Integration von Entscheidungsverfahren in den induktiven Theorembeweiser QuodLibet

René Rondot

Oktober 2004

Diplomarbeit

Technische Universität Kaiserslautern

Betreuer:
Prof. Dr. Jürgen Avenhaus
Dipl.-Inform. Tobias Schmidt-Samoa

Inhaltsverzeichnis

0. Motivation und Zielsetzung

Theorembeweiser für die induktive Theorie haben bei der Beweisfindung einen sehr großen Suchraum zu untersuchen. Dies liegt daran, dass im Vergleich zu deduktiven Systemen Induktionsordnungen bestimmt und Lemmata spekuliert werden müssen. Zudem ist das Problem, ob eine Aussage induktiv gültig ist, nicht rekursiv aufzählbar. Derzeit existierende Systeme sind daher in der Regel nur interaktiv einsetzbar und benötigen einen sehr hohen Rechenaufwand. Dies führt dazu, dass ihre Bedienung kompliziert und zeitaufwändig ist.

Für viele spezielle Anwendungsgebiete existieren jedoch Verfahren, die es ermöglichen, deutlich schneller und vollkommen automatisch zu entscheiden, ob eine Aussage gültig ist oder nicht. Solche sogenannten Entscheidungsverfahren existieren beispielsweise für Teile der Arithmetik der natürlichen Zahlen und die Theorie der Listen. Es scheint daher sinnvoll, solche Entscheidungsverfahren für häufig auftretende Theorien mit einem induktiven Theorembeweiser zu kombinieren. Das Entscheidungsverfahren kann immer dann genutzt werden, wenn während eines Beweises Probleme der entsprechenden Theorie auftreten. Dadurch sollte es möglich sein, die Komplexität eines allgemeinen induktiven Theorembeweisers für diese Probleme zu vermeiden und durch die Verwendung eines spezialisierten Verfahrens schneller zum Ziel zu kommen.

Ziel dieser Arbeit ist es, die Integration von Entscheidungsverfahren in einen induktiven Theorembeweiser am Beispiel der linearen Arithmetik über den natürlichen Zahlen zu untersuchen. Für diese Theorie sind bereits seit längerer Zeit einige Entscheidungsverfahren bekannt. Ein solches Entscheidungsverfahren soll in den induktiven Theorembeweiser QuodLibet integriert werden, der an der Universität Kaiserslautern entwickelt wurde [Küh00]. Dabei wird untersucht, wie diese Integration möglich ist und welche Vorteile und Nachteile sie bringt. Der erhoffte positive Effekt sollte vor allem eine deutliche Reduzierung der Beweiskomplexität sowie der notwendigen manuellen Eingriffe in den Beweisvorgang sein. Dabei ist insbesondere zu erwarten, dass die Zahl der Lemmata, die für einen Beweis zu einer Spezifikation mit natürlichen Zahlen benötigt werden, stark verringert werden kann. Als Nachteil ist ein relativ hoher Aufwand für die Zusammenarbeit des Entscheidungsverfahrens mit dem restlichen System zu erwarten. Diese Zusammenarbeit ist notwendig, da das Entscheidungsverfahren auch für eine erweiterte Theorie mit definierten Operatoren verwendet werden soll. Zudem wird das Gesamtsystem durch die Integration des Entscheidungsverfahrens größer und somit für den Benutzer unübersichtlicher. Kann der Beweisvorgang wie erhofft durch die Integration des Entscheidungsverfahrens jedoch stärker automatisiert werden, fällt dieser Nachteil weniger ins Gewicht.

Im ersten Kapitel dieser Arbeit werden zunächst die Grundlagen erläutert. Dazu gehören die theoretischen Grundlagen über die Syntax und Semantik der verwen-

deten Spezifikationen. Außerdem wird der induktive Theorembeweiser QuodLibet in seiner bisherigen Form beschrieben. In diesen Theorembeweiser soll ein Entscheidungsverfahren für lineare Arithmetik integriert werden, weshalb die beiden letzten Abschnitte des ersten Kapitels allgemeine Grundlagen zu Entscheidungsverfahren und eine Beschreibung der linearen Arithmetik enthalten.

Wie die lineare Arithmetik in den Theorembeweiser QuodLibet integriert werden kann, ist Thema des zweiten Kapitels. Da QuodLibet bisher keine Strukturen für natürliche Zahlen enthielt, wird zunächst beschrieben, wie diese in das System integriert werden können. Anschließend wird erläutert, wie die Operatoren und Prädikate definiert sind und wie diese in das System integriert werden. Die beiden letzten Abschnitte des zweiten Kapitels sind der Polynomdarstellung von Termen und Literalen gewidmet. Hier wird beschrieben, wie Terme und Literale der linearen Arithmetik in eine Polynom-Normalform überführt werden können, mit der dann effizient ein Entscheidungsverfahren ausgeführt werden kann.

Die Integration des Entscheidungsverfahrens selbst wird dann im dritten Kapitel behandelt. Hier wird zunächst eine Menge neuer Inferenzregeln vorgestellt, mit deren Hilfe die einzelnen Schritte des Entscheidungsverfahrens realisiert werden. Um diese Inferenzregeln so in den formalen Rahmen des Systems QuodLibet integrieren zu können, dass zentrale Ergebnisse über Eigenschaften des Systems gültig bleiben, wird die Korrektheit und Sicherheit dieser Inferenzregeln bewiesen. Im Anschluss wird beschrieben, wie diese neuen Inferenzregeln in die Taktiken des Theorembeweisers eingearbeitet werden können.

Das vierte Kapitel ist schließlich der Auswertung der Ergebnisse der in den beiden vorhergehenden Kapitel beschriebenen Änderungen gewidmet. Hier wird aufgrund von empirisch erhobenen Daten ausgewertet, wie sich die Integration des Entscheidungsverfahrens bemerkbar macht und wie stark die erhofften positiven Effekte durch die negativen Effekte beeinträchtigt werden.

Zuletzt werden im fünften Kapitel die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf weitere Entwicklungsmöglichkeiten gegeben, die sich basierend auf den erreichten Ergebnissen aufzeigen.

1. Grundlagen

1.1. Theoretische Grundlagen und Schreibweisen

Die dieser Arbeit zugrunde liegenden theoretischen Konzepte und Schreibweisen basieren auf [Küh00] und [Ave95]. Hier wird nur eine kurze Übersicht über die für diese Arbeit wichtigen Konzepte gegeben, weitere Details und Beweise einiger Aussagen können in den beiden angegebenen Werken nachgelesen werden.

1.1.1. Syntax von Spezifikationen mit Konstruktoren

Eine *Signatur* $sig = (S, F, \alpha)$ besteht aus einer Menge von *Sortensymbolen* S, einer Menge von *Funktionssymbolen* F und einer *Stelligkeitsfunktion* α, die F auf S^+ abbildet. Zu jeder Signatur $sig = (S, F, \alpha)$ wird für jede Sorte $s \in S$ eine Menge von *Variablensymbolen* V_s definiert. Diese Mengen sind paarweise disjunkt und zudem disjunkt zu F. Die Menge aller Variablensymbole ist $V = (V_s)_{s \in S}$. Zusammen mit diesen Variablenmengen beschreibt die Signatur wie üblich die Menge der korrekten Terme $\mathcal{T}(sig, V)_s$ einer Sorte s sowie die Menge aller korrekten Terme $\mathcal{T}(sig, V) = (\mathcal{T}(sig, V)_s)_{s \in S}$. Die Menge aller variablenfreien *Grundterme* wird mit $\mathcal{GT}(sig) = (\mathcal{GT}(sig)_s)_{s \in S}$ bezeichnet.

Eine Position p in einem Term t ist eine Folge von natürlichen Zahlen. Mit t/p wird der Teilterm von t an der Position p bezeichnet und $t[u]_p$ bezeichnet den Term, der aus t durch Ersetzen des Teilterms t/p durch u entsteht. Mit $Pos(t)$ wird die Menge aller Positionen in t bezeichnet.

Ein *Gewicht* über einer Signatur sig und einer Variablenmenge V ist eine Folge von Termen $(t_1, \ldots, t_k)$, so dass $0 \leq k \leq k_0$ für einen festen Wert k_0 und $t_1, \ldots, t_k \in \mathcal{T}(sig, V)$.

Eine *Gleichung* ist ein Term-Paar $t_1 = t_2$, so dass $t_1, t_2 \in \mathcal{T}(sig, V)_s$ für eine Sorte $s \in S$. Ein *Ordungsatom* ist ein Ausdruck der Form $w_1 < w_2$, wobei w_1 und w_2 Gewichte sind. Ein *Definiertheitsatom* ist ein Ausdruck der Form $\mathrm{def}\,(t)$, wobei t ein Term ist. Ein *Atom* ist eine Gleichung, ein Ordnungsatom oder ein Definiertheitsatom. Ein *positives Literal* ist ein Atom und ein *negatives Literal* ein *negiertes Atom* in der Form $\neg A$, wobei A ein Atom ist. Ein *Literal* λ ist ein positives oder negatives Literal. Das *Komplement* $\overline{\lambda}$ eines positiven Literals λ ist $\neg\lambda$ und das Komplement $\overline{\neg\lambda}$ eines negativen Literals $\neg\lambda$ ist λ. Die Schreibweise $t_1 \doteq t_2$ steht für $t_1 = t_2$ oder $t_2 = t_1$ und analog steht $t_1 \dot{\neq} t_2$ für $t_1 \neq t_2$ oder $t_2 \neq t_1$.

Eine *Klausel* ist eine (möglicherweise leere) Folge von Literalen $\lambda_1 \ldots \lambda_n$, zur besseren Lesbarkeit und um den disjunktiven Charakter der Klauseln hervorzuheben

meist geschrieben als $\lambda_1 \vee \ldots \vee \lambda_n$. Die leere Klausel wird als $\square$ geschrieben. Eine *bedingte Gleichung* ist ein Ausdruck der Form $l = r \leftarrow \Delta$ wobei Δ eine (möglicherweise leere) Folge von Literalen, den sogenannten *Bedingungsliteralen*, ist. Die *Klauseldarstellung* der bedingten Gleichung $l = r \leftarrow \lambda_1 \ldots \lambda_n$ ist $(l = r)\overline{\lambda_1} \ldots \overline{\lambda_n}$.

Für Ausdrücke (das heißt Terme, Literale, Klauseln oder bedingte Gleichungen) $e_1, \ldots, e_n$ bezeichnet $Var(e_1, \ldots, e_n)$ die Vereinigung der Mengen der Variablen, die in den Ausdrücken $e_1, \ldots, e_n$ vorkommen. Die Definition von Positionen wird auf Literale fortgesetzt, indem bei Definiertheitsliteralen die Positionen des Terms verwendet werden und bei Literalen der Form $l \;\; op \;\; r$ die erste Stelle der Position bezeichnet, ob die restliche Position auf den linken Term (1) oder auf den rechten Term (2) bezogen ist. Dementsprechend werden für ein Literal analog zu Termen die Bezeichnungen λ/p, $\lambda[u]_p$ und $Pos(\lambda)$ verwendet.

Eine *Spezifikation mit Konstruktoren* $spec = (sig, C, E)$ besteht aus einer Signatur sig, einer Menge der aus den Funktionssymbolen F ausgezeichneten *Konstruktorsymbole* C und einer Menge von *bedingten Gleichungen* E, den sogenannten *definierenden Gleichungen*. Die Konstruktoren C heißen *frei*, wenn jede definierende Gleichung mindestens ein Nicht-Konstruktorsymbol außerhalb der Bedingungsliterale enthält. Entsprechend der Unterteilung der Funktionssymbole in die Konstruktorsymbole C und die Symbole für definierte Operatoren $D = F \setminus C$ wird auch die Menge der Variablensymbole V disjunkt in sogenannte *Konstruktorvariablen* $V^C = (V_s^C)_{s \in S}$ und *generelle Variablen* $V^G = (V_s^G)_{s \in S}$ unterteilt, so dass gilt $V = V^C \cup V^G$ und $V^C \cap V^G = \emptyset$.

Eine Substitution $\sigma : V \longrightarrow \mathcal{T}(sig, V)$ wird *Konstruktorsubstitution* genannt, falls $\sigma(V^C) \subseteq \mathcal{T}(sig^C, V^C)$ und σ wird *induktive Substitution* genannt, falls $\sigma(V^C) \subseteq \mathcal{GT}(sig^C, V^C)$ und $\sigma(V^G) \subseteq \mathcal{T}(sig, V^G)$.

Die Intention dieser Definitionen ist es, durch eine Spezifikation mit Konstruktoren einen Datentyp zu spezifizieren, dessen Daten die Konstruktorgrundterme sind und auf dem durch die Gleichungen E Operatoren definiert werden. Dies wird durch die folgende Semantik formalisiert.

1.1.2. Semantik von Spezifikationen mit Konstruktoren

Eine *sig-Algebra* $\mathcal{A}$ ist ein Tupel $(A, F^{\mathcal{A}})$, wobei $A = (A_s)_{s \in S}$ eine (nicht-leere) Trägermenge für die Sorten der Signatur und $F^{\mathcal{A}} = (f^{\mathcal{A}})_{f \in F}$ eine Menge von Funktionen passender Stelligkeit ist, die die Funktionssymbole der Signatur interpretieren.

Ein *sig-Homomorphismus* $h : \mathcal{A} \longrightarrow \mathcal{B}$ zwischen zwei *sig*-Algebren $\mathcal{A}$ und $\mathcal{B}$ ist eine Familie $h = (h_s)_{s \in S}$ von Funktionen $h_s : A_s \longrightarrow B_s$, so dass für alle $a_i \in A_{s_i}$ gilt:

$$h_s(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

Mit $eval^{\mathcal{A}}$ wird der eindeutige *sig*-Homomorphismus von $\mathcal{GT}(sig)$ auf eine *sig*-Algebra $\mathcal{A}$ bezeichnet, der durch

$$eval^{\mathcal{A}}(f(t_1, \ldots, t_n)) = f^{\mathcal{A}}(eval^{\mathcal{A}}(t_1), \ldots, eval^{\mathcal{A}}(t_n))$$

für alle $f \in F$ und $t_i \in \mathcal{GT}(sig)_{s_i}$ definiert ist. Statt $eval^{\mathcal{A}}(t)$ wird üblicherweise $t^{\mathcal{A}}$ geschrieben.

Definition 1.1 *Das* Datenredukt *einer sig-Algebra $\mathcal{A} = (A, F^{\mathcal{A}})$ ist die sig^C-Algebra $\mathcal{A} = (A^C, C^{\mathcal{A}^C})$, die folgende Bedigungen erfüllt:*

- *Für jedes $s \in S$ ist $A^C_s = \{t^{\mathcal{A}} \in A_s \mid t \in \mathcal{GT}(sig^C)_s\}$.*
- *Für jedes $c \in C$ und jedes $a_i \in A^C_{s_i}$ gilt $c^{\mathcal{A}^C}(a_1, \ldots, a_n) = c^{\mathcal{A}}(a_1, \ldots, a_n)$, wobei $\alpha(c) = s_1 \ldots s_n s$.*

Zur Auswertung von Ordnungsliteralen wird die Ordnung $<^{lex}_{\mathcal{A}}$ verwendet, die die strikte Komponente der lexikographischen Erweiterung $\leq^{lex}_{\mathcal{A}}$ der wie folgt definierten Ordnung ist:

Definition 1.2 *Sei $sig = (S, F, \alpha)$ eine Signatur, so dass $C \subseteq F$ eine Konstruktormenge für sig ist und sei $\mathcal{A} = (A, F^{\mathcal{A}})$ eine sig-Algebra. Die mit $\mathcal{A}$ assoziierte Relation $\leq_{\mathcal{A}}$ ist definiert auf A durch $a_1 \leq_{\mathcal{A}} a_2$ genau dann, wenn*

- *$a_1 = a_2$ oder*
- *es gibt $t_1, t_2 \in \mathcal{GT}(sig^C)$, so dass $t_i^{\mathcal{A}} = a_i$ für $i = 1, 2$ und $|t_1| < |t_2|$.*

Die Modellsemantik einer Spezifikation mit Konstruktoren kann dann wie folgt definiert werden:

Definition 1.3 *Sei sig eine Signatur und $\mathcal{A} = (A, F^{\mathcal{A}})$ eine sig-Algebra.*

- *Sei $X \subseteq V$. Ein* Belegung *von X in $\mathcal{A}$ ist eine Funktion $\varphi : X \longrightarrow A$, so dass $\varphi(x) \in A^C_s$ für jedes $x \in X \cap V^C_s$ und $\varphi(x) \in A_s$ für jedes $x \in X \cap V^G_s$. Mit $eval^{\mathcal{A}}_{\varphi}$ wird der eindeutige sig-Homomorphismus von $\mathcal{T}(sig, X)$ nach A bezeichnet, der φ erweitert. Weiterhin wird $eval^{\mathcal{A}}_{\varphi}$ so erweitert, dass für ein Gewicht $w = (t_1, \ldots, t_k)$ gilt: $eval^{\mathcal{A}}_{\varphi}(w) = (eval^{\mathcal{A}}_{\varphi}(t_1), \ldots, eval^{\mathcal{A}}_{\varphi}(t_k))$.*
- *Sei φ eine Belegung von V in $\mathcal{A}$. Dann* erfüllt *$\mathcal{A}$*
  - *eine Gleichung $t_1 = t_2$* mit *φ, falls $eval^{\mathcal{A}}_{\varphi}(t_1) = eval^{\mathcal{A}}_{\varphi}(t_2)$,*
  - *ein Ordnungsatom $w_1 < w_2$ mit φ, falls $eval^{\mathcal{A}}_{\varphi}(w_1) <^{lex}_{\mathcal{A}} eval^{\mathcal{A}}_{\varphi}(w_2)$ und*
  - *ein Definiertheitsatom $def(t)$ mit φ, falls $eval^{\mathcal{A}}_{\varphi}(t) \in A^C_s$.*

  $\mathcal{A}$ erfüllt ein negatives Literal $\neg\lambda$ mit φ falls $\mathcal{A}$ nicht λ mit φ erfüllt. Weiterhin erfüllt $\mathcal{A}$ eine Klausel Γ mit φ falls ein Literal in Γ existiert, das $\mathcal{A}$ mit φ erfüllt.
- *Eine Klausel Γ ist gültig in $\mathcal{A}$, falls $\mathcal{A}$ die Klausel Γ mit jeder Belegung von V in $\mathcal{A}$ erfüllt (Schreibweise: $\mathcal{A} \models \Gamma$). Sei K eine Klasse von sig-Algebren und E eine Klauselmenge. Man schreibt $K \models E$ genau dann, wenn $\mathcal{A} \models \Gamma$ für jede sig-Algebra $\mathcal{A} \in K$ und jede Klausel $\Gamma \in E$.*

Definition 1.4 *Sei $spec = (sig, C, E)$ eine Spezifikation mit Konstruktoren. Eine sig-Algebra $\mathcal{A}$ heißt* (sig-)Modell *von spec falls die Klauseldarstellung jeder bedingten Gleichung E in $\mathcal{A}$ gültig ist. Die Klasse aller sig-Modelle von spec wird mit $Mod(spec)$ bezeichnet.*

Definition 1.5 *Sei $spec = (sig, C, E)$ eine Spezifikation mit Konstruktoren. Ein sig-Modell $\mathcal{A}$ von spec wird ein* Datenmodell *genannt, falls für alle Konstruktor-Grundterme $t_1, t_2 \in \mathcal{GT}(sig^C)$, $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$ impliziert, dass $Mod(spec) \models t_1 = t_2$. Die Klasse aller Datenmodelle von spec wird mit $DMod(spec)$ bezeichnet.*

Definition 1.6 *Ein Spezifikation spec heißt* zulässig, *falls $DMod(spec) \neq \emptyset$.*

Eine Klausel Γ heißt induktiv gültig *oder* induktives Theorem *bezüglich einer zulässigen Spezifikation mit Konstruktoren spec, falls $DMod(spec) \models \Gamma$.*

Ein induktiver Theorembeweiser hat nun zur Aufgabe, die induktive Gültigkeit einer gegebenen Klausel nachzuweisen. Das folgende Lemma bietet hierfür ein nützliches Kriterium für die Gültigkeit einer Klausel in einer sig-Algebra.

Lemma 1.1 *(nach [Küh00, S. 34]) Wenn $\mathcal{A}$ eine sig-Algebra und Γ eine Klausel ist, dann gilt $\mathcal{A} \not\models \Gamma$ genau dann, wenn es eine induktive Substitution σ und eine Belegung φ von V^G in $\mathcal{A}$ gibt, so dass $\mathcal{A}$ nicht $\Gamma\sigma$ mit φ erfüllt. Das Tupel $(\Gamma, \sigma, \varphi)$ wird dann ein* $\mathcal{A}$-Gegenbeispiel *genannt.*

Basierend auf diesem Lemma ist es möglich, eine Klausel als induktiv gültig nachzuweisen, indem systematisch die Existenz eines Gegenbeispiels ausgeschlossen wird. Dieses Verfahren ist die Grundlage des induktiven Theorembeweisers QuodLibet.

1.2. Der Theorembeweiser QuodLibet

In den folgenden Abschnitten werden einige grundlegende Eigenschaften des Theorembeweisers QuodLibet beschrieben, die für die folgenden Kapitel wichtig sind. Eine detaillierte Darstellung des gesamten Systems findet sich in [Küh00] und [Kai02].

1.2.1. Anwendbarkeitsbereich und generelle Funktionsweise

QuodLibet ist ein Beweissystem zum Nachweis von induktiven Theoremen zu positiv/negativ bedingten Spezifikationen mit freien Konstruktoren. Dabei sind auch partiell spezifizierte Funktionen zugelassen. Die Zulässigkeit der Spezifikation, das heißt, die Tatsache, dass die Klasse der Datenmodelle der Spezifikation nicht leer ist, wird von QuodLibet automatisch anhand eines hinreichenden, syntaktischen Kriteriums überprüft.

Um ein induktives Theorem nachzuweisen, arbeitet QuodLibet nach dem Prinzip der Verkleinerung von Gegenbeispielen entsprechend einer wohlfundierten Ordnung. Diese wird solange durchgeführt, bis die Existenz oder Nicht-Existenz eines Gegenbeispiels evident wird. Dieses Verfahren ist sehr stark an der mathematischen

Vorgehensweise beim Finden von Induktionsbeweisen orientiert, die Fermat bereits 1659 unter dem Namen *descente infinie* beschrieb [Wir04].

Der Vorgang wird durch ein festes Inferenzsystem beschrieben, das gültige Schritte zur Verkleinerung von Gegenbeispielen und für die Erkennung der Nicht-Existenz eines Gegenbeispiels beschreibt. Jede Inferenzregel überführt ein Ziel in eine (möglicherweise leere) Menge von Teilzielen. Das Inferenzsystem enthält auch Inferenzregeln zum induktiven Anwenden von Zielen. Hierbei werden automatisch neue Teilziele erzeugt, die sogenannten induktiven Beweisverpflichtungen, die den Nachweis fordern, dass die Instanz, für die die Induktionsannahme angewandt wurde, in einer wohlfundierten *Induktionsordnung* kleiner als das ursprüngliche Beweisziel ist.

Die in QuodLibet verwendete semantische Induktionsordnung $\precsim_{\mathcal{A}}$ basiert darauf, jedem Beweisziel Γ ein Gewicht w zuzuordnen, das gewissermaßen die „Größe“ des Beweisziels angibt. Daher werden Beweisziele in der Form $\langle \Gamma\,;\ w\rangle$ angegeben, das heißt als Tupel einer Klausel Γ mit dem zugehörigen Gewicht w.

1.2.2. Inferenzsystem

Das Inferenzsystem ist der Kern von QuodLibet. Bisher enthielt QuodLibet 25 Inferenzregeln [Küh00]. Allgemein hat eine Inferenzregel des Kalküls von QuodLibet folgende Form:

$$\frac{\langle \Gamma\,;\ w\rangle}{\langle \Gamma_1\,;\ w_1\rangle \ldots \langle \Gamma_n\,;\ w_n\rangle} \qquad \begin{array}{l}\text{mit } \langle \Pi_1\,;\ \hat{w_1}\rangle^{U_1}, \ldots, \langle \Pi_k\,;\ \hat{w_k}\rangle^{U_k} \\ \text{falls } \textit{Anwendbarkeitsbedingung}\end{array}$$

wobei $n, k \in \mathbb{N}$ und $U_j \in \{\mathcal{I}, \mathcal{L}\}$ für $j = 1, \ldots, k$. Eine solche Inferenzregel kann verwendet werden, um ein Ziel $\langle \Gamma\,;\ w\rangle$ zu den neuen (Teil-)Zielen $\langle \Gamma_1\,;\ w_1\rangle \ldots \langle \Gamma_n\,;\ w_n\rangle$ abzuleiten, vorausgesetzt die Anwendbarkeitsbedingung ist erfüllt. Die Inferenzregel kann dazu weitere Ziele $\langle \Pi_1\,;\ \hat{w_1}\rangle^{U_1}, \ldots, \langle \Pi_k\,;\ \hat{w_k}\rangle^{U_k}$ nutzen, die auf $\langle \Gamma\,;\ w\rangle$ entweder als Induktionsannahme ($U_i = \mathcal{I}$) oder als Axiom oder Lemma ($U_i = \mathcal{L}$) angewandt werden. Eine Inferenzregel heißt *applikativ*, falls $k > 0$ und *nicht-applikativ*, falls $k = 0$. Applikative Inferenzregeln beinhalten die Anwendung einer definierenden Gleichung, eines Lemmas oder einer Induktionsvoraussetzung auf ein Ziel. Nicht-applikative Inferenzregeln sind z. B. das Reinigen einer Klausel oder die Aufspaltung nach Fällen.

Im folgenden Abschnitt werden zunächst zwei wichtige Eigenschaften der Inferenzregeln von QuodLibet vorgestellt. Im Anschluss daran werden drei Beispiele für Inferenzregeln beschrieben.

Eigenschaften von Inferenzregeln

Für das Gesamtsystem sind folgende Eigenschaften gefordert:

- Korrektheit: Wenn mit Hilfe des Systems ein abgeschlossener Beweisbaum gefunden wird, so ist das Ziel ein induktives Theorem.

- Widerspruchskorrektheit: Wenn ein Ziel mit dem System zu einem Widerspruch (in der Regel in Gestalt der leeren Klausel) abgeleitet werden kann, so ist das Ziel kein induktives Theorem.

Da Veränderungen am Zustand der Inferenzmaschine generell nur durch Anwendung einer der Inferenzregeln durchgeführt werden können, kann der Nachweis dieser beiden Eigenschaften auf lokale Eigenschaften der Inferenzregeln zurückgeführt werden. Wie in [Küh00] gezeigt wird, genügt für den Nachweis der Korrektheit des gesamten Kalküls der Nachweis der *Korrektheit* der einzelnen Inferenzregeln und für den Nachweis der Widerspruchskorrektheit der Nachweis der *Sicherheit* der Inferenzregeln.

Definition 1.7 *Eine Inferenzregel heißt* korrekt*, falls für jede zulässige Spezifikation spec und jede Instanz*

$$\frac{\langle \Gamma \ ; \ w \rangle}{\langle \Gamma_1 \ ; \ w_1 \rangle \ldots \langle \Gamma_n \ ; \ w_n \rangle} \qquad \textit{mit } \langle \Pi_1 \ ; \ \hat{w_1} \rangle^{U_1}, \ldots, \langle \Pi_k \ ; \ \hat{w_k} \rangle^{U_k}$$

der Inferenzregel sowie für jedes Datenmodell $\mathcal{A} \in DMod(spec)$ und jedes $\mathcal{A}$-Gegenbeispiel der Form $(\langle \Gamma \ ; \ w \rangle, \sigma, \varphi)$ eine der folgenden Aussagen gilt:

1. *Es gibt ein $i \in \{1, \ldots, n\}$ und ein $\mathcal{A}$-Gegenbeispiel der Form $(\langle \Gamma_i \ ; \ w_i \rangle, \tau, \psi)$, so dass $(\langle \Gamma_i \ ; \ w_i \rangle, \tau, \psi) \precsim_{\mathcal{A}} (\langle \Gamma \ ; \ w \rangle, \sigma, \varphi)$.*
2. *Es gibt ein $j \in \{1, \ldots, k\}$, so dass $U_j = \mathcal{L}$ und Π_j nicht induktiv gültig in spec ist.*
3. *Es gibt ein $j \in \{1, \ldots, k\}$ und ein $\mathcal{A}$-Gegenbeispiel der Form $(\langle \Pi_j \ ; \ \hat{w}_j \rangle, \tau, \psi)$, so dass $U_j = \mathcal{I}$ und $(\langle \Pi_j \ ; \ \hat{w}_j \rangle, \tau, \psi) \prec_{\mathcal{A}} (\langle \Gamma \ ; \ w \rangle, \sigma, \varphi)$.*

Sicherheit bedeutet, dass durch die Anwendung einer Inferenzregel keine neuen Gegenbeispiele eingeführt werden:

Definition 1.8 *Eine Inferenzregel heißt* sicher*, falls für jede zulässige Spezifikation spec und jede Instanz*

$$\frac{\langle \Gamma \ ; \ w \rangle}{\langle \Gamma_1 \ ; \ w_1 \rangle \ldots \langle \Gamma_n \ ; \ w_n \rangle} \qquad \textit{mit } \langle \Pi_1 \ ; \ \hat{w_1} \rangle^{U_1}, \ldots, \langle \Pi_k \ ; \ \hat{w_k} \rangle^{U_k}$$

der Inferenzregel gilt, dass aus der induktiven Gültigkeit von jeder der Klauseln $\{\Gamma, \Pi_1, \ldots, \Pi_k\}$ bezüglich spec die induktive Gültigkeit jeder der Klauseln $\{\Gamma_1, \ldots, \Gamma_n\}$ bezüglich spec folgt.

Nachdem nun diese Eigenschaften von Inferenzregeln definiert wurden, werden in den folgenden Abschnitten drei Beispiele für Inferenzregeln von QuodLibet beschrieben, die diese Eigenschaften haben. Eine Beschreibung der weiteren Inferenzregeln kann in [Küh00] und [Kai02] nachgelesen werden.

Beispiel: Literal Hinzufügen

Ein Beispiel für eine nicht-applikative Inferenzregel ist die Regel Literal Hinzufügen. Um diese zu beschreiben, wird die folgende Definition benötigt:

Definition 1.9 *Sei* $\Gamma = \lambda_1, \ldots, \lambda_n$ *eine Klausel mit* $n \in \mathbb{N}$. *Dann besteht die* aus der Klausel Γ resultierende Fallunterscheidung *aus den Klauseln* $\Lambda_1, \ldots, \lambda_n$ *und* Λ, *so dass*

1. $\Lambda_i = \overline{\lambda_i}, \lambda_{i-1}, \lambda_{i-2}, \ldots, \lambda_1$ *für* $i = 1, \ldots, n$
2. $\Lambda = \lambda_n, \ldots, \lambda_1$

Die Inferenzregel Literal Hinzufügen realisiert genau diese Fallunterscheidung und kann wie folgt formal beschrieben werden:

Literal Hinzufügen

$$\frac{\langle \Gamma \,;\, w \rangle}{\langle \Lambda_1, \Gamma \,;\, w \rangle \ldots \langle \Lambda_n, \Gamma \,;\, w \rangle \; \langle \Lambda, \Gamma \,;\, w \rangle}$$

falls $\Lambda_1, \ldots, \Lambda_n, \Lambda$ die Fallunterscheidung ist, die aus den Literalen $\lambda_1, \ldots, \lambda_n$ für $n > 0$ resultiert

Wendet man diese Inferenzregel mit den Literalen $\neg\mathrm{def}\,(f(x))\,, \neg\mathrm{def}\,(g(y))$ an, so kann beispielsweise das Ziel $\langle f(x) + g(y) = g(y) + f(x) \,;\, w \rangle$ zu den Zielen

$$\begin{gathered}\langle \mathrm{def}\,(f(x))\,, f(x) + g(y) = g(y) + f(x) \,;\, (f(x), g(y)) \rangle \\ \langle \mathrm{def}\,(g(y))\,, \neg\mathrm{def}\,(f(x))\,, f(x) + g(y) = g(y) + f(x) \,;\, w \rangle \\ \langle \neg\mathrm{def}\,(g(y))\,, \neg\mathrm{def}\,(f(x))\,, f(x) + g(y) = g(y) + f(x) \,;\, w \rangle\end{gathered}$$

abgeleitet werden.

Beispiel: Konstant Umschreiben

Eine weitere nicht-applikative Inferenzregel ist die Regel Konstant Umschreiben. Diese Inferenzregel nutzt ein negatives Literal der Form $t_1 \mathrel{\dot{\neq}} t_2$ aus, um in einem anderen Literal den Term t_1 durch t_2 zu ersetzen. Sie kann folgendermaßen beschrieben werden:

Konstant Umschreiben

$$\frac{\langle \Gamma, \lambda[t_1]_p, \Delta \,;\, w \rangle}{\langle \Gamma, \lambda[t_2]_p, \Delta \,;\, w \rangle}$$

falls

- $p \in Pos(\lambda)$ und $\lambda/p = t_1$ und
- es ein Literal $t_1 \mathrel{\dot{\neq}} t_2$ in Γ, Δ gibt.

Mit der Inferenzregel Konstant Umschreiben kann demnach beispielsweise das Ziel $\langle x \neq y \;\vee\; y \neq z \;\vee\; x = z \,;\, w \rangle$ in das Ziel $\langle x \neq y \;\vee\; x \neq z \;\vee\; x = z \,;\, w \rangle$ abgeleitet werden.

