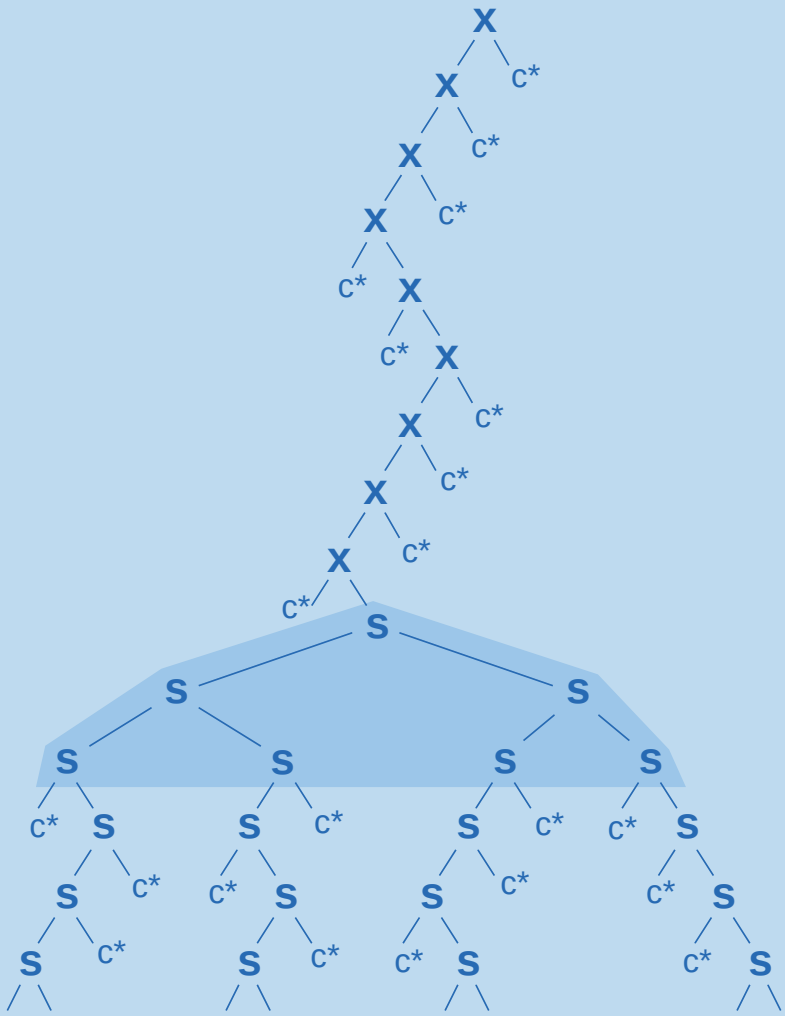
Tim Priesnitz

Entailment von nicht-strukturellen Teiltyp-Constraints



——— Diplomarbeit, Universität des Saarlandes ———

Tim Priesnitz
Forschungsbereich Programmiersysteme
Fachbereich 14, Informatik
Universität des Saarlandes
Im Stadtwald
66041 Saarbrücken, Deutschland
EMail: tim@ps.uni-sb.de
Homepage: http://www.ps.uni-sb.de/~tim/

Titel der Diplomarbeit:	**Entailment von nicht-strukturellen Teiltyp-Constraints**
Leitung:	Prof. Dr. Gert Smolka
Zweitgutachter:	Prof. Dr.-Ing. Jörg Siekmann
Betreuung:	Dr. Joachim Niehren

Eingereicht am 25. Februar 2000.

Finanzielle Unterstützung. Diese Arbeit wurde finanziell durch den Sonderforschungsbereich „Ressourcenadaptive Kognitive Prozesse“ (SFB 378) der Universität des Saarlandes und der Esprit Working Group CCL II (EP 22457) unterstützt.

Danksagung

Besonderer Dank gilt meinem Betreuer Joachim Niehren, an den ich mich jederzeit wenden konnte und der desöfteren durch seine für diese Arbeit außerordentliche wertvolle Erfahrung mich von abenteuerlichen Beweistechniken abbrachte. Desweiteren möchte ich mich bei Martin Müller bedanken, der mir das Thema dieser Arbeit näher brachte und durch zahlreiche Diskussionen zu dieser Arbeit beisteuerte. An dieser Stelle bedanke ich mich auch bei dem gesamten Team des Forschungsbereichs Programmiersysteme, insbesondere bei Prof. Gert Smolka, für die Unterstützung und freundliche Atmosphäre. Bedanken möchte ich mich auch bei Prof. Jörg Siekmann für sein Engagement. Dank geht auch an Katrin Erk, Holger Dewes und Georg Jung für ihre Korrekturvorschläge. Danken möchte ich auch meinen Eltern und Großeltern für die moralische und finanzielle Unterstützung während meines gesamten Studiums.

Ich erkläre hiermit an Eides statt, diese Arbeit selbstständig verfaßt und keine Quellen und Hilfsmittel außer den angegebenen verwendet zu haben.

Saarbrücken, den 25. Februar 2000

Tim Priesnitz

Zusammenfassung

Teiltyp-Entailment ist die Frage, ob eine Implikation zwischen Teiltyp-Constraints $t_1 \leq t_2$ gilt. Algorithmen, die diese Frage lösen, sind für die Vereinfachung von Teiltyp-Constraints relevant. Die Vereinfachung von Teiltyp-Constraints ist hingegen in contraintbasierten Typ-Sytemen dringend erforderlich. Die Komplexität von Teiltyp-Entailment hängt von der gewählten Typsprache ab; schon für ausdrucksschwache Typsprachen ist es überraschend schwierig, einen Algorithmus für Teiltyp-Entailment zu entwerfen. So ist Teiltyp-Entailment für einfache Typen coNP-vollständig und wird durch die Hinzunahme von rekursiven Typen PSPACE-vollständig (Henglein & Rehof, 1997, 1998). Entailment wird durch die Erweiterung um einen kleinsten und größten Typ nicht-strukturell. Henglein und Rehof (1998) haben bewiesen, daß nicht-strukturelles Teiltyp-Entailment, sowohl im einfachen wie auch im rekursiven Fall, PSPACE-schwer ist; einen vollständigen Algorithmus konnten sie allerdings nicht angeben. Es ist eine bekannte offene Frage, ob nicht-strukturelles Teiltyp-Entailment überhaupt entscheidbar ist.

Ausgehend von dieser Situation untersucht diese Arbeit, worin die Schwierigkeiten liegen. Wir isolieren ein Teilproblem von nicht-strukturellem Teiltyp-Entailment, von dem wir zeigen, daß es PSPACE-vollständig ist. Damit zeigen wir zum ersten Mal Entscheidbarkeit für ein nicht-triviales Fragment von nicht-strukturellem Entailment. Wir charakterisieren Teiltyp-Entailment in der Automaten-Theorie; dazu erweitern wir das Konzept der endlichen Automaten zu sogenannten P-Automaten, deren Eigenschaften wir systematisch analysieren. Im nächsten Schritt reduzieren wir nun Teiltyp-Entailment auf das Universalitätsproblem von eingeschränkten P-Automaten. Für unser ausgezeichnetes Fragment können wir uns in der Tat auf endliche Automaten zurückziehen, deren Universalitätsproblem PSPACE-vollständig ist.

Diese Arbeit basiert im wesentlichen auf Resultaten einer vorangegangen Konferenzveröffentlichung (Niehren & Priesnitz, 1999a). In einer anschließenden Untersuchung (Niehren & Priesnitz, 1999b) haben wir den hier vorgestellten Ansatz über das eingeschränkte Fragment hinaus ausgedehnt und auch eine umgekehrte Charakterisierung formuliert; die Entscheidbarkeit von Teiltyp-Entailment bleibt aber weiterhin offen.

Abstract

Entailment of subtype constraints was introduced for constraint simplification in subtype inference systems. Understanding the algorithmic properties of subtype entailment is relevant to the research field of subtype inference. The complexity of entailment depends on the chosen type language. Even for weak type languages designing an efficient algorithm for subtype entailment turned out to be surprisingly difficult. The situation was clarified by Rehof and Henglein who proved structural subtype entailment to be coNP-complete for simple types; when extended with recusive types it becomes PSPACE-complete. The presence of a least or greatest type renders subtyping non-structural. For non-structural subtype entailment of both simple and recursive types they proved PSPACE-hardness and conjectured PSPACE-completeness but failed in finding a complete algorithm. Whether non-structural subtype entailment is decidable is a prominent open problem.

In this work, we investigate the source of complications and isolate a natural subproblem of non-structural subtype entailment that we prove to be PSPACE-complete. This is the first decidability result for a nontrivial subproblem of non-structural subtype entailment. To this purpose we characterize subtype entailment in automata theory by introducing so called *P-automata* which extend finite automata. We then reduce non-structural subtype entailment to the universality problem of P-automata. For a subproblem it is sufficient to reduce entailment to the universality problem of finite automata which is known to be PSPACE-complete. We conjecture (but this is left open) that the presented approach can be extended to the general case.

Inhaltsverzeichnis

Kapitel 1

Einführung

Wir geben zunächst eine Einführung in die Teiltypisierung und der Simplifizierung von Typ-Constraints. Anschließend präsentieren wir die Resultate dieser Arbeit und diskutieren ihre Relevanz. Hierbei gehen wir auch auf verwandte Arbeiten ein.

1.1 Statische Typisierung

Statische Typisierung ist ein wichtiger Punkt bei dem Entwurf vieler Programmiersprachen (siehe auch Fuh und Mishra (1990), Mitchell (1991), Amadio und Cardelli (1993), Eifrig, Smith, und Trifonov (1995), Pottier (1998b)). Zum einen garantiert sie in einem Programm die Abwesenheit von Typ-Fehlern, d. h. Invarianten werden bei der Programmausführung eingehalten; zum anderen gibt sie dem Programmierer ein Gerüst, welches den Programm-Entwurf unterstützt. Für den Programmierer ist die Typ-Annotation seiner Programm-Strukturen aufwendig, eine automatische Typisierung durch ein sogenanntes Typ-Inferenz-System ist wünschenswert (wie beispielsweise in der Programmiersprache ML realisiert). Die Menge aller möglichen Typen ist durch eine Typ-Sprache gegeben. Unterschiedliche Typ-Sprachen sind als Basis für Typ-Inferenz-Systeme untersucht worden. Hierbei wird der Entwurf der zugrundeliegenden Typ-Sprache hauptsächlich durch die folgende Abwägung bestimmt: Eine möglichst hohe Ausdrucksstärke der Typ-Sprache geht mit einer gleichzeitig hohen Laufzeit des entsprechenden Typ-Inferenz-Systems einher.

1.2 Teiltypisierung

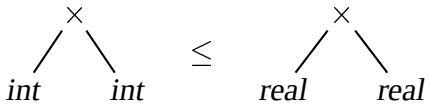
Teiltypisierung wird zur Erhöhung der Ausdrucksstärke von Typ-Sprachen benutzt. Sie wurde unabhängig von verschiedenen Autoren eingeführt (Mitchell, 1984; Cardelli, 1988). Klassische Typ-Sprachen basieren oft auf einer Typ-Gleichheit; eine Funktions-Anwendung ist genau dann erlaubt, wenn der aktuelle Argument-Typ τ_1 mit dem geforderten Argument-Typ τ_2 übereinstimmt, d. h. $\tau_1 = \tau_2$. Diese Forderung

ist unnötig stark. Der aktuelle Argument-Typ τ_1 kann von dem geforderten Argument-Typ τ_2 abweichen, falls τ_1 die operationalen Möglichkeiten von τ_2 subsumiert, d. h. alle Operationen für τ_2 sind auch für τ_1 definiert. Nach dieser Betrachtung ist die folgende Bedingung für eine typ-gerechte Funktions-Anwendung hinreichend: Der aktuelle Argument-Typ τ_1 ist ein Teiltyp von dem geforderten Argument-Typ τ_2; hierbei setzen wir die folgende Definition voraus: Wir nennen einen Typ τ_1 einen Teiltyp vom Typ τ_2, falls $\tau_1 \leq \tau_2$ bezüglich einer gegebenen Teiltyp-Relation $\leq$ gilt. Eine Teiltyp-Relation $\leq$ muß die folgende Ersetzbarkeits-Eigenschaft erfüllen.

Ersetzbarkeits-Eigenschaft. Falls der Ausdruck e den Typ τ_1 besitzt und τ_1 ein Teiltyp von τ_2 ist, dann kann sich e auch wie ein Ausdruck vom Typ τ_2 verhalten (bzgl. einer gegebenen Semantik der Programmiersprache), e besitzt somit auch den Typ τ_2.

Wir betrachten zwei Beispiele für eine Teiltypisierung:

- Durch Teiltypisierung können wir Beziehungen zwischen atomaren Typen beschreiben. So kann zum Beispiel die Menge der Zahlen vom Typ int als eine Teilmenge der Zahlen vom Typ $real$ angesehen werden (diese Betrachtung ist problematisch, da $real$ die rationalen Zahlen nur approximiert). Ein Ausdruck vom Typ int kann überall, wo ein Ausdruck vom Typ $real$ gefordert wird, benutzt werden (Ersetzbarkeits-Eigenschaft); wir setzen $int \leq real$. Diese Ordnung kann auf Paar-Typen erweitert werden: Auch ein Ausdruck vom Typ $int \times int$ kann überall, wo ein Ausdruck vom Typ $real \times real$ gefordert wird, benutzt werden. Stellen wir diese Paar-Typen durch Bäume dar, ergibt sich folgende Ordnung:



- Wir betrachten nun Teiltypisierung auf Rekord-Typen. Die Teiltyp-Ordnung auf Rekord-Typen erlaubt das Konzept des „Vergessens“, hiermit ist das Fehlen von einem oder mehreren Feldern in einem Rekord gemeint. So ist beispielsweise der Rekord-Typ $\{x : int,\ y : int,\ c : color\}$ eines farbigen Punktes ein Teiltyp von dem Rekord-Typ $\{x : int,\ y : int\}$ eines normalen Punktes. Die Ersetzbarkeits-Eigenschaft wird hierbei eingehalten: Ein farbiger Punkt kann offensichtlich überall, wo nur ein normaler Punkt gefordert wird, benutzt werden.

  Rekords mit dieser Teiltyp-Ordnung stellen eine mögliche Basis für objektorientierte Systeme dar.

1.3 Simplifizierung von Typ-Constraints

Typen stellen wir durch sogenannte Typ-Constraints dar. Der Simplifizierung von Typ-Constraints wird eine dominante Rolle bei der Typ-Inferenz zugeschrieben (Eifrig, Smith, & Trifonow, 1995; Rehof, 1997; Pottier, 1998a). Zum einen wächst bei der Teiltyp-Inferenz die Größe der Typ-Constraints schnell an, so daß für eine angemessene Laufzeit eine Simplifizierung der Typ-Constraints unvermeidbar wird. Zum anderen kann der entstehende Typ des gesamten Progamms nicht mehr durch Basis-Typen, sondern nur noch durch Typ-Constraints ausgedrückt werden. Ist dieser entstehende Typ-Constraint zu groß (und somit zu komplex), kann das Ergebnis der Typ-Inferenz vom Programmier nicht mehr nachvollzogen werden. Die Simplifizierung von Typ-Constraints wirkt beiden Phänomenen entgegen.

Teiltyp-Entailment wurde zur Typ-Simplifizierung eingeführt. Teiltyp-Entailment ist das Problem, ob eine Entailment-Aussage, bestehend aus einer Implikation zwischen zwei Teiltyp-Constraints, logisch gültig ist. Hiermit kann ein potentieller Simplifikations-Schritt geprüft werden. So betrachtet Rehof (1998) das Erfüllbarkeitsproblem von Teiltyp-Constraints als das Schlüsselproblem beim Entscheiden, ob ein Programm typ-korrekt ist, wogegen er Teiltyp-Entailment als das Schlüsselproblem der Typ-Darstellung ansieht.

Eifrig et al. (1995, 1995), Pottier (1996) haben Teiltyp-Entailment zur Typ-Simplifizierung in ihren Teiltyp-Inferenz-Systemen benutzt. Henglein und Rehof (1997, 1998) haben daraufhin gezeigt, daß Teiltyp-Entailment selbst einfachster Typ-Sprachen nicht mehr handhabbar ist, worauf Pottier (1998b) und Trifonov und Smith (1996) ihr Typ-Inferenz-System nur noch auf einer Approximation von Teiltyp-Entailment aufbauen. Diese arbeitet unvollständig, ist dafür aber praktikabel. Schwerpunkt ihres neuen Simplifizierungs-Algorithmus ist die Darstellung der Typ-Constraints durch sogenannte Constraint-Automaten. Auf dieser Darstellung agieren ein effizienter Speicherplatz-Bereinigungs-Algorithmus und ein Minimalisierung-Algorithmus für endliche Automaten.

Wir listen die wichtigsten Argumente für Teiltyp-Entailment auf.

- Teiltyp-Entailment kann zur Vereinfachung von Teiltyp-Constraints herangezogen werden (es ist in diesem Kontext aufgeworfen worden). Es kann damit ein zentrales Problem von Typ-Inferenz-Systemen lösen, es eignet sich allerdings nicht als Bestandteil eines praktisch umgesetzten Typ-Inferenz-Systems.

- Es kann zu einer elementaren Theorie über den Strukturen von geordneten Bäumen beitragen.

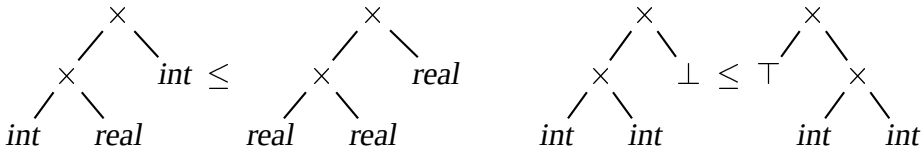
- Es ist ein mathematisch interessantes Problem, da es eine kurze und natürliche Formulierung besitzt, aber trotzdem bislang allen Lösungsversuchen getrotzt hat; es wurde schon in zahlreichen Arbeiten untersucht (Eifrig et al., 1995; Henglein & Rehof, 1997; Pottier, 1998a; Henglein & Rehof, 1998). Es

ist immer noch eine offene Frage, ob Teiltyp-Entailment im sogenannten nicht-strukturellen Fall entscheidbar ist.

1.4 Typ-Sprache dieser Arbeit

Im Rahmen dieser Arbeit untersuchen wir die Komplexität von Teiltyp-Entailment einer einfachen Typ-Sprache. Diese besteht aus den Teiltyp-Constraints der Form $\varphi, \psi \ ::= \ t_1 \leq t_2 \ \mid \ \varphi \wedge \psi$, wobei t_1 und t_2 Terme einer gegebenen Signatur Σ sind. Die Signatur Σ besitzt Konstanten (Basis-Typen wie zum Beispiel int oder $real$) und Konstruktoren (zum Beispiel den Paar-Konstruktor $\times$). Besitzt Σ zusätzlich einen kleinsten $\bot$ oder einen größten Typ $\top$ nennen wir die Typ-Sprache *nicht-strukturell*, ansonsten nennen wir sie *strukturell*.

Die gegebene Teiltyp-Ordnung $\leq$ auf Terme t_1, t_2, t_1', t_2' wird wie folgt interpretiert: Falls $t_1 \leq t_2$ und $t_1' \leq t_2'$ gilt, dann gilt auch $t_1 \times t_1' \leq t_2 \times t_2'$ (n-stellige Konstruktoren werden äquivalent behandelt). Die Konstante $\bot$ ist kleiner und die Konstante $\top$ ist größer als jeder andere Term. Die Ordnung zwischen anderen Konstanten wird einfach vorgegeben, beispielsweise $int \leq real$. Wir betrachten ein Beispiel einer strukturellen Ordnung (links) versus einer nicht-strukturellen Ordnung (rechts), wobei wir Typen durch Bäume (über Σ) darstellen.



Vergleichen wir Bäume mittels der Teiltyp-Ordnung, so können wir auch unendliche Bäume miteinander vergleichen. Wir sprechen von *einfachen Typen* im Falle von endlichen Bäumen und *rekursiven Typen* im Falle von potentiell unendlichen Bäumen.

Viele komplexer aufgebaute Typen stehen im Interesse von getypten Programmiersprachen, wie contra-variante Funktionstypen $\tau \to \tau'$, Rekord-Typen $\{f_1 : \tau_1, \ldots, f_n : \tau_n\}$, Mengen-Typen $a \cup b$ oder polymorphe Typen $\forall x.\ x \to x$. Um Teiltyp-Entailment einfach zu halten, beschränken wir uns auf einfache und rekursive Typen.

1.5 Verwandte Arbeiten

Trifonov und Smith (1996) und Pottier (1996) haben Algorithmen für nicht-strukturelles Teiltyp-Entailment angegeben, die sich aber alle als unvollständig herausstellten. Die Komplexität dieses Problems wurde anscheinend unterschätzt. (Rehof, 1997; Henglein & Rehof, 1997, 1998) haben die Komplexität des strukturellen Falls vollständig exploriert. So haben sie den nicht-rekursiven Fall 1997 als coNP-vollständig, den rekursiven Fall 1998 als PSPACE-vollständig bewiesen. Für den nicht-strukturellen Fall haben sie nur eine untere PSPACE-Schranke gefunden, wir wiederholen diesen Beweis in Kapitel 7.

Kozen, Palsberg, und Schwartzbach (1995) haben das sehr viel einfachere Typ-Inklusions-Problem — gegeben seien zwei Typen τ_1, τ_2 einer einfachen Typ-Sprache, gilt $\tau_1 \leq \tau_2$ — ähnlich zu unserem Vorgehen mit Hilfe der Automaten-Theorie gelöst.

In dieser Arbeit betrachten wir ausschließlich die Constraint-Sprache der nicht-strukturellen Teiltyp-Constraints; hingegen untersuchen zahlreiche Arbeiten das Entailment-Problem anderer Constraint-Sprachen. Allen diesen Sprachen ist eine Interpretation über Bäumen gemeinsam. So datieren Müller, Niehren, und Treinen (1998), Niehren, Müller, und Talbot (1999) zwei Entailment-Probleme als PSPACE-vollständig. Im Gegensatz zu unserem Entailment-Problem hängen diese beiden Entailment-Probleme nur von Eigenschaften regulärer Wort-Mengen des assoziierten Constraint-Graphen ab; die Eigenschaften können also durch endliche Automaten charakterisiert werden. Hingegen werden wir den Begriff des endlichen Automaten erweitern, um geeignet nicht-reguläre Eigenschaften charakterisieren zu können. Wir betrachten diese beiden Entailment-Probleme genauer:

- In der Feature-Logik — eine Logik, um Feature-Bäume (oder Rekord-Typen) zu untersuchen — sind viele Constraint-Sprachen im Hinblick auf ihr Entailment-Problem untersucht worden. So zeigen Ait-Kaci, Podelski, und Smolka (1994), Smolka und Treinen (1994), daß Entailment für Gleichheits-Constraints über Feature-Bäume in quasi linearer Zeit entschieden werden kann. Schwächt man die Gleichheits-Constraints durch Ordnungs-Constraints ab, ist ein kubischer Zeitaufwand vonnöten (Müller, Niehren, & Podelski, 2000). Diese Ordnungs-Constraints über Feature-Bäume (Dörre & Rounds, 1990; Dörre, 1991) können als Rekord-Teiltyp-Constraints angesehen werden. Entailment für Ordnungs-Constraints mit zusätzlichen existenz-quantifizierten Variablen ist PSPACE-vollständig (Niehren et al., 1999).

- Das Entailment-Problem ist auch für Mengen-Constraints über Mengen von endlichen Bäumen untersucht worden (Mengen-Typen werden im allgemeinen durch Mengen-Constraints dargestellt). So zeigen Charatonik und Podelski (1997), daß Entailment von Mengen-Constraints mit einem Schnitt-Operator über einer unendlichen Signatur DEXPTIME-vollständig ist. Niehren et al. (1999) zeigen, daß Entailment von atomaren Mengen-Constraints im Fall einer unendlichen Signatur PSPACE-vollständig und im Fall einer endlichen Signatur DEXPTIME-hart ist.

1.6 Beitrag dieser Arbeit

Diese Arbeit untersucht, worin die Schwierigkeiten beim Teiltyp-Entailment liegen. Wir isolieren ein Teilproblem von nicht-strukturellem Teiltyp-Entailment, von dem wir zeigen, daß es PSPACE-vollständig ist. Damit zeigen wir zum ersten Mal Entscheidbarkeit für ein nicht-triviales Fragment von nicht-strukturellem Entailment. In dem

hier vorgestellten Ansatz charakterisieren wir Teiltyp-Entailment in der Automaten-Theorie; dazu erweitern wir das Konzept der endlichen Automaten zu sogenannten P-Automaten, deren Eigenschaften wir systematisch analysieren. Im nächsten Schritt reduzieren wir nun Teiltyp-Entailment auf das Universalitätsproblem von eingeschränkten P-Automaten. Für unser ausgezeichnetes Fragment können wir uns in der Tat auf endliche Automaten zurückziehen, deren Universalitätsproblem PSPACE-vollständig ist.

Wir fassen die Hauptresultate dieser Arbeit in zwei Punkten zusammen.

- Ein Fragment des Teiltyp-Entailment kann durch Universalität eingeschränkter P-Automaten charakterisiert werden.
- Die Universalität von diesen eingeschränkten P-Automaten ist PSPACE-vollständig, obwohl P-Automaten auch nicht-kontext-freie Sprachen erkennen können.

Beide Resultate sind neu und nicht trivial, ihre Beweise werden in dieser Arbeit ausführlich exploriert.

Diese Arbeit basiert im wesentlichen auf Resultaten einer vorangegangen Konferenzveröffentlichung (Niehren & Priesnitz, 1999a). In einer anschließenden Untersuchung (Niehren & Priesnitz, 1999b) haben wir den hier vorgestellten Ansatz über das eingeschränkte Fragment hinaus ausgedehnt und auch eine umgekehrte Charakterisierung formuliert: Wir reduzieren das Universalitätsproblem von ausgezeichneten P-Automaten auf Teiltyp-Entailment. Die Entscheidbarkeit von Teiltyp-Entailment bleibt aber weiterhin offen. Wir geben eine ausführliche Zusammenfassung dieser Untersuchung in Kapitel 8.

1.7 Vorgehensweise

Zuerst geben wir in Kapitel 2 eine Definition der grundlegenden mathematischen Begriffe und erläutern die Einschränkung unseres Fragmentes. Im Kapitel 3 führen wir den Begriff der P-Automaten ein und untersuchen deren Eigenschaften. Mit dieser erweiterten Automatentheorie können wir in Kapitel 4 einen Algorithmus für eingeschränktes nicht-strukturelles Teiltyp-Entailment angeben. Dieser Algorithmus besteht aus einem bekannten Erfüllbarkeitstest von Teiltyp-Constraints und einem Universalitätstest von eingeschränkten P-Automaten. In Kapitel 5 und 6 beweisen wir, daß unser Entailment-Algorithmus korrekt und widerlegungs-vollständig ist, in Kapitel 7 leiten wir seine Komplexitäts-Klasse PSPACE ab. Anschließend diskutieren wir die Konklusion dieser Arbeit und geben einen kurzen Ausblick auf ein ausdrucksstärkeres Fragment von nicht-strukturellem Teiltyp-Entailment (Kapitel 8). Im Anhang B geben wir noch einmal eine Zusammenfassung aller besprochenen und einiger neuen Entailment-Beispiele. Desweiteren befinden sich im Anhang C ausgelagerte Beweise, die zwar umfangreich, jedoch von einfacher Natur sind.

Kapitel 2

Nicht-strukturelle Teiltyp-Constraints

2.1 Grundlagen

Als grundlegende Datenstruktur dieser Arbeit benutzen wir endliche und unendliche Bäume [1]. Bäume definieren wir über einer endlichen Signatur Σ mit mindestens einem Funktionssymbol. Wir verwenden f, g, h für Funktionssymbole aus Σ und bezeichnen die Stelligkeit von f mit $\mathsf{ar}(f)$. Konstanten, d. h. Funktionssymbole f mit Stelligkeit $\mathsf{ar}(f) \geq 0$, bezeichnen wir mit a, b, ... Die gegebene Signatur Σ definiert zusätzlich eine partielle Ordnung auf ihren Konstanten. Desweiteren sind $\bot$ und $\top$ zwei ausgezeichnete Konstanten. Wir nehmen außerdem eine unendliche Menge von Variablen V an, die wir mit den Zeichen x, y, z, u, v, w bezeichnen.

Wir definieren einen Baum τ durch seine Domäne D_τ und eine Labelfunktion L_τ. Die Domäne D_τ ist eine Menge aller Pfade von τ; die Labelfunktion L_τ ordnet jedem Pfad aus der Domäne D_τ ein Label aus Σ zu. Wir betrachten nur Stelligkeits-Konsistente Bäume, das sind Bäume, die genau dann einen Pfad πi in D_τ besitzen, wenn auch π in D_τ ist und zusätzlich die Bedingung $i \leq \mathsf{ar}(L_\tau(\pi))$ gilt. Wir denotieren die Menge aller endlichen Bäume mit $\mathit{Baum}_\Sigma^{fin}$ und die Menge aller potentiell unendlichen Bäume mit Baum_Σ.

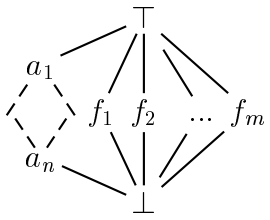
Wir definieren nun eine partielle Ordnung über Bäumen aus Baum_Σ. Hierzu definieren wir zunächst eine Ordnung über Labels und erweitern diese anschließend auf Bäume.

Definition 1 (Ordnung über Labels). Sei eine beliebige Signatur Σ mit einer partiellen Ordnung $\leq$ auf ihren Konstanten gegeben, dann ist die partielle Ordnung $\leq_L$ über Funktionssymbolen aus Σ minimal mit den folgenden Eigenschaften, welche für alle Funktionssymbole f aus Σ definiert sind.

[1] Für eine präzise Definition der grundlegenden mathematischen Begriffe verweisen wir auf den Anhang A.

Konstanten.	Für alle $a, b \in \Sigma$ gilt $a \leq_L b$, falls $\Sigma\ a \leq b$ unterstützt.
Reflexivität.	Für alle $f \in \Sigma$ gilt $f \leq_L f$.
kleinstes Element.	Für alle $f \in \Sigma$ gilt $\bot \leq_L f$, falls $\bot \in \Sigma$.
größtes Element.	Für alle $f \in \Sigma$ gilt $f \leq_L \top$, falls $\top \in \Sigma$.

Somit ist $\bot$ kleiner und $\top$ größer als jedes andere Symbol der Signatur Σ, ansonsten sind alle Symbole, außer den Konstanten aus Σ, untereinander unvergleichbar. Die rechte Abbildung zeigt die Labelordnung $\leq_L$ für eine beliebige Signatur $\Sigma = \{\bot, \top, a_1, \ldots, a_n, f_1, f_2, \ldots, f_m\}$ mit einer beliebigen Ordnung auf ihren Konstanten $a_1, \ldots, a_n$.



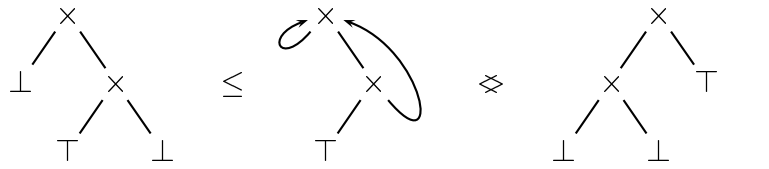
Definition 2 (Ordnung über Bäumen). Sei eine beliebige Signatur Σ mit einer Ordnung auf Konstanten und einer Ordnung $\leq_L$ nach Definition 1 gegeben. Die partielle Ordnung $\leq$ über Bäumen τ_1, τ_2 aus Baum_Σ (beziehungsweise $\mathit{Baum}_\Sigma^{\mathit{fin}}$) ist gegeben durch:

> Es gilt $\tau_1 \leq \tau_2$ genau dann, wenn für alle Pfade $\pi \in D_{\tau_1} \cap D_{\tau_2}$ die Labelordnung $L_{\tau_1}(\pi) \leq_L L_{\tau_2}(\pi)$ gilt.

