CST1
COMPUTER SCIENCE TRIPOS Part IB

Monday 9 June 2025 13:30 to 16:30

COMPUTER SCIENCE Paper 4

*Answer **five** questions.*

*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

1 Compiler Construction

Here is a fragment of a lexer specification for a programming language:

`int`	⇒	`INT`	the type name `int`
`[a-z]+`	⇒	`IDENT`	a user-defined identifier
`type`	⇒	`TYPE`	the keyword `type`
`[ \n]+`	⇒	*skip*	whitespace

The lexing rules map regular expressions to tokens or to the special action *skip*.

(*a*) The specification is ambiguous: multiple lexing rules match the same string.

(*i*) Give an example of ambiguity that is resolved by the *first match* rule. [1 mark]

(*ii*) Give an example of ambiguity that is resolved by the *longest match* rule. [1 mark]

(*iii*) Identify an ineffectual lexing rule in the specification. [1 mark]

(*b*) Convert the specification to a tagged deterministic finite automaton (DFA). [7 marks]

(*c*) Give a new lexing rule that skips over comments. Comments start with the two-character sequence `{-` and end with the two-character sequence `-}`, and can contain any characters except the closing sequence `-}`.

You can use standard operators ($\emptyset$, `c`, ϵ, $r_1|r_2$, r_1r_2, $r*$, $r+$) and positive and negative character sets such as `[a-z]` and `[^a-z]`. [3 marks]

(*d*) One way of constructing lexer DFAs is to use *derivatives*. The derivative of a regular expression r with respect to a character c is another regular expression $\partial_c\, r$ that matches s if r matches cs.

(*i*) Define ∂_c for sequencing r_1r_2, intersection $r_1 \& r_2$ and negation $\neg r$. [3 marks]

(*ii*) Give a set of derivatives that can be used to construct a DFA for the lexer specification, including the lexing rule for comments from part (*c*). Omit cases that produce $\emptyset$. You do not need to actually construct the DFA. [4 marks]

2 Compiler Construction

A library for a Slang-like language supports explicit lazy evaluation via a type `lazy` and functions `delay` and `force`:

```
type 'a cell =
| Val of 'a
| Exn of exn
| Lazy of (unit -> 'a)
type 'a lazy =
    ('a cell) ref
```

```
let delay f = ref (Lazy f)
let force r = match !r with
  | Val v -> v
  | Exn e -> raise e
  | Lazy f -> try (let v = f () in r := Val v; v)
              with e -> r := Exn e; raise e
```

You decide to incorporate similar support for lazy evaluation into the language, adding built-in constructs `delay e` and `force e`, where `e` is an expression.

(*a*) Outline the benefits and drawbacks of implementing laziness in the compiler rather than in a library. [4 marks]

(*b*) Give new Jargon VM instructions that can implement `delay` and `force` and describe their behaviour. [6 marks]

(*c*) Give the translation of the `delay` and `force` constructs into your extended instruction set. [6 marks]

(*d*) You now consider adding an optimization that evaluates the argument of `delay` eagerly rather than creating a delayed computation.

(*i*) Give an expression `e` for which the transformation is valid (that is, behaviour-preserving) and always an optimization. [1 mark]

(*ii*) Give an expression `e` for which the transformation is valid and only sometimes an optimization. [1 mark]

(*iii*) Give an expression `e` for which the compiler cannot ascertain whether the transformation is valid. [1 mark]

(*iv*) Give an expression `e` for which the transformation is not valid. [1 mark]

3 Semantics of Programming Languages

Consider the following syntax of an assembly language. It is similar to RV32 except that each memory location holds a 32-bit word, from a set $\texttt{u32} = \{0, \ldots, 2^{32} - 1\}$. It has instructions to load, store, add, branch-on-equal, and jump-and-link.

$$\begin{array}{l} \mathit{register\_name},\ r,\ rd,\ rs,\ rs_a,\ rs_d ::= \ \texttt{x1} \mid \texttt{x2} \mid \texttt{x3} \\ \mathit{instruction},\ i ::= \ \texttt{lw}\, rd, rs_1, \mathit{imm} \mid \texttt{sw}\, rs_a, rs_d, \mathit{imm} \mid \texttt{addi}\, rd, rs_1, \mathit{imm} \mid \texttt{beq}\, rs_1, rs_2, \mathit{imm} \\ \mid \texttt{jalr}\, rd, rs_1, \mathit{imm} \end{array}$$