Beispiel: Nicht-induktive Termersetzung

Eine applikative Inferenzregel ist zum Beispiel die Regel Nicht-induktive Termersetzung. Um diese beschreiben zu können, wird zunächst folgende Definition benötigt:

Definition 1.10 *Die Menge von Definiertheitsbedingungen einer Substitution μ und einer Klausel Γ ist definiert als*

$$DefCond(\mu, \Gamma) = \{\neg def(x\mu) \mid x \in Var(\Gamma) \cap V^C \text{ und } x\mu \notin \mathcal{T}(sig^C, V^C)\}$$

Damit kann nun die Inferenzregel Nicht-induktive Termersetzung wie folgt formuliert werden:

Nicht-induktive Termersetzung

$$\frac{\langle \Gamma, \lambda, \Delta \,;\, w\rangle}{\langle \Lambda_1, \Gamma, \lambda, \Delta \,;\, w\rangle \ldots \langle \Lambda_n, \Gamma, \lambda, \Delta \,;\, w\rangle \; \langle \Lambda, \Gamma, \lambda[r\mu]_p, \Delta \,;\, w\rangle} \quad \text{mit } \langle \Pi, l \doteq r, \Sigma \,;\, \hat{w}\rangle^{\mathcal{L}}$$

falls es eine Position $p \in Pos(\lambda)$, eine Substitution μ und eine Klausel Θ gibt, so dass

- $\lambda/p = l\mu$
- $\Gamma, \Delta, l\mu = r\mu, \Theta$ enthält $DefCond(\mu, (\Pi, l \doteq r, \Sigma)), \Pi\mu, \Sigma\mu$
- $\Lambda_1, \ldots, \Lambda_n, \Lambda$ ist die aus Θ resultierende Fallunterscheidung

Mit Hilfe dieser Inferenzregel kann beispielsweise unter Anwendung des Lemmas $\langle x + y = y + x \,;\, w\rangle$ mit der Substitution $\mu = \{x \leftarrow \mathsf{f}(u), y \leftarrow v\}$ auf die Position $p = 1$ das Ziel $\langle \mathsf{f}(u) + v = v + \mathsf{f}(u) \,;\, w\rangle$ zu den Zielen

$$\langle \mathrm{def}\,(\mathsf{f}(u)) \;\vee\; \mathsf{f}(u) + v = v + \mathsf{f}(u) \,;\, w\rangle$$
$$\langle \neg \mathrm{def}\,(\mathsf{f}(u)) \;\vee\; v + \mathsf{f}(u) = v + \mathsf{f}(u) \,;\, w\rangle$$

abgeleitet werden.

1.2.3. Beweiszustandsgraphen

Beweise und Beweisversuche der Lemmata werden in sogenannten *Beweiszustandsgraphen* dargestellt. Diese Beweiszustandsgraphen sind UND-ODER-Graphen, die Zielknoten und Inferenzknoten als UND- bzw. ODER-Knoten enthalten. Für jedes zu beweisende Lemma wird ein neuer Zielknoten erzeugt. Eine Veränderung an einem Beweiszustandsgraphen ist nur möglich, indem auf einen Zielknoten eine Inferenzregel angewendet wird. Dadurch erhält dieser Zielknoten als Kind einen Inferenzknoten, dessen Kinder Zielknoten sind. Diese enthalten die Teilziele der Inferenzregel. Bei applikativen Inferenzregeln wird zusätzlich eine Kante zu einem Axiom-Knoten oder einem anderen Zielknoten eingefügt. Dadurch werden Zusammenhänge zwischen einzelnen Lemmata festgehalten. Es ist daher nicht unbedingt notwendig, dass diese

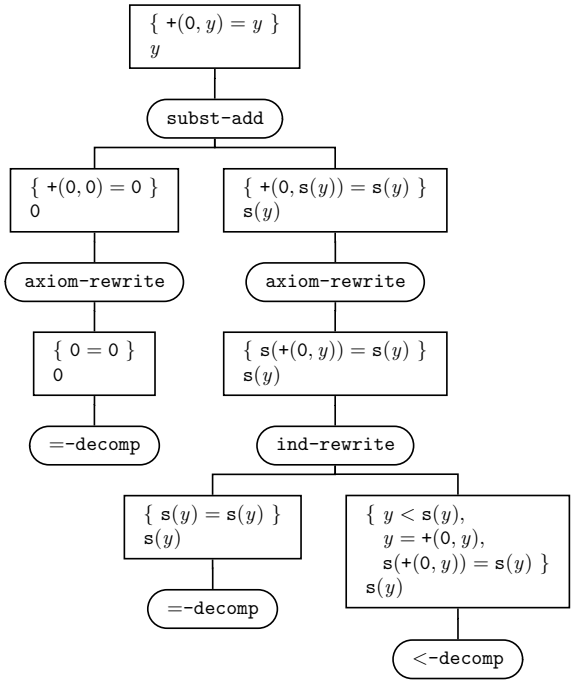

Abbildung 1.1.: Ein abgeschlossener Beweisbaum für das Ziel $+(0, y) = y$.

Lemmata bereits bewiesen sind, da die Abhängigkeiten später überprüft werden können. Lässt man diese Kanten jedoch weg, so zerfällt der Beweiszustandsgraph in den sogenannten *Beweiszustandswald*, dessen Komponenten *Beweiszustandsbäume* genannt werden. Die Wurzeln der Beweiszustandsbäume sind die zu beweisenden Ziele.

Es ist möglich, auf einen Zielknoten mehrere Inferenzregeln anzuwenden. Dadurch entstehen mehrere Inferenzknoten als Nachfolger des Zielknotens. Ein solcher Punkt wird *Entscheidungspunkt* genannt und ermöglicht es, mehrere Beweisversuche für einen Zielknoten parallel auszuführen.

Wenn eine Inferenzregel keine neuen Teilziele erzeugt, so ist dieser Inferenzknoten des Beweiszustandsbaumes *abgeschlossen*. Ein Inferenzknoten mit Teilzielen ist abgeschlossen, wenn jedes Teilziel abgeschlossen ist (daher werden die Inferenzknoten als UND-Knoten angesehen). Ein Zielknoten ist abgeschlossen, wenn mindestens ein nachfolgender Inferenzknoten abgeschlossen ist (daher sind die Zielknoten ODER-Knoten). Reduziert man einen Beweiszustandsbaum auf jeweils einen Inferenzknoten als Nachfolger eines Zielknotens, so erhält man einen *partiellen Beweisversuch*. Ist die Wurzel des partiellen Beweisversuches ein abgeschlossener Zielknoten und ist die induktive Gültigkeit aller angewandten Lemmata bewiesen, so wird der partielle Beweisversuch als *Beweisbaum* bezeichnet. Wie in [Küh00] gezeigt wird, ist ein Ziel induktiv gültig, wenn für das Ziel ein Beweisbaum existiert. Der Beweis eines induktiven Theorems läuft in QuodLibet daher so ab, dass zunächst ein neuer Zielknoten erzeugt wird, der das zu beweisende Theorem enthält. Anschließend wird

durch Anwendung von Inferenzregeln ausgehend von diesem Zielknoten versucht, einen Beweisbaum für das Ziel zu erzeugen.

Ein Beispiel für einen abgeschlossenen Beweisbaum ist in Abbildung 1.1 dargestellt. Zielknoten werden dabei als Rechtecke dargestellt, in denen das Ziel in Form einer Klausel in geschweiften Klammern und eines Gewichtes steht. Inferenzknoten werden als Rechtecke mit abgerundeten Ecken dargestellt, in denen der Name der angewendeten Inferenz steht. Der Beweisbaum beinhaltet einen Beweis für das Ziel $+(\mathsf{0}, y) = y$ in einer Spezifikation, die die Sorte Nat mit den Konstruktoren 0 und s und ein definiertes zweistelliges Funktionssymbol + mit den definierenden Regeln $+(x, \mathsf{0}) = x$ und $+(x, s(y)) = s(+(x, y))$ enthält.

1.2.4. Taktiken

Weil die Inferenzregeln korrekt und sicher sind und die einzige Möglichkeit darstellen, Änderungen an Beweiszustandsgraphen vorzunehmen, ist QuodLibet bereits dafür geeignet, einen manuell erarbeiteten Beweis zu überprüfen. Allerdings ist es auch wünschenswert, noch unbekannte Beweise automatisch zu finden. Dies ist möglich, indem die Anwendung der Inferenzregeln durch sogenannte *Taktiken* automatisiert wird. In QuodLibet werden diese Taktiken in der eigens entwickelten Programmiersprache QML (QuodLibet-Meta-Language) beschrieben. Hierbei handelt es sich um eine imperative Programmiersprache, ähnlich der Sprache Pascal, die neben Befehlen für die Ausführung von Inferenzregeln die notwendigen Kontrollstrukturen bereitstellt, um Taktiken für die Beweisfindung zu beschreiben. Diese Taktiken können vom Benutzer selbst programmiert und in das System geladen werden. Es steht jedoch auch eine Menge von Standard-Taktiken bereit, mit denen zumindest eine Teilautomatisierung des Beweisprozesses bereits realisierbar ist – dennoch ist eine Interaktion mit dem Benutzer in den meisten Fällen notwendig, um komplexere Beweise zu führen. Eine Beschreibung der bisher verwendeten Standard-Taktiken findet sich in [Sch04].

1.3. Entscheidungsverfahren

Mit einem induktiven Theorembeweiser wie QuodLibet ist es möglich, Theoreme aus verschiedenen Theorien zu beweisen. Dazu müssen nur in der Spezifikation die entsprechenden Axiome der Theorie enthalten sein. Diese Flexibilität ist der große Vorteil eines solchen allgemeinen Theorembeweisers. Allerdings tauchen viele Theorien sehr häufig auf, insbesondere in bestimmten Problemdomänen. Eine der häufigsten Theorien ist die Theorie der Arithmetik der natürlichen Zahlen. Hierbei zeigt sich eine Schwäche des allgemeinen Ansatzes, da es für einige dieser Theorien deutlich effizientere Verfahren gibt, die das Wissen über die Struktur der Axiome und damit der Theorie besser ausnutzen. Besonders interessant sind dabei diejenigen Theorien, für die es ein sogenanntes Entscheidungsverfahren gibt. Ein Entscheidungsverfahren liefert für jeden syntaktisch korrekten Ausdruck eine Entscheidung, ob dieser in einer gegebenen Theorie gültig ist oder nicht. Für die Arithmetik der natürlichen Zahlen zeigte Gödel bereits 1930 in [Göd31] mit seinem bekannten Unvollständig-

keitssatz, dass diese unvollständig ist; das heißt auch, dass für diese Theorie kein Entscheidungsverfahren existiert. Allerdings hatte Presburger bereits 1929 in [Pre29] gezeigt, dass eine Teiltheorie der Arithmetik der ganzen Zahlen, die lineare Arithmetik oder sogenannte Presburger-Arithmetik, vollständig ist und somit auch ein Entscheidungsverfahren existiert. Presburgers Argumentation lässt sich analog zur linearen Arithmetik der ganzen Zahlen auch für die der natürlichen Zahlen führen [JBG99].

Die Kombination spezieller Entscheidungsverfahren mit einem allgemeinen Theorembeweiser ist daher sehr viel versprechend. Man hat nach wie vor ein Verfahren, das allgemein für alle Theorien funktioniert, kann jedoch Aussagen, die aus einer der Theorien stammen, für die ein Entscheidungsverfahren zur Verfügung steht, wesentlich effizienter behandeln. Es ist zudem möglich, auch Aussagen, die nur teilweise in den Bereich eines Entscheidungsverfahrens fallen, mit dem Entscheidungsverfahren zu behandeln. Dazu werden Terme mit definierten Operatoren, die nicht zu der jeweiligen Theorie gehören, zu Variablen abstrahiert. Dann ist es auch möglich, auf die Axiome und Lemmata dieser Theorien zu verzichten, so dass das allgemeine Verfahren bei der Beweissuche einen deutlich kleineren Suchraum durchsuchen muss. Zudem wird auch der Benutzer entlastet, da die Axiome und Lemmata, die in den Bereich der Entscheidungsverfahren fallen, nicht mehr explizit formuliert werden müssen. Kritisch ist jedoch die Zusammenarbeit zwischen Theorembeweiser und Entscheidungsverfahren zu sehen. Hierbei kann der Effizienzgewinn durch das Entscheidungsverfahren möglicherweise wieder verloren gehen. Dabei ist es in der Regel nicht realisierbar, das Entscheidungsverfahren als „Black-Box“ zu integrieren, da zu viele Informationen zwischen den Komponenten ausgetauscht werden müssen [BM88].

1.4. Lineare Arithmetik

Lineare Arithmetik ist eine Theorie erster Stufe der natürlichen, ganzen bzw. rationalen Zahlen ohne Multiplikation. Diese Theorie ist im Gegensatz zur allgemeinen Arithmetik der natürlichen, ganzen bzw. rationalen Zahlen entscheidbar. Bereits 1929 bewies Mojzesz Presburger in [Pre29] diese Entscheidbarkeit für die lineare Arithmetik der ganzen Zahlen und dieser Beweis kann leicht auf die natürlichen Zahlen übertragen werden. Allerdings ist auch bewiesen, dass die Komplexität eines solchen Entscheidungsverfahrens für die natürlichen und die ganzen Zahlen bestenfalls in $\mathcal{O}(2^{2^n})$ liegen kann, wobei n die Länge der Aussage ist [FR74]. In der Praxis können viele Beispiele jedoch deutlich schneller entschieden werden.

Im Folgenden wird ausschließlich die lineare Arithmetik der natürlichen Zahlen behandelt, die daher auch kurz als lineare Arithmetik bezeichnet wird. Die lineare Arithmetik der natürlichen Zahlen umfasst die Prädikatenlogik erster Stufe mit den natürlichen Zahlen als Universum und den Operatoren $+$ und $\dot{-}$ sowie dem Prädikat $\leq$. Zusätzlich wird der Operator $*$ zur Multiplikation mit Konstanten als Abkürzung für eine entsprechend häufige Addition definiert: $n * x := x + x + \ldots + x$ für $n \in \mathbb{N}$. Dieser Operator muss als erstes Argument stets eine natürliche Zahl haben.

In dieser Arbeit werden nur Aussagen betrachtet, die keine Quantoren enthalten. Aufgrund der in Abschnitt 1.1.2 beschriebenen Semantik sind die Aussagen implizit allquantifiziert, da für die Gültigkeit einer Klausel, gefordert wird, dass diese von allen Belegungen erfüllt wird. Die so entstehende Arithmetik wird *quantorenfreie lineare Arithmetik* genannt. Im Folgenden wird unter dem Begriff *lineare Arithmetik* stets die *quantorenfreie lineare Arithmetik* verstanden.

Für die Informatik ist die quantorenfreie lineare Arithmetik besonders interessant, da viele Beweisziele, die typischerweise bei Hardware- und Software-Verifikation entstehen, mit den Mitteln dieser Arithmetik formuliert werden können. Daher ist es wünschenswert, solche Aufgaben effizient lösen zu können. Die Integration eines Entscheidungsverfahrens für lineare Arithmetik in einen induktiven Theorembeweiser ist hierfür sehr viel versprechend.

Es kommt jedoch sehr häufig vor, dass Beweisziele nicht in der reinen linearen Arithmetik liegen, sondern zusätzlich definierte Funktionssymbole enthalten. Auch diese Ziele können mit Hilfe des Entscheidungsverfahrens bearbeitet werden, indem man die Teilterme, die außerhalb der lineare Arithmetik liegen, zu Variablen abstrahiert. Natürlich ist das Entscheidungsverfahren dann kein Entscheidungsverfahren für die ursprüngliche, nicht in der linearen Arithmetik liegende, Klausel. Gelingt es jedoch, die Klausel mit den neuen Variablen mittels des Entscheidungsverfahrens zu beweisen, das heißt die Nicht-Existenz eines Gegenbeispiels nachzuweisen, so kann auch in der ursprünglichen Klausel kein Gegenbeispiel enthalten sein; somit ist auch diese bewiesen.

Beispiel: (nach [BM88, S. 92]) Zusätzlich zu der Sorte Nat und den beschriebenen Operatoren und Prädikaten sei die Sorte List definiert, die wie üblich Listen über natürlichen Zahlen beschreibt. Dazu seien die beiden folgenden Funktionen definiert, die das maximale bzw. minimale Element einer Liste bezeichnen sollen:

$$\begin{aligned} \mathsf{max} &: \mathsf{List} \longrightarrow \mathsf{Nat} \\ \mathsf{min} &: \mathsf{List} \longrightarrow \mathsf{Nat} \end{aligned}$$

Seien weiter l und k Variablen der Sorte Natund a eine Variable der Sorte List. Dann ist die Klausel

$$\mathsf{min}(a) < l \ \vee \ k \leq 0 \ \vee \ l < \mathsf{max}(a) + k$$

nicht in der linearen Arithmetik, kann aber durch Abstraktion der Terme $\mathsf{min}(a)$ und $\mathsf{max}(a)$ zu neuen Variablen min und max der Sorte Nat in eine solche umgewandelt werden:

$$min < l \ \vee \ k \leq 0 \ \vee \ l < max + k$$

Diese neue Klausel kann jedoch nicht mehr bewiesen werden, da die Information fehlt, dass $\mathsf{min}(a) \leq \mathsf{max}(a)$ ist. Möglicherweise steht diese Information als Lemma zur Verfügung, kann jedoch nicht mehr angewendet werden, weil die Terme $\mathsf{min}(a)$ und $\mathsf{max}(a)$ abstrahiert wurden. Daher ist es notwendig, diese Abstraktion rückgängig zu machen, wenn das Entscheidungsverfahren nicht das gewünschte Ergebnis bringt. Um dies zu vereinfachen, muss man in der Praxis die Abstraktion nicht durchführen, sondern kann lediglich während des Entscheidungsverfahrens alle Teilterme, die außerhalb der linearen Arithmetik liegen, wie Variablen behandeln. Dies geschieht hier im Rahmen der in Abschnitt 2.3 beschriebenen Polynomdarstellung.

Eine derartige Integration des Entscheidungsverfahrens in den Theorembeweiser ermöglicht es daher, dieses nicht nur für die Entscheidung von Aussagen der reinen linearen Arithmetik zu verwenden, sondern auch Aussagen, die definierte Funktionssymbole enthalten, zu verarbeiten. Diese Möglichkeit ist essentiell für den Nutzen des Entscheidungsverfahrens, da in der Praxis sehr selten Aussagen aus der reinen linearen Arithmetik bewiesen werden müssen.

2. Integration der linearen Arithmetik in QuodLibet

Ein wichtiger Schritt für die Integration eines Entscheidungsverfahrens ist zunächst, die notwendigen Basisstrukturen bereitzustellen. Bisher sind in QuodLibet keinerlei vordefinierte Sorten und Operatoren vorhanden. Um jedoch ein spezielles Entscheidungsverfahren für eine auf den natürlichen Zahlen basierende Arithmetik implementieren zu können, müssen sowohl die natürlichen Zahlen selbst als auch die zu behandelnden Operatoren dem System als solche bekannt sein.

2.1. Die Darstellung natürlicher Zahlen

Zunächst wird für die natürlichen Zahlen eine Sorte definiert, die mit Nat bezeichnet wird. Das heißt, dass zukünftig jede Spezifikation eine Signatur enthält, für die $\mathsf{Nat} \in S$ gilt. In QuodLibet wird dies so realisiert, dass beim Programmstart automatisch die Sorte Nat in die Liste der bekannten Sorten eingetragen wird.

Bisher wurden die natürlichen Zahlen in der Regel mittels der Konstruktoren $\mathsf{0} : \ \rightarrow \mathsf{Nat}$ für die natürliche Zahl Null und $\mathsf{s} : \mathsf{Nat} \rightarrow \mathsf{Nat}$ für die Nachfolgerfunktion dargestellt. Diese Darstellung wird sehr unhandlich, wenn mit Konstanten gearbeitet wird, die größere natürliche Zahlen repräsentieren, da die natürliche Zahl n durch den Konstruktorgrundterm $\mathsf{s}(\mathsf{s}(\mathsf{s}(\ldots(\mathsf{s}(\mathsf{0}))\ldots)))$ (im Folgenden kurz $\mathsf{s}^n(\mathsf{0})$) dargestellt werden muss. Um effizienter mit natürlichen Zahlen arbeiten zu können, werden daher abkürzende Schreibweisen für sämtliche Konstruktorgrundterme der bisherigen Darstellung eingeführt. Dazu wird das Zahlsymbol der natürlichen Zahl n als Abkürzung für $\mathsf{s}^n(\mathsf{0})$ verwendet. Das heißt, dass beispielsweise das Symbol $\mathsf{3}$ als Abkürzung für den Term $\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$ eingeführt wird. Die Menge aller dieser Symbole für alle natürlichen Zahlen wird zukünftig mit $\mathcal{N}$ bzw. als die Konstanten der natürlichen Zahlen bezeichnet. Um weiterhin eine Darstellung der natürlichen Zahlen mit freien Konstruktoren zu ermöglichen, wird dabei die Definition mit $\mathsf{0}$ und s beibehalten und fest in das System integriert. Das heißt, es werden fortan nur noch Spezifikationen verwendet, deren Signatur das Funktionssymbol s sowie die Konstante $\mathsf{0}$ als Konstruktorsymbole der Sorte Nat enthält. Für die Konstanten der natürlichen Zahlen wird ein neuer Symboltyp eingeführt, der überall im System an Stelle eines Konstruktorterms verwendet werden kann. Der Parser wird entsprechend angepasst, so dass automatisch solche Symbole erzeugt werden, wenn eine natürliche Zahl eingelesen wird.

Aus dieser Darstellung ergibt sich jedoch das Problem, dass es für jede natürliche Zahl (außer der Null) mehrere syntaktisch verschiedene, jedoch semantisch gleich-

wertige Darstellungen gibt. Da es aufgrund der unendlichen Anzahl nicht möglich ist, für jede der neuen Konstanten eine definierende Regel im System zu speichern, die deren Äquivalenz zur Darstellung in $\mathsf{s}^n(\mathsf{0})$ beschreibt, ist es notwendig, diese Information fest in das System zu integrieren und die beiden Darstellungen gleichwertig zu behandeln. Dies betrifft Matching, Unifikation und Termgleichheit. Hierbei muss sichergestellt werden, dass die Terme n, $\mathsf{s}^n(\mathsf{0})$, $\mathsf{s}^{n-1}(\mathsf{1})$, usw. für $n \in \mathcal{N}$ als gleich angesehen werden. Eine relativ einfache Lösung für dieses Problem ist die automatische Konvertierung der verkürzten Darstellung in die s-Darstellung. Diese Konvertierung kann schrittweise und nur bei den Operationen, die diese benötigen, durchgeführt werden. Will man beispielsweise die beiden Terme 2 und $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ auf Gleichheit überprüfen, so geschieht dies rekursiv durch Vergleich der Topsymbole und der Argumente. Es wäre an dieser Stelle also möglich, den Term 2 zunächst in $\mathsf{s}(\mathsf{1})$ zu konvertieren. Dann stimmt das Topsymbol von $\mathsf{s}(\mathsf{1})$ und $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ überein. Als nächsten Schritt werden die Argumente verglichen, das heißt 1 mit $\mathsf{s}(\mathsf{0})$. Auch hier kann man zunächst 1 in $\mathsf{s}(\mathsf{0})$ konvertieren, was dann sowohl im Topsymbol s als auch im Argument 0 mit dem zweiten Term übereinstimmt.

Analog kann man auch bei den Operationen Unifikation und Matching vorgehen. Diese erweiterte Form der Termgleichheit ist in vielen Fällen jedoch sehr aufwändig zu überprüfen, so dass bei der Implementierung noch einige Optimierungen notwendig sind. Beispielsweise muss vermieden werden, dass beim Vergleich zweier vordefinierter Nat-Konstanten wie beispielsweise 5 und 7 beide Terme in ihre s-Darstellung expandiert werden.

2.2. Operatoren und Prädikate der linearen Arithmetik

Wie in Abschnitt 1.4 beschrieben, enthält die lineare Arithmetik die Operatoren $+$, $\dot{-}$ und $*$ sowie das Prädikat $\leq$. Die drei Operatoren $+$, $\dot{-}$ und $*$ arbeiten rein auf der Sorte Nat, für die Definition des Prädikates $\leq$ wird zusätzlich die Sorte Bool mit den Konstruktorsymbolen true und false benötigt.

Daraus ergibt sich folgende Definition für eine Signatur, die die Sorten und Funktionssymbole der linearen Arithmetik enthält:

Definition 2.1 *Eine* Signatur mit linearer Arithmetik *ist eine Signatur* $sig = (S, F, \alpha)$, *für die gilt:*

- $\{\mathit{Nat}, \mathit{Bool}\} \subseteq S$
- $\{+, \dot{-}, *, \mathsf{s}, \leq\} \subseteq F$, $\mathcal{N} \subseteq F$
- $\alpha(+) = \mathit{Nat}\ \mathit{Nat}\ \mathit{Nat}$
  $\alpha(\dot{-}) = \mathit{Nat}\ \mathit{Nat}\ \mathit{Nat}$
  $\alpha(*) = \mathit{Nat}\ \mathit{Nat}\ \mathit{Nat}$
  $\alpha(\mathsf{s}) = \mathit{Nat}\ \mathit{Nat}$
  $\alpha(\leq) = \mathit{Nat}\ \mathit{Nat}\ \mathit{Bool}$

Im Folgenden werden die Funktionssymbole $+$, $\dot{-}$ und $*$ mit Infix-Notation verwendet, um die Lesbarkeit zu erhöhen. Alle anderen Funktionssymbole werden jedoch nach wie vor mit der bei QuodLibet üblichen Präfix-Notation verwendet.

Die Bedeutung der Funktionssymbole wird nun durch eine Menge von Axiomen festgelegt, die im Rahmen der linearen Arithmetik vorgegeben sind:

$$
\begin{aligned}
x + \mathsf{0} &= x && (A1)\\
x + \mathsf{s}(y) &= \mathsf{s}(x + y) && (A2)\\
x * \mathsf{0} &= \mathsf{0} && (A3)\\
x * \mathsf{s}(y) &= (x * y) + x && (A4)\\
x \dot{-} \mathsf{0} &= x && (A5)\\
\mathsf{0} \dot{-} x &= \mathsf{0} && (A6)\\
\mathsf{s}(x) \dot{-} \mathsf{s}(y) &= x \dot{-} y && (A7)\\
\leq(\mathsf{0}, y) &= \mathsf{true} && (A8)\\
\leq(\mathsf{s}(x), \mathsf{0}) &= \mathsf{false} && (A9)\\
\leq(\mathsf{s}(x), \mathsf{s}(y)) &= \leq(x, y) && (A10)
\end{aligned}
$$

Die Menge aller dieser Axiome wird mit $\mathcal{AX}_{LA}$ bezeichnet. Daraus ergibt sich dann folgende Definition für eine Spezifikation mit Konstruktoren, die die Axiome der linearen Arithmetik enthält:

Definition 2.2 *Eine* Spezifikation mit Konstruktoren und linearer Arithmetik *ist eine Spezifikation mit Konstruktoren $spec = (sig, C, E)$, für die gilt:*

- *sig ist eine Signatur mit linearer Arithmetik*
- $C_{Nat} = \{0, \mathsf{s}\}$
  $C_{Bool} = \{\mathsf{true}, \mathsf{false}\}$
- $\mathcal{AX}_{LA} \subseteq E$

Enthält die Spezifikation keine weiteren Sorten, Operatoren und Axiome, so spricht man von der Spezifikation der reinen linearen Arithmetik.

Durch diese Definitionen wird die Spezifizierbarkeit nicht eingeschränkt. Alle Spezifikationen ohne lineare Arithmetik lassen sich problemlos zu Spezifikationen mit linearer Arithmetik erweitern, indem die Sorten- und Funktionssymbole sowie die Axiome der linearer Arithmetik zur Spezifikation hinzugenommen werden. Sofern in der ursprünglichen Spezifikation bereits Sorten- oder Funktionssymbole der linearen Arithmetik enthalten waren, können diese umbenannt werden, um Konflikte zu vermeiden. Da es unendlich viele Möglichkeiten für Sorten- und Funktionssymbole gibt, entsteht dadurch keine Einschränkung. Die Menge der induktiv gültigen Theoreme wird durch diese Erweiterung nicht eingeschränkt, sondern wird ebenfalls erweitert. Das heißt, dass alle vorher induktiv gültigen Theoreme auch anschließend induktiv

gültige Theoreme sind. Zusätzlich sind alle in der linearen Arithmetik induktiv gültigen Theoreme hinzugekommen. Dadurch ist es gerechtfertigt, im Folgenden nur noch mit Spezifikationen mit linearer Arithmetik zu arbeiten.

Intention der Spezifikationen mit linearer Arithmetik ist es, die lineare Arithmetik auf den natürlichen Zahlen zu beschreiben. Da die natürlichen Zahlen mit den üblichen Operationen Datenmodell einer Spezifikation *spec* mit reiner linearer Arithmetik sind, gelten alle in *spec* induktiv gültigen Theoreme auch in den natürlichen Zahlen. Für die Bestimmung induktiv gültiger Theoreme sind jedoch alle Datenmodelle zu betrachten, nicht nur die natürlichen Zahlen. Um diese Semantik beizubehalten, werden daher die folgenden Beweise nicht nur für die natürlichen Zahlen, sondern für alle Datenmodelle geführt.

Bei der Integration der linearen Arithmetik in QuodLibet wird statt des Operators $\leq$ mit Ergebnistyp Bool ein vordefiniertes Prädikat in Form eines neuen Literaltyps eingeführt, das sogenannte $\leq$-Atom. Dies dient dazu, im Rahmen des Entscheidungsverfahrens einfacher mit diesem Prädikat arbeiten zu können. Zudem kann dadurch auf die Einführung der Sorte Bool verzichtet werden. Ein $\leq$-*Atom* ist ein Termpaar der Form $t_1 \leq t_2$ mit $t_1, t_2 \in \mathcal{T}(sig, V)_{\mathsf{Nat}}$. Ein Atom ist dann eine Gleichung, ein Ordnungsatom, ein Definiertheitsatom oder ein $\leq$-Atom. Um die Semantik von $\leq$-Atomen festzulegen, wird der zweite Punkt der Definition 1.3 wie folgt erweitert: $\mathcal{A}$ erfüllt ein $\leq$-Atom $t_1 \leq t_2$, falls gilt $eval^{\mathcal{A}}_{\varphi}(\leq(t_1, t_2)) = \mathsf{true}$. Somit kann ein $\leq$-Atom $t_1 \leq t_2$ als abkürzende Schreibweise für $\leq(t_1, t_2) = true$ angesehen werden. Dadurch ist es möglich, bei der Integration in QuodLibet die Sorte Bool nicht mehr explizit zu integrieren, da diese in der Syntax nicht mehr auftaucht.

Um nun das Entscheidungsverfahren für lineare Arithmetik in QuodLibet zu integrieren, ist es zunächst notwendig, QuodLibet auf Spezifikationen mit linearer Arithmetik umzustellen. Dazu werden die Sorten- und Funktionssymbole sowie die Axiome der linearen Arithmetik fest in das System integriert. Dies geschieht, indem beim Systemstart die Funktionssymbole $+$, $\dot{-}$, $*$ und s sowie die Konstante $\mathsf{0}$ automatisch in die Liste der Funktionssymbole eingetragen werden. Die Axiome werden ebenfalls automatisch für jede Spezifikation in das System aufgenommen. Dies dient vor allem dazu, die Analyse der Operatoren zu ermöglichen, beispielsweise um zu einem späteren Zeitpunkt eine geeignete Induktionsvariable zu finden. Allerdings sollten die Axiome für $+$ und $\dot{-}$ nicht angewendet werden. Stattdessen wird die in den folgenden Abschnitten beschriebene Normalisierung angewendet, die die Anwendung dieser Axiome beinhaltet. Die Axiome für $*$ werden jedoch weiterhin angewendet, da im Rahmen der linearen Arithmetik nur Multiplikationen mit Konstanten behandelt werden können. Für nichtlineare Multiplikationen müssen daher nach wie vor die Axiome für den Beweisprozess zur Verfügung stehen.

Alle Grundterme der linearen Arithmetik können, da sämtliche Operatoren total definiert sind, unter Anwendung der Axiome zu Konstruktorgrundtermen ausgewertet werden. Daher kann jeder Grundterm t der linearen Arithmetik zu einer Konstante aus $\mathcal{N}$ ausgewertet werden, für die im Folgenden die Schreibweise $(t)\!\downarrow$ verwendet wird. Beispielsweise gilt $(\mathsf{3} + \mathsf{4})\!\downarrow = (\mathsf{s}^3(\mathsf{0}) + \mathsf{s}^4(\mathsf{0}))\!\downarrow = \mathsf{s}^7(\mathsf{0}) = \mathsf{7}$ und $(\mathsf{12} \dot{-} \mathsf{5})\!\downarrow = (\mathsf{s}^{12}(\mathsf{0}) \dot{-} \mathsf{s}^5(\mathsf{0}))\!\downarrow = \mathsf{s}^7(\mathsf{0}) = \mathsf{7}$. Außerdem kann jeder Konstruktorgrundterm $\mathsf{s}^n(0)$ der Sorte Nat auf eine äquivalente Konstante $n \in \mathbb{N}$ abgebildet werden

und umgekehrt. Das heißt, die Auswertung eines Grundterms zu einer Konstanten entspricht inhaltlich dem „Rechnen“ auf natürlichen Zahlen.

Somit ist es auch möglich, gewisse Konzepte, die auf den natürlichen Zahlen $\mathbb{N}$ verwendet werden, auf die hier verwendeten Konstanten der natürlichen Zahlen $\mathcal{N}$ zu übertragen. Ein solches Konzept, das für die folgenden Abschnitte benötigt wird, ist das eines *gemeinsamen Teilers* natürlicher Zahlen. Ein gemeinsamer Teiler g zweier natürlicher Zahlen a und b ist eine Zahl, die beide Zahlen ohne Rest teilt. Das heißt es existieren $a', b' \in \mathbb{N}$, so dass $a = g * a'$ und $b = g * b'$. Dieses Konzept wird nun folgendermaßen auf die hier verwendeten Konstanten und Operationen übertragen: Seien x und y Konstanten aus $\mathcal{N}$. Dann können die Konstanten x und y auf natürliche Zahlen $\widetilde{x}$ und $\widetilde{y}$ in $\mathbb{N}$ abgebildet werden. Ist nun $\widetilde{g} \in \mathbb{N}$ ein gemeinsamer Teiler von $\widetilde{x}$ und $\widetilde{y}$, so wird die $\widetilde{g}$ entsprechende Konstante g in $\mathcal{N}$ gemeinsamer Teiler von x und y genannt. Dann existieren auch $x', y' \in \mathcal{N}$, so dass $x = (g * x')\downarrow$ und $y = (g * y')\downarrow$. Analog kann auch ein gemeinsamer Teiler mehrerer natürlicher Zahlen bzw. Konstanten definiert werden.

