Test-sets und Termersetzungen für die Generierung rekursiv definierter Algorithmen aus Existenzaussagen

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr. Ing.)
an der Naturwissenschaftlich-Technischen
Fakultät I der Universität
des Saarlandes

vorgelegt von

Idriss Bengeloune

Saarbrücken
Juli 2000

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Entwicklung von Softwaresystemen, die in der Lage sind, nicht nur Routine, sondern auch anspruchsvolle Aufgaben zu bewältigen, ist nach wie vor ein Ziel einiger Teilbereiche der Informatik und der Künstlichen Intelligenz. Zu diesen anspruchsvollen Aufgaben gehört die Programmsynthese, die sich die Automatisierung der Programmierung zum Ziel gesetzt hat. Was genau unter Programmsynthese zu verstehen ist, läßt sich mit den folgenden Sätzen zusammenfassen:

> The design and implementation of correct software meeting given requirements continues to be most relevant, practical, and scientifically challenging problem. There are many lines of research directed towards solving this problem. Automatic programing is one among those. It is the study of techniques for generating executable code from information which may be fragmentary and may only indirectly specify the target behavior.
>
> The field is based on the idea that ultimately we need to engage the machine itself in the process of programming machines, since only machines offer the important property needed for this task which is the ability to work without making mistakes. The "Automatic" in the name does not necessarily refer to a full automation of programming, but rather to a considerably higher degree of automation than in other lines of software production pursued by the literature.

Alan Biermann

Die automatische Unterstützung der Programmentwicklung einerseits und die Sicherstellung der Korrektheit der erstellten Programme andererseits sind die wichtigsten Charakteristika der Programmsynthese. Bei zunehmend internationalem Wettbewerb und wegen der Erschließung neuer Anwendungsgebiete gewinnt die Korrektheit von Software immer mehr an Bedeutung. Bei der formalen Softwareentwicklung besteht der Weg von der Spezifikation bis zum lauffähigen Programm

aus kleinen, mathematisch präzisen Beweisschritten, so daß eine maschinelle Unterstützung unabdingbar ist. Es sollen so viele Schritte wie möglich automatisch ablaufen, so daß der Bediener nur noch eingreift, wenn dies nötig ist.

Die vorliegende Arbeit hat die Automatisierung der Softwareentwicklung zum Gegenstand. Sie beschränkt sich auf einen einfachen Formalismus und zeigt, wie sich Programme in diesem Formalismus systematisch - also automatisierbar - generieren lassen. Damit sind uns Synthesen einfacher Programme gelungen, die bis in die allerletzten Details automatisiert werden können.

Wir stellen **(1)** ein Inferenzsystem vor, das uns ermöglicht, die Gültigkeit von Existenzformeln im initialen Modell einer Menge bedingter Gleichungen nachzuweisen. Es ist wohl bekannt, daß aus einem konstruktiven Beweis einer Existenzaussage ein Programm zur Berechnung des als existent nachgewiesenen Objekts extrahiert werden kann. Es wird **(2)** ein Verfahren präsentiert, mit dem Existenzsätze in bestimmten Fällen widerlegt werden können. Anschließend daran führen wir **(3)** einen Algorithmus zur Extraktion algorithmischer Definitionen für Skolemfunktionen ein.

Zu **(1)**: In unserem Ansatz wird Programmsynthese in erster Linie als ein Beweisproblem aufgefaßt (deduktive Programmsynthese). Ausgehend von Existenzaussagen, die man als formale Spezifikationen von Programmen ansehen kann, werden rekursive Programme generiert. Dabei wird die Prädikatenlogik erster Stufe sowohl als Spezifikation als auch als Programmiersprache verwendet.

Zur Repräsentation von Datenstrukturen und Programmen wird der Begriff der kanonischen erweiterten konstruktiven Spezifikationen (KEKS) eingeführt. In diesen werden die Eigenschaften der Datenstrukturen und Programme definiert. Eine wichtige Eigenschaft der KEKS ist, daß sie durch initiale Konstruktoralgebren gedeutet werden können. Auf dieser formalen Basis wird ein Verfahren entwickelt, mit dem man die Gültigkeit einer Existenzformel im initialen Modell einer KEKS nachweisen kann. Das Verfahren beruht auf Test-sets und einem Vereinfachungsmechanismus.

Test-sets stellen eine Beschreibung des initialen Modells einer KEKS dar. Sie eignen sich besonders für die Automatisierung von Induktionsschemata und für die Widerlegung von Formeln.

Die Grundidee des Vereinfachungsmechanismus besteht darin, Funktionsdefinitionen und bereits bewiesene Lemmata solange auf eine zu beweisende Formel anzuwenden, bis man eine Tautologie oder eine (bzgl. einer vorgegebenen noetherschen Ordnung) kleinere Instanz der zu beweisenden Aussage erhält. Dieser letzte Schritt entspricht der Anwendung einer Induktionshypothese bei einem klassischen Beweis durch Induktion.

Zur Automatisierung des Induktionsprinzips sind zwei Ansätze aus der Literatur bekannt: die explizite und die implizite Induktionsmethode. Letztere wird auch induktionslose Induktion genannt. Die Induktionsmethode, welche in dieser Arbeit verwendet wird, vereint die Vorteile der beiden erwähnten Methoden. Zum einen werden, wie bei der impliziten Induktionsbeweismethode, Lemmata automatisch generiert, so daß in vielen Fällen die Suche nach passenden Lemmata in Form

von Generalisierungen entfällt. Zum anderen hängt die Korrektheit des Verfahrens nicht mit der Konvergenz der betrachteten Axiommenge zusammen. Gleichungen, die nicht als Regeln gerichtet werden können, lassen sich in bestimmen Fällen anwenden, ohne die Terminierung des Verfahrens zu gefährden. Der Grund dafür ist, daß wir eine Erweiterung der klassischen Termersetzung verwenden, bei der bedingte Gleichungen, die nicht orientiert werden können, und die in der Prämisse einer zu beweisenden Implikationsformel enthaltenen Gleichungen in den Termersetzungsvorgang miteinbezogen werden. Außerdem ist zu jeder KEKS ein terminierendes Termersetzungssystem assoziiert, dessen Terminierungsordnung als Induktionsordnung für einen Beweis gewählt werden kann, falls der Benutzer keine vorgibt. Ein solcher Fall würde der von Reddy in [RED89] vorgeschlagenen Termersetzungsinduktion (engl. Term Rewriting Induction) entsprechen.

Zu (2): Von entscheidender Bedeutung für jedes Beweisverfahren ist es, möglichst früh Inkonsistenzen erkennen zu können. Somit können zum einen Fehler in der Beschreibung einer zu beweisenden Aussage gefunden und entfernt werden. Zum anderen wird der Suchraum erheblich dadurch reduziert, daß Sackgassen bzw. Irrwege, die oftmals unendlich lang sind, vermieden werden. In dieser Arbeit wird ein Verfahren eingeführt, das Existenzaussagen in bestimmten Fällen widerlegt.

Zu (3): Ausgehend von den Beweisen, die mittels des Inferenzsystems geführt wurden, generiert unser Extraktionsalgorithmus rekursive Programme zur Berechnung des als existent nachgewiesenen Objekts. Eine wichtige Rolle bei der Extraktion spielen einerseits die Markierungen der bewiesenen Formeln, in denen alle auf sie im Laufe des Beweises angewandten Substitutionen protokolliert werden. Andererseits bestimmt die letzte Inferenzregel, die auf einer Formel angewandt wurde, bevor sie als bewiesen gilt, die Form der vom Algorithmus erzeugten definierenden Gleichung. Die Anwendung der Induktionshypothese im Beweis entspricht z.B. einem rekursiven Aufruf des zu synthetisierenden Programms.

Die neu generierten Programme stellen nicht nur Lösungen für das gestellte Problem dar, sondern können auch als Erweiterung des Wissens, das in einem Synthesesystem bereits vorhanden ist, angesehen werden. Geeignetes Wissen zu benutzen, ist für ein Programmsynthesesystem von zentraler Bedeutung. Je mehr Wissen (in Form von Funktionsdefinitionen und Lemmata) ein System über die in der Problemspezifikation betrachtete Domäne verfügt, desto einfacher und übersichtlicher werden die Beweise und die aus ihnen extrahierten Programme.

Die vorliegende Arbeit ist in 11 Kapitel unterteilt.
Im zweiten Kapitel werden die logischen und algebraischen Grundlagen der Arbeit detailliert vorgestellt.

Daran anschließend werden im dritten Kapitel sowohl unbedingte als auch bedingte Termersetzungssysteme, die in dieser Arbeit eine zentrale Rolle spielen, eingeführt.

Kapitel 4 enthält die formale Definition der kanonischen erweiterten konstruktiven Spezifikationen. Syntax, Semantik und einige Eigenschaften der KEKS werden gezeigt.

Im Kapitel 5 wird eine Möglichkeit zur Beschreibung des initialen Modells einer KEKS präsentiert. Danach wird eine Methode zur Konstruktion von Test-sets beschrieben.

Ein allgemeiner Überblick über das Induktionsprinzip und die Programmsynthese wird in den Kapiteln 6 und 7 gegeben.

Ausgehend von dieser formalen Basis wird im Kapitel 8 zuerst ein informaler Überblick über unsere induktive Beweismethode für eine Klasse von Existenzformeln gegeben. Anschließend stellen wir das Inferenzsystem vor, mit dem sich die Gültigkeit von Existenzformeln im initialen Modell einer KEKS nachweisen läßt. Wir zeigen die Korrektheit des Inferenzsystems und behandlen einige Beispiele.

Bevor der Algorithmus zur Extraktion von Programmen aus Beweisen präsentiert, und seine Korrektheit im Kapitel 10 bewiesen wird, führen wir im Kapitel 9 ein Widerlegungsverfahren für Existenzformeln ein.

Im letzten Kapitel schließen wir mit einer Zusammenfassung und einem Ausblick auf Forschungsarbeiten ab, die auf unseren Ergebnissen aufbauen könnten.

Kapitel 2

Logische und algebraische Grundlagen

In diesem Kapitel werden wir einige grundlegende Begriffe und Techniken einführen, die später für unterschiedliche Anwendungen instanziiert und verfeinert werden. Wir orientieren uns dabei an den formalen Begriffen der Prädikatenlogik erster Stufe, wie sie in [LS87] und [LEW96] dargestellt wurden. Die im weiteren in diesem Kapitel verwendeten Grundbegriffe stammen aus klassischen Arbeiten, wie etwa [DER85], [BUR69], [HUH82], [KR90] und werden hier ausführlich vorgestellt.

2.1 Signaturen

Definition 1 (Signaturen) Eine Signatur ist ein Paar (S, Ω), wobei die Elemente von S Sorten und die von Ω Operationszeichen genannt werden. Jedes Operationszeichen ist ein $(k+2)$-Tupel

$$n : s_1 \times \ldots \times s_k \rightarrow s,$$

wobei $s_1, \ldots, s_k \in S$ und $k \geq 0$; n ist der Operationsname und $n : s_1 \times \ldots \times s_k \rightarrow s$ seine Stelligkeit. Die Sorten $s_1, \ldots, s_k$ werden Argumentsorten und s Zielsorte genannt. Falls $k = 0$, wird die Operation $n :\rightarrow s$ Konstante genannt.

Für eine Signatur $\Sigma = (S, \Omega)$ definieren wir den Begriff Σ-Algebra.

2.2 Algebren

Eine Algebra ordnet einer Signatur eine Bedeutung zu, indem sie zu jeder Sorte eine Datenmenge und zu jeder Operation eine totale Funktion assoziiert.

Definition 2 (Algebra) Sei $\Sigma = (S, \Omega)$ eine Signatur. Eine Σ-Algebra A besteht aus:

- einer Menge $A(s)$ für jede Sorte $s \in S$, die Trägermenge von A der Sorte s genannt wird, und
- einer totalen Funktion

$$A(n : s_1 \times \ldots \times s_k \rightarrow s) : A(s_1) \times \ldots \times A(s_k) \rightarrow A(s)$$

für jede Operation $(n : s_1 \times \ldots \times s_k \rightarrow s) \in \Omega, k \geq 0$.

Notation: $Alg(\Sigma)$ ist die Klasse aller Σ-Algebren

2.3 Homomorphismen

Definition 3 (Homomorphismen) Seien $\Sigma = (S, \Omega)$ eine Signatur und A, B Σ-Algebren. Ein Σ-Homomorphismus von A nach B ist eine Familie $h = (h_s)s \in S$ von Funktionen:

$$h_s : A(s) \rightarrow B(s),$$

so daß für jede Operation $\omega \in \Omega$, $\omega = (n : s_1 \times \ldots \times s_k \rightarrow s), k \geq 0$ die folgende Bedingung gilt:

$$h_s(A(\omega)(a_1, \ldots, a_k)) = B(\omega)(h_{s_1}(a_1), \ldots, h_{s_k}(a_k)) \quad (*)$$

für alle $(a_1, \ldots, a_k) \in A(s_1) \times \ldots \times A(s_k)$.

(*) wird Homomorphiebedingung von h für ω genannt.

2.4 Initiale Algebra

Definition 4 (Initiale Algebra) Sei $\mathcal{C} \subseteq Alg(\Sigma)$ eine Klasse von Σ-Algebren für eine Signatur Σ. Eine Algebra $A \in \mathcal{C}$ ist initial in $\mathcal{C}$, falls es für jede Algebra $B \in \mathcal{C}$ genau einen Homomorphismus von A nach B gibt.

Ein Epimorphismus h ist ein surjektiver Homomorphismus, d.h. jedes

$$h_s : A(s) \to B(s)$$

ist surjektiv.

Ein Monomorphismus (bzw. Isomorphismus) ist ein injektiver (bzw. bijektiver) Homomorphismus.

Definition 5 Zwei Algebren $A,\ B$ heißen isomorph, wenn es einen Isomorphismus von A nach B gibt.

Notation: $A, B \in Alg(\Sigma), A \simeq B \Leftrightarrow A, B$ isomorph

2.5 Variablen und Terme

Definition 6 (Variablen) Eine Variablenfamilie für eine Signatur $\Sigma = (S, \Omega)$ ist eine Familie $V = (V_s)_{s \in S}$ von Mengen von Variablensymbolen, die jeweils paarweise und zu den Operationsnamen disjunkt sind. Ein Element von $V_s,\ s \in S$ wird Variable der Sorte s genannt.

Definition 7 (Syntax von Termen) Seien $\Sigma = (S, \Omega)$ eine Signatur und X eine Variablenmenge für Σ. Wir definieren die Familie $T_\Sigma(X) = (T_{\Sigma(X),s})_{s \in S}$ durch simultane Induktion:

- $X_s \subseteq T_{\Sigma(X),s}$;
- falls $n :\to s$ eine Operation in Ω ist, dann ist $n \in T_{\Sigma(X),s}$;
- falls $n : s_1 \times \ldots \times \to s, k \geq 1$ eine Operation in Ω und $t_i \in T_{\Sigma(X),s_i}, 1 \leq i \leq k$, dann ist $n(t_1 \ldots, t_k) \in T_{\Sigma(X),s}$.

Falls $t \in T_{\Sigma(X),s}$, dann nennt man s die Sorte von t. Falls $t \in T_\Sigma(X)$, dann ist $Var(t)$ die Menge der Variablen, die in t enthalten sind. Ein Term ohne Variable wird Grundterm genannt. Wir schreiben $T_\Sigma = (T_{\Sigma,s})_{s \in S}$, wobei $T_{\Sigma,s}$ die Menge aller Grundterme der Sorte s ist.

Definition 8 (Belegung) Seien $\Sigma = (S, \Omega)$ eine Signatur, X eine Variablenmenge und A eine Σ-Algebra. Eine Belegung für X ist eine Familie $\alpha = (\alpha_s)_{s \in S}$ von Funktionen:

$$\alpha_s : X_s \to A(s).$$

Definition 9 (Semantik von Termen) Seien $\Sigma = (S, \Omega)$ und $\alpha : X \to A$ eine Belegung. Der Wert $A(\alpha)(t)$ eines Termes $t \in T_\Sigma(X)$ für α und A wird durch Induktion über die Struktur von t definiert:

- Falls $t = x \in X$, dann $A(\alpha)(t) = \alpha_s(x)$;
- falls $t = n$ und $\omega = (n :\to s) \in \Omega$, dann $A(\alpha)(t) = A(\omega)$;
- falls $t = n(t_1, \ldots, t_k)$ und $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega,\ k \geq 1,\ t_i \in T_{\Sigma(X), s_i}$, $1 \leq i \leq k$, dann $A(\alpha)(t) = A(\omega)(A(\alpha)(t_1), \ldots, A(\alpha)(t_k))$.

Um formal mit Teiltermen eines Termes umgehen zu können, geben wir eine andere Definition von Termen. Terme können nämlich als Bäume, die über den Symbolen einer Signatur und einer Variablenmenge konstruiert sind, definiert werden.

Definition 10 (Baum) Seien N^+ die Menge der endlichen Folgen von natürlichen Zahlen, ϵ die leere Folge und $\cdot$ die Konkatenation auf Folgen. Seien $\Sigma = (S, \Omega)$ eine Signatur, X eine Variablenmenge, t eine Funktion von N^+ nach $\Omega \cup X$ und $Dom(t)$ ihr Definitionsbereich. t ist genau dann ein Baum, wenn:

- $\epsilon \in Dom(t)$;
- $m \cdot i \in Dom(t) \Leftrightarrow m \in Dom(t), 1 \leq i \leq s$, wobei s die Stelligkeit von $t(m)$ ist.

Der Definitionsbereich $Dom(t)$ eines Baumes wird Menge aller Stellen in t genannt. Die Elemente von $Dom(t)$ sind die Knoten des Baumes:

- $t|_\epsilon = t$;
- $f(t_1, \ldots, t_n)|_{i \cdot m} = t_i|_m$, falls $1 \leq i \leq n$ und $m \in Dom(t)$.

Definition 11 (Tiefe eines Baumes) Die Tiefe eines Baumes t wird induktiv definiert:

- Falls t eine Konstante oder eine Variable ist, dann ist $Tiefe(t) = 0$;

- falls $t = f(t_1, \ldots, t_n)$, dann ist $Tiefe(t) = 1 + max(Tiefe(t_1), \ldots, Tiefe(t_n))$.

Definition 12 (Gleichung) Sei $\Sigma = (S, \Omega)$ eine Signatur und X eine Variablenmenge. Eine (Σ, X)-Gleichung hat die Form:

$$\forall X t_1 = t_2 \text{ mit } t_1, t_2 \in T_{\Sigma(Y),s} \text{ und } Y \subseteq X.$$

Definition 13 (Bedingte Gleichung) Sei $\Sigma = (S, \Omega)$ eine Signatur und X eine Variablenmenge. Eine (Σ, X)-bedingte Gleichung hat die Form:

$$\forall X t_1 = u_1 \wedge \ldots \wedge t_k = u_k \Rightarrow t_{k+1} = u_{k+1},\ k \geq 0,\ t_i, u_i \in T_{\Sigma(X),s_i}, 1 \leq i \leq k+1$$

2.6 Erzeugte Algebren

Definition 14 (Erzeugte Algebren) Seien $\Sigma = (S, \Omega)$ eine Signatur, A eine Σ-Algebra und $\Omega_c \subseteq \Omega$ eine Menge von Operationen, Konstruktoren gennant. Eine Algebra A ist:

- erzeugt durch die Konstruktoren in Ω_c, falls es für jede Sorte $s \in S$ und Träger $a \in A(s)$ einen Grundterm $t \in T_{\Sigma_c,s}$ gibt, mit $a = A(t)$;
- erzeugt, falls sie durch Ω erzeugt ist.

Per Definition ist jeder Träger in einer erzeugten Algebra durch einen Grundterm darstellbar (*no Junk*). Um Eigenschaften der Träger einer erzeugten Algebra zu beweisen, kann man demnach Induktion über die Grundterme verwenden. Diese zusätzliche Eigenschaft erhöht die Attraktivität der erzeugten Algebren.

Definition 15 (Frei erzeugte Algebren) Seien $\Sigma = (S, \Omega)$ eine Signatur, A eine Σ-Algebra und $\Omega_c \subseteq \Omega$ eine Konstruktormenge. Eine Algebra ist dann:

- frei erzeugt durch Ω_c, falls es für jede Sorte $s \in S$ und für jeden Träger $a \in A(s)$ *genau* einen Grundterm $t \in T_{\Sigma_c,s}$ gibt, so daß $a = A(t)$;
- frei erzeugt, falls sie durch Ω frei erzeugt ist.

2.7 Formeln

Definition 16 (Syntax von Formeln) Für jede Signatur $\Sigma = (S, \Omega)$ wird die Menge der Formeln $PL(\Sigma)$ wie folgt definiert:

- Falls X eine Variablenmenge für Σ ist, s eine Sorte in S und $t, u \in T_{\Sigma(X),s}$, dann ist $t = u$ eine Formel;
- falls ϕ_1, ϕ_2 Formeln sind, dann ist $\phi_1 \wedge \phi_2$ eine Formel;
- falls ϕ eine Formel ist, dann ist $(\neg\phi)$ eine Formel;
- falls $s \in S$, x eine Variable der Sorte s, ϕ eine Formel, dann ist $(\forall x : s.\phi)$ eine Formel.

Die folgenden Abkürzungen sind klassisch: ϕ_1, ϕ_2, ϕ_3 sind beliebige Formeln und ϕ ist eine feste aber nicht näher spezifizierte Formel:

- $\phi_1 \vee \phi_2$ für $(\neg((\neg\phi_1) \wedge (\neg\phi_2)))$;
- $\phi_1 \Rightarrow \phi_2$ für $((\neg\phi_1) \vee \phi_2)$;
- $\phi_1 \Leftrightarrow \phi_2$ für $((\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1))$;
- $(\exists x : s.\phi)$ für $(\neg(\forall x : s.(\neg\phi)))$;
- $(\forall x_1 : s_1, \ldots, x_k : s_k.\phi)$ für $(\forall x_1 : s_1.(\ldots(\forall x_k : s_k.\phi))\ldots)$;
- $(\exists x_1 : s_1, \ldots, x_k : s_k.\phi)$ für $(\exists x_1 : s_1.(\ldots(\exists x_k : s_k.\phi))\ldots)$;
- T für $(\phi \Leftrightarrow \phi)$;
- F für $(\neg T)$.

Definition 17 (Universum) Ein Universum $\mathcal{U}$ für eine Signatur Σ ist eine unter Isomorphismus abgeschlossene Klasse von Σ-Algebren.

Definition 18 (Semantik von Formeln) Seien Σ eine Signatur und A eine Σ-Algebra. Der Wert einer Formel $\phi \in PL(\Sigma)$ für eine Belegung $\alpha : X \to A$, wobei X die freien Variablen von ϕ enthält, ist durch Induktion über die Struktur von ϕ definiert:

- $A(\alpha)(t = u) = true$ genau dann, wenn $A(\alpha)(t) = A(\alpha)(u)$;
- $A(\alpha)(\phi_1 \wedge \phi_2) = true$ genau dann, wenn $(A(\alpha)(\phi_1) = true$ und $A(\alpha)(\phi_2) = true)$;

- $A(\alpha)(\neg\phi) = true$ genau dann, wenn $A(\alpha)(\phi) = false$;
- $A(\alpha)(\forall x : s.\phi) = true$ genau dann, wenn $(A(\alpha[a/x])(\phi) = true$ für alle $a \in A(s))$,

wobei für eine Funktion $f : A \to B$ die Funktion $f[b/a] : A \cup \{a\} \to B \cup \{b\}$ wie folgt definiert ist:

$$f[b/a](c) = \begin{cases} b & \text{falls } c = a \\ f(c) & \text{sonst} \end{cases}$$

für alle $c \in A \cup \{a\}$.

Definition 19 (Gültigkeitsrelation) Sei Σ eine Signatur. Die Gültigkeitsrelation der Prädikatenlogik erster Stufe wird wie folgt definiert:

$$A \models_\Sigma \phi \text{ genau dann, wenn } (A(\alpha)(\phi) = true) \text{ für alle Belegungen } \alpha : frei(\phi) \to A$$

für jede Σ-Algebra A und jede Formel $\phi \in PL(\Sigma)$, wobei $frei(\phi)$ für alle freien Variablen in ϕ steht.

2.8 Modelle und logische Folgerungen

Definition 20 (Modell) Seien L eine Logik, Σ eine Signatur und $\Phi \subseteq L(\Sigma)$ eine Formelmenge.

- Eine Σ-Algebra A ist ein Modell von Φ, falls $A \models \phi$ für alle $\phi \in \Phi$.
- Sei $\mathcal{U}$ ein Σ-Universum. Die Klasse aller Algebren von $\mathcal{U}$, die Modelle von Φ sind, wird mit $Mod_{\mathcal{U},\Sigma}(\Phi)$ bezeichnet.

Definition 21 (Substitution) Seien $\Sigma = (S, \Omega)$ eine Signatur und X, Y zwei Σ-Variablenmengen. Eine Familie $\sigma = (\sigma_s)_{s \in S}$ von Funktionen

$$\sigma_s : X_s :\to T_{\Sigma(Y),s}$$

wird Substitution $\sigma : X \to T_\Sigma(Y)$ genannt.

Falls Y leer ist, dann ist σ eine Grundsubstitution. $Dom(\sigma)$ (bzw. $Img(\sigma)$) ist der Definitionsbereich (bzw. Bildbereich) von σ.
Die Anwendung der Substitution $\sigma : X \to T_\Sigma(Y)$ auf $t \in T_\Sigma(X)$ ergibt einen Term $\sigma(t) \in T_\Sigma(Y)$, der induktiv definiert ist:

- Falls $t = x, x \in X_s$, dann $\sigma(t) = \sigma_s(x)$;
- falls $t = n$, wobei $n :\rightarrow s \in \Omega$, dann $\sigma(t) = t$;
- falls $t = n(t_1, \ldots, t_k)$, wobei $n : s_1 \times s_k \rightarrow s \in \Omega,\ k \geq 1$ und $t_i \in T_{\Sigma(x),s_i}, 1 \leq i \leq k$, dann

$$\sigma(t) = n(\sigma(t_1), \ldots, \sigma(t_k)).$$

Definition 22 (Logische Folgerung) Seien Σ eine Signatur, ϕ eine Formel in $PL(\Sigma)$ und $\Phi \subseteq PL(\Sigma)$. Die Formel ϕ ist eine logische Folgerung von Φ, falls $A \models \phi$ für jede Algebra $A \in Mod_\Sigma(\Phi)$. Man schreibt: $\Phi \models \phi$.

2.9 Quotientenalgebra

Definition 23 (Kongruenzrelation) Sei $\Sigma = (S, \Omega)$ eine Signatur und $A \in Alg(\Sigma)$. Eine Kongruenzrelation auf A ist eine Familie $Q = (Q_s)_{s \in S}$ von Äquivalenzrelationen Q_s auf $A(s)$, $s \in S$, so daß die folgende Bedingung gilt:

Für alle $w = (n : s_1 \times \ldots \times s_k \rightarrow s) \in \Omega,\ k \geq 0$ und alle $a_i, a_i' \in A(s_i),\ 1 \leq i \leq n$ gilt:
falls $a_i\ Q_{s_i}\ a_i',\ 1 \leq i \leq k$;
dann $A(w)(a_1, \ldots, a_k) Q_s A(w)(a_1', \ldots, a_k')$.

Die obige Eigenschaft nennt man Substitutionseigenschaft.

Definition 24 (Quotientenalgebra) Seien $\Sigma = (S, \Omega)$ eine Signatur, A eine Σ-Algebra und $Q = (Q_s)_{s \in S}$ eine Kongruenzrelation auf A. Die Quotientenalgebra von A bzgl. Q ist die Σ-Algebra A/Q definiert durch:

- $A/Q(s) = \{[a]_{Q_s} | a \in A(S)\}$, für alle $s \in S$;
- $A/Q(w)([a_1]_{Q_{s_1}}, \ldots, [a_k]_{Q_{s_k}}) = [A(w)(a_1, \ldots, a_k)]_{Q_s}$, für alle $w = (n : s_1 \times \ldots s_k \rightarrow s) \in \Omega,\ k \geq 1$.

Man sagt: A/Q erhält man durch Faktorisierung von A durch Q, welche eine rein semantische Operation ist. Die Signatur wird dadurch nicht verändert.

Wir führen nun Algebren ein, deren Trägermengen aus der Menge aller Grundterme der entsprechenden Sorte bestehen.

Definition 25 (Termalgebra) Die Termalgebra für die Signatur $\Sigma = (S, \Omega)$ und die Variablenmenge X ist die Algebra $T(\Sigma)$, die wie folgt definiert ist:

- $T(\Sigma)(s) = T_{\Sigma,s}$, für jedes $s \in S$;
- $T(\Sigma)(w) = n$, für jedes $w = (n :\rightarrow s) \in \Omega$;
- $T(\Sigma)(w)(t_1, \ldots, t_k) = n(t_1, \ldots, t_k)$, für jedes $w = (n : s_1 \times \ldots \times s_k \rightarrow s) \in \Omega$, $k \geq 1,\ t_i \in T(\Sigma)(s_i),\ 1 \leq i \leq k$.

Wir definieren nun die Kongruenzrelation einer Klasse von Algebren.

Definition 26 (Kongruenzrelation einer Algebraklasse) Seien $\Sigma = (S, \Omega)$ eine Signatur und $\mathcal{C} \subseteq Alg(\Sigma)$ eine Klasse von Σ-Algebren. Die Kongruenzrelation der Klasse $\mathcal{C}$ ist eine Familie $\equiv_{\mathcal{C}} = (\equiv_{\mathcal{C},s})_{s \in S}$ von Relationen auf T_Σ, definiert durch:

$$\equiv_{\mathcal{C},s} = \{(t, u) \in T_{\Sigma,s} \times T_{\Sigma,s} | A(t) = A(u) \text{ für jede } A \in \mathcal{C}\}$$

Definition 27 (Quotiententermalgebra) Sei $\mathcal{C} \subseteq Alg(\Sigma)$ eine nicht-leere Klasse von Σ-Algebren. Die Quotiententermalgebra der Klasse $\mathcal{C}$ ist die Σ-Algebra $T(\Sigma, \mathcal{C})$ definiert durch:

$$T(\Sigma, \mathcal{C}) = T(\Sigma)/_{\equiv_{\mathcal{C}}}.$$

2.10 Matching und Unifi kation

Bevor wir uns der Vorstellung von Termersetzungssystemen widmen, führen wir in diesem Abschnitt zwei Begriffe ein, die bei der Anwendung von Regeln und bei der Umwandlung eines Regelsystems in ein äquivalentes konvergentes System eine fundamentale Rolle spielen.

2.10.1 Matching

Ein Term t ist mit einer Regel $l \rightarrow r$ reduzierbar, falls es eine Stelle u in t und eine Substitution (einen Match) σ gibt, so daß $t|_u = \sigma(l)$. Die Suche nach einer Stelle u und nach einem Match σ gehört also unmittelbar zu der Ausführung eines Reduktionsschritts.

Definition 28 (Matching) Die Substitution σ heißt Match von s auf t, falls $\sigma(s) = t$ gilt. Das Problem, einen Match zu finden, heißt Matching-Problem.

2.10.2 Unifikation

Zwei Terme s und t sind unifizierbar, falls es eine Substitution σ gibt, die s und t gleichmacht.

Definition 29 (Unifikation) Seien s und t zwei Terme. Falls es eine Substitution σ gibt, so daß $\sigma(s) = \sigma(t)$, dann heißen $s,\ t$ unifizierbar und σ Unifikator von s und t.

Kapitel 3

Termersetzungssysteme

Termersetzungssysteme sind ein wichtiges Hilfsmittel zur automatisierten Behandlung von Gleichheitsaxiomen, da sie das Rechnen in (bedingten) gleichungsdefinierten Algebren ermöglichen. Sie finden deshalb im Bereich der algebraischen Spezifikationen, der funktionalen Programmierung und des automatischen Theorembeweisens Verwendung. Termersetzungssysteme dienen weiter zur Beschreibung von ADT's, genauer zur Angabe von ausführbaren Spezifikationen.

Termersetzungssysteme basieren auf der Idee, Terme durch Anwendung von gerichteten Gleichungen in Normalformen zu überführen. Die Semantik eines Termersetzungssystems ist dann wohldefiniert, wenn alle Terme zu Normalformen reduzierbar sind (Terminierung) und das Ergebnis der Berechnung nicht von der Wahl der Regeln abhängt, die zur Reduktion benutzt werden (Konfluenz).

Für Termersetzungssysteme, die konfluent und terminierend sind, ist die Termreduktion eine konsistente und vollständige Inferenzregel, da alle logischen Folgerungen durch die Berechnung von Normalformen herleitbar sind [HUHO80].

Wegen des engen Zusammenhangs von Ersetzungsregeln und Gleichungen werden Termersetzungssysteme für eine breite Klasse von Problemen eingesetzt. Hierzu gehören u.a. die Unifikation in Gleichungstheorien [FAY79], [SIE89], das Wortproblem in der universellen Algebra [KB70], die Ausführung algebraischer Spezifikationen [MUS80B], induktive Beweise für Datentypen [MUS80A], [HUH82], die Programmsynthese [DER85], sowie Theorembeweise in verschiedenen Logikkalkülen.

Das ursprüngliche Ziel bei der Betrachtung von Termersetzungssystemen war die Beschränkung der Anwendbarkeit von Gleichungen, bei denen die Vollständigkeit erhalten bleiben sollte. Um die Expressivität der Termersetzungssysteme zu erhöhen, wurden die bedingten Termersetzungssysteme eingeführt [KAP84]. Die semantische Fundierung bedingter Termersetzungssysteme beruht auf einem verallgemeinerten Reduktionsbegriff. Die Termersetzung ist an zusätzliche Bedingun-

gen gebunden, die als Prämissen von Regeln formuliert werden und die solche Terme bestimmen, die reduzierbar sind. Eine bedingte Regel kann nur dann angewandt werden, wenn ihre Prämisse geeignet auswertet werden kann. Das Hauptproblem bei der Anwendung von bedingten Termersetzungssystemen ist die rekursive Auswertung der Prämissen. Konfluenz und Terminierung eines bedingten Termersetzungssystems garantieren nicht die Terminierung der rekursiven Auswertung der Prämissen [KAP84]. Wir werden in dieser Arbeit nur bedingte Termersetzungssysteme betrachten, bei denen die rekursive Auswertung der Prämissen terminiert.

Ein bedingtes Termersetzungssystem ist eine Menge von bedingten Gleichungen, bei der jede bedingte Regel als eine universell quantifizierte Hornklausel mit der Gleichheit als einziges Prädikatssymbol gesehen werden kann. Bevor wir bedingte Termersetzungssysteme formal definieren, führen wir zunächst die (klassischen) unbedingten Termersetzungssysteme und einige ihrer grundlegenden Eigenschaften ein, die wir später verwenden werden.

3.1 Unbedingte Termersetzungssysteme

Definition 30 (Partielle Ordnung) Eine binäre Relation $\geq$ auf einer Menge M heißt partielle Ordnung, falls sie reflexiv, anti-symmetrisch und transitiv ist. Für den strikten Anteil $>$ von $\geq$ gilt: $x > y$ genau dann, wenn $x \geq y$ und $x \neq y$.

Definition 31 (Termersetzungsregel, Termersetzungssystem, Termersetzungsrelation) Sei $\Sigma = (S, \Omega)$ eine Signatur und X eine Σ-Variablenmenge.

- Eine Regel ist ein Paar (l, r) mit $l, r \in T_{\Sigma(X),s}$ $s \in S$, $l \notin X$und $Var(r) \subseteq Var(l)$; wir schreiben $l \to r$;
- Ein Termersetzungssystem R ist eine Menge von Regeln;
- Ein Termersetzungssystem definiert eine Relation $\to_R$ auf $T_\Sigma(X)$:

  $t \to_R s$ genau dann, wenn es eine Regel $l \to r \in R$, eine Substitution σ und eine Stelle u in t gibt, so daß :

$$t|_u \equiv \sigma(l) \text{ und } s \equiv t[u \leftarrow \sigma(r)].$$

Ein Term t heißt *linear*, wenn jede Variable in $Var(t)$ nur einmal in t vorkommt. Man schreibt $\sharp(x, t)$ für die Anzahl der Vorkommen von x in t. Eine Regel $l \to r$ ist linear, falls l linear ist. Ein Termersetzungssystem R ist dann linear, wenn alle Regeln in R linear sind. Eine Stelle u in t heißt *strikt*, falls $t|_u$ keine Variable ist. Eine Sorte s heißt *finitär*, falls die Menge der Grundterme der Sorte s endlich ist sonst *infinitär*. Ein Term t heißt finitär (bzw. infinitär), falls seine Sorte finitär

(bzw. infinitär) ist. Die *Tiefe* eines Termersetzungssystems R ist das Maximum der Tiefen aller linken Seiten von R. Die strikte Tiefe von R ist analog definiert.

Die Relation $\leftrightarrow_R$, (bzw. $\rightarrow_R^+$, $\rightarrow_R^*$, $\leftrightarrow_R^*$) ist der symmetrische (bzw. transitive, transitive-reflexive, symmetrisch-transitiv-reflexive) Abschluß von $\rightarrow_R$.

Ein Term t heißt *reduzierbar* oder *reduzibel*, falls es einen Term s gibt, so daß $t \rightarrow_R s$, sonst ist t irreduzibel. s heißt eine Normalform zu t, falls $t \rightarrow_R^* s$ und s irreduzibel ist. $t \downarrow_R$ steht dann für die Normalform von t.

Definition 32 (Terminierende Relation) Eine partielle Ordnung $>$ auf M heißt terminierend oder noethersch, wenn es keine unendliche absteigende Kette $t_0 > t_1 > ...$ gibt.

Definition 33 (Stabile Relation) Eine Relation P auf $T_\Sigma(X)$ heißt *stabil*, wenn für alle Substitutionen σ gilt:

$$\text{Falls } s \ P \ t, \text{ dann } \sigma(s) \ P \ \sigma(t).$$

Sie heißt *monoton* , wenn für alle Terme t und Stellen u in t gilt:

$$\text{Falls } s_1 \ P \ s_2 \text{ dann } t[u \leftarrow s_1] \ P \ t[u \leftarrow s_2].$$

Definition 34 Ein Termersetzungssystem ist terminierend, wenn $\rightarrow_R$ terminierend ist.

Definition 35 (Reduktionsordnung) Eine noethersche partielle Ordnung heißt *Reduktionsordnung*, falls sie stabil und monoton ist.

Die Umwandlung einer Menge von Gleichungen in ein Termersetzungssystem gefährdet die Vollständigkeit des darauf aufbauenden Verfahrens (z.B. Deduktionsverfahren), falls gewisse Bedingungen nicht erfüllt sind. Die Vollständigkeit läßt sich sicherstellen, falls das nach der Umwandlung erhaltene Termersetzungssystem terminierend und konfluent ist. Ein Termersetzungssystem ist konfluent, wenn alle divergierenden Reduktionen wieder zusammengeführt werden können. Mit einem konfluenten Termersetzungssystem kann von jedem Term s aus höchstens ein irreduzibler Term erreicht werden.

Definition 36 (Konfluenz) Eine Relation $\rightarrow$ heißt konfluent, wenn für alle Terme s, t_1, t_2 mit $s \rightarrow^* t_1$ und $s \rightarrow^* t_2$ ein Term t existiert, so daß $t_1 \rightarrow^* t$ und $t_2 \rightarrow^* t$.

Definition 37 (Lokale Konfluenz) Eine Relation $\rightarrow$ heißt lokal konfluent, falls für alle Terme s, t_1, t_2 mit $s \rightarrow t_1$ und $s \rightarrow t_2$ ein Term t existiert, so daß $t_1 \rightarrow^* t$ und $t_2 \rightarrow^* t$.

Der Unterschied zwischen (globaler) und (lokaler) Konfluenz liegt darin, daß man bei der lokalen Konfluenz Terme betrachtet, die aus s durch genau einen Schritt erreichbar sind. Jede konfluente Relation ist selbstverständlich auch lokal konfluent. Für terminierende Relationen gilt auch die Umkehrung, so daß dort nur die lokale Konfluenz sichergestellt werden muß, um eine konvergente (terminierende und konfluente) Relation zu erhalten. Ein Termersetzungssystem heißt grundkonfluent bzw. grundterminierend, wenn die Einschränkung von $\rightarrow$ auf $T_\Sigma \times T_\Sigma$ konfluent bzw. terminierend ist.

Definition 38 (Kritische Paare) Seien $g_1 \rightarrow d_1,\ g_2 \rightarrow d_2$ zwei (nicht notwendigerweise verschiedene) Regeln aus einem Termersetzungssystem R. Falls es eine Substitution σ und eine strikte Stelle u von g_1 (also $g_1|_u$ ist keine Variable) gibt, so daß $g_1|_u$ und g_2 durch σ unifizierbar sind. Dann wird das Paar $(\sigma(d_1), \sigma(g_1[u \leftarrow \sigma(g_2)]))$, kritisches Paar zu den beiden Regeln genannt.

Ein kritisches Paar (p, q) heißt konfluent, falls es einen Term t gibt, so daß $p \rightarrow t$ und $q \rightarrow t$.

Satz 1 *Ein Termersetzungssystem ist genau dann lokal konfluent, wenn alle seine kritischen Paare konfluent sind.*

Die Konfluenz garantiert die Eindeutigkeit von Normalformen, während die Terminierung deren Existenz gewährleistet.

3.2 Bedingte Termersetzungssysteme

Eine Menge bedingter Gleichungen auf Terme induziert eine Kongruenz. Zwei Terme sind dann äquivalent, falls man durch eine Folge von Ersetzungen von einem Term zu einem anderen übergehen kann. Dabei ist eine Ersetzung nur erlaubt, wenn die Gültigkeit der assoziierten Prämissen rekursiv gezeigt werden kann. Wir werden in diesem Abschnitt die Eigenschaften der Konfluenz und der Terminierung auf bedingte Termersetzungssysteme verallgemeinern.

Das Konzept der bedingten Termersetzung wurde von Kaplan [KAP84] [KAP85] eingeführt und später durch Jouannaud und Waldmann [JW86] erweitert. Die in den Regeln vorkommenden Bedingungen werden hier nicht auf Formeln beschränkt, die nur Funktionen enthalten, die auf tieferen Ebenen bzgl. einer vorgegebenen Hierarchie definiert wurden [DRO84], [PE82], [ZR85]. Wir verlangen hier, daß die linke und rechte Seite jeder Gleichung in der Prämisse einer Ersetzungsregel (bzgl. einer *abfallenden* Ordnung: siehe Definition 41) kleiner als die linke Seite der Konklusion sind. Diese Einschränkung ist hinreichend, um sowohl die Terminierung und als auch die Entscheidbarkeit der Reduktionsrelation zu garantieren.

Definition 39 (Bedingte Regel) Sei $\Sigma = (S, \Omega)$ eine Signatur und X eine Variablenmenge. Eine bedingte Regel ist eine bedingte Gleichung, deren Konklusion gerichtet ist:

$$t_1 = s_1 \wedge \ldots \wedge t_n = s_n \Rightarrow t_0 \rightarrow s_0.$$

Weiter gilt: $Var(s_0) \subseteq Var(t_0)$ und für $1 \leq i \leq n, Var(t_i), Var(s_i) \subseteq Var(t_0)$.

Die Regeln enthalten keine Extra-Variablen; d.h. jede Variable, die in einer Regel vorkommt, kommt auch in der linken Seite der Konklusion der Regel vor.

Um dem Begriff der bedingten Termersetzung eine Semantik zuzuordnen, soll dem in den Prämissen vorkommenden Gleichheitssymbol eine feste Interpretation gegeben werden. Es bieten sich u.a. folgende Möglichkeiten an:

1. $=$ wird als $\leftrightarrow$ (der symmetrische und transitive Abschluß von $\rightarrow$) interpretiert;

2. $=$ wird als $\downarrow$ (wobei $s \downarrow t$ genau dann, wenn es ein u gibt, so daß $s \rightarrow^* u$ und $t \rightarrow^* u$) interpretiert;

3. $=$ wird als $\equiv$ (wobei $\equiv$ für die syntaktische Gleichheit steht) interpretiert.

In dieser Arbeit werden wir nur bedingte Termersetzungssysteme betrachten, bei denen das Gleichheitssymbol in den Prämissen als Relation verstanden wird, die genau die Terme in Beziehung setzt, die auf einen gemeinsamen Term reduzierbar sind (Möglichkeit 2 oben). Für ein bedingtes Termersetzungssystem R werden die Ersetzungsrelation ($\rightarrow$) und die Zusammenführungsrelation ($\downarrow$) wie folgt definiert:

Definition 40 (Bedingte Termersetzungsrelation) Sei $t_1 = s_1 \wedge \ldots \wedge t_n = s_n \Rightarrow t_0 \rightarrow s_0$ eine bedingte Regel in R (wir schreiben im folgenden in den Prämissen $=$ anstelle von $\downarrow$). Sei t ein Term, und u eine Stelle in t. Wir schreiben:

$$t[\sigma(t_0)] \rightarrow_R t[u \leftarrow \sigma(s_0)],$$

falls es eine Substitution σ gibt, so daß für alle $i \in [1, \ldots, n]$ ein d_i existiert mit:

$$\sigma(t_i) \rightarrow_R^* d_i \text{ und } \sigma(s_i) \rightarrow_R^* d_i.$$

$\rightarrow_R^*$ ist der reflexive und symmetrische Abschluß von $\rightarrow_R$.

