FIRST and FOLLOW Sets in Compiler Design

Course: Principles of Compiler Design (SEng4031)

Student Name: Tsadkan Kelemework

Student ID: BDU1508032

Instructor: Wondimu B.

University: Bahir Dar University

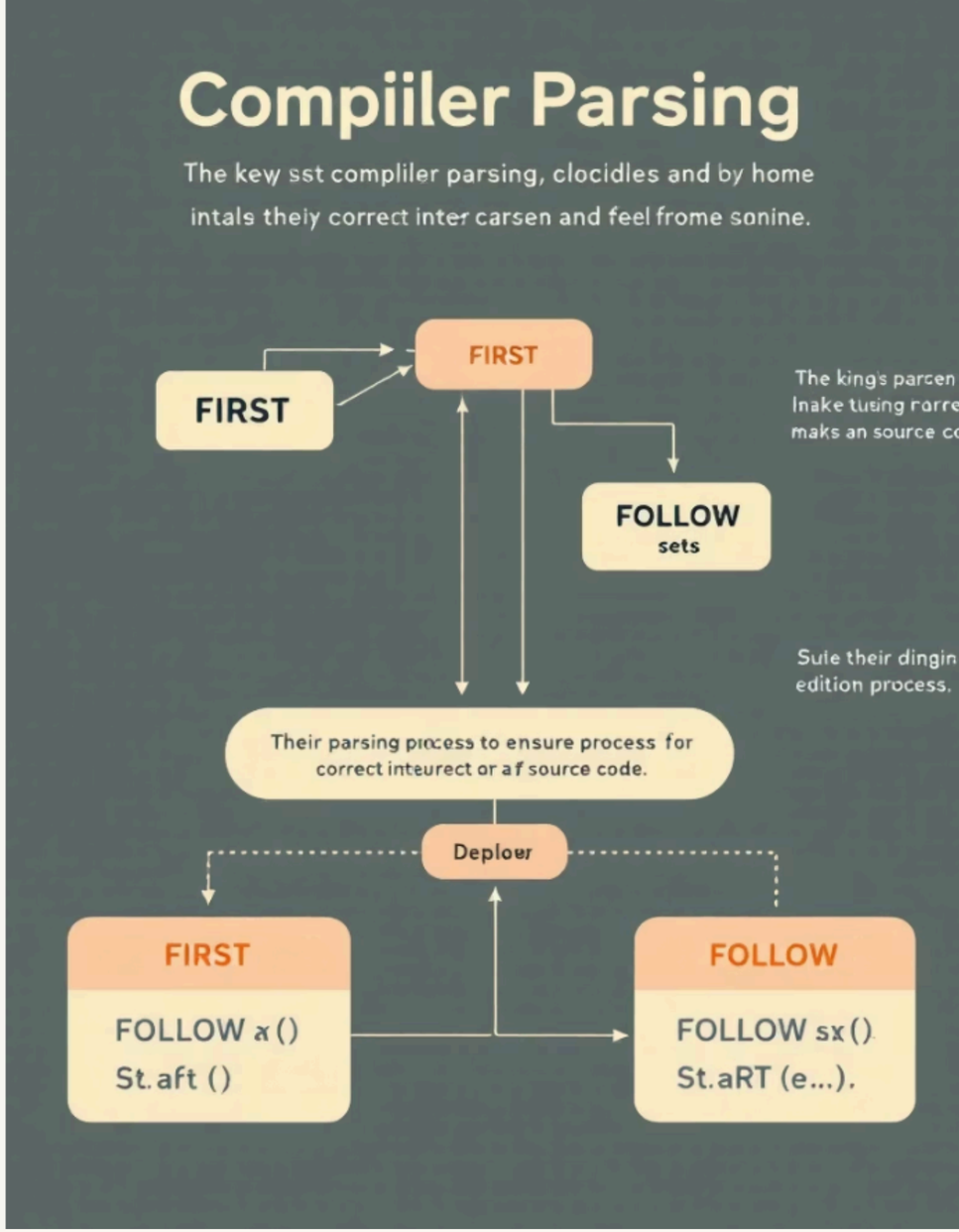
Introduction to Compiler Parsing

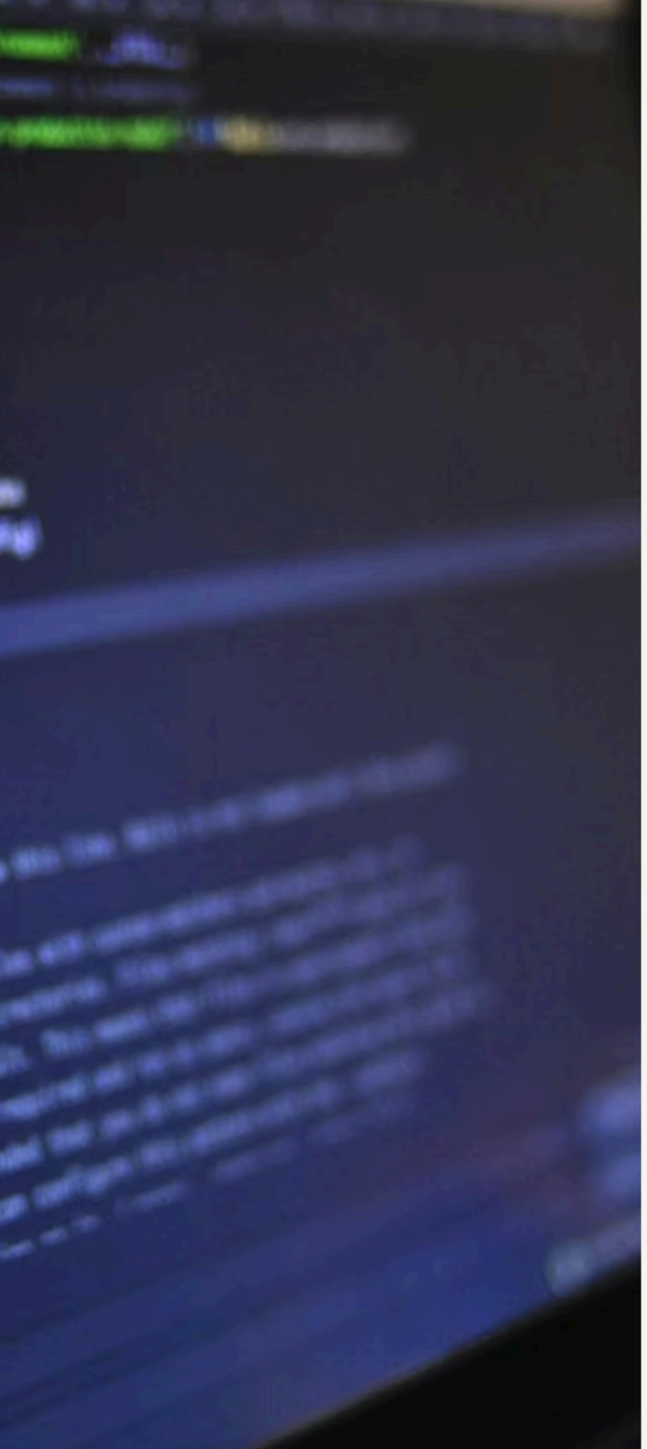
Compiler Design explores the intricate process of translating high-level programming languages into machine-understandable code. A crucial stage in this process is **syntax analysis**, also known as parsing.

Parsing validates that a program adheres to the grammatical rules of its language. To construct efficient parsers, particularly top-down parsers, two foundational concepts are indispensable:

- The FIRST set
- The FOLLOW set

These sets guide the compiler in selecting the appropriate grammar rule during the parsing phase, ensuring correct and unambiguous interpretation of the source code.





Defining FIRST Set

The FIRST set of a grammar symbol comprises all terminal symbols that can initiate strings derived from that symbol.

Handling Empty Strings

If a symbol can derive the empty string (ε), then ε is explicitly included in its FIRST set.

In simple terms, the **FIRST** set answers the question:

> "What symbols can appear first when this specific grammar rule begins its derivation?"

Rules for Computing FIRST Sets

Terminal Symbols

If the symbol (**X**) is a terminal, then **FIRST(X) = { X }**.

