UNIVERSITÄT SIEGEN
FACHBEREICH MATHEMATIK

Diplomarbeit

Theoretische Grundlagen der Objektorientierung

Benedikt Meurer
1. März 2007

INTERNE BERICHTE
INTERNAL REPORTS

Diplomarbeit am
Fachbereich Mathematik
der Universität Siegen

Betreuer:
Privatdozent Dr. Kurt Sieber
Prof. Dr. Dieter Spreen

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Benedikt Meurer)

Zusammenfassung

TODO

Inhaltsverzeichnis

Kapitel 1

Einleitung

Narf, [Rém02, 50f].

Kapitel 2

Funktionale Objekte

2.1 Syntax der Sprache $\mathcal{L}_o$

Menge *Op*:

$$\begin{array}{lcll} op & ::= & + \mid - \mid * & \text{arithmetische Operatoren} \\ & \mid & < \mid > \mid \leq \mid \geq \mid = & \text{Vergleichsoperatoren} \end{array}$$

Menge *Const*:

$$\begin{array}{lcll} c & ::= & () & \text{unit-Element} \\ & \mid & b & \text{boolscher Wert} \\ & \mid & n & \text{Ganzzahl} \\ & \mid & op & \text{binärer Operator} \end{array}$$

Menge *Exp*:

$$\begin{array}{lcll} e & ::= & c & \text{Konstante} \\ & \mid & id & \text{Name} \\ & \mid & e_1\, e_2 & \text{Applikation} \\ & \mid & \lambda id.e_1 & \text{Abstraktion} \\ & \mid & \textbf{rec}\, id.e_1 & \text{Rekursion} \\ & \mid & \textbf{let}\, id = e_1\, \textbf{in}\, e_2 & \text{let-Ausdruck} \\ & \mid & \textbf{if}\, e_0\, \textbf{then}\, e_1\, \textbf{else}\, e_2 & \text{bedingter Ausdruck} \\ & \mid & e_1\#m & \text{Methodenaufruf} \\ & \mid & \textbf{object}\,(id)\; r\; \textbf{end} & \text{Objekt} \\ & \mid & \{\langle id_1 = e_1; \ldots; id_n = e_n \rangle\} & \text{Duplikation} \end{array}$$

Menge *Row*:

$$\begin{array}{lcll} r & ::= & \epsilon & \text{leere Reihe} \\ & \mid & \textbf{val}\; id = e;\, r_1 & \text{Attribut} \\ & \mid & \textbf{method}\; m = e;\, r_1 & \text{Methode} \end{array}$$

Zur Definition der small step Semantik wird darüber hinaus noch der Hilfsausdruck

$$e \quad ::= \quad r\#m$$

benötigt.

Definition 1 $dom\,(r)$ definieren.

2.1.1 Syntaktischer Zucker

$e_1\;op\;e_2$	für	$op\;e_1\;e_2$	Infixnotation
$-\,e$	für	$0 - e$	unäres Minus

2.2 Operationelle Semantik der Sprache $\mathcal{L}_o$

Für jeden Operator $op \in Op$ sei eine Funktion op^I vorgegeben mit

$$\begin{array}{ll} op^I : Int \times Int \to Int & \text{falls } op \in \{+, -, *\} \\ op^I : Int \times Int \to Bool & \text{sonst.} \end{array}$$

Die Funktion op^I heisst *Interpretation* des Operators op. Auf eine exakte Definition wird hier verzichtet. Im Folgenden nehmen wir an, dass diese Funktionen gemäß ihrer intuitiven Bedeutung definiert sind.

$Val_r \subseteq Row$ sei die Menge aller Reihen ω mit den folgenden Eigenschaften:

- jede Attributdeklaration in ω ist von der Form **val** $id = v; \omega'$ mit $\omega' \in Val_r$ und

- die Namen der Attribute in ω sind paarweise verschieden.