Die Auswertung der Prämissen wird rekursiv auf Termersetzung zurückgeführt, was zur Folge haben kann, daß man in eine Endlosschleife gerät. Die Terminierung bedingter Termersetzungssysteme ist also schwieriger zu beweisen als die Terminierung unbedingter Termersetzungssysteme.

Neben dem Beweis, daß es keine unendliche Termersetzungssequenz gibt, muß man zusätzlich beweisen, daß die Auswertung der Prämissen keine unendlichen rekursiven Aufrufe enthält. Um die Terminierung bedingter Termersetzungssysteme zu garantieren, müssen noch einige Restriktionen gemacht werden. Es gibt in der literatur folgende Einschränkungen, die sowohl die Terminierung, als auch die Entscheidbarkeit der Reduktionsrelation eines bedingten Termersetzungssystems garantieren. Sie unterscheiden sich in der Art der verwendeten noetherschen Ordnung:

1. Die rechte und die linke Seite jeder Gleichung in der Prämisse jeder bedingten Regel und die rechte Seite der Konklusion müssen bzgl. einer gegebenen Simplifikationsordnung kleiner als die linke Seite der Konklusion sein (**Simplifizierende Systeme**: siehe [KAP85]).

2. Die rechte und die linke Seite jeder Gleichung in der Prämisse jeder bedingten Regel müssen bzgl. einer Reduktionsordnung kleiner als die linke Seite der Konklusion sein. Die Reduktionsordnung enthält dabei den transitiven Abschluß der Ersetzungsrelation des betrachteten Termersetzungssystems (**Reduktive Systeme**: siehe [JW86]).

3. Die rechte und die linke Seite jeder Gleichung in der Prämisse jeder bedingten Regel müssen bzgl. einer noetherschen Ordnung $<$ kleiner als die linke Seite der Konklusion sein, wobei $<$ den irreflexiv-transitiven Abschluß der Ersetzungsrelation entählt, und die Teiltermeigenschaft besitzt (**Abfallende** oder **Decreasing systems**: siehe [DOS87]).

Sowohl die simplifizierenden als auch die reduktiven Systeme sind Spezialfälle der abfallenden Systeme. Simplifizierende Systeme haben per Definition die Teiltermeigenschaft. Für reduktive Systeme gilt: weil die Teiltermeigenschaft wegen der Monotonie und der Wohlfundiertheit nicht verletzt werden kann, läßt sich jede monotone noethersche Ordnung mit der Teiltermeigenschaft zu einer noetherschen Ordnung erweitern.

Definition 41 (Abfallende Termersetzungssysteme) Ein bedingtes Termersetzungssystem R ist abfallend, falls es eine noethersche Erweiterung $<$ der Teiltermordnung $\triangleleft$ gibt, so daß $<$ den irreflexiv-transitiven Abschluß $\rightarrow_R^+$ der Ersetzungsrelation von R enthält und

$$\sigma(l) > \sigma(s_1), \ldots, \sigma(t_n)$$

für jede Substitution σ und jede Regel $s_1 = t_1 \wedge \ldots \wedge s_n = t_n \Rightarrow l \rightarrow r$ in R, wobei $\triangleleft$ wie folgt definiert ist: $s \triangleleft t$, falls s ein Teilterm von t ist.

Satz 2 *(Dershowitz, Okada, Sivakumar) Falls ein bedingtes Termersetzungssystem R abfallend ist, dann hat es folgende Eigenschaften:*

1. *R ist terminierend.*
2. *Folgende Relationen sind entscheidbar: $\rightarrow$, $\rightarrow_R^+$, $s \downarrow t$, $t \downarrow_R$ (d.h. ist t reduzibel?).*

Analog zur Terminierung ist auch der Beweis der Konfluenz bedingter Termersetzungssysteme komplexer als beim unbedingten Fall. Die bekannten, auf kritischen Paaren basierenden hinreichenden Kriterien greifen nicht mehr bei bedingten Termersetzungssystemen. Das Kritische-Paar-Lemma gilt für bedingte Termersetzungssysteme nicht mehr ohne neue Restriktionen. Es gibt nämlich terminierende bedingte Termersetzungssysteme mit konfluenten kritischen Paaren, die nicht lokal konfluent [DOS87] sind. Für abfallende Termersetzungssysteme gilt aber das Kritische-Paar-Lemma. Die bedingten Termersetzungssysteme, die wir in dieser Arbeit betrachten werden, sind trivialerweise konfluent, da sie abfallend sind und per Definition (siehe erweiterte konstruktive Spezifikationen) keine kritischen Paare zulassen.

Definition 42 (Kritische Paare) Seien $r_1 : \phi_1 \Rightarrow s_1 \rightarrow t_1$ und $r_2 : \phi_2 \Rightarrow s_2 \rightarrow t_2$ zwei bedingte Regeln aus R. Falls es eine Substitution σ gibt und eine Stelle u von s_1 existiert, so daß $\sigma(s_1|u) = \sigma(s_2)$, dann ist die folgende bedingte Gleichung:

$$\sigma(\phi_1) \wedge \sigma(\phi_2) : \sigma(s_1[u \leftarrow s_2]) = \sigma(t_1)$$

ein kritisches Paar der Regeln r_1 und r_2.

Ein kritisches Paar $c = d \Rightarrow s = t$ ist *lösbar* (oder *feasible*), falls es eine Substitution σ gibt, so daß $\sigma(c) \downarrow \sigma(d)$. Es heißt *trivial*, falls s und t identisch sind. Es ist *zusammenführbar* (oder *joinable*), falls für alle Substitutionen σ, für die $\sigma(c) \downarrow \sigma(d)$ gilt, auch $\sigma(s) \downarrow \sigma(t)$ gilt. Ein bedingtes Termersetzungssystem R heißt *nicht-überlappend*, falls es weder triviale noch lösbare kritische Paare besitzt.

Satz 3 *(Dershowitz, Okada, Sivakumar) Ein abfallendes Termersetzungssystem R ist konfluent, falls seine lösbaren kritischen Paare zusammenführbar sind.*

Wir werden in den kommenden Kapiteln lediglich nicht-überlappende Regelsysteme betrachten. Wie bei noetherschen unbedingten Termersetzungssystemen sind auch nicht-überlappende bedingte Termersetzungssysteme konfluent, falls sie terminierend sind.

Wir definieren nun einen weiteren, auf dem Termersetzungsgebiet sehr verbreiteten Begriff, der durch den Wunsch, das Wortproblem zu lösen, motiviert ist.

Definition 43 (Church-Rosser-Eigenschaft) Ein bedingtes Termersetzungssystem R besitzt die *Church-Rosser-Eigenschaft*, wenn für alle Terme $t_1, t_2 \in T_\Sigma(X)$ folgendes gilt:

$$t_1 \leftrightarrow_R^* t_2 \text{ genau dann, wenn } t_1 \downarrow_R t_2.$$

Bemerkung: Konfluenz und Church-Rosser-Eigenschaft sind äquivalente Eigenschaften.

Falls ein bedingtes Termersetzungssystem R konfluent und terminierend (konvergent) ist, dann läßt es sich als Entscheidungsverfahren für das Wortproblem in der durch R spezifizierten algebraischen Struktur einsetzen.

Kapitel 4

Initiale und konstruktive Spezifikationen

Bei der Entwicklung von Programmen ist es wesentlich, darauf zu achten, daß die am Ende erhaltenen Programme transparent, einfach zu modifizieren und fehlerfrei sind. Das Aufteilen von größeren Softwareprodukten in kleinere Programmteile ist für die Erfüllung der obengenannten Eigenschaften unabdingbar. Die Korrektheit von Programmen läßt sich zum Beispiel nur mit kleinen Modulen verifizieren. Dieses Modularisierungsprinzip setzt aber eine gewisse Abstraktion voraus. Auf höherer Ebene abstrahiert man von Programmierdetails, und auf höchster Ebene beschäftigt man sich lediglich mit der Problemstellung.

Eine Methode zur Realisierung der Abstraktion ist die Benutzung von Spezifikationen. Spezifikationen definieren einerseits abstrakte Datentypen (kurz ADT's) und ermöglichen es andererseits, Eigenschaften von solchen definierten ADT's zu beweisen. Es werden zwei Arten von Spezifikationen unterschieden: die Anforderungsspezifikationen und die Designspezifikationen. Erstere sind im allgemeinen als polymorphe Spezifikationen definiert (d.h. sie definieren polymorphe ADT's), letztere sind näher zum Programm, sind im allgemeinen monomorph und ermöglichen Rapid Prototyping. Wir führen nun die Methode der initialen Spezifikation mit der bedingten Gleichungslogik ein.

4.1 Initiale Spezifikationen

Definition 44 (Syntax der initialen Spezifikation) Eine initiale Spezifikation in der bedingten Gleichungslogik ist ein Paar $Sp = (\Sigma, \Phi)$, bestehend aus einer Signatur Σ und aus einer Menge $\Phi \subseteq CEL(\Sigma)$ von bedingten Gleichungen.

Definition 45 (Semantik der initialen Spezifikation) Die Semantik $\mathcal{M}(Sp)$ einer initialen Spezifikation $Sp = (\Sigma, \Phi)$ ist der monomorphe ADT $\mathcal{M}(Sp) = \{A \in Alg(\Sigma) | A \simeq T(\Sigma, \Phi)\}$, wobei $T(\Sigma, \Phi)$ die Quotiententermalgebra von Definition 27 ist.

Satz 4 *(Initialität) Sei Σ eine Signatur und $\Phi \subseteq CEL(\Sigma)$ eine Menge von bedingten Gleichungen. Es gilt: $T(\Sigma, \Phi)$ ist ein Modell von Φ.*

Beweis: siehe [LEW96]

Satz 5 *Sei $Sp = (\Sigma, \Phi)$ eine initiale Spezifikation. Jede Algebra in $\mathcal{M}(Sp)$ ist initial in $Mod_\Sigma(\Phi)$.*

Beweis: siehe [LEW96]

Eine Algebra in $\mathcal{M}(Sp)$ ist nicht irgendein Modell, sondern gerade das Modell, das der Vorstellung des Programmierers beim Entwerfen eines Datentypen entspricht (das intendierte Modell).

4.2 Erweiterte konstruktive Spezifikationen

Wir betrachten nun eine Variante konstruktiver Spezifikationen, die solche in [LEW96] erweitert und in [PAD96] als kanonische Spezifikationen bezeichnet werden. Im folgenden gehen davon aus, daß es für jede Sorte $s \in S$ mindestens einen Grundkonstruktorterm der Sorte s gibt.

Definition 46 (Erweiterte konstruktive Spezifikationen) Eine erweiterte konstruktive Spezifikation (kurz: EKS) ist ein Tripel $Sp = (\Sigma, \Phi, \Omega_c)$, wobei $\Sigma = (S, \Omega)$ eine Signatur, Φ eine Menge bedingter Gleichungen und $\Omega_c \subseteq \Omega$ eine Menge von Konstruktoren ist. Wir schreiben $\Sigma_c = (S, \Omega_c)$. Für Sp gilt:

1. Jede Gleichung in Φ hat die Form:

$$s_1 = t_1 \wedge \ldots \wedge s_n = t_n \Rightarrow f(r_1, \ldots, r_k) = t,$$

   mit $(f : s_1 \times \ldots \times s_k \rightarrow s) \in \Omega - \Omega_c, k \geq 0$,
   $r_i \in T_{\Sigma_c(X), s_i}, 1 \leq i \leq k$ und $t, t_j, s_j \in T_{\Sigma(X), s}, 1 \leq j \leq n$,
   wobei $Var(t), Var(s_i), Var(t_i) \subseteq Var(f(r_1, \ldots, r_k)), 1 \leq i \leq n$.

2. Φ ist nicht überlappend (d.h. es gibt weder lösbare noch triviale kritische Paare zwischen den Gleichungen von Φ).

3. Es gibt eine noethersche Ordnung $<$, unter der das bedingte Termersetzungssystem, das man erhält, indem man die Konklusion jeder Gleichung in Φ von links nach rechts richtet, abfallend ist.

Die obige Definition schließt die Verwendung von Extra-Variablen in den Bedingungen aus. Dies bedeutet, daß alle in den Bedingungen enthaltenen Variablen auch in der rechten Seite der Konklusion vorkommen. Damit sind bedingte Regeln, wie etwa die Transitivität, nicht formulierbar. Die beiden ersten Bedingungen der Definition sind rein syntaktisch, was für die dritte nicht gilt, da sie die Existenz einer noetherschen Ordnung verlangt (siehe Definition 41). Sie garantiert, daß man bei der Auswertung einer definierten Funktion nicht in eine Endlosschleife gerät. Durch die zweite Bedingung wird gewährleistet, daß es für jeden Term $t = f(t_1, \ldots, t_k) \in T_\Sigma(X)$ und $t_i \in T_{\Sigma_c(X)}, 1 \leq i \leq k$ höchstens eine Gleichung in Φ gibt, die auf t anwendbar ist. Die Bedeutung der zweiten Bedingung wird bei der Betrachtung des zu Φ assoziierten Termersetzungssystems deutlicher.

Beispiel 1 Sei $Sp = (\Sigma, \Phi, \Omega_c)$ mit den Konstruktoren $true,\ false,\ 0,\ s,\ nil$ und $cons$, $\Sigma = (\{bool, nat, list\}, \{true :\to bool,\ false :\to bool,\ 0 :\to nat,\ s : nat \to nat,\ nil :\to list,\ cons : nat \times list \to list,\ eq : nat \times nat \to bool,\ del : nat \times list \to list\})$ und

$$\begin{aligned}\Phi = \{\ & eq(0,0) = true;\\ & eq(0, s(x)) = false;\\ & eq(s(x), 0) = false;\\ & del(x, nil) = nil;\\ & eq(x,y) = true \Rightarrow del(x, cons(y,m)) = m;\\ & eq(x,y) = false \Rightarrow del(x, cons(y,m)) = cons(y, del(x,m))\}\end{aligned}$$

Sp ist konfluent, da die linken Seiten der Regeln in Sp nicht überlappend sind (weder lösbare noch triviale kritische Paare). Betrachtet man nun die rekursive Pfadordnung $<_{rpo}$ mit folgender Ordnung auf den Funktionssymbolen: $true < false < 0 < s < nil < cons < eq < del$. So sind für alle Regeln in Φ die Bedingungen 1-3 der Definition 46 erfüllt. Demnach ist Sp eine EKS.

Bemerkung: Unter rekursiver Pfadordnung (bzw. lexikographischer Pfadordnung) verstehen wir im folgenden die übliche Definition mit der folgenden zusätzlichen Bedingung:

$$t\ >_{rpo}\ x \text{ (bzw. } t\ >_{lpo}\ x\text{), falls } x \in Var(t),\ x \neq t.$$

Die durch die zusätzliche Bedingung erhaltene Ordnung ist dann eine Reduktionsordnung (siehe [AVE95]).

Definition 47 Das zu einer EKS B assoziierte Termersetzungssystem R ist das Termersetzungssystem, das man erhält, indem man die Konklusionen der bedingten Gleichungen in B von links nach rechts richtet.

Das zu einer EKS assoziierte Termersetzungssystem ist per Definition abfallend und demnach terminierend. Es ist auch konfluent, da es wegen der zweiten Bedingung keine kritischen Paare zwischen den Regeln gibt.

Die in dieser Arbeit eingeführten EKS unterscheiden sich von den konstruktiven Spezifikationen (kurz: KS), die in [LEW96] vorgestellt wurden in folgenden Punkten:

- EKS enthalten bedingte Gleichungen, während KS per Definition nur Gleichungen erlauben;
- EKS müssen nicht nur lineare Gleichungen enthalten, während die Definition von KS die Linearität der Gleichungen fordert;
- EKS müssen nicht hinreichend vollständig sein, während KS hinreichend vollständig sind.

Es ist also offensichtlich, daß EKS allgemeiner und expressiver als die klassischen konstruktiven Spezifikationen sind. Mit EKS lassen sich ADT's in einer sehr einfachen und natürlichen Weise spezifizieren.

Da wir aber in dieser Arbeit nur totale Funktionen betrachten wollen, fügen wir eine zusätzliche Bedingung zu der Definition der EKS hinzu.

Definition 48 (Hinreichende Vollständigkeit) Sei Sp eine EKS=(Σ, Φ, Ω_c). Eine definierte Funktion $f \in \Omega - \Omega_c$ heißt hinreichend vollständig bzgl. den Konstruktoren in Ω_c, falls es für jeden Grundterm $f(t_1, \ldots, t_n)$ und $t_i \in T_\Sigma, 1 \leq i \leq n$ einen Term $t \in T_{\Sigma_c}$ gibt, so daß :

$$f(t_1, \ldots, t_n) \rightarrow_R^* t,$$

wobei R das Termersetzungssystem ist, das zu Φ assoziiert ist.

Bemerkung: Die obige Definition der Vollständigkeit ist operationell und unterscheidet sich im allgemeinen von dem üblichen Vollständigkeitsbegriff. Die zwei Begriffe sind erst dann äquivalent, wenn die Konfluenz und die Terminierung des betrachteten Termersetzungssystems garantiert sind.

4.2.1 Kanonische erweiterte konstruktive Spezifi kationen

Definition 49 (Kanonische erweiterte konstruktive Spezifikationen, kurz: KEKS) Eine EKS heißt kanonisch, falls alle definierten Funktionen in Φ hinreichend vollständig sind.

Beispiel 2 Die EKS von Beispiel 1, erweitert mit der folgenden bedingten Gleichung:

$$eq(s(x), s(y)) = eq(x, y),$$

ist eine KEKS, da die Funktionen eq und del durch die neue Gleichung hinreichend vollständig geworden sind.

Neben der Terminierung und der Eindeutigkeit ist auch die Vollständigkeit (Erfassung aller möglichen Eingabewerte) der definierten Funktionen in einem KEKS nun gewährleistet. Diese drei Eigenschaften zusammen erfassen den gewöhnlichen Begriff der Totalität von Funktionen. Wir werden im nächsten Abschnitt hinreichende Kriterien vorstellen, die KEKS bzw. ihre assoziierten Termersetzungssysteme auf solche Eigenschaften überprüfen.

Bemerkung: Jede KEKS definiert genau einen Interpreter für alle in ihr definierten Funktionen (Algorithmen). Grundterme, etwa t, werden durch den Interpreter dadurch ausgewertet, daß alle in t enthaltenen Funktionen auf ihre ausgewerteten Argumente angewendet werden. Dieser Prozeß wird solange fortgesetzt, bis keine Auswertung mehr möglich ist. Wegen der Terminierung, der Konfluenz und der hinreichenden Vollständigkeit der KEKS ist das Ergebnis einer solchen Auswertung immer ein eindeutiger Konstruktorgrundterm.

Definition 50 (Semantik der KEKS) Sei $Sp = (\Sigma, \Phi, \Omega_c)$ eine KEKS mit $\Sigma = (S, \Omega)$ und $\Sigma_c = (S, \Omega)$. Die Semantik $\mathcal{M}(Sp)$ von Sp ist der monomorphe ADT $\mathcal{M}(Sp) = \{A \in Alg(\Sigma) | A \simeq C\}$, wobei C die folgende Σ-Algebra ist:

1. $C(s) = T_{\Sigma_c, s}$ für jede Sorte $s \in S$;
2. $C(w) = n$ für jedes $w = (n :\rightarrow s) \in \Omega_c$;
3. $C(w)(w_1, \ldots, w_k) = n(w_1, \ldots, w_k)$ für jedes $w = (n : s_1 \times \ldots \times \rightarrow s) \in \Omega_c$, $k \geq 1, w_i \in T_{\Sigma_c, s_i}, 1 \leq i \leq k$;
4. $C(w)(w_1, \ldots, w_k) = NF(n(w_1, \ldots, w_k))$ für $w = (n : s_1 \times \ldots \times \rightarrow s) \in \Omega - \Omega_c$, $k \geq 0$ und $w_i \in T_{\Sigma_c, s_i}, 1 \leq i \leq k$, wobei $NF(n(w_1, \ldots, w_k))$ die Normalform von $n(w_1, \ldots, w_k)$ ist.

Satz 6 *Die obige Definition der Semantik von KEKS ist konsistent.*

Beweis: Jede Funktion $C(w)$ hat einen eindeutigen Wert für gegebene Argumente und dieser Wert ist ein Konstruktorterm. Dies gilt trivialerweise, da die Spezifikation (bzw. das assoziierte Termersetzungssystem) konvergent und hinreichend vollständig ist.

□

Hilfssatz: Für alle Grundterme $t \in T_\Sigma$ gilt: $C(t) = NF(t)$.

Beweis: Sei $<$ die noethersche Ordnung in der Definition der KEKS.

1. $t = n$ mit $(n :\to s) \in \Omega_c$. Dann ist $C(t) = t = NF(t)$.
2. $t = f(t_1, \ldots, t_k)$ mit $w = (f : s_1 \times \ldots \times s_k \to s) \in \Omega_c, k \geq 1, t_i \in T_{\Sigma,s_i}, 1 \leq i \leq k$. Dann gilt:

   $C(t) = C(w)(C(t_1), \ldots, C(t_k))$
   $= f(NF(t_1), \ldots, NF(t_k))$ wegen Ind. Ann. (da $t_i < t$) und Konfluenz von R.
3. $t = f(t_1, \ldots, t_k)$ mit $w = (f : s_1 \times \ldots \times s_k \to s) \in \Omega - \Omega_c, k \geq 0, t_i \in T_{\Sigma,s_i}, 1 \leq i \leq k$. Dann gilt:

   $C(t) = C(w)(C(t_1), \ldots, C(t_k))$
   $= C(w)(NF(t_1), \ldots, NF(t_k))$ Ind. Ann., da $t_i < t$
   $= NF(f(NF(t_1), \ldots, NF(t_k)))$ Definition von C
   $= NF(f(t_1, \ldots, t_k))$ da R konfluent.

□

Satz 7 *Die kanonische Algebra C einer KEKS $Sp = (\Sigma, \Phi, \Omega_c)$ ist ein Modell von Φ.*

Beweis: Sei C die kanonische Algebra der Spezifikation Sp. Sei $r_1 = r_1' \wedge \ldots \wedge r_l = r_l' \Rightarrow f(t_1 \ldots, t_n) = t$ eine beliebige bedingte Gleichung aus Φ. Zu zeigen ist, für alle Belegungen $\sigma : X \to C$: falls $C(\sigma)(r_1) = C(\sigma)(r_i'), 1 \leq i \leq l$ gilt, dann:

$$C(\sigma)(f(t_1, \ldots, t_n)) = C(\sigma)(t)$$

Annahme: $C(\sigma)(r_1) = C(\sigma(r_i'))$ gilt für eine beliebige Belegung $\sigma : X \to C$.

Da C erzeugt ist, genügt es zu zeigen (siehe [LEW96], Seite 57), daß

$$C(\tau)(f(t_1, \ldots, t_n)) = C(\tau)(t),$$

wobei $\tau : X \to T_\Sigma$ eine beliebige Grundsubstitution ist. Wähle nun $\alpha : X \to T_{\Sigma_c}$ mit $\alpha(x) = C(\tau(x)), \forall x \in X$. Es gilt $C(\tau(r)) = C(\alpha(r))$ für jeden Term $r \in T_\Sigma(X)$ (siehe [LEW96], Seite 156). Insbesondere gilt dann:

$$C(\tau(f(t_1,\ldots,t_n))) = C(\alpha(f(t_1,\ldots,t_n))) \text{ und } C(\tau(t)) = C(\alpha(t)).$$

Zu zeigen bleibt also: $C(f(\alpha(t_1,\ldots,t_n))) = C(\alpha(t))$, aber

$$\begin{aligned} C(\alpha(f(t_1,\ldots,t_n))) &= C(w)(f(\alpha(t_1),\ldots,\alpha(t_n))) \\ &= C(w)(C(\alpha(t_1)),\ldots,C(\alpha(t_n))) \\ &= NF(f(\alpha(t_1),\ldots,\alpha(t_n))) \text{ Definition von } C \\ &= NF(f(\alpha(t))), \text{ da } f(\alpha(t_1),\ldots,\alpha(t_n)) =_\Phi \alpha(t) \text{ und } R \text{ konfluent} \\ &= C(\alpha(t)) \text{ wegen obigem Hilfssatz} \end{aligned}$$

□

Satz 8 *Sei $Sp = (\Sigma, \Phi, \Omega_c)$ eine KEKS. Die kanonische Algebra C ist initial in $Mod(\Phi)$.*

Beweis: Siehe [LEW96], Seite 158.

□

Kapitel 5

Test-sets

Das Beweissystem, das wir in dieser Arbeit vorstellen werden, beruht auf Test-sets [KNZ86], [KR90],[BR95], die im wesentlichen eine Beschreibung des initialen Modells einer Menge von Gleichungen darstellen. Ein Test-set für ein terminierendes Termersetzungssystem R ist mit anderen Worten eine endliche Menge von irreduziblen Termen, die die Menge aller bzgl. R irreduziblen Grundterme erfaßt. Test-sets eignen sich besonders für die automatische Generierung von Induktionsschemata bei Induktionsbeweisen. Sie werden auch eingesetzt, um Eigenschaften von Termersetzungssystemen (z.B. hinreichende Vollständigkeit) zu beweisen. Eine Besonderheit der Test-sets ist, daß sie die Widerlegung der Gültigkeit von Klauseln im initialen Modell einer bedingten Gleichungsmenge ermöglichen.

Für Gleichungstheorien gibt es effiziente Methoden zur Berechnung von Test-sets [KOU90], [HK88]. Die Berechnung von Test-sets für bedingte Gleichungstheorien erfordert aber induktives Beweisen und ist demnach im allgemeinen unentscheidbar. Unter bestimmten Voraussetzungen gibt es dennoch effiziente Methoden zur Konstruktion von Test-sets für bedingte Gleichungstheorien. In dieser Arbeit werden wir die Ergebnisse von [KR90] bzgl. der Berechnung von Test-sets übernehmen. Der entsprechende Algorithmus ist im *SPIKE-System* [BOU94] implementiert.

Definition 51 (Schranke eines Termersetzungssystems) Sei R ein bedingtes Termersetzungssystem. Die Schranke $D(R)$ von R ist wie folgt definiert:

$$D(R) = \begin{cases} Tiefe(R) - 1 & \text{falls S-Tiefe(R)} < \text{Tiefe(R) und R ist left-linear} \\ Tiefe(R) & \text{sonst,} \end{cases}$$

wobei $S - Tiefe(R)$ das Maximum aller Tiefen der strikten Stellen der linken Seiten der Konklusionen der Gleichungen in R ist.

Definition 52 (Test-set) Sei R ein bedingtes Termersetzungssystem. Ein Test-set T für R ist eine endliche Menge von R-irreduziblen Termen, für die gilt:

1. Für jeden R-irreduziblen Grundterm s gibt es einen Term $t \in T$ und eine Grundsubstitution σ, so daß $\sigma(t) = s$.

2. Jeder Term in T enthält nur Variablen, die ab Tiefe $d \geq D(R)$ vorkommen und infinitär sind.

Mit der ersten Eigenschaft läßt sich die Gültigkeit von Formeln durch Induktion auf dem Bereich der irreduziblen Terme statt auf der Menge aller Terme nachweisen. Sie garantiert die Vollständigkeit der Erfassung aller irreduziblen Terme durch die Test-sets. Im Gegensatz dazu ist die letzte Bedingung für die Korrektheit des Test-set-Induktionsprinzips irrelevant. Sie wird aber benötigt, um Inkonsistenzen von Klauseln zu entdecken, und spielt bei der Widerlegung der Gültigkeit von universell quantifizierten Formeln durch Konstruktion von Gegenbeispielen eine wichtige Rolle.

5.1 Die Konstruktion von Test-sets

Wir stellen eine Methode zur Berechnung von Test-sets vor, die von E. Kounalis und M. Rusinowitch in [KR90] erstmals eingeführt wurde. Die Methode basiert auf der Ableitung eines Strukturbaums für jede definierte Funktion in der Spezifikation.

Definition 53 (Strukturbaum) Sei $C = \{c(x_1, \ldots, x_n) | c \text{ ist Konstruktor}\}$. Ein Strukturbaum B für eine definierte Funktion f ist ein Baum, dessen Knoten Terme sind und dessen Wurzel $f(x_1, \ldots, x_n)$ ist. Die Söhne eines Knotens k erhält man, indem man eine Variable in k durch ein Element aus C ersetzt.

Es geht nun darum, einen Strukturbaum zu konstruieren, aus dessen Blättern man ein Test-set extrahieren kann. Die folgenden zwei Definitionen legen diejenigen Knoten fest, die expandiert werden, sowie diejenigen Variablen, die durch Konstruktorterme ersetzt werden.

Definition 54 (Erweiterbarkeit) Ein Term t ist an der Stelle u bezüglich eines bedingten Termersetzungssystems R erweiterbar, falls $t|_u$ eine Variable der Sorte s ist und entweder s finitär ist, oder es eine Regel $c \Rightarrow l \rightarrow r \in R$ gibt, so daß $l|_u$ strikt ist oder eine Variable, die mehrmals in l vorkommt. Ein Term t ist dann erweiterbar bezüglich R, falls t an einigen Stellen u erweiterbar ist.

Definition 55 (Pseudo-reduzibel) Sei R ein bedingtes Termersetzungssystem. Ein Term t ist pseudo-reduzibel durch R, falls es eine Menge von Regeln $\{c_1 \Rightarrow l_1 \rightarrow r_1, \ldots, c_n \Rightarrow l_n \rightarrow r_n\}$ in R gibt und es Stellen $u_1, \ldots, u_n$ in t gibt, so daß $t|_{u_1} = \sigma_1(l_1), \ldots, t|_{u_n} = \sigma_n(l_n)$ und $\sigma_1(c_1) \wedge \ldots \wedge \sigma_n(l_n)$ eine induktive Folgerung (siehe S. 37) von R ist.

Ein Strukturbaum B heißt vollständig, falls jedes Blatt in B entweder pseudo-reduzibel oder nicht erweiterbar ist. Ausgehend von der Wurzel $f(x_1, \ldots, x_n)$ wird der Baum an erweiterbaren Knoten solange expandiert, bis alle Blätter pseudo-reduzibel sind. Die Blätter des Baumes erfassen dann alle möglichen Argumente der Funktion f.

Satz 9 *(Kounalis, Rusinowitch) Sei R eine Menge von bedingten Regeln. Falls es für alle definierten Funktionen f einen vollständigen Strukturbaum gibt, dessen Blätter pseudo-reduzibel sind, dann läßt sich ein Test-set für R wie folgt konstruieren.*

Sei T_1 die Menge aller Argumente der Blätter aller vollständigen Strukturbäume, die man durch das obige Verfahren für alle definierten Funktionen konstruiert hat. Dann ist die folgende Teilmenge T_2 von T_1, für die gilt:

1. *Jedes Element in T_1 hat eine Instanz in T_2;*
2. *kein Element in T_2 hat eine andere Instanz in T_2 (außer sich selbst),*

ein Test-set für R.

Die erste Bedingung garantiert die Vollständigkeit der berechneten Test-sets, während die zweite dessen Minimalität gewährleistet. Diese vorgestellte Methode zur Konstruktion von Strukturbäumen kann auch dazu dienen, die hinreichende Vollständigkeit von Funktionen zu überprüfen. Das Verfahren läßt sich leicht erweitern, um nicht hinreichend vollständig definierte Funktionen zu vervollständigen. Test-sets und hinreichende Vollständigkeit eines Termersetzungssystems hängen sehr stark zusammen, wie der folgende Satz zeigt.

Satz 10 *(Naren, Zhang86) Ein Termersetzungssystem R ist genau dann hinreichend vollständig, wenn keine definierte Funktion in einem Test-set von R vorkommt.*

Es gibt eine einfache Methode zur Berechnung von Test-sets, wenn alle Funktionen hinreichend vollständig definiert sind.

Satz 11 *Sei R ein bedingtes Termersetzungssystem. Falls alle Funktionen hinreichend vollständig über freie Konstruktoren definiert sind, dann ist die Menge aller Terme t, für die gilt:*

1. $t \in T_{\Sigma_c(X)}$;
2. $Tiefe(t) \leq D(R)$;
3. *alle in t vorkommenden infinitären Variablen, kommen nur in Tiefe $D(R)$ vor,*

ein Test-set für R.

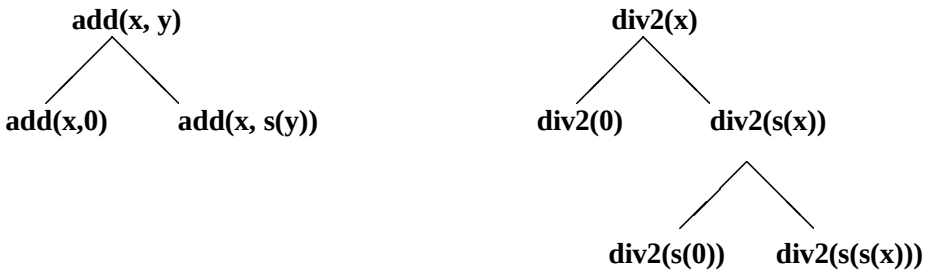

Abbildung 5.1: Strukturbaum für add und $div2$

Beispiel 3 Wir betrachten die natürlichen Zahlen, definiert durch die Konstruktoren 0 und s. Wir definieren die Addition add und die ganzzahlige Division durch 2 $div2$ ohne Rest:

$\Sigma = (\{nat\}, \Omega)$ mit
$\Omega = \{0 :\rightarrow nat,\ s : nat \rightarrow nat,\ add : nat \times nat \rightarrow nat,\ div2 : nat \rightarrow nat\}$

Betrachte die Spezifikation $Sp = (\Sigma, \Phi)$ und das zu Φ assoziierte Termersetzungssystem R:
$R = \{add(x, 0) \rightarrow 0;\ add(x, s(y)) \rightarrow s(add(x, y));\ div2(0) \rightarrow 0;\ div2(s(0)) \rightarrow 0;$
$\quad div2(s(s(x))) \rightarrow s(div2(x))\}$.
Wir konstruieren die Strukturbäume für die definierten Funktionen add und $div2$:

Alle Blätter sind pseudo-reduzibel: ein Test-set für R ist die Menge $T = \{0, s(0), s(s(x))\}$. Die Menge $M = \{0, s(x)\}$ kann nicht als Test-set gewählt werden, da sie die erste Bedingung von Satz 9 nicht erfüllt. $s(0)$ hat nämlich keine Instanz in M. Die zweite Bedingung der Test-sets wäre dadurch verletzt, daß die Tiefe der Variable x in $s(x)$ kleiner ist als $D(R) = 2$.

Wir werden Test-sets verwenden, um Induktionsschemata automatisch zu generieren. Diese Induktionsschemata werden für eine gegebene Theorie einmal berechnet im Gegensatz zu Induktionsbeweisern, wie etwa Nqthm [BOY79] oder INKA [BIHW86], wo sie bei jedem Beweis neu berechnet werden, je nach dem welche Funktionssymbole in den zu beweisenden Aussagen vorkommen. Solche Systeme verwenden Heuristiken, die sich an Rekursionsschemata der in der zu beweisenden Formel vorkommenden rekursiven Funktionen orientieren. Diese so generierten Induktionsschemata können sich aber während des Beweises als ungeeignet entpuppen. Test-sets

dagegen hängen nicht unbedingt mit den Rekursionsschemata von Funktionen zusammen, die in einer zu beweisenden Aussage vorkommen. Ein kurzer Vergleich mit anderen Systemen wird am Ende dieser Arbeit durchgeführt.

Ein entscheidender Aspekt bei der Wahl der Test-sets für die Generierung von Induktionsschemata war, daß man mit ihnen die Gültigkeit von Klauseln im initialen Modell einer Menge bedingter Gleichungen widerlegen kann, falls das assoziierte Termersetzungssystem grundkonvergent ist. Das hat zur Folge, daß der Suchraum erheblich reduziert und die Effizienz des Beweisers dadurch erhöht wird.

Kapitel 6

Induktion als Beweismittel

Induktion ist eine grundlegende Beweistechnik, die in vielen Bereichen der Informatik und der Künstlichen Intelligenz, insbesondere aber in der Verifikation, Transformation und Synthese von Programmen, Anwendung findet. Zur Automatisierung des Induktionsprinzips wurden zwei Ansätze in der Literatur vorgeschlagen: die explizite und die implizite Induktion. Letztere wird auch induktionslose Induktion genannt. Nach einer kurzen Vorstellung des Induktionsprinzips sollen die Vor- und Nachteile der jeweiligen Ansätze zur Automatisierung der Induktion dargestellt werden.

6.1 Das Prinzip der Induktion

Viele Eigenschaften abstrakter Datentypen lassen sich durch eine Menge (bedingter) Gleichungen axiomatisieren. Dennoch kann es vorkommen, daß unendlich viele Eigenschaften eines ADT's nicht aus einer endlichen Menge von (bedingten) Gleichungen logisch folgen. Um solche Eigenschaften, die nur in speziellen Modellen (erzeugten Modellen, initialem Modell oder Konstruktormodell) gültig sind, zu beweisen, muß man das Induktionsaxiom anwenden. Leider ist das Induktionsaxiom in der Prädikatenlogik erster Stufe wegen der Allquantifizierung über Prädikaten (Eigenschaften) nicht formulierbar. Die Aufnahme beliebig vieler Instanzen I_1 des Induktionsaxioms in die betrachtete Axiommenge stellt einen zufriedenstellenden Ausweg dar. Eine Instanz wird dadurch erhalten, daß man die allquantifizierte Eigenschaft durch eine feste prädikatenlogische Formel ersetzt. Man verzichtet also auf eine gewisse Ausdruckskraft zugunsten eines effektiveren Beweisbegriffs. Die Situation läßt sich am besten am Beispiel der natürlichen Zahlen illustrieren, die durch die Peano-Axiome und das Induktionsaxiom eindeutig charakterisiert werden [PEA89]. Das Induktionsaxiom $I_2 = \forall P[P(0) \wedge \forall x[P(x) \Rightarrow P(s(x))] \Rightarrow \forall x P(x)]$ ist in der $PL1$ nicht formulierbar. I_2 wird demnach durch eine unendliche Menge I_1 approximiert. Nach dem Goedelschen Unvollständigkeitssatz ist diese Beschreibung der natürlichen Zahlen nicht mehr eindeutig. Es gibt nämlich im initialen Modell gültige Aussagen über natürliche Zahlen, die mit der vorge-

schlagenen Axiomatisierung nicht ableitbar sind.

6.2 Explizite Induktion

Bei diesem Ansatz wird das Induktionsprinzip in einer offensichtlicheren Weise zum Beweis von Aussagen benutzt.

6.2.1 Schrittweise Induktion

Die schrittweise Induktion ist ein sehr wichtiges Beweisprinzip, um Eigenschaften über natürliche Zahlen zu beweisen. Sie stützt sich auf die Definition der natürlichen Zahlen und geht davon aus, daß eine Eigenschaft P für alle natürlichen Zahlen wie folgt nachgewiesen werden kann:

1. P gilt für $n = 0$;
2. falls P für eine natürliche Zahl n gilt, dann gilt P für $n + 1$.

Die schrittweise Induktion ist nicht nur auf die natürlichen Zahlen beschränkt, sondern läßt sich auf Datenstrukturen wie z.B. Bäume, Listen und Mengen verallgemeinern. Ein Schritt kann dann als der Übergang eines Objekts zum strukturell direkten Nachfolger gesehen werden. Dieses Induktionsprinzip wurde von Burstall in [BUR69] vorgeschlagen. Wenn es aber um den Beweis von Eigenschaften von Funktionen geht, die nicht über Konstruktoren definiert sind, scheitert die strukturelle Induktion. Ein Paradebeispiel ist der Beweis der Kommutativität der Funktion, die nach dem Euklidischen Algorithmus den größten gemeinsamen Teiler (ggt) von zwei natürlichen Zahlen berechnet. In diesem Fall braucht man ein allgemeineres Induktionsprinzip.

6.2.2 Noethersche Induktion

Das oben vorgestellte Induktionsprinzip läßt sich zum Prinzip der noetherschen Induktion verallgemeinern. Nach diesem Induktionsprinzip gilt eine Eigenschaft P für alle Elemente x einer wohlfundierten Menge M (mit der noetherschen Ordnung $<$), wenn

$$\forall x \in M : (\forall y \in M, y < x \Rightarrow P(y)) \Rightarrow P(x).$$

Diese Definition erlaubt es, nicht nur auf die Gültigkeit des direkten Vorgängers, sondern auch auf die Gültigkeit der Eigenschaft P aller bezüglich einer gegebenen noetherschen Ordnung kleineren Elemente zurückzugreifen. In der Informatik werden verschiedene Varianten dieses Induktionsprinzips verwendet. Die noethersche Induktion auf Terme, auch Terminduktion genannt, ist eine

dieser Varianten, die sich auf die Struktur der Terme stützt.

Terminduktion

Induktive Folgerungen sind diejenigen Formeln, die in speziellen Modellen gültig sind. Diese Modelle sind je nach Wahl entweder alle erzeugten Modelle oder das initiale Modell einer Axiommenge. Die Gültigkeit einer Formel in allen erzeugten Modellen unterscheidet sich im allgemeinen von der im initialen Modell [PAD88]. Wenn es sich um eine Gleichung handelt, ist die Gültigkeit im initialen mit der in allen erzeugten Modellen äquivalent. Aus Vereinfachungsgründen betrachten wir nur Gleichungen für den Rest dieses Abschnitts.

Satz 12 *Sei $Sp = (\Sigma, E)$ eine Gleichungsspezifikation, $\ll$ eine wohlfundierte Ordnung auf Tupeln von Grundtermen und P eine Gleichung. $P[x]$ ist dann eine induktive Folgerung von E, wenn für alle Tupel von Grundtermen $(t_1, \ldots, t_n)$*

$$E \cup \{P[s_1, \ldots, s_n] \mid (s_1, \ldots, s_n) \ll (t_1, \ldots, t_n)\} \vdash P[t_1, \ldots, t_n]$$

wobei $t_i \in T_\Sigma,\ 1 \leq i \leq n$ und $\vdash$ die Ableitbarkeit des Kalküls der Gleichungslogik ist.

Die größte Schwierigkeit bei der Automatisierung der Terminduktion liegt in der unbeschränkten Zahl der zu betrachtenden Grundterme. Deshalb versucht man, sie durch Terme aus einer endlichen Menge zu ersetzen.

Test-set-Induktion

Hier werden die in der zu beweisenden Aussage vorkommenden Variablen durch Elemente eines vorher berechneten Test-sets ersetzt. Induktion durch Test-sets läßt sich nur anwenden, wenn die betrachtete (bedingte) Gleichungsmenge in ein terminierendes Termersetzungssystem transformiert werden kann.

Satz 13 *Sei $Sp = (\Sigma, E)$ eine Gleichungsspezifikation, $<$ eine stabile noethersche Ordnung auf $T_\Sigma(X)$, P eine Gleichung, R das zu E assoziierte Termersetzungssystem und T ein Test-set für R. P ist eine induktive Folgerung von E, falls für alle $t \in T$ folgendes gilt:*

$$E \cup \{P[u] \mid u < t\} \vdash P[t],$$

wobei $u \in T_\Sigma(X)$.

Falls man beim Test-set-Induktionsprinzip für $<$ die Terminierungsordnung des Termersetzungssystems wählt, erhält man die von Reddy in [RED89] vorgeschlagene *Term Rewriting Induktion* . Es ist offensichtlich, daß die strukturelle Induktion auch ein Spezialfall der Test-set-Induktion ist. Falls die Test-sets nur aus freien Konstruktoren bestehen und als noethersche Ordnung die Teiltermordnung gewählt wird, reduziert sich die Test-set-Induktion auf die strukturelle Induktion. Verwendet man im obigen Satz Cover-sets [BEN96], [ZKK88], [BAC88] statt Test-sets, erhält man das Cover-set-Induktionsprinzip. Cover-sets sind allgemeiner als Test-sets, dementsprechend ist das auf Cover-sets basierende Induktionsprinzip ebenfalls allgemeiner. Das Problem bei der Benutzung von Cover-sets liegt bei ihrer automatischen Generierung. Cover-sets können nur automatisch aus Funktionsdefinitionen (à la Boyer-Moore oder wie in INKA) generiert werden, wenn diese vollständig definiert sind.

6.3 Implizite Induktion

Die zentrale Idee bei einem impliziten Induktionsbeweis basiert im wesentlichen auf einer Konsistenzüberprüfung zweier Gleichungsmengen. Diese Beweismethode verzichtet auf eine explizite Verwendung des Induktionsaxioms und wird deshalb induktionslose Induktion genannt. Sie geht davon aus, daß die betrachtete Axiommenge Hilbert-Post-vollständig und konsistent ist. Nach der Konsistenz-Methode der First-Order-Logik ist eine Axiommenge E Hilbert-Post-vollständig, falls sich jede mit E konsistente Formel B von E ableiten läßt. Demnach kann man zeigen, daß $E \vdash B$, indem man zeigt, daß $E \cup \{B\}$ vollständig ist (sie enthält keine Formel der Form $\phi \wedge \neg\phi$). In der Gleichungslogik ist dieses Beweisprinzip attraktiver, weil sich dort Konsistenz einfacher nachweisen läßt. Eine Gleichungsmenge wird in der Gleichungslogik als konsistent im algebraischen Sinne genau dann definiert, wenn sie kein triviales Modell besitzt. In der logischen Sicht ist eine Gleichungsmenge E konsistent, falls ihre induktive Theorie keine Gleichungen der Form $x = y$ enthält, wobei x und y Variablen sind. Es ergibt sich dann folgende Alternative zum Gültigkeitsnachweis einer Gleichung e im initialen Modell einer Gleichungsmenge E:

e ist eine induktive Folgerung von E, falls $E \cup \{e\}$ konsistent ist.

