3. Syntax Analysis

Andrea Polini, Luca Tesei

Formal Languages and Compilers
MSc in Computer Science
University of Camerino

ToC

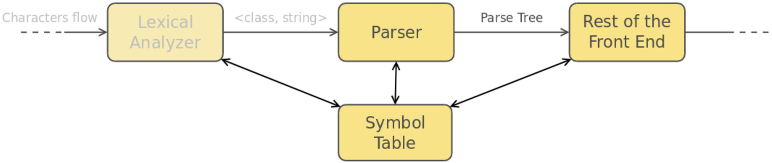
Syntax analysis

Parsing

Parsing is the activity of taking a string of terminals and figuring out how to derive it from the start symbol of a grammar. If a derivation cannot be obtained then syntax errors must be reported within the string.

The Parser

The parser obtains a sequence of tokens and verifies that the sequence can be correctly generated by a given grammar of the source language. For well-formed programs the parser will generate a parse tree that will be passed to the next compiler phase.



Parse Tree

Parse tree

A parse tree shows how the start symbol of a grammar derives the string in the language. If $A \to XYZ$ is a production applied in a derivation, the parse tree will have an interior node labeled with A with three children labeled X, Y, Z from left to right:

- the root is always labeled with the start symbols
- leaves are labeled with terminals or ϵ
- interior nodes are labeled with non-terminal symbols
- parent-children relations among nodes depend from the rules defined by the grammar

Parsing Example

Expressions grammar I

$E \rightarrow E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid id$

Find the sequence or productions for the string "$id + id * id$" and derive the corresponding parse tree

Expressions grammar II

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T/F \mid F$

$F \rightarrow (E) \mid id$

Parsing Example

Expressions grammar I

$E \rightarrow E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid id$

Find the sequence or productions for the string "$id + id * id$" and derive the corresponding parse tree

Expressions grammar II

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T/F \mid F$

$F \rightarrow (E) \mid id$

Type of parsers

Three general type of parsers:

- universal (any kind of grammar)
- top-down
- bottom-up

ToC

Chomsky Hierarchy

A hierarchy of grammars can be defined imposing constraints on the structure of the productions in set $\mathcal{P}$ ($\alpha, \beta, \gamma \in \mathcal{V}^*$, $a \in \mathcal{V}_T$, $A, B \in \mathcal{V}_N$):

T0. Unrestricted Grammars:
- Production Schema: *no constraints*
- Recognizing Automaton: Turing Machines

T1. Context Sensitive Grammars:
- Production Schema: $\alpha A \beta \rightarrow \alpha \gamma \beta$
- Recognizing Automaton: Linear Bound Automaton (LBA)

T2. **Context-Free Grammars**:
- Production Schema: $A \rightarrow \gamma$
- Recognizing Automaton: Non-deterministic Push-down Automaton

T3. Regular Grammars:
- Production Schema: $A \rightarrow a$ or $A \rightarrow aB$
- Recognizing Automaton: Finite State Automaton

Grammar Definition

Context Free Grammar

A Context Free Grammar is a tuple $\mathcal{G} = \langle \mathcal{V_T}, \mathcal{V_N}, \mathcal{S}, \mathcal{P} \rangle$ where:

- $\mathcal{V_T}$ is a finite non-empty set of terminal symbols (alphabet)
- $\mathcal{V_N}$ is a finite non-empty set of non-terminal symbols s.t. $\mathcal{V_N} \cap \mathcal{V_T} = \varnothing$
- $\mathcal{S}$ is the start symbol of the grammar s.t. $\mathcal{S} \in \mathcal{V_N}$
- $\mathcal{P}$ is a finite non-empty set of productions s.t. $\mathcal{P} \subseteq \mathcal{V_N} \times \mathcal{V}^*$ where $\mathcal{V}^* = \mathcal{V_T} \cup \mathcal{V_N}$

Push-down Automata

Definition

A Push-down Automaton is a tuple $\langle \Sigma, \Gamma, \mathcal{Z}_0, \mathcal{S}, s_0, \mathcal{F}, \delta \rangle$ where:

- Σ defines the input alphabet
- Γ defines the alphabet for the stack
- $\mathcal{Z}_0 \in \Gamma$ is the symbol used to represent the empty stack
- $\mathcal{S}$ represents the set of states
- $s_0 \in \mathcal{S}$ is the initial state of the automaton
- $\mathcal{F} \subseteq \mathcal{S}$ is the set of final states
- $\delta : \mathcal{S} \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \ldots$ represents the transition function

Deterministic vs. Non-Deterministic

Push-down automata can be defined according to a deterministic strategy or a non-deterministic one. In the first case the transition function returns elements in the set $\mathcal{S} \times \Gamma^*$, in the second case the returned element belongs to the set $\mathscr{P}(\mathcal{S} \times \Gamma^*)$

Push-down Automata - How do they proceed?

