Compiler Construction

Syntactic Analysis

Syntactic Analysis

Reminders from the THL lectures!

How to check that a text (stream of token) is valid according to a given grammar?
$\Rightarrow$ We need something more powerful than finite automata!

Languages

Language

A **Language** is a set of strings,
each string is a sequence of **symbols** from an **alphabet**

Remark

In our context, the alphabet is the set of token types returned by the lexical analyzer.

Context-free grammars (1/2)

Regular expressions

Regular expressions are not enough to represent programming languages.

Context-free Grammar

A **context-free grammar** is a set of recursive rules used to generate patterns of symbols.

Context-free grammars (2/2)

Production rules

A grammar has production rules of the form $symbol \rightarrow symbol \;\; symbol \;\; symbol$
Symbols can be:

- **terminal**, meaning that this is a token from the alphabet.
- **non-terminal**, meaning that it appears on the left-hand side of some production.

Example

```
E -> E + E
E -> id
```

Derivations

To detect if a sentence is in the expected language we can perform **derivations**:

1. Start with a symbol
2. Apply productions rules (*Replace any non terminal by one of its right-hand side*)
3. Repeat until no more replacement

Derivations

To detect if a sentence is in the expected language we can perform **derivations**:

1. Start with a symbol
2. Apply productions rules (*Replace any non terminal by one of its right-hand side*)
3. Repeat until no more replacement

There are many derivations:

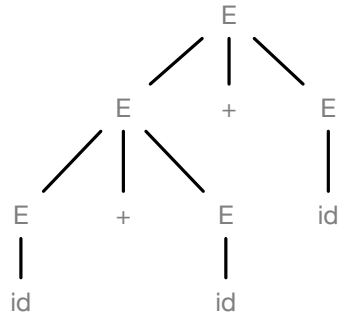
- the **leftmost** non-terminal symbol is always expanded
- the **rightmost** non-terminal symbol is always expanded

Parse tree

Parse tree

A **parse tree** is made by connecting each symbol in a derivation to the one from which it was derived

Example input: id + id + id
(One possible) derivation:
E
E + E
E + id
E + E + id
E + id + id
id + id + id

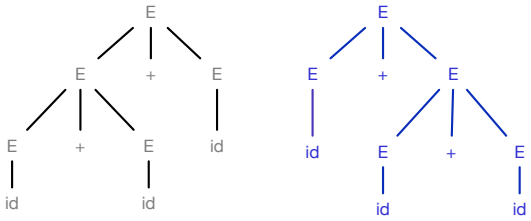


Ambiguous Grammar

Ambiguous Grammar

A grammar is **ambiguous** if we can derive two different parse tree for a sentence

Example: id + id + id



Predictive Parsing – Unambiguous Grammar

A predictive parser is a recursive descent parser which is able to predict which production rule is used to replace the input string.
$\Rightarrow$ each sub-expression must provide enough information to choose a production rule.

Predictive Parsing – Unambiguous Grammar

A predictive parser is a recursive descent parser which is able to predict which production rule is used to replace the input string.
$\Rightarrow$ each sub-expression must provide enough information to choose a production rule.

- LL(k): Left-to-right, Leftmost derivation
- LR(k): Left-to-right, Rightmost derivation

Summary