Der *größte gemeinsame Teiler* (ggT) zweier oder mehrerer Zahlen ist dann der bezüglich der Ordnung der natürlichen Zahlen größte der gemeinsamen Teiler der Zahlen. Auf den natürlichen Zahlen ist der ggT eindeutig bestimmt. Auch das Konzept des größten gemeinsamen Teilers lässt sich analog zum Vorgehen beim gemeinsamen Teiler auf die hier definierten Konstanten für die natürlichen Zahlen übertragen.

2.3. Terme in Polynomdarstellung

Um effizient mit Termen der linearen Arithmetik arbeiten zu können, bietet es sich an, diese als Polynome darzustellen. Dies ermöglicht es, die Operationen des Entscheidungsverfahrens effizient auszuführen.

Definition 2.3 *Ein* Term-Polynom *ist eine Summe der Form*

$$a_1 t_1 + a_2 t_n + \ldots + a_n t_n + a \quad \textit{mit } a_i, a \in \mathcal{N}, a_i \neq 0, n \in \mathbb{N}, t_i \in \mathcal{T}(F, V)_{\mathsf{Nat}}.$$

Als abkürzenden Schreibweise wird im Folgenden meist die Form

$$\sum_{i=1}^{n} a_i t_i + a$$

verwendet.

Man nennt die t_i die Summanden*, die a_i die* Koeffizienten*, die $a_i t_i$ die* Monome *und a die* Konstante *des Term-Polynoms.*

Hierbei ist es sinnvoll, zusätzliche Eigenschaften zu fordern, die eine Normalform für Term-Polynome beschreiben. Durch Anwendung der Axiome der linearen Arithmetik und einfacher, daraus ableitbarer, Lemmata ist es möglich, jeden Term in eine solche Normalform zu überführen. Dies erleichtert die Vergleichbarkeit von Polynomen und erhöht nochmals die Effizienz bei der Behandlung der Polynome im

Rahmen des Entscheidungsverfahrens. Zudem ist es möglich, auf die explizite Anwendung der Axiome der linearen Arithmetik zu verzichten, wenn die Umwandlung eines Terms in ein Term-Polynom in Normalform in das System integriert wird.

Für die Normalform wird gefordert, dass die Monome minimal sind, das heißt nicht selbst wieder eine Addition oder eine Multiplikation mit einer Konstanten sind. Falls ein Monom eine Subtraktion beschreibt, so wird diese unter Einsatz der Axiome möglichst stark vereinfacht. Weiterhin ist es möglich, alle Monome zu sortieren, da die Kommuntativität und Assoziativität von + bewiesen werden können. Hierbei kann auch zusätzlich sichergestellt werden, dass kein Summand mehrfach vorkommt. Daraus ergibt sich die folgende Definition:

Definition 2.4 *Sei $\prec$ eine strikte und totale Ordnung auf Termen. Ein Term-Polynom t der Form*

$$t = \sum_{i=1}^{n} a_i t_i + a \qquad \textit{mit } a_i, a \in \mathcal{N}, a_i \neq 0, n \in \mathbb{N}, t_i \in \mathcal{T}(F,V)_{\mathsf{Nat}}$$

ist in Polynom-Normalform (PNF) bezüglich $\prec$*, falls für alle t_i*

- *t_i ist eine Variable oder*
- *$top(t_i) \notin \{+, *, \dot{-}, s\}$ oder*
- *$t_i = l * r$ und $l, r \notin \mathcal{N}$, l, r Term-Polynome in PNF oder*
- *$t_i = l \dot{-} r$ und*
  - *l ist ein Term-Polynom in PNF der Form $\sum_{j=1}^{n'} a'_j t'_j + a'$*
  - *r ist ein Term-Polynom in PNF der Form $\sum_{k=1}^{n''} a''_k t''_k + a''$*
  - *alle t'_j, t''_k sind paarweise disjunkt*
  - *$a' = 0$ oder $a'' = 0$*
  - *alle a'_j, a''_k, a', a'' haben keinen gemeinsamen Teiler außer 1*
  - *$r \neq 0$, $l \neq 0$*

und $t_1 \prec t_2 \prec \ldots \prec t_n$ gilt. Damit sind auch alle t_i paarweise verschieden, da die Ordnung $\prec$ strikt ist.

Term-Polynome werden im Folgenden auch anstelle von Termen verwendet, obwohl sie syntaktisch keine Terme darstellen. Dazu wird die folgende *Termrepräsentation* eines Term-Polynoms verwendet, wenn notwendig:

Definition 2.5 *Die* Termrepräsentation $[a_i t_i]_T$ *eines Monoms $a_i t_i$ ist*

$$[a_i t_i]_T := \begin{cases} t_i & \textit{falls } a_i = 1 \\ (a_i * t_i) & \textit{sonst} \end{cases}$$

Die Termrepräsentation $[t]_T$ *eines Term-Polynoms $t = \sum_{i=1}^{n} a_i t_i + a$ ist*

$$[p]_T := \begin{cases} a + ([a_1 t_1]_T + ([a_2 t_2]_T + \ldots + [a_n t_n]_T) \ldots)) & \textit{falls } a \neq 0 \\ [a_1 t_1]_T + ([a_2 t_2]_T + \ldots + [a_n t_n]_T) \ldots) & \textit{sonst} \end{cases}$$

Im Folgenden wird diese Umwandlung eines Term-Polynoms in seine Termdarstellung nicht immer explizit angegeben, sondern immer dann, wenn ein Term-Polynom als Term behandelt wird, angenommen. Auch für Termpolynome selbst werden oft zur besseren Lesbarkeit eine Konstante, die 0 ist, und Faktoren, die 1 sind, weggelassen.

Da die Ordnung $\prec$ in der Definition beliebig, aber fest ist, wird sie fortan nicht mehr explizit aufgeführt. Der Ausdruck *ein Term-Polynom t ist in PNF* bedeutet dann, dass das Term-Polynom t bezüglich einer beliebigen, aber festen Ordnung $\prec$ in PNF ist.

Um zu zeigen, dass es mit den bisherigen Inferenzregeln von QuodLibet möglich ist, einen Term in ein Term-Polynom in PNF zu überführen, wird eine vereinfachte Ableitbarkeit definiert, die sich insbesondere dadurch auszeichnet, dass keine induktiven Inferenzregeln zugelassen sind.

Definition 2.6 *Sei spec eine Spezifikation. Ein Term t heißt* in *spec* mit einer Klausel Θ zu einem Term s nicht-induktiv ableitbar*, wenn jedes Ziel $\langle \Gamma\,;\, w\rangle$, für das $\Gamma/p = t$ gilt, durch einmalige Anwendung der* QuodLibet*-Inferenzregel Literal Hinzufügen mit Θ und anschließende Anwendung von nicht-induktiven Inferenzregeln (wobei nur in spec gültige Lemmata und Axiome für applikative Inferenzregeln verwendet werden) in die Ziele*

$$\langle \Lambda_1, \Gamma\,;\, w\rangle \ldots \langle \Lambda_n, \Gamma\,;\, w\rangle\ \langle \Lambda, \Gamma[s]_p\,;\, w\rangle$$

überführt werden kann. Dabei ist $\Lambda_1, \ldots, \Lambda_n, \Lambda$ die aus Θ resultierende Fallunterscheidung ist.

Man sagt dann auch umgekehrt, dass der Term s in spec mit Θ aus t nicht-induktiv ableitbar ist. Ein Term t ist in jeder Spezifikation spec mit $\emptyset$ zu sich selbst nicht-induktiv ableitbar.

Die folgenden zwei Lemmata beschreiben Eigenschaften der nicht-induktiven Ableitbarkeit von Termen, die in den folgenden Beweisen benötigt werden.

Lemma 2.1 *Sei spec eine Spezifikation mit Konstruktoren. Ist in spec der Term t mit Θ_1 zum Term t_1 nicht-induktiv ableitbar und der Term t_1 mit Θ_2 zum Term t_2 nicht-induktiv ableitbar, so ist in spec auch t mit $\Theta = \Theta_1 \cup \Theta_2$ zu t_2 nicht-induktiv ableitbar.*

Beweis: Ist in *spec* der Term t mit Θ_1 zum Term t_1 nicht-induktiv ableitbar und der Term t_1 mit Θ_2 zu t_2 nicht-induktiv ableitbar, so kann ein Ziel $\langle \Gamma\,;\, w\rangle$, für das $\Gamma/p = t$ gilt, durch einmalige Anwendung der QuodLibet-Inferenzregel Literal Hinzufügen mit Θ in die Ziele

$$\langle \Lambda_1, \Gamma\,;\, w\rangle \ldots \langle \Lambda_n, \Gamma\,;\, w\rangle\ \langle \Lambda, \Gamma\,;\, w\rangle$$

überführt werden, wobei $\Lambda_1, \ldots, \Lambda_n, \Lambda$ die aus Θ resultierende Fallunterscheidung ist.

Anschließend kann das Ziel $\langle \Lambda, \Gamma\,;\, w\rangle$ in *spec* durch aufeinander folgende Ausführung der Instanzen der nicht-induktiven, applikativen Inferenzregeln, die für die

Ableitung von t nach t_1 und von t_1 nach t_2 verwendet wurden, zu $\langle \Lambda, \Gamma[t_2]_p \,;\, w \rangle$ abgeleitet werden. □

Lemma 2.2 *Sei spec eine Spezifikation und t ein Term für den $t/p = t'$ gilt. Ist in spec der Term t' mit Θ zum Term s nicht-induktiv ableitbar, so ist in spec auch t mit Θ zu $t[s]_p$ nicht-induktiv ableitbar.*

Beweis: Folgt unmittelbar aus der Definition von *nicht-induktiv ableitbar*. □

In den folgenden Beweisen wird stets angegeben, welche Instanzen von Inferenzregeln für die nicht-induktive Ableitbarkeit angewendet werden. Sind lediglich Axiome und induktiv gültige Lemmata angegeben, so bedeutet dies, dass diese mit der Inferenzregel Nicht-induktive Termersetzung angewendet werden, sofern es sich um eine bedingte Gleichung handelt. Handelt es sich um ein Lemma der Form $l \leftrightarrow r$, so wird das Lemma zunächst in einer Richtung mit der Inferenzregel Nicht-induktive Termersetzung und anschließend, um das ursprüngliche Lemma zu entfernen, in die andere Richtung mit der Inferenzregel Applikative Literal-Beseitigung angewendet. Da Gültigkeit der Bedingungsliterale meist aus dem Kontext klar ist, wird deren Nachweis dann nicht explizit aufgeführt.

Einige Axiome und einfache Lemmata werden hierbei zur Verbesserung der Übersichtlichkeit oft nicht explizit aufgeführt, sofern sie für den Nachweis der nicht-induktiven Ableitbarkeit nicht wesentlich sind. Dazu gehören beispielsweise die Assoziativität und Kommutativität von +, die häufig zunächst angewandt werden, um Summanden so umzusortieren, dass andere Lemmata anwendbar sind. Das folgende Lemma enthält eine Liste der benötigten, in jeder Spezifikation mit linearer Arithmetik induktiv gültigen Lemmata.

Lemma 2.3 *Die in Abbildung 2.1 aufgeführten Lemmata sind in jeder Spezifikation mit linearer Arithmetik spec induktiv gültig (u, v, w, x, y und z sind Konstruktorvariablen der Sorte Nat).*

Die Schreibweise $\Gamma \leftarrow \Delta$ steht hierbei (als Erweiterung des Konzeptes des bedingten Gleichung) für eine bedingte Klausel, deren Klauselrepräsentation $\Gamma, \overline{\delta_1} \ldots \overline{\delta_n}$ ist, falls $\Delta = \delta_1 \ldots \delta_n$ ist. Weiter steht $\Gamma \leftrightarrow \Delta$ (Γ und Δ sind Klauseln) als Abkürzung für zwei Lemmata $\Gamma \leftarrow \Delta$ und $\Delta \leftarrow \Gamma$.

Beweis: Siehe Anhang A. □

Mit Hilfe dieser Lemmata kann nun folgender Satz bewiesen werden:

Satz 2.1 *Sei spec eine Spezifikation mit Konstruktoren und linearer Arithmetik. Zu jedem Term t der Sorte Nat gibt es ein Term-Polynom t_p in PNF, dessen Termdarstellung $[t_p]_T$ in spec mit einer Klausel PNFDefCond(t) aus t nicht-induktiv ableitbar ist.*

$$
\begin{array}{rcll}
\mathrm{def}\,(x+y) & & & (L1)\\
\mathrm{def}\,(x*y) & & & (L2)\\
1*x & = & x & (L3)\\
x+y & = & y+x & (L4)\\
(x+y)+z & = & x+(y+z) & (L5)\\
(x*z)+(y*z) & = & (x+y)*z & (L6)\\
(x+1)*y & = & (x*y)+y & (L7)\\
\mathsf{s}(x) & = & x+1 & (L8)\\
(x*y)\dot{-}(x*z) & = & x*(y\dot{-}z) & (L9)\\
(x+y)\dot{-}(x+z) & = & y\dot{-}z & (L10)\\
x+y=w+z & \leftrightarrow & x+(y\dot{-}z)=w+(z\dot{-}y) & (L11)\\
x+(y*u)=w+(z*u) & \leftrightarrow & x+((y\dot{-}z)*u)=w+((z\dot{-}y)*u) & (L12)\\
{\leq}(x*y,x*z) & = & {\leq}(y,z) \leftarrow x\neq 0 & (L13)\\
{\leq}(u*v,(u*w)+y) & = & {\leq}(v,w) \leftarrow {\leq}(u,y)=\mathsf{false} & (L14)\\
{\leq}((u*x)+v,u*y) & = & {\leq}(x+1,y) \leftarrow {\leq}(u,v)=\mathsf{false}, v\neq 0 & (L15)\\
x*y & \neq & (x*z)+u \leftarrow u\neq 0, {\leq}(x,u)=\mathsf{false} & (L16)\\
(x*y=x*z) & \leftrightarrow & y=z) \leftarrow x\neq 0 & (L17)\\
{\leq}(x,y)=\mathsf{false} & \leftrightarrow & {\leq}(y+1,x)=\mathsf{true} & (L18)\\
x<y & \leftrightarrow & {\leq}(x+1,y)=\mathsf{true} & (L19)\\
x\not<y & \leftrightarrow & {\leq}(y,x)=\mathsf{true} & (L20)\\
x\dot{-}y & = & 0 \leftarrow {\leq}(y,x)=\mathsf{false} & (L21)\\
(x+(y\dot{-}z)=u & \leftrightarrow & x+y=u+z) \leftarrow {\leq}(z,y)=\mathsf{true} & (L22)\\
({\leq}(x+(y\dot{-}z),u)=\mathsf{true} & \leftrightarrow & {\leq}(x+y,u+z)=\mathsf{true}) \leftarrow {\leq}(z,y)=\mathsf{true} & (L23)\\
x & = & y \leftarrow {\leq}(x,y)=\mathsf{true}, {\leq}(y,x)=\mathsf{true} & (L24)\\
{\leq}(x,y) & = & \mathsf{true} \leftarrow x=y & (L25)\\
{\leq}(x,y) & = & \mathsf{true} \leftarrow {\leq}(y,x)\neq\mathsf{true} & (L26)
\end{array}
$$

Abbildung 2.1.: In jeder Spezifikation mit linearer Arithmetik induktiv gültige Lemmata.

Beweis: Induktion über den Termaufbau.

Induktionsanfang:

1. $t \in V$: $t_p = 1t$ ist ein Polynom-Term in PNF und t ist in *spec* mit

$$\mathit{PNFDefCond}(t) = \begin{cases} \emptyset & \text{falls } t \in V^C \\ \{\neg\mathrm{def}\,(t)\} & \text{sonst} \end{cases}$$

   zu $[t_p]_T$ nicht-induktiv ableitbar, denn es gilt $[t_p]_T = [1t]_T = t$. Auch wenn t keine Konstruktorvariable ist, reicht $\mathit{PNFDefCond}(t) = \emptyset$ für die Ableitbarkeit aus, für den weiteren Beweis ist es jedoch nützlich, hier bereits $\neg\mathrm{def}\,(t)$ aufzunehmen.

2. $t \in \mathcal{N}$: $t_p = t$ ist ein Polynom-Term in PNF, der in *spec* mit $\mathit{PNFDefCond}(t) = \emptyset$ zu $[t_p]_T$ nicht-induktiv ableitbar ist, denn es gilt $[t_p]_T = [t]_T = t$.

Induktionsschritt: Fallunterscheidung nach dem Topsymbol des Terms:

1. $t = l + r$: Seien l_p und r_p Term-Polynome in PNF, deren Termdarstellungen in *spec* mit $\mathit{PNFDefCond}(l)$ bzw. $\mathit{PNFDefCond}(r)$ aus l bzw. r nicht-induktiv ableitbar sind. Diese existieren nach Induktionsvoraussetzung und haben die Form $l_p = \sum_{i=1}^{n} a_i t_i + a$ und $r_p = \sum_{j=1}^{m} b_j s_j + b$. Dann ist $t' := l_p + r_p$ in *spec* mit $\mathit{PNFDefCond}(t) = \mathit{PNFDefCond}(l) \cup \mathit{PNFDefCond}(r)$ aus t nicht-induktiv ableitbar (nach Lemma 2.1 und 2.2). Sei $c_j := (b_j + a_i)\downarrow$ für alle i, j mit $s_j = t_i$ (für jedes s_j existiert maximal ein solches t_i, da l_p und r_p in PNF sind) und sei $c := (a + b)\downarrow$. Dann ist auch die Termdarstellung des Polynoms

$$t'' := \sum_{\{i \mid 1 \le i \le n, \forall j : s_j \neq t_i\}} a_i t_i + \sum_{\{j \mid 1 \le j \le m, \forall i : t_i \neq s_j\}} b_i s_j + \sum_{\{j \mid 1 \le j \le m, \exists i : t_i = s_j\}} c_j s_j + c$$

   mit $\mathit{PNFDefCond}(t)$ in *spec* aus t nicht-induktiv ableitbar (Anwendung von *A1*, *L3*, *L4*, *L5* und *L6*). Sortiert man die Summanden von t'' entsprechend einer totalen Ordnung $\prec$ auf Termen, so erhält man ein Term-Polynom t_p in PNF, dessen Termdarstellung ebenfalls in *spec* mit $\mathit{PNFDefCond}(t)$ aus t nicht-induktiv ableitbar ist (Lemma 2.1 und Anwendung von *L4* und *L5*).

2. $t = l * r$: Seien l_p und r_p Term-Polynome in PNF, deren Termdarstellung in *spec* mit $\mathit{PNFDefCond}(l)$ bzw. $\mathit{PNFDefCond}(r)$ aus l bzw. r nicht-induktiv ableitbar ist. Diese existieren nach Induktionsvoraussetzung und haben die Form $l_p = \sum_{i=1}^{n} a_i t_i + a$ und $r_p = \sum_{j=1}^{m} b_j s_j + b$. Dann ist $t' := l_p * r_p$ in *spec* mit $\mathit{PNFDefCond}(t) = \mathit{PNFDefCond}(l) \cup \mathit{PNFDefCond}(r)$ aus t nicht-induktiv ableitbar (nach Lemma 2.1 und Lemma 2.2). Daraus können dann folgende vereinfachte Term-Polynome abgeleitet werden, die in PNF sind:

   a) $n = 0, m = 0$: $t_p := (a * b)\downarrow$ ist ein Term-Polynom in PNF, dessen Termdarstellung in *spec* mit $\emptyset$ aus t' und damit mit $\mathit{PNFDefCond}(t)$ aus t nicht-induktiv ableitbar ist (Lemma 2.1).

b) $n > 0, m = 0$: $t_p := \sum_{i=1}^{n}(b * a_i)\downarrow t_i + (b * a)\downarrow$ ist ein Term-Polynom in PNF, dessen Termdarstellung in *spec* mit *PNFDefCond*(t_1) aus t' und damit mit *PNFDefCond*(t) aus t nicht-induktiv ableitbar ist (Lemma 2.1, Anwendung von *L6*).

c) $n = 0, m > 0$: $t_p := \sum_{j=1}^{m}(a * b_j)\downarrow s_j + (a * b)\downarrow$ ist ein Term-Polynom in PNF, dessen Termdarstellung in *spec* mit *PNFDefCond*(t_2) aus t' und damit mit *PNFDefCond*(t) aus t nicht-induktiv ableitbar ist (Lemma 2.1, Anwendung von *L6*).

d) $n > 0, m > 0$: $t_p := 1(t_{1_p} * t_{2_p})$ ist ein Term-Polynom in PNF, dessen Termdarstellung $[t_p]_T$ in *spec* mit *PNFDefCond*(t) aus t nicht-induktiv ableitbar ist, da $[t_p]_T = l_p * r_p = t'$.

3. $t = \mathsf{s}(l)$: Sei l_p ein Term-Polynom in PNF, dessen Termdarstellung in *spec* mit *PNFDefCond*(l) aus l nicht-induktiv ableitbar ist. Dieser existiert nach Induktionsvoraussetzung und hat die Form $l_p = \sum_{i=1}^{n} a_i t_i + a$. Dann ist $t_p := \sum_{i=1}^{n} a_i t_i + (a+1)\downarrow$ ein Term-Polynom in PNF, dessen Termdarstellung mit *PNFDefCond*(t) = *PNFDefCond*(l) aus t nicht-induktiv ableitbar ist (Lemma 2.1 und Lemma 2.2, Anwendung von *L8*).

4. $t = l \dot{-} r$: Seien l_p und r_p Term-Polynome in PNF, deren Termdarstellung in *spec* mit *PNFDefCond*(l) bzw. *PNFDefCond*(r) aus l bzw. r nicht-induktiv ableitbar ist. Diese existieren nach Induktionsvoraussetzung und haben die Form $l_p = \sum_{i=1}^{n} a_i t_i + a$ und $r_p = \sum_{j=1}^{m} b_j s_j + b$. Seien

$$\begin{aligned} c_i &:= \begin{cases} a_i & \text{falls } s_j \neq t_i \text{ für alle } j \\ (a_i \dot{-} b_j)\downarrow & \text{falls } s_j = t_i \end{cases} \\ d_j &:= \begin{cases} b_j & \text{falls } s_j \neq t_i \text{ für alle } i \\ (b_j \dot{-} a_i)\downarrow & \text{falls } s_j = t_i \end{cases} \\ c &:= (a \dot{-} b)\downarrow \\ d &:= (b \dot{-} a)\downarrow \end{aligned}$$

Sei weiterhin k der größte gemeinsame Teiler aller c_i und d_j sowie c und d, das heißt es existieren c_i', d_j', c' und d', so dass $c_i = (k * c_i')\downarrow$ für alle $i \in \{1, \ldots, n\}$ und $d_j = (k * d_j')\downarrow$ für alle $j \in \{1, \ldots, m\}$ sowie $c = (k * c')\downarrow$ und $d = (k * d')\downarrow$. Seien dann $t_{1_p}' := \sum_{\{i|1 \leq i \leq n, c_i \neq 0\}} c_i' t_i + c'$ und $t_{2_p}' = \sum_{\{j|1 \leq j \leq m, d_j \neq 0\}} d_j' s_j + d'$. Weiterhin sei

$$t_p := \begin{cases} 0 & \text{falls } c_i = 0 \text{ für alle } i \in \{1, \ldots, n\}, c = 0 \\ \sum_{\{i|1 \leq i \leq n, c_i \neq 0\}} c_i t_i + c & \text{falls } d_j = 0 \text{ für alle } j \in \{1, \ldots, m\}, d = 0 \\ k\left(t_{1_p}' \dot{-} t_{2_p}'\right) & \text{sonst} \end{cases}$$

Dann ist t_p ein Term-Polynom in PNF, dessen Termdarstellung in *spec* mit *PNFDefCond*(t_p) aus t nicht-induktiv ableitbar ist (Anwendung von *A5*, *A6*, *L6*, *L9* und *L10*).

5. $t = f(t_1, \ldots, t_n)$ mit $f \notin \{+, *, s, \dot{-}\}$: Das Polynom $t_p := 1(f(t_1, \ldots, t_n))$ ist in PNF und seine Termdarstellung in *spec* mit *PNFDefCond*(t) = $\{f(t_1, \ldots, t_n)\}$ aus t nicht-induktiv ableitbar, da $[t_p]_T = t$ gilt. □

Der Beweis von Satz 2.1 ist konstruktiv und lässt daher die unmittelbare Ableitung eines Konstruktionsverfahrens für die Bestimmung eines aus einem Term nicht-induktiv ableitbaren Term-Polynoms in PNF zu.

Beispiele: Sei eine Ordnung auf den Termen gegeben, die diese im Wesentlichen nach der Multimenge der Variablen, der Termlänge, dem Topsymbol und schließlich lexikographisch nach den Teiltermen ordnet.

Der Term $((x + (3 * y)) + 5) + (x + (x * y))$ ist entsprechend dem Verfahren aus Satz 2.1 mit $\emptyset$ nicht-induktiv ableitbar zu dem Polynom $2x + 3y + xy + 5$, das in PNF ist.

Der Term $((3 * x) + (f(z) * 2)) + (3 + (y \dot{-} x))$ ist entsprechend dem Verfahren aus Satz 2.1 mit $\{\neg\mathrm{def}(f(z))\}$ nicht-induktiv ableitbar zu dem Polynom $3x+2f(z)+(y \dot{-} x) + 3$, das in PNF ist.

2.4. Literale in Polynomdarstellung

Auch für Literale der Form l *op* r, wobei $op \in \{\neq, =, \leq\}$ ist und l und r Terme der Sorte Nat sind, kann eine Polynomdarstellung definiert werden. Diese beinhaltet, dass die beiden Terme l und r Term-Polynome sind.

Definition 2.7 *Ein* Polynom-Literal *ist ein Literal λ der Form l op r, bei dem l und r Polynome der Form $l = \sum_{i=1}^{n} a_i t_i + a$ bzw. $r = \sum_{j=1}^{m} b_j s_j + b$ sind*

Analog zu Term-Polynomen kann auch für Polynom-Literale eine Normalform definiert werden. Diese besteht zunächst daraus, dass die beiden Polynome l und r in PNF sind. Darüber hinaus können hier jedoch noch weitere Forderungen gestellt werden. Die Summanden können über beiden Seiten paarweise disjunkt gemacht werden, indem bei gleichen Summanden auf der Seite mit dem größeren Koeffizienten der kleinere Koeffizient abgezogen wird. Dasselbe kann auch mit den Konstanten gemacht werden, so dass nur noch eine der Konstanten ungleich 0 ist. Zuletzt können mittels der Distributivität gemeinsame Koeffizienten ausgeklammert und anschließend eliminiert werden. Hierbei müssen jedoch die Konstanten gesondert behandelt werden, falls sie den gemeinsamen Teiler der Summanden nicht enthalten. Die Behandlung ist abhängig von der Literalart. Daraus ergibt sich folgende Definition für die Polynom-Normalform erster Stufe für Literale:

Definition 2.8 *Ein Polynom-Literal $\lambda = \sum_{i=1}^{n} a_i t_i + a$ op $\sum_{j=1}^{m} b_j s_j + b$ ist in* Polynom-Normalform erster Stufe (PNF1)*, falls gilt:*

1. *$op \in \{\neq, =, \leq\}$*
2. *die beiden Polynome $\sum_{i=1}^{n} a_i t_i + a$ und $\sum_{j=1}^{m} b_j s_j + b$ sind in PNF*
3. *alle t_i, s_j sind paarweise verschieden*
4. *$a = 0$ oder $b = 0$*
5. *alle a_i, b_j haben keinen gemeinsamen Teiler außer 1*

Definition 2.9 *Sei spec eine Spezifikation mit Konstruktoren. Ein Literal λ heißt in spec mit einer Klausel Θ zu einer Klauselmenge $K = \{\Gamma_1, \ldots, \Gamma_n\}$ mit $\Gamma_i = \lambda_{i,1} \ldots \lambda_{i,k_i}$ für alle $i \in \{1, \ldots, n\}$ nicht-induktiv ableitbar, wenn das Ziel $\langle \lambda \,;\, w \rangle$ sich durch einmalige Anwendung der* QuodLibet*-Inferenzregel* Literal Hinzufügen *mit Θ und anschließende Anwendung von nicht-induktiven Inferenzregeln (wobei nur in spec gültige Lemmata und Axiome für applikative Inferenzregeln verwendet werden) in die Ziele*

$$\langle \Lambda_1, \lambda \,;\, w \rangle, \ldots, \langle \Lambda_n, \lambda \,;\, w \rangle, \langle \Lambda, \lambda_{1,1}, \ldots, \lambda_{1,k_1} \,;\, w \rangle, \ldots, \langle \Lambda, \lambda_{j,1}, \ldots, \lambda_{j,k_j} \,;\, w \rangle$$

überführen lässt. Dabei ist $\Lambda_1, \ldots, \Lambda_n, \Lambda$ die aus Θ resultierende Fallunterscheidung.

Besteht dabei die Klauselmenge K nur aus einer Klausel mit einem Literal, das heißt $K = \{\lambda_{1,1}\}$, so sagt man auch, dass λ in spec mit Θ zu $\lambda_{1,1}$ nicht-induktiv ableitbar ist.

Bemerkung: Ist ein Literal λ in *spec* mit Θ zur Klauselmenge $K = \emptyset$ ableitbar, so bedeutet dies, dass für den Beweis des Ziels $\langle \lambda \,;\, w \rangle$ nur noch die aus der Fallunterscheidung resultierenden Ziele $\langle \Lambda_1, \lambda \,;\, w \rangle, \ldots, \langle \Lambda_n, \lambda \,;\, w \rangle$ bewiesen werden müssen.

Ist ein Literal λ in *spec* mit Θ zur Klauselmenge $K = \{\Box\}$ ableitbar, so bedeutet dies, dass das Literal λ aus jedem Ziel entfernt werden kann, sofern zuvor eine Fallunterscheidung nach Θ gemacht wird.

Auch hier lässt sich, analog zu Lemma 2.1, die nicht-induktive Ableitbarkeit verketten.

Lemma 2.4 *Sei spec eine Spezifikation mit Konstruktoren und das Literal λ in spec mit Θ_1 zur Klauselmenge $K = \{\Lambda_1, \ldots, \Lambda_n\}$ mit $\Lambda_i = \lambda_{i,1} \ldots \lambda_{i,k_i}$ für $i \in \{1, \ldots, n\}$ nicht-induktiv ableitbar. Sei weiterhin das Literal $\lambda_{j,k}$ mit $j \in \{1, \ldots, n\}$ und $k \in \{1, \ldots, k_j\}$ in spec mit Θ_2 zu einer Klauselmenge $K_2 = \{\Delta_1, \ldots, \Delta_m\}$ nicht-induktiv ableitbar und $\Gamma_l := \lambda_{j,1} \ldots \lambda_{j,k-1}, \Delta_l, \lambda_{j,k} \ldots \lambda_{j,k_i}$ für $l \in \{1, \ldots, m\}$. Dann ist λ in spec mit $\Theta = \Theta_1 \cup \Theta_2$ zur Klauselmenge $\{\Lambda_1, \ldots, \Lambda_{j-1}, \Gamma_1, \ldots, \Gamma_m, \Lambda_{j+1}, \ldots, \Lambda_n\}$ nicht-induktiv ableitbar.*

Beweis: Analog Lemma 2.1. □

Lemma 2.5 *Sei spec eine Spezifikation mit Konstruktoren und linearer Arithmetik. Jedes Literal λ der Form $\lambda = l \;\; op \;\; r$, wobei $op \in \{\neq, =, \leq\}$ ist und l und r Terme der Sorte* Nat *sind, ist in spec mit einer Klausel PNFDefCond(λ) zu einer Klauselmenge K_p nicht-induktiv ableitbar, in der alle Literale in PNF1 sind.*

Beweis: Seien $l_N = \sum_{i=1}^{n} a_i t_i + a$ und $r_N = \sum_{j=1}^{m} b_j s_j + b$ Term-Polynome in PNF, die sich in *spec* mit *PNFDefCond*(l) bzw. *PNFDefCond*(r) aus l bzw. r ableiten lassen. Diese existieren nach Satz 2.1. Sei $\lambda' := l_N \;\; op \;\; r_N$. Dann ist λ' in *spec* mit *PNFDefCond*(λ) := *PNFDefCond*(l) $\cup$ *PNFDefCond*(r) aus λ nicht-induktiv ableitbar. Es sind dann in λ' alle t_i paarweise verschieden, ebenso alle s_j. Jedoch kann es i und j mit $t_i = s_j$ geben. Diese müssen wie folgt zusammengefasst werden: Seien $c_j := (b_j \dot{-} a_i)\downarrow$ und $d_i := (a_i \dot{-} b_j)\downarrow$ für alle i, j mit $t_i = s_j$ sowie $a' := (a \dot{-} b)\downarrow$ und $b' := (b \dot{-} a)\downarrow$. Sei weiterhin

$$\begin{aligned}\lambda'' \;:=\; & \sum_{\{i|1\le i\le n,\forall j:s_j\neq t_i\}} a_i t_i + \sum_{\{i|1\le i\le n,\exists j:(s_j=t_i\wedge d_i\neq 0)\}} d_i t_i + a' \;\; op \\ & \sum_{\{j|1\le j\le m,\forall i:s_j\neq t_i\}} b_j s_j + \sum_{\{j|1\le j\le m,\exists i:(s_j=t_i\wedge c_j\neq 0)\}} c_j s_j + b' \\ =:\; & \sum_{i=1}^{n'} a'_i t'_i + a' \;\; op \sum_{j=1}^{m'} b'_j s'_j + b'\end{aligned}$$

λ'' erfüllt nun bereits die ersten vier Bedingungen der PNF1. Außerdem ist λ'' in *spec* mit *PNFDefCond*(λ) aus λ' (Anwendung von *L11* und *L12*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar.

