmals ist aber wünschenswert auch Pläne unterscheiden zu können, die dazu führen, dass nur eine Teilmenge der Formeln aus S_G erfüllt ist. Dies gilt insbesondere für die Fälle, in denen kein Plan existiert, der obiges Planungsproblem löst. Dies ist z.B. dann gegeben, wenn sich einzelne Formeln aus S_G widersprechen.

Reale Planungsanwendungen sind oftmals vor die Aufgabe gestellt, Pläne unter der Berücksichtigung von Ressourcenbeschränkung zu finden. Beispielsweise wurde das OASIS System [9] für ein weitgehend autonomes Erheben von Messdaten von Mars-Rovern entwickelt. Eine wichtige Komponente dieses System beschäftigt sich mit dem Ablauf und der Zeitplanung von Rover-Aktionen um bestimmte Messungen durchführen zu können. Andere Beispiele stammen aus den Bereichen Satellitenüberwachung [10], der Zeitplanung des Hubble Space Teleskops [11] usw. Viele Nebenbedingungen, z.B. der maximale Energieverbrauch und die Zeitdauer von Aktionen, sind kritisch für einen erfolgreichen Einsatz dieser Systeme und müssen berücksichtigt werden. Die entsprechende Planungsdomäne wird als *Planning and Scheduling* bezeichnet und ähnelt in vielen Aspekten der Domäne der partiellen Erfüllbarkeitsprobleme. Ein Unterschied ist jedoch, dass Probleme des *Planning and Scheduling* ihr Hauptaugenmerk auf einen konfliktfreien Einsatz der vorhandenen Ressourcen richten, wobei die Zeitplanung eine besonders wichtige Rolle spielt. Die Optimierungsaufgabe besteht darin, einen Zeitablauf zu finden, in dem so viele Ziele wie möglich erfüllt bzw. so wenige Ressourcen wie nötig eingesetzt wurden. Übliche Lösungsansätze starten mit einem nichtoptimalen Plan und verbessern ihn iterativ [12], indem sie Teilpläne für weitere Ziele hinzufügen und die Anzahl der Konflikte reduzieren. Sofern der Startplan nicht zu optimistisch war, erhält man schließlich einen konfliktfreien und qualitativ guten Plan. Alternativ können derartige Pläne auch konstruktiv aufgebaut werden [13]. Dabei werden immer weitere Teilpläne hinzugefügt – solange sie sich zu einem konfliktfreien Gesamtplan kombinieren lassen.

Das Hauptaugenmerk in partiellen Erfüllbarkeitsproblemen (PSP – *partial satisfaction problems*) liegt hingegen auf den Zielen selbst [14]. Typischerweise werden wesentlich mehr potentielle Ziele angestrebt, als sich überhaupt erreichen lassen. Entweder aufgrund der fehlenden Ressourcen oder weil die Zielmenge widersprüchlich ist. Die Optimierungsaufgabe besteht darin, eine (im gewissen Sinn) maximale Teilmenge dieser Zielmenge zu finden und einen Plan, sie zu erreichen. Vermutlich lassen sich jedoch in einem Kontext, der explizite Zeitmodellierung erlaubt, Probleme des *Planning and Scheduling* ebenfalls als partielle Erfüllbarkeitsprobleme auffassen. Es ist möglich, partielle Erfüllbarkeitsprobleme mittels einer kleinen Veränderung von Definition 1 einzuführen. Im Folgenden ist stets ein klassisches partielles Erfüllbarkeitsproblem gemeint, wenn von partiellen Erfüllbarkeitsproblemen die Rede ist.

Definition 3 (Partielles Erfüllbarkeitsproblem – PSP). Ein 6-Tupel $\Gamma = (S, A, \gamma, s_I, S_G, \succeq)$ bestehend aus einer endlichen Menge von Zuständen S , einer endlichen Menge von Aktionen A , einer Übergangsfunktion $\gamma : S \times A \rightarrow S$, einem Anfangszustand $s_I \in S$, einer Beschreibung der Zielzustände $S_G \subseteq S$ und einer reflexiven und transitiven Relation $\succeq \subseteq S_G \times S_G$ heißt partielles Erfüllbarkeitsproblem oder PSP.

Andere Begriffe, wie z.B. Anwendbarkeit von Aktionen oder Erreichbarkeit von Zuständen, übertragen sich entsprechend, vgl. Abschnitt 1.1.

Definition 4 (Lösung des partiellen Erfüllbarkeitsproblems). Sei ein partielles Erfüllbarkeitsproblem Γ gegeben. Es heißt lösbar, falls mindestens ein Zustand $s_G \in S_G$ existiert, der vom Ausgangszu-

stand s_I erreichbar ist. Ein maximaler Zielzustand ist ein Zustand $s \in S_G$ derart, dass für keinen Zustand $s' \in S_G$ gilt: $s' \succeq s$ und $s \not\preceq s'$. Eine linear geordnete, endliche Sequenz von Aktionen $\Delta = \langle a_1, \dots, a_n \rangle$ heißt Lösung des Problems Γ oder Plan, wenn die Anwendung der Aktionssequenz Δ auf den Ausgangszustand s_I in einen maximalen Zielzustand führt⁴.

Ein lineares partielles Erfüllbarkeitsproblem (LPSP) ist ein PSP mit einer *linearen* Relation \succeq . Die Relation \succeq lässt sich äquivalent durch die \geq Relation über den natürlichen Zahlen beschreiben, wenn Zielzustände geeignet in natürliche Zahlen übersetzt werden. Die Lösung des LPSP ist analog der Lösung des PSP definiert.

Ein einfacheres Problem stellt das lineare partielle Erfüllbarkeitsproblem mit Schranke dar (LPSP*).

Definition 5 (Lineares partielles Erfüllbarkeitsproblem mit Schranke – LPSP*). Ein 7-Tupel $\Gamma = (S, A, \gamma, s_I, S_G, \succeq, s_C)$ bestehend aus einer endlichen Menge von Zuständen S , einer endlichen Menge von Aktionen A , einer Übergangsfunktion $\gamma : S \times A \rightarrow S$, einem Anfangszustand $s_I \in S$, einer Beschreibung der Zielzustände $S_G \subseteq S$, einer reflexiven, transitiven und *linearen* Relation $\succeq \subseteq S_G \times S_G$ und einem Zielzustand $s_C \in S_G$ heißt lineares partielles Erfüllbarkeitsproblem mit Schranke oder LPSP*.

Die Lösung des LPSP* ist wie folgt gegeben.

Definition 6 (Lösung des linearen partiellen Erfüllbarkeitsproblems mit Schranke). Sei ein lineares partielles Erfüllbarkeitsproblem mit Schranke Γ gegeben. Es heißt lösbar, falls eine linear geordnete, endliche Sequenz von Aktionen $\Delta = \langle a_1, \dots, a_n \rangle$ existiert, die angewendet auf den Ausgangszustand s_I in einen Zielzustand s mit $s \succeq s_C$ führt. Eine derartige Sequenz heißt Lösung des Problems Γ oder kurz Plan.

Sei nun ein LPSP* $\Gamma = (S, A, \gamma, s_I, S_G, \succeq, s_C)$ gegeben. Dies lässt sich in ein äquivalentes klassisches Planungsproblem mit $\Gamma' = (S, A, \gamma, s_I, S'_G = \{s \mid s \in S_G, s \succeq s_C\})$ übersetzen, wobei Äquivalenz die hier offensichtlich geltende Eigenschaft bezeichnet, dass eine Aktionssequenz genau dann Lösung von Γ ist, wenn sie Lösung von Γ' ist.

In der weiteren Arbeit wird nun folgendes Konzept zur Lösung partieller Erfüllbarkeitsprobleme detailliert ausgeführt.

1. Es ist zu klären, wie die Relation \succeq , die eine Präferenzierung zwischen verschiedenen Zuständen angibt, effektiv beschrieben werden kann. Dies erfolgt in Kapitel 2.
2. Anschliessend wird die Übersetzung eines PSP in ein LPSP vorbereitet, indem in Kapitel 3 die Linearisierung der Relation \succeq vorgestellt wird. Es wird eine Präferenzbeschreibungssprache eingeführt, mit der sich effektiv lineare Präferenzen ausdrücken lassen.
3. In Kapitel 4 wird zunächst aufgezeigt, wie sich eine Lösung eines LPSP durch einen Algorithmus finden lässt, der ein LPSP* löst. Dabei erfolgt die Übersetzung eines LPSP* in ein klassisches Planungsproblem, wobei als Repräsentationssprachen PDDL2.1 und PDDL2.1' verwendet werden (siehe Abschnitt 1.3.3).

⁴Da die Zustandsmenge S als endlich angenommen wird, existiert, sofern das Problem Γ überhaupt lösbar ist, für jede reflexive und transitive Relation \succeq ein maximaler Zielzustand.

1.3 Repräsentationssprachen für Planungsprobleme

1.3.1 Historische Entwicklung

Von 1966 bis 1972 wurde am Stanford Research Institut ein mobiler Roboter namens „Shakey“ entwickelt. Als Planungskomponente des Roboters wurde das STRIPS System [2] entwickelt und eingesetzt⁵. Seine Kontrollstruktur baute auf dem 1961 entwickelten GPS (*General Problem Solver*) [15] auf. Für die Suche im Zustandsraum wurde eine Mittel-Zweck-Analyse (*means-end analysis*) verwendet. Einen wesentlichen Einfluß auf die weitere Entwicklung der künstlichen Intelligenz hatte jedoch die in diesem Zusammenhang entworfene Repräsentationssprache (STRIPS-Sprache). In verschiedenen Varianten und Erweiterungen wird sie bis heute eingesetzt und weiterentwickelt. Eine formale Definition und Analyse der Sprache wurde in [16] durchgeführt. Eine historische Entwicklung der Sprache und ihr Bezug zu heutigen Entwicklungen in der künstlichen Intelligenz findet sich in [17]. Dass diese Sprache trotz ihrer Restriktivität hinreichend komplex ist, zeigt sich darin, dass bereits einfache STRIPS Planungsprobleme PSPACE-hart sind [18]. Mit ADL (Action Description Language) [19] wurden einige Restriktionen gelockert und eine realistischere Problembeschreibung ermöglicht. 1998 wurde PDDL (*Problem Domain Description Language*) [20] vorgestellt. Eine Sprache, die es erlaubt, innerhalb einer einheitlichen Syntax die bereits vorhandenen Sprachen STRIPS und ADL zu integrieren und zu erweitern. Sie stellt seitdem die Standardsprache für Planungswettbewerbe (AIPS, ICAPS, ...) dar.

1.3.2 STRIPS

Die heutzutage verwendete STRIPS-Variante definiert sich üblicherweise als Untersprache von PDDL und weicht in einigen Punkten von der Originalsprache [2] ab. Die Grundobjekte von STRIPS stellen aussagenlogische Atomformeln (Atome, Fakten) dar. Syntax und Semantik der Sprache werden im Folgenden kurz vorgestellt.

Die Menge aller Fakten werde mit F bezeichnet. Ein Zustand $s \subseteq F$ ist syntaktisch durch eine endliche Menge von Fakten repräsentiert. Die semantische Interpretation einer derartigen Formelmenge erfolgt unter der folgenden Abgeschlossenheitsannahme (*closed world assumption*). Eine Formel $f \in F$ sei im Zustand s genau dann wahr, wenn $f \in s$. Insbesondere werden nicht in s vorkommende Formeln als falsch interpretiert. Eine Aktion a ist syntaktisch gegeben als Tripel von Mengen von Fakten $a = (\text{pre}(a), \text{eff}^+(a), \text{eff}^-(a))$. Hierbei nennt man $\text{pre}(a)$ die Vorbedingungen, $\text{eff}^+(a)$ die positiven Effekte oder add-Liste, $\text{eff}^-(a)$ die negative Effekte oder del-Liste der Aktion a und es gilt die Einschränkung $\text{eff}^+(a) \cap \text{eff}^-(a) = \emptyset$. Der Ausgangszustand $s_I \subseteq F$ wird durch eine Menge von Fakten angegeben. Die Zielbeschreibung $S_G \subseteq F$ erfolgt ebenfalls durch eine Menge von Fakten. Sie repräsentiert alle Zielzustände s_G mit $s_G \subseteq S_G$. Die Übergangsfunktion γ , welche Aktionen semantisch beschreibt, ist nun wie folgt definiert:

$$\gamma(s, a) = \begin{cases} s \setminus \text{eff}^-(a) \cup \text{eff}^+(a) & , \text{ wenn } \text{pre}(a) \subseteq s \\ \text{undefiniert} & , \text{ anderenfalls.} \end{cases}$$

⁵STRIPS ist das Akronym für STanford Research Institute Problem Solver.

```

(define (domain satellite)
(:requirements :strips)
(:predicates
  (on_board ?i ?s) (power_avail ?s)
  (power_on ?i) (satellite ?x) (instrument ?x) )
(:action switch_on
:parameters ( ?i ?s)
:precondition
  (and (instrument ?i) (satellite ?s) (on_board ?i ?s) (power_avail ?s))
:effect
  (and (power_on ?i) (not (power_avail ?s)) )
)

```

Abbildung 1.2: Auszug aus der STRIPS-Domänenbeschreibung der Satelliten-Domäne

Diese Definition der Übergangsfunktion legt auch fest, was mit einer Formel f geschieht, die nicht durch die Anwendung der Aktion a beeinflusst wird. Sie gilt in Zustand $\gamma(s, a)$ genau dann, wenn sie in Zustand s gilt. Das aus dem Situationskalkül bekannte Problem der Persistenz der Eigenschaften (*frame problem*) wird so umgangen.

Ein STRIPS-Planungsproblem ist durch das 5-Tupel $(\mathcal{P}(F), A, \gamma, s_I, S_G)$ gegeben. Die Definition einer Lösung des STRIPS-Planungsproblems erfolgt in analoger Weise zur allgemeinen Definition in 1.1. Insbesondere werden wir uns in dieser Arbeit auf klassische lineare Aktionsfolgen als Lösungen beschränken, d.h. keine gleichzeitige Ausführung von Aktionen oder Aktionen endlicher Dauer betrachten. Dies vereinfacht erheblich die Beschreibung der Semantik, z.B. ist jetzt nicht zu klären, wann sich Aktionen behindern und nicht simultan ausgeführt werden können.

Die genaue Syntax der STRIPS-Sprache als Untersprache von PDDL2.1 wird in Anhang A beschrieben.

Am Beispiel einer Probleminstanz aus der Satelliten-Domäne sollen nun die Konstrukte der STRIPS-Sprache veranschaulicht werden. Die Aufgabe besteht in dieser Domäne darin, bestimmte Observationsmessungen mittels verschieden ausgestatteter Satelliten durchzuführen. Eine Sammlung von Probleminstanzen in dieser Domäne und von Varianten der Domänenbeschreibungen formuliert in STRIPS und PDDL findet sich in [21]. Doch zunächst einige Anmerkungen. In der STRIPS-Sprache kommen nur aussagenlogische Formeln vor, die insbesondere variablenfrei sind. Nun bietet es sich zur effizienten Beschreibung der Aktionen⁶ jedoch an, Aktionsschemata zu verwenden. Ein Aktionsschema kann sowohl variablenbehaftete als auch variablenfreie Formeln verwenden. Aus einem Schema ergeben sich die jeweiligen Aktionen durch Substitution der Variablen mit geeigneten Objekten⁷. In analoger Weise sind Atomschemata zu verstehen.

Zudem wird die Problembeschreibung oft in eine Domänenbeschreibung und eine Faktenbeschreibung aufgetrennt. Beschreibungen der vorhandenen Aktions- und Atomschemata werden der Domänenbeschreibung zugeordnet. So legt sie die allgemeinen Eigenschaften der betrachteten Modellwelt fest. Die Angabe

⁶ Aktionen bzw. Aktionsschemata werden auch als Operatoren bzw. Operatorschemata bezeichnet.

⁷ Als Objekte werden die in den Atomen vorkommenden Konstanten der Sprache bezeichnet.

```

(define (problem strips-sat-x-1)
 (:domain satellite)
 (:objects
  satellite0
  satellite1
  instrument0)
 (:init
  (satellite satellite0) (satellite satellite1) (instrument instrument0)
  (on_board instrument0 satellite0) (power_avail satellite0))
 (:goal
  (power_on instrument0))
)

```

Abbildung 1.3: Auszug aus einer STRIPS-Faktenbeschreibung der Satelliten-Domäne

der vorhandenen Objekte, des Anfangszustandes und des Zielzustandes, d.h. die Konkretisierung der Modellwelt, erfolgt separat als Faktenbeschreibung. Die Auftrennung der Beschreibung einer Problem Instanz dient vor allem der Wiederverwendung der Domänenbeschreibung für verschiedene Konkretisierungen.

In Abbildung 1.2 findet sich ein Auszug aus der Domänenbeschreibung der Satelliten-Domäne. Die Syntax ist weitgehend selbsterklärend. Die Angabe `(:requirements :strips)` weist auf die ausschließliche Verwendung von STRIPS Sprachkonstrukten hin. Neben den verwendeten Atomschemata `(:predicates)` ist exemplarisch ein Aktionsschema angegeben `(:action)`. Hier fällt auf, dass die Effekte nicht in positive und negative Effekte getrennt sind. Vielmehr werden positive Effekte durch positive Literale oder positive Literalschemata definiert, negative Effekte hingegen durch negative Literale oder negative Literalschemata⁸.

Die Anwendung der Aktion `switch_on(instrument0,satellite0)` überführt den Anfangszustand s_I in den Zustand s' :

$$\begin{aligned}
 s_I &= \{(\text{satellite satellite0}), (\text{satellite satellite1}), (\text{instrument instrument0}), \\
 &\quad (\text{on_board instrument0 satellite0}), (\text{power_avail satellite0})\} \\
 &\quad \Downarrow \text{switch_on}(\text{instrument0}, \text{satellite0}) \\
 s' &= \{(\text{satellite satellite0}), (\text{satellite satellite1}), (\text{instrument instrument0}), \\
 &\quad (\text{on_board instrument0 satellite0}), (\text{power_on instrument0})\}.
 \end{aligned}$$

Aufgrund von $(\text{power_on instrument0}) \in s'$ stellt somit die einelementige Aktionssequenz

$$\langle \text{switch_on}(\text{instrument0}, \text{satellite0}) \rangle$$

eine Lösung für die in Abbildung 1.2 und 1.3 definierte Problem Instanz dar.

⁸Atome (bzw. negierte Atome) werden auch als positive (bzw. negative) Literale bezeichnet.

1.3.3 PDDL2.1

ADL: Für viele reale Planungsprobleme erwies sich STRIPS als unzureichend oder zu umständlich. ADL [19] erweitert STRIPS in folgenden Aspekten.

- Als Vorbedingungen von Aktionen und Zielbeschreibungen sind beliebige Formeln der Prädikatenlogik erster Stufe mit funktionsfreien Literalen erlaubt. Somit sind insbesondere quantifizierte Ausdrücke und das Verwenden von Variablen erlaubt.
- Aktionen können bedingte Effekte besitzen. Dies sind Effekte, die nur ausgeführt werden, sofern die entsprechenden Effekt-Bedingungen vor Ausführung der Aktion gelten.
- Die Typisierung von Variablen wird unterstützt.
- Eine Gleichheitsrelation wird unterstützt.

In der ursprünglichen Version werden in ADL Zustände s durch Literalismengen definiert. Sei l ein derartiges Literal und bezeichne $\neg l$ das enthaltene Atom, falls l ein negatives Literal ist, und ansonsten die Negation von l . Dann besagt die Offenheitsannahme (*open world assumption*) von ADL, dass

- l in Zustand s gilt, falls $l \in s$
- l in Zustand s nicht gilt, falls $\neg l \in s$

und ansonsten der Wahrheitswert von l unbekannt ist.

PDDL2.1: Im Jahr 1998 wurde mit PDDL [20] eine weitere Sprache für Planungsprobleme veröffentlicht, welche ADL erweitert und inzwischen zum Standard für die Darstellung von Planungsdomänen herangereift ist. Die prominente Rolle dieser Sprache ist auch mit der derzeitigen Entwicklung in der künstlichen Intelligenz verbunden, die Lösung immer komplexerer und realistischerer Planungsaufgaben anzustreben. Die zu beschreibenden, realitätsnahen Modellwelten besitzen oftmals Eigenschaften, die sich mit den Beschränkungen einer STRIPS-Sprache gar nicht oder nicht effektiv beschreiben lassen. Insbesondere muss ein moderner Planer auch in der Lage sein, mit numerischen Größen umzugehen, mit der gleichzeitigen Ausführung mehrerer Aktionen und mit Aktionen, die eine endliche Dauer besitzen. Als Repräsentationssprache für Planungsprobleme wurde im 3. Internationalen Planungswettbewerb (2002) die Version PDDL2.1 verwendet, die nun vorgestellt werden soll. Die Beschreibung folgt [3].

In Abbildung 1.4 wird eine PDDL2.1 Domänenbeschreibung der Satelliten-Domänen dargestellt. Folgende Unterschiede im Vergleich zu Abbildung 1.2 stehen hervor.

- Die Verwendung von bedingten Effekten der Form (**when P Q**). Der Effekt Q wird demnach nur ausgeführt, wenn die Bedingung P vor Ausführung der Aktion gilt.
- Es werden typisierte Objekte und Variablen unterstützt. Dies vereinfacht z.B. Beschreibung der Aktionsschemata.
- Die Deklaration und Anwendung von Funktionen. Funktionen können Variablen als typisierte Argumente besitzen und bilden in die Menge der rationalen Zahlen ab⁹.

⁹Tatsächlich bilden Funktionen in den Bereich der auf einem Computer darstellbaren Fließkommazahlen endlicher Länge

```

(define (domain satellite)
  (:requirements
   :conditional-effects :negative-preconditions
   :typing :fluents :equality)
  (:types satellite direction instrument mode)
  (:predicates
   (on_board ?i - instrument ?s - satellite)
   (power_avail ?s - satellite)
   (power_on ?i - instrument)
   (have_image ?d - direction ?m - mode))
  (:functions
   (data_capacity ?s - satellite)
   (data ?d - direction ?m - mode)
   (data-stored))
  (:action switch_on
   :parameters (?i - instrument ?s - satellite)
   :precondition (and (on_board ?i ?s) (power_avail ?s))
   :effect (and (power_on ?i) (not (power_avail ?s))))
  (:action take_image
   :parameters (?s - satellite ?d - direction ?i - instrument ?m - mode)
   :precondition (and (on_board ?i ?s)
                      (power_on ?i)
                      (>= (data_capacity ?s) (data ?d ?m)))
   :effect (when (not (have_image ?d ?m))
             (and (decrease (data_capacity ?s) (data ?d ?m))
                  (have_image ?d ?m)(increase (data-stored) (data ?d ?m))))))
)

```

Abbildung 1.4: Auszug aus der PDDL2.1-Domänenbeschreibung der Satelliten-Domäne.

- Der Einsatz negativer Vorbedingungen, z.B. im erwähnten bedingten Effekt.
- Im `:requirements`-Feld sind entsprechende Einträge angegeben, um Nutzer der PDDL2.1-Domänenbeschreibung auf die sprachlichen Erweiterungen hinzuweisen.

PDDL2.1 wird in mehrere Sprachebenen (*levels*) unterteilt. Das STRIPS Fragment von PDDL2.1 wird als PDDL2.1 level 1 bezeichnet. Die zusätzliche Unterstützung von numerischen und ADL-Konstrukten definiert PDDL2.1 level 2. Weitere Ebenen ermöglichen die Verwendung von Aktionen mit endlicher Ausdehnung. Diese Arbeit beschränkt sich auf PDDL2.1 level 2. Im Folgenden bezeichne PDDL2.1 stets PDDL2.1 level 2.

Die Semantik der nichtnumerischen Komponente von PDDL2.1 wird über eine Transformation in die STRIPS-Sprache definiert. Insbesondere werden anstelle von Literal Mengen (wie in ADL) wieder Mengen

ab.

von Atomformeln zur Zustandsbeschreibung verwendet. Im Folgenden Absatz sei kurz angedeutet auf welche Weise Quantoren, bedingte Effekte und Typisierung von Objekten und Variablen mittels STRIPS-Konstrukten ausgedrückt werden können. Für eine formale und wesentlich umfangreichere Darstellung sei auf [3, S.22ff] verwiesen. Anschließend wird auf die numerische Komponente eingegangen.

Transformation von ADL nach STRIPS: Um die Semantik des ADL Fragments von PDDL2.1 zu definieren, werden alle ADL Konstrukte mittels STRIPS-Sprachelementen ausgedrückt. Gegebene Planer müssen natürlich eine derartige Transformation nicht unbedingt durchführen. So wurde z.B. in [22] gezeigt, dass eine Kompilation von bedingten Effekten in STRIPS-Konstrukte im Allgemeinen zu einer exponentiellen Vergrößerung der Domänenbeschreibung führt, sofern die Länge der Lösung des Planungsproblems bewahrt werden soll¹⁰. Die Transformation der ADL-Sprachelemente folgt [23].

1. Transformation aller Aktionsschemata. Sei a ein Aktionsschema. Die Menge S_a enthalte anfangs das Schema a und werde nun wie folgt konstruktiv definiert:

- Falls S_a ein Aktionsschema b mit einem bedingten Effekt (when P Q) enthält, werden zwei neue Aktionsschemata gleichen Namens erzeugt, die b bis auf folgende Unterschiede gleichen. In beiden neuen Schemata wird der bedingte Effekt entfernt. Im ersten neuen Aktionsschema wird die Bedingung P zu den Vorbedingungen hinzugefügt¹¹ und Q zu den Effekten. Im zweiten neuen Schema werden die Vorbedingungen um (not P) erweitert. Anschliessend wird b aus S_a entfernt und die beiden neuen Schemata hinzugefügt.
- Sofern S_a ein Aktionsschema b mit einer quantifizierten Formel $f = (\text{QUANT } (\text{Var}_1 \dots \text{Var}_n) P)$ enthält, wird aus b in folgender Weise ein Aktionsschema erzeugt. Die Formel f wird durch eine Konjunktion (falls QUANT der All-Quantor ist) oder Disjunktion (falls QUANT der Existenz-Quantor ist) aller derjenigen Ausdrücke P' ersetzt, die aus P durch die Substitution der Variablen $\text{Var}_1 \dots \text{Var}_n$ mit Objekten auf jede mögliche Weise gebildet werden können.
- Die beiden Schritte werden solange wiederholt, bis keiner der beiden Schritte mehr anwendbar ist. Anschließend wird mit dem nächsten Aktionsschema a' fortgesetzt. Anstelle der ursprünglichen Aktionsschemata wird nun die Vereinigung aller nach obigen Regeln gebildeten Mengen S_a verwendet.

2. Instanziierung alle Schemata, d.h. das Ersetzen von Objektvariablen durch Objekte, die bezüglich der Typisierung erlaubt sind.

3. Überführung aller Formeln in disjunktive Normalform. Negative Literale werden durch die Einführung von Hilfsatomen aufgelöst¹².

4. Disjunktionen in Formeln werden, wie in [23] beschrieben, aufgelöst durch Trennen der Aktionen, Effekte bzw. Zielbedingungen.

¹⁰Die Länge eines Planes ist durch die Anzahl der enthaltenen Aktionen gegeben.

¹¹d.h. die neue Vorbedingung ergibt sich aus einer UND Verknüpfung der alten Vorbedingung mit P.

¹²z.B. wird (not_power_avail satellite0) anstelle von (not (power_avail satellite0)) verwendet.

Die numerische Komponente von PDDL2.1 In diesem Abschnitt werden Syntax und Semantik von numerischen Effekten, numerischen Bedingungen und numerischen Ausdrücken kurz erläutert. Die Ausführung folgt dabei weitgehend [24]. Nach der im vorherigen Abschnitt beschriebenen Transformation der ADL-eigenen Konstrukte besteht die nun vorliegende Sprache lediglich aus Elementen der STRIPS-Sprache und numerischen Konstrukten.

In Abschnitt 1.3.2 wurde ein Zustand als Menge von Atomformeln interpretiert. Aufgrund der Verwendung von Funktionen und numerischen Variablen muss die Zustandsdefinition nun erweitert werden. Sei mit $F = \{f_1, \dots, f_i\}$ die Menge aller Atomformeln (Fakten) bezeichnet und mit $V = \{v_1, \dots, v_k\}$ die Menge der numerischen Variablen. Ein Zustand $s \in S = \mathcal{P}(F) \times \mathbb{Q}^k$ ist nun definiert als Paar $s = (f(s), \nu(s))$, wobei $f(s) \subseteq F$ und $\nu(s) = (\nu_1(s), \dots, \nu_k(s)) \in \mathbb{Q}^k$ ein k -Tupel rationaler Zahlen ist¹³. Zunächst wird eine Kurzfassung der Syntax der Sprache vorgestellt¹⁴.

Ein *numerischer Ausdruck* ist ein arithmetischer Ausdruck unter Verwendung der Variablen aus V und rationalen Zahlen mit den Operatoren $+$, $-$, $*$ und $/$. Ein *numerischer Effekt* ist ein Tripel (v_i, ass, exp) , mit $v_i \in V$, $ass \in \{\text{assign, increase, decrease, scale-up, scale-down}\}$ ist ein Zuweisungsoperator und exp ein numerischer Ausdruck. Ein *numerischer Vergleich* ist ein Tripel $(exp, comp, exp')$, wobei exp und exp' numerische Ausdrücke darstellen und $comp \in \{<, >, =, \leq, \geq\}$ einen Vergleichoperator bezeichnet.

Die bereits aus der STRIPS-Sprache bekannten Begriffe Bedingung, Effekt und Aktion werden nun um numerische Konstrukte erweitert. Eine *Bedingung* ist ein Paar (f, c) , wobei $f \subseteq F$ und c eine Menge numerischer Vergleiche darstellt. Ein *Effekt* ist ein Triple (f^+, f^-, nef) , wobei $f^+ \subseteq F$, $f^- \subseteq F$ und nef eine Menge numerischer Effekte bestimmt. Die Menge nef ist nicht ganz beliebig, sondern muss gewisse Konsistenzbedingungen erfüllen, die sicherstellen, dass die Reihenfolge der Ausführung numerischer Effekte aus nef keine Rolle spielt. Eine *Aktion* ist ein Paar (pre, eff) , das aus einer Bedingung pre und einem Effekt eff besteht.

Die Semantik der definierten Konstrukte wird nun erklärt. Der Wert $exp(\nu(s))$ eines numerischen Ausdrucks exp im Zustand $s = (f(s), \nu(s))$ ist eine rationale Zahl, die durch Substitution der Variablen v_i mit ihren Werten $\nu_i(s)$ und der anschließenden üblichen Auswertung von arithmetischen Ausdrücken entsteht. Sofern eine Division durch 0 auftritt, ist der Wert des numerischen Ausdrucks nicht definiert¹⁵. Ein numerischer Vergleich $(exp, comp, exp')$ ist im Zustand s erfüllt, sofern die Werte der Ausdrücke exp und exp' im Zustand s definiert sind und der Vergleich der Werte mit der üblichen Interpretation des Vergleichsoperators $comp$ gilt. Eine Bedingung (f, c) gilt in Zustand s , falls $f \subseteq f(s)$ und alle numerischen Vergleiche aus c in Zustand s erfüllt sind.

Die Übergangsfunktion beschreibt die Wirkung von Aktionen auf Zustände. Die Wirkung einer Aktion $(pre, (f^+, f^-, nef))$ auf den Vektor $\nu(s)$ ergibt sich aus der Hintereinanderausführung der numerischen Effekte aus nef auf $\nu(s)$. Ein numerischer Effekt heißt anwendbar auf den Vektor ν , falls $exp(\nu)$ definiert

¹³Dieser Definition liegt offensichtlich die Vorstellung zugrunde, dass der i -te Eintrag im Vektor $\nu(s)$ dem Wert der numerischen Variable v_i im Zustand s entspricht.

¹⁴Die Syntax wird hier in einer etwas verkürzten und vereinfachten Form vorgestellt. Insbesondere wird die Infix Notation verwendet. Die exakte PDDL2.1 Syntax in Präfixform kann in Anhang A nachgelesen werden.

