Zur Korrektheit und Sicherheit von Inferenzregeln (3)

Beispiel

Betrachte die Regel

$$\frac{\langle \Gamma_1, \lambda'[t_1]_p, \Gamma_2; \omega\rangle}{\langle \Gamma_1, \lambda'[t_2]_p, \Gamma_2; \omega\rangle} \quad \text{falls } t_1 \neq t_2 \text{ in } \Gamma_1 \cup \Gamma_2$$

Sei $\mathcal{A} \in$ DMod(spec) und $\Gamma = \Gamma_1 \cup \{\lambda'[t_1]_p\} \cup \Gamma_2$ und $\Gamma' = \Gamma_1 \cup \{\lambda'[t_2]_p\} \cup \Gamma_2$ und τ eine induktive Substitution. Gilt $\mathcal{A} \models \tau(\lambda)$ für ein $\lambda \in \Gamma_1 \cup \Gamma_2$, so gilt $\mathcal{A} \models \tau(\Gamma)$ und $\mathcal{A} \models \tau(\Gamma')$.

Sei $\mathcal{A} \not\models \tau(\lambda)$ für alle $\lambda \in \Gamma_1 \cup \Gamma_2$. Dann gilt $\mathcal{A} \models \tau(t_1) = \tau(t_2)$, also gilt $\mathcal{A} \models \tau(\lambda'[t_1]_p)$ gdw $\mathcal{A} \models \tau(\lambda'[t_2]_p)$.

Dies zeigt $\mathcal{A} \models \Gamma$ gdw $\mathcal{A} \models \Gamma'$.

Also ist die Regel korrekt und sicher.

Eigenschaften der QUODLIBET-Regeln

Satz 4.5.

Die Inferenzregeln von QUODLIBET sind korrekt und sicher.

Beweis:

Übung. Man beachte dazu die Vorüberlegungen.

Bemerkungen zu den Inferenzregeln

1. Bemerkung

Die Anwendung der Inferenzregeln ist hochgradig nicht-deterministisch.

(1) Es sind in der Regel mehrere Inferenzregeln auf ein Ziel $\langle \Gamma; \omega \rangle$ anwendbar.

(2) Es ist i. Allg. eine feste Inferenzregel auf mehrere $\lambda \in \Gamma$ anwendbar, manchmal sogar mit unterschiedlichen Lemmata bzw. Induktions-Hypothesen.

Zu (2)

Die Inferenzregeln sind so geschrieben, dass es sinnvoll ist, eine Regel jeweils auf das erste λ aus (der Liste) Γ anzuwenden.

Zu (1)

In QUODLIBET sind Folgen von Inferenzregeln zu **Taktiken** zusammengefasst.

4.5 Die Beweisführung

Sei

sig $= (S, C, F, \alpha)$ eine Signatur,
spec $=$ (sig, R) eine zulässige Spezifikation,
$\mathcal{L}$ eine Menge von (gültigen) Lemmata, $R \subseteq \mathcal{L}$,
Γ eine Klausel.

Zu zeigen sei DMod(spec) $\models \Gamma$.

Bemerkung

Ein Beweisversuch zu Γ wird als Baum organisiert.

Ein Beweisbaum hat als Knoten Ziel- und Inferenzknoten.

Ein Zielknoten trägt als Marke das Ziel (eine Klausel und ein Gewicht). Er wird durch Anwendung einer Inferenzregel abgearbeitet. Dies erzeugt einen Inferenzknoten mit dem Namen der anzuwendenden Inferenz.

Ein Inferenzknoten trägt als Marke den Namen einer Inferenzregel. Wird die Anwendung der Inferenz auf das Ziel abgearbeitet, so entstehen n neue Ziele mit $n \geq 0$.

Der Beweis ist abgeschlossen, wenn alle Ziel- und Inferenzknoten abgearbeitet sind.

Inferenzfolgen können zu Strategien zusammengefasst werden.

Beweisbäume

Bezeichnung

Ein **Beweisbaum** ist ein Und-Oder-Baum.

