Functions
(the first draft)

Dmitry Boulytchev

October 9, 2018

1 Procedures

Procedures are unit-returning functions. We consider adding procedures as a separate step, since introducing full-fledged functions would require essential redefinition of the semantics for expressions; at the same time the code generation for funictions is a little trickier, than for procedures, so it is reasonable to split the implementation in two steps.

At the source level procedures are added as a separate syntactic category — definition $\mathscr{D}$:

$$\begin{array}{rcl}\mathscr{D} & = & \varepsilon \\ & & (\mathrm{fun}\ \mathscr{X}\ (\mathscr{X}^*)\ \mathrm{local}\ \mathscr{X}^*\ \{\mathscr{S}\})\mathscr{D}\end{array}$$

In other words, a definition is a (possibly empty) sequence of procedure descriptions. Each description consists of a name for the procedure, a list of names for its arguments and local variables, and a body (statement). In concrete syntax a single definition looks like

```
fun name (a_1, a_2, ..., a_k)
  local l_1, l_2, ..., l_n {
  s
}
```

where *name* — a name for the procedure, a_i — its arguments, l_i — local variables, s — body.

We also need to add a call statement to the language:

$$\mathscr{S} += \mathscr{X}\,(\mathscr{E}^*)$$

In a concrete syntax a call to a procedure f with arguments $e_1,\dots,e_k$ looks like

$$f\,(e_1,\dots,e_k)$$

Finally, we have to redefine the syntax category for programs at the top level:

$$\mathscr{L} = \mathscr{D}\mathscr{S}$$

In other words, we extend a statement with a set of definitions.

With procedures, we need to introduce the notion of scope. When a procedure is called, its arguments are associated with actual parameter values. A procedure is also in "posession" of its local variables. So, in principle, the context of a procedure execution is a set of arguments, local variables and, possibly, some other variables, for example, the global ones. However, the exact details of procedure context manipilation can differ essentially from language to language.

In our case, we choose a static scoping — each procedure, besides its arguments and local variables, has an access only to global variables. To describe this semantics, we need to change the definition of a state we've used so far:

$$\Sigma = (\mathscr{X} \to \mathbb{Z}) \times 2^{\mathscr{X}} \times (\mathscr{X} \to \mathbb{Z})$$

Now the state is a triple: a global state, a set of variables, and a local state. Informally, in a new state $\langle \sigma_g, S, \sigma_l \rangle$ S describes a set of local variables, σ_l — their values, and σ_g — the values of all other accessible variables.

We need to redefine all state-manipulation primitives; first, the valuation of variables:

$$\langle \sigma_g, S, \sigma_l \rangle\ x = \left\{ \begin{array}{lcl} \sigma_g\, x & , & x \notin S \\ \sigma_l\, x & , & x \in S \end{array} \right.$$

Then, updating the state:

$$\langle \sigma_g, S, \sigma_l \rangle\, [x \leftarrow z] = \left\{ \begin{array}{lcl} \langle \sigma_g[x \leftarrow z], S, \sigma_l \rangle & , & x \notin S \\ \langle \sigma_g, S, \sigma_l[x \leftarrow z] \rangle & , & x \in S \end{array} \right.$$

As an empty state, we take the following triple:

$$\bot = \langle \bot, \varnothing, \bot \rangle$$

Finally, we need two transformations for states:

$$\begin{array}{rcl} \text{enter}\ \langle \sigma_g, \_, \_ \rangle\ S & = & \langle \sigma_g, S, \bot \rangle \\ \text{leave}\ \langle \sigma_g, \_, \_ \rangle\ \langle \_, S, \sigma_l \rangle & = & \langle \sigma_g, S, \sigma_l \rangle \end{array}$$

The first one simlulates entering the new scope with a set of local variables S; the second one simulates leaving from an inner scope (described by the first state) to an outer one (described by the second).

All exising rules for big-step operational semantics have to be enriched by functional environment Γ, which binds procedure names to their definitions. As this binding never changes during the program interpretation, we need only to propagate this environment, adding $\Gamma \vdash ...$ for each transition "$... \Longrightarrow ...$". The only thing we need now is to describe the rule for procedure calls:

$$\frac{\Gamma \vdash \left\langle \text{enter}\,\sigma\,(\bar{a}@\bar{l})\overline{[a \leftarrow [\![e]\!]\sigma]},\, i,\, o \right\rangle \xRightarrow{S} \langle \sigma',\, i',\, o' \rangle}{\Gamma \vdash \langle \sigma,\, i,\, o \rangle \xRightarrow{f(\bar{e})} \langle \text{leave}\,\sigma'\,\sigma,\, i', o' \rangle} \qquad [\text{Call}_{bs}]$$

where $\Gamma f = \text{fun}\ f\ (\bar{a})\ \text{local}\ \bar{l}\ \{S\}$.

2 Extended Stack Machine

In order to support procedures and calls, we enrich the stack machine with three following instructions:

$$\begin{array}{lcl} \mathscr{I} & += & \text{BEGIN}\ \mathscr{X}^*\ \mathscr{X}^* \\ & & \text{CALL}\ \mathscr{X} \\ & & \text{END} \end{array}$$

Informally speaking, instruction BEGIN performs entering into the scope of a procedure; its operands are the lists of argument names and local variables; END leaves the scope and returns to the call site, and CALL performs the call itself.

We need to enrich the configurations for the stack machine as well:

$$\mathscr{C}_{SM} = (\mathscr{P} \times \Sigma) \times \mathbb{Z}^* \times \mathscr{C}$$

Here we added a control stack — a stack of pairs of programs and states. Informally, when performing the CALL instruction, we put the following program and current state on a stack to use them later on, when corresponding END instruction will be encountered. As all other instructions does not affect the control stack, it gets threaded through all rules of operational semantics unchanged.

Now we specify additional rules for the new instructions:

$$\frac{P \vdash \left\langle cs,\, st,\, \left\langle \text{enter}\,\sigma\,(\bar{a}@\bar{l})\overline{[a \leftarrow z]},\, i,\, o \right\rangle \right\rangle \xRightarrow{p} c'}{P \vdash \langle cs,\, \bar{z}@st,\, \langle \sigma,\, i,\, o \rangle \rangle \xRightarrow{(\text{BEGIN}\ \bar{a}\,\bar{l})p} c'} \qquad [\text{Begin}_{SM}]$$

$$\frac{P \vdash \langle (p,\, \sigma) :: cs,\, st,\, \langle \sigma,\, i,\, o \rangle \rangle \xRightarrow{P[f]} c'}{P \vdash \langle cs,\, st,\, \langle \sigma,\, i,\, o \rangle \rangle \xRightarrow{(\text{CALL}\ f)p} c'} \qquad [\text{Call}_{SM}]$$

$$\frac{P \vdash \langle cs,\, st,\, \langle \text{leave}\,\sigma\,\sigma',\, i,\, o \rangle \rangle \xRightarrow{p'} c'}{P \vdash \langle (p',\, \sigma') :: cs,\, st,\, \langle \sigma,\, i,\, o \rangle \rangle \xRightarrow{\text{END}p} c'} \qquad [\text{EndRet}_{SM}]$$

$$P \vdash \langle \varepsilon,\, st,\, c \rangle \xRightarrow{\text{END}p} \langle \varepsilon,\, st,\, c \rangle \qquad [\text{EndStop}_{SM}]$$