Information Science and Technology College of
Northeast Normal University

Franklin's 13 virtues and Puritan ethics

Temperance
Silence

(3)Order : Let all your things have their places; Let each part business have its time.

条理：物归其位，做事之前制定时间表，按照计划行事

(4)Resolution: Resolve to perform what you ought; Perform without fail what you resolve.

决断：做事应该果断，不要拖泥带水，一旦决定就要付诸行动，还要坚持到底。

Always bear in mind that your own resolution to succeed is more important than any one thing.（疯狂渴求，强烈呼唤）

------Abraham Lincion

Information Science and Technology College of
Northeast Normal University

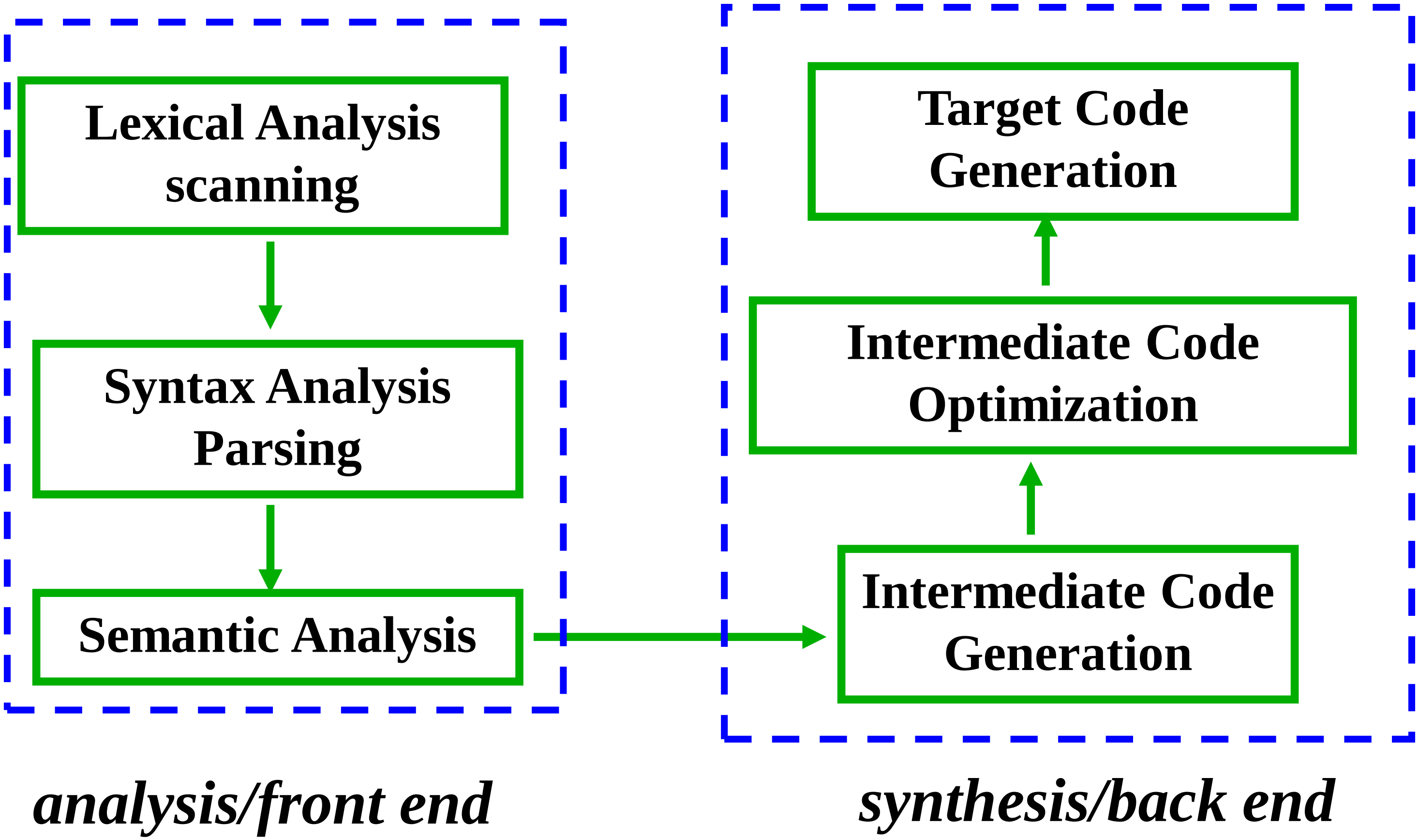
Compiling and Running of Program

Dr. Zheng Xiaojuan
Professor

October. 2019

Information Science and Technology College of
Northeast Normal University

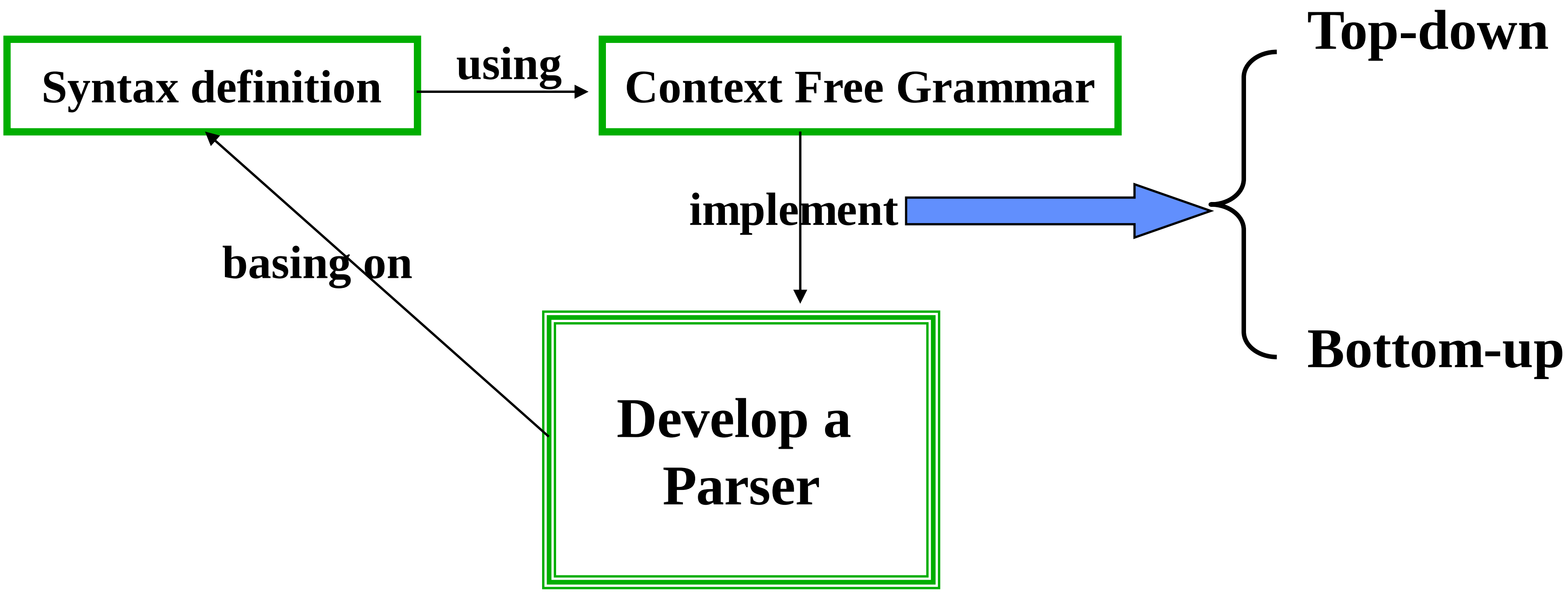
Role of Parsing in a Compiler



What will be introduced

- **About Parsing**
  - **General information about a Parser (parsing)**
    - **Functional requirement (input, output, main function)**
    - ***Process***
  - **General techniques in developing a Parser**
    - **How to define the syntax of a programming language?**
      - **Why not regular expressions? (not enough)**
