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

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

Then, we add three extra control flow instructions:

$$\begin{array}{rcl}\mathscr{I} & += & \texttt{LABEL}\,\mathscr{L}\\ & & \texttt{JMP}\,\mathscr{L}\\ & & \texttt{CJMP}_x\,\mathscr{L},\ \text{where } x \in \{\texttt{nz}, \texttt{z}\}\end{array}$$

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 smeantics. Instead of the rules in the form

$$\frac{c \xRightarrow[\mathscr{SM}]{p} c'}{c' \xRightarrow[\mathscr{SM}]{p'} c''}$$

we use the following form

$$\frac{\Gamma' \vdash c \xRightarrow[\mathscr{SM}]{p} c'}{\Gamma \vdash c' \xRightarrow[\mathscr{SM}]{p'} c''}$$

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 $p'[\texttt{LABEL}\ l]P[l]$. 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:

$$\frac{p \vdash \langle \varepsilon, \langle \Lambda, \langle i, \varepsilon\rangle\rangle\rangle \xRightarrow[\mathscr{SM}]{p} \langle s, \langle \sigma, \omega\rangle\rangle}{[\![p]\!]_{\mathscr{SM}}\ i = \mathbf{out}\ \omega}$$

$$\frac{P \vdash c \xRightarrow[\mathscr{SM}]{p} c'}{P \vdash c \xRightarrow[\mathscr{SM}]{[\texttt{LABEL}\ l]p} c'} \qquad [\textsc{Label}_{SM}]$$

$$\frac{P \vdash c \xRightarrow[\mathscr{SM}]{P[l]} c'}{P \vdash c \xRightarrow[\mathscr{SM}]{[\texttt{JMP}\ l]p} c'} \qquad [\mathrm{JMP}_{SM}]$$

$$\frac{z \neq 0, \quad P \vdash \langle s, \theta\rangle \xRightarrow[\mathscr{SM}]{P[l]} c'}{P \vdash \langle zs, \theta\rangle \xRightarrow[\mathscr{SM}]{[\texttt{CJMP}_{nz}\ l]p} c'} \qquad [\mathrm{CJMP}^{+}_{nz\,SM}]$$

$$\frac{z = 0, \quad P \vdash \langle s, \theta\rangle \xRightarrow[\mathscr{SM}]{p} c'}{P \vdash \langle zs, \theta\rangle \xRightarrow[\mathscr{SM}]{[\texttt{CJMP}_{nz}\ l]p} c'} \qquad [\mathrm{CJMP}^{-}_{nz\,SM}]$$

$$\frac{z = 0, \quad P \vdash \langle s, \theta\rangle \xRightarrow[\mathscr{SM}]{P[l]} c'}{P \vdash \langle zs, \theta\rangle \xRightarrow[\mathscr{SM}]{[\texttt{CJMP}_{z}\ l]p} c'} \qquad [\mathrm{CJMP}^{+}_{z\,SM}]$$

$$\frac{z \neq 0, \quad P \vdash \langle s, \theta\rangle \xRightarrow[\mathscr{SM}]{p} c'}{P \vdash \langle zs, \theta\rangle \xRightarrow[\mathscr{SM}]{[\texttt{CJMP}_{z}\ l]p} c'} \qquad [\mathrm{CJMP}^{-}_{z\,SM}]$$

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

$$[\![p]\!]^{comp}_{\mathscr{S}}\,\Gamma = \left\langle c, \Gamma'\right\rangle$$

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 auxilliary semantic primitive $[\![\bullet]\!]^{comp}_{\mathscr{S}}$. We need one primitive to operate on environments which allocates a number of fresh labels and returns a new environment:

$$\textbf{labels}\ \Gamma$$

The number of labels allocated is determined by context.
We give an example of compiler specification rule for the while-loop:

$$\frac{\langle l_e, l_s, \Gamma' \rangle = \textbf{labels}\ \Gamma, \quad [\![s]\!]^{comp}_{\mathscr{S}}\, \Gamma' = \langle c_s, \Gamma'' \rangle}{[\![\texttt{while}\ e\ \texttt{do}\ s\ \texttt{od}]\!]^{comp}_{\mathscr{S}}\, \Gamma \quad = \quad \left\langle \begin{array}{l} \texttt{JMP}\ l_e \\ \texttt{LABEL}\ l_s \\ c_s \\ \texttt{LABEL}\ l_e \\ [\![e]\!]^{comp}_{\mathscr{E}} \\ \texttt{CJMP}_{nz}\ l_s, \quad \Gamma'' \end{array} \right\rangle}$$

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:

$$\frac{[\![p]\!]^{comp}_{\mathscr{S}}\, \Gamma_0 = \langle c, \_ \rangle}{[\![p]\!]^{comp} = c}$$

where Γ_0 — empty environment.