Cornell University
Computer Science 611

Problem Set 3 DUE: 18 October 2004

Instructions

Turn in the written part of the assignment by 5PM on the due date in Upson 4119.

1. State

   Writing programs that deal gracefully with failure can be difficult in the usual imperative style, especially in a distributed environment where failure is unavoidable. A failed subcomputation can leave the the system in an unpredictable state where important invariants are violated. A *transaction* mechanism is one way to deal with this. Consider the following extension to uML$_!$:

   $$e ::= \ldots \mid \mathsf{transaction}\ e \mid \mathsf{abort}\ e$$

   The idea is that a transaction expression evaluates e and if e evaluates successfully, the whole expression has the same result as e. However, if the evaluation of e leads to the evaluation of an expression abort e', the most recently started transaction halts and its result is the result of evaluating e'. Since this is an extension to uML$_!$, there is a state that might be modified by expressions that are evaluated. When a transaction completes successfully, the resulting state is the same as the state after evaluating e. However, when a transaction is aborted, the resulting state then reverts to the state that existed at the beginning of the transaction. Transactions can be nested inside other transactions, so an abort only aborts the innermost ongoing transaction.

   (a) Give an operational semantics for this language. You may uniformly lift rules given in class for uML or uML$_!$

   (b) Give a direct state-passing translation from this extended language to uML.

   (c) Writing a formal semantics for this mechanism exposes some ambiguities in the above description, which entail design choices. What are these choices, how did you make them, and where do these choices show up in the semantics you wrote? Discuss briefly.

2. CPS semantics

   In class we saw a CPS translation of a simple exception mechanism, which exposed the existence of a dynamic exception handler environment. Translations were given for the raise and try...catch constructs. Some languages, such as Lisp, Modula-3, and Java, have an additional try...finally construct (in Lisp it's called unwind-protect). Let us give a semantics for this construct. Suppose we add an expression try e_1 finally e_2, where the result of the expression is the result of e_1, but the code in e_2 is executed even if e_1 causes an exception to be raised. In addition, if e_1 raises an exception, the whole try...finally raises the same exception once e_2 completes.

   (a) Why might try...finally be useful to have in a programming language?

   (b) Extend the exception translation given in class to support try...finally (without breaking the translations for raise and try...catch).

3. CPOs

   In this problem you will prove some facts about the mathematical structure P_ω, which can serve as a model for the untyped lambda calculus. P_ω is a pair $(2^{\mathbb{N}}, \subseteq)$, where $2^{\mathbb{N}}$ is the set of all subsets of $\mathbb{N}$.

   (a) Prove that P_ω is a CPO (in fact, it is a complete lattice). What are the least and greatest upper bound operations in P_ω?

(b) Characterize continuous functions from P_ω to P_ω (that is, write what they are and prove that these are exactly the continuous functions).

(c) **(Bonus question)** Try to come up with the 1–1 continuous function from $[P_\omega \to P_\omega]$ to P_ω. You may find the following functions useful:

- The bijective pairing function $p : \mathbb{N}^2 \to \mathbb{N}$.
- The bijective finite set coding function $s : Fin(\mathbb{N}) \to \mathbb{N}$, where $Fin(\mathbb{N})$ denotes the set of all finite sets of natural numbers.

You don't need to know what these functions are, but for the curious, here are possible definitions:

- $p(x, y) = \frac{(n+m)(n+m+1)}{2} + m$.
- $s(\{k_0, \ldots, k_l\}) = \sum 2^{k_i}$.

4. Approximation

An element x of a CPO *approximates* another element y, written $x \ll y$, if all chains z_n whose LUB is at least y contain an element that is at least x:

$$y \sqsubseteq \bigsqcup_{n \in \omega} z_n \implies \exists n \in \omega . x \sqsubseteq z_n$$

An element of a CPO is *compact* (or *finite*) if it approximates itself.

(a) Show that $x \ll y \implies x \sqsubseteq y$.

What are the compact elements of these domains?

(b) natural numbers ω with discrete ordering

(c) $\omega \cup \{\infty\}$ with $\leq$ ordering ($\forall n . n \leq \infty$)

(d) $\mathbb{Z} \to \mathbb{Z}$ with pointwise ordering

(e) $\mathbb{Z} \to \mathbb{Z}_\perp$ with pointwise ordering

5. Domain isomorphism

A set S is *isomorphic* to another set S' if there is a one-to-one and onto function (a bijection) mapping S to S'. Similarly, a domain (for our purposes, a CPO) D is isomorphic to a domain D' if there is a *continuous* bijection mapping D to D'. That is, the two domains must have not only corresponding elements but also corresponding structure. Because the function is continuous, it preserves not only ordering but also least upper bounds (suprema). So it is harder to show that domains are isomorphic than it is for sets.

For each of the following pairs of domains, show that they are isomorphic or that they are not.

(a) The domains $D \times \mathbb{U}$ and D for any CPO D (where $\mathbb{U}$ is the unit domain $(\{unit\}, =)$).

(b) The domains $D \to E$ and $\{f \in D_\perp \to E_\perp \mid f(\perp) = \perp\}$ (both ordered pointwise), for any CPO's D and E.

(c) The domains $D \times E \to F$ and $D \to E \to F$ for any CPO's D, E, F.