Examination 2023-06-07

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 **20**. 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 might of course use Swedish, but note that for most of the questions only extremely short answers using natural language are required. Besides your A4 page, you are also allowed to use dictionaries to/from English but **no other material**. Whenever in *real doubt* for what a particular question might mean, make sure to **state your assumptions clearly**.

DON'T PANIC!

---

1. **Grammars (5 pts total).** Write a context-free grammar for each of the following languages over the terminals $\{\texttt{a}, \texttt{b}\}$. Your grammar may be ambiguous.

   (a) **(2 pts)** Strings that are palindromes (having the same reading forwards or backwards). Some examples of palindromes are `a`, `aba`, and `abba`.

   (b) **(3 pts)** Strings containing an equal number of `a`'s and `b`'s.

---

2. **LR Parsing (6 pts total).** Consider the following grammar over terminals `a`, and `b`:

$$\begin{array}{lcl} S & \rightarrow & A \\ A & \rightarrow & \texttt{a}\ A \mid A\ \texttt{a}\ A \mid \texttt{b} \end{array}$$

   (a) **(1pt)** Draw all the parse trees for the string "`b a a b a b`" on the answer sheet.

   (b) **(5pts)** On the answer sheet, there is a partially completed LR(1) DFA for this grammar (state 0 is the initial state). Complete the DFA. You need to do the following:

   - Fill in items for states 2, 4, 6 and the rest of state 0 by performing closure operation. (You can of course fill all states if you want, but you do not need to.)
   - Fill in the missing transition labels on edges.
   - Write the necessary "Reduce by ... on ..." and "Accept on ..." labels on states. (State 1 has an already filled "Reduce by" label.) Not all states need a label.
   - Add all missing transition edges, if any, that are needed.

3. **LL Parsing (6 pts total; 1 for each row).** Consider the following grammar:

$$
\begin{array}{ll}
1 & E \rightarrow FE' \\
2 & E' \rightarrow AFE' \\
3 & E' \rightarrow \epsilon \\
4 & A \rightarrow a \\
5 & A \rightarrow \epsilon \\
6 & F \rightarrow GF' \\
7 & F' \rightarrow *F \\
8 & F' \rightarrow \epsilon \\
9 & G \rightarrow (E) \\
10 & G \rightarrow fG \\
11 & G \rightarrow c
\end{array}
$$

Construct an LL(1) parsing table for this grammar. Your table should look as follows:

	$	a	f	c	(	)	$*$
E							
E'							
F							
F'							
G							
A							

4. **Activation Records (5 pts).** Instead of (or in addition to) using static links, there are other ways of getting access to non-local variables. One way is just to leave the variable in a register! Consider the following program:

```
function f() : int =
  let var a := 5
      function g() : int = (a+1)
   in g() + g()
end
```

If a is left in, say register r_7, while g is called, then g can just access it from there.

What properties must a local variable, the function in which it is defined, and the functions in which it is used, have for this trick to work?

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

```
main() {
  a := 3;
  b := 4;
  f(a, b, a + b);
  print(a);
}
```

```
f(x, y, z) {
  y := y + 5;
  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. **Code Generation & Optimization (6 pts in total).** Consider the following code snippet:

```
1  #define N 100
2  void main(){
3    int i;
4    int a = 12;
5
6    for (i=0; i<N; i++){
7      a = a + i*3;
8      if (a%2 == 5)
9        break;
10     else a = a^2 + a^2 - i*3;
11   }
12 }
```

(a) **(2 pts)** Generate the intermediate 3-address code representation.

(b) **(1 pt)** Build the control flow graph.

(c) **(3 pts)** Apply all possible local optimizations and document each step. For example, *"applying algebraic representation on code lines 1,2,3 in version 1, leads to version 2."*

---

7. **Basic Blocks, Liveness Analysis, and Register Allocation (9 pts total).** Consider the following fragment of intermediate code:

```
L0: x := y + x
    w := 2 * x
    if s = u goto L1
    x := w + u
    u := u - 1
    goto L2
L1: s := w + 1
L2: y := s + x
    if y > 1000 goto L0
L3:
```

(a) **(2 pts)** Break the above piece of code into basic blocks and draw its control-flow graph. Place each basic block in a single node.

(b) **(3 pts)** Annotate your control-flow graph with the set of variables live before and after each statement (not just before and after every block!), assuming that only `s` and `u` are live at label `L3`. Make sure it is clear where your annotations are placed.

(c) **(4 pts)** Draw the register inference graph for this intermediate code and assign each temporary to a register for a machine that has only 3 registers. If you need to spill some temporary into memory, rewrite the program using psuedoinstructions `load x` and `store x` to load and spill the value of `x` from and to memory.

Good luck !

Page to be used for answers to Question 2

(a) The parse trees for the string "b a a b a b" are:

(b) The LR(1) DFA for this grammar is: