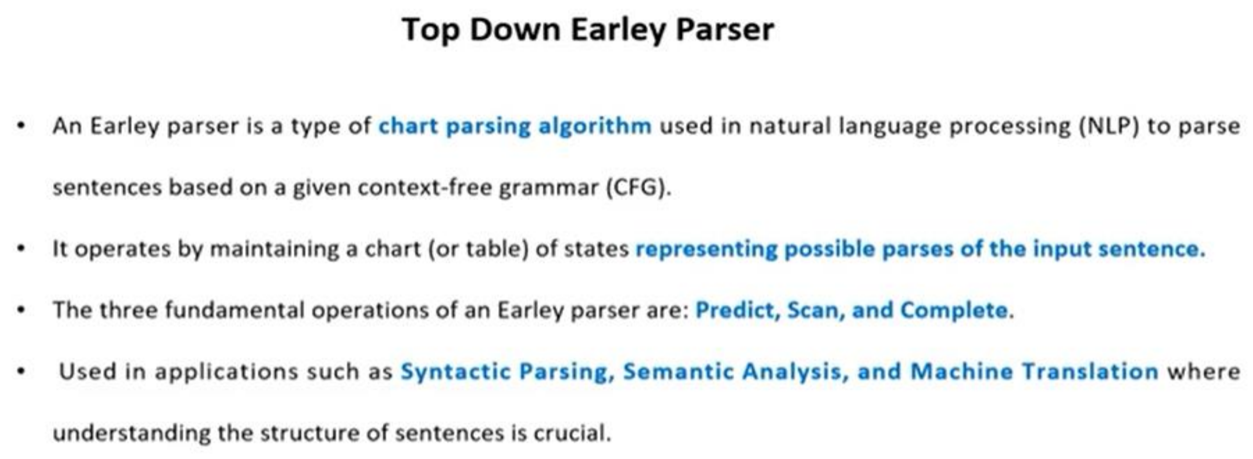
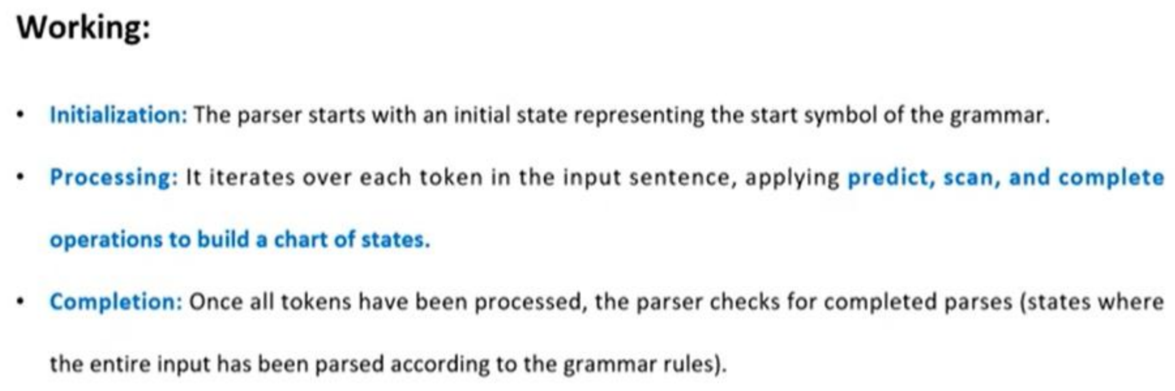
Earley Algorithm (Chart Parsing)

The Earley algorithm is a chart parsing algorithm for context-free grammars (CFGs), particularly useful for parsing ambiguous and complex grammatical structures in natural language processing (NLP). It is designed to parse any CFG and is especially efficient for grammars with a high degree of ambiguity.



