Material zur Vorlesung "Semantik von Programmiersprachen"

Prof. Gert Smolka und Martin Müller

1 Äquivalenz von F-Termen

Definition. $t\downarrow \quad :\Leftrightarrow \quad t \in \mathbf{GTer} \;\wedge\; \exists c : t \to c$

Ein Kontext C ist aufgebaut wie ein F-Term, der an genau einer Termposition ein Loch enthält. Das Einsetzen eines Terms t in das Loch eines Kontextes C schreiben wir als $C[t]$. Dabei ist das Kapern von Variablen erlaubt und sogar gewollt.

Wir definieren Äquivalenz von F-Termen als die folgende binäre Relation $\sim\, \subseteq \mathbf{Ter} \times \mathbf{Ter}$:

$$t_1 \sim t_2 \quad :\Leftrightarrow \quad \forall C : C[t_1]\downarrow \Leftrightarrow C[t_2]\downarrow$$

Proposition 1.1 *$t_1 \sim t_2$ ist eine Kongruenz auf $\mathbf{Ter} \times \mathbf{Ter}$, d.h. eine Äquivalenzrelation, so daß*

$$\forall C\ \forall t_1\ \forall t_2 : \quad t_1 \sim t_2 \Rightarrow C[t_1] \sim C[t_2]$$

Beweis. Einfach. □

Diese Definition stimmt mit der aus dem Winskel überein mit einem wesentlichen Unterschied: Winskel betrachtet eine getypte Sprache und schränkt die Quantifikation auf solche Kontexte ein, für die $C[t_1]$ und $C[t_2]$ typisierbar sind. Im Unterschied dazu ist F ungetypt, und der Allquantor läuft über beliebige Kontexte.

2 Eine gleichungsdefinierte Kongruenz

α-β-Kongruenz ist die kleinste binäre Relation $\approx\, \subseteq \mathbf{Ter} \times \mathbf{Ter}$, die die folgenden Regeln erfüllt.

$$t \approx t \qquad \frac{t_1 \approx t_2}{t_2 \approx t_1} \qquad \frac{t_1 \approx t_2 \qquad t_2 \approx t_3}{t_1 \approx t_3} \qquad \frac{t_1 \approx t_2}{t[t_1/x] \approx t[t_2/x]}$$

$$(\alpha) \qquad \lambda x.t \approx \lambda y.t[y/x] \qquad (\text{falls } y \text{ frei für } x \text{ in } t)$$

$$(\beta) \qquad (\lambda x.t_1)\ c \approx t_1[c/x] \qquad (c \text{ frei für } x \text{ in } t)$$

$$(\gamma) \qquad \lambda y.((\lambda x.t)\ y) \approx \lambda x.t \qquad (y \text{ nicht frei in } \lambda x.t)$$

Satz 2.1 $t_1 \approx t_2 \Rightarrow t_1 \sim t_2$

(Die Umkehrung ist falsch, da $t_1 \sim t_2$ nicht rekursiv aufzählbar ist.)

Beweis. Der Beweis über die operationale Semantik ist mühsam (aber machbar). Mit der denotationalen Semantik ist er dagegen einfach. Beachte allerdings, daß die denotationale Semantik nur über wohlgetypte Terme (also über TF) spricht. Wir zeigen also für wohlgetypte Terme t_1 und t_2:

$$t_1 \approx t_2 \;\Rightarrow\; [\![t_1]\!]^e = [\![t_2]\!]^e$$

und verwenden den folgenden Satz aus der Vorlesung:

$$[\![t_1]\!]^e = [\![t_2]\!]^e \;\Rightarrow\; t_1 \sim t_2$$

Wenn D eine beliebige VPO ist, so definieren wir eine *partielle* Funktion $\lceil\cdot\rceil : D_\perp \rightharpoonup D$ durch

$$\lceil d \rceil \;=\; \begin{cases} d' & \text{falls } \lfloor d' \rfloor = d \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Nach der Definition von $\lfloor\cdot\rfloor$ (siehe Winskel) ist $\lceil\cdot\rceil$ wohldefiniert; weiter folgt, daß $\lceil\perp\rceil$ undefiniert ist und daß $\forall d \in D : \lceil\lfloor d\rfloor\rceil = d$.

(β) Wir halten zunächst fest, daß für jeden kanonischen Term c der Ausdruck $\lceil[\![c]\!]^e\rho\rceil$ definiert ist (aber nicht für jeden nicht-kanonischen Term).

$$\begin{array}{rcl} [\![\lambda x.t\ c]\!]^e\rho & = & \lceil[\![\lambda x.t]\!]^e\rho\rceil\ \lceil[\![c]\!]^e\rho\rceil \\ & = & (\lambda v.[\![t]\!]^e\rho[v/x])\ \lceil[\![c]\!]^e\rho\rceil \\ & = & [\![t]\!]^e\rho[\lceil[\![c]\!]^e\rho\rceil/x] \\ & = & [\![t[c/x]]\!]^e\rho \end{array}$$

Die letzte Gleichung folgt aus dem Substitutionslemma 11.6 im Winskel.

