COMPUTER SCIENCE TRIPOS Part IB

Monday 3 June 2019 1.30 to 4.30

COMPUTER SCIENCE Paper 4

*Answer **five** questions: **up to four** questions from Section A, and **at least one** question from Section B.*

*Submit the answers in five **separate** bundles, each with its own cover sheet. On each cover sheet, write the numbers of **all** attempted questions, and circle the number of the question attached.*

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

STATIONERY REQUIREMENTS
Script paper
Blue cover sheets
Tags

SPECIAL REQUIREMENTS
Approved calculator permitted

SECTION A

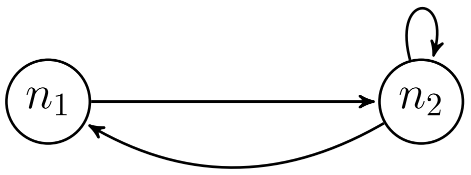
1 Programming in C

Consider the following structure declaration for a general directed graph data structure. In this structure, the `size` field gives the number of outgoing edges, and the `children` field is a pointer to an array of pointers to the child nodes.

```
typedef struct node Node;
struct node {
  bool flag;
  int size;
  Node **children; // pointer to an array of Node pointers
};
```

(*a*) Define a function `Node *node(int n, Node **children)` which builds a new node from its arguments, taking ownership of the `children` argument and initializing the `flag` field to false. [2 marks]

(*b*) Write a function `Node *example(void)` which returns a new graph with the following structure, with the return value corresponding to n_1: [2 marks]



$$\begin{tikzcd} n_1 \arrow[r] & n_2 \arrow[l, bend left] \arrow[loop above] \end{tikzcd}$$

(*c*) Define a structure for representing a linked list of `Node *` pointers, with a `Nodelist` typedef for the structure. [2 marks]

(*d*) Supposing we represent the empty linked list with the `NULL` pointer, and a cons cell with a pointer to a `Nodelist`, define a function `Nodelist *cons(Node *head, Nodelist *tail)` to add an element to this linked list. [2 marks]

(*e*) Write a function `Nodelist *reachable(Node *node)` which returns a list of all the nodes reachable from the argument `node`, including `node` itself. This list should contain every reachable node, and have no duplicates. You may assume that the `flag` field of every reachable node is set to `false` on entry to this function, and that your routine may modify it as you wish. [7 marks]

(*f*) Define a function `void free_node(Node *node)` which deallocates all the node objects reachable from the argument `node`. You may assume that the `flag` field of every reachable node is set to `false` on entry to this function, and that your routine may modify it as you wish. [5 marks]

2 Programming in C and C++

(*a*) Find at least 2 sources of undefined behaviour in the following program, and write a corrected version of this function. [5 marks]

```
int main(void) {
  char *s = "abcde"; int len = strlen(s);
  for (int i = 0; i <= len; i++)
    s[i] += 1;
  return printf("'%s' is %d characters long\n", ++s, strlen(s));
}
```

(*b*) Restructure the program below to be more cache-efficient, giving the code and explaining your changes. [5 marks]

```
typedef struct point { double x, y, z; } Point;

int find_max_x_argument(int n, Point *elems) {
  double max = 0; int max_index = 0;
  for (int i = 0;  i < n; i++)
    if (max < elems[i].x) { max_index = i; max = elems[i].x; }
  return max_index;
}
```

(*c*) The following definition forms part of a legal C++ program:

```
int foo() {
  MyClass x(1,2);
  MyClass y = C(3,4);
  MyClass z = x;
  MyClass t;
  z = x;
  z.f = x.f;
  return z.f;
}
```

(*i*) Give a declaration of `MyClass` which enables `foo` to compile and run, noting any methods or constructors in `MyClass` which are invoked when `foo` is called. [*Note:* Precise C++ syntax is not necessary to obtain full marks.] [4 marks]

(*ii*) Having seen your declaration of `MyClass`, a colleague points out some of the lines of `foo` may be redundant. Which are these? [2 marks]

(*iii*) Your boss now replaces your declaration of `MyClass`. Not having access to the new declaration, explain, giving reasons, which if any lines of `foo` are now redundant. [4 marks]

3 Compiler Construction

We will extend our language SLANG and the JARGON virtual machine with data type definitions such as

```
type int_list = Nil | Cons of int * int_list

type int_tree = Leaf of int | Node of int * int_tree * int_tree
```

(Note that we will not consider polymorphic types.)

We also extend the language with a `match` expression and pattern matching so that we can write functions such as

```
let tail (l : int_list) : int_list =
    match l with
    | Cons(x, l') -> l'
    end
end

let is_nil (l : int_list) : bool =
    match l with
    | Nil -> true
    | l' -> false
    end
end

let sum (x : int_tree) : int =
    match x with
    | Leaf y -> y
    | Node(y, t1, t2) -> y + (sum t1) + (sum t2)
    end
end
```

The semantics of the `match` expression: Match clauses are attempted from first to last. If no match is found then there is a run-time error and the program halts (we don't have exceptions).

