Verifikation eines abstrakten Compilers

Praktikum
Formale Entwicklung objektorientierter Software

Th. Käufl Institut für Theoretische Informatik

Als beispielhafte Aufgabe eines Compilers betrachten wir die Übersetzung von Zuweisungen der Form $x := y_1 + \cdots + y_n$, wobei x nicht auf der rechten Seite vorkommt in eine Folge $x := 0; x := x + y_1; \ldots x := x + y_n$ maschinennaher Anweisungen.

Bezeichne *Sexp* die Menge der syntaktisch korrekten Zuweisungen der Quellsprache und *Texp* die der Zielsprache. Den Compiler können wir abstrakt als Abbildung *gen-texp*: $Sexp \to Texp$ ansehen. Man wird erwarten, daß ein Compiler

1. syntaktisch korrekte Zuweisungen der Quellsprache in syntaktisch korrekte der Zielsprache übersetzt und

2. die Bedeutung der Anweisungen in der Quellsprache mit der der übersetzten der Zielsprache übereinstimmt.

1. Allgemeines

Wir verwenden eine sortierte Logik. Sorten sind Teilmengen des Universums. Durch Einführung von (einstelligen Sorten-) Prädikaten kann eine sortierte Logik in eine unsortierte umgewandelt werden.

Die Schreibweise der logischen Formeln und der Festlegungen von Sorten sind nicht in der Syntax von KeY. Das gleiche gilt für die Spezifikation der Prädikate, Funktionen und Konstantenzeichen.

Variablen und Folgen von Variablen

Die beiden Sorten benötigen wir für die Quell- und für die Zielsprache.

Variablen

sort *Var*

Listen (Folgen) von Variablen.

sort *Vars*

Konstruktoren

type v_ε: *Vars*	die leere Folge
type *vcons*(*Var Vars*): *Vars*	nichtleere Folgen

Jede Folge von Variablen ist leer oder wird durch *vcons* erzeugt.

$$\forall vs\colon Vars\ vs = v_\varepsilon \vee (\exists v_1\colon Var\ (\exists vs_1\colon Vars\ vs = vcons(v_1, vs_1)))$$

Eine durch *vcons* erzeugte Liste ist nicht leer:

$$\forall v\colon Var\ (\forall vs\colon Vars\ \neg vcons(v, vs) = v_\varepsilon)$$

2. Die Quellsprache

Die in der Quellsprache zulässigen Zuweisungen haben die Form

$$x := y_1 + \cdots + y_n,$$

wobei x verschieden von den y_i ist und $n \geq 1$.

2.1. Syntax

Sorten

Die Sorte der Zuweisungen ist *Sexp*.

sort *Sexp*

Funktionen

Konstruktorfunktion

type *mk-sexp*(*Var*, *Vars*): *Sexp*

Var ist die linke und *Vars* die rechte Seite einer Zuweisung der Zielsprache.

Selektorfunktionen

type *s-l*(*Sexp*): *Var*
type *s-r*(*Sexp*): *Vars*

s-l wählt die linke, *s-r* die rechte Seite einer Zuweisung aus.

Für den Konstruktor- und die Selektoren gelten

$$\forall v\colon Var\ (\forall vsVars\colon\ s\text{-}l(mk\text{-}sexp(v, vs)) = v)$$
$$\forall v\colon Var\ (\forall vs\colon Vars\ s\text{-}r(mk\text{-}sexp(v, vs)) = vs)$$
$$\forall se\colon Sexp\ mk\text{-}sexp(s\text{-}l(se), s\text{-}r(se)) = se$$

Sytaktisch korrekte Zuweisungen
Das Prädikat *Wf-Sexp*kennzeichnet syntaktisch korrekte Zuweisungen der Quellsprache.

type *Wf-Sexp*(*Sexp*)

In einer syntaktisch korrekten Zuweisung (Sexp) kommt die Variable auf der linken Seite von := nicht auf der rechten vor und die rechte Seite besteht aus mindestens einer Variablen.

$$\forall z\ (\mathit{Wf\text{-}Sexp}(z) \leftrightarrow (\exists x\colon \mathit{Var}\ \exists y\colon \mathit{Vars}\ (z = \mathit{mk\text{-}sexp}(x,y) \wedge \neg \mathit{In}(x,y) \wedge y \neq v_\varepsilon)))$$

In[1] ist ein Prädikat, das wahr ist, wenn das erste Argument im zweiten, einer Liste, auftritt.

