

Abgeleitete small step Regeln

(ANDALSO-LEFT)	$\frac{e_1 \rightarrow e'_1}{e_1 \text{ andalso } e_2 \rightarrow e'_1 \text{ andalso } e_2}$
(ANDALSO-TRUE)	$true \text{ andalso } e \rightarrow e$
(ANDALSO-FALSE)	$false \text{ andalso } e \rightarrow false$
(ORELSE-LEFT)	$\frac{e_1 \rightarrow e'_1}{e_1 \text{ orelse } e_2 \rightarrow e'_1 \text{ orelse } e_2}$
(ORELSE-TRUE)	$true \text{ orelse } e \rightarrow true$
(ORELSE-FALSE)	$false \text{ orelse } e \rightarrow e$
(BETA-V-n)	$(\lambda(id_1 : \tau_1, \dots, id_n : \tau_n). e) (v_1, \dots, v_n) \rightarrow e[v_1/id_1] \dots [v_n/id_n]$
(LET-EVAL-n)	$\frac{e_1 \rightarrow e'_1}{\text{let val } (id_1, \dots, id_n) = e_1 \text{ in } e_2 \text{ end} \rightarrow \text{let val } (id_1, \dots, id_n) = e'_1 \text{ in } e_2 \text{ end}}$
(LET-EXEC-n)	$\text{let val } (id_1, \dots, id_n) = (v_1, \dots, v_n) \text{ in } e \text{ end} \rightarrow e[v_1/id_1] \dots [v_n/id_n]$

Ableitung der Regel (BETA-V-n) für $n = 2$

$(\lambda(id_1 : \tau_1, id_2 : \tau_2). e) (v_1, v_2)$

steht für

$(\lambda id : \tau_1 * \tau_2. \text{let val } id_1 = \#1 \text{ id}$
 $\quad \text{in let val } id_2 = \#2 \text{ id}$
 $\quad \quad \text{in } e$
 $\quad \quad \text{end}$
 $\quad \text{end}) (v_1, v_2)$

$\rightarrow \text{let val } id_1 = \#1 (v_1, v_2)$
 $\quad \text{in let val } id_2 = \#2 (v_1, v_2)$
 $\quad \quad \text{in } e$
 $\quad \quad \text{end}$
 $\quad \text{end}$

mit Regel (BETA-V), da $id \neq id_1$ und $id \notin free(e)$

$\rightarrow \text{let val } id_1 = v_1$
 $\quad \text{in let val } id_2 = \#2 (v_1, v_2)$
 $\quad \quad \text{in } e$
 $\quad \quad \text{end}$
 $\quad \text{end}$

mit Regel (PROJ)

$\rightarrow \text{let val } id_2 = \#2 (v_1, v_2)$
 $\quad \text{in } e[v_1/id_1]$
 $\quad \text{end}$

mit Regel (LET-EXEC)

$\rightarrow \text{let val } id_2 = v_2$
 $\quad \text{in } e[v_1/id_1]$
 $\quad \text{end}$

mit Regel (PROJ)

$\rightarrow e[v_1/id_1][v_2/id_2]$
 mit Regel (LET-EXEC)

Ableitung der Regel (LET-EXEC-n) für $n = 2$

```
let val (id1, id2) = (v1, v2)  
in e  
end
```

steht für

```
let val id = (v1, v2)  
in let val id1 = #1 id  
    in let val id2 = #2 id  
        in e  
        end  
    end  
end
```

→ let val id₁ = #1 (v₁, v₂)
 in let val id₂ = #2 (v₁, v₂)
 in e
 end
 end

mit Regel (LET-EXEC), da $id \neq id_1$ und $id \notin \text{free}(e)$

→ let val id₁ = v₁
 in let val id₂ = #2 (v₁, v₂)
 in e
 end
 end

mit Regel (PROJ)

→ let val id₂ = #2 (v₁, v₂)
 in e[v₁/id₁]
 end

mit Regel (LET-EXEC)

→ let val id₂ = v₂
 in e[v₁/id₁]
 end

mit Regel (PROJ)

→ e[v₁/id₁][v₂/id₂]
mit Regel (LET-EXEC)

Ableitung der übrigen Regeln

Sie ergeben sich unmittelbar durch die Übersetzung des syntaktischen Zuckers.