Empty String Derivation

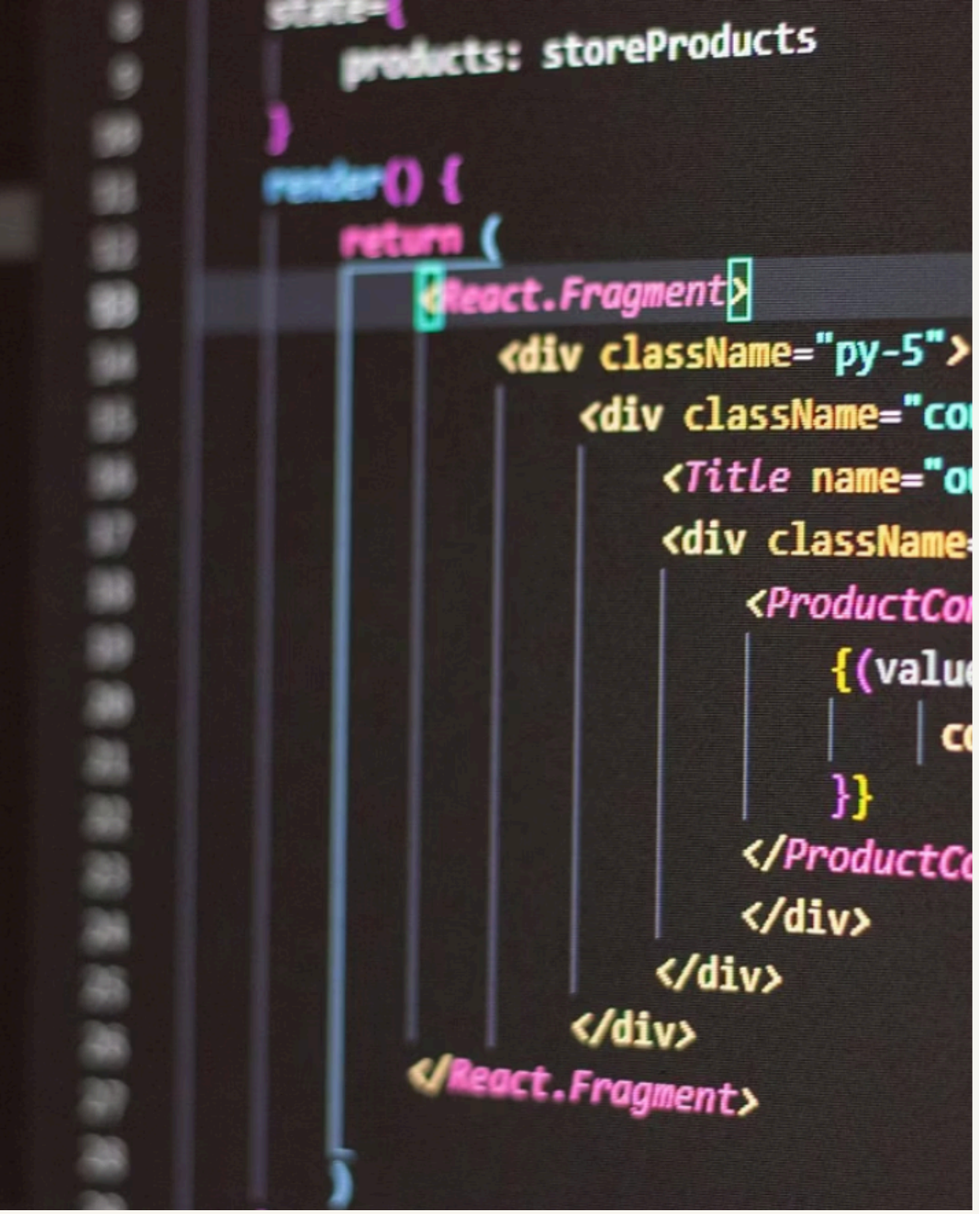
If the symbol (**X**) can derive the empty string (ε), then **FIRST(X)** must include ε.