(*a*) Ignoring lexing and parsing, what changes to the compiler's front-end would this require? [3 marks]

(*b*) Discuss possible runtime representations of values of types such as `int_list` and `int_tree`. [3 marks]

[continued . . .]

(*c*) Assuming your runtime representation uses tags, do you need distinct tags for `Cons(x, l)` and `Node(x, t1, t2)`? Justify your answer. [4 marks]

(*d*) Suppose that our language allows nested patterns such as

```
match t with
| Node(x, Leaf y, t2) -> e1(x, y, t2)
| Node(x, t1, Leaf y) -> e2(x, t1, y)
| Node(x, Node(y, t1 t2), t3) -> e3(x, y, t1, t2, t3)
end
```

but our front-end generates abstract syntax that cannot contain nested patterns. How would you represent the code above in the same language without nested patterns? [4 marks]

(*e*) Carefully describe the code you would generate for the JARGON virtual machine for the body of the function `sum` defined above. If you need to extend the virtual machine with new instructions, then define their semantics. (You do not need to remember the exact syntax of the JARGON instructions as long as you clearly explain what your code is doing.) [6 marks]

4 Compiler Construction

This question explores how exceptions might be added to SLANG and the JARGON virtual machine. We will raise an exception with

```
raise e
```

where `e` is an expression. We will "trap" an exception with the following expression.

```
try e with f end
```

If `e` evaluates to a value `v`, then `v` is the result of the `try`-expression. Otherwise, the evaluation of `e` raises an exception `E` and the `try`-expression continues by evaluating the function application `f(E)`. To simplify things we will assume that each f is an identifier. Uncaught exceptions at the top-level will result in a runtime error.

(*a*) Do we need to define a fixed type for exceptions? Justify your answer. [3 marks]

(*b*) What typing rule or rules would you implement for the expression `raise e`? Justify your answer. [3 marks]

(*c*) A compiler may rewrite expressions in order to optimise generated programs. For example, here are two rewrite rules to simplify conditional expressions:

	code	replacement
1	`if true then e1 else e2`	`e1`
2	`if false then e1 else e2`	`e2`

For each of the rules below, argue that it is, or is not, a valid optimisation rule.

	code	replacement
1	`raise (raise e)`	`raise e`
2	`e1 + (raise e2)`	`raise e2`
3	`try (raise e) with f end`	`f(e)`
4	`try e with (fun x -> raise x) end`	`e`

[6 marks]

(*d*) Carefully describe the stack-oriented code you would generate for both the `raise`- and `try`-expressions. [8 marks]

5 Further Java

A programmer designs a client-server booking system for a meeting room. The role of the server is to distribute bookings between clients when they connect. Clients open a socket connection to the server regularly for a short period. When a client connects, the client first sends to the server an instance of `Message` which contains any new bookings made by the client; in response, the server sends an instance of `Message` containing all bookings made by other clients since the client last connected; the server then closes the connection. The key parts of the `Message` and `Booking` classes are defined as follows:

```
public class Message implements Serializable {
  private final String uniqueClientId;
  private final java.util.List<Booking> bookings;
  ...
}

public class Booking implements Serializable {
  private final String uniqueClientId;
  private final java.util.Date startTime;
  private final java.util.Date endTime;
  private final String description;
  ...
}
```

(*a*) Write a Java implementation of the server, using a single thread to serve each client in turn. You may assume the existence of a static method `processBookings`, which accepts a list of new bookings from a specified client and returns a list of bookings to be sent back to the client. You may assume suitable accessor methods for `Message` and `Booking`; you do not need to handle exceptions. [8 marks]

(*b*) The programmer decides to extend the booking system with *vector clocks*.

(*i*) Write down a suitable data structure for a vector clock in Java. [2 marks]

(*ii*) Describe in words how the system can be modified to incorporate vector clocks and allow clients to compute a partial order of `Message` objects. Discuss how vector clocks are initialised and updated. [6 marks]

(*iii*) The programmer wants to use vector clocks to determine which booking occurred first, allowing clients to mark any subsequent bookings as in conflict and therefore cancelled. Describe when the vector clock algorithm cannot determine which booking is first, how this is detected, and propose a solution which resolves the ambiguity. [4 marks]

6 Security

(*a*) What is the purpose of the `HttpOnly` flag in the HTTP protocol? Briefly describe an attack that this flag was intended to prevent. [4 marks]

(*b*) Users of web sites often commit transactions by filling out an HTML form and pressing a "Submit" button to update some state stored on a server (e.g., password change, purchase).

