Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Olaf Müller

WS 1998/99
Lösungsvorschläge zu Blatt 9
2. 2. 1999

Übungen zur Semantik von Programmiersprachen

Aufgabe 2 **(Lösungsvorschlag)**

(a) Die cpo $\mathbf{T}_\perp = \{false, true, \perp\}$ entsteht aus der Menge $\mathbf{T} = \{false, true\}$ durch Lifting. Damit ist $\mathbf{T}_\perp$ flach geordnet.

Auf flachen cpo's sind alle monotonen Funktionen stetig (warum?). Deshalb bestimmen wir im folgenden alle monotonen Erweiterungen von $\vee$ und $\wedge$ auf $\mathbf{T}_\perp$.

Da $\vee : \mathbf{T}_\perp \times \mathbf{T}_\perp \to \mathbf{T}_\perp$ monoton sein soll, ergeben sich folgende Einschränkungen für den Wert von $\perp \vee false$:

$$\begin{array}{l} \perp \vee false \sqsubseteq false \vee false = false \text{ und} \\ \perp \vee false \sqsubseteq true \vee false = true \end{array}$$

Daraus folgt

$$\perp \vee false = \perp$$

und mit symmetrischer Argumentation

$$false \vee \perp = \perp \ .$$

Aus der Monotonie von $\vee$ folgt für den Wert $\perp \vee true$:

$$\begin{array}{l} \perp \vee true \sqsubseteq false \vee true = true \text{ und} \\ \perp \vee true \sqsubseteq true \vee true = true \end{array}$$

Daraus folgt

$$\perp \vee true \in \{\perp, true\}$$

und mit symmetrischer Argumentation

$$true \vee \perp =\in \{\perp, true\} \ .$$

Schließlich impliziert die Monotonie von $\vee$ für den Wert $\perp \vee \perp$:

$$\begin{array}{l} \perp \vee \perp \sqsubseteq false \vee false = false \text{ und} \\ \perp \vee \perp \sqsubseteq true \vee true = true \end{array}$$

und damit

$$\perp \vee \perp = \perp \ .$$

Für $\wedge$ lassen sich entsprechende Einschränkungen finden.

Diese Einschränkungen lassen folgende Erweiterungen zu:

Strikte Erweiterungen:

$\vee$	$\perp$	$false$	$true$
$\perp$	$\perp$	$\perp$	$\perp$
$false$	$\perp$	$false$	$true$
$true$	$\perp$	$true$	$true$

$\wedge$	$\perp$	$false$	$true$
$\perp$	$\perp$	$\perp$	$\perp$
$false$	$\perp$	$false$	$false$
$true$	$\perp$	$false$	$true$

Sequentielle Erweiterungen:

Linkssequentiell:

$\lor$	$\bot$	$false$	$true$
$\bot$	$\bot$	$\bot$	$\bot$
$false$	$\bot$	$false$	$true$
$true$	$true$	$true$	$true$

$\land$	$\bot$	$false$	$true$
$\bot$	$\bot$	$\bot$	$\bot$
$false$	$false$	$false$	$false$
$true$	$\bot$	$false$	$true$

Rechtssequentiell:

$\lor$	$\bot$	$false$	$true$
$\bot$	$\bot$	$\bot$	$true$
$false$	$\bot$	$false$	$true$
$true$	$\bot$	$true$	$true$

$\land$	$\bot$	$false$	$true$
$\bot$	$\bot$	$false$	$\bot$
$false$	$\bot$	$false$	$false$
$true$	$\bot$	$false$	$true$

Parallele Erweiterungen:

$\lor$	$\bot$	$false$	$true$
$\bot$	$\bot$	$\bot$	$true$
$false$	$\bot$	$false$	$true$
$true$	$true$	$true$	$true$

$\land$	$\bot$	$false$	$true$
$\bot$	$\bot$	$false$	$\bot$
$false$	$false$	$false$	$false$
$true$	$\bot$	$false$	$true$

Alle diese Erweiterungen sind monoton.

(b) Wir geben die operationelle Semantik $\to$ jeweils in Form von Regeln an.

