

1 Extended Stack Machine

In order to compile a language with structural control flow constructs into a program for the stack machine the latter has to be extended. First, we introduce a set of label names

$$\mathcal{L} = \{l_1, l_2, \dots\}$$

Then, we add three extra control flow instructions:

$$\begin{aligned} I \quad + = \quad & \text{LABEL } \mathcal{L} \\ & \text{JMP } \mathcal{L} \\ & \text{CJMP}_x \mathcal{L}, \text{ where } x \in \{\text{nz}, \text{z}\} \end{aligned}$$

In order to give the semantics to these instructions, we need to extend the syntactic form of rules, used in the description of big-step operational semantics. Instead of the rules in the form

$$\frac{c \xRightarrow[p]{\mathcal{S}\mathcal{M}} c'}{c' \xRightarrow[p']{\mathcal{S}\mathcal{M}} c''}$$

we use the following form

$$\frac{\Gamma' \vdash c \xRightarrow[p]{\mathcal{S}\mathcal{M}} c'}{\Gamma \vdash c' \xRightarrow[p']{\mathcal{S}\mathcal{M}} c''}$$

where Γ, Γ' — environments. The structure of environments can be different in different cases; for now environment is just a program. Informally, the semantics of control flow instructions can not be described in terms of just a current instruction and current configuration — we need to take the whole program into account. Thus, the enclosing program is used as an environment.

Additionally, for a program P and a label l we define a subprogram $P[l]$, such that P is uniquely represented as $p'[\text{LABEL } l]P[l]$. In other words $P[l]$ is a unique suffix of P , immediately following the label l (if there are multiple (or no) occurrences of label l in P , then $P[l]$ is undefined).

All existing rules have to be rewritten — we need to formally add a $P \vdash \dots$ part everywhere. For the new instructions the rules are given on Fig. 1.

Finally, the top-level semantics for the extended stack machine can be redefined as follows:

$$\frac{p \vdash \langle \epsilon, \langle \Lambda, \langle i, \epsilon \rangle \rangle \rangle \xRightarrow[p]{\mathcal{S}\mathcal{M}} \langle s, \langle \sigma, \omega \rangle \rangle}{\llbracket p \rrbracket_{\mathcal{S}\mathcal{M}} i = \mathbf{out} \, \omega}$$

$$\begin{array}{c}
\frac{P \vdash c \xRightarrow{P} c'}{\frac{P \vdash c \xRightarrow{[\text{LABEL } l]p} c'}{SM}} \quad [\text{Label}_{SM}] \\
\\
\frac{P \vdash c \xRightarrow{P[l]} c'}{\frac{P \vdash c \xRightarrow{[\text{JMP } l]p} c'}{SM}} \quad [\text{JMP}_{SM}] \\
\\
\frac{z \neq 0, \quad P \vdash \langle s, \theta \rangle \xRightarrow{P[l]} c'}{\frac{P \vdash \langle zs, \theta \rangle \xRightarrow{[\text{CJMP}_{nz} l]p} c'}{SM}} \quad [\text{CJMP}_{nz SM}^+] \\
\\
\frac{z = 0, \quad P \vdash \langle s, \theta \rangle \xRightarrow{P} c'}{\frac{P \vdash \langle zs, \theta \rangle \xRightarrow{[\text{CJMP}_{nz} l]p} c'}{SM}} \quad [\text{CJMP}_{nz SM}^-] \\
\\
\frac{z = 0, \quad P \vdash \langle s, \theta \rangle \xRightarrow{P[l]} c'}{\frac{P \vdash \langle zs, \theta \rangle \xRightarrow{[\text{CJMP}_z l]p} c'}{SM}} \quad [\text{CJMP}_z SM^+] \\
\\
\frac{z \neq 0, \quad P \vdash \langle s, \theta \rangle \xRightarrow{P} c'}{\frac{P \vdash \langle zs, \theta \rangle \xRightarrow{[\text{CJMP}_z l]p} c'}{SM}} \quad [\text{CJMP}_z SM^-]
\end{array}$$

Figure 1: Big-step operational semantics for extended stack machine

2 A Compiler for the Stack Machine

A compiler for the language with structural control flow into the stack machine can be given in the form of static semantics. Similarly to the big-step operational semantics, the compiler also operates on environment. For now, the environment allows us to generate fresh labels. Thus, a compiler specification for statements has the shape

$$[[p]]_S^{comp} \Gamma = \langle c, \Gamma' \rangle$$

where p is a source program, Γ, Γ' — some environments, c — generated program for the stack machine. As we can see, the environment changes during the code generation, hence auxiliary semantic primitive $[[\bullet]]_S^{comp}$. We need one primitive to operate on environments which allocates a number of fresh labels and returns a new environment:

labels Γ

The number of labels allocated is determined by context.

We give an example of compiler specification rule for the while-loop:

$$\frac{\langle l_e, l_s, \Gamma' \rangle = \mathbf{labels} \ \Gamma, \quad \llbracket s \rrbracket_S^{comp} \Gamma' = \langle c_s, \Gamma'' \rangle}{\llbracket \text{while } e \text{ do } s \text{ od} \rrbracket_S^{comp} \Gamma = \langle \begin{array}{l} \text{JMP } l_e \\ \text{LABEL } l_s \\ c_s \\ \text{LABEL } l_e \\ \llbracket e \rrbracket_{\mathcal{E}}^{comp} \\ \text{CJMP}_{nz} l_s, \quad \Gamma'' \end{array} \rangle}$$

Note, the compiler for expressions is not changed and completely reused.

Finally, the top-level compiler for the whole program can be defined as follows:

$$\frac{\llbracket p \rrbracket_S^{comp} \Gamma_0 = \langle c, _ \rangle}{\llbracket p \rrbracket^{comp} = c}$$

where Γ_0 — empty environment.