Menge $Val_e \subseteq Exp$:

$$\begin{array}{lll} v & ::= & c \\ & | & id \\ & | & op\,v_1 \\ & | & \lambda id.e \\ & | & \textbf{object}\,(id)\;\omega\;\textbf{end} \end{array}$$

2.2.1 Frei vorkommende Namen und Substitution

Definition 2 (Frei vorkommende Namen) Die Mengen $\mathit{free}\,(e)$ aller im Ausdruck e und $\mathit{free}\,(r)$ aller in der Reihe r *frei vorkommenden Namen* werden wie folgt induktiv über die Größen von e und r definiert.

$$
\begin{array}{rcl}
\mathit{free}\,(c) & = & \emptyset \\
\mathit{free}\,(id) & = & \{id\} \\
\mathit{free}\,(e_1\,e_2) & = & \mathit{free}\,(e_1) \cup \mathit{free}\,(e_2) \\
\mathit{free}\,(\lambda id.e) & = & \mathit{free}\,(e) \setminus \{id\} \\
\mathit{free}\,(\mathbf{rec}\,id.e) & = & \mathit{free}\,(e) \setminus \{id\} \\
\mathit{free}\,(\mathbf{let}\,id = e_1\,\mathbf{in}\,e_2) & = & \mathit{free}\,(e_1) \cup (\mathit{free}\,(e_2) \setminus \{id\}) \\
\mathit{free}\,(\mathbf{if}\,e_0\,\mathbf{then}\,e_1\,\mathbf{else}\,e_2) & = & \mathit{free}\,(e_0) \cup \mathit{free}\,(e_1) \cup \mathit{free}\,(e_2) \\
\\
\mathit{free}\,(r\#m) & = & \mathit{free}\,(r) \\
\mathit{free}\,(e\#m) & = & \mathit{free}\,(e) \\
\mathit{free}\,(\mathbf{object}\,(id)\;r\;\mathbf{end}) & = & \mathit{free}\,(r) \setminus \{id\} \\
\mathit{free}\,(\{\langle id_1 = e_1; \ldots; id_n = e_n\rangle\}) & = & \bigcup_{i=1}^{n} \mathit{free}\,(e_i) \\
\\
\mathit{free}\,(\epsilon) & = & \emptyset \\
\mathit{free}\,(\mathbf{val}\;id = e;\;r) & = & \mathit{free}\,(e) \cup \big(\mathit{free}\,(r) \setminus \{id\}\big) \\
\mathit{free}\,(\mathbf{method}\;m = e;\;r) & = & \mathit{free}\,(e) \cup \mathit{free}\,(r)
\end{array}
$$

Definition 3 Ein Ausdruck e heisst *abgeschlossen*, wenn $\mathit{free}\,(e) = \emptyset$.

Es ist leicht zu sehen, dass gemäß dieser Definition, ein Ausdruck genau dann abgeschlossen ist, wenn er keine freien Vorkommen von Namen enthält. Zum Beispiel sind im Ausdruck

$$f\,(x+1)$$

die beiden Vorkommen der Namen f und x frei, der Ausdruck also nicht abgeschlossen. Erweitert man hingegen den Ausdruck zu

$$\mathbf{let}\,f = \lambda y.y * y\,\mathbf{in}\,\mathbf{let}\,x = 2\,\mathbf{in}\,f\,(x+1)$$

so werden nun beide zuvor freien Vorkommen von Namen durch **let**-Ausdrücke gebunden und der Gesamtausdruck ist abgeschlossen.