Da die Konsistenz aus der algebraischen Sicht schwer nachzuweisen ist, wurden in mehreren Varianten der induktionslosen Induktion stärker hinreichende Kriterien für Konsistenz verwendet. Der erste Vorschlag kam von Musser [MUS80A]. Seine Methode setzt voraus, daß neben der hinreichenden Vollständigkeit der definierten Funktionen, das Gleichheitsprädikat zu jeder Sorte der Signatur gehört. Inkonsistenzen, in diesem Fall Gleichungen der Form $true = false$, werden dann durch eine zu Knuth-Bendix analogen Prozedur entdeckt, vorausgesetzt, daß das zu der Gleichungsmenge assoziierte Termersetzungssystem konvergent ist. Die Methode wurde dann später durch Huet und Hullot [HUH82] verfeinert, indem die Bedingung der expliziten Axiomatisierung der Gleichheit fallengelassen wurde und Inkonsistenzen auf Gleichungen zwischen Grundkonstruktortermen reduziert wurden. Danach wurde die Methode u.a. von Dershowitz [DER82],

Jouannaud und Kounalis [JK86], Fribourg [FRI86], Narendran und Zhang [KNZ86] und Bachmair [BAC88] erweitert. Alle Varianten unterschieden sich in den hinreichenden Kriterien, die sie für Konsistenz festlegen. In der von Bachmair vorgeschlagenen Prozedur [BAC88] ist es sogar möglich, mit Gleichungen umzugehen, die nicht in Termersetzungsregeln umzuwandeln sind (z.B. die Kommutativität der Addition).

Beispiel 4 (Huet, Hullot) Betrachte die folgende Spezifikation $Sp = (\Sigma, \Phi)$ von Listen mit den Konstruktoren nil und $cons$:
$\Sigma = (\{ list \}, \Omega)$ mit
$\Omega = \{nil :\rightarrow list, cons : el \times list \rightarrow list, append : list \times list \rightarrow list, rev : list \rightarrow list\}$ und

$\Phi = \{append(nil, x) = x$; (1)
$append(cons(x, y), z) = cons(x, append(y, z))$; (2)
$rev(nil) = nil$; (3)
$rev(cons(x, y)) = append(rev(y), cons(x, nil))$; (4) $\}$

Φ erfüllt das Definitionsprinzip von Huet und Hullot, d.h. die Konstruktoren in Φ sind frei, und Φ ist hinreichend vollständig. Sei R das zu Φ assoziierte Termersetzungssystem. R ist terminierend (wähle die RPO mit folgender Präzedenz: $nil < cons < append < rev$) und konfluent.

Wir zeigen durch induktionslose Induktion, daß $rev(rev(x)) = x$ (5) eine induktive Folgerung von Φ ist. Zu zeigen ist: Der Knuth-Bendix-Algorithmus gestartet mit $\Phi \cup \{rev(rev(x))\}$, terminiert erfolgreich. Es entstehen folgende neue Regeln durch die Vervollständigung:
$rev(append(rev(x), cons(y, nil))) \rightarrow cons(y, x)$ (6)
$rev(append(x, cons(y, nil))) \rightarrow cons(y, rev(x))$ (7)

Der Algorithmus terminiert und liefert ein konvergentes Termersetzungssystem bestehend aus den Regeln (1), (2), (3), (4), (5) und (7). $rev(rev(x)) = x$ ist also eine induktive Folgerung von Φ.

6.4 Kurzer Vergleich beider Ansätze

Das Prinzip der impliziten Induktion wird von manchen Autoren als Konsistenz-Methode bezeichnet und nicht als reine induktive Methode betrachtet. Diese Kritik war für die Anhänger der impliziten Induktion nicht ohne weiteres zu akzeptieren. Deshalb wurden nach der Veröffentlichung von [GG88], einige Artikeln [RED89], [BRH94] geschrieben mit dem Ziel, die impliziten Methode als induktive Methode zu rechtfertigen und die Vorteile dieser Methode hervorzuheben. Reddy zeigte in [RED89], daß die auf der Knuth-Bendix-Vervollständigungsprozedur basierenden induktiven Beweisprozeduren tatsächlich induktive Beweise (im Sinne von Term Rewriting Induktion) liefern. Laut Bronsard [BRH94] liegt der Unterschied zwischen den beiden Methoden nur darin, daß die in der impliziten Induktion verwendete noethersche Ordnung in dem Deduktionsmechanismus (hier Termersetzung) beinhaltet ist. Sie wäre demnach als noethersche Induktion über Terme

zu betrachten. Damit wäre eine Unterscheidung der beiden Methoden überflüssig. Wir glauben trotzdem, daß es in dieser Arbeit angebracht ist, folgende Bemerkungen anzubringen:

- Die Terminierungsordnung des zu einer Gleichungstheorie assoziierten Termersetzungssystems ist die wichtigste Komponente der impliziten Induktionsmethode, da die Korrektheit des Knuth-Bendix-Algorithmus darauf basiert. Bei der expliziten Methode dagegen hängt die Korrektheit der Methode nicht mit der Terminierungseigenschaft der betrachteten Axiommenge zusammen. Ihre Effektivität ist wohl aber mit der gewählten Termersetzungsstrategie verbunden [BEN96], [DOS93]. Die Einschränkung der für die implizite Induktionsmethode benötigten fundierten Ordnung auf die Terminierungsordnung des Termersetzungssystems verlangt selbstverständlich ihren Tribut: Beweise, die eine andere Ordnung als die Terminierungsordnung brauchen, können mit der impliziten Induktionsmethode nicht erfolgreich durchgeführt werden. In solchen Fällen würde man zuerst zeigen müssen, daß das Termersetzungssystem mit der neuen Ordnung terminiert (bzw. daß beide Ordnungen kompatibel sind), und erst dann kann ein neuer Versuch gestartet werden. Dieser Schwachpunkt der induktionslosen Induktion wurde in [BAC88] teilweise behoben, dort aber wird die Methode vom Autor selbst nicht als induktionslose Induktion sondern als Beweis durch Konsistenz bezeichnet.

- Im Gegensatz zu der expliziten Methode verhält sich die implizite Methode nicht monoton. D.h. der Beweis, daß eine Gleichung e eine induktive Folgerung von E ist, ist nicht mehr mit Hilfe der Methode zu wiederholen, wenn man eine neue Gleichung e_1 in E hinzufügt, und zwar dann, wenn sich $E \cup \{e_1\}$ nicht als konvergentes Termersetzungssystem transformieren läßt.

- Der große Vorteil der auf Vervollständigung basierenden Induktionsmethode besteht darin, daß sie gegenseitige Induktion (mutual induction) in einer einfachen Weise ermöglicht. Unter einem gegenseitigen Induktionsbeweis ist ein Induktionsbeweis einer Eigenschaft P zu verstehen, bei dem man ein Lemma L benötigt, dessen Beweis aber wiederum die Eigenschaft E verlangt. Die Suche nach passenden Lemmata in Form von Generalisierungen entfällt dadurch, daß die Methode der induktionslosen Induktion bis zu einem gewissen Grad ohne Generalisierungen auskommt.

- Ein anderer erwähnenswerter Vorteil der impliziten Methode ist, daß man Induktionshypothesen als Regeln verwenden kann, ohne überprüfen zu müssen, ob sie bzgl. der verwendeten Ordnung kleiner als die zu beweisenden Formeln sind.

- Mit der impliziten Methode durchgeführte Beweise sind im allgemeinen kürzer als die mit der expliziten Induktion. In [BR95] wurde der Gilbreath-card-trick-Satz mit nur zwei Hilfssätzen gezeigt, während klassische, auf expliziter Induktion basierende Systeme, wie etwa COQ, Nqthm oder RRL, zwischen 15 und 20 Zwischenlemmata benötigten.

- Beweise mit der klassischen Induktionsmethode sind übersichtlicher und einfacher nachzuvollziehen. Die Schwierigkeit bei der expliziten Methode bleibt aber das Finden einer für

einen Beweis geeigneten noetherschen Ordnung.

- Die Auswahl der verwendeten noetherschen Ordnung tendiert bei induktionsloser Induktion mehr zu syntaktischen Ordnungen, bei den Beweisen mit expliziter Induktion mehr zu semantischen Ordnungen.

Kapitel 7

Programmsynthese

Die Entwicklung von Softwaresystemen, die in der Lage sind, nicht nur Routine, sondern auch anspruchsvolle Aufgaben zu bewältigen, war und ist nach wie vor ein Ziel einiger Teilbereiche der Informatik und der Künstlichen Intelligenz. Programmsynthese ist eine dieser anspruchsvollen Aufgaben, die sich die Automatisierung der Programmierung zum Ziel gesetzt hat. Was genau unter Programmsynthese zu verstehen ist, läßt sich mit den folgenden Sätzen zusammenfassen:

> The design and implementation of correct software meeting given requirements continues to be most relevant, practical, and scientifically challenging problem. There are many lines of research directed towards solving this problem. Automatic programming is one among those. It is the study of techniques for generating executable code from information which may be fragmentary and may only indirectly specify the target behavior.
>
> The field is based on the idea that ultimately we need to engage the machine itself in the process of programming machines, since only machines offer the important property needed for this task which is the ability to work without making mistakes. The "Automatic" in the name does not necessarily refer to a full automation of programming but rather to a considerably higher degree of automation than in other lines of software production pursued by the literature.

Alan Biermann

Die automatische Unterstützung der Programmentwicklung einerseits und die Sicherstellung der Korrektheit der erstellten Programme andererseits sind die wichtigsten Charakteristika der Programmsynthese. Bei zunehmend internationalem Wettbewerb und wegen der Erschließung neuer Anwendungsgebiete gewinnt die Korrektheit von Software immer stärkere Bedeutung. Bei der formalen Softwareentwicklung besteht der Weg von der Spezifikation bis zum lauffähigen Programm aus kleinen, mathematisch präzisen Beweisschritten, deshalb ist eine maschinelle Unterstützung

unabdingbar. Es sollen so viele Schritte wie möglich automatisch ablaufen, so daß der Bediener nur noch eingreift, wenn dies nötig ist.

Nach etwa zwanzig Jahren intensiver Forschung haben sich im Bereich der Programmsynthese zwei formale Methodologien durchgesetzt: die induktive und die deduktionsbasierte Programmsynthese.

7.1 Induktive Programmsynthese

Ansätze zur induktiven Programmsynthese gehen üblicherweise von einer Menge von Beispielen aus, die die gewünschte Ein-Ausgabe-Relation des zu synthetisierenden Programms darstellen sollen. Das gewünschte Programm läßt sich dann aus den Beispielen nach einer Reihe von Verallgemeinerungen inferieren. Induktive Programmsynthese ist ein Bereich eines Teilgebiets der Künstlichen Intelligenz, *Maschinelles Lernen* (Machine Learning) genannt, das sich mit der Automatisierung des Lernens beschäftigt. Es gibt in der Literatur eine Reihe von Ansätzen zur induktiven Programmsynthese [MUG92], [SUM77], [BIG83], [WYS87]. Wir werden uns in dieser Arbeit mit einem deduktionsbasierten Ansatz zur Programmsynthese beschäftigen.

7.2 Deduktionsbasierte Programmsynthese

Bei der deduktionsbasierten Programmsynthese wird eine formale Spezifikation vorgegeben, die die Ein-Ausgabe-Relation des gewünschten Programms beschreibt. Ausgehend von der Spezifikation werden logische oder Transformationsregeln solange angewandt, bis man ein Programm erhält. Die Ansätze der deduktionsbasierten Programmsynthese unterscheiden sich im allgemeinen qualitativ durch die Eigenschaften der festgelegten Ziel- und Spezifikationssprachen einerseits und durch die Art der während der Synthese angewandten Regeln anderseits.

7.2.1 Synthese durch Transformation

Aus einer formalen Spezifikation wird ein Programm schrittweise durch Anwendung korrektheitserhaltender Transformationsregeln konstruiert. Die Anwendbarkeit der Regeln muß noch bei einigen Schritten bewiesen werden. Systeme, die auf diesem Ansatz basieren, stellen dem Benutzer eine Menge Transformationsregeln zur Verfügung, aus der die geeigneten Regeln während der Programmentwicklung auswählt und instanziiert werden. Die ersten Transformationen einer Programmkonstruktion bringen die initiale Problemspezifikation in eine angebrachte Form, während die letzten dazu dienen, Teile des Programms zusammenzufügen. Transformationen ermöglichen es, Probleme auf kleinere Teilprobleme zurückzuführen, Teile zusammenzufügen, Iterationen oder Rekursionen einzuführen etc. Eine nützliche Charakteristik dieses Ansatzes ist die Tatsache, daß

viele Informationen in einen einzigen Schritt eingebettet werden können. Systeme, die auf diesem Ansatz basieren, findet man in [BRO83], [BUR77], [PAR90], [HOF93] und [BIB84].

7.2.2 Konstruktive Programmsynthese

Hier wird ein Programm aus einem (hinreichend) konstruktiven Beweis seiner formalen Spezifikation gewonnen. Bei diesem Ansatz läuft die Erstellung des Programms Hand in Hand mit dessen Verifikation ab. Die Spezifikation des Programms wird dabei durch eine Formel einer vorgegebenen Logik ausgedrückt. In [MAW84] wird die formale Spezifikation des gewünschten Programms als Existenzformel formuliert (oder kann als solche interpretiert werden), und die Beweise werden durch eine zweidimensionale Struktur, deduktive Tabelle genannt, dargestellt. Resolutionsbasierte Techniken und Termersetzung werden angewandt, um Beweise durchzuführen. In den letzten Jahren hat sich dieser Ansatz bei den auf konstruktiven Typtheorien basierenden Systemen [DOW], [LP92], [MAG92], [CON86] stark verbreitet. Die in diesen Typtheorien enthaltenen Logiken sind konstruktiv. Deshalb sind Beweise in solchen Typtheorien ebenfalls konstruktiv und bestehen aus einem logischen und einem algorithmischen Teil. Aus dem algorithmischen Teil eines solchen Beweises läßt sich dann das gewünschte Programm problemlos automatisch extrahieren. Die Struktur des Beweises spiegelt sich in dem aus ihm synthetisierten Programm wider; man spricht von Dualität zwischen Beweisteilen und korrespondierenden Programmteilen. Zum Beispiel hängt die Art der Induktion, die man im Beweis angewandt hat, sehr stark mit der Rekursion im extrahierten Programm zusammen.

Da viele natürliche und intuitive Beweisschritte nicht konstruktiv sind (z.B. Satz vom ausgeschlossenen Dritten), wäre es zu einschränkend, nur konstruktive Logiken zu betrachten. Es genügt, Beweise in einer klassischen Logik durchzuführen und nur solche Beweisschritte konstruktiv einzuschränken, die für die spätere Programmextraktion relevant sind.

Es gibt auch Mischformen, die sowohl zu den induktiven als auch zu den deduktiven Ansätzen gehören. Die schema-orientierte Synthese von D. Smith [SMI85] ist eine davon. Man kann aber auch die Methoden zur Programmsynthese anders kategorisieren. In [DOS93] wurden sie als Vorwärts- und Rückwärtssynthese klassifiziert. Methoden, wie die Fold-Unfold-Strategie von Burstall und Darlington [BUR77] und die Synthese durch Vervollständigung [DER87], [RED88], gehören demnach zur Vorwärtssynthese. Dort werden alle relevanten logischen Folgerungen der Programmspezifikation und der Axiomatisierung der betrachteten Theorie generiert. Aus diesen logischen Folgerungen werden dann, diejenigen herausgefiltert, die das gewünschte Programm darstellen soll. Zur Rückwärtssynthese zählen dann unter anderem die Methode von Manna und Waldinger, die das induktive Folgern in den deduktiven Prozeß integriert.

Obwohl Programmsynthese seit Jahren ein intensiv bearbeitetes Forschungsgebiet ist, sind wir der Meinung, daß man dieses Gebiet mit Konzepten, wie automatischer Lemmagenerierung und/oder Widerlegung von Spezifikationsformeln bereichern kann. Wir werden uns in dieser Arbeit mit konstruktiver Programmsynthese beschäftigen. Es wird eine Methode zur Widerlegung von Spezifika-

tionen präsentiert, die es möglich macht, fehlerhaften oder inkonsistenten Programm-Beschreibungen auf die Spur zu kommen und den großen Suchraum bei der Lösungsfindung erheblich zu reduzieren. Durch die automatische Generierung von Zwischenlemmata werden in unserem Synthesesystem viele direkte Eingriffe des Benutzers nicht mehr notwendig, um Beweise erfolgreich abzuschließen. Die Fähigkeit eines Systems, den Benutzer von aufwendigen Aufgaben zu entlasten und ihm dennoch bei entscheidenden Situationen die Möglichkeit zu geben einzugreifen, zeichnet die Leistungsfähigkeit eines Beweissystems aus.

In dieser Arbeit werden wir uns auf die Automatisierung der konstruktiven Programmsynthese konzentrieren. Die Methode beruht auf Test-sets und einfachen algebraischen Vereinfachungen. Unser erstes Ziel dabei ist, zum einen einen hohen Automatisierungsgrad zu erreichen, zum anderen aber auch eine Interaktivität zuzulassen, falls es nötig ist oder der Benutzer es wünscht (z.B. bei der Wahl der zu benutzenden noetherschen Ordnung).

Kapitel 8

Induktive Beweise f¨ur eine Klasse von Existenzformeln

Wir stellen nun unseren Ansatz zum induktiven Beweisen von Existenzformeln vor. Unsere Beweismethode beruht auf Test-sets (endliche Beschreibung des initialen Modells) und auf einfachen algebraischen Vereinfachungen [KR90]. Die Grundidee des Vereinfachungsmechanismus besteht darin, Funktionsdefinitionen und bereits bewiesene Lemmata solange auf die Problemspezifikation anzuwenden, bis man eine bezüglich einer vorgegebenen noetherschen Ordnung kleinere Instanz der zu beweisenden Aussage erhält. Dieser letzte Schritt entspricht der Anwendung der Induktionshypothese bei einem klassischen Induktionsbeweis.

Unter Problemspezifikation oder Spezifikationsklausel verstehen wir, die Beschreibung des Programms, das wir synthetisieren wollen. Bei der deduktiven Programmsynthese wird sie üblicherweise durch eine Formel der Prädikatenlogik erster Stufe dargestellt und hat im allgemeinen die Form:

$$\forall\, x\ \exists\, y\ R(x,\ y)\ \text{(*)}$$

Diese Formel besagt, daß für alle Eingaben x eine Ausgabe y existiert, die die Eingabe-Ausgabe-Relation $R(x, y)$ erfüllt.
Wir werden in dieser Arbeit eine eingeschränkte Variante von (*) betrachten, die zwar weniger expressiv, aber viel einfacher zu handhaben ist. Wegen der speziellen Form der Formeln, die wir als Problemspezifikationen zulassen werden, können wir für solche maßgeschneiderte Regeln (z.B. Termersetzung mit Prämissen, siehe Definition 59) und einen einfachen induktiven Folgerbarkeitsbegriff einführen. Außerdem läßt sich die Gültigkeit bei solchen Spezifikationsklauseln leichter widerlegen als bei Formeln wie (*), deren Ungültigkeit im initialen Modell einer bedingten Gleichungsspezifikation nur bei seltenen trivialen Fällen nachzuweisen ist.

Definition 56 (Spezifikationsklausel) Eine Spezifikationsklausel K ist eine Formel der Form:

$$\forall x_1, \ldots, x_k \exists y_1, \ldots, y_l \; \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_n \;\; k, l, n \geq 0 \text{ (**)},$$

wobei ϕ eine Konjunktion von Gleichungen und negierten Gleichungen ist, und die Φ_i, $1 \leq i \leq n$ Konjunktionen von Gleichungen sind. Wir schreiben $Varall$ bzw. $Varex$ für $\{x_1, \ldots, x_k\}$ bzw. $\{y_1, \ldots y_l\}$.

Eine Spezifikationsklausel läßt sich wie folgt interpretieren:

Unter der Annahme, daß die Formel ϕ erfüllt ist, kann die Disjunktion der Sukzedenzformeln Φ_i (Konjunktion von Gleichungen) abgeleitet werden. Eine solche Spezifikationsklausel wird im Kontext einer kanonischen erweiterten konstruktiven Spezifikation (KEKS) gegeben. Falls der Gültigkeitsnachweis einer Spezifikationsklausel im initialen Modell einer KEKS hinreichend konstruktiv geführt wird, beinhaltet er genügend Informationen darüber, wie man das spezifizierte Programm automatisch generieren kann. Diese Informationen werden die Basis für unseren Algorithmus zur Extraktion von Programmen aus Beweisen sein.

Beispiel 5 Betrachte die folgende KEKS $Sp = (\Sigma, \Phi, \Omega_c)$ mit den Konstruktoren $0, s, true$ und $false$.

$\Sigma = (\{bool, nat\}, \{+ : nat \times nat \to nat, * : nat \times nat \to nat, <: nat \times nat \to bool\})$

$\Phi = \{\; true :\to bool;$
$\quad false :\to bool;$
$\quad (x < 0) = false;$
$\quad (0 < s(x)) = true;$
$\quad s(x) < s(y) = x < y;$
$\quad 0 + x = x;$
$\quad s(x) + y = s(x + y);$
$\quad 0 * x = 0;$
$\quad s(x) * y = x * y + y\}$

Die folgende Klausel stellt dann eine Spezifikation der euklidischen Division dar, die Quotient und Rest zweier natürlichen Zahlen bestimmt.

$$\forall x, y \; \exists q, r \; \neg(y = 0) \Rightarrow x = y * q + r \wedge (r < y) = true$$

Spezifikationsklauseln sind allgemeiner als Hornklauseln (falls diese die Gleichheit als einziges Prädikat enthalten), da sie Disjunktion von Zielen in der Konklusionen erlauben und mehrere

positive Literale enthalten können.
Zur Erinnerung: Ein Ziel ist eine Konjunktion von Gleichungen.

Logische Folgerungen einer Menge von Formeln Φ sind diejenigen Formeln, die in allen Modellen von Φ gültig sind (siehe Definition 22). Induktive Folgerungen sind dagegen jene Formeln, die in speziellen Modellen von Φ, wie etwa in allen erzeugten Modellen oder im initialen Modell, gültig sind. Für Gleichungen ist die Gültigkeit in allen erzeugten Modellen mit der im initialen Modell äquivalent. Für Spezifikationsklauseln fallen die beiden Gültigkeitsbegriffe jedoch auseinander (siehe [PAD88]). Uns interessieren in dieser Arbeit Spezifikationsklauseln, die im initialen Modell einer KEKS gültig sind. Die Wahl des initialen Modells als Semantik einer KEKS Sp entspricht dem Wunsch, den durch Sp definierten Datentyp bis auf Isomorphie eindeutig zu charakterisieren. Andererseits fällt dann das Beweisen von Eigenschaften in solchen Datentypen mit dem induktiven Beweisen zusammen.

Wegen den in einer Spezifikationsklausel enthaltenen Prämissen müssen wir zwischen zwei möglichen Definitionen der induktiven Folgerbarkeit von Spezifikationsklauseln bezüglich einer KEKS wählen. Die erste besagt, daß eine Spezifikationsklausel

$$K \equiv \forall x_1, \ldots, x_n \exists y_1, \ldots, y_m \; \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l,$$

eine induktive Folgerung einer KEKS Sp ist, falls es für alle Grundsubstitutionen σ, Grundsubstitutionen η_1 und η_2 gibt, so daß :

$$Sp \cup (\eta_1(\sigma(\phi))) \models \eta_2(\Phi_i), \text{ für ein } i \leq l,$$

wobei $\eta_2(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$, $\eta_2(x) = \eta_1(x)$ für alle $x \in Dom(\eta_1)$, $Dom(\sigma) \subseteq X$, $Dom(\eta_1) \subseteq Y$ und $Dom(\eta_2) \subseteq X \cup Y$.

Diese Definition fällt mit der Gültigkeit von K in allen erzeugten Modellen von Sp zusammen, während die folgende, die wir in dieser Arbeit betrachten werden, mit der Gültigkeit im initialen Modell von Sp äquivalent ist.

Definition 57 (Induktive Folgerbarkeit einer KEKS) Sei Σ eine Signatur und Sp eine KEKS. Eine Spezifikationsklausel K der Form:

$$\forall x_1, \ldots, x_n \exists y_1, \ldots y_m \; \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$$

ist eine induktive Folgerung von Sp, falls es für alle Grundsubstitutionen σ, Grundsubstitutionen η_1 und η_2 gibt, so daß :

$$(Sp \models \eta_1(\sigma(\phi))) \text{ impliziert } (Sp \models \eta_2(\Phi_i)) \text{ für ein } i \leq l,$$

wobei $\eta_2(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$, $\eta_2(x) = \eta_1(x)$ für alle $x \in Dom(\eta_1)$, $Dom(\sigma) \subseteq X$, $Dom(\eta_1) \subseteq Y$ und $Dom(\eta_2) \subseteq X \cup Y$. Man schreibt dann: $Sp \models_{ind} K$.

Beispiel 6 Die folgenden Spezifikationsklauseln sind induktive Folgerungen der KEKS in Beispiel 4.

$\forall x, y \; \exists z \; x = y + z \vee y = x + z$
$\forall x, y \; \exists z, r \; x = z * z + r \wedge r < z + z + s(0)$
$\forall x, y \; \exists q, r \; \neg(y = 0) \Rightarrow \; x = y * q + r \wedge r < y$

Satz 14 *Eine Spezifikationsklausel K ist genau dann eine induktive Folgerung einer KEKS Sp, wenn sie im initialen Modell von Sp gültig ist.*

Beweis: Analog zum Beweis von Satz 4.3.3 in [PAD88].

Wegen der speziellen Form der Spezifikationsklauseln ist es möglich, für sie maßgeschneiderte Termersetzungsrelationen zu entwickeln. Wir werden die in Kapitel 3 (Termersetzungssysteme) eingeführte Termersetzungsrelation so erweitern, daß wir beim Überprüfen der Prämissen einer auf eine Spezifikationsklausel K anzuwendenden bedingten Regel, Lemmata und die Prämissen von K (als Implikationsformel betrachtet) ebenfalls anwenden. Wir werden weiterhin die bedingten Gleichungen Φ einer KEKS in zwei disjunkte Mengen unterteilen: die Menge der definierenden Gleichungen, die dazu dienen, neue Funktionen zu definieren, und die Menge der Lemmata (induktive Folgerungen), die zusätzliches Wissen über die betrachtete Theorie darstellen. Diese zweite Menge kann (bedingte) Gleichungen enthalten, die nicht als Regeln zu richten sind (z.B. Kommutativität der Addition). Sie werden aber nur angewandt, wenn sichergestellt wird, daß die aus der Anwendung der Gleichung resultierende Klausel bezüglich der betrachteten noetherschen Ordnung kleiner ist als diejenige vor der Anwendung. Diese Art der Termersetzung nennt man geordnete Termersetzung [BADP89], [DOS93] und erlaubt es, Terme zu reduzieren, bei denen die konventionell bedingte Termersetzung wegen der Terminierungsbedingung scheitert.

Die Termersetzungsrelation, die wir jetzt einführen, ist eine Kombination von kontextueller Termersetzung [ZHA93] und geordneter Termersetzung. Sie unterscheidet sich von der in [BR95] dadurch, daß wir bei der Reduktion einer Spezifikationsklausel nur bereits bewiesene Lemmata anwenden. Kontextuelle Termersetzung ist eine Erweiterung der klassisch bedingten Termersetzung und wurde von mehreren Autoren [ZHA93], [REM82], [GAN87], [ZKK88] untersucht. Wir orientieren uns hier an der von H. Zhang in [ZHA93] eingeführten Definition. Die Grundidee, die hinter der kontextuellen Termersetzung steckt, ist die Reduktion eines Terms mit Hilfe eines Termersetzungssystems und einer Menge von Gleichungen, die nicht als Regeln gerichtet werden können. Die Gleichungsmenge wird Kontext des zu reduzierenden Terms genannt. Bei einer Spezifikationsklausel, etwa K, läßt sich z.B. die Menge der Gleichungen (außer der, in der der zu reduzierende Term vorkommt), die als Prämissen in K vorkommen, als Kontext betrachten.

Wir führen nun einige Begriffe ein, die wir für die Definition der geordneten kontextuellen Termersetzung verwenden werden.

Definition 58 (fokussierte Gleichung) Sei $K \equiv (a_1 = b_1)^\epsilon \wedge \ldots \wedge (a_n = b_n)^\epsilon \Rightarrow (a_{n+1} = b_{n+1} \wedge \ldots \wedge a_{n+1+r} = b_{n+1+r}) \vee \ldots \vee (a_{n+1+r+l} = b_{n+1+r+l} \wedge \ldots \wedge a_{n+1+r+l+p} = b_{n+1+r+l+p})$, $n, r, l, p \geq 0$ eine Spezifikationsklausel, wobei $(a_i = b_i)^\epsilon$ für $\neg a_i = b_i$ steht, falls $\epsilon = -$ und für $a_i = b_i$, falls $\epsilon = +$, $i \leq n$. Dann heißt:

- eine fokussierte Gleichung (bzw. negierte Gleichung), die Gleichung $(a_i = b_i)^\epsilon$, $1 \leq i \leq n + 1 + r + l + p$, deren Teilterm t man mit einem Termersetzungssystem reduzieren will;
- ein Kontext für die fokussierte Gleichung (bzw. negierte Gleichung) $(a_i = b_i)^\epsilon$, $i \leq n + 1 + r + l + p$, die Menge $Präm(K, i)$ definiert durch:

$$Präm(K, i) = \{(a_j = b_j)^\epsilon \mid \epsilon = +, 1 \leq j \leq n \wedge\ i \neq j\}$$

Die skolemisierte Klausel $\overline{K}$ von K erhält man, indem man alle in K vorkommenden Variablen durch neue Konstanten ersetzt.

Bemerkung: Jede Ordnung auf Terme läßt sich zu einer Ordnung auf Terme mit neuen Konstanten erweitern.

Definition 59 (geordnete kontextuelle Termersetzung) Sei $Sp = (\Sigma, \Phi_1 \cup \Phi_2, \Omega_c)$ eine KEKS und Φ_2 eine möglicherweise leere Menge bedingter Gleichungen. Sei R das zu Φ_1 assoziierte und unter $<$ abfallende Termersetzungssystem. Sei

$K \equiv (a_1 = b_1)^\epsilon \wedge \ldots \wedge (a_n = b_n)^\epsilon \Rightarrow (a_{n+1} = b_{n+1} \wedge \ldots \wedge a_{n+1+r} = b_{n+1+r}) \vee \ldots \vee (a_{n+1+r+l} = b_{n+1+r+l} \wedge \ldots \wedge a_{n+1+r+l+p} = b_{n+1+r+l+p})$, $n, r, l, p \geq 0$

eine Spezifikationsklausel und $\overline{K}$ ihr skolemisiertes Gegenstück. Dann gilt:

$$a_i \longmapsto^{K,i}_{R[\Phi_2]} a_i' \text{ für ein i, } 1 \leq i \leq n + 1 + r + l + p,$$

falls entweder

- $\overline{a_i} \mapsto_{Präm(\overline{K},i)} \overline{a_i'}$ und $a_i > a_i'$;
- oder es eine Stelle u in a_i, eine Substitution σ und eine bedingte Regel $c_1 = d_1 \wedge \ldots \wedge c_m = d_m \Rightarrow s \rightarrow t$ in R oder ein Lemma $c_1 = d_1 \wedge \ldots \wedge c_m = d_m \Rightarrow s = t$ in Φ_2 gibt, so daß :
  1. $a_i = a_i[u \leftarrow \sigma(s)]$ und $a_i' = a_i[u \leftarrow \sigma(t)]$;
  2. $\forall k \in [1 \ldots m]\ a_i > \sigma(c_k), a_i > \sigma(d_k)$ und $a_i > a_i'$, falls das Lemma in Φ_2 angewandt wird;

3. $\forall k \in [1 \ldots m]\ \exists v_k, w_k$, so daß $\sigma(c_k) \longmapsto^{*\,K,i}_{R[\Phi_2]} v_k$, $\sigma(d_k) \longmapsto^{*\,K,i}_{R[\Phi_2]} w_k$ und $\overline{v}_k =_{Präm(\overline{K},i)} \overline{w}_k$,

wobei $=_{Präm(\overline{K},i)}$ die Kongruenzrelation ist, die durch die Prämissen $Präm(\overline{K}, i)$ induziert wird.

Die zweite Bedingung steht für den Fall, daß eine bedingte Gleichung, die nicht als Regel gerichtet werden kann, angewandt wird. Sie wird von den Termersetzungsregeln aus R trivialerweise erfüllt, sorgt aber dafür, daß der nach der Ersetzung mit Hilfe der Gleichung erhaltene Term bezüglich der gegebenen noetherschen Ordnung kleiner wird als der Term zuvor. Damit können permutative Gleichungen angewandt werden, ohne die Terminierung der Termersetzungsrelation zu gefährden. Der erste Fall der Definition ermöglicht es, Prämissen der zu beweisenden Spezifikationsklausel K bei der Vereinfachung von K in Betracht zu ziehen.

Die nach einem geordneten kontextuellen Termersetzungsschritt erhaltene Klausel ist offensichtlich kleiner (bzgl. folgender Ordnung auf Klauseln) als die initiale Klausel vor der Regelanwendung. Betrachte dazu jede Gleichung $t = s$ (bzw. negierte Gleichung) in einer Klausel als Multiset $\{s, t\}$ von Termen und jede Klausel K als Multiset von Gleichungen (also als Multiset von Multisets von Termen). Sei nun $<$ die Terminierungsordnung von R, dann ist die Multiseterweiterung $\ll_G$ von $<$ eine Ordnung auf Gleichungen [DM79]. Die zweifache Multiseterweiterung $\ll_K$ von $<$ ist dann eine Ordnung auf Klauseln. Da $<$ noethersch ist, sind auch $\ll_G$, $\ll_K$ noethersch. Aus der Definition der geordneten kontextuellen Termersetzung geht dann hervor:

$$a \ > \ a', \text{ falls } a \longmapsto^{K}_{R[\Phi_2]} \ a'.$$

Nach der obigen noetherschen Ordnung auf Klauseln gilt dann auch:

$$K[a'] \ \ll_K \ K[a],$$

wobei $K[a']$ die Klausel ist, die man aus K durch Ersetzung von a an einer beliebigen Stelle u durch a' erhält.

Wir schreiben:

$$K[a] \longmapsto^{K}_{R[\Phi_2]} K[a'] \text{ falls } a \longmapsto^{K}_{R[\Phi]} a'.$$

Beispiel 7 Sei $Sp = (\Sigma, \Phi_1 \cup \Phi_2, \Omega_c)$ die folgende KEKS mit den Konstruktoren 0 und s.

$\Sigma = (\{nat\}, \{0 :\rightarrow nat, s : nat \rightarrow nat\})$

$\Phi_1 = \{0 + x = x;$
$\quad s(x) + y = s(x + y)\}$ und

$\Phi_2 = \{x + y = y + x\}$.

Sei R das zu Φ_1 assoziierte Termersetzungssystem und lpo die lexikographische Ordnung, die wie folgt definiert ist:

$$t \equiv f(t_1, \ldots, t_m) \ >_{lpo} \ g(s_1, \ldots, s_n) \equiv s \ \text{ genau dann, wenn}$$

- entweder $t_i \geq_{lpo} s$ für ein i, $i = 1, \ldots, m$;
- oder $f > g$ in der Präzedenzordnung und $t >_{lpo} s_i$ für alle i, $1 \leq i \leq n$;
- oder $f = g$ $(m = n)$, $(t_1, \ldots, t_m) >>_{lpo} (s_1, \ldots, s_m)$ und $t >_{lpo} t_i$, für alle i, $1 \leq i \leq m$,

wobei $\geq$ eine Präzedenz auf Σ ist und $>>_{lpo}$ die (links-rechts) lexikographische Erweiterung von $>$ ist.
Betrachte nun die lexikographische Pfadordnung, die durch die folgende Präzedenz $<$ induziert wird: $0 < s < + <$. R terminiert trivialerweise unter $<_{lpo}$ (rechts-links). Sei nun die folgende Spezifikationsklausel gegeben:

$$K \equiv \forall x \ \exists y \ x = x + y \Rightarrow x + s(x + y) = s(x + x)$$

K läßt sich durch geordnete kontextuelle Termersetzung zeigen. Da $s(x+y)+x <_{lpo} x+s(x+y)$, können wir die Gleichung in Φ_2 anwenden und erhalten:

$\forall x \ \exists y \ x = x + y \Rightarrow s(x + y) + x = s(x + x)$;

$\forall x \ \exists y \ x = x + y \Rightarrow s(x + y + x) = s(x + x)$ durch die zweite Regel in Φ_1;

$\forall x \ \exists y \ x = x + y \Rightarrow s(x + x) = s(x + x)$ da $\overline{x} + \overline{y} \mapsto_{\overline{x}=\overline{x}+\overline{y}} \overline{x}$ und $s(x + x) <_{lpo} s(x + y + x)$.

Damit ist K bewiesen, was uns ohne geordnete kontextuelle Termersetzung nicht gelungen wäre.

Beim Versuch eine Spezifikationsklausel durch ein bedingtes Termersetzungssystem zu vereinfachen, kann es passieren, daß sich keine Regel anwenden läßt, obwohl Teilterme der zu beweisenden Klausel K Instanzen der linken Seiten einiger bedingter Regeln aus R bilden. Dies ist der Fall, wenn die Prämissen der entsprechenden Regeln nicht erfüllt sind. In einer solchen Situation kann man aber trotzdem die zu beweisende Klausel mit Hilfe der Regeln auf einfachere Teilformeln zurückführen, vorausgesetzt, die Disjunktion der Prämissen der in Frage kommenden Regeln bildet eine vollständige Fallunterscheidung. Jede dieser Teilformeln erhält man aus K, indem man die in K vorkommende Instanz der linken Seite der bedingten Regeln durch die entsprechende rechte Seite ersetzt. Als Zusatzbedingung werden dann die Prämissen der angewandten Regeln in die Prämissen der neu erhaltenen Klauseln eingefügt. Diese Termersetzungstechnik, die wir

im folgenden als Fallunterscheidung bezeichnen werden, spielt beim automatischen Beweisen mit bedingten Termersetzungssystemen eine wichtige Rolle. Varianten dieser Technik wurden in der Literatur von mehreren Autoren in [BRH94], [KR90], [BL93], [BR95] vorgestellt.

Definition 60 (Fallunterscheidung) Seien $\phi_i \Rightarrow l_i \rightarrow r_i,\ 1 \leq i \leq n$ Regeln eines zu einer KEKS Sp assoziierten Termersetzungssystems R und K eine Spezifikationsklausel. Falls es Substitutionen σ_i und Stellen u_i in K gibt, so daß

$$K|_{u_i} = \sigma_i(l_i),\ R \models_{ind} (\sigma_1(\phi_1) \vee \ldots \vee \sigma_n(\phi_n)) \text{ und } Var(K|_{u_i}) \subseteq Varall(K),\ 1 \leq i \leq n$$

dann

$$Fallunterscheidung(K) = \{\sigma_i(\phi_i) \Rightarrow K[u_i \leftarrow \sigma_i(r_i)] \mid 1 \leq i \leq n\},$$

wobei $\models_{ind}$ für die induktive Folgerbarkeit im initialen Modell von Sp steht.

Beispiel 8 Wir betrachten wieder die KEKS aus Beispiel 2 sowie die folgende Spezifikationsklausel:

$$K \equiv \forall m_1 : list, a_1, x : nat\ \exists\, l_2 : list.\ del(x, cons(a_1, m_1)) = l_2.$$

Dann ist $K' \equiv Fallunterscheidung(K)$

$$K' = \begin{cases} \forall m_1 : list, a_1, x : nat\ \exists\, l_2 : list\ eq(x, a_1) = true \Rightarrow m_1 = l_2 \\ \forall m_1 : list, a_1, x : nat\ \exists\, l_2 : list\ eq(x, a_1) = false \Rightarrow cons(a_1, del(x, m_1)) = l_2, \end{cases}$$

da $R \models_{ind} (\forall x, a_1\ eq(x, a_1) = true \vee eq(x, a_1) = false)$ trivialerweise erfüllt ist und $Var(del(x, cons(a_1, m_1))) \subseteq Varall(K)$.

Bemerkung: Die Disjunktion der Prämissen der Regeln, die für eine Fallunterscheidung in Frage kommen, ist wiederum eine Spezifikationsklausel. Die induktive Folgerbarkeit solcher Formeln bezüglich der betrachteten KEKS kann demnach innerhalb unseres Systems bewiesen werden.

8.1 Induktionsvariablen

Beim induktiven Beweisen, sei es von Hand oder automatisch, ist die Wahl der Variablen, über die man induziert, von entscheidender Bedeutung. Da eine Spezifikationsklausel in der Regel nur

einige allquantifizierte Variablen enthält, könnte man meinen, daß der Suchraum in dieser Hinsicht nicht sehr groß wäre, so daß man ohne weiteres alle Möglichkeiten durchspielen könnte. Dies ist aber leider nicht der Fall, wie die folgende Überlegung deutlich illustriert.

Sei Sp eine KEKS, R das zu Sp assoziierte Termersetzungssystem, T ein Test-set für R und K eine zu beweisende Spezifikationsklausel. Mit unserer Beweismethode, die wir im nächsten Abschnitt ausführlich vorstellen, werden die allquantifizierten Variablen in K mit den Elementen von T der entsprechenden Sorten zuerst instanziiert. Ausgehend von den so erhaltenen Lemmata wird dann ein Vereinfachungsprozeß gestartet und angewandt, bis man auf Tautologien oder auf bezüglich einer stabilen und noetherschen Ordnung kleineren Instanzen der zu beweisenden Formeln kommt. Sei nun n_i die Anzahl der allquantifizierten Variablen in K der Sorte s_i und m_i die Anzahl der Elemente von T der Sorte s_i. Dann entstehen mit einer naiven Methode, die darauf basiert, alle in K vorkommenden allquantifizierten Variablen als Induktionsvariablen zu betrachten, $\prod_{i=1}^{r} m_i^{n_i}$ zu beweisende Lemmata, wobei r die Anzahl der Sorten der in K auftretenden allquantifizierten Variablen ist. Diese Zahl kann schon bei einfachen Beispielen unannehmbar groß sein, so daß bei den meisten automatischen Beweissystemen [BOY79], [BOU94], [BIHW86] Heuristiken eingesetzt werden, die ausgehend von den in der zu beweisenden Formel vorkommenden Funktionen die Induktionsvariablen bestimmen. Die Idee, die hinter diesen Verfahren steckt, beruht darauf, daß Induktionsvariablen als jene Variablen betrachtet werden, die nur in Teiltermen vorkommen, die erst nach einer Instanziierung dieser Variablen durch die Regeln des betrachteten Termersetzungssystems zu reduzieren sind. Mit dem Einsatz solcher Heuristiken für die Bestimmung der Induktionsvariablen wird die Zahl der zu beweisenden Formelinstanzen drastisch reduziert.

Definition 61 (Induktionsvariablen) Sei Sp eine KEKS, R das zu Sp assoziierte Termersetzungssystem und K eine Spezifikationsklausel. Die Menge der induktiven Variablen von K $Indvar(K)$ ist die kleinste Teilmenge der allquantifizierten Variablen von K, so daß :

- falls x die linke bzw. die rechte Seite einer Gleichung in K (etwa $x = s,\ s \in T_\Sigma(x)$) bildet, dann $x \in Indvar(k)$;
- falls x in einem strikten Teilterm t von K an der Stelle u vorkommt und es eine Regel $\phi \Rightarrow l \rightarrow r \in R$ gibt, so daß t und l unifizierbar sind und
  - entweder u ist strikt in l;
  - oder $l|_u$ ist nicht-linear;
  - oder $l|_u \in (\cup_{i\in[1\ldots n]}\{Indvar(a_i), Indvar(b_i)\})$,

  dann $x \in Indvar(K)$.

Die obige Definition ist eine leicht veränderte Version der in SPIKE [BOU94] implementierten Definition von Induktionsvariablen.

Notation: Wir schreiben $Subind(K)$ für die Menge aller Substitutionen, die Induktionsvariablen von K in das betrachtete Test-set T abbilden.

Beispiel 9 Wir betrachten die natürlichen Zahlen, definiert durch die Konstruktoren 0 und s.