Zielknoten sind Oder-Knoten. **Zielknoten** tragen ein Teilziel $\langle \Gamma; \omega \rangle$. Söhne von Zielknoten sind Inferenzknoten.

Ein **Inferenzknoten** ist ein Und-Knoten. Er trägt den Namen der angewendeten Inferenz und Informationen darüber, welche Klausel $\langle \Pi; \hat{\omega} \rangle$ wie auf seinen Vaterknoten $\langle \Gamma; \omega \rangle$ angewendet wird. Söhne des Inferenzknotens sind Zielknoten zu den neu entstandenen Teilzielen $\langle \Gamma_i; \omega_i \rangle$.

Beachte

Die Wahl, welche Inferenz auf $\langle \Gamma; \omega \rangle$ angewendet wird, ist frei. Daher sind Zielknoten Oder-Knoten.

Ein Inferenzknoten erzeugt neue Zielknoten, die alle abgearbeitet werden müssen, er ist also ein Und-Knoten.

Ein Beweisbaum ist **abgeschlossen**, wenn jedes Blatt ein Inferenzknoten ist. (Dann ist sein Vater abgearbeitet, ohne dass ein neues Teilziel erzeugt wurde.)

Ein abgeschlossener Beweisbaum mit Wurzel $\langle \Gamma; \omega \rangle$ stellt einen Beweis für die Gültigkeit von Γ dar.

Behandlung von Gewichten

Start eines Beweisversuchs zu Γ

Es wird ein Beweisbaum mit einzigem Knoten $\langle \Gamma; \omega_0 \rangle$ erzeugt. Dabei ist ω_0 zunächst nur ein Name $W_\Gamma(x_1, \ldots, x_n)$ mit $\{x_1, \ldots, x_n\} = \mathsf{Var}(\Gamma)$.

Instanziierung von Gewichten

Nur wenige Inferenzregeln, z. B. Fallunterscheidung nach Substitutionen, verändern das Gewicht: $\langle \Gamma; \omega \rangle$ hat Teilziele der Form $\langle \Gamma'; \sigma(\omega) \rangle$. Die anderen Inferenzregeln erzeugen Teilziele der Form $\langle \Gamma'; \omega \rangle$.

Solange das Gewicht noch ein Name der Form $W_\Gamma(t_1, \ldots, t_n)$ ist, wird der Name durch σ instanziiert.

Setzen von Gewichten

Irgendwann muss ein Name $W_\Gamma(t_1, \ldots, t_n)$ zu $(t'_1, \ldots, t'_k) \in \mathsf{Term}(F, V)^k$ „geeignet“ konkretisiert werden. Dies macht der Benutzer direkt oder durch den Aufruf einer Taktik.

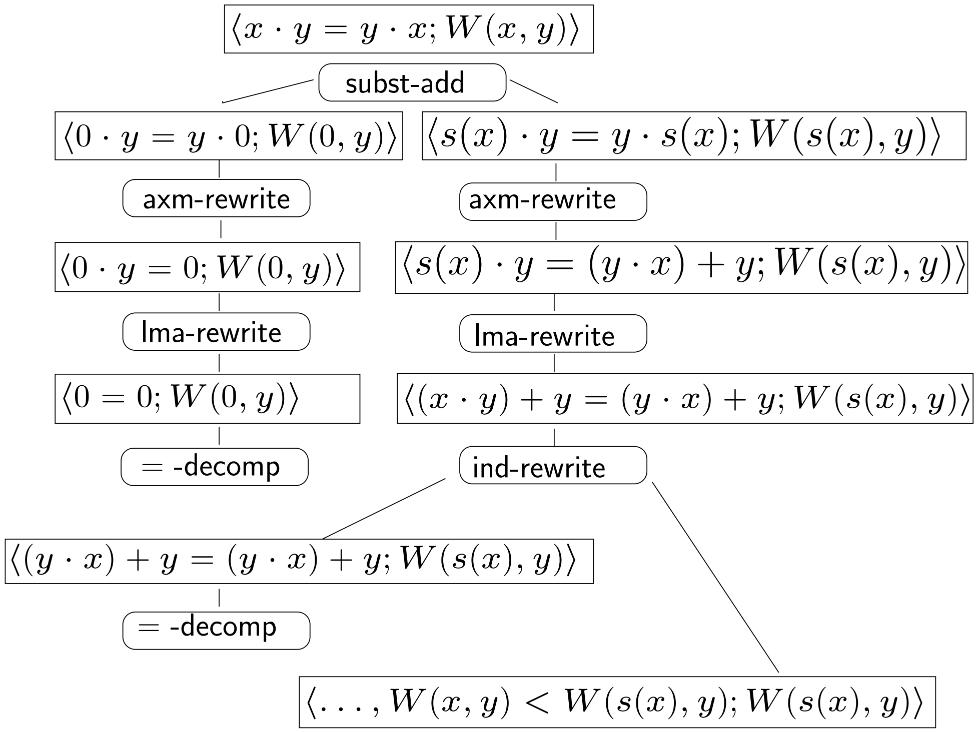
Es entstehen konkrete Beweisaufgaben, die mit den Inferenzregeln abgearbeitet werden müssen.

Beachte

Dies realisiert die Idee, dass die Induktionsordnung erst sehr spät (“bei Bedarf‘) festgelegt wird, und zwar durch das Setzen des Gewichts.

Beispiel: Multiplikation (1)

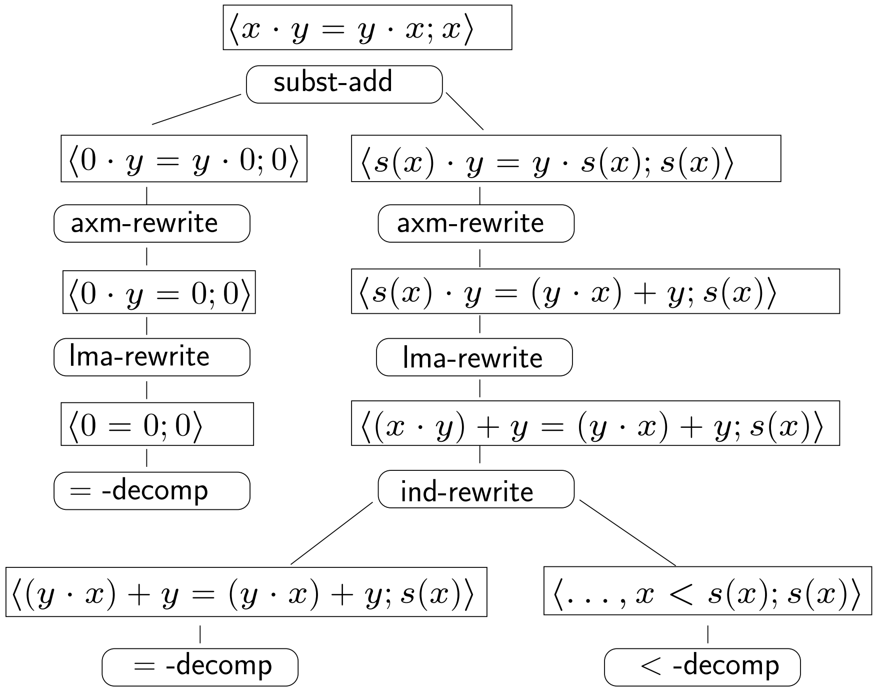
Axiome	$x + 0 = x$	$x + s(y) = s(x + y)$
	$x \cdot 0 = 0$	$x \cdot s(y) = (x \cdot y) + x$
Lemmata	$0 \cdot y = 0$	$s(x) \cdot y = (x \cdot y) + y$
Ziel	$x \cdot y = y \cdot x$	



Beispiel: Multiplikation (2)

Setzen des Gewichtes

$$W(x, y) :\equiv x$$



Proof Engineering

Beobachtung

Die Beweisführung ist nur schwer automatisierbar. Man braucht Kriterien

a) für die Auswahl der „richtigen" Inferenzregel,
b) für die Definition der Gewichte (dies definiert die Ind.-Ordnung),
c) für die „richtigen" Lemmata.

Dies bezeichnet man mit **Proof Engineering**.

Ansatz Taktiken entwickeln.