A machine state is a tuple $\langle pc, R, M \rangle$ of a PC value $pc \in \texttt{u32}$, a register state $R : \mathit{register\_name} \to \texttt{u32}$, and a partial memory state $M : \texttt{u32} \rightharpoonup \texttt{u32}$, not necessarily defined for the whole address space. Assume there is a partial function $\texttt{decode} : \texttt{u32} \rightharpoonup \mathit{instruction}$, then the semantics of the load instruction can be defined by:

$$\frac{\begin{array}{l} M(pc)\,\texttt{defined} \\ \texttt{decode}\,(M(pc)) = \texttt{lw}\, rd, rs_1, \mathit{imm} \\ n = R(rs_1) + \mathit{imm} \\ M(n)\,\texttt{defined} \\ M(n) = n' \end{array}}{\langle pc, R, M \rangle \to \langle pc + 1, R + \{rd \mapsto n'\}, M \rangle}\ \textsc{lw}$$

(a) Give operational semantics rules for the other instructions. Comment briefly on any choices you had to make. [8 marks]

From now on, consider just the load, store, and add instructions. We want to impose a type discipline that distinguishes between integer and pointer values, both in registers and in memory, with types T that are either `uint` for integer values, or T`*` for pointers to values of type T. Suppose that no pointer arithmetic is allowed. Let Γ range over finite partial functions from register names and addresses to types.

(b) Define a judgement $\Gamma \vdash \langle R, M \rangle$ that checks that the register state R and memory state M are consistent with Γ, with every pointer-typed value being dereferenceable with a value of the appropriate type. Explain your definition briefly. [4 marks]

(c) Define a typing judgement $\Gamma \vdash i \dashv \Gamma'$ for instructions i where Γ is the type environment before i executes and Γ' is the type environment that can be assumed by the following instruction. Explain your definition briefly.

Your definition should be sound with respect to the operational semantics: if $\Gamma \vdash i \dashv \Gamma'$, assuming $\Gamma \vdash \langle R, M \rangle$ for some Γ before the instruction executes, with $M(pc)\,\texttt{defined}$ and $\texttt{decode}\,(M(pc)) = i$, then (1) there should exist some transition $\langle pc, R, M \rangle \to \langle pc', R', M' \rangle$, and (2) for any such transition, $\Gamma' \vdash \langle R', M' \rangle$. You should **not** prove this. [8 marks]

4 Prolog

This question concerns a two-player game (white pieces vs. black pieces) played on an 8x8 game board represented as an 8-element list for the rows, each row being a list of eight values for the contents of squares on that row. The board columns/rows are numbered 1 . . . 8 starting at bottom/left from the white player perspective. Both players have one type of piece, attacking one or more squares diagonally up and to the right. An empty square is represented by the atom `e`, a white piece by `w` and a black piece by `b`. So the empty board is,

```
[ [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ],
  [ e,e,e,e,e,e,e,e ] ]
```

In your answers ensure each relation has a comment giving a declarative reading of its behaviour. Avoid unnecessary use of *cut* or other extra-logical relations. The library relations `\=` and `is` may be used. Other library relations should not be assumed.

(*a*) Write a relation `set/4` which, if given a board, square location and a piece, will place that piece on that square. For example `set(Board1,sq(3,2),Piece,Board2)` will succeed with `Board2` having the same contents as `Board1` except that the square on the 3rd column from left and 2nd row from bottom contains the given piece. [4 marks]

(*b*) Write a relation `contains(Board,sq(Column,Row),S)` which, if given a board and square position, will succeed with `S` being the content of that square. [5 marks]

(*c*) Write a relation `white_move(sq(Column,Row),sq(Column1,Row1))` which given a starting position for a white piece at `sq(Column,Row)` will generate in `sq(Column1,Row1)` each position that could be reached on an empty board by moving diagonally up and to the right from the starting position. [4 marks]

(*d*) Assuming a board populated with pieces from both sides, write a relation `white_attack(Board,sq(Col,Row),sq(Col1,Row1))` which, given the current board state and `sq(Col,Row)` containing a white piece, will succeed with `sq(Col1,Row1)` containing the location of an enemy piece that can be successfully attacked, if one exists. An attack on an enemy piece would be blocked by a piece of either colour earlier on the same diagonal. [7 marks]

5 Programming in C and C++

A handheld device uses a rotary encoder with a circle of LED indicators around it. All behaviour is implemented in permanent C code running on an internal micro-controller.

