ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Diplomarbeit

Subsumption deterministischer Aktionsschemata

Patrick Eyerich

Lehrstuhl für Grundlagen der Künstlichen Intelligenz
Betreuer: Prof. Dr. Bernhard Nebel
und Dipl.-Inf. Michael Brenner

April 2007

Danksagungen

Zunächst möchte ich mich bei Prof. Dr. Bernhard Nebel und Dipl.-Inf. Michael Brenner für die hervorragende Betreuung während der Anfertigung dieser Diplomarbeit bedanken.

Mein großer Dank gilt meinen Eltern und meinem Bruder, die mit ihrer enormen Unterstützung mein Studium, diese Arbeit und noch so vieles mehr erst ermöglicht haben.

Schließlich bedanke ich mich ganz besonders bei meiner Frau Birgit, die immer rückhaltlos hinter mir stand und steht und ohne die diese Arbeit wohl nie zu Stande gekommen wäre.

Widmen möchte ich diese Arbeit Lukas, Kevin und Lenja.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit von mir selbständig und nur unter Verwendung der aufgeführten Hilfsmittel erstellt wurde. Alle Stellen, die ich wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen habe, wurden als solche gekennzeichnet. Diese Diplomarbeit wurde weder als Ganzes noch in Auszügen für eine andere Prüfung angefertigt.

Freiburg, den 20.April 2007 Patrick Eyerich

Zusammenfassung

Aktionen und Aktionsschemata spielen eine zentrale Rolle in wichtigen Bereichen der Künstlichen Intelligenz wie zum Beispiel der Handlungsplanung. Die Möglichkeit, eine Menge von Aktionen auf Subsumptionsbeziehungen hin untersuchen und anhand dieser Beziehungen ordnen zu können, so dass eine Taxonomie beziehungsweise eine hierarchische Ordnung entsteht, würde eine Reihe von Vorteilen bieten. So könnte man zum Beispiel unabhängig voneinander entworfene Aktionsdomänen sehr viel besser miteinander vergleichen oder auch den Entwurf von Aktionsdomänen erleichtern, indem man neue Aktionsschemata als Generalisierungen oder Spezialisierungen bereits vorhandener Aktionschemata erstellt. Mit dieser Arbeit wird ein erster Schritt in diese Richtung gemacht. Aktionen, Aktionsschemata und deren Subsumption werden geeignet definiert und es wird gezeigt, dass es möglich ist, durch syntaktische Operationen zu entscheiden, ob ein Aktionsschema von einem anderen subsumiert wird. Es wird erläutert, warum eine natürliche Modellierung des Subsumptionsproblemes in OWL nicht möglich ist. Das Subsumptionsproblem wird durch Einschränkungen der Ausdruckskraft der beteiligten Aktionsschemata in Klassen eingeteilt und die Komplexität von drei wichtigen Klassen wird untersucht. Es wird gezeigt, dass das Subsumptionsproblem $\mathcal{NP}$-vollständig ist, wenn man die Ausdruckskraft auf die Planungssprache *STRIPS* beschränkt, Σ_2^p-vollständig, wenn man boolesche Operatoren zulässt und unentscheidbar, wenn man die Ausdruckskraft der Planungssprache *ADL* erlaubt. Für die ersten beiden Fälle werden Algorithmen vorgestellt und deren Implementation beschrieben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein zentraler Begriff in wichtigen Teilbereichen der Künstlichen Intelligenz, so zum Beispiel in der Handlungsplanung, ist der der *Aktion*. Eine Aktion überführt einen Zustand der Welt in einen anderen. In der Handlungsplanung wird eine Aktion typischerweise beschrieben, indem man angibt, was gelten muss, damit die Aktion durchgeführt werden kann (die *Vorbedingung* der Aktion) und welche Änderungen sich durch das Ausführen der Aktion in der Welt ergeben (die *Effekte* der Aktion).

Ein Aktionsschema (auch Operator genannt) ist eine Aktionsbeschreibung, in der Platzhalter, sogenannte Aktionsparameter, vorhanden sind. Durch die Instanzierung dieser Parameter mit Objekten entstehen dann Aktionen, die deshalb auch Aktions-Instanzen genannt werden.

Zur Beschreibung von Planungsdomänen im Bereich der Handlungsplanung, die im Wesentlichen aus Beschreibungen von Aktionsschemata und Objekten bestehen, hat sich in den letzten Jahren die *Planning Domain Definition Language* (*PDDL*) durchgesetzt. Mittlerweile existiert eine große Anzahl von separat entworfenen Aktionsdomänen in PDDL für sich teilweise überschneidende Anwendungsgebiete. Um herauszufinden, welche Zusammenhänge und Gemeinsamkeiten zwischen diesen Domänen und den einzelnen darin enthaltenen Aktionen existieren, beziehungsweise worin diese sich unterscheiden, kann man die einzelnen Aktionen verschiedener Domänen miteinander vergleichen. Hier ist man bis jetzt

jedoch auf einen reinen Gleichheitstest beschränkt, da noch kein Automatismus entwickelt wurde, der es ermöglicht, Aktionen darauf hin zu testen, ob sie in einer Subsumptionsbeziehung zueinander stehen, ob also ein Aktionsschema eine Generalisierung beziehungsweise eine Spezialisierung eines anderen Aktionsschemas ist.

Hier setzt diese Arbeit an: Der Subsumptionsbegriff zweier Aktionsschemata wird geeignet definiert und das Problem, zu entscheiden, ob eine Subsumptionsbeziehung zwischen zwei Aktionsschemata besteht, wird komplexitätstheoretisch untersucht. Dazu wird dieses Problem durch Beschränkungen der Ausdruckskraft der beteiligten Aktionsschemata in verschiedene Klassen aufgeteilt. Anschließend werden für zwei entscheidbare Klassen Algorithmen, die dieses Problem lösen, angegeben und deren Implementation beschrieben.

Die Möglichkeit, Aktionen hierarchisch gliedern zu können, wäre auch für das Design einer neuen Domäne sehr hilfreich. Ein solches Vorgehen wäre weniger fehleranfällig und sehr viel effizienter. Desweiteren würde eine hierarchisch gegliederte Aktionsdomäne einen höheren Grad an Modularität bieten: Einzelne Teile der Domäne könnten als Module in anderen Domänen eingesetzt werden, genauso könnten bereits vorhandene Module während des Designs weiterverwendet werden. PDDL ist für diese Zwecke im Allgemeinen nicht ausreichend, da bis jetzt keine Möglichkeit besteht, neue Aktionen auf der Basis von Generalisierungen oder Spezialisierungen bereits vorhandener Aktionen zu generieren.

Bei geeigneter Wahl der Subsumptionsbeziehung, so dass abstrakte Aktionen näher an der Wurzel stehen als alle ihre Spezialisierungen, könnte man zudem die verschiedenen Ebenen der Abstraktionshierarchie nutzen, um Heuristiken für Planungssysteme zu entwerfen.

Auch für das automatische Lernen von Aktionen ist es sinnvoll, zunächst von abstrakten Aktionen auszugehen und es dem Agenten dann zu ermöglichen, diese während des Lernprozesses genauer zu spezialisieren.

Die weitere Arbeit ist wie folgt gegliedert: In Kapitel 3 werden zunächst kurz die relevanten Grundlagen vorgestellt. In Kapitel 4 wird die Subsumption von Aktionsschemata definiert, ein syntaktisches Kriterium präsentiert, anhand dessen entschieden werden kann, ob ein Aktionsschema von einem anderen subsumiert

wird und schließlich wird noch auf die Modellierung des Subsumptionsproblemes in OWL eingegangen. In Kapitel 5 wird die Komplexität dreier Teilklassen von Aktionsschemata untersucht. Anschließend werden in Kapitel 6 Algorithmen für zwei dieser Klassen angegeben und deren Implementation kurz in Kapitel 7 beschrieben. Eine Zusammenfassung der erzielten Ergebnisse und ein Ausblick finden sich schließlich in Kapitel 8.

Kapitel 2

Verwandte Arbeiten

In diesem Kapitel geben wir einen Überblick über verwandte Arbeiten.

RAT [Heinsohn u. a., 1992], CLASP [Devanbu und Litman, 1991] und T-REX [Weida und Litman, 1992] sind Systeme zur Repräsentation von Aktionen in Beschreibungslogiken. RAT unterscheidet zwischen primitiven und komplexen Aktionen. Dabei sind komplexe Aktionen aus primitiven zusammengestellt und verfügen über globale Vor- und Nachbedingungen. Es ist aber nicht möglich, einzelne Aktionen zu spezialisieren. In CLASP und T-REX werden Subsumptionskriterien zwischen Plänen, nicht jedoch zwischen atomaren Aktionen definiert.

Liebig und Rösner [Liebig und Rösner, 1997] haben eine Syntax und Semantik für Aktionen basierend auf KL-ONE [Brachman und Schmolze, 1985] entworfen, wobei es möglich ist, Aktionen zu spezialisieren. Allerdings ist die Ausdruckskraft hierbei auf Disjunktionen und Konjunktionen von *Feature Chain Agreements* (Gleichheit der Rollenfüller funktionaler Rollen) beschränkt und es werden weder komplexitätstheoretische Untersuchungen angestellt, noch Algorithmen, die die Subsumption von Aktionen feststellen, präsentiert.

Im Bereich des hierarchischen Planens werden im Wesentlichen zwei verschiedene Abstraktions-Techniken verwendet: Auf der einen Seite werden mehrere Aktionen zu einer gemeinsamen zusammengefasst [Korf, 1987], während auf der anderen Seite Details der Welt-Zustands-Repräsentation weggelassen werden. Im zweiten Ansatz unterscheidet man zwischen *reduzierten Modellen* und *relaxierten Mo-*

dellen. Den reduzierten Modellen [Knoblock, 1994] liegt zu Grunde, dass einige Terme weggelassen werden, um verschiedene Abstraktionsräume zu erzeugen. In einem Abstraktionsraum, der durch das Weglassen von Termen entsteht, entsprechen dann mehrere Zustände des speziellen Raumes einem einzelnen Zustand. In den relaxierten Modellen [Sacerdoti, 1974] werden den Prädikaten *criticality values* zugeordnet, anhand derer die Vorbedingungen der Aktionen vereinfacht werden können, indem man alle Prädikate einer Vorbedingung, die unter einem bestimmten Grenzwert liegen, weglässt.

Gemeinsam haben diese beiden Ansätze, dass Abstraktionsbeziehungen zwischen Aktionen in verschiedenen Abstraktionsräumen bestehen, nicht jedoch zwischen Aktionen in derselben Domäne. Außerdem ist die Ausdruckskraft der untersuchten Aktionen auf die Ausdruckskraft der Planungssprache STRIPS [Fikes und Nilsson, 1971] beschränkt.

Kemke hat eine formale Theorie entworfen, um Aktionen in terminologischen Wissensbasen zu beschreiben und ist dabei auf das Subsumptions-Problem von Aktionen eingegangen [Kemke, 2003a,b]. Zusammen mit Walker hat sie einen Planungsalgorithmus entwickelt, der Aktions-Abstraktion und Plandekompositions-Hierarchien integriert [Kemke und Walker, 2006]. Auch hier werden jedoch keine komplexitätstheoretischen Untersuchungen gemacht und keine Algorithmen vorgestellt.

Kapitel 3

Grundlagen

In diesem Kapitel gehen wir kurz auf die für diese Arbeit relevanten formalen Grundlagen ein.

3.1 Propositionale Logik

3.1.1 Syntax

Wir gehen aus von einem abzählbaren *Alphabet* Σ eindeutiger atomarer Aussagen: $a, b, c \ldots$ und definieren darauf basierend die *aussagenlogischen Formeln*:

φ	$\longrightarrow$	a	atomare Aussage
	$\mid$	$\bot$	falsche Aussage
	$\mid$	$\top$	wahre Aussage
	$\mid$	$\neg\varphi'$	Negation
	$\mid$	$\varphi' \wedge \varphi''$	Konjunktion
	$\mid$	$\varphi' \vee \varphi''$	Disjunktion
	$\mid$	$\varphi' \Rightarrow \varphi''$	Implikation
	$\mid$	$\varphi' \Leftrightarrow \varphi''$	Äquivalenz

Dabei gelten folgende Operator-Vorrangsregeln: $\neg > \wedge > \vee > \Rightarrow = \Leftrightarrow$ (diese können durch Klammersetzung überlagert werden).

3.1.2 Semantik

Eine *Interpretation* für Σ ist eine Funktion

$$I : \Sigma \mapsto \{0, 1\}.$$

Eine aussagenlogische Formel φ wird von einer Interpretation I *erfüllt* $I \models \varphi$, wenn:

$$\begin{array}{rll} I \models a & \text{gdw.} & I(a) = 1 \\ & I \models \top & \\ & I \not\models \bot & \\ I \models \neg\varphi & \text{gdw.} & I \not\models \varphi \\ I \models \varphi \wedge \varphi' & \text{gdw.} & I \models \varphi \text{ und } I \models \varphi' \\ I \models \varphi \vee \varphi' & \text{gdw.} & I \models \varphi \text{ oder } I \models \varphi' \\ I \models \varphi \Rightarrow \varphi' & \text{gdw.} & \text{wenn } I \models \varphi, \text{ dann } I \models \varphi' \\ I \models \varphi \Leftrightarrow \varphi' & \text{gdw.} & I \models \varphi \text{ gdw. } I \models \varphi' \end{array}$$

3.1.3 Terminologie

Eine aussagenlogische Formel φ ist

- *erfüllbar* genau dann, wenn es eine Interpretation I gibt, so dass $I \models \varphi$
- *unerfüllbar* genau dann, wenn es keine Interpretation I gibt, so dass $I \models \varphi$
- *gültig* genau dann, wenn für alle Interpretationen I gilt, dass $I \models \varphi$
- *falsifizierbar* genau dann, wenn es eine Interpretation I gibt, so dass $I \not\models \varphi$

Atomare Aussagen, negierte atomare Aussagen, $\top$ und $\bot$ sind *Literale*. Eine aussagenlogische Formel ist in *konjunktiver Normalform* (*KNF*) genau dann, wenn sie eine Konjunktion von Disjunktionen von Literalen ist.

3.1.4 Quantifizierte Boolesche Formeln (*QBF*)

Wenn φ eine aussagenlogische Formel, P die Menge aller booleschen Variablen in φ, und ψ eine Sequenz von $\exists p$ und $\forall p$, wobei für jede Variable aus P genau ein Quantor vorkommt, ist, dann ist $\psi\varphi$ eine *Quantifizierte Boolesche Formel*.

Eine Formel $\exists x\phi$ ist genau dann wahr, wenn $\phi[\top, x] \vee \phi[\bot, x]$ wahr ist, während eine Formel $\forall x\phi$ ist genau dann wahr ist, wenn $\phi[\top, x] \wedge \phi[\bot, x]$ wahr ist.

3.2 Logik der ersten Stufe

3.2.1 Syntax

Wir gehen aus von

- eindeutigen *Variablen-Symbolen*: $x, y, z, \ldots$
- eindeutigen n-stelligen *Funktions-Symbolen*: $f, g, \ldots$
- eindeutigen *Konstanten-Symbolen*: $a, b, c, \ldots$
- eindeutigen n-stelligen *Prädikaten-Symbolen*: $P, Q, \ldots$ und
- *logischen Symbolen*: $\forall, \exists, =, \neg, \wedge, \ldots$

und definieren darauf aufbauend *Terme*:

t	$\longrightarrow$	x	Variable
	$\mid$	$f(t_1, \ldots, t_n)$	funktionaler Term
	$\mid$	a	Konstante

und *FOL-Formeln*:

φ	$\longrightarrow$	$P(t_1, \ldots, t_n)$	atomare Formel
	$\mid$	$\bot$	falsche Formel
	$\mid$	$\top$	wahre Formel
	$\mid$	$t = t'$	Identitäts-Formel
	$\mid$	$\neg\varphi'$	Negation
	$\mid$	$\varphi' \wedge \varphi''$	Konjunktion
	$\mid$	$\varphi' \vee \varphi''$	Disjunktion
	$\mid$	$\varphi' \Rightarrow \varphi''$	Implikation
	$\mid$	$\forall x(\varphi')$	Allquantifikation
	$\mid$	$\exists x(\varphi')$	Existenzquantifikation

Kommen in einem Term (einer Variablen, einem Prädikat,...) keine Variablen vor, so spricht man von *Grund*-Termen (*Grund*-Variablen, *Grund*-Prädikaten,...).

3.2.2 Semantik

Eine *Interpretation* $I = \langle \mathcal{D}, \cdot^I \rangle$ besteht aus einer beliebigen nicht-leeren Menge $\mathcal{D}$ und einer Funktion I, die

- n-stellige Funktions-Symbole f auf n-stellige Funktionen $f^I \in [\mathcal{D}^n \mapsto \mathcal{D}]$,
- Konstanten-Symbole a auf Objekte $a^I \in \mathcal{D}$ und
- n-stellige Prädikate P auf n-stellige Relationen $P^I \subseteq \mathcal{D}^n$

abbildet.

V ist die Menge der Variablen. Eine *Belegung* ist eine Funktion $\alpha : V \mapsto \mathcal{D}$. $\alpha[d, x]$ ist für $V \setminus x$ identisch zu α und $\alpha[d, x](x) = d$.

Interpretation von Termen unter I, α:

$$\begin{aligned} x^{I,\alpha} &= \alpha(x) \\ a^{I,\alpha} &= a^I \\ (f(t_1, \ldots, t_n))^{I,\alpha} &= f^I(t_1^{I,\alpha}, \ldots, t_n^{I,\alpha}) \end{aligned}$$

Da für Grundterme die Belegung keine Rolle spielt, gilt für diese:

$$(f(t_1, \ldots, t_n))^{I,\alpha} = f^I(t_1^I, \ldots, t_n^I)$$

Für Grundatome gilt:

$$I, \alpha \models P(t_1, \ldots, t_n) \text{ gdw. } \langle t_1^I, \ldots, t_n^I \rangle \in P^I$$

Eine Formel φ wird von I, α erfüllt ($I, \alpha \models \varphi$) ($I, \alpha$ ist ein Modell von φ) genau dann, wenn:

$$
\begin{aligned}
I, \alpha \models P(t_1, \ldots, t_n) &\quad \text{gdw.} \quad \langle t_1^{I,\alpha}, \ldots, t_n^{I,\alpha} \rangle \in P^I \\
I, \alpha \models t_1 = t_2 &\quad \text{gdw.} \quad t_1^{I,\alpha} = t_2^{I,\alpha} \\
I, \alpha \models \neg\varphi &\quad \text{gdw.} \quad I, \alpha \not\models \varphi \\
I, \alpha \models \varphi \wedge \psi &\quad \text{gdw.} \quad I, \alpha \models \varphi \text{ und } I, \alpha \models \psi \\
I, \alpha \models \varphi \vee \psi &\quad \text{gdw.} \quad I, \alpha \models \varphi \text{ oder } I, \alpha \models \psi \\
I, \alpha \models \varphi \Rightarrow \psi &\quad \text{gdw.} \quad \text{wenn } I, \alpha \models \varphi, \text{ dann } I, \alpha \models \psi \\
I, \alpha \models \varphi \Leftrightarrow \psi &\quad \text{gdw.} \quad I, \alpha \models \varphi \text{ gdw. } I, \alpha \models \psi \\
I, \alpha \models \forall x\varphi &\quad \text{gdw.} \quad I, \alpha[d, x] \models \varphi \text{ für alle } d \in \mathcal{D} \\
I, \alpha \models \exists x\varphi &\quad \text{gdw.} \quad I, \alpha[d, x] \models \varphi \text{ für mindestens ein } d \in \mathcal{D}
\end{aligned}
$$

3.2.3 Terminologie

Erfüllbarkeit, *Nichterfüllbarkeit*, *Falsifizierbarkeit* und *Gültigkeit* sind analog zum propositionalen Fall definiert.

Atomare Formeln, die falsche Formel und die wahre Formel sind *Atome*. Atome und negierte Atome sind *Literale*.

Zwei Formeln φ und ψ sind *logisch äquivalent* $\varphi \equiv \psi$ genau dann, wenn für alle I, α:

$$I, \alpha \models \varphi \text{ gdw. } I, \alpha \models \psi$$

Alle Variablen, die nicht von einem Quantor *gebunden* werden, sind *frei*.

3.3 Komplexitätstheorie

Wir gehen davon aus, das *Deterministische Turingmaschinen* (DTM) und *Nichtdeterministische Turingmaschinen* (NTM) bekannt sind [Garey und Johnson, 1979; Papadimitriou, 1994].

Ein *Problem* P ist eine Menge von Paaren $\langle I, A \rangle$ von Strings aus $\{0,1\}^*$, wobei I die *Instanz* und A die *Antwort* ist. Wenn $A \in \{0,1\}$, so nennt man P auch

Entscheidungsproblem. Ein Entscheidungsproblem definiert eine *Sprache* als die Menge aller Instanzen mit Antwort 1.

Ein Algorithmus *entscheidet* ein Problem, wenn er die richtige Antwort für alle Instanzen berechnet.