¹⁵In [3] wird der Wert `undef` explizit in den Wertebereich von numerischen Variablen aufgenommen. Sofern in einem arithmetischen Ausdruck eine Division durch 0 auftritt oder eine Variable den Wert `undef` annimmt, ist Wert des Ausdrucks ebenfalls `undef`.

ist. Die Anwendung $\omega_e : \mathbb{Q}^k \rightarrow \mathbb{Q}^k$ eines numerischen Effektes $e = (v_i, ass, exp)$ auf einen Vektor rationaler Zahlen ν ist dabei gegeben durch die mittels *ass* angegebene Ersetzung der Vektorkomponente ν_i von ν mit dem Wert des numerischen Ausdruckes $exp(\nu)$. Analog heißt die Hintereinanderausführung von numerischen Effekten $\omega_{nef} = \omega_{e_1} \circ \dots \circ \omega_{e_l}$ (mit $nef = \{e_1, \dots, e_l\}$) anwendbar auf den Vektor ν , falls e_l auf ν anwendbar ist und jeder numerische Effekt e_i mit $i \in \{1, \dots, l-1\}$ auf ν^i mit $\nu^i = \omega_{e_{i+1}} \circ \dots \circ \omega_{e_l}(\nu)$ anwendbar ist. Damit die Hintereinanderausführung der numerischen Effekte nicht von ihrer Reihenfolge abhängt, sind gewisse Bedingungen an die Menge der numerischen Effekte zu stellen (siehe oben). Ein Effekt (f^+, f^-, nef) heißt anwendbar, falls ω_{nef} anwendbar ist. Die Wirkung der Übergangsfunktion $\gamma : S \times A \rightarrow S$ auf das Paar (s, a) mit $s = (f, \nu)$ und $a = (pre, (f^+, f^-, nef))$ ist nicht definiert, falls der Effekt (f^+, f^-, nef) nicht auf ν anwendbar ist oder die Bedingung *pre* nicht in s erfüllt ist. Ansonsten ist die Wirkung definiert als

$$\gamma(s, a) = (f', \nu') \text{ mit } f' = f \setminus f^- \cup f^+ \text{ und } \nu' = \omega_{nef}(\nu).$$

Ein Planungsproblem der PDDL2.1-Sprache ist durch das 5-Tupel (S, A, γ, s_I, S_G) gegeben. Hierbei ist die Zustandsmenge S durch $\mathcal{P}(F) \times \mathbb{Q}^k$ gegeben, wobei $k = |V|$ die Anzahl der numerischen Variablen angibt. A ist eine Menge von Aktionen, γ bezeichnet die oben definierte Übergangsfunktion und $s_I \in S$ ist eine Beschreibung des Anfangszustandes. Die Menge der Zielzustände wird durch eine Bedingung S_G spezifiziert. Aus γ läßt sich analog zu der in Abschnitt 1.1 angegebenen Weise eine Funktion $\gamma^* : S \times A^* \rightarrow S$ konstruieren. Eine Sequenz von Aktionen $\langle a_1, \dots, a_n \rangle \in A^*$ heißt eine Lösung des Planungsproblems oder Plan, falls die Bedingung S_G im Zustand $\gamma^*(s_I, \langle a_1, \dots, a_n \rangle)$ erfüllt ist.

Lineare numerische Ausdrücke – die Sprache von Metric-FF Der erfolgreichste vollautomatische Planer des AIPS-2000 Planungswettbewerbs war das FF Planungssystem [25]. Dieser Planer verwendet eine heuristischen Vorwärtssuche im Zustandsraum zur Planlösung. Zunächst wird das Planungsproblem dahingehend vereinfacht, dass negative Effekte von Aktion ignoriert werden. Das so vereinfachte Planungsproblem wird dann mit einem Algorithmus gelöst, welcher das Graphplan-Verfahren [26] adaptiert. Die Anzahl der in der Planlösung des vereinfachten Planungsproblems enthaltenen Aktionen ergibt eine Schätzung des Abstandes des aktuellen Zustand zu einem Zielzustand. Diese Schätzung (Heuristik) lenkt die Vorwärtssuche im Zustandsraum des ursprünglichen Planungsproblems. Eine Limitation dieses System bestand jedoch darin, dass Planungsprobleme in STRIPS oder ADL gestellt werden mussten. Metric-FF [27] stellt eine Erweiterung des FF-Systems dahingehend dar, dass jetzt eine Untersprache PDDL2.1' von PDDL2.1 unterstützt wird. Die einzige Einschränkung von PDDL2.1' in Bezug auf PDDL2.1 liegt darin, dass lediglich *lineare* numerische Ausdrücke verwendet werden dürfen.

Definition 7 (Lineare numerische Ausdrücke). Eine numerische Variable v aus PDDL2.1 heißt problemkonstant, wenn sie durch keinen Effekt einer Aktion geändert werden kann. Ein numerischer Ausdruck heißt problemkonstant, wenn er nur aus problemkonstanten numerischen Variablen und rationalen Zahlen gebildet ist. Sei nun ein numerischer Ausdruck exp gegeben. Ist für jeden enthaltenen numerischen Ausdruck der Form $(exp_1 * exp_2)$ mindestens einer der numerischen Unterausdrücke exp_1 bzw. exp_2 problemkonstant und ist für jeden enthaltenen numerischen Ausdruck der Form (exp_1/exp_2) der numerische Ausdruck exp_2 problemkonstant, so heißt exp linear.

Offenbar spezifizieren numerische Ausdrücke diejenigen arithmetischen Ausdrücke, die in jeder enthaltenen numerischen Variable linear sind.

Wenn ein Planungsproblem in PDDL2.1' gegeben ist, kann es in eine Normalform gebracht werden, in welcher numerische Ausdrücke die Form $\sum_j c_j * v_j + c$ und numerische Vergleiche die Form $(\sum_j c_j * v_j, comp, c)$ besitzen. Für die Konstanten c_j bzw. c gilt hierbei $c_j \in \mathbb{Q}^+$ bzw. $c \in \mathbb{Q}$. Als Vergleichoperator $comp$ ist $>$ oder \geq erlaubt. Ferner sind **assign** und **increase** die einzigen verwendeten Zuweisungsoperatoren der numerischen Effekte. Wenn das Planungsproblem in Normalform vorliegt und ein numerischer Vergleich in einem Zustand s gültig ist, dann ist er offenbar auch in all denjenigen Zuständen s' gültig, in denen keine numerische Variable einen kleineren Wert als in s annimmt. Das **Metric-FF** System verwendet nun ein zum **FF** System analoges Verfahren um eine Heuristik für die Problemlösung bereitzustellen. Dazu wird ein gegebenes PDDL2.1' Planungsproblem zunächst in Normalform überführt. Anschliessend wird ein vereinfachtes Planungsproblem gelöst, das dadurch entsteht, dass in den Effekten von Aktionen alle negativen aussagenlogischen Effekte und alle numerischen Effekte, die den Wert von numerischen Variablen senken, entfernt werden. Dieses Verfahren verallgemeinert damit das Vorgehen des **FF** Planers.

1.4 Spezielle Lösungsansätze in der Literatur

In diesem Abschnitt sollen aktuelle Ansätze zur Lösung von partiellen Erfüllbarkeitsproblemen und verwandter Problemstellungen kurz vorgestellt werden. Dabei ist zu beachten, dass die betrachteten Lösungszugänge oft im Rahmen spezieller Aufgabenstellungen entwickelt wurden und sich deshalb nicht immer mit dem in dieser Arbeit entwickelten Ansatz vergleichen lassen. Anschließend erfolgt eine kleine Diskussion mit dem Ziel, diese Zugänge von den Ergebnissen dieser Arbeit abzugrenzen bzw. eventuelle Gemeinsamkeiten hervorzuheben.

1.4.1 Integer Programming

In [28] wird die Anwendung von IP (*Integer Programming*)-Techniken zur Lösung klassischer Planungsprobleme vorgeschlagen. Ein Vorteil von IP liegt darin, dass auf natürliche Weise numerische Zwangsbedingungen und komplexe Zielbeschreibungen ermöglicht werden. Allerdings ist es für einen effizienten Einsatz von IP-Techniken notwendig, gegebene Planungsprobleme geeignet als IP-Probleme zu formulieren. Wie diese Formulierung ausfällt, bestimmt entscheidend die Leistungsfähigkeit des IP-Ansatzes.

Die Grundidee der Lösung von Planungsproblemen mittels IP besteht darin, die im Problem vorkommenden Aktionen und Zustände durch numerische Variablen auszudrücken und Zustandsübergänge mittels Zwangsbedingungen festzulegen. Das Problem entspricht der Suche nach einer ganzzahligen Belegung der Variablen, so dass alle Zwangsbedingungen erfüllt sind.

In [28] wird eine auf SATPLAN basierende Kodierung (vgl. auch [29]) vorgestellt, die STRIPS-artige Planungsprobleme als IP-Problem formuliert. Dabei werden Atomformeln f in numerische Variablen $x_{f,i} \in \{0, 1\}$ übersetzt, wobei genau dann $x_{f,i} = 1$ gilt, wenn die entsprechende Atomformel f im Zeitschritt i gilt (d.h. f in dem Zustand $s(i)$ gilt, den das Planungssystem nach Ausführung der Aktion a_i annimmt¹⁶). In einer alternativen Kodierung werden nun anstelle von Atomformeln Zustandsübergänge durch numerische Variablen repräsentiert. Diese Formulierung wollen wir uns kurz anschauen.

Zustandsübergänge werden durch die *Übergangsvariablen* $x_{f,i}^{\text{add}}$, $x_{f,i}^{\text{pre-add}}$, $x_{f,i}^{\text{pre-del}}$ und $x_{f,i}^{\text{maintain}}$ ausgedrückt. Außerdem verwendet die IP-Kodierung Variablen $y_{a,i}$ mit Werten 0 oder 1, welche Aktionen repräsentieren. Hierbei gilt $y_{a,i} = 1$ genau dann, wenn eine Aktion a im Zeitschritt i ausgeführt wird. Sei pre_f die Menge aller Aktionen, für die die Atomformel f eine Vorbedingung darstellt und sei add_f (del_f) diejenige Menge von Aktionen, die f als positiven (negativen) Effekt besitzen. Der Ausgangszustand s_I bzw. die Beschreibung S_G der Zielzustände ist durch eine Menge von Atomformeln gegeben.

Die Semantik dieser Variablen ist wie folgt gegeben. $x_{f,i}^{\text{add}}$ ist genau dann 1, wenn eine Aktion a zum Zeitpunkt i ausgeführt wird, die f als positiven Effekt, aber nicht als Vorbedingung besitzt. Analog ist $x_{f,i}^{\text{pre-add}}$ genau dann 1, wenn eine Aktion a ausgeführt wird, die f als Vorbedingung besitzt, jedoch nicht als negativen Effekt. Man beachte, dass implizit angenommen wird, dass die Atomformel f nach Anwendung von a weiterhin gilt, sofern sie nicht explizit als negativer Effekt angegeben ist. Sofern $x_{f,i}^{\text{pre-del}} = 1$ ist, tritt

¹⁶Die IP-Formulierung ist geeignet, um eine gleichzeitige Ausführung von Aktionen zu erfassen. Dann wird ein Plan nicht mehr als Aktionssequenz $\langle a_1, \dots, a_n \rangle$, sondern als Sequenz von Aktionsmengen $\langle A_1, \dots, A_n \rangle$ aufgefasst. Die Zahl $i \in \{1, \dots, n\}$ wird als Zeitschritt bezeichnet und alle Aktionen $a \in A_i$ eines Zeitschrittes werden gleichzeitig ausgeführt.

genau der Fall ein, dass f eine Vorbedingung und ein negativer Effekt von a ist. Die Übergangsvariable $x_{f,i}^{\text{maintain}}$ wiederum nimmt den Wert 1 genau dann an, wenn f im Zeitschritt i gilt und keine Aktion zum Zeitpunkt i als Vorbedingung oder Effekt besitzt. Diese Variable repräsentiert somit eine Formel f , die, zum Zeitpunkt i geltend, durch eine *no-op* Aktion [26] in den Zustand zum Zeitschritt $i + 1$ propagiert.

Nun werden Zwangsbedingungen eingeführt, um das gegebene Planungsproblem zu repräsentieren. Diese Bedingungen gelten, sofern nicht anders angegeben, für alle Atomformeln f :

$$\sum_{a \in \text{add}_f \setminus \text{pre}_f} y_{a,i} \geq x_{f,i}^{\text{add}} \quad (1.1a)$$

$$y_{a,i} \leq x_{f,i}^{\text{add}}, \quad \forall a \in \text{add}_f \setminus \text{pre}_f \quad (1.1b)$$

$$\sum_{a \in \text{pre}_f \setminus \text{del}_f} y_{a,i} \geq x_{f,i}^{\text{pre-add}} \quad (1.1c)$$

$$y_{a,i} \leq x_{f,i}^{\text{pre-add}}, \quad \forall a \in \text{pre}_f \setminus \text{del}_f \quad (1.1d)$$

$$\sum_{a \in \text{pre}_f \cap \text{del}_f} y_{a,i} = x_{f,i}^{\text{pre-del}} \quad (1.1e)$$

$$x_{f,i}^{\text{add}} + x_{f,i}^{\text{maintain}} + x_{f,i}^{\text{pre-del}} \leq 1 \quad (1.1f)$$

$$x_{f,i}^{\text{pre-add}} + x_{f,i}^{\text{maintain}} + x_{f,i}^{\text{pre-del}} \leq 1 \quad (1.1g)$$

$$x_{f,i}^{\text{pre-add}} + x_{f,i}^{\text{maintain}} + x_{f,i}^{\text{pre-del}} \leq x_{f,i-1}^{\text{pre-add}} + x_{f,i-1}^{\text{add}} + x_{f,i-1}^{\text{maintain}} \quad (1.1h)$$

$$x_{f,0}^{\text{add}} = \begin{cases} 1 & \text{falls } f \in s_I \\ 0 & \text{sonst} \end{cases} \quad (1.1i)$$

$$x_{f,t}^{\text{pre-add}} + x_{f,t}^{\text{add}} + x_{f,t}^{\text{maintain}} \geq 1, \quad \forall f \in S_G \quad (1.1j)$$

Die Bedingungen (1.1a) – (1.1e) beschreiben logische Zusammenhänge zwischen Aktionen, ihren Effekten und Vorbedingungen in ihrer IP-Darstellung. Die Ungleichungen (1.1f) und (1.1g) legen fest, dass eine gegebene Formel f entweder mittels positiven Effekts zur Zustandsbeschreibung hinzugefügt oder als negativer Effekt entfernt wird oder durch eine *no-op* Aktion propagiert wird. In (1.1h) wird festgelegt, dass eine Formel im Zeitschritt i nur dann als Vorbedingung einer Aktion dienen bzw. als negativer Effekt oder als Folge einer *no-op* Aktion auftreten kann, wenn im Zeitschritt i auch gilt, d.h. im Zeitschritt $i - 1$ hinzugefügt wurde oder gültig war und nicht entfernt wurde. Die letzten beiden Bedingungen (1.1i) und (1.1j) repräsentieren den Anfangs- bzw. den Zielzustand des als IP-Problem kodierten Planungsproblems. Es ist nun möglich, beliebige lineare Funktionen der verwendeten Variablen zu konstruieren und damit Optimierungsaufgaben auszudrücken. Seien beispielsweise bestimmten Atomformeln f Gewichte (*utilities*) u_f zugeordnet. Dann mag das Optimierungsziel darin bestehen, einen Zielzustand zu finden, in dem die Summe der Gewichte maximal ist, d.h.

$$\sum_f u_f (x_{f,t}^{\text{add}} + x_{f,t}^{\text{pre-add}} + x_{f,t}^{\text{maintain}}) = \max.$$

Es werden allerdings nur Pläne einer begrenzten Länge (hier t) betrachtet. Obwohl der vorgestellte Ansatz somit optimale Lösungen für Pläne begrenzter Länge findet, wird möglicherweise ein echtes Optimum deshalb verpasst, weil kein hinreichend großes t gewählt wurde.

1.4.2 Heuristische Planer – AltAlt^{PS} und Sapa^{PS}

Ein Ansatz ganz anderer Natur als der Zugang mittels *Integer Programming* stellen die in [14] bzw. [30] vorgestellten Systeme AltAlt^{PS} bzw. Sapa^{PS} dar. Diese System wurden entwickelt, um folgendes Problem zu lösen:

Definition 8 (Gesamtnutzenproblem – *Net benefit problem*). Sei ein STRIPS Planungsproblem $\Gamma = (\mathcal{P}(F), A, \gamma, s_I, S_G)$ gegeben, eine Kostenfunktion $c : A \rightarrow \mathbb{Q}^{\geq}$, die jeder Aktion a ihre „Ausführungskosten“ zuordnet, eine Menge $G \subseteq F$ von Formeln und eine Nutzenfunktion $u : G \rightarrow \mathbb{Q}^{\geq}$, welche jeder in G enthaltenen Formel ihren „Nutzen“ zuordnet. Sei weiterhin $k \in \mathbb{Q}^+$ gegeben. Das 5-Tupel (Γ, G, c, u, k) heißt Gesamtnutzenproblem.

Definition 9 (Lösung des Gesamtnutzenproblems). Ein linear geordnete, endliche Sequenz von Aktionen¹⁷ $\Delta = \langle a_1, \dots, a_n \rangle$ heißt Lösung des Gesamtnutzenproblems (Γ, G, c, u, k) , falls Δ Lösung des STRIPS Planungsproblems $\Gamma = (\mathcal{P}(F), A, \gamma, s_I, S_G)$ ist und für den erreichten Zielzustand $s = \gamma^*(s_I, \Delta)$ gilt¹⁸:

$$k \leq \sum_{f \in (G \cap s)} u(f) - \sum_{a \in \Delta} c(a).$$

Die rechte Seite dieser Ungleichung bezeichnet den Gesamtnutzen.

Oft wird eine optimale bzw. maximale Lösung des Gesamtnutzenproblems gesucht, d.h. eine Lösung, so dass k maximal ist.

AltAlt^{PS} ist ein heuristischer Regressionsplaner, der Ideen des Planens mit Plangraphen [26] mit einer heuristischen Zustandsraumsuche (z.B. [31]) kombiniert. Speziell besteht er aus folgenden Schritten:

- AltAlt^{PS} berechnet, ausgehend von der leeren Menge, iterativ eine Menge von Formeln $G' \subseteq G$, die möglicherweise einen optimalen Gesamtnutzen besitzt. Sei also G' gegeben, dann wird nun für jede Zielformel $g \in G \setminus G'$ ein vereinfachtes Planungsproblem, das die Zielmenge $G' \cup \{g\}$ erfüllt, gelöst und daraus abgeschätzt, welche zusätzlichen Kosten die Hinzunahme der Zielformel g bedingt. Das vereinfachte Planungsproblem entsteht aus dem ursprünglichen Problem durch das Nichtberücksichtigen von negativen Effekten der Aktionen. Für jede Formel g wird dann der Gesamtnutzen als Differenz von Nutzen $u(g)$ und geschätzten Kosten berechnet. Falls keine Formel mit positiven Gesamtnutzen existiert, so bricht der Algorithmus ab und G' wird zurückgegeben. Ansonsten wird die Formel, die den höchsten Gesamtnutzen hat, zu G' hinzugefügt und eine neue Iteration gestartet.
- Für die so gefundene Menge G' wird nun ein (nicht vereinfachter) Plan gesucht, der zudem möglichst geringe Aktionskosten aufweisen sollte, um den Gesamtnutzen weiter zu verbessern. Dabei wird eine Regressionssuche (Rückwärtssuche) eingesetzt, beginnend bei G' mit dem Ziel, den Startzustand zu erreichen. Die Suche wird durch die Verwendung einer Kostenheuristik verbessert, welche mittels Plangraph-Techniken erstellt wurde [32].

Ein Vorteil des Systems AltAlt^{PS} liegt darin, dass, sobald eine Menge G' ausgewählt wurde, lediglich ein klassisches Planungsproblem mit der Zielmenge G' zu lösen ist. Ein Nachteil von AltAlt^{PS} besteht darin, dass das System nicht garantieren kann, optimale Mengen G' zu finden. Der beschriebene Algorithmus ist

¹⁷bzw. von Aktionsmengen, siehe vorherige Fußnote.

¹⁸Die Schreibweise $a \in \Delta$ bedeutet, dass a in der Sequenz Δ enthalten ist.

außerdem kein *anytime* Algorithmus. Der gefundene, endgültige Lösungsvorschlag G' kann im Rahmen des Algorithmus nicht weiter verbessert werden.

Eine etwas andere Herangehensweise verfolgt daher das Planungssystem Sapa^{PS} . Natürlicherweise werden zunächst alle ausführbaren Aktionsfolgen als mögliche Lösungen betrachtet und der Wert eines Planes durch seinen Gesamtnutzen bestimmt. Die Schwierigkeit besteht selbstverständlich darin, in der Menge aller Pläne jenen Plan (jene Pläne) zu finden, der (die) einen maximalen Gesamtnutzen besitzt (besitzen). Sapa^{PS} ist ein lokaler *best-first* Suchalgorithmus, wobei die Effizienz derartiger Algorithmen durch die verwendete heuristische Funktion begrenzt wird. Während der Suche im Zustandsraum wird folgende Nutzenfunktion $f : S \rightarrow \mathbb{Q}$ ausgebeutet, die jedem Zustand s des Zustandsraumes einen Wert zuweist:

$$f(s) = g(s) + wh(s).$$

Die Funktionen $g : S \rightarrow \mathbb{Q}$ bzw. $h : S \rightarrow \mathbb{Q}$ geben im Kontext des Gesamtnutzenproblems an, welcher Gesamtnutzen der Zustand s besitzt bzw. welcher weitere Gesamtnutzen ausgehend von Zustand s noch erreicht werden kann. Die Funktion h wird als heuristische Funktion bezeichnet. Der Faktor w bestimmt, wie stark die heuristische Funktion die Suche beeinflusst. Unter gewissen, strengen Voraussetzungen lässt sich zeigen, dass ein derartiger Algorithmus stets eine optimale Lösung findet, wenn eine solche existiert. Die Funktion g lässt sich in einem Vorwärtsplanungsalgorithmus leicht aus s und derjenigen Aktionsfolge, die zum Zustand s führte, berechnen. Die heuristische Funktion wird in Sapa^{PS} durch das Lösen vereinfachter Planungsprobleme, d.h. bei Nichtberücksichtigung von negativen Effekten, abgeschätzt. Die Heuristik verletzt allerdings gewisse Zulässigkeitseigenschaften, weshalb Sapa^{PS} nicht garantieren kann, Zustände mit maximalen Gesamtnutzen auch zu finden. Dafür kann Sapa^{PS} als *anytime* Algorithmus eingesetzt werden und bereits nach kurzer Zeit brauchbare Lösungsvorschläge ausgeben.

Das Gesamtnutzenproblem kann als partielles Erfüllbarkeitsproblem, vgl. Abschnitt 1.2, aufgefasst werden, wenn Aktionen keine Kosten verursachen, d.h. $c \equiv 0$ gilt. Obwohl dies ein Grenzfall des Gesamtnutzenproblems ist, erscheint es aufgrund folgender Überlegungen nicht aussichtsreich, die Algorithmen $\text{AltAlt}^{\text{PS}}$ und Sapa^{PS} zum Lösen von partiellen Erfüllbarkeitsproblemen einzusetzen:

- Als Lösung eines partiellen Erfüllbarkeitsproblems ist stets eine optimale Lösung, im Sinne des Gesamtnutzenproblems, gesucht. Beide Algorithmen garantieren nicht, eine optimale Lösung auch zu finden.
- Sofern jeder Zustand des Zustandsraumes von jedem anderen erreicht werden kann, ist die bestmögliche heuristische Funktion eine konstante Funktion. Von jedem Zustand aus ist es dann nämlich möglich, in den Zustand mit maximalem Nutzen zu gelangen. Die von Sapa^{PS} verwendete heuristische Funktion kann aber per definitionem nicht genauer schätzen. Somit ist fraglich, ob eine Suche effizient durchgeführt werden kann. In partiellen Erfüllbarkeitsproblemen kommt es eben nicht darauf an, eine Menge von Zielen möglichst schnell zu erfüllen, sondern es ist zu klären, ob die Menge von Zielen überhaupt erfüllt werden kann.

1.4.3 Unterscheidung vom Ansatz dieser Arbeit

Der Ansatz dieser Arbeit besitzt im Vergleich zu den eben vorgestellten Zugängen folgende Eigenschaften.

- In der in dieser Arbeit verwendeten Definition eines partiellen Erfüllbarkeitsproblems spielen Aktionskosten keine Rolle.
- Die Präferenzierung von Zielzuständen wird formal untersucht und nicht per se numerisch ausgedrückt. In Kapitel 3 werden wir allerdings Präferenzen linearisieren und zu ihrer Beschreibung natürliche Zahlen verwenden.
- Zur Lösung von partiellen Erfüllbarkeitsproblemen wird ein einfacher Algorithmus vorgestellt, welcher auf die Lösung klassischer Planungsprobleme zurückgreift. Dies geschieht allerdings auf ganz andere Weise als im System `AltAltPS`.
- Die Übersetzung in das klassische Planungsproblem erfolgt unter Verwendung numerischer Variablen, um anzuzeigen, ob eine Atomformel gilt oder nicht. Dies ähnelt dem in Abschnitt 1.4.1 präsentierten, IP-basierten Ansatz. Ebenfalls wird eine numerische Zielfunktion verwendet, die es zu maximieren gilt.
- Zur eigentlichen Plansuche kann jeder Planer eingesetzt werden, der den Sprachumfang von `PDDL2.1` beherrscht. Das Lösen von partiellen Erfüllbarkeitsproblemen wird somit durch Weiterentwicklungen der Planer für klassische Planungsprobleme beschleunigt. `AltAltPS` und `SapaPS` verwenden hingegen fest implementierte Heuristiken und Suchverfahren.

Kapitel 2

Wissensbasen und Präferenzen

In diesem Kapitel sollen Möglichkeiten vorgestellt werden, komplexe Präferenzausdrücke zu beschreiben. Der erste Abschnitt folgt dabei der Darstellung aus [33]. Im nachfolgenden Abschnitt wird ein alternativer Ansatz vorgestellt. Anschließend werden beide Möglichkeiten verglichen.

2.1 Präferenzausdrücke auf gewichteten Wissensbasen

Präferenzausdrücke spielen in vielen Gebieten der künstlichen Intelligenz eine wichtige Rolle. So z.B. in der logischen Programmierung [4] oder bei der Lösung von beschränkten Erfüllbarkeitsproblemen (*constraint satisfaction problems*). Auch im Bereich des Automatischen Planens besitzen Präferenzausdrücke eine große Bedeutung. So werden bei der Darstellung von Planungsproblemen Ziele oft durch Formelmengen S_G beschrieben. Da es oftmals nicht möglich ist, alle Zielformeln $f \in S_G$ zu erfüllen, ist man auch an einer Teillösung des Planungsproblems interessiert, bei der lediglich eine (in einem zu spezifizierenden Sinne) optimale Teilmenge der Menge S_G erfüllt wird. Zunächst soll nun der konzeptionelle Rahmen für die Behandlung von Präferenzausdrücken eingeführt werden. Anschließend wird gezeigt, wie Präferenzen mithilfe von gewichteten Wissensbasen ausgedrückt werden können.

2.1.1 Gewichtete Wissensbasen

Sei Hintergrundwissen in Form einer Menge von aussagenlogischen Formeln B gegeben. Wir betrachten im Folgenden Modelle von B , d.h. eine Interpretation, die jeder Formel aus B den booleschen Wert *wahr* zuordnet. Sei nun eine weitere aussagenlogische Formel f gegeben. Ein Modell¹ m heißt genau dann bevorzugt gegenüber einem Modell m' bezüglich f , wenn gilt²:

$$m' \models f \rightarrow m \models f. \tag{2.1}$$

¹Ein Modell in Bezug auf das Hintergrundwissen B wollen wir im Weiteren kurz als Modell bezeichnen.

²Hierbei bedeutet $m \models f$, dass die Formel f im Modell m erfüllt ist.

Sei zusätzlich zum Hintergrundwissen B eine Menge F von aussagenlogischen Formeln gegeben. Da sich meist kein Modell finden lässt, welches alle Formeln aus F erfüllt, ist es sinnvoll eine Präferenzrelation \succeq über den Modellen anzugeben. Ein Modell m heißt bevorzugt gegenüber einem Modell m' bzw. besser als ein Modell m' bezüglich der Präferenzrelation \succeq , wenn $m \succeq m'$ gilt³. Die Optimierungsaufgabe besteht jetzt darin, ein, mehrere oder auch alle optimalen Modelle bezüglich der Präferenzrelation zu finden. Dabei sei ein Modell m optimal, wenn es maximal bezüglich der gegebenen Präferenzrelation ist, d.h. wenn für kein Modell m' gilt:

$$m' \succ m.$$

Abgeleitet von der Präferenzrelation \succeq , werden die strikte Präferenzrelation \succ und die Gleichheit $=$ bzgl. der Präferenzierung zwischen zwei Modellen m und m' wie folgt definiert:

$$m \succ m' \text{ gdw. } m \succeq m' \text{ und nicht } m' \succeq m \quad (2.2)$$

$$m = m' \text{ gdw. } m \succeq m' \text{ und } m' \succeq m. \quad (2.3)$$

Im weiteren Verlauf dieses Kapitels werden folgende kleine Lemmata benötigt. Sie werden daher an dieser Stelle eingeführt und bewiesen.

Sei \succeq eine reflexive und transitive Relation, dann ist

- die strikte Relation \succ transitiv.
- die abgeleitete Gleichheitsrelation $=$ transitiv.

Beweis. Zum ersten Punkt: Aus $m_1 \succ m_2$ und $m_2 \succ m_3$ folgt $m_1 \succeq m_3$ aufgrund der Transitivität von \succeq . Allerdings ist $m_1 = m_3$ unvereinbar mit $(m_1 \succ m_2 \text{ und } m_2 \succ m_3)$, denn aus $m_1 = m_3$ und $m_1 \succ m_2$ folgt $m_3 \succeq m_1$ und $m_1 \succeq m_2$. Dies impliziert $m_3 \succeq m_2$ im Widerspruch zu $m_2 \succ m_3$.

Zum zweiten Punkt: Aus $m_1 = m_2$ und $m_2 = m_3$ folgt: $m_1 \succeq m_2$ und $m_2 \succeq m_1$ und $m_2 \succeq m_3$ und $m_3 \succeq m_2$. Daher gilt auch $m_1 \succeq m_3$ und $m_3 \succeq m_1$, woraus $m_1 = m_3$ folgt. \square

In diesem Abschnitt wird die Präferenzrelation mithilfe von gewichteten Wissensbasen eingeführt. Derartige Wissensbasen haben sich in verschiedenen Gebieten als nützlich erwiesen. Kurzgefasst wird jeder Formel aus F noch ein zusätzliches Gewicht zugeordnet, welches die relative Bedeutung der Formel angibt. Eine Darstellung der Gewichte durch natürliche Zahlen ist üblich. Es ist möglich, zwischen quantitativen Präferenzrelationen, welche die Werte der Gewichte explizit verwenden, und qualitativen Präferenzrelationen, welche nur die implizite Ordnungsstruktur der Gewichte ausnutzen, zu unterscheiden. Relationen des ersteren Typs werden häufig in beschränkten Erfüllbarkeitsproblemen eingesetzt. Die Gewichte werden dort als Belohnungen interpretiert und das Ziel besteht darin, die Gesamtbelohnung zu maximieren. Eine Zusammenfassung quantitativer Ansätze findet sich z.B. in [34]. Der Darstellung in [33] folgend beschränken wir uns an dieser Stelle auf qualitative Präferenzrelationen und verweisen auf die weiterführende Diskussion in Abschnitt 2.3 und die in Kapitel 3 vorgestellte numerische Beschreibungssprache NLPD.

³In dieser Arbeit wird sowohl die Notation $m \succeq m'$ als auch die Schreibweise $(m, m') \in \succeq$ verwendet. Beide bedeuten, dass m zu m' in Relation \succeq steht.

Eine gewichtete Wissensbasis (RKB) ist eine Paar (F, \geq) bestehend aus einer Menge von aussagenlogischen Formeln F zusammen mit einer 2-stelligen linearen Quasiordnung \geq über F . Eine Quasiordnung ist eine transitive und reflexive Relation. In einer linearen Quasiordnung über F gilt für jedes Paar $(f_1, f_2) \in F \times F$: $f_1 \geq f_2$ oder $f_2 \geq f_1$, d.h. zwei beliebige Elemente sind stets vergleichbar. Eine RKB kann nun auf folgende Weisen dargestellt werden:

- Als Sequenz (F_1, \dots, F_n) von Formelmengen F_i , so dass für beliebige Formeln $f^i \in F_i$, $f^j \in F_j$ gilt: $i \geq j$ gdw. $f^i \geq f^j$.
- Als Menge von Paaren der Form (f_k, r_k) , wobei r_k den Rang (das Gewicht) der entsprechenden Formel f_k angibt. Dabei gilt: $r_i \geq r_j$ gdw. $f_i \geq f_j$.

