Control Flow Constructs
(the first draft)

Dmitry Boulytchev

October 3, 2018

1 Structural Control Flow

We add a few structural control flow constructs to the language:

$$\begin{array}{rcl}\mathscr{S} & += & \text{skip} \\ & & \text{if } \mathscr{E} \text{ then } \mathscr{S} \text{ else } \mathscr{S} \\ & & \text{while } \mathscr{E} \text{ do } \mathscr{S}\end{array}$$

The big-step operational semantics is straightforward and is shown on Fig. 1.

In the concrete syntax for the constructs we add the closing keywords " if " and "od" as follows:

$$\begin{array}{c}\text{if } e \text{ then } s_1 \text{ else } s_2 \text{ fi} \\ \text{while } e \text{ do } s \text{ od}\end{array}$$

2 Extended Stack Machine

In order to compile the extended language 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:

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

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

$$\frac{c \xRightarrow{p} c'}{c' \xRightarrow{p'} c''}$$

$$c \xrightarrow{\text{skip}} c \qquad [\text{Skip}_{bs}]$$

$$\frac{c \xrightarrow{S_1} c'}{c \xrightarrow{\text{if } e \text{ then } S_1 \text{ else } S_2} c'},\ [\![e]\!]\ s \neq 0 \qquad [\text{If-True}_{bs}]$$

$$\frac{c \xrightarrow{S_2} c'}{c \xrightarrow{\text{if } e \text{ then } S_1 \text{ else } S_2} c'},\ [\![e]\!]\ s = 0 \qquad [\text{If-False}_{bs}]$$

$$\frac{c \xrightarrow{S} c',\ c' \xrightarrow{\text{while } e \text{ do } S} c''}{c \xrightarrow{\text{while } e \text{ do } S} c''},\ [\![e]\!]\ s \neq 0 \qquad [\text{While-True}_{bs}]$$

$$c \xrightarrow{\text{while } e \text{ do } S} c,\ [\![e]\!]\ s = 0 \qquad [\text{While-False}_{bs}]$$

Figure 1: Big-step operational semantics for control flow statements

we use the following form

$$\frac{\Gamma \vdash c \overset{p}{\Longrightarrow} c'}{\Gamma' \vdash c' \overset{p'}{\Longrightarrow} c''}$$

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. 2.

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

$$\forall p \in \mathscr{P}, \forall i \in \mathbb{Z}^*\ :\ [\![p]\!]_{SM}\ i = o \Leftrightarrow p \vdash \langle \varepsilon, \langle \bot, i, \varepsilon \rangle \rangle \overset{p}{\Longrightarrow} \langle \_, \langle \_, \_, o \rangle \rangle$$

3 Syntax Extensions

With the structural control flow constructs already implemented, it is rather simple to “saturate” the language with more elaborated control constructs, using the method of syntactic extension. Namely, we may introduce the following constructs

$$\frac{P \vdash c \overset{p}{\Longrightarrow} c'}{P \vdash c \xRightarrow{(\text{LABEL } l)p} c'} \qquad [\text{Label}_{SM}]$$

$$\frac{P \vdash c \overset{P[l]}{\Longrightarrow} c'}{P \vdash c \xRightarrow{(\text{JMP } l)p} c'} \qquad [\text{JMP}_{SM}]$$

$$\frac{P \vdash c \overset{P[l]}{\Longrightarrow} c'}{P \vdash \langle z :: st, c\rangle \xRightarrow{(\text{CJMP}_{nz}\ l)p} c'},\ z \neq 0 \qquad [\text{CJMP}^{+}_{nz\,SM}]$$

$$\frac{P \vdash c \overset{p}{\Longrightarrow} c'}{P \vdash \langle z :: st, c\rangle \xRightarrow{(\text{CJMP}_{nz}\ l)p} c'},\ z = 0 \qquad [\text{CJMP}^{-}_{nz\,SM}]$$

$$\frac{P \vdash c \overset{P[l]}{\Longrightarrow} c'}{P \vdash \langle z :: st, c\rangle \xRightarrow{(\text{CJMP}_{z}\ l)p} c'},\ z = 0 \qquad [\text{CJMP}^{+}_{z\,SM}]$$

$$\frac{P \vdash c \overset{p}{\Longrightarrow} c'}{P \vdash \langle z :: st, c\rangle \xRightarrow{(\text{CJMP}_{z}\ l)p} c'},\ z \neq 0 \qquad [\text{CJMP}^{-}_{z\,SM}]$$

Figure 2: Big-step operational semantics for extended stack machine

```
  if   $e_1$  then  $s_1$
elif   $e_2$  then  $s_2$
...
elif   $e_k$  then  $s_k$
[ else  $s_{k+1}$  ]
  fi
```

and

```
for  $s_1$ ,  $e$ ,  $s_2$  do  $s_3$  od
```

only at the syntactic level, directly parsing these constructs into the original abstract syntax tree, using the following conversions:

```
  if  $e_1$  then  $s_1$                        if  $e_1$  then  $s_1$
elif  $e_2$  then  $s_2$                  else  if  $e_2$  then  $s_2$
...                          $\rightsquigarrow$      ...
elif  $e_k$  then  $s_k$                  else  if  $e_k$  then  $s_k$
else  $s_{k+1}$                            else  $s_{k+1}$
fi                                              fi
                                                ...
                                                fi
```

```
  if  $e_1$  then  $s_1$                        if  $e_1$  then  $s_1$
elif  $e_2$  then  $s_2$                  else  if  $e_2$  then  $s_2$
...                          $\rightsquigarrow$      ...
elif  $e_k$  then  $s_k$                  else  if  $e_k$  then  $s_k$
fi                                        else  skip
                                                fi
                                                ...
                                                fi
```

```
for  $s_1$,  $e$,  $s_2$  do  $s_3$  od        $\rightsquigarrow$        $s_1$;
                                                 while  $e$  do
                                                   $s_3$;
                                                   $s_2$
                                                 od
```

The advantage of syntax extension method is that it makes it possible to add certain constructs with almost zero cost — indeed, no steps have to be made in order to implement the extended constructs (besides parsing). Note, the semantics of extended constructs is provided for free as well (which is not always desirable). Another potential problem with syntax extensions is that they can easily provide unreasonable results. For example, one may be tempted to implement a post-condition loop using syntax extension:

```
repeat  $s$  until  $e$                  $\rightsquigarrow$        $s$;
                                                 while  $e$ == 0  do
                                                   $s$
                                                 od
```

However, for nested repeat constructs the size of extended program is exponential w.r.t. the nesting depth, which makes the whole idea unreasonable.