Die *Komplexität eines Algorithmus* ist eine Funktion

$$T : \mathbb{N} \mapsto \mathbb{N},$$

die die Anzahl der atomaren Berechnungsschritte (oder des Speicherplatzverbrauches) angibt, die der Algorithmus in Abhängigkeit von der Größe der Instanz benötigt, um die Antwort zu finden.

Die *Komplexität eines Problems* ist die Komplexität des effizientesten Algorithmus, der das Problem löst.

Für diese Arbeit sind vor allem drei Klassen von Problemen interessant: Die Klasse von Problemen, die auf DTM's in polynomieller Zeit lösbar sind ($\mathcal{P}$), die Klasse von Problemen, die auf NDTM's in polynomieller Zeit lösbar sind ($\mathcal{NP}$) und die Klasse von Problemen, die auf DTM's mit polynomiellem Platz lösbar sind ($\mathcal{PSPACE}$).

Eine polynomielle Reduktion von einer Sprache L_1 auf eine andere Sprache L_2 ist eine in Polynomialzeit berechenbare Funktion f, so dass $x \in L_1$ genau dann, wenn $f(x) \in L_2$ für alle Instanzen x.

3.3.1 $\mathcal{NP}$-Vollständigkeit

Ein Problem P ist $\mathcal{NP}$-vollständig genau dann, wenn es $\mathcal{NP}$-hart ist und in $\mathcal{NP}$ liegt. $\mathcal{NP}$-hart bedeutet, dass alle Probleme $P' \in \mathcal{NP}$ polynomiell auf P reduzierbar sind. Das bekannteste $\mathcal{NP}$-vollständige Problem ist SAT: das Erfüllbarkeitsproblem der Aussagenlogik.

3.3.2 Polynomielle Hierachie

Zwischen den Komplexitätsklassen $\mathcal{NP}$ und $\mathcal{PSPACE}$ lassen sich unendlich viele weitere Komplexitätsklassen definieren, die die sogenannte *Polynomielle Hiera-*

chie ($\mathcal{PH}$) [Meyer und Stockmeyer, 1972] bilden. Diese wird im Folgenden kurz vorgestellt.

Sei $\mathcal{X}$ eine Klasse von Entscheidungsproblemen. Wir benötigen zunächst den Begriff einer $\mathcal{X}$-Orakel-Turingmaschine. Informal ist eine $\mathcal{X}$-Orakel-Turingmaschine eine DTM oder eine NTM, die die Möglichkeit besitzt, ein Orakel für eine Instanz I von $\mathcal{X}$ aufzurufen. Das Orakel gibt in konstanter Zeit die zu I gehörige Antwort A aus.

$\mathcal{P}^{\mathcal{X}}$ ($\mathcal{NP}^{\mathcal{X}}$) bezeichnet nun diejenige Klasse von Entscheidungsproblemen, die in polynomieller Zeit auf einer DTM (NTM) unter Verwendung eines Orakels, dass $\mathcal{X}$ entscheidet, entschieden werden kann. Das Befragen des Orakels hat hierbei nur einen konstanten Zeitverbrauch. Die Mengen Δ_i^p, Σ_i^p und Π_i^p sind nun folgendermaßen definiert:

$$\begin{array}{lll} \Sigma_0^p = \mathcal{P} & \Pi_0^p = \mathcal{P} & \Delta_0^p = \mathcal{P} \\ \Sigma_{i+1}^p = \mathcal{NP}^{\Sigma_i^p} & \Pi_{i+1}^p = \text{co-}\Sigma_{i+1} & \Delta_{i+1}^p = \mathcal{P}^{\Sigma_i^p} \end{array}$$

Folglich ist $\Sigma_1^p = \mathcal{NP}$ und $\Delta_1^p = \text{co-}\mathcal{NP}$.

Wir definieren:

$$\mathcal{PH} = \bigcup_{i \geq 0} (\Sigma_i^p \cup \Pi_i^p \cup \Delta_i^p)$$

Es gilt:

$$\mathcal{NP} \subseteq \mathcal{PH} \subseteq \mathcal{PSPACE}$$

Analog zur unbeantworteten $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ Frage ist es nicht bekannt, ob es sich hierbei um echte Teilmengen-Beziehungen handelt oder nicht. Genau wie dort wird allerdings auch hier sehr stark vermutet, dass dem tatsächlich so ist.

Das bekannteste Π_k^p-vollständige Problem ist es, zu entscheiden, ob

$$\overbrace{\forall \vec{x_1} \exists \vec{x_2} \forall \ldots}^{k} \phi(\vec{x_1}, \vec{x_2}, \ldots)$$

gültig ist. Zu entscheiden, ob das dazu komplementäre Problem

$$\overbrace{\exists \vec{x_1} \forall \vec{x_2} \exists \ldots}^{k} \phi(\vec{x_1}, \vec{x_2}, \ldots)$$

gültig ist, ist Σ_k^p-vollständig.

Kapitel 4

Subsumption von Aktionen

4.1 Aktionen / Aktionsschemata

In diesem Kapitel definieren wir die für die weitere Arbeit wesentlichen Begriffe wie zum Beispiel Aktionen, Aktionsschemata sowie deren Subsumption. Anschließend stellen wir ein Kriterium vor, mit dessen Hilfe alleine durch syntaktische Operationen entschieden werden kann, ob zwei gegebene Aktionsschemata gemäß unserer Definition in einer Subsumptions-Beziehung stehen. Schließlich gehen wir darauf ein, warum es nicht möglich ist, das Subsumptionsproblem von Aktionsschemata auf eine natürliche Weise in der Web-Ontologie-Sprache OWL zu modellieren.

Zunächst definieren wir, was wir unter einer Aktion verstehen.

Definition 1 ***(Aktion).*** Eine (deterministische) *Aktion* a ist ein Tupel $a = \langle pre(a), eff(a) \rangle$. $pre(a)$ (die *Vorbedingung* von a) ist eine Formel der Prädikatenlogik der ersten Stufe ohne freie Variablen und ohne funktionale Terme[1]. $eff(a)$ (der *Effekt* von a) ist eine Konjunktion der Form $\bigwedge_{1 \leq i \leq n}(c_i \rhd e_i)$ mit endlichem n, wobei c_i eine Formel der Prädikatenlogik der ersten Stufe ohne freie Variablen und ohne funktionale Terme ist und e_i ein Grund-Literal.

[1] Als funktionale Terme bezeichnen wir an dieser Stelle Terme der Stelligkeit ≥ 1; Terme der Stelligkeit $= 0$, also Konstanten, sind natürlich erlaubt (siehe Kapitel 3.2.1).

In Definition 1 haben wir keine freien Variablen erlaubt. Alle Argumente der beteiligten Prädikate sind also Konstanten. Deshalb sprechen wir hierbei auch von einer *Aktions-Instanz*. Lassen wir dagegen freie Variablen zu, so erhalten wir die sogenannten *Aktionsschemata*. Für deren Definition müssen wir zunächst noch den Begriff der Objekt-Domäne und des Typs definieren:

Definition 2 ***(Typ, Objekt-Domäne).*** Ein *Typ* T ist eine Mengenvariable. Eine *Objekt-Domäne* $\Delta = \langle \Lambda, \theta \rangle$ besteht aus einer endlichen Menge von Objekten (Individuen, Konstanten) Λ und einer Abbildung θ, die allen Typen eine Teilmenge von Λ zuordnet (dabei werden nur die Typen T angegeben, für die $\theta(T) \neq \emptyset$). Es gibt in jeder Objekt-Domäne einen speziellen Typ THING, für den immer $\theta(\text{THING}) = \Lambda$ gilt.

Für eine Objekt-Domäne $\Delta = \langle \Lambda, \theta \rangle$ und eine Konstante c schreiben wir statt $c \in \Lambda$ ($c \notin \Lambda$) der Einfachheit halber auch $c \in \Delta$ ($c \notin \Delta$).

Definition 3 ***(Typhierarchie).*** Eine *Typhierachie* $\boldsymbol{T}$ ist eine endliche Menge von Regeln der Form $T \Rightarrow T'$, wobei T und T' Typen sind. Eine Objekt-Domäne $\Delta = \langle \Lambda, \theta \rangle$ *respektiert* eine Typhierarchie $\mathbf{T}$ genau dann, wenn für jede Regel $T \Rightarrow T'$ aus $\mathbf{T}$ gilt: $\theta(T) \subseteq \theta(T')$.

Definition 4 ***(Aktionsschema).*** Ein (deterministisches) *Aktionsschema* A ist ein Tripel $A = \langle pre(A), eff(A), types_A \rangle$. $pre(A)$ (die *Vorbedingung* von A) ist eine Formel der Prädikatenlogik der ersten Stufe ohne funktionale Terme. $eff(A)$ (der *Effekt* von A) ist eine Konjunktion der Form $\bigwedge_i (V_i, T_i : (C_i \rhd E_i))$, wobei C_i eine Formel der Prädikatenlogik der ersten Stufe ohne funktionale Terme und E_i ein Literal ist. V_i ist eine Menge von Variablen und T_i bildet jede Variable in V_i auf einen Typen ab. $types_A$ ist eine Abbildung, die jeder freien Variablen in $pre(A)$ und $eff(A)$, die nicht in einem V_j vorkommt, einen Typ zuordnet. Die freien Variablen in $pre(A)$ sowie die Elemente aller V_i sind paarweise verschieden.

Den Grundbereich der Abbildung $types_A$ in einem Aktionsschema A bezeichnen wir als *Schemavariablen* oder *Parameter* $\mathcal{S}(A)$ von A.

Ein Aktionsschema A ist *definiert* in einer Objekt-Domäne $\Delta = \langle \Lambda, \theta \rangle$, wenn $\theta(types_A(x)) \neq \emptyset$ für alle $x \in \mathcal{S}(A)$, wenn also die Typen aller Parameter von A jeweils mindestens ein Objekt enthalten.

Ein in einer Objekt-Domäne $\Delta = \langle \Lambda, \theta \rangle$ definiertes Aktionsschema repräsentiert in Abhängigkeit von Δ eine Menge von Aktionen:

Definition 5 ***(Repräsentierte Aktion).*** Wir erhalten eine in Δ von A repräsentierte Aktion $a = \langle pre(a), eff(a) \rangle$, indem wir zunächst eine zulässige Belegung α für $\mathcal{S}(A)$ wählen. Zulässig bedeutet hier, dass $\forall x \in \mathcal{S}(A) : \alpha(x) \in \theta(types_A(x))$. Ist $V_i \neq \emptyset$, so enthält $eff(a)$ für jede zulässige Belegung β_j von V_i einen Effekt (c_{ij}, e_{ij}), der entsteht, indem in $(C_i \rhd E_i)$ alle $y \in V_i$ durch $\beta_j(y)$ ersetzt werden (hier bedeutet zulässig, dass $\forall y \in V_i : \beta_j(x) \in \theta(T_i(x))$). Dies nennen wir das Abflachen von allquantifizierten Effekten.

Die durch ein Aktionsschema A in einer Objekt-Domäne Δ repräsentierten Aktionen nennen wir $Actions_\Delta(A)$.

Für die spätere Klassifizierung unterschiedlich schwieriger Subsumptionsprobleme ist es hilfreich, Aktionsschemata in unterschiedliche Klassen einzuteilen. Dies geschieht durch Einschränkungen der Ausdruckskraft von $pre(A)$ und $eff(A)$:

Definition 6 ***(Klassen des Subsumptionsproblems).*** Beschränken wir $pre(A)$ auf Konjunktionen von positiven Prädikaten und in $eff(A)$ alle c_i auf $\top$ und alle V_i auf $\emptyset$, so erhalten wir die Klasse $\mathcal{A}_{STRIPS}$. Erlauben wir zusätzlich die allgemeine Negation und Disjunktionen in $pre(A)$ und allen c_i und Konjunktionen in den c_i, so erhalten wir die Klasse $\mathcal{A}_{BOOL}$. Erlauben wir zusätzlich noch Existenzquantifizierungen und Allquantifizierungen in $pre(A)$ sowie in den c_i von $eff(A)$ und allquantifizierte Effekte (also beliebige Variablenmengen für die V_i), so erhalten wir die Klasse $\mathcal{A}_{ADL}$.

Diese Definitionen wurden so gewählt, dass die Ausdruckskraft von $\mathcal{A}_{STRIPS}$ derjenigen der Planungssprache STRIPS [Fikes und Nilsson, 1971] und die Ausdruckskraft von $\mathcal{A}_{ADL}$ derjenigen der Planungssprache ADL [Pednault, 1989] entspricht.

Ist ein Effekt nicht allquantifiziert ($V_i = \emptyset$), so schreiben wir kurz ($C_i \rhd E_i$) statt ($\emptyset, \emptyset : C_i \rhd E_i$).

Beispiel 1. Wir betrachten das Aktionsschema $Move$:

$$\begin{aligned} Move = \langle \quad & At(x,y) \wedge Connected(y,z), \\ & (\top \rhd \neg At(x,y)) \wedge \\ & (\top \rhd At(x,z)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(x,b) \rhd \neg At(b,y)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(x,b) \rhd At(b,z)), \\ & \{x \mapsto Agent, y \mapsto Location, z \mapsto Location\}\rangle \end{aligned}$$

Es ist $\mathcal{S}(Move) = \{x, y, z\}$ und $Move$ ist auf Grund der verwendeten allquantifizierten Effekte in der Klasse $\mathcal{A}_{ADL}$.

Jetzt betrachten wir eine Objekt-Domäne $\Delta = \langle \Lambda, \theta \rangle$ mit

$$\Lambda = \{agentA, agentB, locationA, locationB, locationC, ballA, ballB\}$$

und

$$\begin{aligned} &\theta(Agent) = \{agentA, agentB\} \\ &\theta(Location) = \{locationA, locationB, locationC\} \\ &\theta(Ball) = \{ballA, ballB\}. \end{aligned}$$

Es gibt insgesamt $|\theta(Agent)| * |\theta(Location)| * |\theta(Location)| = 18$ zulässige Belegungen für $\mathcal{S}(Move)$ in Δ. Eine davon ist z.B. $\alpha_1 = \{x \mapsto agentA, y \mapsto locationA, z \mapsto locationB\}$

Wir belegen zunächst $\mathcal{S}(A)$ mit α_1:

$$\begin{aligned} Move'_1 = \langle \quad & At(agentA, locationA) \wedge Connected(locationA, locationB), \\ & (\top \rhd \neg At(agentA, locationA)) \wedge \\ & (\top \rhd At(agentA, locationB)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(agentA, b) \rhd \neg At(b, locationA)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(agentA, b) \rhd At(b, locationB)\rangle \end{aligned}$$

Für jede zulässige Belegung von $\{b\}$ (dies sind $\beta_1 = \{b \mapsto ballA\}$ und $\beta_2 = \{b \mapsto$

$ballB\}$ ersetzen wir dann die beiden unteren Effektschemata und erhalten so:

$$\begin{aligned} Move_1 = \langle \quad & At(agentA, locationA) \wedge Connected(locationA, locationB), \\ & (\top \rhd \neg At(agentA, locationA)) \wedge \\ & (\top \rhd At(agentA, locationB)) \wedge \\ & (Carry(agentA, ballA) \rhd \neg At(ballA, locationA)) \wedge \\ & (Carry(agentA, ballB) \rhd \neg At(ballB, locationA)) \wedge \\ & (Carry(agentA, ballA) \rhd At(ballA, locationB)) \wedge \\ & (Carry(agentA, ballB) \rhd At(ballB, locationB)) \rangle \end{aligned}$$

Auf analoge Weise erhalten wir für jede der 17 anderen möglichen Belegungen von $\mathcal{S}(Move)$ eine Aktion. Es ist $Actions_{\Delta}(Move) = \{Move_1, \ldots, Move_{18}\}$.

4.2 Abstrakte Subsumption

Welche Kriterien müssen nun erfüllt sein, damit eine Aktion von einer anderen subsumiert wird? Hierfür gibt es verschiedene Möglichkeiten [Liebig und Rösner, 1997; Weida und Litman, 1992; Heinsohn u. a., 1992].

Die beiden für uns in Frage kommenden Alternativen hierbei sind die *Anwendbarkeits-Subsumption* sowie die *Abstraktions-Subsumption*.

Die Definition der Anwendbarkeits-Subsumption basiert auf der Idee, dass eine Aktion B genau dann eine andere Aktion A subsumiert, wenn B immer dann anwendbar ist, wenn A anwendbar ist. Um dies zu gewährleisten, darf A offensichtlich keine schwächere Vorbedingung als B sowie höchstens die gleichen Effekte (mit nicht schwächeren Vorbedingungen) wie B haben. Wann immer eine Aktion B also eine Aktion A gemäß der Anwendbarkeits-Subsumption subsumiert, können wir statt A auch B anwenden.

Dies ist ein sehr interessanter Ansatz; er entspricht jedoch nicht unserem intuitiven Verständnis von Generalisierung und Spezialisierung von Aktionen. Wir wollen stattdessen, dass eine Spezialisierung A einer Aktion B alle Eigenschaften von B besitzt - also implizit deren Vorbedingung sowie alle Effekte (wobei die Vorbedingung eines Effektes in A nicht stärker sein darf als die des entsprechenden Effektes in B). Dieser Intuition entspricht die Abstraktions-Subsumption, die

wir im Folgenden genauer definieren.

Wir schreiben kurz $c_i \in eff(a)$ bzw. $e_i \in eff(a)$, falls (c_i, e_i) ein Konjunkt in $eff(a)$ ist.

Definition 7 ***(Abstrakte Subsumption von Aktionen).*** Eine Aktion a wird von einer Aktion b *abstrakt subsumiert* $a \subseteq_{as} b$ genau dann, wenn

1. $pre(a) \Rightarrow pre(b)$
2. $\forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

Hiermit können wir nun die abstrakte Subsumption zweier Aktionsschemata definieren. Da ein Aktionsschema in Abhängigkeit von einer Objekt-Domäne Δ für eine Menge von Aktionen steht, liegt es nahe, die Subsumption von Aktionsschemata auf der Subsumption von Aktionen zu basieren (siehe Abbildung 4.1):

Definition 8 ***(Abstrakte Subsumption von Aktionsschemata).*** Ein Aktionsschema A wird von einem Aktionsschema B *abstrakt subsumiert* $A \subseteq_{as} B$ genau dann, wenn für jede Objekt-Domäne Δ, in der A und B definiert sind, vollständige Abbildungen

$$f_1 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

und

$$f_2 : Actions_\Delta(B) \mapsto Actions_\Delta(A)$$

existieren, für die gilt, dass

1. $\forall a \in Actions_\Delta(A) : a \subseteq_{as} f_1(a)$
2. $\forall b \in Actions_\Delta(B) : f_2(b) \subseteq_{as} b.$

Intuitiv besagt Definition 8, dass es zu jeder Aktion $a \in Actions_\Delta(A)$ eine Aktion $b \in Actions_\Delta(B)$ und zu jeder Aktion $b \in Actions_\Delta(B)$ eine Aktion $a \in Actions_\Delta(A)$ gibt, so dass a abstrakt von b subsumiert wird.

Eine Abbildung von $Actions_\Delta(A)$ nach $Actions_\Delta(B)$ alleine genügt nicht, wie Beispiel 2 illustriert:

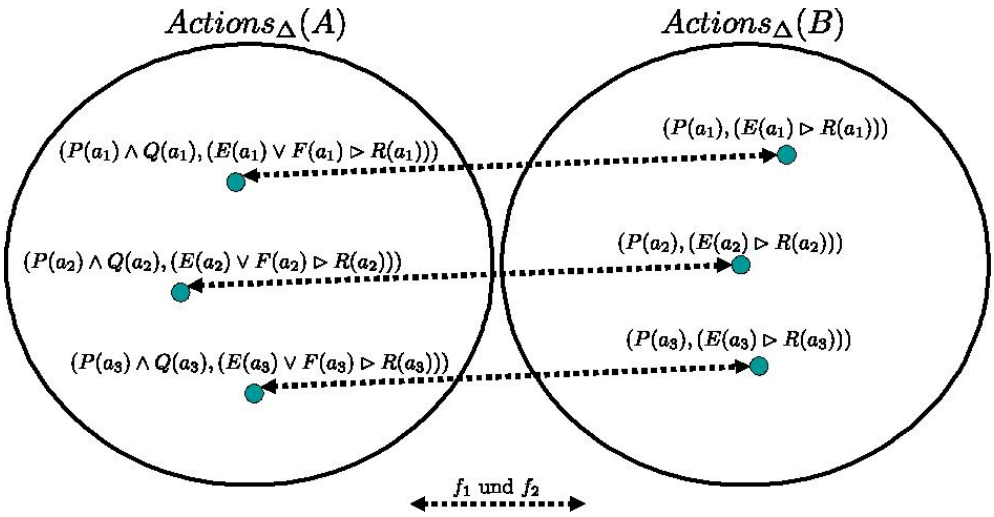

Abbildung 4.1: $\Delta = \langle\{a_1, a_2, a_3\}, \{Agent \mapsto \{a_1, a_2, a_3\}\}\rangle$, $A = \langle P(x) \wedge Q(x), (E(x) \vee F(x) \rhd R(x)), \{x \mapsto Agent\}\rangle$, $B = \langle P(x), (E(x) \rhd R(x)), \{x \mapsto Agent\}\rangle$, $A \subseteq_{as} B$

Beispiel 2. Sei $A = \langle P(x), (), \{x \mapsto T\}\rangle$ und $B = \langle P(x) \wedge P(y), (), \{x \mapsto T\}\rangle$. In der Objekt-Domäne $\Delta = \langle\{c_1, c_2\}, \{T \mapsto \{c_1, c_2\}\}\rangle$ gibt es zwei Aktionen in $Actions_\Delta(A)$: $a_1 = \langle P(a), ()\rangle$ und $a_2 = \langle P(b), ()\rangle$. In $Actions_\Delta(B)$ gibt es unter anderem die Aktionen $b_1 = \langle P(a) \wedge P(a), ()\rangle$ und $b_2 = \langle P(b) \wedge P(b), ()\rangle$. Mit $f_1 : \{a_1 \mapsto b_1, a_2 \mapsto b_2\}$ hat man nun eine vollständige Abbildung von $Actions_\Delta(A)$ nach $Actions_\Delta(B)$, so dass für alle Aktionen $a \in Actions_\Delta(A)$ gilt, dass $a \subseteq_{as} f_1(a)$. Es ist leicht zu sehen, dass auf diese Weise solche Abbildungen für alle Objekt-Domänen konstruiert werden können. Würde man also nur eine Abbildung für diese eine Richtung fordern, würde hier $A \subseteq_{as} B$ gelten – was wir natürlich nicht wollen.

