UNIVERSITY OF OSLO

Faculty of mathematics and natural sciences

Examination in INF5110 — Kompilatortteknikk

Day of examination: 12. June 2018

Examination hours: 09.00 – 13:00

This problem set consists of 11 pages.

Appendices: 2 pages

Permitted aids: All written and printed

Please make sure that your copy of the problem set is complete before you attempt to answer anything.

- You should read the whole problem set before you start, getting an overview can help to make wise use of the time.
- Besides writing in a readable manner, draw requested figures in a clear way.
- Give concise and clear explanations!
- You may answer parts of Problem 4 and 5 by filling in the pages in the appendix and hand them in together with the rest of the answers (in the "white version").

Good luck!

(Continued on page 2.)

Problem 1 Regular expressions (weight 10%)

1a Regular languages (weight 2%)

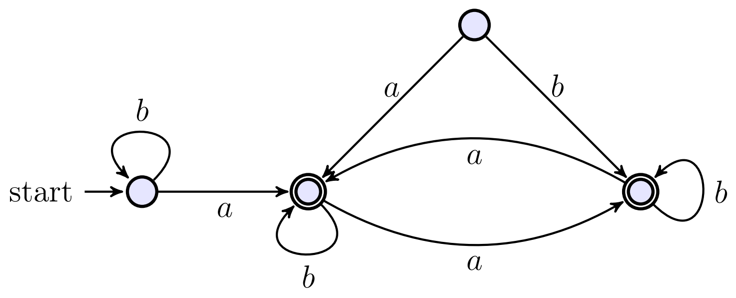
Are the following statements true or not?

(a) Every subset of a regular language is regular.

(b) The union of two regular languages is regular.

1b Automata and regular expressions (weight 4%)

Consider the following finite state automaton over the alphabet $\Sigma = \{\boldsymbol{a}, \boldsymbol{b}\}$:



Which of the following regular expressions captures the language of the automaton? One answer only.

$$\boldsymbol{b^*ab^*ab^*ab^*} \tag{1}$$
$$(\boldsymbol{a} \mid \boldsymbol{b})^* \tag{2}$$
$$\boldsymbol{b^*a}(\boldsymbol{a} \mid \boldsymbol{b})^* \tag{3}$$
$$\boldsymbol{b^*ab^*ab^*} \tag{4}$$

1c Regular languages (weight 4%)

Consider the following regular expression

$$\boldsymbol{a^*b^*(ba)^*a^*}$$

over the alphabet $\Sigma = \{a, b\}$. Give the shortest word over Σ which is *not* in the language specified by the regular expression.

(Continued on page 3.)

Problem 2 Context-free grammars (weight 10%)

Consider the following context-free grammar.

$$
\begin{array}{lcl}
S & \rightarrow & A\ B \\
A & \rightarrow & \mathbf{x}B \mid \boldsymbol{\epsilon} \\
B & \rightarrow & \mathbf{y}A \mid \mathbf{z}B
\end{array}
\tag{5}
$$

2a Language (weight 3%)

Give 3 words in the language defined by the grammar.

2b Derivation (weight 3%)

Give an example of a derivation starting from S in *two steps by a right-most* derivation, i.e., give an example of an α in the following situation

$$S \Rightarrow_r \ \Rightarrow_r \alpha$$

2c Parse tree (weight 4%)

Draw a parse tree for a derivation for the sentential form $\mathbf{xyy}A$, i.e., for the deriviation

$$S \Rightarrow^* \mathbf{xyy}A$$

(Continued on page 4.)

Problem 3 Top-down parsing (weight 25%)

Consider the following context-free grammar (for a simple form of "email-addresses"):

$$
\begin{array}{rcl}
\mathit{Addr} & \to & \mathit{Name}\,@\,\mathit{Name}\ \textbf{.}\ \textbf{id} \\
\mathit{Name} & \to & \textbf{id} \mid \textbf{id}\,\textbf{.}\,\mathit{Name}
\end{array}
\tag{6}
$$

3a LL(1)-table (weight 5%)

Give the LL(1) parsing table for the grammar. Point out LL(1)-conflict(s) in the table.

3b Left factorization (weight 4%)

One reason for the conflict in the previous subtask is that the grammar suffers from a *common left factor*. Repair *this* conflict by transforming the given grammar from equation (6) using the left-factorisation algorithm. Giving the resulting left-factored grammar suffices as answer.

3c LL(1)-table again (weight 8%)

Give the LL(1) parsing table for the grammar after having performed left-factorization in subproblem 3b.

3d LL(1) parsing (weight 8%)

Find an equivalent grammar for the language described by the grammar (6) which is LL(1)-parseable. It's not required to provide another table as part of the solution, an LL(1)-grammar is enough.

