Examination 2011-12-15

Kompilatorteknik 1 (5 hours)

Please make sure to write your exam code on each page you hand in. Note that some questions (1–4, 6a and 6d) require the answers to be written in the pages provided at the end of this exam. When you are finished, please staple the pages containing your solutions together in an order that corresponds to the order of the questions. You can take with you the first four pages with the questions.
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 **35**. To get a grade of **4** you must score at least **29** and to get a final grade of **3** you must score at least **22**. 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. Whenever in *real doubt* for what a particular question might mean, make sure to **state your assumptions clearly**.

DON'T PANIC!

Please use the supplied page 5 for your answers to Question 1 and the supplied page 7 for answers to Questions 2, 3 and 4.

---

1. **Regular expressions and finite automata (6 pts total; 1+1 pts each).** Write regular expressions for the following languages over the alphabet $\Sigma = \{a, b\}$.

   (a) All strings that do not end with aa.

   (b) All strings that contain an even number of b's.

   (c) All strings that do not contain the substring ba.

   Also, for each language draw a deterministic finite automaton (DFA) that recognizes the language. None of your DFAs may contain more than four states.

---

2. **Parsing (4 pts total).**

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

   (b) **(3pts)** Give an example of a grammar that is unambiguous, left-factored, and not left-recursive that is also not LL(1). Briefly explain why this is so.

3. **First and follow sets (5 pts total; 1pt each).** Consider the following three grammars (where A is the start symbol):

   1. $A \rightarrow BC$
      $B \rightarrow Ax \mid x$
      $C \rightarrow yC \mid y$
   2. $A \rightarrow BC$
      $B \rightarrow Ax \mid x \mid \epsilon$
      $C \rightarrow yC \mid y$
   3. $A \rightarrow BC$
      $B \rightarrow Ax \mid x \mid \epsilon$
      $C \rightarrow yC \mid y \mid \epsilon$

   For each of the following statements about $First$ and $Follow$ sets, indicate for which grammar(s) the following statement is correct. You will receive credit only if you precisely list the set of grammars for which the statement is correct. Note that grammar 3 is a small extension of grammar 2, which in turn is a small extension of grammar 1.

   (a) $First(A) = \{x, y\}$ is correct for grammars:
   (b) $Follow(A) = \{\$, x\}$ is correct for grammars:
   (c) $Follow(B) = \{\$, x, y\}$ is correct for grammars:
   (d) $First(C) = \{y\}$ is correct for grammars:
   (e) $Follow(C) = \{\$, x\}$ is correct for grammars:

---

4. **Parameter passing mechanisms (4 pts total; 1pt each).** In the following program

```
fun bar() =
{  int a := 2;
      void foo(int b) {
        b := b * a;
        a := a - b;
      }
   {
      int a := 10;
      foo(a);
      print a;
   }
}
```

   what does `bar()` print under:

   (a) call-by-value and lexical scope
   (b) call-by-value and dynamic scope
   (c) call-by-reference and lexical scope, and
   (d) call-by-reference and dynamic scope.

---

5. **Runtime organization (10pts total).** Recall the following MIPS instructions:

   `sw $a0 n($sp)` store the value of the accumulator at address `n+$sp`

   `lw $ra n($sp)` load the return address register with the value stored at address `n+$sp`

   `addiu $sp $sp n` adjust the value of the stack pointer by `n`

   `move $a0 $t1` move the contents of `$t1` into `$a0`

   `jal f` jump to address `f` and store the address of the next instruction in register `$ra`

   `jr $ra` jump to address stored in register `$ra`

   Imagine a compiler that uses a variation of the calling convention that we used in the lectures. We give below the code that this compiler generates to setup the activation record for a function and also the code that is used to access the i^{th} $(i = 1, \ldots, n)$ formal argument of a function with n arguments. The caller pops the arguments off the stack in this calling convention. On entry to a function the return address is in `$ra` and on exit the return value is in `$a0`.

```
cgen(f(x1,...,xn) begin e end) =
   sw    $fp 0($sp)
   addiu $sp $sp -4   ; push the old FP on the stack
   move  $fp $sp      ; set the new FP
   cgen(e)            ; evaluate the body of the function
   ...                ; code to return

cgen(xi) =            ; get the ith argument
   lw    $a0 z($fp)   ; where z = 8 + 4 * (n - i)
```

   (a) **(2.5pts)** Draw the stack contents at the point where the function `f` has been invoked and its execution is just before the code that evaluates the function body (i.e., the `cgen(e)` above). Put larger addresses at the top. Clearly mark on the stack the location of the frame pointer, the stack pointer, and the positions where the arguments and the old frame pointer are stored.