Taktiken zu b)

Dies lässt sich häufig aus der Beweisverpflichtung $W(t_1, \ldots, t_n) < W(t'_1, \ldots, t'_n)$ raten: Als Taktik formulieren.

Taktiken zu c)

Man errate aus dem aktuellen Teilziel $\langle \Gamma; \omega \rangle$ eine Hilfsbehauptung, führe sie mit dem assume-Befehl ein, führe damit den Beweis weiter und beweise sie bei Erfolg.

Vorsicht: (1) Die Hilfsbehauptung kann falsch sein. (2) Dies kann zu unendlichem Prozess führen.

Taktiken zu a)

(1) Inferenzfolgen zusammenfassen

- für die Reinigung von Klauseln,
- für die Termersetzung und Subsumption,
- für die Behandlung von def- und $<$-Literalen.

(2) Analyse der definierten Funktionssymbole f

- dies liefert Hinweise für die Fallunterscheidung nach Substitutionen (Expansion von f),
- dies liefert außerdem das Lemma def $f(x_1, \ldots, x_n)$ und hilft bei dem Beweis von def-Literalen.

(3) Auswahl des „richtigen" Literals in $\langle \Gamma; \omega \rangle$: Beachte, dass es reicht, **ein** Literal aus Γ zu beweisen

- wähle stets das erste Literal aus Γ, für das eine Inferenzregel anwendbar ist. Die Inferenzregeln sind so entworfen, dass das sinnvoll ist.

Beispiel Multiplikation

Sei

Gegeben: $R: x + 0 = x \qquad x + s(y) = s(x + y)$
$\qquad\qquad x \cdot 0 = 0 \qquad x \cdot s(y) = (x \cdot y) + x$

Ziel: $x \cdot y = y \cdot x$

Man benötigt folgende Hilfsbehauptungen

(1) $0 \cdot y = 0$ (2) $s(x) \cdot y = (x \cdot y) + y$

Es ist (1) leicht, für (2) benötigt man

(3) $((x \cdot y) + y) + x = ((x \cdot y) + x) + y$

Für (3) rate man die Hilfsbehauptung

(4) $(x + y) + z = x + (y + z)$ (5) $x + y = y + x$

oder

(4') $(z + y) + x = (z + x) + y$ (Generalisierung von (3))

Es gilt

Alle Hilfsbehauptungen lassen sich beweisen.

Damit lässt sich das Ziel beweisen.

Also Hilfsbehauptungen lassen sich häufig beim Beweisen erraten.
$\Rightarrow$ Taktiken dafür entwickeln.

Beispiel Mergesort: Die Spezifikation

merge: Mischen

merge(nil, l_2) = l_2 merge(l_1, nil) = l_1
merge($x.l_1, y.l_2$) = x.merge($l_1, y.l_2$) $\Leftarrow$ leq(x, y) = true
merge($x.l_1, y.l_2$) = y.merge($x.l_1, l_2$) $\Leftarrow$ leq(x, y) $\neq$ true,
def leq(x, y)

split1, **split2**: Aufspalten

split1(nil) = nil split2(nil) = nil
split1(x.nil) = x.nil split2(x.nil) = nil
split1($x.y.l$) = x.split1(l) split2($x.y.l$) = y.split2(l)

msort: Sortieren durch Mischen

msort(nil) = nil
msort(x.nil) = x.nil
msort($x.y.l$) = merge(msort(split1($x.y.l$)), msort(split2($x.y.l$)))

sorted: Sortiertheit

sorted(nil) = true
sorted($x.l$) = sorted(l) $\Leftarrow$ leq-list(x, l) = true
sorted($x.l$) = false $\Leftarrow$ leq-list(x, l) $\neq$ true, def leq-list(x, l)

Ziel

sorted(msort(l)) = true.

