Principles of Compiler Construction

Prof. Wen-jun LI
School of Computer Science and Engineering
lnslwj@mail.sysu.edu.cn

Lecture 2. Formal Languages

语法

1. **What** Is a Language, Formally?
2. **How** to Define a Specific Language?
3. Context-Free Grammar
4. Parse Tree
5. Ambiguity
6. Chomsky Hierarchy

Formal System

- Don't care the meaning of symbols.
  - Symbols can be substituted.
- Only depend on formally defined rules.
- Programs are formal systems.

1. What Is a Language, Formally?

- Alphabet ::= a set of symbols
  - Non-empty and finite
  - Symbol is a meta-definition元定义
- String (sentence or word) ::= a sequence of symbols which are from the alphabet
  - 了解signature Finite; Permit empty string ε [*epsilon*]零元
  - 创建空串 求长度 前缀 后缀
  - Manipulation: empty, length, prefix, suffix,
  - 子串 substring, proper ~去除自身, subsequence,可跳过部分字母 concatenation, Kleene star (closure)
  - 联接 克林闭包：把给定string的k次方并在一起，k个相连接
- Σ^* :: = the set of all possible strings on Σ
  - 有字母形成的最大集合 集合上求集合闭包
  - Mathematically, a monoid (semigroup with an identity) generated by Σ.

Formal Definition of Language

- Language ::= A set of strings
  - Maybe infinite, or an empty set {} (or $\varnothing$)
- Language manipulation
  - Union: $L_1 \cup L_2$
  - Concatenation: $L_1 L_2$
  - Exponential: $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$, ...
  - Kleene closure (star): $L^* = \bigcup_{i=0}^{\infty} L^i$
  - Positive closure (plus): $L^+ = \bigcup_{i=1}^{\infty} L^i = L^* L$
- What is a meta-definition?

Mapping (Binding) to Practice

- Binding
  - Abstract: formal definition of a language
  - Concrete: programming languages
- **Discussion**
  - For C programs
    - Alphabet Σ → ? ASCII
    - Σ^* → ? ASCII的任意组合
    - Language → ? Σ*中语法正确的真子集
    - A string in the language → ? 符合语法的程序
  - For Java programs
    - Alphabet Σ → ? Unicode
    - A string in the language → ?

2. How to Define a Specific Language?

- Essentially, how to define a set:
  - Enumeration
    - { aa, ab, ba, bb }
  - Partial enumeration 部分枚举
    - { a, ab, abb, abbb, ... }
  - Predicate description 描述法
    - 谓语
    - $\{ x \mid \mathbf{P}(x) \wedge x \in \Sigma^* \}$
- Problem: how to define the predicate **P**?
  - Described (by an expression)
  - Generated (by a grammar)
  - Recognized (by an automaton)
  自动机

3. Context-Free Grammar (CFG)

- CFG is a 4-tuple (Σ, N, P, S)
  - Σ: alphabet, set of terminals 终结符号
  - $N \cap \Sigma = \emptyset$: set of nonterminals 非终结符 大写
    无交集
  - P: set of rewriting rules (productions)
    - Each production has the form $A \rightarrow \alpha$,
      where $A \in N \wedge \alpha \in (\Sigma \cup N)^*$
      A被α定义
      随意组合
  - $S \in N$: start (goal) symbol

An Example

- A Context-Free Grammar
  - $G_1 = (\{a, b, c\}, \{A, B\}, \{A \rightarrow aB, A \rightarrow bB, A \rightarrow cB, B \rightarrow a, B \rightarrow b, B \rightarrow c\}, A)$
- Brief notation

简化

第一行为开始标志

$$A \rightarrow aB \mid bB \mid cB$$

$$B \rightarrow a \mid b \mid c$$

大写：非终结
小写：终结

也有些为 : :=
=

Derivation

- Derive (rewrite) 推导
  - $A \Rightarrow bB \Rightarrow bc$
  - Direct derivation: $bB \Rightarrow bc$
  - n-step derivation: $A \Rightarrow^2 bc$ 推导两步
  - 0 or more steps derivation: $A \Rightarrow^* bc$ 若干步
  - 1 or more steps derivation: $A \Rightarrow^+ bc$ 至少1步
  - Left-most derivation: $A \Rightarrow^*_{lm} bc$ —直选最左边的非终结符推导
    - Every step must be left-most derivation
  - Right-most derivation: $A \Rightarrow^*_{rm} bc$
    - Every step must be right-most derivation
  - Canonical derivation: right-most
    经典推导