(*a*) For the C code running in the device, what is the minimum support needed (if any) from an operating system or run-time system? [4 marks]

(*b*) Can C code in the device or in general operate without a heap? Would it then have no pointers? [4 marks]

(*c*) The following code turns two adjacent LEDs on. Explain four features of the code.

```
#define IO_BASE 0xE000
((volatile unsigned char *)IO_BASE)[4] = 0x60;
```

[4 marks]

(*d*) The rotary encoder has two output bits (as in the ECAD classes) that advance through the following infinitely repeating pattern in the low two bits read at offset 8 from the `IO_BASE` when rotating clockwise, ...0132013... The sequence is reversed when the encoder is turned the other way. Define a C subroutine `poll()` that is to be repeatedly called at a suitable rate (eg 1 kHz). The outcome should be that a single LED is lit at any time, with which one being suitably adjusted by the encoder. [8 marks]

6 Programming in C and C++

(*a*) You are building a complex-number library where the expression $R = R * A * B$ is coded roughly like

```
struct cpx { double re, im; };
extern cpx A, B; cpx R = { 1.0, 1.0 };
mul(R, R, mul(A, B));
```

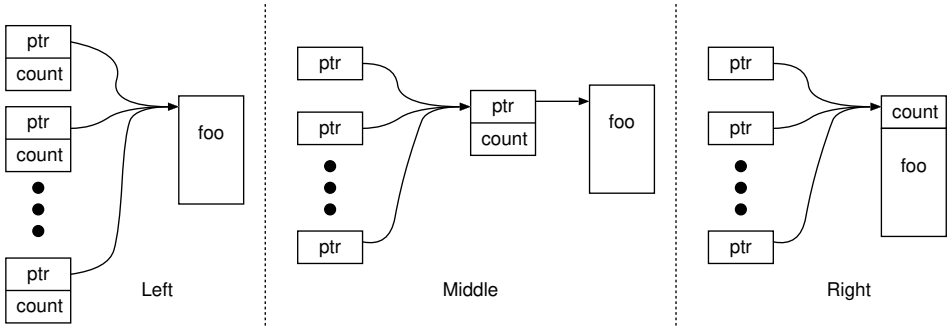
The outer call to `mul` returns its result using pass-by-reference in its first argument: the other two arguments are the operands.

(*i*) Give a complete implementation of the multiply function (and any overloads needed) and an example of using it, using C++ where all arguments to `mul` are references. There might not be any deviations from the original coding style, but if there are, justify changes with a brief comment. [4 marks]

(*ii*) Likewise, give a complete C implementation, including the caller and callee, where all arguments are pointers. To achieve overloading in C, variations on the method name can be used. [4 marks]

(*iii*) Explain how the passing of `R` to `mul` twice (aliasing) can cause an incorrect result in some simple implementations and suggest a fix if either of your implementations might fail. [2 marks]

(*b*) You have no access to the standard library and must code a flexible C++ reference counter class `refct<T>`. It will automatically delete a heap object when its reference count reaches zero.



(*i*) Briefly assess each of the above three design sketches. [2 marks]

(*ii*) Write a C++ implementation of the middle design so that it would behave sensibly under the following (far from ideal) artificial use pattern: [5 marks]

```
foo *copy1 = new foo();  // An object to be managed.
refct<foo> rcf(copy1);   // Add management to the first reference.
foo *copy2 = rcf.new_user(); // Create a second reference to it.
...
rcf.drop();      // Drop one of the references.
rcf.drop();      // Drop the last one, causing foo to be deleted.
```

(*iii*) Criticise the inclusion of the `drop()` method, suggesting an improvement to the overall design that follows the RAII (Resource Acquisition Is Initialization) programming idiom. [3 marks]

7 Cybersecurity

A web application uses an SQL database that contains a `student` table with fields `id`, `studentName`, `pwdHash` (all `varchar`) and `studentGrade` (`integer`). The `pwdHash` field contains the unsalted SHA-256 hash of the user's password, encoded in Base 32 (A–Z and 2–7) with `b32enc()`. [*Note:* In this question, SQL statements are written over several lines for greater legibility, but assume there are no newlines.]

