Semantische Induktionsordnung

Beispiel

Sortieren durch Mischen: msort.

R: split1(nil) $\rightarrow$ nil　　split2(nil) $\rightarrow$ nil
split1(x.nil) $\rightarrow$ x.nil　　split2(x.nil) $\rightarrow$ nil
split1($x.y.l$) $\rightarrow$ x.split1(l)　　split2($x.y.l$) $\rightarrow$ y.split2(l)
merge(nil, nil) $\rightarrow$ nil
merge($x.l_1$, nil) $\rightarrow$ $x.l_1$
merge(nil, $y.l_2$) $\rightarrow$ $y.l_2$
merge($x.l_1, y.l_2$) $\rightarrow$ x.merge($l_1, y.l_2$) $\Leftarrow x \leq y =$ true
merge($x.l_1, y.l_2$) $\rightarrow$ y.merge($x.l_1, l_2$) $\Leftarrow x \leq y =$ false
msort(nil) $\rightarrow$ nil
msort(x.nil) $\rightarrow$ x.nil
msort($x.y.l$) $\rightarrow$ merge(msort(split1($x.y.l$)), msort(split2($x.y.l$)))

Behauptung: sorted(msort(l)) $=$ true.

Beweis-Idee:
Setze le(l)=Länge von l und zeige

le($x.y.l$) $>$ le(split1($x.y.l$))
le($x.y.l$) $>$ le(split2($x.y.l$))
Setze $\sigma \succ_i \tau$ gdw le($\sigma(l)$) $>$ le($\tau(l)$)

Also: Induktion nach der Länge von l.

3.2 Die Coverset-Methode

Sei sig, spec $=$ (sig, E), $C \subseteq F$ wie immer.
$e \equiv u = v \Rightarrow s = t$.

Frage

Wie findet man eine Induktionsordnung $\succ_i$ und ein Induktionsschema für den Beweis zu $e \in ITh$(spec)?

Idee dazu

Wähle Induktionsschema nach dem Definitionsschema eines „geeigneten“ $f \in F - C$. Dies Induktionsschema wird durch ein „Coverset“ beschrieben.

Benutze $>_{TT}$ zur Definition von $\succ_i$.

Falls spec $=$ (sig, R) und R simplifizierend bezüglich $\succ$, so benutze evtl. auch $\succ$ zur Definition von $\succ_i$. Beispiel: Sei $t \in$ Term(F, V) gegeben. Setze $\sigma \succ_i \tau$ gdw $\sigma(t) \succ \tau(t)$.

Ein motivierendes Beispiel (1)

Sei $F = \{0, s, \text{double}, \text{half}, \text{true}, \text{false}, \text{even}\}$,
$C = \{0, s, \text{true}, \text{false}\}$.

R: even$(0) \to$ true | half$(0) \to 0$
even$(s(0)) \to$ false | half$(s(0)) \to 0$
even$(s(s(x))) \to$ even(x) | half$(s(s(x))) \to s(\text{half}(x))$
double$(0) \to 0$
double$(s(x)) \to s(s(\text{double}(x)))$

$e \equiv \text{even}(x) = \text{true} \Rightarrow \text{double}(\text{half}(x)) = x \in ITh(\text{spec})$.

Dann: C ist eine Konstruktormenge.

Idee

Induktion nach dem Definitionsschema von even.

1. Fall: $\sigma_1(x) \equiv 0$ Basisfall.
2. Fall: $\sigma_2(x) \equiv s(0)$ Basisfall.
3. Fall: $\sigma_3(x) \equiv s(s(x))$ reduziert $\sigma_3(e)$ auf $\tau(e)$ mit $\tau(x) \equiv x$.

Also

$M = \{0, s(0), s(s(x))\}$ ist eine „Überdeckung“ von Term(C).

$C_{\text{even}} = \{\langle 0, \varnothing\rangle, \langle s(0), \varnothing\rangle, \langle s^2(x), \{x\}\rangle\}$ ist ein „Coverset“ für even $\in F$.

Ein motivierendes Beispiel (2)

C_{even} liefert das Induktionsschema

(1) $\Rightarrow \sigma_1(e)$
(2) $\Rightarrow \sigma_2(e)$
(3) $\tau(e) \Rightarrow \sigma_3(e)$

und die Induktionsordnung

$\sigma' \succ_i \sigma \text{ gdw } \sigma'(x) >_{TT} \sigma(x)$

Es gilt

Zu (1):
even$(0) = $ true $\Rightarrow$ double(half$(0)) = 0 \in ITh(\text{spec})$.

Zu (2):
even$(s(0)) = $ true $\Rightarrow$ double(half$(s(0))) = s(0) \in ITh(\text{spec})$
denn even$(s(0)) \neq_R$ true.

Zu (3): Es reduziert sich

even$(s^2(x)) = $ true $\Rightarrow$ double(half$(s^2(x))) = s^2(x)$.

zu

even$(x) = $ true $\Rightarrow s^2(\text{double}(\text{half}(x))) = s^2(x)$.