Falls die ausgezeichneten Konstanten $\bot$ oder $\top$ in der Signatur Σ vorkommt, heißt die Ordnung $\leq$ *nicht-strukturell*, ansonsten heißt sie *strukturell*. Die gegebene Ordnung $\leq$ impliziert, daß im strukturellen Fall nur Bäume mit einer gleichen Form verglichen werden können. Im Gegensatz können im nicht-strukturellen Fall auch Bäume mit einer unterschiedlichen Form verglichen werden, wie das nachfolgende Beispiel zeigt.

Wir denotieren endliche Bäume aus $\mathit{Baum}_\Sigma^{\mathit{fin}}$ durch Grundterme aus Σ und unendliche Bäume aus Baum_Σ durch eine μ-Notation über Konstruktortermen aus Σ (μ-Terme).

Beispiel 1. Wir betrachten Bäume über der nicht-strukturellen Signatur $\Sigma = \{\bot, \top, \times\}$. Im folgenden benutzen wir die Relation $\not\leq$ für unvergleichbare Bäume, d. h. $\tau_1 \not\leq \tau_2 \ :\Leftrightarrow\ \neg\tau_1 \leq \tau_2 \wedge \neg\tau_2 \leq \tau_1$. Es gilt die folgende nicht-strukturelle Ordnung für die Bäume τ_1, τ_2 und τ_3:



$\tau_1 : \bot \times (\top \times \bot)$ $\quad$ $\tau_2 : \mu x.x \times (\top \times x)$ $\quad$ $\tau_3 : (\bot \times \bot) \times \top$

Wir beweisen $\tau_2 \not\leq \tau_3$ durch $L_{\tau_2}(2) \leq_L L_{\tau_3}(2)$ und $L_{\tau_3}(11) \leq_L L_{\tau_2}(11)$. Hingegen gilt $\tau_1 \leq \tau_2$, da für alle Pfade $\pi \in D_{\tau_1} \cap D_{\tau_2} = \{\varepsilon, 1, 2, 21, 22\}$ die Labelordnung $L_{\tau_1}(\pi) \leq_L L_{\tau_2}(\pi)$ gilt.

In dieser Arbeit betrachten wir nur eine Ordnung über Bäumen, die nicht-strukturell ist. Diese Ordnung stellt die Grundlage der Interpretation der folgenden Sprache dar.

Definition 3 (Sprache NS_Σ der nicht-strukturellen Teiltyp-Constraints über Σ). Gegeben sei eine Signatur Σ, die $\bot$, $\top$ und mindestens einen weiteren Konstruktor f unterstützt und eine Menge V der Variablen. Die Sprache NS_Σ definiert über der Signatur $\Sigma \cup \{\leq\}$ besteht aus den *flachen Teiltyp-Constraints*

$$\psi ::= x{=}f(x_1, \ldots, x_n) \mid x{=}\top \mid y{=}\bot \mid x{\leq}y \mid \psi \wedge \psi' \qquad (f \in \Sigma \text{ mit } \mathsf{ar}(f) = n).$$

In der Sprache NS_Σ interpretieren wir

- Variablen in der Domäne der potentiell unendlichen Bäume $Baum_\Sigma$,
- das Zeichen $\leq$ als nicht-strukturelle Teiltypisierung und
- Funktionssymbole $f \in \Sigma$ als Baumkonstruktoren.

Analog definieren wir die Sprache NS_Σ^{fin} mit der Ausnahme, daß Variablen nur in der Domäne der endlichen Bäume $Baum_\Sigma^{fin}$ interpretiert werden.

Die Sprachen NS_Σ und NS_Σ^{fin} besitzen die Einschränkung, daß sie nur flache Teiltyp-Constraints zulassen. In flachen Teiltyp-Constraints gibt es im Gegensatz zu tiefen Teiltyp-Constraint keine Terme, die aus geschachtelten Konstruktoren aufgebaut sind. Die Ausdrucksstärke der Sprachen NS_Σ und NS_Σ^{fin} wird durch diese Einschränkung nicht herabgesetzt, da wir zu jedem tiefen Teiltyp-Constraint φ einen logisch äquivalenten flachen Teiltyp-Constraints in Polynomialzeit konstruieren können: Sei f ein beliebiger Konstruktor aus φ. Für alle Terme t, die in φ in einem Term der Form $f(\ldots, t, \ldots)$ auftreten, erweitern wir φ um einen neuen Constraint $u_t{=}t$ (u_t ist eine neue Variable). Anschließend flachen wir φ ab, indem wir alle Terme der Form $f(\ldots, t, \ldots)$ durch $f(\ldots, u_t, \ldots)$ ersetzen. Diesen Algorithmus führen wir induktiv fort, bis alle Teiltyp-Constraints in φ vollständig abgeflacht sind.

Gegeben sei die Constraint-Sprache NS_Σ (entsprechend NS_Σ^{fin}) mit zwei Teiltyp-Constraints φ und ψ. Eine *Entailment-Aussage* $\varphi \models \psi$ gilt in NS_Σ, wenn alle Belegungen, die den Constraint φ in NS_Σ erfüllen, auch den Constraint ψ in NS_Σ erfüllen, d. h. $\forall \alpha : NS_\Sigma\ \alpha \models \varphi \implies NS_\Sigma\ \alpha \models \psi$. Ohne Beschränkung der Allgemeinheit betrachten wir nur Entailment-Aussagen der Form $\varphi \models x{\leq}y$ mit $x, y \in V(\varphi)$, da wir jede beliebige Entailment-Aussage in mehrere Entailment-Aussagen der Form $\varphi \models t_1{\leq}t_2$ zerlegen können (t_1 und t_2 sind flache Terme). Eine Entailment-Aussage der Form $\varphi \models t_1{\leq}t_2$ ist wiederum äquivalent zu $\varphi \wedge t_1{\leq}x \wedge y{\leq}t_2 \models x{\leq}y$ (x und y sind neue Variablen).

Beispiel 2. Wir betrachten die folgenden Entailment-Aussagen über einer Signatur Σ mit einem zweistelligen Funktionssymbol $\times$.

- A_1 : $x{\leq}z \wedge z{\leq}y \models x{\leq}y$ gilt in den Sprachen NS_Σ und NS_Σ^{fin}.
- A_2 : $x{\leq}x{\times}x \wedge y{\times}y{\leq}y \models x{\leq}y$ gilt in NS_Σ und NS_Σ^{fin}.
- A_3 : $x{\leq}y{\times}y \wedge x{\times}x{\leq}y \models x{\leq}y$ gilt nicht in NS_Σ und NS_Σ^{fin}.
- A_4 : $x{\leq}y{\times}x \wedge z{\times}x{\leq}y \wedge z{\times}\top{\leq}z \models x{\leq}y$ gilt in NS_Σ und NS_Σ^{fin}.

Die Entailment-Aussage A_1 gilt, da die Transitivität von $\leq$ offensichtlich ist. Die Gültigkeit von Aussage A_2 und A_4 ist hingegen keineswegs offensichtlich. Der Constraint $x{\leq}x{\times}x$ aus Aussage A_2 impliziert, daß alle Labels aus dem Belegungsbaum von x gleich dem Konstruktor $\bot$ oder $\times$ sind. Analog impliziert der Constraint $y{\times}y{\leq}y$, daß alle Labels aus der Belegungsbaum von y gleich dem Konstruktor $\times$ oder $\top$ sind. In der Aussage A_2 implizieren somit die beiden gerade genannten Constraints $x{\leq}y$. Die Entailment-Aussage A_3 wird einfach durch die Belegung x nach $\bot{\times}(\top{\times}\bot)$ und y nach $\top{\times}(\bot{\times}\top)$ widerlegt. Allen Entailment-Aussagen A_1 bis A_3 ist gemeinsam, daß sie syntaktisch kein $\bot$ oder $\top$ enthalten.

Wir werden in dieser Arbeit ein Verfahren angeben, das Entailment-Aussagen dieser Problemklasse, Kontanten $\bot$ und $\top$ treten syntaktisch nicht auf, verifiziert oder widerlegt. Hierzu führen wir eine Erweiterung des Begriffs des endlichen Automaten, den sogenannten P-Automaten, ein. Im nächsten Schritt reduzieren wir nun Teiltyp-Entailment dieser Problemklasse auf das Universalitätsproblem von speziellen P-Automaten. Die Entailment-Aussage A_4 geht aber über diese Problemklasse hinaus, da sie syntaktisch eine Konstante $\top$ besitzt. Wir skizzieren den Beweis der Gültigkeit von A_4 in Kapitel 8. Eine Analyse dieser um $\bot$ und $\top$ erweiterten Problemklasse befindet sich in unserer nachfolgenden Untersuchung (Niehren & Priesnitz, 1999b). Auch in dieser Untersuchung ist die Grundlage unserer Entailmentanalyse eine ausgezeichnete Klasse von P-Automaten. In dem Anhang B (Entailment-Beispiele) befindet sich eine weitergehende Analyse der Entailment-Aussagen A_3 und A_4.

2.2 Einschränkungen des Entailmentproblems

Wir müssen zunächst das grundlegende Fragment der Spachen NS_Σ und NS_Σ^{fin} definieren, auf welches wir unsere späteren Komplexitäts- und Entscheidbarkeitsresultate anwenden können:

Definition 4 (eingeschränkte Sprachen NS_n und NS_n^{fin}). Gegeben sei eine Signatur Σ, die genau aus $\bot$, $\top$ und einem Konstruktor f der Stelligkeit n besteht und eine Menge V der Variablen. Die Sprache NS_n definiert über der Signatur $\{\bot, \top, f, \leq\}$ besteht aus den *flachen Teiltyp-Constraints*

$$\psi \ ::= \ x{=}f(x_1, \ldots, x_n) \mid x{\leq}y \mid \psi \wedge \psi'.$$

Die Interpretation (Semantik) der Sprachen NS_n und NS_n^{fin} ist identisch mit der Interpretation der Sprache NS_Σ beziehungsweise NS_Σ^{fin} (gegeben in Definition 3).

Wir vergleichen unsere allgemeinen Constraintsprachen (Definition 3) mit den jetzt gegebenen eingeschränkten Constraintsprachen (Definition 4). Im folgenden sei $\times$ ein binäres und g ein unäres Funktionssymbol, hingegen benutzen wir f und h für Funktionssymbole beliebiger Stelligkeit. Wir finden in den eingeschränkten Sprache die folgenden Einschränkungen:

- *Die Syntax der eingeschränkten Sprachen erlaubt keine Constraints der Bauart $x{=}\bot$ oder $x{=}\top$.*

  Wir können syntaktisch ein $\bot$ oder $\top$ zwar nicht ausdrücken, wohl aber semantisch darf die Belegung einer Variable auf $\bot$ oder $\top$ gesetzt werden. Durch diese Einschränkung erhalten wir als Grundlage für unseren Entailment-Test spezielle P-Automaten; die Sprache ihres zugrundeliegenden endlichen Automaten ist präfix-abgeschlosssen. Diese Eigenschaft ist entscheidend für unsere Komplexitätsresultate der P-Automaten.

- *Die Signatur der eingeschränkten Sprachen besitzt gegenüber den beiden Konstanten $\bot$ und $\top$ genau ein weiteres Funktionssymbol.*

  Betrachten wir die allgemeine Sprache NS_Σ mit der Signatur $\Sigma = \{\bot, f, h\}$, so kann die Konstante $\bot$ auch indirekt durch die beiden zusätzlichen Konstruktoren f und h dargestellt werden. Entailment ist von dieser indirekt dargestellten Konstante abhängig. Wir betrachten hierzu die Entailment-Aussage

  $$x{\leq}f(\ldots) \wedge x{\leq}h(\ldots) \models x{\leq}z.$$

  Diese besitzt keine syntaktische $\bot$ Konstante. Trotzdem impliziert $x{\leq}f(\ldots) \wedge x{\leq}h(\ldots)$ den Constraint $x{=}\bot$ (ein indirekt dargestelltes $\bot$). Dieser Constraint impliziert wiederum für alle beliebigen Variablen z den Constraint $x{\leq}z$; die gegebene Entailment-Aussage gilt somit.

Wir benötigen noch eine weitere für uns wichtige Eigenschaft.

- *Wir betrachten nur einen zweistelligen Konstruktor $\times$, also die Sprachen NS_2 und NS_2^{fin}.*

  Der in dieser Arbeit vorgestellte Ansatz funktioniert zwar auch für höherstellige Konstruktoren, aber wir begnügen uns damit, den Ansatz für den zweistelligen Fall examplarisch zu beweisen. Wichtig ist hingegen, daß wir keinen einstelligen Konstruktor in der Signatur besitzen. Obwohl die Beispiele mit einem einstelligen Konstruktor ziemlich leicht aussehen — in diesem Fall gibt es für Entailment ein einfaches Entscheidungsverfahren — treten hier ähnliche Probleme auf wie im Falle von Entailment-Aussagen mit syntaktischen $\bot$ Konstanten.

Trotz dieser genannten Einschränkungen ist unser betrachtetes Sprachfragment NS_2, beziehungsweise NS_2^{fin} für den endlichen Fall, nicht trivial. Die von Henglein und Rehof (1998) gegebene untere PSPACE-Schranke für nicht-strukturelles Teiltyp-Entailment begnügt sich mit unseren eingeschräkten Sprachfragmenten NS_2 bzw. NS_2^{fin}:

Proposition 1 (eingeschränktes Entailment ist PSPACE-schwer). Nicht-strukturelles Teiltyp-Entailment ist sowohl für die eingeschränkte Sprache NS_2, als auch für NS_2^{fin} PSPACE-schwer.

Wir beweisen diese Proposition in Kapitel 7.

2.3 Erfüllbarkeitstest

In diesem Abschnitt wiederholen wir einen bekannten Erfüllbarkeitstest für nicht-strukturelle Teiltyp- Constraints. Dieser besteht aus einem Saturierungsalgorithmus mit einem Konsistenztest. Im Falle einer Interpretation über endlichen Bäumen erfolgt noch ein zusätzlicher Occur-Test.

Wir wiederholen einen bekannten Algorithmus von Henglein und Rehof (1998), welcher die Erfüllbarkeit eines nicht-strukturellen Teiltyp-Constraints über einer beliebigen Signatur in der Sprache NS_Σ oder NS_Σ^{fin} testet. Wir geben in der folgenden Tabelle die Eigenschaften *S0–S4* an.

S0	falls $x \in V(\psi)$ dann $x \leq x$ in ψ
S1	falls $x \leq y \wedge y \leq z$ in ψ dann $x \leq z$ in ψ
S2	falls $x = f(x_1, ..., x_n) \wedge x \leq y \wedge y = f(y_1, ..., y_n)$ in ψ dann $\bigwedge_{i=1}^{n} x_i \leq y_i$ in ψ
S3	nicht $x = f_1(x_1, ..., x_n) \wedge x \leq y \wedge y = f_2(y_1, ..., y_n)$ in $\psi,\ f_1 \not\leq_L f_2,\ n \geq 0$
S4	nicht $\bigwedge_{i=1}^{n} x_{i-1} = f(..., y_i, ...) \wedge y_i \leq x_i \wedge x_i \leq y_i$ in $\varphi,\ n \geq 1,\ x_0 = x_n$

Eigenschaften $S0 - S3$ für NS_Σ und $S0 - S4$ für NS_Σ^{fin}

Die Eigenschaften *S0-S2* beschreiben einen Abschluß unter Reflexivität, Transitivität und Absteigen, dieser kann auch als ein Saturierungsalgorithmus angesehen werden, welcher den Abschluß eines (flachen) Constraints in kubischer Zeit berechnet. Wir nennen einen (flachen) Constraint *abgeschlossen* falls er die Eigenschaften *S0–S2* erfüllt. Besteht ein (flacher) Constraint den Test *S3* (Konsistenz-Test), so heißt er *konsistent*, besteht er den Test *S4* (Occur-Test), so heißt er *zykelfrei*. Desweiteren nennen wir einen Constraint *fehlerfrei* unter NS_Σ, falls er konsistent ist und wir nennen ihn fehlerfrei unter NS_Σ^{fin}, falls er zusätzlich noch zykelfrei ist. Der Konsistenztest

unterbindet einen Labelfehler bezüglich der gegebenen Ordnung $\leq$, der zum Beispiel in dem Constraint $f(x,x) \leq h(x,x)$ auftritt. Der Occur-Test unterbindet unendliche Bäume in den Variablen-Denotationen, er akzeptiert zum Beispiel den Constraint $x=g(x)$ nicht, der offensichtlich in NS_Σ^{fin} nicht erfüllbar ist.

Proposition 2 (Erfüllbarkeit). Ein nicht-struktureller Teiltyp-Constraint φ ist genau dann in NS_Σ erfüllbar, wenn sein Abschluß konsistent ist. Der Constraint φ ist genau dann in NS_Σ^{fin} erfüllbar, wenn sein Abschluß konsistent und zykelfrei ist. Gegeben sei φ, dann kann der Konsistenz- und der Occur-Test von dem Abschluß von φ in kubischer Zeit berechnet werden.

Wir bemerken, daß ein unter NS_Σ^{fin} abgeschlossener Constraint automatisch auch unter NS_Σ abgeschlossen ist, die umgekehrte Schlußfolgerung jedoch nicht gilt.

Beispiel 3. Wir berechnen den Abschluß und die Erfüllbarkeit des Constraints φ_1:

$$x=h(y,x) \wedge x \leq y \wedge y=h(u,u) \wedge u \leq x$$

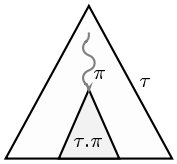
Durch Anwendung der Reflexivitätsregel *S0* fügen wir die trivialen Constraints $x \leq x \wedge y \leq y \wedge u \leq u$ hinzu. Als nächstes können wir durch Anwenden der Absteigeregel *S2* den Constraint $y \leq u \wedge x \leq u$ hinzufügen. Die Transitivitätsregel *S1* angewendet auf $y \leq u \wedge u \leq x$ ergibt den Constraint $y \leq x$. Jetzt können wir die Absteigeregel *S2* auf $y=h(u,u) \wedge y \leq x \wedge x=h(y,x)$ anwenden und erhalten $u \leq y \wedge u \leq x$. Der sich ergebende Constraint ist nun abgeschlossen (*S0–S2* leiten keine neuen Constraints mehr ab) und konsistent, somit ist φ_1 nach Proposition 2 erfüllbar über NS_Σ. Der Constraint φ_1 ist hingegen nicht erfüllbar über NS_Σ^{fin}, da der Occur-Test *S4* bedingt durch den Teilconstraint $x=h(y,x)$ in φ_3 fehlschlägt.

Wir merken an, daß Proposition 2 nicht im *strukturellen* Fall gilt. So ist zum Beispiel der Constraint $x \leq f(x,x) \wedge x \leq h(x,x) \wedge x \leq x$ abgeschlossen, konsistent und zykelfrei, aber über einer strukturell interpretierten Ordnung $\leq$ nicht erfüllbar.

2.4 Pfadconstraints

Wir führen Formeln ein, welche Eigenschaften eines Teilbaums an einem gegebenen Pfad ausdrücken. Diese sogenannten Pfadconstraints sind grundlegend zum Verständnis von Entailment vieler Constraintsprachen, (siehe auch Henglein und Rehof (1997, 1998), Rehof (1998), Müller et al. (1998), Müller et al. (2000), Niehren et al. (1999)).

2.4.1 Teilbaum-Constraints



Zunächst definieren wir einen Zugriffsoperator „ . “ auf Bäume τ. Hiermit greifen wir auf Teilbäume von τ an einem gegebenen Pfad π zu. Sei also τ ein Baum und $\pi \in D_\tau$, dann schreiben wir $\tau.\pi$ für den *Teilbaum von τ an π*, d. h. $D_{\tau.\pi} = \{\pi' \mid \pi\pi' \in D_\tau\}$ und $L_{\tau.\pi}(\pi') = L_\tau(\pi\pi')$ für alle $\pi' \in D_{\tau.\pi}$. Analog beschreibt der Term $x.\pi$ den Teilbaum der Denotation von x an π. Ein *Teilbaum-Constraint* $x.\pi{=}y$ verlangt, daß der Pfad π in der Domäne der Denotation von x ist und daß der Teilbaum $x.\pi$ gleich der Denotation von y ist.

2.4.2 Bedingte Teilbaum-Constraints

Bedingte Teilbaum-Constraints besitzen die Form $x?\sigma \leq_L y?\pi$. Das Fragezeichen betont, daß der Constraint abhängig von der Existenz der beiden Teilbäumen $x.\sigma$ und $y.\pi$ ist. Existiert einer der beiden Teilbäume nicht, so ist der obige Constraint automatisch erfüllt. Ansonsten gilt der Constraint genau dann, wenn das Label von $x.\sigma$ kleiner als das Label von $y.\pi$ ist. Wir geben nun seine exakte Semantik an: Ein *bedingter Teilbaum-Constraint* $x?\pi_1 \leq_L y?\pi_2$ ist genau dann von einer Variablebelegung α erfüllt, wenn aus $\pi_1 \in D_{\alpha(x)}$ und $\pi_2 \in D_{\alpha(y)}$ folgt, daß $L_{\alpha(x)}(\pi_1) \leq_L L_{\alpha(y)}(\pi_2)$. Die Bedingung $?\varepsilon$ lassen wir im folgenden einfach weg, da der Pfad ε in jedem Baum existiert. Ferner benutzen wir $x?\sigma \leq_L f$ anstelle von $\exists y(x?\sigma \leq_L y \wedge y{\leq}f(\top,\ldots,\top))$ und symmetrisch $f \leq_L x?\sigma$ anstelle von $\exists y(f(\bot,\ldots,\bot){\leq}y \wedge y \leq_L x?\sigma)$.

2.4.3 Charakterisierung von Entailment

Die nachfolgende Proposition 3 zeigt, daß bedingte Teilbaum-Constraints ausdrucksstark genug sind, um nicht-strukturelles Entailment zu charakterisieren.

Proposition 3 (Charakterisierung von Entailment). Gegeben sei eine beliebige Signatur Σ mit ihrer maximalen Stelligkeit m_Σ, d. h. m_Σ ist minimal unter der Eigenschaft $\forall f \in \Sigma:\ m_\Sigma \geq ar(f)$. Für alle Variablen u, v ist die folgende Äquivalenz in den Sprachen NS_Σ und NS_Σ^{fin} gültig:

$$u{\leq}v \ \leftrightarrow\ \bigwedge\{u?\pi \leq_L v?\pi \mid \pi \text{ ist ein Pfad aus } \{1,\ldots,m_\Sigma\}^*\}$$

Beweis. Die Implikationsrichtung von links nach rechts folgt direkt aus der Definition 2 der Ordnung über Bäumen und der vorangegangenen Definition der bedingten Teilbaum-Constraints. Es verbleibt die Gültigkeit von

$$u{\not\leq}v \ \rightarrow\ \exists\pi \in \{1,\ldots,m_\Sigma\}^*:\ u?\pi \not\leq_L v?\pi$$

zu zeigen. Sei also der Constraint $u{\not\leq}v$ gültig, das heißt, es gibt eine Belegung α, mit $\alpha(u) \not\leq \alpha(v)$. Wir betrachten nun den minimalen Pfad π mit den beiden Eigenschaften, π ist in den beiden Bäumen $\alpha(u)$ und $\alpha(v)$ definiert und es gilt $\alpha(u.\pi) \not\leq \alpha(v.\pi)$. Nach Definition 2 existiert dieser Pfad π immer. Aus diesen beiden Eigenschaften folgt automatisch die Gültigkeit des bedingten Teilbaum-Constraints $u?\pi \not\leq_L v?\pi$.

2.4.4 Weitere Vorgehensweise

Proposition 3 zeigt, daß eine Entailment-Aussage $\psi \models x \leq y$ genau dann gilt, falls für alle Pfade π der Constraint ψ den Constraint $x?\pi \leq_L y?\pi$ impliziert. Diese Eigenschaft ist zentral für die gesamte weitere Arbeit unserer Entailmentanalyse.

Wir nennen einen Pfad π einen *Widerspruchspfad* für ein gegebenes Entailmentproblem $\psi \models x \leq y$ dann und nur dann, falls ψ nicht $x?\pi \leq_L y?\pi$ impliziert. In dieser Terminologie gilt nach Proposition 3 eine Entailment-Aussage genau dann, wenn es hierfür keinen Widerlegungspfad gibt.

Wir werden in dem Kapitel 4 einen Automaten aus einem gegebenen Constraint φ bauen, der gerade alle Wörter π erkennt, für die der Teilbaum-Constraint $u?\pi \leq_L v?\pi$ gilt. Der Automat kann nicht exakt syntaktisch die erwähnten Teilbaum-Constraints erkennen, so daß wir im Kapitel 5, eine schwächere Form der bedingten Teilbaum-Constraints einführen, die aber die Sprache des Automaten exakt beschreibt.

Kapitel 3

Theorie der P-Automaten

Der Begriff des P-Automaten erweitert den klassischen Begriff des endlichen Automaten: Gegeben sei die Sprache L eines endlichen Automaten, dann kann man einen P-Automaten konstruieren, der die Möglichkeit besitzt, ein Wort aus L mit einem beliebig oft konkatenierten Suffix aus diesem Wort zu erkennen. Diese P-Automaten stellen die Grundlage unserer Entailmentanalyse dar.

3.1 Definition

In diesem Kapitel präsentieren wir die Notation und Eigenschaften der *P-Automaten*, die die Grundlage unserer Entailmentanalyse darstellen. Der Begriff des P-Automaten ist eine echte Erweiterung des klassischen Begriffs des endlichen Automaten, er kann somit auch nicht-reguläre Sprachen erkennen.

In der Literatur werden zahlreiche Erweiterungen von endlichen Automaten diskutiert. So charakterisiert zum Beispiel Büchi (1964) einen endlichen Automaten als ein Eingabeband mit einem pop-Operator und erweitert dieses Modell um einen zusätzlichen push-Operator (gegenüber einem Kellerautomaten agieren beide Operatoren auf dem Eingabeband). Eine weitere Erweiterung stellen die sogenannten *alternierenden* endlichen Automaten dar; von einem alternierenden Knoten muß nicht ein Pfad, sondern alle erreichbaren Pfade von dem Automaten akzeptiert werden. Eine klassische Erweiterung sind Baumautomaten, welche ähnliche Übergangsregeln wie Wortautomaten besitzen, anstelle von Wörtern aber ganze Bäume einlesen. Alle diese Erweiterungen können bestimmte Schleifen, welche durch nicht-strukturelle Teiltyp-Constraints ausgelöst werden, nicht charakterisieren, so daß wir in dieser Arbeit eine neue Automatenerweiterung präsentieren.

Grundlage dieser Arbeit sind P-Automaten. Diese erweitern endliche Automaten um sogenannte P-Kanten. Ob eine P-Kante von dem Automaten durchlaufen werden kann ist abhängig von dem schon zuvor eingelesen Wort. Gegeben sei die Sprache L eines endlichen Automaten, dann kann man einen P-Automaten konstruieren, der die Möglichkeit besitzt, ein Wort aus L mit einem beliebig oft konkatenierten Suffix

aus diesem Wort zu erkennen. Welche Suffixe genau erkannt werden, ist abhängig von seinen P-Kanten; ein zuvor von einem endlichen Automaten eingelesener Suffix muß durch eine P-Kante markiert sein.

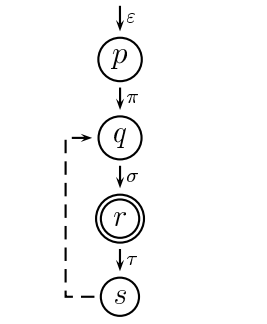
Definition 5 (Erreichbarkeitsrelation). Sei $\mathcal{A} = (A, Q, I, F, \Delta)$ ein endlicher Automat mit *Alphabet* A*, Zuständen* Q*, Anfangszuständen* I*, Endzuständen* F und *Übergangskanten* Δ. Wir definieren $\mathcal{A} \vdash p \xrightarrow{\pi} q$ für Zustände $p, q \in Q$ und $\pi \in A^*$ falls der Automat $\mathcal{A}$ einen Übergang von p nach q erlaubt und hierbei das Wort π liest.

Die von $\mathcal{A}$ erkannte Sprache $\mathcal{L}(\mathcal{A})$ ist die Menge $\{\pi \in A^* \mid \mathcal{A} \vdash p \xrightarrow{\pi} q,\ p{\in}I, q{\in}F\}$.

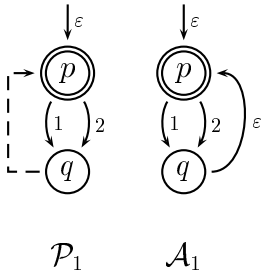
Definition 6 (P-Automat). Ein *P-Automat* $\mathcal{P}$ ist ein Tupel $(\mathcal{A}, P)$, bestehend aus einem endlichen Automaten $\mathcal{A} = (A, Q, I, F, \Delta)$ und einer Menge aus *P-Kanten* $P \subseteq Q \times Q$ zwischen den Zuständen von $\mathcal{A}$. Der P-Automat $\mathcal{P}$ erkennt die Sprache $\mathcal{L}(\mathcal{P}) \subseteq A^*$, gegeben durch:

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{A}) \cup \bigcup\{\pi(\sigma\tau)^+\sigma \mid \mathcal{A} \vdash p \xrightarrow{\pi} q \xrightarrow{\sigma} r \xrightarrow{\tau} s,\ (s,q){\in}P,\ p{\in}I,\ r{\in}F\}$$

Wir zeichnen einen P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ als einen endlichen Automaten $\mathcal{A}$ mit zusätzlichen gestrichelten Linien für seine P-Kanten P. Der P-Automat $\mathcal{P}$ unterstützt alle Übergänge seines zugrundeliegenden endlichen Automaten $\mathcal{A}$. Zusätzlich erlaubt $\mathcal{P}$ seine P-Kanten P als gewöhnliche ε-Kanten zu traversieren; hierbei bestimmt die erste Traversierung einer P-Kante die Periode des noch zu lesenden Wortes. Als Periode bezeichnen wir das Wort $\sigma\tau$ in Definition 6.



Beispiel 4. Wir erklären die Funktionsweise von P-Kanten anhand der folgenden Automaten: Gegeben sei der endliche Automat $\mathcal{A}$ mit Alphabet $\{1, 2\}$, Zuständen $\{p, q\}$, Anfangs- und Endzustand $\{p\}$ und Übergangskanten $p \xrightarrow{1,2} q$. Diesen Automaten erweitern wir zu dem P-Automaten $\mathcal{P}_1$ durch Hinzunahme der P-Kante (q, p) und zu dem endlichen Automaten $\mathcal{A}_1$ durch Hinzunahme der ε-Kante (q, p). Beide Automaten sind rechts dargestellt. Der endliche Automat $\mathcal{A}_1$ akzeptiert offensichtlich die Sprache $\{1, 2\}^*$. Die P-Kante des P-Automaten $\mathcal{P}_1$ kann genau wie die ε-Kante von $\mathcal{A}_1$ behandelt werden, mit der Einschränkung, daß die Wahl zwischen den Kanten 1 und 2, einmal getroffen, beibehalten werden muß. Somit erkennt $\mathcal{P}_1$ die Sprache $1^* \cup 2^*$.

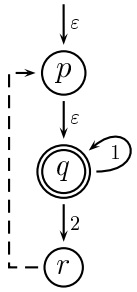


3.2 Eigenschaften der P-Automaten

Der P-Automat besitzt die Möglichkeit, einen mit einer P-Kante markierten Abschnitt (die Periode $\sigma\tau$) beliebig oft zu iterieren. Die Länge der Periode ist hierbei unbeschränkt, da das Wort $\sigma\tau$ aus einer regulären Menge gewählt werden kann. Diese kann im allgemeinen unendlich sein, was der Grund für das nachfolgende Lemma ist:

Lemma 1. Es existiert ein P-Automat, dessen Sprache nicht kontextfrei (und somit nicht regulär) ist.

