Midterm Exam
CS6848

07-Mar-2012

1. [4] Write the interpreter code for *dynamic assignments*. Example of dynamic assignment:

```
let x = 4
in let p = lambda (y) (+ x y)
   in (+ (x:= 7 during (p 1))
         (p 2))
```

   During the invocation of `(p 1)`, the value of `x` is set to `7`, and is reverted back to `4` for the evaluation of `(p 2)`; giving the answer $(8 + 6 = 14)$.

2. [2] In the class we have studied small step semantics of simply typed lambda calculus assuming eager evaluation. Write small step semantics for simply typed lambda calculus assuming lazy evaluation.

3. [4] Prove that the following two commands are axiomatically equivalent.

```
1: do c while (b)

2: c;
   if (b) {c};
   while (b) {c}
```

4. [2] Derive the universal pre-condition and universal post conditions. [Hint: use the consequence proof rule.]

5. [8] Prove the type soundness for the simply typed lambda calculus extended with pairs.

   - An expression is derived from the grammar

$$\begin{aligned}
&e \in Expression\\
&e ::= c|(e_1, e_2)|e.1|e.2\\
&c ::= IntegerConstant
\end{aligned}$$

- A value is given by: $v ::= c|(v_1, v_2)$
- Types: $t ::= \mathsf{Int}\ |t_1 \times t_2$

Small step operational semantics using $\rightarrow_V$.
$\rightarrow_V \subseteq Expression \times Expression$

(1) $(Pair\ \beta 1)(v_1, v_2).1 \rightarrow_V v_1$

(2) $(Pair\ \beta 2)(v_1, v_2).2 \rightarrow_V v_2$

(3) $\text{Proj 1}\dfrac{e \rightarrow_V e'}{e.1 \rightarrow e'.1}$

(4) $\text{Proj 2}\dfrac{e \rightarrow_V e'}{e.2 \rightarrow_V e'.2}$

(5) $\text{Eval 1}\dfrac{e_1 \rightarrow_V e_1'}{(e_1, e_2) \rightarrow_V (e_1', e_2)}$

(6) $\text{Eval 2}\dfrac{e_2 \rightarrow_V e_2'}{(v_1, e_2) \rightarrow_V (v_1, e_2')}$

The type rules are given below:

(7) $\text{Pair}\dfrac{A \vdash e_1 : t_1 \qquad A \vdash e_2 : t_2}{A \vdash (e_1, e_2) : t_1 \times t_2}$

(8) $\text{Proj 1}\dfrac{A \vdash e : t_1 \times t_2}{A \vdash e.1 : t_1}$

(9) $\text{Proj 2}\dfrac{A \vdash e : t_1 \times t_2}{A \vdash e.2 : t_2}$

(10) $\vdash c : \mathsf{Int}$

Definitions.

- An expression e is *stuck* if it is not a value and there is no expression e' such that $e \rightarrow_V e'$.
- An expression e *goes wrong* if $\exists e' : e \rightarrow_V^* e'$ and e' is stuck.
- An expression is *well typed* iff there exists a type t such that $\vdash e : t$.

Prove that a well typed expression cannot go wrong.

6. **Bonus** [2] Prove that the following type inference algorithm terminates.

   Input: G: set of type equations (derived from a given program).
   Output: Unification σ

   (a) failure = `false`; $\sigma = \{\}$.
   (b) while $G \neq \phi$ and $\neg$ failure do
      i. Choose and remove an equation e from G. Say $e\sigma$ is $(s = t)$.
      ii. If s and t are variables, or s and t are both Int then continue.
      iii. If $s = s_1 \rightarrow s_2$ and $t = t_1 \rightarrow t_2$, then $G = G \cup \{s_1 = t_1, s_2 = t_2\}$.
      iv. If ($s = \mathsf{Int}$ and t is an arrow type) or vice versa then failure = `true`.
      v. If s is a variable that does not occur in t, then $\sigma = \sigma\ o\ [s := t]$.
      vi. If t is a variable that does not occur in s, then $\sigma = \sigma\ o\ [t := s]$.
      vii. If $s \neq t$ and either s is a variable that occurs in t or vice versa then failure = `true`.
   (c) end-while.
   (d) if (failure = true) then output "Does not type check". Else o/p σ.