Beide Darstellungen sind offensichtlich äquivalent. Der Rang r_k entspricht gerade dem Index der Formelmengruppe F_{r_k} . Welche Darstellungsmöglichkeit gerade verwendet wird, erschließt sich aus dem Kontext. Weiterhin bezeichnet f_k^i die k -te Formel aus der Menge F_i , während die Angabe f_k die Formelkomponente eines Paares (f_k, r_k) spezifiziert. Sei eine RKB (F, \geq) gegeben und das Problem, eine im zu spezifizierenden Sinne optimale Teilmenge von F zu finden. Intuitiv sollten dabei Formeln mit höherem Rang bedeutender sein als Formeln von niedrigerem Rang. Sofern nur nicht-negative Gewichte vorkommen, sollte zudem folgende Aussage für die Präferenzrelation gelten. Seien P bzw. P' in F maximal erfüllbare Teilmengen⁴ bzgl. der Modelle m bzw. m' , dann gilt: $P \supseteq P'$ impliziert $m \succeq m'$. Diese Monotonie-Aussage beschreibt, dass es immer günstiger sein sollte, mehr Formeln zu erfüllen. In der später definierten Präferenzbeschreibungssprache LPD gilt die Monotonie-Aussage aufgrund der Existenz des „-“ Operators jedoch nicht.

2.1.2 Basis-Präferenzausdrücke

Trotz dieser Einschränkungen ist die Präferenzstrategie durch die Angabe der RKB allein nicht festgelegt. In [33] erfolgt die Festlegung der Präferenzrelation durch Präferenzausdrücke. Dabei werden zunächst Basis-Präferenzausdrücke eingeführt und dann, darauf aufbauend, komplexere Ausdrücke definiert.

Definition 10. Ein Basis-Präferenzausdruck K^s ist ein Paar (s, K) bestehend aus einem Basis-Identifikator Symbol $s \in \{\top, \kappa, \supseteq, \#\}$ und einer gewichteten Wissensbasis K .

Um die Präferenzrelation zu spezifizieren sind folgende Hilfskonstrukte nützlich:

$$K^i(m) = \{f \mid (f, i) \in K, m \models f\}$$

$$\maxsat^K(m) = \begin{cases} -\infty & \text{falls } \forall (f, r) \in K : m \not\models f \\ \max\{r \mid (f, r) \in K, m \models f\} & \text{sonst} \end{cases}$$

$$\maxunsat^K(m) = \begin{cases} -\infty & \text{falls } \forall (f, r) \in K : m \models f \\ \max\{r \mid (f, r) \in K, m \not\models f\} & \text{sonst} \end{cases}$$

$K^i(m)$ definiert diejenige Menge von Formeln aus F_i , die im Modell m erfüllt sind.

⁴Eine Menge $A \subseteq B$ heie in B maximal erfüllbar bzgl. m gdw. $m \models A$ und für keine in B liegende Obermenge C von A $m \models C$.

Definition 11. Die Basis-Präferenzausdrücke K^\top , K^κ , K^\supseteq und $K^\#$ spezifizieren die entsprechenden Basis-Präferenzrelationen \succeq_\top , \succeq_κ , \succeq_\supseteq und $\succeq_\#$.

$$m \succeq_\top m' \text{ gdw. } \max\text{sat}^K(m) \geq \max\text{sat}^K(m'),$$

$$m \succeq_\kappa m' \text{ gdw. } \max\text{unsat}^K(m') \geq \max\text{unsat}^K(m),$$

$$m \succeq_\supseteq m' \text{ gdw. } \forall i (K^i(m) = K^i(m')) \text{ oder}$$

$$(\exists i (K^i(m) \supset K^i(m'))) \text{ und } \forall j > i : (K^j(m) = K^j(m')),$$

$$m \succeq_\# m' \text{ gdw. } \forall i (\text{card } K^i(m) = \text{card } K^i(m')) \text{ oder}$$

$$(\exists i (\text{card } K^i(m) > \text{card } K^i(m'))) \text{ und } \forall j > i : (\text{card } K^j(m) = \text{card } K^j(m')).$$

Im Gegensatz zur Relation \succeq_\supseteq sind die Relationen $\succeq_\#$, \succeq_κ , \succeq_\top linear. Die Präferenzrelation \succeq_\top wurde im Kontext bipolarer Repräsentationen eingesetzt [35]. Die Relation \succeq_κ kam im System Z zum Einsatz [36, 37] und auf die Präferenzrelationen \succeq_\supseteq bzw. $\succeq_\#$ wurde in [38] bzw. [39] zurückgegriffen. Die Basis-Präferenzrelationen sind nicht unabhängig voneinander. Nach [33] gilt

Satz 1. Seien m und m' Modelle und K eine gewichtete Wissensbasis, dann gelten u.a. die folgenden Meta-Relationen zwischen den aus den Basis-Präferenzrelation abgeleiteten strikten Präferenzrelationen.

$$m \succ_\top m' \text{ impliziert } m \succ_\supseteq m',$$

$$m \succ_\kappa m' \text{ impliziert } m \succ_\supseteq m',$$

$$m \succ_\supseteq m' \text{ impliziert } m \succ_\# m'.$$

Diese Meta-Relationen sind in Abbildung 2.1 illustriert. Die Relation $\succ_\#$ spielt eine besondere Rolle, denn offensichtlich ist sie von den gegebenen strikten Präferenzrelationen die stärkste (d.h. ihre Kardinalität ist größer oder gleich der Kardinalität jeder anderen gegebenen strikten Präferenzrelationen). Zudem lassen sich mit der Relation $\succeq_\#$ auch die Relationen \succeq_κ , \succeq_\top ausdrücken.

Satz 2. Seien m und m' Modelle und K eine gewichtete Wissensbasis. Sei weiterhin

$$K_\wedge = \{(C_i, r) \mid C_i \text{ ist Konjunktion von allen } f \text{ mit } (f, l) \in K, l \geq r\},$$

$$K_\vee = \{(C_i, r) \mid C_i \text{ ist Disjunktion von allen } f \text{ mit } (f, l) \in K, l \geq r\}.$$

Dann gilt (siehe [33])

$$m \succeq_\kappa m' \text{ bzgl. } K \text{ gdw. } m \succeq_\# m' \text{ bzgl. } K_\wedge,$$

$$m \succeq_\top m' \text{ bzgl. } K \text{ gdw. } m \succeq_\# m' \text{ bzgl. } K_\vee.$$

2.1.3 Komplexe Präferenzausdrücke

Nach [33] werden komplexe Präferenzausdrücke iterativ aus den Basis-Präferenzausdrücken aufgebaut.

Definition 12. Die Syntax der Sprache LPD (*L*anguage for *P*reference *D*escriptions) ist induktiv wie folgt definiert:

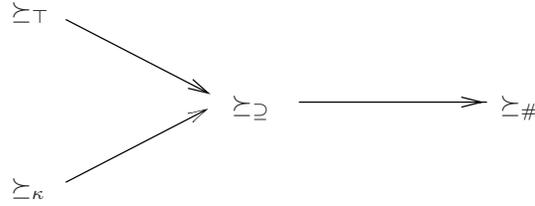


Abbildung 2.1: Meta-Relationen zwischen den strikten Präferenzrelationen \succeq_{\top} , \succeq_{κ} , \succeq_{\supseteq} und $\succeq_{\#}$. Inklusionsbeziehungen zwischen den Relationen sind durch Pfeile dargestellt.

1. Jeder Basis-Präferenzausdruck gehört zu LPD.
2. Seien d und d' Ausdrücke der LPD. Dann gehören auch die Ausdrücke $(d \wedge d')$, $(d \vee d')$, $(d > d')$ und $-d$ zu LPD.

Die durch einen beliebigen Präferenzausdruck definierte Präferenzrelation stellt die Semantik dieses Ausdrucks dar. Sie wird ebenfalls induktiv definiert, wobei die Semantik der Basis-Präferenzausdrücke bereits im letzten Abschnitt erläutert wurde.

Definition 13. Seien \succeq_d und $\succeq_{d'}$ die zu den Ausdrücken d und d' gehörenden Präferenzrelationen. Bezeichne $\text{tr}(\succeq)$ den transitiven Abschluss einer Relation \succeq und bezeichne \succeq^{-1} die zu \succeq inverse Relation. Dann ist die Präferenzrelation eines komplexen LPD-Ausdrucks wie folgt definiert:

- $\succeq_{(d \wedge d')} = \succeq_d \cap \succeq_{d'}$
- $\succeq_{(d \vee d')} = \text{tr}(\succeq_d \cup \succeq_{d'})$
- $\succeq_{-d} = \succeq_d^{-1}$
- $\succeq_{(d > d')} = \succeq_{(d \wedge d')} \cup (\succeq_d \setminus \succeq_d^{-1})$

$\succeq_{(d \wedge d')}$ entspricht der Pareto-Ordnung, d.h. ein Modell m ist mindestens so gut wie ein anderes Modell m' , wenn es mindestens so gut ist bzgl. sowohl d als auch d' . Das Modell m ist strikt besser als das Modell m' , falls es mindestens so gut ist wie m' und bzgl. d oder d' strikt besser. Die zu $(d \vee d')$ gehörende Relation gibt einem Modell m mindestens die gleiche Präferenz wie einem Modell m' , sofern m mindestens so gut ist wie m' bzgl. d oder d' oder es ein Modell m'' gibt, so dass $m \succeq_{(d \vee d')} m''$ und $m'' \succeq_{(d \vee d')} m'$. Man beachte, dass letzterer Teil den transitiven Abschluss der Relation rekursiv beschreibt. Die Verwendung des transitiven Abschlusses ist nötig, da die Vereinigung zweier transitiver Relationen i.A. nicht transitiv ist. \succeq_{-d} führt zur inversen Präferenzrelation und $\succeq_{(d > d')}$ beschreibt die lexikographische Ordnung der Relationen \succeq_d und $\succeq_{d'}$.

Ein Beispiel soll die Verwendung der Präferenzbeschreibungssprache LPD veranschaulichen. Ein Student suche nach einer passenden Promotionsstelle im Fachgebiet Informatik. Gleichmaßen wichtig erscheinen ihm das Promotionsthema und das Arbeitsumfeld (Betreuung und Bezahlung). Weiterhin habe eine

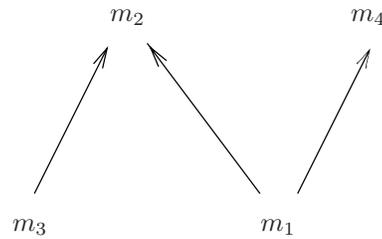


Abbildung 2.2: Anwendungsbeispiel der Präferenzbeschreibungssprache LPD.

intensive Betreuung Vorrang vor einer guten Bezahlung. Es seien daher folgende Wissensbasen gegeben:

$$\begin{aligned}
 K_{\text{Thema}} &= (\{\text{Bildverarbeitung, Robotik}\}, \{\text{Planung}\}), \\
 K_{\text{Betreuung}} &= (\{\text{Gruppenseminar, Konferenz}\}, \{\text{intensive_Betreuung, gutes_Arbeitsklima}\}), \\
 K_{\text{Bezahlung}} &= (\{\text{halbe_Stelle}\}, \{\text{ganze_Stelle}\}).
 \end{aligned}$$

Obige Präferenzierung lässt sich nun durch folgenden LPD Ausdruck wiedergeben

$$K_{\text{Thema}}^{\top} \wedge (K_{\text{Betreuung}}^{\#} > K_{\text{Bezahlung}}^{\top}).$$

Eine Recherche des Studenten ergab folgende Bewerbungsmöglichkeiten:

*Stelle*₁ : Bildverarbeitung, Gruppenseminar, gutes_Arbeitsklima, halbe_Stelle

*Stelle*₂ : Planung, Gruppenseminar, intensive_Betreuung, halbe_Stelle

*Stelle*₃ : Planung, Konferenz, ganze_Stelle

*Stelle*₄ : Robotik, Gruppenseminar, intensive_Betreuung, gutes_Arbeitsklima

Die Stellenbeschreibungen *Stelle*₁, ..., *Stelle*₄ definieren die entsprechenden zu betrachtenden Modelle⁵ m_1, \dots, m_4 .

m_2 besitzt gegenüber m_1 das bessere Thema. Das Arbeitsumfeld ist in beiden Modellen gleich gut, daher ist m_2 strikt besser als m_1 . Das Modell m_2 wird auch gegenüber m_3 bevorzugt, da es eine bessere Betreuung verspricht. Aus denselben Gründen wird m_4 gegenüber m_1 bevorzugt, obwohl keine Bezahlung zugesichert werden kann. m_1 und m_3 sind nicht vergleichbar, da m_1 die bessere Betreuung aufweist und m_3 die bevorzugte Themenwahl. Aus ähnlichen Gründen sind m_4 und m_2 bzw. m_4 und m_3 nicht vergleichbar. In Abbildung 2.2 sind die strengen Präferenzrelationen zwischen den Modellen m_1, \dots, m_4 durch Pfeile dargestellt. Es ist ersichtlich, dass es in diesem Beispiel zwei optimale Modelle gibt: m_2 und m_4 .

2.2 Alternative Darstellung von Präferenzausdrücken

Die Verwendung von gewichteten Wissensbasen für die Beschreibung komplexer Präferenzen wurde im letzten Abschnitt dargelegt. Hier soll nun die Präferenzbeschreibungssprache LPD* vorgestellt werden,

⁵Die Menge des Hintergrundwissens B ist in diesem Beispiel leer. Damit stellt jede Σ -Struktur ein *Modell* bzgl. B dar.

Für, ist $m_1 \succeq_n m_2$ gdw. ...
$n = (\text{LEV } n_1 \dots n_k)$	$\forall j, 1 \leq j \leq k (m_1 =_{n_j} m_2)$ oder $\exists i, 1 \leq i \leq k (m_1 \succ_{n_i} m_2 \text{ und } \forall j, i < j \leq k (m_1 =_{n_j} m_2))$
$n = (\text{CAR } n_1 \dots n_k)$	$\text{card}\{i \mid m_1 \succeq_{n_i} m_2\} \geq \text{card}\{i \mid m_2 \succeq_{n_i} m_1\}$
$n = (\text{ALL } n_1 \dots n_k)$	$\forall i, 1 \leq i \leq k (m_1 \succeq_{n_i} m_2)$
$n = (\text{EX } n_1 \dots n_k)$	$\exists i, 1 \leq i \leq k (m_1 \succeq_{n_i} m_2)$ oder $\exists m (m_1 \succeq_n m \succeq_n m_2)$
$n = (\text{INV } n_1)$	$m_2 \succeq_{n_1} m_1$
$n = f, f$ aussagenlog. Formel	$m_2 \models f \rightarrow m_1 \models f$

Tabelle 2.1: Definition der Semantik der Sprache LPD*

welche sich zum einen an der oben definierten Sprache LPD orientiert, zum anderen jedoch auf die Verwendung von gewichteten Wissensbasen verzichtet. Nach Einführung von Syntax und Semantik wird gezeigt werden, dass sich LPD Ausdrücke in LPD* äquivalent darstellen lassen. Anschließend wird der Sprachumfang von LPD* einer Analyse unterzogen, wobei insbesondere die Ergebnisse aus [40] einfließen. Es wird sich herausstellen, dass bereits eine Untersprache von LPD* zur Beschreibung von Präferenzen ausreicht, sofern bestimmte Forderungen an die zu beschreibenden Präferenzrelationen gestellt werden. Ein Vergleich zwischen der Präferenzbeschreibung mittels Wissensbasen und dem alternativen Ansatz dieses Abschnittes findet sich im nachfolgenden Abschnitt 2.3.

2.2.1 Die Sprache LPD*

Die Syntax der Sprache LPD* wird nun in Backus-Naur-Form (BNF) beschrieben. Die Elemente der Sprache sind dann von folgender Form:

```

<node> ::= (LEV <node>+)
<node> ::= (CAR <node>+)
<node> ::= (ALL <node>+)
<node> ::= (EX <node>+)
<node> ::= (INV <node>)
<node> ::= <formula>

```

Das Nichtterminalsymbol <formula> expandiert zu einer gewöhnlichen aussagenlogischen Formel aus F , d.h. einer Verknüpfung von Atomformeln mittels der aussagenlogischen Operatoren \wedge , \vee , \neg , \rightarrow , \leftrightarrow . Die durch Sprachelemente von LPD* beschriebenen Präferenzrelationen stellen die Semantik dieser Sprache dar. In Tabelle 2.1 wird die Semantik induktiv definiert.

Definition 14. Ein Ausdruck n einer Beschreibungssprache L und ein Ausdruck n' einer Beschreibungssprache L' heißen äquivalent, wenn sie dieselbe Relation beschreiben, d.h. genau dann, wenn für eine feste Menge M von Modellen und beliebige Modelle $m_1, m_2 \in M$ gilt:

$$m_1 \succeq_n m_2 \text{ bzgl. der } L \text{ Semantik gdw. } m_1 \succeq_{n'} m_2 \text{ bzgl. der } L' \text{ Semantik.}$$

Es soll nun gezeigt werden, dass für jeden LPD Ausdruck ein äquivalenter LPD* Ausdruck existiert. In einem ersten Schritt wird dazu die $\succeq_{\#}$ Relation mittels LPD* Konstrukten nachgebildet.

Satz 3. Sei $K = (\{f_1^1, \dots, f_{l_1}^1\}, \dots, \{f_1^s, \dots, f_{l_s}^s\})$ eine gewichtete Wissensbasis. Sei weiterhin $n = (\text{LEV } n_1 \dots n_s)$ mit $n_j = (\text{CAR } f_1^j \dots f_{l_j}^j)$ für alle $1 \leq j \leq s$ ein LPD* Ausdruck über der gleichen Formelmenge. Dann sind $K^\#$ und n äquivalente Ausdrücke.

Dieser Satz lässt sich mit folgendem Hilfssatz leicht beweisen.

Satz 4 (Hilfssatz). Die LPD und LPD* Ausdrücke seien wie eben definiert. Dann gilt unter Verwendung der Hilfskonstrukte aus Abschnitt 2.1.2:

$$\text{card } K^j(m_1) = \text{card } K^j(m_2) \text{ gdw. } m_1 =_{n_j} m_2.$$

Beweis.

$$\begin{aligned} m_1 =_{n_j} m_2 \text{ gdw. } & \text{card } \{f_k^j \mid m_1 \succeq_{f_k^j} m_2\} = \text{card } \{f_k^j \mid m_2 \succeq_{f_k^j} m_1\} \\ & \text{gdw. } \text{card } \{f_k^j \mid m_1 \succ_{f_k^j} m_2\} = \text{card } \{f_k^j \mid m_2 \succ_{f_k^j} m_1\} \\ & \text{gdw. } \text{card } \{f_k^j \mid m_1 \models f_k^j \wedge m_2 \not\models f_k^j\} = \text{card } \{f_k^j \mid m_2 \models f_k^j \wedge m_1 \not\models f_k^j\} \\ & \text{gdw. } \text{card } \{f_k^j \mid m_1 \models f_k^j\} = \text{card } \{f_k^j \mid m_2 \models f_k^j\} \\ & \text{gdw. } \text{card } K^j(m_1) = \text{card } K^j(m_2) \end{aligned}$$

Der erste Schritt des Beweises verwendet die Definition des CAR Konstrukts. Die nächste Umformung benutzt die Identität

$$\text{card } \{f \mid m_1 \succeq_f m_2\} = \text{card } \{f \mid m_1 =_f m_2\} + \text{card } \{f \mid m_1 \succ_f m_2\}.$$

Im dritten Schritt wurde $m_1 \succ_f m_2$ gdw. $m_1 \models f \wedge m_2 \not\models f$ verwendet, wie es aus (2.1) und (2.2) folgt. Anschließend wurde die folgende Umformung benutzt

$$\{f \mid m_1 \models f \wedge m_2 \not\models f\} = \{f \mid m_1 \models f\} \setminus \{f \mid m_1 \models f \wedge m_2 \models f\}$$

und ausgenutzt, dass $\text{card}(A \setminus B) = \text{card } A - \text{card } B$ gilt, falls $B \subseteq A$ und alle Mengen endlich sind. \square

Bemerkung. Sei n_j wie eben gegeben, dann gilt auch

$$\begin{aligned} \text{card } K^j(m_1) &\geq \text{card } K^j(m_2) \text{ gdw. } m_1 \succeq_{n_j} m_2 \\ \text{card } K^j(m_1) &> \text{card } K^j(m_2) \text{ gdw. } m_1 \succ_{n_j} m_2. \end{aligned}$$

Der Beweis erfolgt analog zum Beweis des Hilfssatzes.

Beweis von Satz 3.

$$\begin{aligned} m_1 \succeq_{\#} m_2 \quad \text{gdw.} \quad & \forall j, 1 \leq j \leq s (\text{card } K^j(m_1) = \text{card } K^j(m_2)) \text{ oder} \\ & \exists i, 1 \leq i \leq s (\text{card } K^i(m_1) > \text{card } K^i(m_2) \text{ und} \\ & \quad \forall j, i < j \leq s (\text{card } K^j(m_1) = \text{card } K^j(m_2))) \\ \text{gdw.} \quad & \forall j, 1 \leq j \leq s (m_1 =_{n_j} m_2) \text{ oder} \\ & \exists i, 1 \leq i \leq s (m_1 \succ_{n_i} m_2 \text{ und } \forall j, i < j \leq s (m_1 =_{n_j} m_2)) \\ \text{gdw.} \quad & m_1 \succeq_n m_2 \end{aligned}$$

\square

Satz 5. Sei $K = (\{f_1^1, \dots, f_{l_1}^1\}, \dots, \{f_1^s, \dots, f_{l_s}^s\})$ eine gewichtete Wissensbasis. Sei weiterhin $n = (\text{LEV} (\text{ALL} f_1^1 \dots f_{l_1}^1) \dots (\text{ALL} f_1^s \dots f_{l_s}^s))$ ein LPD* Ausdruck über der gleichen Formelmenge. Dann gilt: $K \succeq$ und n sind äquivalente Ausdrücke.

Beweis. Der Beweis wird analog zum Beweis des Satzes 3 durchgeführt. Zunächst wird ein entsprechender Hilfssatz gezeigt. Dieser entspricht obigem Hilfssatz, wobei anstelle von $n_j = (\text{CAR} f_1^j \dots f_{l_j}^j)$ der Ausdruck $n_j = (\text{ALL} f_1^j \dots f_{l_j}^j)$ verwendet wird und anstelle eines Vergleiches der Kardinalität die entsprechenden Mengen auf Mengengleichheit getestet werden. Man beachte, dass alle betrachteten Formelmengen als endlich angenommen werden, da alle syntaktischen Ausdrücke endliche Längen besitzen. \square

Satz 6. Sei $K = (\{f_1^1, \dots, f_{l_1}^1\}, \dots, \{f_1^s, \dots, f_{l_s}^s\})$ eine gewichtete Wissensbasis. Seien außerdem K_\wedge und K_\vee wie in Satz 2 definiert, d.h. $K_\wedge = (\{g_\wedge^1\}, \dots, \{g_\wedge^s\})$ bzw. $K_\vee = (\{g_\vee^1\}, \dots, \{g_\vee^s\})$ mit $g_\wedge^i = \bigwedge_{j=i}^s \bigwedge_{k=1}^{l_j} f_k^j$ bzw. $g_\vee^i = \bigvee_{j=i}^s \bigvee_{k=1}^{l_j} f_k^j$. Seien weiterhin $n_\wedge = (\text{LEV} g_\wedge^1 \dots g_\wedge^s)$ bzw. $n_\vee = (\text{LEV} g_\vee^1 \dots g_\vee^s)$ LPD* Ausdrücke, dann gilt:

1. K^κ und n_\wedge sind äquivalente Ausdrücke.
2. K^\top und n_\vee sind äquivalente Ausdrücke.

Beweis. Zunächst wird gezeigt, dass die LPD* Ausdrücke $n' = (\text{CAR} f)$ und $n = f$ für Formeln f dieselbe Präferenzrelation beschreiben, d.h. für beliebige Modelle m_1 und m_2 gilt:

$$m_1 \succeq_{n'} m_2, \text{ gdw. } m_1 \succeq_n m_2.$$

Dies wird wie folgt gezeigt:

$$\begin{aligned} m_1 \succeq_{n'} m_2, \text{ gdw. } & m_1 \succeq_f m_2 \text{ oder nicht } m_2 \succeq_f m_1, \\ & \text{gdw. } (m_2 \models f \rightarrow m_1 \models f) \text{ oder nicht } (m_1 \models f \rightarrow m_2 \models f), \\ & \text{gdw. } (m_2 \not\models f \vee m_1 \models f) \text{ oder } (m_1 \models f \wedge m_2 \not\models f), \\ & \text{gdw. } m_2 \not\models f \vee m_1 \models f, \\ & \text{gdw. } m_1 \succeq_n m_2. \end{aligned}$$

Man beachte, dass i.A. zwei Ausdrücke $n' = (\text{CAR} n_1)$ und $n = n_1$ nicht dieselbe Präferenzordnung beschreiben. Denn seien zwei Modelle m_1 und m_2 bzgl. n nicht vergleichbar⁶, d.h. gelte weder $m_1 \succeq_n m_2$ noch $m_2 \succeq_n m_1$, so sind sie jedoch per Definition vergleichbar bzgl. n' , wobei dann sogar gilt $m_1 =_{n'} m_2$.

Seien nun die Ausdrücke n'_\wedge und n'_\vee wie folgt definiert:

$$\begin{aligned} n'_\wedge &= (\text{LEV} (\text{CAR} g_\wedge^1) \dots (\text{CAR} g_\wedge^s)), \\ n'_\vee &= (\text{LEV} (\text{CAR} g_\vee^1) \dots (\text{CAR} g_\vee^s)). \end{aligned}$$

Nach obigen Überlegungen definieren sie dieselbe Präferenzrelation wie die Ausdrücke n_\wedge bzw. n_\vee . Damit ist der LPD* Ausdruck n_\wedge äquivalent zum LPD Ausdruck $K_\wedge^\#$ und nach Satz 2 auch äquivalent zu K^κ . Analog ist der Ausdruck n_\vee äquivalent zum LPD Ausdruck $K_\vee^\#$ und somit auch äquivalent zu K^\top . \square

⁶Bezüglich einer Formel sind zwei Modelle stets vergleichbar.

Es wurde nun gezeigt, dass alle Basispräferenzausdrücke aus LPD eine Entsprechung in LPD* besitzen. Die Frage, ob sich auch komplexe Präferenzausdrücke in LPD* darstellen lassen, wird im Folgenden Satz beantwortet.

Satz 7. *Seien der LPD Ausdruck d_1 und der LPD* Ausdruck n_1 , sowie der LPD Ausdruck d_2 und der LPD* Ausdruck n_2 jeweils äquivalent. Dann gilt:*

$$\begin{aligned} d_1 > d_2 &\text{ ist äquivalent zu } (\mathbf{LEV} \ n_2 \ n_1), \\ d_1 \wedge d_2 &\text{ ist äquivalent zu } (\mathbf{ALL} \ n_1 \ n_2), \\ d_1 \vee d_2 &\text{ ist äquivalent zu } (\mathbf{EX} \ n_1 \ n_2), \\ -d_1 &\text{ ist äquivalent zu } (\mathbf{INV} \ n_1). \end{aligned}$$

Beweis. Sei $n = (\mathbf{LEV} \ n_2 \ n_1)$. Dann gilt $(m_1, m_2) \in \succeq_{d_1 > d_2}$ nach Definition 13

$$\text{gdw. } (m_1, m_2) \in (\succeq_{d_1 \wedge d_2}) \cup (\succeq_{d_1} \setminus \succeq_{d_1}^{-1}),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_{d_1 \wedge d_2} \text{ oder } (m_1, m_2) \in \succ_{d_1},$$

$$\text{gdw. } ((m_1, m_2) \in =_{d_1} \text{ und } (m_1, m_2) \in \succeq_{d_2}) \text{ oder } (m_1, m_2) \in \succ_{d_1},$$

$$\text{gdw. } \forall i, 1 \leq i \leq 2 ((m_1, m_2) \in =_{d_i}) \text{ oder } \exists i, 1 \leq i \leq 2 ((m_1, m_2) \in \succ_{d_i} \text{ und } \forall j, 1 \leq j < i ((m_1, m_2) \in =_{d_j})),$$

$$\text{gdw. } \forall i, 1 \leq i \leq 2 ((m_1, m_2) \in =_{n_i}) \text{ oder } \exists i, 1 \leq i \leq 2 ((m_1, m_2) \in \succ_{n_i} \text{ und } \forall j, 1 \leq j < i ((m_1, m_2) \in =_{n_j})),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_n .$$

Sei $n = (\mathbf{ALL} \ n_1 \ n_2)$. Dann gilt $(m_1, m_2) \in \succeq_{d_1 \wedge d_2}$

$$\text{gdw. } (m_1, m_2) \in \succeq_{d_1} \text{ und } (m_1, m_2) \in \succeq_{d_2},$$

$$\text{gdw. } \forall i, 1 \leq i \leq 2 ((m_1, m_2) \in \succeq_{n_i}),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_n .$$

Sei $n = (\mathbf{EX} \ n_1 \ n_2)$. Dann gilt $(m_1, m_2) \in \succeq_{d_1 \vee d_2}$

$$\text{gdw. } (m_1, m_2) \in \succeq_{d_1} \cup \succeq_{d_2} \text{ oder } \exists m ((m_1, m) \in \succeq_{d_1 \vee d_2} \text{ und } (m, m_2) \in \succeq_{d_1 \vee d_2}),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_{d_1} \cup \succeq_{d_2} \text{ oder}$$

$$\exists k > 2 \exists (\tilde{m}_1, \dots, \tilde{m}_k) (\forall j, 2 \leq j \leq k ((\tilde{m}_{j-1}, \tilde{m}_j) \in \succeq_{d_1} \cup \succeq_{d_2}) \text{ und } \tilde{m}_1 = m_1 \text{ und } \tilde{m}_k = m_2),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_{n_1} \cup \succeq_{n_2} \text{ oder}$$

$$\exists k > 2 \exists (\tilde{m}_1, \dots, \tilde{m}_k) (\forall j, 2 \leq j \leq k ((\tilde{m}_{j-1}, \tilde{m}_j) \in \succeq_{n_1} \cup \succeq_{n_2}) \text{ und } \tilde{m}_1 = m_1 \text{ und } \tilde{m}_k = m_2),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_{n_1} \cup \succeq_{n_2} \text{ oder } \exists m ((m_1, m) \in \succeq_n \text{ und } (m, m_2) \in \succeq_n),$$

$$\text{gdw. } (m_1, m_2) \in \succeq_n .$$

Sei $n = (\mathbf{INV} \ n_1)$. Dann gilt $(m_1, m_2) \in \succeq_{-d_1}$

$$\text{gdw. } (m_2, m_1) \in \succeq_{d_1},$$

$$\text{gdw. } (m_2, m_1) \in \succeq_{n_1},$$

$$\text{gdw. } (m_1, m_2) \in \succeq_n .$$

□

Aus all den vorangegangenen Sätzen folgt unmittelbar folgende, bereits am Anfang dieses Abschnittes aufgestellte, Behauptung.

Satz 8. *Jede durch einen LPD Ausdruck gegebene Präferenzrelation lässt sich durch einen entsprechenden LPD* Ausdruck definieren.*

2.2.2 Analyse der Sprachelemente

In [40] wurde die Möglichkeit der Bildung komplexer Präferenzrelationen aus gegebenen Präferenzrelationen unter allgemeinen Gesichtspunkten untersucht. Um die Ergebnisse dieses Kapitels in die Resultate obiger Arbeit einordnen zu können, sind einige vorbereitende Begriffsbildungen nötig. Da in unserem Kontext nur Ausdrücke endlicher Länge von Bedeutung sind, werden alle folgenden Begriffe für den Fall der Kombination *endlich* vieler Präferenzrelationen definiert.

Definition 15 (Operator). Ein Operator endlicher Arität V ist eine Abbildung, die jedem Tupel von Präferenzrelationen $(R_v)_{v \in V}$ eine Präferenzrelation R zuordnet.

Sei im Folgenden O ein Operator endlicher Arität und sei $R = O((R_v)_{v \in V})$.

Definition 16 (Unabhängigkeit). Wenn die Präferenzrelation R über einer Menge M von Modellen nur von den Präferenzrelationen $(R_v)_{v \in V}$ über M abhängt, d.h. falls

$$\forall M' \subseteq M, \quad O((R_v)_{v \in V})|_{M'} = O((R_v|_{M'})_{v \in V})$$

gilt, so besitzt der Operator O die Eigenschaft der Unabhängigkeit.

