ST0244 Programming Languages
2. Syntax

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2023-2

Preliminaries

Conventions

- The number and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook [Lee 2017].
- The source code examples are in course's repository.

Introduction

Syntax and Semantics

- Syntax is how programs look (well-formed programs)
- Semantics is how programs work (meaning of programs)

Introduction

Syntax and Semantics

- Syntax is how programs look (well-formed programs)
- Semantics is how programs work (meaning of programs)

Question

When you are learning/using a programming language are its syntax and its semantics equally important?

Terminology

Syntax and semantics issues

Type	Static (compile-time)	Dynamic (run-time)
Syntax	✓	
Semantic	✓	✓

Terminology

Example (p. 32)

Is the code

```
a = b + c;
```

a correct C++ statement?

Terminology

Example (p. 32)

Is the code

```
a = b + c;
```

a correct C++ statement?

Some questions:

1. Do `b` and `c` have values? (answered in run-time, dynamic semantic issue or answered in compile-time, static semantic issue)
2. Have `b` and `c` been declared as a type that allows the `+` operation? (answered in compile-time, static semantic issue)
3. Is `a` assignment compatible with the result of the expression `b + c`? (answered in compile-time, static semantic issue)
4. Does the assignment statement have the proper form? (answered in compile-time, syntactic issue)

Terminology

Definition

A **terminal** symbol (or **token**) is an elementary symbol of the language.

Example

Keywords, types, operators, numbers, identifiers, among others, are terminal symbols in a programming language.

Terminology

Definition

A **non-terminal** symbol (or **syntactic category** or **syntactic variable**) represents a sequence of terminal symbols.

Example

- C++, Java, Python and other

  Statements, expressions, if-statements, among others.

- Haskell, Standard ML and other

  Types, expressions, function applications, function abstractions, among others.

Backus-Naur Form (BNF)

Definition

Backus Naur-Form (BNF) is a formal (i.e. non-ambiguous) meta-language (i.e. a language for describing or analysing other language) for describing language syntax.

Backus-Naur Form (BNF)

BNF Rules

A BNF for a language is a set of rules such as

$$\langle\text{non-terminal}\rangle \quad ::= \quad \texttt{expression}_1 \mid \texttt{expression}_2 \mid \ldots \mid \texttt{expression}_n$$

where

(i) $\texttt{expression}_i$ is a string of terminals and non-terminals,

(ii) the symbol $::=$ means that the non-terminal symbol on the left must be replaced with one expression on the right and

(iii) the symbol $\mid$ means a choice.

Backus-Naur Form (BNF)

Example

Let P a set of propositional letters (atomic formulae) and let $p \in P$, we can define the wff's (well-formed formulae) of propositional logic by

$$
\begin{aligned}
\langle \text{formula} \rangle \quad ::= \quad & \mathtt{p} \\
\mid \quad & \neg \ \langle \text{formula} \rangle \\
\mid \quad & \mathtt{(} \ \langle \text{formula} \rangle \wedge \langle \text{formula} \rangle \ \mathtt{)} \\
\mid \quad & \mathtt{(} \ \langle \text{formula} \rangle \vee \langle \text{formula} \rangle \ \mathtt{)} \\
\mid \quad & \mathtt{(} \ \langle \text{formula} \rangle \rightarrow \langle \text{formula} \rangle \ \mathtt{)} \\
\mid \quad & \mathtt{(} \ \langle \text{formula} \rangle \leftrightarrow \langle \text{formula} \rangle \ \mathtt{)}
\end{aligned}
$$

Remark

Note the recursive definition of wff's.

