AOR Priv.-Doz. Dr. Thomas Noll

Friedrich Gretz, Christina Jansen

Exercise 1 (Prefix-free Languages): (2 Points)

A language $L \in \Sigma^*$ is called prefix-free, if $L \cap L\Sigma^+ = \emptyset$, i.e. if no proper prefix of a word in L is in L, too.

Show that the following holds for all non prefix-free languages L: $L \notin \mathfrak{L}(LR(0))$.

Exercise 2 (Causes of Nondeterminism): (5 Points)

In the lecture we have seen four different causes, that led to nondeterminism in a NBA:

1. choice between shift and reduce
2. choice between different left-hand sides of the productions
3. choice between different right-hand sides of production
4. choice on when to terminate parsing

a) Explain in what situations point 4 may appear and how to resolve it.

b) Points 1, 2 and 3 may introduce conflicts into LR(0)-items. Please give minimal examples for those conflicts (provide a grammar + a conflicting LR(0) item).

c) We considered reduce/reduce as well as shift/reduce conflicts in LR(0)-items. Explain briefly why we do not have to consider shift/shift conflicts.

Exercise 3 (Nondeterministic Bottom-Up Automaton): (3 Points)

Consider the following grammar G:

$$\begin{array}{lcl} S & \rightarrow & expr \\ expr & \rightarrow & ID \mid ID + expr \end{array}$$

a) Specify $NBA(G)$ (for the transitions it suffices to provide examples for shifting and reduction steps).

b) Give an accecpting run of $NBA(G)$ on the input: $ID + ID$.

c) Is $G \in LR(0)$? Argue why!

Exercise 4 (Implementation of LR(0) sets): (5 Points)

After building a lexer in the previous exercise we now want to build a parser for our WHILE language. In this exercise we take the first steps towards a parser.
Assume the following grammar for the *WHILE* language. The terminal alphabet is the set of tokens (coming from the lexer), non-terminals and starting symbol are obvious. The production rules are given below:

start	→	program
program	→	program program | statement
statement	→	declaration SEM | assignment SEM | branch | loop | out SEM
declaration	→	INT ID
assignment	→	ID ASSIGN expr | ID ASSIGN READ LBRAC RBRAC
out	→	WRITE LBRAC expr RBRAC | WRITE LBRAC STRING RBRAC
branch	→	IF LBRAC guard RBRAC LCBRAC program RCBRAC |
		IF LBRAC guard RBRAC LCBRAC program RCBRAC ELSE LCBRAC program RCBRAC
loop	→	WHILE LBRAC guard RBRAC LCBRAC program RCBRAC
expr	→	NUM | ID | subexpr | LBRAC subexpr RBRAC
subexpr	→	expr PLUS expr | expr MINUS expr | expr TIMES expr | expr DIV expr
guard	→	relation | subguard | LBRAC subguard RBRAC | NOT LBRAC guard RBRAC
subguard	→	guard AND guard | guard OR guard
relation	→	expr LT expr | expr LEQ expr | expr EQ expr | expr NEQ expr | expr GEQ expr | expr GT expr

Task:
Your program should generate the $LR(0)$ items from a hard-coded grammar. List all $LR(0)$ sets and their items like in Example 9.13 in the lecture. Indicate where conflicts occur. Give the total number of sets and conflicts. Please summarize the main implementation ideas briefly and hand in your documentation together with the other written exercises. Some questions that you should answer there are:

- what is your data structure for the grammar?
- which necessary methods does it provide?
- how are the $LR(0)$ items and sets stored?
- how do you detect conflicts?