Finite representation of the computation tree for PCF

William Blum

May 23, 2008

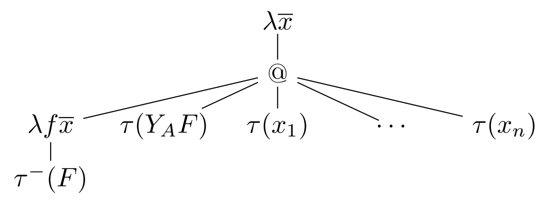
Abstract

It would be nice if we could have a finite transition system (such as a higher-order DPDA) that recognizes the set of traversals of the compuation tree for some fragment of PCF. The idea would be to have an automaton whose states are the nodes of the computation tree. The problem with PCF is that computation tree are infinite due to the presence of the Y combinator. Here we present an alternative definition of computation tree for PCF.

The original definition as a least-upper bound

The computation tree of a PCF term, written $\tau(M)$, is defined as the least upper bound of the computation trees of its approximants: Take a term of the form $Y_A F$ where $F : A \rightarrow A$ and $A = (A_1, \ldots, A_n, o)$. We define the eta-normal form $\lceil Y_A F \rceil$ of $Y_A F$ to be the unique infinite term satisfying the recursive equation $X \equiv \lambda \overline{x} : \overline{A}. \lceil F \rceil X \lceil x_1 \rceil \ldots \lceil x_n \rceil$ where $\overline{x} = x_1 \ldots x_n$ are fresh variables. Similarly, the computation tree $\tau(Y_A F)$ is given by the unique infinite tree verifying the recursive equation:

$\tau(Y_A F) =$



for some fresh variables $\overline{x} : \overline{A}$ and where $\tau^{-}(F)$ denotes $\tau(F)$ with the root removed. This implies that $\tau(Y_A F) = \tau(F(Y_A F))$.

Example 0.1. *Take $F \equiv \lambda f x . f x$. We have:*

$$\lceil Y F \rceil \equiv \lambda x . \lceil F \rceil \lceil Y F \rceil \lambda . x \equiv \lambda x . (\lambda f x . f \lambda . x) \lceil Y F \rceil \lambda . x$$

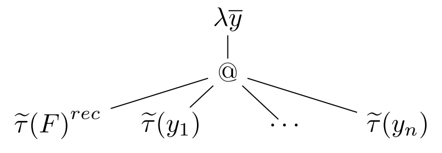
therefore its computation tree is the solution of the equation $\tau = \lambda x . (\lambda f x . f \lambda . x) \ \tau \ (\lambda . x)$ in τ.

We write $\mathcal{T}rav_\tau(Y_A F)$ to denote the set of traversals of the computation tree $\tau(Y_A F)$.

A finite representation

Since the subtree $\tau(Y_A F)$ in the second branch of the application node in $\tau(Y_A F)$ is the same tree as the main computation tree itself, we can identify them and remove the first one from the computation tree. This gives rise to a finite version $\widetilde{\tau}(Y_A F)$ of the computation tree:

$\widetilde{\tau}(Y_A F) =$



where the exponent rec means that the root of the subtree $\widetilde{\tau}(F)$ has a mark indicating that it is a special recursion lambda-node. The notion of traversal can be changed accordingly: whenever a traversal meets a recursion variable f in the subtree $\tau(F)$ then instead of jumping to the second child of the application node, it jumps to the root of the main tree. (Thus with this notion of traversal, the root can occur multiple times in the same traversal.) The definition of the mapping ψ from nodes to moves remains consistent with this notion of computation tree, and the game-semantic correspondence still holds:

$$\psi : \mathcal{T}rav_{\widetilde{\tau}}(Y_A F)^{\upharpoonright\circledast} \xrightarrow{\cong} [\![Y_A F]\!]$$

where $\mathcal{T}rav_{\widetilde{\tau}}(M)^{\upharpoonright\circledast}$ denotes the set of reductions of traversals of the computation tree $\widetilde{\tau}(M)$.

An alternative finite representation

In order to handle the Y combinators, we introduce a new kind of abstraction node. Such node is labelled $\lambda\overline{x}$ where each variable occurring in $\overline{x}$ is either a standard variable or a "recursion variable". Recursion variables are put into parenthesis in order to distinguish them from standard ones. The set L of node labels is given by the grammar:

$$\begin{aligned} L &::= \lambda\ V_{abs}^{*} \mid @ \mid \mathcal{V} \\ V_{abs} &::= \mathcal{V} \mid (\mathcal{V}) \end{aligned}$$

for some countable set $\mathcal{V}$ of variables.