Backus-Naur Form (BNF)

Example

A BNF describing the integer numbers, with or without sign (e.g. -344, 56, $+9784$, 8, 0000).

$$
\begin{aligned}
\langle \text{integer} \rangle \quad &::= \quad \langle \text{sign} \rangle \langle \text{digits} \rangle \ \mid \ \langle \text{digits} \rangle \\
\langle \text{digits} \rangle \quad &::= \quad \langle \text{digit} \rangle \langle \text{digits} \rangle \ \mid \ \langle \text{digit} \rangle \\
\langle \text{digit} \rangle \quad &::= \quad \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\langle \text{sign} \rangle \quad &::= \quad \texttt{+} \mid \texttt{-}
\end{aligned}
$$

Backus-Naur Form (BNF)

Example

The set of λ-terms of the λ-calculus can be defined by

$$
\begin{aligned}
\langle \text{variable} \rangle &::= \texttt{x} \mid \texttt{x'} \mid \texttt{x''} \mid \dots \\
\langle \lambda\text{-term} \rangle &::= \langle \text{variable} \rangle \\
&\quad \mid \texttt{(}\ \lambda\ \langle \text{variable} \rangle\ \texttt{.}\ \langle \lambda\text{-term} \rangle\ \texttt{)} \\
&\quad \mid \texttt{(}\ \langle \lambda\text{-term} \rangle\ \langle \lambda\text{-term} \rangle\ \texttt{)}
\end{aligned}
$$

Backus-Naur Form (BNF)

Example

A BNF describing a part of Java (pp. 33–34).

$$
\begin{aligned}
\langle\text{primitive-type}\rangle &::= \texttt{boolean} \mid \texttt{char} \mid \texttt{byte} \mid \texttt{short} \mid \texttt{int} \mid \texttt{long} \mid \texttt{float} \mid \ldots \\
\langle\text{argument-list}\rangle &::= \langle\text{expression}\rangle \mid \langle\text{argument-list}\rangle \texttt{ , } \langle\text{expression}\rangle \\
\langle\text{selection-statement}\rangle &::= \texttt{if (} \langle\text{expression}\rangle \texttt{) } \langle\text{statement}\rangle \\
&\quad \mid \texttt{if (} \langle\text{expression}\rangle \texttt{) } \langle\text{statement}\rangle \texttt{ else } \langle\text{statement}\rangle \\
&\quad \mid \texttt{switch (} \langle\text{expression}\rangle \texttt{) } \langle\text{block}\rangle \\
\langle\text{m[ethod]-declaration}\rangle &::= \langle\text{modifiers}\rangle\ \langle\text{type-specifier}\rangle\ \langle\text{m-declarator}\rangle\ \langle\text{throws-clause}\rangle\ \langle\text{m-body}\rangle \\
&\quad \mid \langle\text{modifiers}\rangle\ \langle\text{type-specifier}\rangle\ \langle\text{m-declarator}\rangle\ \langle\text{m-body}\rangle \\
&\quad \mid \langle\text{type-specifier}\rangle\ \langle\text{m-declarator}\rangle\ \langle\text{throws-clause}\rangle\ \langle\text{m-body}\rangle \\
&\quad \mid \langle\text{type-specifier}\rangle\ \langle\text{m-declarator}\rangle\ \langle\text{m-body}\rangle
\end{aligned}
$$

Backus-Naur Form (BNF)

Extended BNF (EBNF)

We shall extended BNF with the following definitions:

i) ‘item?’ or ‘[item]’ means the item is optional.

ii) ‘item*’ or ‘{item}’ means zero or more occurrences of the item are allowable.

iii) ‘item+’ means one or more occurrences of the item are allowable.

iv) Parentheses may be used for grouping.

Context-Free Grammars

Definition

A **context-free grammar** is a 4-tuple

$$G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S}),$$

where

- $\mathcal{N}$ is a finite set of non-terminal symbols,
- $\mathcal{T}$ is a finite set of terminal symbols,
- $\mathcal{P}$ is a finite set of productions of the form $A \to \alpha$, with $A \in \mathcal{N}$ and $\alpha \in \{\mathcal{N} \cup \mathcal{T}\}^*$,
- $\mathcal{S} \in \mathcal{N}$ is the start symbol,

$\{\mathcal{N} \cup \mathcal{T}\}^*$: String of terminals and non-terminals symbols including the empty word ϵ.

Context-Free Grammars

Example (infix expressions grammar (§ 2.3.1))

We can define a context-free grammar $(\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ for infix expressions by

$$\begin{aligned}\mathcal{N} &= \{E, T, F\},\\ \mathcal{T} &= \{\text{identifier}, \text{number}, +, -, *, /, (,)\},\end{aligned}$$

and the productions in the set $\mathcal{P}$ are

$$\begin{aligned}E &\to E + T \mid E - T \mid T\\ T &\to T * F \mid T/F \mid F\\ F &\to (\ E\ ) \mid \text{identifier} \mid \text{number}\end{aligned}$$

Derivations

Definition

A **sentence** of a grammar G is a string of tokens (terminal symbols) from G.

