CST1
COMPUTER SCIENCE TRIPOS Part IB

Tuesday 15 June 2021 11:30 to 14:30 BST

COMPUTER SCIENCE Paper 4

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

*Submit each question answer in a **separate** PDF. As the file name, use your candidate number, paper and question number (e.g., `1234A-p4-q6.pdf`). Also write your candidate number, paper and question number at the start of each PDF.*

You must follow the official form and conduct instructions for this online examination

SECTION A

1 Programming in C and C++

(*a*) (*i*) In programming languages in general, what is the difference between statically and dynamically-allocated variables? Without using the `static` keyword, give examples of both types of variable in the C language. [2 marks]

(*ii*) What effect do the `static` and `extern` keywords have in C for both forms of variable? [3 marks]

(*b*) There are 32 characters in the ASCII set between 'A' and 'a'. All are printable. A potentially-infinite stream of 8-bit bytes can be made printable by encoding into base 32 and represented by a longer stream just using these characters. Give a syntactically-accurate definition in C for `encode32(unsigned char a)`, called for each byte, which invokes `putchar(char c)` once or twice per call to render the encoded output. [5 marks]

(*c*) Show how the `for` statement in C can be used to traverse a linked list. Explain the benefits of this coding style, mentioning whether the `continue` statement works appropriately. [4 marks]

(*d*)
```
typedef char * mystring;
mystring s201 = "201";
mystring s202 = s201;
s202[2] += 1;
```

A novice programmer writes the above code. What do you think they are intending to do and what two problems might they suffer? [2 marks]

(*e*) An interpreter for a string processing language is written in C. Describe four storage and efficiency-related considerations when designing the module for storing strings for the interpreter? [4 marks]

2 Programming in C and C++

(*a*) A key/value store holds pairs of strings in a non-volatile memory (NVM). The NVM is mapped as a memory region between two hardcoded virtual addresses, NV_START and NV_END. This memory is all set to zero as manufactured and can be freely read, but any write operations result in a logical ORing of the new data with the old data at the addressed location. Bits can never be cleared. Strings representing keys and values will be fewer than 250 characters long. Eventually the memory will fill up, but sufficient is available for the intended application.

The API provides the following two operations:

```
int store(char *key, char *value) // returns non-zero on error, and
char *lookup(char *key)           // returns NULL if not found.
```

Provide a full C implementation of the `store` routine, explaining how the lookup function and changes to values under a given key would work, if this is not obvious from your code. Code efficiency is not important.
Hint: You can directly access the non-volatile store using a construct such as `((char *)NV_START)[offset]`. [10 marks]

(*b*) The following text *almost* constitutes both a C++ and Java program:

```
class Foo { public: int x, y; };
class Test {
  private: void f1(Foo p) { p.x = 1; }
  private: void f2(Foo &q) { q.y = 2; }  //[Java]: ignore '&'
  public: void test() {
    Foo p;  //[Java]: replace with: 'Foo p = new Foo();'
    p.x = p.y = 99;
    f1(p);
    Foo q = p;
    p = q;
    f2(q);
    print("HERE");
  }
};
```

(*i*) Explain the storage structure accessible from variables `x` and `y` when control reaches `print("HERE")` following a call to `test()`, both in the Java and the C++ interpretations. [3 marks]

(*ii*) For the C++ interpretation, show how adjustments *only to the definition* of `class Foo` can cause (useful) debugging-style output to be produced at *as many places as possible* during the execution of method `test()`. [7 marks]

3 Compiler Construction

(*a*) Suppose we are writing a compiler for an ML-like language and we want to employ the equation

```
(map f l1) @ (map f l2) = map f (l1 @ l2)
```

as a left-to-right rewrite rule for optimisation. The symbol `@` represents list append.