Definition 17 (Präferenzbasierung). Ein Isomorphismus $\phi : M \rightarrow M'$ zwischen zwei Modellmengen M und M' und zwei Mengen von Relationen $(R_v)_{v \in V}$ und $(R'_v)_{v \in V}$ ist eine Bijektion derart, dass gilt: $\forall v \in V \forall m, m' \in M ((m, m') \in R_v \text{ gdw. } (\phi(m), \phi(m')) \in R'_v)$. O hat die Eigenschaft der Präferenzbasierung, wenn er eine Funktion der Präferenzrelationen R_v ist (und damit insbesondere nicht die Identität von Modellen aus M berücksichtigt), d.h. dann, wenn für einen beliebigen Isomorphismus $\phi : M \rightarrow M'$ und für zwei beliebige Modelle m_1 und m_2 aus M gilt:

$$(m_1, m_2) \in O((R_v)_{v \in V}) \text{ gdw. } (\phi(m_1), \phi(m_2)) \in O((R'_v)_{v \in V}).$$

Definition 18 (Einstimmigkeit). Informal bedeutet diese Eigenschaft, dass R eine Einstimmigkeit aller kombinierten Präferenzrelationen $(R_v)_{v \in V}$ widerspiegeln soll, wobei Gleichheit als Enthaltung zählt. Genauer, sei R eine Relation, dann bezeichnet $R^>$ die entsprechende strikte Relation⁷, $R^< = (R^>)^{-1}$ das Inverse der strikten Relation und $R^=$ die entsprechende Gleichheitsrelation, vgl. (2.2) und (2.3). Die Inkomparabilitätsrelation $R^\#$ ist definiert als Menge aller Paare (m, m') , so dass weder $(m, m') \in R$ noch $(m', m) \in R$. Der Operator O besitzt die Eigenschaft der Einstimmigkeit genau dann, wenn für alle $\alpha \in \{<, >, =, \#\}$ gilt:

Sei $V' \subseteq V$ nicht leer, seien m_1 und m_2 Modelle aus M und sei $\forall v \in V' ((m_1, m_2) \in R_v^\alpha$ und $\forall v' \in V \setminus V' ((m_1, m_2) \in R_{v'}^=)$, dann gilt $(m_1, m_2) \in R^\alpha$.

⁷In [40] wird die in Bezug auf die vorliegende Arbeit, vgl. Definition (2.2) und (2.3), abweichende Konvention getroffen: $(m, m') \in R^<$ gdw. $(m, m') \in R$ und $(m', m) \notin R$.

Definition 19 (Transitivitätserhaltung). R ist transitiv, falls alle Präferenzrelationen $(R_v)_{v \in V}$ transitiv sind.

Definition 20 (Präferenzierte Algebra). Sei eine Menge von Relationen gegeben, die abgeschlossen gegenüber den binären Operatoren $/$ und \parallel ist, welche wie folgt definiert sind:

$$R_1/R_2 =_{\text{def}} (R_1 \cap R_2) \cup R_2^>$$

$$R_1 \parallel R_2 =_{\text{def}} R_1 \cap R_2.$$

Eine derartige Menge von Relationen zusammen mit den binären Operatoren $/$ und \parallel heißt präferenzierte Algebra.

Definition 21 (Sprache der präferenzierten Algebren). Die Syntax der Sprache der präferenzierten Algebren ist in BNF wie folgt definiert:

```

<term> ::= <term>/<term>
<term> ::= <term>\parallel<term>
<term> ::= <variable>.

```

Die Semantik ergibt sich aus der Ersetzung der Variablen (Relationssymbolen) mit den entsprechenden Relationen und der syntaktischen Operatoren $/$ und \parallel mit den oben definierten binären Operatoren.

Ein wichtiges Ergebnis der Arbeit [40] ist der folgende Satz.

Satz 9. *Jeder Operator endlicher Arität, der die obigen Eigenschaften der Unabhängigkeit, Präferenzbasierung, Einstimmigkeit und Transitivitätserhaltung erfüllt, lässt sich syntaktisch durch einen Term der Sprache der präferenzierten Algebren beschreiben.*

Ein Vergleich der Semantik der syntaktischen Operatoren $/$ und \parallel mit der Semantik der LPD Sprachkonstrukte \wedge und $>$, vgl. Definition 13, offenbart folgende Äquivalenz.

Satz 10. *Seien $R_1 = \succeq_{d_1}$ bzw. $R_2 = \succeq_{d_2}$ durch LPD Ausdrücke d_1 bzw. d_2 spezifizierte Relationen und n_1 bzw. n_2 äquivalente LPD* Ausdrücke, dann beschreiben jeweils*

der Term R_1/R_2 , der LPD Ausdruck $(d_2 > d_1)$ und der LPD Ausdruck $(\text{LEV } n_1 n_2)$, sowie*

der Term $R_1 \parallel R_2$, der LPD Ausdruck $d_1 \wedge d_2$ und der LPD Ausdruck $(\text{ALL } n_1 n_2)$*

dieselbe Relation⁸.

Beweis. R_1/R_2 beschreibt laut Definition des Operators $/$ die Relation $(R_1 \cap R_2) \cup R_2^>$. Da $R_1 = \succeq_{d_1}$, $R_2 = \succeq_{d_2}$ und $R_2^> = \succ_{d_2}$ gilt, spezifiziert der LPD Ausdruck $(d_2 > d_1)$ laut Definition der LPD Semantik dieselbe Relation. Nach Satz 7 beschreibt dann auch $(\text{LEV } n_1 n_2)$ diese Relation. Dass $R_1 \parallel R_2$ und $d_1 \wedge d_2$ dieselbe Relation spezifizieren, folgt analog. \square

Es stellt sich daher die Frage, wofür die anderen Operatoren der Sprachen LPD und LPD* nützlich sind, welche nicht den Operatoren $/$ bzw. \parallel entsprechen. Erstens werden diese Sprachkonstrukte eingesetzt, um Relation zu erzeugen, welche anschließend kombiniert werden können. Zweitens erfüllen die verwendeten Operatoren nicht notwendigerweise die obigen Eigenschaften der Unabhängigkeit, Präferenzbasierung,

⁸Man beachte, dass die Relation R_i durch eine Variable R_i gleichen Namens dargestellt wird.

Einstimmigkeit und Transitivitätserhaltung. Für die Sprache LPD* betrifft dies die Operatoren EX, CAR und INV. Für letzteren Operator ist die Antwort einfach.

Satz 11. *Sei n ein LPD* Ausdruck, dann existiert ein äquivalenter LPD* Ausdruck \tilde{n} , in welchem der Operator INV nicht vorkommt.*

Beweis. Sei im Folgenden $n = (\text{INV } n')$ vereinbart.

Sei $n' = (\text{ALL } n_1 \dots n_k)$, dann sind n und $\tilde{n} = (\text{ALL } (\text{INV } n_1) \dots (\text{INV } n_k))$ äquivalente LPD* Ausdrücke, denn es gilt:

$$\begin{aligned} m_1 \succeq_n m_2 &\text{ gdw. } m_2 \succeq_{n'} m_1 \\ &\text{ gdw. } \forall i, 1 \leq i \leq k (m_2 \succeq_{n_i} m_1) \\ &\text{ gdw. } \forall i, 1 \leq i \leq k (m_1 \succeq_{(\text{INV } n_i)} m_2) \\ &\text{ gdw. } m_1 \succeq_{\tilde{n}} m_2. \end{aligned}$$

Sei $n' = (\text{EX } n_1 \dots n_k)$, dann sind n und $\tilde{n} = (\text{EX } (\text{INV } n_1) \dots (\text{INV } n_k))$ äquivalente LPD* Ausdrücke, denn es gilt:

$$\begin{aligned} m_1 \succeq_n m_2 &\text{ gdw. } m_2 \succeq_{n'} m_1 \\ &\text{ gdw. } \exists l \geq 2 \text{ und } \exists (\tilde{m}_1, i_1, \tilde{m}_2, \dots, i_{l-1}, \tilde{m}_l), \text{ so dass} \\ &\quad (\tilde{m}_1 = m_2 \text{ und } \tilde{m}_l = m_1 \text{ und } \forall j, 1 \leq j \leq l-1 (\tilde{m}_j \succeq_{n_{i_j}} \tilde{m}_{j+1})) \\ &\text{ gdw. } \exists l \geq 2 \text{ und } \exists (\tilde{m}_l, i_{l-1}, \tilde{m}_{l-1}, \dots, i_1, \tilde{m}_1), \text{ so dass} \\ &\quad (\tilde{m}_l = m_1 \text{ und } \tilde{m}_1 = m_2 \text{ und } \forall j, 1 \leq j \leq l-1 (\tilde{m}_{j+1} \succeq_{(\text{INV } n_{i_j})} \tilde{m}_j)) \\ &\text{ gdw. } m_1 \succeq_{\tilde{n}} m_2. \end{aligned}$$

Sei $n' = (\text{INV } n_1)$, dann sind offensichtlich n und $\tilde{n} = n_1$ äquivalente LPD* Ausdrücke.

Sei $n' = (\text{LEV } n_1 \dots n_k)$, dann sind n und $\tilde{n} = (\text{LEV } (\text{INV } n_1) \dots (\text{INV } n_k))$ äquivalente LPD* Ausdrücke, denn es gilt:

$$\begin{aligned} m_1 \succeq_n m_2 &\text{ gdw. } m_2 \succeq_{n'} m_1 \\ &\text{ gdw. } \forall i, 1 \leq i \leq k (m_2 =_{n_i} m_1) \text{ oder} \\ &\quad \exists i, 1 \leq i \leq k (m_2 \succ_{n_i} m_1 \text{ und } \forall j, i < j \leq k (m_2 =_{n_j} m_1)) \\ &\text{ gdw. } \forall i, 1 \leq i \leq k (m_1 =_{(\text{INV } n_i)} m_2) \text{ oder} \\ &\quad \exists i, 1 \leq i \leq k (m_1 \succ_{(\text{INV } n_i)} m_2 \text{ und } \forall j, i < j \leq k (m_1 =_{(\text{INV } n_j)} m_2)) \\ &\text{ gdw. } m_1 \succeq_{\tilde{n}} m_2. \end{aligned}$$

Hierbei wurde ausgenutzt, dass $m \succ_n m'$ genau dann gilt, wenn $m' \succ_{(\text{INV } n)} m$. Dieser Fakt folgt unmittelbar aus der Definition von strikter Präferenzrelation und der Semantik des INV Konstruktes.

Sei $n' = (\text{CAR } n_1 \dots n_k)$, dann sind n und $\tilde{n} = (\text{CAR } (\text{INV } n_1) \dots (\text{INV } n_k))$ äquivalente LPD* Ausdrücke,

denn

$$\begin{aligned}
& m_1 \succeq_n m_2 \text{ gdw. } m_2 \succeq_{n'} m_1 \\
& \text{gdw. } \text{card}\{i \mid m_2 \succeq_{n_i} m_1\} \geq \text{card}\{i \mid m_1 \succeq_{n_i} m_2\} \\
& \text{gdw. } \text{card}\{i \mid m_1 \succeq_{(\text{INV } n_i)} m_2\} \geq \text{card}\{i \mid m_2 \succeq_{(\text{INV } n_i)} m_1\} \\
& \text{gdw. } m_1 \succeq_{\tilde{n}} m_2.
\end{aligned}$$

Sei n' eine aussagenlogische Formel f und bezeichne nf die entsprechende negierte Formel oder eine zur negierten Formel $\neg f$ im aussagenlogischen Sinne äquivalente Formel, dann sind n und $\tilde{n} = nf$ äquivalente LPD* Ausdrücke, denn für aussagenlogische Formeln f gilt $m \models f$ genau dann, wenn $m \not\models nf$. Es folgt

$$\begin{aligned}
& m_1 \succeq_n m_2 \text{ gdw. } m_2 \succeq_{n'} m_1 \\
& \text{gdw. } m_1 \not\models f \text{ oder } m_2 \models f \\
& \text{gdw. } m_1 \models nf \text{ oder } m_2 \not\models nf \\
& \text{gdw. } m_1 \succeq_{\tilde{n}} m_2.
\end{aligned}$$

□

Der INV Operator, welcher gegen die Eigenschaft der Einstimmigkeit verstößt, kann demnach aus der Sprache entfernt werden, ohne die Ausdrucksmächtigkeit in Bezug auf Präferenzrelationen zu verringern. Dieser Satz gilt insbesondere, weil als Grundobjekte aussagenlogische Formeln angenommen werden und die Menge aller aussagenlogischen Formeln gegenüber der Negation abgeschlossen ist. Der INV Operator kann deshalb auch eliminiert werden, falls als Grundobjekte Literale zugelassen sind. Besteht hingegen die Menge der Grundobjekte aus Atomformeln, so ist die oben vorgestellte Elimination nicht möglich.

Der CAR Operator in der vorliegenden Form besitzt folgende, nicht unbedingt wünschenswerte, Eigenschaften.

- $n = (\text{CAR } (\text{CAR } f_1 f_2 f_3) (\text{CAR } f_4 f_5 f_6) (\text{CAR } f_7 f_8 f_9))$ definiert eine andere Relation als $n' = (\text{CAR } f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8 f_9)$. Um dies zu sehen, seien die folgenden Modelle m_1 und m_2 gegeben. Es gelte $m_1 \models f_i \leftrightarrow i \in \{1, 2, 4, 5\}$ und $m_2 \models f_i \leftrightarrow i \in \{3, 6, 7, 8, 9\}$. Dann gilt einerseits $m_1 \succ_n m_2$, aber andererseits $m_2 \succ_{n'} m_1$.
- Verschachtelungen von CAR Knoten führen i.A. nicht zu einer transitiven Relation. Seien die Modelle m_1 und m_2 wie eben definiert und des Weiteren folgende Modelle gegeben: $m_3 \models f_i \leftrightarrow i \in \{1, 4, 5, 7\}$ und $m_4 \models f_i \leftrightarrow i \in \{1, 4, 7, 8\}$. Nun gilt: $m_2 \succeq_n m_4 \succeq_n m_3 \succeq_n m_1$, allerdings gilt nicht $m_2 \succeq_n m_1$.

Um einen transitiven Operator zu erhalten, gibt es nun mehrere Möglichkeiten. Sei durch einen CAR Knoten eine Relation R spezifiziert, dann ist es natürlich möglich, analog zum Fall des EX Operators, eine transitive Relation durch das explizite Bilden der transitiven Hülle von R zu erzeugen. Dieser Ansatz hat den Nachteil, dass die so gebildete Relation $\text{tr}(R)$ i.A. wesentlich größer ist als R und mithin sehr viele Modelle m die gleiche Präferenz besitzen. Eine alternativer Ansatz besteht in einer geeigneten Einschränkung der Syntax der CAR Knoten. Es wird später ersichtlich werden, dass damit jede spezifizierete Relation transitiv ist. Zudem lässt sich trotz dieser Einschränkung jeder beliebige LPD Ausdruck äquivalent darstellen.

Der Operator **EX** verstößt gegen die Eigenschaft der Unabhängigkeit. Dazu ein Beispiel.

Beispiel. Sei $M = \{m_1, m_2, m_3\}$ eine Menge von Modellen und $M' = \{m_1, m_3\}$ eine Teilmenge von M . Sei $F = \{f_1, f_2, f_3\}$ eine Menge von aussagenlogischen Formeln und gelte

$$\begin{aligned} m_1 \models f_j &\text{ genau dann, wenn } j \in \{1, 4\}, \\ m_2 \models f_j &\text{ genau dann, wenn } j \in \{2, 4\}, \\ m_3 \models f_j &\text{ genau dann, wenn } j \in \{2, 3\}. \end{aligned}$$

Sei weiterhin $n = (\mathbf{EX} \ n_1 \ n_2)$ mit $n_1 = (\mathbf{ALL} \ f_1 \ f_2 \ f_3)$ und $n_2 = (\mathbf{ALL} \ f_2 \ f_3 \ f_4)$. Den durch die Unterknoten definierten Präferenzrelationen über M entsprechen explizit folgende Mengen: $\succeq_{n_1} = \{(m_3, m_2)\}$ und $\succeq_{n_2} = \{(m_2, m_1)\}$. Laut Definition des **EX** Operators gilt nun für die resultierende Präferenzrelation $\succeq_n = \{(m_3, m_2), (m_2, m_1), (m_3, m_1)\}$. Die Einschränkung auf M' liefert $\succeq_n \upharpoonright_{M'} = \{(m_3, m_1)\}$. Die durch die gleichen Ausdrücke spezifizierten Relationen eingeschränkt auf die Teilmenge M' sind offenbar leer, d.h. $\succeq_{n_1} \upharpoonright_{M'} = \emptyset$ und $\succeq_{n_2} \upharpoonright_{M'} = \emptyset$. Wendet man den **EX** Operator darauf an, so erhält man im Unterschied zu $\succeq_n \upharpoonright_{M'}$ die leere Relation.

Dass der **EX** Operator nicht die Eigenschaft der Unabhängigkeit besitzt, beruht letztlich auf der expliziten Bildung des transitiven Abschlusses (vgl. Definition aus Tabelle 2.1). Die Verletzung dieser Eigenschaft hat sehr unangenehme Auswirkungen. Sei z.B. der **EX** Knoten $n = (\mathbf{EX} \ f_1 \ f_2 \ f_3)$ gegeben und erfülle ein erstes Modell m_1 keine der Formeln f_1 bis f_3 . Ein zweites Modell m_2 erfülle alle Formeln. Falls die Menge aller Modelle durch $M = \{m_1, m_2\}$ gegeben ist, so gilt $m_2 \succ_n m_1$. Sei nun ein weiteres Modell m_3 gegeben, welches nur die Formel f_1 erfüllt. Dann gilt plötzlich $m_1 =_n m_2$ auf der Menge $M = \{m_1, m_2, m_3\}$, d.h. die Präferenzrelation kollabiert, denn $m_1 =_n m_3 =_n m_2$.

2.2.3 Die Sprache TLPD

Aufgrund der oben angestellten Überlegungen über die Konstrukte **EX**, **CAR** und **INV** der LPD* Sprache wird im Weiteren eine Untersprache von LPD* verwendet. Diese Sprache verzichtet auf die **INV** und **EX** Elemente und schränkt die Syntax des **CAR** Konstruktes ein.

Sei TLPD (*Transitive Language for Preference Descriptions*) die durch folgende Syntax definierte Untersprache von LPD.

$$\begin{aligned} \langle \text{node} \rangle & ::= (\mathbf{LEV} \ \langle \text{node} \rangle^+) \\ \langle \text{node} \rangle & ::= (\mathbf{CAR} \ \langle \text{formula} \rangle^+) \\ \langle \text{node} \rangle & ::= (\mathbf{ALL} \ \langle \text{node} \rangle^+) \\ \langle \text{node} \rangle & ::= \langle \text{formula} \rangle \end{aligned}$$

Ihre Semantik ist durch die entsprechende Einschränkung der Semantik von LPD* definiert. Dann gilt:

Satz 12. *Jede durch einen \vee -freien LPD Ausdruck gegebene Präferenzrelation lässt sich durch einen entsprechenden TLPD Ausdruck definieren.*

Beweis. Dies folgt unmittelbar aus dem entsprechenden Satz über die LPD* Sprache und den Eigenschaften, dass jeder LPD* Ausdruck in einen äquivalenten **INV**-freien LPD* Ausdruck überführt werden kann

und dass zur äquivalenten Darstellung von LPD Konstrukten nur solche CAR Knoten benötigt werden, deren Unterknoten aussagenlogische Formeln sind. Die Forderung der \vee -Freiheit des LPD Ausdrucks ist nötig, da in TLPD das EX Konstrukt nicht mehr zur Verfügung steht. \square

Überdies gilt für TLPD im Gegensatz zu LPD* der folgende Satz.

Satz 13. *Jede durch einen TLPD Ausdruck definierte Präferenzrelation ist transitiv.*

Beweis. Der Beweis erfolgt mittels Induktion. Zunächst wird für aussagenlogische Formeln, als die fundamentalsten Präferenzausdrücke, die Transitivität der entsprechend definierten Präferenzrelation gezeigt. Anschließend wird die Transitivitätserhaltung der höheren Sprachkonstrukte nachgewiesen.

Induktionsanfang:

Sei $n = f$ eine aussagenlogische Formel. Dann gilt $m_1 \succeq_n m_2$ und $m_2 \succeq_n m_3$ genau dann, wenn $m_2 \models f \rightarrow m_1 \models f$ und $m_3 \models f \rightarrow m_2 \models f$. Dies impliziert $m_3 \models f \rightarrow m_1 \models f$ und damit $m_1 \succeq_n m_3$.

Sei $n = (\text{CAR } f_1 \dots f_k)$ ein TLPD Sprachausdruck. Seien $m_1 \succeq_n m_2$ und $m_2 \succeq_n m_3$, dann gilt (siehe Beweis des Hilfssatzes 4): $\text{card}\{f_i \mid m_1 \models f_i\} \geq \text{card}\{f_i \mid m_2 \models f_i\}$ und $\text{card}\{f_i \mid m_2 \models f_i\} \geq \text{card}\{f_i \mid m_3 \models f_i\}$. Daher gilt auch $\text{card}\{f_i \mid m_1 \models f_i\} \geq \text{card}\{f_i \mid m_3 \models f_i\}$ und somit $m_1 \succeq_n m_3$.

Induktionsschritt:

Sei $n = (\text{ALL } n_1 \dots n_k)$ ein TLPD Sprachausdruck. Seien $m_1 \succeq_n m_2$ und $m_2 \succeq_n m_3$, dann gilt $\forall i, 1 \leq i \leq k$ ($m_1 \succeq_{n_i} m_2$ und $m_2 \succeq_{n_i} m_3$). Aufgrund der angenommenen Transitivität bzgl. der durch die n_i Ausdrücke definierten Relationen gilt damit $\forall i, 1 \leq i \leq k$ ($m_1 \succeq_{n_i} m_3$), d.h. $m_1 \succeq_n m_3$.

Sei $n = (\text{LEV } n_1 \dots n_k)$ ein TLPD Sprachausdruck. Dann folgt aus $m_1 \succeq_n m_2$ und $m_2 \succeq_n m_3$: $\forall j, 1 \leq j \leq k$ ($m_1 =_{n_j} m_2$ und $m_2 =_{n_j} m_3$) oder $\exists i, 1 \leq i \leq k$ ($m_1 \succ_{n_i} m_2$ und $\forall j, i < j \leq k$ ($m_1 =_{n_j} m_2$)) oder $\exists i', 1 \leq i' \leq k$ ($m_2 \succ_{n_{i'}} m_3$ und $\forall j, i' < j \leq k$ ($m_2 =_{n_j} m_3$)). Sofern der erste Term der Disjunktion gilt, folgt sofort $\forall j, 1 \leq j \leq k$ ($m_1 =_{n_j} m_3$) und damit $m_1 \succeq_n m_3$. Sei also nun der erste Term der Disjunktion nicht wahr, dann ist $i^\dagger = \max\{i, i'\}$ der höchste Index $j = i^\dagger$ eines Knotens n_j mit ($m_1 \succ_{n_j} m_2$ oder $m_2 \succ_{n_j} m_3$). i^\dagger ist eindeutig, da i und i' eindeutig bestimmt sind. Damit gilt nun ($m_1 \succ_{n_{i^\dagger}} m_3$ und $\forall j, i^\dagger < j \leq k$ ($m_1 =_{n_j} m_3$)). Es folgt also auch in diesem Fall $m_1 \succeq_n m_3$. \square

Abschließend soll das Beispiel aus dem ersten Abschnitt nun in TLPD dargestellt werden. Nach Satz 6 werden die Ausdrücke K_{Thema}^\top bzw. $K_{\text{Bezahlung}}^\top$ durch die folgenden TLPD Ausdrücke n_{Thema} und $n_{\text{Bezahlung}}$ ersetzt:

$$\begin{aligned} n_{\text{Thema}} &= (\text{LEV Robotik} \vee \text{Bildverarbeitung} \vee \text{Planung} \quad \text{Planung}), \\ n_{\text{Bezahlung}} &= (\text{LEV halbe_Stelle} \vee \text{ganze_Stelle} \quad \text{ganze_Stelle}). \end{aligned}$$

Der LPD Ausdruck $K_{\text{Betreuung}}^\#$ wird laut Satz 3 durch folgendes Konstrukt ersetzt:

$$\begin{aligned} n_{\text{Betreuung}} &= (\text{LEV } (\text{CAR Gruppenseminar Konferenz}) \\ &\quad (\text{CAR intensive_Betreuung gutes_Arbeitsklima})). \end{aligned}$$

Die Präferenz des Studenten wird nun durch folgenden TLPD Ausdruck spezifiziert:

$$(n_{\text{Thema}} (\text{LEV } n_{\text{Bezahlung}} n_{\text{Betreuung}})).$$

2.3 Vergleich beider Ansätze

In diesem Abschnitt sollen die vorgestellten Präferenzbeschreibungssprachen LPD, LPD* und TLPD kurz verglichen und bezüglich der Ergebnisse aus [40] eingeordnet werden.

Die Verwendung von LPD zur Beschreibung von Präferenzen bietet folgende Vor- und Nachteile:

Vorteile:

- Sie liefert auf intuitive Art eine Rangfolge zwischen einzelnen Formeln durch den Einsatz von Wissensbasen.
- Komplexe Strategien lassen sich aus Basisstrategien zusammensetzen.

Nachteile:

- Sie definiert in erster Linie einen qualitativen Ansatz. Auf der Ebene komplexer Präferenzen ist nicht klar, wie man numerische Vergleiche integrieren will.
- Die gewichtete Wissensbasis schränkt die Klasse möglicher Präferenzrelationen ein, legt sie aber nicht fest. Dazu werden zusätzliche Präferenzausdrücke benötigt.

Der letzte Punkt besagt, dass bei der Verwendung von gewichteten Wissensbasen keine strikte Trennung zwischen der Definition der Grundobjekte (den aussagenlogischen Formeln) und der Präferenzrelation vorherrscht. Sei z.B. eine RKB gegeben, welche die Elemente (f_1, r_1) und (f_2, r_2) mit $r_1 > r_2$ enthält. Seien weiter zwei Modelle m_1 und m_2 gegeben, für die gilt: $m_1 \models f_1$, $m_1 \not\models f_2$, $m_2 \not\models f_1$, $m_2 \models f_2$ und $\forall i > 2 (m_1 \models f_i \leftrightarrow m_2 \models f_i)$. Intuitiv sollte es dann keine Präferenzrelation geben, die m_2 gegenüber m_1 bevorzugt, da m_1 eine Formel höheren Ranges erfüllt. Sofern dies zutrifft, schränkt die Verwendung der gewichteten Wissensbasis die Präferenzrelationen ein. Sollten andererseits doch Relationen erlaubt sein, die m_2 gegenüber m_1 bevorzugen, so ist die Bezeichnung „Rang“ für die Gewichte in der Wissensbasis unzutreffend. In diesem Fall könnte man von einer Einteilung der Wissensbasis in Klassen sprechen, wobei zunächst keinerlei Bevorzugung der einen oder anderen Klasse vorhanden sein darf. Die Zuweisung von Prioritäten zu den Klassen sollte dann erst durch den Präferenzausdruck erfolgen.

Diesem Ansatz folgt die Beschreibungssprache LPD*. Sie besitzt im Vergleich zu LPD folgende Eigenschaften

Gemeinsamkeiten mit LPD:

- flexibel und intuitiv anwendbar
- mindestens gleiche Ausdruckfähigkeit wie LPD
- keine Unterstützung von numerischen Konstrukten

Vorteile:

- einfache Syntax- und Semantikdefinition
- strikte Trennung zwischen Formeln und ihrer Präferenzierung

Nachteile:

- \succeq_{κ} und \succeq_{\top} werden nicht effizient ausgedrückt. Dies ließe sich u.U. durch Hinzufügen neuer Sprachkonstrukte verändern.
- LPD* erzeugt auch nichttransitive Relationen. Es kann jedoch die Untersprache TLPD verwendet werden, welche lediglich transitive Präferenzrelationen definiert.

Kapitel 3

Linearisierung und numerische Präferenzbeschreibungen

3.1 Linearisierung einer Quasiordnung

In dieser Arbeit wird unter einer Quasiordnung eine reflexive und transitive Relation verstanden. Eine antisymmetrische Quasiordnung wird als Ordnung bezeichnet. Eine Quasiordnung heißt linear, falls zwei beliebige Elemente stets vergleichbar sind. Dieser Abschnitt beschäftigt sich mit der Transformation einer Quasiordnung in eine lineare Quasiordnung. Zuvor soll diese Aufgabenstellung motiviert werden.

Mithilfe der Präferenzbeschreibungssprache TLPD kann eine Quasiordnung \succeq auf der Menge M der betrachteten Modelle definiert werden. In praktischen Anwendungen ist die genaue Relation jedoch oft nicht von Interesse. Vielmehr besteht die Aufgabe meist darin, maximale Modelle zu finden. Im Planungskontext beispielsweise entsprechen Modelle erreichbaren Zielzuständen, beispielsweise kodiert als Mengen von Formeln (vgl. 1.3.2). Im partiellen Erfüllbarkeitsproblem des Planens wird nun eine bestmögliche erfüllbare Zielmenge gesucht. Die Wertung verschiedener Zielmengen erfolgt dabei mittels der spezifizierten Präferenzrelation.

Sei R eine Quasiordnung und seien $R^>$ bzw. $R^=$ die abgeleitete strikte Relation bzw. Gleichheitsrelation. An die linearisierte Quasiordnung R_{lin} , die abgeleitete strikte Relation $R_{\text{lin}}^>$ und die abgeleitete Gleichheitsrelation $R_{\text{lin}}^=$ werden folgende Forderungen erhoben:

$$R_{\text{lin}} \supseteq R, \tag{3.1a}$$

$$R_{\text{lin}}^= \supseteq R^=, \tag{3.1b}$$

$$R_{\text{lin}}^> \supseteq R^>. \tag{3.1c}$$

Die Inklusion (3.1a) beschreibt die natürliche Forderung, dass zwei Elemente bzgl. der linearisierten Quasiordnung in Relation stehen, wenn sie es schon bzgl. der ursprünglichen Quasiordnung tun. Die letzte Inklusion verhindern ein Abschwächen der Relation durch die Linearisierung. Die drei Forderungen sind nicht unabhängig. Aus (3.1a) folgt sofort (3.1b). Aus (3.1c) und (3.1b) folgt (3.1a).

Sofern die letzte Forderung erfüllt ist, hat dies folgende interessante Konsequenz.

Satz 14. *Sei m^* ein maximales Element in der linearisierten Quasiordnung R_{lin} , welche die Forderung (3.1c) erfüllt, dann ist m^* auch ein maximales Element in der Quasiordnung R .*

Beweis. Sei $(m, m') \in R_{\text{lin}}$. Dann gilt $(m', m) \notin R^>$. Falls nämlich $(m', m) \in R^>$ wäre, so implizierte dies $(m', m) \in R_{\text{lin}}^>$, im Widerspruch zu $(m, m') \in R_{\text{lin}}$. \square

Wenn also die Aufgabe vorliegt, ein maximales Element in der Quasiordnung R zu finden, dann besteht die Grundidee des in diesem Kapitel vorgestellten Ansatzes darin, stattdessen ein maximales Element in der linearisierten Quasiordnung R_{lin} zu suchen. Wir werden später sehen, dass dies das Finden eines derartigen Elements sehr erleichtern kann.

Eine hinreichende Bedingung für eine derartige Linearisierung benennt der folgende Satz.

Satz 15. *Sei R eine Quasiordnung über der endlichen Menge M mit (mindestens¹) einem kleinsten Element ω , dann existiert eine lineare Quasiordnung, die (3.1a) – (3.1c) erfüllt.*

Beweis. Der Beweis erfolgt konstruktiv.

Sei H^n wie folgt induktiv definiert:

$$H^0 = \{m \mid (m, \omega) \in R^=\} \text{ und} \\ H^n = H^{n-1} \cup \{m \mid \forall m' \in M ((m, m') \in R^> \rightarrow m' \in H^{n-1})\}.$$

Für hinreichend großes n ist jedes Element aus M in H^n . Zunächst ist klar, dass $H^{n'} = H^n$ für alle $n' \geq n$ gilt, falls $H^{n+1} = H^n$, d.h. sobald ein Fixpunkt erreicht ist. Dies ist spätestens für $n^* = \text{card } M - 1$ erfüllt. Ein Element m wird nur dann nicht zu H^{2n^*} hinzugefügt, wenn mindestens ein echt kleineres Element m' noch nicht in $H^{2n^*-1} = H^{n^*}$ ist. Dieselbe Überlegung gilt nun auch für m' usw. Die Kette endet beim kleinsten Element ω . Somit dürfte auch ω nicht in $H^{2n^*-k} = H^{n^*}$ mit $k \leq n^*$ enthalten sein. Dies steht aber im Widerspruch zu $\omega \in H^n$ für beliebige $n \in \mathbb{N}$.