Beweis. Wir betrachten den P-Automaten $\mathcal{P}$ mit Alphabet $\{1, 2\}$, Zuständen $\{p, q, r\}$, Anfangszustand $\{p\}$, Endzustand $\{q\}$, Übergangskanten $p \xrightarrow{\varepsilon} q$, $q \xrightarrow{1} q$, und $q \xrightarrow{2} r$ und einer einzigen P-Kante (r, p). Der rechts dargestellte Automat $\mathcal{P}$ akzeptiert die Sprache $\{\sigma \mid \exists n : \sigma \leq \pi^n, \pi \in 1^*2\}$. Diese Sprache $\mathcal{L}(\mathcal{P})$ ist nicht kontextfrei, da ansonsten der Schnitt von $\mathcal{L}(\mathcal{P})$ mit der regulären Sprache $1^*21^*21^*2$ kontextfrei wäre. Der Schnitt ist aber gleich der Sprache $\{1^n21^n21^n2 \mid n \geq 0\}$, welche sicher nicht kontextfrei ist.



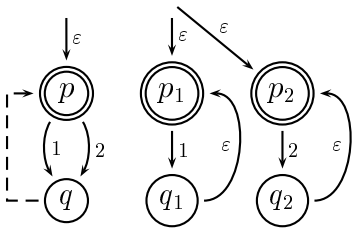
In diesem Beispiel haben wir die Periode aus einer unendlichen Menge gewählt; wählen wir hingegen die Periode $\sigma\tau$ aus einer endlichen Menge, ist die resultierende Sprache $\mathcal{L}(\mathcal{P})$ regulär, da wir nun durch eine Automatentransformation alle P-Kanten in ε-Kanten übersetzen können:

Lemma 2. Die durch einen gegebenen P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ gegebene Sprache $\mathcal{L}(\mathcal{P})$ ist regulär, falls für jede P-Kante $(r, p) \in P$ die Menge $\{\pi \mid \mathcal{A} \vdash p \xrightarrow{\pi} r\}$ endlich ist.

Beweis. Falls für jede P-Kante $(r, p) \in P$ die Menge $\{\pi \mid \mathcal{A} \vdash p \xrightarrow{\pi} r\}$ endlich ist, ist sicherlich auch die Menge $M_{(r,p)} = \{(\sigma, \tau) \mid \mathcal{A} \vdash p \xrightarrow{\sigma} q \xrightarrow{\tau} r,\ q \in I\}$ für jede P-Kante $(r, p) \in P$ endlich; sei nun also $M_{(r,p)} = \{(\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots, (\sigma_n, \tau_n)\}$. Desweiteren gibt es für jede P-Kante $(r, p) \in P$ einen regulären Ausdruck $reg_{(p)}$ mit $reg_{(p)} = \{\pi \mid \mathcal{A} \vdash o \xrightarrow{\pi} p,\ o \in I\}$. Nun kann die Menge $\mathcal{L}(\mathcal{P})$ durch die folgenden regulären Ausdrücke äquivalent zu Definition 6 charakterisiert werden:

$$\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{A}) \cup \bigcup_{(r,p)\in P} \bigcup_{(\sigma_i,\tau_i)\in M_{(r,p)}} reg_{(p)}(\sigma_i\tau_i)^+\sigma_i$$

Beispiel 5. Wir übersetzen den P-Automat $\mathcal{P}_1$ aus Beispiel 4 in einen äquivalenten endlichen Automat $\mathcal{A}_{\mathcal{P}_1}$. Dies ist nach Lemma 2 möglich, da die Menge $\{\pi \mid \mathcal{A} \vdash p \xrightarrow{\pi} q\}$ gleich der Menge $\{1, 2\}$ ist. Wir konstruieren den endlichen Automaten $\mathcal{A}_{\mathcal{P}_1}$ mit den Zuständen $\{p_1, q_1, p_2, q_2\}$, Anfangs- und Endzuständen $\{p_1, p_2\}$ und Übergangskanten $p_1 \xrightarrow{1} q_1$, $p_2 \xrightarrow{2} q_2$, $q_1 \xrightarrow{\varepsilon} p_1$ und $q_2 \xrightarrow{\varepsilon} p_2$. Der neue Automat $\mathcal{A}_{\mathcal{P}_1}$ ist äquivalent zu $\mathcal{P}_1$, d.h. $\mathcal{L}(\mathcal{A}_{\mathcal{P}_1}) = \mathcal{L}(\mathcal{P})$. Wir merken an, daß durch die Übersetzung eines P-Automaten in einen endlichen Automaten die Größe polynomial stark anwachsen kann, da jeder Teilautomat mit einer P-Kante in n neue Teilautomaten mit einer ε-Kante übersetzt werden muß.



$\mathcal{P}_1$ $\quad$ $\mathcal{A}_{\mathcal{P}_1}$

Eine analoge Übersetzung des P-Automaten aus dem Beweis von Lemma 1 in einen endlichen Automaten würde fehlschlagen, da hier die Menge $\{\pi \mid \mathcal{A} \vdash p \xrightarrow{\pi} r\} = 1^*2$ unendlich viele Wörter besitzt.

3.3 Traversierungsbäume

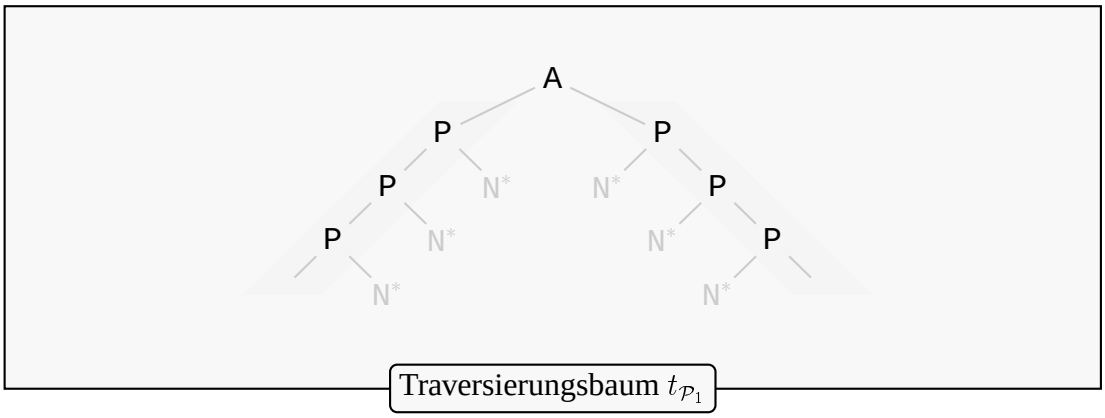
Zur Darstellung der von einem P-Automaten $\mathcal{P}$ erkannten Sprache $\mathcal{L}(\mathcal{P})$ führen wir den Begriff der Traversierungsbäume ein. Ein Traversierungsbaum für einen P-Automat $\mathcal{P} = (\mathcal{A}, P)$ sei eine Funktion $t_\mathcal{P} : A^* \to \{\mathsf{A}, \mathsf{P}, \mathsf{N}\}$, die für jedes Wort über dem Alphabet des Automaten festhält, ob der Automat $\mathcal{A}$ oder $\mathcal{P}$ oder keiner von beiden das Wort akzeptiert.

Definition 7 (Traversierungsbaum). Gegeben sei ein P-Automat $\mathcal{P} = (\mathcal{A}, P)$ mit Alphabet A. Sein *Traversierungsbaum* $t_\mathcal{P}$ sei die Funktion $t_\mathcal{P} : A^* \to \{\mathsf{A}, \mathsf{P}, \mathsf{N}\}$, gegeben durch

$$t_\mathcal{P}(\pi) = \begin{cases} \mathsf{A}, & \text{falls } \pi \text{ in } \mathcal{L}(\mathcal{A}) \\ \mathsf{P}, & \text{falls } \pi \text{ in } \mathcal{L}(\mathcal{P}), \text{ aber nicht in } \mathcal{L}(\mathcal{A}) \\ \mathsf{N}, & \text{falls } \pi \text{ nicht in } \mathcal{L}(\mathcal{P}) \end{cases}$$

Einen Traversierungsbaum $t_\mathcal{P}$ über einem Alphabet A stellen wir graphisch durch einen Baum $(A^*, t_\mathcal{P})$ dar. Dieser Baum $(A^*, t_\mathcal{P})$ genügt aber nicht der in Anhang A gegebenen Definition für Bäume, da wir einen Pfad in einem Baum als eine Sequenz über natürliche Zahlen, nicht über einem beliebigen Alphabet, definiert haben. Abweichend von dieser Definition betrachten wir jetzt einen Baum über beliebige Pfade aus A^*. Um Traversierungsbäume endlich und übersichtlich darstellen zu können, benutzen wir A^* als Abkürzung für den vollständig gelabelten A-Baum (entsprechende Abkürzungen gelten für den vollständig gelabelten P-Baum und N-Baum). Wir nennen einen P^*-Teilbaum, einen *unendlichen P-Cluster*, hingegen nennen wir einen Teilbaum, welcher vollständig bis zur Pfadtiefe n mit P gelabelt ist, einen P-Cluster der Tiefe n. Desweiteren nennen wir in einem Traversierungsbaum einen unendlichen Zweig, der vollständig mit P gelabelt ist, eine *P-Spur*.

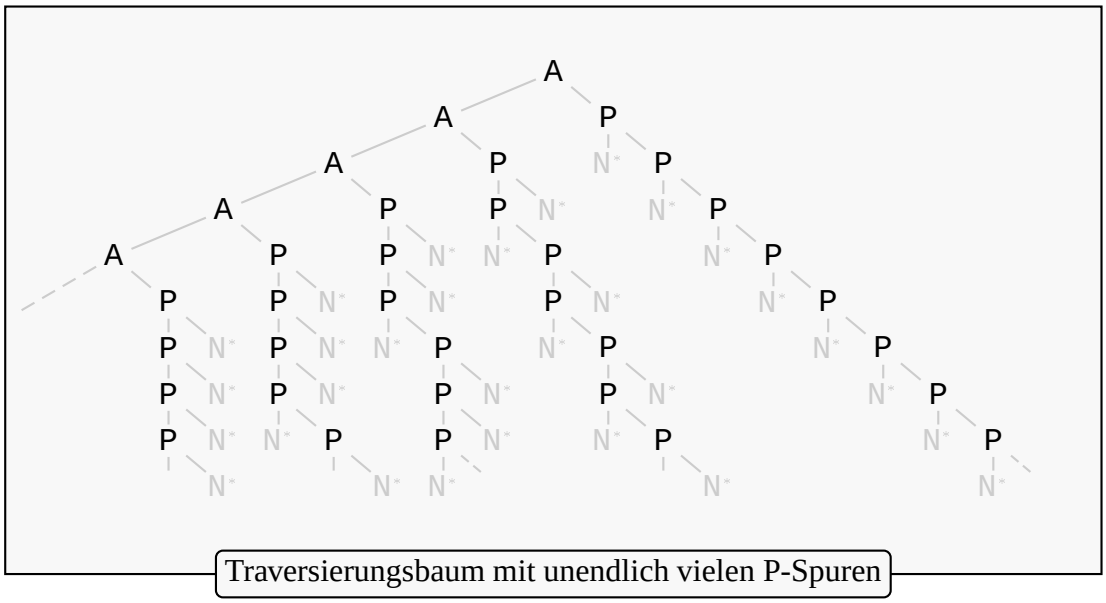
Beispiel 6. Wir geben den Traversierungsbaum $t_{\mathcal{P}_1}$ des P-Automat $\mathcal{P}_1$ aus Beispiel 4 an. Dieser besitzt zwei P-Spuren 1^+ und 2^+. Für eine endliche Darstellung begnügen wir uns mit einem hinreichend großen Teilbaum von $t_{\mathcal{P}_1}$:



Traversierungsbaum $t_{\mathcal{P}_1}$

3.4 P-Cluster in Traversierungsbäumen

Ziel ist es, einen Universalitätstest für P-Automaten $\mathcal{P}$ anzugeben. Hierbei spielen Effekte der P-Kanten, die P-Spuren in dem Traversierungsbaum $t_{\mathcal{P}}$, die zentrale Rolle. Ein P-Automat besitzt nur endlich viele P-Kanten, diese können jedoch unendlich viele P-Spuren in seinem Traversierungsbaum induzieren. Als Beispiel hierfür betrachten wir den Traversierungsbaum für den P-Automat aus dem Beweis von Lemma 1.



Traversierungsbaum mit unendlich vielen P-Spuren

Je dichter die P-Spuren im dem Traversierungsbaum liegen, desto schwieriger wird eine Berechnung der Universalität. Man kann komplizierte Beispiele für P-Automaten angeben, die ihre P-Spuren so dicht anordnen, daß sie sich überlagern; es kommt zu einer endlichen P-Cluster-Bildung. Wir verweisen auf den Anhang B, wo wir zunächst in Beispiel B5 in einem denkbar einfachen P-Automaten mit nur zwei Zuständen eine Überlagerung seiner P-Spuren beobachten. Im nächsten Beispiel B6 erzeugen wir durch einen nun sehr komplizierten P-Automaten einen P-Cluster der Tiefe 2. Die wichtigste globale Eigenschaft der P-Automaten $\mathcal{P}$ ist jedoch, daß ein unendlicher P-Cluster in dem Traversierungsbaum $t_{\mathcal{P}}$ nicht existieren kann:

Proposition 4 (unendliche P-Cluster existieren nicht). Sei $\mathcal{P}$ ein beliebiger P-Automat mit einem Alphabet A von mindestens zwei Zeichen, dann gibt es im Traversierungsbaum $t_{\mathcal{P}}$ keinen unendlichen P-Cluster P^*, d. h. es gibt kein Wort π, so daß $\forall \pi' \in A^* : \; t_{\mathcal{P}}(\pi\pi') = \mathsf{P}$.

Beweis. Wir betrachten einen P-Automaten $\mathcal{P} = (\mathcal{A}, P)$, wobei wir zwei Zeichen 1 und 2 in seinem Alphabet voraussetzen. Wir zeigen durch Widerspruch, daß es kein Wort π mit der Eigenschaft $\forall \pi' \in A^* : \; t_{\mathcal{P}}(\pi\pi') = \mathsf{P}$ gibt. Zu gegebenen π untersuchen wir das Wort $\pi' := 1^{|\pi|}2$, wobei 1^n ein Wort mit genau n-mal dem Zeichen 1 denotiert.

Nach Definition 7 des Traversierungsbaumes folgt aus $t_{\mathcal{P}}(\pi) = \mathsf{P}$ und $t_{\mathcal{P}}(\pi\pi') = \mathsf{P}$ direkt, daß π und $\pi\pi'$ nicht in $\mathcal{L}(\mathcal{A})$, aber in $\mathcal{L}(\mathcal{P})$ sind. Nach Definition 6 gibt es nun eine Zerlegung von π mit $\pi = \pi_1\pi_2\pi_3\pi_4$, so daß $\pi\pi'$ in der regulären Menge $\pi_1(\pi_2\pi_3)^+\pi_2$ ist. Es gibt somit eine natürliche Zahl $n \geq 1$ mit

$$(1) \qquad \pi 1^{|\pi|} 2 \ \in \ \pi_1(\pi_2\pi_3)^n\pi_2.$$

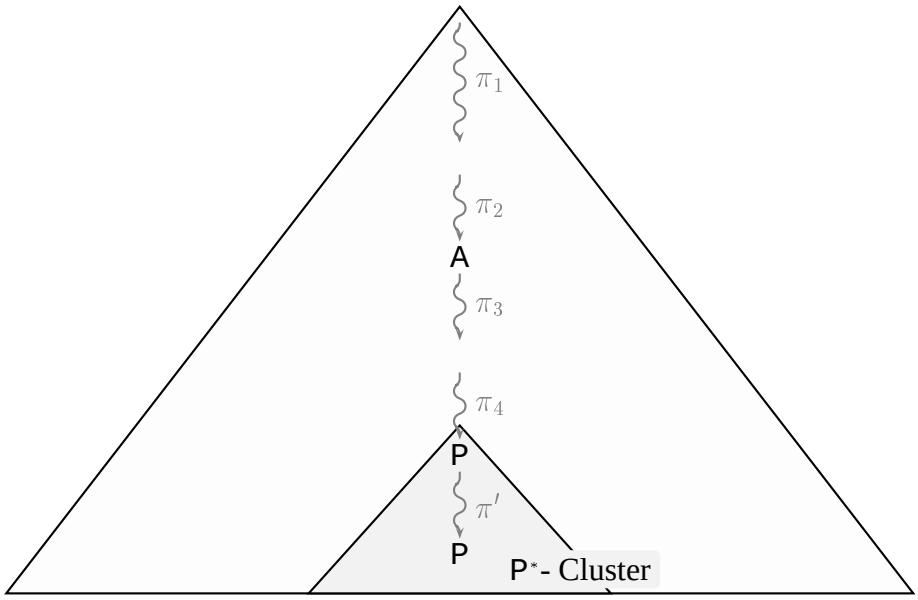
Die Länge von $\pi 1^{|\pi|}2$ ist $2{\cdot}|\pi| + 1$ und die Länge von $\pi_1(\pi_2\pi_3)^n\pi_2$ ist kleinergleich $|\pi^{n+1}|$. Wenden wir diese Längenberechnung auf Gleichung (1) an, dann folgt $n \geq 2$. Wir eliminieren n in Gleichung (1) und erhalten

$$\pi 1^{|\pi_1\pi_4\pi_3\pi_2|} 2 \ \in \ \pi_1(\pi_2\pi_3)^*(\pi_2\pi_3)(\pi_2\pi_3)\pi_2.$$

Wir vergleichen die beiden Ausdrücke von hinten nach vorne: Falls das Wort $\pi_3\pi_2$ leer ist, erhalten wir den Widerspruch $\pi_1\pi_4 1^{|\pi_1\pi_4|}2 = \pi_1$. Ansonsten ist das letzte Zeichen von $\pi_3\pi_2$ eine 2, sei also $\pi_3\pi_2 := \pi_3'\pi_2'2$ und es folgt der Widerspruch

$$\pi 1^{|\pi_1\pi_4|} \ \mathbf{1} \ 1^{|\pi_3'\pi_2'|} 2 \ \in \ \pi_1(\pi_2\pi_3)^*\pi_2\pi_3'\pi_2' \ \mathbf{2} \ \pi_3'\pi_2'2.$$

Vergleichen wir wieder beide Ausdrücke von hinten, steht $\mathbf{2}$ im Widerspruch zur $\mathbf{1}$, da beide Zeichen an der gleichen Position auftreten.



Die Graphik zeigt das Wort $\pi_1\pi_2\pi_3\pi_4\pi'$ in dem Traversierungsbaum $t_{\mathcal{P}}$ und illustriert den obigen Beweis der Cluster-Proposition. Der Beweis basiert auf der folgenden Idee: Wir nehmen an dem Pfad $\pi_1\pi_2\pi_3\pi_4$ einen P^*-Cluster in $t_{\mathcal{P}}$ an. Dann starten von dem Pfad $\pi_1\pi_2\pi_3\pi_4$ unendlich viele P-Spuren in $t_{\mathcal{P}}$. Das steht jedoch im Widerspruch zur

Definition 6 der P-Automaten $\mathcal{P}$, da $\mathcal{P}$ nach Definition nur endlich viele P-Spuren – ausgelöst durch seine P-Kanten – an dem Pfad $\pi_1\pi_2\pi_3\pi_4$ in $t_\mathcal{P}$ vereinbaren kann.

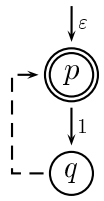
Wir weisen nocheinmal darauf hin, daß schon ein P-Automat $\mathcal{P}$ mit nur einer P-Kante unendlich viele P-Spuren in seinem Traversierungsbaum $t_\mathcal{P}$ induzieren kann. Allerdings beginnnen diese P-Spuren dann nicht an einem, sondern an unterschiedlichen Pfaden in $t_\mathcal{P}$. Dieses Phänomen illustrieren die Beispiele B4 und B5 aus dem Anhang B.

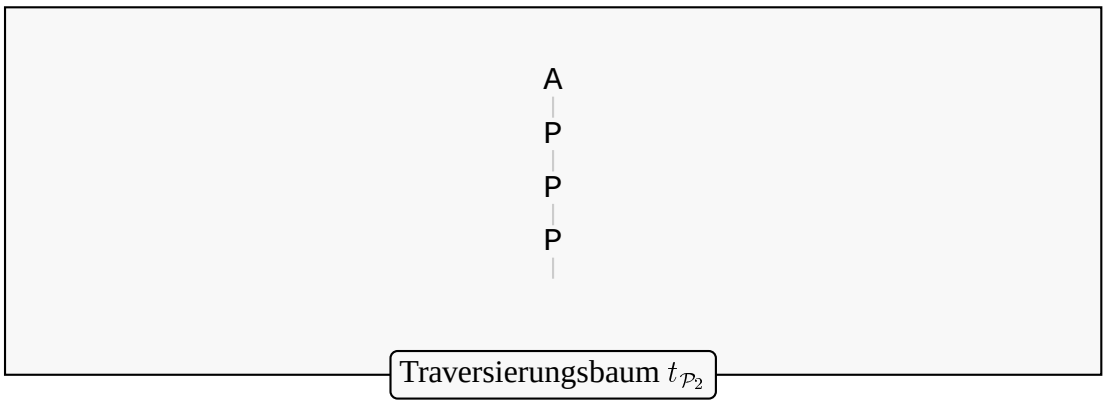
Korollar 1 (Flucht). Sei $\mathcal{P} = (\mathcal{A}, P)$ ein beliebiger P-Automat mit einem Alphabet von mindestens zwei Zeichen und sei $\mathcal{L}(\mathcal{A})$ präfix-abgeschlossen. Falls $\pi \notin \mathcal{L}(\mathcal{A})$, dann gibt es ein Wort $\pi' \notin \mathcal{L}(\mathcal{P})$ mit $\pi \leq \pi'$.

Beweis. Falls $\pi \notin \mathcal{L}(\mathcal{A})$, gibt es nach dem Beweis der Cluster-Proposition 4 ein Wort $\pi' := \pi 1^{|\pi|} 2$ mit $t_\mathcal{P}(\pi') \neq \mathsf{P}$ und $\pi \leq \pi'$. Nach Definition 7 des Traversierungsbaumes folgt aus $t_\mathcal{P}(\pi') \neq \mathsf{P}$ direkt, $\pi \in \mathcal{L}(\mathcal{A})$ oder $\pi \notin \mathcal{L}(\mathcal{P})$. Da $\mathcal{L}(\mathcal{A})$ präfix-abgeschlossen ist, folgt aus $\pi \notin \mathcal{L}(\mathcal{A})$ auch $\pi' \notin \mathcal{L}(\mathcal{A})$ und somit ist π' nicht in $\mathcal{L}(\mathcal{P})$.

Wie das nachfolgende Beispiel zeigt, stellt ein P-Automat mit einem Alphabet von nur einem Zeichen ein Sonderfall dar.

Beispiel 7. Wir betrachten einen P-Automaten mit einem Alphabet von nur einem Zeichen und zeigen, daß in diesem Fall das Korollar 1 und somit auch die Proposition 4 nicht gelten. Sei also $\mathcal{P} = (\mathcal{A}, P)$ ein P-Automat mit Alphabet $A = \{1\}$, Zuständen $\{p, q\}$, Anfangs- und Endzustand $\{p\}$, Übergangskante $p \xrightarrow{1} q$ und P-Kante (q, p). Der Automat ist mit seinem Traversierungsbaum angegeben. Es gilt $\mathcal{L}(\mathcal{A}) = \{\varepsilon\}$ und $\mathcal{L}(\mathcal{P}) = 1^*$ und somit existiert im Traversierungsbaum am Pfad 1 ein P^*-Teilbaum, was im Gegensatz zur Proposition 4 steht. Es existiert ein Wort $\pi \notin \mathcal{L}(\mathcal{A})$, hingegen gibt es kein Wort π' mit $\pi' \notin \mathcal{L}(\mathcal{P})$, was wiederum im Gegensatz zum Korollar 1 steht.





Mit dem nachfolgenden Korollar beweisen wir, daß das Universalitätsproblem

von einem P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ invariant gegenüber seinen P-Kanten P ist, falls A präfix-abgeschlossen ist.

Korollar 2. Sei $\mathcal{P} = (\mathcal{A}, P)$ ein beliebiger P-Automat mit einem Alphabet A von mindestens zwei Zeichen und sei $\mathcal{L}(\mathcal{A})$ präfix-abgeschlossen, dann gilt:

$$\mathcal{L}(\mathcal{P}) = A^* \iff \mathcal{L}(\mathcal{A}) = A^*$$

Beweis. Die Implikations-Richtung „$\Leftarrow$“ ist offensichtlich, da nach Definition 6 die Menge $\mathcal{L}(\mathcal{A})$ eine Teilmenge von $\mathcal{L}(\mathcal{P})$ ist.

Wir zeigen nun die andere Implikations-Richtung, indem wir $\mathcal{L}(\mathcal{A}) \neq A^* \Rightarrow \mathcal{L}(\mathcal{P}) \neq A^*$ beweisen. Sei also π ein Wort, das nicht in $\mathcal{L}(\mathcal{A})$ ist. Nach dem Flucht-Korollar 1 gibt es dann auch ein Wort π', das nicht in $\mathcal{L}(\mathcal{P})$ ist und somit ist $\mathcal{L}(\mathcal{P})$ nicht universell.

3.5 Komplexitätsfragen der P-Automaten

In diesem Abschnitt starten wir eine systematische Untersuchung von Komplexitätsfragen der P-Automaten. So bestimmen wir die Komplexität des Wortproblems (welches für diese Arbeit nicht weiter relevant ist) und des Universalitätsproblems. Bei dem Universalitätsproblem treten Schwierigkeiten auf, welche wir im nächsten Abschnitt genauer untersuchen.

Lemma 3. Das Wortproblem für P-Automaten ist in der Komplexitätsklasse P.

Beweis. Als obere Schranke für das Wortproblem geben wir einen deterministischen polynomialen Algorithmus an: Gegeben sei ein Wort w und ein P-Automat $\mathcal{P} = (\mathcal{A}, P)$, zu beantworten ist, ob $w \in \mathcal{L}(\mathcal{P})$. Zunächst testen wir, ob w in $\mathcal{L}(\mathcal{A})$ liegt, falls ja, liegt w auch in $\mathcal{L}(\mathcal{P})$, ansonsten testen wir, ob:

$$w \in \bigcup \{\pi(\sigma\tau)^+\sigma \mid \mathcal{A} \vdash p \xrightarrow{\pi} q \xrightarrow{\sigma} r \xrightarrow{\tau} s,\ (s,q) \in P,\ p \in I,\ r \in F\}$$

Hierzu testen wir für alle Präfixe π von w und für alle P-Kanten (s, q) aus P und für alle Pfade $(\sigma\tau)$ mit der Eigenschaft, $\pi(\sigma\tau)$ ist Präfix von w, die Bedingung: Ist w in der regulären Sprache $\pi(\sigma\tau)^+\sigma$ und gilt zusätzlich $\mathcal{A} \vdash p \xrightarrow{\pi} q \xrightarrow{\sigma} r \xrightarrow{\tau} s$ mit $(s, q) \in P$, $p \in I$ und $r \in F$? Falls ja, ist obige Bedingung erfüllt und w ist in $\mathcal{L}(\mathcal{P})$, falls nein, ist w *nicht* in $\mathcal{L}(\mathcal{P})$. Offensichtlich läuft dieser Algorithmus in Polynomialzeit.

Ausgehend von dem wichtigen Korollar 2 beweisen wir nun die zentrale Komplexitätsaussage dieser Arbeit. Wir nennen einen endlichen Automat *präfix-abgeschlossen*, falls alle seine Zustände Endzustände sind. Diese Definition ist syntaktisch überprüfbar. Hingegen sei ein endlicher Automat $\mathcal{A}$ *semantisch präfix-abgeschlossen*, falls seine Sprache $\mathcal{L}(\mathcal{A})$ präfix-abgeschlossen ist. Es ist leicht einzusehen, daß ein (syntakisch) präfix-abgeschlossener endlicher Automat auch semantisch präfix-abgeschlossen ist. Andererseits gibt es zu jeder von einem semantisch

präfix-abgeschlossen endlichen Automaten erkannten Sprache $\mathcal{L}(\mathcal{A})$ einen (syntaktisch) präfix-abgeschlossenen endlichen Automaten, der die gleiche Sprache $\mathcal{L}(\mathcal{A})$ erkennt. Wie man jedoch ausgehend von einem semantisch präfix-abgeschlossenen, den äquivalenten (syntaktisch) präfix-abgeschlossenen endlichen Automaten konstruiert, ist aufwendiger; wir gehen auf diese Konstruktion nicht weiter ein.

Proposition 5. Das Universalitätsproblem von P-Automaten $\mathcal{P} = (\mathcal{A}, P)$, mit der Einschränkung einer präfix-abgeschlossenen Sprache des endlichen Automaten $\mathcal{A}$ und einem Alphabet von mindestens zwei Zeichen, ist PSPACE-vollständig.

Für den Beweis wiederholen wir zunächst aus (Rehof, 1998) das folgende Komplexitätsresultat:

Lemma 4 (Rehof). Das Universalitätsproblem eines präfix-abgeschlossenen endlichen Automaten mit einem Alphabet von mindestens zwei Zeichen ist PSPACE-vollständig.

Beweis. Die Idee des Beweises beruht auf einer Simulationstechnik: Wir simulieren die Traversierung eines beliebigen endlichen Automaten durch die Traversierung eines präfix-abgeschlossenen endlichen Automaten. Anschließend zeigen wir, daß das Universalitätsproblem invariant gegenüber dieser Simulation ist.

Sei also ein beliebiger endlicher Automat $\mathcal{A}$ gegeben. Wir konstruieren in Linearzeit zu $\mathcal{A}$ einen präfix-abgeschlossenen endlichen Automaten $\mathcal{A}'$. Das Alphabet von $\mathcal{A}'$ ist gegenüber dem Alphabet von $\mathcal{A}$ um ein neues Zeichen erweitert, sei also $A' = A \cup \{a^{\text{neu}}\}$. Zusätzlich erweitern wir die Zustände von $\mathcal{A}'$ gegenüber den Zuständen von $\mathcal{A}$ um einen neuen Zustand q^{neu}, dieser ist auf ewig akzeptierend, das heißt, er ist ein Endzustand und besitzt für jedes Zeichen aus A' eine reflexive Übergangskante. Wir erhalten nun $\mathcal{A}'$ aus $\mathcal{A}$, indem wir von allen Endzuständen aus $\mathcal{A}$ eine Kante mit der Beschriftung a^{neu} zu dem Zustand q^{neu} einführen und anschließend alle Nicht-Endzustände zu Endzuständen erklären.

Es ist nun zu zeigen, daß das Universalitätsproblem invariant gegenüber dieser Automatentransformation bleibt. Sei also die Sprache $\mathcal{L}(\mathcal{A})$ universell, dann gilt nach Konstruktion von q^{neu}, daß die Sprache $\mathcal{L}(\mathcal{A}')$ gleich $\{\pi a^{\text{neu}} \sigma \mid \pi \in A^*, \sigma \in A'^*\}$ ist, also universell. Sei nun die Sprache $\mathcal{L}(\mathcal{A})$ nicht universell, d. h. es gibt ein Wort $\pi \notin \mathcal{L}(\mathcal{A})$. Nach obiger Transformationsvorschrift ist dann das Wort πa^{neu} nicht in $\mathcal{L}(A')$, also ist auch $\mathcal{L}(A')$ nicht universell.