2.2. Semantik

Hier nicht.

3. Die Zielsprache

Syntaktisch zulässige Zuweisungen in der Zielsprache haben die Form $x := 0$ oder $x := x + y$, wobei x und y zwei verschiedene Variablen sind.

Eine syntaktisch korrekte Folge von Zuweisungen in der Zielsprache hat die Gestalt

$$x := 0;\ x := x + y_1;\ \ldots\ x := x + y_n$$

3.1. Abstrakte Syntax

Sorten
Initialisierungen sind Zuweisungen der Form $x := 0$.

sort *Ass*

Zuweisungen in Zweiaddressform haben die Gestalt $x := x + y$.

sort *RAsg*

Folgen von Zuweisungen der Zielsprache, denen eine Initialisierung vorausgeht, gehören zur Sorte *Texp*.

sort *Texp*

[1] Siehe Anhang

Funktionen

Konstruktorfunktionen

type $mk\text{-}ass(Var)$: Ass	für Initialisierungen
type $mk\text{-}rasg(Var, Var, Var)$: $RAsg$	für Zuweisungen

Selektorfunktionen für Zuweisungen

type $s\text{-}tar(RAsg)$: Var	die linke Seite
type $s\text{-}so_1(RAsg)$: Var	die erste Variable auf der rechten Seite
type $s\text{-}so_2(RAsg)$: Var	die zweite Variable auf der rechten Seite

mk-rasg und die Selektorfunktionen erfüllen

$$\forall v_1, v_2, v_3 : Var\ s\text{-}tar(mk\text{-}rasg(v_1, v_2, v_3)) = v_1$$
$$\forall v_1, v_2, v_3 : Var\ s\text{-}so_1(mk\text{-}rasg(v_1, v_2, v_3)) = v_2$$
$$\forall v_1, v_2, v_3 : Var\ s\text{-}so_2(mk\text{-}rasg(v_1, v_2, v_3)) = v_3$$
$$\forall r\text{-}ass : RAsg\ mk\text{-}rasg(s\text{-}l(r\text{-}ass), s\text{-}so_1(r\text{-}ass), s\text{-}so_2(r\text{-}ass)) = r\text{-}ass$$

Prädikate für syntaktisch korrekte Anweisungen

Die beiden folgenden Prädikate kennzeichnen korrekte Anweisungen der Zielsprache

type $Is\text{-}Ass$	für Initialisierungen
type $Is\text{-}R\text{-}Asg$	für Zuweisungen

Sie sind festgelegt durch

$$\forall z\ (Is\text{-}Ass(x) \leftrightarrow \exists v : Var\ z = mk\text{-}ass(v))$$
$$\forall r\ (Is\text{-}R\text{-}Asg(r) \leftrightarrow (\exists v_0, v_1 : Var\ r = mk\text{-}rasg(v_0, v_0, v_1) \wedge v_0 \neq v_1))$$

Folgen von Zuweisungen in Zweiadressform

Die Festlegungen in diesem Teil benötigen wir zur Definition syntaktisch korrekter Zuweisungen der Zielsprache. Dazu führen wir mit

sort *Tlist*

die Sorte der Listen ein, deren Elemente zur Sorte *RAsg* gehören.