A term of the form $Y_A(\lambda f : A.M)$ where $M : A$ will be written $\lambda(f).M$. This notation also accounts for ground type recursion, for instance the `while` operator of Idealized Algol defined as `while` C `do` $I \equiv Y(\lambda f.$`cond` C `skip` (`seq` $I f$)) is rewritten as $\lambda(f).$`cond` C `skip` (`seq` $I f$).

To construct the computation tree of a PCF term, we first compute the η-long normal form as usual except for the Y combinator case which is treated as follows. For each term $Y_A F$ where $F : A \to A$, let $\lceil F \rceil = \lambda f\overline{x}.M$ where $f : A$ and $M : o$. The *finite eta-normal form* of $Y_A F$ is defined to be $\lceil Y_A F \rceil = \lambda(f)\overline{x}.M$. The ***finite computation tree*** is then obtained from the finite eta-normal form in the standard way:

$$\begin{tikzcd}[row sep=small] \hat{\tau}(Y_A F) = & \lambda(f)\overline{x} \arrow[d, no head] \\ & \tau^{-}(F) \end{tikzcd}$$

The definition of the order of an abstraction node does not change: it is the order of the term represented by the subtree rooted at this node. In other word, the order of $\lambda\overline{x}$ is defined as the order of $\lambda\overline{y}$ where $\overline{y}$ is the sublist of $\overline{x}$ obtained by removing all the recursion variables in parenthesis.

The numbering of variables bound in a λ-node labelled $\lambda\overline{x}$ is as follows: the i^{th} standard variable in $\overline{x}$ is denoted by i and the the i^{th} recursion variable in $\overline{x}$ is denoted by (i). The links in a justified sequence of nodes are labelled accordingly.

We can now define a notion of traversals adapted to this new kind of computation tree as follows. All the traversal rules are kept unmodified. The recursion variables in the λ-nodes are automatically ignored by the rules since such variables are numbered differently from standard variables. In particular, the (Var) rules only applies to non-recursion variables. We add a new rule to handle recursion variable:

($\mathsf{Var}_{\mathsf{rec}}$) If $\begin{tikzcd}[column sep=small] t' \cdot n \cdot & \lambda\overline{x} & \ldots & f_i \arrow[ll, bend right, "(i)"'] \end{tikzcd}$ is a traversal for some *recursion* variable f_i bound by $\lambda\overline{x}$ then so is $\begin{tikzcd}[column sep=small] t' \cdot n \cdot & \lambda\overline{x} & \ldots & f_i \arrow[ll, bend right, "(i)"'] & \cdot\ \lambda\overline{x} \arrow[l, bend right] \end{tikzcd}$.

With the original definition of computation tree, the proof of the Correspondence Theorem for PCF was straightforward: it was done by a reduction to the simply-typed lambda calculus case using the technique of approximants. With the finite version of computation trees, it is not obvious to see which uncovering of moves gives rises to a correspondence with the set of traversals. However it is easy to show that the set of traversal reductions of the finite computation tree is equal to the set of traversal reductions of the standard computation tree. Therefore the Correspondence Theorem still holds.

Proof of soundness

We now show that the finite representation $\hat{\tau}$ of a PCF computation tree together with its set of traversals is equivalent to the original presentation.

We define a partial map α from the set of nodes of $\widetilde{\tau}(Y_A F)$ to the set of nodes of $\hat{\tau}(Y_A F)$ as follows. Each node labelled by a variable x_i is mapped to the child of the root of the sub-tree $\tau(y_i)$ in $\widetilde{\tau}(Y_A F)$ labelled with y_i. Also, α maps each node hereditarily enabled by x_i to the corresponding node hereditarily enabled by y_i in $\widetilde{\tau}(Y_A F)$.

Lemma 0.2. $\alpha(\mathcal{T}rav_{\hat{\tau}}(Y_A F)^{\upharpoonright \circledast}) = \mathcal{T}rav_{\widetilde{\tau}}(Y_A F)^{\upharpoonright \circledast}$.

Proof. We prove $\mathcal{T}rav_{\widetilde{\tau}}(Y_A F)^{\upharpoonright \circledast} \subseteq \alpha(\mathcal{T}rav_{\hat{\tau}}(Y_A F)^{\upharpoonright \circledast})$ by induction on the traversal rules for $\widetilde{\tau}$. The base case is trivial. For the step case we consider the (Var) rule, the other cases being trivial.