Binding to Practice

- Arithmetic expressions

  越下面优先级越高

  $E \rightarrow E + T \mid T$ term

  $T \rightarrow T * F \mid F$ factor

  $F \rightarrow (E) \mid \boxed{n}$ 怎样写合法表达式

- Derivation of **n+n*n**
  - $\mathbf{E} \Rightarrow E+\mathbf{T} \Rightarrow \mathbf{E}+T*F \Rightarrow \mathbf{T}+T*F \Rightarrow F+\mathbf{T}*F \Rightarrow F+F*\mathbf{F} \Rightarrow F+\mathbf{F}*n \Rightarrow \mathbf{F}+n*n \Rightarrow n+n*n$
  - Not canonical!

- Canonical derivation of **n+n*n**
  - $\mathbf{E} \Rightarrow E+\mathbf{T} \Rightarrow E+T*\mathbf{F} \Rightarrow E+\mathbf{T}*n \Rightarrow E+\mathbf{F}*n \Rightarrow \mathbf{E}+n*n \Rightarrow \mathbf{T}+n*n \Rightarrow \mathbf{F}+n*n \Rightarrow n+n*n$

Each nonterminal represents a subset of the language

Reduction 归约

- Reduce
  - Derive in reverse order
  - Direct reduction
  - n-step reduction
  - Left-most (canonical) reduction
  - Right-most reduction

Sentences and Handles

- Sentential form 句型 $\alpha \in (\Sigma \cup N)^*$ 随意组合
  - If S $\Rightarrow^* \alpha$, α is a sentential form of G.
  - Right (Left) sentential form
- Sentence
  - A sentential form with no nonterminals.
- Phrase
  - If S $\Rightarrow^* \alpha A \omega \wedge A \boxed{\Rightarrow^+} \beta$, β is a phrase of $\alpha\beta\omega$ on A.
- Simple phrase
  - If S $\Rightarrow^* \alpha A \omega \wedge A \Rightarrow \beta$, β is a simple phrase.
- Handle 最左直接短语
  - If S $\Rightarrow^*_{rm} \alpha A \omega \Rightarrow_{rm} \alpha\beta\omega$, A $\rightarrow \beta$ in the position following α is a handle of $\alpha\beta\omega$.

规约：从给定串推导开始符合

Define a Language by Grammar

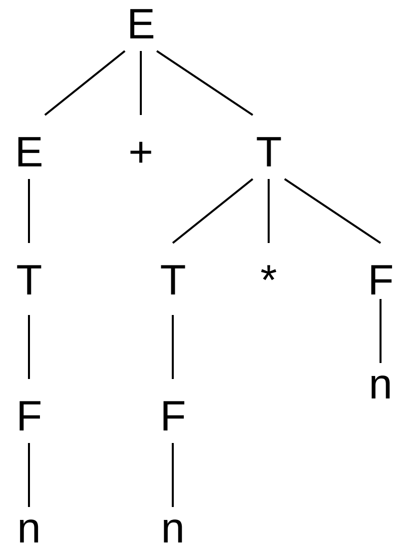
- Language defined by a given CFG G
  - $L(G) = \{\alpha \mid S \Rightarrow^* \alpha \wedge \alpha \in \Sigma^*\}$ 可换为+符号
  - That is the set of all sentences of G
- Set of all sentential forms of G is
  - 不能换+
  - $SF(G) = \{\alpha \mid S \Rightarrow^* \alpha \wedge \alpha \in (\Sigma \cup N)^*\}$
- Thus we have $L(G) = SF(G) \cap \Sigma^*$.

BNF (Backus-Naur Form)

- Equivalent to CFG
  - Change "→" to "::="
  - De facto standard to define a computer language
- EBNF (Extended BNF)
  - ISO/IEC 14977:1996(E). *The Standard Metalanguage Extended BNF*.
  - =, "terminals", [optional], {repetition}, (group), (*comment*), etc.