Deshalb fordern wir ebenfalls eine Abbildung für die andere Richtung. In Beispiel 2 sieht man sofort, dass es zum Beispiel für $b_3 = \langle P(a) \wedge P(b), ()\rangle$ keine Aktion $a \in Actions_\Delta(A)$ geben kann, so dass $a \subseteq_{as} b_3$. Nach Definition 8 gilt also nicht $A \subseteq_{as} B$.

Bemerkung 1. Abstrakte Subsumption von Aktionsschemata ist transitiv.

Beispiel 3. Betrachte

$$\begin{aligned} Move = \langle \quad & At(x,y) \wedge Connected(y,z), \\ & (\top \rhd \neg At(x,y)) \wedge \\ & (\top \rhd, At(x,z)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(x,b) \rhd \neg At(b,y)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(x,b) \rhd At(b,z)), \\ & \{x \mapsto Agent, y \mapsto Location, z \mapsto Location\}\rangle \end{aligned}$$

und

$$\begin{aligned} MoveWithTrain = \langle \quad & At(q,r) \wedge Connected(r,s) \wedge At(t,r), \\ & (\top \rhd \neg At(q,r)) \wedge \\ & (\top \rhd At(q,s)) \wedge \\ & (\top \rhd \neg At(t,r)) \wedge \\ & (\top \rhd At(t,s)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(q,b) \rhd \neg At(b,r)) \wedge \\ & (\{b\}, \{b \mapsto Ball\} : Carry(q,b) \rhd At(b,s)), \\ & \{q \mapsto Agent, r \mapsto Location, s \mapsto Location, t \mapsto Train\}\rangle \end{aligned}$$

Behauptung 1. *Es gilt* $MoveWithTrain \subseteq_{as} Move$.

Beweis 1 *(Behauptung 1).* Wir müssen zeigen, dass für jede Objekt-Domäne Δ vollständige Abbildungen f_1 von $Actions_\Delta(MoveWithTrain)$ nach $Actions_\Delta(Move)$ und f_2 von $Actions_\Delta(Move)$ nach $Actions_\Delta(MoveWithTrain)$ existieren, für die gilt, dass

i) für alle Aktionen a aus $Actions_\Delta(MoveWithTrain)$ gilt, dass

   (a) $pre(a) \Rightarrow pre(f_1(a))$

   (b) $\forall i(e_i \in eff(f_1(a)) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

ii) für alle Aktionen b aus $Actions_\Delta(Move)$ gilt, dass

   (a) $pre(f_2(b)) \Rightarrow pre(b)$

   (b) $\forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(f_2(b)) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

Dazu betrachten wir eine beliebige Objekt-Domäne

$$\begin{aligned}\Delta = \langle \quad & \{a_1, a_2, \ldots, l_1, l_2, \ldots, t_1, t_2, \ldots, b_1, b_2, \ldots\}, \\ & \{Agent \mapsto \{a_1, a_2, \ldots\}, Location \mapsto \{l_1, l_2, \ldots\}, \\ & Train \mapsto \{t_1, t_2, \ldots\}, Ball \mapsto \{b_1, b_2, \ldots\}\}\rangle.\end{aligned}$$

Sei a eine beliebige Aktion aus $Actions_\Delta(MoveWithTrain)$ und α die dazugehörige Belegung von $\mathcal{S}(MoveWithTrain)$. Wir wählen die Belegung β für $f_1(a)$ in Abhängigkeit von α: $\beta(x) = \alpha(q)$, $\beta(y) = \alpha(r)$ und $\beta(z) = \alpha(t)$. Man kann sich leicht überzeugen, dass dann $a \subseteq_{as} f(a)$ gilt.

Analog finden wir für eine beliebige Aktion b aus $Actions_\Delta(Move)$ eine passende Aktion $f_2(b) \in Actions_\Delta(A)$.

Da Δ und in Δ dann a beziehungsweise b beliebig gewählt waren, folgt $MoveWithTrain \subseteq_{as} Move$.

4.2.1 Prädikaten-Inklusions-Hierarchien

Oft ist es sinnvoll, zusätzliche Axiome zu erlauben. In Beispiel 3 wäre es zum Beispiel gut denkbar, dass $pre(MoveWithTrain)$ nicht $At(q, r) \wedge Connected(r, s) \wedge At(t, s)$ lautet, sondern $At(q, r) \wedge Connected\_by\_Rail(r, s) \wedge At(t, s)$. Dann würde natürlich nicht länger $MoveWithTrain \subseteq_{as} Move$ gelten, weil kein semantischer Zusammenhang zwischen $Connected(r, s)$ und $Connected\_by\_Rail(r, s)$ existiert. Man würde deshalb gerne formulieren, dass $Connected(x, y)$ immer dann gilt, wenn $Connected\_by\_Rail(x, y)$ gilt. Genau solche Axiome wollen wir jetzt erlauben:

Definition 9 ***(Prädikaten-Inklusions-Axiom).*** Ein Prädikaten-Inklusions-Axiom ist ein Satz der Form

$$\forall \vec{x}(P(\vec{x}) \Rightarrow Q(\vec{x})).$$

Definition 10 ***(Prädikaten-Inklusions-Hierachie).*** Eine Prädikaten-Inklusions-Hierachie $\mathbf{P}$ ist eine endliche Menge von Prädikaten-Inklusions-Axiomen.

Bemerkung 2. Da in einem Prädikaten-Inklusions-Axiom $\mathbf{P}_1 = \forall\vec{x}(P(\vec{x}) \Rightarrow Q(\vec{x}))$ in P und Q dieselben Argumente auftreten, die alle allquantifiziert sind, und Prädikatennamen eindeutig sind, genügt es, $\mathbf{P}_1$ als $P \Rightarrow Q$ zu schreiben.

Mit Definition 10 können wir nun Definition 7 erweitern:

Definition 11 ***(Abstrakte Subsumption von Aktionen mit Prädikaten-Inklusions-Hierachie).*** Eine Aktion a wird von einer Aktion b abstrakt subsumiert bezüglich einer Menge $\mathbf{P}$ von Prädikaten-Inklusions-Axiomen $a \subseteq_{as}^{\mathbf{P}} b$ genau dann, wenn

1. $pre(a) \wedge \bigwedge_{p\in\mathbf{P}} p \Rightarrow pre(b)$
2. $\forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(a) \wedge (e_j \wedge \bigwedge_{p\in\mathbf{P}} p \Rightarrow e_i) \wedge (c_i \wedge \bigwedge_{p\in\mathbf{P}} p \Rightarrow c_j)))$

Auch Definition 8 kann direkt angepasst werden:

Definition 12 ***(Abstrakte Subsumption von Aktionsschemata mit Prädikaten-Inklusions-Hierachie).*** Ein Aktionsschema A wird von einem Aktionsschema B abstrakt subsumiert bezüglich einer Menge $\mathbf{P}$ von Prädikaten-Inklusions-Axiomen $A \subseteq_{as}^{\mathbf{P}} B$ genau dann, wenn für jede Objekt-Domäne Δ vollständige Abbildungen

$$f_1 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

und

$$f_2 : Actions_\Delta(B) \mapsto Actions_\Delta(A)$$

existieren, für die gilt, dass

1. $\forall a \in Actions_\Delta(A) : a \subseteq_{as}^{\mathbf{P}} f_1(a)$
2. $\forall b \in Actions_\Delta(B) : f_2(b) \subseteq_{as}^{\mathbf{P}} b$

4.2.2 Typ-Hierarchien

Unsere bisherige Definition der Subsumption verlangt, dass alle gemeinsamen Parameter zweier Aktionsschemata A und B den selben Typ haben müssen, wenn $A \subseteq_{as} B$ gelten soll. Es ist jedoch sinnvoll, einen Operator spezialisieren zu können, indem man den Typen eines Parameters einschränkt. Deshalb erweitern wie die bisherige Definition ein weiteres Mal:

Definition 13 ***(Abstrakte Subsumption von Aktionsschemata mit Prädikaten-Inklusions-Hierachie und Typ-Hierarchie).*** Ein Aktionsschema A wird von einem Aktionsschema B abstrakt subsumiert bezüglich einer Prädikaten-Inklusion-Hierarchie $\mathbf{P}$ und einer Typ-Hierachie $\mathbf{T}$ $A \subseteq_{as}^{\mathbf{P},\mathbf{T}} B$ genau dann, wenn für jede Objekt-Domäne Δ vollständige Abbildungen

$$f_1 : Actions_{\Delta}(A) \mapsto Actions_{\Delta}(B)$$

und

$$f_2 : Actions_{\Delta}(B) \mapsto Actions_{\Delta_{\mathbf{T}}}(A)$$

existieren, für die gilt, dass

1. $\forall a \in Actions_{\Delta}(A) : a \subseteq_{as}^{\mathbf{P}} f_1(a)$
2. $\forall b \in Actions_{\Delta}(B) : f_2(b) \subseteq_{as}^{\mathbf{P}} b$

Dabei wird $\Delta_{\mathbf{T}}$ konstruiert, indem man die Typ-Hierarchie $\mathbf{T}$ in Δ hineinkompiliert: Man beginnt mit Δ und erweitert es folgendermaßen: Für jedes $c \in \Delta$, für das eine Regel $T_1 \Rightarrow T_2$ in $\mathbf{T}$ existiert und $\theta(T_2) = \{\ldots, c, \ldots\}$ sowie $c \notin \theta(T_1)$ gilt, setzen wir $\theta(T_1) = \{\ldots, c, \ldots\}$.

Die Notwendigkeit dieser Erweiterung wird deutlich, wenn wir das folgende Beispiel betrachten: Angenommen wir haben die beiden Aktionsschemata $A = \langle P(x), (), \{x \mapsto T_1\}\rangle$ und $B = \langle P(x), (), \{x \mapsto T_2\}\rangle$ gegeben. Dann gilt $A \subseteq_{as} B$ natürlich nicht, da $types_A(x) \neq types_B(x)$. Wenn wir allerdings die Typ-Hierarchie $\mathbf{T}$=$\{T_1 \Rightarrow T_2\}$ in Betracht ziehen, dann sollte $A \subseteq_{as}^{\mathbf{T}} B$ gelten. Betrachten wir zum Beispiel die Objekt-Domäne $\Delta = \langle\{c_1, c_2\}, \{T_1 \mapsto \{c_1\}, T_2 \mapsto \{c_1, c_2\}\}\rangle$. Dann gibt es keine Aktion a in $Actions_{\Delta}(A)$ so dass $a \subseteq_{as} \langle P(c_2), ()\rangle$

gilt; also gilt auch $A \subseteq_{as} B$ nicht. Wenn wir aber $\mathbf{T}$ in Δ hineinkompilieren, so erhalten wir $\Delta_{\mathbf{T}} = \langle\{c_1, c_2\}, \{T_1 \mapsto \{c_1, c_2\}, T_2 \mapsto \{c_1, c_2\}\}\rangle$. Nun finden wir natürlich sofort eine solche Aktion, genau wie für jede andere Aktion in jeder anderen Domäne – also gilt $A \subseteq_{as}^{\mathbf{T}} B$.

4.2.3 Substitution von Parametern

Ersetzen wir einen oder mehrere paarweise verschiedene Variablennamen $y_1, y_2, y_3, \ldots \in \left(\mathcal{S}(A) \cup \bigcup_i V_i\right)$ in einem Aktionsschema A durch paarweise verschiedene Variablennamen $x_1, x_2, x_3, \ldots \notin \left(\mathcal{S}(A) \cup \bigcup_i V_i\right)$, so schreiben wir das entstehende Aktionsschema als $A[\frac{x_1}{y_1}\frac{x_2}{y_2}\frac{x_3}{y_3}\ldots]$ und nennen es die Substitution A^s von A bezüglich der Parameterersetzung $s = [\frac{x_1}{y_1}\frac{x_2}{y_2}\frac{x_3}{y_3}\ldots]$. Wichtig hierbei ist, dass alle Variablen paarweise verschieden sind (siehe Theorem 1).

Definition 14 ***(Identität von Aktionsschemata).*** Zwei Aktionsschemata A und B sind identisch $A \equiv B$ genau dann, wenn $Actions_\Delta(A) = Actions_\Delta(B)$ für jede Objekt-Domäne Δ gilt.

Proposition 1. *Es gilt* $A \equiv A[\frac{x}{y}]$ *für* $y \in \left(\mathcal{S}(A) \cup \bigcup_i V_i\right)$ *und* $x \notin \left(\mathcal{S}(A) \cup \bigcup_i V_i\right)$.

Beweis 2 *(Proposition 1).* Wir müssen zeigen, dass für jede Objekt-Domäne Δ gilt, dass $Actions_\Delta(A) = Actions_\Delta(A[\frac{x}{y}])$. Sei dazu Δ eine beliebige Objekt-Domäne und a eine beliebige Aktion aus $Actions_\Delta(A)$.

1. $x \in \mathcal{S}(A)$
   Sei α die Belegung der Variablen in $\mathcal{S}(A)$, die a erzeugt. Wir betrachten die Belegung α', die x auf $\alpha(y)$ abbildet und ansonsten identisch zu α ist. Belegen wir die Variablen aus $\mathcal{S}(A[\frac{x}{y}])$ mit α', so entsteht offensichtlich die Aktion a.

2. $x \in \bigcup_i V_i$
   Seien β_j die zulässigen Belegungen von y. Wir betrachten die Belegungen β'_j, die x auf $\beta_j(y)$ abbilden und ansonsten identisch zu β_j sind. Offensichtlich entstehen durch die β'_j dieselben Effekte wie durch die β_j und damit dieselbe Aktion a.

Also gilt $a \in Actions_\Delta(A) \Rightarrow a \in Actions_\Delta(A[\frac{x}{y}])$. Die Rückrichtung kann analog gezeigt werden.

Korollar 1. Es gilt $A \equiv B$ für zwei Aktionsschemata A und B genau dann, wenn es eine Substitution A' von A gibt, so dass $A' \equiv B$.

Beweis 3 *(Korollar 1).*

1. „⇒“
   Es gelte $A \equiv B$ für zwei Aktionsschemata A und B. Da A eine Substitution von sich selbst ist, folgt direkt die Behauptung.

2. „⇐“
   Es gäbe eine Substitution A' von A, so dass $A' \equiv B$. Aus Proposition 1 folgt, dass $A \equiv A'$ und damit dann direkt die Behauptung.

Korollar 2. Es gilt $A \subseteq_{as} B$ für zwei Aktionsschemata A und B genau dann, wenn es eine Substitution A' von A gibt, so dass $A' \subseteq_{as} B$.

Beweis 4 *(Korollar 2).*

1. „⇒“
   Es gelte $A \subseteq_{as} B$ für zwei Aktionsschemata A und B. Da A eine Substitution von sich selbst ist, folgt direkt die Behauptung.

2. „⇐“
   Es gäbe eine Substitution A' von A, so dass $A' \subseteq_{as} B$. Betrachte eine beliebige Objekt-Domäne Δ. Aus Proposition 1 folgt, dass $A \equiv A'$, also gilt nach Definition 14, dass $Actions_\Delta(A) = Actions_\Delta(A')$. Nach Voraussetzung gibt es nun eine vollständige Abbildung f_1, so dass für jede Aktion $a' \in Actions_\Delta(A')$ eine Aktion $f_1(a') \in Actions_\Delta(B)$ existiert, so dass $a' \subseteq_{as} f_1(a')$. Mit $Actions_\Delta(A) = Actions_\Delta(A')$ folgt daraus direkt, dass für f_1 auch gilt, dass es eine vollständige Abbildung von $Actions_\Delta(A)$ nach $Actions_\Delta(B)$ ist, so dass für jede Aktion $a \in Actions_\Delta(A)$ eine Aktion $f_1(a) \in Actions_\Delta(B)$ existiert, so dass $a \subseteq_{as} f_1(a)$.

Analog finden wir eine geeignete Abbildung f_2.

Da Δ beliebig gewählt war, gilt damit $A \subseteq_{as} B$.

4.3 Ein syntaktisches Kriterium für abstrakte Subsumption

Wie können wir nun für zwei Aktionsschemata A und B entscheiden, ob $A \subseteq_{as} B$ gilt? Nach Definition 8 gilt $A \subseteq_{as} B$ genau dann, wenn für jede Objekt-Domäne Δ vollständige Abbildungen

$$f_1 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

und

$$f_2 : Actions_\Delta(B) \mapsto Actions_\Delta(A)$$

existieren, für die gilt, dass

1. $\forall a \in Actions_\Delta(A) : a \subseteq_{as} f_1(a)$
2. $\forall b \in Actions_\Delta(B) : f_2(b) \subseteq_{as} b$.

Man könnte also alle Objekt-Domänen betrachten und für jede die Existenz entsprechender Abbildungen nachweisen. Da es aber unendlich viele Objekt-Domänen gibt, würde ein solches Vorgehen im Allgemeinen nicht terminieren, falls tatsächlich $A \subseteq_{as} B$ gilt. Deshalb sind wir an syntaktischen Kriterien interessiert, anhand derer wir entscheiden können, ob $A \subseteq_{as} B$ für zwei Aktionsschemata A und B gilt.

Theorem 1 ***(Syntaktisches Kriterium für*** $\boldsymbol{A \subseteq_{as} B}$***).*** *Es gilt $A \subseteq_{as} B$ für zwei Aktionsschemata A und B ohne allquantifizierte Effekte bezüglich einer Typhierachie $\boldsymbol{T}$ genau dann, wenn es eine Substitution A^s von A gibt, so dass*

(S1) $pre(A^s) \Rightarrow pre(B)$

(S2) $\forall i(E_i \in eff(B) \Rightarrow \exists j(E_j \in eff(A^s) \wedge E_i = E_j \wedge C_i \Rightarrow C_j))$

(S3) $types_{A^s}(x) \subseteq types_B(x)$ *folgt aus* $\boldsymbol{T}$*, falls* $x \in \big(\mathcal{S}(A^s) \cap \mathcal{S}(B)\big)$

Hier sieht man, warum in einer Substitution $s = [\frac{x_0}{y_0}\frac{x_1}{y_1}\frac{x_2}{y_2}\ldots]$ die paarweise Disjunktheit aller Variablen gefordert wird: Die paarweise Unterschiedlichkeit der y_i garantiert uns, dass alle Substitutionen eindeutig sind, wir also nicht eine Variable durch zwei verschiedene Variablen ersetzen, während die paarweise Disjunktheit der x_i garantiert, dass nicht zwei unterschiedliche Variablen durch dieselbe Variable ersetzt werden.

Warum diese zweite Bedingung notwendig ist, illustriert Beispiel 4:

Beispiel 4. Betrachte

$$B = \langle P(x_0, x_1) \wedge P(x_1, x_2), (), \{x_0 \mapsto T, x_1 \mapsto T, x_2 \mapsto T\}\rangle$$

und

$$A = \langle P(y_0, y_1) \wedge P(y_2, y_3), (), \{y_0 \mapsto T, y_1 \mapsto T, y_2 \mapsto T, y_3 \mapsto T\}\rangle$$

Betrachten wir die Substitution $s = [\frac{x_0}{y_0}\frac{x_1}{y_1}\frac{x_1}{y_2}\frac{x_2}{y_3}]$, so gelten offensichtlich S1-S3. In der Objekt-Domäne $\Delta = \langle\{a, b, c, d\}, \{T \mapsto \{a, b, c, d\}\}\rangle$ gibt es aber die Aktion $a = \langle P(a, b) \wedge P(c, d), ()\rangle \in Actions_\Delta(A)$, zu der es offensichtlich keine Aktion $b \in Actions_\Delta(B)$ gibt, so dass $pre(a) \Rightarrow pre(b)$, es gilt also nicht $A \subseteq_{as} B$, obwohl es eine Substitution A^s von A gibt, so dass S1-S3 erfüllt sind.