Definition 4 (Substitution) Der Ausdruck $e[e'/id]$, der aus e durch *Substitution* von e' für id entsteht, und die Reihe $r[e'/id]$, die aus r durch *Substitution* von e' für id entsteht, sind durch Induktion über die Größen von e und r wie folgt definiert.

$$
\begin{array}{rcl}
c[e'/id] & = & c \\
id'[e'/id] & = & \begin{cases} e' & \text{falls } id = id' \\ id' & \text{sonst} \end{cases} \\
(e_1\, e_2)[e'/id] & = & (e_1[e'/id])\,(e_2[e'/id]) \\
(\lambda id'.e)[e'/id] & = & \begin{cases} \lambda id'.e & \text{falls } id = id' \\ \lambda id''.e[id''/id'][e'/id] & \text{sonst} \end{cases} \\
 & & \text{mit } id'' \notin \mathit{free}\,(\lambda id'.e) \cup \{id\} \cup \mathit{free}\,(e') \\
(\textbf{rec}\, id'.e)[e'/id] & = & \begin{cases} \textbf{rec}\, id'.e & \text{falls } id = id' \\ \textbf{rec}\, id''.e[id''/id'][e'/id] & \text{sonst} \end{cases} \\
 & & \text{mit } id'' \notin \mathit{free}\,(\textbf{rec}\, id'.e) \cup \{id\} \cup \mathit{free}\,(e') \\
(\textbf{let}\, id' = e_1\, \textbf{in}\, e_2)[e'/id] & = & \begin{cases} \textbf{let}\, id' = e_1\, \textbf{in}\, e_2 & \text{falls } id = id' \\ \textbf{let}\, id'' = e_1[e'/id]\, \textbf{in}\, e_2[id''/id'][e'/id] & \text{sonst} \end{cases} \\
 & & \text{mit } id'' \notin \mathit{free}\,(\lambda id'.e_2) \cup \{id\} \cup \mathit{free}\,(e') \\
(\textbf{if}\, e_0\, \textbf{then}\, e_1\, \textbf{else}\, e_2)[e'/id] & = & \textbf{if}\, e_0[e'/id]\, \textbf{then}\, e_1[e'/id]\, \textbf{else}\, e_2[e'/id] \\
 & & \\
(r\#m)[e'/id] & = & (r[e'/id])\#m \\
(e\#m)[e'/id] & = & (e[e'/id])\#m \\
(\textbf{object}\,(id')\ r\ \textbf{end})[e'/id] & = & \begin{cases} \textbf{object}\,(id')\ r\ \textbf{end} & \text{falls } id = id' \\ \textbf{object}\,(id'')\ r[id''/id'][e'/id]\ \textbf{end} & \text{sonst} \end{cases} \\
 & & \text{mit } id'' \notin (\mathit{free}\,(r) \setminus \{id'\}) \cup \{id\} \cup \mathit{free}\,(e') \\
\{\langle id_i = e_i{}^{1\le i\le n}\rangle\}[e'/id] & = & \begin{cases} \textbf{object}\,(id)\ \omega\langle id_i = e_i[e'/id]^{1\le i\le n}\rangle\ \textbf{end} \\ \quad \text{falls } e' = \textbf{object}\,(id)\ \omega\ \textbf{end} \\ \{\langle id_i = e_i[e'/id]^{1\le i\le n}\rangle\} \\ \quad \text{sonst} \end{cases} \\
\epsilon[e'/id] & = & \epsilon \\
(\textbf{val}\ id' = e;\ r)[e'/id] & = & \begin{cases} \textbf{val}\ id' = e;\ r & \text{falls } id = id' \\ \textbf{val}\ id'' = e[e'/id];\ r[id''/id'][e'/id] & \text{sonst} \end{cases} \\
 & & \text{mit } id'' \notin (\mathit{free}\,(r) \setminus \{id'\}) \cup \{id\} \cup \mathit{free}\,(e') \\
(\textbf{method}\ m = e;\ r)[e'/id] & = & \textbf{method}\ m = e[e'/id];\ r[e'/id]
\end{array}
$$

Die Reihe $\omega\langle id_1 = e_1; \ldots; id_n = e_n\rangle$ im ersten Fall der Anwendung Substitution auf eine Duplikation entsteht aus ω durch Einsetzung der Ausdrücke $e_1, \ldots, e_n$ auf den rechten Seiten der Attribute $id_1, \ldots, id_n$. Die exakte Definition dieser Reiheneinsetzung folgt später.