Non-Terminal Productions

For a production $X \rightarrow Y_1\ Y_2\ \dots\ Y_n$:

- Add all symbols from **FIRST(Y_1)** (excluding ε) to **FIRST(X)**.
- If **FIRST(Y_1)** contains ε, then repeat the process for Y_2, and so on, until a Y_i whose **FIRST(Y_i)** does not contain ε, or all **Y** symbols have been processed.

These comprehensive rules guarantee that all potential starting symbols are accurately identified for each grammar production.



FIRST Set Example: A Simple Expression Grammar

Example Grammar:

```
E  → T E'
E' → + T E' | ε
T  → F
F  → ( E ) | id
```

Computed FIRST Sets:

- **FIRST(E) = { (, id }**
- **FIRST(E′) = { +, ε }**
- **FIRST(T) = { (, id }**
- **FIRST(F) = { (, id }**

This analysis reveals that an expression in this grammar can begin with either an opening parenthesis **(** or an identifier **id**. This information is crucial for predictive parsing.

Definition of FOLLOW Set

The FOLLOW set of a non-terminal symbol consists of all terminal symbols that can appear immediately to its right in any valid sentential form.

Non-Terminal Specific

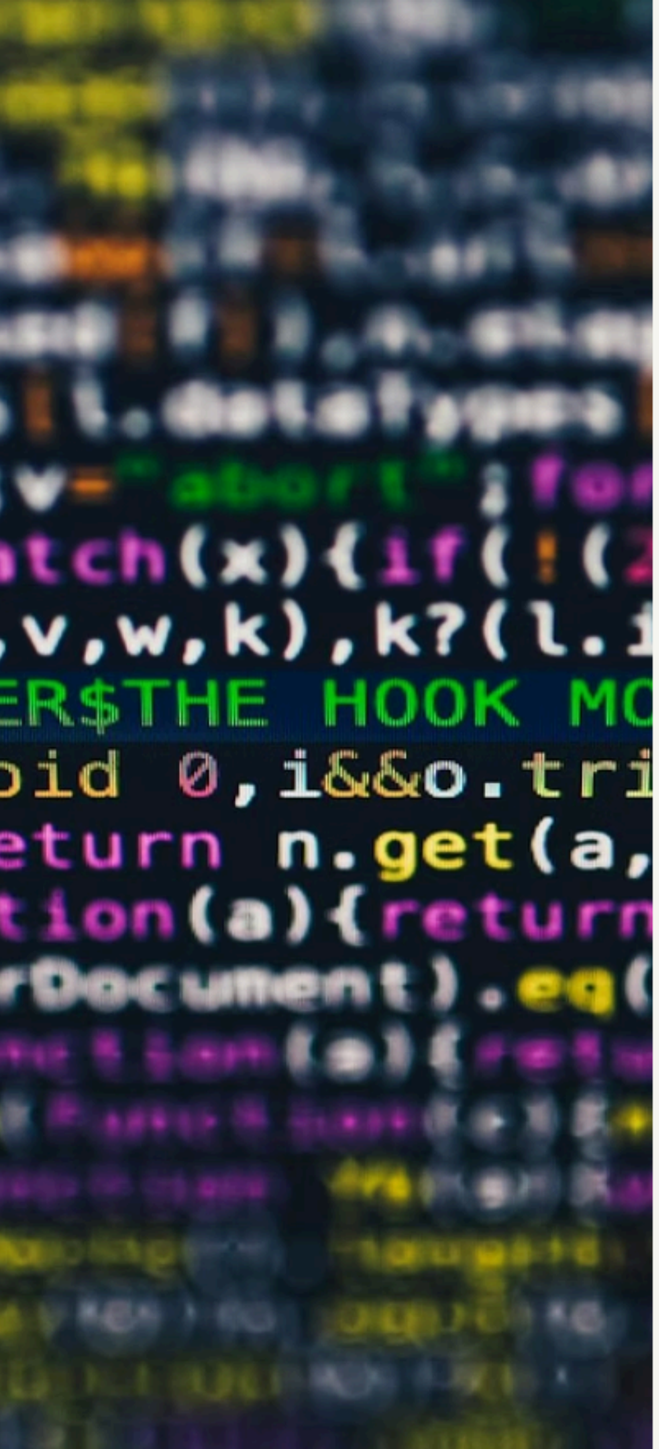
FOLLOW sets are exclusively defined and computed for non-terminal symbols within the grammar.