$[\frac{x_0}{y_0}\frac{x_1}{y_1}\frac{x_1}{y_2}\frac{x_2}{y_3}]$ ist jedoch keine zulässige Substitution, da zwei Variablen durch x_1 ersetzt werden. Theorem 1 wird hierdurch also nicht verletzt. Dass es tatsächlich korrekt ist, zeigen wir jetzt:

Beweis 5 *(Theorem 1).*

1. „⇒“

   Es gelte $A \subseteq_{as} B$.

   O.B.d.A. gäbe es keinen gemeinsamen Variablennamen in A und B.

   Wir betrachten eine beliebige Objekt-Domäne $\Delta = \langle\Lambda, \theta\rangle$, für die gilt, dass

(T1) $|\theta(T)| \geq n$ für alle Typen T aus dem Zielbereich von $types_A$ und $types_B$, wobei n=max(# Parameter von A, # Parameter von B).

(T2) Δ respektiert $\mathbf{T}$.

(T3) Folgt für zwei Typen T_1 und T_2 nicht $T_1 \Rightarrow T_2$ aus $\mathbf{T}$, so haben T_1 und T_2 keine gemeinsamen Objekte.

Da nun $A \subseteq_{as} B$ gilt, gibt es für Δ vollständige Abbildungen

$$f_1 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

und

$$f_2 : Actions_\Delta(B) \mapsto Actions_\Delta(A)$$

mit:

(a) i. $\forall a \in Actions_\Delta(A) : pre(a) \Rightarrow pre(f_1(a))$

ii. $\forall a \in Actions_\Delta(A) : \forall i(e_i \in eff(f_1(a)) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

(b) i. $\forall b \in Actions_\Delta(B) : pre(f_2(b)) \Rightarrow pre(b)$

ii. $\forall b \in Actions_\Delta(B) : \forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(f_2(b)) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

Eine Aktion $a \in Actions_\Delta(A)$ entsteht gemäß Definition 5 aus A durch eine zulässige Belegung aller Variablen $x \in \mathcal{S}(A)$ und durch das Abflachen der allquantifizierten Effekte. Deshalb hat jede Aktion $a \in Actions_\Delta(A)$ dieselbe syntaktische Struktur Υ^A; damit ist gemeint, dass alle Aktionen $a \in Actions_\Delta(A)$ bis auf die Argumente der Prädikate identisch sind. Ordnet man alle Argumente aller Prädikate von links nach rechts, so kann man eine Aktion $a \in Actions_\Delta(A)$ also eineindeutig durch deren syntaktische Struktur Υ^A zusammen mit den Prädikaten-Argumenten in der Reihenfolge ihres Auftretens in der syntaktischen Struktur beschreiben.

Da keine allquantifizierten Effekte verwendet werden, ist die syntaktische Struktur der Aktionen identisch zur syntaktischen Struktur des Aktionsschemas und somit ist durch das Aktionsschema und die Belegung α der Parameter $x \in \mathcal{S}(A)$ eine Aktion eineindeutig beschrieben.

Analoges gilt natürlich für $b \in Actions_\Delta(B)$.

Wir konstruieren nun aus A und $Actions_\Delta(A)$ ein Aktionsschema A', so dass S1-S3 für A' und B gelten und A' eine Substitution von A ist (siehe auch Beispiel 5).

Wir betrachten die Menge A_0 aller Aktionen $a \in Actions_\Delta(A)$, in denen jede Variable $x \in \mathcal{S}(A)$ durch eine unterschiedliche Konstante belegt wurde – es muss wegen T1 mindestens eine solche Aktion geben.

Wir machen eine Fallunterscheidung:

a) Es gibt eine Aktion $b \in Actions_\Delta(B)$, für die gilt:
   - $a \subseteq_{as} b \wedge f_1(a) = b$ oder $a \subseteq_{as} b \wedge f_2(b) = a$ für ein $a \in A_0$
   - in b wurde jede Variable $y \in \mathcal{S}(B)$ durch eine unterschiedliche Konstante belegt.

   Wir betrachten die erzeugende Belegung β von b und die dazugehörige Aktion a mit ihrer erzeugenden Belegung α.

   Es kann zu jeder Variablen $x \in \mathcal{S}(A)$ höchstens eine Variable $y \in \mathcal{S}(B)$ geben, so dass x in α und y in β durch dieselbe Konstante c belegt wurden.

   Wir ersetzen nun in a und $f_1(a)$ alle Vorkommen von c durch y und wiederholen dieses Vorgehen für alle $x \in \mathcal{S}(A)$ und $y \in \mathcal{S}(B)$, die durch dieselbe Konstante ersetzt wurden.

   Wenn eine Variable $x \in \mathcal{S}(A)$ in α durch c' belegt wird und es keine Variable $y \in \mathcal{S}(B)$ gibt, die in β ebenfalls mit c' belegt wird, so ersetzen wir c' in a durch x.

   Wenn eine Variable $y \in \mathcal{S}(B)$ in β durch c'' belegt wird und es keine Variable $x \in \mathcal{S}(A)$ gibt, die in α ebenfalls mit c'' belegt wird, so ersetzen wir c'' in $f_1(a)$ durch y.

   So entstehen Aktionsschemata A' und B', für die gilt:

   i. $B' = B$.

   ii. A' ist eine Substitution von A.

   iii. Aus $pre(a) \Rightarrow pre(b)$ und der Art unserer Ersetzung folgt $pre(A') \Rightarrow pre(B')$.

iv. Aus $\forall i(e_i \in eff(f_1(a)) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$ und der Art unserer Ersetzung folgt $\forall i(E_i \in eff(B) \Rightarrow \exists j(E_j \in eff(A') \wedge E_i = E_j \wedge C_i \Rightarrow C_j))$

v. Falls nach der Ersetzung $x \in \big(\mathcal{S}(A') \cap \mathcal{S}(B)\big)$, so stand in A und B an dieser Stelle dieselbe Konstante, also gilt wegen T2 und T3 auch S3.

b) Es gibt keine Aktion $b \in Actions_\Delta(B)$, für die gilt:

- $a \subseteq_{as} b \wedge f_1(a) = b$ oder $a \subseteq_{as} b \wedge f_2(b) = a$ für ein $a \in A_0$
- in b wurde jede Variable $y \in \mathcal{S}(B)$ durch eine unterschiedliche Konstante belegt.

Dann wählen wir die Aktion a, in deren korrespondierender Aktion b die meisten unterschiedlichen Konstanten vorkommen.

Es gilt, dass es für die Aktionen $b \in Actions_\Delta(B)$, in denen jede Variable $y \in \mathcal{S}(B)$ durch eine andere Konstante belegt wurde (diese Aktionen gibt es wegen T1), eine Aktion $f_2(b) \in Actions_\Delta(A)$ gibt, so dass

i. $pre(f_2(b)) \Rightarrow pre(b)$

ii. $\forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(f_2(b)) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

Es kann nun für eine Variable $x \in \mathcal{S}(A)$, die in α durch c ersetzt wurde, mehrere Variablen $y_i \in \mathcal{S}(B)$ geben, die ebenfalls durch c ersetzt wurden. Aus

i. $pre(a) \Rightarrow pre(f_1(a))$

ii. $\forall i(e_i \in eff(f_1(a)) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

folgt jedoch, dass es mindestens ein y_i gibt, so dass wir genau wie in a) in a und $f_1(a)$ jedes Vorkommen von c durch y_i ersetzen können.

Wenn eine Variable $x \in \mathcal{S}(A)$ in α durch c' belegt wird und es keine Variable $y \in \mathcal{S}(B)$ gibt, die in β ebenfalls mit c' belegt wird, so ersetzen wir c' in a durch x.

Wenn eine Variable $y \in \mathcal{S}(B)$ in β durch c'' belegt wird und es keine Variable $x \in \mathcal{S}(A)$ gibt, die in α ebenfalls mit c'' belegt wird, so ersetzen wir c'' in $f_1(a)$ durch y.

So entstehen Aktionsschemata A' und B', für die dasselbe gilt wie in a).

2. „$\Leftarrow$“
Zunächst ist klar, dass, wenn $\Phi \Rightarrow \Psi$ für zwei FOL-Formeln Φ und Ψ gilt, auch $\Phi' \Rightarrow \Psi'$ gilt, wenn Φ' aus Φ und Ψ' aus Ψ entsteht, indem alle Vorkommen einer freien Variablen durch eine Konstante ersetzt werden (natürlich muss eine freie Variable, die sowohl in Φ als auch in Ψ auftaucht, auch in beiden Formeln durch dieselbe Konstante ersetzt werden). (*)
Es gäbe nun eine Substitution A' von A, so dass S1, S2 und S3 gelten. Wir müssen zeigen, dass für jede Objekt-Domäne Δ vollständige Abbildungen

$$f_1 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

und

$$f_2 : Actions_\Delta(B) \mapsto Actions_\Delta(A)$$

existieren mit:

(a) i. $\forall a \in Actions_\Delta(A) : pre(a) \Rightarrow pre(f_1(a))$
ii. $\forall a \in Actions_\Delta(A) : \forall i(e_i \in eff(f_1(a)) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

(b) i. $\forall b \in Actions_\Delta(B) : pre(f_2(b)) \Rightarrow pre(b)$
ii. $\forall b \in Actions_\Delta(B) : \forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(f_2(b)) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

Dazu betrachten wir eine beliebige Objekt-Domäne $\Delta = \langle \Lambda, \theta \rangle$ sowie eine beliebige Aktion a in $Actions_\Delta(A')$.
Wir betrachten jetzt die Belegung α von $\mathcal{S}(A')$, die a erzeugt hat. Wir erzeugen nun eine Belegung β, für die gilt, dass $\beta(x) = \alpha(x)$, falls $x \in \mathcal{S}(A') \cap \mathcal{S}(B)$ und $\beta(x)$ beliebig, falls $x \notin \mathcal{S}(A') \cap \mathcal{S}(B)$. Die von β erzeugte Belegung ist unser gesuchtes $f_1(a)$. Wegen (*) gilt $a \subseteq_{as} f_1(a)$. Auf analoge Weise erzeugen wir aus einer beliebigen Aktion $b \in Actions_\Delta(B)$ eine passende Aktion $f_2(b) \in Actions_\Delta(A')$.

Da Δ, a und b beliebig gewählt waren, folgt $A' \subseteq_{as} B$ und mit Proposition 2 dann direkt $A \subseteq_{as} B$.

Beispiel 5. Wir betrachten

$$A : \langle P(x) \wedge Q(y), P(y) \rhd E(y), \{x \mapsto T, y \mapsto T\}\rangle$$

und

$$B : \langle P(u) \vee Q(v), P(u) \wedge R(v) \rhd E(u), \{u \mapsto T, v \mapsto T\}\rangle$$

Es gelte $A \subseteq_{as} B$. Wir betrachten jetzt eine ausreichend große Objekt-Domäne Δ:

$$\Delta = \langle \{c_1, c_2\}, \{T \mapsto \{c_1, c_2\}\}\rangle$$

Dann ist $Actions_\Delta(A)$=

$$\begin{aligned}
a_1 &: \langle P(c_1) \wedge Q(c_1), P(c_1) \rhd E(c_1)\rangle \\
a_2 &: \langle P(c_1) \wedge Q(c_2), P(c_2) \rhd E(c_2)\rangle \\
a_3 &: \langle P(c_2) \wedge Q(c_1), P(c_1) \rhd E(c_1)\rangle \\
a_4 &: \langle P(c_2) \wedge Q(c_2), P(c_2) \rhd E(c_2)\rangle
\end{aligned}$$

und $Actions_\Delta(B)$=

$$\begin{aligned}
b_1 &: \langle P(c_1) \vee Q(c_1), P(c_1) \wedge R(c_1) \rhd E(c_1)\rangle \\
b_2 &: \langle P(c_1) \vee Q(c_2), P(c_1) \wedge R(c_2) \rhd E(c_1)\rangle \\
b_3 &: \langle P(c_2) \vee Q(c_1), P(c_2) \wedge R(c_1) \rhd E(c_2)\rangle \\
b_4 &: \langle P(c_2) \vee Q(c_2), P(c_2) \wedge R(c_2) \rhd E(c_2)\rangle
\end{aligned}$$

Da nun $A \subseteq_{as} B$ gilt, gibt es für Δ vollständige Abbildungen

$$f_1 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

und

$$f_2 : Actions_\Delta(A) \mapsto Actions_\Delta(B)$$

mit:

1. (a) $\forall a \in Actions_\Delta(A) : pre(a) \Rightarrow pre(f_1(a))$
   (b) $\forall a \in Actions_\Delta(A) : \forall i(e_i \in eff(f_1(a)) \Rightarrow \exists j(e_j \in eff(a) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$
2. (a) $\forall b \in Actions_\Delta(B) : pre(f_2(b)) \Rightarrow pre(b)$

(b) $\forall b \in Actions_\Delta(B) : \forall i(e_i \in eff(b) \Rightarrow \exists j(e_j \in eff(f_2(b)) \wedge e_i = e_j \wedge c_i \Rightarrow c_j))$

Dies gilt zum Beispiel für

$$f_1 = \{a_1 \mapsto b_1, a_2 \mapsto b_2, a_3 \mapsto b_3, a_4 \mapsto b_4\}$$

und

$$f_2 = f_1^-.$$

Wir betrachten nun eine Aktion $a \in Actions_\Delta(A)$, für die gilt, dass jeder Parameter durch ein unterschiedliches Objekt belegt wurde. Dies ist zum Beispiel der Fall für a_2. Da in a_2 der Parameter x von A durch dieselbe Konstante wie in $f_1(a_2)$ der Parameter u von B ersetzt wurde (nämlich durch c_1), ersetzen wir c_1 in a_2 und c_1 in $f(a_2)$ durch u. Genauso ersetzen wir c_2 in a_2 und c_2 in $f(a_2)$ durch v und erhalten so zwei Aktionsschemata

$$A' : \langle P(u) \wedge Q(v), P(v) \rhd E(v), \{u \mapsto T, v \mapsto T\}\rangle$$

und

$$B' : \langle P(u) \vee Q(v), P(u) \wedge R(v) \rhd E(u), \{u \mapsto T, v \mapsto T\}\rangle$$

für die gilt:

1. A' ist eine Substitution von A.
2. $B' = B$
3. S1-S3.

Wir können Theorem 1 auch relativ einfach um eine Prädikaten-Inklusions-Hierarchie erweitern:

Theorem 2 ***(Syntaktisches Kriterium für $A \subseteq_{as} B$ bezüglich einer Prädikaten-Inklusions-Hierarchie).*** *Es gilt $A \subseteq_{as}^{\mathbf{P}} B$ für zwei Aktionsschemata A und B bezüglich einer Typhierachie $\boldsymbol{T}$ und einer Prädikaten-Inklusions-Hierarchie $\boldsymbol{P}$ genau dann, wenn es eine Substitution A^s von A gibt, so dass*

(S1) $pre(A^s) \wedge \bigwedge_{p \in \mathbf{P}} p \Rightarrow pre(B)$

(S2) $\forall i(E_i \in eff(B) \Rightarrow \exists j(E_j \in eff(A^s) \wedge (E_j \wedge \bigwedge_{p \in \mathbf{P}} p \Rightarrow E_i) \wedge (C_i \wedge \bigwedge_{p \in \mathbf{P}} p \Rightarrow C_j)))$

(S3) $types_{A^s}(x) \subseteq types_B(x)$ *folgt aus* $\boldsymbol{T}$, *falls* $x \in \big(\mathcal{S}(A^s) \cap \mathcal{S}(B)\big)$

Der Beweis von Theorem 2 funktioniert analog zu Beweis 5, wobei an den entsprechenden Stellen $\mathbf{P}$ berücksichtigt wird.

4.4 Modellierung in OWL

Für die Modellierung einer Abstraktionshierarchie von Aktionsschemata bietet sich ein auf Beschreibungslogiken [Baader u.a., 2003] basierendes System an, in dem auf natürliche und übersichtliche Weise Ontologien beschrieben werden können. Besonders interessant erscheint hierbei der W3C-Standard *OWL* [McGuinness und van Harmelen, 2004]. Obwohl speziell für das Web entworfen, kann diese Sprache auch für die Pflege einer Aktionsdomäne eine zentrale Rolle übernehmen. Hierfür lassen sich im Wesentlichen drei Gründe nennen:

Zunächst kann neben der reinen Organisation der Objektklassenbeschreibung durch den Einsatz von Werkzeugen zum Erstellen und Pflegen von Ontologien (z.B. Protege) der Prozess der Erstellung von Objektmodellen signifikant unterstützt werden. Desweiteren existieren einige sehr gut funktionierende Reasoningsysteme (z.B. Fact++, Pellet oder Racer) für OWL, mit dessen Hilfe aus den vorhandenen expliziten Domäneninformationen zusätzliche, für den weiteren Prozess möglicherweise sehr hilfreiche hierarchische Strukturen der Domäne gewonnen werden können. Und nicht zu Letzt gibt es sehr ausgereifte und intensiv getestete API's für Programmiersprachen wie Java (z.B. Jena-API), mit deren Hilfe die Implementation einer für den Planungsprozess notwendigen Übersetzung nach PDDL wesentlich vereinfacht werden kann und somit weniger fehleranfällig ist.

Da die Taxonomieklassifizierung von OWL $\mathcal{NEXPTIME}$-vollständig und AS_{BOOL} Σ_2^p-vollständig ist (siehe Kapitel 5), erwartet man, dass eine solche Modellierung möglich sein sollte. Wir suchen jedoch nicht irgendeine Modellierung, sondern eine „natürliche“, die über weitere Eigenschaften verfügen sollte:

Eine „natürliche“ Modellierung in OWL sollte jedes Aktionsschemata als ein eigenes Konzept modellieren. Wir bezeichnen im Folgenden ein Aktionsschema mit einem kursiven Großbuchstaben (A,B,C,...) und das modellierende Konzept dazu mit demselben Buchstaben in Schreibmaschinen-Schrift (`A`,`B`,`C`,...).

Um die Lösung des Subsumptionsproblems zweier OWL-Konzepte dazu nutzen zu können, das Subsumptionsproblem zweier Aktionsschemata zu lösen, sollte die Modellierung auf jeden Fall folgende Eigenschaft haben:

(M1) $A_1 \subseteq_{as} A_2$ genau dann, wenn $\mathtt{A}_1 \sqsubseteq \mathtt{A}_2$.

Außerdem sollte ein Zusammenhang zwischen den von einem Aktionsschema A repräsentierten Aktionen und den Instanzen von A bestehen. Dazu fordern wir, dass in jeder Objekt-Domäne Δ die folgende Eigenschaft gilt:

(M2) Es gibt eine bijektive Abbildung zwischen den Aktionen $a \in \mathit{Actions}_\Delta(A)$ und den möglichen Instanzen a' von A.

Aktionsschemata lassen sich jedoch nicht als OWL-Konzepte definieren, wenn man M1 und M2 gleichzeitig fordert. Betrachte hierzu zwei Aktionsschemata A und B, für die gilt:

i) $A \subseteq_{as} B$ und

ii) es gibt eine Objekt-Domäne Δ, so dass $|\mathit{Actions}_\Delta(A)| > |\mathit{Actions}_\Delta(B)|$ (dies ist zum Beispiel erfüllt für $A = \langle P(x) \wedge P(y), (), \{x \mapsto \textsc{Thing}, y \mapsto \textsc{Thing}\}\rangle$ und $B = \langle P(x), (), \{x \mapsto \textsc{Thing}\}\rangle$

Sei $\#\mathit{inst}(\mathtt{X})$ die Anzahl der möglichen Instanzen des Konzeptes X.

Da $A \subseteq_{as} B$ gilt, soll wegen M1 auch $\mathtt{A} \sqsubseteq \mathtt{B}$ gelten. Wenn aber $\mathtt{A} \sqsubseteq \mathtt{B}$ gilt, so muss $\#\mathit{inst}(\mathtt{A}) \leq \#\mathit{inst}(\mathtt{B})$ sein(*), denn jede Instanz von A muss dann natürlich auch eine Instanz von B sein. Gleichzeitig soll aber wegen M2 auch eine bijektive Abbildung zwischen den Aktionen $a \in \mathit{Actions}_\Delta(A)$ und den Instanzen a' von A sowie den Aktionen $b \in \mathit{Actions}_\Delta(B)$ und den Instanzen b' von B existieren. Deshalb muss $\#\mathit{inst}(\mathtt{A}) = |\mathit{Actions}_\Delta(A)|$ und $\#\mathit{inst}(\mathtt{B}) = |\mathit{Actions}_\Delta(B)|$ sein. Damit wäre aber dann $\#\mathit{inst}(\mathtt{A}) > \#\mathit{inst}(\mathtt{B})$, was ein Widerspruch zu (*) ist.

Wir können also keine bijektive Abbildung fordern, wenn M1 gilt, sondern müssen hinnehmen, dass mehrere Aktionen $a \in \mathit{Actions}_\Delta(A)$ auf dieselbe OWL-Instanz von A abgebildet werden. Außerdem muss, wenn $A \subseteq_{as} B$ gilt, für jede Aktion $a \in \mathit{Actions}_\Delta(A)$ auch $a \in \mathit{Actions}_\Delta(B)$ gelten, da ja nach M1 dann $\mathtt{A} \sqsubseteq \mathtt{B}$ gilt und deshalb jede Instanz von A auch eine Instanz von B sein muss.