Intuition

- The automaton starts with an empty stack and a string to read
- On the base of its status (state, symbol at the top of the stack), and of the character at the begining of the input string it changes its status consuming the character from the input string.
- The status change consists in the insertion of one or more symbol in the stack after having removed the one at the top, and in the transition to another internal state
- the string is accepted when all the symbols in the input stream have been considered and the automaton reach a status in which the state is final or the stack is empty

Push-down Automata

Configuration

Given a Push-dow Automaton $\mathcal{A} = \langle \Sigma, \Gamma, \mathcal{Z}_0, \mathcal{S}, s_0, \mathcal{F}, \delta \rangle$ a configuration is given by the tuple $\langle s, x, \gamma \rangle$ where:

- $s \in \mathcal{S}, x \in \Sigma^*, \gamma \in \Gamma^*$

The configuration of an automaton represent its global state and contains the information to know its future states.

Transition

Given $\mathcal{A} = \langle \Sigma, \Gamma, \mathcal{Z}_0, \mathcal{S}, s_0, \mathcal{F}, \delta \rangle$ and two configurations $\chi = \langle s, x, \gamma \rangle$ and $\chi' = \langle s', x', \gamma' \rangle$ it can happen that the automaton passes from the first configuration to the second ($\chi \vdash_A \chi'$) iff:

- $\exists a \in \Sigma . x = ax'$
- $\exists Z \in \Gamma, \eta, \sigma \in \Gamma^* . \gamma = Z\eta \wedge \gamma' = \sigma\eta$
- $\delta(s, a, Z) = (s', \sigma)$

Push-down Automata

Acceptance by empty stack

Given $\mathcal{A} = \langle \Sigma, \Gamma, \mathcal{Z}_0, \mathcal{S}, s_0, \mathcal{F}, \delta \rangle$ a configuration $\chi = \langle s, x, \gamma \rangle$ accepts a string iff $x = \gamma = \epsilon$

Acceptance by final state

Given $\mathcal{A} = \langle \Sigma, \Gamma, \mathcal{Z}_0, \mathcal{S}, s_0, \mathcal{F}, \delta \rangle$ a configuration $\chi = \langle s, x, \gamma \rangle$ accepts a string iff $x = \epsilon$ and $s \in \mathcal{F}$

Push-down Automata - Exercise

- Define a push-down automaton that accept the language $\mathcal{L} = \{a^n b^n | n \in \mathbb{N}^+\}$
- Define a push-down automaton that accept the language $\mathcal{L} = \{w\overline{w} | w \in \{a, b\}^+\}$
- Define a push-down automaton that accept the language $\mathcal{L} = \{a^n b^m c^{2n} | n \in \mathbb{N}^+ \wedge m \in \mathbb{N}\}$

Push-down Automata - Exercise

- Define a push-down automaton that accept the language $\mathcal{L} = \{a^n b^n | n \in \mathbb{N}^+\}$
- Define a push-down automaton that accept the language $\mathcal{L} = \{w\overline{w} | w \in \{a, b\}^+\}$
- Define a push-down automaton that accept the language $\mathcal{L} = \{a^n b^m c^{2n} | n \in \mathbb{N}^+ \wedge m \in \mathbb{N}\}$

Push-down Automata - Exercise

- Define a push-down automaton that accept the language $\mathcal{L} = \{a^n b^n | n \in \mathbb{N}^+\}$
- Define a push-down automaton that accept the language $\mathcal{L} = \{w\overline{w} | w \in \{a, b\}^+\}$
- Define a push-down automaton that accept the language $\mathcal{L} = \{a^n b^m c^{2n} | n \in \mathbb{N}^+ \wedge m \in \mathbb{N}\}$

Derivations

Derivation

The construction of a parse tree can be made precise by taking a derivational view, in which production are considered as rewriting rules.

A sentence belongs to a language if there is a derivation from the initial symbol to the sentence.
e.g. $E \rightarrow E + E|E * E| - E|(E)|\mathbf{id}$

Kind of derivations

Each sentence can be generated according to two different strategies leftmost and rightmost. Parsers generally return one of this two derivations.

Derivations

Derivation

The construction of a parse tree can be made precise by taking a derivational view, in which production are considered as rewriting rules.

A sentence belongs to a language if there is a derivation from the initial symbol to the sentence.
e.g. $E \rightarrow E + E | E * E | - E | (E) | \textbf{id}$

Kind of derivations

Each sentence can be generated according to two different strategies leftmost and rightmost. Parsers generally return one of this two derivations.

Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguos. An ambiguous grammar has more then one left-most derivation or more than one rightmost derivation for the same sentence.

Ambiguity and Precedence of Operators

Using the simplest grammar for expressions let's derive again the parse tree for:

$$\textbf{id} + \textbf{id} * \textbf{id}$$

Now consider the following grammar:

$E \rightarrow E + T | E - T | T$

$T \rightarrow T * F | T / F | F$

$F \rightarrow (E) | \textbf{id}$

Use of ambiguos grammar

In some case it can be convenient to use ambiguous grammar, but then it is necessary to define precise disambiguating rules

Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguos. An ambiguous grammar has more then one left-most derivation or more than one rightmost derivation for the same sentence.

Ambiguity and Precedence of Operators

Using the simplest grammar for expressions let's derive again the parse tree for:

$$\mathbf{id} + \mathbf{id} * \mathbf{id}$$

Now consider the following grammar:

$E \rightarrow E + T | E - T | T$
$T \rightarrow T * F | T / F | F$
$F \rightarrow (E) | \mathbf{id}$

Use of ambiguos grammar

In some case it can be convenient to use ambiguous grammar, but then it is necessary to define precise disambiguating rules

Ambiguity

A grammar that produces more than one parse tree for some sentence is said to be ambiguos. An ambiguous grammar has more then one left-most derivation or more than one rightmost derivation for the same sentence.

Ambiguity and Precedence of Operators

Using the simplest grammar for expressions let's derive again the parse tree for:

$$\mathbf{id} + \mathbf{id} * \mathbf{id}$$

Now consider the following grammar:
$E \rightarrow E + T | E - T | T$
$T \rightarrow T * F | T / F | F$
$F \rightarrow (E) | \mathbf{id}$

Use of ambiguos grammar

In some case it can be convenient to use ambiguous grammar, but then it is necessary to define precise disambiguating rules

Ambiguity

Conditional statements

Consider the following grammar:

$$\begin{array}{lcl} \textit{stmt} & \rightarrow & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \\ & | & \textbf{if } \textit{expr} \textbf{ then } \textit{stmt} \textbf{ else } \textit{stmt} \\ & | & \textbf{other} \end{array}$$

decide if the following sentence belongs to the generated language:

$$\textbf{if } E_1 \textbf{ then if } E_2 \textbf{ then } S_1 \textbf{ else } S_2$$

Exercises

Consider the grammar:

$$S \rightarrow SS + |SS * |a$$

and the string $aa + a*$

- Give the leftmost derivation for the string
- Give the rightmost derivation for the string
- Give a parse tree for the string
- Is the grammar ambiguous or unambiguous?
- Describe the language generated by this grammar?

Define grammars for the following languages:

- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ is palindrom}\}$
- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ contains the same occurrences of 0 and 1}\}$
- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ does not contain the substring 011}\}$

Exercises

Consider the grammar:

$$S \rightarrow SS + |SS * |a$$

and the string $aa + a*$

- Give the leftmost derivation for the string
- Give the rightmost derivation for the string
- Give a parse tree for the string
- Is the grammar ambiguous or unambiguous?
- Describe the language generated by this grammar?

Define grammars for the following languages:

- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ is palindrom}\}$
- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ contains the same occurrences of 0 and 1}\}$
- $\mathscr{L} = \{w \in \{0,1\}^* | w \text{ does not contain the substring 011}\}$

CF grammars are capable to describe most, but not all, of the syntax of programming languages. For instance, the requirement that identifiers must be dclared before their usage cannot be expressed in CF grammar.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an
- A Turing machine cannot decide whether a context-free language is ambiguous or not

CF grammars are capable to describe most, but not all, of the syntax of programming languages. For instance, the requirement that identifiers must be dclared before their usage cannot be expressed in CF grammar.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language
- A Turing machine cannot decide whether a context-free language is ambiguous or not

CF grammars are capable to describe most, but not all, of the syntax of programming languages. For instance, the requirement that identifiers must be dclared before their usage cannot be expressed in CF grammar.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language
- A Turing machine cannot decide whether a context-free language is ambiguous or not

CF grammars are capable to describe most, but not all, of the syntax of programming languages. For instance, the requirement that identifiers must be dclared before their usage cannot be expressed in CF grammar.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language
- A Turing machine cannot decide whether a context-free language is ambiguous or not

CF grammars are capable to describe most, but not all, of the syntax of programming languages. For instance, the requirement that identifiers must be dclared before their usage cannot be expressed in CF grammar.

So what we can do?

Ambiguity

- Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars
- A language that only admits ambiguous grammars is called an inherently ambiguous language
- A Turing machine cannot decide whether a context-free language is ambiguous or not