$\Sigma = (\{nat\}, \Omega)$ mit
$\Omega = \{0 :\rightarrow nat, s : nat \rightarrow nat, + : nat \times nat \rightarrow nat\}$

Betrachte die fogende KEKS $Sp = (\Sigma, \Phi, \Omega_c)$ und das zu Φ assoziierte Termersetzungssystem R:

$R = \{x + 0 \rightarrow x;\ x + s(y) \rightarrow s(x + y)\}$

Wir wählen $T = \{0,\ s(x)\}$ als Test-set für R. Sei nun die folgende Spezifikationsklausel K zu beweisen:

$$\forall\, x_1, x_2, x_3\ \exists z\ x_1 + (x_2 + x_3) = z + x_3$$

Wenn wir alle Möglichkeiten durchgespielt hätten, würden die 8 folgenden zu beweisenden Lemmata entstehen:

$\exists z\ 0 + 0 + 0 = z + 0$
$\forall x'_3\ \exists z\ 0 + 0 + s(x'_3) = z + s(x'_3)$
$\forall x'_2\ \exists z\ 0 + s(x'_2) + 0 = z + 0$
$\forall x'_2, x'_3\ \exists z\ 0 + s(x'_2) + s(x'_3) = z + s(x'_3)$
$\forall x'_1\ \exists z\ s(x'_1) + 0 + 0 = z + 0$
$\forall x'_1, x'_3\ \exists z\ s(x'_1) + 0 + s(x'_3) = z + s(x'_3)$
$\forall x'_1, x'_2\ \exists z\ s(x'_1) + s(x'_2) + 0 = z + 0$
$\forall x'_1, x'_2, x'_3\ \exists z\ s(x'_1) + s(x'_2) + s(x'_3) = z + s(x'_3)$

Nach dem obigen Verfahren sind nun nur folgende 2 Lemmata zu beweisen, da nur x_3 als Induktionsvariable in Frage kommt:

$\forall x_1, x_2\ \exists\ z\ x_1 + x_2 + 0 = z + 0$
$\forall x_1, x_2, x'_3\ \exists\ z\ x_1 + x_2 + s(x'_3) = z + s(x'_3)$

Beispiel 10 Betrachten wir nun das Beispiel 3, erweitert mit der Funktion $rest2$, die wie folgt definiert ist:

$div2(x) + div2(x) = x \Rightarrow rest2(x) = 0;$
$s(div2(x) + div2(x)) = x \Rightarrow rest2(x) = s(0).$

Sei nun die folgende zu beweisende Formel Φ:

$$\forall x\ \exists\ y\ rest2(x) = y \vee rest2(x) = s(y).$$

Es ist offensichtlich, daß hier x als Induktionsvariable gewählt werden muß, denn durch die letzte Bedingung der obigen Definition für Induktionsvariablen läßt sich x als Induktionsvariable festlegen, da $Indvar(\Phi) = Indvar(div2(x))) = \{x\}$.

Es kann dennoch vorkommen, daß keine der Bedingungen in der obigen Definition für Induktionsvariablen greifen. In einem solch seltenen Fall wählen wir alle allquantifizierten Variablen als Induktionsvariablen.

Während eines Gültigkeitsbeweises einer Spezifikationsklausel K im initialen Modell einer KEKS (bzw. während eines Beweises einer existenziell quantifizierten Formel durch Induktion) spielt nicht nur die Wahl der Induktionsvariablen eine wichtige Rolle, sondern auch die der zu instanziierenden Existenzvariablen. Eine gute Wahl dieser Variablen, die mit den Elementen des betrachteten Test-sets ersetzt werden, gefolgt von simplen algebraischen Vereinfachungen (Termersetzungen, Lemmaanwendungen) kann rasch zu einer Reduktion der zu beweisenden Formel auf eine Tautologie oder auf eine bzgl. einer vorgegebenen stabilen noetherschen Ordnung kleinere Instanz von K führen.

Die während eines Beweises zu instanziierenden Variablen werden bei unserer Beweismethode jene existenziell quantifizierten Variablen sein, die die Definition 61 erfüllen. Induktionsvariablen und die in einem Beweis zu instanziierenden existenziell quantifizierten Variablen werden demnach nach denselben Kriterien bestimmt.

Definition 62 (Zu instanziierende Existenzvariable) Eine existenziell quantifizierte Variable wird eine zu instanziierende Existenzvariable genannt, falls sie eine der Bedingungen in Definition 61 erfüllt.

Notation: Wir schreiben $Subex(K)$ für die Menge aller Substitutionen, die die zu instanziierenden Existenzvariablen von K in das betrachtete Test-set T abbilden.

8.2 Informaler Überblick unserer Beweismethode

In Kapitel 6 haben wir die beiden wichtigsten Ansätze zur Automatisierung der Induktion kennengelernt. Beide Ansätze zeigten einige Schwächen: Zum einen erfordert die explizite Methode das Auffinden von zum Erfolg führenden Induktionshypothesen und Lemmata, zum anderen verlangen Beweise durch Konsistenz im allgemeinen entweder die Konvergenz oder die Grundkonvergenz des Termersetzungssystems bestehend aus der initialen Axiommenge und der zu beweisenden Formel (i.a. Gleichung). In den letzten Jahren wurde ein neuer Ansatz vorgeschlagen, der die Stärken der expliziten Methode und die des Beweisens durch Konsistenz in sich vereinigt [KR90], [KRU90], [RED89], [BR95]. Das System SPIKE [BOU94] basiert auf dieser Kombination der expliziten und impliziten Induktion.

Wir werden in dieser Arbeit eine auf Test-sets basierende Induktionsmethode kennenlernen, bei der Test-sets und damit verfeinerte Induktionsschemata automatisch generiert werden. Diese Methode kann als Erweiterung der auf Test-sets basierenden Beweismethoden angesehen werden, da sie Existenzquantoren in Formeln zuläßt.

8.2.1 Vorgehensweise

Vorgegeben sind:

- eine KEKS Sp;
- eine zu beweisende Spezifikationsklausel K;
- eine stabile und auf Grundterme noethersche Ordnung $<$ (optional: siehe untere Bemerkung) und ihre Multiseterweiterung $\ll$.

1. Aus dem zu Sp assoziierten Termersetzungssystem R wird zuerst ein Test-set berechnet (Da Sp vollständig ist und die Konstruktoren frei sind, läßt sich ein Test-set für R leicht berechnen: siehe Satz 11);
2. die Induktionsvariablen und die zu instanziierenden Existenzvariablen in K werden bestimmt;
3. die Induktionsvariablen werden dann durch die Elemente des Test-sets der entsprechenden Sorten ersetzt, wodurch eine Menge von Instanzen der initialen Spezifikationsklausel entsteht. Jede dieser Instanzen wird mit allen möglichen Kombinationen der Elemente des Test-sets für die zu instanziierenden Variablen expandiert. Jede dieser so entstandenen Formelmengen, etwa M_1, wird mit Hilfe des Termersetzungssystems und der Lemmata solange vereinfacht, bis man auf eine Tautologie (Fall a) oder auf eine Formelmenge M_2 kommt, die von einer Instanz von K subsumiert wird, die bzgl. $<$ kleiner als M_2 ist (Fall b). Dieser letzte Fall entspricht der Anwendung der Induktionshypothese bei einem klassischen Beweis durch Induktion. Falls keiner der Fälle (a) und (b) auftritt, wird M_1 als neue zu beweisende Spezifikationsklausel betrachtet, und der Prozeß wiederholt.

Bemerkung: Die Vorgabe einer noetherschen Ordnung ist optional. Falls der Benutzer keine Ordnung vorgibt, wird automatisch die Terminierungsordnung des Termersetzungssystems R gewählt. Das zu Sp assoziierte Termersetzungssystem R ist auf jeden Fall terminierend, da Sp eine KEKS ist. Wir gehen davon aus, daß diese Terminierungsordnung dem System bekannt ist.

Wir stellen nun zwei einfache Beispiele vor, um die Beweismethode zu illustrieren.

Beispiel 11 Betrachte die folgende KEKS $Sp = (\Sigma, \Phi, \Omega_c)$ mit den Konstruktoren $0,\ s, true$ und $false$, wobei

$\Sigma = (\{bool, nat\}, \{0 :\rightarrow nat;\ s : nat \rightarrow nat;\ true :\rightarrow bool;\ false :\rightarrow bool;\ + : nat \times nat \rightarrow nat;\ \geq : nat \times nat \rightarrow bool\})$

und $\Phi = \{\ x + 0 \rightarrow x;$
$x + s(y) \rightarrow s(x + y);$
$0 \geq s(x) = false;$
$x \geq 0 = true;$
$s(x) \geq s(y) = x \geq y\}.$

Sei nun die folgende Spezifikationsklausel K zu beweisen:

$$\forall x, y\ \exists\ z\ x \geq y = true \Rightarrow x = z + y$$

- Gemäß Definition 61 sind sowohl x als auch y als Induktionsvariablen zu betrachten. Gemäß Definition 62 brauchen wir z während des Beweises nicht zu instanziieren.
- Als Ordnung wählen wir die Anzahlordnung (count ordering) auf den natürlichen Zahlen.
- Als Test-set betrachten wir die Menge $T = \{0,\ s(x),\ true,\ false\}$.
- Es gibt dann für x und y die folgenden 4 Test-set-Substitutionen:

$$\{(x \leftarrow 0, y \leftarrow 0),\ (x \leftarrow 0,\ y \leftarrow s(y')),\ (x \leftarrow s(x'), y \leftarrow 0),\ (x \leftarrow s(x'), y \leftarrow s(y'))\}$$

Folgende Instanzen von K sind demnach zu beweisen:

1. $\exists z \quad 0 \geq 0 = true \Rightarrow 0 = z + 0;$
2. $\forall y' \quad \exists z \quad 0 \geq s(y') = true \Rightarrow 0 = z + s(y');$
3. $\forall x' \quad \exists z \quad s(x') \geq 0 = true \Rightarrow s(x') = z + 0;$
4. $\forall x', y' \exists z \quad s(x') \geq s(y') = true \Rightarrow s(x') = z + s(y')$

Sie lassen sich mit Hilfe des zu Φ assoziierten Termersetzungssystems R und wegen der Freiheit der Konstruktoren in Sp zu folgenden Formeln vereinfachen:

1.1 $\exists z\ true = true \Rightarrow 0 = z;$
2.1 $\forall y' \quad \exists z \quad false = true \Rightarrow 0 = s(z + y');$
3.1 $\forall x' \quad \exists z \quad true = true \Rightarrow s(x') = z;$
4.1 $\forall x', y' \exists z \quad x' \geq y' = true \Rightarrow x' = z + y'$

1.1, 2.1 und 3.1 sind Tautologien, während 4.1 eine Instanz von K ist, die kleiner als 4 ist, da $(x', y') \ll (s(x'), s(y'))$, wobei $\ll$ die Multiseteerweiterung der Anzahlordnung ist.

Damit ist die Gültigkeit von K im initialen Modell von Sp gezeigt.

Im allgemeinen werden mehrere Schleifen benötigt, um einen Beweis abzuschließen, wie das folgende Beispiel zeigt.

Beispiel 12 Gegeben sei nun folgende KEKS $Sp = (\Sigma, \Phi_1 \cup \Phi_2, \Omega_c)$ mit

$\Sigma = (\{nat, bool\}, \{0 :\to nat, s : nat \to nat,\ true :\to bool,\ false :\to bool,\ even : nat \to bool,\ odd : nat \to bool\})$;

$\Omega_c = \{0, s, true, false\}$;

$$\begin{aligned}\Phi_1 = \{\ & x + 0 = x;\\ & x + s(y) = s(x + y);\\ & even(0) = true;\\ & even(s(x)) = odd(x);\\ & even(x) = true \Rightarrow odd(x) = false;\\ & even(x) = false \Rightarrow odd(x) = true\}\end{aligned}$$

$$\begin{aligned}\text{und } \Phi_2 = \{\ & x + y = y + x;\\ & even(s(s(x))) = even(x)\}\end{aligned}$$

Sei nun die folgende Spezifikationsklausel K zu beweisen:

$$\forall x\ \exists\ y\ even(x) = true \Rightarrow x = y + y$$

- Gemäß Definitionen 61 und 62 ist x als Induktionsvariable und y als zu instanziierende Variable zu wählen;
- als Ordnung wählen wir die lexikographische Pfadordnung $<_{lpo}$ (von rechts nach links) mit folgender Präzedenz $0 < s < true < false < even < odd$;
- Wir wählen wie im letzten Beispiel das Test-set $T = \{0, s(x), true, false\}$

Es sind nun folgende Instanzen von K zu beweisen:

$$\begin{array}{lll} 1. & \exists\ y & even(0) = true \Rightarrow 0 = y + y;\\ 2.\ \forall x' & \exists\ y & even(s(x')) = true \Rightarrow s(x') = y + y \end{array}$$

Die Existenzvariable y wird jetzt durch 0 bzw. $s(y')$ ersetzt, und wir erhalten folgende Instanzen:

1.1 $even(0) = true \Rightarrow 0 = 0 + 0$;
1.2 $\exists y'$ $even(0) = true \Rightarrow 0 = s(y') + s(y')$;
2.1 $\forall x'$ $even(s(x')) = true \Rightarrow s(x') = 0 + 0$;
2.2 $\forall x'$ $\exists y'$ $even(s(x')) = true \Rightarrow s(x') = s(y') + s(y')$.

Da 1.1 eine Tautologie ist, haben wir einen Wert für y gefunden, wenn x gleich 0 ist. Es soll nun ein Wert für y gefunden werden, wenn sich x als $s(x')$ darstellen läßt.
2.1 führt zu einer Sackgasse, und wird nicht weiter betrachtet. Auf 2.2 wenden wir wieder unsere Beweismethode an. Daraus entstehen folgende Formeln:

2.2.1 $\exists y'\ even(s(0)) = true \Rightarrow s(0) = s(y') + s(y')$;
2.2.2 $\forall x''\ \exists y'\ even(s(s(x''))) = true \Rightarrow s(s(x'')) = s(y') + s(y')$

2.2.1 läßt sich zu einer Tautologie vereinfachen, da $even(s(0)) \rightarrow_R^* false$. Bei 2.2.2 wird die Existenzvariable y' durch 0 bzw. $s(y'')$ ersetzt, und wir erhalten:

2.2.2.1 $\forall x''\ even(s(s(x''))) = true \Rightarrow s(s(x'')) = s(0) + s(0)$;
2.2.2.2 $\forall x''\ \ \exists\, y''\ \ even(s(s(x''))) = true \Rightarrow s(s(x'')) = s(s(y'')) + s(s(y''))$.

2.2.2.1 führt zu einer Sackgasse, während 2.2.2.2 sich wie folgt vereinfachen läßt:

$\forall x''\exists\ y''\ even(x'') = true \Rightarrow s(x'') = s(s(y'')) + s(y'')$ wegen der Gleichung in Φ_2 und der Freiheit der Konstruktoren;

zudem gilt:

$\forall x''\exists\ y''\ even(x'') = true \Rightarrow s(x'') = s(y'') + s(s(y''))$ wegen der Kommutativität von $+$ und weil $s(y'') + s(s(y'')) <_{lpo} s(s(y'')) + s(y'')$;

daraus folgt:

(*) $\forall x''\ \exists\, y''\ even(x'') = true \Rightarrow x'' = s(y'') + s(y'')$ wegen der zweiten Regel der Addition und wegen der Freiheit der Konstruktoren.

Aber (*) ist eine Instanz von K, die kleiner ist als 2.2.2. Damit ist die Gültigkeit von K im initialen Modell von Sp bewiesen.

Bemerkung: Im obigen Beispiel wurde die Kommutativität der Addition in den Beweis mit einbezogen. Dies war nur möglich, weil unsere Methode geordnete Termersetzung zuläßt.

Fakt: Wie bei der impliziten Beweismethode gibt es auch bei unserer Methode keine Hierarchie zwischen den zu beweisenden Formeln. Im letzten Beispiel wurde z.B. die Formel K mit Hilfe

eines Zwischenlemmas bewiesen, dessen Beweis wiederum die ursprüngliche Formel K benötigte (mutual induction). Das für den Beweis benötigte Zwischenlemma wurde ohne Eingriff des Benutzers generiert und bewiesen. Ein Vorteil der impliziten Beweismethode ist, daß sie gerade solche Beweise zuläßt.

Im letzten Beispiel sind aus der Formel 1 (bzw. 2) die Unterfälle 1.1 und 1.2 (bzw. 2.1 und 2.2) dadurch entstanden, daß die Existenzvariable y durch die Elemente aus dem Test-set $T = \{0, s(y')\}$ ersetzt wurde. Der Nachweis der Gültigkeit eines Unterfalls 1.1 oder 1.2 (bzw 2.1 oder 2.2) genügt, um die Gültigkeit der Formel 1. (bzw. 2.) nachzuweisen. Dies ist aber nur möglich, wenn keine Fallunterscheidung für den weiteren Verlauf des Beweises notwendig ist. Falls eine Fallunterscheidung unumgänglich ist, läßt sich eine Instanz einer Spezifikationsklausel K im allgemeinen nicht getrennt von den anderen beweisen, wohl aber die Disjunktion jener Instanzen von K, die aus K nach Ersetzen der Existenzvariablen durch den Elementen des betrachteten Test-sets entstehen. Letztere kann wiederum in eine Spezifikationsklausel umgeformt werden (siehe unten), falls in den Prämissen von K keine Existenzvariablen vorkommen. Während dieser Transformation werden die zu beweisenden Formeln mit den Substitutionen markiert, die auf sie angewandt worden sind. Durch die Markierung läßt sich im Beweis später feststellen, welche Instanzen der zu beweisenden Formel als Induktionsannahme angewandt werden können. Sie hat außerdem die weitere Eigenschaft, die Konstruktivität der Beweise zu verdeutlichen. Diese Markierungen spielen deshalb eine Schlüsselrolle bei der Extraktion von Programmen aus Beweisen (siehe Kapitel 10). Wir unterscheiden zwei Arten von Markierungen: Die äußeren Markierungen, in denen die auf die Induktionsvariablen bereits angewandten Substitutionen gespeichert werden und die inneren Markierungen, die jene Substitutionen festhalten, die auf die Existenzvariablen angewandt worden sind. Sie werden bei jeder Initialisierung neuer Induktionsschritte aktualisiert, indem man die die alten mit den neuen angewandten Substitutionen überschreibt.

Definition 63 (Umformung expandierter Klauseln in Spezifikationsklauselform)
Sei $K \equiv < \tau > : \forall X \exists Y\ \phi \Rightarrow < \tau_{\mathbf{1}} > : \Phi_1 \vee \ldots \vee < \tau_{\mathbf{n}} > : \Phi_n$ eine Spezifikationsklausel, für die gilt: $Var(\phi) \cap Y = \emptyset$. Seien σ, $\eta_i,\ 1 \leq i \leq l$ Substitutionen, für die gilt: $Dom(\sigma) \subseteq X$, $Dom(\eta_i) \subseteq Y$ und $Img(\eta_i) \cap Img(\eta_j) = \emptyset$ für $i \neq j$. Dann ist

$Norm(\bigvee_{i=1}^{l} \eta_i(\sigma(K))) \equiv < \sigma > : \forall X' \exists Y'\ \sigma(\phi) \Rightarrow\ < \eta_{\mathbf{1}} > : (\eta_1(\sigma(\Phi_1)) \vee \ldots \vee \eta_1(\sigma(\Phi_n))) \vee$
$\ldots \vee < \eta_{\mathbf{l}} > : (\eta_l(\sigma(\Phi_1)) \vee \ldots \vee \eta_l(\sigma(\Phi_n)))$,

wobei $X' = (X \backslash Dom(\sigma)) \cup Var(Img(\sigma))$ und $Y' = (Y \backslash (\cup_{i=1}^{l} Dom(\eta_i))) \cup (\cup_{i=1}^{l} Var(Img(\eta_i)))$.

Für eine Spezifikationsklausel K wird dann $Norm(\bigvee_{i=1}^{l} \eta_i(\sigma(K)))$ die mit σ und $\eta_i,\ 1 \leq i \leq l$ expandierte Instanz von K genannt, wobei l die Anzahl der Test-set-Substitutionen für die zu instanziierenden Existenzvariablen in K ist. Die Prämisse ϕ der expandierten Instanz von K wird der Übersichtlichkeitshalber nicht markiert, zumal alle während eines Beweises benötigte Informationen über die angewandten Substitutionen in ihrer äußeren Markierung enthalten sind.

Eine Spezifikationsklausel K läßt sich mit allen Test-set-Substitutionen $\eta_i,\ i \geq 1$ für ihre zu instanziierenden Existenzvariablen in eine Spezifikationsklausel K_1 äquivalent expandieren, indem man die Disjunktion jener Instanzen von K bildet, die aus K durch Anwendung der η_i entstehen.

Satz 15 *Sei Sp eine KEKS, R das zu Sp assoziierte Termersetzungssystem und T ein Test-set für R. Seien weiterhin $K \equiv \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_n$ eine Spezifikationsklausel und $\eta_i,\ 1 \leq i \leq l$ die Test-set-Substitutionen für die zu instanziierenden Existenzvariablen. Falls $Var(\phi) \cap Y = \emptyset$, dann gilt:*

$$Sp \models_{ind} K \Leftrightarrow Sp \models_{ind} Norm(\bigvee_{i=1}^{l} \eta_i(K)).$$

Beweis: $Norm(\bigvee_{i=1}^{l} \eta_i(K)) = \forall X \exists Y' \phi \Rightarrow \eta_1(\Phi_1) \vee \ldots \vee \eta_1(\Phi_n) \vee \ldots \vee \eta_l(\Phi_1) \vee \ldots \vee \eta_l(\Phi_n)$
Weil $Var(\phi) \cap Y = \emptyset$ gilt auch: (*) $Sp \models_{ind} Norm(\bigvee_{i=1}^{l} \eta_i(K)) \Leftrightarrow$ Für alle Grundsubstitutionen σ gibt es eine Grundsubstitution τ sowie i und $j,\ 1 \leq i \leq n,\ 1 \leq j \leq n$, so daß gilt:

$$(Sp \models \sigma(\phi)) \text{ impliziert } (Sp \models (\tau(\eta_i(\Phi_j)))) \text{ und } \tau(x) = \sigma(x) \text{ für alle } x \in Dom(\sigma).$$

(**) $Sp \models_{ind} K \Leftrightarrow$ Für alle Grundsubstitutionen τ_1 gibt es eine Grundsubstitution τ_2 und ein $k,\ 1 \leq k \leq n$, so daß gilt:

$$(Sp \models \tau_1(\phi)) \text{ impliziert } (Sp \models \tau_2(\Phi_k)) \text{ und } \tau_2(x) = \tau_1(x) \text{ für alle } x \in Dom(\tau_1).$$

„$\Leftarrow$“:
Annahme: $Sp \models \tau_1(\phi)$.
Zu zeigen ist: $Sp \models \tau_2(\Phi_k),\ 1 \leq k \leq n$ und $\tau_2(x) = \tau_1(x)$ für alle $x \in Dom(\tau_1)$.

Wähle dazu in (**) $\tau_2 = \tau \circ \eta_i$ und $k = j$.
Da $Sp \models \tau_1(\phi)$, gilt auch $Sp \models \tau(\eta_i(\Phi_j))$ und $\tau(x) = \tau_1(x)$ für alle $x \in Dom(\tau_1)$ (wegen (*) und der Annahme). Weiterhin gilt $\tau \circ \eta_i(x) = \tau_1(x)$ für alle $x \in Dom(\tau_1)$, da $Dom(\eta_i) \subseteq Y$.

„$\Rightarrow$“:
Annahme: $Sp \models \sigma(\phi)$.
Zu zeigen ist: $Sp \models (\tau(\eta_i(\Phi_j)))$ und $\tau(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$.

Da $Dom(\tau_2) \subset X \cup Y$ und $X \cap Y = \emptyset$, gibt es τ_2' und τ_2'', so daß $\tau_2 = \tau_2' \circ \tau_2''$ und $Dom(\tau_2') \subseteq X$ und $Dom(\tau_2'') \subseteq Y$. Es genügt τ_2 in (**) als irreduzible Grundsubstitution zu betrachten, da R terminierend und hinreichend vollständig ist. Demnach sind auch τ_2' und τ_2'' irreduzible Grundsubstitutionen. Gemäß Definition 52 gibt es dann eine Test-set-Substition δ und eine Grundsubstitution γ, so daß $\tau_2'' = \gamma \circ \delta$. Wähle dann in (*) $\tau = \tau_2' \circ \gamma,\ n_i = \delta$ und $j = k$. Wegen der Annahme und weil $Dom(\gamma) \subseteq Y$ und $Var(\phi) \cap Y = \emptyset$, gilt auch $\tau(x) = \tau_2' \circ \gamma(x) = \tau_2'(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$.

□

Beispiel 13 Die Klauseln 1.1 und 1.2 bzw. 2.1 und 2.2 aus Beispiel 12 werden nun durch die obige Transformation in die folgenden Klauseln überführt:

$< \mathbf{x} \leftarrow \mathbf{0} > : \exists\, y'\ even(0) = true \Rightarrow (< (\mathbf{y} \leftarrow \mathbf{0}) > : 0 = 0 + 0)\ \vee (< (\mathbf{y} \leftarrow \mathbf{s}(\mathbf{y'})) > : 0 = s(y') + s(y'))$ bzw.

$< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{x'}) > : \forall\, x'\ \exists\, y'\ even(s(x')) = true\ \Rightarrow (< (\mathbf{y} \leftarrow \mathbf{0}) > : s(x') = 0 + 0)\ \vee$
$(< (\mathbf{y} \leftarrow \mathbf{s}(\mathbf{y'})) > : s(x') = s(y') + s(y'))$.

Bemerkung: Bei der Umformung einer bereits markierten Spezifikationsklausel (dies kann bei Formeln vorkommen, die durch geschachtelte Induktion bewiesen werden) wird die alte Markierung durch die aktuelle (siehe Definition 63) ersetzt. Die neu umgeformte Formel wird dann als neue zu beweisende Spezifikationsklausel betrachtet, und nur die auf diese Klausel angewandten Test-Set-Substitutionen sind für den Nachweis ihrer Gültigkeit relevant. Bei der Extraktion rekursiver Programme aus einem solchen Beweis bedeutet dieser Schritt die Einführung einer neuen Funktion, die vom initialen zu extrahierenden Programm aufgerufen wird.

Für alle expandierte Instanzen einer Spezifikationsklauseln K gilt die folgende Distributivregel.

Satz 16 *Sei* $K \equiv \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_n$ *eine Spezifikationsklausel und* $\sigma,\ \eta_i,\ 1 \leq i \leq l$ *Substitutionen mit den Eigenschaften:* $Dom(\sigma) \subseteq X$ *und* $Dom(\eta_i) \subseteq Y,\ 1 \leq i \leq l$. *Dann gilt:*

$$\sigma(Norm(\bigvee_{i=1}^{l} \eta_i(C))) = Norm(\bigvee_{i=1}^{l} \eta_i(\sigma(C))).$$

Beweis: Trivial.

8.3 Eine Beweisprozedur f¨ur kanonische erweiterte konstruktive Spezifikationen

Wir führen nun eine Beweisprozedur für den Gültigkeitsnachweis von Spezifikationsklauseln bzgl. kanonischer erweiterter konstruktiver Spezifikationen ein, mit anderen Worten: Mit der Prozedur wird die Gültigkeit einer Existenzformel im initialen Modell einer Menge bedingter Gleichungen bewiesen. Der hier verwendete Formalismus verallgemeinert diejenigen, die in [BAC88], [RED89] und [BR95] bereits eingeführt wurden.

Sei Sp eine KEKS, K eine Spezifikationsklausel, R das zu Sp assoziierte Termersetzungssystem, Φ_2 eine möglicherweise leere Menge bereits bewiesener Lemmata in Sp und $<$ eine stabile auf Grundterme noethersche Ordnung. Wir schreiben $\ll$ für die Multiseterweiterung von $<$. Da $<$

noethersch ist, ist auch $\ll$ noethersch [DM79]. Unsere Beweisprozedur konstruiert und verändert zwei Klauselmengen Kl und H, wobei Kl die zu beweisenden Spezifikationsklauseln enthält und H die Klauseln enthält, deren Instanzen (falls diese bzgl. $\ll$ kleiner als die aktuelle zu beweisende Formel sind) als Induktionshypothesen angewandt werden können. Da das zu Sp assoziierte Termersetzungssystem R und die Menge Φ_2 während eines Beweises unverändert bleiben, werden sie als globale Komponenten betrachtet und kommen deshalb explizit in den Regeln der Prozedur nicht vor.

Um die Regeln des Systems so einfach wie möglich zu halten, werden nur die Regeln markiert, deren Anwendbarkeit von den in den Markierungen enthaltenen Informationen abhängt. Die Regeln unseres Systems sind in vier Gruppen eingeteilt, nämlich:

I. **Regeln für Lemmagenerierung**:

Diese Gruppe besteht aus zwei Regeln, die ausgehend von einer zu beweisenden Spezifikationsklausel neue Lemmata generieren und Induktionsschritte initialisieren.

II. **Regeln für Klauselreduktion**:

Neben der kontextuell geordneten Reduktion von Spezifikationsklauseln (siehe Definition 58) wird hier u.a. die Freiheit der Konstruktoren ausgenutzt, um die in den zu beweisenden Spezifikationsklauseln enthaltenen Gleichungen zu vereinfachen oder zu eliminieren.

III. **Regeln für Klauseleliminierung**:

Mit den Regeln dieser Gruppe werden Tautologien sowie von Induktionshypothesen subsumierte oder redundante Spezifikationsklauseln eliminiert.

IV. **Regeln für Klauselzerlegung**:

Eine zu beweisende Spezifikationsklausel kann unter bestimmten Voraussetzungen in mehrere Teilklauseln aufgeteilt werden. Das ursprünglich komplexere Beweisproblem läßt sich dann durch die Regeln dieser Gruppe auf kleinere Teilprobleme zurückführen.

Im folgenden werden diese vier Gruppen von Regeln vorgestellt und einige Erläuterungen dazu gegeben.

Gruppe I: **Regeln für Lemmagenerierung**

expand_1 $\frac{Kl \cup (\cup_\sigma C_\sigma),\ H \cup \{C\}}{Kl \cup \{C\},\ H}$

Falls $C \equiv < \tau >: \forall X \exists Y \phi \Rightarrow < \gamma_1 >: \Phi_1 \vee \ldots \vee < \gamma_n >: \Phi_n$ und $Var(\phi) \cap Y = \emptyset$, dann $C_\sigma = \{Norm(\bigvee_{i=1}^{l} \eta_i(\sigma(C)))\}$ für alle $\sigma \in Subind(C)$ und $\eta_i \in Subex(C)$, wobei $l = \sharp Subex(C)$.

expand_2 $\frac{Kl \cup (\cup_\sigma C_\sigma),\ H \cup \{C\}}{Kl \cup \{C\},\ H}$

Falls $C \equiv < \tau >: \forall X \exists Y \phi \Rightarrow < \gamma_1 >: \Phi_1 \vee \ldots \vee < \gamma_n >: \Phi_n$ und $Var(\phi) \cap Y = \emptyset$, dann $C_\sigma = \{< \sigma >: \forall X' \exists Y' \sigma(\eta_i(\phi)) \Rightarrow (< \eta_i >: \sigma(\eta_i(\Phi_1))) \vee \ldots \vee (< \eta_i >: \sigma(\eta_i(\Phi_n)))\}$ für alle $\sigma \in Subind(C)$, wobei $\eta_i \in Subex(C)$ für ein $i,\ 1 \leq i \leq \sharp Subex(C)$,
$X' = (X \setminus Dom(\sigma) \cup Var(Img(\sigma))$ und $Y' = (Y \setminus Dom(\eta_i)) \cup Var(Img(\eta_i))$.

Gruppe II: **Regeln für Klauselreduktion**

simplify $\frac{Kl \cup \{C'\},\ H}{Kl \cup \{C\},\ H}$

falls $C \mapsto_{R[\Phi_2]} C'$.

positive decomposition $\frac{Kl \cup \{C'\},\ H}{Kl \cup \{C\},\ H}$,

falls es ein u gibt, so daß $C|_u = k_1(t_1, \ldots, t_n) = k_1(s_1, \ldots, s_n)$, wobei k_1 ein Konstruktor ist und $C' = C[u \leftarrow \bigwedge_{i=1}^{n}(t_i = s_i)]$.

negative decomposition $\frac{Kl \cup (\cup_{i=1}^{n}\{C_i\}),\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv \forall X \exists Y \neg(k_1(s_1,\ldots,s_n) = k_1(t_1,\ldots,t_n)) \wedge \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$, k_1 ein Konstruktor ist und $C_i = \forall X \exists Y \neg(s_i = t_i) \wedge \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$.

delete_trivial_eq $\frac{Kl \cup \{C'\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv \forall X \exists Y \phi \Rightarrow ((s = t) \wedge \psi) \vee \Phi_1 \vee \ldots \vee \Phi_l$, wobei ψ eine nicht leere Konjunktion von Gleichungen ist und,
entweder s und t syntaktisch gleich sind
oder $\phi \equiv s = t \wedge \phi_1$ und,
$C' = \forall X \exists Y \phi \Rightarrow \psi \vee \Phi_1 \vee \ldots \vee \Phi_l$.

delete_redundant_eq_1 $\frac{Kl \cup \{C'\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv < \sigma >: \forall X \exists Y (s = t)^{\epsilon} \wedge \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$ und
entweder $\epsilon = -$, $s \equiv k_1(s_1,\ldots,s_n)$, $t \equiv k_2(t_1,\ldots,t_m)$ und k_1, k_2 verschiedene Konstruktoren sind
oder $\epsilon = -,\ s \in Var(t),\ s \neq t$ und t ein Konstruktorterm ist
oder $\epsilon = +$ und $s,\ t$ syntaktisch gleich sind
oder $\epsilon = +,\ s \in X,\ x \notin Var(t)$ (*)
und $C' = < \alpha \circ \sigma >: \forall X \exists Y \alpha(\phi) \Rightarrow \alpha(\Phi_1 \vee \ldots \vee \Phi_l)$, wobei $\alpha : s \to t$, falls (*) eintritt und die leere Substitution sonst.

delete_redundant_eq_2 $\frac{Kl \cup \{C'\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv \forall X \exists Y \phi \Rightarrow (s = t \wedge \psi) \vee \Phi_1 \vee \ldots \vee \Phi_l$, $\Phi_i,\ 1 \leq i \leq l$ eine nicht-leere Konjunktion von Gleichungen ist, und
entweder $s \in Var(C),\ s \in Var(t),\ s \neq t$ und t ein Konstruktorterm ist
oder $s \equiv k_1(s_1,\ldots,s_n),\ t \equiv k_2(t_1,\ldots,t_m)$ und k_1, k_2 verschiedene Konstruktoren sind
und $C' = \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$.

delete_explicit_eq $\dfrac{Kl \cup \{C'\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv \forall X \exists Y \phi \Rightarrow (< \eta >: y = t \wedge \psi) \vee \Phi_1 \vee \ldots \vee \Phi_l$ und $y \in Y,\ y \notin Var(t),\ y \in Var(\psi)$, ψ nicht leer und $C' = \forall X \exists Y \phi \Rightarrow (< \alpha \circ \eta >: \alpha(\psi)) \vee \Phi_1 \vee \ldots \vee \Phi_l$, wobei $\alpha : y \rightarrow t$.

Gruppe III: **Regeln für Klauseleliminierung**

delete_tautology $\dfrac{Kl,\ H}{Kl \cup \{C\},\ H}$,

falls C eine Tautologie ist.

delete_explicit_clause $\dfrac{Kl,\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv < \sigma >: \forall X \exists Y \phi \Rightarrow < \eta >: (y = t) \vee \Phi_1 \vee \ldots \vee \Phi_l$, $y \in Y$ und $Var(t) \subseteq X$.

delete_redundant_clause $\dfrac{Kl,\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv \forall X \exists Y (s = t)^{\epsilon} \wedge \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$ und
entweder $\epsilon = -$ und $s,\ t$ syntaktisch gleich sind,
oder $\epsilon = +$, $s \equiv k_1(s_1, \ldots, s_n)$, $t \equiv k_2(t_1, \ldots, t_m)$ und k_1, k_2 verschiedene Konstruktoren sind,
oder $\epsilon = +$, $s \in Var(C),\ s \in Var(t),\ s \neq t$ und t ein Konstruktorterm ist,

subsumption_1 $\frac{Kl,\ H}{Kl \cup \{C\},\ H}$,

falls es ein $h \in H$ und eine Substitution τ gibt, so daß C mit τ durch h subsumiert wird: D.h. $C \equiv < \sigma >: \forall X \exists Y \psi \wedge \phi \Rightarrow < \eta_1 >: (\Phi_{11} \vee \ldots \vee \Phi_{n1}) \vee \ldots \vee < \eta_l >: (\Phi_{1l} \vee \ldots \vee \Phi_{nl})$, $\tau(h) \equiv \forall X' \exists Y' \phi \Rightarrow (\Phi_{1i} \vee \ldots \vee \Phi_{ni}) \wedge R,\ i \in \{1, \ldots, l\}$, wobei $X' \subseteq X, Y' \subseteq Y$, $\tau|_{Indvar(h)} \ll \sigma$, R eine Konjunktion von Gleichungen ist und für alle $y \in Varex(h)$, $\tau(y) = \eta_j(y)$ oder $\tau(y) \in Var(\eta_j(y)),\ 1 \leq j \leq l$.

Gruppe IV: **Regeln für Klauselzerlegung**

case_simplify $\frac{Kl \cup C',\ H}{Kl \cup \{C\},\ H}$,

falls $C' = Fallunterscheidung(C)$.

binary_split $\frac{Kl \cup \{C_1,\ C_2\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv \forall X \exists Y\ \phi \Rightarrow (s = t \wedge \psi) \vee R$, $s \neq t$, $(Var(s) \cup Var(t)) \cap Y = \emptyset$, R nicht leer und
$C_1 = \forall X \exists Y\ \phi \wedge (s = t) \Rightarrow (s = t \wedge \psi) \vee R$,
$C_2 = \forall X \exists Y\ \phi \wedge \neg(s = t) \Rightarrow R$.

subsumption_2 $\frac{Kl \cup \{C_1', C_2'\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv <\sigma>: \forall X \exists Y \phi \wedge \phi_1 \Rightarrow (<\eta_1>: P_1) \vee (<\eta_2>: P_2) \vee \Phi$,
es ein $h \in H$ und Substitutionen τ_1 und τ_2 gibt, so daß $h \equiv \phi_2 \Rightarrow P \wedge R_1,\ P_1 \equiv \tau_1(P)$,
$P_2 \equiv \tau_2(P) \wedge R$, $\phi_1 \equiv \tau_1(\phi_2)$, $\tau_1|_{Indvar(h)} \equiv \tau_2|_{Indvar(h)}$ und $\tau_1|_{Indvar(h)} \ll \sigma$,
es eine nicht leere Menge $J = \{y | y \in Varex(h)$ und $\tau_1(y) \neq \eta_1(y)$ oder $\tau_1(y) \notin Var(\eta_1(y))\}$ gibt,
es eine Substitution γ (o.B.d.A die allgemeinste) gibt, so daß $\gamma(\tau_1(y)) = \gamma(\tau_2(y))$ für alle
$y \in Varex(h)$ gilt, wobei $Dom(\gamma) \subseteq Y$ und
$C_1' = <\sigma> : \forall X \exists Y \phi \wedge \phi_1 \wedge (\bigwedge_{y_j \in J} \gamma(\tau_1(y_j)) = \tau_2(y_j)) \Rightarrow <\eta_1>: P_1[\tau_1(y_j) \leftarrow \tau_2(y_j)] \vee <\eta_2>: P_2 \vee \Phi$
$C_2' = <\sigma>: \forall X \exists Y \phi \wedge \phi_1 \wedge \neg(\bigwedge_{y_j \in J}(\gamma(\tau_1(y_j)) = \tau_2(y_j))) \Rightarrow <\eta_1>: P_1 \vee <\eta_2>: P_2 \vee \Phi$

Erläuterungen zu Gruppe I:

expand_1: Diese Regel ist das Herzstück unserer Beweisprozedur: Sie generiert neue Lemmata und initialisiert Induktionsschritte. Für eine zu beweisende Spezifikationsklausel C bildet und markiert sie alle ihrer mit den Test-set-Substitutionen für die Induktionsvariablen und mit den Test-set-Substitutionen für die Existenzvariablen expandierten Instanzen (siehe Definition 63). Danach können die expandierten Instanzen von C entweder durch Vereinfachungsregeln z.B. einen kontextuellen geordneten Termersetzungsschritt oder eine Fallunterscheidung reduziert. Letztere ist nur dann möglich, wenn die Disjunktion der Bedingungen einiger Termersetzungsregeln aus R aus der betrachteten KEKS Sp induktiv folgt. Die so expandierten Spezifikationsklauseln werden dann in die Menge Kl der noch zu beweisenden Klauseln und die initiale Spezifikationsklausel C in die Menge H der Induktionshypothesen aufgenommen. Jede Instanz einer Klausel in H, die bzgl. der vorgegebenen noetherschen Ordnung kleiner als die aktuelle zu beweisende Klausel ist, kann dann später als Induktionshypothese angewandt werden. **expand_1** läßt sich allerdings nicht auf Spezifikationsklauseln anwenden, die Existenzvariable in den Prämissen enthalten. Der Grund dafür liegt darin, daß expandierte Instanzen solcher Klauseln im allgemeinen keine Spezifikationsklauseln mehr sind, da sich ihre Prämisse nicht als Konjunktion von Gleichungen (bzw. negierten Gleichungen) darstellen läßt. In einem solchen Fall wird **expand_2** angewandt.

expand_2: Falls Existenzvariablen in der Prämisse einer Spezifikationsklausel vorkommen, wird

(aus dem oben genannten Grund) für jede Test-set-Substitution für die Induktionsvariablen lediglich eine Test-set-Substitution für die Existenzvariablen betrachtet. Der Übersichtlichkeit halber werden in den Beispielen nur diejenigen Substitutionen angewandt, die zum Erfolg führen. Analog zu **expand_1** generiert **expand_2** neue Lemmata und initialisiert Induktionsschritte.

Erläuterungen zu Gruppe II

simplify vereinfacht eine Klausel C mit Hilfe einer Regel des Termersetzungssystems R oder mit Hilfe eines bereits bewiesenen Lemmas.

positive decomposition: Eine Dekomposition einer Gleichung $e \equiv k(t_1, \ldots, t_n) = k(s_1, \ldots, s_n)$ in einer Spezifikationsklausel C läßt sich immer dann durchführen, wenn k ein Konstruktor ist. In diesem Fall wird die Gleichung e in C mittels eines Dekompositionsschrittes durch die n Gleichungen

$$\begin{aligned} t_1 &= s_1 \\ &\vdots \\ t_1 &= s_n \end{aligned}$$

ersetzt.

negative decomposition: Wie bei der obigen Regel wird hier ein Dekompositionsschritt ausgeführt, falls in der Prämisse der zu beweisenden Spezifikationsklausel ein Ausdruck der Form $\neg(k(t_1, \ldots, t_n) = k(s_1, \ldots, s_n))$ enthalten ist, wobei k ein Konstruktor ist.

delete_trivial_eq: Hier werden triviale Gleichungen in der Konklusion einer Spezifikationsklausel C gelöscht.

delete_redundant_eq_1: Diese Regel ermöglicht, neben der Streichung von Konstruktor-clash-Gleichungen in der Prämisse einer Spezifikationsklausel auch die Eliminierung jener Gleichungen, deren linke (bzw. rechte) Seite eine Variable ist und in deren rechter (bzw. linker) Seite enthalten ist (occur-check). Zudem werden triviale Gleichungen wie etwa $t = t$, in der Prämisse einer Klau-

sel gelöscht.

delete_redundant_eq_2: Analog zu der obigen Regel. Allerdings werden hier redundante Gleichungen in der Konklusion einer Klausel eliminiert.

delete_explicit_eq: Die explizite Gleichung $e \equiv x = t$ in C wird gelöscht, und alle Vorkommen der Variable x in der Teilklausel mit der gleichen Markierung wie e werden durch ihren expliziten Wert t ersetzt.

Erläuterungen zu Gruppe III

delete_tautology: Falls C eine Tautologie ist, wird sie durch diese Regel aus der Menge der zu beweisenden Spezifikationsklauseln entfernt.

delete_explicit_clause: Spezifikationsklauseln in expliziter Form sind trivialerweise wahr, da eine Ersetzung alle Vorkommen von x in ψ durch t zu einer Tautologie führt.

delete_redundant_clause: Redundante Spezifikationsklauseln werden aus der Menge der zu beweisenden Klauseln entfernt.

subsumption_1: Diese Regel entspricht der Anwendung einer Induktionshypothese bei einem traditionellen Beweis durch Induktion. Jede Instanz, etwa $\tau(h)$, einer Klausel h in H, die bzgl. der Multiseterweiterung $\ll$ von $<$ kleiner als die Instanz $\sigma(h)$ ist, darf als Induktionshypothese angewandt werden, vorausgesetzt σ ist die Markierung der aktuellen zu beweisenden Spezifikationsklausel C (also die bei dem Induktionsschritt auf die Induktionsvariablen angewandte Substitution) und h subsumiert die Klausel C mit der Substitution τ. Die Überprüfung der *kleiner*-Beziehung für eine Hypothese $\tau(h)$ und des Induktionsschrittes $\sigma(h)$, durch den C erzeugt wurde, geschieht durch einen Vergleich der Substitutionen τ und σ, allerdings beschränkt auf die Menge $Indvar(h)$ der Induktionsvariablen von h. Die noethersche Ordnung $<$ muß dabei nicht unbedingt mit der Terminierungsordnung des Termersetzungssystems R übereinstimmen. Der Benutzer kann jede beliebige stabile und auf Grundterme noethersche Ordnung vorgeben. Ansonsten wird die Terminierungsordnung des zu Sp assoziierten Termersetzungssystems defaultmäßig vom

System gewählt. Die gewählte Ordnung wird dann der Terminierungsordnung der Funktion entsprechen, die bei der Programmsynthese später generiert wird.