4. Parse Tree

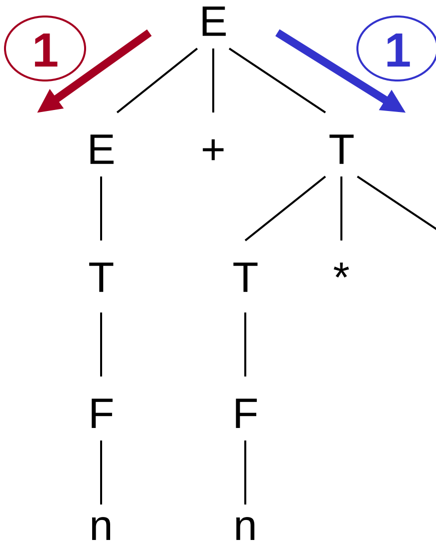
- Graphic representation of derivations
  - Root: start symbol
  - Leafs: terminals or ε
  - Interior nodes: nonterminals
  - Offspring of a nonterminal: a production
- An example
  - $\mathbf{E} \Rightarrow E+\mathbf{T} \Rightarrow E+T*\mathbf{F}$
    $\Rightarrow E+\mathbf{T}*n \Rightarrow E+\mathbf{F}*n$
    $\Rightarrow \mathbf{E}+n*n \Rightarrow \mathbf{T}+n*n$
    $\Rightarrow \mathbf{F}+n*n \Rightarrow n+n*n$



Insight into a Parse Tree

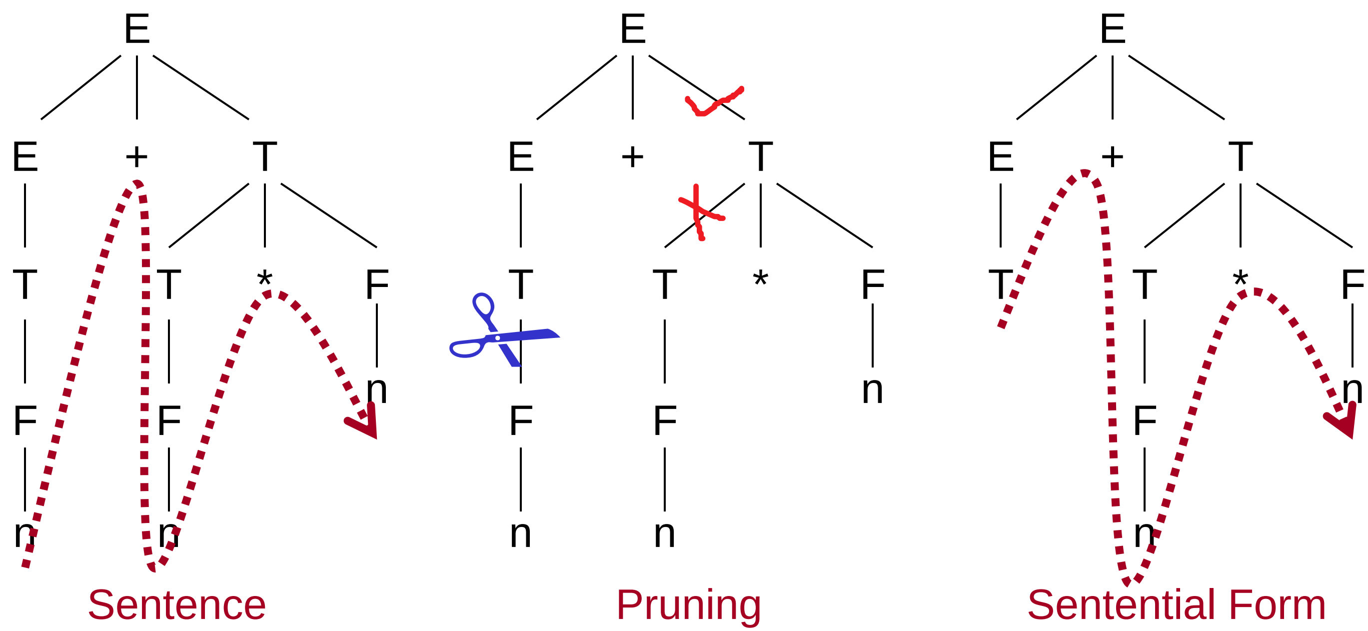
抽象层次更高，蕴藏了推到顺序

- An **abstraction** of derivations
  - Derivation order is discarded.



