03.11.05

Theorie der Programmierung I

(Mitschrift von Rudolf Koslowski)

Nachtrag:
Damit die small step Semantik tatsächlich deterministisch ist, legt man bei der Definition der Substitution auf einen bestimmten neuen Namen fest.
Weil jeder einzelne small step deterministisch ist, gibt es für jeden Ausdruck e genau eine Berechnung.

Rekursion: Bisher: Mit λ-Abstraktion können wir (namenlose) Funktionen hinschreiben, z.B. λx. x * x. Mit let können wir Namen für Werte, also insbesondere für Funktionen, einführen.

Man beachte: Die beiden Konzepte „Funktionen" (mit Parametern, Parameterübergabe, …) und „Deklaration" sind sauber getrennt. Vorteil (zumindest für die Theorie): Einfach Regeln für die Semantik und für Typüberprüfung. Diese saubere Trennung soll beibehalten werden. Deshalb: Eigenständige Schreibweise für rekursive Funktionen.

Formal: Die KFG für $\mathcal{L}_1$ wird erweitert durch eine Produktion e ::= rec id. e_1 (rekursiver Ausdruck)

Erweiterte Sprache: L2

Intuition: rec id. e steht für das Element (mit Namen id), das durch die rekursive Gleichung id = e gegeben ist.

Beispiel: rec fact. λ x. if x = 0 then 1 else x * fact (x-1)
Erlaubt sind Ausdrücke, die keine Funktion sind, z.B. rec x. x+1.

Syntaktische Zucker: 1) let rec id = e_1 in e_2 steht für: let id = rec id. e_1 in e_2

Beispiel: let rec fact = λx. if x = 0 then 1 else x * fact (x-1) in fact 3
| (entspricht O'Camel (in SML: let val rec fact = …))
steht für let fact = rec fact λx. if … in fact 3

2) let rec id id_1, …, id_n = e_1 in e_2 steht für let rec id = λid_1, … λid_n e_1 in e_2

Beispiel: let rec fact x = if x = 0 then 1 else x * fact (x-1) steht für let rec fact = λx. …
let rec exp x n = if n = 0 then 1 else x * exp x (n-1) steht für let rec exp = λx. λn. …
(vgl. zu SML: Man schreibt let fun id id_1, …, id_n = e_1 in e_2)

Small step Semantik: Die Definition von free(e) und e'[e/id] wird auf rekursive Ausdrücke erweitert: rec ist (wie λ) ein Bindungsmechanismus und wird deshalb ganz analog behandelt.
free(rec id e) = free(e) \ {id}

$$(\text{rec id' } e_1)\ [e^*/id^*] = \begin{cases} \text{rec id } e_1 \text{ falls id = id'} \\ \text{rec id'' } e_1\ [\text{id''/id'}][e/\text{id}] \text{ mit id'' } \notin \\ \text{free(rec id' } e_1) \cup \text{free(e)} \cup \{\text{id}\} \text{ sonst} \end{cases}$$

An der Definition von Werten v wird nichts geändert. Die small step Semantik wir durch eine neue Regel erweitert:
(UNFOLD) rec id. e → e [rec id e/id]

Intuition: Für jedes freie Vorkommen von id in e (d.h. für jeden „rekursivn Aufruf von id") setzt man die gesamte rekursive Funktion ein.

Warum führt das nicht immer zur Endlos-Schleife? Üblicherweise ist e eine λ-Abstraktion, also ist auch e [rec id e/id] eine λ-Abstraktion und die ist vor weiterer Auswertung

„geschützt“. (Mit anderen Worten: Die weitere Auswertung findet erst dann statt, wenn ein aktueller Parameter vorhanden ist.)

Beispiel: let rec fact = λx. if x=0 then 1 else x ∗ fact(x-1) in fact 1
→ (UNFOLD) let fact = λx. if x=0 then 1 else x ∗ (rec fact …) (x-1) in fact 1
→ (LET-EXEC) λx. if x=0 then 1 else x ∗ (rec fact …) (1-1)
→ (BETA-V) if 1=0 then 1 else 1 ∗ (rec fact …) (1-1)
→ (OP) if false then 1 else 1 ∗ (rec fact …) (1-1)
→ (COND-FALSE) 1 ∗ (rec fact …) (1-1)
| an dieser Stelle weiter auswerten
→ (UNFOLD) 1 ∗ λx. if x=0 then 1 else x ∗ (rec fact …) (1-1)
→ (OP) 1 ∗ λx. if x=0 then 1 else x ∗ (rec fact …) 0
→ (BETA-V) 1 ∗ if 0=0 then 1 else …
→ (OP) 1 ∗ if true then 1 else …
→ (COND-TRUE) 1 ∗ 1
→ (OP) 1

Bemerkung: Satz 2 gilt nach wie vor. Für jeden Ausdruck e existiert höchstens ein r ∈ Exp ∪ Exn mit e→r (denn: im Beweis kommt als einziger neuer Fall der rec-Ausdruck hinzu. Für rec id e kommt nur die Regel (UNFOLD) in Frage, und die führt zu einem eindeutigen Ergebnis.)
Lemma 3 gilt auch noch, wenn e→e’, dann ist free(e’) ⊆ free(e)

Big step Semantik:

Orientiert sich weniger an den einzelnen Umformungsschritten, sondern mehr am **Endergebnis** einer Berechnung.

Formal: Ein big step ist von der Form $e \Downarrow v$ oder $e \Downarrow exn$

Intuition: $e \Downarrow r$ soll bedeuten, e terminiert mit Resultat r (∈ Exp ∪ Exn)

Ein big step heißt **gültig**, wenn er sich mit den folgenden Regeln herleiten lässt:

(VAL) $v \Downarrow v$

(OP) $op\ n_1\ n_2 \Downarrow op^I(n_1\ n_2)$
| ∈ Val ∪ Exn

(BETA-V) $\dfrac{e[v/id] \Downarrow v'}{(\lambda\ id.\ e)\ v \Downarrow v'}$

(APP) $\dfrac{e_1 \Downarrow v_1\ e_2 \Downarrow v_2\ v_1 v_2 \Downarrow v}{e_1 e_2 \Downarrow v}$