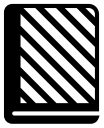
FIRST and FOLLOW Sets

LAB 7 ACTIVITY

🎯Objective

- Input grammar production rules
- Calculate FIRST and FOLLOW sets for each non-terminal
- Display results in a structured way

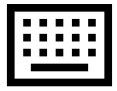


Key Concepts

- FIRST set: terminals that appear at the start of derivations
- FOLLOW set: terminals that appear immediately after a non-terminal in some derivation
- Epsilon (~): Represents an empty production

🔧Program Structure

- Main Function: Handles input, calls processing functions
- ComputeFirst(): Recursively calculates FIRST sets
- ComputeFollow(): Calculates FOLLOW sets by traversing production rules
- Uses Dictionary and HashSet collections for storage



Input Format

- Production rules in the form: S->A B
- Use | to separate alternatives: A->a | ~
- Use '~' to represent epsilon (empty string)
- Type 'end' when finished

Code: Main & Input Handling

```
Console.WriteLine("Enter production rules (S->A B):");
while (true) {
    input = Console.ReadLine();
    if (input == "end") break;
    var parts = input.Split("->");
    var lhs = parts[0].Trim();
    var rhs = parts[1].Trim().Split('|');
    foreach (var alt in rhs)
        productionRules[lhs].Add(alt.Split());
}
```

Code: ComputeFirst Function

```
static HashSet<string> ComputeFirst(string symbol) {
    if (!productionRules.ContainsKey(symbol))
        return new HashSet<string>{ symbol };
    foreach (var prod in productionRules[symbol]) {
        foreach (var sym in prod) {
            var first = ComputeFirst(sym);
            result.UnionWith(first);
            if (!first.Contains("~")) break;
        }
    }
}
```

Code: ComputeFollow Function

```
static void ComputeFollow(string nonTerminal) {
   foreach (var head in productionRules.Keys) {
      foreach (var prod in productionRules[head]) {
         for (int i = 0; i < prod.Length; i++) {
            if (prod[i] == nonTerminal) {
               if (i+1 < prod.Length)
                  followSet.UnionWith(First(prod[i+1]));
               else if (head != nonTerminal)
                  followSet.UnionWith(followSets[head]);
            }
         }
      }
   }
}
```

Sample Output

- Input: S->A B | A->a | ~ | B->b
- FIRST(S) = { a, b, ~ }
- FOLLOW(A) = { b }
- FOLLOW(B) = { $ }

✓ Conclusion

- Efficiently calculates FIRST and FOLLOW sets
- Supports error checking and recursion
- Helps understand grammar parsing in compilers