INFR11114 TYPES AND SEMANTICS FOR PROGRAMMING LANGUAGES

Thursday 15th December 2016
14:30 to 16:30

INSTRUCTIONS TO CANDIDATES

Answer QUESTION 1 and ONE other question.

Question 1 is COMPULSORY.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

Year 4 Courses
Convener: I. Murray
External Examiners: A. Cohn, A. Donaldson, S. Kalvala

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY
1. THIS QUESTION IS COMPULSORY

This question uses the library definition of list in Coq, which includes the functions \texttt{rev} and \texttt{app} (\texttt{++}). You may use any facts about \texttt{rev} and \texttt{app} that you find with \texttt{SearchAbout}, except in (b) you may not use \texttt{rev_app_distr}, which is the theorem you are trying to prove.

Here is an informal definition of the predicate \texttt{pal}.

\[
\begin{array}{c}
\text{pal_empty} \quad \text{pal_unit} \quad \text{pal} \quad \text{pal_step} \quad \text{pal}\,xs \\
\text{pal [ ]} \quad \text{pal [ } x \text{ ]} \quad \text{pal ( } [ x ] \text{ ++ } xs \text{ ++ [ } x \text{ ])}
\end{array}
\]

(a) Formalise the definition above. \hfill [8 marks]

(b) Prove the following.

\textbf{Theorem rev_app_distr'}: \forall (X : \text{Type}) \ (xs \ ys : \text{list } X),

\[ \text{rev} \ (xs ++ ys) = (\text{rev} \ ys) ++ (\text{rev} \ xs). \]

\hfill [8 marks]

(c) Prove the following.

\textbf{Theorem pal_rev}: \forall (X : \text{Type}) \ (xs : \text{list } X),

\[ \text{pal } xs \rightarrow xs = \text{rev} \ xs \]

\hfill [9 marks]
2. ANSWER EITHER THIS QUESTION OR QUESTION 3

You will be provided with a definition of a simple imperative language in Coq. Mark any changes you make to the definition with (* !!! *).

Consider a guarded command construct satisfying the following rules:

Evaluation:

\[ \text{EGuardEnd} \]

\[
\begin{align*}
\text{beval } st \ b_1 &= \text{false} \\
\text{beval } st \ b_2 &= \text{false} \\
\text{DO } b_1 \text{ THEN } c_1 \text{ OR } b_2 \text{ THEN } c_2 \text{ OD } \downarrow st
\end{align*}
\]

\[ \text{EGuardLoop1} \]

\[
\begin{align*}
\text{beval } st \ b_1 &= \text{true} \\
\text{c}_1/st \downarrow st' \\
\text{DO } b_1 \text{ THEN } c_1 \text{ OR } b_2 \text{ THEN } c_2 \text{ OD } st' \downarrow st''
\end{align*}
\]

\[ \text{EGuardLoop2} \]

\[
\begin{align*}
\text{beval } st \ b_2 &= \text{true} \\
\text{c}_2/st \downarrow st' \\
\text{DO } b_1 \text{ THEN } c_1 \text{ OR } b_2 \text{ THEN } c_2 \text{ OD } st' \downarrow st''
\end{align*}
\]

Hoare logic:

\[ \text{hoareguarded} \]

\[
\begin{align*}
\{ \{P \land b_1\} \} \ c_1 \ \{ \{P\}\} \\
\{ \{P \land b_2\} \} \ c_2 \ \{ \{P\}\}
\end{align*}
\]

\[
\{ \{P\}\} \ DO \ b_1 \ \text{THEN} \ c_1 \ \text{OR} \ b_2 \ \text{THEN} \ c_2 \ \text{OD} \ \{ \{P \land \neg b_1 \land \neg b_2\}\}
\]

Note that guarded commands are non-deterministic in the case that both \( b_1 \) and \( b_2 \) are true.

(a) Extend the given definition to formalise the evaluation rules. \[ 12 \text{ marks} \]

(b) Prove the Hoare rule. You will be provided with proofs of Hoare rules for the simple imperative language that you may modify. \[ 13 \text{ marks} \]
3. ANSWER EITHER THIS QUESTION OR QUESTION 2
You will be provided with a definition of simply-typed lambda calculus in Coq.
Mark any changes you make to the definition with (* !!! *).
Consider constructs satisfying the following rules:

Values:

\[
\text{value (tdelay } t) \quad \text{v}_{\text{delay}}
\]

Evaluation:

\[
\begin{align*}
\text{tforce (tdelay } t) \Rightarrow t & \quad \text{ST}_{\text{ForceDelay}} \\
tforce t \Rightarrow t' & \quad \text{ST}_{\text{Force}}
\end{align*}
\]

Typing:

\[
\begin{align*}
\Gamma \vdash t \in T & \quad \text{T}_{\text{Delay}} \\
\Gamma \vdash \text{tdelay } t \in \text{TLift } T & \quad \Gamma \vdash t \in T
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash t \in \text{TLift } T & \quad \text{T}_{\text{Force}} \\
\Gamma \vdash \text{tforce } t \in T & \quad \Gamma \vdash t \in T
\end{align*}
\]

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