THEORY OF
COMPUTATION

TOC

CONTEXT-FREE GRAMMAR AND PARSING ALGORITHMS IN COMPILER DESIGN.

16010421073 - KEYUR PATEL
16010421076 - PRANAV PATIL
16010421077 - RUTUJA PATIL

CONTENT

COMPILER

A COMPILER IS A SPECIALIZED SOFTWARE TOOL THAT TRANSLATES HUMAN-READABLE SOURCE CODE WRITTEN IN A HIGH-LEVEL PROGRAMMING LANGUAGE INTO MACHINE-EXECUTABLE CODE (USUALLY IN THE FORM OF BINARY CODE OR ASSEMBLY LANGUAGE).

THEY TRANSLATE HIGH-LEVEL SOURCE CODE INTO A LOWER-LEVEL REPRESENTATION THAT CAN BE EXECUTED BY A COMPUTER'S HARDWARE.

COMPILER PHASES

Parsing

Structurally analyzing source code.

Intermediate Code Generation

Producing an intermediate representation.

Lexical Analysis

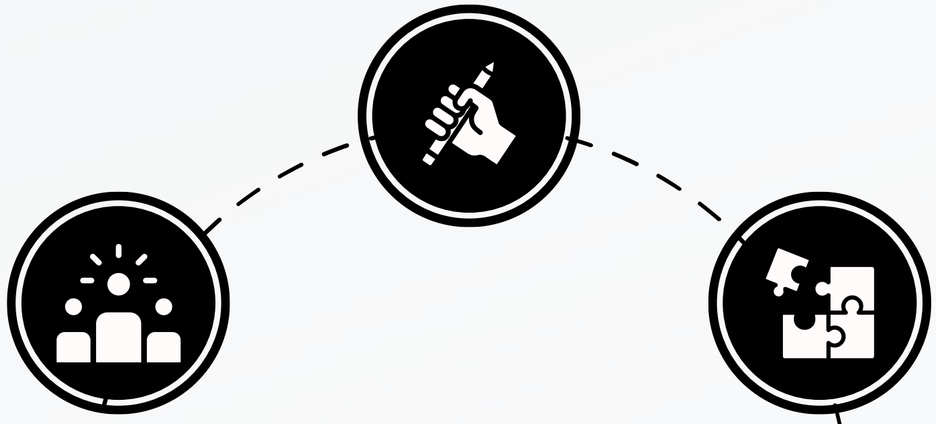
Scanning and tokenizing source code.

Semantic Analysis

Ensuring semantic correctness.

Code Generation

Generating target machine code.



CONTEXT-FREE GRAMMAR

The Context Free Grammar(CFG) is a collection of four-tuple (V,T,P,S)

V is a set of Non-Terminals

- Defines a finite set of non-terminal symbols or variables
- Represented in Uppercase letters

T is a set of Terminals

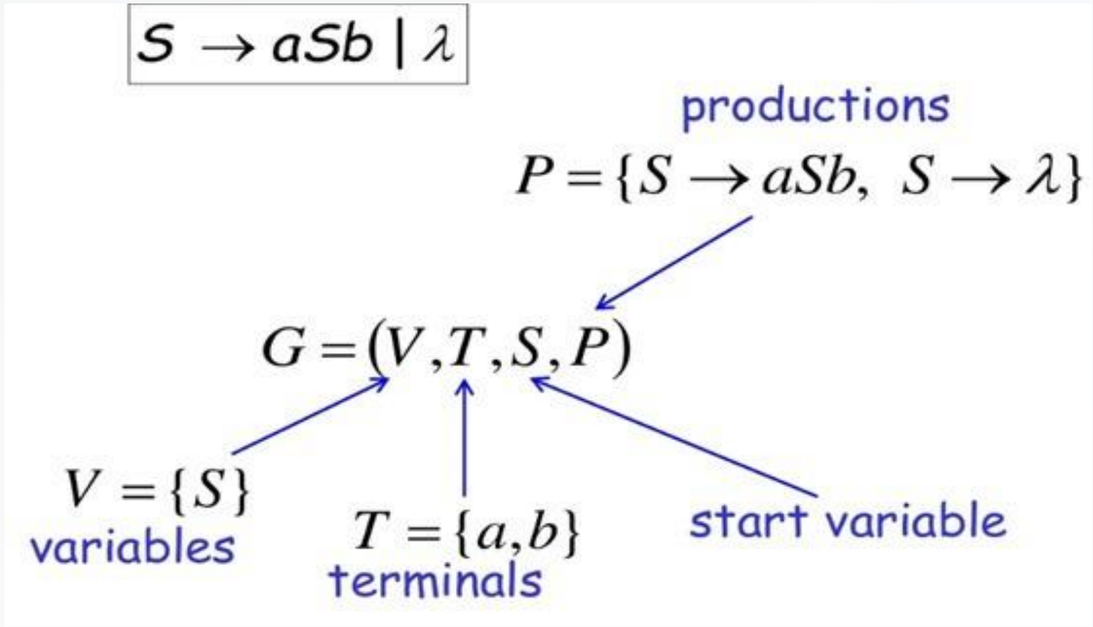
- Used to form language constructs
- Represented in lowercase letters.