   (b) **(2.5pts)** Write the code for return (the `...` in the above code). To return you must use the instruction `jr $ra`. Recall that the return value must be in `$a0` and that the calling function pops the arguments.

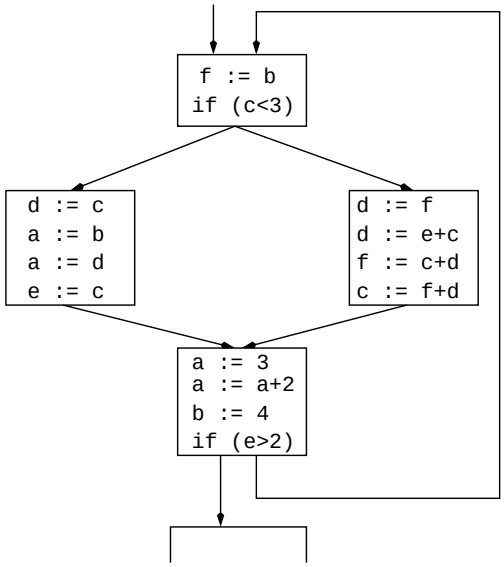
   (c) **(5pts)** Write the code for function call (with comments as above). To perform the actual call use the instruction `jal f`.

---

6. **Liveness analysis and register allocation (10pts total).** The top of the next page contains a control-flow graph that will be used. In Figure 1 (on page 9) there is a "magnified" copy of this graph which can be used to write your solutions (within the given braces).

   (a) **(2.5pts)** Given that only variable `e` is live on exit of the loop (as shown in Figure 1), what variables are live at each program point? (Use page 9 for your answer).

   (b) **(1.5pt)** Purely based on the information in your answer to question 6a above, what is the minimum number of colors that we need to color the register allocation graph? Please give a brief reason for your answer.



(c) **(2pts)** Draw and color the register interference graph. Use the minumum number of colors possible.

(d) **(1.5pts)** Using the liveness information collected in question 6a, cross out any statement in the program that is unnecessary. Again use the supplied page with the figure for your answer.

(e) **(2.5pts)** Can the optimization performed in 6d affect the register allocation done earlier? If so, how? If not, why?

---

7. **"Philosophical" question (1pt for any reasonable answer + 1pt "bonus" if your answer is similar to the lecturer's!).** In your opinion, what is the *single* most important reason why it is worthwhile for a student in your study program to have taken a course like 'Compiler Design 1' during her/his studies? One or two sentences will do (no essays please!).

Good luck !

Page to be used for answers to Question 1

Answer to Question 1

(a) Regular expression for all strings that do not end with aa:

The DFA for the language is:

(b) Regular expression for all strings that contain an even number of b's:

The DFA for the language is:

(c) Regular expression for all strings that do not contain the substring ba:

The DFA for the language is:

Page to be used for answers to Questions 2, 3, and 4

Answer to Question 2

(a) The LL(1) parsing table contains multiple entries for the pair:

The entries are:

(b) A grammar which is unambiguous, left-factored, not left-recursive and also not LL(1) is:

Answer to Question 3

1. $First(A) = \{x, y\}$ is correct for grammars:
2. $Follow(A) = \{\$, x\}$ is correct for grammars:
3. $Follow(B) = \{\$, x, y\}$ is correct for grammars:
4. $First(C) = \{y\}$ is correct for grammars:
5. $Follow(C) = \{\$, x\}$ is correct for grammars:

Answer to Question 4

1. With call-by-value and lexical scope the program prints:
2. With call-by-value and dynamic scope the program prints:
3. With call-by-reference and lexical scope the program prints:
4. With call-by-reference and dynamic scope the program prints:

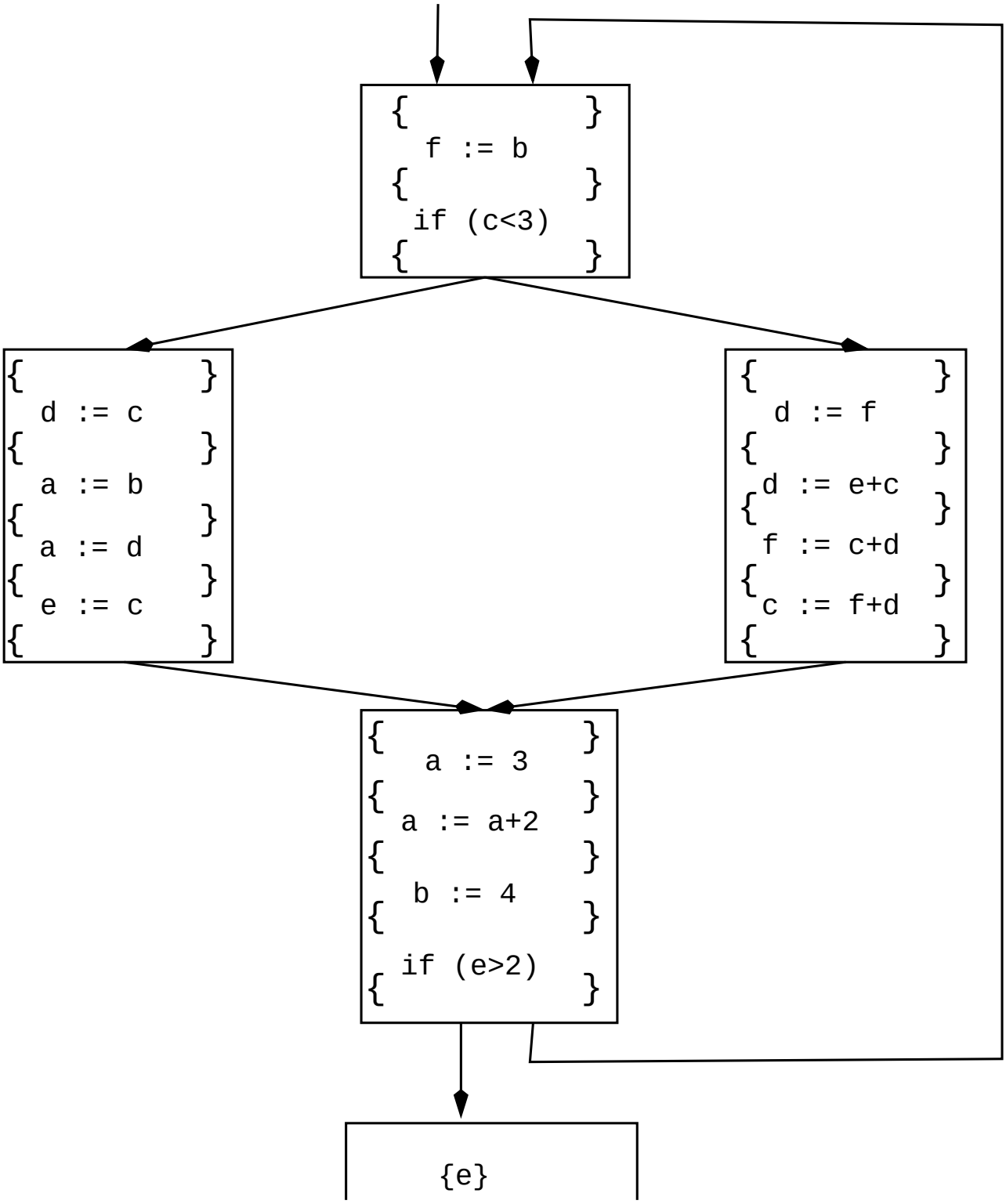


Figure 1: Control flow graph to be used for the answers to Questions 6a and 6d.