(*i*) HTML forms can submit such requests using either the `GET` or `POST` method of HTTP. Which is more appropriate here? Give *three* reasons. [6 marks]

(*ii*) Some web servers place an additional token value into an invisible field of HTML forms that are used to commit security-critical transactions. What security risk can such a token mitigate? [4 marks]

(*iii*) Explain *three* additional checks that a web server may implement to reduce this risk? [6 marks]

7 Security

(*a*) In a Linux shell session, you can see the following information:

```
$ ls -la
drwxr-xr-x   2 root    root    4096 Jun  3 13:29  .
drwxr-xr-x  25 root    root    4096 Jun  3 13:29  ..
-rwxr-xr-x   1 root    root    4675 Jun  3 13:29  script.pl
```

Consider how you need to change the file access-control information shown above in order to achieve the following additional goals:

- Only members of the group `staff` who are not also members of the group `interns` can execute `script.pl`.
- When `script.pl` is called, it should be able to switch between using the access privileges of the caller and those of the user `primary`.
- All members of group `staff` should be able to read the contents of `script.pl`.

What would "`ls -la`" output after you have applied these changes? [6 marks]

(*b*) Sending a password over a network connection is vulnerable to replay attacks by eavesdroppers. Briefly describe three other forms of unilateral (or one-pass) authentication suitable for human keyboard entry that reduce that risk with the help of a hardware token, and name one advantage of each. [6 marks]

(*c*) The Windows NT operating-system family offers two variants of many API functions that receive a string: one for strings using ASCII (or one of its 8-bit "code page" extensions) and one for 16-bit Unicode strings. Linux and many Internet protocols instead use an ASCII-compatible encoding of Unicode called UTF-8.

(*i*) Briefly explain how UTF-8 is decoded. [4 marks]

(*ii*) What particular security risk can emerge when UTF-8 is used in a system along with another Unicode encoding, such as the 16-bit wide characters on Windows, and how can this be avoided? [4 marks]

SECTION B

8 Semantics of Programming Languages

Consider the following C-like language, tinyC. It has locally-scoped mutable variables, and functions that take a single argument. Its operational semantics is defined as a transition system over configurations $\langle e, E, s\rangle$ where E is an environment $\{x_1 \mapsto n_1, .., x_j \mapsto n_j\}$, mapping the variable names currently in scope to their addresses, and s is a store $\{n_1 \mapsto v_1, .., n_k \mapsto v_k\}$, mapping each currently allocated address to either an integer n or **undef**. In this question n ranges over $0 \ldots 2^{63}{-}1$. Programs p consist of finite sets of definitions with distinct names.

$$\begin{array}{ll} \textit{expression},\ e ::= & n \mid x \mid x{=}e' \mid \{\,\mathbf{int}\,x;\, e\} \mid e_1; e_2 \mid f(e) \mid \mathbf{undef} \mid \mathbf{kill}\,x \\ \textit{definition},\ d ::= & \mathbf{int}\,f(\,\mathbf{int}\,x)\{e\} \end{array}$$

$$\frac{E(x){=}n \quad s(n){=}n'}{\langle x, E, s\rangle \longrightarrow \langle n', E, s\rangle}\ \text{DEREF} \qquad \frac{E(x){=}n \quad n \in \mathbf{dom}\,(s)}{\langle \mathbf{kill}\,x, E, s\rangle \longrightarrow \langle 0, E\backslash x, s\backslash n\rangle}\ \text{KILL}$$

$$\frac{\langle e, E, s\rangle \longrightarrow \langle e', E', s'\rangle}{\langle x{=}e, E, s\rangle \longrightarrow \langle x{=}e', E', s'\rangle}\ \text{AS1} \qquad \frac{E(x){=}n \quad s(n){=}v}{\langle x{=}n', E, s\rangle \longrightarrow \langle n', E, s + [n \mapsto n']\rangle}\ \text{AS2}$$

$$\frac{x \notin \mathbf{dom}\,(E) \quad n \notin \mathbf{dom}\,(s) \quad \neg\exists n' < n.\ n' \notin \mathbf{dom}(s)}{\langle\{\,\mathbf{int}\,x; e\}, E, s\rangle \longrightarrow \langle e; \mathbf{kill}\,x, E + [x \mapsto n], s + [n \mapsto \mathbf{undef}]\rangle}\ \text{LOCAL}$$

$$\frac{\langle e_1, E, s\rangle \longrightarrow \langle e_1', E', s'\rangle}{\langle e_1; e_2, E, s\rangle \longrightarrow \langle e_1'; e_2, E', s'\rangle}\ \text{SEQ1} \qquad \frac{}{\langle n; e, E, s\rangle \longrightarrow \langle e, E, s\rangle}\ \text{SEQ2}$$

$$\frac{\langle e, E, s\rangle \longrightarrow \langle e', E', s'\rangle}{\langle f(e), E, s\rangle \longrightarrow \langle f(e'), E', s'\rangle}\ \text{CL1} \qquad \frac{\mathbf{int}\,f(\,\mathbf{int}\,x)\{e\} \in p}{\langle f(n), E, s\rangle \longrightarrow \langle\{\,\mathbf{int}\,x; (x{=}n; e)\}, E, s\rangle}\ \text{CL2}$$

