

Übersetzungsfunktionen:

Anweisungen:

$\text{code } e; \rho = \text{code}_R e \rho$
 pop

$\text{code } (s \text{ } ss) \rho = \text{code } s \rho$
 $\text{code } ss \rho$

$\text{code } \varepsilon \rho = // \text{ leere Folge von Befehlen}$

$\text{code } (if (e) s) \rho = \text{code}_R e \rho$
 jumpz A
 $\text{code } s \rho$

A: ...

$\text{code } (if (e) s_1 \text{ else } s_2) \rho = \text{code}_R e \rho$
 jumpz A
 $\text{code } s_1 \rho$
 jump B

A: $\text{code } s_2 \rho$

B: ...

$\text{code } (while (e) s) \rho =$
 A: $\text{code}_R e \rho$
 jumpz B
 $\text{code } s \rho$
 jump A

B: ...

$\text{code } (for (e_1; e_2; e_3) s) \rho = \text{code}_R e_1$
 pop
 A: $\text{code}_R e_2 \rho$
 jumpz B
 $\text{code } s \rho$
 $\text{code}_R e_3 \rho$
 pop
 jump A

B: ...

$\text{code } s \rho = \text{code}_R e \rho$ $C_0: \text{code } ss_0 \rho$ B: $\text{jump } C_0$
 $\text{check } 0 \ k \ B$ jump D ...
 $\text{jump } C_k$
 $C_k: \text{code } ss_k \rho$ D: ...
 jump D

wenn $s =$ **switch** (e) {
 case 0: ss_0 **break**;
 case 1: ss_1 **break**;
 :
 case $k - 1$: ss_{k-1} **break**;
 default: ss_k
 }

wobei $\text{check } 0 \ k \ B$ = dup dup $\text{jumpi } B$
 $\text{loadc } 0$ $\text{loadc } k$ $A:$ pop
 geq leq $\text{loadc } k$
 $\text{jumpz } A$ $\text{jumpz } A$ $\text{jumpi } B$

Ausdrücke:

$\text{code}_L (e_1[e_2]) \rho$ = $\text{code}_R e_1 \rho$
 $\text{code}_R e_2 \rho$
 $\text{loadc } |t|$
 mul
 add sofern e_1 Typ $t []$ hat

$\text{code}_L (e.a) \rho$ = $\text{code}_L e \rho$
 $\text{loadc } (\rho a)$
 add

$\text{code}_L (*e) \rho$ = $\text{code}_R e \rho$

$\text{code}_L x \rho$ = $\text{loadc } (\rho x)$

$\text{code}_R (\&e) \rho$ = $\text{code}_L e \rho$

$\text{code}_R (\text{malloc}(e)) \rho$ = $\text{code}_R e \rho$
 new

$\text{code}_R e \rho$ = $\text{code}_L e \rho$ falls e ein Feld ist

$\text{code}_R (e_1 \square e_2) \rho$ = $\text{code}_R e_1 \rho$
 $\text{code}_R e_2 \rho$
 op op Befehl zu Operator ' \square '
 $\text{code}_R q \rho$ = $\text{loadc } q$ q Konstante

$\text{code}_R (e_1 = e_2) \rho$ = $\text{code}_R e_2 \rho$
 $\text{code}_L e_1 \rho$
 store

$\text{code}_R e \rho$ = $\text{code}_L e \rho$
 load sonst

Funktionen:

$\text{code}_R g(e_1, \dots, e_n) \rho$ = mark
 $\text{code}_R e_1 \rho$
 \dots
 $\text{code}_R e_n \rho$
 $\text{code}_R g \rho$
 $\text{call } m$

wobei m der Platz für die aktuellen Parameter ist.

$\text{code}_R f \rho = \text{loadc } (\rho f)$ f ein Funktions-Name
 $\text{code}_R (*e) \rho = \text{code}_R e \rho$ e ein Funktions-Zeiger
 $\text{code}_R e \rho = \text{code}_L e \rho$
 $\text{move } k$ e eine Struktur der Größe k

$\text{code } t f (\text{specs})\{V_defs \ ss\} \rho =$
 $_f :$ $\text{enter } q$ // *setzen des EP*
 $\text{alloc } k$ // *Anlegen der lokalen Variablen*
 $\text{code } ss \rho_f$
 return // *Verlassen der Funktion*

$\text{code}_L x \rho = \begin{cases} \text{loadc } j & \text{tag} = G \\ \text{loadrc } j & \text{tag} = L \end{cases}$

$\text{code return } e; \rho = \text{code}_R e \rho$
 $\text{storer } -3$
 return

Ganze Programme:

$\text{code } p \emptyset =$ $\text{enter } (k + 6)$
 $\text{alloc } (k + 1)$
 mark
 $\text{loadc } _main$
 $\text{call } 0$
 pop
 halt
 $_f_1 :$ $\text{code } F_def_1 \rho$
 \vdots
 $_f_n :$ $\text{code } F_def_n \rho$

wobei $p \equiv V_defs \ F_def_1 \dots F_def_n$
 $\emptyset \hat{=} \text{leere Adress-Umgebung;}$
 $\rho \hat{=} \text{globale Adress-Umgebung;}$
 $k \hat{=} \text{Platz für globale Variablen}$