Beweis der Proposition 5. Nach Lemma 4 ist das Universalitätsproblem eines präfix-abgeschlossenen endlichen Automaten mit einem Alphabet von mindestens zwei Zeichen PSPACE-schwer. Jeder präfix-abgeschlossene endliche Automat ist auch semantisch präfix-abgeschlossen. Somit ist auch das Universalitätsproblem der endlichen Automaten $\mathcal{A}$, mit der Einschränkung einer präfix-abgeschlossenen Sprache und einem Alphabet von mindestens zwei Zeichen, PSPACE-schwer. Durch die kanonische Einbettung vom endlichen Automaten $\mathcal{A}$ in den P-Automaten $\mathcal{P} = (\mathcal{A}, \emptyset)$ erhalten

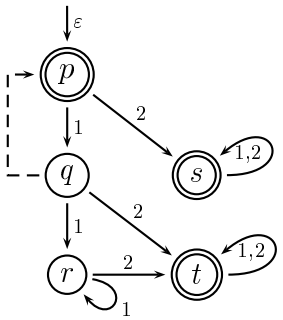
wir den Satz: Das Universalitätsproblem eines P-Automaten $\mathcal{P} = (\mathcal{A}, P)$, mit der Einschränkung einer präfix-abgeschlossenen endlichen Automaten $\mathcal{A}$ und einem Alphabet von mindestens zwei Zeichen, ist PSPACE-schwer.

Das Universalitätsproblem von diesen eingeschränkten P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ ist in PSPACE, da wir dieses auf das Universalitätsproblem von (semantisch präfix-abgeschlossenen) endlichen Automaten $\mathcal{A}$ zurückführen können (nach Korollar 2).

3.6 Ausblick

Für P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ mit der Einschränkung einer präfix-abgeschlossenen Sprache von $\mathcal{A}$ und einem Alphabet von mindestens zwei Zeichen gilt Proposition 5: Ihr Universalitätsproblem ist PSPACE-vollständig. Ohne Präfix-Abgeschlossenheit von der Sprache des Automaten $\mathcal{A}$ treten Schwierigkeiten auf:

Beispiel 8. Wir betrachten hierzu den P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ mit Alphabet $\{1, 2\}$, Zuständen $\{p, q, r, s, t\}$, Anfangszustand $\{p\}$, Endzustände $\{p, s, t\}$, Übergangskanten $p \xrightarrow{1} q \xrightarrow{1} r \xrightarrow{1} r \xrightarrow{2} t$,$p \xrightarrow{2} s \xrightarrow{1,2} s$ und $q \xrightarrow{2} t \xrightarrow{1,2} t$ und eine einzige P-Kante (q, p). Der rechts dargestellte Automat $\mathcal{P}$ akzeptiert die universelle Sprache $\{1, 2\}^*$. Hingegen ist die von seinem zugrundeliegenden endlichen Automaten $\mathcal{A}$ akzeptierte Sprache gleich $\{1, 2\}^* \setminus 1^+$, somit nicht präfix-abgeschlossen und nicht universell. In diesem Fall kann ein Universalitätstest nicht einfach die P-Kanten unberücksichtigt lassen. Andererseits können wir nicht die Effekte aller P-Kanten dem endlichen Automaten einfach zufügen, da die Sprache eines P-Automaten im allgemeinen nicht regulär zu sein braucht. Es stellt sich somit die folgende Frage, die wir im Rahmen dieser Arbeit nicht weiter explorieren.



Gibt es einen Algorithmus, der entscheidet, ob die Sprache eines gegebenen P-Automaten universell ist?

Kapitel 4

Entailment und P-Automaten

Wir charakterisieren das Gültigkeitproblem einer Entailment-Aussage $\varphi \models \psi$ in den eingeschränkten Sprachen NS_n und NS_n^{fin} durch Universalität von P-Automaten. Der P-Automat erkennt dabei widerspruchsfreie Pfade bezüglich der gegebenen Entailment-Aussage $\varphi \models \psi$; ein Pfad π heißt widerspruchsfrei, falls $\varphi \models \psi$ an π nicht widerlegt werden kann. Sind alle Pfade widerspruchsfrei, gilt Entailment.

4.1 Zugrundeliegende Idee

Wir betrachten eine Signatur $\Sigma = \{\bot, \top, f\}$ mit genau einem n-stelligen Funktionssymbol f. Gegeben sei nun eine Entailment-Aussage $\varphi \models \psi$ in der eingeschränkten Sprache NS_n oder NS_n^{fin}. Wir nennen den Constraint ψ die Zielaussage; ohne Beschränkung der Allgemeinheit habe diese die Form $x{\leq}y$ (andere Formen können wir auf diese zurückführen, siehe Kapitel 2). Ziel ist es, einen P-Automaten aus φ zu konstruieren, der alle widerspruchsfreien Pfade akzeptiert; das sind nach Definition alle Wörter π mit $\varphi \models x?\pi \leq_L y?\pi$, das heißt, das Label von $x.\pi$ ist kleiner als das Label von $y.\pi$. Gilt dieser bedingte Teilbaumconstraint für alle Pfade, gilt nach Proposition 3 Entailment.

Der Automatenkonstruktion liegt nun eine Zerlegung der Ziel-Aussage in zahlreiche Teil-Aussagen zugrunde. Wir illustrieren diese Zerlegung anhand von zwei Beispielen: Wenn wir den Constraint φ fixieren, ist eine Entailment-Aussage $\varphi \models x{\leq}y$ in einer Sprache NS_n oder NS_n^{fin} nur noch durch seine Ziel-Aussage $x{\leq}y$ charakterisiert, in den folgenden Beispielen stellen wir sie als Tupel (x, y) dar.

Beispiel 9. Wir untersuchen unter der Signatur $\Sigma{=}\{\bot, \top, \times\}$ mit dem binären Funktionssymbol $\times$ in der Sprache NS_2 bzw. NS_2^{fin} die Entailment-Aussage

$$(1) \qquad \underbrace{\varphi' \wedge\ x{=}x_1{\times}x_2 \wedge y{=}y_1{\times}y_2}_{\varphi} \models x{\leq}y.$$

Da die Variablen x und y einen Term der Form $\ldots \times \ldots$ denotieren, können wir unsere Ziel-Aussage auch angeben als

$$(1) \quad \leftrightarrow \qquad \varphi \models x_1 \times x_2 \leq y_1 \times y_2.$$

Durch weitere Umformungsschritte erhalten wir die beiden Äquivalenzen:

$$(1) \quad \leftrightarrow \qquad \varphi \models x_1 \leq y_1 \wedge x_2 \leq y_2$$

$$(1) \leftrightarrow \varphi \models x_1 \leq y_1 \text{ und } \varphi \models x_2 \leq y_2$$

Somit ist die gegebene Entailment-Aussage $\varphi \models x \leq y$ äqivalent zu den beiden Aussagen $\varphi \models x_1 \leq y_1$ und $\varphi \models x_2 \leq y_2$. Wir sagen, wir können von der Ziel-Aussage (x, y) zu den Ziel-Aussagen (x_1, y_1) und (x_2, y_2) *absteigen,* falls der Constraint $x = x_1 \times x_2 \wedge y = y_1 \times y_2$ in φ vorkommt.

Beispiel 10. Wir untersuchen unter einer beliebigen Signatur Σ in der Sprache NS_Σ bzw. NS_Σ^{fin} die Entailment-Aussage

$$(2) \qquad \underbrace{\varphi' \wedge\ x \leq x'}_{\varphi} \models x \leq y.$$

Falls die Entailment-Aussage $\varphi' \wedge\ x \leq x' \models x' \leq y$ gültig ist, gilt nach Transitivität auch $\varphi' \wedge\ x \leq x' \models x \leq y$, die Umkehrung dieser Implikation gilt jedoch nicht.

$$(2) \qquad \leftarrow \qquad \varphi' \wedge\ x \leq x' \models x' \leq y$$

Wir können somit von der Ziel-Aussage (x, y) zu der Ziel-Aussage „(x, y) oder (x', y)“ absteigen, falls der Constraint $x \leq x'$ in φ vorkommt. Diese Disjunktion ist dafür verantwortlich, daß wir einen nicht-deterministischen Automaten erhalten werden.

Diese Zerlegung ist unabhängig davon, ob Constraints über endlichen oder unendlichen Bäumen interpretiert werden. In der Tat erhalten wir in beiden Fällen den gleichen Algorithmus, nur daß wir im endlichen Fall einen zusätzlichen Occur-Test auf dem gegebenen Constraint durchführen. Beide Versionen des Algorithmus unterscheiden sich also nur in dem Erfüllbarkeitstest, der der gegebenen Sprache zugrunde liegt.

4.2 Der Sicherheitsautomat

Gegeben sei eine Entailment-Aussage $\varphi \models x \leq y$ in der Sprache NS_Σ oder NS_Σ^{fin} mit einem abgeschlossenen und fehlerfreien Constraint φ. Wir konstruieren nun einen P-Automaten, der alle widerspruchsfreien Pfade erkennt, sprich alle *sicheren* Pfade. Wir nennen diesen aus einem gegebenen Constraint φ erzeugten Automaten einen *Sicherheitsautomaten* für φ. Hierzu fixieren wir die zwei globalen Variablen x und y aus der

gegebenen Aussage $\varphi \models x \leq y$ und erzeugen die Übergangskanten, indem wir von der Ziel-Aussage (x, y) in immer weitere Ziel-Aussagen absteigen. Das Ergebnis ist ein endlicher Automat, dessen Knoten Paare von Variablen aus φ, die Ziel-Aussagen,sind. Der Startknoten ist die gegebene Ziel-Aussage (x, y), die hiervon erreichbaren Knoten sind die möglichen Ziel-Aussagen, die wir durch Absteigen erreichen können. Hierbei beschriften wir die dazwischenliegende Kante mit 1 oder 2, wenn wir analog zum Beispiel 9 absteigen, und mit ε, wenn wir analog zum Beispiel 10 absteigen. Wir nennen im folgenden die Übergangskanten Konstruktorkanten. Zusätzlich benötigen wir noch P-Kanten, die bestimmte Pfade als sicher beweisen. Die Notwendigkeit von P-Kanten in Hinblick auf Entailment ist nicht leicht einzusehen. Wir werden im nächsten Abschnitt hierauf ausführlich eingehen.

Wir weisen noch einmal darauf hin, daß wir diese Konstruktion nur für die eingeschränkten Sprachen NS_n bzw. NS_n^{fin} definieren. Die Relevanz dieser Methode für die vollständige Sprache NS_Σ bzw. NS_Σ^{fin} diskutieren wir in der Konklusion, Kapitel 8.

Definition 8 (Sicherheitsautomat). Gegeben sei ein Constraint φ in der *eingeschränkten* Sprache NS_n oder NS_n^{fin}, der bezüglich seiner Sprache abgeschlossen und fehlerfrei ist. Der *Sicherheitsautomat* $\mathcal{P}_\varphi$ sei für zwei gegebene Variablen x und y aus $V(\varphi)$ durch die nachfolgende Tabelle definiert und kann in Polynomialzeit erzeugt werden. Wir sagen, daß P_φ die Entailment-Aussage $\varphi \models x \leq y$ berechnet.

endl.Automat	$\mathcal{A}_\varphi = (A_\varphi, Q_\varphi, I_\varphi, F_\varphi, \Delta_\varphi)$	
Alphabet	$A_\varphi = \{1, \ldots, n\}$	für Signatur $\Sigma_n = \{\bot, \top, f\}$
Zustände	$Q_\varphi = \{(u, v) \mid u, v \in V(\varphi)\}$	
Startzustand	$I_\varphi = \{(x, y)\}$	
Abschwächen	$(u, v) \xrightarrow{\varepsilon} (u', v) \in \Delta_\varphi$	falls $u \leq u'$ in φ
	$(u, v) \xrightarrow{\varepsilon} (u, v') \in \Delta_\varphi$	falls $v' \leq v$ in φ
Dekomposition	$(u, v) \xrightarrow{i} (u_i, v_i) \in \Delta_\varphi$, $(u, v) \in F_\varphi$	falls $u = g(u_1, \ldots, u_n)$ in φ, $v = g(v_1, \ldots, v_n)$ in φ, und $i \in A_\varphi$
Reflexivität	$(u, u) \xrightarrow{i} (u, u) \in \Delta_\varphi$, $(u, u) \in F_\varphi$	falls $i \in A_\varphi$
P−Kanten	$((u, v), (v, u)) \in P_\varphi$	falls $u, v \in V(\varphi)$

Sicherheitsautomat $\mathcal{P}_\varphi = (\mathcal{A}_\varphi, P_\varphi)$ für $\varphi \models x \leq y$

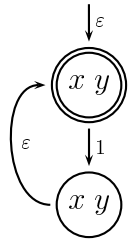
Der Automat $\mathcal{P}_\varphi$ akzeptiert alle Wörter über den natürlichen Zahlen, von 1 bis zu der maximalen Stelligkeit des gegebenen Konstruktors. Seine Zustände sind Paare von

Variablen aus $V(\varphi)$, sein Anfangszustand ist (x, y). Ordnungs-Constraints $u{\leq}v$ korrespondieren mit ε-Übergängen, wobei die Variable der linken Seite in der Ordnung aufsteigt und die der rechten Seite in der Ordnung absteigt. Die Dekompositionsregel vereinbart Konstruktorkanten für alle $i \in A_\varphi$, welche parallel für beide Variablen zum i-ten Kind absteigen. Zustände, auf die die Dekompositionsregel angewandt wird, sind Endzustände. Desweiteren sind alle Zustände der Form (u, u) Endzustände und besitzen reflexive Kanten für alle $i \in A_\varphi$. Zusätzlich besitzt der Automat $\mathcal{P}_\varphi$ eine P-Kante zwischen jedem Zustandspaar (u, v) und (v, u).

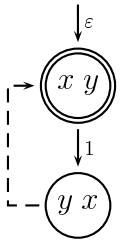
4.3 Konstruktorkanten versus P-Kanten

Wir erklären anhand von einfachen Beispielen die Effekte und Notwendigkeit von P-Kanten und stellen sie gewöhnlichen Konstruktorkanten gegenüber. Im Überblick befinden sich diese und andere wichtige Beispiele im Anhang B.

Beispiel 11. Wir betrachten die Entailment-Aussage $x{=}g(x) \wedge y{=}g(y) \models x{\leq}y$ in der Sprache NS_1. Der Sicherheitsautomat besitzt nur einfache Konstruktorkanten; wir haben ihn nicht als Minimalautomaten angegeben, um ihn mit dem nächsten Beispiel besser vergleichen zu können. Wir untersuchen seine Semantik. Der Sicherheitsautomat versucht Entailment zu widerlegen. Am Pfad ε ist das nicht möglich, da $x.\varepsilon$ und $y.\varepsilon$ mit dem gleichen Konstruktor gelabelt sind. Also steigt der Automat über den Pfad 1 ab und findet das gleiche Problem wie zuvor vor. Alle Pfade sind somit sicher und Entailment gilt.



Beispiel 12 (siehe Anhang B3). Wir betrachten die Entailment-Aussage $x{\leq}g(y) \wedge g(x){\leq}y \models x{\leq}y$ in der Sprache NS_1. Am Pfad ε kann Entailment nicht widerlegt werden, da $x.\varepsilon$ und $y.\varepsilon$ wieder mit dem gleichen Konstruktor g gelabelt sind. Also nehmen wir den Constraint $\psi : x{=}g(x_1) \wedge y{=}g(y_1)$ für zwei frische Variablen x_1 und y_1 an, um die Entailment-Aussage am Pfad 1 zu widerlegen. Wenn wir nun über 1 absteigen, kommen wir nicht zu dem Ausgangsproblem, sondern zu dem Zustand (y, x). Bei dem Abstieg über den Pfad 1 muß nun auch die Annahme ψ gelten, ansonsten kann keine Widerlegung erfolgen; hierdurch wird aber auch der Pfad 1 sicher. Die Sicherheit am Pfad 1 garantieren nicht die Konstruktorkanten (erzeugt durch den gegebenen Constraint), sondern eine notwendige Annahme (in unserem Fall ψ) beim Versuch, Entailment zu widerlegen. Dieser Effekt, eine notwendige Annahme erzwingt Knoten als sicher, pflanzt sich weiter fort. Hierdurch werden alle Pfade aus 1^+ sicher, was genau mit der Semantik einer P-Kante übereinstimmt. Somit erkennt der Sicherheitsautomat genau wie im vorangegangenen Beispiel die Sprache 1^*. Entailment gilt, ist jetzt aber abhängig von den Effekten der P-Kante.

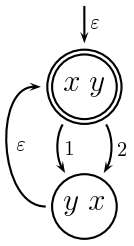


Konstruktorkanten und P-Kanten haben eine unterschiedliche Semantik. Dieses Argument haben wir in Kapitel 3 ausführlich diskutiert. Im Fall eines unären Konstruktors haben wir noch keinen Unterschied dieser beiden Semantiken ausmachen können: Sowohl Konstruktor-, wie auch P-Kante haben eine Schleife erzeugt, die die Sprache 1^+ erkannt hat.

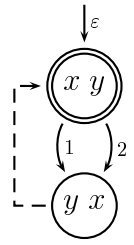
4.4 Entailment-Beispiele mit binärem Konstruktor

Wir betrachten nun beide vorangegangenen Beispiele mit einem binären anstelle eines unären Konstruktors.

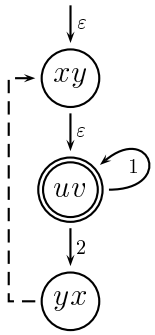
Beispiel 13. Wir betrachten die Entailment-Aussage $x{=}x{\times}x \ \wedge \ y{=}y{\times}y \ \models \ x{\leq}y$ in der Sprache NS_2. Sowohl x als auch y denotieren den vollständig mit $\times$ gelabelten Baum $\mu z.\ z{\times}z$. Somit gibt es keinen Widerspruchspfad, und Entailment gilt. Dieses Ergebnis wird auch durch den rechts gegebenen Sicherheitsautomaten berechnet: Der Sicherheitsautomat erkennt die Sprache $\{1,2\}^*$ und ist somit universell.



Beispiel 14 (siehe Anhang B2). Wir betrachten die Entailment-Aussage $x{\leq}y{\times}y \ \wedge \ x{\times}x{\leq}y \ \models \ x{\leq}y$ in der Sprache NS_2. Entailment kann beispielsweise am Pfad 12 widerlegt werden, wie wir schon im Beispiel 2, Kapitel 2 gezeigt haben. Der Hauptpunkt unserer Argumentation kommt jetzt. Entailment gilt bedingt durch den möglichen Widerspruch somit nicht (im Gegensatz zu Beispiel 12). Dies erscheint zunächst überraschend, da der einzige Unterschied zu Beispiel 12 die Wahl eines unären versus eines binären Konstruktors ist. Diese Situation wird durch den rechts dargestellten Sicherheitsautomaten geklärt. Der zugrundeliegende endliche Automat akzeptiert die Sprache $\{\varepsilon\}$. Die P-Schleifen akzeptieren zusätzlich die Wörter aus $1^+ \cup 2^+$. Da die Semantik der P-Kanten von der gewöhnlicher ε-Kanten abweicht, akzeptiert der Sicherheitsautomat nicht die Wörter 12 und 21, welche in der Tat Widerspruchspfade für Entailment sind.



Beispiel 15 (siehe Anhang B4). Der Sicherheitsautomat der Entailment-Aussage $\varphi_5 : x{\leq}u \ \wedge \ v{\leq}y \ \wedge \ u{=}u{\times}y \ \wedge \ v{=}v{\times}x \models x{\leq}y$ in der Sprache NS_2 oder NS_2^{fin} ist äquivalent zu dem nicht-kontextfreien P-Automaten aus dem Beweis von Lemma 1. Somit kann die Sprache eines Sicherheitsautomaten nicht regulär sein. Andererseits spielt der nicht-kontextfreie Teil der Sprache des Sicherheitsautomaten keine Rolle in Bezug auf Entailment und wir brauchen ihn nicht in unserem Entailment-Test mit zu berücksichtigen.



Die Beispiele zeigen, daß nicht-strukturelles Teiltyp-Entailment in Gegenwart eines einzelnen binären Konstruktors nicht abhängig von den Effekten der P-Kanten ist. Hierbei erfüllen alle bisherigen Sicherheitsautomaten die Bedingung: Ihre zugrundeliegenden endlichen Automaten erkennen eine präfix-abgeschlossene Sprache. Mit dieser Beobachtung sagt die Theorie der P-Automaten, daß P-Kanten die Universalität und somit auch Entailment nicht beeinflussen können (siehe Korollar 2). Es steht nun noch der Beweis aus, daß die Entailment-Charakterisierung durch P-Automaten korrekt und vollständig ist.

4.5 Entailment-Algorithmus

Die vorangegangenen Beispiele haben gezeigt, daß der in Definition 8 gegebene Sicherheitsautomat Entailment in der eingeschränkten Sprache berechnen kann. Im Allgemeinen testen wir hierzu einfach die Sprache des Sicherheitsautomaten auf Universalität. Dieses Universalitätsproblem von P-Automaten ist jedoch ein offenes Problem (P-Automaten können auch nicht-reguläre Sprachen akzeptieren, siehe Beispiel 15). Wir können aber dieses Universalitätsproblem mit der folgenden Eigenschaft der Sicherheitsautomaten einschränken:

Lemma 5 (Basisautomat des Sicherheitsautomaten ist semantisch präfix-abgeschlossen). Alle nach Definition 8 gegebene Sicherheitsautomaten erfüllen die folgende Bedingung: Ihre zugrundeliegenden endlichen Automaten erkennen eine präfix-abgeschlossene Sprache.

Beweis. Wir nehmen an, daß der nach Definition 8 gegebene Sicherheitsautomat $\mathcal{P}_\varphi = (\mathcal{A}_\varphi, P)$ die oben genannte Bedingung nicht erfüllt. Es existiert also ein Wort $\pi \notin \mathcal{L}(\mathcal{A}_\varphi)$ mit $\pi i \in \mathcal{L}(\mathcal{A}_\varphi)$ für ein i aus dem zugrundeliegenden Alphabet. Wir nehmen somit an, daß mindestens eine Transition in dem Sicherheitsautomaten existiert, die von einem Nicht-Endzustand über i zu einem beliebigen Zustand führt. Nach Definition 8 des Sicherheitsautomaten gibt es zwei Metaregeln (**Dekomposition** und **Reflexivität**), die eine Transition über einen Buchstaben i in den Automaten $\mathcal{A}_\varphi$ einfügen. Beide Metaregeln verlangen aber, daß der Startknoten für eine solche Transition ein Endzustand ist, was im Widerspruch zu unserer Annahme steht.

Nach diesem Lemma 5 und Korollar 2 erhalten wir einen einfachen Universalitätstest für unseren Sicherheitsautomaten. Einzige Bedingung für Anwendung dieses Universalitätstests ist ein nicht-triviales Alphabet (d.h. das Alphabet besitzt mehr als ein Zeichen). Wir können somit die folgende Proposition als einen Algorithmus für eingeschränktes Entailment in den Sprachen NS_2 und NS_2^{fin} verstehen.

Proposition 6 (Algorithmus für eingeschränktes Entailment). Gegeben sei eine Entailment-Aussage $\varphi \models x \leq y$ mit $x, y \in V(\varphi)$ in der *eingeschränkten* Sprache NS_2 oder NS_2^{fin}, wobei φ bezüglich seiner Sprache abgeschlossen ist. Die gegebene

Entailment-Aussage gilt genau dann in der gegebenen Sprache, wenn φ bezüglich seiner Sprache nicht fehlerfrei ist oder Universalität des Sicherheitsautomaten $\mathcal{P}_\varphi$ (gebaut aus φ nach Definition 8) gilt.

Beweis. Wir machen eine Fallunterscheidung. Ist φ abgeschlossen und nicht fehlerfrei, dann gilt Entailment $\varphi \models x \leq y$, da φ nach Proposition 2 nicht erfüllbar ist. Ist im anderen Fall φ abgeschlossen und fehlerfrei, dann subsumieren nach der Entailment-Charakterisierung Proposition 3 und der Definition des Widerspruchspfads die folgenden zwei Propositionen den ausstehenden Beweis.

Proposition 7 (Korrektheit). Gegeben sei ein Constraint φ in der *eingeschränkten* Sprache NS_n oder NS_n^{fin}, der bezüglich seiner Sprache abgeschlossen und fehlerfrei ist. Desweiteren seien die Variablen x, y in $V(\varphi)$. Kein vom P-Automaten $\mathcal{P}_\varphi$ akzeptiertes Wort ist ein Widerspruchspfad für $\varphi \models x \leq y$ unter NS_n beziehungsweise NS_n^{fin}.

Proposition 8 (Widerlegungs-Vollständigkeit). Gegeben sei ein Constraint φ in der *eingeschränkten* Sprache NS_2 oder NS_2^{fin}, der bezüglich seiner Sprache abgeschlossen und fehlerfrei ist. Desweiteren sei $\mathcal{P}_\varphi$ der aus φ gebaute Sicherheitsautomat, x, y Variablen aus $V(\varphi)$. Falls $\mathcal{L}(\mathcal{P}_\varphi)$ nicht universell ist, dann gibt es einen Widerlegungspfad für die Entailment-Aussage $\varphi \models x \leq y$ in der Sprache NS_2 beziehungsweise NS_2^{fin}.

Die Beweise dieser beiden Propositionen sind diffizil; wir geben sie in den nächsten zwei Kapiteln 5 (für Korrektheit) und 6 (für Vollständigkeit) an. Der Entailment-Test ist nach dem Algorithmus (Proposition 6) für die Sprachen NS_2 und NS_2^{fin} identisch, bis auf den zusätzlich Occur-Test im Falle NS_2^{fin}.

Wir merken an, daß die Korrektheits-Proposition 7 im Gegensatz zu der Vollständigkeits-Proposition 8 und der Algorithmus-Proposition 6 keine Einschränkung auf einen zweistelligen Konstruktor besitzt (sprich NS_2 und NS_2^{fin}). In der Tat können wir auch die anderen beiden Propositionen auf den n-stelligen Fall erweitern; wir beschränken uns aber aus Gründen der Übersichtlichkeit, vor allem was den Beweis der Widerlegungs-Vollständigkeit betrifft, auf den zweistelligen Fall. Da wir aber in dieser Arbeit auch Sicherheitsautomaten für den n-stelligen Fall diskutieren, beweisen wir die Korrektheits-Proposition 7 für die Sprachen NS_n und NS_n^{fin}. Wir garantieren mit diesem Beweis die Korrektheit aller in dieser Arbeit betrachteten Sicherheitsautomaten für die Sprachen NS_n und NS_n^{fin}. Für eine allgemeinere Betrachtung des n-stelligen Falls verweisen wir auf unsere erweiterte Untersuchung von nicht-strukturellem Entailment (Niehren & Priesnitz, 1999b).

Kapitel 5

Korrektheit des Sicherheitsautomaten

Wir zeigen nun die Korrektheit unseres Sicherheitsautomaten $\mathcal{P}_\varphi$. Die Schwierigkeit liegt darin, die Korrektheit der P-Kanten zu zeigen: Wir zeigen, daß es für jede P-Kante ein induktives Argument gibt, welches im Falle des Vertauschens der Variablen, also $\mathcal{A}_\varphi \vdash (x,y) \xrightarrow{\pi} (u,v) \xrightarrow{\sigma} (v,u)$, die Pfade aus der Menge $\{\pi\sigma' \mid \exists n:\ \sigma' \leq \sigma^n\}$ als widerspruchsfrei beweist.

In diesem Kapitel beweisen wir die Korrektheit des zuvor im Kapitel 4 angegebenen Sicherheitsautomaten. Nach der in Kapitel 2.4 gegebenen Definition eines Widerspruchspfades können wir die Korrektheit durch folgende Proposition angeben.

Proposition 7 (Korrektheit). Gegeben sei ein Constraint φ in der *eingeschränkten* Sprache NS_n oder NS_n^{fin}, der bezüglich seiner Sprache abgeschlossen und fehlerfrei ist. Desweiteren seien die Variablen x, y in $V(\varphi)$. Kein vom P-Automaten $\mathcal{P}_\varphi$ akzeptiertes Wort ist ein Widerspruchspfad für $\varphi \models x{\leq}y$ unter NS_n beziehungsweise NS_n^{fin}.

Für den Beweis charakterisieren wir zunächst die Sprache des Sicherheitsautomaten durch Pfadconstraints. Zu diesem Zwecke führen wir eine neue Sorte *bedingter Teilbaum-Constraints* der Form $x?\pi{\leq^{pr}}y$, $x{\leq^{pr}}y?o$ und $x?\pi{\leq^{pr}}y?o$ ein. Im Gegensatz zu den bisherigen bedingten Teilbaum-Constraints beschränken diese nicht nur den denotierten Teilbaum, sondern alle Präfixe der Denotationen von x und y. Die Semantik dieser neuen Constraints geben wir mit der nachfolgenden Tabelle an:

$$
\begin{array}{lcl}
x?\pi{\leq^{pr}}y & \leftrightarrow & x.\pi{\leq}y \vee \bigvee_{\pi'<\pi} x.\pi'{\leq}\bot \\
x{\leq^{pr}}y?o & \leftrightarrow & x{\leq}y.o \vee \bigvee_{o'<o} \top{\leq}y.o' \\
x?\pi{\leq^{pr}}y?o & \leftrightarrow & \exists u\,(x?\pi{\leq^{pr}}u \wedge u{\leq^{pr}}y?o)
\end{array}
$$

Semantik der bedingten Teilbaum-Constraints

So ist die Semantik des bedingten Teilbaum-Constraints $x?\pi{\leq^{pr}}y$, daß entweder der Teilbaum-Constraint $x.\pi{\leq}y$ gilt oder die Denotation von x zuvor auf einem echten Präfix von π mit $\bot$ abbricht. Die Semantik des Constraints $x{\leq^{pr}}y?o$ ist analog definiert. Die nachfolgenden zwei Lemmata zeigen die engen Zusammenhänge zwischen den einzelnen Sorten der Pfadconstraints.

Lemma 6. Für alle Variablen u, v, Pfade $\pi, \pi' \in \{1, \ldots, n\}^*$ mit $\pi' < \pi$ und Konstruktoren $f \in \Sigma$ sind die folgenden zwei Aussagen in den Sprachen NS_n und NS_n^{fin} gültig.

$$u?\pi{\leq^{pr}}v \rightarrow u?\pi' \leq_L f \qquad u{\leq^{pr}}v?\pi \rightarrow f \leq_L v?\pi'$$

Beweis. Einfache Induktion über den induktiven Aufbau der Pfadconstraints.

Lemma 7 (Teilbaum- versus bedingte Teilbaum-Constraints). Für alle Variablen u, v, Pfade $\pi \in \{1, \ldots, n\}^*$ und Konstruktoren $f \in \Sigma$ sind die folgenden Äquivalenzen in den Sprachen NS_n und NS_n^{fin} gültig.

$$u?\pi{\leq^{pr}}v \;\leftrightarrow\; \exists w(u{\leq}w \wedge w.\pi{=}v) \qquad u{\leq^{pr}}v?\pi \;\leftrightarrow\; \exists w(w{\leq}v \wedge w.\pi{=}u)$$

Beweis. Da die Beweise beider Äquivalenzen analog sind, beweisen wir nur die Gültigkeit der linken Äquivalenz. Die Implikationsrichtung von rechts nach links folgt aus der Semantik des bedingten Teilbaum-Constraints $u?\pi{\leq^{pr}}v$. Es bleibt die Implikationsrichtung von links nach rechts zu zeigen. Sei α eine Belegung, die $u?\pi{\leq^{pr}}v$ erfüllt. Wir machen eine Fallunterscheidung. Falls der Pfad π in $\alpha(u)$ existiert, dann definieren wir $\alpha(w)$ als den Baum $\alpha(u)$, wobei wir den Teilbaum am Pfad π durch $\alpha(v)$ ersetzen. Falls der Pfad π in $\alpha(u)$ nicht existiert, dann gibt es ein π' mit $\pi = \pi'\pi''$, so daß $x.\pi'{\leq}\bot$ durch α erfüllt ist. Sei τ nun ein Baum mit $\tau.\pi'' = \alpha(v)$; dieser Baum existiert für einen Pfad $\pi'' \in \{1, \ldots, n\}^*$ immer. Wir definieren nun $\alpha(w)$ als den Baum $\alpha(u)$, wobei wir den Teilbaum am Pfad π' durch τ ersetzen. In beiden Fällen ist der Constraint $\exists w(u{\leq}w \wedge w.\pi{=}v)$ durch α erfüllt.

