Assignment 5

George Gkikas-Panousis 1820253079
Anastasios Dyoilis 1820253075
Vlad Kovalev 1820253050

April 30, 2025

Introduction

In this assignment we solve five exercises on context-free grammars and parsers:

1. LL(1) analysis for seven grammars.
2. Proving ambiguity for three small grammars.
3. LR(0) automaton and parse table.
4. LR(1) automaton and parse table.
5. Fixing and extending a partial C++ EBNF.

Problem 1: LL(1) Analysis

Compute FIRST and FOLLOW sets, then check the LL(1) conditions.

Grammar 1

$$S \to a\,A\,B\,C \mid \epsilon, A \to a \mid bbD, B \to a \mid \epsilon, C \to b \mid \epsilon, D \to c \mid \epsilon.$$

FIRST:

$\text{FIRST}(S) = \{a, \epsilon\},\ \text{FIRST}(A) = \{a, b\},\ \text{FIRST}(B) = \{a, \epsilon\}, \text{FIRST}(C) = \{b, \epsilon\},\ \text{FIRST}(D) = \{c, \epsilon\}.$

FOLLOW:

$\text{FOLLOW}(S) = \{\$\},\ \text{FOLLOW}(A) = \{a, b, \$\},\ \text{FOLLOW}(B) = \{b, \$\}, \text{FOLLOW}(C) = \{\$\},\ \text{FOLLOW}$

All intersections are empty and the ϵ checks pass. *Grammar 1 is LL(1).*

Grammar 2

$$A \to B\,C\,c \mid e\,D\,B, B \to \epsilon \mid b\,C\,D, C \to D\,a\,B \mid c\,a, D \to \epsilon \mid d\,D.$$

FIRST: FIRST$(D) = \{d, \epsilon\}$, but $d \in$ FOLLOW(D) conflict. *Grammar 2 is NOT LL(1).*

Grammar 3

Clarified as:

$$S \to' ('\,X\,')' \mid' ['\,E\,']' \mid F, X \to E\,')' \mid F\,']', E \to A, \quad F \to A, \quad A \to \epsilon.$$

Compute FIRST and FOLLOW (all disjoint, no conflicts). *Grammar 3 is LL(1).*

Grammar 4

$$S \to X\,b \mid Y\,d, X \to a\,X \mid \epsilon, \quad Y \to c\,Y \mid \epsilon.$$

FIRST$(Xb) = \{a, b\}$ vs FIRST$(Y d) = \{c, d\}$ disjoint, ϵ-cases OK. *Grammar 4 is LL(1).*

Grammar 5

$$S \to M\,N\,O\,P\,Q, \quad M, N, O, P, Q \to t \mid \epsilon$$

Each $t \notin$ FOLLOW LL(1). *Grammar 5 is LL(1).*

Grammar 6

Eliminate left recursion:

$$S \to A, \quad A \to a\,B\,A', \quad A' \to d\,A' \mid \epsilon, \quad B \to b.$$

No conflicts LL(1). *Grammar 6 is LL(1).*

Grammar 7

$$S \to A\,a\,A\,b \mid B\,b\,B\,a, \quad A \to \epsilon,\ B \to \epsilon.$$

FIRST-sets $\{a\}$ vs $\{b\}$ LL(1). *Grammar 7 is LL(1).*

Problem 2: Ambiguity Proofs

Each grammar admits a string with two parse-trees:

- Grammar 1: $S \to a \mid aAb \mid abSb,\ A \to aAAb \mid bS$. String `abab`:

$$S \Rightarrow abSb \Rightarrow abab, \quad S \Rightarrow aAb \Rightarrow a(bS)b \Rightarrow abab.$$

- Grammar 2: $S \to AB \mid aaaB,\ A \to a \mid Aa,\ B \to b$. String `aaab`:

$$S \Rightarrow AB \Rightarrow Aa\,B \Rightarrow aa\,b, \quad S \Rightarrow aaaB \Rightarrow aaab.$$

- Grammar 3: $S \to xyXxX \mid xyXyy \mid yy,\ X \to xxX \mid yy \mid xx$. String `xyxxxxxyy` by two different expansions of X.

All three are ambiguous.

Problem 3: LR(0) Automaton

Augmented Grammar

```
0. S' -> S
1. S  -> A A
2. A  -> a A
3. A  -> b
```

SLR Parse Table

State	a	b	$	A	S
0	s3	s4		2	1
1			acc		
2	s3	s4		5	
3	s3	s4		6	
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

where r1: S→AA, r2: A→aA, r3: A→b.

Problem 4: LR(1) Automaton

Augmented Grammar

```
0. S' -> S
1. S  -> a A d
2. S  -> b B d
3. S  -> a B e
4. S  -> b A e
5. A  -> g
6. B  -> g
```

LR(1) ACTION / GOTO

State	a	b	g	d	e	$	A	B	S
0	s1	s2							13
1			s11				3	4	
2			s12				6	5	
3				s7					
4					s9				
5				s8					
6					s10				
7						r1			
8						r2			
9						r3			
10						r4			
11				r5	r6				
12				r6	r5				
13						acc			

r1:S→aAd, r2:S→bBd, r3:S→aBe, r4:S→bAe, r5:A→g, r6:B→g.

Problem 5: C++ Grammar Corrections

Below I show what I *fixed* (in red) and what I *added* (in blue).

```
<const-def>    ::= const <type> <var-name> = <expression> ;

<switch-stmt>   ::= switch(<expression>){<case-list><default-case>?}
<case-list>     ::= <case> | <case-list><case>
<case>          ::= case <literal> : <statement>*
```

```
<default-case>  ::= default : <statement>*

<class-def>     ::= class <var-name> { <member-list> } ;
<member-list>   ::= <member> | <member-list><member>
<member>        ::= <var-def> | <fun-def>

<try-catch>     ::= try{<statement>*}catch(<type><var-name>){<statement>*}

 Precedence-enforced expressions
<logical-or-expr>    ::= <logical-and-expr> ( "||" <logical-and-expr> )*
<logical-and-expr>   ::= <equality-expr>  ( "&&" <equality-expr>  )*
<equality-expr>      ::= <relational-expr> (("=="|"!=") <relational-expr>) *
<relational-expr>    ::= <additive-expr>  (("<"|"<="|">"|">=") <additive-expr>) *
<additive-expr>      ::= <multiplicative-expr> (("+"|"-") <multiplicative-expr>)*
<multiplicative-expr>::= <unary-expr>      (("*"|"/"|"%") <unary-expr>)*
<unary-expr>         ::= ("-"|"!") <unary-expr> | "(" <expression> ")" | <literal> |
```