SEMESTER 2 EXAMINATION 2021 - 2022

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 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.

6 page examination paper.

Question 1.

(a) Consider the simple BNF specification of a LOGO like language below:

$$
\begin{array}{rcl}
\texttt{<lStmt>} & ::= & \mathbf{FD}\,\texttt{<num>} \\
& | & \mathbf{LEFT}\,\texttt{<num>} \\
& | & \mathbf{RIGHT}\,\texttt{<num>} \\
& | & \mathbf{REPEAT}\,\texttt{<num>}\,\texttt{<lStmt>} \\
& | & \texttt{<lStmt>}\,\texttt{<lStmt>} \\
\texttt{<num>} & ::= & \texttt{<digit>} \mid \texttt{<digit><num>} \\
\texttt{<digit>} & ::= & \mathbf{0} \mid \mathbf{1} \mid \ldots \mid \mathbf{9}
\end{array}
$$

where the symbol ␣ represents a whitespace character. The intended semantics are that of a drawing cursor being moved in a forwards direction, rotating left or right and executing simple loops. For each of the following, state whether or not the string is generated by the grammar (assuming top level non-terminal <lStmt>).

(i) **FD 0 RIGHT 3 FD 1 LEFT 90 FD 13**

(ii) **FD 10 REPEAT RIGHT 45 FD 20**

(iii) **REPEAT 5 LEFT 90 FD 10 REPEAT 5 RIGHT 90 FD 10**

(iv) **REPEAT 10 FD 20 REPEAT 10 RIGHT 90 REPEAT 5**

(v) **FD 1 RIGHT 20 LEFT 4 5**

[5 marks]

(b) Explain what it means for a BNF specification to be *ambiguous*. Is ambiguity likely to be problematic for the above programming language? [5 marks]

(c) With the aid of two examples, show that the BNF specification given in Part (a) is ambiguous in at least two distinct ways. [5 marks]

(d) Rewrite and extend the grammar above so that all sources of ambiguity are removed. The language accepted by the grammar may change but you should be able to express the same range of behaviours. You may introduce block delimiters such as parentheses as extra symbols in the concrete syntax. Grammars that allow the use of block delimiters sparingly are preferred.

[10 marks]

Question 2.

In this question we consider a programming language, StPL, given by the following grammar:

$$E ::= \mathbf{true} \mid \mathbf{false} \mid n \mid \mathsf{skip} \mid \mathsf{stop} \mid \mathsf{add} \mid \mathsf{dup} \mid \mathsf{pop} \mid \mathsf{push}\ E \mid \mathsf{isZero} \mid \mathsf{if}\ E\ \mathsf{then}\ E\ \mathsf{else}\ E \mid E; E \mid \mathsf{repeat}\ E$$

where n ranges across integer literals. The language is intended as a stack-based language for simple calculations. The small step operational semantics for this language consists of reduction between states comprising a finite stack containing integer values and an expression. We write this in the form $(S, E) \ \rightarrow\ (S', E')$. We write $n : S$ for stack with value n at the top with stack S below. The reduction rules are given as follows:

$$\frac{}{(S, \mathbf{true}; E) \ \rightarrow\ (S, E)} \quad \frac{}{(S, \mathbf{false}; E) \ \rightarrow\ (S, E)} \quad \frac{}{(S, n; E) \ \rightarrow\ (S, E)}$$

$$\frac{}{(S, \mathsf{skip}; E) \ \rightarrow\ (S, E)} \quad \frac{}{(S, \mathsf{stop}; E) \ \rightarrow\ (S, \mathsf{stop})} \quad \frac{(S, E_1) \ \rightarrow\ (S', E_1')}{(S, E_1; E_2) \ \rightarrow\ (S', E_1'; E_2)}$$

$$\frac{}{(n : m : S, \mathsf{add}) \ \rightarrow\ (n + m : S, \mathsf{skip})} \quad \frac{}{(n : S, \mathsf{dup}) \ \rightarrow\ (n : n : S, \mathsf{skip})}$$

$$\frac{}{(n : S, \mathsf{pop}) \ \rightarrow\ (S, n)} \quad \frac{}{(S, \mathsf{push}\ n) \ \rightarrow\ (n : S, \mathsf{skip})} \quad \frac{(S, E) \ \rightarrow\ (S', E')}{(S, \mathsf{push}\ E) \ \rightarrow\ (S', \mathsf{push}\ E')}$$

$$\frac{}{(0 : S, \mathsf{isZero}) \ \rightarrow\ (0 : S, \mathbf{true})} \quad \frac{n \neq 0}{(n : S, \mathsf{isZero}) \ \rightarrow\ (n : S, \mathbf{false})}$$