Durch die neu eingeführten bedingten Teilbaum-Constaints können wir die Erreichbarkeitsrelation unseres Sicherheitsautomaten äquivalent charakterisieren.

Lemma 8 (Charakterisierung durch Pfadconstraints). Sei $\mathcal{P}_\varphi = (\mathcal{A}_\varphi, P)$ ein beliebiger Sicherheitsautomat für das Entailmentproblem $\varphi \models x{\leq}y$ in der Sprache NS_n oder NS_n^{fin}. Die folgenden Aussagen sind gültig in den Sprachen NS_n und NS_n^{fin}.

1. Falls $\mathcal{A}_\varphi \vdash (u, v) \xrightarrow{\pi} (u', v')$ mit $u' \neq v'$, dann $\varphi \models u?\pi{\leq^{pr}}u' \wedge v'{\leq^{pr}}v?\pi$.
2. Falls $\mathcal{A}_\varphi \vdash (u, v) \xrightarrow{\pi} (u', u')$, dann $\varphi \models u?\pi\sigma{\leq^{pr}}v?\pi\sigma$ für alle Pfade σ.

Beweis. Einfache Induktion über den Aufbau des Sicherheitsautomaten, Definition 8.

Die folgende Eigenschaft nicht-struktureller Teiltyp-Constraints induziert die Notwendigkeit von P-Kanten und somit auch des Konzepts des P-Automaten. Die neu

eingeführten Pfadconstraints können die Situation für das Vorliegen einer P-Kante, nämlich das Vertauschen der beiden betrachteten Variablen in dem Sicherheitsautomaten, beschreiben. Das folgende Lemma zeigt somit die Korrektheit der P-Kanten und stellt den Kern des Korrektheitsbeweises dar. Sein Beweis erfolgt durch einen trickreichen Induktionsbeweis und ist nicht trivial.

Lemma 9 (Pfadconstraints erzeugen P-Schleifen). Für alle Variablen u, v, Pfade $\sigma < \pi$ und Zahlen $k \geq 0$ ist die folgende Implikation in NS_n und NS_n^{fin} gültig:

$$u?\pi{\leq^{pr}}v \wedge u{\leq^{pr}}v?\pi \quad \rightarrow \quad u?\pi^k\sigma \leq_L v?\pi^k\sigma$$

Beweis. Der Beweis erfolgt mit Hilfe von Induktion über k. Sei $k = 0$. Da $\sigma < \pi$ ist, folgt nach Lemma 6, daß $u?\pi{\leq^{pr}}v \wedge u{\leq^{pr}}v?\pi$ impliziert $u?\sigma \leq_L g \wedge g \leq_L v?\sigma$, was wiederum $u?\sigma \leq_L v?\sigma$ impliziert. Angenommen $k > 0$ und es gelte $u?\pi{\leq^{pr}}v \wedge u{\leq^{pr}}v?\pi$. Definitionsgemäß gilt:

$$u?\pi{\leq^{pr}}v \leftrightarrow u.\pi{\leq}v \vee \bigvee_{\varrho<\pi} u.\varrho{\leq}\bot, \quad v{\leq^{pr}}v?\pi \leftrightarrow u{\leq}v.\pi \vee \bigvee_{\varrho<\pi} \top{\leq}u.\varrho$$

Falls es einen Pfad $\varrho < \pi$ gibt mit $u.\varrho{\leq}\bot$ oder $\top{\leq}u.\varrho$, dann wird $u.\varrho{\leq}v.\varrho$ für einen Präfix von π impliziert. In diesem Fall folgt $u?\pi^k\sigma \leq_L v?\pi^k\sigma$ nach Proposition 3. Anderenfalls ist $u.\pi{\leq}v \wedge u{\leq}v.\pi$ gültig. Seien u', v' Variablen mit $u' = u.\pi$ und $v' = v.\pi$. Somit gilt $u'{\leq}v \wedge v.\pi{=}v'$ und impliziert $u'?\pi{\leq^{pr}}v'$ nach Lemma 7. Nach symmetrischer Schlußfolgerung wird auch $u'{\leq^{pr}}v'?\pi$ impliziert. Wir können nun die Induktions-Hypothese anwenden mit dem Resultat $u'?\pi^{k-1}\sigma \leq_L v'?\pi^{k-1}\sigma$, und somit gilt $u?\pi^k\sigma \leq_L v?\pi^k\sigma$, was zu zeigen war.

Beweis der Korrektheit (Proposition 7). Zu zeigen ist, daß $\pi \in \mathcal{L}(\mathcal{P}_\varphi)$ die Gültigkeit der Entailment-Aussage $\varphi \models x?\pi \leq_L y?\pi$ impliziert. Sei $\pi \in \mathcal{L}(\mathcal{P}_\varphi)$ bedingt durch eine Transition $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\pi} (u, v) \in F_\varphi$, welche in dem Endzustand (u, v) endet (**Dekompositions**-Regel). Es gibt dann für alle $i \in A_n$ den Übergang $\mathcal{A}_\varphi \vdash (u, v) \xrightarrow{i} (u', v') \in F_\varphi$. Somit gilt nach Lemma 8 entweder $x?\pi \leq_L y?\pi$ oder ($x?\pi i{\leq^{pr}}u'$ und $v'{\leq^{pr}}y?\pi i$), was wiederum $x?\pi \leq_L y?\pi$ impliziert (Lemma 6).

Sei $\pi \in \mathcal{L}(\mathcal{P}_\varphi)$ bedingt durch eine Transition $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\pi} (u, u) \in F_\varphi$, welche in dem Endzustand (u, u) endet (**Reflexivitäts**-Regel). Es gilt dann $x?\pi \leq_L y?\pi$ nach Lemma 8.

Es verbleibt zu zeigen, daß die durch P-Kanten erzeugten Schleifen sich nicht mit Widerspruchspfaden überlagern können. Falls ein Pfad mittels einer P-Kante in die Sprache $\mathcal{L}(\mathcal{P}_\varphi)$ induziert wird, dann hat der Pfad die Form $\pi(\sigma\,\varrho)^k\sigma$ mit $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\pi} (u, v) \xrightarrow{\sigma\varrho} (v, u)$ für Variablen $u, v \in V(\varphi)$ (nach der Definition 6 und der **P-Kanten**-Regel aus der Tabelle auf Seite 40). Wir nehmen an, daß u verschieden von v ist, ansonsten subsumiert die Reflexivitäts-Regel die aus der P-Kante erzeugten Pfade. Aus $\mathcal{A}_\varphi \vdash (u, v) \xrightarrow{\sigma\varrho} (v, u)$ folgt, daß φ den Constraint $u?\sigma\varrho{\leq^{pr}}v \wedge u{\leq^{pr}}v?\sigma\varrho$ impliziert (Lemma 8). Somit gilt nach dem vorherigen Lemma 9, daß φ den Constraint $u?(\sigma\varrho)^k\sigma \leq_L v?(\sigma\varrho)^k\sigma$ impliziert. Mit $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\pi} (u, v)$ folgt, daß φ auch $x?\pi(\sigma\varrho)^k\sigma \leq_L y?\pi(\sigma\varrho)^k\sigma$ impliziert.

5.1 Anmerkung zu den Pfadconstraints

Es ergeben sich folgende nicht-trivialen Beziehungen zwischen den einzelnen Klassen der Pfad-Consraints. Die folgenden Lemmata sind zwar nicht vonnöten für diese Arbeit, zeigen doch aber schön die komplizierten Zusammenhänge.

Das nachfolgende Lemma hat Ähnlichkeit mit Lemma 6.

Lemma 10. Für alle Variablen u, v und Pfade $\pi \in \{1, \ldots, n\}^*$ ist die folgende Aussage in den Sprachen NS_n und NS_n^{fin} gültig.

$$u?\pi \leq^{pr} v \;\rightarrow\; \exists w(u.\pi = w \rightarrow w \leq v)$$

Beweis. Einfache Induktion über den induktiven Aufbau der Pfadconstraints.

Zu bemerken ist, daß die andere Implikationsrichtung von Lemma 10 nicht gilt. Der Grund ist, daß der Constraint $x?\pi \leq^{pr} y$ sogar dann eine Aussage über x macht, falls $x.\pi$ nicht definiert ist. Wir betrachten hierzu das folgende Gegenbeispiel: Der Constraint $x \leq f(y)$ impliziert $x?1 \leq^{pr} y$, was wiederum $w = x.1 \;\rightarrow\; w \leq x$ impliziert. Der Constraint $w = x.1 \rightarrow w \leq x$ impliziert nicht $x?1 \leq^{pr} y$, da, falls $x.1$ nicht definiert ist, anstelle des Constraints $x \leq \bot$ überhaupt keine Aussage für x impliziert wird.

Lemma 11 (Teilbaum- versus bedingte Teilbaum-Constraints). Für alle Variablen u, v, Pfade $\pi \in \{1, \ldots, n\}^*$ und Konstruktoren $f \in \Sigma$ ist die folgende Äquivalenz in den Sprachen NS_n und NS_n^{fin} gültig.

$$u.\pi = v \;\leftrightarrow\; u?\pi \leq^{pr} v \wedge v \leq^{pr} u?\pi$$

Beweis. Die Implikationsrichtung von links nach rechts ist offensichtlich. Es verbleibt, die Implikationsrichtung von rechts nach links zu beweisen. Wir nehmen also den bedingten Teilbaum-Constraint $u?\pi \leq^{pr} v \wedge v \leq^{pr} u?\pi$ als gültig an. Für beliebige Präfixe $\pi' < \pi$ beweist Lemma 6 die Gültigkeit von $u?\pi' \leq_L f$ und $f \leq_L u?\pi'$. Somit muß $u.\pi$ definiert sein und es gilt der Teilbaum-Constraint $u.\pi = v$.

Kapitel 6

Vollständigkeit des Sicherheitsautomaten

Wir zeigen nun die Widerlegungs-Vollständigkeit unseres Sicherheitsautomaten: Falls seine Sprache nicht universell ist, dann gibt es einen Widerlegungspfad für die gegebene Entailment-Aussage. Wir definieren für diesen Beweis eine syntaktische Unterstützung von Pfadconstraints, mit der wir die Sprache des Sicherheitsautomaten formal charakterisieren. Im Abschnitt 6.3 definieren wir für den gegebenen Constraint eine Saturierung, diese benutzen wir in Abschnitt 6.5 zum Beweis der Widerlegungs-Vollständigkeit. Für diesen Beweis erweitern wir zuvor in Abschnitt 6.4 die Konstruktionsvorschrift unseres Sicherheitsautomaten um eine weitere Regel. Die in Kapitel 3 untersuchte Theorie der P-Automaten zeigt, daß die Widerlegungs-Vollständigkeit invariant gegenüber dieser Erweiterung ist, somit ist dieser Schritt der Erweiterung mathematisch korrekt.

6.1 Resultat

In diesem Kapitel beweisen wir die Vollständigkeit des zuvor im Kapitel 4 gegebenen Sicherheitsautomaten. Nach der in Kapitel 2.4 gegebenen Definition eines Widerspruchspfad können wir die Vollständigkeit durch folgende Proposition charakterisieren.

Proposition 8 (Widerlegungs-Vollständigkeit). Gegeben sei ein Constraint φ in der *eingeschränkten* Sprache NS_2 oder NS_2^{fin}, der bezüglich seiner Sprache abgeschlossen und fehlerfrei ist. Desweiteren sei $\mathcal{P}_\varphi$ der aus φ gebaute Sicherheitsautomat, x, y Variablen aus $V(\varphi)$. Falls $\mathcal{L}(\mathcal{P}_\varphi)$ nicht universell ist, dann gibt es einen Widerlegungspfad für die Entailment-Aussage $\varphi \models x \leq y$ in der Sprache NS_2 beziehungsweise NS_2^{fin}.

Der Rest dieses Kapitels behandelt den Beweis dieser Proposition.

6.2 Syntaktische Unterstützung von Pfadconstraints

Wir definieren in der nachfolgenden Tabelle einen Kalkül der *syntaktischen Unterstützung*, um bedingte Teilbaum-Constraints mit Hilfe von einfachen syntaktischen Argumenten zu beweisen. Ist μ ein bedingter Teilbaum-Constraint der Form $x?\pi{\leq^{pr}}y$, $x{\leq^{pr}}y?o$ oder $x?\pi{\leq^{pr}}y?o$, dann lesen wir die Aussage $\varphi \vdash \mu$ als „φ unterstützt μ syntaktisch“.

$\varphi \vdash x?\varepsilon{\leq^{pr}}y$	falls $x{\leq}y$ in φ
$\varphi \vdash x?\pi i{\leq^{pr}}y$	falls $\varphi \vdash x?\pi{\leq^{pr}}z$ und $z{=}f(z_1,\ldots z_i\ldots,z_n),\ z_i{\leq}y$ in φ
$\varphi \vdash x{\leq^{pr}}y?\varepsilon$	falls $x{\leq}y$ in φ
$\varphi \vdash x{\leq^{pr}}y?\pi i$	falls $\varphi \vdash z{\leq^{pr}}y?\pi$ und $z{=}f(z_1,\ldots z_i\ldots,z_n), x{\leq}z_i$ in φ
$\varphi \vdash x?\pi{\leq^{pr}}y?\pi'$	falls $\exists z : \varphi \vdash x?\pi{\leq^{pr}}z$ und $\varphi \vdash z{\leq^{pr}}y?\pi'$

Syntaktische Unterstützung

Der gegebene Kalkül ist bezüglich der Semantik der Pfadconstraints korrekt, aber nicht vollständig.

Lemma 12 (Korrektheit der syntaktischen Unterstützung). Für alle Pfadconstraints μ gilt: Aus $\varphi \vdash \mu$ folgt auch $\varphi \models \mu$ in den Sprachen NS_Σ und NS_Σ^{fin}.

Beweis. Einfache Induktion über den Aufbau der Pfadconstraints.

Die Vollständigkeit wird durch das folgende Beispiel widerlegt: Es gilt $u{\leq}v \wedge v{\leq}u \models u?1{\leq^{pr}}v?1$, aber es gilt nicht $u{\leq}v \wedge v{\leq}u \vdash u?1{\leq^{pr}}v?1$. Trotzdem ist dieser Kalkül ausdrucksstark genug, die erkannte Sprache $\mathcal{L}(\mathcal{P}_\varphi)$ unseres Sicherheitsautomaten formal zu charakterisieren.

Lemma 13 (Charakterisierung duch syntaktische Unterstützung). Gegeben sei ein abgeschlossener und fehlerfreier Constraint φ mit einem Sicherheitsautomaten $\mathcal{P}_\varphi = (\mathcal{A}_\varphi, P)$ für die Entailment-Aussage $\varphi \models x{\leq}y$, dann gilt

$$\varphi \vdash x?\pi{\leq^{pr}}y?\pi \quad \rightarrow \quad \mathcal{A}_\varphi \vdash (x,y) \xrightarrow{\pi} (u,u).$$

Beweis. Einfache Induktion über den induktiven Aufbau des Sicherheitsautomaten und der Pfadconstraints.

Definition 9. Wir definieren zwei Funktionen l_φ und r_φ für die Entailment-Aussage $\varphi \models x{\leq}y$.

$$\begin{array}{lcll} l_\varphi(\sigma) & = & \max\{\pi \mid \pi{\leq}\sigma \wedge \exists u.\varphi \vdash x?\pi 1{\leq^{pr}}u\} & \text{(Links)} \\ r_\varphi(\sigma) & = & \max\{\pi \mid \pi{\leq}\sigma \wedge \exists v.\varphi \vdash v{\leq^{pr}}y?\pi 1\} & \text{(Rechts)} \end{array}$$

Der Sicherheitsautomat vergleicht beim Absteigen über ein Wort σ zwei Constraints parallel, zum einen $x?\pi{\leq}u$, zum anderen $v{\leq}y?\pi$. Die beiden obigen Funktionen testen, welcher der beiden Constraints zuerst für einen Pfad π abbricht (also nicht mehr definiert ist). Falls $l_\varphi(\sigma) \leq r_\varphi(\sigma)$, dann ist $r_\varphi(\sigma)$ der längste Präfix von σ aus $\mathcal{L}(\mathcal{A}_\varphi)$. Anderenfalls gilt $l_\varphi(\sigma) > r_\varphi(\sigma)$ und $l_\varphi(\sigma)$ ist der längste Präfix von σ aus $\mathcal{L}(\mathcal{A}_\varphi)$. Das folgende Lemma benutzt diese Fallunterscheidung und subsumiert die Vollständigkeits-Proposition 8.

Lemma 14 (Widerspruch). Gegeben sei ein Constraint φ in der *eingeschränkten* Sprache NS_2 oder NS_2^{fin}, der bezüglich seiner Sprache abgeschlossen und fehlerfrei ist. Desweiteren sei $\mathcal{P}_\varphi = (\mathcal{A}_\varphi, P)$ der aus φ gebaute Sicherheitsautomat, x, y Variablen aus $V(\varphi)$ und o ein Pfad mit $o \notin \mathcal{L}(\mathcal{P}_\varphi)$. Es gibt ein Pfad ϱ mit $o{\leq}\varrho$, so daß

1. die Aussage $\varphi \wedge x.\varrho{=}\top \wedge y.\varrho{=}_L \times$ erfüllbar ist, falls $l_\varphi(\varrho) \leq r_\varphi(\varrho)$ gilt.
2. die Aussage $\varphi \wedge x.\varrho{=}_L \times \wedge y.\varrho{=}\bot$ erfüllbar ist, falls $l_\varphi(\varrho) > r_\varphi(\varrho)$ gilt.

Der Beweis von Lemma 14 nimmt den Rest dieses Kapitels ein. Da seine beiden Fälle symmetrisch sind, beschränken wir uns auf seinen ersten Fall.

Wir benutzen im folgenden einen abgeschlossenen und fehlerfreien Constraint φ und die beiden global fixierten Variablen $x, y \in V(\varphi)$ der gegebenen Entailment-Aussage $\varphi \models x{\leq}y$.

6.3 Saturierung

Zu zeigen ist, daß $\varphi \wedge x.\varrho{=}\top \wedge y.\varrho{=}_L \times$ erfüllbar ist. Hierzu definieren wir einen abgeschlossenen und fehlerfreien Constraint $s(\varphi, \varrho)$, welcher erfüllbarkeits-äquivalent zu $\varphi \wedge x.\varrho{=}u \wedge y.\varrho{=}_L \times$ ist (u ist eine frische Variable).

Definition 10. Wir nennen eine Menge $D \subseteq \{1,2\}^*$ *Domänen-abgeschlossen,* falls D präfix-abgeschlossen ist und für alle $\pi \in \{1,2\}^*$ die folgende Eigenschaft erfüllt: $\pi 1 \in D \;\leftrightarrow\; \pi 2 \in D$. Der *Domänen-Abschluß* $dc(D)$ ist die kleinste Domänen-abgeschlossene Menge, welche D enthält.

Definition 11 (Saturierung). Sei φ ein Constraint über $\Sigma = \{\bot, \top, \times\}$, $x, y \in V(\varphi)$ und $\varrho \in \{1,2\}^*$. Für jedes $z \in \{x, y\}$ und $\pi \in dc(\{\varrho 1, \varrho 2\})$ sei q_π^z eine frische Variable und $W(\varphi, \varrho)$ die Menge aller dieser frischen Variablen. Die *Saturierung* $s(\varphi, \varrho)$ von φ am Pfad ϱ ist der Constraint minimaler Größe, welcher die folgenden Eigenschaften a–f erfüllt:

a. φ in $s(\varphi, \varrho)$.

b. Für alle $q_\pi^x \in W(\varphi, \varrho)$: $q_\pi^x{=}q_{\pi 1}^x \times q_{\pi 2}^x$ in $s_{x',y'}(\varphi, \varrho)$, falls $\pi < \varrho$.

c. Für alle $q^y_\pi \in W(\varphi, \varrho):\ q^y_\pi{=}q^y_{\pi 1}{\times}q^y_{\pi 2}$ in $s_{x',y'}(\varphi, \varrho),$ falls $\pi \leq \varrho$.

d. Für alle $q^z_\pi \in W(\varphi, \varrho), u \in V(\varphi):\ q^z_\pi{\leq}u$ in $s_{x',y'}(\varphi, \varrho),$ falls $\varphi \vdash z?\pi{\leq}^{pr}u$.

e. Für alle $q^z_\pi \in W(\varphi, \varrho), u \in V(\varphi):\ u{\leq}q^z_\pi$ in $s_{x',y'}(\varphi, \varrho),$ falls $\varphi \vdash u{\leq}^{pr}z?\pi$.

f. Für alle $q^z_{o\pi}, q^{z'}_{o'\pi} \in W(\varphi, \varrho): q^z_{o\pi}{\leq}q^{z'}_{o'\pi}$ in $s_{x',y'}(\varphi, \varrho),$ falls $\varphi \vdash z?o{\leq}^{pr}z'?o'$.

Lemma 15. Falls der Constraint φ unter NS_Σ (entsprechend NS^{fin}_Σ) abgeschlossen und fehlerfrei ist, dann ist auch die Saturierung $s(\varphi, \varrho)$ unter NS_Σ (entsprechend NS^{fin}_Σ) abgeschlossen und fehlerfrei.

Der Beweis dieses Lemmas ist aufwendig, wir geben ihn im Anhang C an.

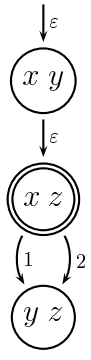
6.4 Wechsel des P-Automaten

Wir haben aus φ einen Sicherheitsautomaten $\mathcal{P}_\varphi$ gebaut und zeigen nun seine Widerlegungs-Vollständigkeit: Falls $\mathcal{L}(\mathcal{P}_\varphi)$ nicht universell ist, dann gibt es einen Widerlegungspfad für die Entailment-Aussage $\varphi \models x{\leq}y$ in der Sprache NS_2 beziehungsweise NS^{fin}_2. Unter der Voraussetzung $l_\varphi(\varrho) \leq r_\varphi(\varrho)$ ist diese Aussage äquivalent zu der folgenden: Falls $\pi \notin \mathcal{L}(\mathcal{P}_\varphi)$, dann ist $\varphi \wedge x.\varrho{=}\top \wedge y.\varrho{=}_L{\times}$ erfüllbar. Die Frage ist nun, an welcher Stelle ϱ wir widerlegen können. Wir müssen ϱ in Abhängigkeit von π definieren. Hierzu stellen wir die folgende Betrachtung an. Die **P-Kanten**-Regel der Definition 8 des Sicherheitsautomaten ist problematisch, sie lautet:

$$\mathbf{P{-}Kanten} \qquad ((u,v),(v,u)) \in P_\varphi, \text{falls } u, v \in V(\varphi).$$

Sie ist nach der Korrektheits-Proposition 7 korrekt, das heißt, alle von dieser Regel verifizierten Wörter sind keine Widerspruchspfade für Entailment. Sie ist jedoch nicht vollständig, in dem Sinne, daß nicht alle vom Sicherheitsautomaten abgewiesenen Wörter Widerspruchspfade für Entailment sind. Wir zeigen diese Aussage im folgenden Beispiel.

Beispiel 16 (Sicherheitsautomat akzeptiert nicht alle Nicht-Widerlegungsknoten). Sei φ der Constraint $x{=}y{\times}y \wedge z{=}z{\times}z \wedge z{\leq}y$. Wir betrachten den Sicherheitsautomaten $\mathcal{P}_\varphi$ für die Entailment-Aussage $\varphi \models x{\leq}y$ in der Sprache NS^{fin}_2. Er besitzt ein Alphabet $\{1, 2\}$, Zustände $\{xy, xz, yz\}$, Anfangszustand xy, Endzustand xz, Übergangskanten $xy \xrightarrow{\varepsilon} xz \xrightarrow{1,2} yz$ und keine P-Kanten. Er ist rechts dargestellt.



Wir analysieren die Situation am Pfad 1. Dieser wird offensichtlich nicht von dem Sicherheitsautomat $\mathcal{P}_\varphi$ akzeptiert. Trotzdem können wir nicht am Pfad 1 Entailment widerlegen, wie die folgende Betrachtung zeigt: Der Constrant φ impliziert, daß das Label von $y.1$, falls definiert, größer oder gleich dem Konstruktor $\times$ ist. Um an 1 widerlegen zu können, müssen wir

also $y.1 = u_1 \times u_2$ zu unserem Constraint hinzunehmen (u_1, u_2 sind frische Variablen). Nun impliziert aber $\varphi \wedge y.1 = u_1 \times u_2$, daß das Label von $x.1$ kleiner oder gleich dem Konstruktor $\times$ ist. Somit ist eine Widerlegung an 1 nicht möglich; der Pfad 1 ist sicher.

Um dieses Problem zu vermeiden fügen wir noch zusätzliche P-Kanten in den Sicherheitsautomaten ein. Nach Korollar 2 und Lemma 5 ist das Universalitätsproblem invariant gegenüber diesen neuen P-Kanten, somit ist dieser Schritt mathematisch korrekt. Im weiteren Beweis setzen wir die folgende zusätzliche Regel bei der Sicherheitsautomaten-Konstruktion (Definition 8) voraus:

$\mathbf{einseitige\ P-Kanten}$ $((u,v),(w,u)),((v,u),(u,w)) \in P_\varphi$, falls $u, v, w \in V(\varphi)$.

Die neuen P-Kanten subsumieren die alten P-Kanten, so daß die ursprüngliche Regel **P-Kanten** redundant wird. Man könnte sich nun fragen, wieso wir nicht gleich die neue Regel in der Definition des Sicherheitsautomaten benutzt haben. Durch die neue Regel **einseitige P-Kanten** akzeptiert der P-Automat Wörter, die Widerlegungspfade sind, wir könnten somit seine Korrektheit nicht mehr mit der im vorherigen Kapitel 5 benutzten Technik beweisen.

6.5 Beweis der Vollständigkeit

Für den Beweis der Widerlegungs-Vollstängigkeit beweisen wir zunächst das folgende Hilfs-Lemma:

Lemma 16. Seien σ, o Pfade. Es gilt $\sigma < o\sigma$ genau dann, wenn $\sigma < o^n$ für ein $n \in \mathrm{N}$ gilt.

Beweis. Die Implikation von links nach rechts zeigen wir mit Induktion über die Länge von σ. Zunächst folgt aus $\sigma < o\sigma$ die Beschränkung $o \neq \varepsilon$. Sei $|\sigma| \leq |o|$. Dann gilt nach der Voraussetzung $\sigma < o\sigma$ auch $\sigma \leq o^1$. Sei also $|\sigma| > |o|$. Nach der Voraussetzung $\sigma < o\sigma$ existiert somit ein σ' mit $\sigma = o\sigma'$ und $\sigma' < \sigma$. Es folgt nun $o\sigma' < oo\sigma'$ und somit $\sigma' < o\sigma'$. Da $\sigma' < \sigma$, können wir nun die Induktions-Hypothese auf $\sigma' < o\sigma'$ anwenden mit dem Resultat $\sigma' < o^n$ für ein $n \in \mathrm{N}$, und somit gilt auch $\sigma < o^{n+1}$, was zu zeigen war.

Es verbleibt die Implikation von rechts nach links zu zeigen: Falls $n = 0$ oder $o = \varepsilon$, so ist σ nicht definiert und die linke Seite automatisch erfüllt; sei also $\sigma < o^n$ mit $o \neq \varepsilon$ und $n \geq 1$. Wir machen wieder eine Induktion über die Länge von σ. Sei $|\sigma| \leq |o|$. Dann gilt $\sigma < o$ und somit $\sigma < o\sigma$. Sei also $|\sigma| > |o|$. Nach der Voraussetzung $\sigma < o^n$ existiert dann ein σ' mit $\sigma = o\sigma'$ und $\sigma' < \sigma$. Es folgt $o\sigma' < o^n$ und somit $\sigma' < o^{n-1}$. Da $\sigma' < \sigma$, können wir nun die Induktions-Hypothese auf $\sigma' < o^{n-1}$ anwenden mit dem Resultat $\sigma' < o\sigma'$, und somit gilt auch $o\sigma' < oo\sigma'$ und $\sigma < o\sigma$, was zu zeigen war.

Wir beweisen jetzt den entscheidenden Schritt: Falls $\pi \notin \mathcal{L}(\mathcal{P}_\varphi)$, wobei die Sprache $\mathcal{L}(\mathcal{P}_\varphi)$ durch die **einseitige P - Kanten**-Regel erweitert ist, dann ist $\varphi \wedge x.\pi{=}\top \wedge y.\pi{=}_L \times$ erfüllbar. Zu zeigen ist also, daß $s(\varphi, \varrho) \wedge q^x_\pi{=}\top$ auch abgeschlossen und fehlerfrei bezüglich der gegebenen Sprache ist. Abgeschlossenheit und Zykelfreiheit folgt automatisch aus Lemma 15, aber die Konsistenz (Fehlerfreiheit) erfordert Arbeit. Da der gegebene Constraint φ frei von Konstanten $\bot, \top$ ist, kann prinzipiell nur ein Fehler in der Saturierung von φ auftreten. Für $w, w_1, w_2 \in V(s(\varphi, \varrho))$ ist die folgende Aussage ein Fehler für Konsistenz, da $\top \not\leq_L \times$ gilt:

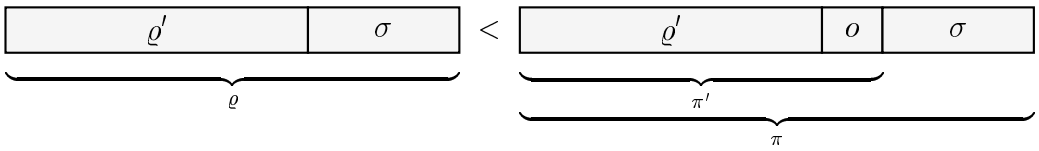
$$q^x_\pi{=}\top \text{ und } q^x_\pi{\leq}w \wedge w{=}w_1{\times}w_2 \text{ in } s(\varphi, \varrho)$$

Wir haben nun jede mögliche Wahl für $w \in V(s(\varphi, \varrho))$ zu untersuchen, wir beschränken uns auf den entscheidenden Fall $w \in W(\varphi, \varrho)$. In diesem Fall wurde $q^x_\pi{\leq}w$ durch Regel f der Saturierungs-Definition 11 dem Constraint $s(\varphi, \varrho)$ hinzugefügt. Die Annahme $w{=}w_1{\times}w_2$ in $s(\varphi, \varrho)$ mit $w \in W(\varphi, \varrho)$ impliziert entweder $w = q^y_\pi$ oder $w = q^z_\varrho$ für $\varrho < \pi$ und $z \in \{x, y\}$; wir machen eine Fallunterscheidung:

Sei $w = q^y_\pi$. Regel f der Saturierungs-Definition benötigt $\varphi \vdash x?\sigma{\leq}^{pr}y?\sigma$ für einen Präfix $\sigma{\leq}\pi$. Nach dem Charakterisierungs-Lemma 13 ist diese Bedingung äquivalent zu $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\sigma} (u, u)$ für ein u. Aus der **Reflexivitäts**-Regel der Sicherheitsautomaten-Konstruktion folgt $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\pi} (u, u)$ und somit $\pi \in \mathcal{L}(\mathcal{P}_\varphi)$, ein Widerspruch zu unserer Annahme.