Take the tree $\widetilde{\tau}$, when traversing the sub-tree $\tau^-(F)$, each time a variable node hereditarily justified by $\lambda(f)\overline{x}$ is encountered, the next step consists in taking a jump to a node in $\tau(y_i)$ hereditarily justified by the root of $\tau(y_i)$. Then the traversal of $\tau(y_i)$ continues until it reaches a variable node z hereditarily justified by the root of $\tau(y_i)$ after which it jumps back to a node $\lambda\overline{\eta}$ in $\tau^-(F)$. $\lambda\overline{\eta}$ is the child of a previously visited node p hereditarily justified by $\lambda(f)\overline{x}$ and from which a previous jump to $\tau(y_i)$ was taken:

$$t = t_1 \cdot \underline{p} \cdot \lambda\overline{\nu} \cdot s \cdot z \cdot \underline{\lambda\overline{\eta}}$$

for some traversal t_1 of $\widetilde{\tau}$, sequence of nodes s of even-length and where the nodes belonging to the sub-tree $\tau^-(F)$ are underlined.

By construction of $\tau(y_i)$, each node of $\tau(y_i)$ is either hereditarily justified by the root of $\tau(y_i)$ or by the root of $\widetilde{\tau}$. Consequently,

$$t \upharpoonright r = (t_1 \upharpoonright r) \cdot (s \upharpoonright r) \ .$$

If $s = \epsilon$ then we conclude immediately using the induction hypothesis. Suppose that $|s| \geq 2$ then $s = u \cdot \ldots \cdot v$ where u the child of $\lambda\overline{\nu}$ and v is the parent of z. Necessarily, u and v are not hereditarily justified by the root of $\tau(y_i)$ and therefore are hereditarily justified by the root of $\widetilde{\tau}$. Hence:

$$t \upharpoonright r = (t_1 \upharpoonright r) \cdot u \cdot v \ .$$

By the induction hypothesis we have $t_1 \upharpoonright r = \alpha(t'_1 \upharpoonright r)$ for some traversal t'_1 of $\hat{\tau}$.

The nodes p and $\lambda\overline{\eta}$ belong to the sub-tree $\tau^-(F)$ of $\widetilde{\tau}$. Let us write p' and $\lambda\overline{\eta}'$ to denote the corresponding nodes in the sub-tree $\tau^-(F)$ of $\hat{\tau}$. We have $\alpha(p') = u$ and $\alpha(\lambda\overline{\eta}') = v$.

After the traversal $t'_1 \upharpoonright r$ of $\hat{\tau}$, it is possible to visit the node p' using the same rule that has permitted to visit p in $\widetilde{\tau}$ after $t_1 \upharpoonright r$. After this step, since p' is hereditarily justified by the root of $\hat{\tau}$, it is possible to visit the child node $\lambda\overline{\eta}'$ of p' using the (InputVar) rule. Hence

$$(t'_1 \upharpoonright r) \cdot p' \cdot \lambda\overline{\eta}' \in \mathcal{T}rav(\hat{\tau}(Y_A F))$$

and $\alpha((t'_1 \upharpoonright r) \cdot p' \cdot \lambda\overline{\eta}') = t \upharpoonright r$.

The proof of $\mathcal{T}rav_{\widetilde{\tau}}(Y_A F)^{\upharpoonright \circledast} \supseteq \alpha(\mathcal{T}rav_{\hat{\tau}}(Y_A F)^{\upharpoonright \circledast})$ is similar. □

Although α is not injective (it maps two different nodes of $\hat{\tau}$ labelled by the same variable to the same node of $\widetilde{\tau}$), its node-wise extension to the set of reductions of traversals is injective.

The proof is by an easy induction on the traversal rules. Hence α defines an isomorphism from $\mathcal{T}rav_{\tilde{\tau}}(Y_A F)^{\upharpoonright \circledast}$ to $\mathcal{T}rav_{\hat{\tau}}(Y_A F)^{\upharpoonright \circledast}$, and consequently we have:

$$\psi \circ \alpha : \mathcal{T}rav_{\hat{\tau}}(Y_A F)^{\upharpoonright \circledast} \xrightarrow{\cong} [\![Y_A F]\!]$$

which shows that the finite representation $\hat{\tau}$ of the computation tree is sound in the sense that the game-semantics characterisation sill holds.