Funktionen für *Tlist*

type t_ε: $Tlist$	die leere Liste
type $tcons(RAsg, Tlist)$: $Tlist$	Konstruktor für Listen

Kennzeichendes Prädikat

type $Is\text{-}Asg\text{-}List(Var\ Tlist)$,

das für Folgen von Zuweisungen der Zielsprache wahr ist. Für den Korrektheitsbeweis benötigen wir die folgenden beiden Eigenschaften.

$$\forall x : Var\ Is\text{-}Asg\text{-}List(x, t_\varepsilon)$$
$$\forall x : Var\ (\forall t : Tlist\ (\forall u : RAsg\ (\forall s : Tlist$$
$$t = tcons(u, s) \wedge x = s\text{-}tar(u) \wedge Is\text{-}R\text{-}Asg(u) \wedge Is\text{-}Asg\text{-}List(x, s)$$
$$\rightarrow Is\text{-}Asg\text{-}List(x, t))))$$

Syntaktisch korrekte Zuweisungen der Zielsprache
Diese Zuweisungen werden gekennzeichnet durch das Prädikat *Wf-Texp*

type *Wf-Texp*(*Texp*)

das durch

$$\begin{array}{l}\forall te\colon Texp\\ \quad (Wf\text{-}Texp(te)\\ \quad \leftrightarrow \exists v\colon Var\ \exists tli\colon Tlist\ te = mk\text{-}texp(mk\text{-}ass(v), tli) \wedge IsAsgList(v, tli))\end{array}$$

festgelegt ist.

3.2. Semantik

Hier nicht

4. Der Compiler

4.1. Die Spezifikation des Compilers

gen-texp ist der Übersetzer

type *gen-texp*(*Sexp*): *Texp*

Als (Hilfs-) Funktion verwendet *gen-texp*die Funkion f.

type $f(Var, Vars, Tlist)$: *Texp*

*gen-texp*ist festgelegt durch

$$\forall se\colon Sexp\ gen\text{-}texp(se) = f(s\text{-}l(se), s\text{-}r(se), t_\varepsilon)$$

Bei der Festlegung von f betrachten wir zuerst den Fall, daß das zweite Argument leer ist. In diesem Fall muß dem dritten Argument eine Initialisierung vorangestellt werden. Dies leistet *mk-texp*.

type *mk-texp*(*Ass Tlist*): *Texp*
$\forall x\colon Var\ (\forall tli\colon Tlist\ f(x, v_\varepsilon, tli) = mk\text{-}texp(mk\text{-}ass(x), tli))$

Ist das zweite Argument, die Variablenliste nichtleer, muß eine neue Zuweisung am Ende des dritten Arguments angehängt werden. Dazu verwenden wir die Funktion $conc_1$.[2]

$$\begin{array}{l}\forall x\colon Var\ (\forall vs\colon Vars\ (\forall tli\colon Tlist\ (\forall z\colon Var\ (\forall vs_1\colon Vars\\ \quad vs = vcons(z, vs_1)\\ \quad \rightarrow f(x, vs, tli) = f(x, vs_1, conc_1(tli, mk\text{-}rasg(x, x, z)))))))\end{array}$$

[2]Siehe Anhang

4.2. Die Korrektheit des Compilers

Ein korrekter Übersetzer erzeugt zu jeder syntaktisch korrekten Zuweisung der Quellsprache eine syntaktisch korrekte Folge von Anweisungen der Zielsprache.

$$\forall s\ (\mathit{Wf\text{-}Sexp}(s) \rightarrow \mathit{Wf\text{-}Texp}(\mathit{gen\text{-}texp}(s)))$$

Zum Nachweis benötigen wir die Aussage, daß $f(x, vars, t_\varepsilon)$ zu *Texp* gehört, also eine korrekte Folge von Zuweisungen ist, sofern x nicht zu *vars* gehört.

$$\forall x\colon \mathit{Var}\ \forall \mathit{vars}\colon \mathit{Vars}\ (\neg \mathit{In}(x, \mathit{vars}) \rightarrow \mathit{Wf\text{-}Texp}(f(x, \mathit{vars}, t_\varepsilon)))$$