Sei d_m das kleinste $n \in \mathbb{N}$ mit $m \in H^n$, dann ist die folgende Relation R_{lin} reflexiv, transitiv, linear und erfüllt (3.1a) – (3.1c):

$$(m, m') \in R_{\text{lin}} \leftrightarrow d_m \geq d_{m'}.$$

Die Eigenschaften der Reflexivität und Transitivität sind offensichtlich erfüllt. Aus der Endlichkeit der Menge M folgt, dass für jedes m ein kleinstes $n \in \mathbb{N}$ mit $m \in H^n$ existiert. Daher ist die Relation auch linear.

Sei nun $(m, m') \in R^>$, dann kann offenbar m erst in H^n enthalten sein, wenn m' in H^{n-1} enthalten ist. Deshalb gilt $d_m > d_{m'}$ und somit $(m, m') \in R_{\text{lin}}^>$.

Sei $(m, m') \in R^=$, dann impliziert $m \in H^n$ auch $m' \in H^n$ und umgekehrt. Laut Definition gilt nämlich $m \in H^{d_m}$ und damit auch $\forall \tilde{m} \in M ((m, \tilde{m}) \in R^> \rightarrow \tilde{m} \in H^{d_m-1})$. Aus $(m, m') \in R^=$ und $(m, \tilde{m}) \in R^>$ folgt jedoch $(m', \tilde{m}) \in R^>$. Somit gilt $\forall \tilde{m} \in M ((m', \tilde{m}) \in R^> \rightarrow \tilde{m} \in H^{d_m-1})$ bzw. $d_m \geq d_{m'}$. Aus der

¹Da die Relation R i.A. nicht antisymmetrisch ist, kann es mehrere kleinste Elemente geben, d.h. mehrere Elemente ω , so dass für alle Elemente m gilt: $(m, \omega) \in R$

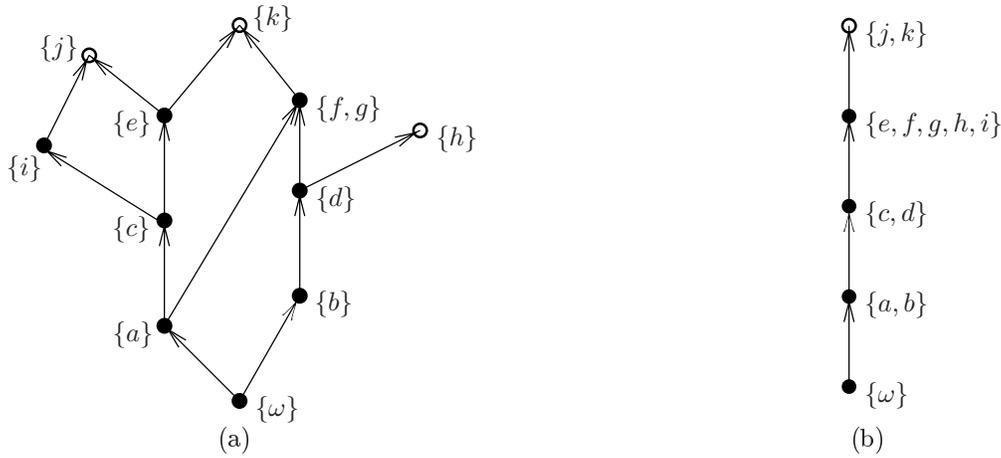


Abbildung 3.1: (a) Hasse-Diagramm einer Quasiordnung über einer endlichen Menge. (b) Die entsprechend linearisierte Quasiordnung. Die maximalen Elemente der jeweiligen Quasiordnung sind durch nichtausgefüllte Symbole gekennzeichnet.

analoges Überlegung beginnend bei $m' \in H^{d_{m'}}$ erhält man $d_{m'} \geq d_m$. Somit gilt $d_m = d_{m'}$ und deshalb $(m, m') \in R_{\text{lin}}^-$.

Sei $(m, m') \in R$, dann gilt $(m, m') \in R^>$ oder $(m, m') \in R^-$. Daraus folgt $(m, m') \in R_{\text{lin}}^>$ oder $(m, m') \in R_{\text{lin}}^-$ und somit $(m, m') \in R_{\text{lin}}$. \square

In Abbildung 3.1 sind die Hasse Diagramme einer nichtlinearen Quasiordnung und einer entsprechenden linearisierten Quasiordnung dargestellt, die nach der im Beweis von Satz 15 angegebenen Methode konstruiert wurde.

3.2 Linearisierung von TLPD

In diesem Abschnitt soll die Sprache TLPD linearisiert werden. Dies bedeutet, dass eine Sprache L angegeben ist, die lineare Relationen beschreibt und eine Vorschrift, welche TLPD Ausdrücke n in entsprechende L Ausdrücke \tilde{n} unter Berücksichtigung der folgenden Forderung übersetzt. Sei R die durch n beschriebene Quasiordnung, dann beschreibt \tilde{n} eine lineare Quasiordnung R_{lin} mit den Eigenschaften (3.1a) – (3.1c).

Diese Linearisierung könnte mithilfe von Satz 15 nach dem im Beweis angegebene Verfahren erfolgen. Tatsächlich lassen sich die Voraussetzungen des Satzes auch für die Sprache TLPD erfüllen. Dazu betrachtet man die endliche Menge F der in einem TLPD Ausdruck n vorkommenden Formeln und definiert eine Äquivalenzrelation \simeq auf der Menge M_Σ aller Σ -Strukturen derart, dass zwei Strukturen äquivalent zueinander sind, wenn sie sich auf der Menge F nicht unterscheiden. Der Satz 15 ist dann auf die durch n beschriebene Relation über der Menge M_Σ / \simeq der Äquivalenzklassen anwendbar² und man erhält

²Inbesondere ist M_Σ / \simeq endlich und die beschriebene Relation besitzt als kleinstes Element diejenige Äquivalenzklasse von Σ -Strukturen, die keine Formel aus F erfüllen.

Knotentyp	Wertungsfunktion val	Funktion maxval
$n = d, d \in \mathbb{N}$	$\text{val}(m, n) = d$	$\text{maxval}(n) = d$
$n = f, f$ ist Formel	$\text{val}(m, n) = \begin{cases} 1 & \text{falls } m \models f \\ 0 & \text{sonst} \end{cases}$	$\text{maxval}(n) = 1$
$n = (\text{CAR } n_1 \dots n_k)$	$\text{val}(m, n) = \sum_{i=1}^k \text{val}(m, n_i)$	$\text{maxval}(n) = \sum_{i=1}^k \text{maxval}(n_i)$
$n = (\text{LEV } n_1 \dots n_k)$	$\text{val}(m, n) = \sum_{i=1}^k c_i \text{val}(m, n_i)$ mit $c_i = \begin{cases} 1 & \text{falls } i = 1 \\ \prod_{j=1}^{i-1} (\text{maxval}(n_j) + 1) & \text{falls } i > 1 \end{cases}$	$\text{maxval}(n) = c_{k+1} - 1$
$n = (\text{MULT } n_1 \dots n_k)$	$\text{val}(m, n) = \prod_{i=1}^k \text{val}(m, n_i)$	$\text{maxval}(m, n) = \prod_{i=1}^k \text{maxval}(m, n_i)$

Tabelle 3.1: Die Wertungsfunktion $\text{val} : M \times \text{NLPD} \rightarrow \mathbb{N}$ und die Funktion $\text{maxval} : \text{NLPD} \rightarrow \mathbb{N}$. Die Wahl des maxval -Funktionswertes für einen LEV Knoten ist aufgrund einer Eigenschaft der Gewichte c_i , die im Beweis des Satzes 16 vorgestellt wird, korrekt.

eine linearisierte Relation. Durch Einschränkung auf die Menge M der Modelle (dies sind diejenigen Σ -Strukturen, die jede Formel des Hintergrundwissens B erfüllen) erhält man eine lineare Relation über M/\simeq . Die Anwendung dieses Verfahren erscheint jedoch recht umständlich. Es wird daher in diesem Abschnitt eine entsprechende Linearisierung direkt angegeben.

Die Sprache NLPD (*Numerical Language for Preference Descriptions*) besitze folgende Syntax.

```

<node> ::= (LEV <node>+)
<node> ::= (CAR <node>+)
<node> ::= (MULT <node>+)
<node> ::= <formula>
<node> ::= <non-negative integer>

```

Diese Sprache ähnelt in ihrer Syntax der Sprache TLPD. Wir werden später sehen, dass diese Wahl nicht ganz zufällig getroffen wurde. Die Semantik wird durch eine Wertungsfunktion val beschrieben. Diese Funktion bildet einen NLPD Ausdruck n und ein Modell m auf eine natürliche Zahl d ab und ist in Tabelle 3.1 angegeben. $\text{maxval}(n)$ beschreibt intuitiv den Wert, den $\text{val}(\cdot, n)$ maximal annehmen kann. Da dies natürlich von der betrachteten Modellmenge abhängt, wird die Funktion maxval in der Tabelle 3.1 definiert und dabei sichergestellt, dass $\text{maxval}(n) \geq \max_{m \in M} \text{val}(m, n)$ gilt. Die durch einen NLPD Ausdruck n beschriebene Präferenzrelation \succeq_n zwischen zwei Modellen m_1 und m_2 ist nun definiert als:

$$m_1 \succeq_n m_2 \text{ genau dann, wenn } \text{val}(m_1, n) \geq \text{val}(m_2, n).$$

Offenbar ist die so gegebene Relation \succeq_n reflexiv, transitiv und linear, d.h. eine lineare Quasiordnung, da die Funktion val für jeden Ausdruck und jedes Modell definiert ist.

Nun ist eine Übersetzung $\phi : \text{TLPD} \rightarrow \text{NLPD}$ derart anzugeben, dass für die durch die jeweiligen Ausdrücke spezifizierten Relationen die Bedingungen (3.1a) – (3.1c) gelten.

Definition 22 (Übersetzungsfunktion). Sei n ein TLPD Ausdruck und $\phi(n)$ der entsprechende NLPD Ausdruck. Die Übersetzungsfunktion $\phi : \text{TLPD} \rightarrow \text{NLPD}$ ist dann wie folgt gegeben:

$$\phi(n) = f, \text{ falls } n \text{ eine Formel } f \text{ ist.}$$

$$\phi(n) = (\text{CAR } f_1 \dots f_k), \text{ falls } n = (\text{CAR } f_1 \dots f_k) \text{ ist.}$$

$$\phi(n) = (\text{CAR } \phi(n_1) \dots \phi(n_k)), \text{ falls } n = (\text{ALL } n_1 \dots n_k) \text{ ist.}$$

$$\phi(n) = (\text{LEV } \phi(n_1) \dots \phi(n_k)), \text{ falls } n = (\text{LEV } n_1 \dots n_k) \text{ ist.}$$

Mit dieser Übersetzung gilt nachstehender Satz:

Satz 16. Sei n ein TLPD Ausdruck und $\phi(n)$ der mit der oben definierten Übersetzungsfunktion ϕ gebildete NLPD Ausdruck. Dann gelten für die Relation $R = \succeq_n$ und die lineare Relation $R_{\text{lin}} = \succeq_{\phi(n)}$ die Bedingungen (3.1a) – (3.1c).

Beweis. Der Beweis erfolgt induktiv. n bzw. $\phi(n)$ bezeichnet einen TLPD Ausdruck bzw. den entsprechenden NLPD Ausdruck.

Induktionsanfang:

Sei $n = f$ und seien zwei Modelle m_1 und m_2 gegeben. Dann gilt $m_1 =_n m_2$ genau dann, wenn $m_1 \models f \leftrightarrow m_2 \models f$. Dies gilt aber genau dann, wenn $\text{val}(m_1, f) = \text{val}(m_2, f)$ und damit genau dann, wenn $m_1 =_{\phi(n)} m_2$. Sei nun $m_1 \succ_n m_2$. Dies gilt genau dann, wenn $m_1 \models f$ und $m_2 \not\models f$. Dies ist genau dann möglich, wenn $\text{val}(m_1, f) = 1$ und $\text{val}(m_2, f) = 0$ bzw. wenn $m_1 \succ_{\phi(n)} m_2$.

Sei nun $n = (\text{CAR } f_1 \dots f_k)$. Dann gilt $m_1 =_n m_2$ genau dann, wenn $\text{card}\{i \mid m_1 \models f_i\} = \text{card}\{i \mid m_2 \models f_i\}$ (vgl. Beweis des Hilfssatzes 4). Dies ist äquivalent zu $\sum_i \text{val}(m_1, f_i) = \sum_i \text{val}(m_2, f_i)$ bzw. zu $m_1 =_{\phi(n)} m_2$. Analog gilt $m_1 \succ_n m_2$ genau dann, wenn $m_1 \succ_{\phi(n)} m_2$.

Induktionsschritt:

Sei $n = (\text{ALL } n_1 \dots n_k)$. Es gilt $m_1 =_n m_2$ genau dann, wenn für alle i mit $1 \leq i \leq k$ $m_1 =_{n_i} m_2$ gilt. Aufgrund der Induktionsvoraussetzung gilt für alle i : $m_1 =_{\phi(n_i)} m_2$, d.h. $\text{val}(m_1, \phi(n_i)) = \text{val}(m_2, \phi(n_i))$. Damit gilt auch $\sum_i \text{val}(m_1, \phi(n_i)) = \sum_i \text{val}(m_2, \phi(n_i))$ bzw. $m_1 =_{\phi(n)} m_2$. Sei $m_1 \succ_n m_2$, dann gilt für alle i mit $1 \leq i \leq k$: $m_1 \succeq_{n_i} m_2$ und es gibt ein j mit $1 \leq j \leq k$, so dass $m_1 \succ_{n_j} m_2$. Deshalb folgt $\sum_i \text{val}(m_1, \phi(n_i)) > \sum_i \text{val}(m_2, \phi(n_i))$ und $m_1 \succ_{\phi(n)} m_2$.

Es soll jetzt eine Eigenschaft der Gewichte c_i bewiesen werden, die für den weiteren Beweis wichtig ist. Mit der Schreibweise $\text{maxval}(n_0) =_{\text{def}} 0$ gilt

$$\begin{aligned} \sum_{l=1}^k c_l \text{maxval}(n_l) &= \sum_{l=1}^k \prod_{j=0}^{l-1} (\text{maxval}(n_j) + 1) \text{maxval}(n_l) \\ &= \sum_{l=1}^k \left[\prod_{j=0}^{l-1} (\text{maxval}(n_j) + 1) (\text{maxval}(n_l) + 1) - c_l \right] \\ &= \sum_{l=1}^k [c_{l+1} - c_l] \\ &= c_{k+1} - c_1 = c_{k+1} - 1. \end{aligned}$$

Sei $m_1 =_n m_2$ mit $n = (\text{LEV } n_1 \dots n_k)$. Dies gilt nach Definition des LEV Knotens in der TLPD Semantik offenbar genau dann, wenn für alle i mit $1 \leq i \leq k$ gilt: $m_1 =_{n_i} m_2$. Somit gilt $\sum_i \text{val}(m_1, \phi(n_i)) = \sum_i \text{val}(m_2, \phi(n_i))$ bzw. $m_1 =_{\phi(n)} m_2$. Sei nun $m_1 \succ_n m_2$. Dies ist lt. Definition genau dann erfüllt, wenn ein i mit $1 \leq i \leq k$ existiert, so dass $m_1 \succ_{n_i} m_2$ und zudem für alle j mit $i < j \leq k$ gilt: $m_1 =_{n_j} m_2$. Nach Induktionsvoraussetzung gilt also

$$\exists i, 1 \leq i \leq k (\text{val}(m_1, \phi(n_i)) > \text{val}(m_2, \phi(n_i)) \text{ und } \forall j, i < j \leq k (m_1 =_{\phi(n_j)} m_2))$$

Sei dieses i im Weiteren mit i^* bezeichnet. Die Wertungsfunktion val bildet in die Menge der natürlichen Zahlen ab. Deshalb ist $\text{val}(m_1, \phi(n_{i^*}))$ mindestens so groß wie $\text{val}(m_2, \phi(n_{i^*})) + 1$. Wir betrachten nun die folgende Aufspaltung.

$$\begin{aligned} \text{val}(m_1, \phi(n)) &= \sum_{i=1}^k c_i \text{val}(m_1, \phi(n_i)) \\ &= \sum_{i=1}^{i^*-1} c_i \text{val}(m_1, \phi(n_i)) + c_{i^*} \text{val}(m_1, \phi(n_{i^*})) + \sum_{l=i^*+1}^k c_l \text{val}(m_1, \phi(n_l)) \end{aligned}$$

Eine analoge Aufspaltung ist natürlich auch für $\text{val}(m_2, \phi(n))$ möglich und es gilt folgende Abschätzung:

$$\begin{aligned} \text{val}(m_1, \phi(n)) &\geq c_{i^*} \text{val}(m_1, \phi(n_{i^*})) + \sum_{l=i^*+1}^k c_l \text{val}(m_1, \phi(n_l)) \\ &\geq c_{i^*} (\text{val}(m_2, \phi(n_{i^*})) + 1) + \sum_{l=i^*+1}^k c_l \text{val}(m_2, \phi(n_l)) \\ &> \sum_{i=1}^{i^*-1} c_i \text{val}(m_2, \phi(n_i)) + c_{i^*} \text{val}(m_2, \phi(n_{i^*})) + \sum_{l=i^*+1}^k c_l \text{val}(m_2, \phi(n_l)) \\ &\geq \text{val}(m_2, \phi(n)) \end{aligned}$$

Die erste Ungleichung ist der Tatsache zu schulden, dass die Bewertungsfunktion und die Gewichte c_i nichtnegativ sind. Die dritte Ungleichung ist durch die oben betrachtete Eigenschaft der Gewichte gegeben $c_{i^*} = \sum_{i=1}^{i^*-1} c_i \max \text{val}(n_i) + 1$. Aus der Abschätzung folgt unmittelbar $m_1 \succ_{\phi(n)} m_2$. \square

Die LEV und CAR Konstrukte der Sprache NLPD sind notwendig, um die Sprache TLPD geeignet linearisieren zu können. Die Definition der Übersetzungsfunktion erklärt zudem die Namenswahl der Sprachelemente von NLPD. LEV Konstrukte werden in LEV Konstrukte übersetzt, CAR Konstrukte in CAR Konstrukte. Das CAR Sprachelement aus NLPD unterscheidet sich jedoch von demjenigen aus TLPD insofern, dass es beliebige Knoten als Unterknoten besitzen darf. Es unterscheidet sich jedoch auch von dem CAR Sprachkonstrukt aus LPD*, da es stets eine transitive Relation definiert. Insbesondere ist eine Verschachtelung einzig bestehend aus CAR Knoten und Formeln äquivalent zu einem Ausdruck, der nur einen CAR Knoten enthält. Beispielsweise beschreiben die NLPD Ausdrücke $n = (\text{CAR } (\text{CAR } f_1 f_2 f_3) (\text{CAR } f_4 f_5 f_6) (\text{CAR } f_7 f_8 f_9))$ und $n' = (\text{CAR } f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8 f_9)$ dieselbe Relation, denn $\text{val}(m, n) = \sum_{i=0}^2 (\sum_{j=1}^3 \text{val}(m, f_{3i+j})) = \sum_{i=0}^9 \text{val}(m, f_i) = \text{val}(m, n')$.

Das **MULT** Konstrukt ist ein numerisches Konstrukt, das es ermöglicht, das Gewicht einzelner Knoten zu vervielfachen. Eine additive Erhöhung des Gewichtes ist mit dem **CAR** Konstrukt möglich. Somit lassen sich mit NLPD sehr einfach quantitative Präferenzen beschreiben. Andere zusätzliche Konstrukte sind denkbar. Beispielhaft seien ein **AND** und ein **OR** Sprachelement mit folgender Syntax und Semantik angegeben.

$\langle \text{node} \rangle ::= (\text{AND } \langle \text{node} \rangle^+)$

$\langle \text{node} \rangle ::= (\text{OR } \langle \text{node} \rangle^+)$

Sei $n_{\text{AND}} = (\text{AND } n_1 \dots n_k)$, $n_{\text{OR}} = (\text{OR } n_1 \dots n_k)$, dann ist $\text{val}(m, n_{\text{AND}})$ bzw. $\text{val}(m, n_{\text{OR}})$ definiert als

$$\text{val}(m, n_{\text{AND}}) = \begin{cases} 1 & \text{falls } \forall i, 1 \leq i \leq k (\text{val}(m, n_i) = \max \text{val}(m, n_i)) \\ 0 & \text{sonst,} \end{cases}$$

$$\text{val}(m, n_{\text{OR}}) = \begin{cases} 1 & \text{falls } \exists i, 1 \leq i \leq k (\text{val}(m, n_i) > 0) \\ 0 & \text{sonst.} \end{cases}$$

Kapitel 4

Übersetzung in PDDL2.1

4.1 Lösung partieller Erfüllbarkeitsprobleme

In Abschnitt 1.2 wurde bereits konzeptionell angedeutet, welchen Ansatz diese Arbeit zur Lösung partieller Erfüllbarkeitsprobleme verfolgt. Die Grundidee besteht demnach darin, ein PSP $\Gamma = (S, A, \gamma, s_I, S_G, \succeq)$ zunächst in ein LPSP $\Gamma' = (S, A, \gamma, s_I, S_G, \succeq_{\text{lin}})$ zu überführen. In Kapitel 3 wurde unter allgemeinen Gesichtspunkten gezeigt, dass eine derartige Übersetzung unter Erhalt der Maximalitätseigenschaft durchgeführt werden kann, d.h. maximale Elemente der reflexiven und transitiven Relation \succeq sind auch maximal bezüglich der entsprechend linearisierten Relation \succeq_{lin} . Damit stellt jede Lösung von Γ' auch eine Lösung von Γ dar. Um dies zu sehen, muss der Zusammenhang zwischen Zuständen s eines Planungsproblems und den Modellen m der allgemeinen Betrachtung aus Kapitel 2 etabliert werden. Modelle wurden als Σ -Strukturen vorgestellt¹, die eine Menge von Formeln B , genannt Hintergrundwissen, erfüllen. Im Kontext partieller Erfüllbarkeitsprobleme entsprechen Σ -Strukturen den Zuständen $s \in S$ und die Modelle m den Zielzuständen $s \in S_G$. Das Hintergrundwissen B stellt eine Beschreibung der Menge S_G aller Zielzustände dar. Damit übertragen sich die Ergebnisse über Modellmengen auf entsprechende Mengen von Zielzuständen im Planungskontext.

Es wurde bereits dargelegt, dass linearisierte partielle Erfüllbarkeitsprobleme mit Schranke (LPSP*) äquivalent zu klassischen Planungsproblemen sind, welche mit herkömmlichen Planern gelöst werden können. In diesem Abschnitt soll gezeigt werden, wie sich das Lösen von LPSP auf das Lösen von klassischen Planungsproblemen zurückführen lässt. Jeder derartige Algorithmus wird vermutlich in Bezug auf Rechenzeit recht kostspielig sein, da sich ein PSP eben nicht einfach in ein klassisches Planungsproblem übersetzen lässt. Allerdings bedeutet dies nicht notwendigerweise eine Erhöhung der Komplexität, denn

¹In der Aussagenlogik entsprechen Σ -Strukturen Interpretationen, die jeder in einer aussagenlogischen Formel f enthaltenen Atomformel einen Wahrheitswert $w \in \{\text{wahr, falsch}\}$ zuordnen und den Wahrheitswert von f aus den Wahrheitswerten der Atomformeln, je nach verwendeten logischen Funktoren, bestimmen. Σ -Strukturen lassen sich jedoch nur für STRIPS-artige Sprachen direkt als Formelmengen deuten. Die Probleme, die z.B. der numerische Teil von PDDL2.1 mit sich bringt, sollen allerdings nicht an dieser Stelle und damit in aller Abstraktheit besprochen werden. Stattdessen wird sich damit an gegebener Stelle (vgl. Abschnitt 4.2.2) befaßt.

Algorithmus 1 Lösung des LPSP Problems $\Gamma' = (S, A, \gamma, s_I, S_G, \succeq_{\text{lin}})$

 $n' := 0;$ **repeat** $n := n' + 1;$ $n' := \phi(\text{solve}(S, A, \gamma, s_I, S'_G = \{s \mid s \in S_G, \phi(s) \geq n\}));$ **until** ($n' = 0$)

bereits das klassische Planen ist PSPACE-hart [18].

Bemerkung. Sei ein LPSP $\Gamma' = (S, A, \gamma, s_I, S_G, \succeq_{\text{lin}})$ gegeben, dann existiert eine Übersetzung $\phi : S_G \cup \{\text{unsolvable}\} \rightarrow \mathbb{N}$ dergestalt, dass

- für beliebige $s, s' \in S_G$ genau dann $s \succeq_{\text{lin}} s'$ gilt, wenn $\phi(s) \geq \phi(s')$ und dass
- für beliebige $\tilde{s} \in S_G \cup \{\text{unsolvable}\}$ genau dann $\phi(\tilde{s}) = 0$ gilt, wenn $\tilde{s} = \text{unsolvable}$.

Beweis. Offensichtlich lässt sich der erste Punkt stets erfüllen. Sei ein ϕ' gegeben, das den ersten aber nicht den zweiten Punkt erfüllt, dann erfüllt folgende Übersetzung ϕ beide Forderungen:

$$\phi(s) = \begin{cases} 0 & \text{falls } s = \text{unsolvable} \\ \phi'(s) + 1 & \text{sonst.} \end{cases}$$

□

Sei `solve` ein vollständiger und korrekter Planungsalgorithmus für klassische Planungsprobleme $\tilde{\Gamma}$, welcher die erreichte Zielmenge zurückliefere². Sei weiterhin $\Gamma' = (S, A, \gamma, s_I, S_G, \succeq_{\text{lin}})$ das zu lösende LPSP, dann liefert Algorithmus 1 eine Lösung von Γ' .

Der Algorithmus arbeitet wie folgt:

1. Er setzt den Subalgorithmus `solve` auf das klassische Planungsproblem $\tilde{\Gamma} = (S, A, \gamma, s_I, S_G)$ an, denn für $n = 1$ ist $S'_G = \{s \mid s \in S_G, \phi(s) \geq 1\} = S_G$. Sollte dies nicht lösbar sein, ist auch das LPSP nicht lösbar und der Algorithmus bricht ab.
2. Er erschwert das klassische Planungsproblem, indem er alle Zielzustände s ausschließt, die nicht besser als der bereits erreichte Zielzustand s_E sind, d.h. falls $s_E \succeq_{\text{lin}} s$ bzw. $\phi(s_E) \geq \phi(s)$ gilt, und setzt dann `solve` auf $\tilde{\Gamma} = (S, A, \gamma, s_I, S'_G)$ mit $S'_G = \{s \mid s \in S_G, \phi(s) > \phi(s_E)\}$ an.
3. Sukzessive erhöht sich so die Qualität der Zielzustände. Formal endet der Algorithmus, falls kein besserer Zielzustand mehr gefunden werden kann. Der beste gefundene Zielzustand ist dann aufgrund der angenommenen Vollständigkeit und Korrektheit des Planers eine Lösung des LPSP.

Im Algorithmus wird eine Übersetzung $\phi : S_G \cup \{\text{unsolvable}\} \rightarrow \mathbb{N}$ der oben angegebenen Gestalt verwendet. Ein Grund hierfür liegt darin, dass es nicht unbedingt einfach ist, zu einem gegebenen Zustand s einen

²Der Planungsalgorithmus gibt normalerweise als Lösung die gefundene Aktionsfolge aus. Hier möge er primär den Zielzustand, der sich durch Anwendung dieser Aktionsfolge auf den Startzustand ergibt, zurückliefern. Die entsprechende Aktionsfolge möge er an anderer Stelle verwahren.

Zustand s' zu finden, der strikt besser als s ist. Dieselbe Aufgabe ist im Bereich der natürlichen Zahlen aber ganz einfach: es wird schlicht die nächstgrößere Zahl gewählt. Ob diese Zahl n einem Zustand s entspricht, d.h. ob es ein s mit $\phi(s) = n$ gibt, ist dabei unwichtig. Die Abbildung auf natürliche Zahlen empfiehlt sich also, um ein partielles Erfüllbarkeitsproblem mit Schranke in PDDL2.1 ausdrücken zu können. Die Bedingung einen Zustand zu finden, der besser als ein gegebener Grenzzustand ist, wird in PDDL2.1 durch einen numerischen Vergleich ausgedrückt. Dabei wird der Grenzzustand durch die entsprechende Grenzzahl ersetzt.

Bemerkung. Für die Präferenzbeschreibungssprache NLPD kann eine geeignete Übersetzung übrigens leicht angegeben werden. Sei d ein NLPD Ausdruck, welcher die lineare Quasirelation \succeq_{lin} beschreibt, dann ist die Funktion $\phi^{\text{val}} : S_G \cup \{\text{unsolvable}\} \rightarrow \mathbb{N}$ mit

$$\phi^{\text{val}}(s) = \begin{cases} 0 & \text{falls } s = \text{unsolvable} \\ \text{val}(s, d) + 1 & \text{sonst} \end{cases}$$

eine geeignete Übersetzung. Hierbei wurde die in Abschnitt 3.2 definierte Funktion val verwendet, sowie die oben erwähnte Identifikation von Modellen und Zuständen.

Folgende Aspekte spielen bei der Effizienz und Flexibilität des Algorithmus' eine wichtige Rolle:

- In STRIPS-artigen Repräsentationssprachen werden Zielzustände durch Formelmengen F beschrieben. In Bezug auf F lassen sich damit $\text{card } \mathcal{P}(F)$ potentielle Zielmengen unterscheiden. Es ist praktisch unmöglich für moderate F alle potentiellen Zielmengen durchzuprobieren und anschließend von den erwiesenen Zielmengen die beste auszuwählen. Die Abbildung auf eine lineare Präferenzrelation weist Zielmengen Äquivalenzklassen zu. Selbst das naive Durchprobieren aller Äquivalenzklassen schränkt bereits den Suchaufwand ein, da die Anzahl der Äquivalenzklassen stets kleiner oder gleich $\text{card } \mathcal{P}(F)$ ist. Abhängig von der verwendeten Präferenzstrategie ist die Anzahl sogar deutlich kleiner.
- Durch Transformation in ein klassisches Planungsproblem kann der jeweils effektivste klassische Planer angewendet werden. Insbesondere wurde im Rahmen dieser Arbeit der heuristische Vorwärtsplaner **Metric-FF** zu Testzwecken verwendet. Jedoch lassen sich die dargestellten Ergebnisse prinzipiell für jeden Planer verwenden, der nur einen gewissen Mindestsprachumfang besitzt. Dies wird in den nachfolgenden Abschnitten ausgeführt.
- Es kommt oft vor, dass der jeweils eingesetzte Planer (besonders während der ersten Iterationen des Algorithmus' 1) Zielzustände ermittelt, die wesentlich besser als gefordert sind. Somit lassen sich bereits nach relativ kurzer Zeit qualitativ gute Zielzustände auffinden.
- Erfolgloses Suchen nach einer Lösung eines klassischen Planungsproblems nimmt sehr viel Zeit in Anspruch. Eine derartige Suche findet in Algorithmus 1 jedoch nur höchstens einmal statt, nämlich genau dann³, wenn vorher ein maximaler Zielzustand entdeckt wurde.
- Der Algorithmus findet immer einen maximalen Zielzustand und damit eine Lösung des partiellen Erfüllbarkeitsproblems, sofern er genügend lange ausgeführt wird. Er kann jedoch auch als *anyti-*

³Vollständigkeit und Korrektheit des eingesetzten Planers vorausgesetzt.

me-Algorithmus verwendet werden und die im Laufe der Zeit gefundenen (und immer besseren) Zielzustände ausgeben.