(γ) Sei x nicht frei in $\lambda y.t$.

$$\begin{array}{rcll} [\![\lambda x.(\lambda y.t)x]\!]^e & = & \lfloor\lambda v.(\lceil[\![\lambda y.t]\!]^e\rho[v/x]\rceil\ \lceil[\![x]\!]^e\rho[v/x]\rceil)\rfloor & \\ & = & \lfloor\lambda v.(\lambda w.[\![t]\!]^e\rho[v/x][w/y]\ v)\rfloor & \\ & = & \lfloor\lambda v.(\lambda w.[\![t]\!]^e\rho[w/y]\ v)\rfloor & (x \text{ nicht frei in } \lambda y.t) \\ & = & \lfloor\lambda v.[\![t]\!]^e\rho[v/y]\rfloor & \\ & = & [\![\lambda y.t]\!]^e & \end{array}$$

□

Lemma 2.2 Falls $t_2 \to c$, so gilt $[\![(\lambda x.t_1)\ t_2]\!]^e = [\![t_1[t_2/x]]\!]^e$ (vorausgesetzt, daß t_2 frei für x in t_1).

Beweis. Aus $t_2 \to c$ folgt $[\![t]\!]^e = [\![c]\!]^e$. Wegen Kompositionalität der denotationalen Semantik und Satz 2.1 folgt daher

$$[\![t_1[t_2/x]]\!]^e = [\![t_1[c/x]]\!]^e = [\![\lambda x.t_1\ c]\!]^e = [\![\lambda x.t_1\ t_2 x]\!]^e$$

□

3 Rückführung von Rekursion auf Selbstanwendung

Satz 3.1 (Rekursion ist simulierbar) $\textbf{rec}\ f.t\ \sim\ (\lambda x.x\ x)\ \lambda f.t[f f/f]$

Beweisidee. Sei $A = \lambda f.t[f f/f]$ geschlossen. Dann $A\ A\ \to_1\ t[A\ A/f]$.

Es gilt aber auch $\textbf{rec}\ f.t\ \to_1\ t[\textbf{rec}\ f.t/f]$.

Also sind wir gewillt zu glauben, daß $A\ A \sim \textbf{rec}\ f.t$ gilt.

Also $\textbf{rec}\ f.t\ \sim\ (\lambda x.x\ x)\ A\ \sim\ (\lambda x.x\ x)\ \lambda f.t[f f/f]$ □

Satz 3.2 (Rekursion ist abstrahierbar)

Sei $R = \textbf{rec}\ r.\lambda g.\lambda x.(g\ (r\ g))\ x$. Dann $\textbf{rec}\ f.t\ \sim\ R\ (\lambda f.t)$.

Beweis.
$$\begin{array}{lcll} [\![R\ (\lambda f.t)]\!]^e & = & [\![(\lambda g.\lambda x.(g\ (R\ g))\ x)\ (\lambda f.t)]\!]^e & \\ & = & [\![\lambda x.((\lambda f.t)\ (R(\lambda f.t)))\ x]\!]^e & (\beta) \\ & = & [\![\lambda x.(t[R\ (\lambda f.t)/f]\ x)]\!]^e & (*) \\ & = & t[R\ (\lambda f.t)/f]) & \end{array}$$

Die mit $(*)$ markierte Gleichheit folgt aus Lemma 2.2 und der Tatsache, daß die Applikation $R(\lambda f.t)$ nach genau einem Schritt konvergiert. Die letzte Gleichheit gilt, weil t nach Syntaxdefinition von F nur eine λ-Abstraktion sein kann.

Also analog zu Satz 3.1 $\textbf{rec}\ f.t\ \sim\ R\ (\lambda f.t)$ □

Satz 3.3 Sei $Y = (\lambda x.x\ x)(\lambda r.\lambda g.\lambda x.(g\ ((r\ r)\ g))\ x)$. Dann $\textbf{rec}\ f.t\ \sim\ Y\ (\lambda f.t)$.

Beweisidee. Satz 3.1 und Satz 3.2. □

4 $\mathbf{F}_0$ ist Turing-vollständig

Für Turing-Vollständigkeit genügt es zu zeigen, daß man in der Lage ist, **Imp** oder Turingmaschinen zu simulieren. Das ist nicht schwer. Wir benötigen

1. Natürliche Zahlen mit Gleichheit, um das Alphabet und die Zustände zu simulieren,
2. Paare, um das Band zu simulieren (2 Stacks), und
3. Konditional und Rekursion, um den Ablauf zu simulieren.

Die Technik zur Simulation von Rekursion haben wir oben kennengelernt. Hier sind die übrigen grundlegenden Datenstrukturen.