Operationelle Semantik für strikte Erweiterungen:

$$\frac{b_0 \to false \quad b_1 \to false}{b_0 \lor b_1 \to false} \qquad \frac{b_0 \to false \quad b_1 \to false}{b_0 \land b_1 \to false}$$

$$\frac{b_0 \to false \quad b_1 \to true}{b_0 \lor b_1 \to true} \qquad \frac{b_0 \to false \quad b_1 \to true}{b_0 \land b_1 \to false}$$

$$\frac{b_0 \to true \quad b_1 \to false}{b_0 \lor b_1 \to true} \qquad \frac{b_0 \to true \quad b_1 \to false}{b_0 \land b_1 \to false}$$

$$\frac{b_0 \to true \quad b_1 \to true}{b_0 \lor b_1 \to true} \qquad \frac{b_0 \to true \quad b_1 \to true}{b_0 \land b_1 \to true}$$

Es werden also stets beide Argumente (b_0 und b_1) ausgewertet. Dadurch wird die Striktheit implementiert.

Sequentielle Erweiterungen:

Linkssequentiell:

$$\frac{b_0 \to false \quad b_1 \to false}{b_0 \lor b_1 \to false} \qquad \frac{b_0 \to false}{b_0 \land b_1 \to false}$$

$$\frac{b_0 \to false \quad b_1 \to true}{b_0 \lor b_1 \to true} \qquad \frac{b_0 \to true \quad b_1 \to false}{b_0 \land b_1 \to false}$$

$$\frac{b_0 \to true}{b_0 \lor b_1 \to true} \qquad \frac{b_0 \to true \quad b_1 \to true}{b_0 \land b_1 \to true}$$

Es wird das erste Argument (b_0) ausgewertet; liefert dies den Wert $true$ ($false$), dann wird das zweite Argument nicht mehr ausgewertet, und die Disjunktion (Konjunktion) erhält den Wert $true$ ($false$). Nur wenn die Auswertung des ersten Arguments $false$ ($true$) liefert, wird das zweite Argument ausgewertet zur Bestimmung des

Werts der Disjunktion (Konjunktion). Diese Auswertungsweise erklärt den Namen "(links-)sequentielle" Erweiterung.

Rechtssequentiell:

$$\begin{array}{c|c}
\dfrac{b_0 \to false \;\; b_1 \to false}{b_0 \vee b_1 \to false} & \dfrac{b_1 \to false}{b_0 \wedge b_1 \to false} \\
\dfrac{b_0 \to true \;\; b_1 \to false}{b_0 \vee b_1 \to true} & \dfrac{b_0 \to false \;\; b_1 \to true}{b_0 \wedge b_1 \to false} \\
\dfrac{b_1 \to true}{b_0 \vee b_1 \to true} & \dfrac{b_0 \to true \;\; b_1 \to true}{b_0 \wedge b_1 \to true}
\end{array}$$

Hier wird analog ausgewertet wie im linkssequentiellen Fall; die Auswertung beginnt jedoch mit dem zweiten Argument.

Parallel:

$$\begin{array}{c|c}
\dfrac{b_0 \to true}{b_0 \vee b_1 \to true} & \dfrac{b_0 \to false}{b_0 \wedge b_1 \to false} \\
\dfrac{b_1 \to true}{b_0 \vee b_1 \to true} & \dfrac{b_1 \to false}{b_0 \wedge b_1 \to false} \\
\dfrac{b_0 \to false \;\; b_1 \to false}{b_0 \vee b_1 \to false} & \dfrac{b_0 \to true \;\; b_1 \to true}{b_0 \wedge b_1 \to true}
\end{array}$$

Beide Argumente werden gleichzeitig ausgewertet. Liefert eines davon den Wert $true$ ($false$), dann braucht das andere Argument zur Bestimmung des Werts der Disjunktion (Konjunktion) nicht fertig ausgewertet zu werden. Diese Auswertungsweise erklärt den Namen "parallele" Erweiterung.

Aufgabe 3 (Lösungsvorschlag)

Zu zeigen: fix ist stetig, d.h. die Funktion

$$\lambda f. \bigsqcup_{m \in \omega} f^m(\perp) \text{ ist stetig.}$$

Beweis: Wir haben zwei Möglichkeiten, den Beweis zu führen: Wir können uns auf die *Definition* der Stetigkeit stützen oder auf einen *Satz*, den wir bereits wissen. Suchen wir nach einem Satz. Nach der Gestalt der Behauptung brauchen wir einen Satz, der aussagt, wann eine Funktion stetig ist. In Blatt 8/Aufgabe 3 wurde gezeigt, daß die kleinste obere Schranke stetiger Funktionen *stetig* ist. Wenn wir $\lambda f. \bigsqcup_{m \in \omega} f^m(\perp)$ als kleinste obere Schranke stetiger Funktionen schreiben könnten, wäre die Behauptung gezeigt.