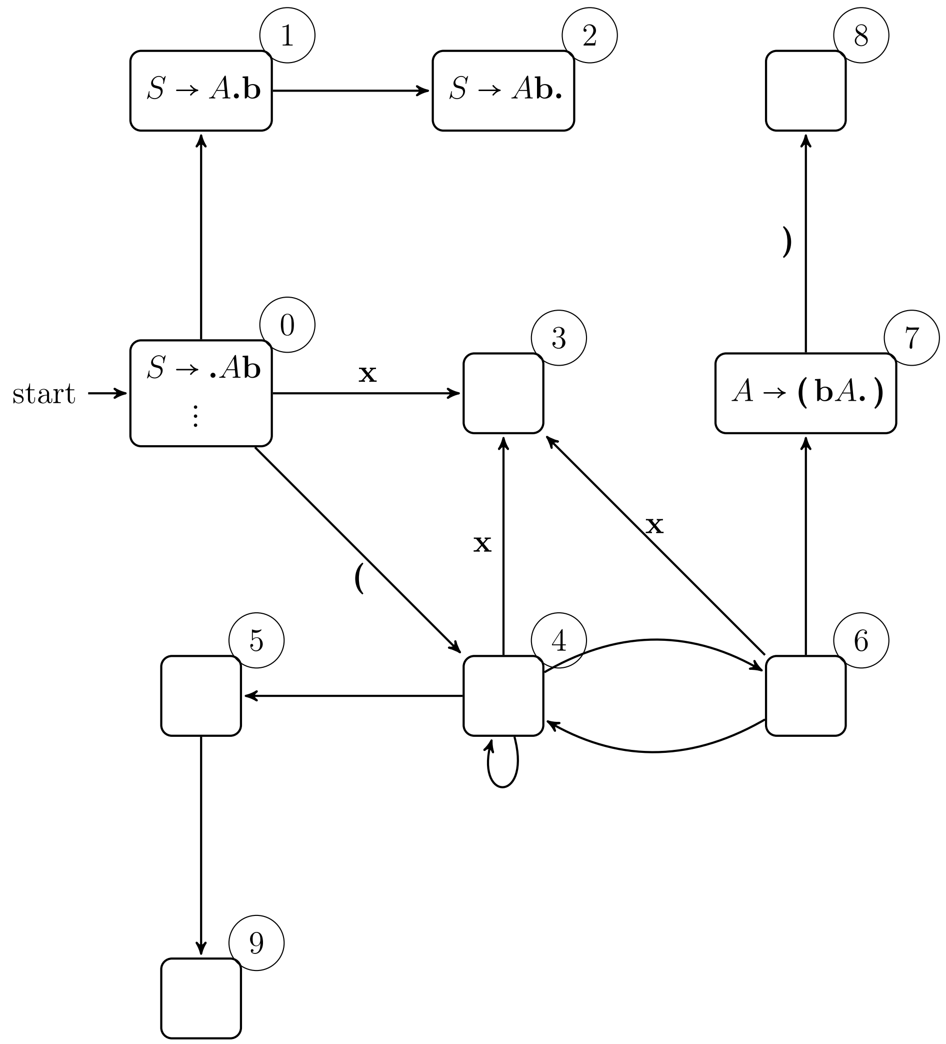
(Continued on page 5.)

Problem 4 Bottom-up parsing (weight 25%)

Consider the following context-free grammar.

$$\begin{array}{lcl} S & \rightarrow & A\mathbf{b} \\ A & \rightarrow & \mathbf{(}\,\mathbf{b}A\,\mathbf{)} \mid \mathbf{(}\,A\,\mathbf{)} \mid \mathbf{x} \end{array} \tag{7}$$

Note: the start symbol S does not occur on the right-hand side of any production, so don't extend the grammar by an additional start symbol S'. The following figure partially shows an LR(0)-DFA for the grammar. All states and all transitions of the automaton are given. Left out are some of the *LR(0)-items* inside the states. Also some of the *transition labels* are left out.



4a Fill in state 0 (weight 5%)

Fill out the remaining items in the starting state 0.

(Continued on page 6.)

4b Fill in the rest (weight 8%)

Fill out the remainder of the automaton, including all items and including the labels on the transitions.

The automaton is reproduced in the attachment, which you can use for your solution. It's advisable to make a sketch first on a separate sheet, to copy it in (readably) afterwards.

4c Complete items (weight 5%)

- List the states (giving their numbers) of the states containing a complete item.
- Which role do such states play in the context of bottom-up parsing.

4d Reduction (weight 7%)

Assume the parser parses the following string of terminals:

$$\mathbf{(\,b\,(\,x\,)\,)\,b}$$

List the *actions* or *steps* the parser does when parsing this word.