P is Production Rule

- Built production rules using non-terminal and terminal symbols

S is Start Symbol

- Derivation should always be with start symbol



CFG-EXAMPLE

S → SA

This production rule states that the non-terminal symbol 'S' can be replaced by the sequence of 'S' followed by 'A'.

S → A

This production rule states that 'S' can also be replaced by 'A' directly.

A → bSe

This production rule defines the non-terminal symbol 'A'. It says that 'A' can be replaced by the sequence of 'b', followed by 'S', followed by 'e'.

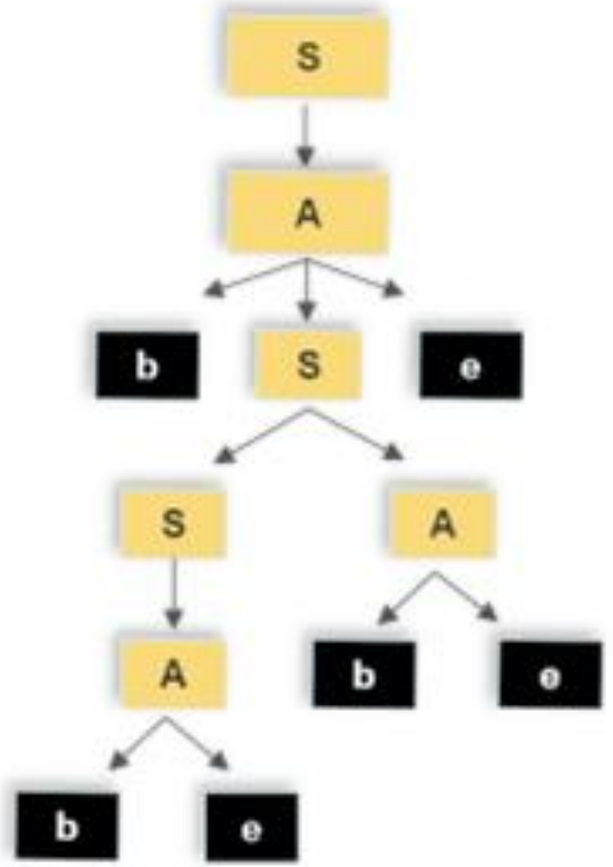
A → be

This production rule states that 'A' can also be replaced by 'be' directly. This provides an alternative way to generate strings that consist of 'be' when 'A' is encountered.