Discuss the merits of this idea. Is it always correct? If so, state clearly what assumptions you are making about `@` and `map`. [5 marks]

(*b*) A compiler's front-end will often expand some syntactic constructs into lower-level representations. Consider the following fragment for the abstract syntax of a SLANG-like language.

```
type var = string

type exp =
  (* abstract syntax *)      (* concrete syntax *)
  | Var of var               (* x *)
  | Project of int * exp     (* proj i e *)
  | Tuple of exp list        (* (e1, e2, ..., en) *)
  | Let of var * exp * exp   (* let x = e1 in e2 *)
  | Apply of exp * exp       (* e1 e2 *)
  | Function of var * arg_pattern * exp   (* fun f p = e *)

and arg_pattern =
  | APvar of var                    (* x *)
  | APtuple of arg_pattern list     (* (p1, p2, ... pn) *)
```

This language has general projections for n-tuples so

$$\text{proj } i \ (e_1,\ e_2, \cdots, e_k)$$

will evaluate to v_i, the value of e_i. If i is not in the range between 1 and k there will be a run-time error.

In this language we can write functions with simple (possibly nested) patterns for function arguments:

```
fun f (a, b, (c, (d, e)) = b a
```

[continued ...]

Now suppose we want our front-end to eliminate such patterns. That is, we want to write a function of type

```
eliminate_tuple_patterns : exp -> exp
```

so that the resulting expression contains functions with patterns only of the form `APvar x` for some (new) variable `x`. For example, the code for `f` above should be translated to a semantically equivalent expression of the form

```
fun f x =  ...
```

that contains only simple variable arguments (that is, only `APvar` patterns in the abstract syntax).

Your task is to write this function in OCaml. You can assume that you have a function for generating fresh variable strings.

```
new_var : unit -> string
```

[15 marks]

4 Compiler Construction

(a) Show that the following grammar for the language of balanced parenthesis is ambiguous.

$$\begin{array}{rcl} S & \rightarrow & \epsilon \\ & | & (S) \\ & | & SS \end{array}$$

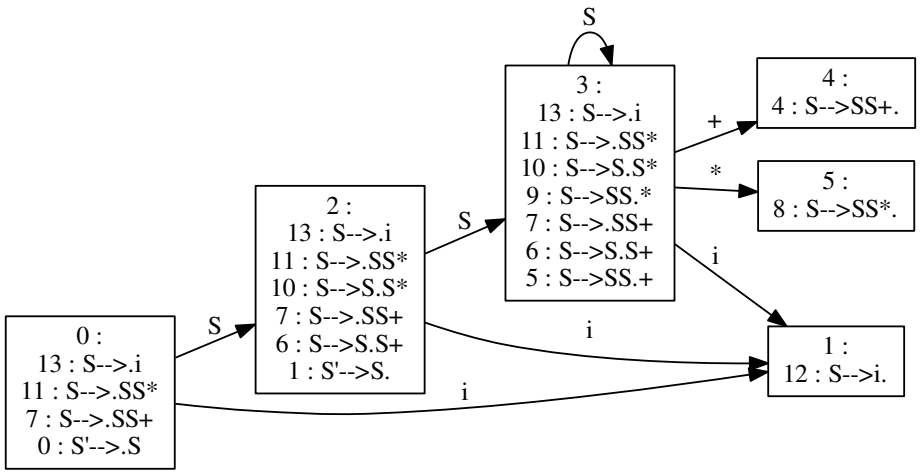
[5 marks]

(b) An arithmetic expression such as $x * ((z * u) + y)$ can be represented without parenthesis in postfix notation as $xzu * y + *$. This representation is ideal for evaluation using a stack machine.

Below is a simple grammar for postfix expressions:

$$\begin{array}{rcl} S & \rightarrow & i \\ & | & SS+ \\ & | & SS* \end{array}$$

The terminal i represents the lexical class of identifiers. Here is a DFA for the LR(0) items of this grammar.



(i) Using the DFA, construct the SLR(1) $ACTION$ and $GOTO$ tables for this grammar. Explain your work. [6 marks]

(ii) Show a trace of a parsing of $w = iii * i + *$. Justify every step. [9 marks]

5 Further Java

Clients of an online service execute code remotely at runtime by sending and receiving serialised Java objects to and from a server. Clients use the following classes:

```
public class NewClass implements Serializable {
  private String name;
  private byte[] bytecode;
}

public class Result implements Serializable {}
public class Params implements Serializable {}

abstract public class Function<R extends Result, P extends Params>
      implements Serializable {
  abstract public R run(P args);
}

class Invoke implements Serializable {
  private Function function;
  private Params args;
}
```

Clients send the bytecode of new classes using `NewClass` and an instance of `Invoke` to request the server execute a specified function. The server responds to such a request by executing the `run` method in the `Function` object and serialising and returning the result back to the client. You may assume the above classes have appropriate constructors as well as getter and setter methods.