Sei $w = q^z_\varrho$ für einen Pfad $\varrho < \pi$ und $z \in \{x, y\}$. Die Saturierungs-Regel f nimmt die Existenz von σ, π', ϱ' an, so daß $\varphi \vdash x?\pi'{\leq}^{pr}y?\varrho'$ mit $\pi = \pi'\sigma$ und $\varrho = \varrho'\sigma$. Aus $\varrho < \pi$ folgt $\varrho'\sigma < \pi'\sigma$ und somit $\varrho' < \pi'$. Wir betrachten einen Pfad $o \neq \varepsilon$ mit $\varrho' o = \pi'$. Durch unsere global gemachte Voraussetzung $l_\varphi(\varrho) \leq r_\varphi(\varrho)$ folgt, daß es Variablen $u, v \in V(\varphi)$ gibt mit $\varphi \vdash x?\varrho'{\leq}^{pr}u$ und $\varphi \vdash v{\leq}^{pr}y?\pi'$. Jetzt können wir mit Hilfe des Charakterisierungs-Lemmas 13 eine Automaten-Transition $\mathcal{A}_\varphi \vdash (x, y) \xrightarrow{\varrho'} (u, z) \xrightarrow{o} (z, v)$ identifizieren. Der zentrale Punkt des gesamten Vollständigkeitsbeweises kommt jetzt: Die neu eingeführte **einseitige P - Kanten**-Regel der Sicherheitsautomaten-Konstruktion ergibt eine P-Kante vom Zustand (u, z) zum Zustand (z, v). Da $o \neq \varepsilon$ gilt, ist (u, z) nach Definition 8 des Sicherheitsautomaten ein Endzustand. Aus $\varrho'\sigma < \pi'\sigma$ und $\varrho' o = \pi'$ folgt $\sigma < o\sigma$. Nach Lemma 16 gilt $\sigma < o^n$ für ein $n \in \mathrm{N}$ und somit $\pi < \varrho' o^{n+1}$. Dieses Wort π ist nach Definition 6, der Sprache des P-Automaten, in der Sprache $\mathcal{L}(\mathcal{P}_\varphi)$. Dies ist ein Widerspruch zur untersuchten Annahme.

Die folgende Graphik illustriert die betrachteten Pfade:



Dieser Beweis hat gezeigt, daß das Konzept der P-Kanten entscheidend für unsere Entailment-Analyse ist.

Kapitel 7

Komplexität von Entailment

Wir präsentieren unser Resultat: Entailment von nicht-strukturellen Teiltyp-Constraints ist sowohl in der Sprache NS_2 als auch in NS_2^{fin}, PSPACE-vollständig. Sowohl als untere wie auch als obere Schranke benutzen wir das Universalitätsproblem von speziellen nicht-deterministischen endlichen Automaten.

7.1 Untere Komplexitäts-Schranke

Wir illustrieren die Expressivität unserer eingeschränkten Sprachen NS_2 und NS_2^{fin} durch die Einbettung von endlichen Automaten in diese Sprachen. Diese beiden Sprachen haben wir über Ordnungs-Constraints $\varphi, \psi ::= t_1 \leq t_2 \mid \varphi \wedge \psi$ mit $t_1, t_2 ::= u \mid t_1 \times t_2,\ u \in V$ definiert. Variablen werden als (un-)endliche Bäume über $\Sigma = \{\bot, \top, \times\}$ und $\leq$ als nicht-strukturelle Ordnung über diesen Bäumen interpretiert. Hierzu beweisen wir die folgende Proposition, welche eine Spezialisierung einer Proposition von Henglein und Rehof (1998) ist.

Proposition 1 (eingeschränktes Entailment ist PSPACE-schwer). Nicht-strukturelles Teiltyp-Entailment ist sowohl für die eingeschränkte Sprache NS_2, als auch für NS_2^{fin} PSPACE-schwer.

Henglein und Rehof (1998) benutzen für ihren Beweis sogenannte Constraintautomaten. Ein Constraintautomat agiert auf dem Constraintgraphen, welcher durch einen gegebenen Constraint aufgespannt wird. Der Constraintautomat verfolgt hierbei nur einen Pfad; im Gegensatz hierzu verfolgt unser Sicherheitsautomat bei der Traversierung zwei Pfade parallel.

In dem Rest dieses Abschnittes beweisen wir Proposition 1. Wir beginnen mit der folgenden Definition eines aus einem Automaten konstruierten Constraints. Wir nennen einen endlichen Automat *präfix-abgeschlossen,* falls alle seine Zustände Endzustände sind, siehe auch Abschnitt 3.5.

Definition 12. Sei $\mathcal{A} = (A, Q, I, Q, \Delta)$ ein (nicht-deterministischer) präfix-abgeschlossener endlicher Automat mit Alphabet $A = \{1, 2\}$, Zuständen Q, wobei y nicht in Q ist, Anfangszustand x, Endzuständen Q und Übergangskanten Δ. Wir definieren den Constraint $\varphi_{\mathcal{A}}$, indem wir jeder Kante aus Δ einen Teilconstraint von $\varphi_{\mathcal{A}}$ zuordnen. Sei der Constraint $\varphi_{\mathcal{A}}$ minimal mit

1. $u{\leq}v$ in $\varphi_{\mathcal{A}}$, falls $\mathcal{A} \vdash u \xrightarrow{\varepsilon} v$ mit $u, v \in Q$
2. $u{\leq}v{\times}\_$ in $\varphi_{\mathcal{A}}$, falls $\mathcal{A} \vdash u \xrightarrow{1} v$ mit $u, v \in Q$
3. $u{\leq}\_{\times}v$ in $\varphi_{\mathcal{A}}$, falls $\mathcal{A} \vdash u \xrightarrow{2} v$ mit $u, v \in Q$

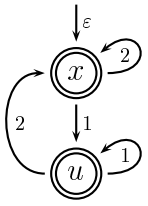
Hierbei steht das Zeichen _ für eine frische Variable aus $V(\varphi_{\mathcal{A}})$.

Lemma 17. Sei A ein endlicher Automat und $\varphi_{\mathcal{A}}$ ein Constraint der entsprechend Definition 12 konstruiert wurde. Es gilt für alle Pfade π aus $\{1, 2\}^*$ und Zustände u und v aus Q:

$$\varphi_{\mathcal{A}} \vdash u?\pi{\leq}v \iff \mathcal{A} \vdash u \xrightarrow{\pi} v$$

Beweis. Einfache Induktion über π.

Beispiel 17. Gegeben sei der präfix-abgeschlossene Automat $\mathcal{A}$ mit Alphabet $A = \{1, 2\}$, Zuständen $\{x, u\}$, Anfangszuständen $\{x\}$, Endzuständen $\{x, u\}$ und Übergangskanten $x \xrightarrow{1} u$, $x \xrightarrow{2} x$, $u \xrightarrow{1} u$ und $u \xrightarrow{2} x$. Die Sprache von $\mathcal{A}$ ist universell, d. h. $\mathcal{L}(\mathcal{A}) = \{1, 2\}^*$. Wir übersetzen nun nach Definition 12 den Automaten $\mathcal{A}$ in einen eingeschränkten nicht-strukturellen Teiltyp-Constraint $\varphi_{\mathcal{A}}$.



$\mathcal{A}$

$$\mathcal{A} \vdash \quad x \xrightarrow{1} u, \quad x \xrightarrow{2} x, \quad u \xrightarrow{1} u, \quad u \xrightarrow{2} x \qquad \varphi_{\mathcal{A}} \equiv \quad x{\leq}u{\times}\_ \wedge x{\leq}\_{\times}x \wedge u{\leq}u{\times}\_ \wedge u{\leq}\_{\times}x$$

Man kann nun über Induktion zeigen, daß $\varphi_{\mathcal{A}}$ den Pfad-Constraint $x?\pi{\leq}\times$ für alle Pfade π aus A impliziert. Somit gilt $\varphi_{\mathcal{A}} \models x{\leq}\times^{\infty}$, wobei $\times^{\infty}$ der total mit $\times$ gelabelte (unendliche) Baum $\mu z.z{\times}z$ ist. Wir axiomatisieren $\times^{\infty}$ und erhalten die Gültigkeit der folgenden Entailment-Aussage:

$$\varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y \models x{\leq}y \text{ in den Modellen } \mathit{NS}_2 \text{ und } \mathit{NS}_2^{\mathit{fin}}$$

Im Falle des Modells $\mathit{NS}_2^{\mathit{fin}}$ wird die Unendlichkeit von $\mathcal{L}(\mathcal{A})$ durch die Menge aller, nicht nur einer Belegung des Constraints $\varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y$ ausgedrückt. Im Falle des Modells NS_2 wird hingegen die Unendlichkeit von $\mathcal{L}(\mathcal{A})$ allein durch die Belegung $I(x){=}I(u){=}I(y){=}\times^{\infty}$ ausgedrückt.

Das Ergebnis dieses Beispiels läßt sich verallgemeinern zu:

Lemma 18. Sei $\mathcal{A}$ ein präfix-abgeschlossener endlicher Automat und $\varphi_{\mathcal{A}}$ ein Constraint, wie in Definition 12 beschrieben, dann gilt in den Sprachen NS_2 und NS_2^{fin}

$$\mathcal{L}(\mathcal{A}) = \{1,2\}^* \quad \Leftrightarrow \quad \varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y \models x{\leq}y.$$

Beweis. Der Constraint $\varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y$ ist in der Sprache NS_2^{fin} (und somit auch in NS_2) erfüllbar, da y nicht in $\varphi_{\mathcal{A}}$ vorkommt und nach Lemma 17 in $\varphi_{\mathcal{A}}$ ein Zykel der Form $u.\pi{=}u$ nicht auftreten kann.

Falls $\mathcal{L}(\mathcal{A})$ gleich $\{1,2\}^*$ ist, dann impliziert der Constraint $\varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y$ die Pfadconstraints $x?\pi \leq_L \times$ und $\times \leq_L y?\pi$ für alle Pfade π. Somit wird nach Proposition 3 auch der Constraint $x{\leq}y$ impliziert.

Falls es einen Pfad π gibt, welcher nicht in $\mathcal{L}(\mathcal{A})$ ist, dann ist die Konjunktion von $\varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y$ und $x.\pi{=}\top \wedge y.\pi{=}\times$ erfüllbar, sowohl über NS_2 als auch über NS_2^{fin}. Somit ist der Pfad π ein Widerspruchspfad für die Entailment-Aussage $\varphi_{\mathcal{A}} \wedge y{\times}y{\leq}y \models x{\leq}y$.

Aus dem gerade gezeigten Lemma 18 folgt direkt das folgende Korollar.

Korollar 3. Die Komplexität des Universalitätsproblems von präfix-abgeschlossenen endlichen Automaten mit einem Alphabet von genau zwei Zeichen ist kleiner als die Komplexität des Entailmentproblems von nicht-strukturellen Teiltyp-Constraints in der Sprache NS_2 oder NS_2^{fin}.

Hieraus folgt nun die untere PSPACE-Schranke unseres engeschränkten Entailments.

Beweis der Proposition 1 (eingeschränktes Entailment ist PSPACE-schwer). Lemma 4 besagt, daß das Universalitätsproblem von präfix-abgeschlossenen endlichen Automaten PSPACE-vollständig ist. Dieses Resultat gilt auch mit der zusätzlichen Einschränkung, daß die endlichen Automaten ein Alphabet von genau zwei Zeichen besitzen. Der Beweis erfolgt analog zum Beweis von 4 durch Simulationstechnik: Wir bauen zu dem gegebenen endlichen Automaten $\mathcal{A}$ einen endlichen Automaten $\mathcal{A}'$ mit einem Alphabet $\{1,2\}$, indem wir jede Transition aus $\mathcal{A}$ in eine eindeutige Transition von $\mathcal{A}'$ in Linearzeit kodieren. Wörter über $\{1,2\}$, die wir durch die Kodierung nicht identifizieren können, akzeptiert der Automat $\mathcal{A}'$ zusätzlich. Das Universalitätsproblem ist gegenüber dieser Transformation von $\mathcal{A}$ nach $\mathcal{A}'$ invariant.

Die zu zeigende Proposition folgt nun aus Korollar 3.

7.2 PSPACE-Vollständigkeit

Wir fügen jetzt alle Resultate dieser Arbeit zusammen und erhalten das folgende Theorem.

Theorem 1 (eingeschränktes Entailment ist PSPACE-vollständig).
Nicht-strukturelles Entailment von Teiltyp-Constraints ist sowohl für die eingeschränkte Sprache NS_2 als auch für NS_2^{fin} PSPACE-vollständig.

Beweis. Die untere PSPACE-Schranke folgt aus der gerade gezeigten Proposition 1 (eingeschränktes Entailment ist PSPACE-schwer).

Zu zeigen ist noch eine obere PSPACE-Schranke des gegebenen Problems. Wir haben in Proposition 6 einen Algorithmus für eingeschränktes Entailment in den Sprachen NS_2 und NS_2^{fin} angegeben. Dieser berechnet Entailment durch die Universalität eines Sicherheitsautomaten mit einem Alphabet von genau zwei Zeichen. Nach Lemma 5 akzeptiert der endliche Automat, der dem Sicherheitsautomat zugrunde liegt, eine präfix-abgeschlossene Sprache. Nach Proposition 5 ist das Universalitätsproblem von P-Automaten $\mathcal{P} = (\mathcal{A}, P)$ mit der Einschränkung einer präfix-abgeschlossenen Sprache des endlichen Automaten $\mathcal{A}$ und einem Alphabet von mindestens zwei Zeichen PSPACE-vollständig.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung dieser Arbeit

In dieser Arbeit haben wir nicht-strukturelles Teiltyp-Entailment für die eingeschränkten Sprachen NS_2 und NS_2^{fin} gelöst. Diese beiden Sprachen haben wir über Ordnungs-Constraints $\varphi, \psi ::= t_1 \leq t_2 \mid \varphi \wedge \psi$ mit $t_1, t_2 ::= u \mid t_1 \times t_2,\ u \in V$ definiert. Variablen werden als (un-)endliche Bäume über $\Sigma = \{\bot, \top, \times\}$ und $\leq$ als nicht-strukturelle Ordnung über diesen Bäumen interpretiert. Wir wiederholen noch einmal unser Resultat.

Theorem 1 (eingeschränktes Entailment ist PSPACE-vollständig).
Nicht-strukturelles Entailment von Teiltyp-Constraints ist sowohl für die eingeschränkte Sprache NS_2 als auch für NS_2^{fin} PSPACE-vollständig.

Für den Beweis haben wir einen Algorithmus angegeben, welcher in PSPACE entscheidet, ob eine gegebene Entailment-Aussage über einfache oder rekursive Typen gilt. Abgesehen von einem zusätzlichen Occur-Test, ist unser Algorithmus für beide Typ-Sorten identisch. Er beruht auf einem neuen Automatenkonzept: Der klassische Begriff der endlichen Automaten wird zu dem Begriff der P-Automaten erweitert. Die Theorie der P-Automaten haben wir in Kapitel 3 systematisch exploriert und ihre Relevanz für das Verständnis von Teiltyp-Entailment illustriert. Die beiden wichtigsten Eigenschaften der P-Automaten geben wir noch einmal an:

- In Gegenwart von P-Kanten kann ein P-Automat nicht kontext-freie Sprachen erkennen.

- Das Universalitätsproblem der P-Automaten mit einem nicht-trivialen Alphabet ist PSPACE-vollständig, falls ihre zugrundeliegenden endlichen Automaten eine präfix-abgeschlosse Sprache akzeptieren.

8.2 Ausblick

Ausgehend von den Ergebnissen dieser Arbeit liegt folgende Frage nahe:

> *Welche Komplexität hat das nicht-strukturelle Entailment-Problem von Teiltyp-Constraints in den nicht-eingeschränkten Sprachen NS_Σ und NS_Σ^{fin}?*

Einen ersten Schritt zur Beantwortung dieser Frage haben wir in unserer nachfolgenden Untersuchung (Niehren & Priesnitz, 1999b) präsentiert. In dieser Untersuchung haben wir die folgenden Resultate gezeigt, die oben genannte Frage bleibt jedoch auch hier offen.

- P-Automaten schränken wir durch ein einfaches syntaktisches Argument ein; wir nennen diese eingeschränkten Automaten *relevante* P-Automaten.
- Analog zur Vorgehensweise dieser Diplomarbeit haben wir in dieser nachfolgenden Untersuchung nicht-strukturelles Entailment in den Sprachen NS_Σ und NS_Σ^{fin} mit Hilfe des Universalitätsproblems von (nun relevanten) P-Automaten charakterisiert. Die einzige Einschränkung ist hierbei, daß die Signatur Σ neben den Konstanten $\bot$ und $\top$ nur einen weiteren, aber nun beliebig-stelligen Konstruktor besitzen darf, d. h. wir betrachten $\Sigma = \{\bot, \top, f\}$.
- Der Konstruktor besitzt zusätzlich die Möglichkeit, die Ordnung beim Absteigen umzudrehen. Wir betrachten also neben den in dieser Arbeit betrachteten *co-varianten* auch *contra-variante* Konstruktoren.
- Wir zeigen in dieser nachfolgenden Untersuchung, daß wir auch umgekehrt das Universalitätsproblem relevanter P-Automaten in nicht-strukturelles Entailment der Sprachen NS_Σ oder NS_Σ^{fin} einbetten können. Beide Probleme besitzen somit die gleiche Komplexität. Es stellt sich somit die folgende hochinteressante und bislang unbeantwortete Frage:

  > *Welche Komplexität hat das Universalitätsproblem von (relevanten) P-Automaten?*

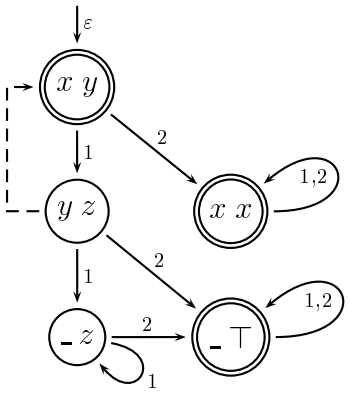
  Diese Frage kann auch in dieser nachfolgenden Untersuchung nicht beantwortet werden. Allerdings ergibt allein die Charakterisierung von nicht-strukturellem Entailment durch relevante P-Automaten die beiden folgenden interessanten Resultate. Beide Fragen sind bislang offen gewesen.
- Nicht-strukturelles Entailment in den Sprachen NS_Σ und NS_Σ^{fin} hat im einfachen und im rekursiven Fall die gleiche Komplexität. Der Beweis besitzt folgenden Aufbau: Wir charakterisien eine Variante des Entailments durch einen relevanten P-Automaten und charakterisieren diesen wiederum durch die entsprechende andere Variante des Entailments. Hierbei setzen wir wieder voraus, daß die Signatur nur einen Konstruktor besitzt.

- Die Komplexität von nicht-strukturellem Entailment in den Sprachen NS_Σ und NS_Σ^{fin} ist unabhängig von der Stelligkeit oder der Co- oder Contra-Varianz des Konstruktors $f \in \Sigma$. Der Beweis ist analog zum vorherigen Punkt.

- Wir charakterisieren durch das Universalitätsproblem eines modifizierten Sicherheitsautomaten auch strukturelles Entailment und subsumieren somit die Komplexitäts-Resultate von Henglein und Rehof (1997, 1998): Einfaches strukturelles Entailment ist coNP-vollständig, rekursives strukturelles Entailment ist PSPACE-vollständig.

Wir geben nun ein Beispiel für einen Sicherheitsautomaten eines Entailment-Problems in der nicht eingeschränkten Sprache NS_Σ an. Eine vollständige Explorierung der Problematik dieser erweiterten Sicherheitsautomaten geht über den Rahmen dieser Arbeit hinaus.

Beispiel 18 (siehe Anhang B8). Wir geben den Sicherheitsautomaten für die Entailment-Aussage
$A_4 : \ x{\leq}y{\times}x \ \wedge \ z{\times}x{\leq}y \ \wedge \ z{\times}\top{\leq}z \models x{\leq}y$ in NS aus dem Beispiel 2 an. Wir merken hierzu an, daß wir die eingeschränkte Constraint-Sprache verlassen haben und nun auch die Konstante $\top$ in der Syntax erlauben. Der Sicherheitsautomat ist rechts angegeben; seine Konstruktions-Regeln sind umfangreicher als die in dieser Arbeit betrachteten Regeln (Defintion 8), da die Konstanten $\top, \bot$ mit in Betracht gezogen werden müssen. Desweiteren werden P-Schleifen durch eine erweiterte **einseitige P - Kanten**-Regel in den Automaten eingeführt (siehe hierzu auch den Abschnitt 6.4). In diesem Beispiel gilt Universalität des P-Automaten und somit Entailment, obwohl die Sprache des zugrundeliegenden endlichen Automat nicht universell ist. Dieses Phänomen tritt in diesem Beispiel zum ersten Mal auf, da nun die Sprache des zugrundeliegenden endlichen Automat nicht mehr präfix-abgeschlossen ist: Der rechts dargestellte endliche Automat akzeptiert zum Beispiel die Wörter ε und 12, aber nicht das Wort 1. Für eine weitere Diskussion dieser Automaten-Klasse verweisen wir auch auf Beispiel 8 in Kapitel 3. Mit diesem Beispiel haben wir demonstriert, wie wir nicht-strukturelles Entailment in der Sprache NS_Σ durch unser Konzept der P-Automaten charakterisieren können. Für eine detailiertere Analyse des Constraints A_4 verweisen wir auf unsere nachfolgende Untersuchung (Niehren & Priesnitz, 1999b).



Anhang A

Mathematische Grundlagen

A.1 Grundlegende Datenstrukturen

Pfade. Ein *Pfad* ist ein Wort über den natürlichen Zahlen $n \geq 1$, welches wir mit π, o, ϱ, oder σ bezeichnen. Der *leere Pfad* wird mit ε denotiert und das freie Monoid der *Konkatenation* von Pfaden π und π' mit $\pi\pi'$, wobei $\varepsilon\pi = \pi\varepsilon = \pi$ ist. Ein *Präfix* eines Pfades π ist ein Pfad π', für welchen ein Pfad π'' existiert mit $\pi = \pi'\pi''$. Ein *echter Präfix* von π ist ein Präfix von π, aber nicht π selbst. Falls π' ein Präfix von π ist, schreiben wir $\pi' \leq \pi$ und falls π' ein echter Präfix von π ist, schreiben wir $\pi' < \pi$.

Signatur. Eine *Signatur* Σ ist eine Menge von Funktionssymbolen, wobei jedem Symbol genau eine Stelligkeit zugeordet wird, d. h. es gibt eine Stelligkeitsfunktion $\mathsf{ar} : \Sigma \to \mathbf{N}$. Zusätzlich definiert Σ eine partielle Ordnung auf ihren Konstanten.

Bäume. Wir setzen eine Signatur Σ mit mindestens einem Funktionssymbol voraus. Ein *Baum* τ ist ein Paar (D_τ, L_τ) mit einer Domäne D_τ und einer Labelfunktion L_τ. Die Domäne D_τ ist eine nichtleere Präfix-abgeschlossene Menge von Pfaden. Die Labelfunktion L_τ ordnet jedem Pfad aus der Domäne D_τ ein Label aus Σ zu, d. h. $L_\tau : D \to \Sigma$. Bäume müssen *Stelligkeits-Konsistent* sein: Ein Pfad πi ist genau dann in D_τ, wenn auch π in D_τ ist und zusätzlich die Bedingung $i \leq \mathsf{ar}(L_\tau(\pi))$ gilt. Ein Baum ist *endlich*, falls seine Baum-Domäne endlich ist und *unendlich* anderenfalls. Wir denotieren die Menge aller endlichen Bäume mit $\mathit{Baum}_\Sigma^{\mathit{fin}}$ und die Menge aller potentiell unendlichen Bäume mit Baum_Σ.

Terme. Wir setzen eine Signatur Σ mit mindestens einem Funktionssymbol und eine unendliche Menge von Variablen V voraus. Ein *Term* t ist entweder eine Variable aus V oder eine Konstruktion $f(t_1, \ldots, t_n)$, wobei $t_1, \ldots, t_n$ Terme sind und f aus Σ ist mit Stelligkeit n.

$$t \ ::= \ x \mid f(t_1, \ldots, t_n) \qquad (f \in \Sigma \text{ mit } \mathsf{ar}(f) = n\,)$$

Ein *Grundterm* über Σ ist ein Term ohne Variablen. Das Modell aller Grundterme über Σ ist isomorph zu dem Modell aller endlicher Bäume aus $\mathit{Baum}_\Sigma^{\mathit{fin}}$.

μ-Terme. Wir setzen wieder eine Signatur Σ mit mindestens einem Funktionssymbol und eine unendliche Menge von Variablen V voraus. Ein *μ-Term* t ist entweder eine Variable aus V oder eine Konstruktion $f(t_1, \ldots, t_n)$, wobei $t_1, \ldots, t_n$ μ-Terme sind und f aus Σ ist mit Stelligkeit n, oder eine Konstruktion $\mu x.t$, wobei x eine gebundene Variable und t ein μ-Term ist.

$$t \ ::= \ x \mid f(t_1, \ldots, t_n) \mid \mu x.t \qquad (f \in \Sigma \text{ mit } \mathsf{ar}(f) = n\,)$$

Die Semantik von $\mu u.t$ ist gegeben durch den μ-Term $t[u/\mu u.t]$. Hierbei denotiert $t[u/s]$ den μ-Term t, wobei t alle freien Auftreten von u ersetzt (falls notwendig nach einer Umbenennung gebundener Variablen). Ein *μ-Grundterm* über Σ ist ein μ-Term ohne freie Variablen, d. h. alle Variablen sind durch μ-Binder gebunden. Das Modell aller μ-Grundterme über Σ ist isomorph zu dem Modell aller regulären (oder rationalen) Bäume aus Baum_Σ.

A.2 Automatentheorie

Endlicher Automat. Ein *endlicher Automat* $\mathcal{A}$ ist ein nicht-deterministisches Berechnungsmodell, gegeben durch ein Tupel (A, Q, I, F, Δ) mit *Alphabet* A, *Zuständen* Q, *Anfangszuständen* I, *Endzuständen* F und *Übergangskanten* Δ, wobei Δ eine Relation $Q \times A \cup \{\varepsilon\} \times Q$ ist. Ausgehend von der Definition 5 der Erreichbarkeit $A \vdash p \xrightarrow{\pi} q$, definieren wir die von A erkannte Sprache $\mathcal{L}(\mathcal{A})$ als die Menge $\{\pi \in A^* \mid \mathcal{A} \vdash p \xrightarrow{\pi} q,\ p \in I, q \in F\}$.

Universalitätsproblem. Gegebenen sei ein endlicher Automat $\mathcal{A}$ mit Alphabet A. Wir nennen seine Sprache $\mathcal{L}(\mathcal{A})$ genau dann *universell*, wenn $\mathcal{L}(\mathcal{A}) = A^*$. Das *Universalitätsproblem* von endlichen Automaten $\mathcal{A}$ ist die Frage, ob $\mathcal{L}(\mathcal{A})$ universell ist. Dieses Problem ist als PSPACE-vollständig bekannt.

Reguläre Ausdrücke. Die von einem endlichen Automaten $\mathcal{A}$ erkannte Sprache $\mathcal{L}(\mathcal{A})$ kann durch einen regulären Ausdruck charakterisiert werden. Ein regulärer Ausdruck ist durch folgende Sprache gegeben:

$$r \ ::= \ \emptyset \mid \varepsilon \mid a \mid r_1 r_2 \mid r_1, r_2 \mid r^* \qquad (a \text{ ist im Alphabet } A)$$

Reguläre Ausdrücke r werden als Mengen R von Zeichenketten über A interpretiert (geschrieben als R=r):

- $\emptyset$ wird als die leere Menge $\{\}$ interpretiert.
- ε wird als die Menge $\{\varepsilon\}$ interpretiert.

- a aus A wird als die Menge $\{a\}$ interpretiert.
- Seien $R_1 = r_1$ und $R_2 = r_2$, dann wird $r_1 r_2$ als die Menge $R_1 \cup R_2$ interpretiert.
- Seien $R_1 = r_1$ und $R_2 = r_2$, dann wird r_1, r_2 als $\{\pi_1 \pi_2 \in A \mid \pi_1 \in R_1, \pi_2 \in R_2\}$ interpretiert.
- Sei $R = r$, dann wird r^* als die Menge $\{\pi^n \mid \pi \in R, n \in \mathbf{N}\}$ interpretiert.

A.3 Grundlagen der Logik

Sprachen. Eine *(logische) Sprache* ist ein Tupel bestehend aus einer Menge von logischen Formeln (Syntax) und einem Modell (Semantik).

Formeln. Gegeben sei eine Signatur Σ, welche Prädikats- und Funktionssymbole beinhaltet. Ein Funktionssymbol mit Stelligkeit 0 nennen wir Konstante. Zusätzlich seien eine Menge von Variablen und eine Menge von Quantoren gegeben. Wir definieren *Formeln* als (quantifizierte) Prädikate, aufgebaut aus Terme über Σ.

Modell. Ein Modell $\mathcal{M}$, auch Semantik oder Σ-Struktur genannt, ist ein Paar (A, I), wobei A, eine nicht-leere Menge, die *Domäne* von A, ist und I eine $\mathcal{M}$-Interpretation.

Interpretation. Eine $\mathcal{M}$-Interpretation ist eine Funktion, die jedem Prädikatssymbol der Stelligkeit n aus Σ eine n-äre Relation über A und jedem Funktionssymbol der Stelligkeit n aus Σ eine Funktion $f : A^n \to A$ zuordnet.

Belegung. Eine *Belegung* α ordet jeder Variable aus der gegebenen Variablenmenge V einen Wert aus der Domäne A (gegeben durch ein Modell) zu. Hierauf bauen wir, wie üblich, die *Erfüllbarkeitsrelation* $\mathcal{M}\ {}_{\alpha}\models \varphi$ auf.

Erfüllbarkeit. Eine Formel φ ist unter einem gegebenen Model $\mathcal{M}$ *erfüllbar*, falls es eine Belegung α gibt, so daß die Erfüllbarkeitsrelation $\mathcal{M}\ {}_{\alpha}\models \varphi$ gilt.

Gültigkeit. Eine Formel ist in einem gegebenen Model $\mathcal{M}$ *gültig*, falls für alle Belegungen α die Erfüllbarkeitsrelation $\mathcal{M}\ {}_{\alpha}\models \varphi$ gilt.

Entailment. Gegeben sei eine Sprache $\mathcal{M}$ mit zwei Formeln φ und ψ aus $\mathcal{M}$. Es gilt die *Entailment-Aussage* $\varphi \models \psi$ in $\mathcal{M}$ definitionsgemäß genau dann, wenn die Formel $\varphi \to \psi$ in $\mathcal{M}$ gültig ist.

Anhang B

Interessante Entailment-Beispiele

In diesem Anhang geben wir nocheinmal eine Zusammenfassung aller besprochenen und einiger neuen Entailment-Beispiele. Die nachfolgende Tabelle zeigt, an welchen Stellen dieser Arbeit das gleiche Beispiel betrachtet wurde.

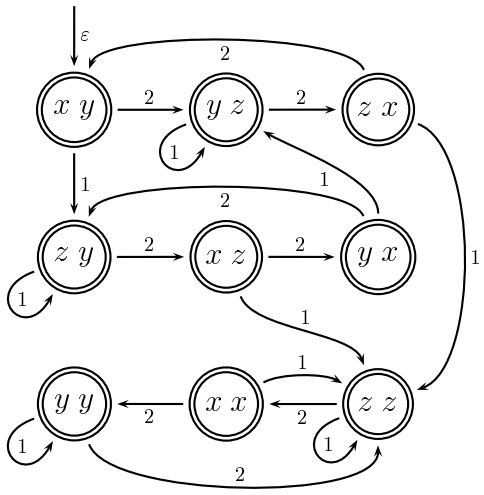
Beispiel im Anhang	Beispiel im Text	zugehörige P-Automat
B1		
B2	Beispiel 2, Seite 22 Beispiel 14, Seite 42	Beispiel 4, 5, 6, Seite 29
B3	Beispiel 12, Seite 41	Beispiel 7 , Seite 34
B4	Beispiel 15, Seite 42	Lemma 1, Seite 30
B5		
B6		
B7		
B8	Beispiel 2 , Seite 22 Beispiel 18 , Seite 61	Beispiel 8, Seite 37

B.1 Entailment-Beispiel 1

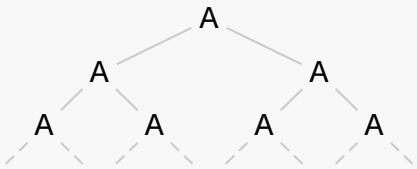
Entailment-Aussage. $\varphi_1 \ : \ x{=}z{\times}y \wedge z{=}z{\times}x \wedge y{=}y{\times}z \ \models x{\leq}y$ in NS_2, NS_2^{fin}

Widerlegung. Entailment kann nicht widerlegt werden.