Erläuterungen zu Gruppe IV

case_simplify: Die Klausel C wird mit Hilfe einiger Regeln aus dem Termersetzungssystem R in Teilklauseln aufgeteilt, vorausgesetzt die Disjunktion der Prämissen der Ersetzungsregeln folgt induktiv aus Sp.

binary_split: Hier werden Zusatzforderungen an Terme, die nur allquantifizierte Variablen enthalten, in die Prämisse der Spezifikationsklausel eingefügt. Auf diese Weise kann eine Klausel so modifiziert werden, daß weitere Auswertungen durchgeführt werden können. Die Idee dabei ist, eine Klausel C bzgl. einer Gleichung $s = t$ in zwei Teilklauseln $C_1 \equiv s = t \Rightarrow C$ und $C_2 \equiv \neg(s = t) \Rightarrow C$ aufzuteilen, wobei die Gleichung $s = t$ als Sukzedenzformel in der Konklusion von C enthalten ist. Zur Erinnerung: Falls $C \equiv \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_n$, dann ist ϕ das Antezedenz und jedes $\Phi_i,\ 1 \leq i \leq n$ eine Sukzedenzformel von C. Um zu verhindern, daß die Regel immer wieder angewandt wird, verlangen wir, daß s und t nur allquantifizierte Variablen enthalten und daß R nicht leer ist. Diese Regel kann als Schnittregel angesehen werden.

subsumption_2: In einigen Fällen kann eine Induktionshypothese nur dann auf eine Spezifikationsklausel C angewandt werden, wenn Zusatzbedingungen erfüllt sind. Die Klausel C wird bzgl. dieser Zusatzforderungen in zwei Teilklauseln aufgeteilt. In der ersten wird davon ausgegangen, daß die Zusatzbedingungen gelten, so daß die Induktionshypothese angewandt werden kann. Damit ist auch die Gültigkeit der ersten Teilklausel trivialerweise nachgewiesen. Die zweite Teilklausel erhält man, indem man die Zusatzbedingungen negiert und sie in die Prämisse von C einfügt. Diese Regel ermöglicht es, daß wir später bei der Programmsynthese rekursive Programme generieren können, die rekursive Aufrufe in ihrer Bedingung enthalten. Sie ist mit der Resolutionsregel bei Manna und Waldinder [MAW84] vergleichbar.

8.4 Korrektheit der Beweisprozedur

Im vorangegangenen Abschnitt haben wir ein Verfahren zum Gültigkeitsnachweis von Spezifikationsklauseln im initialen Modell einer KEKS eingeführt. Das Verfahren, dargestellt durch das Inferenzsystem I, konstruiert und verändert zwei Klauselmengen Kl und H, wobei Kl bzw. H

die zu beweisenden Spezifikationsklauseln bzw. die Induktionshypothesen enthalten. Der Übergang von einem Beweiszustand (Kl_1, H_1) zu einem Beweiszustand (Kl_2, H_2) durch die Anwendung einer Regel unseres Inferenzsystems I bezeichnen wir im folgenden als Ableitungsschritt und schreiben:

$$(Kl_1, H_1) \vdash_I (Kl_2, H_2)$$

Statt $\vdash_I$ schreiben wir im folgenden $\vdash$.

Definition 64 (Ableitung) Eine I-Ableitung $(Kl_1, H_1) \vdash (Kl_2, H_2) \vdash \ldots \vdash (Kl_n, H_n) \vdash \ldots$ ist eine möglicherweise unendliche Folge von Ableitungsschritten. Wir schreiben $\vdash^*$ für die reflexiv-transitive Hülle von $\vdash$.

Wir führen nun einige Begriffe ein, die für den Korrektheitsbeweis unseres Beweissystems eine zentrale Rolle spielen werden. Im folgenden steht $GI(K)$ für die Menge aller irreduziblen Grundsubstitutionen, deren Definitionsbereiche in der Menge der allquantifizierten Variablen von K enthalten sind und deren Bildbereiche nur irreduzible Grundterme enthalten: d.h. $GI(K) = \{\theta \mid Img(\theta) \subseteq T_{\Sigma_c}$ und $Dom(\theta) \subseteq Varall(K)\}$.

Definition 65 (Induktiver Beweiszustand) Ein Beweiszustand (Kl, H) heißt induktiv (bzgl. einer KEKS Sp und einer stabilen auf Grundtermen noetherschen Ordnung $<$), falls für alle Spezifikationsklauseln $< \gamma >$: $k \in Kl$ und $\theta \in GI(k)$ folgendes gilt:

$$(\forall \tau \ll \theta \circ \gamma,\ h \in H\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k),$$

wobei $\ll$ die Multiseteerweiterung von $<$ ist.

Die Intention, die dieser Definition zugrunde liegt, ist, die Anwendung jeder Instanz einer Klausel k (während eines Beweises einer anderen Instanz von k) als Induktionshypothese zuzulassen, solange sie (bzgl. $\ll$) kleiner als die zu beweisende Instanz ist. Die Überprüfung dieser Kleiner-Beziehung für eine Induktionshypothese h und eine zu beweisende Spezifikationsklausel geschieht durch den Vergleich der auf h angewandten Substitutionen τ und der Komposition der in der Markierung von k gespeicherten Substitution γ mit der aktuellen auf k angewandten Substitution θ. Während eines Beweises einer Klausel k stellt die in ihrer Markierung gespeicherte Substitution eine obere Schranke für die zulässigen Induktionshypothesen dar. Ähnliche Definitionen für

induktive Beweiszustände findet man u.a. auch in [AM95], [FRA94] und [BEC93].

Eine zentrale Eigenschaft unseres Inferenzsystems I ist seine induktive Monotonie[1], die wie folgt definiert ist.

Definition 66 (Induktive Monotonie) Ein Inferenzsystem I ist induktiv monoton (bzgl. Sp), falls für alle Beweiszustände (Kl_1, H_1) folgendes gilt:

$$(Kl_1, H_1) \text{ ist induktiv, falls } (Kl_1, H_1) \vdash (Kl_2, H_2) \text{ und } (Kl_2, H_2) \text{ induktiv ist.}$$

Hinter den beiden obigen Definitionen steckt folgende Idee: Offensichtlich ist $(Kl, \emptyset)$ genau dann induktiv bzgl. einer KEKS Sp, wenn alle in K enthaltenen Spezifikationsklauseln induktive Folgerungen von Sp sind. Zudem ist $(\emptyset, H)$ trivialerweise induktiv. Um nun zu zeigen, daß $(K, \emptyset)$ induktiv ist, genügt es, eine Ableitung $(K, \emptyset) = (K_0, H_0) \vdash (k_1, H_1) \vdash \ldots \vdash (K_n, H_n)$ zu berechnen, wobei $K_n = \emptyset$. Aus der induktiven Monotonie des Inferenzsystems I (siehe Satz 18) und aus der Tatsache, daß $(\emptyset, H_n)$ induktiv ist, folgt dann, daß $(K, \emptyset)$ induktiv ist. Was wiederum bedeutet, daß alle in K enthaltene Formeln induktive Folgerungen von Sp sind.

Bevor wir zeigen, daß das Inferenzsystem I induktiv monoton ist, beschreiben nun einige Lemmata noch wichtige Eigenschaften induktiver Folgerungen einer KEKS.

Bemerkung: Der Übersichtlichkeit halber werden wir o.B.d.A bei allen weiteren Beweisen in dieser Arbeit nur Spezifikationsklauseln betrachten, die keine Existenzvariablen in ihrer Prämisse enthalten.

Satz 17 *Sei $K \equiv \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$ eine Spezifikationsklausel und Sp eine KEKS. Die folgenden Aussagen sind äquivalent:*

1. $\forall \theta \in GI(K),\ Sp \models_{ind} \theta(K)$.
2. $\forall \sigma,\ Sp \models_{ind} \sigma(K)$, *wobei σ Test-set-Substitutionen sind, für die gilt:* $Dom(\sigma) \subseteq X$.
3. $Sp \models_{ind} K$.

Beweis: Wir zeigen „1. $\Rightarrow$ 2. $\Rightarrow$ 3. $\Rightarrow$ 1.“

(“1. $\Rightarrow$ 2.“):

[1] Der Ausdruck induktive Monotonie bzw. induktiv Monoton wurde hier gew¨ahlt, weil wir keine ad¨aquate Übersetzung des im Englischen verwendeten Begriffs inductively sound, gefunden haben

1. bedeutet, daß es für alle Grundsubstitutionen σ_1 eine Grundsubstitution τ_1 und ein $i,\ 1 \leq i \leq l$ gibt, so daß

$$(Sp \models \sigma_1(\theta(\phi))) \Rightarrow (Sp \models \tau_1(\theta(\Phi_i))),$$

wobei $\tau_1(x) = \sigma_1(x)$ für alle $x \in Dom(\sigma_1)$.

2. bedeutet, daß es für alle Grundsubstitutionen σ_2 eine Grundsubstitution τ_2 und ein $j,\ 1 \leq j \leq l$ gibt, so daß

$$(Sp \models \sigma_2(\sigma(\phi))) \Rightarrow (Sp \models \tau_2(\sigma(\Phi_j))),$$

wobei $\tau_2(x) = \sigma_2(x)$ für alle $x \in Dom(\sigma_2)$.

Annahme: $Sp \models \sigma_2(\sigma(\phi))$.
Zu zeigen ist: Es gibt ein τ_2 und ein $j,\ 1 \leq j \leq l$, so daß $Sp \models \tau_2(\sigma(\Phi_j))$ und $\tau_2(x) = \sigma_2(x)$ für alle $x \in Dom(\sigma_2)$. Da Sp konvergent und hinreichend vollständig ist, genügt es, σ_2 als irreduzible Grundsubstitution zu betrachten. Es gibt dann immer eine irreduzible Grundsubstitution γ, so daß $\gamma(x) = \sigma_2(x)$ für alle $x \in Dom(\sigma_2)$ und $\gamma \circ \sigma$ eine irreduzible Grundsubstitution ist. Aus der Annahme folgt dann, daß $Sp \models \sigma_2(\gamma(\sigma(\phi)))$. Daraus und aus 1. folgt, daß es eine Substitution τ_1 und ein $i,\ 1 \leq i \leq l$ gibt, so daß $Sp \models \tau_1(\gamma(\sigma(\Phi_i)))$ und $\tau_1(x) = \sigma_2(x)$ für alle $x \in Dom(\sigma_2)$. Wähle nun $\tau_2 = \tau_1 \circ \gamma$ und $j = i$. Es gilt: Für alle $x \in Dom(\sigma_2)$ $\tau_2(x) = \sigma_2(x)$, da $\tau_1(x) = \sigma_2(x)$ und $\gamma(x) = \sigma_2(x)$ für alle $x \in Dom(\sigma_2)$

(“2. $\Rightarrow$ 3.“):

3. bedeutet daß es für alle Grundsubstitutionen σ_3 eine Grundsubstitution τ_3 und ein $m,\ 1 \leq m \leq l$ gibt, so daß

$$(Sp \models \sigma_3(\phi)) \Rightarrow (Sp \models \tau_3(\Phi_m)),$$

wobei $\tau_3(x) = \sigma_3(x)$ für alle $x \in Dom(\sigma_3)$.

Annahme: $Sp \models \sigma_3(\phi)$.
Es genügt σ_3 als irreduzible Grundsubstitution zu betrachten, da Sp konvergent und hinreichend vollständig ist. Es gibt dann eine Test-set-Substitution σ_4 und eine irreduzible Grundsubstitution σ_5, so daß $\sigma_3 = \sigma_5 \circ \sigma_4$ (siehe Definition 52). Wegen 2. existiert eine Substitution τ_2 und ein $j,\ 1 \leq j \leq l$, so daß $Sp \models \tau_2(\sigma_4(\Phi_j))$, wobei $\tau_2(x) = \sigma_5(x)$ für alle $x \in Dom(\sigma_5)$. Wähle nun $\tau_3 = \tau_2 \circ \sigma_4$ und $m = j$. Zu zeigen bleibt, daß $\tau_3(x) = \sigma_3(x)$ für alle $x \in \sigma_3(x)$: d.h. $\tau_2(\sigma_4(x)) = \sigma_5(\sigma_4(x))$ für alle $x \in Dom(\sigma_5 \circ \sigma_3)$. Letzteres gilt aber trivialerweise, da (wegen 2.) $\tau_2(x) = \sigma_5(x)$ für alle $x \in Dom(\sigma_5)$.

(“3. $\Rightarrow$ 1.“):

Annahme: $Sp \models \sigma_1(\theta(\phi))$.
Zu zeigen ist: Es gibt eine Grundsubstitution τ_1 und ein $i,\ i \leq i \leq l$, so daß $Sp \models \tau_1(\theta(\Phi_i))$ und $\tau_1(x) = \sigma_1(x)$ für alle $x \in Dom(\sigma_1)$. Wegen der 3. und der Annahme gibt es ein τ_3 und ein $m,\ 1 \leq m \leq l$, so daß $Sp \models \tau_3(\Phi_m)$ und $\tau_3(x) = \sigma_1(\theta(x))$, für alle $x \in Dom(\sigma_1 \circ \theta)$. Wähle $\tau_1 = \sigma_1$ und $i = m$. Daraus folgt sofort die Behauptung.

□

Aus Definition 65 und aus Satz 17 lassen sich nun folgende Eigenschaften für Beweiszustände ableiten.

Korollar 1 Finale Beweiszustände sind induktiv; d.h.: $(\emptyset, H)$ ist induktiv.

Korollar 2 Initiale Beweiszustände sind genau dann induktiv bzgl. Sp, wenn alle ihre zu beweisenden Spezifikationsklauseln induktive Folgerungen von Sp sind; d.h.

$$(Kl, \emptyset) \text{ ist induktiv} \Leftrightarrow Sp \models_{ind} k, \text{ für alle } k \in K.$$

Theorem 1 *(Induktive Monotonie) Sei Sp eine KEKS. Das Inferenzsystem I ist bzgl. Sp induktiv monoton.*

Beweis: Wir führen den Beweis durch eine Fallunterscheidung über alle Regeln des Inferenzsystems I; d.h. für jede Regel r von I wird gezeigt, daß , falls wir durch die Regel r von einem Beweiszustand (Kl_1, H_1) zu einem Beweiszustand (Kl_2, H_2) übergehen und (Kl_2, H_2) induktiv ist, dann auch (Kl_1, H_1) induktiv ist.

Fall 1.) expand_1:

Unter der Voraussetzung, daß $(Kl \cup (\cup_\sigma C_\sigma), H \cup \{C\})$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma, h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma, h \in H.\ Sp \models_{ind} \tau(h))$.
Zu zeigen ist: $Sp \models_{ind} \theta(k)$.
Wir betrachten zwei Fälle:

a) Es gelte: $(\forall \tau \ll \theta \circ \gamma.\ Sp \models_{ind} \tau(C))$.
Aus a) und der Annahme folgt, daß (*) $(\forall \tau \ll \theta \circ \gamma, h \in H \cup \{C\}.\ Sp \models_{ind} \tau(h))$ gilt.
a.1) $k \in Kl$
Aus (*) und der Induktivität von $(Kl \cup (\cup_\sigma C_\sigma), H \cup \{C\})$ folgt, daß $Sp \models_{ind} \theta(k)$

gilt.

a.2) $k \equiv C$

Aus (*) und der Induktivität von $(Kl \cup (\cup_\sigma C_\sigma), H \cup \{C\})$ folgt, daß $Sp \models_{ind} \theta(Norm(\bigvee_{1=l}^{l} \eta_i(\sigma(C))))$ für alle $\sigma \in Subind$ gilt. Nach Satz 16 gilt dann $Sp \models_{ind} Norm(\bigvee_{i=1}^{l} \eta_i(\theta(\sigma(C))))$. Daraus und aus Satz 15 folgt, daß $Sp \models_{ind} \theta(\sigma(C))$ gilt. Da $\theta \in GI(k)$, gibt es eine Test-set-Substitution σ_1 und eine irreduzible Grundsubstitution θ_1, so daß $\theta \equiv \theta_1\sigma_1$. Also gilt insbesondere für σ_1, daß $Sp \models_{ind} \theta_1(\sigma_1(C))$ gilt: D.h. $Sp \models_{ind} \theta(C)$.

b) Es gibt $\tau, \tau \ll \theta \circ \gamma$ und $Sp \not\models_{ind} \tau(C)$.
Betrachte ein minimales τ mit der obigen Eigenschaft: d.h. $\forall \tau_1 \ll \tau.\ Sp \models_{ind} \tau_1(C)$. Aus der Annahme und aus b) folgt, daß $(\forall \tau_1 \ll \tau, h \in H \cup \{C\}.\ Sp \models_{ind} \tau_1(h))$ gilt. Da aber $(Kl \cup (\cup_\sigma C_\sigma), H \cup \{C\})$ induktiv ist, gilt nun insbesondere für τ: $\forall k' \in Kl \cup (\cup_\sigma C_\sigma).\ Sp \models_{ind} \tau(k')$. Analog zu der Überlegung in a.2) folgt aber, daß $Sp \models_{ind} \tau(C)$ gilt, was ein Widerspruch zu der Annahme b) ist.

Fall 2.) expand_2: analog zu **expand_1**.

Fall 3.) simplify:

Unter der Voraussetzung, daß $(Kl \cup \{C'\}, H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$

Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$

Aus $C \mapsto_{R[\Phi_2]} C'$ folgt, daß es eine Regel $\phi \Rightarrow l \rightarrow \in R$ oder ein Lemma $\phi \Rightarrow l = r \in \Phi_2$, eine Substitution α und ein u gibt, so daß $C|_u \equiv \alpha(l)$, $Sp \models \alpha(\phi)$ und $C' \equiv C[\alpha(r)]_u$. Daraus folgt (wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist), daß $Sp \models_{ind} \theta|_{Varall(C')}(C')$ gilt und demzufolge auch $Sp \models_{ind} \theta(C[\alpha(r)]_u) \equiv \theta(C')$, da $Varall(C') \subseteq Dom(\theta)$. Zudem gilt: $Sp \models_{ind} \theta(\alpha(l)) = \theta(\alpha(r))$, da $\phi \Rightarrow l \rightarrow \in R$ bzw. $\phi \Rightarrow l = r \in \Phi_2$. Also gilt auch: $Sp \models_{ind} \theta(C[\alpha(l)]_u) \equiv \theta(C)$.

Fall 4.) positive decomposition:

Unter der Voraussetzung, daß $(Kl \cup \{C'\}, H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Sei nun $C \equiv \forall X \exists Y \phi \Rightarrow \Phi[K_1(s_1, \ldots, s_n) = K_1(t_1, \ldots, t_n)]_u$ (der Fall, bei dem u in ϕ vorkommt ist analog). Aus der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, folgt, daß $Sp \models_{ind} \theta(C')$. D.h. für alle Grundsubstitutionen σ gibt es dann eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\Phi[\bigwedge_{i=1}^{n}(s_i = t_i)]_u)))$ und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Da aus $Sp \models \bigwedge_{i=1}^{n}(s_i = t_i)$ folgt, daß $Sp \models K_1(s_1, \ldots, s_n) = K_1(t_1, \ldots, t_n)$ gilt, gibt es dann für alle Grundsubstitutionen σ eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\Phi[K_1(s_1, \ldots, s_n) = K_1(t_1, \ldots, t_n)]_u)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. D.h. $Sp \models_{ind} \theta(C)$.

Fall 5.) negative decomposition:

Unter der Voraussetzung, daß $(Kl \cup (\cup_{i=1}^{n}\{C_i\}), H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup (\cup_{i=1}^{n}\{C_i\}), H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Aus der Annahme und weil $(Kl \cup (\cup_{i=1}^{n}\{C_i\}), H)$ induktiv ist, folgt, daß für alle i, $1 \leq i \leq n$ $Sp \models_{ind} \theta(C_i)$ gilt, d.h. (*) Für alle Grundsubstitutionen gibt es eine Grundsubstitution η und ein $j,\ 1 \leq j \leq l$, so daß $(Sp \models \neg(\sigma(\theta(s_i = t_i))) \wedge \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\Phi_j)))$ und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Falls nun $Sp \models \neg(\sigma(\theta(K_1(s_1, \ldots, s_n) = K_1(t_1, \ldots, t_n)))) \wedge (\sigma(\theta(\Phi)))$ für ein σ gilt, dann muß es ein $i,\ 1 \leq i \leq n$ geben, so daß $Sp \models \neg(\sigma(\theta(s_i = t_i))) \wedge \sigma(\theta(\phi))$ gilt. Aus (*) folgt dann, daß es ein $j,\ 1 \leq j \leq l$ und

ein η gibt, so daß $Sp \models \eta(\theta(\Phi_j))$ und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Also gilt: $Sp \models_{ind} \theta(C)$.

Fall 6.) delete_trivial_eq:

Unter der Voraussetzung, daß $(Kl \cup \{C'\}, H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta|_{Varall(C')}(C')$ und demnach $Sp \models_{ind} \theta(C')$, da $Varall(C') \subseteq Dom(\theta)$.
D.h. Für alle Grundsubstitutionen σ gibt eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Aber da $Sp \models \eta(\theta(t = t))$ trivialerweise gilt, gibt es auch für alle Grundsubstitution σ eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta((t = t) \wedge \psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ bzw. $(Sp \models \sigma(\theta(s = t \wedge \phi))) \Longrightarrow (Sp \models \eta(\theta((s = t) \wedge \psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Also gilt $Sp \models_{ind} \theta(C)$.

Fall 7.) delete_redundant_eq_1:

Unter der Voraussetzung, daß $(Kl \cup \{C'\}, H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$

Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta|_{Varall(C')}(C')$ und demnach auch $Sp \models_{ind} \theta(C')$, da $Varall(C') \subseteq Dom(\theta)$.

D.h. (*) Für alle Grundsubstitutionen σ gibt eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$.

Betrachte nun $(s = t)^{\epsilon}$:

b.1)$\epsilon = -, t = K_1(t_1, \ldots, t_n),\ s = K_2(s_1, \ldots, s_m)$ und $K_1,\ K_2$ sind zwei verschiedene Konstruktoren.

Wegen der Freiheit der Konstruktoren gilt $Sp \models \neg(K_1(t_1, \ldots, t_n) = K_2(s_1, \ldots, s_m))$ und demnach gilt $Sp \models \sigma(\theta(\neg(K_1(t_1, \ldots, t_n) = K_2(s_1, \ldots, s_m))))$ für alle Grundsubstitutionen σ und θ. Aus (*) folgt nun, daß es für alle Grundsubstitutionen σ eine Grundsubstitution η gibt, so daß $(Sp \models (\sigma(\theta(\neg(K_1(t_1, \ldots, t_n) = K_2(s_1, \ldots, s_m)))) \wedge \sigma(\theta(\phi)))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$.

D.h. $Sp \models_{ind} \theta(C)$ gilt.

b.2)$\epsilon = -,\ s \in Var(t)$ und t ist ein Konstruktorterm.

Da t ein Konstruktorterm ist, s eine Variable, die in t vorkommt und $s \neq t$, gilt $Sp \models \neg(s = t)$ und demnach gilt $Sp \models \sigma(\theta(\neg(s = t)))$ für alle Substitutionen σ und θ. Aus (*) folgt dann, daß es für alle Grundsubstitutionen σ eine Grundsubstitution η gibt, so daß $(Sp \models \sigma(\theta(\neg(s = t))) \wedge \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. D.h. $Sp \models_{ind} \theta(C)$ gilt.

b.3)$\epsilon = +$ und $s,\ t$ sind syntaktisch gleich.

Es gilt $Sp \models s = t$ und demnach $Sp \models \sigma(\theta(s = t))$, da s und t syntaktisch gleich sind. Aus (*) folgt nun, daß es für alle Grundsubstitutionen σ eine Grundsubstitution η gibt, so daß $(Sp \models \sigma(\theta(s = t)) \wedge \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und. $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. D.h. $Sp \models_{ind} \theta(C)$ gilt.

Fall 8.) delete_redundant_eq_2:

Unter der Voraussetzung, daß $(Kl \cup \{C'\}, H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta|_{Varall(C')}(C')$ und demnach $Sp \models_{ind} \theta(C')$, da $Varall(C') \subseteq Dom(\theta)$.
D.h. (*) Für alle Grundsubstitutionen σ gibt eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l)))$ gilt und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$.
Betrachte nun die Terme s und t:

b.1)$s \in Var(t)$, $s \neq t$ und t ist ein Konstruktorterm.
Falls für eine Grundsubstitution σ, $Sp \models \sigma(\theta(\phi))$, dann gibt es nach (*) eine Substitution η, so daß $Sp \models \eta(\theta(\psi \vee \Phi_1 \vee \ldots \vee \Phi_l))$ gilt. Demnach gilt auch $Sp \models \eta(\theta((s = t) \wedge \psi \vee \Phi_1 \vee \ldots \vee \Phi_l))$, da $Sp \not\models \eta(\theta(s = t))$ trivialerweise für alle Substitutionen η und θ gilt. Also gilt $Sp \models \theta(C)$.

b.2)$s \equiv K_1(t_1, \ldots, t_n),\ s = K_2(s_1, \ldots, s_m)$ und $K_1,\ K_2$ sind zwei verschiedene Konstruktoren.
Analog zu b.1)

Fall 9.) delete_explicit_eq:

Unter der Voraussetzung, daß $(Kl \cup \{C'\}, H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >$: $k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.
b) $k \equiv C$
Wegen der Annahme und weil $(Kl \cup \{C'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(C')$.
D.h. (*) Für alle Grundsubstitutionen σ gibt es eine Grundsubstitution η_1 und ein i, $1 \leq i \leq l$, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta_1(\theta(\alpha(\psi))) \vee \eta_1(\theta(\Phi_i))))$ und $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$.
Falls nun $Sp \models \eta_1(\theta(\Phi_i))$ gilt, dann gilt auch $Sp \models \eta_1(\theta((y = t \wedge \psi) \vee \Phi_1 \vee \ldots \vee \Phi_l))$.
Daraus folgt, daß $Sp \models_{ind} \theta(C)$. Falls aber $Sp \models \eta_1(\theta(\alpha(\psi)))$ gilt, dann gilt auch $Sp \models \eta_1(\theta(\alpha(\theta((y = t \wedge \psi)))))$, da $\theta(\alpha(y = t \wedge \psi)) \equiv \theta(\alpha(\theta(y = t \wedge \psi))))$ (weil $Dom(\theta) \cap Dom(\alpha) = \emptyset$). Daraus folgt, daß es für alle Grundsubstitutionen σ eine

Grundsubstitution $\eta_1 \circ \theta \circ \alpha$ gibt, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta_1(\theta(\alpha(\theta((y = t \wedge \psi))))))$ gilt. Daraus folgt, daß $Sp \models_{ind} \theta(C)$ gilt.

Fall 10.) delete_tautology:

Unter der Voraussetzung, daß $(Kl,\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl,\ H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.
b) $k \equiv C$
Da C eine Tautologie ist, ist auch $\theta(C)$ eine Tautologie. Daraus folgt, daß $Sp \models_{ind} \theta(C)$ gilt.

Fall 11.) delete_explicit_clause:

Unter der Voraussetzung, daß $(Kl,\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ und γ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl,\ H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.
b) $k \equiv C$
Für alle σ, so daß $Sp \models \sigma(\phi)$ gilt, wähle eine Grundsubstitution η_1, so daß $\eta_1 \circ \alpha$ eine Grundsubstitution ist, wobei $\alpha : y \leftarrow t$ und $\eta_1(\alpha(x)) = \sigma(x)$ für alle $x\ inDom(\sigma)$. D.h. $Sp \models_{ind} C$ und insbesondere dann für θ gilt nach Satz 17 $Sp \models_{ind} \theta(C)$.

Fall 12.) delete_redundant_clause:

Unter der Voraussetzung, daß $(Kl,\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl,\ H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Betrachte den Ausdruck $(s = t)^{\epsilon}$

b.1) Falls $\epsilon = -,\ s$ und t syntaktisch gleich sind.
Dann gilt $Sp \not\models \neg(s = t)$ und demnach auch $Sp \models_{ind} \forall X' \exists Y \neg(\theta(s) = \theta(t)) \wedge \theta(\phi) \Rightarrow \theta(\phi_1 \vee \ldots \vee \Phi_l)$. D.h. $Sp \models_{ind} \theta(C)$.

b.2) Falls $\epsilon = +,\ s \equiv K_1(s_1, \ldots, s_s),\ t \equiv K_2(t_1, \ldots, t_m)$ und $K_1,\ K_2$ zwei verschiedene Konstruktoren sind.
Analog zu **b.1)**.

b.3) Falls $\epsilon = +,\ s \in Var(t),\ s \neq t$ und t ein Konstruktorterm ist.
Analog zu **b.1)**.

b.4) Falls $\epsilon = +,\ s \in Var(C),\ s \notin Var(t)$ und es ein $i,\ 1 \leq i \leq l$ gibt, so daß $\alpha(\Phi_i)$ eine Tautologie ist, wobei $\alpha : s \to t$.
Für alle Grundsubstitutionen σ, so daß $Sp \models \sigma(s = t) \wedge \phi)$, wähle eine Grundsubstitution η, so daß $\eta \circ \alpha$ eine Grundsubstitution ist und $\sigma(x) = \eta(\alpha(x)) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Da $\alpha(\Phi_i)$ eine Tautologie ist, gibt es für alle Grundsubstitutionen σ eine Grundsubstitution $\eta \circ \alpha$, so daß
$(Sp \models \sigma(s = t \wedge \phi)) \Longrightarrow (Sp \models (\eta(\alpha(\Phi_1 \vee \ldots \vee \Phi_l))))$ und $\eta(\alpha(x)) = \sigma(x)$ für alle $x \in Dom(\sigma)$ gilt. D.h. $Sp \models_{ind} C$ und insbesondere für θ gilt nach Satz 17: $Sp \models_{ind} \theta(C)$.

Fall 13.) subsumption_1:

Unter der Voraussetzung, daß $(Kl,\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k) (\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl,\ H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Es gilt $\tau|_{Indvar(h)} \ll \sigma$ und demnach $\theta(\tau|_{Indvar(h)}) \ll \theta(\sigma)$, da $\ll$ stabil ist.
Wegen der Annahme gilt nun: $Sp \models_{ind} \theta(\tau|_{Indvar(h)}(h))$. Daraus folgt, daß $Sp \models_{ind} \theta(C)$.

Fall 14.) case_simplify:

Unter der Voraussetzung, daß $(Kl \cup C',\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup C', H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.
b) $k \equiv C$
Falls es eine Menge von Regeln $P_i \Rightarrow l_i \rightarrow r_i \in R,\ 1 \leq i \leq n$, Substitutionen σ_i, einige Vorkommen u_i in C gibt, so daß $C|_{u_i} = \sigma_i(l_i)$ und $Sp \models_{ind} \bigvee_{i=1}^{n} \sigma_i(P_i)$ gelten, dann ergibt die Anwendung der Fallunterscheidungsregel auf C die Klauselmenge $C' \equiv \{\sigma_i(P_i) \Rightarrow C[u_i \leftarrow \sigma_i(r_i)]\}$. Es gibt dann ein $v,\ 1 \leq v \leq n$, so daß $Sp \models \theta(\sigma_v(P_v))$. Zudem gilt auch $Sp \models \theta(\sigma_v(l_v)) = \theta(\sigma_v(r_v))$, da $P_v \Rightarrow l_v \rightarrow r_v$ eine Regel aus R ist. Aus der Annahme und aus der Induktivität von $(Kl \cup C', H)$ folgt nun, daß $Sp \models_{ind} \theta(\sigma_v(P_v)) \Rightarrow \theta(C[u_v \leftarrow (\sigma_v(r_v))])$ gilt. Daraus folgt, daß $Sp \models \theta(C[\sigma_v(l_v)]_u)$. D.h. $Sp \models_{ind} \theta(C)$.

Fall 15.) binary_split:

Unter der Voraussetzung, daß $(Kl \cup \{C_1, C_2\},\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >: k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C_1, C_2\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.
b) $k \equiv C$
Es gilt $Sp \models \sigma(\theta(s = t)) \vee \neg(\sigma(\theta(s = t)))$ für alle Grundsubstitutionen σ. Falls nun $Sp \models \sigma(\theta(s = t))$ für eine Grundsubstitution σ gilt, dann gibt es (wegen der Annahme und weil $Sp \models_{ind} \theta(C_1)$) eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(\psi \vee R)))$ gilt. Insbesondere gilt dann auch $Sp \models \eta(\sigma((s = t \wedge \psi) \vee R))$. D.h. $Sp \models_{ind} \theta(C)$ gilt. Sonst falls $Sp \models \neg(\sigma(\theta(s = t)))$ gilt, dann gibt es (wegen der Annahme und weil $Sp \models_{ind} \theta(C_2)$) eine Grundsubstitution η, so daß $(Sp \models \sigma(\theta(\phi))) \Longrightarrow (Sp \models \eta(\theta(R)))$ gilt. Insbesondere gilt dann auch $Sp \models \eta(\theta((s = t \wedge \psi) \vee R))$. D.h. $Sp \models_{ind} \theta(C)$ gilt.

Fall 16.) subsumption_2:

Unter der Voraussetzung, daß $(Kl \cup \{C_1', C_2'\},\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >: k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl \cup \{C_1', C_2'\}, H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.
b) $k \equiv C$
Analog zu **subsumption_1** und **binary_split**.

□

Nun können wir die induktive Korrektheit unserer Beweisprozedur beweisen. Aus der induktiven Korrektheit des Inferenzsystems I folgt dann seine Korrektheit. Dadurch, daß I induktiv monoton ist und, daß finale Beweiszustände (Korollar 1) induktiv sind, wird die Induktivität von initialen Beweiszuständen gewährleistet. Letztere bedeutet, daß alle in in den initialen Beweiszuständen enthaltenen Spezifikationsklauseln induktive Folgerungen der betrachteten KEKS sind (Korollar 2).

Ausgehend von einer Spezifikationsklauselmenge Kl_0 wird eine I-Ableitung $(Kl_0, H_0) \vdash \ldots \vdash (Kl_n, H_n)$ ausgeführt. Erfüllt die Ableitung die folgende Bedingung, so sind alle in Kl_0 enthaltenen Spezifikationsklauseln induktive Folgerungen der betrachteten KEKS Sp.

Theorem 2 *(Induktive Korrektheit) Sei Sp eine KEKS und Kl_0 eine Spezifikationsklauselmenge. Für alle I-Ableitungen $(Kl_0, H_0) \vdash^n (\emptyset, H_{n+1})$ gilt:*
(Kl_0, H_0) ist induktiv und falls, $H_0 = \emptyset$, dann sind alle $k \in Kl_0$ induktive Folgerungen von Sp. Also ist I induktiv korrekt.

Beweis: Durch Induktion über die Länge der I-Ableitungen.

1. $n = 1$
   D.h. $(Kl_0, H_0) \vdash (\emptyset, H_1)$. Aus Korollar 1 und Theorem 1 folgt, daß (Kl_0, H_0) induktiv ist. Falls $H_0 = \emptyset$, dann folgt aus Korollar 2 der zweite Teil der Behauptung.
2. $n > 1$
   Nach der Induktionsannahme gilt die Behauptung für $(Kl_1, H_1) \vdash^{n-1} (\emptyset, H_{n+1})$. D.h. (Kl_1, H_1) ist induktiv. Da I induktiv monoton ist (Theorem 1) und $(Kl_0, H_0) \vdash (Kl_1, H_1)$, ist auch (Kl_0, H_0) induktiv. Falls $H_0 = \emptyset$, gilt wegen Korollar 2 die Behauptung.

□

Vielmehr sind alle in einer I-Ableitung zu beweisende Spezifikationsklauseln induktive Folgerungen von Sp. Diese kann man als Zwischenlemmata betrachten, die ohne Eingriff des Benutzers vom System automatisch generiert werden.

Satz 18 *In einer I-Ableitung $(Kl_0, \emptyset) \vdash^n (\emptyset, H_{n+1})$ sind alle $k \in \cup_{i=0}^{n} K_i$ induktive Folgerungen von Sp.*

Beweis:

1. $i = 1$
   Aus Korollar 2 folgt sofort die Behauptung.
2. $i > 1$.
   Aus der Induktionsannahme folgt, daß alle (*) $k \in \cup_{j=1}^{i-1} Kl_j$ induktive Folgerungen von Sp sind. Da $(\emptyset, H_{n+1})$ induktiv ist und I induktiv monoton ist, muß auch (Kl_i, H_i) induktiv sein. Zudem gilt, daß $H_i \subseteq \cup_{j=1}^{i-1} Kl_j$. Aus (*) folgt nun, daß alle $h \in H_i$ induktive Folgerungen von Sp sind. Daraus und aus der Induktivität von (Kl_i, H_i) folgt, daß alle $k \in Kl_i$ induktive Folgerungen von Sp sind.

□

8.5 Einige Beispiele

Um unsere Beweismethode zu illustrieren, soll nun das Inferenzsystem I an einigen einfachen Beispielen vorgeführt werden. Die ersten drei Beispiele behandeln die natürlichen Zahlen und zeigen, wie einfach, kurz und übersichtlich die Beweise sind, die mit Hilfe unserer Beweismethode geführt werden. Die drei darauf folgenden in diesem Abschnitt haben Sequenzen von natürlichen Zahlen zum Gegenstand und zeigen vor allem noch einmal ausführlich den Ablauf der Beweise, die mit dem Inferenzsystem I generiert werden. Nach der Einführung einer neuen Subsumptionsregel werden noch zwei weitere Beipiele behandelt, die die Bedeutung dieser Regel veranschaulichen. Die meisten in dieser Arbeit ausgewählten Beispiele sind aus der Literatur entnommen und haben in erster Linie einen didaktischen Hintergrund.

Beispiel 14 Die positive Differenz zweier natürlicher Zahlen

Sei Sp die folgende KEKS bestehend aus den Konstruktoren 0 und s und den beiden folgenden Gleichungen, die die Addition beschreiben:

$\Phi = \{x + 0 = x;$
$\quad x + s(y) = s(x + y)\}$

Sei nun K die folgende Spezifikationsklausel zu beweisen:

$$\forall x, y\ \exists z\ x = z + y \vee y = z + x$$

Als Induktionsordnung wählen wir die Anzahlordnung, als Test-set die Menge $\{0, s(x)\}$ und als Induktionsvariablen werden x und y gemäß Definition 61 festgelegt. Demnach sind die folgenden vier Substitutionspaare zu betrachten: $\{(x \leftarrow 0, y \leftarrow 0), (x \leftarrow 0, y \leftarrow s(y')), (x \leftarrow s(x'), y \leftarrow 0), (x \leftarrow s(x'), y \leftarrow s(y'))\}$. Nach der Anwendung von **expand_1** auf K erhalten wir folgende zu beweisende Spezifikationsklauseln:

(1) $< \mathbf{x} \leftarrow \mathbf{0}, \mathbf{y} \leftarrow \mathbf{0} >: \exists z\ 0 = z + 0 \vee 0 = z + 0;$
(2) $< \mathbf{x} \leftarrow \mathbf{0}, \mathbf{y} \leftarrow \mathbf{s(y')} >:\ \forall y'\ \exists z\ 0 = z + s(y') \vee s(y') = z + 0;$
(3) $< \mathbf{x} \leftarrow \mathbf{s(x')}, \mathbf{y} \leftarrow \mathbf{0} >:\ \forall x'\ \exists z\ s(x') = z + 0 \vee 0 = z + s(x');$
(4) $< \mathbf{x} \leftarrow \mathbf{s(x')}, \mathbf{y} \leftarrow \mathbf{s(y')} >:\ \forall x', y'\ \exists z\ s(x') = z + s(y') \vee s(y') = z + s(x').$

Die ersten drei Spezifikationsklauseln werden durch die Anwendung der Regeln **simplify** und **delete_explicit_clause** aus der Menge der zu beweisenden Klauseln entfernt. Die vierte läßt sich zuerst mit **simplify** wie folgt vereinfachen:

$< \mathbf{x} \leftarrow \mathbf{s(x')}, \mathbf{y} \leftarrow \mathbf{s(y')} >:\ \forall x', y'\ \exists z\ s(x') = s(z + y') \vee s(y') = s(z + x').$

Danach erhalten wir durch eine zweifache Anwendung der Regel **positive decomposition**:

$< \mathbf{x} \leftarrow \mathbf{s(x')}, \mathbf{y} \leftarrow \mathbf{s(y')} >:\ \forall x', y'\ \exists z\ x' = z + y' \vee y' = z + x'.$

Schließlich läßt sich die obige Klausel durch Anwendung der Regel **subsumption_1** aus der Menge der zu beweisenden Spezifikationsklauseln entfernen, da $(x', y') \ll (s(x'), s(y'))$ gilt. Nach Theorem 2 ist dann K eine induktive Folgerung von Sp.

□

Das Beispiel, das wir jetzt betrachten werden, ist eine Variante von der bereits im Beispiel 12 eingeführten KEKS. Es soll zeigen, wie sich eine adäquate Wahl eines Test-sets auf den Beweisablauf auswirkt. Im Gegensatz zu Beispiel 12 wird diesmal nur eine Schleife benötigt, um den Beweis abzuschließen.

Beispiel 15 Division einer geraden Zahl durch 2

Sei Sp_1 die KEKS aus dem vorangegangenen Beispiel, erweitert um die folgenden Konstruktoren T, F der Sorte $Bool$, die folgenden bedingten Gleichungen und die Kommutativität der Addition als zusätzliches Lemma:

$\Phi_1 = \{\ ev(0) = T;$
$ev(s(0)) = F;$
$ev(s(s(x))) = ev(x);$
$ev(x) = T \Rightarrow odd(x) = F;$
$ev(x) = F \Rightarrow odd(x) = T\}$

$\Phi_2 = \{x + y = y + x\}$.

Sei nun die folgende Spezifikationsklausel K zu beweisen:

$$\forall x\ \exists y\ ev(x) = T \Rightarrow x = y + y$$

- Gemäß Definition 61 bzw. 62 ist x als Induktionsvariable und y als zu instanziierende Existenzvariable zu wählen.
- Als noethersche Ordnung wählen wir die lexikographische Pfadordnung (von links nach rechts) mit folgender Präzedenz $0 < s < F < T < ev < odd$.
- Gemäß Satz 9 bzw. 11 wird $T = \{0, s(0), s(s(x)), T, F\}$ als Test-set festgelegt.

Auf K wird nun **expand_1** angewandt und wir erhalten die folgenden zu beweisenden Spezifikationsklauseln:

$< \mathbf{x} \leftarrow \mathbf{0} > : \exists y'\ ev(0) = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : 0 = 0 + 0 \vee < \mathbf{y} \leftarrow \mathbf{s(0)} > : 0 = s(0) + s(0) \vee$
$< \mathbf{y} \leftarrow \mathbf{s(s(y')}) > : 0 = s(s(y')) + s(s(y'))$ **(1)**

$< \mathbf{x} \leftarrow \mathbf{s(0)} > : \exists y'\ ev(s(0)) = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(0) = 0 + 0 \vee$
$< \mathbf{y} \leftarrow \mathbf{s(0)} > : s(0) = s(0) + s(0) \vee < \mathbf{y} \leftarrow \mathbf{s(s(y')}) > : s(0) = s(s(y')) + s(s(y'))$ **(2)**

$< \mathbf{x} \leftarrow \mathbf{s(s(x')}) > : \forall x'\ \exists y'\ ev(s(s(x')) = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(s(x')) = 0 + 0 \vee$
$< \mathbf{y} \leftarrow \mathbf{s(0)} > : s(s(x')) = s(0) + s(0) \vee < \mathbf{y} \leftarrow \mathbf{s(s(y')}) > : s(s(x')) = s(s(y')) + s(s(y'))$ **(3)**

(1) läßt sich mit den Regeln **simplify** und **delete_tautology** aus der Menge der zu beweisenden Klauseln entfernen.

(2) wird durch die Anwendung der Regeln **simplify** und **delete_redundant_clause** weggelöscht.

(3) ergibt die folgende Klausel nach dreimaliger Anwendung der Regel **simplify**:

3.1 $< \mathbf{x} \leftarrow \mathbf{s(s(x')}) > : \forall x' \exists y'\ ev(x') = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(s(x')) = 0 \vee < \mathbf{y} \leftarrow \mathbf{s(0)} > :$
$s(s(x')) = s(0) + s(0) \vee < \mathbf{y} \leftarrow \mathbf{s(s(y')}) > : s(s(x')) = s(s(s(y')) + s(y'))$.

simplify läßt sich auf 3.1 noch zweimal mal anwenden: zuerst weil $s(y') + s(s(y')) <_{lpo} s(s(y') + s(y')$ gilt, und dann durch die zweite Gleichung der Addition. Wir erhalten:

3.2 $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{x}')) > : \forall x' \exists y'\ ev(x') = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(s(x')) = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{0}) > : s(s(x')) = s(0) + s(0) \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{y}')) > : s(s(x')) = s(s(s(y') + s(y')))$.

Eine zweimalige Anwendung von **positive decomposition** auf 3.2 ergibt dann:

3.3 $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{x}')) > : \forall x' \exists y'\ ev(x') = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(s(x')) = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{0}) > : s(s(x')) = s(0) + s(0) \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{y}')) > : x' = s(y') + s(y')$.

3.3 läßt sich mit der Regel **subsumption_1** aus der Menge der zu beweisenden Spezifikationsklauseln entfernen, da $x <_{lpo} s(s(x'))$ gilt.

Nach Theorem 2 ist demnach K eine induktive Folgerung von Sp_1.

□

Der Einfachheit halber zeigen wir die nächsten Beispiele ohne Markierungen, die während des Gültigkeitsnachweises einer Spezifikationsklausel im initialen Modell einer KEKS lediglich beim Versuch, eine Induktionshypothese anzuwenden, eine wesentliche Rolle spielen. Um aber später Programme aus einem Beweis extrahieren zu können, müssen alle während des Beweises relevanten angewandten Substitutionen protokolliert werden.