Um auch die fünfte Bedingung zu erfüllen, werden gemeinsame Faktoren aus den Koeffizienten von λ'' zunächst ausgeklammert (Anwendung von *L6*) und anschließend eliminiert. Sei dazu g der größte gemeinsame Teiler aller a'_i und b'_j, das heißt $a'_i = (g * a''_i)\downarrow$ für alle $i \in \{1,\dots,n'\}$ und $b'_j = (g * b''_j)\downarrow$ für alle $j \in \{1,\dots,m'\}$. Seien weiterhin $a'', a''', b'', b''' \in \mathcal{N}$, so dass $a' = (g * a'' + a''')\downarrow$ und $b' = (g * b'' + b''')\downarrow$ sowie $a''' < g$ und $b''' < g$ erfüllt sind. Je nach Literalart wird dann wie folgt verfahren:

- *op* $= \neq$:
  Gilt $a''' = 0$ und $b''' = 0$, so ist $\lambda_p := \sum_{i=1}^{n'} a''_i t'_i + a'' \neq \sum_{j=1}^{m'} b''_j s'_j + b''$ ein Literal in PNF1, das in *spec* aus λ'' mit *PNFDefCond*(λ) (Anwendung von *L17*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar ist.

  Anderenfalls ist $\emptyset$ eine Klauselmenge in PNF1, die in *spec* mit *PNFDefCond*(λ) aus λ'' (Anwendung der Inferenzregel Nicht-induktive Subsumption mit *L16*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar ist.

- *op* $= \leq$:
  Gilt $a''' = 0$, so ist $\lambda_p := \sum_{i=1}^{n'} a''_i t'_i + a'' \leq \sum_{j=1}^{m'} b''_j s'_j + b''$ ein Literal in PNF1, das in *spec* aus λ mit *PNFDefCond*(λ) (Anwendung von *L14*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar ist.

  Anderenfalls ist $\lambda_p := \sum_{i=1}^{n'} a''_i t'_i + (a'' + 1)\downarrow \leq \sum_{j=1}^{m'} b''_j s'_j + b''$ ein Literal in PNF1, das in *spec* aus λ mit *PNFDefCond*(λ) (Anwendung von *L15*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar ist.

- *op* $= =$:
  Gilt $a''' = 0$ und $b''' = 0$, so ist $\lambda_p := \sum_{i=1}^{n'} a''_i t'_i + a'' = \sum_{j=1}^{m'} b''_j s'_j + b''$ ein Literal in PNF1, das in *spec* aus λ'' mit *PNFDefCond*(λ) (Anwendung von *L17*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar ist.

  Anderenfalls ist die Menge $\{\square\}$ eine Klauselmenge in PNF1, die in *spec* mit *PNFDefCond*(λ) aus λ'' (Applikative Literal-Beseitigung mit *L16*) und damit nach Lemma 2.4 aus λ nicht-induktiv ableitbar ist. $\square$

Beispiel: An das Beispiel zu Satz 2.1 anknüpfend, sei folgendes Literal gegeben:

$$((x + (3 * y)) + 5) + (x + (x * y)) = ((3 * x) + (f(z) * 2)) + (3 + (y \dot{-} x))$$

Dieses Literal ist entsprechend dem Verfahren aus Satz 2.1 mit $\{\neg\mathrm{def}\,(f(z))\}$ nicht-induktiv ableitbar zum Polynom-Literal

$$3y + xy + 2 = x + 2f(z) + (y \dot{-} x) \quad ,$$

das in PNF1 ist.

Diese Polynom-Normalform erster Stufe lässt sich auch auf negierte $\leq$-Literale sowie eine bestimmte Art von Ordungsliteralen erweitern. Dies geschieht, indem diese Literale zunächst in $\leq$-Literale umgewandelt werden und anschließend mit Lemma 2.5 zu einer Klauselmenge in PNF1 abgeleitet werden.

Lemma 2.6 *Sei spec eine Spezifikation mit Konstruktoren und linearer Arithmetik. Dann ist in spec jedes Literal λ der Form $\lambda = l \not\leq r$, $\lambda = l < r$ oder $\lambda = l \not< r$ wobei l und r Terme der Sorte Nat sind, mit einer Klausel PNFDefCond(λ) zu einem Literal der Form $l' \leq r'$ nicht-induktiv ableitbar.*

Beweis:

1. Habe λ die Form $\lambda = l \not\leq r$. Dann ist λ in *spec* mit *PNFDefCond*(λ) unter Anwendung von *L18* zu $r + 1 \leq l$ nicht-induktiv ableitbar.
2. Habe λ die Form $\lambda = l < r$. Dann ist λ in *spec* mit *PNFDefCond*(λ) unter Anwendung von *L19* zu $l + 1 \leq r$ nicht-induktiv ableitbar.
3. Habe λ die Form $\lambda = l \not< r$. Dann ist λ in *spec* mit *PNFDefCond*(λ) unter Anwendung von *L20* zu $r \leq l$ nicht-induktiv ableitbar. □

Satz 2.2 *Sei spec eine Spezifikation mit Konstruktoren und linearer Arithmetik. Jedes Literal λ der Form $\lambda = l$ op r, wobei op $\in \{\neq, =, \leq, \not\leq, <, \not<\}$ ist und l und r Terme der Sorte Nat sind , ist in spec mit einer Klausel PNFDefCond(λ) zu einer Klauselmenge K_p nicht-induktiv ableitbar, in der alle Literale in PNF1 sind.*

Beweis: Folgt unmittelbar aus Lemma 2.6 und Lemma 2.5. □

Von der Polynom-Normalform erster Stufe ausgehend, wird die Polynom-Normalform zweiter Stufe bestimmt, indem für Minuszeichen im linearen Anteil des Literals eine Fallunterscheidung gemacht wird. Dies resultiert in einer konjunktiv verknüpften Menge von Klauseln, in deren linearem Anteil keine Minuszeichen mehr vorkommen.

Definition 2.10 *Die* Anzahl der Minuszeichen im linearen Anteil *eines Terms t, bezeichnet mit $mp(t)$, ist wie folgt rekursiv definiert:*

$$mp(f(t_1, \ldots, t_n)) = \begin{cases} \sum_{i=1}^{n} mp(t_i) & \text{falls } f \in \{+, s\} \\ & \text{oder } f = * \text{ und } t_1 \in \mathbb{N} \text{ oder } t_2 \in \mathbb{N} \\ \sum_{i=1}^{n} mp(t_i) + 1 & \text{falls } f = \dot{-} \\ 0 & \text{sonst} \end{cases}$$

Die Anzahl der Minuszeichen im linearen Anteil eines Literals $\lambda = l\ op\ r$ ist die Summe aus $mp(l)$ und $mp(r)$ und wird als $mp(\lambda)$ bezeichnet.

Definition 2.11 *Ein Literal $\lambda = l\ op\ r$ ist in Polynom-Normalform zweiter Stufe (PNF2), falls es in PNF1 ist und somit die Form $\lambda = \sum_{i=1}^{n} a_i t_i + a\ op\ \sum_{j=1}^{m} b_j s_j + b$ hat und $top(t_i) \neq \dot{-}$ für alle $i \in \{1, \ldots, n\}$ und $top(s_j) \neq \dot{-}$ für alle $j \in \{1, \ldots, m\}$ gilt.*

Eine Klausel $\Lambda = \lambda_1 \ldots \lambda_l$ für $l \in \mathbb{N}$ ist in Polynom-Normalform zweiter Stufe (PNF2), falls alle Literale $\lambda_1, \ldots, \lambda_l$ in PNF2 sind.

Eine Klauselmenge $K = \{\Lambda_1, \ldots, \Lambda_k\}$ für $k \in \mathbb{N}$ ist in Polynom-Normalform zweiter Stufe (PNF2), falls alle Klauseln $\Lambda_1, \ldots, \Lambda_l$ in PNF2 sind.

Für den Beweis, dass bestimmte Literale zu einer Klauselmenge in PNF2 ableitbar sind, wird noch die folgende Definition benötigt:

Definition 2.12 *Für zwei Klauseln $\Gamma = \gamma_1 \ldots \gamma_n$ und $\Delta = \delta_1 \ldots \delta_m$ bezeichne $\Gamma\Delta$ die Konkatenation der beiden Klauseln, das heißt die Klausel $\gamma_1 \ldots \gamma_n \delta_1 \ldots \delta_m$. Die* disjunktive Verknüpfung *$K_1 \vee K_2$ zweier Klauselmengen K_1 und K_2 ist dann wie folgt definiert:*

$$K_1 \vee K_2 := \{\Gamma\Delta \mid \Gamma \in K_1, \Delta \in K_2\}$$

Satz 2.3 *Sei spec eine Spezifikation mit Konstruktoren und linearer Arithmetik. Jedes Literal $\lambda = \sum_{i=1}^{n} a_i t_i + a\ op\ \sum_{j=1}^{m} b_j s_j + b$ in PNF1 ist in spec mit einer Klausel PNFDefCond(λ) zu einer Klauselmenge K_p nicht-induktiv ableitbar, die in PNF2 ist.*

Beweis: Vollständige Induktion über die Anzahl $mp(\lambda)$ der Minuszeichen im linearen Anteil von λ

- **Induktionsanfang**: $mp(\lambda) = 0$
  Da λ keine Minuszeichen im linearen Anteil enthält, enthält λ auch keine Minuszeichen an Top-Position eines Summanden und ist somit in PNF2. Somit ist die Klauselmenge $\{\lambda\}$ ebenfalls in PNF2.

- **Induktionsschritt**: $mp(\lambda) > 0$
  Fallunterscheidung nach dem Summanden, in dem ein Minuszeichen vorkommt:

  1. t_k enthält ein Minuszeichen für ein $k \in \{1, \ldots, n\}$: dieses Minuszeichen muss das Top-Symbol von t_k sein, da t_k sonst keinen linearen Anteil hätte (weil λ in PNF1 ist); sei deshalb $t_k = t'_k \dot{-} t''_k$. Seien weiterhin

$$\begin{aligned}
\lambda_1 &= t''_k \not\leq t'_k \\
\lambda_2 &= \sum_{i \in \{1,\ldots,n, i \neq k\}} a_i t_i + a_k t'_k + a\ op\ \sum_{j=1}^{m} b_j s_j + a_k t''_k + b \\
\lambda_3 &= t''_k \leq t'_k \\
\lambda_4 &= \sum_{i \in \{1,\ldots,n, i \neq k\}} a_i t_i + a\ op\ \sum_{j=1}^{m} b_j s_j + b
\end{aligned}$$

und K_l für $l \in \{1,2,3,4\}$ in *spec* mit *PNFDefCond*(λ_l) aus λ_l nicht-induktiv ableitbare Klauselmengen, deren Literale in PNF1 sind. Diese existieren nach Satz 2.2. Besteht die Klauselmenge K_l aus genau einer Klausel mit genau einem Literal λ'_l, so sei K'_l eine in *spec* mit *PNFDefCond*(λ'_l) aus λ'_l nicht-induktiv ableitbare Klauselmenge in PNF2. Besteht die Klauselmenge K_l jedoch nur aus der leeren Klausel oder ist selbst leer, so sei $K'_l = K_l$. Eine andere Möglichkeit für K'_l gibt es nicht (siehe Ergebnisse von Satz 2.2). Alle Klauselmengen K'_l für $l \in \{1,2,3,4\}$ sind dann in PNF2. Die abgeleiteten Klauselmengen K'_l existieren nach Induktionsvoraussetzung, da die Anzahl der Minuszeichen im linearen Anteil von λ'_l jeweils um mindestens eins kleiner ist als die von λ. Dann ist die Klauselmenge $(K'_1 \vee K'_2) \cup (K'_3 \vee K'_4)$ eine in *spec* mit *PNFDefCond*$(\lambda) \supseteq \bigcup_{i\in\{1,\ldots,4\}} (\mathit{PNFDefCond}(\lambda_l) \cup \mathit{PNFDefCond}(\lambda'_l))$ aus λ nicht-induktiv ableitbare Klauselmenge in PNF2 (Fallunterscheidung mit Literal Hinzufügen nach dem Literal $t''_k \leq t'_k$ und Anwendung von *L21* und *L22* bzw. *L23*).

2. $s_k = s'_k \dot{-} s''_k$ für ein $k \in \{1,\ldots,m\}$: analog □

Beispiel: Anknüpfend an das Beispiel zu Lemma 2.5 ist das Literal

$$3y + xy + 2 = x + 2f(z) + (y \dot{-} x) \quad ,$$

das in PNF1 ist, entsprechend dem Verfahren aus Satz 2.3 mit $\{\neg\mathrm{def}\,(f(z))\}$ nicht-induktiv ableitbar zur Klauselmenge

$$\{y + 1 \leq x, 2y + xy + 2 = 2f(z)\}, \{x \leq y, 3y + xy + 2 = x + 2f(z)\} \quad ,$$

in der alle Literale in PNF2 sind.

Da das Entscheidungsverfahren, das im folgenden Kapitel vorgestellt wird, sehr stark auf die Verarbeitung von $\leq$-Literalen und Ungleichungen ausgelegt ist, ist es manchmal nützlich, Gleichungen in zwei $\leq$-Literale umzuwandeln. Dazu wird die Polynom-Normalform dritter Stufe wie folgt definiert:

Definition 2.13 *Ein Literal $\lambda = l \ op \ r$ ist in Polynom-Normalform dritter Stufe (PNF3), falls es ein PNF2 ist und $op \in \{\neq, \leq\}$ ist.*

Dass eine Umwandlung eines Literals in PNF2 in die PNF3 möglich ist, besagt der folgende Satz:

Satz 2.4 *Sei spec eine Spezifikation mit Konstruktoren und linearer Arithmetik. Jedes Literal $\lambda = l \ op \ r$ in PNF2 ist in spec mit einer Klauselmenge PNFDefCond(λ) zu einer Klauselmenge K_p nicht-induktiv ableitbar, in der jedes Literal in PNF3 ist.*

Beweis: Ist $op \in \{\neq, \leq\}$, so ist λ bereits in PNF3. Sei $op = =$. Dann ist $\{l \leq r, r \leq l\}$ eine in *spec* aus λ nicht-induktiv ableitbare Klauselmenge, deren Literale in PNF3 sind. Die Ableitbarkeit kann wie folgt gezeigt werden: Zunächst wird mittels der Inferenzregel Nicht-induktive Subsumption mit *L24* das Ziel $\langle l = r \ ; \ w\rangle$ in die beiden Ziele

$$\langle l \leq r, l = r \ ; \ w\rangle$$
$$\langle r \leq l, l \not\leq r, l = r \ ; \ w\rangle$$

abgeleitet (zusätzliche Definiertheitsziele können sofort mit *L1* und *L2* sowie den Literalen aus $\mathit{PNFDefCond}(\lambda)$ gezeigt werden). Das erste Ziel kann dann mittels der Inferenzregel Applikative Literal-Beseitigung mit *L25* in das Ziel $\langle l \leq r\,;\ w\rangle$ abgeleitet werden. Das zweite Ziel kann mittels Applikative Literal-Beseitigung mit *L26* in das Ziel $\langle r \leq l, l = r\,;\ w\rangle$ abgeleitet werden. Diese kann wiederum mittels der Inferenzregel Applikative Literal-Beseitigung mit *L25* in das Ziel $\langle r \leq l\,;\ w\rangle$ abgeleitet werden. □

Beispiel: Das Literal $2y + xy + 2 = 2f(z)$ aus dem Beispiel zu Satz 2.3 ist in PNF2 und kann, entsprechend dem Verfahren aus Satz 2.4, mit $\{\neg\mathrm{def}\,(f(z))\}$ nicht-induktiv zur Klauselmenge

$$\{2y + xy + 2 \leq 2f(z)\}, \{2f(z) \leq 2y + xy + 2\} \quad ,$$

abgeleitet werden, in der alle Literale in PNF3 sind.

Die Normalform für Polynome-Literale wird somit in drei Stufen definiert. Dies erleichtert die Definition, da diese sukzessive aufgebaut werden kann. Außerdem kann in der im nächsten Kapitel beschriebenen Inferenzregel für die Normalisierung ebenfalls ein stufenweises Vorgehen realisiert werden. Dies hat insbesondere den Vorteil, dass je nach Notwendigkeit eine Aufspaltung in mehrere Fälle vermieden werden kann. Eine genauere Beschreibung zum Einsatz der verschiedenen Stufen der Normalisierung ist in der Beschreibung der neuen Taktiken im Abschnitt 3.3 enthalten.

3. Integration eines Entscheidungsverfahrens für lineare Arithmetik

3.1. Das Entscheidungsverfahren

Entscheidungsverfahren für lineare Arithmetik über den natürlichen oder ganzen Zahlen sind sehr aufwändig im Vergleich zu Entscheidungsverfahren für lineare Arithmetik über rationalen Zahlen. Dies liegt daran, dass das Problem für natürliche oder ganze Zahlen NP-vollständig ist und daher kein Verfahren bekannt ist, das eine lineare Komplexität hat. Für rationale Zahlen sind jedoch Verfahren mit linearer Komplexität bekannt. Aus diesem Grund entschieden sich beispielsweise Boyer und Moore [BM88] dazu, ein Entscheidungsverfahren für lineare Arithmetik über rationalen Zahlen als hinreichendes Kriterium für die Erfüllbarkeit von Formeln der linearen Arithmetik über den ganzen Zahlen zu verwenden. Wenn eine (quantorenfreie) lineare Formel keine Lösung in den rationalen Zahlen besitzt, so besitzt Sie auch keine Lösung in den ganzen Zahlen. Umgekehrt gilt dies natürlich nicht: besitzt eine lineare Formel eine Lösung in den rationalen Zahlen, so heißt das nicht, dass sie auch eine Lösung in den ganzen Zahlen besitzt. Dieser Fall tritt jedoch in der Praxis sehr selten auf, weshalb es durchaus vertretbar sein kann, diese Schwäche in Kauf zu nehmen. Man muss sich dann jedoch bewusst sein, dass man damit kein Entscheidungsverfahren für die lineare Arithmetik der ganzen Zahlen einsetzt. Somit nimmt man in Kauf, dass durchaus auch reine Formeln der linearen Arithmetik der ganzen Zahlen mit den aufwändigeren Mitteln außerhalb des „Entscheidungsverfahrens“ entschieden werden müssen.

Eine andere Möglichkeit, diesen Fall zu behandeln, zeigt Pugh in seiner Beschreibung des Omega-Tests [Pug92]. Die grundlegende Idee ist, in einem dieser seltenen Fälle, in denen das Entscheidungsverfahren für rationale Zahlen nicht ausreicht, eine aufwändigere Berechnung zu starten, um durch systematische Durchsuchung des rationalen Lösungsbereichs ganzzahlige Lösungen aufzufinden. Man hofft dann, dass dies nicht allzu häufig auftritt. Im Extremfall kann dies allerdings zu einer sehr ungünstigen Laufzeit führen, wie Pugh in seinem Beispiel „An omega test nightmare“ [Pug92] demonstriert, in dem gezeigt wird, dass die Laufzeit von der Größe der Koeffizienten abhängt.

Wie in Abschnitt 1.3 beschrieben, ermöglicht ein Entscheidungsverfahren, die Erfüllbarkeit eines gegebenen Ausdrucks zu bestimmen. Da in QuodLibet jedoch nicht die Erfüllbarkeit, sondern die Gültigkeit eines (quantorenfreien) Ausdrucks gezeigt werden soll, müssen die Ausdrücke vor Anwendung des Entscheidungsverfahrens ne-

giert werden. Dies ist leicht möglich, da in QuodLibet mit Klauseln, das heißt disjunktiv verknüpften Literalen, gearbeitet wird und die meisten Entscheidungsverfahren auf Aussagen in Form von konjunktiv verknüpften Literalen beschrieben sind. Somit müssen lediglich die einzelnen Literale negiert werden. Denn liefert das Entscheidungsverfahren das Ergebnis, dass der Ausdruck $\Lambda = \lambda_1 \wedge \lambda_2 \wedge \ldots \wedge \lambda_n$ unerfüllbar ist (das heißt, in keinem Datenmodell und von keiner Belegung erfüllt wird), so heißt dies, dass die Klausel $\neg\Lambda = \neg\lambda_1 \vee \neg\lambda_2 \vee \ldots \vee \neg\lambda_n$ gültig ist. Ist Λ erfüllbar, so ist $\neg\Lambda$ nicht gültig. Bei der folgenden Beschreibung des verwendeten Entscheidungsverfahrens wird daher ebenfalls von konjunktiv verknüpften Literalen ausgegangen und einer Entscheidung über die Erfüllbarkeit ausgegangen. Für den Einsatz in QuodLibet wird das Entscheidungsverfahren dann so modifiziert, dass ohne Negation direkt auf den Literalen der Klauseln gearbeitet werden kann.

In dieser Arbeit wird der Ansatz verwendet, der auch dem Omega-Test zugrunde liegt. Das heißt, basierend auf einem Entscheidungsverfahren für rationale Zahlen wird ein Entscheidungsverfahren für natürliche Zahlen entwickelt, indem der rationale Lösungsraum auf natürlichzahlige Lösungen durchsucht wird. Als grundlegendes Entscheidungsverfahren für rationale Zahlen wird die *Fourier-Motzkin-Variablenelimination* [DE73] verwendet. Dieses Verfahren ist für Mengen von $\leq$-Literalen definiert und basiert auf der einfachen Idee, zwei $\leq$-Literale so mit Konstanten zu multiplizieren, dass bei einer anschließenden Addition der beiden Literale eine Variable eliminiert wird. Hierfür werden zunächst alle Ungleichungen in die Form $\sum_{j=1}^{n} a_j x_j + a \leq \sum_{i=1}^{m} b_i y_i + b$ gebracht, wobei alle a_j, a, b_i und b positiv sind.

Wenn dann im Literal $s_1 \leq t_1$ die Variable x in s_1 mit dem Faktor c_1 und im Literal $s_2 \leq t_2$ in t_2 mit dem Faktor c_2 vorkommt, dann können diese Literale zu $c_2 s_1 + c_1 s_2 \leq c_2 t_1 + c_1 t_2$ addiert werden. In diesem neuen Literal kommt auf beiden Seiten die Variable x mit dem Faktor $c_1 * c_2$ vor. Durch anschließende Normalisierung wird damit die Variable x aus dem neuen Literal eliminiert. Eine Belegung, die $s_1 \leq t_1$ und $s_2 \leq t_2$ erfüllt, erfüllt auch stets das neue Literal.

Eine Menge von $\leq$-Literalen kann nun in eine neue Menge überführt werden, in der die Variable x nicht mehr vorkommt. Dazu wird für jedes Paar von Literalen in der ursprünglichen Menge, bei dem die Variable x bei einem Literal auf der linken und bei einem auf der rechten Seite vorkommt, eine Variablenelimination durchgeführt. Die Menge der Literale, die dadurch entstehen, wird zusammen mit den Literalen, in denen die Variable x nicht vorkommt, als neue Ausgangsmenge verwendet, um die nächste Variable zu eliminieren. Kommt eine Variable x nur auf linken oder nur auf rechten Seiten vor, so können alle Literale, in denen die Variable x vorkommt, entfernt werden. Dies ist möglich, weil stets ein hinreichend kleiner oder großer Wert für x gefunden werden kann, so dass diese Literale erfüllt sind.

Somit kann effektiv jede Variable vollständig aus der gesamten Literalmenge eliminiert werden. Führt man dies für alle Variablen durch, die in den Literalen vorkommen, so erhält man letztendlich nur noch variablenfreie Literale. Hat mindestens eines dieser Literale die offensichtlich widersprüchliche Form $c \leq 0$ mit $c \in \mathbb{Q}$ und $c > 0$, so ist die Klausel unerfüllbar. Enthält die Klausel jedoch kein solches Literal, sondern nur noch offensichtlich allgemeingültige Literale der Form $0 \leq c$ mit $c \in \mathbb{Q}$

und $c \geq 0$, so ist die Klausel erfüllbar. Einen anderen Fall gibt es nicht, weshalb das Verfahren stets zu einer Entscheidung führt.

Dieses Verfahren muss für die Anwendung im Rahmen von QuodLibet noch angepasst werden. Zum einen dürfen hier Variablen nicht komplett eliminiert werden, da die Variablen der linearen Arithmetik auch ganze Terme repräsentieren können. Würde man nun diese Terme komplett aus allen Literalen eliminieren, so ginge eine eventuell in den zu Variablen abstrahierten Termen enthaltene Information verloren. Kann mittels des Entscheidungsverfahrens die Unerfüllbarkeit nachgewiesen werden, ist dies natürlich unerheblich. Stellt das Entscheidungsverfahren jedoch die Erfüllbarkeit der Literalmenge fest, so müssen die Informationen aus den abstrahierten Termen verwendet werden, um eventuell doch noch die Unerfüllbarkeit nachweisen zu können. Dies liegt daran, dass für die Variablen der linearen Arithmetik, die einen Term außerhalb der linearen Arithmetik repräsentieren, nicht beliebige Werte eingesetzt werden können. Daher werden auch die ursprünglichen Literale weiterhin in der Literalmenge mitgeführt. Auch die Löschung von Literalen, wenn eine Variable nur auf einer Seite aller Literale vorkommt, wird nicht durchgeführt. Führt man dabei auf eine andere Weise Buch, welche Variablen bereits eliminiert worden sind, so führt das Verfahren nach wie vor zu einer Entscheidung.

Weiterhin ist eine Erweiterung des Verfahrens auf den Zahlenbereich der natürlichen Zahlen notwendig. Erweist sich eine Literalmenge als erfüllbar über den rationalen Zahlen, so wird anschließend die gefundene Lösungsmenge auf Lösungen in den natürlichen Zahlen durchsucht, indem vor dem letzten Schritt der Variablenelimination eine Fallunterscheidung durchgeführt wird. Dies geschieht, indem eine oder mehrere neue Literalmengen erzeugt werden, die alle möglichen Fälle abdecken, für die die zuletzt übrig gebliebenen Variablen Werte aus den natürlichen Zahlen annehmen könnten. Lässt sich in all diesen Fällen nachweisen, dass keine natürlichzahlige Lösung existiert, so ist die Unerfüllbarkeit der Literalmenge über den natürlichen Zahlen nachgewiesen. Anderenfalls ist die Literalmenge auch über den natürlichen Zahlen erfüllbar. Welche Möglichkeiten hierbei genau überprüft werden müssen, ist in [Pug92] beschrieben. Ein konkretes Beispiel wird im Rahmen der Beschreibung der Inferenzregel $\leq$-Fallunterscheidung im Abschnitt 3.2.5 erläutert.

Neben den $\leq$-Literalen gibt es in QuodLibet auch Ungleichungen, die durch die eingangs beschriebene Negation der Literale zu Gleichungen werden. Eine solche Gleichung $l = r$ könnte man in zwei $\leq$-Literale $l \leq r$ und $r \leq l$ umwandeln und das oben beschrieben Verfahren zur Lösung anwenden. Die in den Gleichungen enthaltene Information kann jedoch deutlich effizienter eingesetzt werden, indem die Gleichung nach einer Variablen aufgelöst wird und anschließend in den anderen Literalen eine entsprechende Ersetzung dieser Variablen vorgenommen wird. Dies ist jedoch nur möglich, wenn der Koeffizient der Variablen 1 ist. Um in einer Gleichung, in der keiner der Koeffizienten 1 ist, mindestens einen der Koeffizienten auf 1 zu bringen, wird das in [Knu81] beschriebene Verfahren verwendet. Dieses basiert auf dem euklidischen Algorithmus beruht und erreicht die Verkleinerung des minimalen Faktors in einem Literal dadurch, dass das Literal durch diesen minimalen Faktor dividiert wird, wobei die Ergebnisse der Division auf- bzw. abgerundet werden. Um die durch die Rundung entstehende Differenz auszugleichen wird eine zusätzliche Variable eingeführt. Anschließend wird die Gleichung nach der Variablen aufgelöst,

die ursprünglich den minimalen Faktor hatte und nun den Faktor 1 hat. Der Wert für diese Variable wird dann in die ursprüngliche Gleichung eingesetzt. Danach ist der minimale Faktor in der Gleichung kleiner dem ursprünglichen minimalen Faktor. Falls nötig, wird dieser Schritt so lange wiederholt, bis der minimale Faktor 1 ist.

Beispiel: Das Literal $2x+5y=7$, in dem keiner der Koeffizienten 1 ist, kann durch Division durch den minimalen Faktor 2 zu $x+2y+z=4$ umgeformt werden (auf der linken Seite wird abgerundet, auf der rechten Seite aufgerundet). Dieses resultierende Literal kann dann nach x aufgelöst werden und bestimmt somit einen Term, der für x im ursprünglichen Literal (und auch in alle anderen Literalen) eingesetzt werden kann. Das ursprüngliche Literal hat dann die Form $y+1=2z$. Der minimale Faktor ist nun 1, so dass diese Gleichung nun nach y aufgelöst und das Resultat in allen anderen Literalen für y eingesetzt werden kann. Damit wurden die Variable x und y aus allen Literalen eliminiert, die Variable z wurde neu eingeführt. Die Gesamtzahl der Variablen wurde somit um eins reduziert.

Die Realisierung des beschriebenen Entscheidungsverfahrens erfolgt in QuodLibet durch die Erweiterung der Inferenzregeln um einzelne Regeln für die Teilschritte des Entscheidungsverfahrens. Diese werden dann durch Taktiken angesteuert, die das eigentliche Verfahren realisieren. Hierbei ist eine enge Verzahnung mit den bereits vorhandenen Inferenzregeln und somit eine Erweiterung der vorhandenen Taktiken notwendig.

3.2. Inferenzregeln für das Entscheidungsverfahren

3.2.1. Vorbemerkungen

In den folgenden Abschnitten werden zunächst die einzelnen Inferenzregeln, die zur Realisierung des Entscheidungsverfahrens in QuodLibet integriert wurden, im Detail beschrieben. Die Beschreibung richtet sich dabei nach der in [Küh00, Anhang C] verwendeten Form. Das heißt insbesondere, dass die folgenden (Meta-)Variablen ohne weitere Erklärung verwendet werden:

1. m, n, j, k für natürliche Zahlen
2. t, u, v, l, r für Terme
3. p für Positionen
4. w für Gewichte
5. λ für Literale
6. Γ, Δ, Π für Klauseln

Außerdem wird mit $\Gamma[m]$ das m-te Literal in einer Klausel Γ bezeichnet.

Zur besseren Verständlichkeit wird zusätzlich zu jeder Inferenzregel ein Beispiel für die Anwendung der Regel gegeben. Die Variablen x, y und z sind in diesen Beispielen Konstruktorvariablen der Sorte Nat.

Weiterhin werden zu jeder Inferenzregel die Korrektheit und die Sicherheit gezeigt. Dies dient – wie in Abschnitt 1.2.2 beschrieben – dazu, die Korrektheit und Widerspruchskorrektheit des Gesamtsystems sicherzustellen. Für den Beweis der Sicherheit ist folgendes Lemma sehr nützlich, das besagt, dass eine Inferenzregel sicher ist, wenn jedes Teilziel, das bei der Anwendung der Inferenzregel erzeugt wird, durch hinzufügen von Literalen zum ursprünglichen Ziel entsteht.

Lemma 3.1 *(nach [Küh00, Lemma 5.1.2]) Wenn jede Instanz einer Inferenzregel die Form*

$$\frac{\langle \Gamma\ ;\ w\rangle}{\langle \Lambda_1, \Gamma\ ;\ w_1\rangle \ldots \langle \Lambda_n, \Gamma\ ;\ w_n\rangle} \qquad \textit{mit } \langle \Pi_1\ ;\ \hat{w}_1\rangle^{U_1}, \ldots, \langle \Pi_k\ ;\ \hat{w}_k\rangle^{U_k}$$

hat, wobei $n, k \in \mathbb{N}$, $\Lambda_1, \ldots, \Lambda_n$ Klauseln sind und $U_j \in \{\mathcal{I}, \mathcal{L}\}$ für $i = 1, \ldots, k$ gilt, dann ist die Inferenzregel sicher.

Beweis: Folgt unmittelbar aus Definition 1.8. □

Kann die Sicherheit nicht mit diesem Lemma gezeigt werden, so wird sie in den folgenden Beweisen auf Inferenzregeln zurückgeführt, deren Sicherheit bereits bewiesen wurde. Um das dafür benötigte Lemma beweisen zu können, wird zunächst folgendes Lemma gezeigt:

Lemma 3.2 *Sei das Ziel $\langle \Gamma\ ;\ w\rangle$ durch die aufeinander folgende Anwendung einer Folge von k Instanzen ($k \in \mathbb{N}$) von nicht-induktiven und sicheren Inferenzregeln auf $\langle \Gamma\ ;\ w\rangle$ oder bereits abgeleitete Teilziele unter Verwendung ausschließlich gültiger Lemmata in die Teilziele $\langle \Gamma_1\ ;\ w_1\rangle, \ldots, \langle \Gamma_n\ ;\ w_n\rangle$ ableitbar. Ist dann die Klausel Γ induktiv gültig, so sind auch die Klauseln $\Gamma_1, \ldots, \Gamma_n$ induktiv gültig.*

Beweis: Vollständige Induktion nach k.

Induktionsanfang: Ist $k = 0$, so ist $n = 1$ und $\Gamma_1 = \Gamma$. Da Γ induktiv gültig ist, ist auch Γ_1 induktiv gültig.

