UNIVERSITY OF SOUTHAMPTON COMP2212W1

SEMESTER 2 EXAMINATIONS 2022 - 2023

PROGRAMMING LANGUAGE CONCEPTS

DURATION 120 MINS (2 Hours)

This paper contains 4 questions

Answer **all** of the four questions.

An outline marking scheme is shown in square brackets to the right of each question.

University approved calculators MAY be used. A foreign language translation dictionary (paper version) is permitted provided it contains no notes, additions or annotations

A foreign language dictionary is permitted ONLY IF it is a paper version of a direct 'Word to Word' translation dictionary AND it contains no notes, additions or annotations.

8 page examination paper.

Accompanying Reference Sheet

This sheet is for use in Question 2 and it describes the λEx language. The syntax of expressions, values and types of the λEx language is

$$
\begin{array}{lllr}
E & ::= & \mathsf{n} & \text{Integer literals} \\
 & & \mathbf{true} & \text{Boolean True literal} \\
 & & \mathbf{false} & \text{Boolean False literal} \\
 & & E < E & \text{Comparison} \\
 & & E + E & \text{Addition} \\
 & & x & \text{Variable} \\
 & & \mathsf{if}\ E\ \mathsf{then}\ E\ \mathsf{else}\ E & \text{Conditional} \\
 & & \lambda(x : T) \to E & \text{Abstraction} \\
 & & E\ E & \text{Application} \\
 & & \mathsf{throw}\ E & \text{Throw an Exception} \\
 & & \mathsf{try}\ E\ \mathsf{catch}\ E & \text{Exception Handling} \\
 & & & \\
V & ::= & \mathsf{n} & \text{Integer value} \\
 & & \mathbf{true} & \text{True value} \\
 & & \mathbf{false} & \text{False value} \\
 & & \lambda(x : T) \to E & \text{Abstraction value} \\
T & ::= & \mathsf{Int} & \text{Type of Integers} \\
 & & \mathsf{Bool} & \text{Type of Booleans} \\
 & & T \to T & \text{Type of functions}
\end{array}
$$

The (partial) type rules are

$$
\frac{}{\Gamma \vdash \mathsf{n} : \mathsf{Int}} \qquad \frac{}{\Gamma \vdash \mathbf{true} : \mathsf{Bool}} \qquad \frac{}{\Gamma \vdash \mathbf{false} : \mathsf{Bool}}
$$

$$
\frac{\Gamma \vdash E_1 : \mathsf{Int} \quad \Gamma \vdash E_2 : \mathsf{Int}}{\Gamma \vdash E_1 < E_2 : \mathsf{Bool}} \qquad \frac{\Gamma \vdash E_1 : \mathsf{Int} \quad \Gamma \vdash E_2 : \mathsf{Int}}{\Gamma \vdash E_1 + E_2 : \mathsf{Int}} \qquad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}
$$

$$
\frac{\Gamma \vdash E_1 : \mathsf{Bool} \quad \Gamma \vdash E_2 : T \quad \Gamma \vdash E_3 : T}{\Gamma \vdash \mathsf{if}\ E_1\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3 : T}
$$

$$
\frac{\Gamma, x : T_1 \vdash E : T_2}{\Gamma \vdash \lambda(x : T_1) \to E : T_1 \to T_2} \qquad \frac{\Gamma \vdash E_1 : T_2 \to T_1 \quad \Gamma \vdash E_2 : T_2}{\Gamma \vdash E_1 E_2 : T_1}
$$

Where Γ is a type context given by the grammar $\Gamma ::= \emptyset \mid \Gamma, x : T$