Sentences and Sentential Forms in a Parse Tree

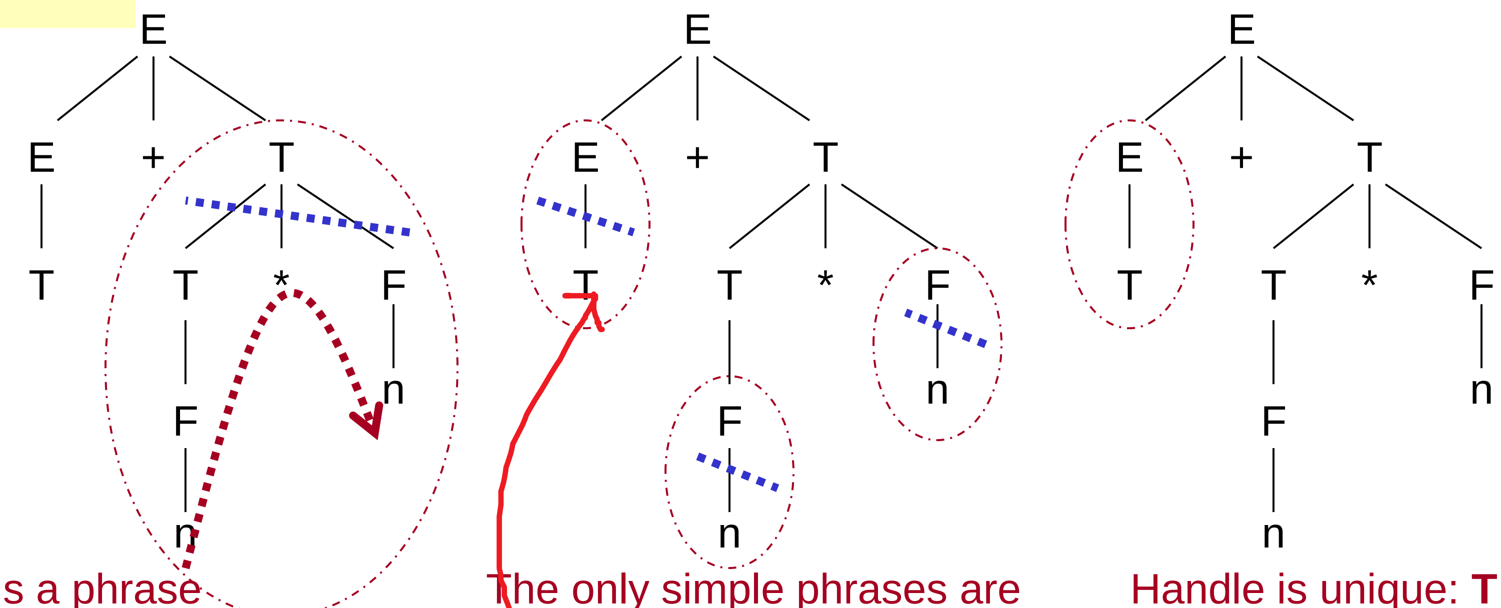
- Sentence
  - Frontier of the parse tree, e.g. n+n*n
- Sentential Form
  - Frontier of the pruned parse tree, e.g. T+n*n



两层即为一个直接短语

Phrases and Handles in a Parse Tree

- For a given sentential form, e.g. T+n*n
  - Phrase: frontier of a nonterminal subtree
  - Simple phrase: frontier of a 2-level subtree
  - Handle: left-most simple phrase