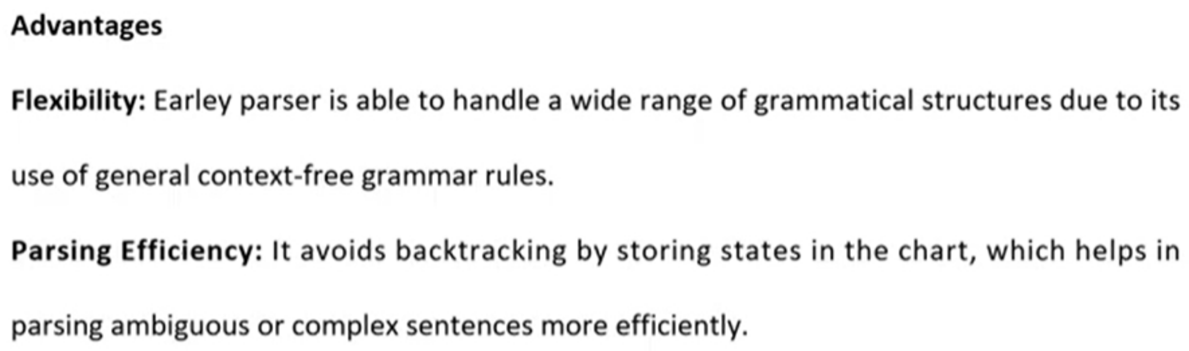
Top Down Earley Parser

- An Earley parser is a type of **chart parsing algorithm** used in natural language processing (NLP) to parse sentences based on a given context-free grammar (CFG).
- It operates by maintaining a chart (or table) of states **representing possible parses of the input sentence.**
- The three fundamental operations of an Earley parser are: **Predict, Scan, and Complete**.
- Used in applications such as **Syntactic Parsing, Semantic Analysis, and Machine Translation** where understanding the structure of sentences is crucial.



Working:

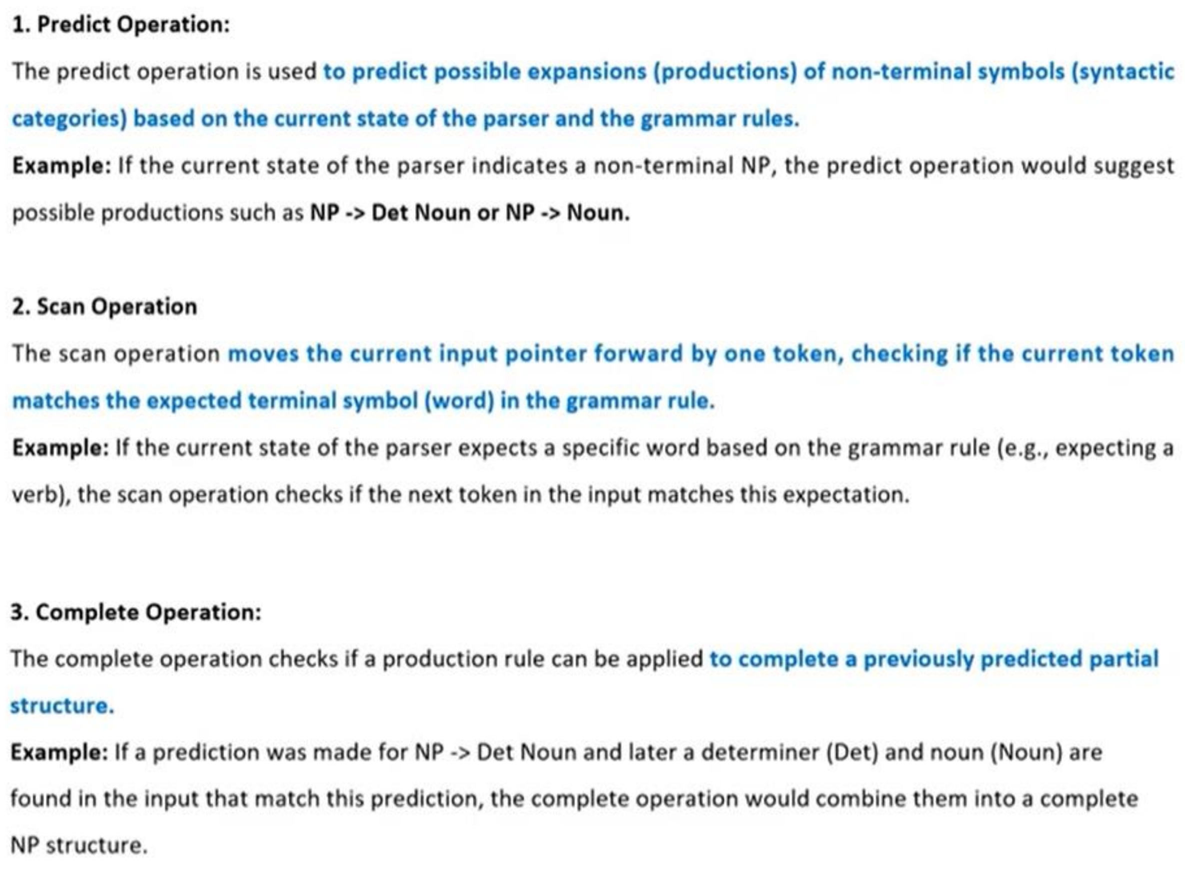
- **Initialization:** The parser starts with an initial state representing the start symbol of the grammar.
- **Processing:** It iterates over each token in the input sentence, applying **predict, scan, and complete operations to build a chart of states.**
- **Completion:** Once all tokens have been processed, the parser checks for completed parses (states where the entire input has been parsed according to the grammar rules).