(*a*) Define subclasses of `Params`, `Result` and `Function` (called `Squared`) to compute x^2 where x is of type `double`. [3 marks]

(*b*) Implement a client which executes `Squared`. Your solution should accept an argument x from the user, remotely execute the code and print out the result. You may assume a static method `byte[] getClassBytes(Class c)`. [5 marks]

(*c*) An `Invoke` object can contain incompatible `Function` and `Params` references. Provide an example and describe how to refactor your client to include a compile-time compatibility check. Can the server also check? [5 marks]

(*d*) Describe the security challanges in the server implementation. How might these be addressed? [3 marks]

(*e*) Describe how you could improve runtime error handling by the server as well as reporting such errors to the client. [4 marks]

6 Security

(*a*) Consider the access rights of user `alice` regarding the following files and directories on a Linux file system:

```
$ id
uid=1000(alice) gid=1000(alice) groups=505(safety)
$ ls -ld foo foo/* foo/*/*
drwxrwxrwx 4 root  root   4096 Apr 20 14:40 foo
-rw-rw-r-- 1 root  root    100 Apr 20 14:40 foo/bar
-rw-rw-r-- 1 alice alice   100 Apr 20 14:40 foo/baz
-rw----r-- 1 root  safety  100 Apr 20 14:40 foo/cux
drwx------ 2 root  root   4096 Apr 20 14:40 foo/dir1
drwxrwxr-t 2 root  alice  4096 Apr 20 14:40 foo/dir2
-rw-rw---- 1 root  root    100 Apr 20 14:40 foo/dir2/flop
-rw-rw-r-- 1 alice alice   100 Apr 20 14:40 foo/dir2/wibble
-rw-rw---- 1 root  safety  100 Apr 20 14:40 foo/dir2/wobble
```

(*i*) For each of the eight files and directories under `foo/`, list which of the following three operations user `alice` is able to perform: read (`R`), delete (`D`), rename (`N`). Your answer should be eight lines of the form

```
R-N  foo/qux
```

where the letters `RDN` indicate that the corresponding operation is allowed, and replacing any of these letters with `-` indicates that it is denied. [8 marks]

(*ii*) Which sequence of shell commands can user `alice` use to take ownership of file `foo/bar`, without affecting its content, such that it appears afterwards in the above directory listing as in

```
-rw-rw-r-- 1 alice alice   100 Apr 20 14:40 foo/bar
```

[2 marks]

(*iii*) Name *two* operations on the metadata of file `foo/bar` that user `alice` can execute only after taking ownership. [2 marks]

(*b*) List *four* mechanisms that a web server can use to maintain authentication session state across a web browser's HTTP requests that form part of the same login session, along with one disadvantage associated with each. [8 marks]

7 Security

(*a*) An SQLite database set up with

```
CREATE TABLE users(name varchar(32), password varchar(32));
CREATE TABLE prices(commodity varchar(32), value varchar(32));
INSERT INTO users VALUES ('alice', 'SeCreT');
INSERT INTO prices VALUES ('gold', 1335.33);
```

is used by a Perl web application for looking up commodity prices. The application receives a string in variable `$metal` from a user-provided HTML form, forms an SQL statement to look up the corresponding price with

```
$sql = "SELECT value FROM prices WHERE commodity='$metal';";
```

and displays to the user the value it finds in the first column of the first row of the table returned.

