

Compiler Construction 2016

— Series 4 —

Hand in until May 31st before the exercise class.



Credit: <https://xkcd.com/1421/>

General Remarks

- Follow the naming convention for the zip file: `ex4_MATRNO1_MATRNO2_MATRNO3` and include the complete framework provided to you via our webpage.
- It is allowed to hand in your solutions for the theoretical part via email as a **separately attached PDF file**.
- Please hand in your solutions in groups of 3 or 4.

Exercise 1

(3 Points)

Show that every regular language can be generated by a $LL(1)$ -grammar.

Solution: _____

Assume the regular language to be L over alphabet Σ . Then there exists a DFA $\mathfrak{A}$ over Σ , such that $L(\mathfrak{A}) = L$. Now construct a grammar $G = (Q, \Sigma, P, q_0)$ out of $\mathfrak{A}$:

$$P : \begin{aligned} q &\rightarrow aq', \text{ if } \delta(q, a) = q' \\ q &\rightarrow \varepsilon, \quad \text{if } q \in F \end{aligned}$$

By construction it is clear that $L(G) = L(\mathfrak{A})$. (Note: G is by construction right linear/right regular)

Now show: $G \in LL(1)$.

Let $\Sigma = \{a_1, \dots, a_n\}$. Then for all $q \in Q$ it holds that

1. $q \rightarrow a_1 q_1 \mid \dots \mid a_n q_n \in P$
2. $q \rightarrow \varepsilon \in P$, if $q \in F$

We calculate the lookahead sets:

1. $la(q \rightarrow a_i q_i) = fi(a_i q_i fo(q)) = \{a_i\}$
Therefore the lookahead sets are different for all production rules from 1.
2. $la(q \rightarrow \varepsilon) = fi(\varepsilon fo(q)) = fi(fo(q)) = fo(q) = \{\varepsilon\}$
Therefore the lookahead sets are different for all production rules in P .

$\rightarrow G \in LL(1)$.

Exercise 2

(4 Points)

Consider the grammar $G = (N, \Sigma, P, start)$ covering some boolean expressions:

- $N := \{start, guard, rel\}$
- $\Sigma := \{AND, OR, ID, EQ, LEQ\}$
- $$\begin{array}{ll} start & \rightarrow guard \\ guard & \rightarrow rel \mid guard \text{ AND } guard \mid guard \text{ OR } guard \\ rel & \rightarrow ID \text{ EQ } ID \mid ID \text{ LEQ } ID \end{array}$$

- (a) Construct $NTA(G)$.
(Either give a transition table or depict the automaton and specify what the edge labelling means. Do not forget to give a numbering to the grammar rules.)
- (b) Provide a run of $NTA(G)$ on the input $ID \text{ EQ } ID \text{ AND } ID \text{ LEQ } ID$.
- (c) Construct an equivalent grammar G' with $G' \in LL(1)$.
- (d) Specify the deterministic top-down parsing automaton $DTA(G')$.
(Again, either give a transition table as in the lecture or depict the automaton and specify what the edge labelling means. As before, do not forget to give a numbering to the grammar rules of G' .)
- (e) Provide a run of $DTA(G')$ on the input $ID \text{ EQ } ID \text{ AND } ID \text{ LEQ } ID$.

Solution: _____

1. $start \rightarrow guard$
2. $guard \rightarrow rel$
3. $guard \rightarrow guard \text{ AND } guard$
4. $guard \rightarrow guard \text{ OR } guard$
5. $rel \rightarrow ID \text{ EQ } ID$
6. $rel \rightarrow ID \text{ LEQ } ID$

- (a) The non-deterministic push down automaton $NTA(G)$:

- Input alphabet: Σ
- Pushdown alphabet: $X = \Sigma \cup N$
- Output alphabet: $P = \{1, 2, 3, 4, 5, 6\}$
- Configurations: $\Sigma^* \times X^* \times P^*$
- Initial configuration: $(w, start, \varepsilon)$ for $w \in \Sigma^*$
- Final configuration: $\{\varepsilon\} \times \{\varepsilon\} \times P^*$
- Transitions: for $w \in \Sigma^*, \alpha \in X^*, z \in P^*$:

- (a) $(w, \text{start } \alpha, z) \rightarrow (w, \text{guard } \alpha, z1)$
 (b) $(w, \text{guard } \alpha, z) \rightarrow (w, \text{rel } \alpha, z2)$
 (c) $(w, \text{guard } \alpha, z) \rightarrow (w, \text{guard AND guard } \alpha, z3)$
 (d) $(w, \text{guard } \alpha, z) \rightarrow (w, \text{guard OR guard } \alpha, z4)$
 (e) $(w, \text{rel } \alpha, z) \rightarrow (w, \text{ID EQ ID } \alpha, z5)$
 (f) $(w, \text{rel } \alpha, z) \rightarrow (w, \text{ID LEQ ID } \alpha, z6)$
 (g) $(T w, T \alpha, z) \rightarrow (w, \alpha, z)$, where $T \in \Sigma$
- (b) The input word is $w := \text{ID EQ ID AND ID LEQ ID}$. An accepting computation is as follows:
- $(w, \text{start}, \varepsilon)$
 $\vdash (w, \text{guard}, 1)$
 $\vdash (w, \text{guard AND guard}, 13)$
 $\vdash (w, \text{rel AND guard}, 132)$
 $\vdash (w, \text{ID EQ ID AND guard}, 1325)$
 $\vdash^* (\text{ID LEQ ID}, \text{guard}, 1325)$
 $\vdash (\text{ID LEQ ID}, \text{rel}, 13252)$
 $\vdash (\text{ID LEQ ID}, \text{ID LEQ ID}, 132526)$
 $\vdash^* (\varepsilon, \varepsilon, 132526)$
- (c) $G' :=$
- | | | | |
|----|----------|---------------|---------------|
| 1. | start | $\rightarrow$ | guard |
| 2. | guard | $\rightarrow$ | rel subguard |
| 3. | subguard | $\rightarrow$ | ε |
| 4. | subguard | $\rightarrow$ | AND guard |
| 5. | subguard | $\rightarrow$ | OR guard |
| 6. | rel | $\rightarrow$ | ID subrel |
| 7. | subrel | $\rightarrow$ | EQ ID |
| 8. | subrel | $\rightarrow$ | LEQ ID. |