Gebundene Umbenennung oder *alpha@α-Konversion* (**TODO:** guter Literaturverweis), Idee beim ersten Fall der Anwendung einer Substitution auf eine Duplikation u.a. Zusammenhang zwischen id und **object** $(id)\ \omega$ **end** bei der Substitution.

Definition 5 (Umbenennung) $e\{e'/id\}$ und $r\{e'/id\}$ definieren.

Definition 6 (Reiheneinsetzung) Die Reihe $r\langle id_1 = e_1; \ldots; id_n = e_n\rangle$ entsteht aus r durch *Reiheneinsetzung*, indem die rechten Seiten der Attribute $id_1, \ldots, id_n$ durch die Ausdrücke $e_1, \ldots, e_n$ ersetzt werden, ist durch Induktion über die Größe von r wie folgt definiert.

$$\begin{aligned}
\epsilon\langle id_i = e_i{}^{1\leq i\leq n}\rangle &= \epsilon \\
(\mathbf{method}\ m = e;\ r)\langle id_i = e_i{}^{1\leq i\leq n}\rangle &= \mathbf{method}\ m = e;\ r\langle id_i = e_i{}^{1\leq i\leq n}\rangle \\
(\mathbf{val}\ id = e;\ r)\langle id_i = e_i{}^{1\leq i\leq n}\rangle &= \begin{cases} \mathbf{val}\ id' = e;\ r\{id'/id\}\langle id_i = e_i{}^{1\leq i\leq n}\rangle \\ \text{ falls } id \notin \{id_1, \ldots, id_n\} \\ \mathbf{val}\ id' = e_j;\ r\{id'/id\}\langle id_i = e_i{}^{1\leq i\leq n \wedge i\neq j}\rangle \\ \text{ falls } \exists 1 \leq j \leq n : id = id_j \end{cases} \\
&\quad\ \text{mit } id' \notin (\mathit{free}\,(r) \setminus \{id\}) \cup \bigcup_{i=1}^{n} \mathit{free}\,(e_i)
\end{aligned}$$

Bei der Einsetzung in Attributdeklarationen muss gegebenenfalls eine gebundene Umbenennung durchgeführt werden, da die Ausdrücke $e_1, \ldots, e_n$ nicht unbedingt abgeschlossen sind und durch naive Einsetzung dieser Ausdrücke, frei vorkommende Namen durch zuvor in der Reihe deklarierte Attribute neu gebunden werden könnten. Hierbei ist zu beachten, dass, falls id nicht frei in $e_1, \ldots, e_n$ vorkommt, keine Umbenennung durchgeführt werden muss.

Beispiel 1 Betrachten wir die Reiheneinsetzung

$$(\mathbf{val}\ x = 1;\ \mathbf{val}\ y = 2;\ \epsilon)\langle y = x\rangle$$

so würde bei einer naiven Ersetzung von 2 durch x die Reihe

$$\mathbf{val}\ x = 1;\ \mathbf{val}\ y = x;\ \epsilon$$

entstehen. Das zuvor freie Vorkommen von x wäre nun an das Attribut x gebunden, was in der Regel zu einer Änderung der Semantik des Gesamtausdrucks führen würde. Gebundene Umbenennung des Attributs x liefert das gewünschte Ergebnis

$$\mathbf{val}\ x' = 1;\ \mathbf{val}\ y = x;\ \epsilon.$$

2.2.2 Small step Semantik

Definition 7 Ein *small step* ist eine Formel der Gestalt $e \rightarrow_e e'$ mit $e, e' \in$ *Exp* oder $r \rightarrow_r r'$ mit $r, r' \in$ *Row*.