(*i*) What text could an attacker provide in `$metal`, such that

(A) the value displayed is the password of user `alice`? [3 marks]

(B) the password of user `alice` is changed to `qwerty`. [3 marks]

(*ii*) Briefly describe *three* measures that the designer of the web application can take to reduce the risks created by the attack described in Part (*a*)(*i*)(A). [6 marks]

(*iii*) Describe how the TCB of the web application could be structured to reduce the risk of the attack described in Part (*a*)(*i*)(B). [2 marks]

(*b*) The *WikiHash* web application stores for each registered user U in its user table the tuple (U, V) with $V = H(P)$, where H is a collision-resistant hash function and P is U's password. When an HTTP request arrives, it applies the following authentication procedure:

- if the request arrives without a session cookie, the user is presented with a password login form
- when the user submits username U and password P via that form, the web application checks the user table for entry $(U, H(P))$ and if it exists sets the session cookie to $(U, H(H(P)))$
- if the request arrives with a session cookie (U, C), the web application loads the user's user-table entry (U, V) and checks if $H(V) = C$ before granting access to pages restricted to user U

(*i*) What risk does storing $H(P)$ (as opposed to storing P) in a user table aim to mitigate? [2 marks]

(*ii*) Are all these risks still mitigated by the above procedure? If not, change the procedure to remove the vulnerability it introduced. [4 marks]

SECTION B

8 Semantics of Programming Languages

Languages like FORTH and POSTSCRIPT are *stack-based languages*; they store intermediate values on a stack rather than binding to variable names. In this question we will look at how to give a type system and operational semantics for a simple stack-based language. The syntax and informal meaning of our language is given by:

e	::=	$\underline{n}$	Push the numeral n on the stack
	|	$\underline{b}$	Push the Boolean b on the stack
	|	Add	Replace the top two stack elements with their sum
	|	Eql	Replace the top two stack elements with the result of comparing them for equality
	|	$\mathsf{Cond}(e_1, e_2)$	Delete the top stack element and execute e_1 or e_2, depending on if the top of the stack was True or False
	|	Skip	No-op
	|	$e_1; e_2$	Run e_1 and then e_2
v	::=	$\underline{b} \mid \underline{n}$	Values
s	::=	$\cdot \mid s, v$	Stacks
τ	::=	$\mathsf{bool} \mid \mathsf{num}$	Types
Γ	::=	$\cdot \mid \Gamma, \tau$	Stack Types

We take a value v to be a Boolean or numeral, and define a stack s to be a stack of values (growing at the right). Correspondingly, there are types bool and num for values, and stack types Γ for stacks s.

The small-step operational semantics is then defined by a transition relation $\langle e_1 \mid s_1 \rangle \mapsto \langle e_2 \mid s_2 \rangle$. One rule for this relation is:

$$\frac{}{\langle \mathsf{Add} \mid s, \underline{n}, \underline{m} \rangle \mapsto \langle \mathsf{Skip} \mid s, \underline{n+m} \rangle}$$

The typing relation is given as a relation $\Gamma \vdash e \dashv \Gamma'$, which means that e, when run with a stack of shape Γ, yields a stack of shape Γ'. One rule for this relation is:

$$\frac{}{\Gamma, \mathsf{num}, \mathsf{num} \vdash \mathsf{Add} \dashv \Gamma, \mathsf{num}}$$

(a) Give the remaining rules for the operational semantics. [7 marks]

(b) Give the remaining rules for the typing judgement. [7 marks]

(c) Formulate and state the progress and preservation lemmas for this language. [6 marks]

9 Semantics of Programming Languages

(a) Suppose that $l : \mathsf{intref} \in \Gamma$. Now, consider the following program equivalence for L1:

$$(\mathsf{if}\ !l \leq 0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2); e_3 \simeq^{\Gamma}_{\mathsf{unit}} (\mathsf{if}\ !l \leq 0\ \mathsf{then}\ e_1; e_3\ \mathsf{else}\ e_2; e_3)$$

(i) Explain informally but carefully why this equivalence holds. [3 marks]

(ii) Using the definition of semantic equivalence, prove that this equivalence holds. [7 marks]

(b) Now, consider the following *non*-equivalence:

$$e_3; (\mathsf{if}\ !l \leq 0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2) \not\simeq^{\Gamma}_{\mathsf{unit}} (\mathsf{if}\ !l \leq 0\ \mathsf{then}\ e_3; e_1\ \mathsf{else}\ e_3; e_2)$$

(i) Give a well-typed example exhibiting a counterexample of this equivalence. [5 marks]

(ii) Give a statically decidable condition under which the transformation is valid. [5 marks]

END OF PAPER