For shift-steps, indicate the state *into which* the automaton moves to. For example: write "shift-to-5" or `S5` when the action is do a shift step and moving to state 5. For *reduce step*, indicate also the *rule* of the grammar used for reduction. Assume the rules of the grammar from equation (7) numbered from I, II, III, and IV in the order of appearance. For example, when doing a reduce step accoding to the second production $A \to \mathbf{(\,b}A\mathbf{)}$, write "reduce with *II*" or "`R II`" for the action.

It's not required to give the sequence of stack contents during the parse or the remaining inputs; the list of actions in the form indicated is enough as answer.

(Continued on page 7.)

Problem 5 Attribute grammars (weight 15%)

We want to do *symbolic differentiation* of polynomials using attribute grammars. As a reminder or illustration: the following is a polynomial expression over x and y in math notation:

$$x^3 + 10x + 4x^7 + 17y$$

It represents a function, let's call it $f(x, y)$, over real numbers. The derative of f over, for instance, the variable x, is often noted as $\frac{\partial f}{\partial x}$. The result of the differentiation, its derivative, is the following polynomial:

$$\frac{\partial f}{\partial x} = 3x^2 + 10 + 28x^6$$

Now: consider the following grammar specifying a (simplified) syntax of polynomials,[1] represented by the non-terminal P; the right-hand side **deriv (** var, P **)** of D represent the derivative of the polynomial over the specified variable represented by var.

$$\begin{array}{rcl} D & \to & \textbf{deriv}\,\textbf{(}\,var\,\textbf{,}\,P\,\textbf{)} \\ P & \to & P\,\texttt{+}\,T \mid T \\ T & \to & C \mid C\ \ast\, var \uparrow C \\ C & \to & \textbf{const} \\ var & \to & \textbf{id} \end{array}$$

Assume that the nonterminal C has an attribute `val`, with the value of the constant already filled out. Assume for the value a *positive* integer. Assume further an attribute `name` for the nonterminal var, which also is already filled in.

Now:

> design an *attribute grammar* that, when evaluated on a syntax tree, gives the "symbolic derivation" of a given non-terminal D.

Use an attribute D.`deriv` to contain the symbolic derivation and an attribute `name` containing the name of the variable which is used in the derivative (x in the example $\frac{\partial f}{\partial x}$). Concretely:

(a) fill out the semantic rules in the following table, making appropriate use of attributes. C.`val` is already filled out with a positive integer, var .`name` with a string.

[1] The simplification is: we don't consider "mixed" summands like $7x^3y^4$; this is for making the task easier. We also simplify in that we don't consider negative numbers and `-`, as it would just add cases analogous to `+`.

(Continued on page 8.)

(b) Indicate for each of our attributes whether its *synthesized, inherited*, or neither-nor.

You can use *string* as the type for the attribute `deriv`. Also: for convenience, you can make use of "+" for concatenating strings (as in Java).

You may use the corresponding form in the appendix (by tearing it out and deliver it with the "white sheets").

	productions/grammar rules	semantic rules
1	$D \rightarrow \textbf{deriv}\,\textbf{(}\,var, P\,\textbf{)}$	
2	$P_0 \rightarrow P_1 + T$	
3	$P \rightarrow T$	
4	$T \rightarrow C_1 * var \uparrow C_2$	
5	$T \rightarrow C$	
6	$var \rightarrow \textbf{id}$	$var\,.\texttt{name} = valueof(\textbf{id})$
7	$C \rightarrow \textbf{const}$	$C.\texttt{val} = valueof(\textbf{const})$

□

(Continued on page 9.)

Problem 6 Code generation (weight 15%)

Assume code generation as covered in the "notat" which corresponds to parts of Chapter 9 of the old "dragon book" (*Compilers: Principles, Techniques, and Tools, A. V. Aho, R. Sethi, and J. D. Ullman, 1986*).

For all subproblems here: it's not required to give exactly the generated code as answer.

6a Registers (weight 5%)

Assume 3 registers, all initially free. With the code generation from the notat and assuming *local* liveness information: Would increasing the number of registers beyond 3 *improve* ("optimize") the code generated from Listing 1? Explain.

Listing 1: Three address intermediate code

```
 z :=  x + y;
t1 :=  z + y;
 z :=  y + x;
```

6b No liveness information (weight 5%)

The code generator from the lecture makes use of liveness information, Give a simple example of straight-line three-address intermediate code, where *ignoring* all liveness information leads to less good code. Point out in your three-address example one occurence of a variable where this happens ("line 2, variable x on the right-hand side" or similar). Explain.