Derivations

Definition

A **sentence** of a grammar G is a string of tokens (terminal symbols) from G.

Example

Two sentences of the infix expressions grammar are `(5*x)+y` and `)4++(`.

Derivations

Definition

A **sentential form** of a grammar G is a string of terminals and non-terminals symbols from G.

Derivations

Definition

A **sentential form** of a grammar G is a string of terminals and non-terminals symbols from G.

Example

Two sentential forms of the infix expressions grammar are $(T * F) + T$ and $(5 * F) + T$.

Derivations

Definition

A **derivation** of a sentence S in a grammar G is a sequence of sentential forms of G that starts with the start symbol of G and ends with S.

Derivations

Definition

A **derivation** of a sentence S in a grammar G is a sequence of sentential forms of G that starts with the start symbol of G and ends with S.

Remark

Every sentential form in the derivation is obtained from the previous one by replacing $A \in \mathcal{N}$ (non-terminal symbol) by $\alpha \in \{\mathcal{N} \cup \mathcal{T}\}^*$ (string of terminals and non-terminals symbols), if $A \rightarrow \alpha$ is a production of G.

Derivations

Definition

A sentence S of a grammar G is **valid** iff there exists at least one derivation for S in G.

Derivations

Example

The sentence (5*x)+y of the infix expressions grammar is valid because has the following derivation:

$$
\begin{aligned}
\underline{E} &\Rightarrow \underline{E} + T \\
&\Rightarrow \underline{T} + T \\
&\Rightarrow \underline{F} + T \\
&\Rightarrow (\underline{E}) + T \\
&\Rightarrow (\underline{T}) + T \\
&\Rightarrow (\underline{T} * F) + T \\
&\Rightarrow (\underline{F} * F) + T \\
&\Rightarrow (5 * \underline{F}) + T \\
&\Rightarrow (5 * x) + \underline{T} \\
&\Rightarrow (5 * x) + \underline{F} \\
&\Rightarrow (5 * x) + y
\end{aligned}
$$

$$
\left(
\begin{array}{l}
E \rightarrow E + T \\
E \rightarrow E - T \\
E \rightarrow T \\
T \rightarrow T * F \\
T \rightarrow T / F \\
T \rightarrow F \\
F \rightarrow (\ E\ ) \\
F \rightarrow \text{identifier} \\
F \rightarrow \text{number}
\end{array}
\right)
$$

Derivations

Definition

Let G be a grammar. The **language** of G, denoted $L(G)$, is the set of valid sentences of G.

Derivations

Types of derivations

- Left-most derivation (always replace the left-most non-terminal symbol).
- Right-most derivation (always replace the right-most non-terminal symbol).

Derivations

Example

Left-most and right-most derivations of `(5*x)+y`.

$$
\text{left-most}\begin{cases}
\underline{E} \Rightarrow \underline{E} + T \\
\quad \Rightarrow \underline{T} + T \\
\quad \Rightarrow \underline{F} + T \\
\quad \Rightarrow (\underline{E}) + T \\
\quad \Rightarrow (\underline{T}) + T \\
\quad \Rightarrow (\underline{T} * F) + T \\
\quad \Rightarrow (\underline{F} * F) + T \\
\quad \Rightarrow (5 * \underline{F}) + T \\
\quad \Rightarrow (5 * x) + \underline{T} \\
\quad \Rightarrow (5 * x) + \underline{F} \\
\quad \Rightarrow (5 * x) + y
\end{cases}
\quad
\text{right-most}\begin{cases}
\underline{E} \Rightarrow E + \underline{T} \\
\quad \Rightarrow E + \underline{F} \\
\quad \Rightarrow \underline{E} + y \\
\quad \Rightarrow \underline{T} + y \\
\quad \Rightarrow \underline{F} + y \\
\quad \Rightarrow (\underline{E}) + y \\
\quad \Rightarrow (\underline{T}) + y \\
\quad \Rightarrow (T * \underline{F}) + y \\
\quad \Rightarrow (\underline{T} * x) + y \\
\quad \Rightarrow (\underline{F} * x) + y \\
\quad \Rightarrow (5 * x) + y
\end{cases}
$$