Derivation of bbebee

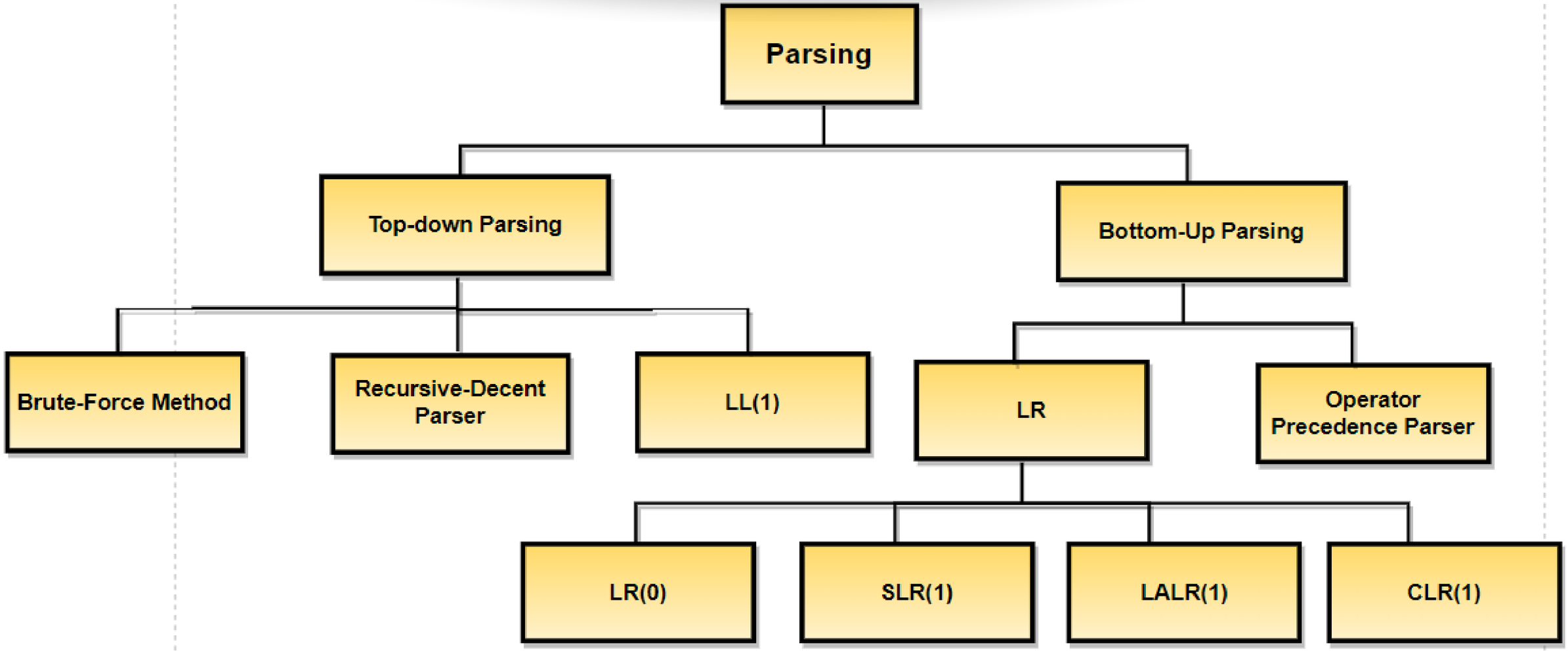
S	S -> A
A	A -> bSe
b S e	S -> SA
b S A e	S -> A
b A A e	A -> be
b b e A e	A -> be
b b e b e e	

Parsing Tree



S,A(uppercase)-Set of Non-Terminals : b,e(lowercase)-Set of Terminals

PARSING ALGORITHMS



LR(0) ALGORITHM

The LR(0) algorithm is used to determine whether a given input string can be generated by a given context-free grammar and, if so, to construct a parse tree for the input string. It works by building a finite automaton that represents the possible states of a parser as it processes the input string. This automaton is used to construct a parsing table that guides the parser through the input string.

Various steps involved in the LR (0) Parsing:

- For the given input string write a context free grammar.
- Check the ambiguity of the grammar.
- Add Augment production in the given grammar.
- Create Canonical collection of LR (0) items.
- Draw a data flow diagram (DFA).
- Construct a LR (1) parsing table.
- Conflict Resolution: The LR(0) algorithm may encounter conflicts, such as shift-reduce or reduce-reduce conflicts, in the parsing table. Conflict resolution rules are used to handle these situations.

Augment Grammer:

It will be generated if we add one more production in the given grammer G. It helps the parser to identify when to stop the parsing and announce the acceptance of the input.