Look-head table for G' :

π	$\text{la}(\pi)$
1	ID
2	ID
3	ε
4	AND
5	OR
6	ID
7	EQ
8	LEQ

$\rightarrow G' \in LL(1)$.

- (d) The deterministic push down automaton $DTA(G')$:
- Input alphabet: Σ
 - Pushdown alphabet: $X = \Sigma \cup N$
 - Output alphabet: $P = \{1, 2, 3, 4, 5, 6, 7, 8\}$
 - Configurations: $\Sigma^* \times X^* \times P^*$
 - Initial configuration: $(w, \text{start}, \varepsilon)$ for $w \in \Sigma^*$
 - Final configuration: $\{\varepsilon\} \times \{\varepsilon\} \times P^*$
 - Transitions: transition table as follows

	start		guard		subguard		rel		subrel		
ID	(guard, 1)		(rel subguard, 2)				(ID subrel, 6)				
EQ									(EQ ID, 7)		
LEQ									(LEQ ID, 8)		
ε					(ε , 3)						
AND					(AND guard, 4)						
OR					(OR guard, 5)						
	ID	EQ	LEQ	ε	AND	OR					
ID	pop										
EQ		pop									
LEQ			pop								
ε				accept							
AND					pop						
OR						pop					

(e) Let $w := \text{ID EQ ID AND ID LEQ ID}$. The accepting computation is as follows:

$(w, \text{start}, \varepsilon)$
 $\vdash (w, \text{guard}, 1)$
 $\vdash (w, \text{rel subguard}, 12)$
 $\vdash (w, \text{ID subrel subguard}, 126)$
 $\vdash (\text{EQ ID AND ID LEQ ID}, \text{subrel subguard}, 126)$
 $\vdash (\text{EQ ID AND ID LEQ ID}, \text{EQ ID subguard}, 1267)$
 $\vdash^* (\text{AND ID LEQ ID}, \text{subguard}, 1267)$
 $\vdash (\text{AND ID LEQ ID}, \text{AND guard}, 12674)$
 $\vdash (\text{ID LEQ ID}, \text{guard}, 12674)$
 $\vdash (\text{ID LEQ ID}, \text{rel subguard}, 126742)$
 $\vdash (\text{ID LEQ ID}, \text{ID subrel subguard}, 1267426)$
 $\vdash (\text{LEQ ID}, \text{subrel subguard}, 1267426)$
 $\vdash (\text{LEQ ID}, \text{LEQ ID subguard}, 12674268)$
 $\vdash^* (\varepsilon, \text{subguard}, 12674268)$
 $\vdash (\varepsilon, \varepsilon, 126742683)$

Exercise 3

(3 Points)

After finishing the lexer, the next step is to implement a parser. The goal of this exercise is to build a recursive descent parser which transforms the list of symbols (returned from the lexer) into a list of grammar rules. The grammar is as follows and covers assignments:

1. $\text{start} \rightarrow \text{assignment SEMICOLON EOF}$
2. $\text{assignment} \rightarrow \text{INT ID ASSIGN expr}$
3. $\text{expr} \rightarrow \text{ID subexpr}$
4. $\text{expr} \rightarrow \text{NUMBER subexpr}$
5. $\text{expr} \rightarrow \text{LPAR expr RPAR}$
6. $\text{expr} \rightarrow \text{READ LPAR RPAR subexpr}$
7. $\text{subexpr} \rightarrow \text{PLUS expr}$
8. $\text{subexpr} \rightarrow \text{MINUS expr}$
9. $\text{subexpr} \rightarrow \text{TIMES expr}$
10. $\text{subexpr} \rightarrow \text{DIV expr}$
11. $\text{subexpr} \rightarrow \text{MOD expr}$
12. $\text{subexpr} \rightarrow \varepsilon$

Hint: as before we provide a framework which can be downloaded from the course webpage.

Implement the methods in `parser.RecursiveDescentParserAssignment` for the given grammar. The superclass `parser.RecursiveDescentParser` offers useful methods for getting the next token (`next()`),



printing a grammar rule (`print(id)`) and throwing an error (`printError(msg)`).

Test your implementation! For example, given the following input

```
int b = read() % 2;
```

your implementation should generate a list of grammar rules like this:

```
1, 2, 6, 11, 4, 12
```

Solution: _____

For the solution see the code in `While2JasminCompilerSolution`.