Advantages

Flexibility: Earley parser is able to handle a wide range of grammatical structures due to its use of general context-free grammar rules.

Parsing Efficiency: It avoids backtracking by storing states in the chart, which helps in parsing ambiguous or complex sentences more efficiently.



1. Predict Operation:

The predict operation is used **to predict possible expansions (productions) of non-terminal symbols (syntactic categories) based on the current state of the parser and the grammar rules.**

Example: If the current state of the parser indicates a non-terminal NP, the predict operation would suggest possible productions such as **NP -> Det Noun or NP -> Noun.**

2. Scan Operation

The scan operation **moves the current input pointer forward by one token, checking if the current token matches the expected terminal symbol (word) in the grammar rule.**

Example: If the current state of the parser expects a specific word based on the grammar rule (e.g., expecting a verb), the scan operation checks if the next token in the input matches this expectation.

3. Complete Operation:

The complete operation checks if a production rule can be applied **to complete a previously predicted partial structure.**

Example: If a prediction was made for NP -> Det Noun and later a determiner (Det) and noun (Noun) are found in the input that match this prediction, the complete operation would combine them into a complete NP structure.

Here is an explanation of the process, step-by-step:

Steps of the Earley Algorithm

The Earley algorithm operates using three primary operations: **prediction**, **scanning**, and **completion**. It builds a chart where each entry, called a state, keeps track of parsing progress. Each state is represented as:

[X -> α • β, i]

Here, X -> α • β indicates a production rule X -> αβ with a dot marking how much of the rule has been parsed, and i is the position in the input string where the state started.

Initialization

1. **Initialize the Chart**: Create an array chart of length n+1, where n is the length of the input string. Each chart[k] will store a list of states.
2. **Start State**: Add the initial state [S' -> • S, 0] to chart[0], where S' is a new start symbol, and S is the original start symbol of the grammar.

Main Loop

For each position k from 0 to n, process all states in chart[k] using the following operations:

1. Prediction

For each state [A -> α • B β, i] in chart[k], if B is a non-terminal:

- Add [B -> • γ, k] to chart[k] for each production B -> γ in the grammar.

2. Scanning

For each state [A -> α • a β, i] in chart[k], if a matches the current input symbol at position k:

- Add [A -> α a • β, i] to chart[k+1].

3. Completion

For each state [B -> γ •, j] in chart[k], if the dot is at the end of the rule (indicating completion of B):

- For each state [A -> α • B β, i] in chart[j], add [A -> α B • β, i] to chart[k].

Termination

After processing all positions k from 0 to n, the algorithm checks if a valid parse has been found by looking for a state [S' -> S •, 0] in chart[n].

Example

Let's parse the sentence "the dog saw a cat" using the following grammar:

S -> NP VP

NP -> Det N

VP -> V NP

Det -> 'the' | 'a'

N -> 'dog' | 'cat'

V -> 'saw'

Initialization:

- chart[0]: [S' -> • S, 0]

Position 0:

1. **Prediction**: [S' -> • S, 0] predicts [S -> • NP VP, 0].
2. [S -> • NP VP, 0] predicts [NP -> • Det N, 0].
3. [NP -> • Det N, 0] predicts [Det -> • 'the', 0] and [Det -> • 'a', 0].

