Continuation Passing Style Semantics

Scribe: André Allavena, Ben Viglietta
Lecturer: Radu Rugina

So far, we have discussed the semantics of language constructs with structured control-flow (e.g. if-then-else, while, procedures). Next, we want to explore constructs with unstructured (non-local) control flow, such as goto's and exceptions. However, it is difficult to express the semantics of such constructs with our current formalism. We therefore introduce a new style of semantics, called Continuation Passing Style, which is more appropriate for constructs with non-local flows. A *continuation* is a piece of computation that remains to be executed. The semantic function $\mathcal{C}[\![e]\!]$ for an expression e now takes a variable environment and a current continuation, and yields the result of the evaluation:

$$\mathcal{C}[\![e]\!] : \mathit{Env} \to \mathit{Cont} \to \mathit{Result}$$

Hence, the denotation of e is $\mathcal{C}[\![e]\!]\rho\kappa$, where ρ is the environment and κ is the continuation, i.e., the computation left to execute. To compute the result, we must define $\mathcal{C}[\![e]\!]\rho\kappa$ such that it first evaluates e and then it invokes the continuation κ with the result of the evaluation of e as argument.

Consider the example:

$$e = \mathsf{if}\ (x \leq 0)\ \mathsf{then}\ 0\ \mathsf{else}\ f(x-1)$$

rewritten as:

$$\underbrace{(\lambda b.\ \mathsf{if}\ b\ \mathsf{then}\ 0\ \mathsf{else}\ f(x-1))}_{\text{cont}}\ \underbrace{(x \leq 0)}_{\text{exp}}$$

The right-hand expression is the expression currently being computed, and the left-hand expression — the continuation — is the computation that remains to be done after $(x \leq 0)$ is completed.

1 CPS Semantic Domains

We formally model the computation that a continuation represents using a mapping that takes the result of a previous computation and yields a new result. Hence, the semantic domain for continuations is:

$$\mathit{Cont} = \mathit{Result} \to \mathit{Result}$$

We also need to pass continuations as parameters to functions. This is required because we must dynamically determine the continuation at each call site. Hence, the semantic domain for functions is:

$$\mathit{Function} \quad = \quad \mathit{Value} \to \mathit{Cont} \to \mathit{Result}$$

2 CPS Denotations

To evaluate a program P, we use the following denotation:

$$\mathcal{C}[\![P]\!]\rho_0\kappa_0$$

where:

- ρ_0 is the initial environment;
- κ_0 is the *halt* continuation. It takes the result of the evaluation of the program as a parameter and produces the final answer. A natural choice for *halt* is the identity function:

  $$\kappa_0 = \lambda x.\ x$$

  However, we could use a different *halt* continuation, which tells us if there was an error in the execution of the program: $\kappa_0 = \lambda r \in \mathit{Result}.\ \mathsf{if}\ (r = \mathit{error})\ \mathsf{then}\ 1\ \mathsf{else}\ 0$.

2.1 Numbers and variables

The CPS denotations for numbers and variables are given by the following rules:

$$\begin{array}{rcl} \mathcal{C}[\![n]\!]\rho\kappa & = & \kappa(\mathbb{Z}(n)) \\ \mathcal{C}[\![x]\!]\rho\kappa & = & \kappa(\rho(x)) \end{array}$$

2.2 Conditional statements

The CPS denotation of the if statement is:

$$\mathcal{C}[\![\textsf{if } e_0 \textsf{ then } e_1 \textsf{ else } e_2]\!]\rho\kappa \quad = \quad \mathcal{C}[\![e_0]\!]\rho\kappa'$$

where κ' is the piece of computation performed after the evaluation of the test condition; this continuation takes the result of the evaluation of the test condition as parameter. Hence κ' is:

$$\begin{array}{rl} \kappa' = \lambda r \in \mathit{Result}\ .\ \mathit{kcase}\ \ r\ \ \mathit{of} & \mathbb{T}(\mathit{true}) \to \mathcal{C}[\![e_1]\!]\rho\kappa \\ \mid & \mathbb{T}(\mathit{false}) \to \mathcal{C}[\![e_2]\!]\rho\kappa \end{array}$$

We use here the *kcase* construct, which is similar to the *ecase* construct, except that it calls the halt continuation for bottom or for the unmatched cases:

- if the argument is $\perp$, then *kcase* returns $\kappa_0(\perp)$;
- for an argument that doesn't match any of the patterns, *kcase* returns $\kappa_0(\mathit{error})$;

We can write the denotation of the if construct more compactly using a higher-order function *check_bool* that filters out non-boolean arguments:

$$\mathcal{C}[\![\textsf{if } e_0 \textsf{ then } e_1 \textsf{ else } e_2]\!]\rho\kappa = \mathcal{C}[\![e_0]\!]\rho(\mathit{check\_bool}\,(\lambda b \in \mathbb{T}.\ \mathit{cond}(b, \mathcal{C}[\![e_1]\!]\rho\kappa, \mathcal{C}[\![e_2]\!]\rho\kappa)))$$