Dies gilt, denn nach IV gilt

$\tau(e) \equiv \text{even}(x) = \text{true} \Rightarrow \text{double}(\text{half}(x)) = x$.

Frage

Wie kommt man (automatisch) zu dem Induktionsschema?

Konstruktor-basierte Spezifikationen (1)

Beachte

Stärkere Voraussetzungen an die Spezifikationen erlauben effizientere Algorithmen.

Idee

spec $=$ (sig, R), sig $= (S, F, \alpha)$, $F = C \cup D$ mit $C \cap D = \varnothing$.

Das grund-konvergente R definiert die $f \in D$ (definierte Operatoren) eventuell partiell über den Konstruktortermen in Term(C). Die $c \in C$ heißen **syntaktische Konstruktoren**.

Ab jetzt

Sei $F = C \cup D$ mit $C \cap D = \varnothing$.

Sei $f \in D$. Eine f-**Definition** ist eine endliche Menge bedingter Regeln der Form $l \to r \Leftarrow u = v$ mit
(1) $l \equiv f(t_1, \ldots, t_n)$ mit $t_i \in$ Term(C, V),
(2) $r, u_i, v_i \in$ Term(F, V),
(3) Var$(r, u_i, v_i) \subseteq$ Var(l) und
(4) $l \succ r, u_i, v_i$ für eine Simplifikationsordnung $\succ$.

Konstruktor-basierte Spezifikationen (2)

Definition 3.1

Sei sig $= (S, F, \alpha)$, $F = C \cup D$ und $C \cap D = \varnothing$.

Eine **konstruktor-basierte Spezifikation** ist spec $=$ (sig, R) mit $R = R_C \cup \bigcup_{f \in D} R_f$.

Dabei sind die R_f f-Definitionen und R_C ist ein Regelsystem über sig$_c = (S, C, \alpha_c)$. Weiter ist R simplifizierend bezüglich einer Reduktionsordnung $\succ$.

Beachte

Ist $R_C = \varnothing$, so sind die Konstruktoren in C **frei**.

In der Praxis sind viele Spezifikationen konstruktor-basiert.

Beispiele

a) Die Spezifikationen zu ins und double/half sind konstruktor-basiert.
b) Die Spezifikation der Ackermann-Funktion ist konstruktor-basiert.

$$
\begin{aligned}
R\colon\ & a(0, y) \to s(y) \\
& a(s(x), 0) \to a(x, s(0)) \\
& a(s(x), s(y)) \to a(x, a(s(x), y))
\end{aligned}
$$

Coversets

Definition 3.2

Sei spec $=$ (sig, R) eine konstruktor-basierte Spezifikation.

sig $= (S, F, \alpha)$ und $C \subseteq F$.

Sei $l \to r \Leftarrow u = v$ in R_f und $l \equiv f(t_1, \ldots, t_n)$. Sei M die Menge der Argument-n-Tupel von f in r. Dann heißt

$$\langle (t_1, \ldots, t_n), M, u = v \rangle$$

das **Überdeckungstripel** zu $l \to r \Leftarrow u = v$.

Sei C_f die Menge der Überdeckungstripel zu den Regeln in R_f. Dann heißt C_f ein **Coverset** für f.

Beispiel Ackermann-Funktion

C_a besteht aus

$\langle (0, y), \varnothing, \varnothing \rangle \qquad \langle (s(x), 0), \{(x, s(0))\}, \varnothing \rangle$

$\langle (s(x), s(y)), \{(x, a(s(x), y)), (s(x), y)\}, \varnothing \rangle$

Induktionsschemata (1)

Sei

spec $=$ (sig, R) konstruktor-basiert.

$e \equiv u = v \Rightarrow s = t$.

Zu zeigen: e gilt in $\mathcal{I} = \mathcal{I}(\text{spec})$.

Beachte

Ein Induktionsschema wird mit Hilfe eines Teilterms t_0 definiert. Dazu gibt es verschiedene Möglichkeiten. Mit einer guten Heuristik ist t_0 zu wählen.

Hier

Definition eines Induktionsschemas zu $t_0 \equiv f(x_1, \ldots, x_n)$ mit $f \in F - C$.

Beispiel

$e \equiv \text{even}(x) = \text{true} \Rightarrow \text{double}(\text{half}(x)) = x$.

Wähle $t_0 \equiv \text{even}(x)$, also $f = \text{even}$.

Induktionsschemata (2)

Definition 3.3

Sei $c = \langle (s_1, \ldots, s_n), M, u' = v' \rangle$ ein Überdeckungstripel zu f.

Sei $M = \{(t_1^i, \ldots, t_n^i) \mid 1 \leq i \leq k\}$, $e \equiv u = v \Rightarrow s = t$ und $e/\pi \equiv f(x_1, \ldots, x_n)$. Dies definiert

$$\sigma = \{x_j \leftarrow s_j \mid 1 \leq j \leq n\} \qquad \tau_i = \{x_j \leftarrow t_j^i \mid 1 \leq j \leq n\}$$

Dann ist

$$[u' = v' \wedge \bigwedge_{i=1}^{k} \tau_i(e)] \Rightarrow \sigma(e)$$

das **Induktions-Teilziel** zum Überdeckungstripel c. Die Menge dieser Teilziele bildet das **Induktionsschema** zu spec, e, $f(x_1, \ldots, x_n)$.