n*n is a phrase

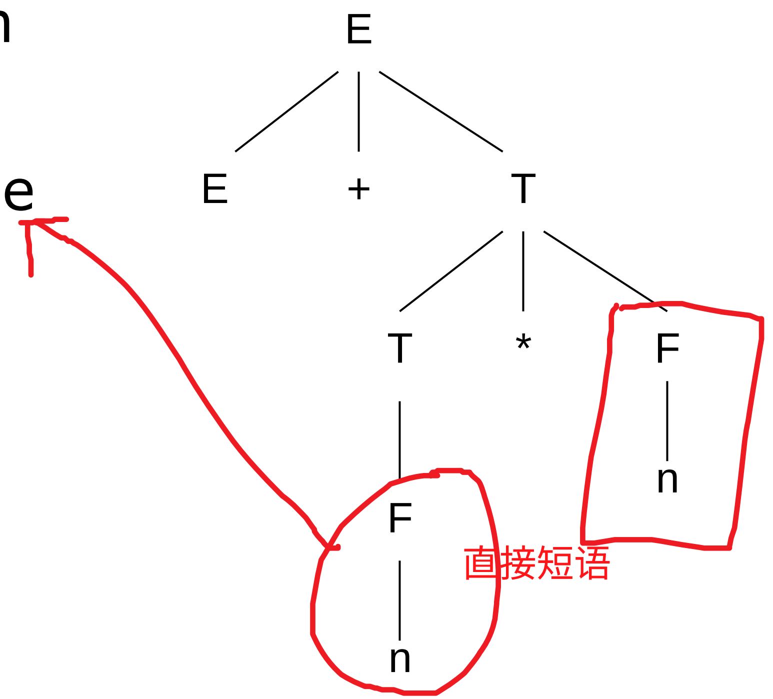
The only simple phrases are **T**, **n**, and **n**

Handle is unique: **T**

最左的两层的树即为句柄

Discussions

- What is ...
  - The sentential form
  - All simple phrases
  - The only one handle



不断pruning，最后即为开始

Parsing

- There are 2 questions that must be answered by parsing
  - Is the source program well-formed (legal in syntax)?
    - That is, is it a sentence of the language?
  - If it is, what's the syntax construction of the program?
    - Tree is an ideal data structure to present **part-of** relationship.
- 2 parsing strategies
  - Top-down vs. bottom-up

Top-Down Parsing

- Beginning at the start symbol
  - Usually expanding nonterminals in depth-first manner (predictive in nature)
  - Left-most derivation
  - Pre-order traversal of the parse tree
- Example: LL(k)
  - Read from **L**eft, and **L**eft-most derivation, with **k** lookaheads
  - Recursive descent (predictive) parsing

递归下降预测分析

Bottom-Up Parsing

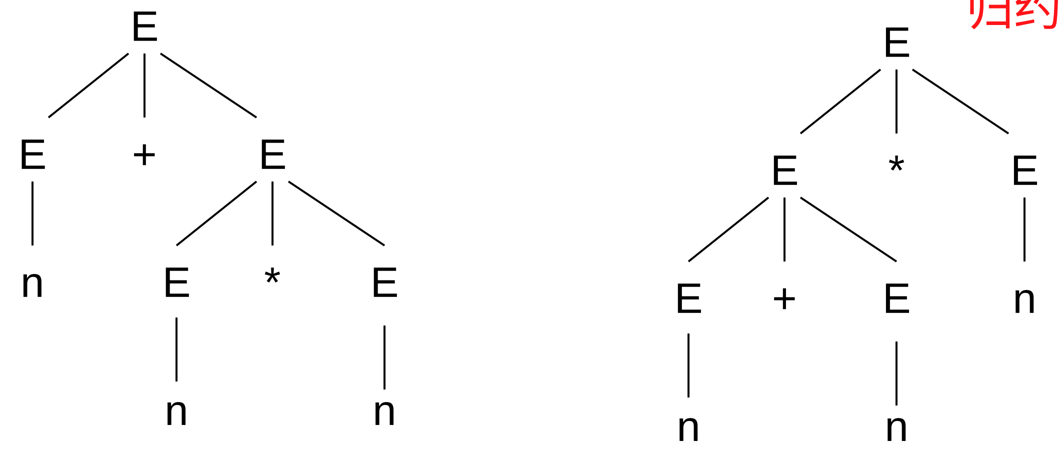
- Beginning from the terminal input string
  - Determining the production used to generate leaves
  - Right-most derivation in reverse order
  - Post-order traversal of the parse tree
- Example: LR(k)
  - Read from **L**eft, and **R**ight-most derivation, with **k** lookaheads 看几个候选式，越多越好。理论k可以=0，只看一个
  - Parser generator yacc

5. Ambiguity

12 recovering concepts

- A grammar with more than one $\boxed{\text{parse tree}}$ for at least one sentence in the language
  - $E \rightarrow E + E \mid E * E \mid (E) \mid n$