Mergesort: Die Voraussetzungen

Voraussetzungen

a) Die Sorten BOOL, NAT und LIST mit den üblichen Operationen sind spezifiziert, z. B.
leq(x, y) = true gdw $x \leq y$
leq-list(x, l) = true gdw $x \leq y$ für alle y in der Liste l

b) Es sind hinreichend viele Lemmata bewiesen, z. B.
L-leq-trans$\{$leq(x, y) = true $\wedge$ leq(y, z) = true
$\Rightarrow$ leq(x, z) = true$\}$
L-leq-list-def$\{$def leq-list$(x, l)\}$

c) Alle spezifizierten Operatoren sind analysiert, also sind sinnvolle Fallunterscheidungen nach Substitutionen für ihre Expansion bekannt.

d) Die Totalität aller Operatoren ist bewiesen, z. B.
L-def-merge$\{$def merge$(l_1, l_2)\}$

Beachte

Diese Voraussetzungen sind hier didaktisch für die Vorlesung eingeführt.

In der Praxis muss im Beweisversuch sichergestellt werden, dass sie erfüllt sind.

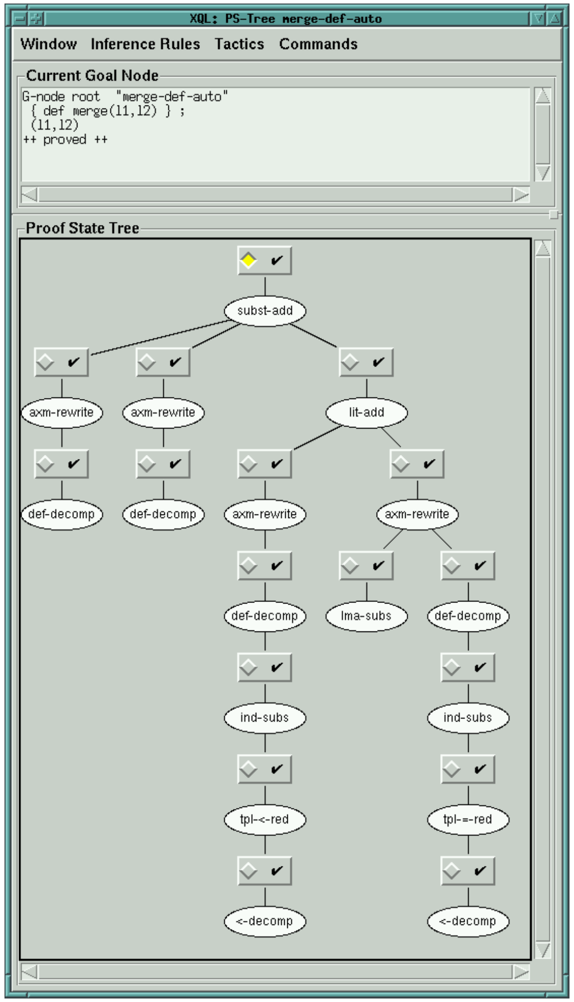
Beweis zu L-def-merge

1 $\{$def merge$(l_1, l_2); W(l_1, l_2)\}$ subst-add
1.1 $\{$def merge(nil, l_2); W(nil, l_2)$\}$ axm-rewrite
$\{$def l_2; W(nil, l_2)$\}$ $\surd$ mit def-decomp
1.2 $\{$def merge(l_1, nil); W(l_1, nil)$\}$ $\surd$ analog zu 1.1
1.3 $\{$def merge$(x.l_1, y.l_2); W(x.l_1, y.l_2)\}$ lit-add
1.3.1 $\{$leq$(x, y) \neq$ true, def merge$(x.l_1, y.l_2); W(x.l_1, y.l_2)\}$
axm-rewrite
$\{$def x.merge$(l_1, y.l_2), \ldots; W(x.l_1, y.l_2)\}$ def-decomp
$\{$def merge$(l_1, y.l_2), \ldots; W(x.l_1, y.l_2)\}$ ind-subs
$\{W(l_1, y.l_2) < W(x.l_1, y.l_2), \ldots; W(x.l_1, y.l_2)\}$ offen
1.3.2 $\{$leq(x, y) = true, def merge$(x.l_1, y.l_2); W(x.l_1, y.l_2)\}$
axm-rewrite
1.3.2.1 $\{$def leq$(x, y), \ldots; W(x.l_1, y.l_2)\}$ $\surd$ lma-subs
1.3.2.2 $\{$def y.merge$(x.l_1, l_2), \ldots; W(x.l_1, y.l_2)\}$ def-decomp
$\{$def merge$(x.l_1, l_2), \ldots; W(x.l_1, y.l_2)\}$ ind-subs
$\{W(x.l_1, l_2) < W(x.l_1, y.l_2), \ldots; W(x.l_1, l_2)\}$ offen