The (partial) small step reduction rules are

$$\frac{n < m}{\mathsf{n} < \mathsf{m} \;\rightarrow\; \mathbf{true}} \qquad \frac{n \not< m}{\mathsf{n} < \mathsf{m} \;\rightarrow\; \mathbf{false}}$$

$$\frac{E \;\rightarrow\; E'}{\mathsf{n} < E \;\rightarrow\; \mathsf{n} < E'} \qquad \frac{E_1 \;\rightarrow\; E_1'}{E_1 < E_2 \;\rightarrow\; E_1' < E_2}$$

$$\frac{n + m = n'}{\mathsf{n} + \mathsf{m} \;\rightarrow\; \mathsf{n}'} \qquad \frac{E \;\rightarrow\; E'}{\mathsf{n} + E \;\rightarrow\; \mathsf{n} + E'} \qquad \frac{E_1 \;\rightarrow\; E_1'}{E_1 + E_2 \;\rightarrow\; E_1' + E_2}$$

$$\frac{}{\text{if } \mathbf{true} \text{ then } E_2 \text{ else } E_3 \;\rightarrow\; E_2} \qquad \frac{}{\text{if } \mathbf{false} \text{ then } E_2 \text{ else } E_3 \;\rightarrow\; E_3}$$

$$\frac{E_1 \;\rightarrow\; E_1'}{\text{if } E_1 \text{ then } E_2 \text{ else } E_3 \;\rightarrow\; \text{if } E_1' \text{ then } E_2 \text{ else } E_3}$$

$$\frac{}{(\lambda(x : T) \rightarrow E)V \;\rightarrow\; E[V/x]} \qquad \frac{E_1 \;\rightarrow\; E_1'}{E_1 E_2 \;\rightarrow\; E_1' E_2} \qquad \frac{E_2 \;\rightarrow\; E_2'}{V E_2 \;\rightarrow\; V E_2'}$$

TURN OVER

Question 1.

In this question we will be considering the following simple BNF specification of a language:

$$
\begin{array}{rcl}
\texttt{<E>} & ::= & <E> \# <T> \mid <T> \\
\texttt{<T>} & ::= & <T> \$ <C> \mid <C> \\
\texttt{<C>} & ::= & <num> \mid (<E>) \\
\texttt{<num>} & ::= & \texttt{<digit>} \mid \texttt{<digit><num>} \\
\texttt{<digit>} & ::= & \mathbf{0} \mid \mathbf{1} \mid \ldots \mid \mathbf{9}
\end{array}
$$

where $\#$ and $\$$ are two binary operations with unspecified interpretations.

(a) What is the difference between a lexer and a parser?
[3 marks]

(b) What is the difference between an LR and an LL parser?
[3 marks]

(c) For each of the following, state whether or not the string is generated by the grammar (assuming top level non-terminal <E>).

(i) $23 \$ 2 \# 9 \# 1000$ (ii) $2 \$ 2 \$ 017$
(iii) $(2 \# 34) \# (22 \$ 1) \$ (1)$ (iv) $983 \# 2 \$ (\$) 23$

[4 marks]

(d) Would it be best to implement a parser for the grammar using an LL or an LR parser? Explain your answer.
[3 marks]

(e) Draw a parse tree for the input string ”1 # 2 # 3 $ 4 # 5”
[5 marks]

(f) Rewrite the grammar to be a right recursive grammar that accepts the same language.
[4 marks]

(g) How might using your grammar from Part (f) have an effect on the possible semantics of the language? Give example interpretations of the # and $ operations to support your answer.
[3 marks]

Question 2.

This question is based on the language named λ*Ex* described in the accompanying reference sheet. In particular we extend the simply typed lambda calculus as follows:

$$E ::= \ldots \mid \text{throw } E \mid \text{try } E \text{ catch } E$$

The term throw E represents an exception being thrown with an $\mathtt{Int}$ valued error code. The term try E catch E' represents an exception handling mechanism in which the code E is evaluated and, should an exception be thrown during its evaluation, the $\mathtt{Int}$ valued error code will be passed to the function represented by E'.

(a) Write type checking rules for the exception operators described above.