Sicherheitsautomat.



Traversierungsbaum.



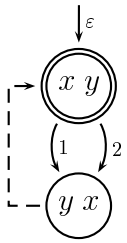
Anmerkung. Dieses Beispiel zeigt die Funktionsweise des Gleichheits-Tests, der dem Sicherheitsautomaten zugrundeliegt. P-Kanten geben wir in dem Sicherheitsautomaten nicht an, da sie keine Relevanz für dieses Problem besitzen. Wir erhalten somit einen gewöhnlichen (regulären) endlichen Automaten.

B.2 Entailment-Beispiel 2

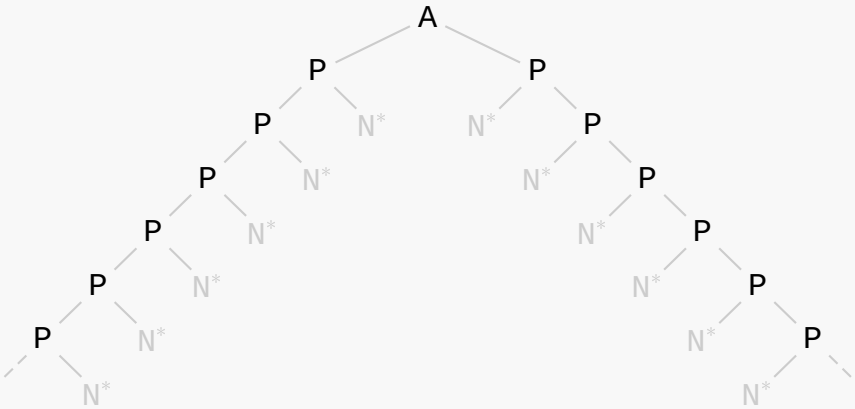
Entailment-Aussage. $\varphi_1 \;:\; x \leq y \times y \;\wedge\; x \times x \leq y \;\models x \leq y$ in NS_2 oder NS_2^{fin}

Widerlegung. Entailment kann an den Pfaden $12, 21, \ldots$ widerlegt werden, beispielsweise mit der Belegung x nach $\bot \times (\top \times \bot)$ und y nach $\top \times (\bot \times \top)$.

Sicherheitsautomat.



Traversierungsbaum.



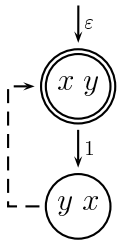
Anmerkung. Dieses Beispiel zeigt ein nicht-strukturelles Teiltyp-Entailment, dessen Sicherheitsautomat zwei P-Schleifen besitzt. Der zugrundeliegende endliche Automat akzeptiert die Sprache $\{\varepsilon\}$. Die P-Schleifen akzeptieren zusätzlich die Wörter aus $1^+ \cup 2^+$. Da die Semantik der P-Kanten von der gewöhnlicher ε-Kanten abweicht, akzeptiert der Sicherheitsautomat nicht die Wörter 12 und 21, welche in der Tat Widerspruchspfade für Entailment sind. Die Gültigkeit der Entailment-Aussage hängt nicht von der Existenz dieser beiden P-Schleifen ab. Wir vergleichen dieses Beispiel mit dem nachfolgenden.

B.3 Entailment-Beispiel 3

Entailment-Aussage. $\varphi_2 \ : \ x \leq g(y) \wedge g(x) \leq y \ \models x \leq y$ in NS_1 oder NS_1^{fin}

Widerlegung. Entailment kann nicht widerlegt werden.

Sicherheitsautomat.



Traversierungsbaum.

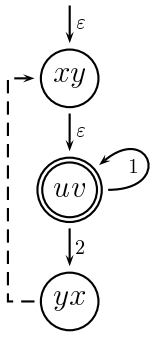
Anmerkung. Wir befinden uns hier in der eingeschränkten Constraint-Sprache NS_1, beziehungsweise NS_1^{fin}. Entailment gilt und ist abhängig von der gegebenen P-Kante. Dieses ist erstaunlich in Anbetracht der Tatsache, daß im vorangegangenen Beispiel Entailment unabhängig von P-Kanten war und der einzige Unterschied zum voherigen Beispiel die Ersetzung eines binären in einen unären Konstruktor ist. Diesen Symmetriebruch erklärt der gegebene P-Automat mit seinem Traversierungsbaum: Die einzelne P-Kante erzeugt eine Spur 1^* von P-Knoten. Bedingt durch das eingeschränkte Alphabet $A = \{1\}$ reicht das schon aus, um alle potentiellen Widerspruchspfade zu eliminieren. Im binären Fall hingegen umfaßt das Alphabet $\{1, 2\}$ und somit kann von einer Spur 1^* von P-Knoten immer noch durch eine Abzweigung 1^*2 ein Widerspruchspfad gefunden werden.

B.4 Entailment-Beispiel 4

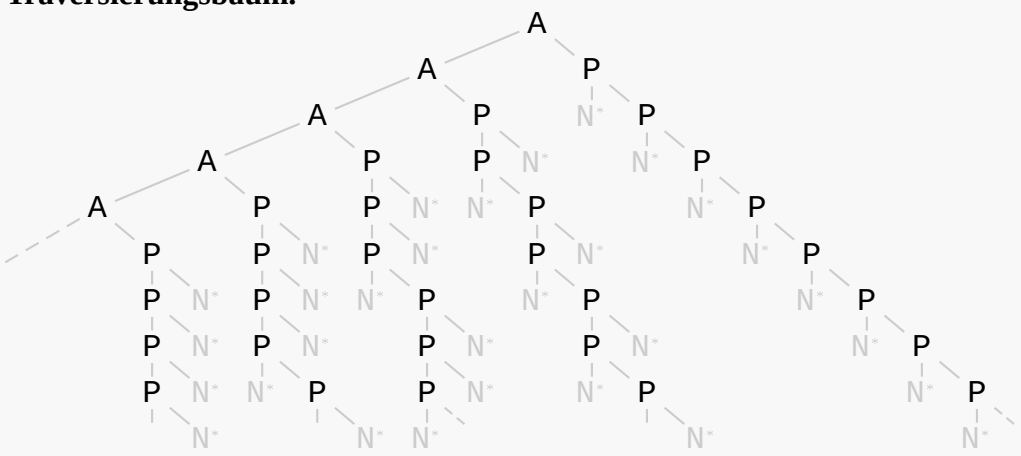
Entailment-Aussage. $\varphi_3 : x{\leq}u \wedge v{\leq}y \wedge u{=}u{\times}y \wedge v{=}v{\times}x \models x{\leq}y$ in NS_2, NS_2^{fin}

Widerlegung. Entailment kann an dem Pfad 21 widerlegt werden.

Sicherheitsautomat.



Traversierungsbaum.



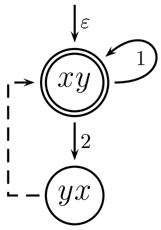
Anmerkung. Dieses Beispiel zeigt, daß der nicht-kontext-freie P-Automat aus Lemma 1 auch durch einen eingeschränkten nicht-strukturelle Teiltyp-Constraint beschrieben werden kann. Somit braucht die Menge aller sicheren Knoten A und P nicht regulär zu sein.

B.5 Entailment-Beispiel 5

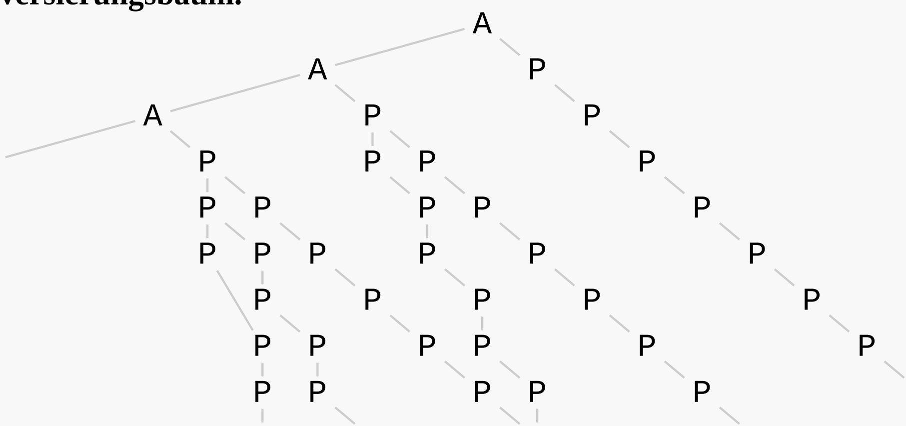
Entailment-Aussage. $\varphi_3 : x \leq x \times y \wedge y \times x \leq y \models x \leq y$ in NS_2, NS_2^{fin}

Widerlegung. Entailment kann an dem Pfad 21 widerlegt werden.

Sicherheitsautomat.



Traversierungsbaum.



Anmerkung. Dieses Beispiel ähnelt dem vorangegangenen, jedoch wurde in diesem Beispiel der P-Automat so modifiziert, daß die P-Schleifen dichter liegen. An manchen Stellen überlagen sich sogar die P-Schleifen, so daß es zu einer Clusterbildung kommt. In dem Traversierungsbaum haben wir alle $\mathtt{N}^*$ Knoten weggelassen. Die Sprache dieses P-Automaten ist wieder nicht kontext-frei; der Beweis läuft analog zum Beweis von Lemma 1.

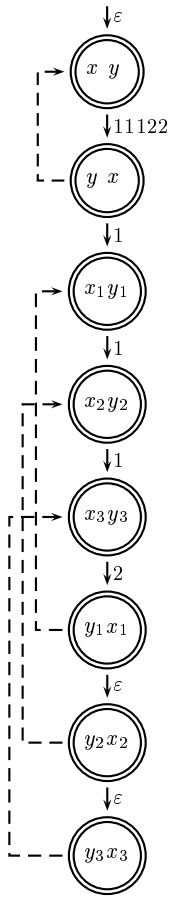
B.6 Entailment-Beispiel 6

Entailment − Aussage.

$$
\begin{array}{lll}
 & x?11122{\leq}y & \wedge\ x{\leq}y?11122 \\
\wedge & y\ ?1{\leq}x_1 & \wedge\ y_1{\leq}x\ ?1 \\
\wedge & x_1?1{\leq}x_2 & \wedge\ y_2{\leq}y_1?1 \\
\wedge & x_2?1{\leq}x_3 & \wedge\ y_3{\leq}y_2?1 \\
\wedge & x_3?2{\leq}y_1 & \wedge\ x_1{\leq}y_3?2 \\
\wedge & x_3{\leq}x_2{\leq}x_1 & \wedge\ y_1{\leq}y_2{\leq}y_3 \\
 & & \models x{\leq}y \text{ in } \mathit{NS}_2,\ \mathit{NS}_2^{\mathit{fin}}
\end{array}
$$

Widerlegung. Entailment kann an dem Pfad 2 widerlegt werden.

Sicherheitsautomat.



Traversierungsbaum. Siehe die Titelseite dieser Arbeit.

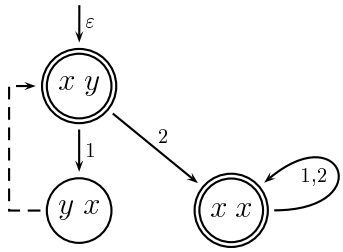
Anmerkung. In diesem Beispiel erzeugen wir einen P-Cluster der Tiefe zwei. Wir vermuten, daß wir mit analogen P-Automaten P-Cluster jeder endlicher Tiefe erzeugen können, jedoch existiert ein P-Cluster unendlicher Tiefe nach der Cluster-Proposition 4 nicht.

B.7 Entailment-Beispiel 7

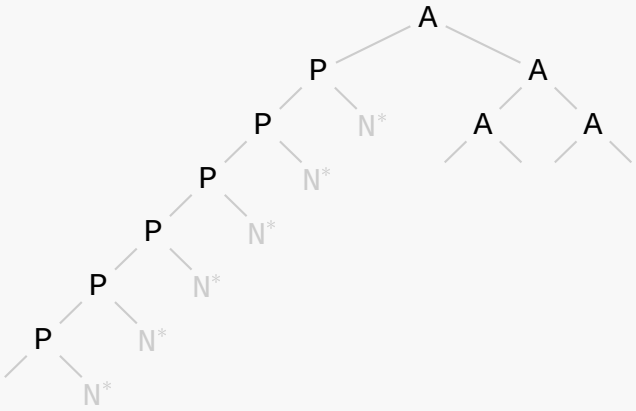
Entailment-Aussage. $\varphi_6 \ : \ x{\leq}y{\times}x \wedge x{\times}x{\leq}y \ \models x{\leq}y$ in NS_2 oder NS_2^{fin}

Widerlegung. Entailment kann an den Pfaden 12 mit der Belegung x nach $\bot{\times}(\top{\times}\bot)$ und y nach $\top{\times}(\bot{\times}\top)$ widerlegt werden.

Sicherheitsautomat.



Traversierungsbaum.



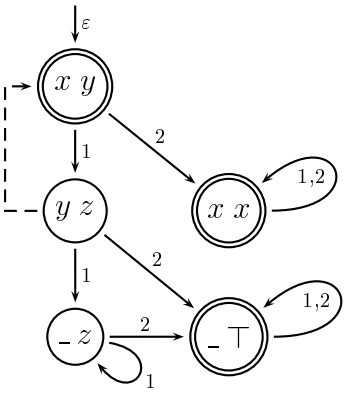
Anmerkung. Dieses Beispiel zeigt den einfachen Fall des Auftretens einer einzelnen P-Schleife. Es dient zum Vergleich mit dem nächsten Beispiel.

B.8 Entailment-Beispiel 8

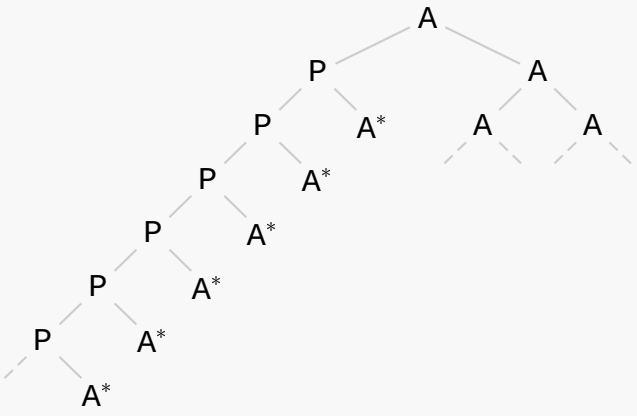
Entailment-Aussage. $\varphi_6 \ : \ x{\leq}y{\times}x \wedge z{\times}y{\leq}y \wedge z{\times}\top{\leq}z \ \models x{\leq}y$ in NS_Σ, NS_Σ^{fin}

Widerlegung. Entailment kann nicht widerlegt werden.

Erweiterter Sicherheitsautomat.



Traversierungsbaum.



Anmerkung. Dieses Beispiel geht über die in dieser Arbeit gezeigten Techniken hinaus. In den bisherigen Beispielen konnten wir die P-Schleifen einfach ignorieren, ohne die Gültigkeit der Entailment-Aussage zu beeinflussen. Hingegen beeinflußt in diesem Beispiel die P-Schleife die Gültigkeit der Entailment-Aussage. Für dieses Beispiel haben wir die eingeschränkte Constraint-Sprache verlassen und erlauben nun auch $\top$ in der Syntax.

Anhang C

Abgeschlossenheit der Saturierung

In diesem Kapitel führen wir den noch ausstehenden Beweis aus dem Vollständigkeitsbeweis aus Kapitel 6: Wir beweisen, daß die aus φ gebaute Saturierung abgeschlossen und fehlerfrei ist.

Zunächst wiederholen wir nocheinmal aus dem Vollständigkeitsbeweis (Kapitel 6) die Saturierung $s(\varphi, \varrho)$ mit ihrer wichtigen Eigenschaft aus Lemma 15.

Definition 11 (Saturierung). Sei φ ein Constraint über $\Sigma = \{\bot, \top, \times\}$, $x, y \in V(\varphi)$ und $\varrho \in \{1, 2\}^*$. Für jedes $z \in \{x, y\}$ und $\pi \in dc(\{\varrho 1, \varrho 2\})$ sei q^z_π eine frische Variable und $W(\varphi, \varrho)$ die Menge aller dieser frischen Variablen. Die *Saturierung* $s(\varphi, \varrho)$ von φ am Pfad ϱ ist der Constraint minimaler Größe, welcher die folgenden Eigenschaften a–f erfüllt:

a. φ in $s(\varphi, \varrho)$.

b. Für alle $q^x_\pi \in W(\varphi, \varrho)$: $q^x_\pi = q^x_{\pi 1} \times q^x_{\pi 2}$ in $s_{x',y'}(\varphi, \varrho)$, falls $\pi < \varrho$.

c. Für alle $q^y_\pi \in W(\varphi, \varrho)$: $q^y_\pi = q^y_{\pi 1} \times q^y_{\pi 2}$ in $s_{x',y'}(\varphi, \varrho)$, falls $\pi \leq \varrho$.

d. Für alle $q^z_\pi \in W(\varphi, \varrho), u \in V(\varphi)$: $q^z_\pi \leq u$ in $s_{x',y'}(\varphi, \varrho)$, falls $\varphi \vdash z?\pi \leq^{pr} u$.

e. Für alle $q^z_\pi \in W(\varphi, \varrho), u \in V(\varphi)$: $u \leq q^z_\pi$ in $s_{x',y'}(\varphi, \varrho)$, falls $\varphi \vdash u \leq^{pr} z?\pi$.

f. Für alle $q^z_{o\pi}, q^{z'}_{o'\pi} \in W(\varphi, \varrho)$: $q^z_{o\pi} \leq q^{z'}_{o'\pi}$ in $s_{x',y'}(\varphi, \varrho)$, falls $\varphi \vdash z?o \leq^{pr} z'?o'$.

Lemma 15. Falls der Constraint φ unter NS_Σ (entsprechend NS^{fin}_Σ) abgeschlossen und fehlerfrei ist, dann ist auch die Saturierung $s(\varphi, \varrho)$ unter NS_Σ (entsprechend NS^{fin}_Σ) abgeschlossen und fehlerfrei.

Beweis. Die Saturierung $s(\varphi, \varrho)$ ist konsistent, da sowohl φ aufgrund der Einschränkung der Syntax als auch $s(\varphi, \varrho)$ aufgrund seiner Definition 11 keine Constraints mit $\top$ oder $\bot$ besitzen. Der Beweis, daß $s(\varphi, \varrho)$ abgeschlossen ist, genauer abgeschlossen unter *Reflexivität*, *Transitivität* und *Absteigen*, erfolgt durch die Lemmata 19, 20 und 22. Im Lemma 24 zeigen wir, daß $s(\varphi, \varrho)$ zykelfrei ist, falls φ zykelfrei ist.

In den nachfolgenden Abschnitten C.1 bis C.4 werden wir nun die vier oben erwähnten Lemmata einzeln beweisen.

C.1 Abschluß unter Reflexivität

Lemma 19. Falls der Constraint φ unter*Reflexivität* abgeschlossen ist, dann ist auch die Saturierung $s(\varphi, \varrho)$ unter*Reflexivität* abgeschlossen.

Beweis. Sei u eine beliebige Variable aus $V(s(\varphi, \varrho))$. Falls $u \in V(\varphi)$, dann ist $u{\leq}u$ in $s(\varphi, \varrho)$, da φ reflexiv ist und in $s(\varphi, \varrho)$ enthalten. Ansonsten ist $u{\leq}u$ aufgrund von Regel f in der Saturierung $s(\varphi, \varrho)$, da $\varphi \vdash z?\varepsilon{\leq}z$, $\varphi \vdash z{\leq}z?\varepsilon$ und somit auch $\varphi \vdash z?\varepsilon{\leq}z?\varepsilon$ gilt.

C.2 Abschluß unter Transitivität

Lemma 20. Falls der Constraint φ unter *Transitivität* und *Absteigen* abgeschlossen ist, dann ist auch die Saturierung $s(\varphi, \varrho)$ unter *Transitivität* abgeschlossen.

Wir bereiten den Beweis mit einer Analyse vor.

Lemma 21 (Analyse). Wir untersuchen den Aufbau der Saturierung $s(\varphi, \varrho)$: Seien u, v Variablen aus $V(\varphi)$ und q_π^z, $q_{\pi'}^{z'}$ Variablen aus $W(\varphi, \varrho)$. Constraints vom Muster $u{\leq}v$ werden nicht in $s(\varphi, \varrho)$ erzeugt, Constraints vom Muster $q_\pi^z{\leq}v$, $u{\leq}q_{\pi'}^{z'}$ oder $q_\pi^z{\leq}q_{\pi'}^{z'}$ werden durch genau einer Regel erzeugt, es gilt also:

1. Falls $u{\leq}v$ in $s(\varphi, \varrho)$, dann $u{\leq}v$ in φ.
2. Ein Constraint $q_\pi^z{\leq}v$ in $s(\varphi, \varrho)$ ist durch Regel d. erzeugt worden.
3. Ein Constraint $u{\leq}q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$ ist durch Regel e. erzeugt worden.
4. Ein Constraint $q_\pi^z{\leq}q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$ ist durch Regel f. erzeugt worden.

Beweis. Durch einfache Untersuchung der Definition 11 von $s(\varphi, \varrho)$.

Beweis von Lemma 20. Wir machen eine Fallunterscheidung:

1. Seien u, v, w Variablen aus $V(\varphi)$. Zu beweisen ist: Falls die Constraints $u{\leq}v$ und $v{\leq}w$ in $s(\varphi, \varrho)$ sind, dann ist auch $u{\leq}w$ in $s(\varphi, \varrho)$.

$$
\begin{array}{lrll}
 & u{\leq}v, v{\leq}w & \text{in } s(\varphi, \varrho) & \text{(Voraussetzung)} \\
\Rightarrow & u{\leq}v, v{\leq}w & \text{in } \varphi & \text{(Analyse-Lemma 21)} \\
\Rightarrow & u{\leq}w & \text{in } \varphi & (\varphi \text{ ist abgeschlossen unter Trans.}) \\
\Rightarrow & u{\leq}w & \text{in } s(\varphi, \varrho) & \text{(Regel a der Definition von } s(\varphi, \varrho)\text{)}
\end{array}
$$

2. Seien v aus $V(\varphi)$ und q_π^z, $q_{\pi'}^{z'}$ aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $q_\pi^z \leq v$ und $v \leq q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$ sind, dann ist auch $q_\pi^z \leq q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$.

	$q_\pi^z \leq v, v \leq q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$	$\varphi \vdash z?\pi \leq v, \varphi \vdash v \leq z'?\pi'$	(Analyse-Lemma 21)
$\Rightarrow$	$\varphi \vdash z?\pi \leq z'?\pi'$	(Definition $\vdash$)
$\Rightarrow$	$q_\pi^z \leq q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$	(Regel f der Definition von $s(\varphi, \varrho)$)

3. Seien u, v aus $V(\varphi)$ und q_π^z aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $u \leq v$ und $v \leq q_\pi^z$ in $s(\varphi, \varrho)$ sind, dann ist auch $u \leq q_\pi^z$ in $s(\varphi, \varrho)$.

		$u \leq v$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$	(1)	$u \leq v$ in φ	(Analyse-Lemma 21)
		$v \leq q_\pi^z$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$		$\varphi \vdash v \leq z?\pi$	(Analyse-Lemma 21)

Wir machen eine weitere Fallunterscheidung:

3.1. Fall $\pi = \varepsilon$:

	$\Rightarrow$		$\varphi \vdash v \leq z?\varepsilon$	(Fallunterscheidung)
	$\Rightarrow$	(2)	$v \leq z$ in φ	(Definition $\vdash$)
(1) $\wedge$ (2)	$\Rightarrow$		$u \leq z$ in φ	(φ ist abgeschlossen unter Trans.)
	$\Rightarrow$		$\varphi \vdash u \leq z?\varepsilon$	(Definition $\vdash$)
	$\Rightarrow$		$\varphi \vdash u \leq z?\pi$	(Fallunterscheidung)
	$\Rightarrow$		$u \leq q_\pi^z$ in $s(\varphi, \varrho)$	(Definition von $s(\varphi, \varrho)$)

3.2. Fall $\pi = \pi' 1$:

	$\Rightarrow$		$\varphi \vdash v \leq z?\pi' 1$	(Fallunterscheidung)
	$\Rightarrow$	(2)	$\exists r.\varphi \vdash r \leq z?\pi'$	
		$\wedge$(3)	$r = f(r_1, r_2)$ in φ	
		$\wedge$(4)	$v \leq r_1$ in φ	(Definition $\vdash$)
(1) $\wedge$ (4)	$\Rightarrow$	(5)	$u \leq r_1$ in φ	(φ ist abg. unter Trans.)
(2) $\wedge$ (3) $\wedge$ (5)	$\Rightarrow$		$\varphi \vdash u \leq z?\pi' 1$	(Definition $\vdash$)
	$\Rightarrow$		$\varphi \vdash u \leq z?\pi$	(Fallunterscheidung)
	$\Rightarrow$		$u \leq q_\pi^z$ in $s(\varphi, \varrho)$	(Definition von $s(\varphi, \varrho)$)

3.3. Fall $\pi = \pi' 2$: Der Beweis dieses Falles ist analog zu dem Beweis von Fall 3.2

4. Dieser Fall ist ähnlich zu Fall 3, aber nicht analog. Seien v, w aus $V(\varphi)$ und q_π^z aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $q_\pi^z \leq v$ und $v \leq w$ in $s(\varphi, \varrho)$ sind, dann ist auch $q_\pi^z \leq w$ in $s(\varphi, \varrho)$.

		$v \leq w$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$	(1)	$v \leq w$ in φ	(Analyse-Lemma 21)
		$q_\pi^z \leq v$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$		$\varphi \vdash z?\pi \leq v$	(Analyse-Lemma 21)

Wir machen eine weitere Fallunterscheidung.

4.1. Fall $\pi = \varepsilon$:

	$\Rightarrow$		$\varphi \vdash z?\varepsilon \leq v$	(Fallunterscheidung)
	$\Rightarrow$	(2)	$z \leq v$ in φ	(Definition $\vdash$)
$(2) \wedge (1)$	$\Rightarrow$		$z \leq w$ in φ	(φ ist abg. unter Trans.)
	$\Rightarrow$		$\varphi \vdash z?\varepsilon \leq w$	(Definition $\vdash$)
	$\Rightarrow$		$\varphi \vdash z?\pi \leq w$	(Fallunterscheidung)
	$\Rightarrow$		$q_\pi^z \leq w$ in $s(\varphi, \varrho)$	(Definition von $s(\varphi, \varrho)$)

4.2. Fall $\pi = \pi' 1$:

	$\Rightarrow$		$\varphi \vdash z?\pi' 1 \leq v$	(Fallunterscheidung)
	$\Rightarrow$	(2)	$\exists r. \varphi \vdash z?\pi' \leq r$	
		$\wedge(3)$	$r = f(r_1, r_2)$ in φ	
		$\wedge(4)$	$r_1 \leq v$ in φ	(Definition $\vdash$)
$(4) \wedge (1)$	$\Rightarrow$	(5)	$r_1 \leq w$ in φ	(φ ist abg. unter Trans.)
$(2) \wedge (3) \wedge (5)$	$\Rightarrow$		$\varphi \vdash z?\pi' 1 \leq w$	(Definition $\vdash$)
	$\Rightarrow$		$\varphi \vdash z?\pi \leq w$	(Fallunterscheidung)
	$\Rightarrow$		$q_\pi^z \leq w$ in $s(\varphi, \varrho)$	(Definition von $s(\varphi, \varrho)$)

4.3. Fall $\pi = \pi' 2$: Der Beweis dieses Falles ist analog zu dem Beweis von Fall 4.2.

5. Seien u, w aus $V(\varphi)$ und q_π^z aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $u \leq q_\pi^z$ und $q_\pi^z \leq w$ in $s(\varphi, \varrho)$ sind, dann ist auch $u \leq w$ in $s(\varphi, \varrho)$.

		$u \leq q_\pi^z$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$	(1)	$\varphi \vdash u \leq z?\pi$	(Analyse-Lemma 21)
		$q_\pi^z \leq w$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$	(2)	$\varphi \vdash z?\pi \leq w$	(Analyse-Lemma 21)

Wir setzen diesen Fall mit einer Induktion über den Pfad π fort.

Induktionshypothese. $u \leq v$ in $s(\varphi, \varrho)$ folgt nach (1) und (2).

Induktionsanfang $\pi = \varepsilon$**.**

(1)	$\Rightarrow$		$\varphi \vdash u{\leq}z?\varepsilon$	(dieser Induktionsfall)
	$\Rightarrow$	$(1')$	$u{\leq}z$ in φ	(Definition $\vdash$)
(2)	$\Rightarrow$		$\varphi \vdash z?\varepsilon{\leq}w$	(Fallunterscheidung)
	$\Rightarrow$	$(2')$	$z{\leq}w$ in φ	(Definition $\vdash$)
$(1') \wedge (2')$	$\Rightarrow$		$u{\leq}w$ in φ	(φ ist abg. unter Trans.)
	$\Rightarrow$		$u{\leq}w$ in $s(\varphi, \varrho)$	(Regel a Definition $s(\varphi, \varrho)$)

Induktionsschritt von π **nach** $\pi 1$**.** Sei $\pi = \pi' 1$.

(1)	$\Rightarrow$		$\varphi \vdash u{\leq}z?\pi' 1$	(dieser Induktionsfall)
	$\Rightarrow$	(3)	$\exists p.\varphi \vdash p{\leq}z?\pi'$	
		$\wedge(4)$	$p{=}f(p_1, p_2)$ in φ	
		$\wedge(5)$	$u{\leq}p_1$ in φ	(Definition $\vdash$)
(2)	$\Rightarrow$		$\varphi \vdash z?\pi' 1{\leq}w$	(Fallunterscheidung)
	$\Rightarrow$	(6)	$\exists r.\varphi \vdash z?\pi'{\leq}r$	
		$\wedge(7)$	$r{=}f(r_1, r_2)$ in φ	
		$\wedge(8)$	$r_1{\leq}w$ in φ	(Definition $\vdash$)
$(3) \wedge (6)$	$\Rightarrow$		$p{\leq}r$ in $s(\varphi, \varrho)$	(nach Ind.hypothese)
	$\Rightarrow$	(9)	$p{\leq}r$ in φ	(Lemma Ana. 21)
$(4) \wedge (9) \wedge (7)$	$\Rightarrow$	(10)	$p_1{\leq}r_1$ in φ	(φ ist abg. unter Abst.)
$(5) \wedge (10) \wedge (8)$	$\Rightarrow$		$u{\leq}w$ in φ	(φ ist abg. unter Trans.)
	$\Rightarrow$		$u{\leq}w$ in $s(\varphi, \varrho)$	(Regel a Def. $s(\varphi, \varrho)$)

Induktionsschritt von π **nach** $\pi 2$**.** Dieser Schritt ist analog zu dem vorangegangenen Induktionsschritt.

6. Seien die Variablen w aus $V(\varphi)$ und q_π^z, $q_{\pi'}^{z'}$ aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $q_\pi^z{\leq}q_{\pi'}^{z'}$ und $q_{\pi'}^{z'}{\leq}w$ in $s(\varphi, \varrho)$ sind, dann ist auch $q_\pi^z{\leq}w$ in $s(\varphi, \varrho)$.