Definition 8 (Gültige small steps) Ein small step, $e \rightarrow_e e'$ mit $e, e' \in$ *Exp* oder $r \rightarrow_r r'$ mit $r, r' \in$ *Row*, heißt *gültig*, wenn er sich mit den *small step Standard-Regeln*

(OP)	$op\, n_1\, n_2 \rightarrow_e op^I(n_1, n_2)$
(BETA-V)	$(\lambda id.e)\, v \rightarrow_e e[v/id]$
(APP-LEFT)	$\dfrac{e_1 \rightarrow_e e_1'}{e_1\, e_2 \rightarrow_e e_1'\, e_2}$
(APP-RIGHT)	$\dfrac{e_2 \rightarrow_e e_2'}{v_1\, e_2 \rightarrow_e v_1\, e_2'}$
(UNFOLD)	$\mathbf{rec}\, id.e \rightarrow_e e[\mathbf{rec}\, id.e/id]$
(LET-EVAL)	$\dfrac{e_1 \rightarrow_e e_1'}{\mathbf{let}\, id = e_1\, \mathbf{in}\, e_2 \rightarrow_e \mathbf{let}\, id = e_1'\, \mathbf{in}\, e_2}$
(LET-EXEC)	$\mathbf{let}\, id = v_1\, \mathbf{in}\, e_2 \rightarrow_e e_2[v_1/id]$
(COND-EVAL)	$\dfrac{e_0 \rightarrow_e e_0'}{\mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 \rightarrow_e \mathbf{if}\, e_0'\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2}$
(COND-TRUE)	$\mathbf{if}\, true\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 \rightarrow_e e_1$
(COND-FALSE)	$\mathbf{if}\, false\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 \rightarrow_e e_2$

sowie den folgenden *small step Regeln für Objekte*

(OBJECT-EVAL) $\dfrac{r \to_r r'}{\textbf{object}\,(id)\ r\ \textbf{end} \to_e \textbf{object}\,(id)\ r'\ \textbf{end}}$

(SEND-EVAL) $\dfrac{e \to_e e'}{e\#m \to_e e'\#m}$

(SEND-UNFOLD) $(\textbf{object}\,(id)\ r\ \textbf{end})\#m \to_e (r[\textbf{object}\,(id)\ r\ \textbf{end}/id])\#m$

(SEND-ATTR) $(\textbf{val}\ id = v;\ \omega)\#m \to_e (\omega[v/id])\#m$

(SEND-SKIP) $(\textbf{method}\ m' = e;\ \omega)\#m \to_e \omega\#m$
falls $m' \neq m \vee (\textbf{method} : m) \in dom\,(\omega)$

(SEND-EXEC) $(\textbf{method}\ m' = e;\ \omega)\#m \to_e e$
falls $m' = m \wedge (\textbf{method} : m) \notin dom\,(\omega)$

und den *small step Regeln für Reihen*

(ATTR-LEFT) $\dfrac{e \to_e e'}{\textbf{val}\ id = e;\ r \to_r \textbf{val}\ id = e';\ r}$

(ATTR-RIGHT) $\dfrac{r \to_r r'}{\textbf{val}\ id = v;\ r \to_r \textbf{val}\ id = v;\ r'}$
falls $(\textbf{attr} : id) \notin dom\,(r)$

(ATTR-RENAME) $\textbf{val}\ id = v;\ r \to_r \textbf{val}\ id' = v;\ r\{id'/id\}$
falls $(\textbf{attr} : id) \in dom\,(r)$

(METHOD-RIGHT) $\dfrac{r \to_r r'}{\textbf{method}\ m = e;\ r \to_r \textbf{method}\ m = e;\ r'}$

herleiten lässt.

Die Auswertung eines Ausdrucks beschreibt man als eine Aneinanderreihung gültiger small steps, die sich mit den small step Regeln herleiten lassen. Die folgende Definition beschreibt diesen Zusammenhang.

Definition 9 (Berechnung)

(a) Eine *Berechnungsfolge* ist eine endliche oder unendliche Folge von small steps $e_1 \to e_1 \to \ldots$.

(b) Eine *Berechnung* des Ausdrucks e ist eine *maximale*, mit e beginnende Berechnungsfolge, d.h., eine Berechnungsfolge, die sich nicht weiter fortsetzen lässt.