$$
\begin{pmatrix}
E \rightarrow E + T \\
E \rightarrow E - T \\
E \rightarrow T \\
T \rightarrow T * F \\
T \rightarrow T / F \\
T \rightarrow F \\
F \rightarrow (\ E\ ) \\
F \rightarrow \text{identifier} \\
F \rightarrow \text{number}
\end{pmatrix}
$$

Derivations

Prefix expressions

In prefix expressions the operator appears before the operands.

Example

```
4 + (a - b) * x   (infix expression)
  + 4 * - a b x   (prefix expression)
```

Derivations

Example (prefix expressions grammar (§ 2.4.3))

We can define a context-free grammar $(\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ for prefix expressions by

$$\mathcal{N} = \{E\},$$
$$\mathcal{T} = \{\text{identifier}, \text{number}, +, -, *, /\},$$

and the productions in the set $\mathcal{P}$ are

$$E \to +\ E\ E \mid -\ E\ E \mid *\ E\ E \mid /\ E\ E \mid \text{identifier} \mid \text{number}$$

Parser Trees

Definition

Let G be a grammar. A **parser tree** is a tree representing of a sentence of $L(G)$.

Parser Trees

Definition

Let G be a grammar. A **parser tree** is a tree representing of a sentence of $L(G)$.

Properties

Let G be a grammar. A parser tree of a sentence of $L(G)$ has the following properties [Aho, Lam, Sethi and Ullman 2006]:

(i) The root is labelled by the start symbol of G.

(ii) Each leaf is labelled by a terminal symbol of G or by ϵ.

(iii) Each interior node is labelled by a non-terminal symbol of G.

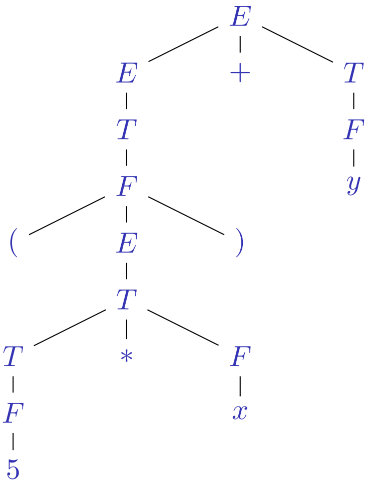
(iv) If A is a non-terminal of symbol of G labelling some interior node and $X_1, X_2, \ldots, X_n$ are the labels of the children of that node from left to right, then there must be a production $A \to X_1, X_2, \ldots, X_n$ in G.

Parser Trees

Example

Parser tree for the sentence `(5*x)+y` of the infix expressions grammar.

$$
\begin{aligned}
\underline{E} &\Rightarrow \underline{E} + T \\
&\Rightarrow \underline{T} + T \\
&\Rightarrow \underline{F} + T \\
&\Rightarrow (\underline{E}) + T \\
&\Rightarrow (\underline{T}) + T \\
&\Rightarrow (\underline{T} * F) + T \\
&\Rightarrow (\underline{F} * F) + T \\
&\Rightarrow (5 * \underline{F}) + T \\
&\Rightarrow (5 * x) + \underline{T} \\
&\Rightarrow (5 * x) + \underline{F} \\
&\Rightarrow (5 * x) + y
\end{aligned}
$$



Abstract Syntax Trees (AST)

Definition

An **abstract syntax tree** is a parser tree without non-essential information required for evaluating (generate code in compilation or execute in interpretation) the sentence (p. 38):

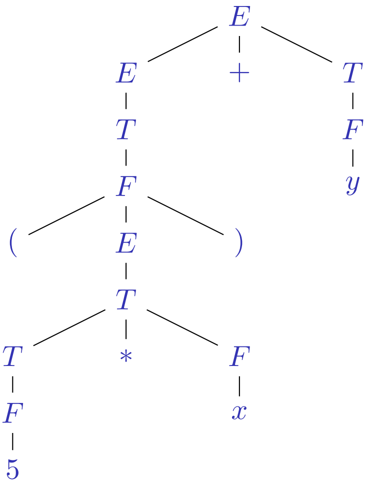
i) 'Non-terminal nodes in the tree are replaced by nodes that reflect the part of the sentence they represent.'

ii) 'Unit productions in the tree are collapsed.'