M1 ist eine grundlegende Eigenschaft, die wir von einer natürlichen Modellierung erwarten, da hier Aktionsschemata so mit OWL-Konzepten in Verbindung

gebracht werden, dass das Subsumptionsproblem der Konzepte dem der Aktionsschemata entspricht. M2 dagegen ist relativ stark, weshalb wir es an dieser Stelle statt dessen mit den folgenden Eigenschaften versuchen:

(M3) Zu jeder Aktion $a \in \mathit{Actions}_\Delta(A)$ gibt es eine mögliche Instanz a' von $\mathtt{A}$, so dass es zwischen a' und allen Objekten $c_i \in \Delta$, die Parameter von a sind, eine funktionale Rolle r_i gibt. Zwischen a' und den c_i existieren Kombinationen von Rollenketten und Konzepten, die die Vorbedingung und die Effekte von a modellieren.

Zu jeder Instanz a' von $\mathtt{A}$ gibt es eine Aktion a_1, für die es eine Aktion $a \in \mathit{Actions}_\Delta(A)$ gibt, so dass alle Objekte c_i, zu denen es eine funktionale Rolle zwischen a' und $c_i \in \Delta$ gibt, Parameter von a_1 sind, durch alle anderen Rollenketten und Konzepte zwischen a' und den c_i genau die Vorbedingung und die Effekte von a_1 modelliert werden und $a_1 \subseteq_{as} a$ gilt.

Die funktionalen Rollen zwischen den Instanzen a' von $\mathtt{A}$ und den beteiligten Objekten $c_i \in \Delta$ sollten auf eine systematische Art und Weise benannt werden und die Zuordnung zwischen der Wahl der Benennung und den Parametern der korrespondierenden Aktion $a \in \mathit{Actions}_\Delta(A)$ sollte eindeutig sein. Am Naheliegendsten ist folgende Zuordnung:

(M4) Eine funktionale Rolle zwischen einer Instanz a' von $\mathtt{A}$ und einem beteiligten Objekt $c_i \in \Delta$ wird eineindeutig nach dem Parameternamen des Parameters $x \in \mathcal{S}(A)$ benannt, der in a durch c_i ersetzt wurde.

Es ist jedoch nicht möglich, M1, M3 und M4 gleichzeitig zu fordern, wie Beispiel 6 zeigt:

Beispiel 6. Wir betrachten

$$A = \langle P(x) \wedge \neg Q(y), (), \{x \mapsto \textsc{Thing}, y \mapsto \textsc{Thing}\}\rangle$$

und

$$B = \langle P(y) \wedge \neg Q(x), (), \{x \mapsto \textsc{Thing}, y \mapsto \textsc{Thing}\}\rangle$$

Man sieht sofort, dass $A \subseteq_{as} B$ und $B \subseteq_{as} A$ gilt, da $Actions_\Delta(A) = Actions_\Delta(B)$ für jede Objekt-Domäne Δ gilt.

Wir betrachten nun die Objekt-Domäne $\Delta = \langle\{c_1, c_2\}, \emptyset\rangle$. Hier gibt es zu der Aktion $a = \langle P(c_1) \wedge \neg Q(c_2), ()\rangle$ wegen M3 eine mögliche Instanz a' von `A`, so dass es eine funktionale Rollenkette r_1 zwischen a' und c_1 und eine funktionale Rollenkette r_2 zwischen a' und c_2 gibt und durch zusätzliche Rollenketten und Konzepte zwischen a', c_1 und c_2 modelliert wird, dass die Vorbedingung von a $P(c_1) \wedge \neg Q(c_2)$ lautet. Genauso gibt es auch eine Instanz b' von `B`, so dass es eine funktionale Rollenkette r_3 zwischen a' und c_1 und eine funktionale Rollenkette r_4 zwischen a' und c_2 gibt und durch zusätzliche Rollenketten und Konzepte zwischen a', c_1 und c_2 modelliert wird, dass die Vorbedingung von a $P(c_1) \wedge \neg Q(c_2)$ lautet.

Wegen M4 müssen wir jedoch in a' r_1 nach x und r_2 nach y und in b' r_3 nach y und r_4 nach x benennen, so dass a' keine Instanz von `B` und b' keine Instanz von `A` ist.

Somit gilt weder $\texttt{A} \sqsubseteq \texttt{B}$ noch $\texttt{B} \sqsubseteq \texttt{A}$.

Warum funktioniert nun eine „natürliche“ Modellierung des Subsumptionsproblems in OWL nicht? Intuitiv gesehen liegt dies daran, dass wir, um zu testen, ob $A \subseteq_{as} B$ gilt, alle zulässigen Parameter-Substitutionen s auf A anwenden und dann S1-S3 für A^s und B testen müssen. Dies müsste man in einer Modellierung dann natürlich auch auf OWL-Ebene tun, was nicht möglich ist, wenn man M1 und M2 beziehungsweise M1, M3 und M4 fordert.

Weitere Abschwächungen von M3 oder M4 würden jedoch die eingangs geforderte Natürlichkeits-Eigenschaft nicht mehr einhalten und sind somit nicht interessant für uns.

Kapitel 5

Komplexität des Subsumptions-Entscheidungsproblems

Das Problem, zu entscheiden, ob bei gegebener Typhierachie $\mathbf{T}$ und gegebener Prädikaten-Inklusions-Hierachie $\mathbf{P}$ für zwei Aktionsschemata A und B gilt, dass $A \subseteq_{as} B$, nennen wir im folgenden AS. Wir untersuchen die drei Fälle, in denen A und B in $\mathcal{A}_{STRIPS}$, $\mathcal{A}_{BOOL}$ bzw. $\mathcal{A}_{ADL}$ (siehe Definition 6) liegen und nennen diese Teilprobleme AS_{STRIPS}, AS_{BOOL} bzw. AS_{ADL}.

5.1 Komplexität von AS_{STRIPS}

Theorem 3 *(**Komplexität von** $\boldsymbol{AS}_{STRIPS}$).* AS_{STRIPS} *ist* $\mathcal{NP}$*-vollständig.*

Wir beweisen $\mathsf{AS}_{STRIPS} \in \mathcal{NP}$ durch die Angabe eines polynomiellen Verifikationsalgorithmus und die $\mathcal{NP}$-Härte von AS_{STRIPS} durch eine polynomielle Abbildungsreduktion des als $\mathcal{NP}$-vollständig bekannten Problems der Subgraph-Isomorphie (SI) auf AS_{STRIPS}.

Dazu definieren wir zunächst SI:

Definition 15 *(**Subgraph-Isomorphie**).* SI *ist das Problem, zu entscheiden, ob für zwei gegebene ungerichtete Graphen* $G_1 = (V_1, E_1)$ *und* $G_2 = (V_2, E_2)$

G_1 isomorph in G_2 eingebettet werden kann. Isomorph einbetten bedeutet hierbei, dass ein Graph $G_2' = (V_2', E_2')$ mit $V_2' \subseteq V_2$, $E_2' = E_2 \cap \{(u,v)|u,v \in V_2\}$ und eine bijektive Abbildung $f : V_1 \mapsto V_2'$ existieren, so dass gilt: $(u,v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2'$.

Beweis 6 *(Theorem 3).*

1. $\mathsf{AS}_{STRIPS} \in \mathcal{NP}$

   Wir haben eine Typhierarchie $\mathbf{T}$, eine Prädikaten-Inklusions-Hierarchie $\mathbf{P}$, sowie zwei Aktionsschemata $A, B \in \mathcal{A}_{STRIPS}$ gegeben und wollen entscheiden, ob $A \subseteq_{as} B$ gilt.

   Dazu „raten“ wir eine gültige Substitution A' von A (hiervon gibt es im Worst-Case exponentiell in der Anzahl der Parameter viele verschiedene) und verifizieren, ob $A' \subseteq_{as} B$ gilt. Nach Theorem 1 genügt es hierzu, S1-S3 für A' und B zu zeigen. Da A und B (und damit auch A') nach Vorraussetzung aus $\mathcal{A}_{STRIPS}$ sind, genügt es für S1, zu zeigen, dass für jedes Prädikat $P(\vec{x})$ aus $pre(A)$ ein Prädikat $P'(\vec{x'})$ in $pre(B)$ existiert, wobei $P = P'$ oder P in $\mathbf{P}$ aus P' folgt und $types_B(x_i') \subseteq types_A(x_i)$ für jedes Argument $x_i' \in \mathcal{S}(B)$, das in P' vorkommt. Dies ist in polynomieller Zeit möglich:

   Sei k die Anzahl der Prädikaten-Inklusions-Axiome, l die Anzahl der Regeln in $\mathbf{T}$, m die Anzahl der Prädikate in $pre(A')$, n die Anzahl der Prädikate in $pre(B)$ und o das Maximum der Anzahlen von Argumenten aller Prädikate. Wir testen für ein Prädikat $P'(\vec{x}) \in pre(B)$, welche Prädikate für P' aus $\mathbf{P}$ folgen (k^2 Vergleiche) und vergleichen jedes Prädikat $P(\vec{x'}) \in pre(A)$ mit diesen Prädikaten ($k^2 * m$ Vergleiche). Gilt $P = P'$ oder folgt P aus P', so testen wir, ob $types(x_i') \subseteq types(x_i)$ für alle i ($o * l^2$ Vergleiche). Gilt dies, so markieren wir $P(\vec{x})$.

   Dies machen wir für jedes Prädikat $P'(\vec{x})$ aus $pre(B)$, insgesamt also n mal. Ist am Ende jedes Prädikat in $pre(A)$ markiert, so gilt $pre(A') \Rightarrow pre(B)$, ansonsten nicht. Somit haben wir insgesamt $n * (k^2 + k^2 * m + o * l^2)$ viele Vergleiche. Dies sind offensichtlich polynomiell viele in der Größe von A, B, $\mathbf{T}$ und $\mathbf{P}$.

Da in $\mathcal{A}_{STRIPS}$ keine konditionalen Effekte zugelassen sind (bzw. alle Konditionen $\top$ sein müssen), vereinfacht sich S2 zu $\forall i(E_i \in eff(B) \Rightarrow \exists j(E_j \in eff(A') \wedge E_j \Rightarrow E_i))$. Für die Effekte ist also dasselbe zu tun, wie für die Vorbedingungen, somit ist dies natürlich ebenfalls in polynomieller Zeit möglich.

S3 lässt sich offensichtlich für jeden Parameter in Zeit polynomiell zur Größe der Typhierachie überprüfen.

Somit gilt $\mathsf{AS}_{STRIPS} \in \mathcal{NP}$.

2. $\mathcal{NP}$-Härte von AS_{STRIPS}

   Wir geben eine polynomielle Abbildungsreduktion von SI auf AS_{STRIPS} an:

   O.B.d.A. nennen wir die Knoten in G_1 x_i für $1 \leq i \leq |V_1|$ und die Knoten in G_2 y_j für $1 \leq j \leq |V_2|$.

   Wir bilden nun G_1 auf ein Aktionsschema $A_1 \in \mathcal{A}_{STRIPS}$ wie folgt ab:

   (a) Für jeden Knoten $x_i \in V_1$ führen wir einen gleichnamigen Parameter vom Typ THING ein.

   (b) Für jede Kante $(x_i, x_j) \in E_1$ fügen wir $pre(A_1)$ die zwei Prädikate $E^+(x_i, x_j)$ und $E^+(x_j, x_i)$ hinzu.

   (c) Für jedes Knotenpaar x_i, x_j, für das $(x_i, x_j) \notin E_1$, für das also keine Kante in E_1 existiert, fügen wir $pre(A_1)$ die zwei Prädikate $E^-(x_i, x_j)$ und $E^-(x_j, x_i)$ hinzu.

   (d) A_1 hat keine Effekte.

   Analog erzeugen wir ein Aktionsschema $A_2 \in \mathcal{A}_{STRIPS}$ für G_2.

   Wir zeigen jetzt: G_1 lässt sich isomorph in G_2 einbetten genau dann, wenn $A_2 \subseteq_{as} A_1$.

   Betrachten wir zunächst den Fall, in dem sich G_1 isomorph in G_2 einbetten lässt: Dann gibt es nach Definition einen Graphen $G_2' = (V_2', E_2')$ mit $V_2' \subseteq V_2$, $E_2' = E_2 \cap \{(u, v) | u, v \in V_2\}$ und eine bijektive Abbildung $f : V_1 \mapsto V_2'$, so dass gilt: $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2'$. Außerdem gibt

es aufgrund unserer Abbildung für jede Kante $(x_i, x_j) \in E_1$ genau zwei Prädikate $E^+(x_i, x_j)$ und $E^+(x_j, x_i)$ in $pre(A_1)$ und für jedes Knotenpaar x_i, x_j, für das keine Kante in E_1 vorhanden ist, genau zwei Prädikate $E^-(x_i, x_j)$ und $E^-(x_j, x_i)$ in $pre(A_1)$; für E_2 und $pre(A_2)$ gilt analoges. Substituieren wir also $f(x_i)$ durch x_i, so kommt jedes $E^+(x_i, x_j) \in pre(A_1)$ und jedes $E^-(x_i, x_j) \in pre(A_1)$ auch in $pre(A_2)$ vor. Da $A_1, A_2 \in \mathcal{A}_{STRIPS}$ und es keine Effekte und keine Typen gibt (S2 und S3 also trivialerweise gelten), folgt hieraus direkt $A_2 \subseteq_{as} A_1$.

Jetzt betrachten wir den Fall, in dem sich G_1 nicht isomorph in G_2 einbetten lässt. Dann kann nach Definition kein Graph $G_2' = (V_2', E_2')$ existieren mit $V_2' \subseteq V_2$, $E_2' = E_2 \cap \{(u, v) | u, v \in V_2\}$, so dass eine bijektive Abbildung $f : V_1 \mapsto V_2'$ existiert, so dass gilt: $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2'$. Also gibt es für jeden Subgraphen G_2' von G_2 und jede Abbildung $f : V_1 \mapsto V_2'$ eine Kante $(x_i, x_j) \in E_1$, so dass $(f(x_i), f(x_j)) \notin E_2$ oder es fehlt eine Kante $(x_i, x_j) \in E_1$, so dass $(f(x_i), f(x_j)) \in E_2$. Unsere Abbildung erzeugt also entweder ein Prädikat $E^+(x_i, x_j) \in pre(A_1)$, so dass $E^+(f(x_i), f(x_j)) \notin pre(A_2)$ oder ein Prädikat $E^-(x_i, x_j) \in pre(A_1)$, so dass $E^-(f(x_i), f(x_j)) \notin pre(A_2)$. Also kann man $f(x_i)$ nicht durch x_i substituieren (denn dann gäbe es ein Prädikat in $pre(A_1)$, das es nicht in $pre(A_2)$ gibt). Also existiert keine Substitution, so dass alle Prädikate aus $pre(A_1)$ auch in $pre(A_2)$ vorkommen. Da $A_1, A_2 \in \mathcal{A}_{STRIPS}$ und es keine Effekte und keine Typen gibt (S2 und S3 also trivialerweise gelten), folgt hieraus direkt $A_2 \not\subseteq_{as} A_1$.

Da die Abbildung zudem polynomiell in der Größe der Kodierung der Graphen ist, haben wir tatsächlich eine polynomielle Abbildungsreduktion angegeben und somit die $\mathcal{NP}$-Härte von AS_{STRIPS} gezeigt.

Aus $\mathsf{AS}_{STRIPS} \in \mathcal{NP}$ und der $\mathcal{NP}$-Härte von AS_{STRIPS} folgt die $\mathcal{NP}$-Vollständigkeit von AS_{STRIPS}.

Beispiel 7. Wir betrachten die beiden Graphen G_1 und G_2 aus Abbildung 5.1. Offensichtlich existiert eine Isomorphie zwischen G_1 und einem Subgraphen von G_2 (genau genommen sogar mehrere); es ist also $(G_1, G_2) \in \mathsf{SI}$. Um dies zu

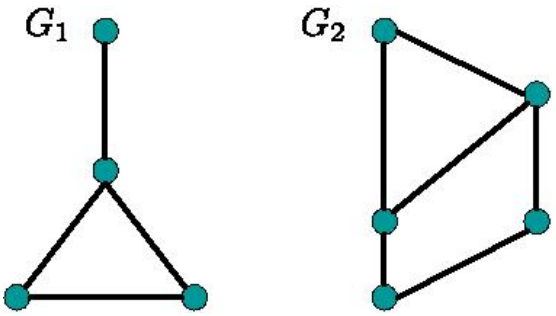

Abbildung 5.1: Eingabe (G_1,G_2) von SI

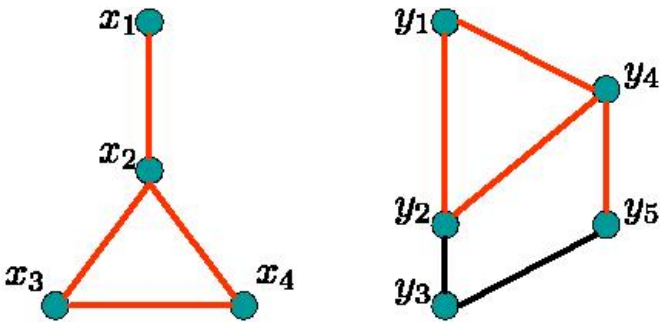

Abbildung 5.2: G_1 ist isomorph zu einem Subgraphen von G_2.

zeigen, wählen wir z.B. $G_2' = \{\{y_1, y_2, y_4, y_5\}, \{(y_1, y_2), (y_1, y_4), (y_2, y_4), (y_4, y_5)\}\}$ und $f = \{x_1 \mapsto y_5; x_2 \mapsto y_4, x_3 \mapsto y_1, x_4 \mapsto y_2\}$. Dann existiert eine Kante in G_1 zwischen x_i und x_j genau dann, wenn eine Kante in G_2' zwischen $f(x_i)$ und $f(x_j)$ existiert.

Wenden wir die oben beschriebene Abbildung auf G_1 und G_2 an, so erhalten wir die beiden Aktionsschemata A_1 und A_2:

$$\begin{aligned} A_1 = \langle \quad & E^+(x_1, x_2) \wedge E^+(x_2, x_1) \wedge E^+(x_2, x_3) \wedge E^+(x_3, x_2) \wedge \\ & E^+(x_2, x_4) \wedge E^+(x_4, x_2) \wedge E^+(x_3, x_4) \wedge E^+(x_4, x_3) \wedge \\ & E^-(x_1, x_3) \wedge E^-(x_3, x_1) \wedge E^-(x_1, x_4) \wedge E^-(x_4, x_1), \\ & (), \\ & \{x_1 \mapsto \texttt{THING}, x_2 \mapsto \texttt{THING}, x_3 \mapsto \texttt{THING}, x_4 \mapsto \texttt{THING}\}\rangle \end{aligned}$$

$$\begin{aligned} A_2 = \langle \quad & E^+(y_1, y_2) \wedge E^+(y_2, y_1) \wedge E^+(y_2, y_3) \wedge E^+(y_3, y_2) \wedge \\ & E^+(y_2, y_4) \wedge E^+(y_4, y_2) \wedge E^+(y_4, y_5) \wedge E^+(y_5, y_4) \wedge \\ & E^+(y_1, y_4) \wedge E^+(y_4, y_1) \wedge E^+(y_3, y_5) \wedge E^+(y_5, y_3) \wedge \\ & E^-(y_1, y_3) \wedge E^-(y_3, y_1) \wedge E^-(y_1, y_5) \wedge E^-(y_5, y_1) \wedge \\ & E^-(y_2, y_5) \wedge E^-(y_5, y_2) \wedge E^-(y_3, y_4) \wedge E^-(y_4, y_3), \\ & (), \\ & \{y_1 \mapsto \texttt{THING}, y_2 \mapsto \texttt{THING}, y_3 \mapsto \texttt{THING}, y_4 \mapsto \texttt{THING}, y_5 \mapsto \texttt{THING}\}\rangle \end{aligned}$$

Wie sich leicht verifizieren lässt, gilt $A_2[\frac{x_3}{y_1}\frac{x_4}{y_2}\frac{x_2}{y_4}\frac{x_1}{y_5}] \subseteq_{as} A_1$ und damit nach Proposition 1 auch $A_2 \subseteq_{as} A_1$.

5.2 Komplexität von AS_{BOOL}

Theorem 4 ***(Komplexität von AS_{BOOL}).*** *AS_{BOOL} ist Σ_2^p-vollständig.*

Wir beweisen $\mathsf{AS}_{BOOL} \in \Sigma_2^p$ durch die Angabe eines nichtdeterministischen polynomiellen Verifikationsalgorithmus, der ein propositionales SAT-Orakel verwendet, und die Σ_2^p-Härte von AS_{BOOL} durch eine polynomielle Abbildungsreduktion des als Σ_2^p-vollständig bekannten Problems 2-QBF auf AS_{BOOL} (siehe Kapitel 3.3.2).