Im Folgenden betrachten wir $\rightarrow_e$ als eine Relation

$$\rightarrow_e \subseteq Exp \times Exp$$

und $\rightarrow_r$ als eine Relation

$$\rightarrow_r \subseteq Row \times Row$$

und benutzen die Infixnotation $e \rightarrow_e e'$ anstelle von $(e, e') \in \rightarrow_e$ und $r \rightarrow_r r'$ anstelle von $(r, r') \in \rightarrow_r$. Statt $\rightarrow_e$ und $\rightarrow_r$ schreiben wir auch $\rightarrow$, wenn wir uns auf beide Relationen beziehen oder es aus dem Zusammenhang hervorgeht, welche Relation gemeint ist.

Wir schreiben $\xrightarrow{+}$ für den transitiven und $\xrightarrow{*}$ für den reflexiven transitiven Abschluss der Reltation $\rightarrow$, d.h. $\xrightarrow{+}$ bildet eine endliche und $\xrightarrow{*}$ bildet eine nicht-leere endliche Folge von small steps.

Ferner schreiben wir $e \not\rightarrow_e$, wenn kein e' existiert, so dass $(e, e') \in \rightarrow_e$, und $r \not\rightarrow_r$, wenn kein r' existiert, so dass $(r, r') \in \rightarrow_r$. Damit sind wir nun in der Lage die erste einfache Eigenschaft der small step Semantik zu formulieren.

Lemma 1

(a) $v \not\rightarrow$ für alle $v \in Val_e$.

(b) $\omega \not\rightarrow$ für alle $\omega \in Val_r$.

Beweis: Der Beweis ergibt sich durch simultane vollständige Induktion über die Grösse von Werten v und Reihenwerten ω: Für Konstanten, Namen, Abstraktionen und die leere Reihe ist die Behauptung unmittelbar klar, da diese weder in den Axiomen noch in den Konklusionen der Regeln links von $\rightarrow$ stehen.

Für $v = op\, v'$ kommt nur ein small step mit (APP-LEFT) oder (APP-RIGHT) in Frage. Dann müsste aber in der Prämisse dieser Regel ein small step für op oder v' stehen, was nach Induktionsvorraussetzung nicht möglich ist.

Im Fall von $v = \textbf{object}\,(id)\ \omega'\ \textbf{end}$ kommt nur ein small step mit (OBJECT-EVAL) in Frage. Nach Induktionsvorraussetzung existiert aber kein small step für ω.

Für $\omega = \textbf{val}\ id = v';\ \omega'$ kommt nur ein small step mit (ATTR-LEFT), (ATTR-RIGHT) oder (ATTR-RENAME) in Frage. Nach Induktionsvoraussetzung existiert weder für ω' noch für v' ein small step, die Regeln (ATTR-LEFT) und (ATTR-RIGHT) können folglich nicht angewandt werden. Nach Definition müssen die Attributnamen in ω paarweise verschieden sein, also

$(\mathbf{attr} : id) \notin dom\,(\omega')$ gelten, was die Anwendung von (ATTR-RENAME) ausschliesst.

Es bleibt der Fall $\omega = \mathbf{method}\ m = e; \omega'$, in dem nur ein small step mit (METHOD-RIGHT) in Frage kommt. Nach Induktionsvoraussetzung existiert aber kein small step für ω'. □

TODO: Überleitung.

Satz 1 (Eindeutigkeit des Übergangsschritts) *Für jeden Ausdruck e existiert höchstens ein $e' \in Exp$ mit $e \to e'$ und für jede Reihe r existiert höchstens ein $r' \in Row$ mit $r \to r'$.*

Beweis: Simultane vollständige Induktion über die Grösse von Ausdrücken e und Reihen r. □

Korollar 1

(a) Für jeden Ausdruck e existiert genau eine Berechnung.

(b) Für jede endliche Berechnung $e_1 \to \ldots \to e_n$ ist das Resultat e_n eindeutig durch e_1 bestimmt.