6c Optimization (weight 5%)

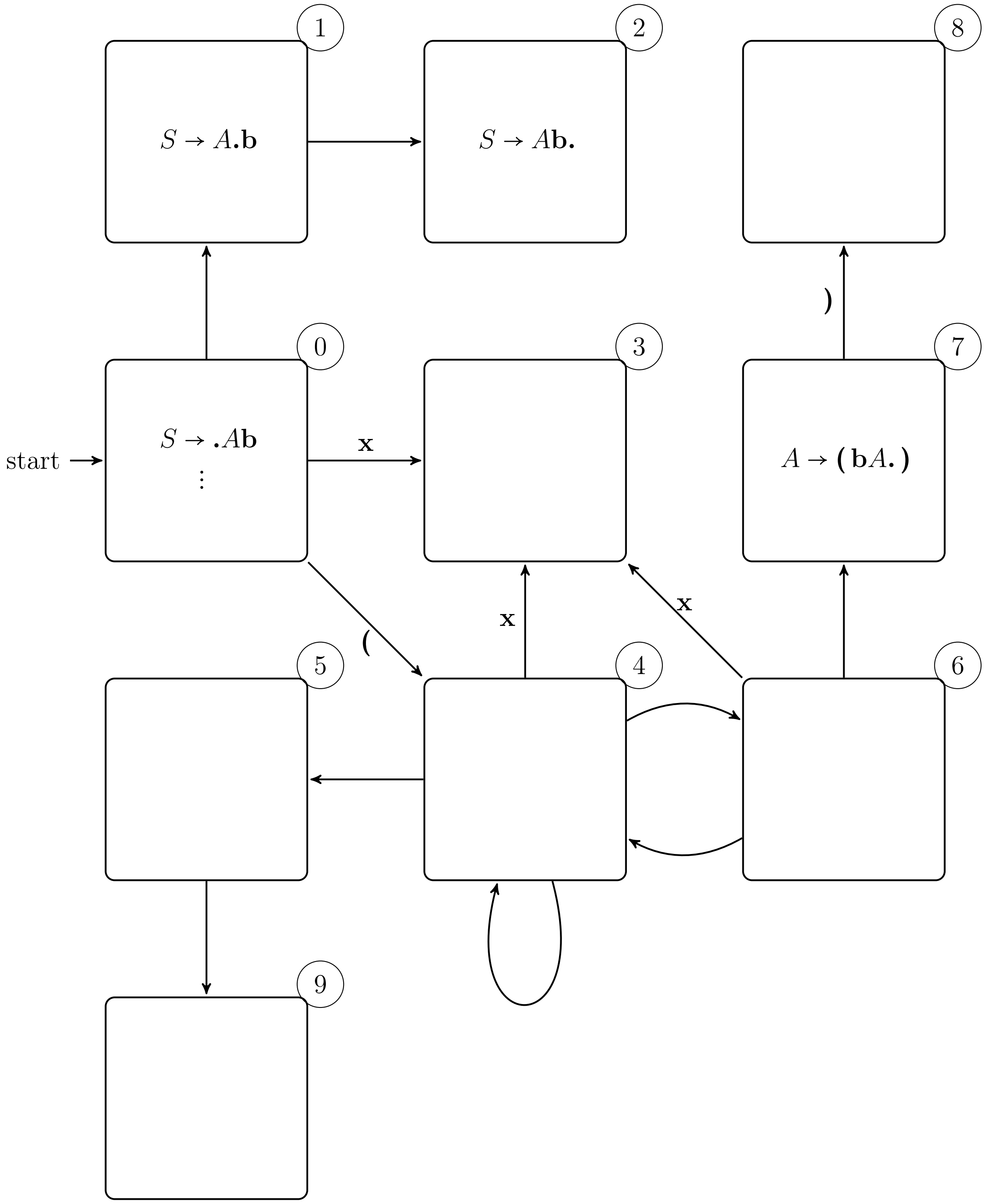
Concerning the code generated from Listing 1: give *two* possible ways one could improve the generated code compared to the code generator from the lecture.

(Continued on page 10.)

Appendix: DFA for Problem 4

Candidate nr.: ....

Date: ....



(Continued on page 11.)

Appendix: Form for Problem 5

Candidate nr.: ....

Date: ....

	productions/grammar rules	semantic rules
1	$D \rightarrow$ **deriv** **(** var **,** P **)**	
2	$P_0 \rightarrow P_1$ **+** T	
3	$P \rightarrow T$	
4	$T \rightarrow C_1$ ***** $var \uparrow C_2$	
5	$T \rightarrow C$	
6	$var \rightarrow$ **id**	var .name = $valueof$ (**id**)
7	$C \rightarrow$ **const**	C.val = $valueof$ (**const**)