Zunächst noch drei dazu notwendige Propositionen:

Proposition 2. *Die Erfüllbarkeit einer FOL-Formel ϕ ohne Quantoren und ohne funktionale Terme kann durch einen propositionalen SAT-Solver entschieden werden.*

Beweis 7 *(Proposition 2).* Da wir keine Quantoren haben, können wir jede Kombination von Prädikatenname und Argumenten als eine atomare Aussage betrachten. Dazu führen wir eine eindeutige Variable für jede solche Kombination ein und ersetzen in ϕ diese Kombination durch die neue Variable. Dazu kann man beispielsweise für ein Prädikat $P(x_1, x_2, c)$ eine Variable mit dem Namen $P\_x_1\_x_2\_c$ einführen und in ϕ jedes Auftreten von $P(x_1, x_2, c)$ durch $P\_x_1\_x_2\_c$ ersetzen (dies ist in polynomieller Zeit möglich). Haben wir alle Prädikate ersetzt, so erhalten wir eine propositionale Formel ϕ' und es gilt, dass ϕ' erfüllbar ist genau dann, wenn ϕ erfüllbar ist.

Aus der mathematischen Logik ist folgende Proposition bekannt (siehe auch Beispiel 8):

Proposition 3. *Eine propositionale Formel ϕ' kann in polynomieller Zeit in eine erfüllbarkeitsäquivalente propositionale Formel ϕ'' in konjunktiver Normalform umgewandelt werden.*

Wir verzichten an dieser Stelle auf einen formalen Beweis und zeigen statt dessen anhand eines Beispiels, wie diese Umformung funktioniert:

Beispiel 8. Betrachte $\phi' = (\neg X \vee Y) \vee ((X \wedge U) \vee Z)$. Wir führen für jede nichtatomare Subformel β_i eine neue Variable X_{β_i} ein:

$$\phi' = \underbrace{\underbrace{(\underbrace{\neg X}_{\beta_1} \vee Y)}_{\beta_2} \vee \underbrace{(\underbrace{(X \wedge U)}_{\beta_3} \vee Z)}_{\beta_4}}_{\beta_5}$$

Hiermit bilden wir nun

$$\begin{aligned}\phi''' = \quad & (X_{\beta_1} \Leftrightarrow \neg X) \wedge && (\text{da } \beta_1 = \neg X)\\ & (X_{\beta_2} \Leftrightarrow X_{\beta_1} \vee Y) \wedge && (\text{da } \beta_2 = \beta_1 \vee Y)\\ & (X_{\beta_3} \Leftrightarrow X \wedge U) \wedge && (\text{da } \beta_3 = X \wedge U)\\ & (X_{\beta_4} \Leftrightarrow X_{\beta_3} \vee Z) \wedge && (\text{da } \beta_4 = \beta_3 \vee Z)\\ & (X_{\beta_5} \Leftrightarrow X_{\beta_2} \vee X_{\beta_4}) \wedge && (\text{da } \beta_5 = \beta_2 \vee \beta_4)\\ & X_{\beta_5}\end{aligned}$$

ϕ''' besteht aus einer Konjunktion von linear in der Anzahl der nichtatomaren Subformeln von ϕ' vielen Äquivalenzen mit höchstens drei Variablen, die alle von der Form 5.1, 5.2 oder 5.3 sind:

$$(X \Leftrightarrow \neg Y) \tag{5.1}$$

$$(X \Leftrightarrow (Y \wedge Z)) \tag{5.2}$$

$$(X \Leftrightarrow (Y \vee Z)) \tag{5.3}$$

Diese Äquivalenzen können wir nun in polynomieller Zeit in konjunktive Normalform umformen:

5.1: $(X \Leftrightarrow \neg Y) \equiv (X \vee \neg Y) \wedge (\neg X \vee Y)$

5.2: $(X \Leftrightarrow (Y \wedge Z)) \equiv (X \vee \neg Y \vee \neg Z) \wedge (\neg X \vee \neg Y) \wedge (\neg X \vee Z)$

5.2: $(X \Leftrightarrow (Y \vee Z)) \equiv (\neg X \vee Y \vee Z) \wedge (X \vee \neg Y) \wedge (X \vee \neg Z)$

Somit erhalten wir schließlich

$$\begin{aligned}\phi'' = \quad & (X_{\beta_1} \vee X) \wedge (\neg X_{\beta_1} \vee \neg X) \wedge\\ & (\neg X_{\beta_2} \vee X_{\beta_1} \vee Y) \wedge (X_{\beta_2} \vee \neg X_{\beta_1}) \wedge (X_{\beta_2} \vee \neg Y) \wedge\\ & (X_{\beta_3} \vee \neg X \vee \neg U) \wedge (\neg X_{\beta_3} \vee X) \wedge (\neg X_{\beta_3} \vee U) \wedge\\ & (\neg X_{\beta_4} \vee X_{\beta_3} \vee Z) \wedge (X_{\beta_4} \vee \neg X_{\beta_3}) \wedge (X_{\beta_4} \vee \neg Z) \wedge\\ & (\neg X_{\beta_5} \vee X_{\beta_2} \vee X_{\beta_4}) \wedge (X_{\beta_5} \vee \neg X_{\beta_2}) \wedge (X_{\beta_5} \vee \neg X_{\beta_4}) \wedge\\ & X_{\beta_5}\end{aligned}$$

Proposition 4. *Sei $\Psi \equiv \forall y_1, \ldots, y_m \Phi(\vec{y})$ eine* 2-QBF*-Formel ohne existenzquantifizierte Variablen. Dann ist die Formel $\Psi' \equiv \exists x \forall y_1, \ldots, y_m (x \wedge \Phi(\vec{y}))$ dazu äquivalent.*

Beweis 8 *(Proposition 4).* Wie in Kapitel 3.1.4 definiert, gilt

$$\exists x \forall y_1, \ldots, y_m(x \wedge \Phi(\vec{y})) \equiv \underbrace{(\forall y_1, \ldots, y_m(1 \wedge \Phi(\vec{y})))}_{\alpha} \vee \underbrace{(\forall y_1, \ldots, y_m(0 \wedge \Phi(\vec{y})))}_{\beta}$$

Offensichtlich ist β unerfüllbar und α genau dann erfüllt, wenn $\Phi(\vec{y})$ erfüllt ist. Somit folgt die Behauptung.

Mit den Propositionen 2, 3 und 4 können wir nun Theorem 4 beweisen:

Beweis 9 *(Theorem 4).*

1. $\mathsf{AS}_{BOOL} \in \Sigma_2^p$

   Wir haben eine Typhierachie $\mathbf{T}$, eine Prädikaten-Inklusions-Hierachie $\mathbf{P}$, sowie zwei Aktionsschemata A und B gegeben und wollen entscheiden, ob $A \subseteq_{as} B$ gilt.

   Sei t die Anzahl der Regeln in $\mathbf{T}$, p die Anzahl der Axiome in $\mathbf{P}$, a das Maximum der beiden Anzahlen an Aktions-Parametern und u die Anzahl unterschiedlicher Prädikate (wobei ein Prädikat $P(\vec{x})$ unterschiedlich zu einem Prädikat $P'(\vec{x'})$ ist, wenn $P \neq P'$ oder $x_i \neq x'_i$ für ein i).

   Wir geben einen nichtdeterministischen Algorithmus an, der ein propositionales SAT-Orakel verwendet:

   1) Wähle eine Substitution A' von A und untersuche im Folgenden $A' \subseteq_{as} B$ (dies ist der nichtdeterministische Teil).
   2) Verifiziere S1.

      Dazu formen wir $pre(A)$ und $pre(B)$ wie in Beweis 7 in zwei propositionale Formeln ϕ_A und ϕ_B um. Als Nächstes müssen wir $\mathbf{P}$ für jedes unterschiedliche Prädikat „instanzieren“ (haben wir beispielsweise eine Regel $\forall \vec{x}(P(\vec{x}) \Rightarrow Q(\vec{x}))$ und zwei Prädikate $P(x_1, c_3)$ und $P(c_2, x_4)$, so führen wir die zwei propositionalen Formeln $(\neg P\_x_1\_c_3 \vee Q\_x_1\_c_3)$ und $(\neg P\_c_2\_x_4 \vee Q\_c_2\_x_4)$ ein).

      Die entstehende Prädikaten-Inklusions-Hierarchie $\mathbf{P'}$ hat die Größe $u*p$, was polynomiell in der Größe von $\mathbf{P}$ und der Anzahl unterschiedlicher Prädikate ist. Diese „Instanzierungen“ bilden eine Konjunktion $\phi_{\mathbf{P}}$.

Dann wandeln wir ϕ_A, ϕ_B und $\phi_{\mathbf{P}}$ wie in Beispiel 8 in drei propositionale Formeln ϕ'_A, ϕ'_B und $\phi'_{\mathbf{P}}$ in konjunktiver Normalform um und verifizieren mittels eines SAT-Orakels, ob $\phi'_A \wedge \neg\phi_B \wedge \phi'_{\mathbf{P}}$ unerfüllbar ist. Falls nein, geben wir nein zurück, falls ja, fahren wir mit 3) fort.

3) Verifiziere S2.

Hierzu müssen wir zunächst $eff(A)$ und $eff(B)$ in polynomieller Zeit vorverarbeiten:

Für alle $(c_i, e_i) \in eff(A)$ testen wir für alle $(c_j, e_j) \in eff(A)$, ob $e_j \Rightarrow e_i$ aus $\mathbf{P}$ folgt. Für jedes so gefundene $(c_j, e_j) \in eff(A)$ setzen wir $c_i := c_i \vee c_j$.

Dasselbe machen wir auch für $eff(B)$.

Danach verifizieren wir in polynomieller Zeit, ob es für jeden primitiven Effekt $e_i \in eff(B)$ einen Effekt $e_j \in eff(A)$ gibt, so dass $e_j \Rightarrow e_i$ und $c_i \Rightarrow c_j$. Die Implikationen überprüfen wir analog wie in 3.

4) Verifiziere S3.

In polynomieller Zeit möglich.

2. Σ_2^p-Härte von AS_{BOOL}

Wir geben eine polynomielle Abbildungsreduktion von 2-QBF auf AS_{BOOL} an. Dazu bilden wir eine 2-QBF-Formel

$$\Psi = \exists x_1, \ldots, x_n \forall y_1, \ldots, y_m \Phi(\vec{x}, \vec{y})$$

auf zwei Aktionsschemata $A, B \in \mathcal{AS}_{BOOL}$ so ab, dass $A \subseteq_{as} B$ genau dann gilt, wenn Ψ gültig ist (siehe auch Beispiel 9):

A und B sind Disjunktionen.

(a) Für jede existenzquantifizierte Variable x_i erzeugen wir einen Parameter x_i in B und zwei Parameter v_i^+ und v_i^- in A mit $types_B(x_i) = types_A(v_i^+) = types_A(v_i^-) = T_i$.

(b) Für jede allquantifizierte Variable y_i erzeugen wir einen Parameter y_i in B mit $types_B(y_i) = T_{-1}$.

(c) Für jede Variable x_i erzeugen wir eine Formel $(P(v_i^+) \Leftrightarrow \top) \wedge (P(v_i^-) \Leftrightarrow \bot$ und fügen sie $pre(A)$ hinzu.

(d) Wir ersetzen jede Variable x_i (y_i) und jede Konstante c_i in $\Phi(\vec{x},\vec{y})$ durch ein Prädikat $P(x_i)$ ($P(y_i)$) bzw. $P(c_i)$ und fügen $\Phi(\vec{x},\vec{y})$ mit diesen Ersetzungen $pre(B)$ hinzu.

(e) A und B haben keine Effekte.

Der Einfachheit halber machen wir im Folgenden keinen Unterschied zwischen einer Variablen x und einem Prädikat $P(x)$ (gemäß Proposition 2).

Nach Anwendung der Abbildung erhalten wir also zwei Aktionsschemata A und B, die folgendermaßen aussehen:

$$\begin{aligned} A = \langle \quad & (P(v_1^+) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot) \\ & \wedge \ldots \wedge \\ & (P(v_n^+) \Leftrightarrow \top) \wedge (P(v_n^-) \Leftrightarrow \bot), \\ & (), \\ & \{v_1^+ \mapsto T_1, v_1^- \mapsto T_1, \ldots, v_n^+ \mapsto T_n, v_n^- \mapsto T_n\}\rangle \end{aligned}$$

und

$$\begin{aligned} B = \langle \quad & \Phi(\vec{x},\vec{y}), \\ & (), \\ & \{x_1 \mapsto T_1, \ldots, x_n \mapsto T_n, y_1 \mapsto T_{-1}, \ldots, y_m \mapsto T_{-1}\}\rangle \end{aligned}$$

Wir zeigen jetzt: Ψ ist gültig genau dann, $A \subseteq_{as} B$.

„$\Rightarrow$“

Ψ ist gültig. Wir zeigen, dass dann $A \subseteq_{as} B$ für die entstehenden Aktionsschemata A und B gilt.

Aufgrund der Wahl der Typen (da wir keine Typhierarchie angegeben haben, sind alle Typen disjunkt) können in A entweder v_i^+ durch x_i oder v_i^- durch x_i substituiert werden – andere Substitutionen sind nicht möglich.

Da $\Psi = \exists x_1, \ldots, x_n \forall y_1, \ldots, y_m \Phi(\vec{x},\vec{y})$ gültig ist, gibt es mindestens eine Belegung α für $\{x_1 \ldots x_n\}$, so dass $\forall y_1, \ldots, y_m \Phi(\alpha(\vec{x}),\vec{y})$ gültig ist (*).

Wir betrachten die Substitution A^s von A, in der v_i^+ durch x_i ersetzt wurde, wenn $\alpha(x_i) = 1$ und v_i^- durch x_i, wenn $\alpha(x_i) = 0$.

O.B.d.A. sei $\alpha(x_i) = 1$ für alle $1 \leq i \leq n$.

Wir betrachten die Kriterien aus Theorem 1: S2 ist trivialerweise erfüllt, da A und B keine Effekte haben. S3 ist erfüllt aufgrund der Wahl unserer Substitution. Um also zu zeigen, dass $A \subseteq_{as} B$ gilt, müssen wir nur noch zeigen, dass S1 gilt:

$$\begin{gathered}(P(x_1) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot) \\ \wedge \ldots \wedge \\ (P(x_n) \Leftrightarrow \top) \wedge (P(v_n^-) \Leftrightarrow \bot) \\ \Rightarrow \\ \Phi(\vec{x}, \vec{y})\end{gathered}$$

Im Folgenden betrachten wir FOL-Atome als atomare propositionale Aussagen (dies ist gerechtfertigt aufgrund von Proposition 2).

Wir machen eine Fallunterscheidung:

i. $I \not\models \big((P(x_1) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot)$
$\wedge \ldots \wedge$
$(P(x_n) \Leftrightarrow \top) \wedge (P(v_n^-) \Leftrightarrow \bot)\big)$
Dann gilt S1 trivialerweise.

ii. $I \models \big((P(x_1) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot)$
$\wedge \ldots \wedge$
$(P(x_n) \Leftrightarrow \top) \wedge (P(v_n^-) \Leftrightarrow \bot)\big)$
Dann muss $\Phi(\vec{x}, \vec{y})$ erfüllt sein, damit S1 gilt.
In (*) hatten wir festgestellt, dass für mindestens eine Belegung α für $\{x_1 \ldots x_n\}$ gilt, dass $\forall y_1, \ldots, y_m \Phi(\alpha(\vec{x}), \vec{y})$ gültig ist und wir hatten o.B.d.A. dieses α so gewählt, dass $\alpha(x_i) = 1$ für alle $1 \leq i \leq n$.
Da hier genau diese Belegung α für die existenzquantifizierten Variablen erzwungen wird, folgt, dass $\Phi(\vec{x}, \vec{y})$ gilt.

Somit folgt also auch S1 und damit $A \subseteq_{as} B$.

„⇐“

$A \subseteq_{as} B$ gilt für die entstehenden Aktionsschemata A und B. Wir zeigen, dass dann Ψ gültig ist.

Da $A \subseteq_{as} B$ gilt, gilt nach Theorem 1 auch $pre(A^s) \Rightarrow pre(B)$ für eine Substitution A^s von A.

Da durch die Art der Aktionsschemata für alle existenzquantifizierten Variablen aus Ψ ein Wahrheitswert festgelegt ist, muss dann auch Ψ gelten.

Da die Abbildung offensichtlich zudem polynomiell in der Länge von Ψ ist, handelt es sich tatsächlich um eine polynomielle Abbildungsreduktion. Somit haben wir die Σ_2^p-Härte von AS_{BOOL} gezeigt.

Aus $\mathsf{AS}_{BOOL} \in \Sigma_2^p$ und der Σ_2^p-Härte von AS_{BOOL} folgt natürlich die Σ_2^p-Vollständigkeit von AS_{BOOL}.

Beispiel 9. Wir betrachten die 2-QBF-Formel $\Psi = \exists x_1 \forall y_1 (\neg y_1 \vee x_1)$. Übersetzen wir Ψ in reine Aussagenlogik, so erhalten wir $\Psi' = ((\neg 1 \vee 1) \vee (\neg 1 \vee 0)) \wedge ((\neg 0 \vee 1) \vee (\neg 0 \vee 0)) \equiv (1 \vee 0) \wedge (1 \vee 1) \equiv 1 \wedge 1 \equiv 1$. Ψ ist also gültig.

Wenden wir die oben beschriebene Abbildung auf Ψ an, so erhalten wir die beiden Aktionsschemata

$$A = \langle (P(v_1^+) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot), (), \{v_1^+ \mapsto T_1, v_1^- \mapsto T_1\} \rangle$$

und

$$B = \langle \neg P(y_1) \vee P(x_1), (), \{x_1 \mapsto T_1, y_1 \mapsto T_{-1}\} \rangle$$

Um jetzt zu entscheiden, ob $A \subseteq_{as} B$ gilt, müssen wir zunächst entscheiden, ob es eine Substitution A^s von A gibt, so dass $pre(A^s) \Rightarrow pre(B)$ gilt, ob also $pre(A^s) \wedge \neg pre(B)$ unerfüllbar ist.

Es gibt zwei mögliche Substitutionen von A: $A[\frac{x_1}{v_1^-}]$ und $A[\frac{x_1}{v_1^+}]$.

Betrachten wir zunächst $A[\frac{x_1}{v_1^-}]$:
Wir untersuchen, ob $(P(v_1^+) \Leftrightarrow \top) \wedge (P(x_1) \Leftrightarrow \bot) \wedge \neg(\neg P(y_1) \vee P(x_1))$ unerfüllbar ist. Mit $\{P(v_1^+) \mapsto 1, P(x_1) \mapsto 0, P(y_1) \mapsto 1\}$ finden wir jedoch eine Belegung.

Somit gilt nicht $A[\frac{x_1}{v_1^-}] \subseteq_{as} A$.

Betrachten wir jetzt $A[\frac{x_1}{v_1^+}]$:
Wir untersuchen, ob $(P(x_1) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot) \wedge \neg(\neg P(y_1) \vee P(x_1))$ unerfüllbar ist. Für jede Belegung muss hier offensichtlich $P(x_1) \mapsto 1$ und $P(v_1^-) \mapsto 0$ gelten. Somit wird das Konjunkt $\neg(\neg P(y_1) \vee P(x_1))$ immer 0 und damit $(P(x_1) \Leftrightarrow \top) \wedge (P(v_1^-) \Leftrightarrow \bot) \wedge \neg(\neg P(y_1) \vee P(x_1))$ unerfüllbar.

Somit gilt $pre(A[\frac{x_1}{v_1^+}]) \Rightarrow pre(B)$. Da offensichtlich auch S2 und S3 gelten, gilt auch $A[\frac{v_1^+}{x_1}] \subseteq_{as} A$ und damit nach Theorem 1 auch $A \subseteq_{as} B$.

5.3 Komplexität von AS_{ADL}

In $\mathcal{A}_{ADL}$ erlauben wir zusätzlich zu den in $\mathcal{A}_{BOOL}$ erlaubten Operatoren noch die Existenz- und die Allquantifizierung in der Vorbedingung der Aktion sowie den Vorbedingungen der Effekte (siehe Definition 6). Somit handelt es sich bei diesen Vorbedingungen um beliebige FOL-Formeln. Auch wenn eine Objekt-Domäne stets aus einer endlichen Anzahl von Objekten besteht (siehe Definition 2), wird AS_{ADL} dadurch unentscheidbar:

Theorem 5 *(**Komplexität von** AS_{ADL}).* AS_{ADL} *ist unentscheidbar.*

Beweis 10 *(Theorem 5).* Der Satz von Trakhtenbrot [Trakhtenbrot, 1950] besagt, dass es nicht entscheidbar ist, ob eine FOL-Formel im Endlichen erfüllbar ist. Somit ist es auch nicht entscheidbar, ob eine FOL-Formel im Endlichen gültig ist. Insbesondere ist es also nicht entscheidbar, ob $pre(A) \Rightarrow pre(B)$ gilt und somit wegen Theorem 1 auch nicht, ob $A \subseteq_{as} B$ gilt.

Kapitel 6

Effiziente Algorithmen für das Subsumptions-Entscheidungsproblem

In Kapitel 5 haben wir gesehen, dass das Subsumptions-Entscheidungsproblem eingeschränkt auf Aktionsschemata der Klassen $\mathcal{A}_{STRIPS}$ beziehungsweise $\mathcal{A}_{BOOL}$ $\mathcal{NP}$-vollständig beziehungsweise Σ_2^p-vollständig ist.

Es ist also recht wahrscheinlich, dass ein „naiver“ Algorithmus zum Scheitern verurteilt ist, so dass wir uns darauf konzentrieren sollten, Algorithmen zu finden, die in der Praxis meistens schneller sind.

