ABSTRACT: The use of continuations in the definition of programming languages has gained considerable currency recently, particularly in conjunction with the lattice-theoretic methods of D. Scott. Although continuations are apparently needed to provide a mathematical semantics for non-applicative control features, they are unnecessary for the definition of a purely applicative language, even when call-by-value occurs. This raises the question of the relationship between the direct and the continuation semantic functions for a purely applicative language. We give two theorems which specify this relationship and show that, in a precise sense, direct semantics are included in continuation semantics.

The heart of the problem is the construction of a relation which must be a fixed-point of a non-monotonic "relational functor." A general method is given for the construction of such relations between recursively defined domains.

Two Definitions of the Same Language

The use of continuations in the definition of programming languages, introduced by Morris (1) and Wadsworth, (2) has gained considerable currency recently, (3) particularly in conjunction with the lattice-theoretic methods of D. Scott. (4) Continuations are apparently needed to provide a mathematical semantics for non-applicative control features such as labels and jumps, Landin's J-operator, (5) or Reynolds' escape functions. (3) However, a purely applicative language, even including call-by-value (to the author's chagrin (3)), can be defined without using continuations. In this paper we will investigate the two kinds of definitions of such a purely applicative language, and prove that they satisfy an appropriate relationship.

The language which we consider is a variant of the lambda calculus which permits both call-by-name and call-by-value. Let $V$ be a denumerably infinite set of variables. Then $R$, the set of expressions, is the minimal set satisfying:

1. If $x \in V$, then $x \in R$.
2. If $r_1, r_2 \in R$, then $(r_1 r_2) \in R$.
3. If $x \in V$ and $r \in R$, then $(\lambda x. r) \in R$.
4. If $x \in V$ and $r \in R$, then $(\lambda_{val} x. r) \in R$.

---

† Work supported by Rome Air Force Development Center Contract No. 30602-72-C-0281, ARPA Contract No. DAHC04-72-C-0003, and National Science Foundation Grant GJ-41540.
Expressions of the fourth form are meant to denote functions which call their arguments by value.

Our first definition uses a typical Scott model of the lambda calculus in which some but not all domain elements are functions. Let \( P \) be any domain of "primitive values." Then let \( D \) be the minimal domain satisfying the isomorphism
\[
D = P + \langle D \to D \rangle
\]
where \( \to \) denotes the formation of a domain of continuous functions, and \( + \) denotes the formation of a separated sum. More precisely, \( D_1 + D_2 \) is the domain
\[
\{1, T\} \cup \{<1, x_1> | x_1 \in D_1\} \cup \{<2, x_2> | x_2 \in D_2\},
\]
with the partial ordering \( x \leq y \) iff
\[
x = 1 \text{ or } y = T \text{ or } x = <1, x_1> \text{ and } y = <1, y_1> \text{ and } x_1 \leq y_1 \text{ or } x = <2, x_2> \text{ and } y = <2, y_2> \text{ and } x_2 \leq y_2.
\]

We introduce the following classification, selection, and embedding functions for the lattice sum:
\[
\tau \in D + \text{Bool} \\
\iota_P \in D \to P \\
\rho_P \in P \to D \\
\rho_F \in (D + D) \to D
\]
which satisfy
\[
\iota_P \cdot \rho_P = \iota_D \\
\iota_F \cdot \rho_F = \iota_D
\]
\[
\lambda x \in D. \text{cond}(\tau(x), \rho_P(\iota_P(x)), \rho_F(\iota_F(x))) = \iota_D.
\]

Here \( \text{Bool} \) denotes the usual four-element domain of truth values, \( \iota_D \) denotes the identity function on a domain \( D \), and \( \text{cond} \) denotes the conditional function which is doubly strict in its first argument (i.e., which maps \( \bot \) into \( \bot \) and \( T \) into \( T \)).

If we take \( D \) to be the set of values described by our language, then the meaning of an expression is a continuous function from environments to values, where an environment is a function from variables to values. More precisely, the meaning of expressions is given by a function \( \mathcal{M} \in R \to D^V \to D \), where the environment domain \( D^V \) is the set of functions from \( V \) to \( D \), partially ordered by the pointwise extension of the partial ordering on \( D \). The following equations define \( \mathcal{M} \) for each of the cases in the syntactic definition of \( R \):
\[
(1) \quad \mathcal{M}[x](e) = e(x) \\
(2) \quad \mathcal{M}[\lambda x. r_1 \ r_2](e) = \text{cond}(\tau(\mathcal{M}[r_1](e)), \bot, \iota_P(\mathcal{M}[r_1](e))(\mathcal{M}[r_2](e))) \\
(3) \quad \mathcal{M}[\lambda x. r](e) = \rho_P(\lambda a \in D. \mathcal{M}[r][e]\mid x\mid a)) \\
(4) \quad \mathcal{M}[\lambda \text{val} x. r](e) = \rho_F(\lambda (\lambda a \in D. \mathcal{M}[r][e]\mid x\mid a))
\]
where \( a \in (D \to D) + (D \to D) \) is the function such that \( a(f)(\bot) = 1 \), \( a(f)(T) = T \), and \( a(f)(a) = f(a) \) otherwise.

Here \( [e]\mid x\mid a \) denotes the environment \( \lambda y \in V. \text{if } y = x \text{ then } a \text{ else } e(y) \).

The only thing surprising about this definition is the fourth case. Essentially, we are interpreting a call-by-value function as the retraction of the