Position 1:

1. **Scanning**: The input symbol is "the", matching [Det -> 'the' •, 0].
2. Add [Det -> 'the' •, 0] to chart[1].
3. **Completion**: Complete [NP -> Det • N, 0] to [NP -> Det N •, 0].

Position 2:

1. **Scanning**: The input symbol is "dog", matching [N -> 'dog' •, 1].
2. Add [N -> 'dog' •, 1] to chart[2].
3. **Completion**: Complete [NP -> Det N •, 0] to [S -> NP • VP, 0].

Position 3:

1. **Prediction**: [S -> NP • VP, 0] predicts [VP -> • V NP, 2].
2. [VP -> • V NP, 2] predicts [V -> • 'saw', 2].

Position 4:

1. **Scanning**: The input symbol is "saw", matching [V -> 'saw' •, 2].
2. Add [V -> 'saw' •, 2] to chart[3].
3. **Completion**: Complete [VP -> V • NP, 2] to [S -> NP VP •, 0].

Position 5:

1. **Prediction**: [VP -> V • NP, 2] predicts [NP -> • Det N, 4].
2. [NP -> • Det N, 4] predicts [Det -> • 'the', 4] and [Det -> • 'a', 4].

Position 6:

1. **Scanning**: The input symbol is "a", matching [Det -> 'a' •, 4].
2. Add [Det -> 'a' •, 4] to chart[5].
3. **Completion**: Complete [NP -> Det • N, 4] to [NP -> Det N •, 4].

Position 7:

1. **Scanning**: The input symbol is "cat", matching [N -> 'cat' •, 5].
2. Add [N -> 'cat' •, 5] to chart[6].
3. **Completion**: Complete [NP -> Det N •, 4] to [VP -> V NP •, 2].
4. Complete [S -> NP VP •, 0] to [S' -> S •, 0].

Final Parse

The presence of [S' -> S •, 0] in chart[7] indicates a successful parse, confirming the sentence "the dog saw a cat" is valid according to the grammar.

Parse Tree

```
        S
       / \
     NP   VP
    / \   / \
  Det  N V   NP
  |    | |   / \
 the  dog saw Det N
              |   |
              a   cat
```

Using the Earley algorithm, parse the sentence "the cat chased the dog" given the following CFG:

S -> NP VP

NP -> Det N | N

VP -> V NP

Det -> 'the' | ‘a’

N -> 'cat' | 'dog'

V -> 'chased' | ‘chases’ | ‘sees’

Earley Algorithm Steps:

1. **Initialization (S0)**: Add S -> . NP VP [0]
2. **Predict NP (S0)**: Add NP -> . Det N [0]
3. **Predict Det (S0)**: Add Det -> . 'the' [0]
4. **Scan 'the' (S1)**: Det -> 'the' . [0]
5. **Complete Det (S1)**: NP -> Det . N [0]
6. **Predict N (S1)**: Add N -> . 'cat' [1], N -> . 'dog' [1]
7. **Scan 'cat' (S2)**: N -> 'cat' . [1]
8. **Complete N (S2)**: NP -> Det N . [0]
9. **Complete NP (S2)**: S -> NP . VP [0]
10. **Predict VP (S2)**: Add VP -> . V NP [2]
11. **Predict V (S2)**: Add V -> . 'chased' [2]
12. **Scan 'chased' (S3)**: V -> 'chased' . [2]
13. **Complete V (S3)**: VP -> V . NP [2]
14. **Predict NP (S3)**: Add NP -> . Det N [3]
15. **Predict Det (S3)**: Add Det -> . 'the' [3]
16. **Scan 'the' (S4)**: Det -> 'the' . [3]
17. **Complete Det (S4)**: NP -> Det . N [3]
18. **Predict N (S4)**: Add N -> . 'cat' [4], N -> . 'dog' [4]
19. **Scan 'dog' (S5)**: N -> 'dog' . [4]

20. **Complete N (S5)**: NP -> Det N . [3]

21. **Complete NP (S5)**: VP -> V NP . [2]

22. **Complete VP (S5)**: S -> NP VP . [0]

```
         S
        / \
      NP  VP
     / \   / \
   Det  N V   NP
   |    | |   / \
  the  cat chased Det  N
              |    |
             the   dog
```