      - ***Context free grammar (上下文无关文法)***
    - **How to implement a parser with respect to its definition of syntax?**
      - ***Top-down Parsing（属于分析判断）***
      - ***Bottom-up Parsing（属于综合判断）***

Knowledge Relation Graph



§ 3 Context Free Grammar & Parsing

3.1 The Parsing Process （语法分析过程）

3.2 Context-free Grammars （上下文无关文法）

3.3 Parse Trees and Abstract Syntax Tree （语法分析树和抽象语法树）

3.4 Ambiguous （二义性）

3.5 Syntax of *C0* Language （简单语言的语法）

3.1 The Parsing Process

- **Function of a Parser**
  - **Input : Token / TokenList**
  - **Output: internal representation of syntactic structure**
  - **Process**
    - ***Read*** **tokens**
    - ***Establish*** **syntactic structure – parse tree/syntax tree, according to the syntax definition (context free grammar);**
    - ***Check*** **syntactical errors**

Syntactic Structures

- **Rules for describing the structure of a well-defined program**

(1) Program

(2) Declaration

- **- Constant declaration**
- **- Type declaration**
- **- Variable Declaration**
- **- Procedure/function declaration**

(3) Body

(4) Statements (assignment, conditional, loop, function call)

(5) Expressions (arithmetic, logical, boolean)