Beachte

a) Wir setzen voraus, dass e und $\langle (s_1, \ldots, s_n), M, u' = v' \rangle$ Variablen-disjunkt sind.

b) Jedes Überdeckungstripel in C_f erzeugt genau ein Teilziel.

Beispiel: Ackermann-Funktion

Sei

$F = \{0, s, a, +, \leq, \text{true}, \text{false}\}$, $C = \{0, s, \text{true}, \text{false}\}$.

R: $a(0, y) \to s(y)$
$a(s(x), 0) \to a(x, s(0))$
$a(s(x), s(y)) \to a(x, a(s(x), y))$
und Definitionen von $+$, $\leq$ wie früher

$e \equiv x' + y' \leq a(x', y') = \text{true}$.

$\succ$ die LPO zur Präzedenz $a >_F s >_F 0$

Also

$$C_a = \{\langle (0, y), \varnothing, \varnothing \rangle, \langle (s(x), 0), \{(x, s(0))\}, \varnothing \rangle, \\ \langle (s(x), s(y)), \{(x, a(s(x), y)), (s(x), y)\}, \varnothing \rangle\}$$

Wähle die Induktionsordnung
$\sigma \succ_i \tau \;\; \text{gdw} \;\; \sigma(a(x, y)) \succ \tau(a(x, y))$

Dies liefert die Teilziele (ohne „$=$ true“)

1. $\Rightarrow 0 + y \leq a(0, y)$.
2. $x + s(0) \leq a(x, s(0)) \Rightarrow s(x) + 0 \leq a(s(x), 0)$.
3. $s(x) + y \leq a(s(x), y), x + a(s(x), y) \leq a(x, a(s(x), y))$
$\Rightarrow s(x) + s(y) \leq a(s(x), s(y))$

Beispiel: Sortieren durch Einfügen

Sei $F = \{0, s, \text{nil}, ., \text{ins}, \text{sorted}, \leq, \text{true}, \text{false}\}$,
$C = \{0, s, \text{true}, \text{false}, \text{nil}, .\}$

R:
$$
\begin{array}{lll}
\text{sorted}(\text{nil}) \to \text{true} & & \\
\text{sorted}(x.\text{nil}) \to \text{true} & & \\
\text{sorted}(x.y.l) \to \text{sorted}(y.l) & \Leftarrow & x \leq y = \text{true} \\
\text{sorted}(x.y.l) \to \text{false} & \Leftarrow & x \leq y = \text{false}
\end{array}
$$
und Definitionen von ins, $\leq$ wie früher

$e \equiv \text{sorted}(l') = \text{true} \Rightarrow \text{sorted}(\text{ins}(x', l')) = \text{true}$

Das Coverset $C_f = C_{\text{sorted}}$ ist

1. $\langle \text{nil}, \varnothing, \varnothing \rangle$
2. $\langle x.\text{nil}, \varnothing, \varnothing \rangle$
3. $\langle x.y.l, \{y.l\}, x \leq y = \text{true} \rangle$
4. $\langle x.y.l, \varnothing, x \leq y = \text{false} \rangle$

Dies liefert das Induktionsschema (ohne „= true“)

1. $\Rightarrow [\text{sorted}(\text{nil}) \Rightarrow \text{sorted}(\text{ins}(x', \text{nil}))]$
2. $\Rightarrow [\text{sorted}(x.\text{nil}) \Rightarrow \text{sorted}(\text{ins}(x', x.\text{nil}))]$
3. $x \leq y = \text{true} \land (\text{sorted}(y.l) \Rightarrow \text{sorted}(\text{ins}(x', y.l)))$
   $\Rightarrow [\text{sorted}(x.y.l) \Rightarrow \text{sorted}(\text{ins}(x', x.y.l))]$
4. $x \leq y = \text{false} \Rightarrow [\text{sorted}(x.y.l) \Rightarrow \text{sorted}(\text{ins}(x', x.y.l))]$

Bemerkung

Sei

Gegeben: spec = (sig, R), $e \equiv u = v \Rightarrow s = t$.

Zu zeigen: e gilt in $\mathcal{I} = \mathcal{I}(\text{spec})$.

Ziel

Sind die Teilziele gültig in $\mathcal{I}$, so ist auch e gültig in $\mathcal{I}$.

Beachte

Dazu benötigt man

a) die Vollständigkeit: Jede Grundinstanz $\sigma(e)$ wird durch einen Fall im Induktionsschema abgedeckt.
b) die Korrektheit: Es gibt eine Noethersche Ordnung (Induktionsordnung) nach der in jedem Teilziel die Voraussetzung kleiner ist als die Behauptung.