# TU Berlin

## Compiler Design – WS 2015/16 – Sample Questions

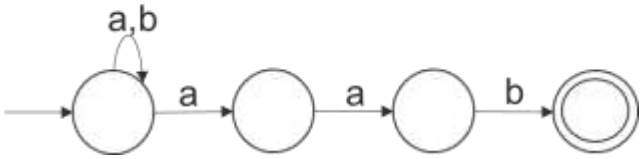
Dozent: Dr. Biagio Cosenza, Prof. Ben Juurlink

This is a set of sample questions, provided to have an idea of the final written examination. You should be prepared on all the topics discussed during the course, not only the one here. Type and number of questions are **not** the same of the final exam.

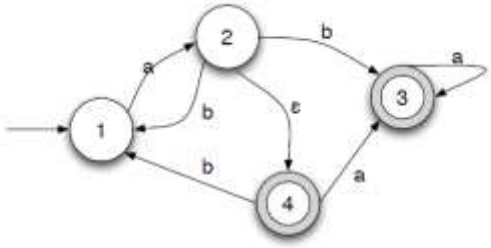
---

**Lexical Analysis**

1. Write a regular expression for all strings of **a**s and **b**s which contains the substring **abba**
   **Answer:** (a|b)*abba (a|b)*
2. Write a regular expression for strings of **x**s and **y**s where every **y** is immediately followed by at least 3 **x**s
   **Answer:** (x | (yxxx))*
3. Build a NFA for the RE: (a|b)*aab
   **Answer:**

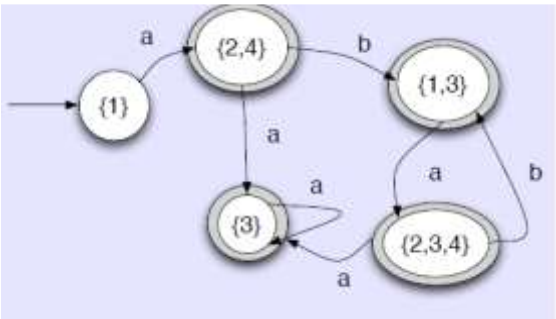



4. Convert the following NFA to a DFA (write both the transition table and diagram for the output DFA, use the correct notation to mark initial state and accepted states):



**Answer:** Apply the subset construction algorithm

| NFA | DFA | a | b |
|---|---|---|---|
| {1} | A | {2,4} | |
| {2,4} | B | {3} | {1,3} |
| {3} | C | {3} | |
| {1,3} | D | {2,3,4} | |
| {2,3,4} | E | {3} | {1,3} |




5. Given the grammar

   S → ABC
   A → aA | ε
   B → bB | ε
   C → cC | ε

   show a leftmost-derivation for the string abbc (at each step underline which non-terminal is derived).

   **Answer:**

   **S** ⇒ **A**BC ⇒ a**A**BC ⇒ a**B**C ⇒ ab**B**C ⇒ abb**B**C ⇒ abb**C** ⇒ abbc**C** ⇒ abbc

**Syntax and Semantic Analysis**

1. Write a Context-Free grammar for the language of all string of open and close parentheses, where the parentheses are balanced.

   **Answer:**

   S → ( S ) | S S | ε

2. Consider the following grammar:

   S → S a S b
   | c
   | Q q
   Q → Q m
   | ε

   Which non-terminals (if any) can derive ε?

   **Answer**: Q

3. What are the FIRST sets of Q and S?

   **Answer**: FIRST(Q) = {m} and FIRST(S) = {m, q, c}

4. Rewrite the grammar so that it accepts the same language but can be parsed by an LL(1) parser
**Answer**: The grammar is left recursive. LL parsers cannot handle left recursion, and this applies for both LL(0) and LL(1). Thus, the answer is the left-factored grammar:

```
S → c S'
  | Q q S'
S'→ a S b S' | ε
Q → m Q | ε
```

5. Associate semantic rules, using only synthesized attributes, to the production rules of the following grammar to calculate the value of the entire expression. The terminal symbol digit may have any value from 0 to 9.

```
G → E
E → E + T
E → T
T → T * F
T → F
F → ( E )
F → digit
```

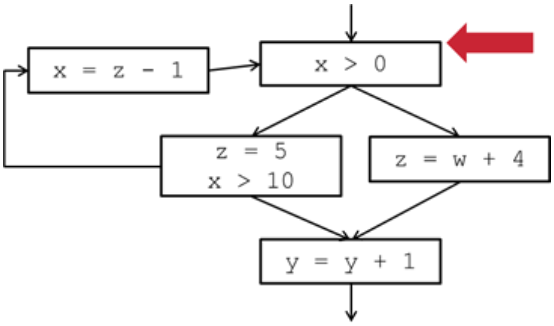
**Answer**:
In synthesized attributes, the value is computed from the values of attributes of the children. Don't forget to use subscript to distinguish symbols in the production rule.

| | |
|---|---|
| G → E | G.val = E.val |
| E → E + T | E.val = $E_1$.val + T.val |
| E → T | E.val = T.val |
| T → T * F | E.val = $T_1$.val x F.val |
| T → F | T.val = F.val |
| F → ( E ) | E.val = T.val |
| F → digit | F.val = digit.lexval |

6. Explain the difference between the following derivations: $\alpha \Rightarrow \beta, \alpha \overset{*}{\Rightarrow} \beta, \alpha \overset{+}{\Rightarrow} \beta, \alpha \underset{lm}{\Rightarrow} \beta, \alpha \underset{rm}{\Rightarrow} \beta$
**Answer**: derivation in one step, derivation in zero or more steps, derivation in one or more steps, leftmost derivation (the leftmost nonterminal in each sentential is always chosen), rightmost derivation (the rightmost nonterminal in each sentential is always chosen)

**Intermediate Representations, Dataflow Analysis, SSA, Dominators**

1. After running the liveness analysis algorithm to completion, which of the w, x, y and z are live at the program point indicated by the red arrow?

