

1 Lambda calculus

- Define an applicative-order operational semantics for the lambda calculus that during evaluation never produces an intermediate expression with holes in scope, assuming that none exist in the original expression to be evaluated.
- An equivalence relation has the properties of reflexivity ($a \sqsubseteq a$), symmetry ($a \sqsubseteq b \iff b \sqsubseteq a$) and transitivity ($a \sqsubseteq b \wedge b \sqsubseteq c \Rightarrow a \sqsubseteq c$). Show that alpha-equivalence is an equivalence relation on lambda calculus expressions.

2 Fixed Points

Consider the expression

$$F = \lambda f : Z_{\perp} \rightarrow Z_{\perp} . \lambda x : Z_{\perp} . \text{if } (x < 2) \text{ then } 1 \text{ else } x * f(x - 1)$$

We can define the factorial function as

$$FACT \stackrel{def}{=} \text{fix}(F) = \bigsqcup_{n \in \omega} F^n(\perp)$$

- Expand $F^2(\perp)$ and $F^3(\perp)$ into normal form for some reasonable definition of normal form.
- For what argument values x do these two functions compute factorial properly?
- Prove inductively the set of argument values x for which the function $F^n(\perp)$ computes factorial correctly.

3 Continuity

- Winskel, exercise 4.14
- Winskel, exercise 5.12 (i)
- Winskel, exercise 8.6

4 Domain theory

- Winskel, exercise 8.12. Show that these truth tables define continuous functions.
- Show that the in and cases functions associated with sum domains are continuous without relying on the desugaring of cases given on page 135 of Winskel.
- Provide the missing step in the proof given in class that the *fix* operator is continuous. Assuming that the *fix* operator has type $(D \rightarrow D) \rightarrow D$, show that

$$\text{fix} = \lambda f : D \rightarrow D . \bigsqcup_{n \in \omega} f^n(\perp) = \bigsqcup_{n \in \omega} \lambda f : D \rightarrow D . f^n(\perp)$$

- Consider the function

$$\text{strict} : D_{\perp} \times (D \rightarrow D_{\perp}) \rightarrow D_{\perp} \stackrel{def}{=} \lambda v : D_{\perp}, f : D \rightarrow D_{\perp} . \text{if } v = \perp \text{ then } \perp \text{ else } f(v)$$

We can use this function to construct denotational semantics if it is continuous.

- (i) Assuming that D is a cpo, show that *strict* is continuous.
- (ii) Use *strict* with $D = Z$ to simplify the denotational semantics for a function call expression in call-by-value REC^+ .

e. Consider the function

$$\text{minall} : (\omega_{\perp} \rightarrow \omega_{\perp}) \rightarrow \omega_{\perp} = \lambda f : \omega_{\perp} \rightarrow \omega_{\perp} . \min_{y \in \omega} f(y)$$

In this definition, ω denotes the whole numbers $(0,1,2,\dots)$, and the function \min gives the smallest number in all of the $f(y)$, or \perp if $f(y) = \perp$ for some integer y . Show that this function is monotonic but not continuous.

5 Parameter Passing

- a. Give a single REC^+ program that has a different meaning in each of call-by-name, call-by-value, and call-by-denotation.
- b. Why can't we write an expression that can be tested at run time to determine which of the parameter passing styles the language is using?

6 Naming

Suppose we add modules to the F_0 language, with the the following new expression forms. The resulting language we will call F_m :

$e ::=$...	(Same as F_0)
	(module $(x_1 e_1) \dots (x_n e_n)$)	(Create a module)
	(select $e' x$)	(Look up x in e')
	(with $e_1 e_2$)	(Evaluate e_2 in extended environment)
	(extend $e_1 e_2$)	(Extend one environment with another)

The module expression creates a new module that binds each of the identifiers x_i to the result of the corresponding expression e_i . The select expression takes a module as its first argument and looks up the value bound to the identifier x_0 . The with expression evaluates the expression e_2 in an environment in which all of the identifiers bound by the module in e_1 are bound accordingly.

In this problem we will extend the denotational semantics for F_0 to describe these new constructs. Values of type module will be considered to be *environment extenders*: members of the domain $\text{Env} \rightarrow \text{Env}$, which extend a given environment with a set of possibly new bindings. We could also define modules as members of the domain Env , but that is not the approach we will take here.

- a. Make all changes to the domain equations of F_0 necessary to support the addition of module values.
- b. Define the semantic function $\mathcal{C}[[e]]$ for the four new F_m expressions. Does its definition need to change for any F_0 expressions?

7 State

The $F!$ language defined in class adds the notion of a *store* that binds *locations* to *values*. There are many imperative structures that we did not add to this language. In this problem we will define the operational and denotational semantics for the $F!$ language extended with support for mutable arrays of fixed size. We will allow four new syntactic forms in this language $F!_a$:

$e ::=$...	(Same as $F!$)
	(array $e_n e_v$)	(Create an array of e_n elements)
	(get $e_a e_i$)	(Get the i th element of the array e_a)
	(set $e_a e_i e_v$)	(Set the i th element of the array e_a)
	(length e_a)	(Get the length of the array e_a)

The result of the array expression is a new array value all of whose elements are bound to the result of evaluating e_v . The result of the set expression is the result of evaluating e_v . The store is also updated to reflect the update to the array.

- a. Assuming array values are members of the domain $Z \times (Z \rightarrow Loc + IndexError)$, where *IndexError* is the result produced by an out-of-bounds array index, what other changes need to be made to the domain equations for $F!$?
- b. Define the semantic function $\mathcal{C}[[e]]$ for the four new kinds of expressions.
- c. Now consider adding the following expression to the language:

$$\begin{array}{ll}
 e & ::= \dots & \text{(Same as } F!_a) \\
 & | \text{ (resize } e_a \ e_n \ e_v) & \text{(Resize array)}
 \end{array}$$

This expression changes the length of the array e_a to be $\max(e_n, 0)$. If any new array cells are created, they are filled in with the result of evaluating e_v .

How can the domain equations be changed to allow this new kind of expression?