THE UNIVERSITY OF NEW SOUTH WALES

Mid-Session Examination

Session 1 2006

COMP3161/COMP9161

Concepts of Programming Languages

ANSWERS INCLUDED

Time allowed: 3 hours

Total number of questions: 5

Do not write your answers on the question sheet — only answers in the provided examination booklets will be marked.
(this page intentionally left blank)
**Question 1  20 marks**

The following set of inference rules define a predicate $Tree$ where $n$ is an integer:

\[
\begin{align*}
(3) \quad \text{leaf } n \quad &\text{Tree} \\
(4) \quad t_1 \quad &\text{Tree} \quad t_2 \quad &\text{Tree} \\
\text{node}(t_1, t_2) \quad &\text{Tree}
\end{align*}
\]

a) **4 marks**  Give three example terms $t$ for which $t \quad \text{Tree}$ is inferrable.

Lots (actually infinitely many) possible solutions:

- leaf 1
- node (leaf 3, leaf 1)
- node (node (leaf 0, leaf 2), leaf 7)

b) **7 marks**  Give the derivation of $t \quad \text{Tree}$, where you can choose any term for $t$ which contains at least three leaf subterms.

\[
\begin{align*}
(1) \quad \text{leaf } 1 \quad &\text{Tree} \\
(2) \quad \text{node } (\text{leaf } 1, \text{leaf } 3) \quad &\text{Tree} \\
(1) \quad \text{leaf } 3 \quad &\text{Tree} \\
(1) \quad \text{leaf } 4 \quad &\text{Tree} \\
\text{node } (\text{node } (\text{leaf } 1, \text{leaf } 3), \text{leaf } 4) \quad &\text{Tree}
\end{align*}
\]

c) **9 marks**  Give an example of a derivable rule for $Tree$ and an example for an admissible rule.

Derivable rule: any rule you can obtain by combining rule (1) and (2), for example:

\[
\text{node } (\text{node } (\text{leaf } a, \text{leaf } b), \text{leaf } c)
\]

or

\[
\text{node } (\text{node } (\text{leaf } a, \text{leaf } b), \text{leaf } c)
\]

Admissible rule: for example:

\[
\text{node } (t_1, t_2) \quad &\text{Tree} \\
\text{t1 } &\text{Tree}
\]
**Question 2 20 marks**

Let the transitive closure \(-\rightarrow^*\) of a relation \(-\rightarrow\) be defined as in the lecture by the following two rules:

1. \[x \rightarrow^* x\]
2. \[x \rightarrow y \quad y \rightarrow^* z \quad \Rightarrow \quad x \rightarrow^* z\]

Show by rule induction that \(-\rightarrow^*\) is indeed transitive, i.e. that if \(a \rightarrow^* b\) and \(b \rightarrow^* c\) then \(a \rightarrow^* c\). Reformulate the property first such that it fits the induction scheme \(a \rightarrow^* b \Rightarrow P(a, b)\). For each case, list \(P(a, b)\) and list the assumptions and induction hypothesis (if any).

Reformulation: \(a \rightarrow^* b \Rightarrow (b \rightarrow^* c \Rightarrow a \rightarrow^* c)\), so \(P(a, b) = b \rightarrow^* c \Rightarrow a \rightarrow^* c\)

Base case (rule 1), show \(P(x, x) = x \rightarrow^* c \Rightarrow x \rightarrow^* c\).
This is trivially true: assuming \(x \rightarrow^* c\), \(x \rightarrow^* c\) holds.

Step case (rule 2), show \(P(x, y) \land P(y, z) \Rightarrow P(x, z)\):
assuming \(P(x, y) = y \rightarrow^* c \Rightarrow x \rightarrow^* c\), and \(P(y, z) = z \rightarrow^* c \Rightarrow y \rightarrow^* c\), show that \(P(x, z) = z \rightarrow^* c \Rightarrow x \rightarrow^* c\) holds. Assuming \(z \rightarrow^* c\), we know from \(P(y, z)\) that \(y \rightarrow^* c\). With this and \(P(x, y)\) we know \(x \rightarrow^* c\). Qed.

**Question 3 20 marks**

Please describe briefly the meaning of the following terms (as discussed in the lecture).

See lecture notes

a) **5 marks** \(\alpha\)-equivalence (give an example of two expressions – in the arithmetic expression language or Haskell – which are not identical, but \(\alpha\)-equivalent)

b) **5 marks** The scope of a variable

c) **5 marks** Concrete and abstract syntax of a programming language

d) **5 marks** Static and dynamic semantics of a programming language
Question 4 20 marks