[6 marks]

(b) Using your type checking rules and the other type checking rules given for λEx, give a type derivation for the term

$$\begin{array}{l} \text{try } (\lambda(x : Int) \to \text{if } x > 0 \text{ then } \mathbf{true} \text{ else } (\text{throw } 1))\ 0 \\ \text{catch } \lambda(code : Int) \to \text{if } code < 2 \text{ then } \mathbf{false} \text{ else } (\text{throw } code) \end{array}$$

[8 marks]

(c) Write small-step operational semantic rules for the exception operators described above. Your rules should implement a call-by-value strategy for passing error codes to exception handlers. Note that throw n is not a value of the extended language and does not need reducing any further.

[11 marks]

TURN OVER

Question 3.

(a) In the context of concurrent programming, what is meant by *deadlock*?

[2 marks]

(b) Explain the difference between *fine-grained* and *coarse-grained* concurrency.

[2 marks]

(c) List **three** advantages and **two** disadvantages of message-passing concurrency as compared to shared-memory concurrency.

[5 marks]

(d) Below is a pseudocode implementation of Peterson's Algorithm:

```
const THREADS = 2;

var turn;
var flag[THREADS];


void enter_critical(process_id) {

  var other_id = 1 - process_id;

  flag[process_id] = true;
  turn = process_id;

  while (turn == process_id AND flag[other_id]) {
   };
  }

void leave_critical(process_id) {
  flag[process_id] = false
}
```

Under what circumstances can the above implementation of Peterson's Algorithm be considered thread-safe? In circumstances in which it would NOT be considered thread-safe (if any), what modification(s) would need to be made in order to make it thread-safe? Use line numbers to refer to specific points of execution.

[5 marks]

(e) Explain what is meant by a *compare and set* operation.

[5 marks]

(f) The following pseudocode shows an example of a Producer/Consumer model, implemented with asynchronous message passing.

```
class Producer implements Runnable {

  String[] messages = { m1, m2, m3, m4, m5, m6}

  void run(){
    for (String message in messages){
      MPI.receive()
      MPI.send()
    }
  }
}

class Consumer implements Runnable {

  const int BUFFER_SIZE = 10

  // TODO: finish Consumer code

}
```

Using pseudocode, complete the Consumer. The Consumer should advertise available buffer space. The Consumer should output any messages created by the Producer. You may assume that the methods `MPI.receive` and `MPI.send` are part of a message passing implementation that handles message buffering.

[6 marks]

TURN OVER

Question 4.

In this question we will use the notation, $x \sim y$ to mean that states x and y of a labelled transition system are bisimilar. Similarly we will write $x \simeq y$ to mean that states x and y are simulation equivalent and we write $x =_{Tr} y$ to mean that states x and y are trace equivalent.

We will also be considering a variation on the notion of equivalence between labelled transition systems defined as follows: we first define the acceptance set $A(x)$ of a state x to be $\{a | x \xrightarrow{a} x' \text{ for some } x'\}$. We then say that a relation R between labelled transition systems is an *accepts simulation* if the following holds:

if $x\ R\ y$ then A(x) = A(y) and
for all $x \xrightarrow{a} x'$ there exists a $y \xrightarrow{a} y'$ such that $x'\ R\ y'$

We write $\prec_A$ to be the largest *accepts simulation* and define *accepts equivalence* as

$$x \sim_A y \text{ if and only if } x \prec_A y \text{ and } y \prec_A x$$

For each of the statements below, give an argument as to why it is true or provide a counter-example to it. For all states x, y of an LTS:

(a) $x \simeq y$ implies $x =_{Tr} y$ [4 marks]
(b) $x =_{Tr} y$ implies $x \simeq y$ [2 marks]
(c) $x \sim_A y$ implies $x \simeq y$ [3 marks]
(d) $x \simeq y$ implies $x \sim_A y$ [4 marks]
(e) $x \sim_A y$ implies $x \sim y$ [6 marks]
(f) $x \sim y$ implies $x \sim_A y$ [6 marks]

END OF PAPER