6.1 Algorithmus für AS_{STRIPS}

Intuitiv gesehen ist die $\mathcal{NP}$-Vollständigkeit von $\mathcal{A}_{STRIPS}$ darin begründet, dass man für alle zulässigen Substitutionen die syntaktischen Kriterien aus Theorem 2 überprüfen muss – dies sind potentiell exponentiell viele in der Anzahl der Parameter der Aktionsschemata. Im Worst Case (wenn A nicht von B subsumiert wird oder die richtige Substitution erst als Letzte gefunden wird), muss man alle Substitutionen durchprobieren.

```
1  Algorithmus :checkSubsumptionStrips
   Eingabe : Action super, Action sub, PredicateInclusions predIncl
   Ausgabe : boolean isSubsumed?
2  wenn Es gibt mehr Parameter in super als in sub dann
3      return false;
4  Bilde Menge possibleSubs aller möglichen Substitutionen;
5  wenn Es gibt für eine Variable in super keine in sub mit selbem
   Typ/Subtyp dann
6      return false;
7  Erstelle Array C1 aller Atome in pre(super);
8  Erstelle Array C2 aller Atome in pre(sub);
9  Erstelle Array E1 aller Atome in eff(super);
10 Erstelle Array E2 aller Atome in eff(sub);
11 Vereinige C1 und E1 zu A1 (Umbenennung der Atomnamen bei
   Gleichheit);
12 Vereinige C2 und E2 zu A2 (Umbenennung der Atomnamen bei
   Gleichheit);
13 return checkSubsumptionRec(A1,0,A2,usedSubs,possibleSubs,predIncl);
```

Algorithmus 1 : checkSubsumptionStrips

Glücklicherweise kann man in vielen praktischen Fällen zusätzliche Einschränkungen an die zulässigen Substitutionen machen.

Zunächst gilt, dass A mindestens so viele Parameter wie B haben muss, denn jedes Atom in $pre(A^s)$ $(eff(A^s))$ muss auch in $pre(B)$ $(eff(B))$ vorkommen. Dies ist aber nur möglich, wenn für alle Parameter $x \in \mathcal{S}(A^s)$ auch $x \in \mathcal{S}(B)$ gilt, wenn also alle Parameter von A^s auch Parameter von B sind, was wiederum nur möglich ist, wenn A mindestens so viele Parameter wie B hat.

Weiterhin kann man ausnutzen, dass viele Aktionsschemata typisierte Parameter haben: Für jede Substitution A^s von A gilt, dass ein Parameter x aus $\mathcal{S}(A)$ nur durch einen Parameter y aus $\mathcal{S}(B)$ substituiert werden kann, wenn $types_A(x) \Rightarrow types_B(y)$ aus der gegebenen Typhierarchie $\mathbf{T}$ folgt (wenn also der Typ von x

```
1  Algorithmus :checkSubsumptionRecStrips
   Eingabe : Array[Atoms] A1, index idx, Array[Atoms] A2,
             Array[Substitutions] usedSubs, Array[Substitutions]
             possibleSubs, PredicateInclusions predIncl
   Ausgabe : boolean isSubsumed?
2  wenn idx größer als Länge von A1 dann
3  |  return true;
4  sonst
5  |  currentAtom = A1[idx];
6  |  für Alle Atome derivedAtom in A2 tue
7  |  |  wenn derivedAtom folgt in predIncl aus currentAtom;
8  |  |  dann
9  |  |  |  Suche passende Substitution s der Parameter von currentAtom
   |  |  |  und derivedAtom;
10 |  |  |  wenn s gefunden dann
11 |  |  |  |  erweitere usedSubs um s;
12 |  |  |  |  bool=checkSubsumptionRec(A1,idx+1,A2,usedSubs,
13 |  |  |  |  possibleSubs,predIncl);
14 |  |  |  |  wenn bool=true dann
15 |  |  |  |  |  return true;
16 |  |  |  |  sonst
17 |  |  |  |  |  return false;
18 |  |  |  sonst
19 |  |  |  |  return false;
20 |  return false;
```

Algorithmus 2 : checkSubsumptionRecStrips

ein Untertyp vom Typen von y ist), denn sonst gäbe es für eine Domäne Δ eine Aktion $a \in Actions_\Delta(A)$, für die es keine Aktion $b \in Actions_\Delta(B)$ gibt, so dass $a \subseteq_{as} b$ gilt. Somit lassen sich alle Substitutionen A^s von A, in denen ein Parameter x aus $\mathcal{S}(A)$ durch einen Parameter y aus $\mathcal{S}(B)$ ersetzt wurde, so dass diese Eigenschaft nicht gilt, ausschließen.

Noch größeren Nutzen in der Praxis hat meist eine weitere für die Zulässigkeit der Substitution notwendige Eigenschaft:

Ein Parameter y, der in einem Prädikat P in $pre(B)$ (beziehungsweise $eff(B)$) an Stelle i als Argument vorkommt, muss auch in einem Prädikat P' in $pre(A)$ (beziehungsweise $eff(A)$) an Stelle i vorkommen, so dass P in $\mathbf{P}$ aus P' folgt. Im Idealfall taucht jedes Prädikat pro Aktionsschema nur einmal auf und $\mathbf{P}$ ist leer, denn dann gibt es höchstens eine zulässige Substitution.

Durch diese notwendige Eigenschaft lässt sich die Anzahl der zulässigen Substitutionen in den meisten praktischen Fällen massiv reduzieren.

Die beiden gerade vorgestellten Ideen sind in den Algorithmen 1 und 2 in Pseudocode festgehalten. Dabei übernimmt Algorithmus 1 die Vorarbeit (unter anderem schränkt er die zulässigen Substitutionen auf jene ein, die die eben beschriebene Typ-Eigenschaft erfüllen) und ruft dann den rekursiven Algorithmus 2 auf. Algorithmus 2 ist ein Backtracking-Algorithmus, der sich ähnlich wie die Davis-Putnam-Prozedur verhält: Zu jedem Prädikat $P(\vec{x}) \in pre(A)$ wird ein Prädikat $P'(\vec{y})$ gesucht, so dass P' aus P folgt. Dann werden die Argumente $\vec{y}$ von P' durch die Argumente $\vec{x}$ von P' substituiert (Konstanten müssen natürlich gleich sein). Sind diese Substitutionen möglich und konsistent zu den bereits durchgeführten Substitutionen, so wird mit dem nächsten Prädikat fortgefahren. Ist eine solche Substitution nicht möglich oder inkonsistent zu einer bereits durchgeführten Substitution, so wird ein Backtracking-Schritt gemacht und ein anderes Prädikat gewählt. Tritt der oben erwähnte Idealfall ein, dass jedes Prädikat nur einmal vorkommt und $\mathbf{P}$ leer ist, so erhalten wir auf diese Weise eine lineare Laufzeit.

```
1 Algorithmus :CheckSubsumptionBool
  Eingabe : Action super, Action sub
  Ausgabe : boolean isSubsumed?
2 generalCondEffects=extractConditionalPrimitiveEffects(generalAction);
3 specifiedCondEffects=extractConditionalPrimitiveEffects(specifiedAction);
4 Bilde Menge possibleSubs aller möglichen Substitutionen;
5 Instanziere Prädikaten-Inklusionen in predInclInst;
6 Fasse gleiche Effekte zusammen;
7 return checkSubsumptionRec(pre(generalAction), pre(specifiedAction),
  generalCondEffects, 0, specifiedCondEffects, possibleSubstitutions,
  usedSubstitutions, predInclInst);
```

Algorithmus 3 : checkSubsumptionBool

6.2 Algorithmus für AS_{BOOL}

In AS_{STRIPS} galt, dass A mindestens so viele Parameter haben muss wie B. Diese Forderung müssen wir jetzt leider aufgeben. Betrachte hierzu Beispiel 10.

Beispiel 10. Es gelte $A \subseteq_{as} B$. Dann lässt sich keine Aussage über das Verhältnis der Anzahlen der Parameter der beiden Aktionsschemata machen:

1. A kann genau so viele Parameter wie B haben: $A = \langle P(x), (), \{x \mapsto T\}\rangle$ und $B = \langle P(x), (), \{x \mapsto T\}\rangle$
2. A kann mehr Parameter als B haben: $A = \langle P(x) \wedge P(y), (), \{x \mapsto T, y \mapsto T\}\rangle$ und $B = \langle P(z), (), \{z \mapsto T\}\rangle$
3. A kann weniger Parameter als B haben: $A = \langle P(x), (), \{x \mapsto T\}\rangle$ und $B = \langle P(y) \vee P(z), (), \{y \mapsto T, z \mapsto T\}\rangle$

Aus Beispiel 10 lässt sich auch ablesen, dass es im Allgemeinen nicht notwendig ist, alle Variablen aus B zu substituieren.

Wir können jedoch die Backtracking-Technik, die für den Algorithmus für AS_{STRIPS} verwendet wurde, eingeschränkt weiterhin benutzen. Zwar lässt sich

```
1 Algorithmus :unsat
  Eingabe : Condition cond
  Ausgabe : boolean isUnsatisfiable?
2 cond'=createSATequivalentCond(cond);
3 cond"=createDIMACSformat(cond");
4 wenn (solve(cond")) dann
5 |  return false;
6 sonst
7 |  return true;
```

Algorithmus 4 : unsat

nicht mehr aussagen, dass für jedes Atom in $pre(A^s)$ ein Atom in $pre(B)$ vorkommen muss, so dass deren Argumente substituierbar sein müssen, dies gilt aber jetzt für jeden Effekt, denn diese müssen atomar sein.

Wenn wir also zunächst S2 testen und keine passende Substitution finden, können wir den Test an dieser Stelle abbrechen. Gibt es jeden Effekt nur einmal und **P** ist leer, so erhalten wir also auch hier eine lineare Laufzeit.

Zwei Dinge sind noch zu beachten:

Zunächst wollen wir ja für den Test $pre(A) \Rightarrow pre(B)$ einen propositionalen SAT-Solver benutzen. Deshalb müssen wir die Prädikaten-Inklusions-Axiome (die ja allquantifiziert sind) für jedes vorkommende Prädikat instanzieren.

Außerdem müssen wir den Fall beachten, dass ein Effekt zweimal mit unterschiedlichen Vorbedingungen auftritt. Deshalb müssen wir solche Effekte zunächst zusammenfassen. Wie dies geschieht, illustriert Beispiel 11.

Beispiel 11. Wir haben ein Aktionsschema

$$A = \langle P(x), \\ (P_1(x) \rhd E_1(x)) \wedge (P_2(x) \rhd E_2(x)) \wedge (P_3(x) \rhd E_3(x)), \\ \{x \mapsto \textsc{Thing}\}\rangle$$

und eine Prädikaten-Inklusions-Hierarchie

$$\mathbf{P} = \{E_1 \Rightarrow E_2, E_2 \Rightarrow E_3\}$$

gegeben. Dann erweitern wir A wie folgt zu A':

$$A' = \langle P(x),$$
$$(P_1(x) \rhd E_1(x)) \wedge (P_2(x) \vee P_1(x) \rhd E_2(x)) \wedge (P_3(x) \vee P_2(x) \vee P_1(x) \rhd E_3(x)),$$
$$\{x \mapsto \textsc{Thing}\}\rangle$$

```
1  Algorithmus :CheckSubsumptionRecBool
   Eingabe : Precondition preSuper, Precondition, preSub, ConEffects
             effSuper, int idx, ConEffects effSub, Subs possSubs, Subs
             usedSubs, PredInclusions predInclInst
   Ausgabe : boolean isSubsumed?
2  wenn idx==genEff.length dann
3      für alle zusätzlichen Subs s tue
4          wenn unsat((and (not(preSuper)) preSub predInclInst)[s]) dann
5              Merke s in usedSubs;
6              return true;
7          sonst
8              return false;
9  nehme nächsten Effekt e in effSuper;
10 für Effekte e' in effSub tue
11     wenn ((e'.predicate == e.predicate) and(Parameter lassen sich durch
       s' substituieren)) dann
12         merke s' in usedSubs;
13         für zusätzliche Subs s" tue
14             wenn unsat((and (not(cond(e'))) cond(e) predInclInst)[s"])
               dann
15                 merke s" in usedSubs;
16                 bool = checkSubsumptionRec(preSuper, preSub, effSuper,
                   idx+1, effSub, possSubs, usedSubs);
17                 wenn bool==true dann
18                     return true;
19                 sonst
20                     entferne s" aus usedSubs;
21                     continue;
22         entferne s' aus usedSubs;
23 return false;
```

Algorithmus 5 : checkSubsumptionRecBool

Kapitel 7

Implementation

7.1 PDDL

Für die Implementation der in Kapitel 6 beschriebenen Algorithmen benötigen wir natürlich zunächst einmal eine Repräsentationssprache für Aktionen. Sehr gut geeignet für diese Zwecke ist die *Planning Domain Definition Language* (*PDDL*), welche 1998 von Drew McDermot entworfen wurde [McDermott, 1998] und seitdem kontinuierlich weiterentwickelt wird. Mittlerweile existiert Version 3.0 [Gerevini und Long, 2006], welche im Rahmen der International Planning Competition 2006 eingesetzt wurde. PDDL basiert auf einer Standardisierung des STRIPS-Formalismus, verfügt jedoch auch über zahlreiche weitere Konstrukte wie z.B. numerische Fluenten, zeitabhängige Aktionen oder abgeleitete Prädikate. Heutzutage gilt PDDL als Standard für die Repräsentation und den Austausch von Planungsdomänen.
Ein Planungsproblem wird zur Formalisierung in PDDL aufgeteilt in eine Domänenbeschreibung, die - z.B. durch parametrisierte Aktionen - das Verhalten einer Domäne beschreibt, sowie eine Problembeschreibung, welche spezifische Objekte, den Startzustand, das Ziel, sowie eine Metrik, mit der die Qualität von Plänen ermittelt werden kann, enthält.
Im Folgenden beschränken wir uns auf die Teilfragmente von PDDL, durch die wir alle Aktionsschemata aus $\mathcal{A}_{BOOL}$ beschreiben können, wobei wir als boole-

sche Operatoren der Einfachheit halber nur `and`, `or` und `not` (und nicht `imply`) verwenden.

Zur Darstellung von Prädikaten-Inklusions-Axiomen verwenden wir die Syntax der abgeleiteten Prädikate [Edelkamp und Hoffmann, 2004], wobei wir diese gemäß unserer Definition der Prädikaten-Inklusions-Axiome einschränken - zusätzlich gehen wir davon aus, dass beide Prädikate dieselben Argumente besitzen. Dabei verzichten wir auf die in PDDL verwendete Konvention einer strikten Separation von abgeleiteten und durch Aktionen veränderbare Prädikate.

Das jeweils erste Prädikat einer Prädikaten-Inklusions-Definition ist immer wahr, wenn das zweite Prädikat wahr ist.

7.2 PDDL-Modell

Wir verwenden ein objektorientiertes Modell der PDDL-Syntax von $\mathcal{AS}_{BOOL}$. Dieses PDDL-Modell wurde in Java implementiert (siehe die UML-Diagramme in Anhang A). Es besteht aus den Paketen `condition`, `effect` und `pddlmodel`.

Das Herzstück von `condition` ist die abstrakte Klasse `Condition`. `AndCondition`, `OrCondition` und `NotCondition` sind aus `Condition` abgeleitet und verwenden als Instanzvariablen Listen von `Condition`-Objekten. Somit ist es möglich, Methoden wie zum Beispiel das Ausgeben einer Condition auf dem Bildschirm oder auch das Umwandeln einer Condition in eine Normalform (siehe hierzu Kapitel 7.3.2) rekursiv und effizient zu implementieren.

Für Details zum `condition`-Paket siehe das UML-Diagramm A.1 in Anhang A.

Analog zum `condition`-Paket wurde das `effect`-Paket erstellt, siehe hierzu das UML-Diagramm A.2 in Anhang A.

Die zentral Klasse im Paket `pddl-model` ist die Klasse `DomainDescription`. Für Details siehe das UML-Diagramm A.3 in Anhang A.

7.2.1 PDDL-Parser

Die Klasse `DomainDescription` des Paketes `pddl-model` stellt Methoden zur Verfügung, durch die eine Domänenbeschreibung neu erzeugt werden kann. Für die Praxis ist es jedoch wichtig, dass man zusätzlich die Möglichkeit hat, bereits vorhandene PDDL-Domänen einzulesen und daraus ein objektorientiertes Modell zu erstellen.

Diese Möglichkeit wird durch die Implementation eines PDDL-Parsers gegeben. Dieser wurde mit Hilfe des Java-Parsergenerators javacc [javacc, 2007] im Paket `parser` implementiert. javacc generiert aus einer Definitionsdatei einige Java-Klassen, unter anderem die Klasse `PDDL-Parser`, die eine PDDL-Domänen-Datei einliest und durch geeignete semantische Anweisungen ein entsprechendes PDDL-Modell generiert.

7.3 SubsumptionChecker

Im Paket `subsumptionchecker` befindet sich unter anderem die Klasse `SubsumptionChecker`, die zunächst mit Hilfe des Parsers eine als Kommandozeilenargument übergebene PDDL-Domänen-Datei einliest und daraus ein objektorientiertes PDDL-Modell erzeugt. Anschließend lassen sich durch Verwendung der von `STRIPSChecker` und `BOOLChecker` bereitgestellten Methoden Subsumptionsprobleme von Aktionsschemata aus $\mathcal{A}_{STRIPS}$ beziehungsweise $\mathcal{A}_{BOOL}$ entscheiden.

7.3.1 STRIPSChecker

Die statische Klasse `STRIPSChecker` stellt die Methode `checkSubsumption` zur Verfügung, die den Algorithmus 1 implementiert.

7.3.2 BOOLChecker

Die statische Klasse `BOOLChecker` stellt die Methode `checkSubsumption` zur Verfügung, die den Algorithmus 3 implementiert.

Für den Erfüllbarkeitstest von booleschen Formeln wird die java-Bibliothek sat4j [sat4j, 2007] verwendet.

An dieser Stelle gehen wir exemplarisch auf zwei Details der Implementierung ein:

createSATPreservingFormula

Mittels dieser Methode wird aus einer booleschen Formel eine erfüllbarkeitsäquivalente Formel in konjunktiver Normalform erzeugt (siehe Proposition 3). Dabei wird die objektorientierte Modellierung einer booleschen Formel ausgenutzt (siehe Abbildung A.1).

Zusätzlich benötigt werden hierfür die Klasse `VarNameMgr`, die neue Prädikaten-Namen erzeugt und die Subklassen der abstrakten Klasse `Equivalence` (`AndEquivalence`, `OrEquivalence` und `NotEquivalence`), die Instanzen der drei Arten von Äquivalenzen

$$(X \Leftrightarrow \neg Y)$$

$$(X \Leftrightarrow (Y \wedge Z))$$

$$(X \Leftrightarrow (Y \vee Z))$$

repräsentieren. Diese Klassen verfügen unter anderem über eine Methode `translateToDimacs`, die eine Äquivalenz in konjunktiver Normalform und in der von sat4j geforderten DIMACS-Syntax ausgibt.

Die Methode bekommt eine Formel (repräsentiert durch ein Objekt `c` der Klasse `Condition`) übergeben und liefert einen Prädikaten-Namen zurück. Sie ist rekursiv definiert:

- `c` ist Objekt der Klasse `EmptyCondition`
  Es werden neue Prädikate P_1, P_2 und P_3 und zwei Äquivalenzen erzeugt:

$P_2 \Leftrightarrow \neg P_3$ und $P_1 \Leftrightarrow P_2 \vee P_3$. Dann wird P_1 zurückgegeben (P_1 ist somit immer wahr).

- `c` ist Objekt der Klasse `PrimitiveCondition`
  Das primitive Prädikat wird zurückgegeben.

- `c` ist Objekt der Klasse `AndCondition`
  Die Anzahl n der Konjunkte wird bestimmt und danach vier Fälle unterschieden:

  - $n = 0$
    `createSATPreservingFormula` wird für eine neu erzeugtes Objekt der Klasse `EmptyCondition` aufgerufen und dessen Rückgabewert zurückgegeben.
  - $n = 1$
    `createSATPreservingFormula` wird für das eine Konjunkt aufgerufen und dessen Rückgabewert zurückgegeben.
  - $n = 2$
    `createSATPreservingFormula` wird für beide Konjunkte aufgerufen und aus deren Rückgabewerten P_1 und P_2 und einem mittels `VarNameMgr` neu erzeugten Prädikat P_3 wird ein neues Objekt der Klasse `AndEquivalence` erstellt, dass die Äquivalenz $P_3 \Leftrightarrow P_1 \wedge P_2$ repräsentiert. Anschließend wird P_3 zurückgegeben.
  - $n > 2$
    `createSATPreservingFormula` wird für das erste Konjunkt und für `c` ohne das erste Konjunkt aufgerufen und aus deren Rückgabewerten P_1 und P_2 und einem mittels `VarNameMgr` neu erzeugten Prädikat P_3 wird ein neues Objekt der Klasse `AndEquivalence` erstellt, dass die Äquivalenz $P_3 \Leftrightarrow P_1 \wedge P_2$ repräsentiert. Anschließend wird P_3 zurückgegeben. Analog zum

- `c` ist Objekt der Klasse `OrCondition`
  Analog zum vorhergehenden Fall.