Assume the tree terms defined in Question 1 represent integer values: a tree represents the sum of all numbers at its leaves. Define the evaluations of a tree term to its corresponding integer value using

a) 6 marks big step semantics, in terms of

- the set of evaluable expressions $S$
  \[ S = \{ t \mid t \text{Tree} \} \]
- the set of values $V$, and
  \[ V = \text{int} \]
- the relation $\Downarrow: S \times V$

\[
\begin{align*}
\text{leaf } n & \Downarrow n \\
\text{node}(t_1, t_2) & \Downarrow n_1 + n_2 \\
\text{node}(t, n) & \Downarrow (t', n')
\end{align*}
\]

b) 14 marks small step semantics (i.e., as a transistion system) in terms of

- the states $S$,
  \[ S = \{ (t, n) \mid t \text{Tree}, n \text{Int} \} \]
- the initial states $I$
  \[ I = \{ (t, 1) \mid t \text{Tree} \} \]
- the final states $F$ of the abstract machine
  \[ F = \{ (\text{leaf } 0, n) \mid n \text{Int} \} \]
- the relation $\rightarrow: S \times S$

\[
\begin{align*}
(\text{leaf } m, n) & \rightarrow (\text{leaf } 0, n + m) \\
(\text{node}(\text{leaf } m, t_2), n) & \rightarrow (t_2, n + m) \\
(t_1, n) & \rightarrow (t_1', n') \\
(\text{node}(t_1, t_2), n) & \rightarrow (\text{node}(t_1', t_2), n')
\end{align*}
\]

Note that you might want to start with the big step semantics, as it is easier to define.
Question 5  20 marks

Given the following EBNF

\[
\begin{align*}
\text{Expr} & \rightarrow \ x \mid \text{num} \mid \text{num.num} \mid (\text{Expr}) \mid \text{Expr} + \text{Expr} \mid \text{Expr} \times \text{Expr} \\
& \mid \text{let } \text{id} = \text{Expr} \text{ in } \text{Expr}
\end{align*}
\]

where \text{id} is an identifier, \text{num} an integer constant and \text{num.num} a floating point constant.

a)  4 marks  Define the first order abstract syntax for this language with inference rules.

\[
\begin{array}{c}
\frac{}{x \ \text{Expr}} \quad \frac{}{\text{num} \ \text{Expr}} \quad \frac{}{\text{num.num} \ \text{Expr}} \quad \frac{}{a \ \text{Expr} \ \ b \ \text{Expr}} \quad \frac{}{a \ \text{Expr} \ \ b \ \text{Expr}} \\
\frac{x \ \text{Id}}{
\frac{e_1 \ \text{Expr} \quad e_2 \ \text{Expr}}{\text{let}(x, e_1, e_2) \ \text{Expr}}}
\end{array}
\]

b)  4 marks  Define the higher order abstract syntax for this language. It is enough to show the rule(s) that are different for the first order case.

\[
\begin{array}{c}
\frac{}{e_1 \ \text{Expr} \quad e_2 \ \text{Expr}} \\
\frac{\text{let}(e_1, x, e_2) \ \text{Expr}}{}
\end{array}
\]

c)  6 marks  Define a typing judgement that ensures error-free execution if no type conversions are available at runtime and only arithmetic with same-type operands works correctly.

\[
\begin{array}{c}
x : \tau \in \Gamma \\
\frac{}{\Gamma \vdash x : \tau} \quad \frac{}{\Gamma \vdash \text{num} : \text{int}} \quad \frac{}{\Gamma \vdash \text{num.num} : \text{float}}
\end{array}
\]

\[
\begin{array}{c}
\frac{}{\Gamma \vdash a : \text{float}} \quad \frac{}{\Gamma \vdash b : \text{float}} \quad \frac{}{\Gamma \vdash \text{plus}(a, b) : \text{float}} \\
\frac{}{\Gamma \vdash a : \text{float}} \quad \frac{}{\Gamma \vdash b : \text{float}} \quad \frac{}{\Gamma \vdash \text{mult}(a, b) : \text{float}} \\
\frac{\Gamma \vdash e_1 : \tau' \quad \Gamma, x : \tau' \vdash e_2 : \tau}{\frac{}{\Gamma \vdash \text{let}(e_1, x, e_2) : \tau}}
\end{array}
\]

d)  6 marks  Define a different typing judgement that ensures error-free execution if automatic type conversions from \text{int} to \text{float} are available at runtime. It is enough to show the rule(s) that are different from the last question.

Additional rules:

\[
\begin{array}{c}
\frac{}{\Gamma \vdash a : \text{int}} \quad \frac{}{\Gamma \vdash b : \text{float}} \quad \frac{}{\Gamma \vdash \text{plus}(a, b) : \text{float}} \\
\frac{}{\Gamma \vdash a : \text{float}} \quad \frac{}{\Gamma \vdash b : \text{int}} \quad \frac{}{\Gamma \vdash \text{plus}(a, b) : \text{float}} \\
\frac{}{\Gamma \vdash a : \text{int}} \quad \frac{}{\Gamma \vdash b : \text{float}} \quad \frac{}{\Gamma \vdash \text{mult}(a, b) : \text{float}} \\
\frac{}{\Gamma \vdash a : \text{float}} \quad \frac{}{\Gamma \vdash b : \text{int}} \quad \frac{}{\Gamma \vdash \text{mult}(a, b) : \text{float}}
\end{array}
\]