Induktionsschritt: Ist $k = l+1$, so werde durch aufeinander folgende Anwendung der ersten l Instanzen von nicht-induktiven und sicheren Inferenzregeln auf das Ziel $\langle \Gamma\ ;\ w\rangle$ oder bereits abgeleitete Teilziele unter Verwendung ausschließlich induktiv gültiger Lemmata das Ziel $\langle \Gamma\ ;\ w\rangle$ in die Teilziele $\langle \Gamma'_1\ ;\ w'_1\rangle, \ldots, \langle \Gamma'_{n'}\ ;\ w'_{n'}\rangle$ abgeleitet. Nach Induktionsvoraussetzung sind alle $\Gamma'_1, \ldots, \Gamma'_{n'}$ induktiv gültig. Leitet nun die $k$te Inferenzregel das Ziel $\langle \Gamma'_i\ ;\ w'_i\rangle$ zu den Teilzielen $\langle \Gamma''_1\ ;\ w''_1\rangle, \ldots, \langle \Gamma''_{n''}\ ;\ w''_{n''}\rangle$ ab, so sind, da auch die $k$te Inferenzregel sicher ist, auch alle $\Gamma''_1, \ldots, \Gamma''_{n''}$ induktiv gültig. Somit sind alle $\Gamma_1, \ldots, \Gamma_n$ induktiv gültig. □

Nun kann das Lemma gezeigt werden, das für die Beweise der Sicherheit der neuen Inferenzregeln verwendet werden kann.

Lemma 3.3 *Die nicht-applikative Inferenzregel*

$$\frac{\langle \Gamma\ ;\ w\rangle}{\langle \Gamma_1\ ;\ w_1\rangle \ldots \langle \Gamma_n\ ;\ w_n\rangle} \qquad \textit{falls}\ \text{Anwendbarkeitsbedingung}$$

ist sicher, falls es für jede Instanz der Inferenzregel eine Folge von Instanzen von nicht-induktiven und sicheren Inferenzregeln gibt, deren aufeinander folgende Anwendung auf $\langle \Gamma\ ;\ w\rangle$ *oder bereits abgeleitete Teilziele unter Verwendung ausschließlich induktiv gültiger Lemmata das Ziel* $\langle \Gamma;\ w\rangle$ *in die Teilziele* $\langle \Gamma_1;\ w_1\rangle, \ldots, \langle \Gamma_n;\ w_n\rangle$ *ableitet.*

Beweis: Sei

$$\frac{\langle \Gamma\ ;\ w\rangle}{\langle \Gamma_1\ ;\ w_1\rangle \ldots \langle \Gamma_n\ ;\ w_n\rangle}$$

eine Instanz der Inferenzregel und Γ induktiv gültig. Weiterhin gebe es eine Folge von Instanzen von nicht-induktiven und sicheren Inferenzregeln, deren aufeinander folgende Anwendung auf $\langle \Gamma\ ;\ w\rangle$ oder bereits abgeleitete Teilziele unter Verwendung ausschließlich induktiv gültiger Lemmata das Ziel $\langle \Gamma\ ;\ w\rangle$ in die Teilziele $\langle \Gamma_1\ ;\ w_1\rangle, \ldots, \langle \Gamma_n\ ;\ w_n\rangle$ ableitet. Nach Lemma 3.2 sind dann auch die Klauseln $\Gamma_1, \ldots, \Gamma_n$ induktiv gültig. Somit ist die Inferenzregel sicher. □

Die Beweise der Korrektheit der neuen Inferenzregeln werden ähnlich wie die der Sicherheit auf die bereits als korrekt bekannten Inferenzregeln zurückgeführt. Um das dafür benötigte Lemma beweisen zu können, wird zunächst folgendes Lemma gezeigt:

Lemma 3.4 *Sei* $\mathcal{A} \in DMod(spec)$*. Sei weiterhin das Ziel* $\langle \Gamma\ ;\ w\rangle$ *durch die aufeinander folgende Anwendung einer Folge von* k *Instanzen (*$k \in \mathbb{N}$*) von nicht-induktiven und korrekten Inferenzregeln auf* $\langle \Gamma\ ;\ w\rangle$ *oder bereits abgeleitete Teilziele unter Verwendung ausschließlich gültiger Lemmata in die Teilziele* $\langle \Gamma_1\ ;\ w_1\rangle, \ldots, \langle \Gamma_n\ ;\ w_n\rangle$ *ableitbar. Existiert dann ein* $\mathcal{A}$*-Gegenbeispiel der Form* $(\langle \Gamma\ ;\ w\rangle, \sigma, \varphi)$*, so existiert auch ein* $\mathcal{A}$*-Gegenbeispiel der Form* $(\langle \Gamma_i\ ;\ w_i\rangle, \tau, \psi)$*, für das* $(\langle \Gamma_i\ ;\ w_i\rangle, \tau, \psi) \precsim_{\mathcal{A}} (\langle \Gamma\ ;\ w\rangle, \sigma, \varphi)$ *gilt.*

Beweis: Vollständige Induktion über k.

Induktionsanfang: Ist $k = 0$, so ist $n = 1$ und $\Gamma_1 = \Gamma$ sowie $w_1 = w$. Dann ist $(\langle \Gamma_1\ ;\ w_1\rangle, \sigma, \varphi)$ ein $\mathcal{A}$-Gegenbeispiel, für das offensichtlich $(\langle \Gamma_1\ ;\ w_1\rangle, \sigma, \varphi) \precsim_{\mathcal{A}} (\langle \Gamma\ ;\ w\rangle, \sigma, \varphi)$ gilt.

Induktionsschritt: Ist $k = l + 1$, so wurde durch aufeinander folgende Anwendung der ersten l Instanzen von nicht-induktiven und korrekten Inferenzregeln auf das Ziel $\langle \Gamma;\ w\rangle$ oder bereits abgeleitete Teilziele unter Verwendung ausschließlich induktiv gültiger Lemmata das Ziel $\langle \Gamma;\ w\rangle$ in die Teilziele $\langle \Gamma'_1;\ w'_1\rangle, \ldots, \langle \Gamma'_{n'};\ w'_{n'}\rangle$ abgeleitet. Nach Induktionsvoraussetzung existiert dann ein $i' \in \{1, \ldots, m\}$ und ein $\mathcal{A}$-Gegenbeispiel der Form $(\langle \Gamma'_{i'}\ ;\ w'_{i'}\rangle, \tau, \psi)$, so dass $(\langle \Gamma'_{i'}\ ;\ w'_{i'}\rangle, \tau, \psi) \precsim_{\mathcal{A}} (\langle \Gamma\ ;\ w\rangle, \sigma, \varphi)$ gilt. Falls mit der $k$ten Inferenzregel eines der Ziele $\langle \Gamma'_j\ ;\ w'_j\rangle$ für $j \neq i'$ abgeleitet wird, so existiert dieses Gegenbeispiel auch anschließend. Wird jedoch $\langle \Gamma'_{i'}\ ;\ w'_{i'}\rangle$ zu den Teilzielen $\langle \Gamma''_1\ ;\ w''_1\rangle, \ldots, \langle \Gamma''_{n''}\ ;\ w''_{n''}\rangle$ abgeleitet, so existiert, da auch die

$k$te Inferenzregel korrekt ist, ein $i'' \in \{1, \ldots, 0\}$ und ein $\mathcal{A}$-Gegenbeispiel der Form $(\langle \Gamma''_{i''} \ ; \ w''_{i''} \rangle, \pi, \rho)$, so dass $(\langle \Gamma''_{i''} \ ; \ w''_{i''} \rangle, \pi, \rho) \precsim_{\mathcal{A}} (\langle \Gamma'_j \ ; \ w'_j \rangle, \tau, \psi)$ gilt. Da weiterhin $(\langle \Gamma'_{i'} ; \ w'_{i'} \rangle, \tau, \psi) \precsim_{\mathcal{A}} (\langle \Gamma ; \ w \rangle, \sigma, \varphi)$ gilt und wegen der Transitivität von $\precsim_{\mathcal{A}}$ gilt dann aber auch $(\langle \Gamma''_{i''} \ ; \ w''_{i''} \rangle, \pi, \rho) \precsim_{\mathcal{A}} (\langle \Gamma \ ; \ w \rangle, \sigma, \varphi)$. □

Mit Hilfe dieses Lemmas kann nun das in den Beweisen der neuen Inferenzregeln verwendete Lemma gezeigt werden.

Lemma 3.5 *Die nicht-applikative Inferenzregel*

$$\frac{\langle \Gamma \ ; \ w \rangle}{\langle \Gamma_1 \ ; \ w_1 \rangle \ldots \langle \Gamma_n \ ; \ w_n \rangle} \qquad \textit{falls} \text{ Anwendbarkeitsbedingung}$$

ist korrekt, falls es für jede Instanz der Inferenzregel eine Folge von Instanzen von nicht-induktiven und korrekten Inferenzregeln gibt, deren aufeinander folgende Anwendung auf $\langle \Gamma ; \ w \rangle$ *oder bereits abgeleitete Teilziele unter Verwendung ausschließlich induktiv gültiger Lemmata das Ziel* $\langle \Gamma \ ; \ w \rangle$ *in die Teilziele* $\langle \Gamma_1 \ ; \ w_1 \rangle, \ldots, \langle \Gamma_n \ ; \ w_n \rangle$ *ableitet.*

Beweis: Da die Inferenzregel nicht-applikativ ist, kann sie nur aufgrund der ersten Bedingung von Definition 1.7 korrekt sein. Sei also

$$\frac{\langle \Gamma \ ; \ w \rangle}{\langle \Gamma_1 \ ; \ w_1 \rangle \ldots \langle \Gamma_n \ ; \ w_n \rangle}$$

eine Instanz der Inferenzregel und $\mathcal{A} \in \mathit{DMod}(\mathit{spec})$ sowie $(\langle \Gamma \ ; \ w \rangle, \sigma, \varphi)$ ein $\mathcal{A}$-Gegenbeispiel. Weiterhin gebe es eine Folge von Instanzen von nicht-induktiven und korrekten Inferenzregeln, deren aufeinander folgende Anwendung auf $\langle \Gamma \ ; \ w \rangle$ oder bereits abgeleitete Teilziele unter Verwendung ausschließlich induktiv gültiger Lemmata das Ziel $\langle \Gamma ; \ w \rangle$ in die Teilziele $\langle \Gamma_1 ; \ w_1 \rangle, \ldots, \langle \Gamma_n ; \ w_n \rangle$ ableitet. Nach Lemma 3.4 gibt es dann ein $i \in \{1, \ldots, n\}$, so dass ein $\mathcal{A}$-Gegenbeispiel der Form $(\langle \Gamma_i ; \ w_i \rangle, \tau, \psi)$ existiert, für das $(\langle \Gamma_i \ ; \ w_i \rangle, \tau, \psi) \precsim_{\mathcal{A}} (\langle \Gamma \ ; \ w \rangle, \sigma, \varphi)$ gilt. Somit ist die Inferenzregel korrekt. □

Um Lemma 3.3 und Lemma 3.5 anwenden zu können, werden die in [Küh00] eingeführten und als korrekt und sicher bewiesenen Inferenzregeln von QuodLibet genutzt (die deutschen Bezeichnungen der Inferenzregeln sind aus [Kai02] übernommen). Die Axiome, die für die Anwendung der applikativen Inferenzregeln zur Verfügung stehen, sind in Abschnitt 2.2 aufgeführt. Die Liste der induktiv gültigen Lemmata aus Abbildung 2.1 wird zusätzlich um die Liste der Lemmata in Abbildung 3.1 ergänzt, deren induktive Gültigkeit ebenfalls in Anhang A gezeigt wird.

Für die Anwendung dieser Inferenzregeln wird jedoch häufig die Definiertheit betroffener Terme gefordert. Diese kann entweder bereits zuvor gezeigt worden sein, was durch ein negiertes Definiertheitsliteral angezeigt wird, oder muss noch gezeigt werden. Bei vielen der hier neu eingeführten Inferenzregeln wird daher gefordert, dass die Definiertheit der Summanden der Polynome bereits gezeigt wurde. Dies geschieht in der Regel bereits im Rahmen der Umwandlung eines Terms in ein Polynom in PNF, die vor Anwendung der anderen Inferenzregeln stattfinden muss. Es

$$0 \neq \mathsf{s}(x) + y \tag{L27}$$
$$\leq(\mathsf{s}(x) + y, 0) = \mathsf{false} \tag{L28}$$
$$\mathrm{def}\,(x \dot{-} y) \tag{L29}$$
$$\leq(x, y) = \mathsf{true} \ \vee \ \leq(u, v) = \mathsf{true}$$
$$\leftarrow \ \leq((z * x) + (w * u), (z * y) + ((w * v) + ((z + w) \dot{-} 1))) = \mathsf{true}, \tag{L30}$$
$$z \neq 0, w \neq 0$$
$$\leq(x, y) = \mathsf{true} \ \vee \ x = y + 1 \ \leftarrow \ \leq(x, y + 1) = \mathsf{true} \tag{L31}$$
$$(x + y) \dot{-} y = x \tag{L32}$$
$$\leq(x + y, x + z) = \leq(y, z) \tag{L33}$$
$$\leq(x \dot{-} v, y + w) = \mathsf{true} \ \leftarrow \ \leq(x, y) = \mathsf{true} \tag{L34}$$
$$\leq(y, z) = \mathsf{true} \ \leftarrow \ x + y = z \tag{L35}$$
$$(x + z = y \ \leftrightarrow x = y \dot{-} z) \ \leftarrow \ \leq(z, y) = \mathsf{true} \tag{L36}$$
$$(x \dot{-} y) * z = (x * z) \dot{-} (y * z) \tag{L37}$$
$$(x \dot{-} y) + y = x \ \leftarrow \ \leq(y, x) = \mathsf{true} \tag{L38}$$
$$x \neq y \ \leftarrow \ z * x \neq z * y \tag{L39}$$
$$x + u = y + v \ \leftarrow \ x = y, u = v \tag{L40}$$
$$u + x \neq v + y \ \leftarrow \ u \neq v, x = y \tag{L41}$$
$$\leq(u + x, v + y) = \mathsf{true} \ \leftarrow \ \leq(u, v) = \mathsf{true}, x = y \tag{L42}$$
$$u = v \ \leftarrow \ u + x = v + y, x = y \tag{L43}$$
$$u \neq v \ \leftarrow \ u + x \neq v + y, x = y \tag{L44}$$
$$\leq(u, v) = \mathsf{true} \ \leftarrow \ \leq(u + x, v + y) = \mathsf{true}, x = y \tag{L45}$$
$$\leq(x + z, y + w) = \mathsf{true} \ \leftarrow \ \leq(x + u, y + v) = \mathsf{true}, \leq(z + v, w + u) = \mathsf{true} \tag{L46}$$

Abbildung 3.1.: Weitere in jeder Spezifikation mit linearer Arithmetik induktiv gültige Lemmata.

müssen daher – außer bei der Normalisierung selbst – keine Definiertheitsziele für die Summanden erzeugt werden, sondern deren Existenz kann vorausgesetzt werden. Sollte eines der negierten Defininiertheitsliterale jedoch nicht im Ziel enthalten sein, beispielsweise, weil der Benutzer es mittels der Inferenzregel Applikative Literal-Beseitigung wieder entfernt hat, so kann stets vor Anwendung der Inferenzregeln mittels der Inferenzregel Literal Hinzufügen ein entsprechendes Literal hinzugefügt werden.

Bei einigen Inferenzregeln müssen jedoch nicht nur einzelne Summanden definiert sein, sondern auch ganze Polynome. Deren Definiertheit lässt sich jedoch einfach zeigen, wenn die Definiertheit der Summanden vorausgesetzt wird. Dies gelingt durch Anwendung von Literal Hinzufügen mit einem negierten Definiertheitsliteral für das Polynom. Das entstehende Ziel mit dem nicht negiertem Definiertheitsliteral $\mathrm{def}\,(t)$ kann durch Anwendung von Nicht-induktive Subsumption mit *L1* gezeigt werden. Anschließend kann die gewünschte Inferenz auf das Ziel mit dem negierten Definiertheitsliteral $\neg\mathrm{def}\,(t)$ angewendet werden. Das negierte Definiertheitsliteral kann

danach wieder entfernt werden, indem die Inferenzregel Applikative Literal-Beseitigung mit *L1* darauf angewendet wird. Dieses Verfahren lässt sich problemlos in eine Kette von Inferenzregel-Anwendungen wie sie für Lemma 3.5 und Lemma 3.3 benötigt wird, eingliedern, da nur korrekte und sichere Inferenzregeln mit induktiv gültigen Lemmata angewendet werden.

Um die Definiertheitsbedingungen für die Inferenzregeln formulieren zu können, wird die Menge *PolyDefCond* wie folgt definiert:

Definition 3.1 *Die Menge der negierten Definiertheitsliterale für alle Summanden eines Polynoms* $t = \sum_{i=1}^{n} a_i t_i + a$ *wird mit PolyDefCond*(t) *bezeichnet und ist wie folgt definiert:*

$$PolyDefCond(t) := \{\neg def(t_i) \mid 1 \leq i \leq n, t_i \notin \mathcal{T}(sig^C, V^C)\}$$

Diese Definition wird für ein Literal $l \ \ op \ \ r$*, bei dem* l *und* r *Polynome sind, wie folgt fortgesetzt:*

$$PolyDefCond(l \ \ op \ \ r) = PolyDefCond(l) \cup PolyDefCond(r)$$

Für alle im Folgenden beschriebenen Inferenzregeln (außer der Normalisierung selbst) ist Bedingung, dass die Literale, auf die eine dieser Inferenzregel angewendet wird, zuvor in PNF gebracht wurden. Nach der Anwendung einer Inferenzregel, die ein neues Literal zu einem Ziel hinzufügt oder ein vorhandenes Literal ändert, ist jedoch in der Regel ein Literal vorhanden, das nicht in PNF ist. Damit nicht ständig erneut eine Normalisierung durchgeführt werden muss, wird bei diesen Inferenzregeln stets automatisch jedes Literal mindestens in PNF1 gebracht. Bei einigen Inferenzregeln wird auch dadurch erst der gewünschte Effekt erzielt, wie beispielsweise bei der Inferenzregel $\leq$-Variablen-Eliminierung, bei der die Variable erst nach der Normalisierung des neuen Literals aus diesem eliminiert ist. Dennoch werden die Inferenzregeln hier ohne die Normalisierung beschrieben, da nur dann die entstehenden Teilziele klar und verständlich definiert werden können. Bei allen Inferenzregeln, bei denen eine solche automatische Normalisierung durchgeführt wird, ist dies im Beschreibungstext erwähnt. Die Beweise der Korrektheit und Sicherheit der Inferenzregeln müssen hierfür nicht angepasst werden, da zunächst die Normalisierung als korrekt und sicher nachgewiesen wird und somit nach Lemma 3.5 und Lemma 3.3 auch die aufeinander folgende Ausführung einer korrekten und sicheren Inferenzregel und der Normalisierung wieder korrekt und sicher ist.

3.2.2. Normalisierung

Zur Normalisierung stehen zwei Inferenzregeln zur Verfügung. Die Inferenzregel LA-Literal-Normalisierung überführt ein gesamtes Literal in die Polynom-Normalform während die Inferenzregel LA-Term-Normalisierung nur einen Teilterm eines Literals in die Polynom-Normalform überführt.

LA-Literal-Normalisierung:

`la-norm` $m\ n$

$$\frac{\langle \Gamma, \lambda, \Delta\ ;\ w\rangle}{\begin{array}{c}\langle \Lambda_1, \Gamma, \lambda, \Delta\ ;\ w\rangle \ldots \langle \Lambda_k, \Gamma, \lambda, \Delta\ ;\ w\rangle \\ \langle \Lambda, \lambda_{1,1}, \ldots, \lambda_{1,k_1-1}, \Gamma, \lambda_{1,k_1}, \Delta\ ;\ w\rangle \ldots \langle \Lambda, \lambda_{j,1}, \ldots, \lambda_{j,k_j-1}, \Gamma, \lambda_{j,k_j}, \Delta\ ;\ w\rangle\end{array}}$$

falls eine Klausel Θ existiert, so dass

- $n \in \{1, 2, 3\}$
- $(\Gamma, \lambda, \Delta)[m] = \lambda$
- $\lambda = l\ \mathit{op}\ r$, wobei l und r Terme bzw. einstellige Gewichte der Sorte Nat sind
- $\{\lambda_{1,1} \ldots \lambda_{1,k_1}, \ldots, \lambda_{j,1} \ldots \lambda_{j,k_j}\}$ ist die aus λ in *spec* mit *PNFDefCond*(λ) herleitbare Klauselmenge in $\mathsf{PNF}n$ entsprechend Satz 2.2, 2.3 bzw. 2.4
- Γ, Δ, Θ enthält *PNFDefCond*(λ)
- $\Lambda_1, \ldots, \Lambda_k, \Lambda$ ist die aus Θ resultierende Fallunterscheidung

Mit dieser Inferenzregel wird das an Position m stehende Literal in die in Abschnitt 2.4 beschriebene Polynom-Normalform überführt. Welche Stufe der PNF erzeugt werden soll, lässt sich über den Parameter n festlegen. Dieser kann daher den Wert 1, 2 oder 3 haben.

Bei der Überführung in PNF werden auch die notwendigen Definiertheitsziele erzeugt, sofern die Definiertheit der entsprechenden Terme nicht bereits gezeigt worden ist.

Beispiel: Das Ziel

$$\langle (x * 2) + (7 + (4 * x)) \leq ((2 * y) \mathbin{\dot{-}} (3 * \mathsf{f}(x)) + ((x * 2) + 3))\ ;\ w\rangle$$

kann durch Anwendung von

`la-norm` 1 2

in die Ziele

$$\begin{array}{c}\langle \mathrm{def}\,(\mathsf{f}(x))\ \vee\ (x * 2) + (7 + (4 * x)) \leq ((2 * y) \mathbin{\dot{-}} (3 * \mathsf{f}(x)) + ((x * 2) + 3))\ ;\ w\rangle \\ \langle 2y + 1 \leq 3\mathsf{f}(x)\ \vee\ \neg\mathrm{def}\,(\mathsf{f}(x))\ \vee\ 4x + 3\mathsf{f}(x) + 4 \leq 2y\ ;\ w\rangle \\ \langle 3\mathsf{f}(x) \leq 2y\ \vee\ \neg\mathrm{def}\,(\mathsf{f}(x))\ \vee\ x + 1 \leq 0\ ;\ w\rangle\end{array}$$

überführt werden. Das erste Ziel dient dazu, die Definiertheit von $\mathsf{f}(x)$ nachzuweisen. In den beiden folgenden Zielen, die aus der Fallunterscheidung für das $\dot{-}$-Zeichen entstehen, wird diese Definiertheit dann vorausgesetzt.

Lemma 3.6 *Die Inferenzregel LA-Literal-Normalisierung ist korrekt und sicher.*

Beweis: Nach Satz 2.2, 2.3 bzw. 2.4 ist für jede Instanz der Inferenzregel LA-Literal-Normalisierung das Literal λ mit *PNFDefCond*(λ) zur Klauselmenge

$$\{\lambda_{1,1} \dots \lambda_{1,k_1}, \dots, \lambda_{j,1} \dots \lambda_{j,k_j}\}$$

nicht-induktiv ableitbar. Das heißt, es gibt eine Folge von Instanzen nicht-induktiver, korrekter und sicherer Inferenzregeln, deren aufeinander folgende Anwendung das Ziel $\langle \lambda\,;\, w\rangle$ in die Ziele

$$\langle \Lambda_1, \lambda\,;\, w\rangle, \dots, \langle \Lambda_n, \lambda\,;\, w\rangle, \langle \Lambda, \lambda_{1,1}, \dots, \lambda_{1,k_1}\,;\, w\rangle, \dots, \langle \Lambda, \lambda_{j,1}, \dots, \lambda_{j,k_j}\,;\, w\rangle$$

ableitet, wobei $\Lambda_1, \dots, \Lambda_n, \Lambda$ die aus *PNFDefCond*(λ) resultierende Fallunterscheidung ist. Da zusätzliche Literale in einem Ziel die Anwendbarkeit von Inferenzregeln nicht verhindern können, gilt dies auch, wenn in jedem Ziel zusätzlich die Klauseln Γ und Δ enthalten sind. Sofern das i-te Literal aus *PNFDefCond*(λ) bereits in Γ oder Δ enthalten ist, kann anschließend dieses anschließend aus allen Zielen, die es enthalten mittels der Inferenzregel Mehrfache Literale wieder entfernt werden. Jedes Ziel, das das Literal negiert enthält, kann mittels der Inferenzregel Komplementäre Literale zur leeren Menge abgeleitet werden. Dadurch entstehen genau die Ziele, die aus einer Fallunterscheidung nach Θ statt nach *PNFDefCond*(λ) resultieren. Daher ist nach Lemma 3.5 und Lemma 3.3 die Inferenzregel LA-Literal-Normalisierung korrekt und sicher. □

LA-Term-Normalisierung:

`la-term-norm` $m\ p$

$$\frac{\langle \Gamma, \lambda, \Delta\,;\, w\rangle}{\langle \Lambda_1, \Gamma, \lambda, \Delta\,;\, w\rangle \dots \langle \Lambda_k, \Gamma, \lambda, \Delta\,;\, w\rangle\ \langle \Lambda, \Gamma, \lambda[t]_p, \Delta\,;\, w\rangle}$$

falls eine Klausel Θ existiert, so dass gilt:

- $(\Gamma, \lambda, \Delta)[m] = \lambda$
- $\lambda/p \in \mathcal{T}(F, V)_{\mathsf{Nat}}$
- t ist die Termrepräsentation eines in *spec* mit *PNFDefCond*(t) aus λ/p nicht-induktiv ableitbaren Polynoms in PNF entsprechend Satz 2.1
- Γ, Δ, Θ enthält *PNFDefCond*(t)
- $\Lambda_1, \dots, \Lambda_k, \Lambda$ ist die aus Θ resultierende Fallunterscheidung

Mit dieser Inferenzregel wird der Teilterm an Position p des Literals m in die in Abschnitt 2.3 beschriebene Polynom-Normalform überführt.

Bei der Überführung in PNF werden auch die notwendigen Definiertheitsziele erzeugt, sofern die Definiertheit der entsprechenden Terme nicht bereits gezeigt worden ist.

Beispiel: Das Ziel

$$\langle((x * 2) + 7) + ((\mathsf{f}(x) * 2) + (3 * x)) \leq (2 * y) + ((x * 2) + 3)) \; ; \; w\rangle$$

kann durch Anwendung von

`la-term-norm` 1 [1]

in die Ziele

$$\langle \mathrm{def}\,(\mathsf{f}(x)) \; \vee \; ((x * 2) + 7) + ((\mathsf{f}(x) * 2) + (3 * x)) \leq (2 * y) + ((x * 2) + 3)) \; ; \; w\rangle$$
$$\langle \neg \mathrm{def}\,(\mathsf{f}(x)) \; \vee \; 5x + 2\mathsf{f}(x) + 7 \leq (2 * y) + ((x * 2) + 3)) \; ; \; w\rangle$$

überführt werden. Das erste Ziel dient dazu, die Definiertheit von $\mathsf{f}(x)$ nachzuweisen. Im zweiten Ziel wird diese Definiertheit dann vorausgesetzt und, da die Position [1] angegeben wurde, die linke Seite des Literals in PNF umgewandelt.

Lemma 3.7 *Die Inferenzregel LA-Term-Normalisierung ist korrekt und sicher.*

Beweis: Nach Satz 2.1 ist für jede Instanz der Inferenzregel LA-Term-Normalisierung der Term λ/p mit *PNFDefCond*(λ/p) zu einem Term t in PNF nicht-induktiv ableitbar. Das heißt, es gibt eine Folge von Instanzen nicht-induktiver, korrekter und sicherer Inferenzregeln, deren aufeinander folgende Anwendung das Ziel $\langle \lambda \; ; \; w\rangle$ in die Ziele

$$\langle \Lambda_1, \lambda \; ; \; w\rangle, \ldots, \langle \Lambda_n, \lambda \; ; \; w\rangle, \langle \Lambda, \lambda[t]_p \; ; \; w\rangle$$

ableitet, wobei $\Lambda_1, \ldots, \Lambda_n, \Lambda$ die aus *PNFDefCond*(λ) resultierende Fallunterscheidung ist. Da zusätzliche Literale in einem Ziel die Anwendbarkeit von Inferenzregeln nicht verhindern können, gilt dies auch, wenn in jedem Ziel zusätzlich die Klauseln Γ und Δ enthalten sind. Sofern das i-te Literal aus *PNFDefCond*(λ) bereits in Γ oder Δ enthalten ist, kann anschließend dieses anschließend aus allen Zielen, die es enthalten mittels der Inferenzregel Mehrfache Literale wieder entfernt werden. Jedes Ziel, das das Literal negiert enthält, kann mittels der Inferenzregel Komplementäre Literale zur leeren Menge abgeleitet werden. Dadurch entstehen genau die Ziele, die aus einer Fallunterscheidung nach Θ statt nach *PNFDefCond*(λ) resultieren. Daher ist nach Lemma 3.5 und Lemma 3.3 die Inferenzregel LA-Term-Normalisierung korrekt und sicher. □

3.2.3. Tautologien

Die Inferenzregeln für Tautologien ermöglichen es, bestimmte induktiv gültige Literale zu nutzen, um ein Ziel zu beweisen. Es stehen zwei Inferenzregeln zur Verfügung, eine für $\leq$-Literale und eine für Ungleichungen. Diese stellen in der Regel den Endpunkt des Entscheidungsverfahrens für ein Ziel dar.

$\leq$-Tautologie:

`<=-taut` m

$$\underline{\langle \Gamma, 0 \leq t, \Delta \,;\, w \rangle}$$

falls

- $(\Gamma, 0 \leq t, \Delta)[m] = 0 \leq t$
- $0 \leq t$ in PNF
- Γ, Δ enthält *PolyDefCond*(t)

Diese Inferenzregel resultiert unmittelbar aus dem Axiom *A8* und dient dazu, am Ende des Entscheidungsverfahrens eine Tautologie zu erkennen.

Beispiel: Das Ziel

$$\langle 0 \leq 4x + 3 \,;\, w \rangle$$

kann durch Anwendung der Inferenzregel $\leq$-Tautologie bewiesen werden.

Lemma 3.8 *Die Inferenzregel $\leq$-Tautologie ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Das Ziel $\langle \Gamma, 0 \leq t, \Delta \,;\, w \rangle$ kann durch Anwendung der korrekten Inferenzregel Nicht-induktive Subsumption mit dem Axiom *A8* und der Substitution $\{y \leftarrow t\}$ zur leeren Menge abgeleitet werden (die dafür benötigte Definiertheit von t wird wie im Abschnitt 3.2.1 beschrieben sichergestellt). Somit ist die Inferenzregel $\leq$-Tautologie nach Lemma 3.5 korrekt.
- **Sicherheit:** Da die Inferenzregel $\leq$-Tautologie keine Teilziele hat, ist sie offensichtlich sicher. □

LA-$\neq$-Tautologie:

`la-=/=-taut` m

$$\underline{\langle \Gamma, 0 \neq t, \Delta \,;\, \rangle}$$

falls

- $(\Gamma, \lambda, \Delta)[m] = 0 \neq t$
- λ ist in PNF und $t = \sum_{i=1}^{n} a_i t_i + a$
- $a \neq 0$
- Γ, Δ enthält *PolyDefCond*(t)

Mit dieser Inferenzregel können auch offensichtlich induktiv gültige Ungleichungen ausgenutzt werden.

Beispiel: Das Ziel

$$\langle 0 \neq x + 1 \,;\, w \rangle$$

kann durch Anwendung der Inferenzregel LA-$\neq$-Tautologie bewiesen werden.

Lemma 3.9 *Die Inferenzregel LA-≠-Tautologie ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Die Termrepräsentation des Polynoms $\sum_{i=1}^{n} a_i t_i + a$ hat, da $a \neq 0$ ist, die Form $\mathsf{s}(x) + y$. Das Ziel $\langle \Gamma, 0 \neq \mathsf{s}(x) + y, \Delta \ ; \ w \rangle$ kann durch Anwendung der korrekten Inferenzregel Nicht-induktive Subsumption mit *L27* und der Substitution $\{x \leftarrow t\}$ zur leeren Menge abgeleitet werden (die dafür benötigte Definiertheit von t wird wie im Abschnitt 3.2.1 beschrieben sichergestellt). Somit ist die Inferenzregel LA-≠-Tautologie nach Lemma 3.5 korrekt.
- **Sicherheit:** Da die Inferenzregel LA-≠-Tautologie keine Teilziele hat, ist sie offensichtlich sicher. □

3.2.4. Redundanz-Elimination

Redundanz-Elimination ist zur Beweisfindung nicht notwendig, trägt jedoch bei manueller Beweissuche zur Übersichtlichkeit und bei automatischer Beweissuche zur Einschränkung des Suchraums bei.

Lemma 3.10 *(nach [Küh00, S. 182])*
Jede ein Literal entfernende Inferenzregel der Form

$$\frac{\langle \Gamma, \lambda, \Delta \ ; \ w \rangle}{\langle \Gamma, \Delta \ ; \ w \rangle}$$

ist korrekt.

Beweis: Angenommen $(\langle \Gamma, \lambda, \Delta \ ; \ w \rangle, \sigma, \varphi)$ ist ein $\mathcal{A}$-Gegenbeispiel für das Ziel der Regel. Dann erfüllt $\mathcal{A}$ keines der Literale $\Gamma\sigma$ und $\Delta\sigma$ mit φ. Daher ist $(\langle \Gamma, \Delta; \ w \rangle, \sigma, \varphi)$ ein $\mathcal{A}$-Gegenbeispiel für das Teilziel der Regel. Außerdem gilt für dieses offensichtlich $(\langle \Gamma, \Delta \ ; \ w \rangle, \sigma, \varphi) \precsim_{\mathcal{A}} (\langle \Gamma, \lambda, \Delta \ ; \ w \rangle, \sigma, \varphi)$. □

≤-Beseitigung:

`<=-removal` m

$$\frac{\langle \Gamma, t \leq 0, \Delta \ ; \ w \rangle}{\langle \Gamma, \Delta \ ; \ w \rangle}$$

falls

- $(\Gamma, \lambda, \Delta)[m] = t \leq 0$
- $t \leq 0 = \sum_{i=1}^{n} a_i t_i + a \leq 0$ in PNF
- $a \neq 0$
- Γ, Δ enthält $\mathit{PolyDefCond}(t)$

Ein Literal, das diese Bedingungen erfüllt, ist offensichtlich von keinem Datenmodell erfüllbar und daher redundant. Es kann somit entfernt werden.