Beweis: Folgt trivialerweise aus der Eindeutigkeit des Übergangsschritts. □

TODO: Was bringt das nun?

2.2.3 Big step Semantik

TODO: Einführung.

Definition 10 Ein *big step* ist eine Formel der Form $e \Downarrow_e v$ mit $e \in Exp, v \in Val_e$ oder $r \Downarrow_r \omega$ mit $r \in Row, \omega \in Val_r$.

Definition 11 (Gültige big steps) Ein big step, $e \Downarrow_e v$ mit $e \in Exp, v \in Val_e$ oder $r \Downarrow_r \omega$ mit $r \in Row, \omega \in Val_r$, heisst *gültig*, wenn er sich mit den *big step Standard-Regeln*

(VAL)	$v \Downarrow_e v$
(OP)	$op\, n_1\, n_2 \Downarrow_e op^I(n_1, n_2)$
(BETA-V)	$\dfrac{e[v/id] \Downarrow_e v'}{(\lambda id.e)\, v \Downarrow_e v'}$
(APP)	$\dfrac{e_1 \Downarrow_e v_1 \quad e_2 \Downarrow_e v_2 \quad v_1\, v_2 \Downarrow_e v}{e_1\, e_2 \Downarrow_e v}$
(UNFOLD)	$\dfrac{e[\mathbf{rec}\, id.e/id] \Downarrow_e v}{\mathbf{rec}\, id.e \Downarrow_e v}$
(LET)	$\dfrac{e_1 \Downarrow_e v_1 \quad e_2[v_1/id] \Downarrow_e v_2}{\mathbf{let}\, id = e_1\, \mathbf{in}\, e_2 \Downarrow_e v_2}$
(COND-TRUE)	$\dfrac{e_0 \Downarrow_e true \quad e_1 \Downarrow_e v}{\mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 \Downarrow_e v}$
(COND-FALSE)	$\dfrac{e_0 \Downarrow_e false \quad e_2 \Downarrow_e v}{\mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 \Downarrow_e v}$

sowie den folgenden *big step Regeln für Objekte*

(OBJECT)	$\dfrac{r \Downarrow_r \omega}{\mathbf{object}\,(id)\ r\ \mathbf{end} \Downarrow_e \mathbf{object}\,(id)\ \omega\ \mathbf{end}}$
(SEND)	$\dfrac{e \Downarrow_e \mathbf{object}\,(id)\ \omega\ \mathbf{end} \quad \omega[\mathbf{object}\,(id)\ \omega\ \mathbf{end}/id]\#m \Downarrow_e v}{e\#m \Downarrow_e v}$
(SEND-ATTR)	$\dfrac{\omega[v/id]\#m \Downarrow_e v'}{(\mathbf{val}\ id = v;\ \omega)\#m \Downarrow_e v'}$
(SEND-SKIP)	$\dfrac{m' \neq m \vee (\mathbf{method} : m) \in dom\,(\omega) \quad \omega\#m \Downarrow_e v}{(\mathbf{method}\ m' = e;\ \omega)\#m \Downarrow_e v}$
(SEND-EXEC)	$\dfrac{m' = m \wedge (\mathbf{method} : m) \notin dom\,(\omega) \quad e \Downarrow_e v}{(\mathbf{method}\ m' = e;\ \omega)\#m \Downarrow_e v}$

und den *big step Regeln für Reihen*

(OMEGA)	$\omega \Downarrow_r \omega$
(ATTR)	$\dfrac{(\mathbf{attr} : id) \notin dom\,(r) \quad e \Downarrow_e v \quad r \Downarrow_r \omega}{\mathbf{val}\ id = e;\ r \Downarrow_r \mathbf{val}\ id = v;\ \omega}$
(RENAME)	$\dfrac{(\mathbf{attr} : id) \in dom\,(r) \quad e \Downarrow_e v \quad r \Downarrow_r \omega}{\mathbf{val}\ id = e;\ r \Downarrow_r \omega}$
(METHOD)	$\dfrac{r \Downarrow_r \omega}{\mathbf{method}\ m = e;\ r \Downarrow_r \mathbf{method}\ m = e;\ \omega}$

herleiten lässt.