End of Input Marker

The special terminal symbol **$** is used to denote the unequivocal end of the input string.

In essence, the **FOLLOW** set clarifies:

> "Which symbols are permitted to immediately succeed this rule once its derivation has concluded?"



Rules for Computing FOLLOW Sets

01

Start Symbol Initialization

Always add the end-of-input marker **$** to the FOLLOW set of the grammar's start symbol.

02

Production Rule: A → α B β

If there is a production where a non-terminal **B** is followed by a sequence of symbols **β**, add all terminals from **FIRST(β)** (excluding ε) to **FOLLOW(B)**.

03

Production Rule: A → α B or FIRST(β) contains ε

If **B** is the last symbol in a production, or if **FIRST(β)** (the symbols immediately following **B**) contains ε, then all symbols in **FOLLOW(A)** must be added to **FOLLOW(B)**.

These rules are fundamental for enabling accurate symbol prediction, which is a cornerstone of efficient predictive parsing algorithms.

FOLLOW Set Example: Understanding Context

Grammar in Focus:

```
E  → T E'
E' → + T E' | ε
T  → F
F  → ( E ) | id
```

Computed FOLLOW Sets:

- **FOLLOW(E) = {), $ }**
- **FOLLOW(E′) = {), $ }**
- **FOLLOW(T) = { +,), $ }**
- **FOLLOW(F) = { +,), $ }**

For instance, **FOLLOW(E)** indicates that an expression can be followed by a closing parenthesis or the end of the input. This granular insight allows the parser to precisely determine when a specific grammar rule or non-terminal derivation should terminate, ensuring the correct structural interpretation of the code.

Crucial Role of FIRST and FOLLOW Sets

LL(1) Parsing Tables

They are indispensable for constructing accurate LL(1) parsing tables, forming the backbone of top-down parsers.

Preventing Conflicts

These sets actively help in detecting and resolving potential parsing conflicts, ensuring unambiguous grammar interpretation.