Beispiel: Im Ziel

$$\langle 3x + 4 \leq 0 \ \vee \ x + y = y + x \,;\ w \rangle$$

kann das erste Literal durch Anwendung der Inferenzregel $\leq$-Beseitigung entfernt werden. Das Resultat ist:

$$\langle x + y = y + x \,;\ w \rangle$$

Lemma 3.11 *Die Inferenzregel $\leq$-Beseitigung ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Die Korrektheit folgt unmittelbar aus Lemma 3.10.
- **Sicherheit:** Die Termrepräsentation t des Polynoms $\sum_{i=1}^{n} a_i t_i + a$ hat, da $a \neq 0$ ist, die Form $\mathsf{s}(x) + y$. Das Ziel $\langle \Gamma, \leq(\mathsf{s}(x) + y, 0) = \mathsf{true}, \Delta \,;\ w \rangle$ kann durch Anwendung der sicheren Inferenzregel Nicht-induktive Termersetzung mit *L28* und der Substitution $\{x \leftarrow t\}$ auf die linke Seite des Literals m zu $\langle \Gamma, \mathsf{false} = \mathsf{true}, \Delta \,;\ w \rangle$ abgeleitet werden (die dafür benötigte Definiertheit von t wird wie im Abschnitt 3.2.1 beschrieben sichergestellt). Dieses Ziel kann durch Anwendung der sicheren Inferenzregel $=$-Beseitigung auf das Literal $\mathsf{false} = \mathsf{true}$ zu $\langle \Gamma, \Delta \,;\ w \rangle$ abgeleitet werden. Somit ist die Inferenzregel $\leq$-Beseitigung nach Lemma 3.3 sicher. □

$\leq$-Subsumptions-Beseitigung:

`<=-subs-removal` $m\ n$

$$\frac{\langle \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle}{\langle \Gamma, t_1 \leq t_2, \Delta, \Pi \ \,;\ w \rangle}$$

falls

- $(\Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi)[m] = t_1 \leq t_2$
- $(\Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi)[n] = s_1 \leq s_2$
- $t_1 \leq t_2 = \sum_{i=1}^{m_1} a_i t'_i + a \leq \sum_{i=1}^{m_2} b_i t''_i + b$ in PNF
- $s_1 \leq s_2 = \sum_{i=1}^{m_1} a_i t'_i + c \leq \sum_{i=1}^{m_2} b_i t''_i + d$ in PNF
- $a + d \leq b + c$ ist induktiv gültig

Dadurch, dass die Differenz zwischen a und b größer sein muss, als die Differenz zwischen d und c, ist immer, wenn $s_1 \leq s_2$ erfüllt ist, auch $t_1 \leq t_2$ erfüllt. Daher kann $s_1 \leq s_2$ entfernt werden, weil dadurch keine Lösungen verloren gehen.

Beispiel: Das Ziel

$$\langle x \leq 7 \ \vee \ x \leq 5 \,;\ w \rangle$$

kann mittels der Inferenzregel $\leq$-Subsumptions-Beseitigung durch Anwendung von

```
<=-subs-removal 1 2
```

zum Ziel

$$\langle x \leq 7 \,;\ w \rangle$$

abgeleitet werden.

Lemma 3.12 *Die Inferenzregel $\leq$-Subsumptions-Beseitigung ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Die Korrektheit folgt unmittelbar aus Lemma 3.10.
- **Sicherheit:** Das Ziel

  $$\langle \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle$$

  kann durch Anwendung der sicheren Inferenzregel Literal Hinzufügen mit dem Literal $a + d \leq b + c$ zu den beiden Zielen

  $$\langle a + d \leq b + c, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle,$$

  $$\langle a + d \not\leq b + c, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle$$

  abgeleitet werden. Das erste dieser beiden Ziele kann durch Anwendung der sicheren Inferenzregel Nicht-induktive Subsumption mit dem induktiv gültigen Literal $a + d \leq b + c$ zur leeren Menge abgeleitet werden. Das zweite Ziel kann durch Anwendung der Inferenzregel Applikative Literal-Beseitigung mit $L46$ und der Substitution

  $$\{x \leftarrow \sum_{i=1}^{m_1} a_i t_i', y \leftarrow \sum_{i=1}^{m_2} b_i t_i'', z \leftarrow a, u \leftarrow c, v \leftarrow d, w \leftarrow b\}$$

  zu

  $$\langle a + d \not\leq b + c, \Gamma, t_1 \leq t_2, \Delta, \Pi \,;\ w \rangle$$

  abgeleitet werden (die dafür benötigte Definiertheit von $\sum_{i=1}^{m_1} a_i t_i'$ und $\sum_{i=1}^{m_2} b_i t_i''$ wird wie im Abschnitt 3.2.1 beschrieben sichergestellt). Anschließend kann das Literal $a + d \not\leq b + c$ durch Anwendung der sicheren Inferenzregel Applikative Literal-Beseitigung mit dem induktiv gültigen Literal $a + d \leq b + c$ wieder entfernt werden. Das daraus resultierende Ziel

  $$\langle \Gamma, t_1 \leq t_2, \Delta, \Pi \,;\ w \rangle$$

  ist das Teilziel der Inferenzregel $\leq$-Subsumptions-Beseitigung. Somit ist die Inferenzregel $\leq$-Subsumptions-Beseitigung nach Lemma 3.3 sicher. □

3.2.5. Variablen-Elimination

Der Kern des beschriebenen Entscheidungsverfahrens ist die Elimination von Variablen. Die folgenden Inferenzregeln realisieren jeweils die Elimination einer Variablen aus zwei $\leq$-Literalen bzw. aus einem $\neq$-Literal. In beiden Fällen werden jedoch aus den in Abschnitt 3.1 beschriebenen Gründen die ursprünglichen Literale im Ziel beibehalten. Zudem wird die Inferenzregel $\leq$-Fallunterscheidung beschrieben, die zwar keine direkte Variablenelimination beinhaltet, aber stets angewendet werden kann, wenn die $\leq$-Variablenelimination zu einem redundanten Literal der Form $k \leq 0$ mit $k \neq 0$ führt.

$\leq$-Variablen-Elimination:

`<=-var-elim` $m\ n\ p$

$$\frac{\langle \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle}{\langle u_1 \leq u_2, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle}$$

falls

- $(\Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi)[m] = t_1 \leq t_2$
- $(\Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi)[n] = s_1 \leq s_2$
- $t_1 \leq t_2 = \sum_{i=1}^{m_1} a_i t'_i + a \leq \sum_{i=1}^{m_2} b_i t''_i + b$ in PNF
- $s_1 \leq s_2 = \sum_{i=1}^{n_1} c_i s'_i + c \leq \sum_{i=1}^{n_2} d_i s''_i + d$ in PNF
- $(t_1 \leq t_2)/p = t''_j$ für ein $j \in \{1, \ldots, m_2\}$
- $t''_j = s'_k$ für ein $k \in \{1, \ldots, n_1\}$
- g ist der größte gemeinsame Teiler von b_j und c_k, das heißt es existieren b'_j und c'_k in $\mathcal{N}$, so dass $b_j = (g * b'_j)\!\downarrow$ und $c_k = (g * c'_k)\!\downarrow$
- $u_1 \leq u_2 = (c'_k * t_1) + (b'_j * s_1) \leq (c'_k * t_2) + ((b'_j * s_2) + ((b'_j + c'_k) \dot{-} 1))$
- Γ, Δ, Π enthält $\mathit{PolyDefCond}(t_i)$ und $\mathit{PolyDefCond}(s_i)$ für $i \in \{1, 2\}$.

Diese Inferenzregel realisiert die Variablen-Elimination für $\leq$-Literale des Entscheidungsverfahrens. Nach der Anwendung der Inferenzregel $\leq$-Variablen-Elimination kommt in u_1 der Term $c'_k * g * b'_j * t''_j$ und in u_2 der Term $b'_j * g * c'_k * s'_k$ vor. Da $t''_j = s'_k$ gilt, wird dieser Summand somit bei der anschließenden Normalisierung aus dem Literal eliminiert. Damit ist die Variablen-Elimination für $\leq$-Literale des Entscheidungsverfahrens realisiert und es steht ein Literal zur Verfügung, aus dem die Variable t''_j bzw. s'_k eliminiert wurde.

Beispiel: Das Ziel

$$\langle 2x \leq y \ \vee \ 2y \leq z \,;\ w \rangle$$

kann mittels der Inferenzregel $\leq$-Variablen-Elimination durch Anwendung von

`<=-var-elim` 1 2 [2]

zu dem Ziel

$$\langle (2 * (2 * x)) + (2 * y) \leq (2 * y) + ((1 * z) + ((2 + 1) \dot{-} 1)) \ \vee \ 2x \leq y \ \vee \ 2y \leq z \,;\ w \rangle$$

abgeleitet werden. Durch die anschließende Normalisierung wird dann das Ziel

$$\langle 4x \leq z + 2 \ \vee \ 2x \leq y \ \vee \ 2y \leq z \,;\ w \rangle$$

erzeugt, bei dem die Variable y aus dem ersten Literal eliminiert ist.

Lemma 3.13 *Die Inferenzregel $\leq$-Variablen-Elimination ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Für jede Instanz der Inferenzregel kann durch Anwendung der korrekten Inferenzregel Nicht-induktive Subsumption mit dem induktiv gültigen Lemma *L30* und der Substitution

  $$\{x \leftarrow t_1, y \leftarrow t_2, u \leftarrow s_1, v \leftarrow s_2, z \leftarrow c'_k, w \leftarrow b'_j\}$$

  auf das Ziel

  $$\langle \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle$$

  dieses zu

  $$\langle u_1 \leq u_2, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle$$

  abgeleitet werden (die dafür benötigte Definiertheit von t_1, t_2, s_1 und s_2 wird wie im Abschnitt 3.2.1 beschrieben sichergestellt). Somit ist nach Lemma 3.5 auch die Inferenzregel $\leq$-Variablen-Elimination korrekt.

- **Sicherheit:** Die Sicherheit folgt unmittelbar aus Lemma 3.1. □

$\leq$-Fallunterscheidung:

`<=-case-split` $m\ n\ p$

$$\frac{\langle \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle}{\langle t_1 \neq t_2 + 1, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle \ldots \langle t_1 \neq t_2 + k, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi \,;\ w \rangle}$$

falls

1. $(\Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi)[m] = t_1 \leq t_2$
2. $(\Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi)[n] = s_1 \leq s_2$

3. $t_1 \leq t_2 = \sum_{i=1}^{m_1} a_i t'_i + a \leq \sum_{i=1}^{m_2} b_i t''_i + b$ in PNF
4. $s_1 \leq s_2 = \sum_{i=1}^{n_1} c_i s'_i + c \leq \sum_{i=1}^{n_2} d_i s''_i + d$ in PNF
5. $(t_1 \leq t_2)/p = t''_j$ für ein $j \in \{1, \ldots, m_2\}$
6. $t''_j = s'_k$ für ein $k \in \{1, \ldots, n_1\}$
7. g ist der größte gemeinsame Teiler von b_j und c_k, das heißt es existieren b'_j und c'_k in $\mathcal{N}$, so dass $b_j = \left(g * b'_j\right)\downarrow$ und $c_k = \left(g * c'_k\right)\downarrow$
8. $c'_k \sum_{i=1}^{m_1} a_i t'_i + b'_j \sum_{i=1}^{n_1} c_i s'_i = c'_k \sum_{i=1}^{m_2} b_i t''_i + b'_j \sum_{i=1}^{n_2} d_i s''_i$ ist induktiv gültig
9. $\mathrm{k} = \left(\left(c'_k a + b'_j c\right) \dot{-} \left(\left(c'_k b + b'_j d\right) + \left(\left(b'_j + c'_k\right) \dot{-} 1\right)\right)\right)\downarrow$
10. $k \neq 0$
11. $\left(c'_k b + b'_j d\right) + \left(\left(b'_j + c'_k\right) \dot{-} 1\right) \leq c'_k a + b'_j c$ ist induktiv gültig
12. Γ, Δ, Π enthält $\mathit{PolyDefCond}(t_i)$ und $\mathit{PolyDefCond}(s_i)$ für $i \in \{1, 2\}$

Diese Inferenzregel dient dazu, in dem Fall, dass die Anwendung der Inferenzregel $\leq$-Variablen-Elimination zu einem Literal $k \leq 0$ mit $k \neq 0$ führen würde, eine Fallunterscheidung für alle k möglichen Werte der Variablen durchzuführen. Dadurch kann auch die im Abschnitt 3.1 beschriebene Fallunterscheidung zur Erweiterung des Entscheidungsverfahrens über rationalen Zahlen auf die natürlichen Zahlen realisiert werden.

Alle neu hinzugefügten Literale werden wie in Abschnitt 3.2.1 beschrieben automatisch normalisiert.

Beispiel: Im Fall $k = 1$ kann durch Anwendung von $\leq$-Fallunterscheidung eine *implizite Gleichung* [KN94] ausgenutzt werden. Die Fallunterscheidung besteht aus genau einem Fall, so dass zum ursprünglichen Ziel eine Ungleichung hinzukommt. Diese kann dann durch Anwendung von Konstant Umschreiben (evtl. nach vorheriger Anwendung von $\neq$-Variablenelimination) genutzt werden.

Beispielsweise kann das Ziel

$$\langle x \leq 1 \ \vee \ 3 \leq x \ \vee \ x = 2 \,;\ w \rangle$$

durch Anwendung von $\leq$-Fallunterscheidung auf die beiden ersten Literale mit für die Variable x zum Ziel

$$\langle x \neq 2 \ \vee \ x \leq 1 \ \vee \ 3 \leq x \ \vee \ x = 2 \,;\ w \rangle$$

abgeleitet werden.

Die Fallunterscheidung für das Entscheidungsverfahren muss nur durchgeführt werden, wenn mindestens zwei Variablen vorhanden sind. Die ist beispielsweise für das Ziel

$$\langle 6 \leq 2x + y \ \vee \ 2x + y \leq 3 \ \vee \ y + 3 \leq 2x \ \vee \ 2x \leq y \,;\ w \rangle$$

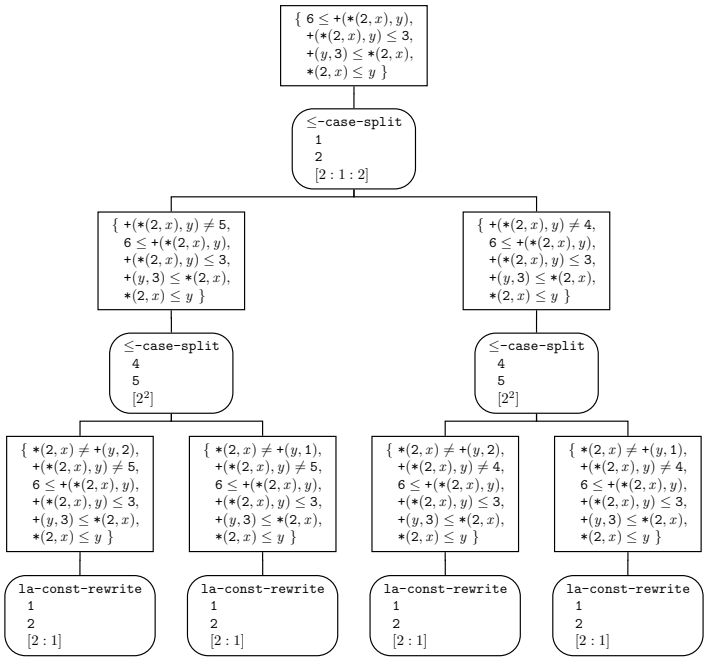

Abbildung 3.2.: Beweisbaum für das zweite Beispiel für ≤-Fallunterscheidung

notwendig. Die induktive Gültigkeit dieses Ziels kann alleine durch Anwendung von ≤-Variablenelimination und ≤-Tautologie nicht nachgewiesen werden. Die liegt daran, dass es über den rationalen Zahlen Gegenbeispiele (z. B. $x = 1{,}5$ und $y = 1{,}5$) gibt, nicht jedoch über den natürlichen Zahlen. Das Ziel kann jedoch durch Anwendung von ≤-Fallunterscheidung mit den beiden ersten Literalen und der Variable x zu den beiden Zielen

$$\langle 2x + y \neq 5 \ \vee \ 6 \leq 2x + y \ \vee \ 2x + y \leq 3 \ \vee \ y + 3 \leq 2x \ \vee \ 2x \leq y \,;\ w \rangle$$
$$\langle 2x + y \neq 4 \ \vee \ 6 \leq 2x + y \ \vee \ 2x + y \leq 3 \ \vee \ y + 3 \leq 2x \ \vee \ 2x \leq y \,;\ w \rangle$$

abgeleitet werden. Führt man anschließend auf jedem dieser Ziele eine ≤-Fallunterscheidung mit den anderen beiden ≤-Literalen und der Variable y aus, so erhält man folgende vier Ziele mit jeweils zwei Ungleichungen:

$$\langle 2x \neq y + 2 \vee 2x + y \neq 5 \vee 6 \leq 2x + y \vee 2x + y \leq 3 \vee y + 3 \leq 2x \vee 2x \leq y \,;\ w \rangle$$
$$\langle 2x \neq y + 1 \vee 2x + y \neq 5 \vee 6 \leq 2x + y \vee 2x + y \leq 3 \vee y + 3 \leq 2x \vee 2x \leq y \,;\ w \rangle$$
$$\langle 2x \neq y + 2 \vee 2x + y \neq 6 \vee 6 \leq 2x + y \vee 2x + y \leq 3 \vee y + 3 \leq 2x \vee 2x \leq y \,;\ w \rangle$$
$$\langle 2x \neq y + 1 \vee 2x + y \neq 6 \vee 6 \leq 2x + y \vee 2x + y \leq 3 \vee y + 3 \leq 2x \vee 2x \leq y \,;\ w \rangle$$

Alle diese Ziele können durch Anwendung von LA Konstant Umschreiben auf die beiden ersten Literale mit der Variable y abgeschlossen werden, so dass die induktive Gültigkeit des ursprünglichen Ziels gezeigt ist. Der Beweisbaum für diesen Beweis ist in Abbildung 3.2 dargestellt.

Lemma 3.14 *Die Inferenzregel ≤-Fallunterscheidung ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Das Ziel $\langle \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle$ kann durch Anwendung der korrekten Inferenz-Regel Literal Hinzufügen mit den Literalen $t_1 = t_2 + 1, \ldots, t_1 = t_2 + k$ zu den Teilzielen

$$\begin{aligned}
&\langle t_1 \neq t_2 + 1, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle,\\
&\langle t_1 \neq t_2 + 2, t_1 = t_1 + 1, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle,\\
&\ldots,\\
&\langle t_1 \neq t_2 + k, \ldots, t_1 = t_1 + 1, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle,\\
&\langle t_1 = t_2 + 1, \ldots, t_1 = t_2 + k, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle
\end{aligned}$$

  abgeleitet werden. Das erste Teilziel ist bereits mit dem ersten Teilziel der Inferenzregel $\leq$-Fallunterscheidung identisch. Das zweite Teilziel kann durch Anwendung der korrekten Inferenzregel Konstant Umschreiben mit dem ersten Literal auf die linke Seite des zweiten Literals zu

$$\langle t_2 + 2 = t_2 + 1, t_1 \neq t_2 + 2, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle$$

  abgeleitet werden. Durch Anwendung der bereits als korrekt nachgewiesenen Inferenzregel LA-Literal-Normalisierung und anschließende Anwendung von $=$-Beseitigung auf das erste Literal wird dieses dann zu

$$\langle t_1 \neq t_2 + 2, \Gamma, t_1 \leq t_2, \Delta, s_1 \leq s_2, \Pi\,;\ w\rangle$$

  abgeleitet. Dies entspricht dem zweiten Teilziel der Inferenzregel $\leq$-Fallunterscheidung. Analog kann man auch für das 3. bis k. Teilziel verfahren.

  Das letzte Teilziel muss eliminiert werden. Dies gelingt durch Nachweis seiner induktiven Gültigkeit. Dazu werden sukzessive durch Anwendung der Inferenzregel Nicht-induktive Subsumption mit *L31* die Literale $t_1 \leq t_2+1$ bis $t_1 \leq t_2+k$ hinzugenommen. Sei m' die Position des Literals $t_1 \leq t_2 + k$ im resultierenden Ziel. Dann kann die bereits als korrekt nachgewiesene Inferenzregel $\leq$-Variablen-Elimination auf die Literale $t_1 \leq t_2 + k$ und $s_1 \leq s_2$ an der Position p angewendet werden. Das Resultat ist ein neues Literal

$$(c'_k * t_1) + (b'_j * s_1) \leq c'_k * (t_2 + k) + (b'_j * s_2) + b'_j + c'_k \dot{-} 1$$

  im Ziel. Durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit *L6* und *L4* kann dieses zu

$$(c'_k * t_1) + (b'_j * s_1) \leq (c'_k * t_2) + (c'_k * k) + (b'_j * s_2) + b'_j + c'_k \dot{-} 1$$

  abgeleitet werden. Anschließend kann dieses durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit Bedingung 8 und *L33* zu

$$(c'_k * a) + (b'_j * c) \leq (c'_k * b) + (b'_j * d) + (c'_k * k) + b'_j + c'_k \dot{-} 1$$

  abgeleitet werden. Dieses kann durch Anwendung der ebenfalls bereits als korrekt nachgewiesenen Inferenzregel Literal-Normalisierung zu $0 \leq ((c'_k \dot{-} 1) * k)\downarrow$

abgeleitet werden. Durch Anwendung der Inferenzregel $\leq$-Tautologie auf dieses Literal kann das Ziel dann zur leeren Menge abgeleitet werden.

Somit gibt es für jede Instanz der Inferenzregel $\leq$-Fallunterscheidung eine Folge von Instanzen von nicht-induktiven und korrekten Inferenzregeln, deren aufeinander folgende Anwendung das Ziel unter Verwendung ausschließlich induktiv gültiger Lemmata in die Teilziele ableitet. Daher ist die Inferenzregel $\leq$-Fallunterscheidung nach Lemma 3.5 korrekt.

- **Sicherheit:** Die Sicherheit folgt unmittelbar aus Lemma 3.1. □

$\neq$-Variablen-Elimination:

`=/=-var-elim` $m\ p\ x$

$$\frac{\langle \Gamma, t_1 \neq t_2, \Delta\ ;\ w\rangle}{\langle u_1 + x \neq u_2, \Gamma, t_1 \neq t_2, \Delta\ ;\ w\rangle}$$

falls

- $(\Gamma, t_1 \neq t_2, \Delta)[m] = t_1 \neq t_2$
- $t_1 \neq t_2 = \sum_{i=1}^{m_1} a_i t_i' + a \neq \sum_{i=1}^{m_2} b_i t_i'' + b$ in PNF
- $x \in V^C \setminus \mathit{Var}(\Gamma, t_1, t_2, \Delta, w)$
- p bezeichnet einen Summanden in t_1, d. h. $(t_1 \neq t_2)/p = a_k t_k'$ für ein $k \in \{1, \ldots, m_1\}$
- $a_k \leq a_i$ ist induktiv gültig für $1 \leq i \leq m_1$
- $a_k \leq b_i$ ist induktiv gültig für $1 \leq i \leq m_2$
- $a_k \neq 1$
- es existieren a_i', a_i'', a', a'', so dass $a_i = (a_k * a_i' + a_i'')\downarrow$ und $a_i'' < a_k$für $1 \leq i \leq m_1$ sowie $a = (a_k * a' + a'')\downarrow$ und $a'' < a_k$
- es existieren b_i', b_i'', b', b'', so dass $b_i = (a_k * b_i' \dot{-} b_i'')\downarrow$ und $b_i'' < a_k$ für $1 \leq i \leq m_2$ sowie $b = (a_k * b' \dot{-} b'')\downarrow$ und $b'' < a_k$
- $u_1 = \sum_{i=1}^{m_1} a_i' t_i' + a'$ und $u_2 = \sum_{i=1}^{m_1} b_i' t_i'' + b'$
- Γ, Δ enthält $\mathit{PolyDefCond}(t_1)$ und $\mathit{PolyDefCond}(t_2)$

Diese Inferenzregel realisiert einen Teil der Variablenelimination für $\neq$-Literale. Mit Hilfe dieser Inferenzregel kann in einem $\neq$-Literal, dessen minimaler Faktor nicht 1 ist, der minimale Faktor reduziert werden. Dazu muss direkt im Anschluss und die Anwendung dieser Inferenzregel die Inferenzregel LA Konstant Umschreiben angewendet werden, um den Wert für die neue Variable in das ursprüngliche Ziel

einzusetzen. Wird dies so lange durchgeführt, bis der minimale Faktor 1 ist, so kann anschließend die Variable eliminiert werden, deren Faktor 1 ist.

Das neu hinzugefügte Literal $u_1 + x \neq u_2$ wird wie in Abschnitt 3.2.1 beschrieben automatisch normalisiert.

Beispiel: Das Ziel

$$\langle 2x + 5y = 7 \,;\; w \rangle$$

kann durch Anwendung von $\neq$-Variablen-Elimination auf die Variable x zu

$$\langle x + 2y + z = 4 \;\vee\; 2x + 5y = 7 \,;\; w \rangle$$

abgeleitet werden, wobei z eine neue Variable ist. Anschließend kann die Inferenzregel LA Konstant Umschreiben, die im nächsten Abschnitt beschrieben ist, angewendet werden. Dieser Schritt ist im Beispiel für die Inferenzregel LA Konstant Umschreiben beschrieben.

Lemma 3.15 *Die Inferenzregel $\neq$-Variablen-Elimination ist korrekt und sicher.*

Beweis:

- **Korrektheit:** Sei

$$\frac{\langle \Gamma, t_1 \neq t_2, \Delta \,;\; w \rangle}{\langle u_1 + x \neq u_2, \Gamma, t_1 \neq t_2, \Delta \,;\; w \rangle}$$

  eine Instanz der Inferenzregel $\neq$-Variablen-Elimination.

  Für den Beweis der Korrektheit wird zunächst die induktive Gültigkeit zweier Lemmata gezeigt (die Definiertheit der Polynome wird hierbei vorausgesetzt, jedoch nicht explizit aufgeführt).

  (H1) $t_1 \dot{-} (\sum_{i=1}^{m_1} a_i'' t_i' + a'') = a_k * u_1$:
  Das Ziel

$$\left\langle \left(\sum_{i=1}^{m_1} a_i t_i' + a \right) \dot{-} \left(\sum_{i=1}^{m_1} a_i'' t_i' + a'' \right) = a_k * u_1 \,;\; w \right\rangle$$

  kann durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit den Axiomen für $+$ und $*$ zu

$$\left\langle \left(\sum_{i=1}^{m_1} ((a_k * a_i') + a_i'') t_i' + ((a_k * a') + a'') \right) \dot{-} \left(\sum_{i=1}^{m_1} a_i'' t_i' + a'' \right) = a_k * u_1 \,;\; w \right\rangle$$

  abgeleitet werden. Durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit *L4*, *L5* und *L6* kann dieses Ziel zu

$$\left\langle \left(a_k * u_1 + \left(\sum_{i=1}^{m_1} a_i'' t_i' + a'' \right) \right) \dot{-} \left(\sum_{i=1}^{m_1} a_i'' t_i' + a'' \right) = a_k * u_1 \,;\; w \right\rangle$$

  abgeleitet werden (da $u_1 = \sum_{i=1}^{m_1} a_i' t_i' + a'$). Wendet man darauf die Inferenzregel Nicht-induktive Termersetzung mit *L32* an, so erhält man das Ziel

$$\langle a_k * u_1 = a_k * u_1 \,;\; w \rangle \quad .$$

Dieses Ziel kann dann durch Anwendung der Inferenzregel =-Zerlegung abgeschlossen werden.

(H2) $t_2 + (\sum_{i=1}^{m_2} b_i'' t_i'' + b'') = a_k * u_2$:
Das Ziel

$$\left\langle \left(\sum_{i=1}^{m_1} b_i t_i'' + b \right) + \left(\sum_{i=1}^{m_1} b_i'' t_i'' + b'' \right) = a_k * u_2 \; ; \; w \right\rangle$$

kann durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit den Axiomen für + und $*$ zu

$$\left\langle \left(\sum_{i=1}^{m_1} ((a_k * b_i') \dot{-} b_i'') t_i'' + ((a_k * b') \dot{-} b'') \right) + \left(\sum_{i=1}^{m_1} b_i'' t_i'' + b'' \right) = a_k * u_2 \; ; \; w \right\rangle$$

abgeleitet werden. Durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit *L4*, *L5*, *L37* und *L38* kann dieses Ziel zu

$$\langle a_k * u_2 = a_k * u_2 \; ; \; w \rangle$$

abgeleitet werden (da $u_2 = \sum_{i=1}^{m_1} b_i' t_i'' + b'$). Dieses Ziel kann dann durch Anwendung der Inferenzregel =-Zerlegung abgeschlossen werden.

Nun kann der eigentliche Beweis begonnen werden. Das Ziel

$$\langle \Gamma, t_1 \neq t_2, \Delta \; ; \; w \rangle$$

kann durch Anwendung der Inferenzregel Literal Hinzufügen mit den beiden Literalen

$$\begin{aligned} \lambda_1 &:= \neg \mathrm{def}\,(u_2 \dot{-} u_1) \\ \lambda_2 &:= u_1 \not\leq u_2 \end{aligned}$$

zu den Zielen

$$\langle \mathrm{def}\,(u_2 \dot{-} u_1), \Gamma, t_1 \neq t_2, \Delta \; ; \; w \rangle \qquad (1)$$

$$\langle u_1 \leq u_2, \neg \mathrm{def}\,(u_2 \dot{-} u_1), \Gamma, t_1 \neq t_2, \Delta \; ; \; w \rangle \qquad (2)$$

$$\langle u_1 \not\leq u_2, \neg \mathrm{def}\,(u_2 \dot{-} u_1), \Gamma, t_1 \neq t_2, \Delta \; ; \; w \rangle \qquad (3)$$

abgeleitet werden. Das erste Ziel kann leicht durch Anwendung von *L29* und anschließender Ausnutzung von *PolyDefCond*(t_1) und *PolyDefCond*(t_2) gezeigt werden.

Das zweite Ziel kann ebenfalls gezeigt werden. Durch Anwendung der Inferenzregel Nicht-induktive Subsumption mit *L25* kann das Ziel zu

$$\langle t_1 \not\leq t_2, u_1 \leq u_2, \neg \mathrm{def}\,(u_2 \dot{-} u_1), \Gamma, t_1 \neq t_2, \Delta \; ; \; w \rangle$$

abgeleitet werden. Dieses Ziel kann anschließend durch Anwendung der Inferenzregel Nicht-induktive Subsumption mit *L34* zum Ziel

$$\left\langle t_1 \dot{-} \left(\sum_{i=1}^{m_1} a_i'' t_i' + a'' \right) \not\leq t_2 + \left(\sum_{i=1}^{m_2} b_i'' t_i'' + b'' \right), \; t_1 \not\leq t_2, u_1 \leq u_2, \neg \mathrm{def}\,(u_2 \dot{-} u_1), \Gamma, t_1 \neq t_2, \Delta \; ; \; w \right\rangle$$

abgeleitet werden. Durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit *H1* und *H2* entsteht daraus das Ziel

$$\langle a_k * u_1 \not\leq a_k * u_2, t_1 \not\leq t_2, u_1 \leq u_2, \neg\mathrm{def}\,(u_2 \dot{-} u_1), \Gamma, t_1 \neq t_2, \Delta\,;\, w\rangle \quad .$$

Durch Anwendung der Inferenzregel Nicht-induktive Termersetzung mit *L13* und anschließende Anwendung der Inferenzregel Komplementäre Literale kann dann das Ziel gezeigt werden.

Im dritten Ziel wird die Inferenzregel Konstrukorvariable Hinzufügen auf das Literal $\neg\mathrm{def}\,(u_2 \dot{-} u_1)$ angewendet. Dadurch entsteht das Ziel

$$\langle u_1 \not\leq u_2, x \neq u_2 \dot{-} u_1, \Gamma, t_1 \neq t_2, \Delta\,;\, w\rangle$$

mit $x \in V^C \setminus \mathit{Var}(\Gamma, t_1, t_2, \Delta, w)$. Durch Anwendung der Inferenzregeln Nicht-induktive Subsumption und Applikative Literal-Beseitigung mit *L36* kann dieses Ziel dann zu

$$\langle u_1 \not\leq u_2, x + u_1 \neq u_2, \Gamma, t_1 \neq t_2, \Delta\,;\, w\rangle$$

abgeleitet werden. Das noch überschüssige Literal $u_1 \not\leq u_2$ kann schließlich durch Anwendung der Inferenzregel Applikative Literal-Beseitigung mit *L35* entfernt werden. Somit kann das Ziel $\langle \Gamma, t_1 \neq t_2, \Delta\,;\, w\rangle$ durch Anwendung korrekter Inferenzregeln in das Ziel $\langle u_1 + x \neq u_2, \Gamma, t_1 \neq t_2, \Delta\,;\, w\rangle$ abgeleitet werden und die Inferenzregel $\neq$-Variablen-Elimination ist nach Lemma 3.5 korrekt.