Beispiel 16 Division einer natürlichen Zahl durch 2

Das folgende Beispiel ist eine Verallgemeinerung des vorangegangenen Beispiels. Es soll aber zeigen, daß unsere Beweismethode es ermöglicht, Beweise durch gegenseitige Induktion automatisch zu führen. Dazu betrachten wir die folgende KEKS $Komnat$ bestehend aus den Konstruktoren 0 und s und den folgenden Gleichungsmengen Φ_1 und Φ_2:

$\Phi = \{x + 0 = x;$
$\quad x + s(y) = s(x + y)\}$

$\Phi_2 = \{x + y = y + x\}$.

Sei nun K die folgende Spezifikationsklausel zu beweisen:

$$\forall x\ \exists y\ x = y + y \vee x = s(y + y)$$

Als Induktionsordnung wählen wir die lexikographische Pfadordnung (von links nach rechts) mit der Präzedenz $0 < s < +$. Die Menge $\{0, s(x)\}$ wird als Test-set und x (bzw. y) als Induktionsvariable (bzw. zu instanziierende Existenzvariable) festgelegt.

Nach der Anwendung von **expand_1** auf K sind folgende Spezifikationsklauseln zu beweisen:

$\exists y'\ (0 = 0 + 0 \vee 0 = s(0 + 0)) \vee (0 = s(y') + s(y') \vee 0 = s(s(y') + s(y')))$ **(1)**

$\forall x'\ \exists y'\ (s(x') = 0 + 0 \vee s(x') = s(0 + 0)) \vee (s(x') = s(y') + s(y') \vee s(x') = s(s(y') + s(y')))$ **(2)**

(1) ist eine Tautologie (nach Anwendung von **simplify**) und kann mit der Regel **delete_tautology** aus der Menge der zu beweisenden Klauseln entfernt werden.

(2) wird mit den Regeln **simplify**, **delete_redundant_eq_2** und **positive decomposition** zu der folgenden Klausel vereinfacht:

$\forall x'\ \exists y'\ x' = 0 \vee (x' = s(y') + y' \vee x' = s(y') + s(y'))$, **(2.1)**

die mit der Regel **binary_split** in die folgenden Klauseln überführt wird:

$\forall x'\ \exists y'\ x' = 0 \Rightarrow x' = 0 \vee (x' = s(y') + y' \vee x' = s(y') + s(y'))$ **(2.1.1)**

$\forall x'\ \exists y'\ \neg(x' = 0) \Rightarrow x' = s(y') + y' \vee x' = s(y') + s(y')$ **(2.1.2)**

(2.1.1) ist eine Tautologie und wird mit der Regel **delete_tautology** aus der Menge der zu beweisenden Klauseln entfernt.

Auf **(2.1.2)** wird wieder **expand_1** angewandt, und wir erhalten die beiden folgenden zu beweisenden Spezifikationsklauseln:

$\exists y''\ \neg(0 = 0) \Rightarrow (0 = s(0) \vee 0 = s(0) + s(0)) \vee (0 = s(s(y'')) + s(y'') \vee$
$0 = s(s(y'')) + s(s(y')))$ **(2.1.2.1)**

$\forall x''\ \exists y''\ \neg(s(x'') = 0) \Rightarrow (s(x'') = s(0) \vee s(x'') = s(0) + s(0)) \vee (s(x'') = s(s(y'')) + s(y'') \vee$
$s(x'') = s(s(y'')) + s(s(y')))$ **(2.1.2.2)**

Die erste der beiden obigen Klauseln kann mit **delete_redundant_clause** entfernt werden, die zweite läßt sich mit **simplify**, **delete_redundant_eq_1** und **positive decomposition** wie folgt vereinfachen:

$\forall x''\ \exists y''\ (x'' = 0 \vee x'' = s(0)) \vee (s(x'') = s(s(y'')) + s(y'') \vee x'' = s(s(y'')) + s(y''))$,

die wegen der Kommutativität der Addition und $s(y'') + s(s(y'')) \ll_{lpo} s(s(y'')) + s(y''))$ mit **simplify** wie folgt transformiert werden kann:

$\forall x''\ \exists y''\ (x'' = 0 \vee x'' = s(0)) \vee (s(x'') = s(y'') + s(s(y'')) \vee x'' = s(y'') + s(s(y')))$.

Die obige Klausel läßt sich dann mit den Regeln **simplify** und **positive decomposition** in die folgende Spezifikationsklausel umwandeln:

$\forall x''\ \exists y''\ (x'' = 0 \vee x'' = s(0)) \vee (x'' = s(y'') + s(y'') \vee x'' = s(s(y'') + s(y'')))$,

die mit **subsumption_1** aus der Menge der zu beweisenden Spezifikationsklauseln entfernt werden

kann, da sie von einer bzgl. $<_{lpo}$ kleineren Instanz von **(2)** subsumiert wird.

Nach Theorem 1 ist damit die Gültigkeit von K im initialen Modell von $Komnat$ bewiesen.

Beispiel 17 Zerlegung einer Liste: Variante I

Wir zeigen, daß jede nicht leere Liste in zwei Listen zerlegt werden kann, wobei die erste Liste aus allen Elementen der initialen Liste außer des letzten und die zweite nur aus dem letzten besteht. Die später aus diesem Beweis extrahierten Programme entsprechen den Funktionen *butlast* und *last* in Common Lisp.

Sei Sp_2 die folgende KEKS bestehend aus den Konstruktoren nil und $cons$ und der Funktion app, die die Konkatenation zweier Listen beschreibt:

$$\begin{aligned}\Phi =&\{app(nil, l) = l,\\ &app(cons(a, m), l) = cons(a, app(m, l))\}\end{aligned}$$

Sei nun K die folgende Spezifikationsklausel zu beweisen:

$$\forall l_1 : list\ \exists l_2 : list, x : el\ \neg(l_1 = nil) \Rightarrow l_1 = app(l_2, cons(x, nil)).$$

Als Induktionsordnung wählen wir die rekursive Pfadordnung mit der Präzedenz $nil < cons < app$, als Test-set die Menge $\{nil, cons(a_1, m_1)\}$ und als Induktionsvariable bzw. zu instanziierende Existenzvariable die Liste l_1 bzw. l_2. Nach der Anwendung von **expand_1** auf K erhalten wir folgende zu beweisende Spezifikationsklauseln:

$\exists m_2 : list, x, a_2 : el\ \neg(nil = nil) \Rightarrow nil = app(nil, cons(x, nil)) \vee$
$nil = app(cons(a_2, m_2), cons(x, nil))$ **(1)**;

$\forall m_1 : list, a_1 : el\ \exists m_2 : list, x, a_2 : el\ \neg(cons(a_1, m_1) = nil) \Rightarrow$
$cons(a_1, m_1) = app(nil, cons(x, nil)) \vee cons(a_1, m_1) = app(cons(a_2, m_2), cons(x, nil))$ **(2)**.

(1) läßt sich mit **simplify** und **delete_redundant_clause** aus der Menge der zu beweisenden Klauseln entfernen. Die Spezifikationsklausel **(2)** kann man mit den Regeln **delete_redundant_eq_1** und **simplify** wie folgt vereinfachen:

$\forall m_1 : list, a_1 : el\ \exists m_2 : list, x, a_2 : el\ cons(a_1, m_1) = cons(x, nil) \vee cons(a_1, m_1) =$
$cons(a_2, app(m_2, cons(x, xil)))$ **(2.1)**

Nach zweimaliger Anwendung der Regel **positive decomposition**, gefolgt von einer Anwendung der Regel, **binary_split** erhalten wir nun die beiden folgenden Klauseln:

$\forall m_1 : list, a_1 : el\ \exists\, m_2 : list, x, a_2 : el\ (m_1 = nil) \Rightarrow (a_1 = x \wedge m_1 = nil) \vee (a_1 = a_2 \wedge m_1 = app(m_2, cons(x, nil)))$ **(2.1.1)**

$\forall m_1 : list, a_1 : el\ \exists\, m_2 : list, x, a_2 : el\ \neg(m_1 = nil) \Rightarrow a_1 = a_2 \wedge m_1 = app(m_2, cons(x, nil))$ **(2.1.2)**

Schließlich läßt sich **(2.2.1)** mit den Regeln **delete_explicit_eq** und **delete_tautology** aus der Menge der zu beweisenden Klauseln entfernen. Die Spezifikationsklausel **(2.1.2)** kann man mit der Regel **delete_explicit_eq** zunächst, wie folgt, vereinfachen:

$\forall m_1 : list\ \exists x : el\ \neg(m_1 = nil) \Rightarrow m_1 = app(m_2, cons(x, nil))$

Die obige Spezifikationsklausel ist eine Instanz von K und kann mit der Regel **subsumption_1** entfernt werden, da $m_1 <_{rpo} cons(a_1, m_1)$ trivialerweise gilt. Damit ist nach Theorem 2 die Gültigkeit der Spezifikationsklausel K im initialen Modell von Sp_2 bewiesen.

□

Das nächste Beispiel stellt einen Gültigkeitsnachweis einer Spezifikationsklausel dar, die besagt, daß man jede nicht leere Liste in zwei kleinere Listen aufteilen kann, wobei die Länge der ersten kleiner oder gleich der der ursprünglichen Liste ist. Mit anderen Worten wird die initiale Liste der Länge l in ihr Präfix, dessen Länge etwa n, vorgegeben ist und in ihr Suffix, dessen Länge $l - n$ ist, zerlegt.

Beispiel 18 Zerlegung einer Liste: Variante II

Sei Sp_3 die folgende KEKS, bestehend aus den Konstruktoren $0, s, nil, cons, T, F$ und den folgenden Funktionen:

$$\begin{aligned}\Phi = \{&x + 0 = x;\\ &x + s(y) = s(x + y);\\ &0 \leq x = T;\\ &s(x) \leq 0 = F;\\ &s(x) \leq s(y) = x \leq y;\\ &app(nil, l) = l;\\ &app(cons(a, m), l) = cons(a, app(m, l));\\ &length(nil) = 0;\\ &length(cons(a, m)) = s(length(m))\ \}\end{aligned}$$

Sei K die folgende Spezifikation zu beweisen:

$$\forall l_1 : list, n : nat\ \exists l_2, l_3 : list\ n \leq length(l_1) \Rightarrow l_1 = app(l_2, l_3) \wedge length(l_2) = n$$

Als Induktionsordnung wählen wir die lexikographische Pfadordnung mit der Präzedenz $0 < s < nil < cons < T < F < \leq < app < length$, als Test-set die Menge $\{0, s(x), nil, cons(a, m)\}$ und als Induktionsvariablen bzw. zu instanziierende Existenzvariablen l_1 und n bzw. l_2. Nach der Anwendung der Regel **expand_1** auf K erhalten wir folgende zu beweisende Spezifikationsklauseln:

$\exists m_2, l_3 : list, a_2 : el\ 0 \leq length(nil) \Rightarrow (nil = app(nil, l_3) \wedge length(nil) = 0) \vee (nil = app(cons(a_2, m_2), l_3)) \wedge length(cons(a_2, m_2)) = 0)$ **(1)**

$\forall n' : nat\ \exists m_2, l_3 : list, a_2 : el\ s(n') \leq length(nil) \Rightarrow (nil = app(nil, l_3) \wedge length(nil) = s(n')) \vee (nil = app(cons(a_2, m_2), l_3) \wedge length(cons(a_2, m_2)) = s(n'))$ **(2)**

$\forall m_1 : list, a_1 : el, n' : nat\ \exists m_2, l_3 : list, a_2 : el\ 0 \leq length(cons(a_1, m_1)) \Rightarrow$
$(cons(a_1, m_1) = app(nil, l_3) \wedge length(nil) = 0) \vee (cons(a_1, m_1) = app(cons(a_2, m_2), l_3) \wedge$
$length(cons(a_2, m_2)) = 0)$ **(3)**

$\forall m_1 : list, a_1 : el, n' : nat\ \exists m_2, l_3 : list, a_2 : el\ s(n') \leq length(cons(a_1, m_1)) \Rightarrow$
$(cons(a_1, m_1) = app(nil, l_3) \wedge length(nil) = s(n')) \vee (cons(a_1, m_1) = app(cons(a_2, m_2), l_3) \wedge$
$length(cons(a_2, m_2)) = s(n'))$ **(4)**

(1) kann man nach mehrmaliger Anwendung der Regeln **simplify**, **delete_redundant_eq_2** und **delete_explicit_clause** aus der Menge der zu beweisenden Klauseln entfernen:

(2) ist eine redundante Spezifikationsklausel, da ihre Prämisse einen Widerspruch enthält, und kann mit der Regel **delete_redundant_clause** entfernt werden.

(3) läßt sich mit den Regeln **simplify**, **delete_redundant_eq_1** und **delete_redundant_eq_2**, wie folgt, vereinfachen:

(3.1) $\forall m_1 : list, a_1 : el\ \exists l_3\ cons(a_1, m_1) = l_3$,

die mit der Regel **delete_explicit_clause** aus der Menge der zu beweisenden Klauseln entfernt werden kann.

(4) kann man nach Anwendung der Regeln **simplify**, **delete_redundant_eq_2** und **positive decomposition** in folgende Klausel überführen:

$\forall m_1 : list, a_1 : el, n' : nat\ \exists m_2, l_3 : list, a_2 : el\ n' \leq length(m_1) \Rightarrow a_1 = a_2 \wedge m_1 = app(m_2, l_3) \wedge length(m_2) = n'$

Die obige Klausel läßt sich weiter mit der Regel **delete_explicit_eq** zu der folgenden Spezifikationsklausel vereinfachen:

$\forall m_1 : list, n' : nat\ \exists m_2, l_3 : list\ n' \leq length(m_1) \Rightarrow m_1 = app(m_2, l_3) \wedge length(m_2) = n'$,

welche eine Instanz der ursprünglichen Spezifikationsklausel **(4)** ist und kann mit der Regel **subsumption_1** aus der Menge der zu beweisenden Klauseln entfernt werden, da $m_1 >_{lpo} cons(a_1, m_1)$

gilt. Nach Theorem 2 ist dann K eine induktive Folgerung von Sp_3.

□

Bevor wir weitere Beispiele vorstellen, führen wir eine zusätzliche Variante der Subsumptionsregel ein. Die Subsumptionsregel, wie sie bereits vorgestellt worden ist, kann als ein syntaktisch leicht erkennbarer Spezialfall der Implikation angesehen werden. Sie läßt sich etwas erweitern, indem man die subsumierte und die subsumierende Klausel nicht nur auf syntaktische Gleichheit sondern auf Implikation in der betrachteten Theorie überprüft. Wir werden allerdings in der neuen Variante nur bestimmte Spezifikationsklauseln in Betracht ziehen. Der Grund dafür ist, daß wir die Gültigkeitsüberprüfung der durch die Implikation gebildeten Klausel von unserem System weiterhin durchführen lassen möchten. Mit anderen Worten: wir werden sicherstellen, daß die neu gebildete Klausel wieder eine Spezifikationsklausel ist. Auf diese Einschränkung könnte man verzichten, würde man die Gültigkeit der Implikation durch ein anderes Beweissystem überprüfen lassen. Aufgrund dieser Einschränkung ist die neue Subsumptionsregel keine Verallgemeinerung sondern eine Ergänzung der Regel **subsumption_1**. Sie entspricht im Gegensatz dazu der Anwendung der Induktionshypothese bei einem klassischen Beweis durch (implizite) Induktion.

subsumption_3 $\dfrac{Kl \cup \{C_1\},\ H}{Kl \cup \{C\},\ H}$,

falls $C \equiv < \sigma >: \forall X \exists Y \phi_1 \wedge \phi_2 \Rightarrow < \eta_1 >: \Phi_1 \vee \ldots \vee < \eta_i >: (R_1 \wedge R_2) \vee \ldots \vee < \eta_n >: \Phi_n$
und es eine Substitution τ und ein $h \in H$ gibt, so daß $\tau(h) \equiv \forall X' \exists Y' \phi_1 \Rightarrow R_1 \wedge R_3$, wobei $R_1,\ R_2,\ R_3$ nicht leere Konjunktionen von Gleichungen sind, $X' \subseteq X, Y' \subseteq Y, \tau|_{Indvar(h)} \ll \sigma$, für alle $y \in Varex(h), \tau(y) = \eta_i(y)$ oder $\tau(y) \in Var(\eta_i(y))$ und
$C_1 \equiv < \sigma >: \forall X \exists Y \phi_1 \wedge \phi_2 \wedge R_3 \Rightarrow < \eta_1 >: \Phi_1 \vee \ldots \vee < \eta_i >: R_2 \vee \ldots \vee < \eta_n >: \Phi_n$.

Die Induktionsannahme $\tau(h)$ in der obigen Regel impliziert die zu beweisende Spezifikationsklausel C in einer KEKS Sp, falls die Klausel C_1 eine induktive Folgerung von Sp ist. Dabei muß die Sukzedenzformel Φ von $\tau(h)$ eine Konjunktion von Gleichungen sein, damit die Klausel C_1 vom System behandelt werden kann. Um die Anwendbarkeit der Regel einzuschränken, wird einerseits verlangt, daß mindestens eine Gleichung sowohl in der zu beweisenden Spezifikationsklausel C als auch in der betrachteten Induktionsannahme vorkommt. Andererseits muß die Prämisse der gewählten Induktionsannahme in der Prämisse der zu beweisenden Klausel C enthalten sein. Die neu gebildete Klausel C_1 wird somit automatisch vom System generiert und kann als neues Lemma betrachtet werden, das für den weiteren Verlauf eines Beweises benötigt wird.

Wir müssen nun zeigen, daß die neue Regel die induktive Monotonie des Inferenzsystems I nicht zerstört, d.h. Theorem 1 trotz Hinzunahme der neuen Regel in I weiterhin gilt.

Beweis: Unter der Voraussetzung, daß $(Kl \cup \{C_1\},\ H)$ induktiv ist, zeigen wir, daß $(Kl \cup \{C\}, H)$ induktiv ist (d.h. $\forall < \gamma >$: $k \in Kl \cup \{C\}, \theta \in GI(k)(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h)) \Rightarrow Sp \models_{ind} \theta(k)$).

Annahme: Für ein beliebiges $< \gamma >$: $k \in Kl \cup \{C\}$ und $\theta \in GI(k)$ gelte: $(\forall \tau \ll \theta \circ \gamma,\ h \in H.\ Sp \models_{ind} \tau(h))$.

a) $k \in Kl$
Wegen der Annahme und weil $(Kl,\ H)$ induktiv ist, gilt dann: $Sp \models_{ind} \theta(k)$.

b) $k \equiv C$
Es gilt $\tau|_{Indvar(h)} \ll \sigma$ und demnach $\theta(\tau|_{Indvar(h)}) \ll \theta(\sigma)$, da $\ll$ stabil ist. Wegen der Annahme gilt aber auch $Sp \models_{ind} \theta(\tau|_{Indvar(h)}(h))$ (*). Falls es nun eine Grundsubstitution σ_1 gibt, so daß $Sp \models \sigma_1(\theta(\phi_1 \wedge \phi_2))$ gilt, dann gibt es wegen (*) eine Grundsubstitution η, so daß $Sp \models \eta(\theta(R_1 \wedge R_3))$ (**) und $\eta(x) = \sigma_1(x)$ für alle $x \in Dom(\sigma_1)$ gilt. Wegen der Annahme und der Induktivität von $(Kl \cup \{C_1\})$ gilt aber auch $Sp \models_{ind} \theta(C_1)$. Also gibt es (wegen (**)) für η eine Grundsubstitution η_1 (weil $Sp \models \eta(\theta(\phi_1 \wedge \phi_2 \wedge R_3))$), so daß $Sp \models \eta_1(\theta(\Phi_i))$ für ein $i,\ 1 \leq i \leq n$ oder $Sp \models \eta_1(\theta(R_1 \wedge R_2))$ und $\eta_1(x) = \eta(x)$ für alle $x \in Dom(\eta)$. Da nun $\eta(x) = \sigma_1(x)$ für alle $x \in Dom(\sigma_1)$ gilt, ist auch $\eta_1(x) = \sigma_1(x)$ für alle $x \in Dom(\sigma_1)$. Für alle Grundsubstitutionen σ_1, für die $Sp \models \sigma_1(\theta(\phi_1 \wedge \phi_2))$ gilt, gibt es dann eine Grundsubstitution η_1, so daß $Sp \models \eta_1(\theta(\Phi_i))$, $i \in \{1, \ldots, n\}$ oder $Sp \models \eta_1(\theta(R_1 \wedge R_2))$ und $\eta_1(x) = \sigma_1(x)$ für alle $x \in Dom(\sigma_1)$, d.h. $Sp \models_{ind} \theta(C_1)$.

□

Die nächsten zwei Beispiele sollen die Bedeutung der neu eingeführten Subsumptionsregel (**subsumption_3**) illustrieren. Das erste behandelt die euklidische Division, während das zweite die Wurzelziehung natürlicher Zahlen zum Gegenstand hat.

Der Übersichtlichkeit halber werden in den nächsten Beweisen nur die Pfade betrachtet, die zum Erfolg führen.

Beispiel 19 Euklidische Division

Gegeben sei die folgende KEKS $Sp_4 = (\Sigma, \Phi \cup \Phi_2, \Omega_c)$, bestehend aus den Konstruktoren $0,\ s$ für nat, T, F für $Bool$, den Funktionen:

$\Phi = \{\ x + 0 = x;$
$\quad x + s(y) = s(x + y);$
$\quad x * 0 = 0;$

$x * s(y) = x * y + x;$
$x < 0 = F;$
$0 < s(x) = T;$
$s(x) < s(y) = x < y\}$

und der folgenden Gleichung, die die Kommutativität der Addition ausdrückt:

$\Phi_2 = \{x + y = y + x\}$.

Wir zeigen, daß die folgende Spezifikationsklausel K

$$\forall x, y\ \exists q, r\ \neg(y = 0) \Rightarrow x = y * q + r\ \wedge\ r < y = T$$

eine induktive Folgerung von Sp_4 ist.

Dazu wählen wir die lexikographische Ordnung (von links nach rechts) mit der Präzedenz $0 < s < (<)\ < + < *$ als Induktionsordnung, die Menge $\{0, s(x)\}$ als Test-set und x, y (bzw. q, r) als Induktionsvariablen (bzw. zu instanziierende Existenzvariablen).

Folgende Spezifikationsklauseln werden durch die Anwendung von der Induktionsregel **expand_1** erzeugt:

1. $\forall y'\ \neg(s(y') = 0) \Rightarrow 0 = s(y') * 0 + 0 \wedge 0 < s(y') = T$

2. $\forall x', y'\ \exists q', q'', r', r''\ \neg(s(y') = 0) \Rightarrow (s(x') = s(y') * 0 + s(r') \wedge s(r') < s(y') = T) \vee$
   $(s(x') = s(y') * s(q') + 0\ \wedge\ 0 < s(y') = T) \vee$
   $(s(x') = s(y') * s(q'') + s(r'') \wedge s(r'') < s(y') = T)$

3. $\exists q', q'', r', r''\neg(0 = 0) \Rightarrow (0 = 0 * 0 + 0) \vee (0 = 0 * 0 + s(r')) \vee (0 = 0 * s(q') + 0) \vee$
   $(0 = 0 * s(q'') + s(r''))$

4. $\forall x'\ \exists q', q'', r', r''\neg(0 = 0) \Rightarrow (s(x') = 0 * 0 + 0) \vee (s(x') = 0 * 0 + s(r')) \vee$
   $(s(x') = 0 * s(q') + 0) \vee (s(x') = 0 * s(q'') + s(r''))$

Die Spezifikationsklauseln 3. und 4. sind redundante Klauseln, da ihre Prämissen Widersprüche enthalten und können mit **delete_redundant_clause** entfernt werden.

1. läßt sich mit **delete_redundant_eq_1**, **simplify** und **delete_tautology** aus der Menge der zu beweisenden Spezifikationsklauseln entfernen, während die Spezifikationsklausel 2. unter Verwendung der Kommutativität der Addition (da $s(r') + 0 <_{lpo} 0 + s(r')$) zuerst mit **simplify** und dann mit **positive decomposition**, **simplify** und **delete_redundant_eq_2** zu der folgenden Klausel vereinfacht wird:

2.1 $\forall x', y'\ \exists q', q'', r''\ \neg(s(y') = 0) \Rightarrow (x' < y' = T) \vee (x' = s(y') * q' + y') \vee$
   $(x' = s(y') * s(q'') + r''\wedge\ r'' < y' = T)$

Aus 2.1 entstehen durch die Anwendung von **subsumption_2** zwei Klauseln:

2.1.1 $\forall x', y' \; \exists q', q'', r'' \; \neg(s(y') = 0) \wedge (r'' = y') \Rightarrow (x' < y' = T) \vee (x' = s(y') * q' + r'') \vee$
$(x' = s(y') * s(q'') + r'' \wedge r'' < y' = T)$

2.1.2 $\forall x', y' \; \exists q', q'', r'' \; \neg(s(y') = 0) \wedge \neg(r'' = y') \Rightarrow (x' < y' = T) \vee (x' = s(y') * q' + y') \vee$
$(x' = s(y') * s(q'') + r'' \wedge r'' < y' = T)$

2.1.1 läßt sich mit **subsumption_1** aus der Menge der zu beweisenden Klauseln entfernen, da die Klausel $\forall x', y' \; \exists q', r'' \; \neg(s(y') = 0) \Rightarrow x' = s(y') * q' + r''$ eine kleinere Instanz der zu beweisenden Klausel 2. ist und kann deshalb als Induktionsannahme angewandt werden.

Die Anwendung der neuen Subsumptionsregel **subsumption_3** auf 2.1.2, gefolgt mit der von **delete_redundant_eq_1** liefert dann:

$\forall x', y' \; \exists q', q'', r'' \; \neg(r'' = y') \wedge (r'' < s(y') = T) \Rightarrow (x' < y' = T) \vee (x' = s(y') * q' + y') \vee$
$(r'' < y' = T)$,

welche mit **expand_2** direkt aus der Menge der zu beweisenden Klauseln zu entfernen ist. Damit ist die Spezifikationsklausel K nach Theorem 2 eine induktive Folgerung von Sp_4.

Das nächste Beispiel berechnet die Wurzel einer natürlichen Zahl.

Beispiel 20 Wurzel einer natürlichen Zahl

Wir betrachten noch einmal die KEKS Sp_4 aus dem letzten Beispiel, wobei die Gleichung Φ_2 hier durch Φ_2' ersetzt wird. Φ_2' drückt die Kommutativität der Multiplikation aus.

$\Phi_2' = \{x * y = y * x\}$

Sei nun die folgende Spezifikationsklausel zu beweisen:

$$\forall x\ \exists w, r\ x = w * w + r\ \wedge\ r < s(w + w)$$

Als Induktionsordnung wählen wir die lexikographische Ordnung (von links nach rechts) mit der Präzedenz $0 < s < + < *$. Die Menge $\{0, s(x)\}$ wird als Test-set und x (bzw. w und r) als Induktionsvariable (bzw. zu instanziierende Existenzvariablen) festgelegt.

Die Anwendung der Induktionsregel **expand_1** auf K liefert folgende Klauseln:

1. $0 = 0 * 0 + 0 \wedge 0 < s(0 + 0)$

2. $\forall x'\ \exists w', w'', r''\ (s(x') = s(w') * s(w') + 0 \wedge 0 < s(s(w') + s(w'))) \vee$
   $(s(x') = s(w'') * s(w'') + s(r'') \wedge\ s(r'') < s(s(w'') + s(w'')))$

1. ist eine Tautologie und kann mit **delete_tautology** aus der Menge der zu beweisenden Klauseln entfernt werden. Die zweite Klausel läßt sich mit **simplify**, **delete_trivial_eq** und **positive decomposition** wie folgt vereinfachen:

2.1 $\forall x'\ \exists w', w'', r''\ (x' = s(w') * w' + w') \vee (x' = s(w'') * s(w'') + r'' \wedge r'' < s(w'') + s(w''))$.

Da $w' * s(w') <_{lpo} s(w') * w$ gilt, kann hier durch **simplify** die Kommutativität der Multiplikation ausgenutzt werden. Wir erhalten dann die folgende Klausel:

2.1.1 $\forall x'\ \exists w', w'', r''\ (x' = w' * s(w') + w') \vee (x' = s(w'') * s(w'') + r'' \wedge r'' < s(w'') + s(w''))$.

Ein weiterer Termersetzungsschritt mit **simplify** auf 2.1.1 ergibt:

2.1.1.1 $\forall x'\ \exists w', w'', r''\ (x' = w' * w' + w' + w') \vee (x' = s(w'') * s(w'') + r'' \wedge r'' < s(w'') + s(w''))$.

Aus 2.1.1.1 entstehen durch die Anwendung von **subsumption_2** die folgenden Klauseln:

2.1.1.1.1 $\forall x'\ \exists w', w'', r''\ (r'' = s(w'') + s(w'')) \Rightarrow (x' = w' * w' + r'') \vee$
$(x' = s(w'') * s(w'') + r'' \wedge\ r'' < s(w'') + s(w''))$

2.1.1.1.2 $\forall x'\ \exists w', w'', r''\ \neg(r'' = s(w'') + s(w'')) \Rightarrow (x' = w' * w' + w' + w') \vee$
$(x' = s(w'') * s(w'') + r'' \wedge\ r'' < s(w'') + s(w''))$

2.1.1.1.1 läßt sich mit **subsumption_1** aus der Menge der zu beweisenden Klauseln entfernen, da $\forall x' \; \exists w', r' x' = w' * w' + r''$ eine kleinere Instanz der zu beweisenden Klausel 2. ist, und kann deshalb als Induktionsannahme angewandt werden.

Auf 2.1.1.1.2 wird nun **subsumption_3** angewandt, und wir erhalten die folgende Klausel:

$\forall x' \; \exists w', w'', r'' \; \neg(r'' = s(w'') + s(w'')) \wedge r'' < s(s(w'') + s(w'')) \Rightarrow (x' = w' * w' + w' + w') \vee (r'' < s(w'') + s(w''))$,

die mit **expand_2** direkt aus der Menge der zu beweisenden Klauseln zu entfernen ist.

Nach Theorem 2 folgt dann K induktiv aus der obigen KEKS.

Kapitel 9

Ein Widerlegungsverfahren f¨ur Spezifi kationsklauseln

Beim Gültigkeitsnachweis einer Spezifikationsklausel im initialen Modell einer KEKS ist es von entscheidender Bedeutung, daß möglichst früh Inkonsistenzen erkannt werden. Somit können zum einen Fehler in der Beschreibung des zu synthetisierenden Programms gefunden und entfernt werden. Zum anderen wird der Suchraum erheblich dadurch reduziert, daß Irrwege bzw. Sackgassen, die oftmals unendlich lang sind, vermieden werden.

Wir stellen nun ein Verfahren vor, mit man die Gültigkeit einer Spezifikationsklausel im initialen Modell einer KEKS (in bestimmten Fällen) widerlegen kann. Dadurch, daß unser Verfahren auf Test-sets basiert, die betrachteten KEKS konvergent und hinreichend vollständig sind und nur freie Konstruktoren enthalten, können wir mehr Formeln widerlegen, als die in der Literatur bereits existierenden Methoden (siehe [BRK95], [BOU97], [CK94] und [PAD96]) in der Lage sind.

Wir führen zuerst einige Begriffe ein, die wir in diesem Abschnitt benötigen werden.

Ein Term t ist irreduzibel bzgl. eines bedingten Termersetzungssystems R, falls keine Regel aus R auf t anwendbar ist. Dies kann aus zweierlei Gründen passieren: Entweder gibt es keinen Match von der linken Seite einer Regel aus R auf einen Teilterm von t oder es gibt zwar einen Match, aber die Prämisse der betreffenden Regeln sind nicht erfüllt. Wir werden im folgenden den ersten Fall betrachten.

Definition 67 (Stark irreduzible Terme) Sei Sp eine KEKS und R das zu Sp assoziierte Termersetzungssystem. Ein Term t ist bzgl. R stark irreduzibel, falls er keinen Teilterm an einer strikten Stelle enthält, der eine Instanz der linken Seite einer Regel aus R darstellt.

Wie das folgende Beispiel zeigt, gibt es irreduzible Terme, die nicht stark irreduzibel sind.

Beispiel 21 Gegeben sei das folgende Termersetzungssystem:

$R = \{ f(x,x) \to 0;$
$\quad x > y \Rightarrow f(x,y) \to s(x);$
$\quad y > x \Rightarrow f(x,y) \to x\}$

Dann ist der Term $f(x,y)$ zwar irreduzibel, aber nicht stark irreduzibel.

Zur Überprüfung der Lösbarkeit bzw. Unifizierbarkeit zweier Terme unter einer nicht leeren Theorie gibt es ein syntaktisches Kriterium, das auf dem Check der führenden Funktionssymbole beider Terme basiert.

Definition 68 (Kompatible Funktionssymbole) Sei $Sp = (\Sigma, \Phi, \Omega_c)$ eine KEKS und R das zu Sp assoziierte Termersetzungssystem. Die Kompabilitätsrelation wird für alle Funktionssymbole $f,\ g \in \Sigma \cup \Omega_c$ wie folgt definiert:

$fPg \Leftrightarrow$ es gibt eine Regel $\phi \Rightarrow f(\ldots) \to t \in R$, wobei $t \equiv g(\ldots)$ oder t eine Variable ist.
P^* ist die reflexive transitive Hülle von P.

Gerade die Funktionssymbole, die nicht in der obigen Relation stehen, werden uns besonders interessieren.

Definition 69 (Kollisionspaare) Sei $Sp = (\Sigma, \Phi, \Omega_c)$ eine KEKS und R das zu Sp assoziierte Termersetzungssystem. Die Funktionssymbole $f,\ g \in \Sigma \cup \Omega_c$ bilden ein Kollisionspaar bzgl. Sp, falls es kein Funktionssymbol $s \in \Sigma \cup \Omega_c$ gibt, so daß fP^*s und gP^*s.

Satz 19 *Die Terme $f(\ldots)$ und $g(\ldots)$ sind unter Sp nicht unifizierbar, falls f und g ein Kollisionspaar bzgl. Sp bilden.*

Beweis: Angenommen die beiden Terme $r \equiv f(\ldots)$ und $t \equiv g(\ldots)$ sind mit σ unifizierbar und f, g bilden ein Kollisionspaar. Für alle Grundsubstitutionen τ gilt dann: $Sp \models \tau(\sigma(r)) = \tau(\sigma(t))$. Da Sp hinreichend vollständig und konvergent ist, gibt es dann einen irreduziblen Term $s \equiv h(\ldots)$, so daß $\tau(\sigma(r)) \to_R^* s$ und $\tau(\sigma(t)) \to_R^* s$. Also gilt auch: fP^*h und gP^*h, da für alle Terme $t_1 \equiv f(\ldots)$ und $t_2 = g(\ldots)$ gilt: $t_1 \to_R^* t_2 \Rightarrow fP^*g$. Was ein Widerspruch dazu ist, daß f und g ein Kollisionspaar bzgl. Sp bilden.

□

Bemerkung: Zwei verschiedene Konstruktoren bilden immer ein Kollisionspaar, da sie in jeder KEKS frei sind.

Ausgehend von den obigen Definitionen können wir nun in bestimmten Fällen feststellen, wann eine Spezifikationsklausel bzgl. einer KEKS inkonsistent ist. Dabei beschränken wir uns, um die Notation einfach zu halten, auf Spezifikationsklauseln, die keine Existenzvariablen in ihrer Prämisse enthalten. Die folgende Definition läßt sich aber für Spezifikationsklauseln, in deren Prämisse Existenzvariablen enthalten sind, leicht modifizieren.

Definition 70 (Inkonsistente Spezifikationsklauseln) Sei $K \equiv \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$ eine Spezifikationsklausel, für die $Var(\phi) \cap Y = \emptyset$ gilt. K heißt bzgl. einer KEKS Sp inkonsistent, falls es eine Test-set-Substitution σ mit $Dom(\sigma) \subseteq X$ gibt, so daß $Sp \models \sigma(\phi)$ und es für jedes $i,\ 1 \leq i \leq l$ eine Gleichung $s = t \in \Phi_i$ gibt, so daß

- entweder $s \equiv f(\ldots)$ und $t \equiv g(\ldots)$ und f und g ein Kollisionspaar bilden,
- oder für alle Test-set-Substitutionen η, für die $\eta(x) = \sigma(x)$ für $x \in Dom(\sigma)$ gilt, $\eta(s)$ und $\eta(t)$ stark irreduzibel und nicht (Robinson)-unifizierbar sind.

Unter der Robinson-Unifikation verstehen wir die Unifikation in der leeren Theorie, d.h. die syntaktische Unifikation zweier Terme.

Theorem 3 *Sei Sp eine KEKS und K eine Spezifikationsklausel. Es gilt: $Sp \not\models_{ind} K$, falls K bzgl. Sp inkonsistent ist.*

Bevor wir den Beweis von Theorem 3 angeben, werden wir zuesrt folgende Hilfssätze beweisen.

Satz 20 *Seien t ein Term, R ein bedingtes Termersetzungssystem und σ eine Test-set-Substitution für die in t vorkommenden Variablen. Falls $\sigma(t)$ bzgl. R stark irreduzibel ist, dann gibt es eine Substitution τ, so daß $\tau(\sigma(t))$ ein stark irreduzibler Grundterm ist.*

Beweis: siehe [BR95].

Satz 21 *Seien t ein Term, R das zu einer KEKS Sp assoziierte Termersetzungssystem, σ_1 eine Test-set-Substitution für die in t vorkommenden Variablen und σ_2 eine irreduzible Grundsubstitution. Falls $\sigma_1(t)$ bzgl. R stark irreduzibel ist, dann ist $\sigma_2(\sigma_1(t))$ bzgl. R stark irreduzibel.*

Beweis: Wir zeigen zuerst, daß falls $\sigma_1(t)$ stark irreduzibel ist, dann ist t ein Konstruktorterm. Dazu nehmen wir an, daß t ein Funktionssymbol f enthält. Also t hat einen Teilterm der Form $f(t_1, \ldots, t_n)$, wobei $t_i \in T_{\Sigma,c}(x), 1 \leq i \leq n$. Nach Satz 20 gibt es eine Substitution τ, so daß $\tau(\sigma_1(t))$ ein stark irreduzibler Grundterm ist. Was ein Widerspruch zu der Vollständigkeit von Sp ist. Also ist t ein Konstruktorterm. Daraus und weil $Img(\sigma_1)$ und $Img(\sigma_2)$ nur aus Konstruktortermen bestehen, folgt, daß $\sigma_2(\sigma_1(t))$ ein Konstruktorterm und wegen der Freiheit der Konstruktoren stark irreduzibel.

□

Beweis von Theorem 3: Falls $K \equiv \forall X \exists Y \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$ inkonsistent bzgl. Sp ist, gibt es eine Test-set-Substitution σ, so daß $Sp \models \sigma(\phi)$ und für alle $i,\ 1 \leq i \leq n$ und Test-set-Substitutionen η, eine Gleichung $s = t \in \Phi_i$, so daß $Sp \not\models \eta(s = t)$, wobei $\eta(x) = \sigma(x)$ für alle $x \in Dom(\sigma)$. Also gibt es eine irreduzible Grundsubstitution σ_0, so daß $Sp \models \sigma_0(\sigma(\phi))$ (da Sp hinreichend vollständig und konvergent ist, genügt es nur irreduzible Grundsubstitutionen zu betrachten). Falls nun die führenden Funktionssymbole von t und s ein Kollisionspaar bilden, dann gilt nach Satz 19: $Sp \not\models \eta(s) = \eta(t)$ für alle Substitutionen η und insbesondere für alle Grundsubstitutionen η_1. Falls der zweite Fall eintritt, dann gilt: $\eta(s)$ und $\eta(t)$ sind für alle Test-set-Substitutionen η stark irreduzibel und nicht (Robinson-)unifizierbar. Für eine beliebige irreduzible Gundsubstitution η_1, gibt es eine Test-set-Substitution η_2 und eine irreduzible Grundsubstitution η_0, so daß $\eta_1 = \eta_0 \circ \eta_2$. Nach Satz 21 sind nun $\eta_0(\eta_2(s))$ und $\eta_0(\eta_2(t))$ stark irreduzibel, da $\eta_2(s)$ und $\eta_2(t)$ nach der Behauptung (K ist inkonsistent) stark irreduzibel sind. Zudem gilt, daß $\eta_2(s)$ und $\eta_2(t)$ ununifizierbar sind. Insbesondere gilt dann: $\eta_0(\eta_2(s)) \neq \eta_0(\eta_2(t))$.
Damit haben wir gezeigt, daß es eine Grundsubstitution, etwa $\sigma_1 = \sigma_0 \circ \sigma$ gibt, so daß $Sp \models \sigma_1(\phi)$ und $\eta_1(s)$, $\eta_1(t)$ für alle Grundsubstitutionen η_1 irreduzibel und syntaktisch verschieden sind. Daraus folgt, daß $Sp \not\models \eta_1(s) = \eta_1(t)$ gilt, da das zu Sp assoziierte Termersetzungssystem konvergent ist.

□

Definition 71 (Widerlegungskorrektheit) Sei Sp eine KEKS und I ein Inferenzsystem. I heißt wiederlegungskorrekt, falls für alle Ableitungsschritte $(Kl_i, H_i) \vdash (Kl_{i+1}, H_{i+1})$ folgendes gilt:

$$Sp \models_{ind} (Kl_i \cup H_i) \text{ impliziert } Sp \models_{ind} (Kl_{i+1} \cup H_{i+1}).$$

Intuitiv stellt die Widerlegungskorrektheit eines Inferenzsystems eine Garantie dafür dar, daß bei einem Ableitungsschritt keine inkonsistente Klausel generiert werden.

Wir zeigen nun, daß das Inferenzsystem I_1, das man erhält, wenn man aus I die Regel **expand_2** herausnimmt, widerlegungskorrekt ist.

Satz 22 *Das Inferenzsystem I_1 ist widerlegungskorrekt.*

Beweis: Wir führen den Beweis durch eine Fallunterscheidung über alle Regeln des Inferenzsystems I_1; d.h. für jede Regel r von I_1 wird gezeigt, daß , falls wir durch r von einem Beweiszustand (Kl_1, H_1) zu einem Beweiszustand (Kl_2, H_2) übergehen, dann gilt:

$$Sp \models_{ind} (Kl_1 \cup H_1) \text{ impliziert } Sp \models_{ind} (Kl_2 \cup H_2)$$

1. **expand_1**:
   Falls $Sp \models_{ind} (KL \cup \{C\} \cup H)$ gilt und **expand_1** auf C angewandt wird, dann übergehen wir zu dem Beweiszustand $(Kl \cup (\cup_\sigma C_\sigma), H \cup C)$.
   Zu zeigen ist: $Sp \models_{ind} (Kl \cup \{C\} \cup (\cup_\sigma C_\sigma) \cup H)$.
   Sei nun $K \in (Kl \cup \{C\} \cup (\cup_\sigma C_\sigma) \cup H)$ beliebig:

   (a) $K \in (Kl \cup \{C\} \cup H))$. Dann gilt die Behauptung.

   (b) $K \in (\cup_\sigma C_\sigma)$. Dann gibt es eine Test-set-Substitution τ für die allquantifizierten Variablen und Test-set-Substitutionen $\eta_i,\ 1 \leq i \leq l$ für die Existenzvariablen von C, so daß $K \equiv Norm(\bigvee_{i=1}^{l} \tau(\eta_i(C)))$. Aus der Annahme folgt, daß $Sp \models_{ind} C$ und nach Satz 15 gilt dann $Sp \models_{ind} Norm(\bigvee_{i=1}^{l} \eta_i(C))$. Gemäß Satz 17.2 gilt : $Sp \models_{ind} \sigma(Norm(\bigvee_{i=1}^{l} \eta_i(C)))$ für alle Test-set-Substitutionen σ für die allquantifizierten Variablen und insbesondere für τ gilt dann auch $Sp \models_{ind} \tau(Norm(\bigvee_{i=1}^{l} \eta_i(C)))$. Aus Satz 16 folgt schließlich, daß $Sp \models_{ind} Norm(\bigvee_{i=1}^{l} \tau(\eta_i(C)))$, d.h.: $Sp \models_{ind} K$.

2. **simplify**:
   Annahme: $Sp \models_{ind} (Kl \cup \{C\} \cup H)$
   Zu zeigen ist: $Sp \models_{ind} (Kl \cup \{C'\} \cup H)$, wobei $C \longmapsto_{R[\Phi_2]} C'$.

   (a) $K \in (Kl \cup H)$, dann gilt die Behauptung.

   (b) $K \equiv C'$. Aus der Annahme folgt, daß $Sp \models_{ind} C$. Auf C wurde entweder ein bereits bewiesenes Lemma oder eine Termersetzungsregel angewandt. Daraus folgt, daß $Sp \models_{ind} C'$ gilt.

3. **positive decomposition**:
   Annahme: $Sp \models_{ind} (Kl \cup \{C\} \cup H)$
   Zu zeigen ist: $Sp \models_{ind} (Kl \cup \{C'\} \cup H)$, wobei o.B.d.A. $C \equiv \forall X \exists Y \phi \Rightarrow k_1(t_1, \ldots, t_n) = k_1(s_1, \ldots, s_n) \wedge \Phi_1 \vee \ldots \vee \Phi_n$.

   (a) $K \in (Kl \cup H)$, dann gilt die Behauptung.