4.1 Boolesche Werte

Wir definieren neue Terme durch Makroexpansion $\Rightarrow$ nach F_0.

$$\begin{array}{lll}
\textbf{true} & \Rightarrow & \lambda x.\lambda y.(x\ x) \\
\textbf{false} & \Rightarrow & \lambda x.\lambda y.(y\ y) \\
\textbf{if}\ t_1\ \textbf{then}\ t_2\ \textbf{else}\ t_2 & \Rightarrow & t_1\ (\lambda x.t_2)\ (\lambda x.t_3) \qquad \text{falls } x \text{ nicht frei in } t_2, t_3
\end{array}$$

4.2 Paare

$$\begin{array}{lll}
t_1, t_2 & \Rightarrow & \lambda x.\lambda y.(\lambda z.\textbf{if}\ z\ \textbf{then}\ x\ \textbf{else}\ y)\ t_1\ t_2 \\
\textbf{fst}\ t & \Rightarrow & t\ \textbf{true} \\
\textbf{snd}\ t & \Rightarrow & t\ \textbf{false}
\end{array}$$

4.3 Natürliche Zahlen

$$\begin{array}{lll}
0 & \Rightarrow & \lambda x.x \\
+1 & \Rightarrow & \lambda x.(\textbf{false}, x) \\
-1 & \Rightarrow & \lambda x.\textbf{snd}\ x \\
=0? & \Rightarrow & \lambda x.x\ \textbf{true}
\end{array}$$

4.4 Gleichheit

Wir schreiben die Applikationen =0? x, +1 x und -1 x als $x = 0$, $(x + 1)$ bzw. $(x - 1)$.

$$\begin{array}{llll}
eq & \Rightarrow & \textbf{rec}\ eq.\lambda x.\lambda y. & \textbf{if}\ x = 0\ \textbf{then}\ y = 0 \\
 & & & \textbf{else if}\ y = 0\ \textbf{then false} \\
 & & & \textbf{else}\ eq\ (x-1)\ (y-1)
\end{array}$$

5 Eine Einschrittsemantik für F

Regelname	Regel		Analoge Bildung
(appl-1)	$\dfrac{}{(\lambda x.t)\ c\ \rightarrow_1\ t[c/x]}$		
(appl-l-1)	$\dfrac{t_1\ \rightarrow_1\ t_1'}{t_1\ t_2\ \rightarrow_1\ t_1'\ t_2}$		(appl-r-1)
(add-1)	$\dfrac{}{n_1 + n_2\ \rightarrow_1\ n}$	falls $n = n_1 + n_2$	
(add-l-1)	$\dfrac{t_1\ \rightarrow_1\ t_1'}{t_1 + t_2\ \rightarrow_1\ t_1' + t_2}$		(add-r-1)
(,-l-1)	$\dfrac{t_1\ \rightarrow_1\ t_1'}{t_1, t_2\ \rightarrow_1\ t_1', t_2}$		(,-r-1)
(fst-r-1)	$\dfrac{t \rightarrow t'}{\mathbf{fst}\ t\ \rightarrow_1\ \mathbf{fst}\ t'}$		(snd-r-1)
(fst-l-1)	$\dfrac{}{\mathbf{fst}\ (c_1, c_2)\ \rightarrow_1\ c_1}$		(snd-l-1)
($\leq$-**false**-1)	$\dfrac{}{n_1 \leq n_2\ \rightarrow_1\ \mathbf{false}}$	falls $n_1 > n_2$	($\leq$-**true**-1)
($\leq$-l-1)	$\dfrac{t_1\ \rightarrow_1\ t_1'}{t_1 \leq t_2\ \rightarrow_1\ t_1' \leq t_2}$		
($\leq$-r-1)	$\dfrac{t_2\ \rightarrow_1\ t_2'}{t_1 \leq t_2\ \rightarrow_1\ t_1 \leq t_2'}$		
(inl-1)	$\dfrac{t\ \rightarrow_1\ t'}{\mathbf{inl}\ t\ \rightarrow_1\ \mathbf{inl}\ t'}$		(inr-1)
(case-inl-1)	$\dfrac{}{\mathbf{case}\ (\mathbf{inl}\ c)\ \mathbf{of}\ \mathbf{inl}\ x_1.t_1\ []\ \mathbf{inr}\ x_2.t_2\ \rightarrow_1\ t_1[c/x_1]}$		(case-inr-1)
(case-?-1)	$\dfrac{t\ \rightarrow_1\ t'}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}\ x_1.t_1\ []\ \mathbf{inr}\ x_2.t_2\ \rightarrow_1\ \mathbf{case}\ t'\ \mathbf{of}\ \mathbf{inl}\ x_1.t_1\ []\ \mathbf{inr}\ x_2.t_2}$		
(if-true-1)	$\dfrac{}{\mathbf{if\ true\ then}\ t_1\ \mathbf{else}\ t_2\ \rightarrow_1\ t_1}$		(if-false-1)
(if-?-1)	$\dfrac{t\ \rightarrow_1\ t'}{\mathbf{if}\ t\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2\ \rightarrow_1\ \mathbf{if}\ t'\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2}$		
(rec-1)	$\dfrac{}{\mathbf{rec}\ x.t\ \rightarrow_1\ t[\mathbf{rec}\ x.t/x]}$		

Satz 5.1 $\forall t\ \forall c\ t \rightarrow_1^* c\ \Leftrightarrow\ t \rightarrow c$

Beweis. Einfach.