Setzen des Gewichts $W(l_1, l_2) :\equiv (l_1, l_2)$ liefert
1.3.1 $\{(l_1, y.l_2) < (x.l_1, y.l_2), \ldots; (x.l_1, y.l_2)\}$ $\surd$
1.3.2.2 $\{(x.l_1, l_2) < (x.l_1, y.l_2), \ldots; (x.l_1, y.l_2)\}$ $\surd$

Schreibweise hier
Ein Knoten ist beschrieben durch seine Position im Beweisbaum, sein Teilziel $\langle \Gamma; \omega \rangle$ und die auf ihn angewandte Inferenzregel. Knoten ohne Position sind einzige Söhne ihres Vaters. Knoten mit Markierung $\surd$ sind abgearbeitet ohne ein Teilziel zu erzeugen oder leicht zu zeigen.

Beweiszustandsbaum zu L-def-merge



Proof Engineering

Zentrale Beweisidee zum Beweis von $\{$sorted(msort(l)) $=$ true$\}$

a) Hilfsbehauptung erzeugen:
L-sorted-merge$\{$sorted(l_1) $=$ true $\wedge$ sorted(l_2) $=$ true
$\Rightarrow$ sorted(merge(l_1, l_2) $=$ true$\}$

b) Der Beweis hierzu benötigt weitere Hilfsbehauptung
L-leq-merge$\{$leq-list(x, l_1) $=$ true $\wedge$ leq-list(x, l_2) $=$ true
$\Rightarrow$ leq-list(x, merge(l_1, l_2)) $=$ true$\}$

Induktionsordnung festlegen

Idee aus der Spezifikation von msort:
split1($x.y.l$) $<$ $x.y.l$ und split2($x.l$) $<$ $x.l$

Also Hilfsbehauptungen
L-split1-ind$\{l \neq$ nil $\wedge$ le(l) $\neq s(0) \Rightarrow$ split1(l) $< l\}$
L-split2-ind$\{l \neq$ nil $\Rightarrow$ split2(l) $< l\}$

Also le(l) (=Länge von l) spezifizieren.

Beachte

$<$ ist ein vordefiniertes Prädikatssymbol (nämlich die semantische Ordnung auf Gewichten).

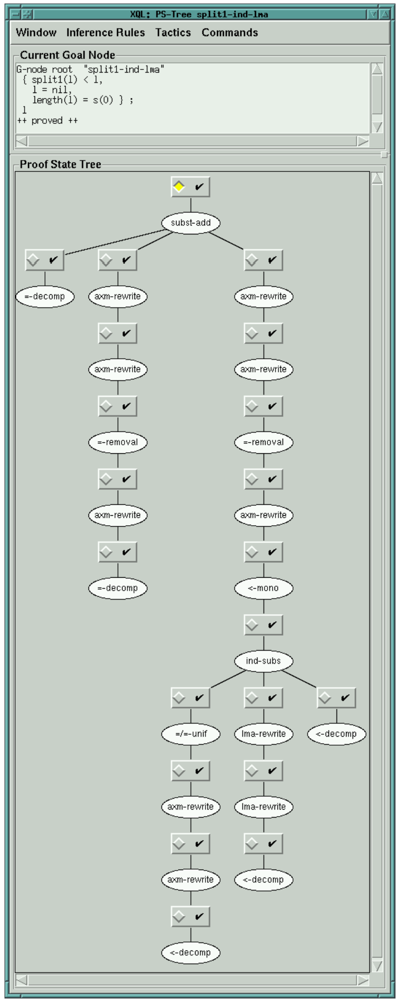
Analog sind $=$ und def vordefinierte Prädikatssymbole.