		$q_\pi^z{\leq}q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$		$\varphi \vdash z?\sigma{\leq}z'?\sigma'$	(Analyse-Lemma 21)
	$\wedge(1)$	$\sigma\tau{=}\pi \wedge \sigma'\tau{=}\pi'$	(Regel f von Sat.)
$\Rightarrow$	(2)	$\exists p.\varphi \vdash z?\sigma{\leq}p$	
	$\wedge(3)$	$\varphi \vdash p{\leq}z'?\sigma'$	(Definition $\vdash$)
	(4)	$q_{\pi'}^{z'}{\leq}w$ in $s(\varphi, \varrho)$	(Voraussetzung)

Wir setzen diesen Fall mit einer Induktion über den Pfad τ fort.

Induktionshypothese $q_\pi^z{\leq}w$ in $s(\varphi, \varrho)$ folgt nach (1), (2), (3) und (4).

Induktionsanfang $\tau = \varepsilon$.

(1)	$\Rightarrow$	$(1')$	$\sigma{=}\pi \wedge \sigma'{=}\pi'$	(dieser Induktionsschritt)
$(2) \wedge (1')$	$\Rightarrow$		$\varphi \vdash z?\pi{\leq}p$	
	$\Rightarrow$	$(2')$	$q_\pi^z{\leq}p$ in $s(\varphi, \varrho)$	(Regel d von Sat.)
$(3) \wedge (1')$	$\Rightarrow$		$\varphi \vdash p{\leq}z'?\pi'$	
	$\Rightarrow$	$(3')$	$p{\leq}q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$	(Regel e von Sat.)
$(3') \wedge (4)$	$\Rightarrow$	(5)	$p{\leq}w$ in $s(\varphi, \varrho)$	(nach Fall 5)
$(2') \wedge (5)$	$\Rightarrow$		$q_\pi^z{\leq}w$ in $s(\varphi, \varrho)$	(nach Fall 4)

Induktionsschritt von τ nach $\tau 1$. Sei τ gleich $\tau' 1$.

(1)	$\Rightarrow$	$(1')$	$\sigma\tau' 1{=}\pi \wedge \sigma'\tau' 1{=}\pi'$	(dieser Induk-) tionsschritt)
$(1')$	$\Rightarrow$	(1^*)	$\sigma\tau'{=}\pi_* \wedge \sigma'\tau'{=}\pi'_*$	(Def. von π_*, π'_*)
$(4) \wedge (1')$	$\Rightarrow$		$q_{\sigma'\tau' 1}^{z'}{\leq}w$ in $s(\varphi, \varrho)$	
	$\Rightarrow$		$\varphi \vdash z'?\sigma'\tau' 1{\leq}w$	(Lemma 21)
	$\Rightarrow$	$(4')$	$\exists r.\varphi \vdash z'?\sigma'\tau'{\leq}r$	
		$\wedge(5)$	$r{=}f(r_1, r_2)$ in φ	
		$\wedge(6)$	$r_1{\leq}w$ in φ	(Definition $\vdash$)
$(4')$	$\Rightarrow$	$(4'')$	$q_{\sigma'\tau'}^{z'}{\leq}w$ in $s(\varphi, \varrho)$	(Regel d von Sat.)
$(4'') \wedge (1^*)$	$\Rightarrow$	(4^*)	$q_{\pi'_*}^{z'}{\leq}w$ in $s(\varphi, \varrho)$	
$(1^*) \wedge (2) \wedge (3) \wedge (4^*)$	$\Rightarrow$		$q_{\pi_*}^z{\leq}r$ in $s(\varphi, \varrho)$	(Ind. Hypothese)
	$\Rightarrow$	(7)	$\varphi \vdash z?\pi_*{\leq}r$	(Lemma 21)
$(7) \wedge (5) \wedge (6)$	$\Rightarrow$	$(7')$	$\varphi \vdash z?\pi_* 1{\leq}w$	(Definition $\vdash$)
$(7') \wedge (1^*)$	$\Rightarrow$	$(7'')$	$\varphi \vdash z?\sigma\tau' 1{\leq}w$	
$(7'') \wedge (1')$	$\Rightarrow$		$\varphi \vdash z?\pi{\leq}w$	
	$\Rightarrow$		$q_\pi^z{\leq}w$ in $s(\varphi, \varrho)$	(Regel d von Sat.)

Induktionsschritt von τ nach $\tau 2$. Dieser Schritt ist analog zu dem vorherigen Induktionsschritt.

7. Seien u aus $V(\varphi)$ und q_π^z, $q_{\pi'}^{z'}$ aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $u{\leq}q_\pi^z$ und $q_\pi^z{\leq}q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$ sind, dann ist auch $u{\leq}q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$.

   Der Beweis ist analog zum Beweis von Fall 6.

8. Seien q_π^z, $q_{\pi'}^{z'}$ und $q_{\pi''}^{z''}$ aus $W(\varphi, \varrho)$. Zu beweisen ist: Falls die Constraints $q_\pi^z{\leq}q_{\pi'}^{z'}$ und $q_{\pi'}^{z'}{\leq}q_{\pi''}^{z''}$ in $s(\varphi, \varrho)$ sind, dann ist auch $q_\pi^z{\leq}q_{\pi''}^{z''}$ in $s(\varphi, \varrho)$.

		$q_\pi^z \leq q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$		$\varphi \vdash z?\sigma_1 \leq z'?\sigma_2$	(Analyse-Lemma 21)
	$\wedge(1)$	$\sigma_1\tau = \pi \wedge \sigma_2\tau = \pi'$	(Regel f der Sat.)
$\Rightarrow$	(2)	$\exists p.\varphi \vdash z?\sigma_1 \leq p$	
	$\wedge(3)$	$\varphi \vdash p \leq z'?\sigma_2$	(Definition $\vdash$)
		$q_{\pi'}^{z'} \leq q_{\pi''}^{z''}$ in $s(\varphi, \varrho)$	(Voraussetzung)
$\Rightarrow$		$\varphi \vdash z'?\sigma_3 \leq z''?\sigma_4$	(Analyse-Lemma 21)
	$\wedge(4)$	$\sigma_3\tau' = \pi' \wedge \sigma_4\tau' = \pi''$	(Regel f der Sat.)
$\Rightarrow$	(5)	$\exists r.\varphi \vdash z'?\sigma_3 \leq r$	
	$\wedge(6)$	$\varphi \vdash r \leq z''?\sigma_4$	(Definition $\vdash$)

Wir machen eine weitere Fallunterscheidung:

8.1. Es gibt einen Pfad π_* mit $\tau = \pi_*\tau'$.

(1)	$\Rightarrow$	$(1')$	$\sigma_1\pi_*\tau' = \pi \wedge \sigma_2\pi_*\tau' = \pi'$	(Voraussetzung)
$(1') \wedge (4)$	$\Rightarrow$	(7)	$\sigma_3 = \sigma_2\pi_*$	
			$q_\pi^z \leq q_{\pi'}^{z'}$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$		$\varphi \vdash z?\sigma_1 \leq z'?\sigma_2$	(Lemma 21)
	$\Rightarrow$	(8)	$q_{\sigma_1\pi_*}^z \leq q_{\sigma_2\pi_*}^{z'}$ in $s(\varphi, \varrho)$	
		falls	$q_{\sigma_1\pi_*}^z, q_{\sigma_2\pi_*}^{z'} \in W(\varphi, \varrho)$	(Regel f der Sat.)
$(7) \wedge (8)$	$\Rightarrow$	$(8')$	$q_{\sigma_1\pi_*}^z \leq q_{\sigma_3}^{z'}$ in $s(\varphi, \varrho)$	
		falls	$q_{\sigma_1\pi_*}^z, q_{\sigma_3}^{z'} \in W(\varphi, \varrho)$	
(5)	$\Rightarrow$	$(5')$	$q_{\sigma_3}^{z'} \leq r$ in $s(\varphi, \varrho)$	(Regel d der Sat.)
		$\wedge$	$q_{\sigma_3}^{z'} \in W(\varphi, \varrho)$	
(6)	$\Rightarrow$	$(6')$	$r \leq q_{\sigma_4}^{z''}$ in $s(\varphi, \varrho)$	(Regel e der Sat.)
$(8') \wedge (5')$	$\Rightarrow$	(9)	$q_{\sigma_1\pi_*}^z \leq r$ in $s(\varphi, \varrho)$	(nach Fall 6)
		falls	$q_{\sigma_1\pi_*}^z \in W(\varphi, \varrho)$	
$(9) \wedge (6')$	$\Rightarrow$		$q_{\sigma_1\pi_*}^z \leq q_{\sigma_4}^{z''}$ in $s(\varphi, \varrho)$	(nach Fall 2)
		falls	$q_{\sigma_1\pi_*}^z \in W(\varphi, \varrho)$	
	$\Rightarrow$		$q_{\sigma_1\pi_*\tau'}^z \leq q_{\sigma_4\tau'}^{z''}$ in $s(\varphi, \varrho)$	(Lemma 21,
		falls	$q_{\sigma_1\pi_*}^z, q_{\sigma_1\pi_*\tau'}^z, q_{\sigma_4\tau'}^{z''} \in W(\varphi, \varrho)$	Regel f der Sat.)
	$\Rightarrow$	(10)	$q_{\sigma_1\pi_*\tau'}^z \leq q_{\sigma_4\tau'}^{z''}$ in $s(\varphi, \varrho)$	($W(\varphi, \varrho)$ ist
		falls	$q_{\sigma_1\pi_*\tau'}^z, q_{\sigma_4\tau'}^{z''} \in W(\varphi, \varrho)$	Domänen-abg.)
$(10) \wedge (1')$	$\Rightarrow$	(11)	$q_\pi^z \leq q_{\sigma_4\tau'}^{z''}$ in $s(\varphi, \varrho)$	
		falls	$q_\pi^z, q_{\sigma_4\tau'}^{z''} \in W(\varphi, \varrho)$	
$(11) \wedge (4)$	$\Rightarrow$	(12)	$q_\pi^z \leq q_{\pi''}^{z''}$ in $s(\varphi, \varrho)$	
		falls	$q_\pi^z, q_{\pi''}^{z''} \in W(\varphi, \varrho)$	
		(13)	$q_\pi^z, q_{\pi'}^{z'}, q_{\pi''}^{z''} \in W(\varphi, \varrho)$	(Voraussetzung)
$(12) \wedge (13)$	$\Rightarrow$		$q_\pi^z \leq q_{\pi''}^{z''}$ in $s(\varphi, \varrho)$	

8.2. Es gibt einen Pfad π_* mit $\tau' = \pi_*\tau$. Dieser Fall ist analog zum Fall 8.1.

8.3. Es gibt keinen Pfad π_* mit $\tau = \pi_*\tau'$ oder $\tau' = \pi_*\tau$. Dies ist ein Widerspruch zu (1) und (4); somit kann dieser Fall nicht auftreten.

C.3 Abschluß unter Absteigen

Lemma 22. Falls der Constraint φ unter *Absteigen* und *Reflexivität* abgeschlossen ist, dann ist auch die Saturierung $s(\varphi, \varrho)$ unter *Absteigen* abgeschlossen.

Wir bereiten den Beweis mit einer weiteren Analyse vor.

Lemma 23 (Analyse). Wir untersuchen den Aufbau der Saturierung $s(\varphi, \varrho)$: Sei u eine Variable aus $V(\varphi)$ und q^z_π eine Variable aus $W(\varphi, \varrho)$. Es gilt:

1. Ein Constraint $u{=}f(\_, \_)$ ist genau dann in $s(\varphi, \varrho)$, falls $u{=}f(\_, \_)$ auch in φ ist.

2. Ein Constraint $q^z_\pi{=}f(\_, \_)$ in $s(\varphi, \varrho)$ ist durch Regel b. oder c. erzeugt worden und besitzt die Form $q^z_\pi{=}f(q^z_{\pi 1}, q^z_{\pi 2})$.

Beweis. Durch einfache Untersuchung der Definition 11 von $s(\varphi, \varrho)$.

Beweis von Lemma 22. Wir machen eine Fallunterscheidung:

1. Seien u, v Variablen aus $V(\varphi)$. Zu beweisen ist: Falls die Constraints $u{=}f(u_1, u_2)$, $v{=}f(v_1, v_2)$ und $u{\leq}v$ in $s(\varphi, \varrho)$ sind, dann sind auch $u_1{\leq}v_1$ und $u_2{\leq}v_2$ in $s(\varphi, \varrho)$.

			$u{\leq}v$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(1)	$u{\leq}v$ in φ	(nach Lemma 21)
			$u{=}f(u_1, u_2)$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(2)	$u{=}f(u_1, u_2)$ in φ	(nach Lemma 23)
			$v{=}f(v_1, v_2)$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(3)	$v{=}f(v_1, v_2)$ in φ	(nach Lemma 23)
$(2) \wedge (1) \wedge (3)$	$\Rightarrow$		$u_1{\leq}v_1, u_2{\leq}v_2$ in φ	(φ ist unter Absteigen abg.)
	$\Rightarrow$		$u_1{\leq}v_1, u_2{\leq}v_2$ in $s(\varphi, \varrho)$	(Regel a der Saturierung)

2. Seien die Variablen u aus $V(\varphi)$ und q^z_π aus $W(\varphi, \varrho)$. Falls der Constraint $q^z_\pi{=}f(\_, \_)$ in $s(\varphi, \varrho)$ ist, dann besitzt er die Form $q^z_\pi{=}f(q^z_{\pi 1}, q^z_{\pi 2})$ nach Lemma 23. Zu beweisen ist: Falls die Constraints $u{=}f(u_1, u_2)$, $q^z_\pi{=}f(q^z_{\pi 1}, q^z_{\pi 2})$ und $u{\leq}q^z_\pi$ in $s(\varphi, \varrho)$ sind, dann sind auch $u_1{\leq}q^z_{\pi 1}$ und $u_2{\leq}q^z_{\pi 2}$ in $s(\varphi, \varrho)$.

			$u \leq q^z_\pi$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(1)	$\varphi \vdash u \leq z?\pi$	(nach Lemma 21)
			$u = f(u_1, u_2)$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(2)	$u = f(u_1, u_2)$ in φ	(nach Lemma 23)
	$\Rightarrow$		$u_1, u_2 \in V(\varphi)$	
	$\Rightarrow$	(3)	$u_1 \leq u_1$	(φ ist abg. unter Refl.)
		$\wedge$(4)	$u_2 \leq u_2$	(φ ist abg. unter Refl.)
(1) $\wedge$ (2) $\wedge$ (3)	$\Rightarrow$		$\varphi \vdash u \leq z?\pi 1$	(Definition $\vdash$)
	$\Rightarrow$		$u \leq q^z_{\pi 1}$ in $s(\varphi, \varrho)$	(Regel e der Sat.)
(1) $\wedge$ (2) $\wedge$ (4)	$\Rightarrow$		$\varphi \vdash u \leq z?\pi 2$	(Definition $\vdash$)
	$\Rightarrow$		$u \leq q^z_{\pi 2}$ in $s(\varphi, \varrho)$	(Regel e der Sat.)

3. Seien die Variablen v aus $V(\varphi)$ und q^z_π aus $W(\varphi, \varrho)$. Falls der Constraint $q^z_\pi = f(\_, \_)$ in $s(\varphi, \varrho)$ ist, dann besitzt er die Form $q^z_\pi = f(q^z_{\pi 1}, q^z_{\pi 2})$ nach Lemma 23. Zu beweisen ist: Falls die Constraints $v = f(v_1, v_2)$, $q^z_\pi = f(q^z_{\pi 1}, q^z_{\pi 2})$ und $q^z_\pi \leq v$ in $s(\varphi, \varrho)$ sind, dann sind auch $q^z_{\pi 1} \leq v_1$ und $q^z_{\pi 2} \leq v_2$ in $s(\varphi, \varrho)$.

   Der Beweis ist analog zu Fall 2.

4. Seien die Variablen q^z_π und $q^{z'}_{\pi'}$ aus $W(\varphi, \varrho)$. Falls der Constraint $q^z_\pi = f(\_, \_)$ in $s(\varphi, \varrho)$ ist, dann besitzt er die Form $q^z_\pi = f(q^z_{\pi 1}, q^z_{\pi 2})$ nach Lemma 23. Falls der Constraint $q^{z'}_{\pi'} = f(\_, \_)$ in $s(\varphi, \varrho)$ ist, dann besitzt er die Form $q^{z'}_{\pi'} = f(q^{z'}_{\pi' 1}, q^{z'}_{\pi' 2})$ nach Lemma 23.

   Zu beweisen ist: Falls die Constraints $q^z_\pi = f(q^z_{\pi 1}, q^z_{\pi 2})$, $q^{z'}_{\pi'} = f(q^{z'}_{\pi' 1}, q^{z'}_{\pi' 2})$ und $q^z_\pi \leq q^{z'}_{\pi'}$ in $s(\varphi, \varrho)$ sind, dann sind auch $q^z_{\pi 1} \leq q^{z'}_{\pi' 1}$ und $q^z_{\pi 2} \leq q^{z'}_{\pi' 2}$ in $s(\varphi, \varrho)$.

			$q^z_\pi \leq q^{z'}_{\pi'}$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(1)	$\varphi \vdash z?\sigma_1 \leq z'?\sigma_2$	(nach Analyse 21)
		$\wedge$(2)	$\sigma_1\tau = \pi \wedge \sigma_2\tau = \pi'$	(Regel f der Sat.)
(1)	$\Rightarrow$	(3)	$q^z_{\sigma_1\tau 1} \leq q^{z'}_{\sigma_2\tau 1}$ in $s(\varphi, \varrho)$	
		falls	$q^z_{\sigma_1\tau 1}, q^{z'}_{\sigma_2\tau 1} \in W(\varphi, \varrho)$	(Regel f der Sat.)
(3) $\wedge$ (2)	$\Rightarrow$	(4)	$q^z_{\pi 1} \leq q^{z'}_{\pi' 1}$ in $s(\varphi, \varrho)$	
		falls	$q^z_{\pi 1}, q^{z'}_{\pi' 1} \in W(\varphi, \varrho)$	
(1)	$\Rightarrow$	(5)	$q^z_{\sigma_1\tau 2} \leq q^{z'}_{\sigma_2\tau 2}$ in $s(\varphi, \varrho)$	
		falls	$q^z_{\sigma_1\tau 2}, q^{z'}_{\sigma_2\tau 2} \in W(\varphi, \varrho)$	(Regel f der Sat.)
(5) $\wedge$ (2)	$\Rightarrow$	(6)	$q^z_{\pi 2} \leq q^{z'}_{\pi' 2}$ in $s(\varphi, \varrho)$	
		falls	$q^z_{\pi 2}, q^{z'}_{\pi' 2} \in W(\varphi, \varrho)$	
			$q^z_\pi = f(q^z_{\pi 1}, q^z_{\pi 2})$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(7)	$q^z_{\pi 1}, q^z_{\pi 2} \in W(\varphi, \varrho)$	(Regel b, c der Sat.)
			$q^{z'}_{\pi'} = f(q^{z'}_{\pi' 1}, q^{z'}_{\pi' 2})$ in $s(\varphi, \varrho)$	(Voraussetzung)
	$\Rightarrow$	(8)	$q^{z'}_{\pi' 1}, q^{z'}_{\pi' 2} \in W(\varphi, \varrho)$	(Regel b, c der Sat.)
(7) $\wedge$ (8) $\wedge$ (4)	$\Rightarrow$		$q^z_{\pi 1} \leq q^{z'}_{\pi' 1}$ in $s(\varphi, \varrho)$	
(7) $\wedge$ (8) $\wedge$ (6)	$\Rightarrow$		$q^z_{\pi 2} \leq q^{z'}_{\pi' 2}$ in $s(\varphi, \varrho)$	

C.4 Saturierung ist zykelfrei

Der Beweis für die Abgeschlossenheit unter *Reflexivität, Transitivität* und *Absteigen* sind zwar langwierig, aber technisch doch einfach. Der Grund hierfür ist, daß die Definition der Saturierung 11 genau auf den Abschluß unter diesen drei Eigenschaften ausgelegt worden ist. Für den *Occur-Test* ist sie hingegen nicht extra ausgelegt worden. Zum weiteren ist der *Occur-Test* gegenüber den anderen drei Abschlüssen eine negative Eigenschaft, in dem Sinne, daß der *Occur-Test* bestimmte Constraints in der Saturierung nicht verlangt, sondern verbietet.

Für den Beweis der Zykelfreiheit der Saturierung gehen wir wie folgt vor: Wir bauen aus dem gegebenen Constraint φ einen neue Saturierung $s^*(\varphi)$, mit der Eigenschaft, daß $s^*(\varphi)$ zykelfrei ist. Anschließend ist nur noch zu beweisen, daß $s(\varphi, \varrho)$ in $s^*(\varphi)$ liegt. Da der *Occur-Test* eine negative Eigenschaft ist, ist somit auch $s(\varphi, \varrho)$ zykelfrei.

Lemma 24. Falls der Constraint φ zykelfrei ist, dann ist auch die Saturierung $s(\varphi, \varrho)$ zykelfrei.

Wir definieren zunächst eine Erweiterung der Saturierung 11, die speziell auf die Zykelfreiheit ausgerichtet ist:

Definition 13 (Occur-Test-Saturierung). Gegeben sei ein Constraint φ. Für alle Variablen $z \in V(\varphi)$ und alle Pfade $\pi \in \{1,2\}^*$ sei q^z_π eine frische Variable und $W^*(\varphi)$ die Menge aller dieser frischen Variablen, i.e. $W^*(\varphi) = \{q^z_\pi \mid z \in V(\varphi), \pi \in \{1,2\}^*\}$. Die *Occur-Test-Saturierung* $s^*(\varphi)$ von φ ist der Constraint minimaler Größe, welcher die folgenden Eigenschaften a–e erfüllt:

a. φ in $s^*(\varphi)$.

b. Für alle $q^z_\pi \in W^*(\varphi)$: $q^z_\pi = f(q^z_{\pi 1}, q^z_{\pi 2})$ in $s^*(\varphi)$.

c. Für alle $q^z_\pi \in W^*(\varphi), u \in V(\varphi)$: $q^z_\pi \leq u$ in $s^*(\varphi)$, falls $\varphi \vdash z?\pi \leq u$.

d. Für alle $q^z_\pi \in W^*(\varphi), u \in V(\varphi)$: $u \leq q^z_\pi$ in $s^*(\varphi)$, falls $\varphi \vdash u \leq z?\pi$.

e. Für alle $q^z_{o\pi}, q^{z'}_{o'\pi} \in W^*(\varphi)$: $q^z_{o\pi} \leq q^{z'}_{o'\pi}$ in $s^*(\varphi)$, falls $\varphi \vdash z?o \leq z'?o'$, $\pi \in \{1,2\}^*$.

Falls nun ein Widerspruch zum *Occur-Test* in der Saturierung $s^*(\varphi)$ liegt, dann liegt dieser auch in der Saturierung $s(\varphi, \varrho)$, da offensichtlich nach Definition 13 die Saturierung $s(\varphi, \varrho)$ in $s^*(\varphi)$ liegt. Anstelle des Beweises von Lemma 24 ist somit der Beweis des folgenden Lemmas hinreichend:

Lemma 25. Falls der Constraint φ zykelfrei ist, dann ist auch die Saturierung $s^*(\varphi)$ zykelfrei.

Beweis. Wir beweisen die Gegenrichtung: Falls $s^*(\varphi)$ nicht zykelfrei ist, dann ist es auch nicht der Constraint φ. Der *Occur-Test* lautet $\psi \not\vdash x.\pi{=}x$ mit $\pi \neq \varepsilon$. Zu zeigen ist also:

1. Falls $s^*(\varphi) \vdash z{\leq}z?\pi$, dann $\varphi \vdash z{\leq}z?\pi$ mit $\pi{\neq}\varepsilon$.
2. Falls $s^*(\varphi) \vdash z?\pi{\leq}z$, dann $\varphi \vdash z?\pi{\leq}z$ mit $\pi{\neq}\varepsilon$.

Wir beweisen Punkt 1, der Beweis von Punkt 2 läuft analog.

Sei $s^*(\varphi)$ nicht zykelfrei; es gibt also eine Variable $z \in V(s^*(\varphi))$ und ein Pfad $\pi{\neq}\varepsilon$, so daß $s^*(\varphi) \vdash z{\leq}z?\pi$. Falls z aus $V(\varphi)$, dann liegt nach Regel c und d der Constraint $q^z_\varepsilon{\leq}z \wedge z{\leq}q^z_\varepsilon$ in $s^*(\varphi)$, und somit gilt nach der Transitivität 26 die Ableitung $s^*(\varphi) \vdash q^z_\varepsilon{\leq}q^z_\varepsilon?\pi$. Ansonsten war z schon aus der Menge der in $s^*(\varphi)$ neu eingeführten Variablen $W^*(\varphi)$.

Es gibt also eine Variable q^z_σ aus $W^*(\varphi)$ und ein Pfad $\pi{\neq}\varepsilon$, so daß $s^*(\varphi) \vdash q^z_\sigma{\leq}q^z_\sigma?\pi$. Dann gilt aber nach dem folgenden Lemma 27 auch $s^*(\varphi) \vdash q^z_\sigma{\leq}q^z_{\sigma\pi}$. Nach Definition 13 Regel e folgt hieraus $\varphi \vdash z?o{\leq}z?o\tau$ und somit

$$\exists u \in V(\varphi).\quad \varphi \vdash z?o{\leq}u{\leq}u?\tau{\leq}z?o\tau.$$

Wir beweisen jetzt noch die im Beweis von Lemma 25 benötigten Eigenschaften der neuen Occur-Test-Saturierung (Definition 13):

Lemma 26. Falls der Constraint φ unter *Transitivität* und *Absteigen* abgeschlossen ist, dann ist auch die Saturierung $s^*(\varphi)$ unter *Transitivität* abgeschlossen.

Der Beweis verläuft analog zum Beweis von Lemma 20 (Transitivität der ersten Saturierung).

Lemma 27. Sei der Constraint φ abgeschlossen und z eine Variable aus $V(\varphi)$, dann gilt für alle Pfade σ und π aus $\{1,2\}^*$:

$$s^*(\varphi) \vdash q^z_\sigma{\leq}q^z_\sigma?\pi \quad\Longrightarrow\quad s^*(\varphi) \vdash q^z_\sigma{\leq}q^z_{\sigma\pi}$$

Beweis. Nach Regel b der Occur-Test-Saturierung (Definition 13) gilt $s^*(\varphi) \vdash q^z_\sigma.\pi{=}q^z_{\sigma\pi}$ und somit auch $s^*(\varphi) \vdash q^z_\sigma?\pi{\leq}q^z_{\sigma\pi}$. Hieraus folgt mit der Voraussetzung $s^*(\varphi) \vdash q^z_\sigma{\leq}q^z_\sigma?\pi$ durch die Transitivität (Lemma 26) die Ableitung $s^*(\varphi) \vdash q^z_\sigma{\leq}q^z_{\sigma\pi}$.

Anmerkung. Die obrige Eigenschaft Lemma 27 ist entscheidend für den Beweis von Lemma 25. In der ersten Saturierung $s(\varphi, \varrho)$ von φ gilt diese Eigenschaft *nicht*, da zum Beispiel für den Constraint $\varphi := z{=}z_1{\times}z_2 \wedge z{\leq}z_1$ die Variablen q^z_ε und q^z_1 in der Saturierung $s(\varphi, \varrho)$ nicht definiert sind. Diese Eigenschaft war der Grund, warum wir im Beweis für die Zykelfreiheit eine neue Saturierung benötigt haben.

Literaturverzeichnis

Ait-Kaci, H., Podelski, A., & Smolka, G. (1994). A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science, 122*(1–2), 263–283.

Amadio, R. M., & Cardelli, L. (1993). Subtyping recursive types. *ACM Transactions on Programming Languages and Systems, 15*(4), 575–631.

Büchi, J. R. (1964). *Regular canonical systems.* (Reprint in Saunders Mac Lane, Dirk Siefers, editors, The collection works of J. Richard Büchi, Springer-Verlag, 1990)

Cardelli, L. (1988). A semantics of multiple inheritance. *Information and Computation, 76*(2/3), 138–164. (Auch in 1984 Semantics of Data Types Symposium, LNCS 173, pages 51–66.)

Charatonik, W., & Podelski, A. (1997). Set constraints with intersection. In *Proceedings of the 12^{th} IEEE symposium on logic in computer science* (pp. 352–361). Warsaw, Poland.

Dörre, J. (1991). Feature logics with weak subsumption constraints. In *Annual meeting of the acl (association of computational logics)* (p. 256-263).

Dörre, J., & Rounds, W. C. (1990). On subsumption and semiunification in feature algebras. In *Proceedings of the 5^{th} IEEE symposium on logic in computer science* (p. 300-310).

Eifrig, J., Smith, S., & Trifonov, V. (1995). Sound polymorphic type inference for objects. *30*(10), 169–184.

Eifrig, J., Smith, S., & Trifonow, V. (1995). Type inference for recursively constrained types and its application to object-oriented programming. *Elec. Notes in Theoretical Computer Science, 1*.

Fuh, Y., & Mishra, P. (1990). Type inference with subtypes. *Theoretical Computer Science, 73*.

Henglein, F., & Rehof, J. (1997). The complexity of subtype entailment for simple types. In *Proceedings of the 12^{th} IEEE symposium on logic in computer science* (pp. 362–372). Warsaw, Poland.

Henglein, F., & Rehof, J. (1998). Constraint automata and the complexity of recursive subtype entailment. In *Proceedings of the 25^{th} int. conference on automata, languages, and programming.*

Kozen, D., Palsberg, J., & Schwartzbach, M. I. (1995). Efficient recursive subtyping. *Mathematical Structures in Computer Science, 5*, 1–13. (Auch in Proc. POPL'93, Seite 419–428)

Mitchell, J. C. (1984). Coercion and type inference. In *11th annual acm symposium on principles of programming languages* (pp. 175–185).

Mitchell, J. C. (1991). Type inference with simple subtypes. *The Journal of Functional Programming, 1*(3), 245–285.

Müller, M., Niehren, J., & Podelski, A. (2000). Ordering constraints over feature trees. *Constraints, an International Journal, Special Issue on CP'97, 5*(1–2).

Müller, M., Niehren, J., & Treinen, R. (1998). The first-order theory of ordering constraints over feature trees. In *IEEE symposium on logic in computer science* (pp. 432–443).

Niehren, J., Müller, M., & Talbot, J.-M. (1999). Entailment of atomic set constraints is PSPACE-complete. In *Fourteenth annual ieee symposium on logic in computer sience (lics99)* (pp. 285–294). Trento, Italy: IEEE Press.

Niehren, J., & Priesnitz, T. (1999a). Entailment of non-structural subtype constraints. In *Asian computing science conference* (Vol. 1742, pp. 251–265). Phuket, Thailand: Springer-Verlag, Berlin.

Niehren, J., & Priesnitz, T. (1999b). *Characterizing subtype entailment in automata theory* (Tech. Rep.). Universität des Saarlandes, Programming Systems Lab.

Pottier, F. (1996). Simplifying subtyping constraints. In *Proceedings of the ACM SIGPLAN international conference on functional programming* (pp. 122–133). ACM Press, New York.

Pottier, F. (1998a). A framework for type inference with subtyping. In *Proceedings of the third acm sigplan international conference on functional programming* (pp. 228–238).

Pottier, F. (1998b). *Type inference in the presence of subtyping: from theory to practice.* Doctoral dissertation, Institut de Recherche d'Informatique et d'Automatique.

Rehof, J. (1997). Minimal typings in atomic subtyping. In *ACM symposium on principles of programming languages.* ACM Press.

Rehof, J. (1998). *The complexity of simple subtyping systems.* Doctoral dissertation, DIKU, University of Copenhagen.

Smolka, G., & Treinen, R. (1994). Records for logic programming. *Journal of Logic Programming*, *18*(3), 229–258.

Trifonov, V., & Smith, S. (1996). Subtyping constrained types. In *Proceedings of the* 3^{rd} *international static analysis symposium* (Vol. 1145, pp. 349–365). Aachen.