

1. Consider the following grammar:

$$S \rightarrow a S b S \mid b S a S \mid \varepsilon$$

- Show that the grammar is ambiguous by constructing two different rightmost derivations for the string *abaabb*.
- Construct the corresponding parse trees for this string.
- Write an unambiguous grammar that describes the same language.

2. Consider the following grammar:

$$\begin{aligned} S &\rightarrow B C z \\ B &\rightarrow x B \mid B y \mid \varepsilon \\ C &\rightarrow u v \mid u \mid \varepsilon \end{aligned}$$

- Calculate nullable, FIRST, and FOLLOW sets.
- Construct the LL(1) parsing table and give evidence that this grammar is not LL(1).
- Give an LL(1) grammar which accepts the same language and build the LL(1) parsing table for that grammar.

3. Consider a simple grammar for pointer expressions in C, consisting of pointer dereference expressions, address-of expressions, assignments, and field accesses:

$$E \rightarrow *E \mid \&E \mid E = E \mid E -> E \mid \text{id}$$

This is an ambiguous grammar. We would like to write an unambiguous grammar for the same language, such that field accesses  $E -> E$  have higher precedence than dereferences and address-of expressions, and all of these have higher precedence than assignments.

- Write an LL(1) grammar which accepts the same language and has the desired operator precedence. Show the LL(1) parsing table for this grammar.
- Write an LR(1) grammar which accepts the same language, respects the desired operator precedence, and is such that assignments are right-associative, and field accesses are left-associative.
- Write the parse tree for the expression  $* * a -> b -> c = \& * d$  using the LR(1) grammar;
- One problem with the grammar above is that it models a superset of the valid C expressions. For instance,  $\&a = b -> *c$  is an invalid expression. We therefore impose the following conditions:
  - only a location can occur on the right-hand side of an assignment, where a location is either a dereference or an identifier;
  - only a location can occur in the address-of construct;
  - the address-of expression can only occur in dereferences or on the right side of an assignment;
  - the expression in the right-hand side of a field access must be an identifier.

Write a LR(1) grammar which precisely accepts this language and has the desired precedence and associativity of operators.

4. Consider the following grammar:

$$E \rightarrow \text{id} \mid \text{id} ( E ) \mid E + \text{id}$$

- (a) Build the LR(0) automaton for this grammar.
- (b) Is this an LR(0) grammar? Give evidence.
- (c) Is this an SLR grammar? Give evidence.
- (d) Is this an LR(1) grammar? Give evidence.

5. Consider the grammar of matched parentheses:

$$\begin{aligned} S &\rightarrow A \$ \\ A &\rightarrow ( A ) A \mid \varepsilon \end{aligned}$$

- (a) Construct the LR(1) automaton.
- (b) Build the LR(1) parsing table to show that the grammar is LR(1).
- (c) Is the grammar LR(0)? Justify your answer.