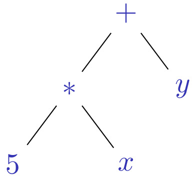
Abstract Syntax Trees (AST)

Example

Parser tree and AST (the interior nodes represent operators and the leafs represent operands) for the sentence `(5*x)+y`.



Parser tree



AST

Ambiguity in Grammars

Definition

A grammar G is **ambiguous** iff there is (at least) a sentence in $L(G)$ that has more than one parse tree.

Ambiguity in Grammars

Definition

A grammar G is **ambiguous** iff there is (at least) a sentence in $L(G)$ that has more than one parse tree.

Remark

Recall that a sentence of a grammar can have various derivations.

Ambiguity in Grammars

Example

Given the grammar $(\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ where

$$\mathcal{N} = \{E\},$$
$$\mathcal{T} = \{*, +, 0, 1, 2, 3, 4, 5, 6, 8, 9\},$$

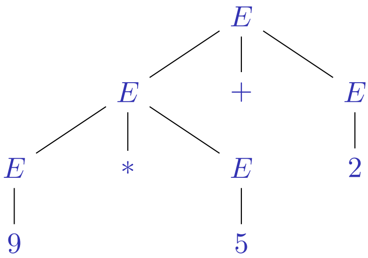
$$E \to E * E \mid E + E$$
$$E \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

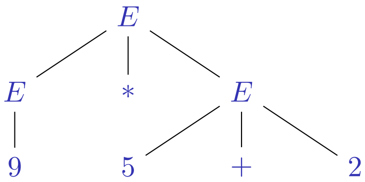
(continued on next slide)

Ambiguity in Grammars

Example

The sentence 9*5+2 has two parser trees.





Limitations of Syntactic Definitions

Some limitations

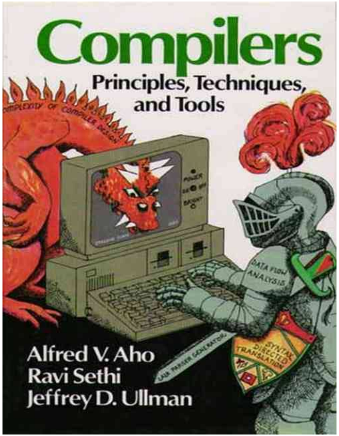
- The syntax of a programming language is an incomplete description of it (e.g. `5 + 4/0`).
- ‘The set of programs in any interesting language is not context-free.’ (p. 50) (e.g. `a + b`)
- A (context-free) grammar does not specify the semantics of a (programming) language.

Limitations of Syntactic Definitions

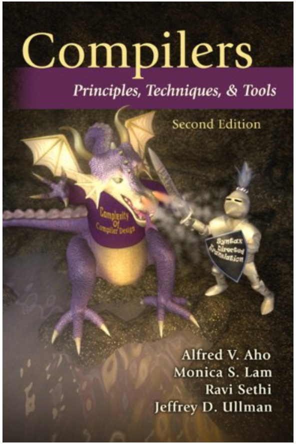
Example (context-sensitive issues (p. 50–51))

- In an array declaration in C++, the array size must be a non-negative value.
- Operands for the && operation must be boolean in Java.
- In a method definition, the return value must be compatible with the return type in the method declaration.
- When a method is called, the actual parameters must match the formal parameter types.

The 'Dragon Book'



(First edition, 1986)



(Second edition, 2006)

References

- Aho, Alfred V., Lam, Monica S., Sethi, Ravi and Ullman, Jeffrey D. [1986] (2006). Compilers: Principles, Techniques, & Tools. 2nd ed. Addison-Wesley (cit. on pp. 32, 33).
- Lee, Kent D. [2014] (2017). Foundations of Programming Languages. 2nd ed. Undergraduate Topics in Computer Science. Springer (cit. on p. 2).