Daß $\sqcup$ auf Funktionen punktweise definiert ist, läßt vermuten, daß $\lambda f. \bigsqcup_{m \in \omega} f^m(\perp)$ die kleinste obere Schranke der Funktionen

$$\lambda f. f^m(\perp) \quad (m \in \omega)$$

ist. Wir müssen 3 Eigenschaften zeigen:

(i) Für alle $m \in \omega$ ist $\lambda f. f^m(\perp)$ stetig.

(ii) $(\lambda f. f^m(\perp))_{m \in \omega}$ ist eine Kette.

(iii) $\lambda f. \bigsqcup_{m \in \omega} f^m(\perp)$ ist die kleinste obere Schranke dieser Kette.

Zu (i):

Wir zeigen zuerst, daß für alle $m \in \omega$ die Funktion $\lambda f.f^m(\perp)$ monoton ist.

Induktion über m:

$m = 0$: Seien f und g beliebige stetige Funktionen mit $f \sqsubseteq g$. Dann gilt:

$$\begin{array}{cl} & f^0(\perp) \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & \perp \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & g^0(\perp) \end{array}$$

$m \rightsquigarrow m+1$: Seien f und g beliebige stetige Funktionen mit $f \sqsubseteq g$. Dann gilt:

$$\begin{array}{cl} & f^{m+1}(\perp) \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & f(f^m(\perp)) \\ \sqsubseteq & \{ \text{ Induktionsvoraussetzung und } f \text{ monoton}\} \\ & f(g^m(\perp)) \\ \sqsubseteq & \{ \ f \sqsubseteq g\} \\ & g(g^m(\perp)) \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & g^{m+1}(\perp) \end{array}$$

Nun zeigen wir, daß für alle $m \in \omega$ die Funktion $\lambda f.f^m(\perp)$ $\sqcup$-erhaltend ist.

Induktion über m:

$m = 0$: Sei $(f.n)_{n\in\omega}$ eine beliebige Kette stetiger Funktionen. Dann gilt:

$$\begin{array}{cl} & (\bigsqcup_{n\in\omega} f.n)^0(\perp) \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & \perp \\ = & \{ \text{ Kette konstant } \perp\} \\ & \bigsqcup_{n\in\omega}(f.n^0(\perp)) \end{array}$$

$m \rightsquigarrow m+1$: Sei $(f.n)_{n\in\omega}$ eine beliebige Kette stetiger Funktionen. Dann gilt:

$$\begin{array}{cl} & (\bigsqcup_{n\in\omega} f.n)^{m+1}(\perp) \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & (\bigsqcup_{n\in\omega} f.n)((\bigsqcup_{n\in\omega} f.n)^m(\perp)) \\ = & \{ \text{ Induktionsvoraussetzung}\} \\ & (\bigsqcup_{n\in\omega} f.n)(\bigsqcup_{n\in\omega}(f.n^m(\perp))) \\ = & \{ \ \sqcup \text{ auf Funktionen punktweise}\} \\ & \bigsqcup_{n\in\omega}(f.n(\bigsqcup_{n\in\omega}(f.n^m(\perp)))) \\ = & \{ \text{ alle } f.n \text{ sind stetig}\} \\ & \bigsqcup_{i\in\omega} \bigsqcup_{n\in\omega} f.i(f.n^m(\perp)) \\ = & \{ \text{ Diagonalisierungslemma Blatt 8/Aufgabe 4a mit } \lambda h.\lambda k.k(h^m(\perp)) \text{ für } g\} \\ & \bigsqcup_{n\in\omega} f.n(f.n^m(\perp)) \\ = & \{ \text{ Def. Funktionsiteration}\} \\ & \bigsqcup_{n\in\omega} f.n^{m+1}(\perp) \end{array}$$

Zu (ii):

Wir haben zu zeigen, daß

$$\forall m \in \omega.(\lambda f.f^m(\perp) \sqsubseteq \lambda f.f^{m+1}(\perp))$$

gilt. Da $\sqsubseteq$ auf Funktionen punktweise definiert ist, zeigen wir die äquivalente Aussage

$$\forall m \in \omega.(\forall f.f^m(\perp) \sqsubseteq f^{m+1}(\perp))$$

durch Induktion über m:

$m = 0$: Sei f eine beliebige stetige Funktion. Dann gilt:

$$\begin{array}{ll} & f^0(\perp) \\ = & \{\ \text{Def. Funktionsiteration}\} \\ & \perp \\ \sqsubseteq & \{\text{kleinstes Element}\} \\ & f^1(\perp) \end{array}$$

