The University of New South Wales

COMP3161/9161
Concepts of Programming Languages

Sample Final Exam

Semester 2, 2018

Time Allowed: 2 Hours

Total Number of Questions: 4

Answer **all** questions.

The questions **are** of equal value.

You are permitted **two** hand-written, single-sided A4 sheets of notes.

Only write your answers on the provided booklets.

Answers must be written in ink, with the exception of diagrams.

Drawing instruments or rules may be used.

Excessively verbose answers may lose marks.

There is a 3% penalty if your name and student number are not filled in correctly.

Question 1 (25 marks)

Consider the following inductive definition of a language of restricted boolean expressions:

$$\frac{}{\texttt{true}\ \mathbf{Bool}} \quad \frac{}{\texttt{false}\ \mathbf{Bool}} \quad \frac{b\ \mathbf{Bool}}{\texttt{not}\ b\ \mathbf{Bool}} \quad \frac{b_1\ \mathbf{Bool} \quad b_2\ \mathbf{Bool}}{(\texttt{and}\ b_1\ b_2)\ \mathbf{Bool}}$$

It has the following small-step semantics:

$$\frac{b \mapsto b'}{\texttt{not}\ b \mapsto \texttt{not}\ b'}1 \quad \frac{}{\texttt{not true} \mapsto \texttt{false}}2 \quad \frac{}{\texttt{not false} \mapsto \texttt{true}}3$$

$$\frac{b_1 \mapsto b_1'}{(\texttt{and}\ b_1\ b_2) \mapsto (\texttt{and}\ b_1'\ b_2)}4 \quad \frac{}{(\texttt{and true}\ b_2) \mapsto b_2}5 \quad \frac{}{(\texttt{and false}\ b_2) \mapsto \texttt{false}}6$$

(a) (3 marks) Derive the step-by-step evaluation of (`and not false (and true not true)`).

(b) (4 marks) Are the rules that define **Bool** unambiguous? If so, briefly explain why. If not, give an example of a term that allows for multiple derivations.

(c) (6 marks) The semantics rules listed above are small-step. Give an equivalent big-step semantics for this language.

(d) (12 marks) Give a new version of the small-step semantics where *all rules are axioms*. It may be helpful to expand the state with supporting data structures, similarly to the C-Machine. Give an inductive definition for any notation you define.

Question 2 (25 marks)

(a) (6 marks) Give the most general type of the following implicitly-typed MinHS expressions.

i. $\mathsf{InR}\ (\mathsf{InR}\ (3, 4))$

ii. $\mathsf{recfun}\ f\ x = \mathsf{fst}\ (\mathsf{snd}\ x)$

iii. $\mathsf{recfun}\ f\ x = \mathsf{case}\ x\ \mathsf{of}\ \mathsf{InL}\ g \to g\ \texttt{False}; \mathsf{InR}\ x \to \texttt{False}$

(b) (6 marks) Explain, with the help of an example, why most-general-type inference is not possible for the simple **rec**-based recursive types we added to MinHS in lectures.
Hint: Consider the type(s) of the term: $\mathsf{roll}\ (\mathsf{InR}\ (\mathsf{roll}\ (\mathsf{InL}\ 3)))$.

(c) (6 marks) What is the most general unifier of the following pairs of type terms? If there is no unifier, explain why.

i. $(a \times b) \to (b \times a)$ and $(\texttt{Int} \times c) \to (c \times c)$

ii. $a \to (a + a)$ and $(b + b) \to b$

iii. $\texttt{Int} \to \texttt{Int}$ and $\texttt{Float} \to \texttt{Int}$.

(d) (7 marks) We wish to specify an abstract data type (ADT) in MinHS for describing sets of integers. We include a function to create an empty set, to insert an integer into a set, and to perform union and intersection on sets. We will provide one implementation using a linked list data type `List`. Assume that `List` is a shorthand for the full recursive linked list type:

$$\mathbf{rec}\ t.\ \mathbf{1} + (\texttt{Int} \times t))$$

$$\begin{array}{l}\mathsf{Pack}\ \texttt{List}\ (\\ \quad \mathsf{recfun}\ \mathit{empty} :: \texttt{List} = \mathsf{roll}\ (\mathsf{InL}\ ()),\\ \quad \mathsf{recfun}\ \mathit{insert} :: (\texttt{Int} \to \texttt{List} \to \texttt{List}) = \cdots,\\ \quad \mathsf{recfun}\ \mathit{union} :: (\texttt{List} \to \texttt{List} \to \texttt{List}) = \cdots,\\ \quad \mathsf{recfun}\ \mathit{size} :: (\texttt{List} \to \texttt{Int}) = \cdots)\end{array}$$

What is the type of the above Pack expression? Write a small MinHS program that, given a set ADT,

1. creates an empty set,
2. inserts an element into the set, and then
3. returns the size of the resulting set.

Question 3 (25 marks)

(a) (15 marks) Suppose we extend MinHS with an (incorrect) subtyping rule that states

$$\frac{A \leq A' \qquad B \leq B'}{(A \to B) \leq (A' \to B')}$$

i. What is the *variance* of the right hand side of the function arrow $\to$?

ii. Assuming `Square` $\leq$ `Rect`, and the variable r : `Rect`, show that the above subtyping rule is incorrect by giving a MinHS expression of type `Square` that evaluates to r.

iii. How does the existence of such an expression violate *progress* or *preservation*?

iv. Provide a different rule for subtyping of functions that corrects this problem.

(b) (5 marks) Suppose we have a Show type class with one method, *show* :: $a \to$ `String`, and we write a function:

$$\begin{aligned} &\mathit{exclaim} :: \mathsf{Show}\ a \Rightarrow a \to \texttt{String} \\ &\mathit{exclaim}\ x = \mathit{show}\ x +\!\!+ \texttt{"!"} \end{aligned}$$

Provide an equivalent version of the *exclaim* function that relies on parametric polymorphism and parameter-passing instead of ad-hoc polymorphism.

(c) (5 marks) Suppose we had defined a monadic interface for managing a bank account:

$$\mathit{withdraw} :: \texttt{Amount} \to \mathsf{Bank}\ \texttt{Cash} \qquad \mathit{deposit} :: \texttt{Amount} \to \mathsf{Bank}\ \texttt{Receipt}$$

i. What purpose does the monadic Bank type serve?

ii. How would that same purpose be accomplished using *linear types* to model the bank account instead?

Question 4 (25 marks)

(a) (8 marks) Decompose the following properties into the intersection of a *safety* and *liveness* property.

i. The process p is the only process that will enter its critical section.

ii. The process q will eventually release the lock.

iii. No STM transaction will be interrupted.

iv. The program will throw an exception, not return a value.

(b) (12 marks) Consider the following concurrent program, consisting of processes **P** and **Q**, which manipulate a shared variable m that starts initialised to zero:

Process P	$\|$	**Process Q**
while $i < 10$ **do**		**while** $j < 10$ **do**
$m := m + 1;$		$m := m - 1;$
$i := i + 1$		$j := j + 1$
od		**od**

i. Assume each line is atomic. What are the possible final values of m?

ii. Now only assume atomicity for load and store instructions, not for each line. What are the possible output values in this case?

iii. How would you ensure *mutual exclusion* such that the final value of m is guaranteed to be 0, regardless of the atomicity model used?

(c) (5 marks) STM is a popular means of concurrency control in programming languages. Describe how STM addresses the main desiderata for concurrency control:

i. Mutual Exclusion

ii. Deadlock-Freedom

iii. Starvation-Freedom