Beweis zu L-split1-ind

Hilfslemma: L-le1-split1$\{\text{le}(l) = s(0) \Rightarrow \text{split1}(l) = l\}$

1	$\{l = \text{nil}, \text{le}(l) = s(0), \text{split1}(l) < l; W(l)\}$	subst-add
1.1	$\{\text{nil} = \text{nil}, \ldots; W(\text{nil})\}$ $\surd$	mit $=$-decomp
1.2	$\{x.\text{nil} = \text{nil}, \text{le}(x.\text{nil}) = s(0), \ldots; W(x.\text{nil})\}$	axm-rewrite
	$\{\ldots, s(0) = s(0), \ldots; W(x.\text{nil})\}$ $\surd$	mit $=$-decomp
1.3	$\{\text{split1}(x.y.l) < x.y.l, \ldots; W(x.y.l)\}$	axm-rewrite
	$\{x.\text{split1}(l) < x.y.l, \ldots; W(x.y.l)\}$	$<$-mono
	$\{\text{split1}(l) < l, x.\text{split1}(l) < x.y.l, \ldots; W(x.y.l)\}$	ind-subs
1.3.1	$\{l \neq \text{nil}, \ldots, x.\text{split1}(l) < x.y.l, \ldots; W(x.y.l)\}$	$\neq$-unif
	$\{x.\text{split1}(\text{nil}) < x.y.\text{nil}, \ldots; W(x.y.l)\}$	axm-rewrite
	$\{x.\text{nil} < x.y.\text{nil}, \ldots; W(x.y.l)\}$ $\surd$	mit $<$-decomp
1.3.2	$\{\text{le}(l) \neq s(0), x.\text{split1}(l) < x.y.l, \ldots; W(x.y.l)\}$	lma-rewrite
	$\{\text{le}(l) \neq s(0); x.l < x.y.l, \ldots; W(x.y.l)\}$ $\surd$	$<$-decomp
1.3.3	$\{W(l) < W(x.y.l), \ldots; W(x.y.l)\}$	Gewicht setzen
	$\{l < x.y.l, \ldots; x.y.l\}$ $\surd$	mit $<$-decomp

Beweiszustandsbaum zu L-split1-ind



Beweis des Ziels (1)

1 $\{\text{sorted}(\text{msort}(l)) = \text{true}; W(l)\}$ subst-add

1.1 $\{\text{sorted}(\text{msort}(\text{nil})) = \text{true}; W(\text{nil})\}$ $\surd$

1.2 $\{\text{sorted}(\text{msort}(x.\text{nil})) = \text{true}; W(x.\text{nil})\}$ $\surd$

1.3 $\{\text{sorted}(\text{msort}(x.y.l)) = \text{true}; W(x.y.l)\}$ axm-rewrite
$\{\text{sorted}(\text{merge}(\text{msort}(x.\text{split1}(l)), \text{msort}(y.\text{split2}(l)))) = \text{true}; W(x.y.l)\}$

1.3.1 $\{\text{def msort}(x.\text{split1}(l)), \ldots ; W(x.y.l)\}$ $\surd$

1.3.2 $\{\text{def msort}(y.\text{split2}(l)), \ldots ; W(x.y.l)\}$ $\surd$

1.3.3 $\{\text{sorted }(\text{msort}(x.\text{split1}(l)) = \text{true}, \ldots ; W(x.y.l)\}$ ind-rewrite

1.3.3.1 $\{\text{def } x.\text{split1}(l), \ldots ; W(x.y.l)\}$ $\surd$

1.3.3.2 $\{\text{true} = \text{true}, \ldots ; W(x.y.l)\}$ $\surd$ mit $=$-decomp

1.3.3.3 $\{W(x.\text{split1}(l)) < W(x.y.l), \ldots ; W(x.y.l)\}$ offen

1.3.4 $\{\text{sorted}(\text{msort}(y.\text{split2}(l))) = \text{true}, ..; W(x.y.l)\}$ ind-rewrite

1.3.4.1 $\{\text{def } y.\text{split2}(l), \ldots ; W(x.y.l)\}$ $\surd$

1.3.4.2 $\{\text{true} = \text{true}, \ldots ; W(x.y.l)\}$ $\surd$ mit $=$-decomp