Die zweite Prämisse der Regel (RENAME) könnte eigentlich auch entfallen, denn der so berechnete Wert v wird im weiteren Verlauf des Programms nicht mehr benötigt. Allerdings sollen die big und small step Semantiken bezüglich des Ergebnisses einer Berechnung äquivalent sein, und in der small step Semantik kann (ATTR-RENAME) erst angewendet werden, wenn hinter dem Attribut ein Wert steht. Mit imperativen Konzepten spielt es später durchaus eine Rolle ob und wann ein Ausdruck ausgewertet wird.

Analog zur small step Semantik betrachten wir $\Downarrow_e \subseteq Exp \times Val_e$ und $\Downarrow_r \subseteq Row \times Val_r$ als Relationen und schreiben lediglich $\Downarrow$, wenn wir uns auf beide Relationen beziehen oder wenn aus dem Zusammenhang hervorgeht auf welche Relation Bezug genommen wird.

Weiter schreiben wir $e \not\Downarrow$, falls kein big step für e existiert, und $r \not\Downarrow$, falls kein big step für r existiert.

Der nun folgende Satz wird es uns erlauben die meisten Eigenschaften der small step Semantik auf die big step Semantik zu übertragen.

Satz 2 (Äquivalenzsatz)

(a) $\forall e \in Exp : \forall v \in Val_e : e \Downarrow v \Leftrightarrow e \xrightarrow{*} v$

(b) $\forall r \in Row : \forall \omega \in Val_r : r \Downarrow \omega \Leftrightarrow r \xrightarrow{*} \omega$

Beweis: Der Beweis erfolgt in zwei Schritten. Zunächst zeigen wir, dass für jeden big step eine äquivalente[1] endliche Berechnung, d.h. eine maximale, endliche Berechnungsfolge von small steps, existiert. Anschliessend zeigen wir, dass für jede endliche Berechnung ein äquivalenter big step existiert.

„$\Rightarrow$" Diese Richtung beweisen wir mittels simultaner Induktion über die Längen der Herleitungen der big steps $e \Downarrow v$ und $r \Downarrow \omega$, und Fallunterscheidung nach der zuletzt angewandten big step Regel.

TODO

„$\Leftarrow$" Der zweite Teil des Beweises erfolgt mittels simultaner Induktion über die Länge der Berechnungen $e \xrightarrow{*} v$ und $r \xrightarrow{*} \omega$, und Fallunterscheidung nach der Form von e und r.

TODO $\square$

Wie bereits angedeutet lassen sich mit Hilfe des Äquivalenzsatzes bereits bewiesene Eigenschaften der small step Semantik auf die big step Semantik übertragen.

[1]Der Begriff der Äquivalenz beudetet in diesem Zusammenhang, dass die Ausführung eines Programms mit den unterschiedlichen Semantiken zum gleichen Ergebnis führen.

Korollar 2

(a) *Existiert ein big step $e \Downarrow v$ mit $e \in Exp, v \in Val_e$, so ist v durch e eindeutig bestimmt.*

(b) *Existiert ein big step $r \Downarrow \omega$ mit $r \in Row, \omega \in Val_r$, so ist ω durch r eindeutig bestimmt.*

Beweis: Folgt mit dem Äquivalenzsatz unmittelbar aus der Eindeutigkeit des Übergangsschritts. □

2.3 Typsystem der Sprache $\mathcal{L}_o$

Index

Literaturverzeichnis

[Rém02] Rémy, Didier: Using, Understanding, and Unraveling the OCaml Language. From Practice to Theory and Vice Versa. In: *Applied Semantics, International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000, Advanced Lectures.* London, UK : Springer-Verlag, 2002. – ISBN 3–540–44044–5, S. 413–536