$m \rightsquigarrow m+1$: Sei f eine beliebige stetige Funktion. Dann gilt:

$$\begin{array}{ll} & f^{m+1}(\perp) \\ = & \{\ \text{Def. Funktionsiteration}\} \\ & f(f^m(\perp)) \\ \sqsubseteq & \{\ \text{Induktionsvoraussetzung; } f \text{ monoton}\} \\ & f(f^{m+1}(\perp)) \\ = & \{\ \text{Def. Funktionsiteration}\} \\ & f^{m+2}(\perp) \end{array}$$

Zu (iii):

Es gilt:

$$\begin{array}{ll} & \bigsqcup_{m\in\omega}(\lambda f.f^m(\perp)) \\ = & \{\ \sqcup \text{ auf Funktionen punktweise}\} \\ & \lambda f.(\bigsqcup_{m\in\omega}.f^m(\perp)) \end{array}$$

Aufgabe 4 (Lösungsvorschlag)

```
-- Denotationelle cbv-Semantik von REC
-- C.P. 31.1.1997


-- Typdeklarationen

type Vvar = String
type Fvar = Int                     -- Funktionen werden zur Vereinfachung
                                    -- bei 0 beginnend durchnummeriert
```

```
type Venv = Vvar -> Int
type Fenv = [[Int] -> Int]

data Term = Num Int | Var Vvar | Term :+ Term | Term :- Term | Term :* Term
          | If Term Term Term | Fun Fvar [Term]

-- Eine Funktionsdeklaration besteht aus der Argumentliste und
-- dem Term, durch den die Funktion definiert wird
type Decl = ([Vvar],Term)


-- strikte map-Funktion, wird fuer die cbv-Auswertung der Funktions-Applikation
-- benoetigt, da hier die Argumente strikt ausgewertet werden sollen

smap :: (a -> b) -> [a] -> [b]
smap f [] = []
smap f (x:xs) = strict (:) (f x)  (smap f xs)


-- Denotationelle cbv-Semantik

cbv :: Term -> Fenv -> Venv -> Int
cbv (Num n)       = \fenv venv -> n
cbv (Var x)       = \fenv venv -> venv(x)
cbv (t0 :+ t1)    = \fenv venv -> (cbv t0 fenv venv) + (cbv t1 fenv venv)
cbv (t0 :- t1)    = \fenv venv -> (cbv t0 fenv venv) - (cbv t1 fenv venv)
cbv (t0 :* t1)    = \fenv venv -> (cbv t0 fenv venv) * (cbv t1 fenv venv)
cbv (If t0 t1 t2) = \fenv venv ->
                         if (cbv t0 fenv venv) == 0
                            then (cbv t1 fenv venv)
                            else (cbv t2 fenv venv)
cbv (Fun f tlist) = \fenv venv ->
                         let nlist = smap (\t -> cbv t fenv venv) tlist
                             in
                             strict (fenv!!f) nlist

-- anstatt eine strikt map-Funktion zu definieren, kann man auch z.B. so
-- die Auswertung von nlist erzwingen:
--
-- cbv (Fun f tlist) = \fenv venv ->
--                         let nlist0 = map (\t -> cbv t fenv venv) tlist
--                             nlist = if nlist0 == nlist0
--                                     then nlist0 else undefined
--                             in
--                             strict (fenv!!f) nlist


-- polymorphe Hilfsfunktionen "upd" zum aktualisieren eines Funktionswerts,
-- "mupd" zum aktualisieren Liste von Werten
```

```
upd:: Eq a => (a -> b) -> a -> b -> (a -> b)
upd f a0 b0 = \a -> if a0==a then b0 else f a

mupd:: Eq a => (a -> b) -> [a] -> [b] -> (a -> b)
mupd f [] [] = f
mupd f (x:xs) (y:ys) = mupd (upd f x y) xs ys

-- kleinster Fixpunkt

fix :: (a -> a) -> a
fix f = f (fix f)

-- initiales Variablen-Environment
-- das zur Interpretation eines geschlossenen Terms verwendetet Environment
-- kann beliebig gewaehlt werden, wir verwenden ein fuer alle Argumente mit
-- 0 initialisiertes Environment

-- Funktion zur Berechnung des Funktions-Environments

big_f :: [Decl] -> Fenv -> Fenv

big_f dcls = \fenv ->
                 map (\(plist,t) ->
                          (\nlist ->
                               cbv t fenv (mupd venv0 plist nlist)))
                 dcls

venv0 :: Venv
venv0 = \x -> 0

-- Interpretation des Programms

prog_cbv:: [Decl] -> Term -> Int

prog_cbv dcls t = cbv t (fix (big_f dcls)) venv0
```