1.3.4.3 $\{W(y.\text{split2}(l)) < W(x.y.l), \ldots ; W(x.y.l)\}$ offen

1.3.5 $\{\text{true} = \text{true}, \ldots ; W(x.y.l)\}$ $\surd$ mit $=$-decomp

Beweis des Ziels (2)

Setzen des Gewichts $W(l) :\equiv l$ liefert

1.3.3.3
$\{x.\text{split1}(l) < x.y.l, \ldots ; x.y.l\}$ $<$-mono
$\{\text{split1}(l) < l, x.\text{split1}(l) < x.y.l, \ldots ; x.y.l\}$ lma-subs

1.3.3.3.1
$\{l \neq \text{nil}, \ldots , x.\text{split1}(l) < x.y.l, \ldots ; x.y.l\}$ $\neq$-unif
$\{\ldots , x.\text{nil} < x.y.\text{nil}, \ldots ; x.y.l\}$ $\surd$ mit $<$-decomp

1.3.3.3.2
$\{\text{le}(l) \neq s(0), x.\text{split1}(l) < x.y.l, \ldots ; x.y.l\}$ lma-rewrite
$\{x.l < x.y.l, \ldots ; x.y.l\}$ $\surd$ mit $<$-decomp

1.3.4.3
$\{y.\text{split2}(l) < x.y.l, \ldots ; x.y.l\}$ $<$-mono
$\{\text{split2}(l) < l, y.\text{split2}(l) < x.y.l, \ldots ; x.y.l\}$ lma-subs
$\{l \neq \text{nil}, \ldots , y.\text{split2}(l) < x.y.l, \ldots ; x.y.l\}$ $\neq$-unif
$\{y.\text{nil} < x.y.\text{nil}, \ldots ; x.y.l\}$ $\surd$ mit $<$-decomp

Beachte

Die angewandten Lemmata sind

- L-sorted-merge bei 1.3
- L-split1-ind bei 1.3.3.3
- L-split2-ind bei 1.3.4.3
- L-le1-split1 bei 1.3.3.3.2

Literatur

Avenhaus, J.; Madlener, K.:

Theorem Proving in Hierarchical Clausal Specifications, in Du, Ko (eds.): Advances in Algorithms, Languages, and Complexity, Kluwer 1997, pp. 1-51.

Bachmair, L.:

Proof by consistency in equational theories, in Proc. Third Annual IEEE Symposium on Logic in Computer Science, 1988, pp. 228-233.

Ehrig, H.; Mahr, B.:

Fundamentals of Algebraic Specification, Springer 1985.

Giesl, J.; Kapur, D.:

Decidable Classes of Inductive Theorems, in Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001), Lecture Notes in Artificial Intelligence 2083, Springer 2001, pp. 469-484.

Giesl, J.:

Induction Proofs with Partial Functions, in Journal of Automated Reasoning, 2001, Vol. 26, No. 1, pp. 1-49.

Jouannaud, J. P.; Kounalis, E.:

Automatic proofs by induction in equational theories without constructors, in Proc. of the Annual IEEE Symposium on Logic in Computer Science, 1986, pp. 358-366.

Kapur, D.; Subramaniam, M.:

New uses of linear arithmetic in automated theorem proving by induction, in Journal of Automated Reasoning, 1996, Vol. 16, pp. 39-78.

Kühler, U.:

A tactic based inductive theorem prover for data types with partial operations, in DISKI, Dissertationen zur künstlichen Intelligenz, 238, 1999.

Walther, C.:

Mathematical induction, in C. J. Hogger D. M. Gabbay and J. A. Robinson (Eds.): Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press 1994, Vol. 2, pp. 122-227.

Zhang, H.; Kapur, D.; Krishnamoorthy, M. S.:

A Mechanizable Induction Principle for Equational Specifications, in E. Lush and R. Overbeek (Eds): Proc. 9th International Conference on Automated Induction, Lecture Notes in Computer Science 310, Springer 1988, pp. 162-181.