不能改为推导或归约，应该是不同的最左、右归约、推导



2 parse trees for the sentence **n + n * n**

Ambiguous: Grammar vs. Language

- Ambiguous Language (inherent)
  - There exists no unambiguous grammar to define the language.
- Ambiguous Grammar (postnatal)
  - May be transformed to an equivalent unambiguous grammar.
  - The following two grammars define the same language
    - E → E + E | E * E | (E) | n
    - E → E + T | T
      T → T * F | F
      F → (E) | n

postnatal
英 [ˌpəʊstˈneɪtl] 美 [ˌpoʊstˈneɪtl]
adj. 产后的，出生后的

优先级高的往下走，因为先形成子树

Binding to Practice

- Ambiguity in Programming Languages
  - Operators in an expression
  - Dangling **`else`** in embedded **`if`** statements
- Both of them are postnatal
  - Can be removed by additional disambiguation rules
    - Precedence and associativity
    - Inner "**`if`**" matched
  - Two approaches to resolving ambiguities
    - Rewrite the ambiguous grammar
    - Ad hoc solution: application of additional rules

6. Chomsky Hierarchy

向一个方向不断展开

Class	Grammar	Restriction	Recognizer
3	Regular 正则文法	$A \to aB$ or $A \to a$, where $A, B \in N \wedge a \in \Sigma \cup \{\varepsilon\}$. $A \to \varepsilon$ permitted if A is the start symbol and does not appear on the right of any production.	Finite-State Automaton (**FSA**) 有限状态指令集
2	Context-Free	$A \to \alpha$, where $A \in N \wedge \boxed{\alpha} \in (\Sigma \cup N)^*$. 串	Push-Down Automaton (**PDA**) 向下
1	Context-Sensitive	$\alpha \to \beta$, where 必须越推越长 $\alpha, \beta \in (\Sigma \cup N)^* \wedge \alpha \neq \varepsilon \wedge \|\alpha\| \le \|\beta\|$. β can't be ε, unless α is the start symbol and does not appear on the right of any production.	Linear-Bounded Automaton (**LBA**) 线性有界
0	Unrestricted	$\alpha \to \beta$, where $\alpha, \beta \in (\Sigma \cup N)^* \wedge \alpha \neq \varepsilon$.	Turing Machine (**TM**)

Useful in Practice

Useful in Theory

过强，难以实现

Regular Language

- Ability of regular languages
  - What is it able to define?
    - Identifiers
    - Decimal constants
  - What isn't it able to define?
    - Matched parentheses 匹配结构
- Satisfy the requirement of lexical descriptions

Context-Free Language

- Ability of context-free languages
  - What is it able to define?
    - Matched constructs
  - What isn't it able to define?
    - Only use declared variables
    - Matched parameter passing
- Satisfy the requirement of syntactic descriptions

Exercise 2.1

- Given the following grammar:

$$S \rightarrow \mathbf{(}\ L\ \mathbf{)}\ |\ \mathbf{a}$$

$$L \rightarrow L\ \mathbf{,}\ S\ |\ S$$

Construct a parse tree for the sentence **(a, ((a, a), (a, a)))**

Exercise 2.2

- Given the following grammar:

  bexpr $\rightarrow$ bexpr **or** bterm | bterm

  bterm $\rightarrow$ bterm **and** bfactor | bfactor

  bfactor $\rightarrow$ **not** bfactor | **(** bexpr **)** | **true** | **false**

  Construct a parse tree for the sentence

  not (true or false)

Exercise 2.3

- Is the grammar:

  $$S \rightarrow a\,S\,b\,S \mid b\,S\,a\,S \mid \varepsilon$$

  ambiguous? Why?

Further Reading

- Dragon Book, 2nd Edition (DBv2)
  - Comprehensive reading:
    - Section 2.1 – 2.3, 4.2.1 – 4.2.6 for the concepts of CFG, derivation, parse tree and ambiguity.
    - Section 4.5.1 – 4.5.2 for the concepts of reduction and handle.
  - Skip reading:
    - Section 2.4 for top-down parsing.
- Skip reading: definition of languages
  - BNF or EBNF of Oberon-0, Oberon, OMG IDL, and Lua from our course website
- Skip reading: EBNF standard
  - ISO/IEC. The Standard Metalanguage EBNF.

Enjoy the Course!