$$\frac{}{(S, \mathsf{if}\ \mathbf{true}\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3) \ \rightarrow\ (S, E_2)} \quad \frac{}{(S, \mathsf{if}\ \mathbf{false}\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3) \ \rightarrow\ (S, E_3)}$$

$$\frac{(S, E_1) \ \rightarrow\ (S', E_1')}{(S, \mathsf{if}\ E_1\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3) \ \rightarrow\ (S', \mathsf{if}\ E_1'\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3)}$$

$$\frac{}{(S, \mathsf{repeat}\ (E)) \ \rightarrow\ (S, E; \mathsf{repeat}\ (E))}$$

This language is presented without any sort of type system. In this question you will be asked to define a type system for StPL.

TURN OVER

Assuming a typing relation $\vdash E : T$ for some types T, type safety for StPL is expressed as follows: for every $\vdash E : T$ and every stack S of size two or more, (S, E) is either (S, stop) or there is a reduction $(S, E) \rightarrow (S', E')$ for some S', E' such that $\vdash E' : T$.

(a) Explain why, for the notion of type safety stated above, for a well-typed expression E and some stack S then (S, E) may still get stuck.

[2 marks]

(b) If we can always choose an arbitrarily large initial stack S does type safety guarantee that well-typed StPL programs never get stuck?

[2 marks]

(c) Define a suitable grammar of types that could be used to represent the different kinds of expressions in this language.

[2 marks]

(d) Provide a set of type rules that guarantee type safety as defined above for StPL.

[13 marks]

(e) Using your type rules, give a type derivation tree for the StPL program

$$\mathsf{repeat}\ (\mathsf{add}; \mathsf{if}\ \mathsf{isZero}\ \mathsf{then}\ \mathsf{push}\ 1; \mathsf{stop}\ \mathsf{else}\ \mathsf{dup}; \mathsf{push}\ -1)$$

[6 marks]

Question 3.

(a) Explain the main difference between the *shared variable* and *message passing* approaches to concurrent programming. [2 marks]

(b) Explain the difference between asynchronous and synchronous message passing communication.

[3 marks]

(c) Consider the following Java interfaces:

```
interface SendEndPoint<T>{
   void send(T o);
}
interface ReceiveEndPoint<T>{
   T receive();
}
interface Channel<T> implements
   SendEndPoint<T>,ReceiveEndPoint<T> {}
```

Using the monitor synchronisation primitives, wait, notify and notifyAll of Java, show how one can give an implemention of a Channel object that supports both synchronous and asynchronous communication between a single sender thread and a single receiver thread. You may use Java-like pseudocode rather than syntactically correct Java.

[10 marks]

(d) What would be the effect on the behaviour of your code if more than one thread is given a `SendEndPoint` channel instance ?

[3 marks]

(e) What would be the effect on the behaviour of your code if more than one thread is given a `ReceiveEndPoint` channel instance ?

[3 marks]

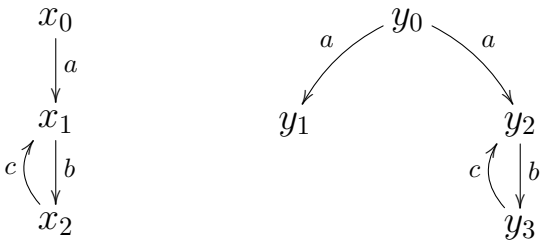
(f) For systems that support `compareAndSet`, explain how this may be used to provide a better implementation of your communication Channel in Part (c).

[4 marks]

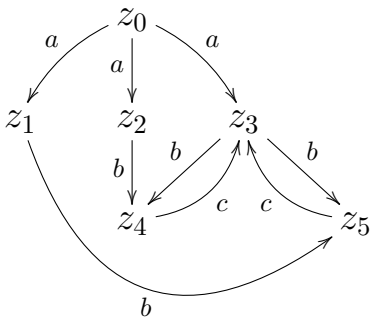
TURN OVER

Question 4.

Consider the following labelled transition systems



and



(a) Are the nodes x_0 and y_0 trace equivalent? Justify your answer. [2 marks]

(b) Show that x_0 simulates y_0. [4 marks]

(c) State what it means for two nodes in a labelled transition systems to be bisimilar. [4 marks]

(d) Using the simulation game, show that x_0 and y_0 are not bisimilar. [6 marks]

(e) Are x_0 and z_0 bisimilar? Justify your answer with full proof. [9 marks]

END OF PAPER