   (b) $K \equiv C'$, wobei $C' \equiv \forall X \exists Y \phi \Rightarrow (\bigwedge_{i=1}^{n}(t_i = s_i)) \wedge \Phi_1 \vee \ldots \vee \Phi_n$. Aus der Annahme folgt, daß $Sp \models_{ind} C$, d.h.: Für alle Grundsubstitutionen τ, für die $Sp \models \tau(\phi)$ gilt, gibt es eine Grundsubstitution η, so daß $Sp \models \eta(k_1(t_1, \ldots, t_n) = k_1(s_1, \ldots, s_n) \wedge \Phi_1 \vee \ldots \vee \Phi_n)$. Da R konvergent ist und die Konstruktoren frei sind, gilt auch $Sp \models \eta(t_i) = \eta(s_i)$ für alle $i,\ 1 \leq i \leq n$. Also gilt $Sp \models_{ind} \forall X \exists Y \phi \Rightarrow (\bigwedge_{i=1}^{n}(t_i = s_i)) \wedge \Phi_1 \vee \ldots \vee \Phi_n$.

4. **negative decomposition**: Analog zu **positive decomposition**.

5. **delete_trivial_eq, delete_redundant_eq_1, delete_redundant_eq_2, delete_explicit_eq**: trivial.

6. **delete_tautology, delete_explicit_clause, delete_redundant_clause, subsumption_1**: Die Behauptung folgt direkt aus der Annahme.

7. **case_simplify**:
   Annahme: $Sp \models_{ind} (Kl \cup \{C\} \cup H)$
   Zu zeigen ist: $Sp \models_{ind} (Kl \cup C' \cup H)$, wobei $C' \equiv Fallunterscheidung(C)$.

   (a) $K \in (Kl \cup H)$, dann gilt die Behauptung.

   (b) $K \in C'$. Dann gibt es eine Menge $\phi_i \Rightarrow l_i \rightarrow r_i \in R$, $1 \leq i \leq n$ von Termersetzungsregeln, Substitutionen σ_i, Vorkommen u_i in C, so daß $C|_{u_i} = \sigma_i(l_i)$ und $Sp \models_{ind} \bigvee_{i=1}^{n} \sigma_i(\phi_i)$. Also ist $C' \equiv \{\sigma_i(\phi_i) \Rightarrow C[u_i \leftarrow \sigma_i(r_i)]\}$. Es muß dann ein j, $1 \leq j \leq n$ geben, so daß $K \equiv \sigma_j(\phi_j) \Rightarrow C[u_j \rightarrow \sigma_j(r_j)]$. Falls es nun eine Grundsubstitution τ gibt, so daß $Sp \models \tau(\sigma_j(\Phi_j))$, dann gilt $Sp \models \tau(\sigma_j(l_j)) = \tau(\sigma_j(r_j))$, da $\phi_j \Rightarrow l_j = r_j \in R$. Zudem gilt $Sp \models \tau(C[u_j \leftarrow \sigma_j(r_j)])$, da $Sp \models_{ind} C$ aus der Annahme folgt. Insbesondere gilt dann $Sp \models \tau(C)$. Für alle Grundsubstitutionen τ, für die $Sp \models \tau(\sigma_j(\phi_j))$, gilt auch $Sp \models \tau(C[u_j \leftarrow \sigma_j(r_j)])$. D.h.: $Sp \models_{ind} K$.

8. **binary_split**:
   Annahme: $Sp \models_{ind} (Kl \cup \{C\} \cup H)$, wobei $C \equiv \forall X \exists Y \phi \Rightarrow (s = t \wedge \psi) \vee R$
   Zu zeigen ist: $Sp \models_{ind} (Kl \cup \{C_1, C_2\} \cup H)$, wobei $C_1 \equiv \forall X \exists Y \phi \wedge (s = t) \Rightarrow (s = t \wedge \psi) \vee R$ und $C_2 \equiv \forall X \exists Y \phi \wedge \neg(s = t) \Rightarrow R$.

   (a) $K \in (Kl \cup H)$, dann gilt die Behauptung.

   (b) $K \in \{C_1, C_2\}$. Da wegen der Annahme $Sp \models_{ind} C$ gilt, gibt es für alle Grundsubstitutionen σ, für die $Sp \models \sigma(\phi)$ gilt, eine Grundsubstitution η, so daß $\eta(s = t \wedge \psi) \vee R)$. Falls nun $Sp \models \sigma(\phi)$, dann gilt auch $Sp \models_{ind} C_1$. Falls $Sp \models \neg(\sigma(\phi))$, dann gilt $Sp \models_{ind} C_2$

9. **subsumption_2**: Analog zu **binary_split**.

10. **subsumption_3**:
    Annahme: $Sp \models_{ind} (Kl \cup \{C\} \cup H)$, wobei $C \equiv < \sigma >: \forall X \exists Y \phi_1 \wedge \phi_2 \Rightarrow \Phi_1 \vee \ldots \vee R_1 \wedge R_2 \vee \ldots \vee \Phi_n$
    Zu zeigen ist: $Sp \models_{ind} (Kl \cup \{C_1\} \cup H)$, wobei $C_1 \equiv < \sigma >: \forall X \exists Y \phi_1 \wedge \phi_2 \wedge R_3 \Rightarrow \Phi_1 \vee \ldots \vee R_2 \vee \ldots \vee \Phi_n$

    (a) $K \in (Kl \cup H)$, dann gilt die Behauptung.

    (b) $K \equiv C_1$. Falls es eine Grundsubstitution γ gibt, so daß $Sp \models \gamma(\phi_1 \wedge \phi_2 \wedge R_3)$, dann gilt trivialerweise auch $Sp \models \gamma(\phi_1 \wedge \phi_2)$. Aus der Annahme folgt, daß $Sp \models_{ind} C$. Also gibt es eine Grundsubstitution η, so daß $Sp \models \eta(\Phi_1 \vee \ldots \vee R_1 \wedge R_2 \vee \ldots \Phi_n)$ gilt. Insbesondere gilt dann auch $Sp \models \eta(\Phi_1 \vee \ldots \vee R_2 \vee \ldots \Phi_n)$, dh.: $Sp \models_{ind} K$.

□

Bemerkung: Die Regel **expand_2** initialisiert Induktionsschritte, indem sie für die Existenzvariablen nur eine Test-set-Substitution betrachtet, welche vom System oder vom Benutzer aus mehreren Kandidaten ausgewählt werden kann. Es kann durchaus vorkommen, daß die gewählte Substitution ungeeignet und damit die generierte Klausel inkonsistent bzgl. der betrachteten KEKS ist. Aus diesem Grunde kann nur die Widerlegungskorrektheit des Inferenzsystems I_1 gewährleistet werden.

Zu dem Inferenzsystem I_1 fügen wir nun die folgende Regel hinzu:

disproof $\frac{Disproof}{Kl \cup \{C\},\ H}$,

falls C inkonsistent ist.

Theorem 4 *Sei Sp eine KEKS und $A \equiv (Kl_0, \emptyset) \vdash (Kl_1, H_1) \vdash \ldots$ eine I_1-Ableitung. Falls A mit der Anwendung der Regel* **disproof** *endet, dann gilt: $Sp \not\models_{ind} Kl_0$.*

Beweis: Da die Regel **disproof** angewandt worden ist, muß es eine Klausel $k \in Kl_i$ geben, die inkonsistent bzgl. Sp ist und nach Theorem 3 keine induktive Folgerung von Sp ist. Aus der Widerlegungskorrektheit von I_1 folgt dann die Behauptung.

□

Beispiel 22 Sei Sp_4 die KEKS vom Beispiel 19 und die folgende zu beweisende Spezifikationsklausel:

$$K \equiv \forall x, y\ \exists\, q, r\ x = y * q + r\ \wedge\ r < y$$

Nach der Anwendung von **expand_1** erhalten wir u.a. die folgende klausel:

$\forall x', y'\ \exists q', q'', r', r''\ (s(x') = 0 * 0 + 0 \wedge 0 < 0 = T) \vee (s(x') = 0 * 0 + s(r') \wedge s(r') < 0 = T)$
$\vee (s(x') = 0 * s(q') + 0 \wedge 0 < 0 = T) \vee (s(x') = 0 * s(q'') + s(r'') \wedge s(r'') < 0 = T)$,

welche sich mit **simplify** wie folgt vereinfachen läßt:

(*) $\forall x', y'\ \exists q', q'', r', r''\ (s(x') = 0 \wedge F = T) \vee (s(x') = 0 + s(r') \wedge F = T) \vee (s(x') = 0 * s(q') \wedge F = T) \vee (s(x') = 0 * s(q'') + s(r'') \wedge F = T)$

(*) ist inkonsistent, da F und T ein Kollisionspaar bilden. Nach Thoerem 4 gilt dann: $Sp \not\models_{ind} K$.

Bemerkung: Die Spezifikationsklausel K vom obigen Beispiel unterscheidet sich von der vom Beispiel 19 dadurch, daß hier die Prämisse $\neg(y = 0)$ weggelassen worden ist. Wie das Beispiel

19 aber zeigt ist K mit der Prämisse eine induktive Folgerung von Sp_4. Falls die Prämisse nicht erfüllt ist (d.h. falls (y=0)), lassen sich aus K inkonsistente Klauseln ableiten. Mit dieser Information kann man also die initiale Spezifikationsklausel K entsprechend ändern und einen neuen Beweis starten.

Manche inkonsistente Spezifikationsklauseln lassen sich durch unser Widerlegungsverfahren direkt abfangen, während bei anderen mehrere Schleifen benötigt werden. Unter einer Schleife verstehen wir jede neue Initialisierung eines Induktionsschritts. Wegen der Widerlegungskorrektheit des Inferenzsystems I_1 ist aber gewährleistet, daß die initiale zu beweisende Spezifikationsklausen in solchen Fällen auch inkonsistent ist. Die beiden nächsten Beispiele behandlen die oben erwähnten Fälle.

Beispiel 23 Gegeben sei die folgende KEKS $Sp_5 = (\Sigma, \Phi, \Omega_c)$, bestehend aus den Konstruktoren $0,\ s$ für nat, T, F für $Bool$, den Funktionen:

$$
\begin{aligned}
\Phi = \{\ & x + 0 = x;\\
& x + s(y) = s(x + y);\\
& s(x) + y = s(x + y);\\
& 0 + x = x;\\
& x * 0 = 0;\\
& 0 * x = 0;\\
& x * s(y) = x * y + x;\\
& s(x) * y = x * y + y;\\
& x < 0 = F;\\
& 0 < s(x) = T;\\
& s(x) < s(y) = x < y;\\
& null(0) = 0;\\
& null(s(x)) = null(x)\}
\end{aligned}
$$

und der folgenden Spezifikationsklausel:

$$K \equiv \forall x\ \exists y\ null(x * y) = s(x + y)$$

Gemäß Definition 69 bilden $null$ und s ein Kollisionspaar. Nach Definition 70 bzw. Theorem 3 ist dann K inkonsistent bzgl. Sp_5 bzw. keine induktive Folgerung von Sp_5.

Beispiel 24 Gegeben sei wie oben die KEKS Sp_5 und die folgende Spezifikationsklausel:

$$K \equiv \forall x\ \exists y\ x * y = x + y$$

Nach einer Anwendung der Regel **expand_1** auf K erhalten wir u.a. folgende zu beweisende Klausel:

$\forall x' \; \exists y' \; (s(x') * 0 = s(x') + 0) \vee (s(x') * s(y') = s(x') + s(y'))$,

welche sich mit **simplify**, **positive decomposition** und **delete_redundant_eq_2** wie folgt vereinfachen läßt:

$\forall x' \; \exists y' \; x' * y' + x' + y' = s(x' + y')$.

Auf (*) wird nochmal **expand_1** angewandt, woraus u.a. folgende Klausel entsteht:

$\exists y'' \; (0 * 0 + 0 + 0 = s(0 + 0) \vee (0 * s(y'') + 0 + s(y'') = s(0 + s(y''))))$.

Weitere Vereinfachungen, nämlich mit **simplify**, **positive decomposition** und **delete_redundant_eq_2** ergeben dann:

(**) $\exists y'' \; y'' = s(y'')$.

(**) ist gemäß Definition 70 inkonsistent, weil $\eta(y'')$ und $\eta(s(y''))$ für alle Test-set-Substitutionen η stark irreduzibel und nicht unifizierfar sind. Nach Theorem 3 ist (**) keine induktive Folgerung von Sp_5. Aus Theorem 4 folgt dann sofort, daß K keine induktive Folgerung von Sp_5 ist.

Wir haben nach zwei Schleifen eine Substitution $\sigma : x \rightarrow s(0)$ gefunden, für die es keine Substitution η für die Variable y gibt, so daß $Sp_4 \models \eta(\sigma(K))$ gilt. Die Substitution σ ist somit ein Gegenbeispiel dafür, daß K eine induktive Folgerung von Sp_5 ist.

Ein wichtiger Aspekt bei der Widerlegung einer Spezifikationsklausel ist die Benutzung von Test-sets, wie sie in Definition 52 eingeführt wurden. Falls die zweite Bedingung der Definition nicht erfüllt ist, d.h. falls Cover-sets [ZKK88] statt Test-sets betrachtet werden, ist die Korrektheit des Widerlegungsverfahrens nicht mehr gewährleistet.

Beispiel 25 Gegeben sei die folgende KEKS $Sp_6 = (\Sigma, \Phi, \Omega_c)$, bestehend aus den Konstruktoren $0,\ s$ für nat, T, F für $Bool$, den Funktionen:

$$\begin{aligned}\Phi = \{ & \; x + 0 = x; \\ & \; x + s(y) = s(x + y); \\ & \; ev(0) = T; \\ & \; ev(s(0)) = F; \\ & \; ev(s(s(x))) = ev(x)\}\end{aligned}$$

und die folgende Spezifikationsklausel:

$$K \equiv \forall x \; ev(x) = T \; \vee \; ev(x) = F$$

K ist trivialerweise eine induktive Folgerung von Sp_6. Wähle nun die Menge $M = \{0,\ s(x)\}$ als Test-set. Aus der Anwendung von **expand_1** auf K ensteht u.a. die folgende Klausel:

$\forall x'\ ev(s(x')) = T \vee\ ev(s(x')) = F,$

die nach Definition 70 inkonsistent ist, weil $ev(s(x')), T$ und F stark irreduzibel und unifizierbar sind. Damit wäre auch K keine induktive Folgerung von Sp_6, was offensichtlich inkorrekt ist. Der Grund dafür ist, daß M die zweite Bedingung der Test-set Definition nicht erfüllt und damit ein Cover-set und kein Test-set ist. Mit der Menge $M_1 = \{0, s(0), s(s(x))\}$ als Test-set läßt sich aber K als induktive Folgerung von Sp_6 sofort beweisen.

Kapitel 10

Ein Algorithmus zur Extraktion von Programmen aus Beweisen

In der deduktiven Programmsynthese werden formale Spezifikationen gesuchter Programme als Existenzformeln aufgefaßt. Aus einem konstruktiven Beweis einer solchen Formel läßt sich ein Programm zur Berechnung des als existent nachgewiesenen Objekts extrahieren. Diese Art der Programmierung wird von Bates und Constable in [BAT85] als "very high level Programming" bezeichnet. Dadurch daß man den bewährten mathematisch-logischen Formalismus als Programmiersprache benutzt, werden nachträgliche Korrektheitsbeweise der erstellten Programme überflüssig.

Wir wollen nun ein Verfahren vorstellen, das ausgehend von einem Beweis einer Existenzformel ein rekursives Programm automatisch generiert, das eine Lösung des mit der Existenzaussage formulierten Problems darstellt. Als Grundlage der automatischen Programmextraktion werden jene Beweise dienen, die mit Hilfe des in dieser Arbeit bereits vorgestellten Inferenzsystems I geführt wurden. Das folgende Beispiel soll die Vorgehensweise illustrieren, bevor wir das eigentliche Verfahren präsentieren.

Beispiel 26 Im Beispiel 14 wurde die Gültigkeit folgender Formel nachgewiesen:

$$K \equiv \forall x, y\ \exists z\ x = z + y\ \vee\ y = z + x,$$

welche besagt, daß die positive Differenz etwa $diff(x, y)$, zweier beliebigen natürlichen Zahlen immer existiert. Wegen der hinreichenden Vollständigkeit der Betrachteten Spezifikationen kann jedes Paar natürlicher Zahlen durch ein Paar (a, b) irreduzibler Grundterme dargestellt werden. Um das spezifizierte Programm zu generieren, brauchen wir dann nur eine Menge von Regeln, die den Wert von (a, b) berechnen. Folgende Paare (Test-set-Substitutionen) sind dabei zu betrachten:

$(0,0),(0,s(y')),(s(x'),0)$ und $(s(x'),s(y'))$. Falls man nun die Induktionsvariablen x bzw. y durch die Werte der ersten bzw. zweiten Komponente der obigen Paare ersetzt, läßt sich K bei den ersten drei Fällen zu Formeln vereinfachen, bei denen der Wert von z explizit wird: ($z = 0$, bzw. $z = s(y')$ bzw. $z = s(x')$). Daraus können folgende Regeln abgeleitet werden:
$diff(0,0) = 0$
$diff(0,s(y')) = s(y')$ und
$diff(0,s(x')) = s(x')$

Falls x bzw. y die Werte $s(x')$ bzw. $s(y')$ annehmen, kann die Spezifikationsklausel K wie folgt vereinfacht werden:

$$K' \equiv \forall x', y' \; \exists z \; x' = z + y' \; \vee \; y' = z + x'.$$

Aus K' folgt, daß $z = diff(x', y')$ und daraus entsteht dann die Regel: $diff(s(x'), s(y')) = diff(x', y')$. Damit läßt sich die positive Differenz zweier natürlichen Zahlen durch die folgende Regeln definieren:

$diff(0,0) = 0$;
$diff(0,s(y')) = s(y')$;
$diff(s(x'),0) = s(x')$;
$diff(s(x'),s(y')) = diff(x',y')$.

Eine wichtige Rolle in unserem Verfahren zur Programmextraktion werden die Markierungen der beweisenen Spezifikationsklauseln (siehe Kapitel 10) spielen, in denen alle auf sie im Laufe des Beweises angewandten Substitutionen protokolliert werden. Besonders wichtig ist die letzte Inferenzregel, die auf eine Klausel angewandt wird, bevor sie als bewiesen gilt oder wieder expandiert wird. Folgende Inferenzregeln werden bei der Programmextraktion betrachtet:
delete_tautologie, **delete_explicit_clause, delete_redundant_clause, subsumption_1, subsumption_3, expand_1** und **expand_2**.

10.1 Der Extraktionsalgorithmus

Sei Sp eine KEKS und $K \equiv \forall x_1, \ldots, x_n \exists y_1, \ldots, y_m \; \phi \Rightarrow \Phi_1 \vee \ldots \vee \Phi_l$ eine Spezifikationsklausel, deren Gültigkeit im initialen Modell von Sp mit Hilfe des Inferenzsystems I nachgewiesen wurde. Wir werden Programme für die Skolemfunktionen f_{y_j}, $1 \leq j \leq m$, generieren, die die Spezifikationsklausel K erfüllen. Dazu betrachten wir alle aus K stammenden Klauseln, die aus der Menge der zu beweisenden Klauseln zu entfernen bzw. zu expandieren sind:

$K' \equiv < \sigma >: \forall x'_1, \ldots, x'_p \exists y'_1, \ldots, y'_r \; \phi_1 \wedge \phi_2 \Rightarrow < \eta_1 >: \Phi'_1 \vee \ldots \vee < \eta_k >: \Phi'_k$,

wobei ϕ_2 eine möglicherweise leere Konjunktion von Gleichungen (bzw. negierten Gleichungen)

ist, die durch Anwendung der Regeln **binary_split, case_simplify** oder **subsumption_2** entstanden ist. Folgende Fälle sind dann zu betrachten:

1. Die Regel **delete_tautology** wird auf K' angewandt, dann gibt es ein $i,\ 1 \leq i \leq k$, so daß $< \sigma >$: $\forall x'_1, \ldots, x'_p \exists y'_1, \ldots, y'_r\ \phi_1 \wedge \phi_2 \Rightarrow < \eta_i >$: Φ'_i eine Tautologie ist. Folgende Regel wird dann für jedes $j,\ 1 \leq j \leq m$ generiert:

   $$\phi_2 \Rightarrow f_{y_j}(\sigma(x_1, \ldots, x_n)) = \begin{cases} \eta_i(y_j) & \text{falls } Var(\eta_i(y_j)) \subseteq Var(\sigma(x_1, \ldots, x_n)) \\ G(\eta_i(y_j)) & \text{sonst,} \end{cases}$$

   wobei $G(\eta_i(y_j))$ eine beliebige Grundinstanz von $\eta_i(y_j)$ ist. Wir betrachten die Grundinstanz von $\eta_i(y_j)$, um zu gewährleisten, daß Funktionen und keine Relationen generiert werden.

2. **delete_explicit_clause** wird auf K' angewandt, dann gibt es ein $i,\ 1 \leq i \leq k$, so daß $\Phi_i \equiv y_u = t$. Für jedes $j,\ 1 \leq j \leq m$ wird dann folgende Regel erzeugt:

   $$\gamma(\phi_2) \Rightarrow f_{y_j}(\sigma(x_1, \ldots, x_n)) = \begin{cases} \gamma(\eta_i(y_j)) & \text{falls } Var(\gamma(\eta_i(y_j))) \subseteq Var(\sigma(x_1, \ldots, x_n)) \\ G(\gamma(\eta_i(y_j))) & \text{sonst,} \end{cases}$$

   wobei $\gamma : y_u \rightarrow t$ und $G(\gamma(\eta_i(y_j)))$ eine beliebige Grundinstanz von $\gamma(\eta_i(y_j))$ ist.

3. **delete_redundant_clause** wird auf K' angewandt. Dann enthält entweder ϕ_1 oder ϕ_2 einen Widerspruch (z.B ϕ_1 oder ϕ_2 enthält eine negierte Gleichung der Form $\neg(t = t)$). Falls Φ_2 einen Widerspruch enthält, dann wird keine Regel generiert. Sonst wird für jedes $j,\ 1 \leq j \leq m$ die folgende Regel erzeugt:

   $$\phi_2 \Rightarrow f_{y_j}(\sigma(x_1, \ldots, x_n)) = s,$$

   wobei s ein beliebiger irreduzibler Grundterm der Sorte von y_j ist.

4. Falls **subsumption_1** auf K' angewandt wird, dann gibt es ein $h \in H$, ein $i,\ 1 \leq i \leq k$ und ein $\tau \ll \sigma$, so daß $\tau(h) \equiv \forall X' \exists Y' \phi_1 \Rightarrow < \eta_i >$: $\Phi'_i \wedge R$, wobei $X' \subseteq \{x'_1, \ldots x'_p\},\ Y' \subseteq \{y'_1, \ldots, y'_r\}$ und R eine möglicherweise leere Konjunktion von Gleichungen ist. Folgende Regel wird dann für jedes $j,\ 1 \leq j \leq m$ generiert:

   $$\phi'_2 \Rightarrow f_{y_j}(\sigma(x_1, \ldots, x_n)) = T,$$

   wobei ϕ'_2 bzw. T entstehen, indem man in ϕ_2 bzw. $\eta_i(y_j)$ alle Vorkommen von $\tau(y_u)$, $1 \leq u \leq m$ durch $f_{y_u}(\tau(x_1, \ldots, x_n))$ ersetzt.

5. **subsumption_3** wird auf K' angewandt. Dieser Fall ist analog zum Fall 4 (siehe Begründung weiter unten).

6. Falls **expand_1** bzw. (**expand_2**) auf K' angewandt wird, dann wird für jedes $j,\ 1 \leq j \leq m$ ein neues Funktionssymbol etwa g_{y_j} eingeführt und folgende Regeln generiert:

$$\phi_2 \Rightarrow f_{y_j}(\sigma(x_1, \ldots, x_n)) = g_{y_j}(x'_1, \ldots, x'_p)$$

und der Prozeß wird mit der Spezifikationsklausel K' als neue Eingabe wiederholt.

10.2 Erläuterungen zu dem Extraktionsalgorithmus

Fall 1.: Dieser Fall ist trivial und entspricht sowohl im Beweis einer Existenzformel als auch in der Funktionsdefinition einem Basisfall. Dabei ist ϕ_2 eine möglicherweise leere Konjunktion von Gleichungen, die während des Beweises als Zusatzbedingung in die entsprechende Spezifikationsklausel eingefügt worden ist. Diese Gleichungen bzw. negierten Gleichungen entstehen nach einer Anwendung der Regeln **binary_split, case_simplify** oder **subsumption_2**. Man kann sie deshalb nach jeder Anwendung dieser Regeln markieren, um sie von den anderen Gleichungen in der Prämisse einer Klausel unterscheiden zu können. Die Bedeutung dieser Unterscheidung wird im Fall 3. deutlich.

Fall 2.: Analog zu Fall 1.

Fall 3.: Im Laufe eines Beweises mit Hilfe des Inferenzsystems I können Fallunterscheidungen mit den Regeln **binary_split, case_simplify** oder **subsumption_2** durchgeführt werden. Sie sind dann zwar vollständig, können aber überflüssige Unterfälle beinhalten. Einen solchen Fall hat man zum Beispiel, wenn man die Regel **case_simplify** anwendet, obwohl **simplify** an derselben Stelle anwendbar ist. Bei der Programmgenerierung wird für einen solchen Fall keine Regel generiert. Der Vollständigskeit halber wird dagegen eine Regel erzeugt, falls die nicht markierte Prämisse einer Klausel einen Widerspruch enthält. Wir wollen nämlich nur totale Funktionen extrahieren, deshalb müssen wir alle möglichen Eingabewerte betrachten. Da der Wert der Funktion an der Stelle, an der die Prämisse falsifiziert wird, für die Spezifikationsklausel ohnehin irrelevant ist, können wir für ihn einen beliebigen irreduziblen Grundterm wählen.

Fall 4.: Der Induktionsfall im Beweis entspricht der Rekursionfall bei der Programmextraktion. Auf der rechten Seite der synthetisierten Funktion (möglicherweise auch in der Prämisse) ist ein rekursiver Aufruf der Funktion, deren Argumente bzgl. einer vorgegebenen noetherschen Ordnung kleiner als die in der linken Seite sind.

Fall 5.: Analog zum obigen Fall. Der Unterschied liegt aber darin, daß mit der Regel **subsumption_3** ein Lemma generiert wird, das die Anwendung einer Induktionshypothese ermöglicht. Der Teil des Beweises nach der Anwendung der Regel ist für die zu synthetisierenden Funktion irrelevant, da er ausschließlich für die Gültigkeit des generierten Lemmas dient. Bei der Synthese eines Programms ist der Effekt der Anwendung der beiden Regeln auf das zu extrahierende Objekt gleich.

Fall 6.: Obwohl wir an der Extraktion einzelner Funktionen interessiert sind, kann es vorkommen, daß durch die Generierung von Zwischenlemmata auch Hilfsfunktionen extrahiert werden. Jedesmal, wenn während eines Beweises eine Spezifikationsklausel K generiert wird, auf die keine Regel außer **expand_1** bzw. **expand_2** anwendbar ist, wird K als neue zu beweisende Spezifikationsklausel betrachtet. Dieser Schritt entspricht bei der Programmextraktion der Generierung bzw. Einführung einer neuen Funktion.

10.3 Korrektheit der Extraktionsprozedur

Nun soll gezeigt werden, daß das im Abschnitt 10.1 eingeführte Verfahren garantiert, daß jede generierte Funktion terminierend, konfluent, total und die vorgegebene Spezifikationsklausel erfüllt. Dazu betrachten solche Funktionsdefinitionen, die der folgenden Variablenbedingung genügen.

Definition 72 (Variablenbedingung) Eine durch das Extraktionsverfahren generierte Funktion f erfüllt die Variablenbedingung, falls für alle definierenden Gleichungen
$\phi \Rightarrow f(x_1, \ldots, x_n) = t$ von f folgendes gilt:

$$Var(\phi) \subseteq \{x_1, \ldots, x_n\} \text{ und } Var(t) \subseteq \{x_1, \ldots, x_n\}.$$

In bestimmten Fällen lassen sich generierte Funktionen so modifizieren, daß sie die obige Variablenbedingung erfüllen. Durch das Weglassen von Gleichungen (bzw. negierten Gleichungen) in der Prämisse läßt sich etwa eine definierende Gleichung $\phi \wedge (r = s) \Rightarrow f(x_1, \ldots, x_n) = t$ in $\phi \Rightarrow f(x_1, \ldots, x_n) = t$ überführen, falls in $Var(r = s)$ Variablen vorkommen, die weder in $\{x_1, \ldots, x_n\}$ noch in $Var(t)$ enthalten sind. Genauso kann man eine definierende Gleichung $\phi \wedge (y = s) \Rightarrow f(x_1, \ldots, x_n) = t$ in $\sigma(\phi) \Rightarrow f(x_1, \ldots, x_n) = \sigma(t)$ umwandeln, falls y eine Variable ist, die in t aber nicht in $\{x_1, \ldots, x_n\}$ vorkommt, wobei $\sigma : y \rightarrow s$ und $Var(s) \subseteq \{x_1, \ldots, x_n\}$. Die nach der Anwendung der beiden Regeln erhaltenen Funktionen sind offensichtlich induktive Folgerungen der ursprünglichlichen und demnach äquivalent.

Theorem 5 *(Korrektheit der Extraktionsprozedur) Sei $Sp = (\Sigma, \Psi)$ eine KEKS, $K \equiv \forall x_1, \ldots, x_n \; \exists y_1, \ldots, y_m \; \Phi$ eine Spezifikationsklausel und f_{y_j}, $1 \leq j \leq m$, Funktionen, die mit der Extraktionsprozedur generiert wurden. Falls alle definierenden Gleichungen von f_{y_j} die Variablenbedingung erfüllen, dann sind die f_{y_j}, $1 \leq j \leq m$ terminierend, konfluent, total und es gilt:*

$$Sp' \models_{ind} \forall x_1, \ldots, x_n \; \Phi[y_j / f_{y_j}(x_1, \ldots, x_n)],$$

wobei $Sp' = (\Sigma \cup \Sigma_f, \Psi \cup \Psi_f)$ und Σ_f bzw. Ψ_f die Menge aller neu generierten Funktionssymbole bzw. Regeln sind.

Beweis:

1. **Terminierung**: Da die generierten Funktionen rekursiv definierte Algorithmen sind, muß gezeigt werden, daß die Argumente der rekursiven Aufrufe kleiner als die zugehörigen Eingaben sind. Alle definierenden Gleichungen haben die folgende Form:

   $$\phi \Rightarrow f(\sigma(x_1, \ldots, x_n)) = t.$$

   Falls ein rekursiver Aufruf in t enthalten ist, dann wurde die Regel im Fall 4. bzw. 5 des Algorithmus generiert. In t kann aber demnach nur einen Aufruf von f mit dem Argument $\tau(x_1, \ldots, x_n)$ enthalten sein, wobei τ bzgl der vorgegebenen noetherschen Ordnung kleiner als σ ist. Falls nun in ϕ ein rekursiver Aufruf vorkommt, müssen wir zwei Fälle betrachten. Wird in Φ die Funktion f aufgerufen, dann sind ihre Argumente auch bzgl. der Ordnung kleiner als die zugehörigen Eingaben. Wird aber in Φ eine andere zu synthetisierende Funktion g aufgerufen, dann haben wir eine verschränkte Rekursion. Die Terminierung der Funktion läßt sich hier zeigen, indem man die verschränkt rekursiven Funktionen identifiziert. Das bedeutet, daß die Aufrufe der Funktion g in der Prämisse von f als rekursive Aufrufe von f und umgekehrt betrachtet werden (siehe [GIE96]). Dieser Fall ist also analog zum ersten. In allen Fällen sind die Argumente der rekursiven Aufrufe kleiner als die Eingaben. Daraus folgt, daß die generierten Funktionen terminieren.

2. **Konfluenz**: Wegen der Vollständigkeit der Fallunterscheidungen, die durch die Regeln **binary_split, case_simplify** und **subsumption_2** im Laufe eines Beweises geführt wurden, schliessen sich die Prämissen der generierten Funktionen bei Überlappungen der linken Seiten gegenseitig aus. Daraus folgt, daß die generierten Funktionen nicht-überlappend sind (d.h. es gibt weder triviale noch lösbare kritische Paare zwischen den definierenden Gleichungen). Zudem erfüllen die definierenden Gleichungen die Variablenbedingung und sind nach 1. terminierend. Daraus folgt, daß sie konfluent sind (siehe [KAP85] und [JW86]).

3. **Totalität**: Wir müssen zeigen, daß jeder Grundterm $f(t_1, \ldots, t_n)$ reduzibel ist, wobei f eine generierte Funktion und $t_1, \ldots, t_n$ Grundterme aus T_Σ sind. Da Sp hinreichend vollständig und konvergent ist, genügt es, jedes $t_i,\ 1 \leq i \leq n$ als irreduziblen Grundterm zu betrachten. Demnach gibt es eine Test-set-Substitution γ und eine irreduzible Grundsubstitution τ, so daß $\tau(\gamma(x_1, \ldots, x_n)) = (t_1\ \ldots, t_n)$. Also ist $f(t_1, \ldots, t_n)$ eine Instanz von mindestens einer linken Seite einer definierenden Gleichungen von f. Wegen der Vollständigkeit der Fallunterscheidungen, die durch die Regeln **binary_split, case_simplify** und **subsumption_2** im Laufe eines Beweises geführt wurden, gilt aber $Sp' \models_{ind} \bigvee_{j=1}^{r} \phi_j$ für alle definierenden Gleichungen:

   $$\begin{aligned} \phi_1 &\Rightarrow f(\gamma(x_1 \ldots, x_n)) = t_1 \\ &\vdots \\ \phi_r &\Rightarrow f(\gamma(x_1 \ldots, x_n)) = t_r \end{aligned}$$

Also gibt es ein $j,\ j \in \{1,\ldots,r\}$, so daß $Sp' \models \tau(\gamma(\phi_j))$. Daraus folgt, daß $f(t_1,\ldots,t_n)$ reduzibel ist.

Aus 1., 2. und 3. folgt nun, daß die generierten Funktionen hinreichend vollständig sind (siehe [MOSR]). Da Ψ und ψ_f bedingte Konstruktorsysteme sind und Σ_f keine neuen Konstruktorsymbole enthält, ist die Anreicherung Sp' von Sp vollständig, konsistent und demnach persistent (siehe [MID92]). Mit anderen Wortern: Durch die Anreicherung werden weder zusätzliche Träger erzeugt noch welche identifiziert, die es in Sp nicht waren.

4. **Korrektheit der generierten Programme**: Nun zeigen wir, daß die generierten Programme auch die vorgegebene Spezifikationsklausel $K \equiv \forall x \exists y\ \Phi$ erfüllen. Um die Notationen übersichtlich zu halten, beweisen wir die Aussage für Spezifikationsklauseln, die nur eine allquantifizierte und eine existenzquantifizierte Variable enthält. Da Sp' hinreichend vollständig und konvergent ist, genügt es zu zeigen, daß für alle irreduziblen Grundterme s folgendes gilt:

$$Sp' \models \Phi[x/s, y/f(s)]$$

Wir führen den Beweis per Induktion über die vorgegebene noethersche Ordnung auf die Grundterme. Da Sp' hinreichend vollständig und konvergent ist, gibt es eine Test-set-Substitution τ und eine irreduzible Grundsubstitution σ, so daß $\sigma(\tau(x)) = s$. Also müssen wir zeigen, daß $Sp' \models \Phi[x/\sigma(\tau(x)), y/f(\sigma(\tau(x)))]$. Es gilt $Sp \models_{ind} K$ und demnach $Sp' \models_{ind} K$, da Sp' eine persistente Erweiterung von Sp ist. Nach Satz 17 gilt aber auch $Sp' \models_{ind} \gamma(K)$ für alle Test-set-Substitutionen γ, für die gilt $Dom(\gamma) = x$. Also gilt insbesondere für die Test-set-Substitution τ: $Sp' \models_{ind} \tau(K)$. Wir betrachten nun alle aus $\tau(K)$ stammenden Klauseln, die aus der Menge der zu beweisenden Klauseln entfernt worden sind. Wie in dem Extraktionsalgorithmus sind diejenigen Spezifikationsklauseln relevant, auf die die Inferenzsregeln **delete_tautologie**, **delete_explicit_clause, delete_redundant_clause, subsumption_1, subsumption_3, expand_1** und **expand_2** angewandt worden sind. Sie lassen sich alle wie folgt schreiben:

$K_i \equiv <\tau>: \forall x'_1,\ldots,x'_p\ \exists y'_1,\ldots,y'_r\ \phi_i \wedge \phi'_i \Rightarrow <\eta_{i_1}>: \Phi_{i_1} \vee \ldots \vee <\eta_{i_l}>: \Phi_{i_l},\ 1 \leq i \leq k,$

wobei die ϕ_i's aus der Anwendung einer der Regeln **binary_split, subsumption_2** und **case_simplify** entstanden sind und bilden demnach eine vollständige Fallunterscheidung, d.h.: $Sp \models_{ind} \bigvee_{i=1}^{k} \phi'_i$. Es gibt also ein $j,\ j \in \{1 \ldots k\}$, so daß $Sp \models \sigma(\tau(\phi_j))$. Auf K_j wurde eine der folgenden Regeln angewandt:

   (a) **delete_tautologie**: Dann ist
   $K_j \equiv <\tau>: \forall x'_1,\ldots,x'_p\ \exists y'_1,\ldots,y'_r\ \phi_j \wedge \phi'_j \Rightarrow <\eta_{j_u}>: \Phi_{j_u}$ für ein $u \in \{1,\ldots,l\}$ eine Tautologie. Daraus folgt, daß $Sp' \models \Phi[x/\sigma(\tau(x)), y/\sigma(\eta_{j_u})]$ gilt. Gemäß dem Extraktionsalgoritmus wurde folgende Regel generiert:

$$\phi'_j \Rightarrow f(\tau(x)) = \begin{cases} \eta_{j_u(y)} & \text{falls } Var(\eta_{j_u}(y)) \subseteq Var(\tau(x)) \\ G(\eta_{j_u}(y)) & \text{sonst,} \end{cases}$$

Also gilt $Sp' \models \Phi[x/\sigma(\tau(x)), y/f(\sigma(\tau(x)))]$.

(b) **delete_explicit_clause**: Dann gibt es ein $u \in \{1,\ldots,l\}$ und ein $v \in \{1,\ldots,p\}$, so daß $\Phi_{j_u} \equiv y_v = t$. Daraus folgt, daß $Sp' \models \Phi[x/\sigma(\tau(x)), y/\sigma(\gamma(\eta_{j_u}))]$ gilt, wobei $\gamma : y_v \to t$. Zudem wurde die folgende Regel von dem Extraktionsalgorithmus generiert:

$$\phi'_j \Rightarrow f(\tau(x)) = \begin{cases} \gamma(\eta_{j_u}(y))) & \text{falls } Var(\gamma(\eta_{j_u}(y))) \subseteq Var(\tau(x)) \\ G(\gamma(\eta_{j_u})) & \text{sonst,} \end{cases}$$

Also gilt auch $Sp' \models \Phi[x/\sigma(\tau(x)), y/f(\sigma(\tau(x)))]$.

(c) **delete_redundant_clause**: Falls ϕ'_j einen Widerspruch enthält, dann gibt es nichts zu zeigen. Sonst ist K_j eine Tautologie und der Beweis ist analog zu Fall 1.

(d) **subsumption_1**: Es gibt also ein $h \in H,\ \tau_1 \ll \tau$ und ein $u, \in \{1,\ldots,l\}$, so daß K_j durch

$$\tau_i(h) \equiv \forall x''_1,\ldots,x''v\exists y''_1,\ldots,y''_q\phi_j \Rightarrow < \eta_{j_u} >: \Phi_{j_u} \wedge R$$

subsumiert wird. Also läßt sich $\Phi[x/\sigma(\tau(x)), y/\sigma(\eta_{j_u}(y))]$ in endlich vielen Schritten durch das Inferenzsystem I zu

$$\Phi[x/\sigma(\tau_1(x), y/\sigma(\tau_1(y))] \qquad (*)$$

reduzieren. Zudem wurde durch den Extraktionsalgorithmus die folgende Regel erzeugt:

$$\phi'_j \Rightarrow f(\tau(x)) = T, \qquad (**)$$

wobei ϕ'_j bzw. T entstanden sind, indem alle Vorkommen von $\tau_1(y)$ in Φ'_j bzw η_{j_u} durch $f(\tau_1(x))$ ersetzt wurden. Also läßt sich $\Phi[x/\sigma(\tau(x)), y/f(\sigma(\tau(x)))]$ mit (**) zu $\phi[x/\sigma(\tau(x)), y/\sigma(T)]$ vereinfachen, welche sich (wegen (*)) zu $\Phi[x/\sigma(\tau_1(x)), y/f(\sigma(\tau_1(y)))]$ reduzieren läßt. Da nun $\sigma(\tau_1(x)) < \sigma(\tau(x))$ gilt, folgt aus der Induktionsannahme, daß $Sp' \models \Phi[x/\sigma(\tau_1(x)), y/f(\sigma(\tau_1(y)))]$ gilt. Denmach gilt auch $Sp' \models \Phi[x/\sigma(\tau(x)), y/f(\sigma(\tau(x)))]$.

(e) **subsumptiom_3**: Analog zu (d).

(f) **expand_1** bzw. **expand_2**: Analog zu (d).

□

Im Laufe eines Beweises mit dem Inferenzsystem I werden durch die Anwendung der Regeln **binary_split, case_simplify** und **subsumption_2** negierte Gleichungen in den Prämissen der zu beweisenden Spezifikationsklauseln eingefügt. Diese negierten Gleichungen kommen dann auch in den Prämissen der mit Hilfe der Extraktionsprozedur generierten Funktionen vor. Kanonische erweiterte konstruktive Spezifikationen enthalten gemäß Definition 46 keine negierten Gleichungen in ihren Prämissen. Um eine KEKS mit neu generierten Funktionen zu erweitern, müssen also

die negierten Gleichungen entfernt werden. Eine Möglichkeit hierfür wäre eine neue Funktion eq und eine neue Sorte mit den Konstruktoren $TRUE$ und $FALSE$ in die Spezifikation einzuführen. Jede Gleichung $s = t$ kann dann durch die Gleichung $eq(s,t) = TRUE$ und jede negierte Gleichung $\neg(s = t)$ durch die Gleichung $eq(s,t) = FALSE$ ersetzt werden. Die binären Funktionen eq_s sollen dann vollständig über s definiert sein, wobei s die Sorte eines Terms ist, der als rechte (bzw. linke) Seite einer Gleichung (bzw. negierte Gleichung) in der Prämisse der neu generierten Funktionen vorkommt. Der folgende Satz garantiert, daß die neu erhaltene KEKS eine persistente Erweiterung der alten Spezifikation ist.

Satz 23 *(Okada). Sei R ein bedingtes Termersetzungssystem und R^* ihre oben erwähnte Erweiterung. Seien s und t zwei Terme, in denen die neuen Funktionen $eq_s, TRUE$ und $FALSE$ nicht enthalten sind. Dann gilt*

$$R^* \vdash s \downarrow t \Leftrightarrow R \vdash s \downarrow t,$$

wobei "$\vdash$" die Ableitbarkeitsrelation ist.

In dem obigen Satz wurde angenommen, daß die Funktionen eq_s und die Konstruktoren $TRUE$ bzw. $FALSE$ in R nicht vorkommen. Falls dies nicht der Fall ist, kann man sie in R^* anders Bezeichnen.

Beweis: (siehe [OKA87]).

10.4 Beispiele

Aus den Beweisen vom Abschnitt 8.5 werden nun mit Hilfe der Extraktionsprozedur rekursive Programme generiert. Die Markierung der bewiesenen Klauseln wird eine Schlüsselrolle bei der automatischen Programmextraktion spielen. Sie enthält alle Substitutionen, die im Laufe eines Beweises auf die Spezifikationsklauseln angewandt worden sind. Eine Spezifikationsklausel gilt (trivialerweise) als bewiesen, falls sie aus der Menge der zu beweisenden Klauseln direkt entfernt werden kann. Dies ist aber nur dann möglich, wenn eine Regel aus der Gruppe III des Inferenzssystems (Regeln für Klauseleliminierung) auf sie angewandt wird. Je nachdem welche Regel aus dieser Gruppe angewandt wurde, erzeugt der Extraktionsalgorithmus eine definierende Gleichung für die zu extrahierende Funktion. Wir werden demnach in den folgenden Beispielen nur die Klauseln betrachten, die entweder zu entfernen, oder wieder zu expandieren sind. Letzteres kann eintretten, falls keine andere Regel des Systems ausser **expand_1** bzw. **expand_2** auf eine Klausel K anwendbar ist. In diesem Fall wird K als neu zu beweisende Spezifikationsklausel betrachtet und der Algorithmus generiert automatisch Funktionen, die die neue Spezifikation erfüllen.

Beispiel 27 Positive Differenz zweier natürlicher Zahlen

Im Beispiel 14 war die folgende Spezifikationsklausel vorgegeben:

$$\forall x, y \; \exists z \; x = z + y \vee y = z + x$$

Wir wollen die Funktion $Diff$ generieren, so daß für alle natürlichen Zahlen x und y die Gleichung $diff(x, y) = z$ gilt.

