FOR INTERNAL SCRUTINY (date of this version: 24/3/2016)

UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

TYPES AND SEMANTICS FOR PROGRAMMING LANGUAGES

Sunday 1st April 2012

00:00 to 00:00

INSTRUCTIONS TO CANDIDATES

MOCK EXAM MOCK EXAM

Answer any TWO questions

All questions carry equal weight

MOCK EXAM MOCK EXAM

Year 4 Courses

Convener: ITO-Will-Determine
External Examiners: ITO-Will-Determine

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. This question uses the library definition of `list` in Coq, which includes the function ++.

   Here is an informal definition of the predicate `member`.

$$\texttt{member\_here}\frac{}{\texttt{member}\ x\ (x :: xs)} \qquad \texttt{member\_later}\frac{\texttt{member}\ x\ xs}{\texttt{member}\ x\ (y :: xs)}$$

   - Formalise the definitions above. *[9 marks]*
   - Prove both of the following.

$$\texttt{Theorem app\_member\_left} : \forall (X : \texttt{Type})\ (x : X)\ (xs\ ys : \texttt{list}\ X),$$
$$\texttt{member}\ x\ xs \longrightarrow \texttt{member}\ x\ (xs \mathbin{++} ys).$$

$$\texttt{Theorem app\_member\_right} : \forall (X : \texttt{Type})\ (x : X)\ (xs\ ys : \texttt{list}\ X),$$
$$\texttt{member}\ x\ ys \longrightarrow \texttt{member}\ x\ (xs \mathbin{++} ys).$$

[8 marks]

   - Prove both of the following.

$$\texttt{Theorem or\_app\_member} : \forall (X : \texttt{Type})\ (x : X)\ (xs\ ys : \texttt{list}\ X),$$
$$\texttt{member}\ x\ xs \lor \texttt{member}\ x\ ys \longrightarrow \texttt{member}\ x\ (xs \mathbin{++} ys).$$

$$\texttt{Theorem app\_or\_member} : \forall (X : \texttt{Type})\ (x : X)\ (xs\ ys : \texttt{list}\ X),$$
$$\texttt{member}\ x\ (xs \mathbin{++} ys) \longrightarrow \texttt{member}\ x\ xs \lor \texttt{member}\ x\ ys.$$

[8 marks]

2. You will be provided with a definition of a simple imperative language in Coq.

   Consider a construct satisfying the following rules.

   Evaluation:

$$\texttt{E\_RepeatEnd}\frac{\begin{array}{c} c/st \Downarrow st' \\ \texttt{beval}\ st'\ b = \texttt{true} \end{array}}{\texttt{REPEAT}\ c\ \texttt{UNTIL}\ b\ \texttt{END}/st \Downarrow st'}$$

$$\texttt{E\_LoopLoop}\frac{\begin{array}{c} c/st \Downarrow st' \\ \texttt{beval}\ st'\ b = \texttt{false} \\ \texttt{REPEAT}\ c\ \texttt{UNTIL}\ b\ \texttt{END}/st' \Downarrow st'' \end{array}}{\texttt{REPEAT}\ c\ \texttt{UNTIL}\ b\ \texttt{END}/st \Downarrow st''}$$

   Hoare logic:

$$\texttt{hoare\_loop}\frac{\begin{array}{c} \{\{P\}\}\ c\ \{\{Q\}\} \\ Q \wedge \neg b\ \texttt{->>}\ P \end{array}}{\{\{P\}\}\ \texttt{REPEAT}\ c\ \texttt{UNTIL}\ b\ \texttt{END}\ \{\{Q \wedge b\}\}}$$

   - Extend the given definition to formalise the evaluation rules. [*12 marks*]
   - Prove the Hoare rule. You will be provided with proofs of Hoare rules for the simple imperative language that you may modify. [*13 marks*]

3. You will be provided with a definition of simply-typed lambda calculus in Coq.

   Consider constructs satisfying the following rules. (There is no T parameter on `leaf`, so types are not unique.)

   Evaluation:

$$\texttt{ST\_BranchLeft}\ \frac{t_1 \Longrightarrow t_1'}{(\texttt{branch}\ t_1\ t_2\ t_3) \Longrightarrow (\texttt{branch}\ t_1'\ t_2\ t_3)}$$

$$\texttt{ST\_BranchMiddle}\ \frac{\texttt{value}\ v_1 \qquad t_2 \Longrightarrow t_2'}{(\texttt{branch}\ v_1\ t_2\ t_3) \Longrightarrow (\texttt{branch}\ v_1\ t_2'\ t_3)}$$

$$\texttt{ST\_BranchRight}\ \frac{\texttt{value}\ v_1 \qquad \texttt{value}\ v_2 \qquad t_3 \Longrightarrow t_3'}{(\texttt{branch}\ v_1\ v_2\ t_3) \Longrightarrow (\texttt{branch}\ v_1\ v_2\ t_3')}$$

$$\texttt{ST\_TCase}\ \frac{t_1 \Longrightarrow t_1'}{\begin{array}{c}(\texttt{tcase}\ t_1\ \texttt{of leaf} \Rightarrow t_2 \mid \texttt{branch}\ xt\ y\ zt \Rightarrow t_3) \\ \Longrightarrow (\texttt{tcase}\ t_1'\ \texttt{of leaf} \Rightarrow t_2 \mid \texttt{branch}\ xt\ y\ zt \Rightarrow t_3)\end{array}}$$

$$\texttt{ST\_TCaseLeaf}\ \frac{}{(\texttt{tcase leaf of leaf} \Rightarrow t_2 \mid \texttt{branch}\ xt\ y\ zt \Rightarrow t_3) \Longrightarrow t_2}$$

$$\texttt{ST\_TCaseBranch}\ \frac{\texttt{value}\ v_{11} \qquad \texttt{value}\ v_{12} \qquad \texttt{value}\ v_{13}}{\begin{array}{c}(\texttt{tcase}\ (\texttt{branch}\ v_{11}\ v_{12}\ v_{13})\ \texttt{of leaf} \Rightarrow t_2 \mid \texttt{branch}\ xt\ y\ zt \Rightarrow t_3) \\ \Longrightarrow [xt := v_{11}][y := v_{12}][zt := v_{13}]t_3\end{array}}$$

   Typing

$$\texttt{T\_Leaf}\ \frac{}{\Gamma \vdash \texttt{leaf} \in \texttt{Tree}\ T}$$

$$\texttt{T\_Branch}\ \frac{\Gamma \vdash t_1 \in \texttt{Tree}\ T \qquad \Gamma \vdash t_2 \in T \qquad \Gamma \vdash t_3 \in \texttt{Tree}\ T}{\Gamma \vdash (\texttt{branch}\ t_1\ t_2\ t_3) \in \texttt{Tree}\ \ T}$$

$$\texttt{T\_TCase}\ \frac{\begin{array}{c}\Gamma \vdash t_0 \in \texttt{Tree}\ T \\ \Gamma \vdash t_1 \in T' \\ \Gamma, xt \in \texttt{Tree}\ T, y \in T, zt \in \texttt{Tree}\ T \vdash t_2 \in T'\end{array}}{\Gamma \vdash (\texttt{tcase}\ t_0\ \texttt{of leaf} \Rightarrow t_1 \mid \texttt{branch}\ xt\ y\ zt \Rightarrow t_2) \in T'}$$

   - Extend the given definition to formalise the evaluation and typing rules. [*12 marks*]
   - Prove progress. You will be provided with a proof of progress for simply-typed lambda calculus that you may extend. [*13 marks*]