Final Exam
CS6848, IIT Madras

28-Apr-2012

1. [12] **Flow analysis** Extend the 0-CFA flow analysis to become *flow sensitive*. As a result given the following code, the flow set for the variable `a` at both lines `L1` and `L3` is a singleton set (instead of {`A, B`}) as is the case with 0-CFA.

```
class A { ... };
class B extends A { ... };
L0: A a = new A();
L1: a.foo(x); // flow set {A}
L2: a = new B(y);
L3: a.foo(); // flow set {B}
```

   The grammar for the statement and expressions is given below.

```
Statement ::= if (Exp) { Statement* } else { Statement* }
          ::= id = Exp
          ::= Exp.id(Exp)
Exp       ::= true | false | id | this | new id()
```

   Hints:

   - `id` denotes an identifier.
   - You can assume that each statement is uniquely labeled.
   - To think: Recall the flow set map in 0-CFA. How will it be different here?

2. [5] **Closure conversion** Translate the following scheme code to semantically equivalent C code.

```
(define f (lambda (x, y)
   (let (g (lambda (y) (+ x  y))) g)))
set! a (f 2 3);
set! b (f 4 5);
set! x (a 5); set !y (b 6);
```

3. [8] **Language extension** We will extend the simply typed lambda calculus with a parallel loop.

   Language:

   $$e ::= x|\lambda x.e|e_1 e_2|ref\ e|!e|let\ x = e_1\ in\ e_2|e_1; e_2$$

   extended with

   $$e ::= \cdots |\ \texttt{ploop}\ (x\ e_1)\ e_2$$

   where e_1 evaluates to a number (say n), then the loop creates n threads, each thread executes e_2 for varying value of `x` $(1\ ..\ n)$ and then waits for each of the thread to terminate. Note: `x` is the loop index and may be free in e_2.

   Provide the extensions to the types, values, type system, and operational semantics (for just the `ploop` extension).

   Assume an *atomic consistency* model; when two threads T_1 and T_2 are running in parallel, the final state gives an impression that the order of the execution is T_1 followed by T_2 or T_2 followed by T_1 without any interleaving.

4. [5] **Partial evaluation** For the partial evaluation algorithm discussed in the class, does it terminate? If yes, then prove the termination or else provide a counter example and give ways to enforce termination.

5. [10] **Type system for program analysis** Here is a new expression language that performs file operations.

   ```
   e ::=  e_1; e_2 | Str | open e | close e | read e
   ```

   where,

   - Str is a character string.
   - `open`: opens the file given by the string argument.
   - `close e`: closes the file given by the string argument.
   - `read e`: reads an element from the file given by the string argument.

   Write a type system that guarantees that a well typed program respects the following protocol:

   - a closed file is not read.
   - an opened file is not reopened.
   - a closed file is not closed again.

   You can assume that all files are closed to start with. Examples:

   - `open 'file1'; read 'file1'; close 'file1'` – should type check.
   - `open 'file2'; close 'file2'; read 'file2'` – should not type check.