If the given Grammer (G)

S -> AA
A-> aA | b

Augment Grammer (G') is represented by-

S'-> S
S -> AA
A-> aA | b

CANONICAL COLLECTION OF LR(0) ITEMS

LR(0) items is useful to indicate that how much of the input has been scanned up to a given point in the process of parsing. Add Augment Production and insert '•' symbol at the first position for every production in G.

Given grammar:

S → AA
A → aA | b

After Adding:

S` → •S
S → •AA
A → •aA
A → •b

- **I0 State**

Add all productions starting with S in to I0 State because "•" is followed by the non-terminal. So, the I0 State becomes

I0 = S` → •S
S → •AA

Add all productions starting with "A" in modified I0 State because "•" is followed by the non-terminal. So, the I0 State becomes.

I0= S` → •S
S → •AA
A → •aA
A → •b

- **I1= Go to (I0, S) = closure (S` → S•) = S` → S•**

Here, the Production is reduced so close the State.

I1= S` → S•

- **I2= Go to (I0, A) = closure (S → A•A)**

Add all productions starting with A in to I2 State because "•" is followed by the non-terminal. So, the I2 State becomes

I2 =S→A•A
A → •aA
A → •b

Go to (I2,a) = Closure (A → a•A) = (same as I3)
Go to (I2, b) = Closure (A → b•) = (same as I4)

- **I3= Go to (I0,a) = Closure (A → a•A)**

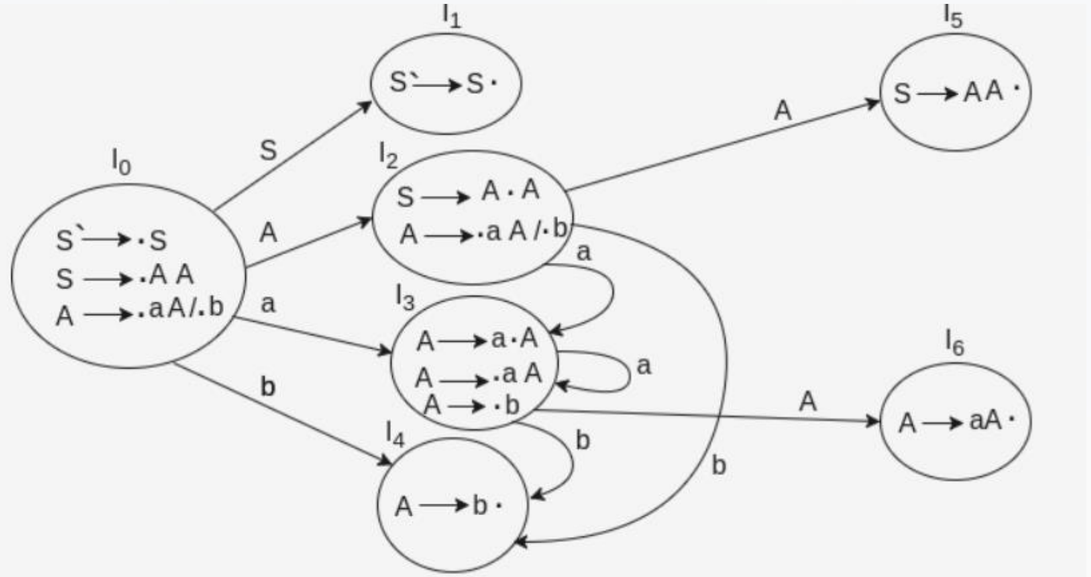
Add productions starting with A in I3.

A → a•A
A → •aA
A → •b

Go to (I3, a) = Closure (A → a•A) = (same as I3)

- **I4= Go to (I0, b) = closure (A → b•) = A → b•**
- **I5= Go to (I2, A) = Closure (S → AA•) = SA → A•**
- **I6= Go to (I3, A) = Closure (A → aA•) = A → aA•**

The DFA contains the 7 states I0 to I6.