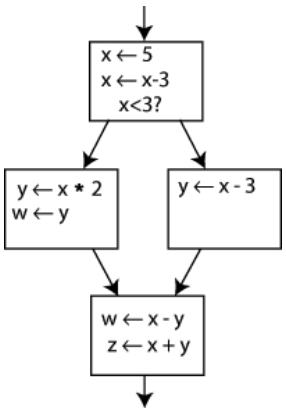



**Answer**: {w, x, y}

2. Write the control flow graph with basic block for the following C code:

```
x = 5;
x = x-3;
if (x < 3) {
  y = x * 2;
  w = y;
}
else {
  y = x - 3;
}
w = x - y;
z = x + y;
```

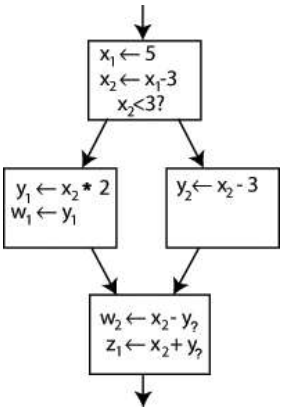
**Answer:**



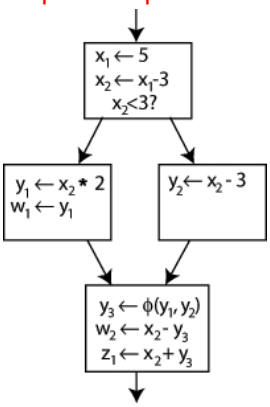

3. Rewrite the previous control flow graph in SSA form and indicate where are phi-nodes.

**Answer:** This code is so simple that can be transformed in SSA-form in only two simple steps (dominance frontiers here are trivial).

Step 1: single assignment variables



Step 2: add phi node on dominance frontiers




4. Write the direction, the domain and the transfer function for the available expression dataflow problem
   **Answer:**
   domain: set of expressions
   direction: forward
   tr. fun.:
   $In(s) = \cap_{s' \in pred(s)} Out(s')$
   $Out(s) = Gen(s) \cup (In(s) - Kill(s))$

5. Given two nodes A and B in a control flow graph, what does it mean that "A dominates B"?
   **Answer:** the node A dominates B if A appears on every path from Entry to B

**Runtime, Codegen, Registry Allocation**

1. What is an activation record? Describe the contents of an activation record.
   **Answer:** (or stack frame) is a region of memory arranged by the compiler for each call to a procedure (runtime support). It is composed of parameters, register save area, return value, return address, access link (may), caller's AR (may), local var. and temporaries
2. Assume a target processor with only three physical registers and apply Best's algorithm to the following code:

```
load r1,@a
load r2,@y
mult r3,r1,r2
load r4,@x
sub  r5,r4,r2
load r6,@z
mult r7,r5,r6
sub  r8,r7,r3
add  r9,r8,r1
```

   Write live intervals and the final code.
   **Answer:** also check lecture 10, slide 9

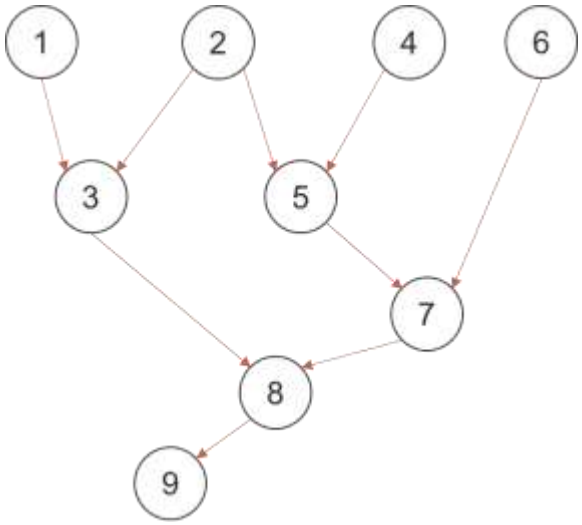
step 1, calculate live intervals

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| r1 | 2 | [1,9] | * | * | * | * | * | * | * | * | * |
| r2 | 2 | [2,5] | | * | * | * | * | | | | |
| r3 | 1 | [3,8] | | | * | * | * | * | * | * | |
| r4 | 1 | [4,5] | | | | * | * | | | | |
| r5 | 1 | [5,7] | | | | | * | * | * | | |
| r6 | 1 | [6,7] | | | | | | * | * | | |
| r7 | 1 | [7,8] | | | | | | | * | * | |
| r8 | 1 | [8,9] | | | | | | | | * | * |
| r9 | - | [9,9] | | | | | | | | | * |
| # of live values | | | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 3 | 2 |

1. load p1,@a
2. load p2,@y
3. mult p3,p1,p2
4. **store p1** // spill the one used farthest
5. load p1,@x
6. sub p1,p1,p2
7. load p2,@z
8. mult p1,p1,p2
9. sub p1,p1,p3
10. **load p2,** … // load spilled value (load r2,@a)
11. add p1,p1,p2

**Instr. Scheduling, Optimizations, Loop transformations**

1. Draw the precedence graph, for the code in the previous exercise.
   **Answer:**




2. Apply constant propagation and dead code elimination to the following C code:

```
x=5;
if(x>5) printf("foo");
else printf ("woo");
```

**Answer:** `printf("woo");`