Mit ihr folgt die Korrektheitsaussage:
Da $\mathit{gen\text{-}texp}(s) = f(\mathit{s\text{-}l}(s), \mathit{s\text{-}r}(s), t_\varepsilon)$ und $\mathit{Wf\text{-}Sexp}(s)$ kommt $\mathit{s\text{-}l}(s)$ nicht in $\mathit{s\text{-}r}(s)$ vor. Also $\mathit{Wf\text{-}Texp}(f(\mathit{s\text{-}l}(s), \mathit{s\text{-}r}(s), t_\varepsilon))$ und das heißt $\mathit{Wf\text{-}Texp}(\mathit{gen\text{-}texp}(s))$

Nachweis der Hilfsaussage für f
Fall 1. $\mathit{vars} = v_\varepsilon$
$f(x, v_\varepsilon, t_\varepsilon) = \mathit{mk\text{-}texp}(\mathit{mk\text{-}ass}(x), t_\varepsilon)$ und diese Liste erfüllt *Is-Asg-List*.
Fall 2. $\mathit{vars} \neq v_\varepsilon$
Hier zeigen wir $\mathit{Is\text{-}Asg\text{-}List}(x, tli) \wedge \neg \mathit{In}(x, \mathit{vars}) \rightarrow \mathit{Wf\text{-}Texp}(f(x, \mathit{vars}, tli))$ durch Induktion über die Länge von *vars*.
(i) $\mathit{vars} = v_\varepsilon$
$f(x, \varepsilon, tli) = (x := 0) \circ tli$. Da $\mathit{Is\text{-}Asg\text{-}List}(x, tli)$ hat jede Zuweisung in *tli* die Form $x := x + u$.
(ii) $\mathit{vars} = (y) \circ \mathit{vars}'$
$f(x, (y) \circ \mathit{vars}', tli) = f(x, \mathit{vars}', tlist \circ (x := x + y))$. Nach Voraussetzung trifft *Is-Asg-List* auf *tli* zu. Alle Zuweisungen in *tli* haben also die Gestalt $x := x + u$. Damit trifft *Is-Asg-List* auch auf $tlist \circ (x := x + y)$ zu. Mit Hilfe der Induktionsvoraussetzung folgt die Behauptung.

5. Anhang

$conc_1$
Für diese Funktion müssen wir nur festlegen, wann mit ihrer Hilfe eine korrekte Asg-Liste erzeugt wird.

$$\begin{aligned}&\forall x\colon \mathit{Var}\ (\forall tli\colon \mathit{Tlist}\ (\forall r\colon \mathit{RAsg}\\ &\quad \mathit{Is\text{-}Asg\text{-}List}(x, tli) \wedge \mathit{Is\text{-}R\text{-}Asg}(r) \wedge x = \mathit{s\text{-}tar}(r)\\ &\quad \rightarrow \mathit{Is\text{-}Asg\text{-}List}(x, conc_1(tli, r))))\end{aligned}$$

In
Das Prädikat ist festgelegt durch

$$\begin{aligned}&\forall x\ (\forall vs\colon \mathit{Vars}\\ &\quad \mathit{In}(x, vs)\\ &\quad \leftrightarrow (\exists v\colon \mathit{Var}\ (\exists vs_1\colon \mathit{Vars}\ (vs = v.cons(v, vs_1) \wedge (x = v \vee \mathit{In}(x, vs_1))))))\end{aligned}$$

Stattdessen können auch die beiden Aussagen

$$\forall x\ \neg In(x, v_\varepsilon)$$

für leere Listen und

$$\forall x\ (\forall v\colon Var\ (\forall vs\colon Vars\ x = v \vee In(x, vs) \rightarrow In(x, vcons(v, vs))))$$

für nichtleere Listen verwendet werden. Sie folgen aus der oben angegebenen Definition.