LR(0) TABLE

States	Action			Go to	
	a	b	$	A	S
I_0	S3	S4		2	1
I_1			accept		
I_2	S3	S4		5	
I_3	S3	S4		6	
I_4	r3	r3	r3		
I_5	r1	r1	r1		
I_6	r2	r2	r2		

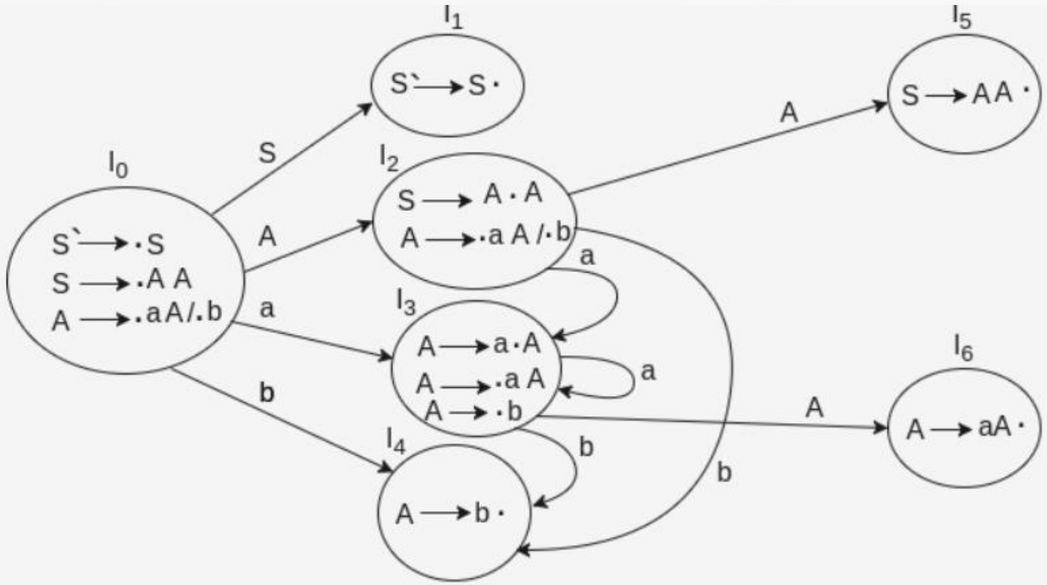
- If a state is going to some other state on a terminal then it correspond to a shift move.
- If a state is going to some other state on a variable then it correspond to go to move.
- If a state contain the final item in the particular row then write the reduce node completely.

Productions are numbered as follows:

- S → AA ... (1)
- A → aA ... (2)
- .A → b ... (3)

- I1 contains the final item which drives(S` → S•), so action {I1, $} = Accept.
- I4 contains the final item which drives A → b• and that production corresponds to the production number 3 so write it as r3 in the entire row.
- I5 contains the final item which drives S → AA• and that production corresponds to the production number 1 so write it as r1 in the entire row.
- I6 contains the final item which drives A → aA• and that production corresponds to the production number 2 so write it as r2 in the entire row.

The DFA constructed with the LR(0) items:



BRUTE-FORCE(BACKTRACKING) PARSER

- Whenever a non-terminal is expanding first time, then go with the first alternative and compare with the i/p string.
- If does not matches, go for the second alternative and compare with i/p string, if does not matches go with the 3rd alternative and continue with each and every alternative.
- If the matching occurs for at least one alternative, then the parsing is successful, otherwise parsing fail.