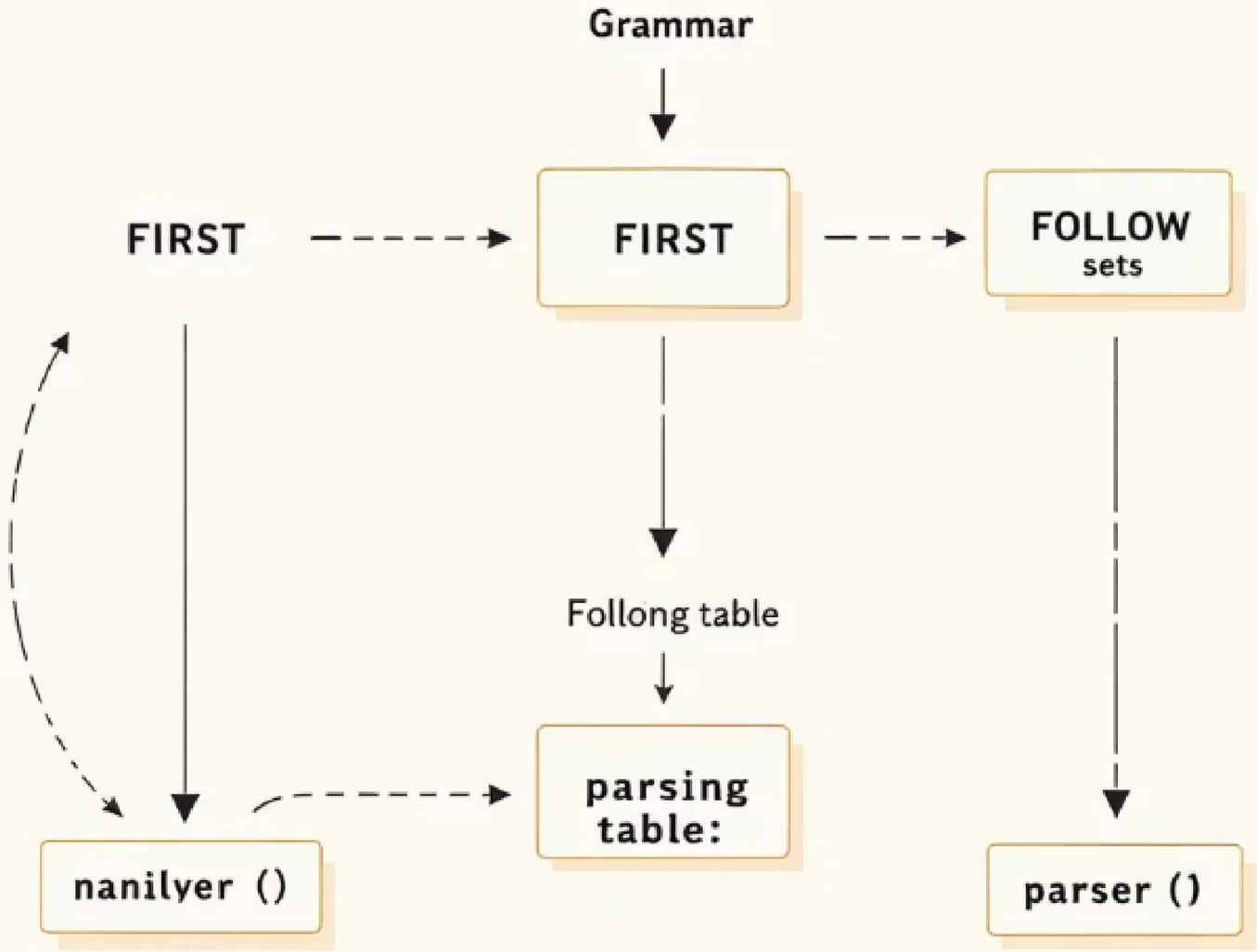
Enabling Predictive Parsing

They facilitate predictive parsing, allowing the compiler to make decisions without the need for costly backtracking.

Enhancing Efficiency

By streamlining the parsing process, FIRST and FOLLOW sets significantly improve the overall efficiency and accuracy of the compiler.

The proper functioning of top-down parsing methodologies is fundamentally dependent on the correct computation and application of FIRST and FOLLOW sets.


Grammar
FIRST
FIRST
FOLLOW
sets
Follong table
parsing
table:
nanilyer ()
parser ()

Real-World Applications in Compiler Technology

The theoretical underpinnings of FIRST and FOLLOW sets find practical application in various critical aspects of compiler development:

- **LL(1) Parsers:** Directly used in the construction and operation of LL(1) parsers, which are widely employed for their simplicity and efficiency.
- **Syntax Analysis Phase:** They are a core component of the syntax analysis phase within virtually every compiler, ensuring the grammatical correctness of source code.
- **Compiler Tools:** These concepts form the conceptual foundation for powerful compiler-construction tools, such as:
  - **ANTLR (ANother Tool for Language Recognition):** A popular parser generator that leverages these principles to build parsers, tree walkers, and translators from grammar specifications.
  - **YACC (Yet Another Compiler Compiler):** While primarily an LALR(1) parser generator, its underlying principles of predicting grammar rules are conceptually tied to FIRST and FOLLOW sets, particularly in handling ambiguities and lookahead.

Ultimately, these sets empower compilers to automatically and intelligently decide which grammar rule to apply next, enabling robust and error-resistant language processing.

Conclusion

- **FIRST set** identifies possible starting symbols
- **FOLLOW set** identifies possible next symbols
- Both are crucial for:
  - Grammar analysis
  - Parser construction
  - Reliable compiler behavior

Understanding FIRST and FOLLOW sets is fundamental to mastering **Compiler Design**.

References

1. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D.
   Compilers: Principles, Techniques, and Tools (Dragon Book)
2. K. C. Louden
   Compiler Construction: Principles and Practice
3. GeeksforGeeks – FIRST and FOLLOW Sets in Compiler Design

Thank You