Folgende Modifikationen des Algorithmus' sind eine Überlegung wert, wenn der maximale Wert der Übersetzung ϕ sehr groß werden kann. Dieser maximale Wert kann beispielsweise für einen Ausdruck der Beschreibungssprache NLPD durch die *maxval* Funktion, nach oben beschränkt, abgeschätzt werden. Anstatt die Grenzzahl n in jedem Schleifendurchlauf von Algorithmus 1 auf den um eins erhöhten Rückgabewert des klassischen Planers zu setzen, wäre eine Erhöhung mittels der folgenden Änderung in Zeile 3 des Algorithmus' denkbar:

$$n := n' + \frac{\text{maxval} - n'}{2};$$

Dies entspricht einer Art binärer Suche. In jedem Schleifendurchlauf wird der noch nicht durchsuchte Wertebereich (mindestens) halbiert. Der große Nachteil dieses Ansatzes besteht darin, dass sehr häufig Suchen ohne Erfolg durchgeführt werden. Eine Bewertung dieses Ansatzes erfordert demnach ein gegeneinander Aufwiegen zweier Zeitfaktoren. Für moderate Größen des Wertebereiches von ϕ wiegt der Zeitverlust durch erfolgloses Suchen wahrscheinlich schwerer. Es existieren jedoch Mittelwege zwischen Algorithmus 1 und der eben vorgestellten Modifikation. So ist es denkbar, die Grenzzahl n derart zu erhöhen, dass höchstens k erfolglose Suchen möglich sind. Somit wird einerseits der Wertebereich von ϕ schneller durchsucht, andererseits werden die Zeitkosten durch fehlgeschlagenes Suchen begrenzt.

4.2 Übersetzung in PDDL2.1

In Abschnitt 4.1 wurde abstrakt der Weg aufgezeigt, PSP mithilfe gängiger Planer zu lösen. Im jetzigen Abschnitt soll dieses Vorgehen unter Verwendung der Repräsentationssprachen PDDL2.1 bzw. PDDL2.1' konkretisiert werden. Dazu wird zunächst eine Erweiterung dieser Sprachen postuliert, die es erlaubt, partielle Erfüllbarkeitsprobleme zu definieren. Durch die zusätzliche Angabe einer Schranke erhält man ein partielles Erfüllbarkeitsproblem mit Schranke (vgl. Abschnitt 1.2). Es wird anschließend gezeigt, wie ein derartiges Problem in PDDL2.1 bzw. PDDL2.1' und damit als klassisches Planungsproblem ausgedrückt werden kann.

4.2.1 Erweiterung von PDDL2.1

Eine Erweiterung von PDDL2.1 (bzw. PDDL2.1') kann an mehreren Punkten ansetzen.

- Präferenzen werden mit TLPD Ausdrücken spezifiziert. Die Linearisierung der dadurch spezifizierten Relation erfolgt automatisch bei der Übersetzung in PDDL2.1.
- Präferenzen werden mittels NLPD Ausdrücken angegeben. Damit sind Präferenzen per se linear.

Die erste Möglichkeit besitzt den Vorteil, dass der Nutzer nichtlineare Präferenzen angeben kann und eine Linearisierung automatisch erfolgt. Allerdings ist eine Linearisierung, wie wir in Kapitel 3 gesehen haben, sehr leicht durchführbar. Die zweite Möglichkeit bietet dem Nutzer den Vorteil numerische Konstrukte

und somit auch quantitative Präferenzen verwenden zu können. Es wird daher an dieser Stelle der zweite Weg gewählt.

Es wird nun eine Erweiterung von PDDL2.1 definiert, vgl. auch Anhang A.5. Dazu wird ein neues Nichtterminalsymbol `<psp>` eingeführt. Die Produktionsregeln dieses Symbols orientieren sich hierbei an den Sprachen PDDL2.1 und NLPD. Der folgende Eintrag, er wird im Weiteren als PSP-Beschreibung bezeichnet, definiert zusammen mit der gegebenen Domänen- und Faktenbeschreibung ein lineares partielles Erfüllbarkeitsproblem in PDDL2.1*.

```

<require-key>          ::= :psp
<require-key>          ::= :psp-full
<psp>                  ::= :psp, :psp-full (define (pspname <name>)
                                           (:problem <name>)
                                           [(:domain <name>)]
                                           <goal>
                                           <psp-def>)

<psp-def>              ::= (:psp <psp-node>)
<psp-node>             ::= (LEV <psp-node>+)
<psp-node>             ::= (CAR <psp-node>+)
<psp-node>             ::= (MULT <psp-node> <non-negative integer>)
<psp-node>             ::= <literal(name)>
<psp-node>             ::= (<binary-comp> <f-exp> <f-exp>)
<psp-node>             ::= <non-negative integer>
<non-negative integer> ::= eine natürliche Zahl
<psp-node>             ::= :psp-full (MULT <psp-node>+)
<psp-node>             ::= :psp-full <GD>

```

Bemerkung. Der Nutzer legt mit der Spezifikation des Nichtterminalsymbols `<require-key>`, d.h. den Sprachspezifikationen `:psp` bzw. `:psp-full`, den Sprachumfang der PSP-Beschreibung fest. Eine hochgestellte Schreibweise von Sprachspezifikationen soll andeuten, dass mindestens eine enthaltene Sprachspezifikation erfüllt sein muss, um die entsprechende Produktionsregel der BNF anwenden zu können.

Sofern die Sprachspezifikation `:psp` aber nicht `:psp-full` angegeben ist, werden lediglich diejenigen Konstrukte der obigen Syntaxbeschreibung erlaubt, für die gilt: Die bei der Übersetzung in die PDDL2.1 Sprache entstehenden numerischen Ausdrücke sind *linear*. Somit wird in die Sprache PDDL2.1' übersetzt, sofern Domänen- und Faktenbeschreibung ebenfalls in PDDL2.1' gegeben sind. Die Erweiterung von PDDL2.1' um die PSP-Beschreibung erzeugt eine Sprache, die mit PDDL2.1'* bezeichnet wird.

Falls hingegen die Sprachspezifikation `:psp-full` verwendet wird, so sind weitere Konstrukte innerhalb der PSP-Beschreibung gestattet und es können auch nichtlineare numerische Ausdrücke in der Übersetzung vorkommen. Diese Erweiterung von PDDL2.1 erzeugt die Sprache PDDL2.1*.

`<GD>` Symbole definieren Zielbeschreibungen. Dies können (vgl. Anhang A) aussagenlogische Formeln

oder quantisierte funktionsfreie Formeln sein, welche u.U. numerische Vergleiche beeinhalt⁴. Sofern nichtgebundene Variablen in den Formeln vorkommen, werden sie als all-quantifiziert betrachtet. Quantisierte funktionsfreie Formeln lassen sich bei gegebener endlicher Zahl von Objekten in aussagenlogische Formeln übersetzen. Wir wollen daher im Folgenden annehmen, dass in einem ersten Prozess jede Zielbeschreibung in eine quantorfreie Form, d.h. aussagenlogische Form, gebracht wird. Des Weiteren kann jeder Ausdruck der Form (`imply <GD1> <GD2>`) durch einen entsprechenden Ausdruck der Form (`or (not <GD1>) <GD2>`) ersetzt werden. Mit derartigen Umformungen vereinfacht, vgl. Anhang A für die nichtvereinfachte Syntaxdarstellung, besitzen Zielbeschreibungen die Form:

```

<GD> ::= ()
<GD> ::= <atomic formula(name)>
<GD> ::= (not <GD>)
<GD> ::= (and <GD>*)
<GD> ::= (or <GD>*)
<GD> ::= (<binary-comp> <f-exp> <f-exp>)

```

Eine Angabe von Zielzuständen mithilfe des `<goal>` Ausdrucks wurde aus praktischen Erwägungen aufgenommen. Die enthaltenen Ziele überschreiben entsprechende Angaben in der Faktenbeschreibung und sind obligatorisch für eine Erfüllung des partiellen Erfüllbarkeitsproblems.

4.2.2 Generelle Überlegungen

Sei eine PDDL2.1* PSP-Beschreibung gegeben. Sie definiert eine Menge⁵ G von Zielbeschreibungen $G = \{g_i \mid 1 \leq i \leq I\}$. Für jede enthaltene Zielbeschreibung g_i gilt, dass sie in einem Zustand s erfüllt sein kann und in einem anderen Zustand s' möglicherweise nicht. Ein PDDL2.1 Zustand $s = (f, \nu) \in S = \mathcal{P}(F) \times \mathbb{Q}^k$ ist ein Paar bestehend aus einer Formelmenge f und einem k -Tupel ν rationaler Zahlen, wobei die i -te Komponente von ν , d.h. ν_i , den Wert der numerischen Variablen v_i angibt. Man kann nun völlig analog zur NLPD Sprache (vgl. Abschnitt 3.2) eine Funktion val' definieren, die für jede PDDL2.1* PSP-Beschreibung jedem Zustand s eine natürliche Zahl zuordnet. Die Definition erfolgt analog zu Tabelle 3.1 auf induktivem Wege, wobei der Induktionsanfang durch

$$\text{val}'(s, g_i) = \begin{cases} 1 & \text{falls } g_i \text{ in } s \text{ erfüllt ist} \\ 0 & \text{sonst} \end{cases}$$

gegeben ist. Die gegebene PDDL2.1* PSP-Beschreibung n sei jetzt fest. Dann definiert die Funktion $\text{val} : S \rightarrow \mathbb{N}$ mit $\text{val}(s) = \text{val}'(s, n)$ eine Abbildung von der Menge der Zustände in die Menge der natürlichen Zahlen. Auf eine Weise, wie sie in Bemerkung 4.1 vorgestellt wurde, erhält man ebenso eine Übersetzung $\phi^{\text{val}} : S_G \cup \{\text{unsolvable}\} \rightarrow \mathbb{N}$ mit den in Bemerkung 4.1 angegebenen Eigenschaften. Das Problem liegt allerdings darin, eine PDDL2.1 Beschreibung anzugeben, welche die Funktionswerte $\text{val}'(s, g_i)$ berechnet, d.h. die prüft, dass g_i in s erfüllt ist.

⁴Ein numerischer Vergleich kann in einem Zustand entweder den Wahrheitswert *wahr* oder *falsch* annehmen und deshalb als Atomformel aufgefasst werden.

⁵Diese kann auch leer sein.

Um dies zu erreichen, wird eine Funktion $w : S \rightarrow \mathbb{N}^I$ eingeführt, deren i -te Komponente w_i durch $w_i(s) = \text{val}'(s, g_i)$ gegeben ist. Sofern sich diese Funktion in PDDL2.1 implementieren lässt, bleibt zu zeigen, wie man darauf aufbauend val bzw. ϕ^{val} im Rahmen von PDDL2.1 berechnet. Die Funktionen val und ϕ^{val} , die Zustände auf natürliche Zahlen abbilden, ergeben sich aus der Komposition einer geeignet definierten Funktion $\widehat{\text{val}} : \mathbb{N}^I \rightarrow \mathbb{N}$ mit der Funktion w , genauer:

$$\begin{aligned} \text{val}(s) &= \widehat{\text{val}}(w(s)) \\ \phi^{\text{val}}(s) &= \begin{cases} 0 & \text{falls } s = \text{unsolvable} \\ \widehat{\text{val}}(w(s)) + 1 & \text{sonst.} \end{cases} \end{aligned}$$

Sei nun ein Zustand $s_C \in S_G$ gegeben. Ein lineares partielles Erfüllbarkeitsproblem mit Schranke s_C zu lösen, bedeutet, einen Zustand $s \in S_G$ zu finden, so dass $\phi^{\text{val}}(s) \geq \phi^{\text{val}}(s_C)$ bzw. $\widehat{\text{val}}(w(s)) \geq \widehat{\text{val}}(w(s_C))$ gilt. Mit anderen Worten wird ein Zustand $s \in S_G$ gesucht, so dass mit $c = w(s) \in \mathbb{N}^I$ und $q_C = \widehat{\text{val}}(w(s_C)) \in \mathbb{N}$ gilt: $\widehat{\text{val}}(c) \geq q_C$.

Die Übersetzung eines linearen partiellen Erfüllbarkeitsproblems mit Schranke s_C (definiert durch einen PDDL2.1* Ausdruck und eine Schranke s_C) in ein klassisches Planungsproblem (definiert durch einen PDDL2.1 Ausdruck) lässt sich daher auf folgende Weise erreichen:

1. Für jede Zielformel g_i wird eine numerische Variable w_i eingeführt.
2. Es wird sichergestellt, dass w_i den Wert 1 annimmt, sobald ein Zustand s eintritt, in dem g_i gilt und dass sonst $w_i = 0$ ist.
3. Die PSP-Beschreibung respektierend wird eine Funktion $\widehat{\text{val}}$ der Variablen w_i konstruiert.
4. Die Schranke s_C wird in einen numerischen Wert q_C übersetzt, wobei $q_C = \widehat{\text{val}}(w(s_C))$ gilt.
5. Als zusätzliches Suchziel wird die numerische Bedingung aufgenommen, dass $\widehat{\text{val}}(w_1, \dots, w_I)$ mindestens so groß wie q_C ist.

Wir werden in den nachfolgenden Abschnitten die Übersetzungsschritte explizit ausführen und anschließend an einem Beispiel verdeutlichen.

4.2.3 Erläuterung der einzelnen Übersetzungsschritte

Zum ersten Schritt: In PDDL2.1 gibt es keine Möglichkeit, numerische Variablen automatisch zu aktualisieren. Nun kann entweder ein derartiges Verfahren emuliert werden, indem Pseudoaktionen der Problembeschreibung hinzugefügt werden, oder man weicht vom generellen Übersetzungsschema im folgenden Punkt ab. Anstatt für jede *Zielbeschreibung* eine numerische Variable einzuführen, wird für jede Atomformel und jeden numerischen Vergleich *innerhalb* einer Zielbeschreibung eine numerische Variable erzeugt. Aus Effizienzgründen wird jedoch sichergestellt, dass für eine Atomformel, möge sie auch in noch so vielen Zielbeschreibungen enthalten sein, nur eine numerische Variable angelegt wird. Die Anzahl der numerischen Variablen hat nämlich einen direkten Einfluß auf die Größe des Suchraumes und damit auf den Umfang der Plansuche.

Nach und nach werden nun die in der PSP-Beschreibung enthaltenen Zielbeschreibungen eingelesen und sukzessive für jede enthaltene Atomformel p und jeden numerischen Vergleich p' eine numerische Variable w bzw. w' angelegt, sofern nicht schon eine solche existiert.

Zur Notation: Die Menge der erhaltenen Atomformeln bzw. numerischen Vergleiche werde mit $P = \{p_i\}_{i \in \{1, \dots, I_P\}}$ bzw. $P' = \{p'_j\}_{j \in \{1, \dots, I_{P'}\}}$ bezeichnet und die Menge der numerischen Variablen mit $W = \{w_i\}_{i \in \{1, \dots, I_P\}} \cup \{w'_j\}_{j \in \{1, \dots, I_{P'}\}}$. Das Anlegen einer numerischen Variable entspricht in PDDL2.1 dem Hinzufügen eines Ausdrucks (`<function-symbol>`) innerhalb der `:functions` Deklaration, wobei `<function-symbol>` durch den Namen der numerischen Variable ersetzt wird. Im nachfolgenden Beispiel wird dies verdeutlicht.

Beispiel. Sei ein LPSP in PDDL2.1* wie folgt spezifiziert. Die enthaltene `:functions` Deklaration ist durch `(:functions (f1) (f2))` gegeben und `(:psp (LEV (CAR p2 (not p2)) (CAR p1 p2)))` ist Teil der PSP-Beschreibung. In ihr kommt die Atomformel p_1 einmal und die Atomformel p_2 dreimal vor. Es werden somit zwei numerische Variablen w_1 und w_2 benötigt. Nach diesem Schritt lautet die `:functions` Deklaration wie folgt: `(:functions (f1) (f2) (w1) (w2))`.

Zum zweiten Schritt: Der zweite Schritt des im letzten Abschnitt vorgestellten generellen Konzepts besteht aus zwei Teilaufgaben. Zunächst werden den numerischen Variablen w_i bzw. w'_j die richtigen Startwerte zugewiesen, d.h. die Werte, die im Zustand s_I gelten. Dies geschieht nach folgendem Schema:

- Für jedes Atom $p_i \in P$ wird geprüft, ob und in welcher Form die Atomformel p_i in der Beschreibung von s_I vorkommt. Sofern p_i als positives Literal (Atomformel) vorkommt, wird die entsprechende numerische Variable w_i auf 1 gesetzt. Anderenfalls, d.h. falls p_i als negierte Atomformel oder gar nicht vorkommt⁶, wird w_i auf 0 gesetzt.
- Für jeden numerischen Vergleich $p'_j \in P'$ wird geprüft, ob er im Startzustand erfüllt ist. Falls er gilt, so wird die entsprechende numerische Variable w'_j auf 1 gesetzt und sonst auf 0. Es wird angenommen, dass jede numerische Variable der PDDL2.1* Beschreibung im Startzustand einen definierten Wert besitzt (ansonsten wird den entsprechenden Variablen ein Wert zugeordnet).

Damit sind die Forderungen $w_i = \text{val}'(s, p_i)$ für Atomformeln p_i und $w'_j = \text{val}'(s, p'_j)$ für numerische Vergleiche p'_j zunächst im Startzustand $s = s_I$ erfüllt. Um sicherzustellen, dass diese Forderungen für alle erreichbaren Zustände s gelten, werden die Aktionen des PDDL2.1* Planungsproblems modifiziert. Bevor wir diese Modifikation betrachten können, müssen wir uns mit einer weiteren Komplikation in PDDL2.1 auseinandersetzen.

In PDDL2.1 werden oft Aktionsschemata verwendet, d.h. Beschreibungen von Klassen von Aktionen. Diese enthalten Formelschemata⁷. Ob eine Atomformel p_i in einer Aktion vorkommt, wird nun dadurch geprüft, dass eine Unifikation der im Aktionsschema enthaltenen Atomschemata mit der Atomformel p_i versucht wird. Gibt es keine derartige Unifikation, so ist p_i sicher in keiner zum Aktionsschema gehörenden Aktion enthalten. Andererseits kann auch der Fall eintreten, dass Unifikationen innerhalb eines Aktionschemas auf verschiedene Arten möglich sind. Dies soll gleich an einem Beispiel verdeutlicht werden. Für einen nume-

⁶Aufgrund der *closed world assumption* sind beide Fälle gleichwertig.

⁷Formeln sind spezielle Formelschemata, die keine freien Variablen enthalten.

rischen Vergleich p'_j wird auf ähnliche Weise getestet, ob er möglicherweise durch eine Aktion verändert werden kann. Dazu wird für jede in p'_j vorkommende numerische Variable geprüft, ob sie im Effekt eines Aktionsschemas verändert wird. Zuvor werden einige Definitionen benötigt.

Definition 23 (Literalunifikator). Sei u ein Paar (σ, not) . Dabei ist σ entweder die Identitätssubstitution ι oder ein Ausdruck der Form $(\text{and } (= \langle \text{variable} \rangle \langle \text{name} \rangle)^+)$, der eine Substitution von Variablen durch Objekte beschreibt. Das Ergebnis der Anwendung der Substitution auf ein Literalschema l_s wird mit $l_s[\sigma]$ bezeichnet und es gelte $l_s = l_s[\iota]$. Weiterhin sei $not \in \{0, 1\}$. Das Paar u heißt Literalunifikator zwischen einem Literalschema l_s und einer Atomformel p , sofern gilt:

$$l_s[\sigma] = \begin{cases} p & \text{falls } not = 0 \\ (\text{not } p) & \text{falls } not = 1. \end{cases}$$

Beispiel. Sei folgendes Formelschema $f_s = (\text{and } (\text{at } ?x ?y) (\text{not } (\text{at } ?x ?z)) (\text{increase LOAD}(?x) 10))$, das z.B. Effektbeschreibung eines bedingten Effektes sein kann, sowie die Atomformel $p_1 = (\text{at TRUCKO PLACEO})$ gegeben. Bei der Unifikation wird nun geprüft, ob für die in f_s enthaltenen maximalen⁸ Literalschemata l_s ein Literalunifikator existiert. In diesem Beispiel existieren zwei Stück:

$$\begin{aligned} u_1 &= ((\text{and } (= ?x TRUCKO) (= ?y PLACEO)), 0), \\ u_2 &= ((\text{and } (= ?x TRUCKO) (= ?z PLACEO)), 1). \end{aligned}$$

Wichtig ist auch folgende Bemerkung. Seien für eine gegebene Atomformel p und ein Formelschema f_s alle Literalunifikatoren der enthaltenen maximalen Literalschemata berechnet worden. Dann kann ein Literalunifikator $u = (\sigma, not)$ durchaus mehrfach ermittelt worden sein, denn p kann mit verschiedenen Literalen aus f_s unifiziert werden. Ein Literalunifikator $u' = (\sigma', not')$ mit $\sigma = \sigma'$ und $not \neq not'$ kann jedoch nicht vorhanden sein, denn sonst wäre der Effekt widersprüchlich und die Wirkung der Überföhrungsfunktion $\gamma : S \times A \rightarrow S$ nicht definiert, vgl. Abschnitt 1.3.3.

In PDDL2.1 besitzen Effekte von Aktionen folgende einfache Form:

- Jede Aktion besitzt einen Effekt. Dieser Effekt werde als Haupteffekt bezeichnet.
- Jeder Haupteffekt ist entweder leer oder ein Untereffekt oder eine Konjunktion von Untereffekten.
- Ein Untereffekt ist entweder bedingt oder unbedingt. Ein unbedingter Effekt lässt sich in eine Konjunktion von Literalen und numerischen Effekten umformen. Ein bedingter Effekt ist entweder
 - ein **when** Effekt, dann besteht er aus einer Bedingung und einer Effektbeschreibung, die eine Konjunktion von Literalen und numerischen Effekten ist. Somit lassen sich **when** Effekte nicht schachteln.
 - ein **forall** Effekt, der freie Variablen bindet und dessen Effektbeschreibung ein Untereffekt ist.
- Jeder Haupteffekt einer Aktion kann nach Elimination der **forall** Effekte als Konjunktion von **when** Effekten⁹ dargestellt werden.

⁸Dies bedeutet an dieser Stelle, dass die betrachteten Literalschemata keine Teilformelschemata von Literalschemata sein dürfen. So ist z.B. $(\text{not } (\text{at } ?x ?z))$ ein maximales Literalschema, nicht jedoch das enthaltene Literalschema $(\text{at } ?x ?z)$.

⁹Deren Vorbedingung ist gegebenenfalls leer, d.h. immer erfüllt.

Es wird nun für jedes Paar bestehend aus einer Atomformel p_i und einem Aktionsschema a die folgende Modifikation des Haupteffekts von a durchgeführt.

- Der Haupteffekt eff eines Aktionsschemas wird in eine Konjunktion von bedingten Effekten überführt. Jede Effektbeschreibung der bedingte Effekte wird in eine Konjunktion von Literalen und numerischen Effekten umgeformt. Dann erfolgt für jeden bedingten Effekt (**when** $cond_j$ eff_j) eine Unifikation zwischen der gegebenen Atomformel p_i und Literalschemata aus der jeweiligen Effektbeschreibung eff_j des bedingten Effekts. Man erhält so eine Menge $U_j = \{u_j^k\}$ von Literalunifikatoren, die p_i mit Literalschemata aus eff_j unifizieren.
- Für jeden Literalunifikator $u_j^k = (\sigma, not)$ aus U_j ist nun folgende Veränderung an der Effektbeschreibung eff_j des jeweiligen bedingten Effektes des Aktionsschemas vorzunehmen:
 - Ein bedingter Effekt eff^* der Form (**when** $cond'$ eff') wird erzeugt.
 - Falls $\sigma \neq \iota$ gilt, ist $cond'$ durch $cond' = (\text{and } \sigma \text{ } cond_j)$ gegeben. Anderenfalls wird $cond' = cond_j$ gesetzt.
 - Der Effekt eff' des neuen **when** Knotens ist ein numerischer Effekt der Form (**assign** w_i not') mit $not' = 1 - not$, $not' \in \mathbb{N}$.
- Der bedingte Effekte eff^* wird zum ursprünglichen Effekt eff des Aktionsschemas hinzugefügt, d.h. der ursprüngliche Effekt eff der Aktion a wird durch den Effekt (**and** eff eff^*) ersetzt.

Anschließend werden für alle Paare bestehend aus einem numerischen Vergleich p'_j und einem Aktionsschema a folgende Schritte durchgeführt. In einigen Punkten verläuft dieser Teil der Übersetzung analog zum eben vorgestellten Verfahren für Atomformeln p_i . Es wird daher, wann immer möglich, eine verkürzte Beschreibung gewählt:

- Sofern noch nicht erfolgt, wird der Haupteffekt des Aktionsschemas in eine Konjunktion von bedingten Effekten umgeformt.
- Für jede im gegebenen numerischen Vergleich p'_j enthaltene numerische Variable v wird geprüft, ob sie in der jeweiligen Effektbeschreibung eines bedingten Effekts eff verändert wird. Dies kann für eine Menge $\{v_l\}_{l \in \{1, \dots, L\}}$ derartiger Variablen zutreffen. Sei mit $comp$ der Vergleichsoperator des numerischen Vergleiches p'_j und mit ass_l derjenige Zuweisungsoperator innerhalb der Effektbeschreibung von eff gemeint, der v_l verändert¹⁰. Die Zuweisung selbst ist ein numerischer Effekt und wird mit n_l bezeichnet. Das *Ergebnis* der Zuweisung $n_l = (ass_l \ v_l \ r_l)$ wird durch die Schreibweise $n_l(v_l)$ notiert und ist wie folgt definiert:

$$n_l(v_l) = r_l, \text{ falls } ass_l = \text{assign},$$

$$n_l(v_l) = (+ \ v_l \ r_l), \text{ falls } ass_l = \text{increase},$$

$$n_l(v_l) = (- \ v_l \ r_l), \text{ falls } ass_l = \text{decrease},$$

¹⁰Der Einfachheit halber nehmen wir hier an, dass es für jede Variable v_l nur einen derartigen Zuweisungsoperator gibt. Falls doch mehrere Zuweisungsoperatoren existieren, so besteht ein möglicher Ausweg darin, alle Zuweisungen, welche die Variable v_l verändern, zu einer Zuweisung zusammenzufassen. Dies sollte aufgrund der Konsistenzbedingungen der numerischen Effekte, vgl. Abschnitt 1.3.3, stets durchführbar sein.

$n_l(v_l) = (* v_l r_l)$, falls $ass_l = \text{scale-up}$,

$n_l(v_l) = (/ v_l r_l)$, falls $ass_l = \text{scale-down}$.

Die Substitution der Variable v_l im numerischen Vergleich p'_j durch einen anderen Ausdruck d_l wird mit $p'_j[v_l/d_l]$ angegeben. Die Angabe $p'_j[v_1/d_1] \cdots [v_L/d_L]$ beschreibt eine Hintereinanderausführung mehrerer derartiger Substitutionen. Ferner wird mit $cond$ die Bedingung des gegebenen bedingten Effekts eff bezeichnet und die zu p'_j gehörige numerische Variable mit w'_j . Es werden nun zwei bedingte Effekte der Form $(\text{when } cond_1 \text{ } eff_1)$ und $(\text{when } cond_2 \text{ } eff_2)$ erzeugt, wobei gilt

$$cond_1 = (\text{and } p'_j[v_1/n_1(v_1)] \cdots [v_L/n_L(v_L)] \text{ } cond), \text{ } eff_1 = (\text{assign } w'_j \text{ } 1),$$

$$cond_2 = (\text{and } (\text{not } p'_j[v_1/n_1(v_1)] \cdots [v_L/n_L(v_L)]) \text{ } cond), \text{ } eff_2 = (\text{assign } w'_j \text{ } 0).$$

- Diese Effekte werden zum Haupteffekt des Aktionsschema a hinzugefügt, d.h. die entsprechende Konjunktion von bedingten Effekten wird um diese neuen Effekte erweitert.

Durch die Modifikation der Aktionen und Aktionsschemata und das Setzen der Anfangswerte der numerischen Variablen wurde erreicht, dass:

- jede ausgeführte Aktion die numerische Variable w bzw. w' auf 1 setzt, falls die Aktion in einen Zustand überführt, in dem die zugehörige Atomformel p bzw. der zugehörige numerische Vergleich p' gilt.
- jede ausgeführte Aktion die numerische Variable w bzw. w' auf 0 setzt, falls die Aktion in einen Zustand überführt, in dem die zugehörige Atomformel p bzw. der zugehörige numerische Vergleich p' nicht gilt.
- die Anfangswerte jeder numerischen Variablen w und w' konsistent mit der Ausgangsbeschreibung s_I unter Verwendung der *closed world assumption* sind.

Es ist somit sichergestellt, dass w bzw. w' genau dann 1 ist, wenn p bzw. p' im gegebenen Zustand gilt. Sonst ist $w = 0$ bzw. $w' = 0$.

Zum dritten Schritt: Zur Konstruktion der Funktion $\widehat{\text{val}}$ wird die PSP-Beschreibung rekursiv durch einen arithmetischen Ausdruck in den numerischen Variablen w_i, w'_j ersetzt. Dazu werden einfach die rekursiven Definitionen der Funktionen val und maxval verwendet, vgl. Tabelle 3.1. Genauer gilt:

Definition 24 (Arithmetische Übersetzung). Eine Abbildung θ , die einer PDDL2.1* PSP-Beschreibung n , welche die Menge P von Atomformeln und die Menge P' von numerischen Effekten enthält, einen arithmetischen Ausdruck $\theta[n]$ in den numerischen Variablen W zuordnet, heißt arithmetische Übersetzung,

wenn sie die Rekursionstruktur von Tabelle 3.1 respektiert. D.h wenn gilt¹¹:

$$\begin{aligned}
\theta[d] &= d, \quad d \in \mathbb{N}, \\
\theta[p_i] &= w_i, \quad p_i \in P, w_i \in W, \\
\theta[p'_j] &= w'_j, \quad p_j \in P', w'_j \in W, \\
\theta[(\text{not } n_1)] &= (- \ 1 \ \theta[n_1]), \\
\theta[(\text{and } n_1 \dots n_k)^{\text{:psp-full}}] &= (* \ \theta[n_1] \dots \theta[n_k]), \\
\theta[(\text{or } n_1 \dots n_k)^{\text{:psp-full}}] &= (- \ 1 \ (* \ (- \ 1 \ \theta[n_1]) \dots (- \ 1 \ \theta[n_k])), \\
\theta[(\text{MULT } n_1 \ n_2)] &= (* \ \theta[n_1] \ \theta[n_2]), \\
\theta[(\text{MULT } n_1 \dots n_k)^{\text{:psp-full}}] &= (* \ \theta[n_1] \dots \theta[n_k]), \\
\theta[(\text{CAR } n_1 \dots n_k)] &= (+ \ \theta[n_1] \dots \theta[n_k]), \\
\theta[(\text{LEV } n_1 \dots n_k)] &= (+ \ (* \ c_1 \ \theta[n_1]) \dots (* \ c_k \ \theta[n_k]))
\end{aligned}$$

wobei die Gewichte $\{c_l\}_{l \in \{1, \dots, k\}}$, wie in Tabelle 3.1 angegeben, mithilfe der `maxval` Funktion bestimmt werden. Man beachte, dass aufgrund der in Abschnitt 4.2.1 spezifizierten Syntax `and`, `or` und `not` Knoten nur innerhalb von Formelausdrücken (die eventuell numerische Vergleiche beinhalten) vorkommen können.

Man erhält so aus einer PSP-Beschreibung n einen arithmetischen Ausdruck $\theta[n]$, bestehend aus Klammern $(,)$, den Operatoren $*$, $+$, natürlichen Zahlen und numerischen Variablen w_i, w'_j , d.h. einen numerischen Ausdruck im Sinne der Sprache PDDL2.1, vgl. Abschnitt 1.3.3.

Satz 17. *Der numerische Ausdruck $\theta[n]$ einer PSP-Beschreibung n besitzt folgende Eigenschaften:*

- Für jede Belegung der Variablen $w_i \in \{0, 1\}$, $w'_j \in \{0, 1\}$ ist die Auswertung des Ausdrucks $\theta[n](w_1, \dots, w_{I_P}, w'_1, \dots, w'_{I_{P'}})$ größer oder gleich 0.
- $\theta[n]$ ist ein linearer Ausdruck in den numerischen Variablen $w_1, \dots, w_{I_P}, w'_1, \dots, w'_{I_{P'}}$, vgl. Abschnitt 1.3.3, wenn keine der hier mit `:psp-full` markierten Knoten in n vorkommen.

Beweis. Beide Eigenschaften können induktiv bewiesen werden. Die erste Eigenschaft folgt aus der Nichtnegativität der Gewichte c_l und daraus, dass als Unterknoten von `and`, `or` und `not` Knoten lediglich Atomformeln oder numerische Vergleiche zugelassen sind. Ein derartiger Unterknoten kann aber nur den Wert 0 oder 1 annehmen, mithin können die angegebenen Ausdrücke der Form $(- \ 1 \ \theta[n_i])$ keine negativen Werte annehmen.