The (non-obvious) signature and definition of *check_bool* is:

$$\begin{array}{rcl} \mathit{check\_bool} & : & (\mathbb{T} \to \mathit{Result}) \to \mathit{Cont} \\ & = & (\mathbb{T} \to \mathit{Result}) \to \mathit{Result} \to \mathit{Result} \\ \mathit{check\_bool} & = & \lambda f.\ \lambda r.\ \mathit{kcase}\ \ r\ \ \mathit{of}\ \ \mathbb{T}(t) \to f\ t \end{array}$$

2.3 Arithmetic expressions

For arithmetic expressions, we define the following semantics:

$$\mathcal{C}[\![e_0 + e_1]\!]\rho\kappa \quad = \quad \mathcal{C}[\![e_0]\!]\rho\kappa'$$

where κ' is the continuation that takes the result of evaluating the first argument and performs the rest of the computation. Hence:

$$\kappa' \quad = \quad \lambda r.\ \mathit{kcase}\, r\ \ \mathit{of}\ \ \mathbb{Z}(n_0) \to \mathcal{C}[\![e_1]\!]\rho\kappa''_{n_0}$$

where κ''_{n_0} is the continuation that takes the result of evaluating the second argument and performs the rest of the computation. This continuation is parameterized on n_0, the result of the evaluation of the first argument:

$$\kappa''_{n_0} \quad = \quad \lambda r.\ \mathit{kcase}\, r\ \ \mathit{of}\ \ \mathbb{Z}(n_1) \to \kappa(n_0 + n_1)$$

Intuitively, the evaluation proceeds as follows. We first evaluate the left operand. We then pass the result to the continuation κ', which evaluates the right operand and passes the result to κ'', which performs the sum. Finally, we pass the sum to the original continuation κ, which computes the final result.

We can compactly write the denotation of arithmetic expressions using filtering functions:

$$\mathcal{C}[\![e_0 + e_1]\!]\rho\kappa = \mathcal{C}[\![e_0]\!]\rho(\mathit{check\_int}\,(\lambda n_0 \in \mathbb{Z}.\ \mathcal{C}[\![e_1]\!]\rho(\mathit{check\_int}\,(\lambda n_1 \in \mathbb{Z}.\ \kappa(n_0 + n_1)))))$$

2.4 Other constructs

The semantics for pairs are defined in a similar way (where π_1 and π_2, recall, are projection functions):

$$\begin{aligned}
\mathcal{C}[\![\langle e_0, e_1 \rangle]\!]\rho\kappa &= \mathcal{C}[\![e_0]\!]\rho(\mathit{check_val}\,(\lambda v_0 \in \mathit{Value}. \\
&\qquad \mathcal{C}[\![e_1]\!]\rho(\mathit{check_val}\,(\lambda v_1 \in \mathit{Value}.\ \kappa(v_0, v_1))))) \\
\mathcal{C}[\![\mathsf{fst}\ e]\!]\rho\kappa &= \mathcal{C}[\![e]\!]\rho(\mathit{check_pair}\,(\lambda p \in \mathit{Pair}.\ \kappa(\pi_1\ p))) \\
\mathcal{C}[\![\mathsf{snd}\ e]\!]\rho\kappa &= \mathcal{C}[\![e]\!]\rho(\mathit{check_pair}\,(\lambda p \in \mathit{Pair}.\ \kappa(\pi_2\ p)))
\end{aligned}$$

For an application, we evaluate the function and the value of its argument. Then we invoke the function, passing the value and the current (dynamic) continuation as parameters:

$$\begin{aligned}
\mathcal{C}[\![e_0\ e_1]\!]\rho\kappa_{\mathrm{dyn}} &= \mathcal{C}[\![e_0]\!]\rho(\mathit{check_fun}\,(\lambda f \in \mathit{Function}. \\
&\qquad \mathcal{C}[\![e_1]\!]\rho(\mathit{check_val}\,(\lambda v \in \mathit{Value}.\ f\ v\ \kappa_{\mathrm{dyn}}))))
\end{aligned}$$

The denotation of an abstraction produces a function which takes a value and a dynamic continuation as parameters. The resulting function is then passed to the continuation enclosing the abstraction:

$$\mathcal{C}[\![\lambda x.\ e]\!]\rho\kappa = \kappa(\lambda v \in \mathit{Value}.\ \lambda\kappa_{\mathrm{dyn}} \in \mathit{Cont}.\ \mathcal{C}[\![e]\!]\rho[x \to v]\kappa_{\mathrm{dyn}})$$

As before, the semantics of the rec construct are similar to those of the abstraction except that they use the fixed point operator:

$$\mathcal{C}[\![\mathsf{rec}\ y\ (\lambda x.\ e)]\!]\rho\kappa = \kappa(\mathrm{fix}(\lambda f.\ \lambda v \in \mathit{Value}.\ \lambda\kappa_{\mathrm{dyn}} \in \mathit{Cont}.\ \mathcal{C}[\![\lambda x.\ e]\!]\rho[y \to f]\kappa_{\mathrm{dyn}}))$$