Examination 2023-08-25

Kompilatorteknik 1 (5 hours)

Please make sure to write your exam code on each page you hand in. When you are finished, please place the pages containing your solutions together in an order that corresponds to the order of the questions. You can take with you the questions; no need to hand them in together with your answers.
This is a **closed book** examination but you might refer to your **A4 sheet of prepared hand-written** notes. It contains **40** points in total and their distribution between sub-questions is clearly identifiable. To get a final grade of **5** you must score at least **34**. To get a grade of **4** you must score at least **28** and to get a final grade of **3** you must score at least **21**. The teacher reserves the right to lower these thresholds. Note that you will get **credit only for answers that are either correct or a very serious attempt**. All answers should preferably be **in English**; if you are uncomfortable with English you can use Swedish, but note that for most of the questions only extremely short answers using natural language are required. Whenever in *real doubt* for what a particular question might mean, make sure to **state your assumptions clearly**.

DON'T PANIC!

1. **Automata & Regular Expressions (4 pts total).**
   (a) **(2 pts)** Show that the language generated by the following grammar:
   $$S \rightarrow aSa \mid a$$
   is a regular language.
   (b) **(2 pts)** Give a deterministic finite state automaton for the regular expression: $(a|b)^* \mid (ab)^*$.

2. **Grammars & Ambiguity (3 pts total; 1 pt each).** Consider the following grammar:
   $$S \rightarrow SS\texttt{+} \mid SS\texttt{*} \mid \texttt{a}$$
   (a) Give a parse tree for the string `a a + a *`
   (b) Is this grammar ambiguous or unambiguous? Justify your answer.
   (c) Describe in words the language generated by this grammar.

3. **LL Parsing (2 pts).** The grammar $S \rightarrow S\ a \mid b \mid \epsilon$ is not LL(1). For what (non-terminal, terminal) pair would the LL(1) table contain more than one entry? Which are these entries?

4. **Parsing (4 pts total; 2 pts each).** In both parts of this question, we are looking for clarity and brevity as well as the right idea. Suppose you are writing a parser for a programming language that includes the following syntax for looping constructs:

$$\begin{array}{lcl} \mathit{Loop} & \rightarrow & \texttt{do}\ \mathit{Stmt}\ \texttt{while}\ \mathit{Expr} \\ & | & \texttt{do}\ \mathit{Stmt}\ \texttt{until}\ \mathit{Expr} \\ & | & \texttt{do}\ \mathit{Stmt}\ \texttt{forever} \end{array}$$

(a) Can a predictive parser use this grammar? Give a *brief* (a couple of sentences) explanation why or why not.

(b) Can a bottom-up parser (e.g. LR(k)) use this grammar? Again, give a *brief* explanation why or why not.

5. **Parameter Passing (3 pts total; 1 pt each).** Consider the following program:

```
main() {
  a := 1;
  b := 2;
  f(a, b, a+b);
  print(a);
}

f(x, y, z) {
  y := y + 1;
  x := x + z;
}
```

Show what is printed if `main` is invoked when the calling convention is:

(a) call-by-value
(b) call-by-reference
(c) call-by-name

6. **Run-time Environments (6 pts; 2 pts each)** Consider a C-like language with functions, integers, and vectors of integers. Vectors are declared with constant sizes (e.g., `int A[100]`). Vector values are implemented as pointers to a block of memory containing the vector elements. When vectors are passed as arguments or returned as the result of functions, it is the pointer that is passed or returned. Thus, vectors are always passed by reference. Integers are always passed by value. Assume that values are communicated between functions only via function arguments and results (i.e., the language has no global variables).

For each of the following combinations of languages fearures, state whether each of activation records, vectors, and integers should be allocated in a global static area, on the stack, or in the heap. Choose the best alternative—the one that is both correct and gives the fastest code. Give a *brief* (1 sentence) justification for your answer to each part.

(a) A language with recursive functions, where integers and vectors can be passed as arguments and returned as the result of functions.

(b) A language without recursive functions, where integers and vectors can be passed as arguments and returned as the result of functions.

(c) A language with recursive functions, where integers and vectors can be passed as arguments but only integers can be returned as the result of functions.

---

7. **Activation Records (4 pts).** Consider the following Pascal program. Pascal is a language that allows nested procedures and procedure names that can be passed as arguments. In Pascal, everything between `(*` and `*)` is a comment. Assuming a user input consisting of the three numbers `1, 2, 0`, draw the stack of activation records when the number `1` is printed the first time. Include all control and access links, as well as parameters and global variables, and assume that all procedures are stored in the environment as closures.

```
program procenv(input, output);

procedure dolist(procedure print);
var x: integer;
     procedure newprint;
     begin
       print;
       writeln(x);
     end;
begin (* dolist *)
  readln(x);
  if x = 0 then begin print; print; end
           else dolist(newprint);
end; (* dolist *)

procedure null;
begin
end;

begin (* main *)
  dolist(null);
end.
```

---

8. **Code Generation (6 pts in total; 3 pts each).**

(a) Describe how a C-like `for` statement can be systematically turned into a corresponding `while` statement. Is it a good idea to use this transformation to generate code?

(b) Describe how a `case` or `switch` statement can be systematically turned into a sequence of nested `if` statements. Is it a good idea to use this transformation to generate code?

---

9. **Control-flow Graphs & Liveness Analysis (8 pts total).** Consider the following fragment of intermediate code (here `a % b` is `a mod b`, i.e., the remainder of dividing `a` with `b`), and the / operator is integer division:

```
    if a = 2 goto L3
L0: b := 2
L1: d := a / 2
    c := a % b
    if c = 0 goto L2
    if b >= d goto L3
    b := b + 1
    goto L1
L2: a := a + 1
    goto L0
L3:
```

(a) **(3 pts)** Draw the control-flow graph for this program. Place each basic block in a single node. Include the conditionals in the appropriate basic blocks.

(b) **(4 pts)** Annotate the control-flow graph with the set of variables live before and after each statement (not just before and after every block!), assuming that only `a` is live at the entry to `L3`.

(c) **(1 pts)** Consisely, what is the function that this program computes?

$\boxed{\text{Good luck !}}$