DerivedPredicates

Ein Objekt der Klasse `DomainDescription` besitzt zwei getrennte Repräsentationen einer Prädikaten-Inklusions-Hierarchie. In der ersten werden nur die Prädikaten-Inklusions-Axiome, die in der PDDL-Datei stehen, gespeichert, während in der anderen auch alle implizit vorhandenen Axiome explizit gespeichert werden.

Durch die zweite Variante ist es möglich, in linearer Zeit zu testen, ob ein Prädikat aus einem anderen Prädikat folgt, indem man einfach jedes Axiom einmal betrachtet.

Die erste Variante hält die Größe der Instanzierungen der Axiome, die in `BOOLChecker` gemacht werden, möglichst klein.

Um die zweite Variante aus der ersten zu erzeugen und konsistent zu halten, wird diese nach jedem Einfügen eines neuen Axioms durch eine rekursive Prozedur aktualisiert.

Kapitel 8

Zusammenfassung

8.1 Ergebnisse

Ziel dieser Arbeit war es, die Subsumption deterministischer Aktionsschemata zu untersuchen.

Dazu wurde zunächst der Subsumptionsbegriff zweier Aktionsschemata formal eingeführt; hierbei wurde die sogenannte Abstraktions-Subsumption gewählt, bei der eine Aktion A von einer Aktion B abstrakt subsumiert wird ($A \subseteq_{as} B$), wenn A eine speziellere Vorbedingung als B hat und alle Effekte, die durch B entstehen, auch durch A entstehen. Diese Definition umfasst für STRIPS-Aktionsschemata zum Beispiel die Definition von Knoblock [Knoblock, 1994].

Anschließend wurde für Aktionsschemata ohne allquantifizierte Effekte ein syntaktisches Kriterium hergeleitet, mit dessen Hilfe das Subsumptionsproblem entschieden werden kann. Somit ist es möglich, das Subsumptionsproblem losgelöst von den unendlich vielen in dessen Definition verwendeten Objekt-Domänen zu untersuchen.

Es wurde dargestellt, warum eine Modellierung von Aktionsschemata in OWL [McGuinness und van Harmelen, 2004] wünschenswert wäre und warum eine solche Modellierung unter einigen als notwendig betrachteten Voraussetzungen nicht möglich erscheint.

Es wurde dann die Komplexität des Subsumptionsproblemes untersucht. Dabei wurde bewiesen, dass das Subsumptionsproblem $\mathcal{NP}$-vollständig ist, wenn man die Ausdrucksmächtigkeit der beteiligten Aktionsschemata auf das STRIPS-Fragment [Fikes und Nilsson, 1971] beschränkt, Σ_p^2-vollständig, wenn man Aktionsschemata mit booleschen Operatoren aber ohne Quantoren zulässt und unentscheidbar, wenn es sich um ADL-Aktionsschemata [Pednault, 1989] handelt.

Anschließend wurden Algorithmen für die ersten beiden Fälle vorgestellt und implementiert.

8.2 Ausblick

Aufbauend auf den Ergebnissen dieser Arbeit kann man eine Reihe weiterer Fragestellungen definieren.

Von Interesse wäre zum Beispiel die Komplexität des Subsumptionsproblems zweier boolescher Aktionsschemata, die allquantifizierte Effekte verwenden. Man könnte empirisch untersuchen, ob Näherungsalgorithmen für das Subsumptionsproblem zweier ADL-Aktionsschemata in der Praxis nützlich sind oder sich relevante Spezialfälle dieses Problemes definieren lassen, für die es entscheidbar wird. Aufgeworfen wird auch die Frage, inwieweit sich das Konzept der Prädikaten-Inklusions-Axiome erweitern lässt, ohne unentscheidbar zu werden, beziehungsweise wie komplex das Subsumptionsproblem für solche Erweiterungen wird.

Literaturverzeichnis

[Armano u. a. 2003] Armano, Giuliano ; Cherchi, Giancarlo ; Vargiu, Eloisa: *An Extension to PDDL for Hierarchical Planning*. Oktober 01 2003

[Baader u. a. 2003] Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. – ISBN 0-521-78176-0

[Brachman und Schmolze 1985] Brachman, Ronald J. ; Schmolze, James G.: An overview of the KL-ONE knowledge representation system. In: *Cognitive Science* 9 (1985), Nr. 2, S. 171–216

[Devanbu und Litman 1991] Devanbu, Premkumar T. ; Litman, Diane J.: Plan-Based Terminological Reasoning. In: *KR*, 1991, S. 128–138

[Ebbinghaus u. a. 1986] Ebbinghaus, Heinz-Dieter ; Flum, Jörg ; Thomas, Wolfgang: *Einführung in die mathematische Logik*. second. Wissenschaftliche Buchgesellschaft Darmstadt, 1986

[Edelkamp und Hoffmann 2004] Edelkamp, Stefan ; Hoffmann, Joerg: PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition / Institut für Informatik, Universität Freiburg. Januar 21 2004. – report00195

[Fikes und Nilsson 1971] Fikes, R. ; Nilsson, N. J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In: *Artificial Intelligence-4, 1971* 2 (1971), S. 189–208

[Fox und Long 2003] Fox, Maria ; Long, Derek: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. In: *J. Artif. Intell. Res. (JAIR)* 20 (2003), S. 61–124

[Garey und Johnson 1979] Garey, M. R. ; Johnson, D. S.: *Computers and intractability; a guide to the theory of NP-completeness.* W.H. Freeman, 1979

[Gerevini und Long 2006] Gerevini, A. ; Long, D.: Preferences and Soft Constraints in PDDL3. In: Gerevini, A. (Hrsg.) ; Long, D. (Hrsg.): *Proceedings of ICAPS workshop on Planning with Preferences and Soft Constraints*, 2006, S. 46–53

[Heinsohn u. a. 1992] Heinsohn, Jochen ; Kudenko, Daniel ; Nebel, Bernhard ; Profitlich, Hans-Jürgen: RAT — Representation of Actions using Terminological Logics. In: Heinsohn, J. (Hrsg.) ; Hollunder, B. (Hrsg.): *DFKI Workshop on Taxonomic Reasoning — Proceedings*, DFKI, Saarbrücken/Kaiserslautern, Germany, 1992 (DFKI Document D-92-08)

[javacc 2007] javacc: 2007. – URL `https://javacc.dev.java.net/`. – [Online; Stand 12. April 2007]

[Kemke 2003a] Kemke, Christel: A Formal Approach to Describing Action Concepts in Taxonomical Knowledge Bases. In: Zhong, Ning (Hrsg.) ; Ras, Zbigniew W. (Hrsg.) ; Tsumoto, Shusaku (Hrsg.) ; Suzuki, Einoshin (Hrsg.): *Foundations of Intelligent Systems, 14th International Symposium, ISMIS 2003, Maebashi City, Japan, October 28-31, 2003, Proceedings* Bd. 2871, Springer, 2003, S. 657–662. – ISBN 3-540-20256-0

[Kemke 2003b] Kemke, Christel: A Formal Theory for Describing Action Concepts in Terminological Knowledge Bases. In: Xiang, Yang (Hrsg.) ; Chaibdraa, Brahim (Hrsg.): *Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2003, Halifax, Canada, June 11-13, 2003, Proceedings* Bd. 2671, Springer, 2003, S. 458–465. – ISBN 3-540-40300-0

[Kemke und Walker 2006] Kemke, Christel ; Walker, Erin: Planning with

Action Abstraction and Plan Decomposition Hierarchies. In: *IAT*, IEEE Computer Society, 2006, S. 447–451

[Knoblock 1994] Knoblock, Craig A.: Automatically Generating Abstractions for Planning. In: *Artificial Intelligence* 68 (1994), Nr. 2, S. 243–302

[Korf 1987] Korf, Richard E.: Planning as Search: A Quantitative Approach. In: *Artificial Intelligence* 33 (1987), Nr. 1, S. 65–88

[Lewis und Papadimitriou 1981] Lewis, Harry R. ; Papadimitriou, Christos H.: *Elements of the theory of computation.* Englewood Cliffs, NJ : Prentice-Hall, 1981 (Prentice-Hall Software Series). – 466 S

[Liebig und Rösner 1997] Liebig, Thorsten ; Rösner, Dietmar: Action Hierarchies in Description Logics. In: Brachman, Ronald J. (Hrsg.) ; Donini, Francesco M. (Hrsg.) ; Franconi, Enrico (Hrsg.) ; Horrocks, Ian (Hrsg.) ; Levy, Alon Y. (Hrsg.) ; Rousset, Marie-Christine (Hrsg.): *Proceedings of the 1997 International Workshop on Description Logics, Université Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique LRI* Bd. 410, 1997

[McDermott 1998] McDermott, D.: *PDDL — the planning domain definition language.* 1998

[McGuinness und van Harmelen 2004] McGuinness, Deborah L. ; Harmelen, Frank van: OWL Web Ontology Language Overview / World Wide Web Consortium. February 2004. – W3C Recommendation

[Meyer und Stockmeyer 1972] Meyer ; Stockmeyer: The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In: *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1972

[Papadimitriou 1994] Papadimitriou, C. H.: *Computational Complexity.* Addison-Wesley, 1994

[Pednault 1989] Pednault, E.: ADL: Exploring the middle ground between STRIPS and the situation calculus. In: *kr89*, 1989, S. 324–332

[Sacerdoti 1974] SACERDOTI, Earl D.: Planning in a Hierarchy of Abstraction Spaces. In: *Artificial Intelligence* 5 (1974), Nr. 2, S. 115–135

[sat4j 2007] SAT4J: 2007. – URL `http://www.sat4j.org/index.php`. – [Online; Stand 12. April 2007]

[Schöning 1992] SCHÖNING, Uwe: *Reihe Informatik.* Bd. 56: *Logik für Informatiker, 3. Auflage.* Bibliographisches Institut, 1992. – ISBN 3-411-14013-5

[Sperschneider und Antoniou 1991] SPERSCHNEIDER, V. ; ANTONIOU, G.: *Logic: A Foundation for Computer Science.* Addison-Wesley: Reading, MA, 1991

[Trakhtenbrot 1950] TRAKHTENBROT, B. A.: Impossibility of an algorithm for the decision problem in finite classes. In: *Doklady Akademii Nauk SSSR* 70 (1950), S. 569–572

[Weida und Litman 1992] WEIDA, Robert A. ; LITMAN, Diane J.: Terminological Reasoning with Constraint Networks and an Application to Plan Recognition. In: *KR*, 1992, S. 282–293

Anhang A

UML-Diagramme

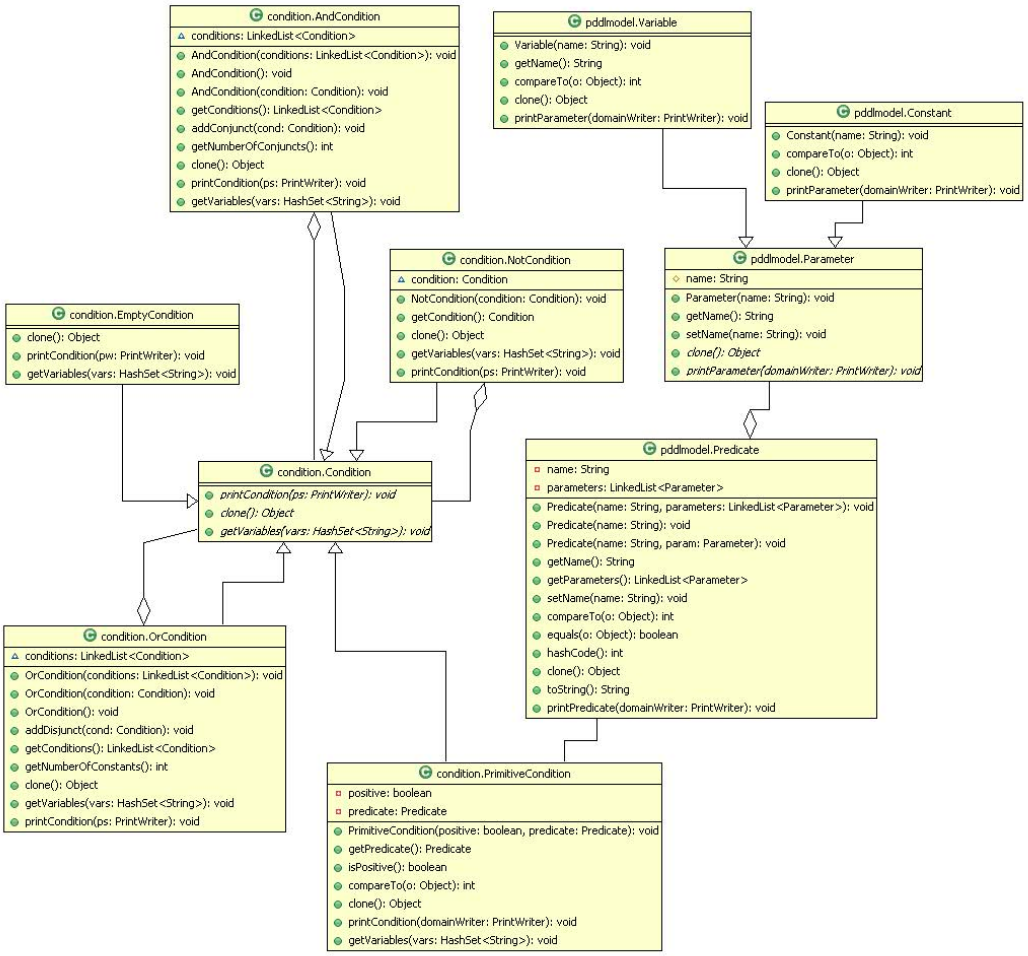

Abbildung A.1: Package condition

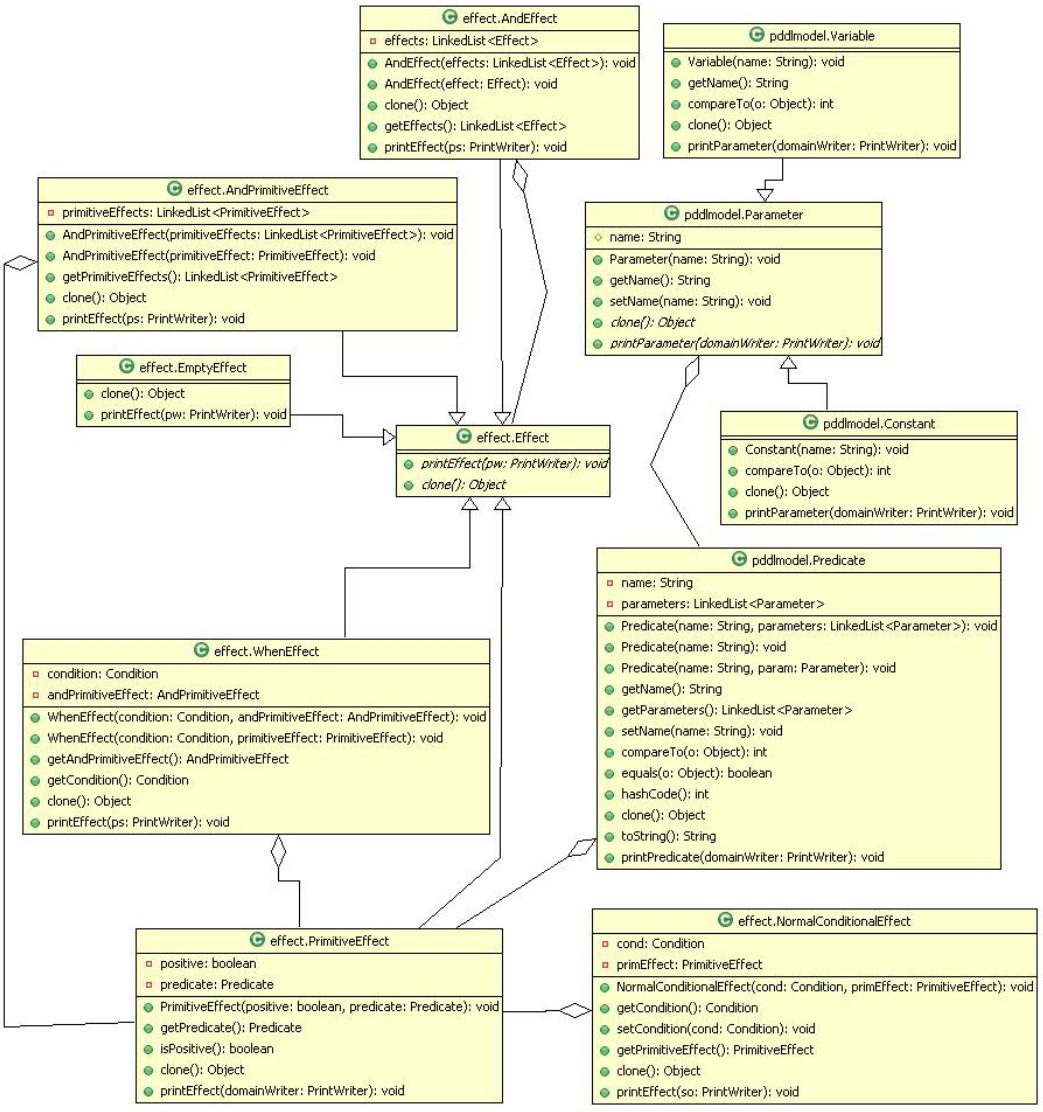

Abbildung A.2: Package effect

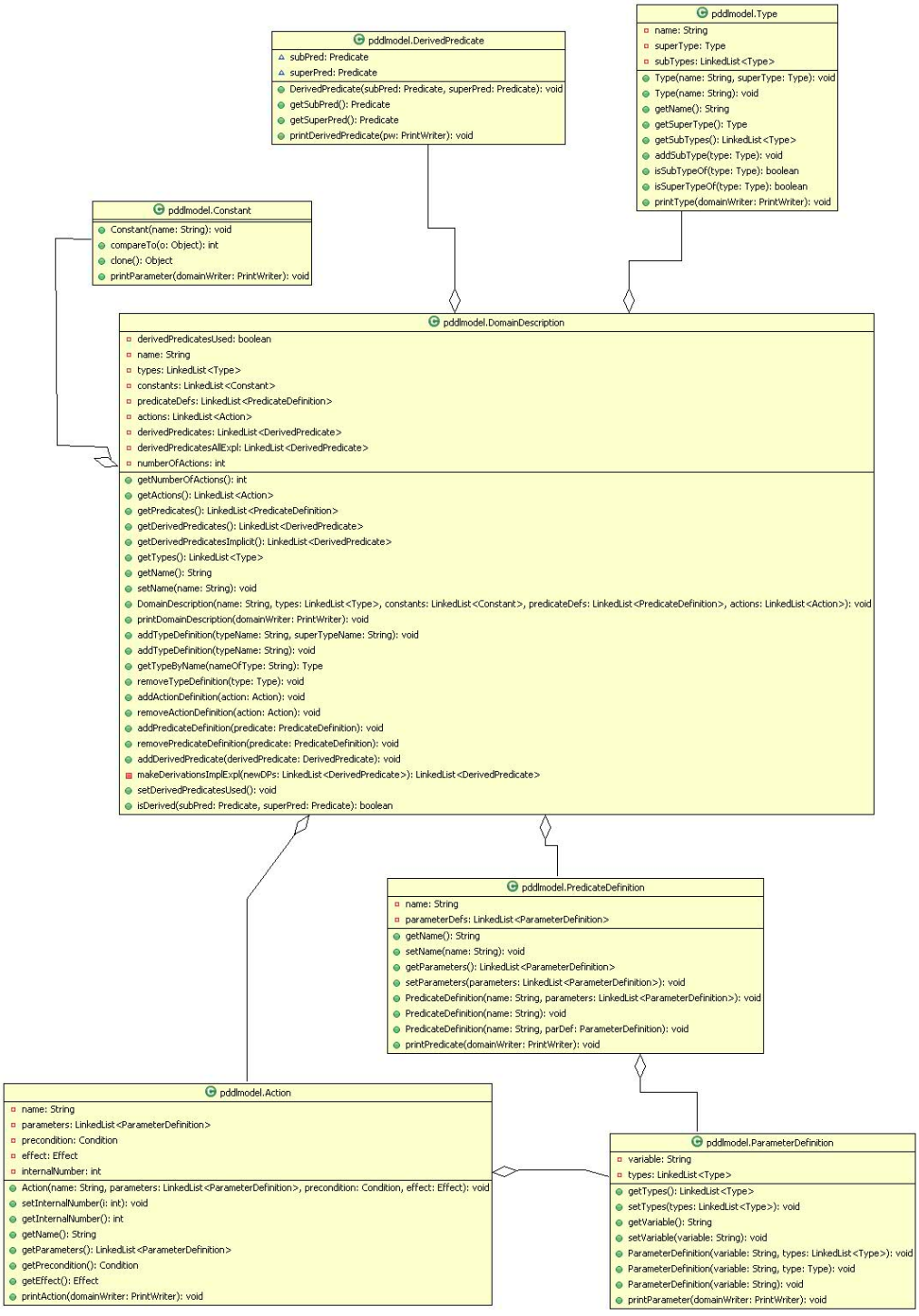

Abbildung A.3: Package pddlmodel