

## 5 Frei vorkommende Namen und Substitution

### 5.1 Frei vorkommende Namen

**Definition:** Die Menge  $free(e)$  aller in  $e$  frei vorkommenden Namen ist induktiv definiert durch

$$\begin{aligned}
 free(c) &= \emptyset \\
 free(id) &= \{id\} \\
 free((e_1, \dots, e_k)) &= \bigcup_{i=1}^k free(e_i) \\
 free(e_1 e_2) &= free(e_1) \cup free(e_2) \\
 free(\mathbf{if} \ e_0 \ \mathbf{then} \ e_1 \ \mathbf{else} \ e_2) &= free(e_0) \cup free(e_1) \cup free(e_2) \\
 free(\mathbf{let} \ \mathbf{val} \ id = e_1 \ \mathbf{in} \ e_2 \ \mathbf{end}) &= free(e_1) \cup (free(e_2) \setminus \{id\}) \\
 free(\lambda id : \tau. e) &= free(e) \setminus \{id\}
 \end{aligned}$$

Ein Ausdruck  $e$  heißt *abgeschlossen* (engl.: *closed*), falls  $free(e) = \emptyset$ .

### 5.2 Substitution

**Definition:** Seien  $e, e' \in Exp$  und  $id \in Id$ . Dann ist der Ausdruck  $e'[e/id]$  induktiv definiert durch

$$\begin{aligned}
 c[e/id] &= c \\
 id'[e/id] &= \begin{cases} e & \text{falls } id \equiv id' \\ id' & \text{sonst} \end{cases} \\
 (e_1, \dots, e_k)[e/id] &= (e_1[e/id], \dots, e_k[e/id]) \\
 (e_1 e_2)[e/id] &= (e_1[e/id])(e_2[e/id]) \\
 (\mathbf{if} \ e_0 \ \mathbf{then} \ e_1 \ \mathbf{else} \ e_2)[e/id] &= \mathbf{if} \ e_0[e/id] \ \mathbf{then} \ e_1[e/id] \ \mathbf{else} \ e_2[e/id] \\
 (\mathbf{let} \ \mathbf{val} \ id' = e_1 \ \mathbf{in} \ e_2 \ \mathbf{end})[e/id] &= \begin{cases} \mathbf{let} \ \mathbf{val} \ id' = e_1[e/id] \ \mathbf{in} \ e_2 \ \mathbf{end} \\ \quad \text{falls } id' \equiv id \\ \mathbf{let} \ \mathbf{val} \ id' = e_1[e/id] \ \mathbf{in} \ e_2[e/id] \ \mathbf{end} \\ \quad \text{falls } id' \not\equiv id \ \text{und} \ id' \notin free(e) \\ \mathbf{let} \ \mathbf{val} \ id'' = e_1[e/id] \ \mathbf{in} \ e_2[id''/id'][e/id] \ \mathbf{end} \\ \quad \text{sonst, wobei } id'' \not\equiv id \ \text{und} \ id'' \notin free(e) \end{cases} \\
 (\lambda id' : \tau. e_1)[e/id] &= \begin{cases} \lambda id' : \tau. e_1 \\ \quad \text{falls } id' \equiv id \\ \lambda id' : \tau. e_1[e/id] \\ \quad \text{falls } id' \not\equiv id \ \text{und} \ id' \notin free(e) \\ \lambda id'' : \tau. e_1[id''/id][e/id] \\ \quad \text{sonst, wobei } id'' \not\equiv id \ \text{und} \ id'' \notin free(e) \end{cases}
 \end{aligned}$$

Man sagt  $e'[e/id]$  entsteht aus  $e'$  durch *Substitution* des Ausdrucks  $e$  für den Namen  $id$ .

## 6 Small step Semantik

Die Menge  $Val$  aller *Werte* (engl.: *values*)  $v$  ist definiert durch

$$v ::= c \\ | id \\ | (v_1, \dots, v_k) \quad (k \geq 2) \\ | \lambda id : \tau. e$$

und für jeden Typ  $\tau$  sei

$$Val^\tau = \{v \in Val \mid v :: \tau\}$$

Vorgegeben sei eine Menge  $Exn$ , deren Elemente  $exn$  wir als *Ausnahmen* (engl.: *exceptions*) bezeichnen und für jeden Operator  $op$  (außer den Projektionen) eine partielle Funktion

$$op^{\mathcal{I}} : Val \hookrightarrow Val \cup Exn$$

so, daß für alle  $\tau_1, \tau_2 \in Type$  mit  $op :: \tau_1 \rightarrow \tau_2$  gilt:

$$Val^{\tau_1} \subseteq dom(op^{\mathcal{I}}) \quad \text{und} \quad op^{\mathcal{I}}(Val^{\tau_1}) \subseteq Val^{\tau_2} \cup Exn$$

Unter einem *small step* verstehen wir eine "Formel" der Form  $e \rightarrow e'$  oder  $e \rightarrow exn$ . Die *gültigen* small steps erhält man mit den Regeln

- (OP)  $op v \rightarrow op^{\mathcal{I}}(v)$  falls  $v \in dom(op^{\mathcal{I}})$
- (PROJ)  $\#n(v_1, \dots, v_k) \rightarrow v_n$  falls  $n \in \{1, \dots, k\}$
- (BETA-V)  $(\lambda id : \tau. e) v \rightarrow e[v/id]$
- (APP-LEFT)  $\frac{e_1 \rightarrow e'_1}{e_1 e_2 \rightarrow e'_1 e_2}$
- (APP-RIGHT)  $\frac{e \rightarrow e'}{v e \rightarrow v e'}$
- (TUPLE)  $\frac{e_i \rightarrow e'_i}{(v_1, \dots, v_{i-1}, e_i, \dots, e_k) \rightarrow (v_1, \dots, v_{i-1}, e'_i, \dots, e_k)}$  falls  $i \in \{1, \dots, k\}$
- (COND-EVAL)  $\frac{e_0 \rightarrow e'_0}{\mathbf{if } e_0 \mathbf{ then } e_1 \mathbf{ else } e_2 \rightarrow \mathbf{if } e'_0 \mathbf{ then } e_1 \mathbf{ else } e_2}$
- (COND-TRUE)  $\mathbf{if } true \mathbf{ then } e_1 \mathbf{ else } e_2 \rightarrow e_1$
- (COND-FALSE)  $\mathbf{if } false \mathbf{ then } e_1 \mathbf{ else } e_2 \rightarrow e_2$
- (LET-EVAL)  $\frac{e_1 \rightarrow e'_1}{\mathbf{let val } id = e_1 \mathbf{ in } e_2 \mathbf{ end} \rightarrow \mathbf{let val } id = e'_1 \mathbf{ in } e_2 \mathbf{ end}}$
- (LET-EXEC)  $\mathbf{let val } id = v \mathbf{ in } e \mathbf{ end} \rightarrow e[v/id]$

und mit den sogenannten *exception-Regeln*. Diese exception-Regeln sind durch folgende "Meta-Regel" definiert: Für jede small step Regel, die eine Prämisse besitzt, d.h. für jede Regel der Form

$$\frac{e_1 \rightarrow e'_1}{e_2 \rightarrow e'_2}$$

sei die zugehörige *exception-Regel* definiert durch

$$\frac{e_1 \rightarrow exn}{e_2 \rightarrow exn}$$