- **Sicherheit:** Die Sicherheit folgt unmittelbar aus Lemma 3.1. □

3.2.6. Benutzen negativer Literale

LA Konstant Umschreiben:

`la-const-rewrite` $m\ n\ p$

$$\frac{\langle \Gamma, t_1\ \mathit{op}\ t_2, \Delta\,;\, w\rangle}{\langle \Gamma, u_1\ \mathit{op}\ u_2, \Delta\,;\, w\rangle}$$

falls

- $(\Gamma, t_1\ \mathit{op}\ t_2, \Delta)[n] = t_1\ \mathit{op}\ t_2$
- $t_1\ \mathit{op}\ t_2 = \sum_{i=1}^{m_1} a_i t_i^1 + a\ \mathit{op}\ \sum_{i=1}^{m_2} b_i t_i^2 + b$ in PNF
- es gibt ein Literal $t_3 \neq t_4$ in Γ, Δ, so dass $(\Gamma, t_1\ \mathit{op}\ t_2, \Delta)[m] = t_3 \neq t_4$
- $t_3 \neq t_4 = \sum_{i=1}^{m_3} c_i t_i^3 + c \neq \sum_{i=1}^{m_4} d_i t_i^4 + d$ in PNF
- p bezeichnet einen Summanden in der
  1. *linken* Seite des $\neq$-Literals, d. h. $(t_3 \neq t_4)|_p = c_j t_j^3$ für ein $j \in \{1, \ldots, m_3\}$
     - $c_j = 1$

- Der Summand t_j^3 kommt auch im Literal $t_1 \ op \ t_2$ vor, auf der
  a) *linken* Seite, d. h. $t_j^3 = t_k^1$ für ein $k \in \{1, \dots m_1\}$:
  $u_1 \ op \ u_2 = t_1 + a_k * t_4 \ op \ t_2 + a_k * t_3$
  b) *rechten* Seite, d. h. $t_j^3 = t_k^2$ für ein $k \in \{1, \dots m_2\}$:
  $u_1 \ op \ u_2 = t_1 + b_k * t_3 \ op \ t_2 + b_k * t_4$

2. *rechten* Seite des $\neq$-Literals, d. h. $(t_3 \neq t_4)|_p = d_j t_j^4$ für ein $j \in \{1, \dots, m_4\}$
   - $d_j = 1$
   - Der Summand t_j^4 kommt auch im Literal $t_1 \ op \ t_2$ vor, auf der
     a) *linken* Seite, d. h. $t_j^4 = t_k^1$ für ein $k \in \{1, \dots m_1\}$:
     $u_1 \ op \ u_2 = t_1 + a_k * t_3 \ op \ t_2 + a_k * t_4$
     b) *rechten* Seite, d. h. $t_j^4 = t_k^2$ für ein $k \in \{1, \dots m_2\}$:
     $u_1 \ op \ u_2 = t_1 + b_k * t_4 \ op \ t_2 + b_k * t_3$

- Γ, Δ enthält *PolyDefCond*(t_1) und *PolyDefCond*(t_2)

Diese Inferenzregel ist eine Erweiterung der bereits zuvor in QuodLibet vorhandenen Inferenzregel Konstant Umschreiben. Hiermit ist es auch möglich, umzuschreiben, wenn der umzuschreibene Term nicht alleine auf einer Seite steht. Das umgeschriebene Literal wird wie in Abschnitt 3.2.1 beschrieben automatisch normalisiert.

Beispiel: Das Ziel

$$\langle x + 2y + z \neq 4 \ \vee \ 2x + 5y \neq 7 \, ; \ w \rangle$$

kann durch Anwendung der Inferenzregel

`la-const-rewrite` 1 2 [1:1]

in das Ziel

$$\langle x + 2y + z \neq 4 \ \vee \ 2x + 5y + 2 * 4 \neq 7 + 2 * (x + 2y + z) \, ; \ w \rangle$$

überführt werden. Durch die automatische anschließende Normalisierung wird dieses Ziel dann zu

$$\langle x + 2y + z \neq 4 \ \vee \ y + 1 \neq 2z \, ; \ w \rangle$$

Lemma 3.16 *Die Inferenzregel LA Konstant Umschreiben ist korrekt und sicher.*

Beweis: Sei

$$\frac{\langle \Gamma, t_1 \ op \ t_2, \Delta \, ; \ w \rangle}{\langle \Gamma, u_1 \ op \ u_2, \Delta \, ; \ w \rangle}$$

eine Instanz der Inferenzregel LA Konstant Umschreiben mit Parametern m, n und p. Bezeichne p einen Summand in der linken Seite des $\neq$-Literals. Außerdem komme dieser Summand in der linken Seite des Literals $t_1 \ op \ t_2$ vor. Das heißt, $u_1 \ op \ u_2 = t_1 + (a_k * t_4) \ op \ t_2 + (a_k * t_3)$. Alle anderen Fälle sind analog beweisbar.

Das Ziel

$$\langle \Gamma, t_1 \ op \ t_2, \Delta \,;\, w \rangle$$

kann durch Anwendung der korrekten und sicheren Inferenzregel Nicht-induktive Subsumption mit *L17* zum Ziel

$$\langle a_k * t_4 \neq a_k * t_3, \Gamma, t_1 \ op \ t_2, \Delta \,;\, w \rangle$$

abgeleitet werden, da $a_k \neq 0$ gilt und Γ, Δ das Literal $t_3 \neq t_4$ enthält. Dieses Ziel kann dann durch eine weitere Anwendung von Nicht-induktive Subsumption mit *L43*, *L44* oder *L45* zu

$$\langle t_1 + (a_k * t_4) \ op \ t_2 + (a_k * t_3), a_k * t_4 \neq a_k * t_3, \Gamma, t_1 \ op \ t_2, \Delta \,;\, w \rangle$$

abgeleitet werden. In diesem Ziel ist das Literal $u_1 \ op \ u_2$ des Teilziels von LA Konstant Umschreiben bereits enthalten. Die beiden zusätzlichen Literale können durch Anwendung der korrekten und sicheren Inferenzregel Applikative Literal-Beseitigung mit *L39* und *L40*, *L41* oder *L42* entfernt werden, so dass das Teilziel

$$\langle \Gamma, u_1 \ op \ u_2, \Delta \,;\, w \rangle$$

abgeleitet werden kann.

Somit ist die Inferenzregel LA Konstant Umschreiben aufgrund von Lemma 3.5 und Lemma 3.3 korrekt und sicher. □

3.3. Taktiken für das Entscheidungsverfahren

Die im vorherigen Abschnitt beschriebenen Inferenzregeln reichen aus, um die manuelle Durchführung des Entscheidungsverfahrens zu ermöglichen. Zusätzlich soll das Entscheidungsverfahren jedoch auch in die Taktiken von QuodLibet integriert werden, so dass das Entscheidungsverfahren im Rahmen der Taktiken automatisch ausgeführt werden kann. Hierzu müssen die vorhandenen Taktiken angepasst und erweitert werden. Ein Anpassung ist insbesondere notwendig, weil durch die Integration des Entscheidungsverfahrens ein zusätzlicher Literaltyp im System verwendet wird, die $\leq$-Literale. Die bisherigen Taktiken können hiermit nicht umgehen und beispielsweise Lemmata für $\leq$-Literale werden nicht angewendet. Zudem müssen die Taktiken erweitert werden, so dass sie die neuen Inferenzregeln für das Entscheidungsverfahren auch anwenden, wenn entsprechende Literale vorhanden sind. Als Ausgangsbasis für die notwendigen Anpassungen und Erweiterungen der Taktiken werden die in [Sch04] beschriebenen neuen Standard-Taktiken von QuodLibet verwendet. Diese Taktiken sind in verschiedene Module aufgeteilt. Die für die Integration des Entscheidungsverfahrens notwendigen Erweiterungen der Taktiken werden im Wesentlichen im Modul Simplification vorgenommen.

3.3.1. Erweiterung des Moduls Simplification

Der Simplifikationsprozess im Modul Simplification ist in fünf Durchläufe gegliedert. Diese fünf Durchläufe werden teilweise erweitert, um die Inferenzregeln des Entscheidungsverfahrens zu integrieren. Zusätzlich wird ein weiterer Durchlauf definiert, in

dem versucht wird, mittels des Entscheidungsverfahrens einen Beweis zu finden. Dieser Durchlauf wird zwischen dem bisher zweiten und dem bisher dritten Durchlauf ausgeführt.

Der bisher erste Durchlauf besteht daraus, einfache Tautologien zu beweisen, die durch die Anwendung einer Inferenzregel ohne die Anwendung von Lemmata gezeigt werden können. Zusätzlich zu den bisherigen Inferenzregeln wird hier nun versucht, die Inferenzregeln aus Abschnitt 3.2.3 anzuwenden. Da bereits für die Anwendung dieser Inferenzregeln Bedingung ist, dass die Literale in PNF sind, werden zu Beginn dieses Durchlaufs alle Literale, die den Bedingungen der Inferenzregel LA-Literal-Normalisierung entsprechen mittels dieser Inferenzregel in PNF1 gebracht.

Im zweiten Durchlauf werden Inferenzregeln angewendet, die redundante Literale entfernen. Hierbei ist es wichtig, einen Kompromiss zwischen Aufwand und Nutzen zu finden. Ein zu hoher Aufwand für die Suche nach redundanten Literalen ist zu vermeiden, da die Entfernung eines solchen Literals zwar den Suchraum verkleinert, für die Beweisfindung aber nicht notwendig ist. Daher wird hier nur versucht, zusätzlich zu den bisherigen Inferenzregeln die Inferenzregel $\leq$-Beseitigung aus Abschnitt 3.2.4 anzuwenden, nicht jedoch die Inferenzregel $\leq$-Subsumptions-Beseitigung.

Nach dem zweiten Durchlauf wird der zusätzliche Durchlauf für die weiteren Inferenzregeln des Entscheidungsverfahrens für lineare Arithmetik durchgeführt. Dieser besteht nun daraus, abhängig vom Literaltyp zu versuchen, die verschiedenen, neuen Inferenzregeln des Entscheidungsverfahrens anzuwenden. Die Heuristiken, die hierbei verwendet werden, sind jedoch lediglich eine erste Annäherung. Sie sind zwar funktionsfähig, jedoch nicht besonders effizient. Die Verbesserung dieser Heuristiken bleibt als weiteres Forschungsfeld offen.

Bevor die weiteren Inferenzregeln des Entscheidungsverfahrens jedoch angewendet werden, werden zunächst alle Literale, die die nötigen Voraussetzungen erfüllen, durch Anwendung der Inferenzregel LA-Literal-Normalisierung in PNF2 gebracht. Literale, die in der reinen linearen Arithmetik liegen, das heißt, keine definierten Operatoren enthalten, werden dabei in PNF3 gebracht. Dieser Kompromiss hinsichtlich der Stufe der Normalformbildung ist dadurch begründet, dass im Rahmen des Entscheidungsverfahrens $\leq$-Literale besser genutzt werden können als Gleichungen. Sind jedoch auch definierte Operatoren vorhanden, so ist eine Umwandlung in $\leq$-Literale nicht unbedingt sinnvoll, da dies dazu führen kann, dass Bedingungsliterale der Lemmata der definierten Operatoren nicht mehr als erfüllt angesehen werden. Beispielsweise wird die Bedingung $x = 0$ nicht als erfüllt angesehen, wenn nur $x \leq 0$ enthalten ist. Dies führt dazu, dass entsprechende Lemmata möglicherweise nicht angewendet werden. Zudem kann es vorkommen, dass Beweise nach der Fallunterscheidung in beiden Fällen genau gleich geführt werden. Auch dies ist unerwünscht, da sich die Beweiskomplexität stark erhöht.

Der weitere Ablauf des zusätzlichen Durchlaufs gliedert sich wiederum in mehrere Phasen. Zunächst wird für jedes $\leq$-Literal, das im zu simplifizierenden Ziel vorkommt, versucht eine $\leq$-Variablenelimination mit einem anderen $\leq$-Literal der Klausel vorzunehmen. Entsteht bei der Variablenelimination ein neues Literal der Form $1 \leq 0$, so wird die Variablenelimination wieder zurückgenommen (da dieses Literal redundant ist) und stattdessen die Inferenzregel $\leq$-Fallunterscheidung angewendet.

Dies führt in diesem Fall dazu, dass genau ein neues Ziel erzeugt wird, in dem eine entsprechende Ungleichung vorkommt. Da dies jedoch nur im Fall $1 \leq 0$ und nicht allgemein im Fall $c \leq 0$ für $c \in \mathcal{N}$ gemacht wird, realisieren die Taktiken nicht vollständig das Entscheidungsverfahren. Hier ist gegebenenfalls ein Benutzereingriff notwendig, wenn das System das Ziel nicht auf andere Weise beweisen kann. Diese Maßnahme soll verhindern, dass automatisch Fallunterscheidungen in sehr viele Fälle gemacht werden. Stattdessen wird dem Benutzer die Entscheidung überlassen, bei wie vielen Fällen versucht werden soll, durch Fallunterscheidung das Ziel zu beweisen.

Anschließend an die Anwendung der Variablenelimination auf $\leq$-Literale wird versucht, Ungleichungen mittels der Inferenzregel LA Konstant Umschreiben auszunutzen. Dazu wird für jedes $\neq$-Literal überprüft, ob dieses einen oder mehrere Summanden mit Faktor 1 enthält. Ist dies der Fall, so wird jeder dieser Summanden auf Vorkommen in anderen Literalen untersucht. Wird ein solches Vorkommen gefunden, so wird überprüft, ob alle Variablen, die im $\neq$-Literal vorkommen auch im zweiten Literal vorkommen. Nur wenn dies der Fall ist, wird die Inferenzregel LA Konstant Umschreiben auf die beiden Literale und den gemeinsamen Summanden angewendet. Diese Heuristik soll verhindern, dass durch die Inferenzregel LA Konstant Umschreiben neue Variablen in Literale eingeführt werden, da dies meist nicht sinnvoll ist.

Zuletzt wird versucht, auch die Ungleichungen nutzbar zu machen, die keinen Summanden mit Faktor 1 enthalten. Dies wird durch Anwendung der Inferenzregel $\neq$-Variablen-Elimination realisiert. Diese Inferenzregel wird auf jede Ungleichung angewendet, deren minimaler Faktor eines Summanden größer als 1 ist. Dadurch entsteht eine neue Ungleichung, die einen kleineren minimalen Faktor enthält. Führt man dies solange durch, bis eine Ungleichung entsteht, die einen Summanden mit Faktor 1 enthält und somit im Rahmen der Inferenzregel LA Konstant Umschreiben nutzbar ist.

Die Anwendung von $\leq$-Variablenelimination, LA Konstant Umschreiben und $\neq$-Variablen-Elimination wird solange versucht, bis keine weitere Anwendung mehr möglich ist. Im Anschluss daran wird versucht, die Inferenzregel LA-Term-Normalisierung anzuwenden. Dazu wird in jedem Literal der Sorte Nat an allen geeigneten Positionen überprüft, ob eine Anwendung von LA-Term-Normalisierung eine Änderung im Term bewirken würde. Ist dies der Fall wird die Inferenz angewendet. Ist keine weitere Anwendung der Inferenz LA-Term-Normalisierung mehr möglich, so wird der zusätzliche Durchlauf beendet und der dritte Durchlauf begonnen.

Die bisher dritten und vierten Durchläufe bleiben im Wesentlichen unverändert. In diesen beiden Durchläufen wird versucht, das Ziel durch Anwendung von Lemmata zu zeigen. Während im dritten Durchlauf nur Lemmata angewendet werden, die direkt anwendbar sind, werden im vierten Durchlauf auch Lemmata angewendet, deren Anwendungsbedingungen gesondert gezeigt werden müssen. Beide Durchläufe werden vor allem um die Möglichkeit erweitert, auch $\leq$-Literale zu behandeln. Dies bedeutet, dass für alle diese Durchläufe zusätzliche Funktionen hinzugefügt werden, die die entsprechenden Operationen auf $\leq$-Literalen vornehmen. Zudem ist es notwendig, bei der Termersetzung die Berechnung alternativer Literalrepräsentationen

zu erweitern. Bisher war es hier beispielsweise bereits möglich, die Anwendungsbedingung $x = \mathsf{false}$ eines Lemmas als erfüllt anzusehen, wenn im Ziel das Literal $x \neq \mathsf{true}$ vorkommt. Dadurch, dass nun eine Theorie fest in das System integriert ist, ergeben sich zusätzliche alternative Literalrepräsentationen. Beispielsweise sollte die Anwendungsbedingung $x \leq y$ als erfüllt betrachtet werden, wenn das Literal $x < y+1$ im Ziel vorkommt und umgekehrt. Derzeit werden nur einige relativ einfach zu überprüfende Alternativen beachtet. Durch einen höheren Aufwand bei der Suche nach Literalen in alternativer Repräsentation ist es möglich, die Suche nach anwendbaren Lemmata zu verbessern. Wie viel Aufwand für die Bestimmung alternativer Literalrepräsentationen hierbei sinnvoll ist, muss in Praxiserprobungen festgestellt werden.

Der fünfte Durchlauf, in dem bisher nur die Inferenzregel Konstant Umschreiben angewendet wurde, bleibt vollkommen unverändert.

3.3.2. Möglichkeiten zur Verbesserung der Taktiken

Die im vorherigen Abschnitt beschriebenen Erweiterungen und Anpassungen der Taktiken realisieren bereits im Wesentlichen das im Abschnitt 3.1 beschriebene Entscheidungsverfahren. Dadurch, dass stets alle möglichen Variableneliminationen durchgeführt werden und danach eine Überprüfung auf Tautologien durchgeführt wird, ist die Fourier-Motzkin-Variablenelimination realisiert.

Die Erweiterung des Fourier-Motzkin-Verfahrens auf natürliche Zahlen wird jedoch nicht vollständig verwirklicht, da die Fallunterscheidung nur angewendet wird, wenn dadurch genau ein Fall entsteht. Dadurch können auch einige induktive Theoreme der reinen linearen Arithmetik, wie das im Abschnitt 3.2.5 beschriebene Beispiel, nicht bewiesen werden. Daher sollte eine bessere Heuristik für die $\leq$-Fallunterscheidung entwickelt werden. Hierbei wäre es sinnvoll, mit verschiedenen Werten für die Grenze der Anzahl der automatisch erzeugten Fälle zu experimentieren.

Im Bereich der Heuristiken gibt es einige weitere Ansatzpunkte für eine Weiterentwicklung der Taktiken. Dass stets alle möglichen Variableneliminationen in willkürlicher Reihenfolge durchgeführt werden, ist zwar ausreichend, um sicherzustellen, dass alle Schritte der Fourier-Motzkin-Variablenelimination auch durchgeführt werden. Hier könnte jedoch eine deutlich bessere Heuristik entwickelt werden, indem (ähnlich wie bei geordneter Paramodulation im Bereich des deduktiven Theorembeweisens) eine Ordnung auf den Summanden etabliert wird, die es erlaubt, nur in dieser Ordnung maximale Variablen zu eliminieren. Dies ist dadurch zu begründen, dass die Variablenelimination nur zum Ziel führt, wenn alle Variablen eliminiert wurden. Kann daher eine maximale Variable nicht eliminiert werden, so müssen auch kleinere Variablen nicht eliminiert werden, da die maximale Variable verhindert, dass eine (variablenfreie) Tautologie entsteht. Die maximale Variable wird auch durch Elimination kleinerer Variablen nicht eliminierbar werden.

Die Anwendung von LA Konstant Umschreiben steht derzeit noch in einem Konflikt mit der Anwendung der ähnlichen Inferenzregel Konstant Umschreiben. Es gibt Literale, auf die beide Inferenzregeln angewendet werden können. Da die Inferenzregel LA Konstant Umschreiben derzeit stets vor Konstant Umschreiben ausgeführt

wird, kann es vorkommen, dass durch die Anwendung von Konstant Umschreiben der Effekt der Anwendung von LA Konstant Umschreiben genau umgekehrt wird. Um dies zu verhindern müsste die Kontrolle über die Anwendung dieser beiden Inferenzregeln koordiniert werden. Alternativ könnte man versuchen, diese beiden sehr ähnlichen Inferenzregeln zu einer Inferenzregel zusammen zu fassen. Dann könnte die Vermeidung des Umkehreffektes lokal für diese Inferenzregel versucht werden.

Ein weiteres großes Potential bietet die Anwendung von Lemmata und Induktionshypothesen. Hier entsteht eine gewisse Problematik dadurch, dass das System zwar nun mit einer eingebauten Theorie arbeitet, aber die Matching-Operation nicht modulo dieser Theorie ausgeführt wird. Erste Versuche, die Matching-Operation entsprechend zu erweitern, erwiesen sich als ungeeignet, da das System zu sehr ausbremst wird. Die Entscheidung, wie stark beim Matching die lineare Arithmetik betrachtet werden soll, kann jedoch ohne Anpassung des Kernsystems in die Taktiken verlagert werden. Dies kann dadurch realisiert werden, dass innerhalb der Taktiken eine Funktion implementiert wird, die für die Anwendung einer Termersetzungs-Inferenzregel mit einem hinreichenden Verfahren einen Match μ modulo der Theorie der linearen Arithmetik bestimmt. Anschließend wird mittels der Inferenzregel Literal Hinzufügen ein der Anwendungsbedingung der Termersetzung entsprechendes Literal $\lambda/p = l\mu$ hinzugefügt. Das Ziel, das anschließend das Literal $\lambda/p = l\mu$ selbst enthält, kann mittels des Entscheidungsverfahrens bewiesen werden. Das zweite Ziel, das entsteht, enthält das Literal $\lambda/p \neq l\mu$. Dieses Literal kann dann durch Anwendung der Inferenzregel Konstant Umschreiben so genutzt werden, dass im ursprünglichen Literal λ/p durch $l\mu$ ersetzt wird. Damit ist der zuvor bestimmte Match modulo der Theorie der linearen Arithmetik auch ein syntaktischer Match und die Inferenzregel für die Termersetzung kann angewendet werden.

Auch im Bereich der Subsumption könnte die Semantik von $\leq$-Literalen noch besser ausgenutzt werden. Hat man beispielsweise das Lemma $\mathsf{f}(x) \leq \mathsf{5}$ bereits gezeigt, so kann das Ziel $\mathsf{f}(x) \leq \mathsf{6}$ leicht gezeigt werden, indem die Inferenzregel Nicht-induktive Subsumption mit $\mathsf{f}(x) \leq \mathsf{5}$ angewendet wird. Anschließend enthält das Ziel die beiden Literale $\mathsf{f}(x) \leq \mathsf{6}$ und $\mathsf{f}(x) \not\leq \mathsf{5}$ und kann durch Anwendung der Literal-Normalisierung auf letzteres und anschließende $\leq$-Variablenelimination zur Tautologie $\mathsf{0} \leq \mathsf{1}$ abgeleitet werden. Dies könnte im Rahmen der Taktiken bei der Suche nach einem Subsumptionslemma beachtet werden, so dass in mehr Fällen eine Subsumption ausgeführt werden könnte, anstatt das neue Ziel auf eine andere Art zu beweisen.

4. Auswertung

Auch wenn die Taktiken noch deutliches Verbesserungspotenzial enthalten, können schon erste Auswertungen vorgenommen werden, um festzustellen, in wie weit die erwarteten Effekte eintreten. Dazu werden im folgenden Abschnitt drei Beispiele betrachtet, aus deren einzelnen Auswertungen anschließend die wesentlichen Grundtendenzen bestimmt werden.

4.1. Auswirkung der Integration des Entscheidungsverfahrens in Beispielspezifikationen

Um den Effekt der Integration des Entscheidungsverfahrens auszuwerten, werden drei Spezifikationen verwendet, die Probleme aus der Domäne der natürlichen Zahlen und damit einen relativ hohen Anteil an linearer Arithmetik beinhalten (die Spezifikationen sind im Anhang B abgedruckt). Diese Spezifikationen werden – sofern möglich – mit dem bisherigen System ohne lineare Arithmetik und mit dem neuen System mit integrierter linearer Arithmetik bearbeitet. Dabei wird jeweils versucht, möglichst viele Beweise automatisch durch Aufruf von Taktiken zu finden. Die Beweise werden jeweils zunächst mit dem System ohne lineare Arithmetik erstellt. Anschließend werden die Beweisskripte im System mit linearer Arithmetik ausgeführt und angepasst, wo es notwendig ist. Zudem werden Lemmata, die beim Beweis mit dem System mit linearer Arithmetik nicht mehr benötigt werden, entfernt. Dieses Vorgehen sichert eine relativ gute Vergleichbarkeit, da ähnliche Beweise geführt werden müssen. Allerdings ergibt sich hieraus auch eine Benachteiligung des Systems mit linearer Arithmetik. Dies liegt daran, dass die Spezifikationen nicht für das System mit linearer Arithmetik optimiert sind. Durch eine im Kontext der linearen Arithmetik bessere Formulierung der Lemmata könnte auch ein besseres Gesamtergebnis erreicht werden. Hierfür wäre es notwendig, die in [Sch04] beschriebenen Richtlinien für gute Spezifikationen zu erweitern und anzupassen. Dann müssten die gesamten Beweise von Beginn an mit dem System mit integrierter Arithmetik neu geführt werden. Im Rahmen dieser Arbeit war ein solches Vorgehen jedoch aufgrund des deutlich höheren Zeitaufwandes leider nicht möglich.

Wie in [Sch04] werden die beiden Versionen des Systems hinsichtlich der folgenden Aspekte verglichen:

- *Lemmata*: die Zahl der Lemmata, die in der Spezifikation verwendet werden. Diese dient als Hinweis auf die Komplexität des Spezifikation. Durch rekursive Beweisstrategien automatisch erzeugte Lemmata werden in Klammern angegeben.

- *Manuelle Anwendungen*: die Zahl der manuell angewendeten Inferenzregeln. Diese lässt erkennen, wie gut die Taktiken automatisch einen Beweis finden.
- *Gewicht*: die Anzahl der notwendigen manuellen Instanziierungen von Gewichtsvariablen um eine passende Induktionsordnung zu bestimmen.
- *Automatische Anwendungen*: die Zahl der automatisch und erfolgreich durch die aufgerufenen Taktiken angewendeten Inferenzen. Diese Zahl lässt Schlüsse auf die Komplexität der automatisch gefundenen Beweise zu.
- *Löschungen*: die Zahl der zurückgenommenen Anwendungen von Inferenzregeln. Diese entstehen durch Fehlschläge beim Beweis eines Bedingungsziels bei bedingter Anwendung von Lemmata aber auch im Rahmen des Entscheidungsverfahrens beispielsweise bei der Rücknahme von $\leq$-Variablen-Elimination, falls dadurch ein redundantes Literal entsteht.
- *Laufzeit*: die Laufzeit in Sekunden für den Beweis der gesamten Spezifikation, gemessen von einem CMU-Common-Lisp-System auf einem Rechner mit 400 MHz Intel Celeron Prozessor und 256 MB RAM. Dabei werden die Zeiten für Benutzerinteraktion nicht berücksichtigt, sondern nur die reine Rechenzeit gemessen.

4.1.1. Das Beispiel ggT

Das Beispiel ggT enthält eine Spezifikation, deren Ziel es ist, zu beweisen, dass der ggT idempotent, kommutativ und assoziativ ist. Dieses Beispiel hat einen relativ hohen Anteil an linearer Arithmetik. Die Daten des Vergleichs der alten Version ohne lineare Arithmetik und der neuen mit linearer Arithmetik sind in Tabelle 4.1 enthalten.

Kriterium	ohne LA	mit LA
Lemmata	81 (+1)	53
manuelle Anwendungen	24	25
Gewicht	2	2
automatische Anwendungen	1059	2614
Löschungen	10	298
Laufzeit	5,99	21,60

Tabelle 4.1.: Ergebnisse des Beispiels ggT

Zunächst fällt bei den Ergebnissen auf, dass die Zahl der benötigten Lemmata bei integrierter linearer Arithmetik deutlich geringer ist. Dies liegt daran, dass viele Lemmata Theoreme der reinen linearen Arithmetik beinhalteten. Sie können nun durch die Anwendung des Entscheidungsverfahrens ersetzt werden. Allerdings führt dies zu einer deutlichen Steigerung der Anzahl der automatischen Anwendungen.

Negativ fällt weiterhin auf, dass eine zusätzliche manuelle Anwendung hinzugekommen ist. Sie besteht aus einer nicht-induktiven Termersetzung mit einem Lemma, dessen Anwendungsbedingung die Form $x = 0$ hat. Diese Anwendungsbedingung ist

an der konkreten Stelle zwar im Ziel enthalten, wurde aber durch Anwendung der Normalisierung dritter Stufe in zwei Fälle $x \leq 0$ und $0 \leq x$ aufgeteilt. Somit ist die Anwendungsbedingung nicht mehr direkt im Ziel enthalten. Dies führt dazu, dass die Taktik zunächst versucht, andere Lemmata anzuwenden, die keine Anwendungsbedingungen haben oder deren Anwendungsbedingungen bereits im Ziel enthalten sind. Dadurch wählt die Taktik das „falsche“ Lemma aus und scheitert im Anschluss. Dieses Problem könnte vermieden werden, indem die Taktiken so erweitert werden, dass auch $x \leq 0$ für die Erfüllung einer Anwendungsbedingung $x = 0$ zugelassen wird, da diese beiden Literale äquivalent sind, falls x definiert ist. Zudem könnte auch die Anwendungsbedingung des Lemmas zu $x \leq 0$ umformuliert werden, was jedoch in den hier nicht betrachteten Bereich der Verbesserung von Spezifikationen im Kontext der linearen Arithmetik fällt.

Die Laufzeit des Beispiels hat sich im Vergleich zum System ohne lineare Arithmetik deutlich (um etwa den Faktor 4) verschlechtert. Dies liegt vor allem daran, dass die Taktik für das Entscheidungsverfahren derzeit noch mit einer sehr einfachen Heuristik arbeitet und daher viele nutzlose Inferenzschritte ausgeführt werden.

4.1.2. Das Beispiel $\sqrt{2}$

Das Beispiel $\sqrt{2}$ enthält eine Spezifikation, deren Ziel es ist, zu beweisen, dass $\sqrt{2}$ irrational ist. Auch dieses Beispiel hat einen relativ hohen Anteil an linearer Arithmetik. Die Daten des Vergleichs der alten Version ohne lineare Arithmetik und der neuen mit linearer Arithmetik sind in Tabelle 4.2 enthalten.

Kriterium	ohne LA	mit LA
Lemmata	51 (+2)	22
manuelle Anwendungen	11	7
Gewicht	1	1
automatische Anwendungen	1005	1144
Löschungen	28	540
Laufzeit	11,76	13,57

Tabelle 4.2.: Ergebnisse des Beispiels $\sqrt{2}$

Bei diesem Beispiel ist eine noch deutlichere Reduzierung der benötigten Lemmata zu erkennen als beim Beispiel ggT. Es werden mit linearer Arithmetik deutlich weniger als die Hälfte der Lemmata benötigt. Auch die Anzahl der manuellen Anwendungen ist deutlich gesunken, wohingegen die Anzahl der automatischen Anwendungen leicht gestiegen ist. Allerdings wird ein sehr großer Anteil der automatischen Anwendungen wieder gelöscht. Die Differenz aus automatischen Anwendungen und gelöschten Anwendungen – das heißt die tatsächlich in den Beweisbäumen enthaltenen Anwendungen – sinkt dadurch sogar von 977 auf 604.

Betrachtet man im Detail die Anzahl der Anwendungen der einzelnen Inferenzregeln, so fällt auf, dass im System mit linearer Arithmetik mit Abstand am häufigsten die Inferenzregeln $\leq$-Variablen-Elimination (326), LA-Literal-Normalisierung (279) und Mehrfache Literale (223) angewendet werden. Die Anzahl der Anwendungen von

Nicht-induktive Termersetzung (62 statt 470) und Nicht-induktive Subsumption (54 statt 246) ist deutlich gesunken. Wie zu erwarten war, wurde somit ein Großteil der Anwendungen nicht-induktiver, applikativer Inferenzregeln durch die Anwendung von Inferenzregeln des Entscheidungsverfahrens ersetzt. Auffällig ist, dass im System mit linearer Arithmetik die Inferenzregel Mehrfache Literale sehr häufig angewendet wird. Durch eine Verbesserung der neuen Inferenzregeln könnte dies reduziert werden, indem mehrfache Literale gar nicht erst erzeugt werden. Dadurch könnten die Beweisbäume weiter verkleinert werden, ein deutlicher Einfluss auf die Laufzeit ist jedoch nicht zu erwarten.

Allerdings ist auch zu beobachten, dass ein sehr großer Anteil der Anwendungen von $\leq$-Variablen-Elimination wieder gelöscht wird (249 von 326). Hier könnte durch eine verbesserte Heuristik noch einiges eingespart werden.

Insgesamt ist eine leichte Erhöhung der Laufzeit um etwa 15% zu beobachten. Der größte Teil dieser Zeit wird damit verbracht, ein einziges Lemma zu beweisen. Bei diesem Beweis wird ein sehr hoher Anteil an Normalisierungen und Variableneliminationen durchgeführt und auch der weitaus größte Teil der Löschungen von Inferenzregelanwendungen tritt hier auf. Verbesserte, zielgerichtetere Taktiken könnten hierbei die Laufzeit deutlich verringern.

4.1.3. Das Beispiel 91er-Funktion

Das letzte Beispiel beinhaltet eine Spezifikation von John McCarthys 91er-Funktion [MM70] und enthält als Ziel den Beweis der Termination der 91er-Funktion. Diese Funktion ist rekursiv wie folgt definiert:

$$f91(x) := \begin{cases} x \dot{-} 10 & \text{falls } n > 100 \\ f91(f91(x+11)) & \text{sonst} \end{cases}$$

Für Werte kleiner oder gleich 101 hat die 91er-Funktion stets das Ergebnis 91 und für alle größeren Werte das Ergebnis $x \dot{-} 10$. Der Beweis, dass die Funktion terminiert, wird als guter Testfall für induktive Theorembeweiser angesehen.

Bisher ist mit QuodLibet ein solcher Beweis nicht gelungen. Dies liegt vor allem daran, dass mit relativ großen Zahlen gerechnet wird, die bisher in QuodLibet durch Konstruktorterme in s-Darstellung repräsentiert wurden. Dadurch entstehen sehr große Ziele, der Termersetzungs-Prozess wird sehr aufwändig und der Suchraum insgesamt sehr groß.