Auf folgende Klauseln wurde eine Regel aus der Gruppe III angewandt:

(1) $< \mathbf{x} \leftarrow \mathbf{0}, \mathbf{y} \leftarrow \mathbf{0} > : \; \exists z \; 0 = z \vee z = 0$

(2) $< \mathbf{x} \leftarrow \mathbf{0}, \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y'}) > : \; \forall y' \exists z \; 0 = z + s(y') \vee s(y') = z$

(3) $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{x'}), \mathbf{y} \leftarrow \mathbf{0} > : \; \forall x' \exists z s(x') = z \vee 0 = z + s(x')$

(4) $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{x'}), \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y'}) > : \; \forall x', y' \exists z x' = z + y' \vee y' = z + x'$

Auf die drei ersten Klauseln wurde die Regel **delete_explicit_clause** angewandt. Gemäß der Extraktionsprozedur (Fall b.) werden die folgenden definierenden Gleichungen generiert:

$Diff(0, 0) = 0$;
$Diff(0, s(y')) = s(y')$
$Diff(s(x'), 0) = s(x')$.

Auf **(4)** wurde die Regel **subsumption_1** angewandt und nach dem vierten Fall des Extraktionsalgorithmus wird folgende Gleichung erzeugt:

$Diff(s(x'), s(y')) = Diff(x', y')$.

Damit ist die gesuchte Funktion $Diff$ vollständig definiert. Nach Theorem 5 ist sie terminierend, nicht überlappend, total und erfüllt die obige Spezifikation.

Beispiel 28 Division einer geraden Zahl durch 2

Aus dem Beweis der Spezifikationsklausel

$$\forall x \; \exists y \; even(x) = T \Rightarrow x = y + y$$

entnehmen wir die folgenden Klauseln, auf die eine Inferenzregel der Gruppe III angewandt wurde.

(1) $< \mathbf{x} \leftarrow \mathbf{0} > : \; \exists y' < \mathbf{y} \leftarrow \mathbf{0} > : 0 = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{0}) > : 0 = s(0) + s(0) \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{y'})) > :$
$0 = s(s(y')) + s(s(y'))$

(2) $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{0}) > : \exists y' \; T = F \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(0) = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{0}) > : s(0) = s(0) + s(0)$
$\vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{y'})) > : s(0) = s(s(y')) + s(s(y'))$

(3) $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{x}')) > : \forall x' \exists y'\ even(x') = T \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : s(s(x')) = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{0}) > : x' = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{s}(\mathbf{y}')) > : x' = s(y') + s(y')$

Auf **(1)** wurde die Regel **delete_tautology**, auf **(2)** die Regel **delete_redundant_clause** und auf **(3)** **subsumption_1** angewandt. Gemäß Fall (a), (c) und (d) des Extraktionsalgorithmus werden die folgenden definierenden Gleichungen für die Funktion div_2 generiert:

$div_2(0) = 0$;
$div_2(s(0)) = 0$;
$div_2(s(s(x')) = s(div_2(x'))$.

Die Funktion div_2 erfüllt nach Theorem 5 die obige Spezifikationsklausel. Sie ist nach demselben Theorem terminierend, nicht überlappend und total.

In dem nächsten Beispiel soll eine Funktion generiert werden, die die Division einer natürlichen Zahl durch 2 berechnet. Die generierte Funktion ist deshalb interessant, weil sie eine negierte Gleichung in ihrer Prämisse enthält. Da die kanonischen erweiterten konstruktiven Spezifikationen, die wir hier betrachten, nur Gleichungen in ihren Prämissen enthalten, soll die generierte Funktion so transformiert werden, daß diese Form auch eingehalten wird.

Beispiel 29 Gegeben war die folgende Spezifikationsklausel

$$\forall x,\ \exists y\ x = y + y\ \vee\ x = s(y + y).$$

Der Beweis, daß die obige Klausel die KEKS $Komnat$ erfüllt, wurde im Beispiel 16 geführt. Daraus entnehmen wir die folgenden Klauseln, die aus der Menge der zu beweisenden Klauseln entfernt worden sind:

(1) $< \mathbf{x} \leftarrow \mathbf{0} > : \exists y' < \mathbf{y} \leftarrow \mathbf{0} > : 0 = 0$

(2) $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{x}') > : \forall x' \exists y'\ x' = 0 \Rightarrow < \mathbf{y} \leftarrow \mathbf{0} > : x' = 0 \vee < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y}') > : (x' = s(y') + y' \vee x' = s(y') + s(y'))$

(3) $< \mathbf{x} \leftarrow \mathbf{s}(\mathbf{x}') > : \forall x' \exists y'\ \neg(x' = 0) \Rightarrow < \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y}') > : (x' = s(y') + y' \vee x' = s(y') + s(y'))$.

Auf **(1)** und **(2)** wurde dann die Regel **delete_tautology** angewandt und der Extraktionsalgorithmus generiert dann für die Funktion div_2' die Regeln:

$div_2'(0) = 0$;
$x' = 0 \Rightarrow div_2'(s(x')) = 0$.

Auf **(3)** wurde die Regel **expand_1** angewandt, und der Extraktionsalgorithmus führt dann ein neues Funktionssymbol, etwa g, ein, und die folgende definierende Gleichung wird erzeugt:

$\neg(x' = 0) \Rightarrow div_2'(s(x')) = g(x')$.

Die Spezifikationsklausel **(3)** wurde expandiert und aus den folgenden Klauseln wird die neue Funktion g generiert:

(3.1) $< \mathbf{x}' \leftarrow \mathbf{0} > : \exists y'' \neg(0 = 0) \Rightarrow < \mathbf{y}' \leftarrow \mathbf{0} > : (0 = s(0) \vee 0 = s(0) + s(0)) \vee < \mathbf{y}' \leftarrow \mathbf{s}(\mathbf{y}'') > : (0 = s(s(y'') + s(y') \vee 0 = s(s(y'')) + s(s(y'')))$

(3.2) $< \mathbf{x}' \leftarrow \mathbf{s}(\mathbf{x}'') > : \forall x'' \exists y'' < \mathbf{y}' \leftarrow \mathbf{0} > : (x'' = 0 \vee x'' = s(0)) \vee < \mathbf{y}' \leftarrow \mathbf{s}(\mathbf{y}'') > : (x'' = s(y'') + s(y'') \vee x'' = s(s(y'') + s(y'')))$.

Aus diesen beiden Klauseln erzeugt der Extraktionsalgorithmus die folgenden definierenden Gleichungen:

$g(0) = 0$;
$g(s(x'')) = s(div_2'(x''))$.

Obwohl wir lediglich die Funktion div_2' extrahieren wollten, hat die automatische Generierung von Lemmata zur Folge, daß eine weitere Funktion nämlich g, extrahiert wurde. Aus dem Beweis der obigen Spezifikation wurden schließlich folgende Regeln generiert:

$div_2'(0) = 0$;
$x' = 0 \Rightarrow div_2'(s(x')) = 0$;
$\neg(x' = 0) \Rightarrow div_2'(s(x')) = g(x)$
$g(0) = 0$;
$g(s(x'')) = s(div_2'(x''))$.

Die negierte Gleichung in der obigen Definition läßt sich entfernen, indem man eine neue Sorte $Bool$ mit den Konstruktoren $TRUE$ und $FALSE$, die folgende binäre Funktion eq

$eq(0, 0) = TRUE$;
$eq(0, s(x) = FALSE$;
$eq(s(x), 0) = FALSE$;
$eq(s(x), s(y)) = eq(x, y)$.

einführt und die Gleichung $x' = 0$ bzw. negierte Gleichung $\neg(x' = 0)$ in der Definition von div_2' durch $eq(x', 0) = TRUE$ bzw. $eq(x', 0) = FALSE$ ersetzt. Die durch die Transformation erweiterte KEKS ist dann nach Satz 23 eine persitente Erweiterung von $komnat$.

Beispiel 30 Zerlegung einer Liste: Variante I

Aus dem Beweis der Spezifikationsklausel (siehe Beispiel 17)

$$\forall l_1 \exists l_2, x \neg(l_1 = nil) \Rightarrow l_1 = app(l_2, cons(x, nil))$$

wollen wir nun die Funktionen $bustlast$ und $last$ extrahieren. Dazu betrachten wir die folgenden

Spezifikationsklauseln, die im Laufe des Beweises aus der Menge der zu beweisenden Klauseln entfernt worden sind:

(1) $< \mathbf{l_1} \leftarrow \mathbf{nil} > : \exists m_2, a_2, x\ \neg(nil = nil) \Rightarrow < \mathbf{l_2} \leftarrow \mathbf{nil} > : nil = app(nil, cons(x, nil)) \vee$
$< \mathbf{l_2} \leftarrow \mathbf{cons(a_2, m_2)} > : nil = app(cons(a_2, m_2), cons(x, nil))$

(2) $< \mathbf{l_1} \leftarrow \mathbf{cons(a_1, m_1)} > : \forall m_1, a_1 \exists m_2, a_2, x\ (m_1 = nil) \Rightarrow < \mathbf{l_2} \leftarrow \mathbf{nil}, \mathbf{x} \leftarrow \mathbf{a_1} > :$
$m_1 = nil \vee < \mathbf{l_2} \leftarrow \mathbf{cons(a_2, m_2)} > : a_1 = a_2 \wedge m_1 = app(m_1, cons(x, nil))$

(3) $<\mathbf{l_1} \leftarrow \mathbf{cons(a_1, m_1)}> : \forall m_1, a_1 \exists m_2, a_2, x\ \neg(m_1 = nil) \Rightarrow <\mathbf{l_2} \leftarrow \mathbf{cons(a_2, m_2)}, \mathbf{a_2} \leftarrow \mathbf{a_1}>:$
$m_1 = app(m_2, cons(x, nil))$

Aus **(1)**, **(2)** und **(3)** generiert nun der Extraktionsalgorithmus die folgenden definierenden Gleichungen:

$butlast(nil) = nil;$
$m_1 = nil \Rightarrow butlast(cons(a_1, m_1)) = nil;$
$\neg(m_1 = nil) \Rightarrow butlast(cons(a_1, m_1)) = cons(a_1, butlast(m_1));$

$last(nil) = c;$
$m_1 = nil \Rightarrow last(cons(a_1, m_1)) = a_1;$
$\neg(m_1 = nil) \Rightarrow last(cons(a_1, m_1)) = last(m_1),$

wobei c ein beliebiger irreduzibler Grundterm der Sorte el ist. Wie im letzten Beispiel läßt hier sich auch die obige Definition der Funktionen $last$ bzw. $butlast$ so modifizieren, daß keine negierte Gleichung in ihrer Prämisse vorkommt.

Analog zu dem letzten generiert der Extraktionsalgoritmus in dem folgenden Beispiel zwei Funktionen. Die erste liefert die n ersten und die zweite die $l - n$ letzten Elemente einer vorgegebenen Liste der Länge l.

Beispiel 31 Die folgende Spezifikationsklausel ist im Beispiel 18 bewiesen worden:

$$\forall l_1 : list, n : nat\ \exists l_2, l_3 : list\ n \leq length(l_1) \Rightarrow l_1 = app(l_2, l_3) \wedge length(l_2) = n.$$

Aus den Klauseln

(1) $< \mathbf{l_1} \leftarrow \mathbf{nil}, \mathbf{n} \leftarrow \mathbf{0} > : \exists l_3 : list < \mathbf{l_2} \leftarrow \mathbf{nil} > : nil = l_3$

(2) $< \mathbf{l_1} \leftarrow \mathbf{nil}, \mathbf{n} \leftarrow \mathbf{s(n')} > : \forall n' : nat\ \exists l_3\ s(n') \leq 0 \Rightarrow < \mathbf{l_2} \leftarrow \mathbf{nil} > : nil = l_3 \wedge 0 = s(n')$

(3) $< \mathbf{l_1} \leftarrow \mathbf{cons(a_1, m_1)}, \mathbf{n} \leftarrow \mathbf{0} > : \forall m_1 : list, a_1 : el\ \exists l_3 < \mathbf{l_2} \leftarrow \mathbf{nil} > : cons(a_1, m_1) = l_3$

(4) $< \mathbf{l_1} \leftarrow \mathbf{cons(a_1, m_1)}, \mathbf{n} \leftarrow \mathbf{s(n')} > : \forall m_1 : list, n' : nat\ \exists m_2, l_3 : list\ n' \leq length(m_1) \Rightarrow$
$< \mathbf{l_2} \leftarrow \mathbf{cons(a_2, m_2)} > : m_1 = app(m_2, l_3) \wedge length(m_2) = n'$

generiert der Extraktionsalgorithmus die folgenden Funktionen $Prafix$ und $Suffix$, so daß $Präfix(l_1, n) = l_2$ und $Suffix(l_1, n) = l_3$:

$Präfix(nil, 0) = nil;$
$Präfix(nil, s(n')) = nil;$
$Präfix(cons(a_1, m_1), 0) = nil;$
$Präfix(cons(a_1, m_1), s(n')) = cons(a_1, Präfix(m_1, n'))$

$Suffix(nil, 0) = nil;$
$Suffix(nil, 0) = nil;$
$Suffix(cons(a_1, m_1), 0) = cons(a_1, m_1);$
$Suffix(cons(a_1, m_1), s(n')) = Suffix(m_1, n')$

In den nächsten Beispielen werden rekursive Programme für die Division mit Rest zweier natürlichen Zahlen und für die Wurzelziehung einer natürlichen Zahl generiert.

Beispiel 32 Euklidische Division

Die Gültigkeit der Spezifikationsklausel

$$\forall x, y\ \exists q, r\ \neg(y = 0) \Rightarrow x = y * q + r\ \wedge\ r < y = T$$

im initialen Modell der KEKS sp_4 wurde bereits im Beispiel 19 bewiesen. Für den Extraktionsalgorithmus sind folgende Klauseln aus dem Beweis von Bedeutung:

(1) $< \mathbf{x} \leftarrow \mathbf{0}, \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y}') > : \forall y' < \mathbf{q} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \mathbf{0} > : 0 = 0$

(2) $<\mathbf{x} \leftarrow \mathbf{s}(\mathbf{x}'), \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y}')>:\forall x', y'\ \exists q', q'', r''\ \neg(s(y') = 0) \wedge (r'' = y') \Rightarrow <\mathbf{q} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \mathbf{s}(\mathbf{r}')>:$
$(x' < y' = T) \vee < \mathbf{q} \leftarrow \mathbf{s}(\mathbf{q}'), \mathbf{r} \leftarrow \mathbf{0} > : (x' = s(y') * q' + r'') \vee < \mathbf{q} \leftarrow \mathbf{s}(\mathbf{q}''), \mathbf{r} \leftarrow \mathbf{s}(\mathbf{r}'') > :$
$(x' = s(y') * s(q'') + r'' \wedge r'' < y' = T)$

(3) $<\mathbf{x} \leftarrow \mathbf{s}(\mathbf{x}'), \mathbf{y} \leftarrow \mathbf{s}(\mathbf{y}')>:\forall x', y'\ \exists q', q'', r''\ \neg(s(y') = 0) \wedge \neg(r'' = y') \Rightarrow <\mathbf{q} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \mathbf{s}(\mathbf{r}')>:$
$(x' < y' = T) \vee < \mathbf{q} \leftarrow \mathbf{s}(\mathbf{q}'), \mathbf{r} \leftarrow \mathbf{0} > : (x' = s(y') * q' + r'') \vee < \mathbf{q} \leftarrow \mathbf{s}(\mathbf{q}''), \mathbf{r} \leftarrow \mathbf{s}(\mathbf{r}'') > :$
$(x' = s(y') * s(q'') + r'' \wedge r'' < y' = T)$

(4) $< \mathbf{x} \leftarrow \mathbf{0}, \mathbf{y} \leftarrow \mathbf{0} > : \exists q', q'', r', r''\ \neg(0 = 0) \Rightarrow < \mathbf{q} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \mathbf{0} > : 0 = 0 \wedge 0 < 0 = T$
$\vee < \mathbf{q} \leftarrow \mathbf{s}(\mathbf{q}'), \mathbf{r} \leftarrow \mathbf{0} > : 0 = 0 \wedge 0 < 0 \vee < \mathbf{q} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \mathbf{s}(\mathbf{r}'') > : 0 = s(r'') \wedge s(r'') < 0$

(5) $<\mathbf{x} \leftarrow \mathbf{s}(\mathbf{x}'), \mathbf{y} \leftarrow \mathbf{0}>:\forall x'\ \exists q', q'', r', r''\neg(0 = 0) \Rightarrow <\mathbf{q} \leftarrow \mathbf{0}, \mathbf{r} \leftarrow \mathbf{0}>:s(x') = 0 \wedge 0 < 0 = T$

Da **(4)** und **(5)** redundante Klauseln sind, generiert der Algorithmus folgende Regeln:

$div(0, 0) = 0;$
$div(s(x'), 0) = 0;$

$rest(0,0) = 0$;
$rest(s(x'),0) = 0$.

Die Klausel **(1)** ist eine Tautologie. Daraus extrahiert der Algorithmus folgende Regeln:

$div(0, s(y')) = 0$ und

$rest(0, s(y')) = 0$.

Auf **(2)** bzw. **(3)** wurden die Regeln **subsumption_1** bzw. **subpsumtiom_3** angewandt. Gemäß der Extraktionsprozedur (Fall (d) bzw. (e)) werden dann die folgende definierenden Gleichungen erzeugt.

$rest(x', s(y')) = y' \Rightarrow div(s(x'), s(y')) = s(div(x', s(y')))$;
$\neg(rest(x', s(y')) = y') \Rightarrow div(s(x'), s(y')) = div(x', s(y'))$

$rest(x', s(y')) = y' \Rightarrow rest(s(x'), s(y')) = 0$;
$\neg(rest(x', s(y')) = y') \Rightarrow rest(s(x'), s(y') = s(rest(x', s(y')))$

Für die Funktionen div bzw. $rest$ für die gilt: $div(x, y) = q$ bzw. $rest(x, y) = r$ wurden folgende Gleichungen automatisch generiert:

$div(0,0) = 0$;
$div(s(x'),0) = 0$;
$div(0, s(y')) = 0$;
$rest(x', s(y')) = y' \Rightarrow div(s(x'), s(y')) = s(div(x', s(y')))$;
$\neg(rest(x', s(y')) = y') \Rightarrow div(s(x'), s(y')) = div(x', s(y'))$

$rest(0,0) = 0$;
$rest(s(x'),0) = 0$
$rest(0, s(y')) = 0$;
$rest(x', s(y')) = y' \Rightarrow rest(s(x'), s(y')) = 0$;
$\neg(rest(x', s(y')) = y') \Rightarrow rest(s(x'), s(y')) = s(rest(x', s(y')))$

Diese Funktionen lassen sich wie im Beispiel 29 so umwandeln, daß keine negierte Gleichung mehr in ihrer Prämisse vorkommen. Nach Theorem 5 sind sie terminierend, nicht überlappend, total und erfüllen die vorgegebene Spezifikationsklausel.

Beispiel 33 Wurzel einer natürlichen Zahl

Aus dem Beweis der folgenden Spezifikationsklausel (siehe Beispiel 20)

$$\forall x\ \exists w, r\ x = w * w + r\ \wedge r < s(w + w)$$

werden analog zu dem Division-Beispiel die folgenden Funktionen $wurzel$ und $rest$, für die $wurzel(x) = w$ bzw. $rest(x) = r$ gilt, automatisch generiert:

$wurzel(0) = 0;$
$rest(x') = wurzel(x') + wurzel(x') \Rightarrow wurzel(s(x')) = s(wurzel(x'));$
$\neg(rest(x') = wurzel(x') + wurzel(x')) \Rightarrow wurzel(s(x')) = wurzel(x')$

$rest(0) = 0;$
$rest(x') = wurzel(x') + wurzel(x') \Rightarrow rest(s(x')) = 0;$
$\neg(rest(x') = wurzel(x') + wurzel(x')) \Rightarrow rest(s(x')) = s(rest(x')).$

Nach Theorem 5 sind $wurzel$ und $rest$ terminierend, nicht überlappend, total und erfüllen die vorgegebene Spezifikation.

Kapitel 11

Zusammenfassung und Schlußbemerkungen

In dieser Arbeit wurde ein Verfahren vorgestellt, mit dem man rekursiv definierte Algorithmen aus Gültigkeitsbeweisen von Existenzformeln extrahieren kann. Das Verfahren beschränkt sich auf einen einfachen Formalismus und basiert auf Test-sets und einem Vereinfachungsmechanismus. Termersetzungen und logische Simplifikationen bilden den Kern dieses Vereinfachungsmechanismus, während Test-sets eine Beschreibung des initialen Modells einer Axiommenge darstellen. Mit anderen Worten: Ein Test-set ist eine endliche Menge irreduzibler Terme, mit der jeder irreduzible Grundterm (als Instanz eines ihrer Elemente) dargestellt werden kann. Wir verwenden Test-sets für die Konstruktion von Induktionsschemata einerseits und für die Widerlegung von Existenzformeln anderseits. Der Vereinfachungsmechanismus und die Initialisierung von Induktionsschritten sind mittels eines Inferenzsystems formalisiert. Darüber hinaus wurde ein Verfahren zur Extraktion rekursiv definierter Algorithmen aus den mit Hilfe des Inferenzsystems geführten Beweisen entwickelt. Die Korrektheit des Inferenzsystems und des Extraktionsalgorithmus garantieren, daß die synthetisierten Algorithmen bzgl. ihrer Spezifikationen korrekt sind. Zusätzlich wurde ein weiteres Verfahren eingefürt, mit dem man bestimmte fehlerhafte Problemspezifikationen widerlegen kann. Der Übersichtlichkeit halber wurden in dieser Arbeit relativ kleine Beispiele behandelt. Um die verwendete Methode aber auf die Praxis übertragen zu können, müßte eine gewaltige Menge an Wissen, das ein Mensch durch seine Programmiertätigkeit gesammelt hat, in einem System vorhanden sein. Überlegungen, wie ein solches Wissen in ein System integriert und effizient benutzt werden kann, wurden für unseren Ansatz der Programmsynthese noch nicht geführt.

Der vorgestellte Programmsyntheseansatz hat folgende Vorteile:

- In unserem Ansatz werden die zu beweisenden Existenzformeln nicht skolemisiert. Dadurch entfällt eine teuere Operation, die Beweise werden übersichtlicher, und die Signaturen der

betrachteten konstruktiven Spezifikationen bleiben unverändert.

- Lemmata bzw. Hilfssätze können beim Beweis einer Existenzformel automatisch generiert werden. Dadurch läßt sich die Synthese einfacher Programme bis in die allerletzten Details automatisieren.
- Induktionshypothesen werden am Anfang eines Beweises nicht explizit angegeben. Jede Instanz einer zu beweisenden Formel kann als Induktionshypothese verwendet werden, vorausgesetzt, sie ist bzgl. der Induktionsordnung kleiner als die aktuelle zu beweisende Formel.
- Fehlerhafte Spezifikationen von Funktionen können durch die Verwendung der Test-sets erkannt und entfernt werden. Dadurch wird der Suchraum reduziert, und die Komplexität der Beweise verkleinert.
- Neben der Korrektheit der generierten Funktionen bzgl. ihrer Spezifikationen sind ihre Totalität, Konfluenz und Terminierung gewährleistet, vorausgesetzt, eine zusätzliche Variablenbedingung ist erfüllt.
- Gleichungen, die in der Prämisse einer zu beweisenden Formel enthalten sind, und solche, die nicht als Regeln gerichtet werden können (z.B. Kommutativität der Addition) werden, ohne die Terminierung des Verfahrens zu gefährden, in den Termersetzungsvorgang miteinbezogen.

Kurzer Vergleich mit anderen Ansätzen bzw. Systemen:

In der Literatur sind diverse Zusammenfassungen und Vergleiche von Programmsyntheseansätzen zu finden (siehe z.B. [BIG83], [NEU92], [DOS93]), Daher sollen hier nur ausgewählte Ansätze bzw. Systeme kurz mit unserem Ansatz verglichen werden.

INKA

In dem Induktionsbeweiser INKA wurde ein Syntheseverfahren (siehe [BIU86]) integriert, das als Systemkomponente zum Beweis von Existenzaussagen dient. Obwohl dort der Aspekt der automatischen Softwareentwicklung im Hintergrund steht, ermöglicht das Synthesesystem die automatische Generierung einfacher Programme. Im Ansatz von S. Biundo wird eine zu beweisende Existenzformel zuerst skolemisiert, dann werden darauf Transformationsregeln wiederholt angewandt, bis eine Formelmenge entsteht, die sich in eine Menge von Definitionsformeln für die Skolemfunktionen und eine Menge von Restformeln zerlegen läßt. Die Restformeln müssen im Anschluß an den Syntheseprozeß noch bewiesen werden, um sicherzustellen, daß die generierten Programme ihre Spezifikationen erfüllen. Die Wahl des in einem Beweis zu verwendenden Induktionsaxioms beruht auf der Form der Funktionsdefinitionen, die in der Theoriespezifikation vorkommen. Diese Wahl ist der erste Schritt zum Beweis einer Existenzformel. Die verwendete Induktion ist dort explizit.

Z. Manna & R. Waldinger

Der klassiche Ansatz der deduktiven Programmsynthese ist der von Z. Manna und R. Waldinder (siehe [MAW84]). Dort wird ein Resolutionsbeweiser eingesetzt, um Programme zu generieren. Ähnlich zu unserem Ansatz entstehen die Programme durch Substitutionen, die während des Beweises auf die zu beweisende Formel angewandt wurden. Die Beweise und die Programmextraktion werden aber dort simultan geführt.

LEMMA

Der von J. Chazarain und E. Kounalis verfolgte Ansatz (siehe [CK94]) ist analog zu unserem. Dort werden allerdings nur Gleichungstheorien betrachtet, so daß nur rekursive Programme in Form von Gleichungen generiert werden können.

Coq, Nuprl, LEGO

Diese Beweissysteme (siehe [DOW], [CON86] und [LP92]) basieren auf konstruktiven Logiken und eignen sich deshalb auch für die Synthese von Programmen. Die Aufgaben, die dort dem Benutzer obliegen, sind meist langwierig. Extraktionsalgorithmen wurden in diesen Systemen bereits integriert, aber Verfahren zum Beweisen von Existenzformeln wurden bis dato nicht automatisiert.

Mögliche Erweiterungen:

Die folgenden Punkte stellen nur einen Teil aller möglichen Erweiterungen dar. Die Vorschläge werden deshalb an dieser Stelle nur kurz dargestellt.

- Es wäre wünschenswert, die in dieser Arbeit betrachteten konstruktiven Spezifikationen so zu erweitern, daß negierte Gleichungen in der Prämisse von Funktionsdefinitionen vorkommen können. Eine weitere sinnvolle Erweiterung wäre, nicht freie Konstruktoren einzuführen und definierte Funktionen in den formalen Argumenten von Funktionsdefinitionen zuzulassen. Dadurch wäre die Repräsentation von Programmen und Datenstrukturen wesentlich einfacher.

- Die Sukzedenzformel einer zu beweisenden Existenzformel sollte eine beliebige quantorfreie Formel der Prädikatenlogik erster Stufe sein. Damit könnte man mehr Funktionen spezifizeren und größere Beispiele behandeln.

- Anstelle von Test-sets wäre die Verwendung von Cover-sets, wie sie in [ZKK88] und in [BEN96] eingeführt wurden, von Vorteil. Dadurch wäre die Generierung allgemein rekursiver Funktionen möglich.

- Eine zusätzliche Erweiterung wäre Rippling [HUT89],[BUN93] anstelle der klassischen Termersetzung einzusetzen, um die Umformung eines Induktionsschlusses besser zu steuern.

Summary

In this thesis we presented a method for extracting recursive defined algorithms from existentially quantified formulas, being based on a simple formalism, test sets and a simplification strategy. Term rewriting and logical simplification represent the core of that simplification strategy and test sets the description of the initial model of a set of axioms. In other words, a test set is a finite set of irreducible terms, allowing every irreducible ground term to be instanciated with one of its elements. On the one hand, we use test sets to build induction schemes and, on the other hand, to disprove existentially quantified formulas. The simplificaton strategy and the initialization of induction steps are formalized by means of an inference system. Futhermore, we developed a procedure for extracting recursive defined algorithms from the proofs carried out by means of the inference system. The correctness of both the inference system and the extraction algorithm guarantees the correctness of the synthesized algorithms with respect to their specifications. Besides, another procedure for disproving certain incorrect existentially quantified formulas was introduced. For the sake of clarity, we delt with relatively small examples. In order to put our method into practice an enormous amount of knowlegde acquired in programming would be required as part of the system considered. As for our approach of program synthesis, is has not yet been considered how to integrate such knowledge in a system and how to use it efficiently.

The advantages of the approach presented here are as follows:

- By not skolemizing the existentially quantified formulas we can dispense with a costly operation, the proofs gain in clarity and the signatures of the constructive specifications remain unchanged.
- When proving an existantially quantified formula, lemmas can be automatically be generated. Thereby, the automatic synthesis of simple programs is facilitated to the utmost.
- There are no explicit induction hypotheses at the outset of a proof. Every instance of a formula to be proved can be used as an induction hypothesis on condition that it is smaller than the formula being proved with respect to the induction ordering.
- With the application of test sets allowing to detect and to remove incorrect specifications of functions, the search space is limited and the complexity of the proofs reduced.

- Besides the correctness of the generated functions with respect to their specifications, their totality, confluence and termination are equally guaranteed, assuming that an additional syntactical condition is met.

- Equations that are either part of the premiss of a formula to be proved or that cannot be transformed into rules (for example, commutativity of addition) can be included in a term rewriting process without jeopardizing the termination of the proof process.

Brief comparison with other approaches and systems respectively

As there is a vast amount of literature, summaries and comparisons of approaches to program synthesis (see [BIG83], [NEU92], [DOS93]) we will limit the following comparison to a selected range of approaches.

INKA

The induction prover **INKA** was integrated with a method of program synthesis which is used as a component to prove existentially quantified formulas. Even though automatic software development is not in the foreground the process of synthesis allows to automatically generate simple programms. In S. Biundo's approach (see [BIU86]) the existentially quantified formula to be proved is first skolemized. Then, transformation rules are repetitively applied to it until a set of formulas is generated that can be split up into a set of defined formulas for the skolem function and a set of rest formulas. Subsequent to the process of synthesis the rest formulas have yet to be proved in order to ensure that the generated programs fulfill their specifications. The choice of the adequate induction axiom is based on the form of the defining functions occurring in the theory specification. Choosing the appropriate induction axiom is the first step towards proving an existentially quantified formula.

Z. Manna & R. Waldinger

The classic approach of deductive program synthesis based on a resolution prover was developed by Z. Manna and R. Waldinger (see [MAW84]). Similar to our approach, the programs result from applying substitutions to the formulas to be proved but with the proofs and the extraction of the programs carried out simultaneously.

LEMMA

J. Chazarain and E. Kounalis' approach (see [CK94]) is analogous to ours generating, however, recursive programs in form of equations only as a result of merely considering equational theories.

Coq, Nuprl, LEGO

These proof systems (see [DOW], [CON86] and [LP92]) are based on constructive logics and as a consequence suitable for the synthesis of programs, yet, with the tasks of the user being mostly time-consuming. Extraction algorithms have already been integrated with these systems whereas

the methods of proving existentially quantified formulas have not been transformed into automatic processes yet.

Possible improvements and generalizations

As there is considerable potential for generalizing our approach the following suggestions only represent brief and concise descriptions of possible ideas.

- It is desirable that the constructive specifications delt with in this thesis should be generalized in a way that would allow the use of negated equations in the premiss of defined functions.
- To introduce constructors that are not free and to allow of defined functions as formal arguments of other defined functions would mean yet another sensible generalization which would facilitate the representation of programs and data structures to a considerable extent.
- With the conclusion of an existentially quantified formula to be a non-restricted quantifier-free formula of the first order predicate logic, a large amount of functions could be specified and, futhermore, larger examples considered.
- The application of cover sets instead of test sets as introduced in [ZKK88] and in [BEN96] would have the advantage of facilitating the generation of generally recursive functions.
- The application of Rippling (see [HUT89] and [BUN93]) instead of classic term rewriting would mean a further possibility to improve the control when transforming the conclusion of an induction proof.

Literaturverzeichnis

[AM95] J. Avenhaus and K. Madlener. Theorem proving in hierarchical clausal specifications. *SEKI-Report SR-95-14 (SFB)*, Fachbereich Informatik, Universität Kaiserlautern, 1995.

[AVE95] J. Avenhaus. *Reduktionssysteme*. Springer-Verlag, 1995.

[BAC88] Bachmair, L. Proof by consistency in equational theories. Proc. 3th Symposium on Logic in Computer Science, Edinburgh, 1988.

[BADP89] L. Bachmair, N. Dershowitz, and M. Plaisted. Completion without failure. In Hassan Ait-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures: Rewriting Techniques*, pages 1-30, New York, 1989.

[BAT85] J. L. Bates and R. L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113-136, 1985.

[BEC93] K. Becker. Proving ground confluence and inductive validity in constructor based equational specification. *TAPSOFT*, LNCS 668, pages 46-60, Springer-Verlag, 1993.

[BEN96] I. Bengeloune. Induction and synthesis by simplification. In P. Moukeli, editor,*Proc. of the 3rd African Conference on Research in Computer Science, Libreville (Gabon)*, 1996.

[BL93] E. Bevers and J. Lewi. Proving termination of (conditional) rewrite systems. *Acta Informatica*, 30:537-568, 1993.

[BIB84] W. Bibel and K. M. Hörnig. LOPS: a system based on a strategical approach to program synthesis. In A. Biermann, G. Guiho, and Yves Kodratoff, editors, *Automatic Program Construction Techniques*, pages 69-89. Macmillan Publ. Co., 1984.

[BIB93] W. Bibel and A. W. Biermann. Special issue: Automatic Programming – foreword of the guest editors. *Journal of Symbolic Computation*, 15 (5 & 6): 463-465, 1993.

[BIG83] A. W. Biermann and G. Guiho, editors, *Computer Program Synthesis Methodologies*, pages 1-49. D. Redel Publ. Co., 1983.

[BIHW86] S. Biundo, B. Hummel, D. Hutter, C. Walther: The Karlsruhe Induction Theorem Proving System. *Proc. of the 8th CADE*, Springer-Verlag, LNCS 230, 1986.

[BIU86] S. Biundo. A synthesis system mechanizing proofs by induction. *Proceeding of the 7th European Conference on Artificial Intelligence*, Brighton, Great Britain, 1986.

[BOU94] A. Bouhoula. SPIKE: a system for sufficient completeness and parameterized inductive proof. In A. Bundy, editor, *Proc. 12th International Conference on Automated Deduction, Nancy (France)*, LNAI 814, pages 836-840. Springer-Verlag, 1994.

[BOU97] A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation* 23(1):47-77, 1997.

[BR95] A. Bouhoula, M. Rusinowitch. Implicit induction in conditional theories. *J. Automated Reasoning* 14, pages 189-235, Kluwer Acad. Publ., 1995.

[BRK95] A. Bouhoula, M. Rusinowitch, and E. Kounalis. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631-668, 1995.

[BOY79] R. S. Boyer and J. S. Moore. A computational logic. Academic Press, New York, 1979.

[BR91] F. Bronsard and U. S. Reddy. Conditional rewriting in focus. In S. Kaplan and M. Okada, editors, *Conditional and Typed Rewriting Systems, Second Intern. CTRS Workshop*, pages 2-13, Springer-Verlag, 1991.

[BRH94] F. Bronsard, U. S. Reddy, and R. W. Hasker. Induction using term orderings. *Proc. 12th CADE*, LNCS vol. 814, 1994.

[BRO83] M. Broy. Programm construction by transformation: A family tree of sorting programs. In A. W. Biermann and G. Guiho, editors, *Computer Program Synthesis Methodologies*, pages 1-49. D. Redel Publ. Co., 1983.

[BUN93] A. Bundy, A. Stevens, F. van Hermelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence* 62:183-253, 1993.

[BUR77] R. M. Burstall and J. Darlington. A transformation system for recursive programms. *J. ACM 24*, pages 44-67, 1977.

[BUR69] R. M. Burstall. Proving propeties of programs by structural induction. *Computer Journal* 12(1), 41-48.

[CK94] J. Chazarain, E. Kounalis. Mechanizable inductive proofs for a class of $\forall\exists$ formulas Proc. 12th CADE, LNCS vol. 814, 1994.

[CON86] R. L. Constable et al. Implementing mathematics with the Nuprl proof development system. *Prentice-Hall, Englewood Cliffs*, 1986.

[DER82] N. Dershowitz. Orderings for Term rewriting systems. *Theoritical Computer Science* 17, 1982.

[DER85] N. Dershowitz. Termination. In *Proc. Rewriting Techniques and Applications*, Dijon, LNCS 202, Springer-Verlag, 1985.

[DER87] N. Dershowitz. Computing with rewrite systems. *Information and Control*, 65, 122-157, 1987.

[DM79] N. Dershowitz and Z. Mana. Proving termination with multiset ordering. *CACM 22*, pages 465-476, 1979.

[DOS87] N. Dershowitz, M. Okada, and G. Sivakumar. Canonical conditional rewrite systems. In *Proc. 9th International Conference on Automated Deduction, Argonne (Illinois, USA)*, LNCS 310, Springer-Verlag, 1988.

[DOS93] N. Dershowitz and U. S. Reddy. Deductive and inductive synthesis of equational programs. *J. of Symbolic Computation* 15, 467-494., 1993.

[DOW] G. Dowek et al. The Coq proof assistant user's guide, version 5.6. *Rapport Technique* 134, INRIA, 1991.

[DRO84] K. Drosten. Towards executable specifications using conditional axioms. *Proc. 1st STACS*. In M. Fontet and K. Mehlhorn, editors, LNCS 166, Springer-Verlag, 1984.

[FAY79] M. Fay. First order unification in equational theories. *Proc. 4th CADE* (W. Bibel and R. Kowalski, eds.), LNCS 874, Springer-Verlag, 1979.

[FRA94] U. Fraus. Mechanizing inductive theorem proving in conditional theories. PhD thesis, Universität Passau, 1994.

[FRI86] L. Fribourg. A strong restriction of the inductive completion procedure. IN *Proc. 13th Int. Coll. on Automata Languages and Programming*, pages 105-115, LNCS 226, Springer-Verlag, 1986.

[GAN87] H. Ganzinger. A completion procedure for conditional equations. In S. Kaplan, J.-P. Jouannaud (eds) *Proc. of Conditional Term Rewriting Systems*. LNCS 308, pages 62-83, 1987.

[GG88] S. J. Garland and J. V. Guttag. Inductive methods reasoning about abstract data types. In *ACM Symp. on Princ. of Program. Languages*, pages 219-228, ACM, 1988.

[GIE96] J. Giesl. Automatisierung von Terminierungsbeweisen für rekursiv definierte Algorithmen. *DISKI96, Dissertation zur Künstlichlen Intelligenz*. Infix 1995.

[HK88] D. Hofbauer and R. D. Kutsche. Proving inductive theorems based on term rewriting systems. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proc. 1st International Worshop on Algebraic and Logic Programming*, pages 180-190. Akademic Verlag, 1988.

[HOF93] Hoffmann B., B. Krieg-Brückner, B. (eds.): PROgram development by SPECification and TRAnsformation, Part I: Methodology, Part II: Language Family, Part III: System. LNCS, 1993.

[HUH82] G. Huet and J. M. Hullot. Proofs by induction in equational theories with constructors. Journal of Comp. and System Sciences, 25(2), 1982.

[HUHO80] G. Huet and D. C. Oppen. Equations and rewrite rules: a survey. *Formal Language Theory: Perspectives and Open Problems*, (R. Bood, ed.), Academic press, New York, 1980.

[HUT89] D. Hutter. Guiding inductive proofs. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449, pages 147-161.

[JK86] J.-P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proc. 1st IEEE Symposium on Logic in Computer Science, Cambridge (Mass. USA)*, pages 358-366, 1986.

[JW86] J. P. Jouannaud and B. Waldmann. Reductive conditional term rewriting systems. In M. Wirsing, editor, *3rd IFIP Conf. on Formal Description of Programming Concepts, Ebberup, Denmark*, Elvesier Science Publishers, Amsterdam, 1980.

[KAP84] S. Kaplan. Conditional rewrite rules. *TCS* 33 (2,3), 1984.

[KAP85] S. Kaplan. Fair conditional term rewriting systems: unification, termination and confluence. In *Recent Trends in Data Type Specification*. H.-J. Kreowski ed., Informatik-Fachberichte 116, Springer-Verlag, 1985.

[KNZ86] D. kapur, P. Narendran, H. Zhang. Proof by induction using test sets. *Prc. 8th CADE*, LNCS 230, pages 99-117, 1986.

[KB70] D.E. Knuth and P. Bendix. Single word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages263-297. Pergamon, Oxford, 1970.

[KOU90] E. Kounalis. Pumping lemmas in tree languages. In *Mathematical Foundation of Computer Science*, 1990.

[KR90] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. *Bulletin of European Association for Theoritical Computer Science*, 41:216-226, 1990.

[KRU90] E. Kounalis and M. Rusinowitch. A proof system for conditional algebraic specifications. In S. Kaplan and M. Okada, editors, *Conditional and Typed Rewriting Systems*, pages 51-63. Springer LNCS 516, 1990.

[LEW96] J. Loeckx, H-D. Ehrig and M. Wolf. *Specification of Abstract Data Types*. J. Wiley & Sons and B.G. Teubner, 1996.

[LS87] J. Loeckx and K. Sieber. *The foundation of Program Verification*. J. Wiley & Sons and B.G. Teubner, 1987.

[LP92] Z. Luo and R. Pollack. Lego proof development system: User's manual. *Technical Report* LFCS-92-211, University of Edinburgh., 1992.

[MAG92] L. Magnusson. The new implementation of ALF. In B. Nordsröm, K. Peterson, and G. Plotkin, editors, *Proc. of the 1992 Workshop on Types for Proofs and Programs*, 1992.

[MAW84] Z. Manna and R. Waldinger. A deductive approach to program syntesis. In A. Biermann, G. Guiho, and Yves Kodratoff, editors, *Automatic Program Construction Techniques*, pages 38-63. Macmillan Publ. Co., 1984.

[MID92] A. Middeldorp. Completeness of combinations of conditional constructor systems. *Conditional Term Rewriting Systems*. 3rd Int. Workshop, CTRS-92, Pont-à-Mousson. LNCS 856, France, 1992.

[MOSR] C. Mohan and M. Srivas. Conditional specifications with inequational assumptions. In S. Kaplan, J.-P. Jouannaud (eds) *Proc. of Conditional Term Rewriting Systems*. LNCS 308, pages 161-178, 1987.

[MUG92] S. Muggleton. Inductive logic Programming. In *Inductive Logic Programming*. Academic Press, 1992.

[MUS80A] D. R. Musser. On proving inductive properties of abstract data types. In Proc. 7th ACM Symp. on Principles of Programming Languages,154-162. Las Vegas, Nevada, 1980

[MUS80B] D. R. Musser. Abstract Data Type Specification in the AFFIRM System. *IEEE Transactions on Software Ingineering* SE 6, 1980.

[NEU92] G. Neugebauer. Pragmatische programmsynthese, volume 18 of *DISKI, Dissertation zur Künstlichen Intelligenz*. Infix, 1992.

[OKA87] M. Okada. A logocal analysis on theory of conditional rewriting. In S. Kaplan, J.-P. Jouannaud (eds) *Proc. of Conditional Term Rewriting Systems*. LNCS 308, pages 179-196, 1987.

[PAD88] P. Padawitz. *Computing in Horn Clause Theories*. Springer-Verlag, 1988.

[PAD96] P. Padawitz. Inductive theorem proving for design specifications. *Journal of Symbolic Computation* 21, 19996, 41-99.

[PAR90] H. Partsch. *Specification and Tranformation of Programs – A Formal Approach to Software Development*. Springer-Verlag, 1990.

[PEA89] G. Peanno: *Arithmetices principia, nova methodo exposita*, Turin, 1889.

[PE82] U. Pletat und G. Engels. Operational semantics of algebraic specifications with conditional equations. *7th CAAp, Lille*, 1982.

[RED88] U. S. Reddy. Rewriting techniques for programm synthesis. In N. Dershowitz, editor, *Rewriting Techniques and Appl.*, pages 388-403, LNCS 355, Springer-Verlag, 1989.

[RED89] U. S. Reddy. Term rewriting induction. Proc. 10th CADE, LNCS vol. 489, 1989.

[REM82] J.-L. Remy. Etudes des systèmes de reécriture conditionelles et applications aux types abstraits algébriques. Thèse d'etat, Université de Nancy I, Nancy, 1982.

[SIE89] J. Siekmann. Universal unification. *Proc. 7th CADE* (R.E. Shostak, ed.), LNCS 170, Springer-Verlag, 1984.

[SMI85] D. R. Smith. Top-down synthesis of divide-and-conquer algorithm. *Artificial Intelligence* 27, 43-96, 1985.

[SUM77] P. D. Summers. A methodology for LISP programm constrution from examples. *J. Assoc. Comp. Mach.* 24, 161-175, 1977.

[WYS87] F. Wysotzki. Program synthesis by hierarchical Planning. I. P. Jorrand and V. Sgurev, editors, *Artificial Intelligence: Methodology, Systems, Applications*, pages 3-11. Elsevier Science, Amsterdam, 1987.

[ZHA93] H. Zhang. Implemeting contextual rewriting. *Proc. of the 3rd CTRS Workshop*, LNCS 656, Springer-Verlag, 1993.

[ZKK88] H. Zhang, D. Kapur and M. S. Krishnamoorthy (1988). A Mechanizable Induction Principle for equational specifications In E.Lusk and R. Overbeek, editors, 9th CADE Conf., LNCS vol 310.

[ZR85] H. Zhang and J.-L. Remy. Contextual rewriting. *1st Int. Conf. on Rewriting Techniques and Application*. J.-P. Jouannaud, ed., LNCS 202, Springer-Verlag, 1985.

Index