Quiz

Compiler Construction, Fall 2014

Wednesday, November 5th, 2014

Version 4

General notes about this quiz

- Have your CAMIPRO card ready on the desk.
- You are allowed to use any printed material (using standard fonts, no cursive or script fonts) that you brought yourself to the exam. You are not allowed to use any notes that were not typed-up. Also, you are not allowed to exchange notes or anything else with other students taking the quiz.
- Use separate sheets, for each question, to write your answers. No sheet of paper should contain answer to two or more questions at the same time.
- Make sure you write your name on each sheet of paper.
- Use a permanent pen with dark ink.
- It is advisable to do the questions you know best first.
- You have in total **2 hours 50 minutes**.

Exercise	**Points**	
1	15	
2	20	
3	25	
4	20	
Total	80	

Problem 1: Lexer for Logical Formulas (15 points)

a) **[10 pts]** Design a lexical analyser that given a stream of characters belonging to the set $\{a, 0, 1, !, =, >, <\}$, tokenizes it into the tokens listed below. The tokens are shown on the left and their defining regular expressions are shown on the right.

Token name	Regular expression
ID	$a(a\|0\|1)^*$
INT	$(0\|1)^*$
NOT	!
IMP	=>
EQV	<=>
EQ	=
LE	<=
GE	>=

b) **[5 pts]** Describe the output (including indicating generated tokens) produced by running your lexer on the following inputs:

i) $a11$ <=>>= $10a10$ <=====>= $a111a1$

ii) $a1$ <=>= $a10$ <=>> a

Mention also the characters of the input stream that correspond to any generated tokens.

Problem 2: Tail-recursive Grammars (20 points)

Consider the grammar shown below where S is the start non-terminal symbol of the grammar, P, Q and R are non-terminals, and a, b are terminals.

$$
\begin{aligned}
S &::= a\ S \mid b\ S \mid a\ P \\
P &::= a\ Q \mid b\ R \\
Q &::= a\ P \mid b\ R \\
R &::= a\ R \mid b\ R \mid \texttt{""}
\end{aligned}
$$

a) **[3 pts]** Find a string accepted by the grammar that has two parse trees. Depict these two parse trees.

b) **[7 pts]** Construct an unambiguous grammar that is equivalent to the grammar shown above.

c) **[10 pts]** We say that a grammar is *tail-recursive* iff every production in the grammar is of the form $A ::= wB$ or $A ::= w$, where w is a (possibly empty) string of terminals, A and B are non-terminals. Describe a procedure that given any tail-recursive grammar constructs an equivalent unambiguous grammar. Your procedure can use other conversion algorithms explained in the lectures as sub-steps and it should be possible in principle to implement it as a transformation that accepts a grammar and produces another grammar.

Problem 3: Parser for Logical Formulas (25 points)

Consider the following grammar that accepts well-formed propositional formulas over the alphabet $A = \{\wedge, \vee, \neg,), (, atom\}$.

$$F' ::= F\ EOF$$
$$F ::= F \wedge F \mid F \vee F \mid \neg F \mid (\ F\ ) \mid atom$$

a) [3 pts] Is this grammar LL(1)? Justify your answer.

b) [7 pts] Modify the above grammar so that it accepts only formulas that are in *conjunctive normal form*, which is described below:

(a) We use the term literal to denote an atomic predicate *atom* or its negation i.e, $\neg atom$.

(b) A clause C_i is either a single literal or a disjunction of literals enclosed within parentheses. That is, each C_i is of the form l or $(l_1 \vee l_2 \vee \cdots \vee l_{m_i})$ where $l, l_1, \cdots, l_{m_i}$ are either *atom* or $\neg atom$.

(c) A conjunctive normal form formula is of the form $C_1 \wedge C_2 \wedge \cdots \wedge C_n$, for $n \geq 1$, where each C_i is a clause.

For example, the following formulas are in conjunctive normal form where a and b are atomic predicates.
a
$a \wedge b$
$(a \vee b)$
$\neg a \wedge \neg b$
$(a \vee \neg c) \wedge b$
$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b)$
Note that in a conjunctive normal form formula parentheses appear around clauses that contain more than one literal and nowhere else. If you are not sure whether some formula is in conjunctive normal form or not, do not hesitate to ask us.

c) [5 pts] Convert the grammar that you designed for accepting conjunctive normal form formulas to LL(1) if it is not already in LL(1).

d) [10 pts] Create the LL(1) parsing table for your grammar. Show the first and follow sets for each non-terminal of your LL(1) grammar.

Problem 4: CYK Parsing (20 points)

Consider the following simple grammar for lambda calculus, where *term* is a non-terminal, and *ID*, λ, (,) and . are terminals

$$term ::= \lambda\ ID\ .\ term \mid term\ term \mid (\ term\ ) \mid ID$$

a) [10 pts] Convert this grammar to Chomsky Normal Form by ensuring the following properties.

1. terminals t occur alone on the right-hand side: X ::= t
2. no unproductive non-terminal symbols

3. no productions of arity more than two
4. no nullable symbols except for the start symbol
5. no single non-terminal productions X ::= Y
6. no non-terminals unreachable from the start symbol

It is sufficient if you only write the final grammar that you obtain.

b) **[5 pts]** Use the CYK algorithm and your normal form to construct a CYK parse table for the string: "$\lambda\ y\ .\ x\ y\ x$". Is this string accepted by the grammar ?

c) **[5 pts]** How many parse trees exist for the above string ? Justify your answer using the CYK parse table for the string. Note that every parse tree for the string should have the start symbol at the root and should cover the entire string.

Hint: If a substring $w_{i,j}$ can be parsed by partitioning it into two in n ways, the number of parse trees for $w_{i,j}$ is the sum of the number parse trees for each partition. Given a partition $w_{i,p}$ and $w_{p+1,j}$ of $w_{i,j}$, the number of parse trees for the partition is the product of the number of parse trees for $w_{i,p}$ and $w_{p+1,j}$.