Mit Hilfe der integrierten linearen Arithmetik, der dazu als Kurzschreibweise definierten Konstanten für natürliche Zahlen und dem Entscheidungsverfahren gelingt es nun, die Termination der 91er-Funktion zu zeigen. Bei diesem Beispiel wurde der Beweis zudem von vornherein im System mit linearer Arithmetik geführt, so dass die bei den beiden vorherigen Beispielen beschriebene Benachteiligung dieses Systems nicht eintritt.

Kriterium	ohne LA	mit LA
Lemmata	-	5
manuelle Anwendungen	-	7
Gewicht	-	3
automatische Anwendungen	-	503
Löschungen	-	157
Laufzeit	-	4,01

Tabelle 4.3.: Ergebnisse des Beispiels 91er-Funktion

Der Vollständigkeit halber sind in Tabelle 4.3 die Ergebnisse der Spezifikation des Beispiels 91er-Funktion aufgeführt, auch wenn hier natürlich kein Vergleich zur Version ohne lineare Arithmetik möglich ist.

4.2. Grundtendenzen in den Auswertungen der Beispiele

Aus der Auswertung der Ergebnisse der drei Beispiele ergeben sich einige Grundtendenzen, die jedoch aufgrund der geringen Anzahl von Beispielen nicht als gefestigt angesehen werden können.

Zunächst ist festzustellen, dass das Ziel einer deutlichen Reduzierung der Anzahl der Lemmata, die für Beweise mit linearer Arithmetik benötigten werden, anscheinend erreicht wurde. Dies entspricht der Erwartung, da viele der bisher benötigten Lemmata einfache Aussagen der linearen Arithmetik enthielten. Diese Lemmata müssen nun nicht mehr explizit formuliert werden, sondern werden durch das Entscheidungsverfahren abgedeckt. Dies bedeutet für den Benutzer eine deutliche Entlastung, da die Lemmata von QuodLibet nicht automatisch gefunden werden, sondern vom Benutzer eingegeben werden müssen. Diese Entlastung könnte zwar auch durch eine Sammlung häufig benötigter Lemmata erreicht werden, die dem Benutzer zur Verfügung gestellt wird. Allerdings müsste der Benutzer dem System dennoch mitteilen, welche dieser Lemmata im Rahmen der Taktiken verwendet werden sollen, da sonst der Suchraum viel zu groß würde.

Weiterhin ließ sich zumindest bei einem Beispiel auch die Zahl der manuellen Anwendungen von Inferenzregeln deutlich reduzieren. Auch dies entlastet den Benutzer, da manuelle Anwendungen stets bedeuten, dass der Benutzer den Stand des Beweises, an dem eine Taktik abgebrochen wurde, erfassen und durch manuellen Eingriff den Beweis fortsetzen muss. Wenn die Beispiele hinsichtlich des Einsatzes mit linearer Arithmetik optimiert würden, könnte hier wahrscheinlich ein noch besserer Wert erzielt werden.

Diesen deutlich erkennbaren Vorteilen stehen jedoch auch zwei Nachteile gegenüber. Zum einen erhöht sich die Gesamtlaufzeit des neuen Systems gegenüber dem System ohne lineare Arithmetik. Diese ist jedoch angesichts der Verringerung der notwendigen manuellen Eingriffe relativ gering. Denn jeder manuelle Eingriff, um ein neues Lemma einzugeben oder manuell Inferenzregeln anzuwenden, bedeutet eine erheblich größere Verzögerung. Dadurch ist die Zeit, die ein Benutzer für die

Beweisfindung benötigt, mit dem neuen System dennoch deutlich geringer. Ähnlich verhält es sich mit der Zahl der automatische angewendeten Inferenzregeln. Diese steigt zwar mit der Integration der linearen Arithmetik teilweise deutlich an, was jedoch dem Benutzer im Wesentlichen nur den bereits diskutierten Nachteil einer längeren Laufzeit beschert und die Beweisbäume etwas komplexer macht. Wenn jedoch im Gegenzug weniger manuelle Eingriffe nötig sind, so sind auch komplexere Beweisbäume verkraftbar, da der Benutzer sich nicht mehr so häufig in einen aktuellen Beweiszustand einarbeiten muss.

Die erwähnten Nachteile resultieren zudem hauptsächlich aus den relativ rudimentären Taktiken. Es ist zu erwarten, dass diese deutlich weniger stark in Erscheinung treten, wenn die in Abschnitt 3.3.2 beschrieben Verbesserungspotentiale ausgeschöpft werden.

5. Zusammenfassung und Ausblick

Zielsetzung dieser Arbeit war es, die Integration von Entscheidungsverfahren in den induktiven Theorembeweiser QuodLibet am Beispiel der linearen Arithmetik zu untersuchen. Um diese Integration zu realisieren, wurden zunächst die notwendigen Basisstrukturen in das System integriert. Dazu gehören die Operatoren der linearen Arithmetik, ein neuer Literaltyp für die Darstellung von $\leq$-Beziehungen und eine effizientere Repräsentation von natürlichen Zahlen im System.

Anschließend wurde das Entscheidungsverfahren der Fourier-Motzkin-Variablenelimination in QuodLibet integriert. Dazu wurde das Verfahren auf den Bereich der natürlichen Zahlen erweitert und für die Verwendung im formalen Kontext von QuodLibet angepasst. Um das Verfahren zu realisieren, wurden dem System zehn neue Inferenzregeln hinzugefügt, die den einzelnen Schritten des Entscheidungsverfahrens entsprechen. Das Entscheidungsverfahren wird dann durch die Anwendung dieser Inferenzregeln in erweiterten Standard-Taktiken ausgeführt.

Die Auswertung der Ergebnisse aus den Beispielspezifikationen hat gezeigt, dass die erhofften positiven Effekte durch diese Integration der linearen Arithmetik erreicht wurden. In Spezifikationen, die Ziele mit linearer Arithmetik enthalten, kann eine deutliche Reduzierung der benötigten Lemmata erreicht werden. Auch die Anzahl der notwendigen manuellen Eingriffe in den Beweisprozess sinkt tendenziell. Im Gegenzug steigt die benötigte Rechenzeit sowie die Anzahl der automatisch angewendeten Inferenzregeln. Dies ist jedoch wesentlich dadurch bedingt, dass die Taktiken noch nicht das volle Potential des Entscheidungsverfahrens ausschöpfen. Insgesamt ist somit eine bessere, aber aufwändigere Automatisierung des Systems gelungen. Insbesondere bei Beispielen mit größeren Zahlkonstanten macht sich die verbesserte Darstellung der Zahlen deutlich bemerkbar, so dass Theoreme bewiesen werden können, deren Beweis zuvor nicht gelang.

Es ist jedoch noch ein großes Potential für Verbesserungen der Taktiken vorhanden. Einige mögliche Verbesserungen wurden im Abschnitt 3.3.2 bereits beschrieben. Dieses Potential kann im Rahmen weiterer Arbeiten zu diesem Thema ausgeschöpft werden. Es ist zu erwarten, dass dadurch die Zahl der automatisch angewendeten Inferenzregeln und auch die Laufzeit deutlich reduziert werden können. Eine bessere Einschätzung des Erfolgs der Integration des Entscheidungsverfahrens wäre durch die Auswertung einer größeren Anzahl von Beispielen möglich. Würde diese im Rahmen weiterer Arbeiten durchgeführt, so erhielte man auch eine gute empirische Grundlage für die weitere Verbesserung der Taktiken.

A. Beweise der verwendeten Lemmata

Die Beweise, dass die in Abbildung 2.1 und Abbildung 3.1 aufgelisteten Lemmata in jeder Spezifikation mit linearer Arithmetik induktiv gültig sind, werden hier in Form eines QuodLibet-Skriptes angegeben. Führt man dieses Skript mit dem bisherigen QuodLibet-System aus, so werden Beweise für alle verwendeten Lemmata erzeugt. Das Skript besteht zunächst aus Anweisungen, die eine Spezifikation mit linearer Arithmetik erzeugen. Anschließend werden einige zusätzliche Lemmata bewiesen, die für die Beweise der eigentlichen Lemmata benötigt werden. Zuletzt wird für jedes der Lemmata *L1* bis *L46* ein Beweis erzeugt. Die Beweise für *L19* und *L20* gelingen jedoch nur in eine Richtung, da die Inferenzregeln von QuodLibet für die Behandlung negativer Ordnungsatome nicht ausgelegt sind. Da diese Lemmata jedoch von untergeordneter Wichtigkeit sind, kann hier auf den Beweis der zweiten Richtung verzichtet werden.

QuodLibet-Skript: Beweise der verwendeten Lemmata

```
initialize
call initialize-database

define sort Bool with constructors
    false :  --> Bool
    true :  --> Bool
    .
declare constructor variables b : Bool.

assume
    { b = false,
      b = true }
    bool-complete
call auto-strategy  bool-complete
call activate-lemma  bool-complete

define sort Nat with constructors
    0 :  --> Nat
    s : Nat --> Nat
    .
declare constructor variables u, v, w, x, y, z : Nat.

declare operators
    leq : Nat Nat --> Bool
```

```
    .
assert leq-1 :
    leq(0,y) = true
    .
assert leq-2 :
    leq(s(x),0) = false
    .
assert leq-3 :
    leq(s(x),s(y)) = leq(x,y)
    .

call analyze-operator  leq
call auto-strategy  leq-def-auto
call activate-lemma  leq-def-auto

assume
    { leq(x,y) = true,
      leq(y,x) = true }
    leq-complete
call auto-strategy  leq-complete
call activate-lemma  leq-complete

assume
    { leq(x,x) = true }
    leq-x-x
call auto-strategy  leq-x-x
call activate-lemma  leq-x-x

assume
    { leq(x,s(y)) = true,
      leq(x,y) =/= true }
    leq-x-sy
call auto-strategy  leq-x-sy
call activate-lemma  leq-x-sy

assume
    { leq(s(x),y) = false,
      leq(x,y) =/= false }
    leq-sx-y
call auto-strategy  leq-sx-y
call activate-lemma  leq-sx-y

assume
    { leq(x,0) =/= true,
      x = 0 }
    leq-x-0
call auto-strategy  leq-x-0
```

```
call activate-lemma  leq-x-0

assume
    { leq(x,y) =/= true,
      leq(y,x) =/= true,
      x = y }
    leq-trichotomy
call auto-strategy  leq-trichotomy

declare operators
    plus : Nat Nat --> Nat
    .
assert plus-1 :
    plus(x,0) = x
    .
assert plus-2 :
    plus(x,s(y)) = s(plus(x,y))
    .

call analyze-operator  plus
call auto-strategy  plus-def-auto
call activate-lemma  plus-def-auto

assume
    { plus(0,y) = y }
    plus-0-y
call auto-strategy  plus-0-y
call activate-lemma  plus-0-y

assume
    { plus(s(x),y) = s(plus(x,y)) }
    plus-sx-y
call auto-strategy  plus-sx-y
call activate-lemma  plus-sx-y

assume
    { plus(x,y) = plus(y,x) }
    plus-commutative
call auto-strategy  plus-commutative
call activate-lemma  plus-commutative

assume
    { plus(x,plus(y,z)) = plus(y,plus(x,z)) }
    plus-comm-ext
call auto-strategy  plus-comm-ext
call activate-lemma  plus-comm-ext
```

```
assume
    { plus(plus(x,y),z) = plus(x,plus(y,z)) }
    plus-ass
call auto-strategy  plus-ass
call activate-lemma  plus-ass

assume
    { leq(x,plus(y,z)) = true,
      leq(x,y) =/= true }
    leq-x-plus-mono
call auto-strategy  leq-x-plus-mono
call activate-lemma  leq-x-plus-mono

assume
    { leq(x,plus(z,y)) = true,
      leq(x,y) =/= true }
    leq-x-plus-mono-2
call auto-strategy  leq-x-plus-mono-2
call activate-lemma  leq-x-plus-mono-2

assume
    { leq(plus(u,v),plus(x,y)) = true,
      leq(u,x) =/= true,
      leq(v,y) =/= true }
    leq-plus-mono
call auto-strategy  leq-plus-mono
call activate-lemma  leq-plus-mono

assume
    { leq(plus(x,z),plus(y,z)) = leq(x,y) }
    leq-plus-2
call auto-strategy  leq-plus-2
call activate-lemma  leq-plus-2

assume
    { leq(plus(z,x),plus(z,y)) = leq(x,y) }
    leq-plus-1
call simplify  leq-plus-1
call activate-lemma  leq-plus-1

declare operators
    times : Nat Nat --> Nat
    .
assert times-1 :
    times(x,0) = 0
    .
assert times-2 :
```

```
    times(x,s(y)) = plus(times(x,y),x)
    .

call analyze-operator  times
call auto-strategy  times-def-auto
call activate-lemma  times-def-auto

assume
    { times(0,y) = 0 }
    times-0-y
call auto-strategy  times-0-y
call activate-lemma  times-0-y

assume
    { times(s(x),y) = plus(times(x,y),y) }
    times-sx-y
call auto-strategy  times-sx-y
call activate-lemma  times-sx-y

assume
    { times(s(0),x) = x }
    times-s0
call auto-strategy  times-s0
call activate-lemma  times-s0

assume
    { times(x,y) = times(y,x) }
    times-commutative
call auto-strategy  times-commutative
call activate-lemma  times-commutative

assume
    { times(x,plus(y,z)) = plus(times(x,y),times(x,z)) }
    times-distr-1
call auto-strategy  times-distr-1
call activate-lemma  times-distr-1

assume
    { times(x,times(y,z)) = times(y,times(x,z)) }
    times-comm-ext
call auto-strategy  times-comm-ext
call activate-lemma  times-comm-ext

call deactivate-lemmas  { times-distr-1 }

assume
    { leq(times(x,z),times(y,z)) = true,
```

```
      leq(x,y) =/= true }
    leq-times-2
call auto-strategy  leq-times-2
call activate-lemma  leq-times-2

assume
    { leq(times(z,x),times(z,y)) = true,
      leq(x,y) =/= true }
    leq-times-1
call simplify  leq-times-1
call activate-lemma  leq-times-1

declare operators
    minus : Nat Nat --> Nat
    .
assert minus-1 :
    minus(x,0) = x
    .
assert minus-2 :
    minus(s(x),s(y)) = minus(x,y)
    .
assert minus-3 :
    minus(0,x) = 0
    .

call analyze-operator  minus
call auto-strategy  minus-def-auto
call activate-lemma  minus-def-auto

assume
    { minus(plus(x,y),y) = x }
    minus-plus-2
call auto-strategy  minus-plus-2
call activate-lemma  minus-plus-2

assume
    { minus(plus(y,x),y) = x }
    minus-plus-1
call simplify  minus-plus-1
call activate-lemma  minus-plus-1

assume
    { minus(plus(x,z),plus(y,z)) = minus(x,y) }
    minus-plus-3
call auto-strategy  minus-plus-3
call activate-lemma  minus-plus-3
```

```
assume
    { minus(plus(z,x),plus(z,y)) = minus(x,y) }
    minus-plus-4
call simplify  minus-plus-4
call activate-lemma  minus-plus-4

assume
    { plus(y,minus(x,y)) = x,
      leq(y,x) =/= true }
    plus-minus-2
call auto-strategy  plus-minus-2
call activate-lemma  plus-minus-2

assume
    { plus(minus(x,y),y) = x,
      leq(y,x) =/= true }
    plus-minus-1
call auto-strategy  plus-minus-1
call activate-lemma  plus-minus-1

assume
    { plus(x,z) =/= plus(y,z),
      x = y }
    plus-inj-1
call auto-strategy  plus-inj-1
call activate-lemma  plus-inj-1

assume
    { plus(z,x) =/= plus(z,y),
      x = y }
    plus-inj-2
call auto-strategy  plus-inj-2
call activate-lemma  plus-inj-2

assume
    { leq(plus(u,v),y) = false,
      leq(v,y) = true }
    less-plus-mono-1
call auto-strategy  less-plus-mono-1
call activate-lemma  less-plus-mono-1

assume
    { leq(plus(y,x),z) = false,
      leq(y,z) =/= false }
    less-plus-mono-2
call auto-strategy  less-plus-mono-2
call activate-lemma  less-plus-mono-2
```

```
assume
    { leq(v,0) = false,
      v = 0 }
    leq-v-0
call auto-strategy  leq-v-0
call activate-lemma  leq-v-0

assume
    { s(minus(x,s(0))) = x,
      x = 0 }
    s-minus
call auto-strategy  s-minus
call activate-lemma  s-minus

assume
    { leq(plus(x,u),s(plus(y,v))) = false,
      leq(x,y) = true,
      leq(u,v) = true }
    less-plus-mono
call auto-strategy  less-plus-mono
call activate-lemma  less-plus-mono

assume
    { leq(times(u,w),plus(times(v,w),minus(w,s(0)))) = false,
      leq(u,v) = true,
      w = 0 }
    less-times-mono
call auto-strategy  less-times-mono
apply lemma-subs
    less-plus-mono
    [ x <-- u, y <-- v, u <-- times(u,w),
      v <-- plus(minus(w,s(0)),times(v,w)) ] ..
    less-times-mono_1
call simplify  less-times-mono_1
apply axiom-rewrite
    1
    [1:2]
    plus-2
    1
    [x <-- v , y <-- plus(minus(w,s(0)),times(v,w))] ..
    less-times-mono_1
apply lemma-rewrite
    1
    [1:2:2]
    plus-sx-y
    1
```

```
    [x <-- minus(w,s(0)) , y <-- times(v,w)] ..
    less-times-mono_1
call simplify  less-times-mono_1
apply lemma-rewrite
    2
    [1:2:2:1]
    s-minus
    1
    [x <-- w] ..
    less-times-mono_1
set weight w less-times-mono
call cont-proof-attempt  less-times-mono_1
call activate-lemma  less-times-mono

assume
    { leq(plus(x,u),plus(y,v)) = false,
      leq(x,y) = true,
      leq(u,v) = true }
    less-plus-mono-3
call auto-strategy  less-plus-mono-3
call activate-lemma  less-plus-mono-3

assume
    { leq(times(u,w),times(v,w)) = false,
      leq(u,v) = true,
      w = 0 }
    less-times-mono-1
call auto-strategy  less-times-mono-1
call activate-lemma  less-times-mono-1

assume
    { leq(times(w,u),times(w,v)) = false,
      leq(u,v) = true,
      w = 0 }
    less-times-mono-2
call auto-strategy  less-times-mono-2
call activate-lemma  less-times-mono-2

assume
    { x =/= y,
      leq(x,y) =/= false }
    uneq-less
call auto-strategy  uneq-less
call activate-lemma  uneq-less

assume
    { times(x,z) =/= times(y,z),
```

```
      x = y,
      z = 0 }
    times-inj-1
apply lemma-subs
    leq-trichotomy
    [x <-- x , y <-- y] ..
    times-inj-1
apply lemma-subs
    less-times-mono
    [u <-- x, v <-- y, w <-- z] ..
    times-inj-1
call simplify  times-inj-1
apply lemma-subs
    less-times-mono
    [u <-- y, v <-- x, w <-- z] ..
    times-inj-1
call simplify  times-inj-1
call activate-lemma  times-inj-1
call deactivate-lemmas  { uneq-less }

assume
    { plus(x,y) =/= y,
      x = 0 }
    uneq-plus-x-y-x
call auto-strategy  uneq-plus-x-y-x
call activate-lemma  uneq-plus-x-y-x

assume
    { x = 0,
      leq(s(0),x) = true }
    leq-1-x
call auto-strategy  leq-1-x
call activate-lemma  leq-1-x

assume
    { minus(s(x),y) = s(minus(x,y)),
      leq(y,x) =/= true }
    minus-s-leq
call auto-strategy  minus-s-leq
call activate-lemma  minus-s-leq

assume
    { minus(plus(x,z),y) = plus(minus(x,y),z),
      leq(y,x) =/= true }
    minus-plus-leq
call auto-strategy  minus-plus-leq
call activate-lemma  minus-plus-leq
```

```
assume
    { minus(plus(z,x),y) = plus(minus(x,y),z),
      leq(y,x) =/= true }
    minus-plus-leq-1
call auto-strategy  minus-plus-leq-1
call activate-lemma  minus-plus-leq-1



assume { def plus(x,y) } L1
call simplify L1

assume { def times(x,y) } L2
call simplify L2

assume { times(s(0),x) = x } L3
call auto-strategy L3

assume { plus(x,y) = plus(y,x) } L4
call auto-strategy L4

assume { plus(plus(x,y),z) = plus(x,plus(y,z)) } L5
call auto-strategy L5

assume { plus(times(x,z),times(y,z)) = times(plus(x,y),z) } L6
call auto-strategy L6

assume { times(plus(x,s(0)),y) = plus(times(x,y),y) } L7
call auto-strategy L7

assume { s(x) = plus(x,s(0)) } L8
call auto-strategy L8

assume { minus(times(x,y),times(x,z)) = times(x,minus(y,z)) } L9
call auto-strategy L9
call activate-lemma L9

assume { minus(plus(x,y),plus(x,z)) = minus(y,z) } L10
call auto-strategy L10

assume { plus(x,y) =/= plus(w,z),
         plus(x,minus(y,z)) = plus(w,minus(z,y)) } L11a
call auto-strategy L11a

assume { plus(x,y) = plus(w,z),
         plus(x,minus(y,z)) =/= plus(w,minus(z,y)) } L11b
```

```
call auto-strategy L11b

assume { plus(x,times(y,u)) =/= plus(w,times(z,u)),
         plus(x,times(minus(y,z),u)) = plus(w,times(minus(z,y),u)) }
       L12a
call auto-strategy L12a

assume { plus(x,times(y,u)) = plus(w,times(z,u)),
         plus(x,times(minus(y,z),u)) =/= plus(w,times(minus(z,y),u)) }
       L12b
call auto-strategy L12b

assume { leq(times(x,y),times(x,z)) = leq(y,z),
         x = 0 } L13
call auto-strategy L13

assume { leq(times(u,v),plus(times(u,w),y)) = leq(v,w),
         leq(u,y) = true } L14
call auto-strategy L14
call activate-lemma  L14

assume { leq(plus(times(u,x),v),times(u,y)) = leq(s(x),y),
         leq(u,v) = true,
         v = 0 } L15
call auto-strategy  L15

assume { times(x,y) =/= plus(times(x,z),u),
         u = 0,
         leq(x,u) = true } L16
apply lemma-subs
    leq-trichotomy
    [ x <-- z ] ..
    L16
apply lemma-subs
    uneq-less
    [y <-- times(x,y) , x <-- plus(times(x,z),u)] ..
    L16
set current g-node L16 [1:2]
apply lemma-subs
    uneq-less
    [x <-- times(x,y) , y <-- plus(times(x,z),u)] ..
    L16
call cont-proof-attempt L16

assume
    { times(z,x) =/= times(z,y),
      x = y,
```

```
      z = 0 } L17a
call auto-strategy L17a

assume { times(x,y) = times(x,z),
         y =/= z,
         x = 0 } L17b
call auto-strategy L17b

assume { leq(x,y) = false,
         leq(s(y),x) = false } L18a
call auto-strategy L18a

assume { leq(x,y) = true,
         leq(s(y),x) = true } L18b
call auto-strategy L18b

assume { ~(x < y),
         leq(s(x),y) = true } L19a

assume { x < y,
         leq(s(x),y) = false } L19b
call auto-strategy L19b

assume { x < y,
         leq(y,x) = true } L20a
call auto-strategy L20a

assume { ~(x < y),
         leq(y,x) = false } L20b

assume { minus(x,y) = 0,
         leq(y,x) = true } L21
call auto-strategy L21

assume { plus(x,minus(y,z)) = u,
         plus(x,y) =/= plus(u,z),
         leq(z,y) =/= true } L22a
call auto-strategy L22a

assume { plus(x,minus(y,z)) =/= u,
         plus(x,y) = plus(u,z),
         leq(z,y) =/= true } L22b
call simplify  L22b

assume { leq(plus(x,minus(y,z)),u) = true,
         leq(plus(x,y),plus(u,z)) = false,
         leq(z,y) =/= true } L23a
```

```
call auto-strategy  L23a

assume { leq(plus(x,minus(y,z)),u) = false,
         leq(plus(x,y),plus(u,z)) = true,
         leq(z,y) =/= true } L23b
call auto-strategy L23b

assume { x = y,
         leq(x,y) =/= true,
         leq(y,x) =/= true } L24
call auto-strategy L24

assume { leq(x,y) = true,
         x =/= y } L25
call auto-strategy L25

assume { leq(x,y) = true,
         leq(y,x) = true } L26
call auto-strategy L26

assume { 0 =/= plus(s(x),y) } L27
call auto-strategy L27

assume { leq(plus(s(x),y),0) = false } L28
call auto-strategy L28

assume { def minus(x,y) } L29
call auto-strategy L29

assume { leq(plus(times(z,x),times(w,u)),
          plus(times(z,y),plus(times(w,v),
          minus(plus(z,w),s(0))))) = false,
         leq(x,y) = true,
         leq(u,v) = true,
         z = 0,
         w = 0 } L30
apply lit-add
    leq(plus(times(z,x),times(w,u)),
     s(plus(plus(times(z,y),minus(z,s(0))),
     plus(times(w,v),minus(w,s(0)))))) =/= false
    . ..
    L30
apply lemma-rewrite
    1
    [1]
    less-plus-mono
    1
```

```
    [x <-- times(z,x) , u <-- times(w,u) ,
     y <-- plus(times(z,y),minus(z,s(0))) ,
     v <-- plus(times(w,v),minus(w,s(0)))] ..
    L30
call cont-proof-attempt L30
apply lemma-rewrite
    11
    [1:2]
    plus-sx-y
    1
    [x <-- minus(w,s(0)) ,
     y <-- plus(minus(z,s(0)), plus(times(v,w),times(y,z)))] ..
    L30
call simplify  L30
apply lemma-rewrite
    12
    [1:2:1]
    s-minus
    1
    [x <-- w] ..
    L30
call simplify  L30

assume
    { leq(x,y) = true,
      x = plus(y,s(0)),
      leq(x,plus(y,s(0))) =/= true }
    L31
call auto-strategy  L31

assume { minus(plus(x,y),y) = x } L32
call auto-strategy L32

assume { leq(plus(x,y),plus(x,z)) = leq(y,z) } L33
call auto-strategy L33

assume { leq(minus(x,v),plus(y,w)) = true,
         leq(x,y) =/= true } L34
call auto-strategy L34

assume { leq(y,z) = true,
         plus(x,y) =/= z } L35
call auto-strategy L35

assume { plus(x,z) = y,
         x =/= minus(y,z),
         leq(z,y) =/= true } L36a
```

```
call auto-strategy L36a

assume { plus(x,z) =/= y,
         x = minus(y,z),
         leq(z,y) =/= true } L36b
call auto-strategy L36b

assume { times(minus(x,y),z) = minus(times(x,z),times(y,z)) }
       L37
apply lemma-rewrite
         1
         [1]
         times-commutative
         1
         [x <-- minus(x,y) , y <-- z] ..
         L37
set current g-node L37 [1:2]
apply lemma-rewrite
         2
         [2:1]
         times-commutative
         1
         [x <-- x , y <-- z] ..
         L37
 apply lemma-rewrite
         2
         [2:2]
         times-commutative
         1
         [x <-- y , y <-- z] ..
         L37
set current g-node L37 root
call simplify-open-subgoals  L37

assume { plus(minus(x,y),y) = x,
         leq(y,x) =/= true } L38
call auto-strategy L38

assume { x =/= y,
         times(z,x)=times(z,y) } L39
call auto-strategy L39

assume { plus(x,u) = plus(y,v),
         x =/= y,
         u =/= v } L40
call auto-strategy L40
```

```
assume { plus(u,x) =/= plus(v,y),
         u = v,
         x =/= y } L41
call auto-strategy L41

assume { leq(plus(u,x),plus(v,y)) = true,
         leq(u,v) = false,
         x =/= y } L42
call auto-strategy L42

assume { u = v,
         plus(u,x) =/= plus(v,y),
         x =/= y } L43
call auto-strategy L43

assume { u =/= v,
         plus(u,x) = plus(v,y),
         x =/= y } L44
call auto-strategy L44

assume { leq(u,v) = true,
         leq(plus(u,x),plus(v,y)) = false,
         x =/= y } L45
call auto-strategy L45

assume { leq(plus(x,z),plus(y,w)) = true,
         leq(plus(x,u),plus(y,v)) = false,
         leq(plus(z,v),plus(w,u)) = false } L46
apply lit-add
    leq(plus(plus(x,u),plus(z,v)),
        plus(plus(y,v),plus(w,u))) =/=  true
    . ..
    L46
apply lemma-subs
    leq-plus-mono
    [u <-- plus(x,u) , v <-- plus(z,v) ,
     x <-- plus(y,v) , y <-- plus(w,u)] ..
    L46
call cont-proof-attempt  L46
```

B. Beispielspezifikationen

In den folgenden drei Abschnitten sind die QuodLibet-Skripte für die Spezifikationen, die in Kapitel 4 verwendet werden, aufgeführt. Diese enthalten nur die Definition der benötigten Operatoren und die eigentlichen Beweisziele, keine Lemmata und keine Beweise. Die Operatoren der linearen Arithmetik werden nicht spezifiziert, da diese im neuen System bereits enthalten sind und für das alte System ohne lineare Arithmetik entsprechend der Beschreibung in Abschnitt 2.2 leicht zur Spezifikation hinzugefügt werden können.

B.1. Das Beispiel ggT

```
declare constructor variables x, y, z : Nat.

declare operators
    gcd : Nat Nat --> Nat
    .
assert gcd-1 :
    gcd(x,y) = x
    if y = 0
    .
assert gcd-2 :
    gcd(x,y) = y
    if x = 0,
       y =/= 0
    .
assert gcd-3 :
    gcd(x,y) = gcd(x,-(y,x))
    if
      x <= y,
      x =/= 0,
      y =/= 0
    .
assert gcd-4 :
    gcd(x,y) = gcd(-(x,y),y)
    if
      ~(x <= y),
      x =/= 0,
      y =/= 0
    .
```

```
assume
    { gcd(x,x) = x }
    gcd-idempotent

assume
    { gcd(x,y) = gcd(y,x) }
    gcd-commutative

assume
    { gcd(gcd(x,y),z) = gcd(x,gcd(y,z)),
      x = 0,
      y = 0,
      z = 0 }
    gcd-associative
```

B.2. Das Beispiel $\sqrt{2}$

```
declare constructor variables x,y : Nat.

declare operators
    sqr    :     Nat --> Nat
    .
assert sqr1 :
    sqr(x)=*(x,x)
    .

assume
    { *(2,sqr(y))=/=sqr(x), y=0 }
    sqrtirrat1
```

B.3. Das Beispiel 91er-Funktion

```
declare constructor variables x : Nat.

declare operators
    f91 : Nat --> Nat
    .
assert f91-1 :
    f91(x)=f91(f91(+(x,11)))
    if x <= 100
    .
```

```
assert f91-2 :
    f91(x) = -(x,10)
    if ~(x <= 100)
    .

assume
    { def f91(x) }
    def-f91
```

Literaturverzeichnis

[Ave95] Avenhaus, Jürgen: *Reduktionssysteme.* Springer, Berlin Heidelberg, 1995.

[BM88] Boyer, Robert S. und J. Strother Moore: *Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic.* Machine Intelligence, 11:83 – 124, 1988.

[DE73] Dantzig, George B. und B. Curtis Eaves: *Fourier-Motzkin elimination and its dual.* Journal of Combinatorial Theory, 14(3):288 – 297, 1973.

[FR74] Fischer, Michael J. und Michael O. Rabin: *Super-Exponential Complexity of Presburger Arithmetic.* In: *Proceedings of the SIAM-AMS Symposium in Applied Mathematics*, Band 7, Seiten 27 – 41, 1974.

[Göd31] Gödel, Kurt: *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.* Monatsheft für Mathematik und Physik, 38:173 – 198, 1931.

[JBG99] Janičić, Predrag, Alan Bundy und Ian Green: *A Framework for the Flexible Integration of a Class of Decision Procedures into Theorem Provers.* Informatics Research Report EDI-INF-RR-0096, University of Edinburgh, Division of Informatics, 1999.

[Kai02] Kaiser, Markus: *Effizientes Beweisen mit einem formalen Beweissystem.* Diplomarbeit, Fachbereich Mathematik, Universität Kaiserslautern, 2002.

[Küh00] Kühler, Ulrich: *A Tactic-Based Inductive Theorem Prover for Data Types with Partial Operations.* Infix, Sankt Augustin, 2000. Dissertation, Fachbereich Informatik, Universität Kaiserslautern.

[KN94] Kapur, Depak und Xumin Nie: *Reasoning about Numbers in Tecton.* In: *Proceedings of 8th International Symposium on Methodologies for Intelligent Systems (ISMIS'94)*, Seiten 57–70, Charlotte, North Carolina, USA, Oktober 1994.

[Knu81] Knuth, Donald E.: *The art of computer programming: Seminumerical algorithms*, Band 2, Seiten 326 – 328. Addison-Wesley, 2. Auflage, 1981.

[MM70] Manna, Zohar und John McCarthy: *Properties of programs and partial function logic.* Machine Intelligence, 5:27 – 37, 1970.

[Pre29] Presburger, Mojzesz: *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt.* Comptes Rendus du I congrès de Mathématiciens des Pays Slaves, Seiten 92 – 101, 1929.

[Pug92] Pugh, William: *The omega test: a fast and practical integer programming algorithm for dependence analysis.* Communications of the ACM, 35(8):102 – 114, 1992.

[Sch04] Schmidt-Samoa, Tobias: *The New Standard Tactics of the Inductive Theorem Prover* QuodLibet. SEKI-Report SR-2004-01, Fachbereich Informatik, TU Kaiserslautern, 2004.

[Wir04] Wirth, Claus-Peter: *Descente Infinie + Deduction.* Logic Journal of the IGPL, 12(1):1–96, 2004.