(*a*) For the configuration $\langle \mathrm{g}(3), \{\,\}, \{\,\}\rangle$ and program $\mathbf{int}\,\mathrm{g}(\,\mathbf{int}\,\mathrm{y})\{\{\,\mathbf{int}\,\mathrm{z}; \mathrm{z}{=}\mathrm{y}\}\}$, give the sequence of 11 configurations it transitions to. For each transition, include the list of rule names involved in its derivation, but not the derivation itself. [9 marks]

(*b*) For each of the following, briefly explain the key points of its tinyC semantics and what it illustrates, referring to the transitions and rules, and to the relationship between tinyC and the full C language, as appropriate.

(*i*) $\langle\{\,\mathbf{int}\,\mathrm{y}; \mathrm{g}(\mathrm{y})\}, \{\,\}, \{\,\}\rangle$. [3 marks]

(*ii*) $\langle\{\,\mathbf{int}\,\mathrm{y}; 4\}; \mathrm{y}, \{\,\}, \{\,\}\rangle$ [3 marks]

(*iii*) $\langle \mathrm{h}(5), \{\,\}, \{\,\}\rangle$, with the program $\mathbf{int}\,\mathrm{h}(\,\mathbf{int}\,\mathrm{y})\{\mathrm{y}{=}6; \mathrm{y}\}$, [3 marks]

(*iv*) $\langle\{\,\mathbf{int}\,\mathrm{y}; (\mathrm{y}{=}3; \{\,\mathbf{int}\,\mathrm{y}; \mathrm{y}{=}4\}); \mathrm{y}\}, \{\,\}, \{\,\}\rangle$. [2 marks]

9 Semantics of Programming Languages

Consider the following pure functional language, in which n ranges over the mathematical integers.

$$
\begin{array}{ll}
T ::= & \textbf{int}1 \mid \textbf{int}8 \mid \textbf{int}16 \mid \textbf{uint}1 \mid \textbf{uint}8 \mid \textbf{uint}16 \mid T \to T' \mid T * T' \mid T + T' \\
e ::= & n \mid e +_T e' \mid x \mid \textbf{fn}\ x{:}T \ \Rightarrow\ e \mid e\ e' \mid (e, e') \mid \#1\ e \mid \#2\ e \mid \textbf{inl}_T\ e \mid \textbf{inr}_T\ e \\
& \mid \textbf{case}\ e\ \textbf{of}\ \textbf{inl}\,(x_1 : T_1) \Rightarrow e_1 \mid \textbf{inr}\,(x_2 : T_2) \Rightarrow e_2
\end{array}
$$

Its operational semantics is defined as a relation $e \longrightarrow e'$ with the standard rules for a pure call-by-value left-to-right functional language, except with the following rules for addition of values. As usual, the expression $n +_T n'$ is stuck if one of these does not apply.

$$
\frac{\begin{array}{l} n \in -2^{N-1} \ldots 2^{N-1} - 1 \\ n' \in -2^{N-1} \ldots 2^{N-1} - 1 \\ n''{=}n + n' \\ n'' \in -2^{N-1} \ldots 2^{N-1} - 1 \end{array}}{n +_{\textbf{int}N}\ n' \longrightarrow n''}\ \text{PLUS\_INT}
\qquad
\frac{\begin{array}{l} n \in 0 \ldots 2^{N} - 1 \\ n' \in 0 \ldots 2^{N} - 1 \\ n''{=}n + n' \\ n''' = n'' \bmod 2^{N} \end{array}}{n +_{\textbf{uint}N}\ n' \longrightarrow n'''}\ \text{PLUS\_UINT}
$$

(*a*) Define a subtype relation $T <: T'$ and type relation $\Gamma \vdash e : T$ for this syntax and operational semantics that will permit flexible use of integers in the appropriate ranges. You can omit the standard type relation rules for the expressions (e, e'), $\#1\ e$, $\#2\ e$, $\textbf{inl}_T\ e$, $\textbf{inr}_T\ e$, and **case**. [14 marks]

(*b*) Explain three main aspects of your definitions, with reference to the programming idioms they permit and the runtime errors they exclude, with examples. [6 marks]

END OF PAPER