Midterm Exam
CS6848

15-Mar-2013

1. [5] **Interpreter** Write an interpreter for the subset of scheme that admits `let`, `lambda`, *application*, `set!` and *statement sequencing* using `begin`. For example, the following code evaluates to 9:

```
(let ((x 3) (y 4))
  (begin
      (set! x 5)
      ((lambda (z) (+ z x)) y)))
```

   You can assume that the we have only integer constants and lambdas as literals. `(begin e1 e2 ...en)` evaluates `e1`, `e2`,..., `en` in that order and returns the value `en`.

2. [5] **Type rules and small step semantics** Write the type rules and small step semantics for the following Java grammar admitting variant objects (similar to the variant records in Pascal and Unions in C).

```
Prog:= (VarClassDecl)* e
VarClassDecl := class id{bool chooseFirst; type1 f1; type2 f2;}
e:= new id(ex, ey) | e.f1 | e.f2 | true | false
type:= id | bool
```

   Here `id` represents any identifier. Each variant class contains three fields. The first field is a boolean field named `chooseFirst` whose value in the object decides if the object contains a value for the first field or the second field. The `new` operator sets the field `f1` (field `f2`) of the newly created object to the value of `ey`, if expression `ex` evaluates to `true` (false). An object `obj` should dereference its `f1` field (`f2` field) only if `obj.chooseFirst` is `true` (`false`). A program returns the value returned by its "main" expression `e`.

3. [5] **Type soundness** Prove the type soundness for following language that admits integer and set minus operations. An expression is derived from the grammar

$$\begin{array}{l|l} e \in Expression & c ::= IntegerConstant \\ e ::= c|s|e_1 - e_2 & s ::= SetOfIntegers \end{array}$$

- A value is given by: $v ::= c|s$ and types are derived from $t ::= \mathsf{Int} \ |Set$

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

(1) $$\frac{e_1 \rightarrow_V e_1'}{e_1 - e_2 \rightarrow_V e_1' - e_2}$$

(2) $$\frac{e_2 \rightarrow_V e_2'}{v - e_2 \rightarrow_V v - e_2'}$$

(3) Integer subtraction
$c_1 - c_2 \rightarrow_V c_3 \ (\lceil c_3 \rceil = \lceil c_1 \rceil - \lceil c_2 \rceil)$

(4) Set minus
$s_1 - s_2 \rightarrow_V s_3 \ (\lceil s_3 \rceil = \lceil s_1 \rceil - \lceil s_2 \rceil)$

(5) Delete element
$s_1 - c \rightarrow_V s_2 \ (\lceil s_2 \rceil = \lceil s_1 \rceil - \{\lceil c \rceil\})$

The type rules are given below:

(6) Integer subtraction
$$\frac{A \vdash e_1 : \mathsf{Int} \qquad A \vdash e_2 : \mathsf{Int}}{A \vdash e_1 - e_2 : \mathsf{Int}}$$

(7) Set Minus
$$\frac{A \vdash e_1 : Set \qquad A \vdash e_2 : Set}{A \vdash e_1 - e_2 : Set}$$

(8) Remove Element
$$\frac{A \vdash e_1 : Set \qquad A \vdash e_2 : Int}{A \vdash e_1 - e_2 : Set}$$

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

(10) $\vdash s : Set$

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.

4. **CPS** [5] Translate the following scheme code to scheme code in imperative-form in a step by step manner.

```
(letrec ((f (lambda (x y)
          (cond ((= x 0) (+ y 1))
                ((= y 0) (f ( (- x 1) 1)))
                (else (f (- x 1) (f x (- y 1))))))))
  (+ (f 1 1) (f 2 2)))
```

5. **Bonus** [2]

- Prove that S and K combinators can be used to derive the I combinator.
- Write the map function in Scheme.