Die zweite Eigenschaft wird sofort klar, wenn man bedenkt, dass in den erlaubten (binären) `MULT` Knoten der zweite Unterknoten stets eine natürliche Zahl ist (siehe Abschnitt 4.2.1) und selbiges für die Gewichte c_l gilt. □

¹¹Die Sprache PDDL2.1 ist leider im Umgang mit verschachtelten Ausdrücken nicht ganz konsistent. So sind beispielsweise `and` Verknüpfungen mit beliebig vielen Formeln erlaubt. Andererseits sind Produkte bzw. Summen mit einer beliebigen Anzahl von Faktoren bzw. Summanden in der Syntaxbeschreibung nicht vorgesehen, obwohl diese Operationen assoziativ sind. An dieser Stelle wird von dieser künstlichen Einschränkung der PDDL2.1 Sprache abgesehen. Die Umformung eines arithmetischen Ausdrucks, der aus einem assoziativen Operator und mehr als zwei Argumenten besteht, in einen verschachtelten Ausdruck, in dem nur eine binäre Variante des Operators vorkommt, ist sehr einfach durchführbar.

Zum vierten Schritt: Sofern eine Schranke tatsächlich als Zustand s_C gegeben wäre, könnten die Werte der numerischen Variablen w_i, w'_j in diesem Zustand bestimmt werden und anschließend eine Auswertung des numerischen Ausdrucks $\theta[n](w_1, \dots, w_{I_P}, w'_1, \dots, w'_{I_{P'}})$ erfolgen. Diese Auswertung ergäbe die Zahl q_C . Eine Inspektion von Algorithmus 1 zeigt jedoch, dass es nicht nötig ist, einen Zustand s_C anzugeben. Vielmehr verwendet der Algorithmus natürliche Zahlen, d.h. direkt q_C , als Schranke.

Zum fünften Schritt: Sei `(:goal g)` die Zielbedingung des in PDDL2.1* beschriebenen Planungsproblems, n die gegebene PSP-Beschreibung und sei g^* der durch $(>= \theta[n] q_C)$ definierte numerische Vergleich. Dann ergibt sich die Zielbedingung des in PDDL2.1 beschriebenen klassischen Planungsproblems zu `(:goal (and g g*))`.

Bemerkung. Die Übersetzung eines PDDL2.1* Problems in PDDL2.1 erfolgt nach den eben ausgeführten Schritten 1 – 5. Dabei gilt:

- Aktionen, Funktionen, Zielbedingungen und die Beschreibung des Startzustandes werden während des Übersetzungsprozesses modifiziert und so in die PDDL2.1 Beschreibung übernommen.
- Die PSP-Beschreibung des PDDL2.1* Problems wird in der Übersetzung weggelassen.
- Alle übrigen PDDL2.1* Sprachelemente werden unverändert übernommen.

In einer Implementation des Algorithmus 1 muss der vollständige Übersetzungsprozess des PDDL2.1* Problems mit Schranke q_C in ein PDDL2.1 Problem natürlich nicht für jede Änderung der Schranke q_C erfolgen. Vielmehr reicht es aus, die Zielbedingung g^* entsprechend anzupassen.

4.2.4 Ein Beispiel

In diesem Abschnitt soll die im letzten Abschnitt vorgestellte Transformation exemplarisch durchgeführt werden. Es wird dazu ein realistisches, im Planungswettbewerb IPC3 verwendetes, Planungsproblem aus der Depot-Domäne verwendet. Diese Domäne ist etwas einfacher, übersichtlicher und deshalb für ein einfaches Beispiel besser geeignet als die in Abschnitt 1.3 in Auszügen vorgestellte Satelliten-Domäne. Die Domäne verbindet die Problemstellung aus der Logistik- und Blockwelt-Domäne und lässt sich kurz wie folgt beschreiben: Die Objekte in einem entsprechenden Planungsproblem sind LKWs (*trucks*), Paletten (*pallets*), Orte (*places*), Container (*blocks*) und Kräne (*hoists*). Von einer Ausgangssituation ausgehend, besteht das Ziel eines klassischen Planungsproblems darin, Container mittels LKWs an bestimmte Orte zu bringen und dort mithilfe von Kränen in vorgegebenen Reihenfolgen auf Paletten zu stapeln. In der hier verwendeten numerischen Variante besitzen die LKWs eine endliche Ladekapazität. Sie begrenzt die Menge der auf einmal transportierbaren Container, da diese ein endliches Gewicht besitzen. Zudem verbrauchen LKWs und Kräne Treibstoff, wenn sie einen Transport vornehmen. Als Nebenbedingung wird ein Plan gesucht, der möglichst wenig Treibstoff verbraucht.

Aus einem derartigen klassischen Planungsproblem wird durch die zusätzliche Angabe einer PSP-Beschreibung ein partiell erfüllbares Planungsproblem. Es sind nun Domänen-, Fakten- und PSP-Beschreibung eines derartigen Problem in PDDL2.1* angegeben.

Man beachte, dass die Sprachen PDDL2.1 und PDDL2.1* bzw. PDDL2.1/* nicht zwischen Groß- und Kleinschreibung unterscheiden.

Die ursprüngliche Domänenbeschreibung:

```
(define (domain Depot)
  (:requirements :typing :fluents)
  (:types place locatable - object
    depot distributor - place
    truck hoist surface - locatable
    pallet crate - surface)
  (:predicates (at ?x - locatable ?y - place)
    (on ?x - crate ?y - surface)
    (in ?x - crate ?y - truck)
    (lifting ?x - hoist ?y - crate)
    (available ?x - hoist)
    (clear ?x - surface)
  )
  (:functions
    (load_limit ?t - truck)
    (current_load ?t - truck)
    (weight ?c - crate)
    (fuel-cost)
```

```

)
(:action Drive
:parameters (?x - truck ?y - place ?z - place)
:precondition (and (at ?x ?y))
:effect (and (not (at ?x ?y)) (at ?x ?z)
            (increase (fuel-cost) 10)))

(:action Lift
:parameters (?x - hoist ?y - crate ?z - surface ?p - place)
:precondition (and (at ?x ?p) (available ?x) (at ?y ?p) (on ?y ?z) (clear ?y))
:effect (and (not (at ?y ?p)) (lifting ?x ?y) (not (clear ?y)) (not (available ?x))
            (clear ?z) (not (on ?y ?z)) (increase (fuel-cost) 1)))

(:action Drop
:parameters (?x - hoist ?y - crate ?z - surface ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (clear ?z) (lifting ?x ?y))
:effect (and (available ?x) (not (lifting ?x ?y)) (at ?y ?p) (not (clear ?z)) (clear ?y)
            (on ?y ?z)))

(:action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (lifting ?x ?y)
                (<= (+ (current_load ?z) (weight ?y)) (load_limit ?z)))
:effect (and (not (lifting ?x ?y)) (in ?y ?z) (available ?x)
            (increase (current_load ?z) (weight ?y))))

(:action Unload
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (available ?x) (in ?y ?z))
:effect (and (not (in ?y ?z)) (not (available ?x)) (lifting ?x ?y)
            (decrease (current_load ?z) (weight ?y))))
)

```

Die ursprüngliche Faktenbeschreibung:

```

(define (problem depotprob1818) (:domain Depot)
(:objects
  depot0 - Depot
  distributor0 distributor1 - Distributor
  truck0 truck1 - Truck
  pallet0 pallet1 pallet2 - Pallet
  crate0 crate1 - Crate
  hoist0 hoist1 hoist2 - Hoist)
(:init
  (at pallet0 depot0)
  (clear crate1)

```

```

    (at pallet1 distributor0)
    (clear crate0)
    (at pallet2 distributor1)
    (clear pallet2)
    (at truck0 distributor1)
    (= (current_load truck0) 0)
    (= (load_limit truck0) 323)
    (at truck1 depot0)
    (= (current_load truck1) 0)
    (= (load_limit truck1) 220)
    (at hoist0 depot0)
    (available hoist0)
    (at hoist1 distributor0)
    (available hoist1)
    (at hoist2 distributor1)
    (available hoist2)
    (at crate0 distributor0)
    (on crate0 pallet1)
    (= (weight crate0) 11)
    (at crate1 depot0)
    (on crate1 pallet0)
    (= (weight crate1) 86)
    (= (fuel-cost) 0)
)
(:goal (and
        (on crate0 pallet2)
        (on crate1 pallet1)
    )
)
(:metric minimize (fuel-cost))

```

Die PSP-Beschreibung:

```

(define
  (pspname PSP001)
  (:problem depotprob1818)
  (:domain Depot)
  (:goal (and
          (at truck1 depot0)
        )
  )
)
(:psp (LEV (CAR (at crate0 distributor0)

```

```

                (at crate0 distributor1)
                (on crate1 pallet1)
            )
        (CAR
            (available hoist0)
            (NOT (at truck1 DEPOT0) )
        )
        (NOT (at truck0 distributor1) )
        (NOT (at crate0 distributor0) )
    )
)
)

```

Offenbar sind nicht alle Ziele der PSP-Beschreibung gleichzeitig erfüllbar, da z.B. die Atomformel

```
(at crate0 distributor0)
```

als positives und negatives Literal enthalten ist. Mit der Angabe eines Zieles im `:goal` Feld wurden entsprechende Angaben aus der ursprünglichen Faktenbeschreibung überschrieben. Das Ziel

```
(at truck1 depot0)
```

muss somit für jede Lösung des partiellen Erfüllbarkeitsproblems erfüllt sein. Im Rahmen einer PSP-Beschreibung können demnach bequem neben optionalen Zielen auch obligatorische Ziele spezifiziert werden.

Die transformierte Domänenbeschreibung:

```

(define (domain DEPOT)
  (:requirements
    :TYPING
    :EQUALITY
    :CONDITIONAL-EFFECTS
    :FLUENTS
  )
  (:types
    PLACE - OBJECT
    LOCATABLE - OBJECT
    DEPOT - PLACE
    DISTRIBUTOR - PLACE
    TRUCK - LOCATABLE
    HOIST - LOCATABLE
    SURFACE - LOCATABLE
    PALLET - SURFACE
    CRATE - SURFACE
  )
)

```

```

)
(:predicates
  (AT ?X - LOCATABLE ?Y - PLACE )
  (ON ?X - CRATE ?Y - SURFACE )
  (IN ?X - CRATE ?Y - TRUCK )
  (LIFTING ?X - HOIST ?Y - CRATE )
  (AVAILABLE ?X - HOIST )
  (CLEAR ?X - SURFACE )
)
(:functions
  (LOAD_LIMIT ?T - TRUCK )
  (CURRENT_LOAD ?T - TRUCK )
  (WEIGHT ?C - CRATE )
  (FUEL-COST )
  (PSP_V-1 )
  (PSP_V-2 )
  (PSP_V-3 )
  (PSP_V-4 )
  (PSP_V-5 )
  (PSP_V-6 )
  (PSP_START_METRIC )
  (PSP_MAX_METRIC )
)
(:action UNLOAD
  :parameters (?X - HOIST ?Y - CRATE ?Z - TRUCK ?P - PLACE )
  :precondition
    (and (AT ?X ?P) (AT ?Z ?P) (AVAILABLE ?X) (IN ?Y ?Z))
  :effect
    (and (not (IN ?Y ?Z))(not (AVAILABLE ?X))
          (LIFTING ?X ?Y)(decrease (CURRENT_LOAD ?Z)(WEIGHT ?Y))
          (when (and (= ?X HOIST0))
                (and (assign (PSP_V-4) 0))
              )
        )
    )
)
(:action LOAD
  :parameters (?X - HOIST ?Y - CRATE ?Z - TRUCK ?P - PLACE )
  :precondition
    (and (AT ?X ?P)(AT ?Z ?P)(LIFTING ?X ?Y)
          (<= (+ (CURRENT_LOAD ?Z)(WEIGHT ?Y))(LOAD_LIMIT ?Z))
        )
  :effect

```

```

        (and (not (LIFTING ?X ?Y))(IN ?Y ?Z)(AVAILABLE ?X)
(increase (CURRENT_LOAD ?Z)(WEIGHT ?Y))
        (when (and (= ?X HOISTO))
          (and (assign (PSP_V-4) 1))
        )
      )
    )
  (:action DROP
    :parameters (?X - HOIST ?Y - CRATE ?Z - SURFACE ?P - PLACE )
    :precondition
      (and (AT ?X ?P)(AT ?Z ?P)(CLEAR ?Z)(LIFTING ?X ?Y))
    :effect
      (and (AVAILABLE ?X)(not (LIFTING ?X ?Y))
        (AT ?Y ?P)(not (CLEAR ?Z))
        (CLEAR ?Y)(ON ?Y ?Z)
        (when (and (= ?Y CRATEO)(= ?P DISTRIBUTORO))
          (and (assign (PSP_V-1) 1))
        )
        (when (and (= ?Y CRATEO)(= ?P DISTRIBUTOR1))
          (and (assign (PSP_V-2) 1))
        )
        (when (and (= ?Y CRATE1)(= ?Z PALLET1))
          (and (assign (PSP_V-3) 1))
        )
        (when (and (= ?X HOISTO))
          (and (assign (PSP_V-4) 1))
        )
      )
    )
  (:action LIFT
    :parameters (?X - HOIST ?Y - CRATE ?Z - SURFACE ?P - PLACE )
    :precondition
      (and (AT ?X ?P)(AVAILABLE ?X)(AT ?Y ?P)(ON ?Y ?Z)(CLEAR ?Y))
    :effect
      (and (not (AT ?Y ?P))(LIFTING ?X ?Y)(not (CLEAR ?Y))
        (not (AVAILABLE ?X))(CLEAR ?Z)(not (ON ?Y ?Z))
        (increase (FUEL-COST) 1)
        (when (and (= ?Y CRATEO)(= ?P DISTRIBUTORO))
          (and (assign (PSP_V-1) 0))
        )
        (when (and (= ?Y CRATEO)(= ?P DISTRIBUTOR1))
          (and (assign (PSP_V-2) 0))
        )
      )
  )

```

```

    )
    (when (and (= ?Y CRATE1)(= ?Z PALLET1))
      (and (assign (PSP_V-3) 0))
    )
    (when (and (= ?X HOISTO))
      (and (assign (PSP_V-4) 0))
    )
  )
)
(:action DRIVE
  :parameters (?X - TRUCK ?Y - PLACE ?Z - PLACE )
  :precondition
    (and (AT ?X ?Y))
  :effect
    (and (not (AT ?X ?Y))(AT ?X ?Z)(increase (FUEL-COST) 10)
      (when (and (= ?X TRUCK1)(= ?Y DEPOTO))
        (and (assign (PSP_V-5) 0))
      )
      (when (and (= ?X TRUCK1)(= ?Z DEPOTO))
        (and (assign (PSP_V-5) 1))
      )
      (when (and (= ?X TRUCKO)(= ?Y DISTRIBUTOR1))
        (and (assign (PSP_V-6) 0))
      )
      (when (and (= ?X TRUCKO)(= ?Z DISTRIBUTOR1))
        (and (assign (PSP_V-6) 1))
      )
    )
)
)
)

```

In der transformierten Domänenbeschreibung fällt, im Vergleich zur ursprünglichen Fassung, die Einführung von numerischen Variablen auf, die mit `PSP_` beginnen. Die Variable `PSP_START_METRIC` bzw. `PSP_MAX_METRIC` definiert die Grenzzahl q_C bzw. den `maxval` Wert der gegebenen PSP-Beschreibung. Die anderen numerischen Variablen der Form `PSP_V-*` entsprechen den in Abschnitt 4.2.2 eingeführten Variablen w_i und korrespondieren auf die in Abschnitt 4.2.3 erläuterte Weise mit den in der PSP-Beschreibung enthaltenen Atomformeln. Notwendige Änderungen der Werte dieser Variablen aufgrund von Zustandsübergängen werden durch die Einführung von bedingten Effekten sichergestellt. Diese Effekte besitzen die Gestalt (`when cond eff`). Die in der Bedingung `cond` angegebene Unifikation kann bei einer Grundinstanziierung der Aktionsschemata direkt berücksichtigt werden. Die Grundinstanziierung eines Aktionsschemas entspricht der Erzeugung einer Menge von Aktionen, indem alle im Schema enthaltenen Objektvariablen auf jede möglich Weise durch Objekte ersetzt werden. Insbesondere ist in den entstehen-

den Aktionen dann kein bedingter Effekt mehr nötig, wenn in der ursprünglichen Domänenbeschreibung keine bedingten Effekte verwendet wurden.

Die transformierte Faktenbeschreibung:

```
(define (problem DEPOTPROB1818)
  (:domain DEPOT)
  (:objects
    DEPOTO - DEPOT
    DISTRIBUTORO - DISTRIBUTOR
    DISTRIBUTOR1 - DISTRIBUTOR
    TRUCKO - TRUCK
    TRUCK1 - TRUCK
    PALLETO - PALLET
    PALLET1 - PALLET
    PALLET2 - PALLET
    CRATEO - CRATE
    CRATE1 - CRATE
    HOISTO - HOIST
    HOIST1 - HOIST
    HOIST2 - HOIST
  )
  (:init
    (AT PALLETO DEPOTO)
    (CLEAR CRATE1)
    (AT PALLET1 DISTRIBUTORO)
    (CLEAR CRATEO)
    (AT PALLET2 DISTRIBUTOR1)
    (CLEAR PALLET2)
    (AT TRUCKO DISTRIBUTOR1)
    (= (CURRENT_LOAD TRUCKO) 0)
    (= (LOAD_LIMIT TRUCKO) 323)
    (AT TRUCK1 DEPOTO)
    (= (CURRENT_LOAD TRUCK1) 0)
    (= (LOAD_LIMIT TRUCK1) 220)
    (AT HOISTO DEPOTO)
    (AVAILABLE HOISTO)
    (AT HOIST1 DISTRIBUTORO)
    (AVAILABLE HOIST1)
    (AT HOIST2 DISTRIBUTOR1)
    (AVAILABLE HOIST2)
    (AT CRATEO DISTRIBUTORO)
```

```

(ON CRATE0 PALLET1)
(= (WEIGHT CRATE0) 11)
(AT CRATE1 DEPOTO)
(ON CRATE1 PALLETO)
(= (WEIGHT CRATE1) 86)
(= (FUEL-COST) 0)
(= (PSP_V-1) 1)
(= (PSP_V-2) 0)
(= (PSP_V-3) 0)
(= (PSP_V-4) 1)
(= (PSP_V-5) 1)
(= (PSP_V-6) 1)
(= (PSP_START_METRIC) 1)
(= (PSP_MAX_METRIC) 47)
)
(:goal
  (and (AT TRUCK1 DEPOTO)
        (<= (PSP_START_METRIC)(+ (* 1(+ (PSP_V-1)(+ (PSP_V-2)(PSP_V-3))))
    (+ (* 4(+ (PSP_V-4)(+ 1(- (PSP_V-5)))))(+ (* 12(+ 1(- (PSP_V-6))))
    (+ (* 24(+ 1(- (PSP_V-1))))(* 48 0))))))
  )
)
(:metric MINIMIZE
  (FUEL-COST)
)
)

```

In dieser transformierten Faktenbeschreibung ist der Grenzwert $q_C = 1$ gesetzt. Während der Laufzeit des Perl-Skripts `SolvePSP` wird dieser Wert durch eine einfache Substitution des Ausdruckes

```
(= (PSP_START_METRIC) 1)
```

mit

```
(= (PSP_START_METRIC)  $\tilde{q}_C$ )
```

verändert. Hierbei gibt \tilde{q}_C den neuen Grenzwert an. Die Werte der eingeführten numerischen Variablen `PSP_V-*` sind außerdem so gesetzt, dass sie mit der Ausgangssituation konsistent sind.

4.2.5 Implementation

Im Rahmen dieser Arbeit wurde zu Testzwecken ein Programm entwickelt, das den Algorithmus 1 implementiert und die Übersetzung einer PDDL2.1* Beschreibung in ein PDDL Problem vornimmt. Es besteht aus folgenden Teilen:

- einem Perl-Skript `SolvePSP`, welches die allgemeine Programmlogik, wie in Algorithmus 1 dargestellt, implementiert.
- einem Übersetzungsprogramm `CreatePSP`, welches das LPSP* (beschrieben durch einen PDDL2.1* Ausdruck und einer Schranke) in ein klassisches Planungsproblem (beschrieben durch einen PDDL2.1 Ausdruck) überführt. Dies geschieht auf die in den vorangegangenen Abschnitten vorgestellte Weise.

Der Quellcode des Perl-Skripts `SolvePSP` und des Programms `CreatePSP` findet sich auf der Homepage des Autors und damit momentan unter folgender Internet-Adresse:

<http://www.physik.uni-leipzig.de/~feldmann>

Das Perl-Skript `SolvePSP`:

Eine vollständige Liste der Programmparameter erhält man durch die Eingabe
> `SolvePSP -h`.

Der Quellcode dieses Programms findet sich in Anhang B.2.

Das Skript verwendet das Programm `CreatePSP` um einen PDDL2.1* Ausdruck in einen entsprechenden PDDL2.1 Ausdruck zu übersetzen. Zudem wird als klassischer Planer das Programm `Metric-FF` (geschrieben von Jörg Hoffmann, siehe [41]) eingesetzt. Eine kleine Modifikation des Programms wurde vorgenommen, damit der Planer den erreichten Grenzwert (der besonders anfangs oft wesentlich größer als der geforderte Grenzwert q_C ist) ausgibt. Dazu wurde im Hauptprogramm von `Metric-FF` eine Bibliothek eingebunden, welche auf die internen Datenstrukturen des Planers zugreift und die gewünschten Werte herausschreibt. In Anhang B.1 ist eine entsprechende Modifikation von `Metric-FF`, sowie der Quellcode der Bibliothek angegeben.

Das Übersetzungsprogramm `CreatePSP`:

Eine vollständige Liste der Programmparameter erhält man durch die Eingabe
> `CreatePSP -h`.

Die genaue Funktionsweise des Programms kann dem Quellcode entnommen werden, welcher aus Platzgründen nicht im Anhang wiedergegeben ist, sondern unter der oben genannten Internet-Adresse gefunden werden kann.

Bevor die Struktur des Hauptprogrammes von `CreatePSP` vorgestellt wird, sind folgende Bemerkungen angebracht:

- Das Programm ist in der Programmiersprache C geschrieben und wurde mit `gcc (GCC) 3.3.5 (Debian 1:3.3.5-2)` kompiliert.
- Die Syntax des Parsers und diverse Datenstrukturen, sowie ein großer Teil des Programmcodes wurde dem Programm `Metric-FF` (implementiert durch Jörg Hoffmann) entlehnt.

- Der Parser verwendet die *open source* Programme `flex version 2.5.4` und `bison (GNU Bison) version 1.75`.
- Aufgrund der Anlehnung an `Metric-FF`, welches lediglich lineare numerische Ausdrücke erlaubt und den Erkenntnissen aus Satz 17, erfolgt die PSP-Beschreibung in der Sprache `PDDL2.1*`.
- Das Verwenden numerischer Vergleiche in der PSP-Beschreibung wurde nicht implementiert und ist daher nicht zugelassen.

Das Programm implementiert die wesentlichen Schritte aus Abschnitt 4.2.2 bzw. 4.2.3. Das Hauptprogramm von `CreatePSP` besitzt daher folgende Gestalt:

1. Einlesen des `PDDL2.1*` Problems und Erkennen der atomaren Formeln der PSP Beschreibung. Tatsächlich wird die Problembeschreibung auf drei Dateien aufgeteilt. Eine beinhaltet die Domänenbeschreibung, die zweite die Faktenbeschreibung der Probleminstanz und in der dritten ist die PSP Beschreibung spezifiziert.
2. Die im letzten Abschnitt im Detail vorgestellten Übersetzungsschritte werden ausgeführt.
3. Eine modifizierte Domänenbeschreibung und eine modifizierte Faktenbeschreibung werden herausgeschrieben. Die Grenzzahl q_C ist hierbei durch einen Platzhalter repräsentiert.

Die Tests mit dem Programm `CreatePSP` zeigen, dass der in dieser Arbeit vorgestellte Ansatz tatsächlich praktisch anwendbar ist. Um die Effizienz dieses Ansatzes zu untersuchen, wäre es nun allerdings notwendig vergleichende Tests mit anderen Programmen durchzuführen. Auch wenn dazu im Rahmen dieser Arbeit keine Zeit bleibt, ist dies doch ein wichtiger Ansatzpunkt für eine weitergehende Beschäftigung mit dem vorliegenden Thema.

Beispielsweise liefert die Anwendung von `SolvePSP` auf das in Abschnitt 4.2.4 vorgestellte partielle Erfüllbarkeitsproblem folgende Ausgabe:

```
>perl SolvePSP.pl
get help with -h ; started with default options:
-d DepotsNum.pddl
-f pfile1
-p psp1
-b foundplan.out
-c 1
Start Metric: 1 Reached Metric: 5
Start Metric: 6 Reached Metric: 28
Start Metric: 29 Reached Metric: 40
Start Metric: 41 Reached Metric: 41
Start Metric: 42 Reached Metric: 42
Start Metric: 43 Reached Metric: 28 /** no plan **/
```

Einer (optimalen) Lösung des partiellen Erfüllbarkeitsproblems entspricht demnach ein Plan, der zu einem Grenzwert $q_C = 42$ führt. Der folgende Plan wurde gefunden:

```

Schritt 0: LIFT HOIST1 CRATEO PALLET1 DISTRIBUTORO
          1: DRIVE TRUCK0 DISTRIBUTOR1 DISTRIBUTORO
          2: LIFT HOISTO CRATE1 PALLETO DEPOTO
          3: LOAD HOISTO CRATE1 TRUCK1 DEPOTO
          4: DRIVE TRUCK1 DEPOTO DISTRIBUTORO
          5: LOAD HOIST1 CRATEO TRUCK1 DISTRIBUTORO
          6: UNLOAD HOIST1 CRATE1 TRUCK1 DISTRIBUTORO
          7: DRIVE TRUCK1 DISTRIBUTORO DISTRIBUTOR1
          8: DROP HOIST1 CRATE1 PALLET1 DISTRIBUTORO
          9: UNLOAD HOIST2 CRATEO TRUCK1 DISTRIBUTOR1
         10: DRIVE TRUCK1 DISTRIBUTOR1 DEPOTO
         11: DROP HOIST2 CRATEO PALLET2 DISTRIBUTOR1

```

Damit wurden folgende Einträge der PSP-Beschreibung, vgl. Abschnitt 4.2.4, erfüllt bzw. nicht erfüllt:

- Literal (AT TRUCK1 DEPOTO) ist erfüllt als obligatorisches Ziel.
- Literal (AT CRATEO DISTRIBUTORO) ist nicht erfüllt.
- Literal (AT CRATEO DISTRIBUTOR1) ist erfüllt.
- Literal (ON CRATE1 PALLET1) ist erfüllt.
- Literal (AVAILABLE HOISTO) ist erfüllt.
- Literal (NOT (AT TRUCK1 DEPOTO)) ist nicht erfüllt.
- Literal (NOT (AT TRUCK0 DISTRIBUTOR1)) ist erfüllt.
- Literal (NOT (AT CRATEO DISTRIBUTORO)) ist erfüllt.

Es ist nicht schwer zu sehen, dass in dieser Lösung tatsächlich eine optimale Menge von Zielformeln erfüllt ist. So sind lediglich zwei Literale nicht erfüllt. Offensichtlich steht eines davon im Widerspruch mit dem gegebenen obligatorischen Ziel und das andere im Widerspruch mit einem Zielliteral der PSP-Beschreibung, das eine höhere Präferenz besitzt.

Das vorgestellte Programmpaket, bestehend aus `SolvePSP` und `CreatePSP`, ermöglicht es, Aspekte von partiellen Erfüllbarkeitsproblemen quantitativ zu untersuchen. Als erstes wäre vielleicht die Frage zu klären, in welchen Domänen der im Programm implementierte Ansatz effizient und schnell partielle Planungsprobleme löst. Eine weitere Problemstellung läge darin, zu untersuchen, wie die verwendete Schranke q_C während der Laufzeit des Programmes anwächst, wobei z.B. über verschiedene PSP-Beschreibungen gemittelt wird. Diese PSP-Beschreibungen könnten für eine gegebene Domänen- und Faktenbeschreibung automatisch erzeugt werden. Es ist zu vermuten, dass dabei prinzipiell ein Verlauf wie in Abbildung 4.1 zu sehen ist. Unklar ist allerdings, inwieweit der genaue Verlauf von Domänen- und Faktenbeschreibung, sowie von der getroffenen Auswahl der PSP-Beschreibungen, abhängt.

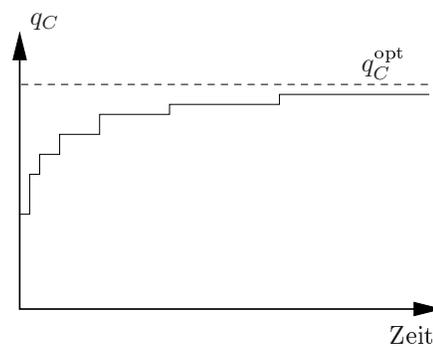


Abbildung 4.1: Schematischer Anstieg der Güte (Schranke q_C) mit der Zeit während der Suche nach einer Lösung des partiellen Erfüllbarkeitsproblems. Der maximal erreichbare Wert ist mit q_C^{opt} bezeichnet.

Zusammenfassung

Die vorliegende Arbeit beschäftigte sich mit zwei Problembereichen. Zum einen wurde untersucht, inwiefern und auf welche geeignete Weise Präferenzen über Zielen ausgedrückt werden können. Die zweite Aufgabe dieser Arbeit lag im Finden eines Ansatzes zur Lösung partieller Erfüllbarkeitsprobleme. Im ersten Kapitel erfolgte zunächst eine Einführung grundlegender Begriffe aus dem Bereich des Automatischen Planens. Dies beinhaltete eine Erläuterung bzw. Definition partieller Erfüllbarkeitsprobleme und der gängigen Repräsentationssprachen für Planungsproblemen. Weiterhin wurden aktuelle, der Literatur entnommene, Lösungsansätze für partielle Erfüllbarkeitsprobleme vorgestellt und wesentliche Unterscheidungsmerkmale zu dem Ansatz dieser Arbeit herausgearbeitet.

Bei der (qualitativen) Beschreibung von Präferenzen stellte sich die Aufgabe, dies auf eine möglichst effiziente und dennoch hinreichend allgemeine Weise durchzuführen. Ein Kandidat für eine derartige Beschreibung stellte zunächst die in [33] eingeführte Sprache LPD dar. In Anlehnung an diese Vorgabe wurden nun zwei alternative Beschreibungssprachen LPD* und TLPD konstruiert und mit der Sprache LPD verglichen. Es zeigte sich, dass insbesondere die Sprache TLPD in der Lage ist, Präferenzen, welche natürliche Eigenschaften aufweisen, auf einfache Weise zu beschreiben.

Die Effektivität des vorgestellten Ansatzes zur Lösung partieller Erfüllbarkeitsprobleme beruht wesentlich auf der Linearisierung der Präferenzrelation. Aus diesem Grund wurde in Kapitel 3 die Sprache TLPD linearisiert und eine numerische Beschreibungssprache NLPD definiert. Die Verwendung der Sprache NLPD bietet zudem den Vorteil, quantitative Präferenzbeschreibungen zu ermöglichen.

Eine Anwendung der in den Kapiteln 2 und 3 ausgearbeiteten Überlegungen auf den Planungskontext erfolgte in Kapitel 4. So wurde eine auf der Sprache NLPD basierende Erweiterung PDDL2.1* der Repräsentationssprache PDDL2.1 vorgeschlagen, mit deren Hilfe sich partielle Erfüllbarkeitsprobleme ausdrücken lassen. Auch wenn diese Erweiterung nur einen geringen Umfang besitzt, ermöglicht sie doch eine intuitive und gleichzeitig hinreichend allgemeine Beschreibung von Präferenzrelationen.

Anschließend erfolgte eine Beschreibung eines allgemeinen Lösungsansatzes, welcher auf der Lösung klassischer Planungsprobleme aufbaut. Es wurde speziell ein Lösungsalgorithmus vorgestellt, der aufzeigt, wie ein in PDDL2.1* gegebenes partielles Erfüllbarkeitsproblem durch klassische, der Sprache PDDL2.1 mächtige, Planer gelöst werden kann. Die hierbei erforderliche Übersetzung eines PDDL2.1* Konstruktes in PDDL2.1 wurde explizit dargelegt und anschließend an einem Beispiel durchgeführt. Ein weiterer wichtiger Teil dieser Arbeit stellte die Automation, d.h. die Implementation als Computerprogramm, des Lösungsansatzes dar. Die Hauptschleife des Lösungsalgorithmus' wurde hierbei durch ein Perl-Skript rea-

liert. Die Übersetzung eines PDDL2.1* Problems (genauer eines in einer Untersprache von PDDL2.1* definierten Problems) in eine PDDL2.1 Beschreibung erfolgte durch ein C-Programm, welches eine PDDL2.1* Spezifikation einliest, parst und schließlich, nach Anwendung obiger Übersetzungsschritte, eine PDDL2.1 Problembeschreibung ausgibt.

