3. Callcc and Throw.

① Some programming languages allow continuations to be denotable values, and provide language constructs for manipulating continuation values.

② Semantically, this means that we change V as follows:

$$V \underset{\psi}{\overset{\phi}{\rightleftarrows}} V_{int} + V_{bool} + V_{fun} + V_{tuple} + V_{alt} + V_{cont}.$$

(V_{cont}: newly added.)

Syntactically, it (often) involves adding the following two constructs:

$$\langle exp \rangle ::= \text{callcc } \langle exp \rangle \mid \text{throw } \langle exp \rangle \langle exp \rangle$$

callcc expects a function as its argument. (callcc ($\lambda f.e$)) reifies the current continuation, binds f to it, and executes e. The bound continuation f can be invoked by throw, as in (throw f 3). This calls the continuation f with the value 3.

③ Here are the semantic clauses for callcc and throw:

(current continuation copied.)

$$[\![\text{callcc } e]\!]\eta\,k = [\![e]\!]\eta\,(\lambda f.\ f\ (\phi\langle s, k\rangle)\ k)_{fun}$$

$$[\![\text{throw } e\ e']\!]\eta\,k = [\![e]\!]\eta\,(\lambda k'.\ [\![e']\!]\eta\,k')_{cont}$$

(k: ignored. The continuation obtained from e is used instead.)

Intuitively, in

(callcc λk. .... throw k 3 ....)

— can be viewed as putting a label denoted by k, and — can be understood as a goto to this label.

④ What are the results of the following expressions?

(i) callcc (λk. 2+ throw k (3×4)).

(ii) (callcc ($\lambda k.\lambda x.$ throw k ($\lambda y.$ x+y))) 6

The first example can be understood as skipping some part of computation. The second shows how we can repeat the computation of some part of an expression using continuation.

⑤ The next example is likely very hard to understand because it uses features not explained so far and it is also quite tricky. The example is from p290. of the textbook.

Imagine that we would like to implement a routine backtrack that takes a function and tries the function with a parameter amb representing a nondeterministic choice between true and false. It collects the results of all those choices and returns a list of all those results. For instance,

backtrack (λamb.
  if amb <> then (if amb <> then 0 else 1)
    (empty tuple.)
  else (if amb <> then 2 else 3))