Context-Free Grammer

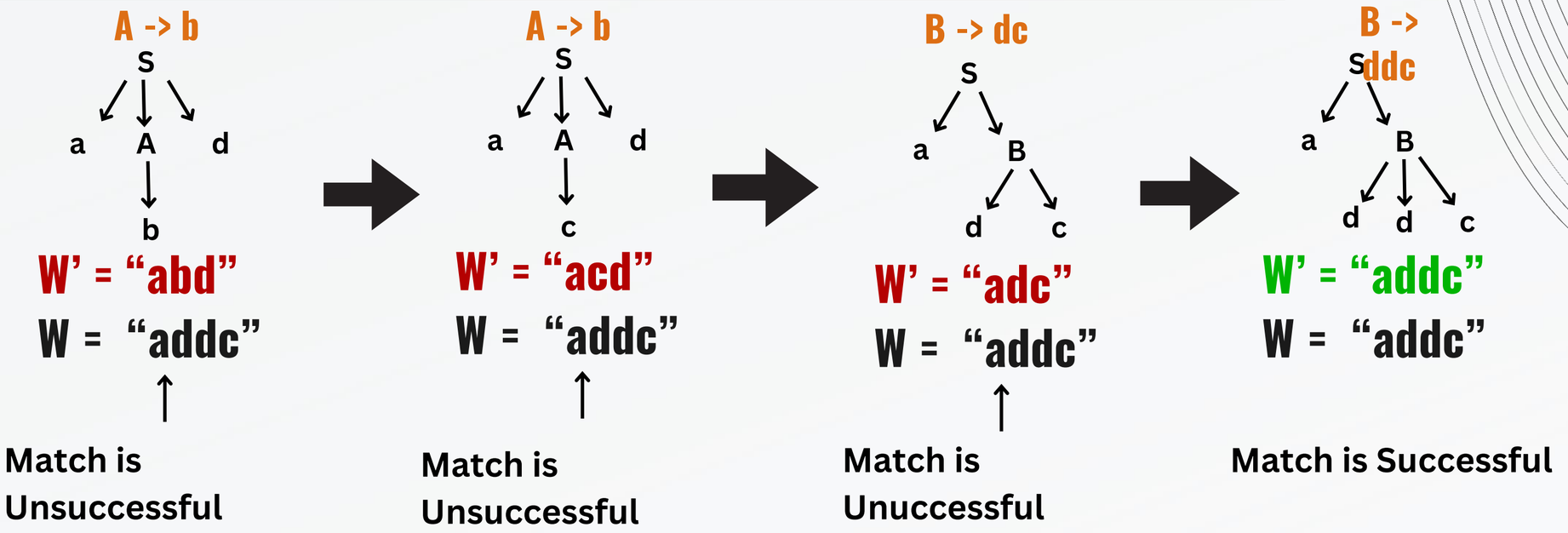
S -> aAd | aB

A -> b | c

B -> dc | ddc

Input String

W = “addc”



Backtracking

- If a parsing function reaches a point where it cannot proceed with any of the available production rules, it backtracks to the previous choice point.
- This means it undoes any partial parsing and tries a different alternative if available.

Error Handling

- If the parser cannot find a valid parse for the input string, it raises a parsing error indicating that the input does not conform to the grammar.

CHALLENGES

Recursive descent parsers, which are based on CFGs, can suffer from backtracking when parsing certain inputs. Backtracking can lead to inefficiency in parsing and can be problematic for error reporting.

BACKTRACKING

CFGs are not well-suited for describing error recovery strategies in parsing. Handling syntax errors and providing meaningful error messages to the programmer can be challenging.

ERROR HANDLING

One of the most significant challenges with CFGs is that they can describe ambiguous grammars. An ambiguous grammar is one in which a single string can have multiple valid parse trees or interpretations

AMBIGUITY

SOME OTHER APPLICATIONS

- **Natural Language Processing (NLP):** Parsing is essential in NLP tasks such as part-of-speech tagging, syntactic parsing, and named entity recognition. It helps understand the grammatical structure of sentences and extract meaningful information from text.
- **Network Protocols:** Parsing algorithms are used in network protocols to decode and encode data packets. This is crucial for communication between devices and systems in computer networks.
- **Data Validation and Transformation:** In data integration and ETL (Extract, Transform, Load) processes, parsing algorithms validate and transform data from one format to another, ensuring data quality and compatibility.

REFERENCES 📖

- https://www.geeksforgeeks.org/problem-on-lr0-parser/t
- https://www.youtube.com/watch?v=ZjQst1xnCuI&ab_channel=Education4u
- https://www.youtube.com/watch?v=PLu8tgUus1k&ab_channel=SuccessGATEway
- https://www.youtube.com/watch?v=5_tfVe7ED3g&ab_channel=NesoAcademy

THANK YOU