Ausblick

Eine Fortsetzung und Weiterentwicklung dieser Arbeit ist an mehreren Stellen möglich. Folgende Punkte stellen sicherlich keine umfassende Liste dar, spiegeln jedoch die möglicherweise naheliegenden nächsten Schritte wider.

- Es wäre interessant, das entwickelte Programmpaket auf verschiedenen Domänen einzusetzen und mit alternativen Lösungsansätzen für partielle Erfüllbarkeitsprobleme zu vergleichen. Dazu müssten Standards für die Beschreibung von partiellen Erfüllbarkeitsproblemen und Bewertungen ihrer Lösungen geschaffen und eingehalten werden. Zur Beschreibung von partiellen Erfüllbarkeitsproblemen wird in dieser Arbeit die Sprache PDDL2.1* vorgeschlagen.
- Angewendet auf verschiedene Domänen können statistische Untersuchungen von partiellen Erfüllbarkeitsproblemen erfolgen. Eine mögliche Aufgabestellung läge beispielsweise darin, empirisch nachzuweisen, wie groß der Anteil der erfüllten Zielformeln in der Lösung eines partiellen Erfüllbarkeitsproblems im Mittel ist. Hierbei kann z.B. über verschiedene Anfangszustände, verschiedene Instanzen von partiellen Erfüllbarkeitsproblemen usw. gemittelt werden.
- Eine Weiterentwicklung bzw. Adaption des Ansatzes auf temporale Domänen wäre sicher von großem Wert, da sich dann Probleme des *Planning and Scheduling* mit dem vorgestellten Ansatz lösen lassen. Die Implementierung ist hierbei entsprechend anzupassen.

Anhang A

BNF Spezifikation von PDDL2.1

In diesem Teil des Anhangs wird die Syntax von PDDL2.1 level 2 in Backus-Naur-Form reproduziert. Die syntaktischen Elemente höherer Level werden nicht dargestellt. Die Beschreibung folgt dabei [3]. Für weiterführende Diskussionen sei ebenfalls auf diese Quelle verwiesen.

Die Syntaxbeschreibung wird aus Gründen der Übersichtlichkeit in mehrere Unterabschnitte aufgeteilt. Eine Regel, in der ein oder mehrere hochgestellte Ausdrücke vorkommen, kann nur dann angewendet werden, wenn mindestens einer der hochgestellten Ausdrücke in einer <require-def> Angabe spezifiziert wird. Die Semantik dieser Angaben besteht darin, dem Planer mitzuteilen, welche PDDL2.1 Sprachkonstrukte in der Problembeschreibung verwendet werden.

A.1 Domänenbeschreibung

```
<domain> ::= (define (domain <name>)
              [<require-def>]
              [<types-def>]
              [<constants-def>]
              [<predicates-def>]
              [<function-def>]:fluents
              <structure-def>*)

<require-def> ::= (:requirements <require-key>+)
<require-key> ::= siehe unten
<types-def> ::= (:types <typed list(name)>)
<constants-def> ::= (:constants <typed list(name)>)
<predicates-def> ::= (:predicates <atomic formula skeleton>+)
<atomic formula skeleton> ::= (<predicate> <typed list (variable)>)
<predicate> ::= <name>
```

```

<name>                ::= eine (fast) beliebige Zeichenkette
<variable>           ::= ?<name>
                        <psp-def>
<functions-def>      ::= :fluents (:functions <function typed list
                        (atomic function skeleton)>)
<atomic function skeleton> ::= (<function-symbol> <typed list (variable)>)
<function-symbol>    ::= <name>
<structure-def>      ::= <action-def>

```

Das Nichtterminalsymbol `<require-key>` kann durch die folgenden Zeichenketten ersetzt werden (die Semantik ist jeweils mit angegeben). Falls kein `:requirements` Feld vorhanden ist wird (`:requirements :strips`) angenommen.

```

<require-key>        Beschreibung
:strips              STRIPS-artige Aktionen, Atomformeln in Zielbeschreibungen erlaubt
:typing              Variablen können typisiert sein
:negative-preconditions Literale in Zielbeschreibungen erlaubt
:disjunctive-preconditions or und andere Konstrukte in Zielbeschreibungen erlaubt
:equality            Es existiert eine interne Unterstützung des = Prädikates
:existential-preconditions Existenz-Quantifizierung in Zielbeschreibungen erlaubt
:universal-preconditions Generalisierung in Zielbeschreibung erlaubt
:quantified-preconditions ::= :existential-preconditions+ :universal-preconditions
:conditional-effects bedingte Effekte werden erlaubt
:adl                 ::= :strips+ :typing+ :negative-preconditions
                    + :disjunctive-preconditions+ :equality
                    + :quantified-preconditions+ :conditional-effects
:fluents             numerische Variablen, Ausdrücke und Vergleiche werden erlaubt

```

A.2 Listen

```

<typed list (x)>      ::= x*
<typed list (x)>      ::= :typing x+ - <type> <typed list (x)>
<primitive-type>    ::= <name>
<type>               ::= (either <primitive-type>+)
<type>               ::= <primitive-type>
<function typed list (x)> ::= x*
<function typed list (x)> ::= :typing x+ - <function type> <function typed list (x)>
<function type>     ::= <number>

```

A.3 Aktionen

<code><action-def></code>	<code>::= (:action <action-symbol> :parameters (<typed list (variable)>> <action-def body>)</code>
<code><action-symbol></code>	<code>::= <name></code>
<code><action-def body></code>	<code>::= [:precondition <GD>] [:effect <effect>]</code>
<code><GD></code>	<code>::= ()</code>
<code><GD></code>	<code>::= <atomic formula(name)></code>
<code><GD></code>	<code>::=:negative-preconditions <literal(term)></code>
<code><GD></code>	<code>::= (and <GD>*)</code>
<code><GD></code>	<code>::=:disjunctive-preconditions (or <GD>*)</code>
<code><GD></code>	<code>::=:disjunctive-preconditions (not <GD>*)</code>
<code><GD></code>	<code>::=:disjunctive-preconditions (imply <GD>*)</code>
<code><GD></code>	<code>::=:existential-preconditions (exists (<typed list(variable)>*) <GD>)</code>
<code><GD></code>	<code>::=:universal-preconditions (forall (<typed list(variable)>*) <GD>)</code>
<code><GD></code>	<code>::=:fluents (<binary-comp> <f-exp> <f-exp>)</code>
<code><literal(t)></code>	<code>::= <atomic formula(t)></code>
<code><literal(t)></code>	<code>::= (not <atomic formula(t)>)</code>
<code><atomic formula(t)></code>	<code>::= (<predicate> t*)</code>
<code><term></code>	<code>::= <name></code>
<code><term></code>	<code>::= <variable></code>
<code><f-exp></code>	<code>::= <number></code>
<code><f-exp></code>	<code>::= (<binary-op> <f-exp> <f-exp>)</code>
<code><f-exp></code>	<code>::= (- <f-exp>)</code>
<code><f-exp></code>	<code>::= <f-head></code>
<code><f-head></code>	<code>::= (<function-symbol> <term>*)</code>
<code><f-head></code>	<code>::= <function-symbol></code>
<code><binary-op></code>	<code>::= +</code>
<code><binary-op></code>	<code>::= -</code>
<code><binary-op></code>	<code>::= *</code>
<code><binary-op></code>	<code>::= /</code>
<code><binary-comp></code>	<code>::= ></code>
<code><binary-comp></code>	<code>::= <</code>
<code><binary-comp></code>	<code>::= =</code>
<code><binary-comp></code>	<code>::= >=</code>
<code><binary-comp></code>	<code>::= <=</code>
<code><number></code>	<code>::= <i>eine beliebige numerische Zeichenkette, z.B. als Fließkommazahl</i></code>
<code><effect></code>	<code>::= ()</code>

```

<effect>          ::= (and <c-effect>*)
<effect>          ::= <c-effect>
<c-effect>        ::=:conditional-effects (forall (<variable>*) <effect>)
<c-effect>        ::=:conditional-effects (when <GD> <cond-effect>)
<c-effect>        ::= <p-effect>
<cond-effect>    ::= (and <p-effect>*)
<cond-effect>    ::= <p-effect>
<p-effect>       ::= (not <atomic formula(term)>)
<p-effect>       ::= <atomic formula(term)>
<p-effect>       ::=:fluents (<assign-op> <f-head> <f-exp>)
<assign-op>      ::= assign
<assign-op>      ::= increase
<assign-op>      ::= decrease
<assign-op>      ::= scale-up
<assign-op>      ::= scale-down

```

A.4 Faktenbeschreibung

```

<problem>          ::= (define (problem <name>)
                        (:domain <name>)
                        [<require-def>]
                        [<object declaration>]
                        <init>
                        <goal>
                        [<metric-spec>])
<object declaration> ::= (:objects <typed list (name)>)
<init>             ::= (:init <init-el>*)
<init-el>          ::= <literal(name)>
<init-el>          ::=:fluents (= <f-head> <number>)
<goal>             ::= (:goal <GD>)
<metric-spec>     ::= (:metric <optimization> <ground-f-exp>)
<optimization>    ::= minimize
<optimization>    ::= maximize
<ground-f-exp>    ::= (<binary-op> <ground-f-exp> <ground-f-exp>)
<ground-f-exp>    ::= (- <ground-f-exp>)
<ground-f-exp>    ::= <number>
<ground-f-exp>    ::= (<function symbol> <name>*)
<ground-f-exp>    ::= total-time
<ground-f-exp>    ::= <function symbol>

```

A.5 PSP Beschreibung

An dieser Stelle werden die Konstrukte aus Abschnitt 4.2.1 zur Sprache PDDL2.1 hinzugefügt. In der vorliegenden Arbeit wird die um diese Konstrukte erweiterte Sprache mit PDDL2.1* bezeichnet. Falls der Ausdruck `:psp-full` nicht angegeben ist, sind nur PSP Beschreibungen erlaubt, die sich in lineare numerische Ausdrücke übersetzen lassen. Diese Teilung der Ausdrucksmächtigkeit wird im Hinblick auf den Einsatz derjenigen automatischen Planer eingeführt, welche numerische Ausdrücke nur dann handhaben können, wenn sie linear sind.

```

<require-key> ::= :psp
<require-key> ::= :psp-full
<psp> ::= :psp, :psp-full (define (pspname <name>)
                           (:problem <name>)
                           [(:domain <name>)]
                           <goal>)
                           <psp-def>)
<psp-def> ::= (:psp <psp-node>)
<psp-node> ::= (LEV <psp-node>+)
<psp-node> ::= (CAR <psp-node>+)
<psp-node> ::= (MULT <psp-node> <non-negative integer>)
<psp-node> ::= <literal(name)>
<psp-node> ::= (<binary-comp> <f-exp> <f-exp>)
<psp-node> ::= <non-negative integer>
<non-negative integer> ::= eine natürliche Zahl
<psp-node> ::= :psp-full (MULT <psp-node>+)
<psp-node> ::= :psp-full <GD>

```


Anhang B

Auszüge aus dem Quellcode

B.1 Modifikation des Planers Metric-FF

Folgende Modifikation wurde am Planer Metric-FF vorgenommen, damit der erreichte Grenzwert ausgegeben wird und verwendet werden kann. Zunächst wurden im Hauptprogramm `main.c` des Metric-FF Programmpakets folgende Zeilen verändert (die modifizierten Zeilen sind durch » RF bzw. « RF eingeklammert):

Im Header:

```
#include "relax.h"
#include "search.h"

/* >> RF */
#include "statistics.h"
/* << RF */
```

Im der Routine `main()`:

```
if ( found_plan ) {
    print_plan();
}

!found_plan = found_plan;

/* >> RF */
print_statistics( );
/* << RF */
```

Die eingebundene Bibliothek besitzt folgende Headerdatei `statistics.h` und Quellcodedatei `statistics.c`:

`statistics.h`:

```

/*****
 * prints some information about the quality
 * of the found plan
 * uses datastructures of the Metric-FF planner
 * author: Robert Feldmann 2005
 *****/

#ifndef __STATISTICS_H
#define __STATISTICS_H

#include <string.h>
#include "ff.h"

void print_statistics ( );

#endif

```

`statistics.c`:

```

/*****
 * prints some information about the quality
 * of the found plan
 * uses datastructures of the Metric-FF planner
 * author: Robert Feldmann 2005
 *****/

#include "statistics.h"
#include "string.h"

/* it prints:
 *  searching time
 *  total time
 *  maximal metric value
 *  metric value of solution
 *  number of formula in rkb
 *  number of fulfilled formula
 */
void print_statistics ( )

{
    float curr_metric,max_metric;
    int fulfilled_formula,relevant_formula;
    int i;
    int c1,c2;
    LnfExpNode *n;
    FILE* file;

    if (gnum_numeric_goal>0 && gnum_lnf_goal>0) { /* print reached value of last numeric goal */
        if (gnumeric_goal_lh[gnum_numeric_goal-1]->connective==NUMBER) {

```

```

/* calculate the offset, which arises since Metric-FF normalizes its expressions */
c1=gnumeric_goal_lh[gnum_numeric_goal-1]->value;
c2=glnf_goal_rh[gnum_lnf_goal-1];

/* now calculate value of last numeric goal */
n=glnf_goal_lh[gnum_lnf_goal-1];

if (n->num_pF!=1 || n->num_nF!=0 || grelevant_fluents_lnf[n->pF[0]] == NULL ) {
fprintf(stderr,"Could not retrieve the final fluent values, please debug");
return;
}
curr_metric=c1-c2+n->pC[0]*gfl_conn[n->pF[0]].level[0]+n->c;
n=grelevant_fluents_lnf[n->pF[0]];
fulfilled_formula=0;
max_metric=0;
for ( i = 0; i < n->num_pF; i++ ) {
if (strstr(grelevant_fluents_name[n->pF[i]],"MINUS-")==NULL) {
if ((int)(gfl_conn[n->pF[i]].level[0]+0.5)>0)
fulfilled_formula++;
if (n->pC[i]>=0)
max_metric+=n->pC[i];
} else {
if ((int)(gfl_conn[n->pF[i]].level[0]+0.5)<=0)
fulfilled_formula++;
/* if (n->pC[i]<=0)
max_metric-=n->pC[i];*/
}
}

relevant_formula=n->num_pF+n->num_nF;
max_metric+=c1-c2+n->c;
/* now print */
file=fopen("PSP_statistic.out","w");
fprintf(file,"PSP_STATISTICS ");
fprintf(file,"Total_Time %f ",gtempl_time + greach_time + grelev_time +
gLNf_time + gconn_time + gsearch_time);
fprintf(file,"Search_Time %f ",gsearch_time);
fprintf(file,"Maximal_Metric %d ",(int)(max_metric+0.5));
fprintf(file,"Solution_Metric %d ",(int)(curr_metric+0.5));
fprintf(file,"Relevant_Formula %d ",relevant_formula);
fprintf(file,"FulFilled_Formula %d ",fulfilled_formula);

fprintf(file,"PSP_VALUES ");
for ( i = 0; i < n->num_pF; i++ )
fprintf(file,"%s %f ",grelevant_fluents_name[n->pF[i]],gfl_conn[n->pF[i]].level[0]);
for ( i = 0; i < n->num_nF; i++ )
fprintf(file,"%s %f ",grelevant_fluents_name[n->nF[i]],gfl_conn[n->nF[i]].level[0]);
fprintf(file,"\n");
}
}
}

```

B.2 Das Perl-Skript

Der Quellcode des Perl-Skript `SolvePSP.pl` lautet:

```
#!/usr/bin/perl -w

#####
# SolvePSP.pl : Solving partial satisfaction problems  #
#               using a translation into PDDL2.1      #
# written by Robert Feldmann (May 2005)             #
#####

use strict;
use Getopt::Std;

my %option=();
my $pathParser='/home/feldmann/Diplom/InfoArbeit/example/parser';
my $execParser='createPSP';
my $pathPlanner='/home/feldmann/Diplom/InfoArbeit/example/planner';
my $execPlanner='ff'; # use Metric-ff
my $pathProblem='/home/feldmann/Diplom/InfoArbeit/example/problem';
my $domainfile='DepotsNum.pddl';
my $factfile='pfile1';
my $pspfile='psp1';
my $foundplanfile='foundplan.out';
my $metric=1;
my $max_metric=1;
my $output_line;
my $foundplan;
my @output;

getopts('hc:d:f:p:', \%option) or exit;

if (defined $option{h}) {
    print "usage:\n";
    print "-d <str>  domain file name\n";
    print "-f <str>  fact file name\n";
    print "-p <str>  PSP description file name\n";
    print "-b <str>  name of the output file for plans and time consumption\n";
    print "-c <int>  starting metric value for partial satisfaction problem\n";
}

if (defined $option{d}) {
    $domainfile=$option{d};
}

if (defined $option{f}) {
    $factfile=$option{f};
}

if (defined $option{p}) {
    $pspfile = $option{p};
}

if (defined $option{b}) {
    $foundplanfile=$option{b};
}
```

```

}
if (defined $option{c}) {
    $metric=$option{c};
}

if (! %option) {
    print "get help with -h ; started with default options:\n";
    print "-d $domainfile\n";
    print "-f $factfile\n";
    print "-p $pspfile\n";
    print "-b $foundplanfile\n";
    print "-c $metric\n";
}

if (! open (RESULTFILE,">SolvePSP.out")) {
    printf "could not open SolvePSP.out for output\n";
    exit;
}

if (! open (FOUNDPLANFILE,">$foundplanfile")) {
    printf "could not open $foundplanfile for output\n";
    exit;
}

# Translate PSP description (factfile, domainfile, pspfile) into
# PDDL2.1 (new factfile, new domainfile) use current metric

`$pathParser/$execParser -p $pathProblem/ -o $domainfile -f $factfile -r $pspfile`;

# Get upper limit metric value of problem (for statistics)

open(PFILE2,"<$pathProblem/PSP_$factfile") || die "$pathProblem/PSP_$factfile not found.";
while (<PFILE2>) {
    if ( /\(=\s\(\PSP_MAX_METRIC\)\s([0-9]*)\)/ ) {
        $max_metric=$1;
    }
}
close(PFILE2);

# Set start metric in PDDL2.1 description

open(PFILE,">$pathProblem/PSPM_$factfile") || die "$pathProblem/PSPM_$factfile not found.";
open(PFILE2,"<$pathProblem/PSP_$factfile") || die "$pathProblem/PSP_$factfile not found.";
while (<PFILE2>) {
    s/\(=\s\(\PSP_START_METRIC\)\s[0-9]*\)/\(\s\(\PSP_START_METRIC\)\s$metric\)/;
    print PFILE $_;
}
close(PFILE2);
close(PFILE);

# main loop: finish if no plan found or by explicit user break

```

```
while (1) {

    # Run planner on the translated problem

    print "\nStart Metric: $metric ";
    $foundplan='$pathPlanner/$execPlanner -p $pathProblem/ -o \PSP_$domainfile\ -f \PSPM_$factfile\ -i 0';
    print FOUNDPLANFILE $foundplan;

    # Get reached metric value from the planners output

    open(STATFILE,"<PSP_statistic.out") || die "PSP_statistic.out not found.";
    while (<STATFILE>){
        $output_line=$_;
    }
    close(STATFILE);

    @output=split(/ /,$output_line);
    print RESULTFILE "Start_Metric $metric $output_line";
    print "Reached Metric: $output[8]";

    # if reached metric is smaller than current start metric -> no plan found

    if ($output[8] < $metric) {
        last;
    }

    # Increase metric by one

    $metric=$output[8]+1;

    # Set new start metric

    open(PFILE,">$pathProblem/PSPM_$factfile") || die "$pathProblem/PSPM_$factfile not found.";
    open(PFILE2,"<$pathProblem/PSP_$factfile")|| die "$pathProblem/PSP_$factfile not found.";
    while (<PFILE2>) {
        s/\(= \PSP_START_METRIC\) [0-9]*\)/\(\(= \PSP_START_METRIC\) $metric\)/;
        print PFILE $_;
    }
    close(PFILE2);
    close(PFILE);

};

print " /** no plan **/\n";
print RESULTFILE "*** finished ***\n";
```

Literaturverzeichnis

- [1] NAU, Dana ; GHALLAB, Malik. *Measuring the Performance of Automated Planning Systems*. Juli 12 2004. URL: <http://www.cs.umd.edu/~nau/papers/nau04measuring.pdf>
- [2] FIKES, Richard E. ; NILSSON, Nils J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In: *Artificial Intelligence* 2 (1971), Nr. 3–4, S. 189–208
- [3] FOX, M. ; LONG, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. In: *Journal of Artificial Intelligence Research* 20 (2003), S. 61–124. – ISSN 1076–9757. URL: <http://www.cs.washington.edu/research/jair/volume20/fox03a.ps>
- [4] BREWKA, Gerhard: Complex Preferences for Answer Set Optimization. In: DUBOIS, Didier (Hrsg.) ; WELTY, Christopher A. (Hrsg.) ; WILLIAMS, Mary-Anne (Hrsg.): *KR*, AAAI Press, 2004. – ISBN 1–57735–199–1, S. 213–223. URL: www.informatik.uni-leipzig.de/~brewka/papers/KR04BrewkaG.pdf
- [5] DELGRANDE, James P. ; SCHAUB, Torsten: Expressing preferences in default logic. In: *Artificial Intelligence* 123 (2000), Nr. 1-2, S. 41–87. URL: citeseer.ist.psu.edu/delgrande00expressing.html
- [6] BREWKA, Gerhard ; EITER, Thomas: Prioritizing Default Logic. In: HÖLDOBLER, Steffen (Hrsg.): *Intellectics and Computational Logic* Bd. 19, Kluwer, 2000. – ISBN 0–7923–6261–6, S. 27–45
- [7] RUSSEL, S.J. ; NORVIG, P.: *Artificial Intelligence, A Modern Approach*. Englewood Cliffs, NJ : Prentice Hall, 1995
- [8] GHALLAB, M. ; NAU, D. ; TRAVERSO, P.: *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004. – ISBN 1–55860–856–7
- [9] ESTLIN, T. ; CASTAÑO, R. ; ANDERSON, B. ; FISHER, F. ; JUDD, M.: Learning and Planning for Mars Rover Science. In: *JCA 2003 workshop notes on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating* Bd. 10. Acapulco, Mexico, 2003
- [10] ONSSON, Ari J. ; SMITH, David E. ; FRANK, Jeremy ; MORRIS, Robert. *Planning and Scheduling for Fleets of Earth Observing Satellites*. April 18 2001. URL: <http://ic-www.arc.nasa.gov/ic/people/jonsson/Papers/isairas01.ps>

- [11] JEFFREY, S. ; MARK, L. ; JOHNSTON, D. ; MILLER, G. ; KRUEGER, A. ; LUCKS, M. ; GIULIANO, M. *An AI Scheduling Environment for the Hubble Space Telescope*. 1991. URL: citeseer.ist.psu.edu/jeffrey91ai.html
- [12] MINTON, Steven ; JOHNSTON, Mark D. ; PHILIPS, Andrew B. ; LAIRD, Philip: Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. In: *Artificial Intelligence* 58 (1992), Nr. 1-3, S. 161–205. URL: citeseer.ist.psu.edu/article/minton92minimizing.html
- [13] POTTER, W. ; GASCH, J.: A photo album of earth: Scheduling landsat 7 mission daily activities. In: *Proc. International Symposium on Space Mission Operations and Ground Data Systems*, 1998
- [14] BRIEL, Menkes Van D. ; SANCHEZ, Romeo ; KAMBHAMPATI, Subbarao. *Over-Subscription in Planning: a Partial Satisfaction Problem*. März 27 2004. URL: <http://rakaposhi.eas.asu.edu/WIPS2004-PSP.pdf>
- [15] NEWELL, A. ; SIMON, H. A.: GPS, a program that simulates human thought. In: FEIGENBAUM, E. A. (Hrsg.) ; FELDMAN, J. (Hrsg.): *Computers and thought*. New York : McGraw-Hill, 1963, S. 279–293
- [16] LIFSCHITZ, Vladimir. *On The Semantics Of Strips*. November 13 1987. URL: <http://www.cs.utexas.edu/users/vl/mypapers/strip.ps>
- [17] FIKES, Richard E. ; NILSSON, Nils J.: STRIPS, a Retrospective. In: *Artificial Intelligence* 59 (1993), Nr. 1–2, S. 227–232
- [18] BYLANDER, Tom. *The Computational Complexity of Propositional STRIPS Planning*. August 07 1994. URL: <http://www.cs.utsa.edu/~bylander/pubs/artificial-intelligence94.ps.gz>
- [19] PEDNAULT, E. P. D.: ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In: *Proceedings of KR'89*. Toronto, Canada, pp 324-331, Mai 1989. – (extended version submitted to Artificial Intelligence, special issue on KR'89).
- [20] BARRETT, Anthony ; KWOK, Chung ; AERONAUTIQUES, Constructions ; ISI, Craig K. ; WELD, Daniel ; CHRISTIANSON, Dave ; SMITH, David E. ; SRI, David W. ; NATIONALE, Ecole ; GOLDEN, Keith ; GHALLAB, Malik ; FRIEDMAN, Marc ; PENBERTHY, Scott ; SUN, Ying. *PDDL - The Planning Domain Definition Language*. Juli 31 1998. URL: <http://www.informatik.uni-freiburg.de/~koehler/aips/PDDL-MANUAL.ps.gz>
- [21] The 2002 International Planning Competition [online]. URL: <http://planning.cis.strath.ac.uk/competition/>
- [22] NEBEL, B.: On the Expressive Power of Planning Formalisms: Conditional Effects and Boolean Preconditions in the STRIPS Formalism. In: MINKER, J. (Hrsg.): *Logic-Based Artificial Intelligence*. 2000, S. 469–490
- [23] GAZEN, B. C. ; KNOBLOCK, C. A.: Combining the expressivity of UCPOP with the efficiency of Graphplan. In: STEEL, Sam (Hrsg.) ; ALAMI, Rachid (Hrsg.): *Proceedings of the 4th European*

- Conference on Planning (ECP-97): Recent Advances in AI Planning* Bd. 1348. Berlin : Springer, September 24–26 1997. – ISBN 3–540–63912–8, S. 221–233
- [24] HOFFMANN, Jörg: Extending FF to Numerical State Variables. In: *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*. Lyon, France, Juli 2002, S. 571–575
- [25] HOFFMANN, Jörg ; NEBEL, Bernhard: The FF Planning System: Fast Plan Generation Through Heuristic Search. 14 (2001), S. 253–302. URL: <http://www.cs.washington.edu/research/jair/volume14/hoffmann01a.ps>
- [26] BLUM, Avrim ; FURST, Merrick: Fast Planning Through Planning Graph Analysis. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1995, S. 1636–1642
- [27] HOFFMANN, Jörg: The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables. In: *Journal of Artificial Intelligence Research* 20 (2003), S. 291–341. URL: <http://www.cs.washington.edu/research/jair/volume20/hoffmann03a.ps>
- [28] VOSSEN, Thomas ; BALL, Michael ; LOTEM, Amnon ; NAU, Dana: On the Use of Integer Programming Models in AI Planning. In: THOMAS, Dean (Hrsg.): *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1)*. S.F. : Morgan Kaufmann Publishers, Juli 31–August 6 1999, S. 304–309. URL: <http://www.cs.umd.edu/~nau/papers/ip-ijcai99.pdf>
- [29] KAUTZ, Henry ; SELMAN, Bart: Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*. Menlo Park : AAAI Press / MIT Press, August 4–8 1996. – ISBN 0–262–51091–X, S. 1194–1201. URL: <http://www.cc.gatech.edu/~jimmyd/summaries/kautz1996.ps>
- [30] VAN DEN BRIEL, Menkes ; NIGENDA, Romeo S. ; DO, Minh B. ; KAMBHAMPATI, Subbarao: Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. In: MCGUINNESS, Deborah L. (Hrsg.) ; FERGUSON, George (Hrsg.): *AAAI, AAAI Press / The MIT Press*, 2004. – ISBN 0–262–51183–5, S. 562–569
- [31] BONET, Blai ; LOERINCS, Gábor ; GEFFNER, Héctor: A Robust and Fast Action Selection Mechanism for Planning. In: *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*. Menlo Park : AAAI Press, Juli 27–31 1997. – ISBN 0–262–51095–2, S. 714–719
- [32] DO, M. ; KAMBHAMPATI, S.: SAPA: A Multi-objective Metric Temporal Planner. In: *Journal of Artificial Intelligence Research* 20 (2003), S. 155–194. – ISSN 1076–9757. URL: <http://www.cs.washington.edu/research/jair/volume20/do03a.ps>
- [33] BREWKA, Gerhard: A Rank Based Description Language for Qualitative Preferences. In: DE MÁNTARAS, Ramon L. (Hrsg.) ; SAITTA, Lorenza (Hrsg.): *ECAI*, IOS Press, 2004. – ISBN 1–58603–452–9, S. 303–307. URL: www.pims.math.ca/science/2004/NMR/papers/paper12.pdf

- [34] LANG, Jérôme: From preference representation to combinatorial vote. In: *Proc. of the 8th International Conference, Principles of Knowledge Representation and Reasoning*, Toulouse, France. San Francisco, California : Morgan Kaufmann Publishers, April 22–25 2002. – BB, S. 277–288
- [35] BENFERHAT, Salem ; DUBOIS, Didier ; KACI, Souhila ; PRADE, Henri: Bipolar representation and fusion of preferences in the possibilistic logic framework. In: *Proc. of the 8th International Conference, Principles of Knowledge Representation and Reasoning*, Toulouse, France. San Francisco, California : Morgan Kaufmann Publishers, April 22–25 2002. – BB, S. 421–432
- [36] GOLDSZMIDT, Moisés ; PEARL, Judea: System Z⁺: A Formalism for Reasoning With Variable-Strength Defaults. In: DEAN, Thomas (Hrsg.) ; MCKEOWN, Kathleen (Hrsg.): *Proceedings of the Ninth National Conference on Artificial Intelligence*. Menlo Park, California : AAAI Press, 1991, S. 399–404
- [37] PEARL, Judea: System Z: a Natural Ordering of Defaults with Tractable Applications to Nonmonotonic Reasoning. In: *Proceedings of the Third Conference on Theoretical Aspects of Reasoning About Knowledge*. Monterey, California : Morgan Kaufmann, März 1990, S. 121–135
- [38] BREWKA, Gerhard: Preferred Subtheories: An Extended Logical Framework for Default Reasoning. In: SRIDHARAN, N. S. (Hrsg.): *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Detroit, MI, USA : Morgan Kaufmann, August 1989. – ISBN 1–55860–094–9, S. 1043–1048
- [39] BENFERHAT, S. ; CAYROL, C. ; DUBOIS, D. ; LANG, J. ; PRADE, H.: Inconsistency Management and Prioritized Syntax-Based Entailment. In: *Proceedings of the Thirteenth International Joint Conferences on Artificial Intelligence* Bd. 1, Morgan Kaufmann, 1993, S. 640–645
- [40] ANDREKA ; RYAN ; SCHOBENS: Operators and Laws for Combining Preference Relations. In: *JLC: Journal of Logic and Computation* 12 (2002). URL: <ftp://ftp.cs.bham.ac.uk/pub/authors/M.D.Ryan/97-operators.ps>
- [41] HOFFMANN, J.: The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Variables. In: *Journal of Artificial Intelligence Research* 20 (2003), S. 291–341. – ISSN 1076–9757. URL: <http://www.cs.washington.edu/research/jair/volume20/hoffmann03a.ps>

Danksagung

Ich möchte mich zuerst bei meinem Betreuer Prof. Dr. Brewka sehr herzlich für die stets hilfreiche Unterstützung während des vergangenen Jahres bedanken. Mein spezieller Dank gilt auch Dr. Jörg Hoffmann für wertvollen Anregungen und die Hilfe bei der Anwendung des `Metric-FF`-Systems.

Ich möchte mich ebenfalls bei meiner Schwester Marianne, bei Alexander Hertsch und besonders bei Nadine Große bedanken, welche mir sehr bei der Fehlerkorrektur der Arbeit geholfen haben. Ich danke außerdem dem Institut für Theoretische Physik für technische Unterstützung.

Erklärung nach §27(9) der Prüfungsordnung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum, Unterschrift