(*a*) A web form of that application has input fields for `id` and `password`. On submission, it displays the corresponding `studentGrade`, obtained by running the following SQL query, where the items inside the single quotes are replaced by the content of the corresponding form fields. Describe an attack that displays the grade of student `abc78`, whose password you do not know. First write out the SQL resulting from your attack, then what to type in the fields. [3 marks]

```
SELECT studentGrade
FROM student
WHERE id = 'id' AND pwdHash = b32enc(sha256('password'));
```

(*b*) Another web form of the same application lets you change your own display name and password: you must supply fields `studentID`, `newStudentName`, `oldPassword`, `newPassword`. On submission, the form sends the following SQL statement to the database, with the items inside the single quotes replaced by the content of the corresponding form fields. Describe an attack to change the grade of existing student `zzz666` to `100` but without changing this student's name or password, neither of which you know. First explain your strategy, then write out the resulting SQL, then what to type in the fields. Assume the database is configured to parse only one SQL statement per supplied string. [10 marks]

```
UPDATE student
SET studentName = 'newStudentName',
    pwdHash = b32enc(sha256('newPassword'))
WHERE id = 'id' AND pwdHash = b32enc(sha256('oldPassword'));
```

(*c*) The developer augments all forms with code that removes every non-alphanumeric character from the fields before inserting them in the SQL statement. Describe the main advantages and drawbacks of this approach. [4 marks]

(*d*) Describe a better approach than the one in Part (*c*) and justify why it is better. [3 marks]

8 Cybersecurity

For the purpose of this question, assume that you are working on a 32-bit x86 system, that `strcpy()` is the C standard library function and that all the other named functions were invented for this question.

(*a*) You discovered a buffer overflow vulnerability in a program. You wish to execute some shellcode you prepared, which is 48 bytes long. During the execution of the vulnerable function, the `ebp` is at `0xbffff3e4` and the 32-byte vulnerable buffer starts at `0xbffff3c0`. The `badstuff` you supply gets copied into the vulnerable buffer via `strcpy()`, provided it does not exceed 512 bytes in length.

You prepare the following `badstuff` input...

16 copies of `0xbffff400` ; 32 bytes of nop sled ; 48 bytes of shellcode

...but your shellcode does not get executed.

(*i*) Assuming `badstuff` has just been copied into the buffer, give hex addresses for: the start of the nop sled; the start of the shellcode; the location of the return address; the first location not overwritten by `badstuff`. [4 marks]

(*ii*) Clearly explain why your shellcode did not run. [2 marks]

(*iii*) Fix the attack by changing some of the numbers, with clear justification, but without altering the structure or length of `badstuff`. [4 marks]

(*iv*) Construct, with explanation, the shortest possible `badstuff` that still executes your shellcode payload. Give the lengths of your new `badstuff` and of the original one. [4 marks]

(*b*) The host system has been strengthened by making the stack not executable. Nonetheless, you wish to exploit a buffer overflow vulnerability you discovered in another program. Your input gets copied via `strcpy()` into a vulnerable buffer in the stack frame of `print()`, at absolute memory position `&buf`. The return address, during the execution of `print()`, is at `&ret`. With your attack, you wish to invoke `unlock(0)`, whose entry point is at `&unlock`. You may use library function `clear(p)` (entry point `&clear`), which writes 0 into the word at address `p`. You may not skip the functions' prologues. You may crash the program after exiting `unlock(0)`.

Clearly explain how to construct the input, describing the evolution of the stack throughout the attack. Your answer must provide the exact content of your input, byte by byte, as in the example provided in Part (*a*), parametric in the symbolic addresses above. [6 marks]

END OF PAPER