Syntax Errors

- **Different Types of Syntax Errors**
  - **Following token for a syntactic structure is wrong (后继单词错)**
  - **Identifier/constant error (标识符或者常量错)**
  - **Keyword error(关键字错)**
  - **Start token for a syntactic structure is wrong(开始单词错)**
  - **Unbalanced parentheses(括号配对错)**

Syntax Errors

```
int  GetMax( int x ; int y)
{ if (x>y) {return x  }
  else return y;
} }

vod main()
{
  real 10, y;
  10 + ;
  GetMax( x, y) ;
}
```

后继单词错

括号配对错

关键字错, 开始单词错

标识符错

开始单词错

Dealing with Errors

- **Once an error is detected, how to deal with it?**
  - **Quit immediately, not practical**
  - **Error recovery**
  - **Error repair**
  - **Error correction**
  - **There is no "perfect" way to do it!**

Different Types of Parsing Methods

- **Universal parsing methods for any grammars**
  - **Cocke-Younger-Kasami algorithm**
  - **Earley's algorithm**

  inefficient
- **Top-down parsing methods (limited grammars)- predictive**
  - **Recursive descendent parsing (递归下降法)**
  - **LL(k) -- k=1**
- **Bottom-up parsing methods (limited grammars) – shift-reduce**
  - **SLR(k)**
  - **LR(k)**
  - **LALR(k)**

  k=1
  - **Operator-precedence parsing (简单优先关系法)**

§ 3 Context Free Grammar & Parsing

3.1 The Parsing Process （语法分析过程）

3.2 Context-free Grammars （上下文无关文法）

3.3 Parse Trees and Abstract Syntax Tree （语法分析树和抽象语法树）

3.4 Ambiguous （二义性）

3.5 Syntax of Sample Language （简单语言的语法）

3.2 Context-free Grammars

- **What is a *<u>Grammar</u>*?**
- **Chomsky classification of Grammars;**
- **Context Free Grammar (some concepts)**

- 文法： 描述语言的语法结构的形式规则
- He gave me a book.

<句子> → <主语><谓语><间接宾语><直接宾语>
<主语> → <代词>
<谓语> → <动词>
<间接宾语> → <代词>
<直接宾语> → <冠词> <名词>
<代词> → He
<代词> → me
<名词> → book
<冠词> → a
<动词> → gave

1. <句子> → <主语><谓语><间接宾语><直接宾语>
2. <主语> → <代词>
3. <谓语> → <动词>
4. <间接宾语> → <代词>
5. <直接宾语> → <冠词> <名词>
6. <代词> → He
7. <代词> → me
8. <名词> → book
9. <冠词> → a
10. <动词> → gave

<句子>
⇒<主语><谓语><间接宾语><直接宾语>
⇒<代词><谓语><间接宾语><直接宾语>
⇒He <谓语><间接宾语><直接宾语>
⇒He <动词> <间接宾语><直接宾语>
⇒He gave <间接宾语><直接宾语>
⇒He gave <代词> <直接宾语>
⇒He gave me <直接宾语>
⇒He gave me <冠词><名词>
⇒He gave me a <名词>
⇒He gave me a book

<句子> → <主语><谓语><间接宾语><直接宾语>
<主语> → <代词>
<谓语> → <动词>
<间接宾语> → <代词>
<直接宾语> → <冠词> <名词>
<代词> → He
<代词> → me
<名词> → book
<冠词> → a
<动词> → gave

What is a Grammar?

- To define syntactic structure.文法是表示无穷字符串集的强有力的一种有限方式。
- A ***grammar*** G is a quadruple (V_T, V_N, S, P)
  - V_T is a finite set of ***terminal symbols***(有限的终极符集合)
  - V_N is a finite set of ***non-terminal symbols***(有限的非终极符集合)
  - S is ***start symbol***， $S \in V_N$
  - P is a set of ***production rules***(产生式的集合)，

    each production rule has following form:

    $\alpha \rightarrow \beta$， where α， $\beta \in (V_T \cup V_N)^*$

文法的分类

- 乔姆斯基(Chomsky)是美国当代有重大影响的语言学家
- www.chomsky.info
- 乔姆斯基于1956年建立形式语言体系，他把文法分成四种类型：0，1，2，3型，都由四部分组成，但对产生式的限制有所不同

文法的分类

- 0型(短语文法，图灵机)
  - 产生式形如： $\alpha \rightarrow \beta$
  - 其中：$\alpha \in (V_T \cup V_N)^*$且至少含有一个非终结符；$\beta \in (V_T \cup V_N)^*$
- 1型(上下文有关文法，线性界限自动机)
  - 产生式形如： $\alpha \rightarrow \beta$
  - 其中：$|\alpha| \leq |\beta|$，仅 $S \rightarrow \varepsilon$ 例外

文法的分类

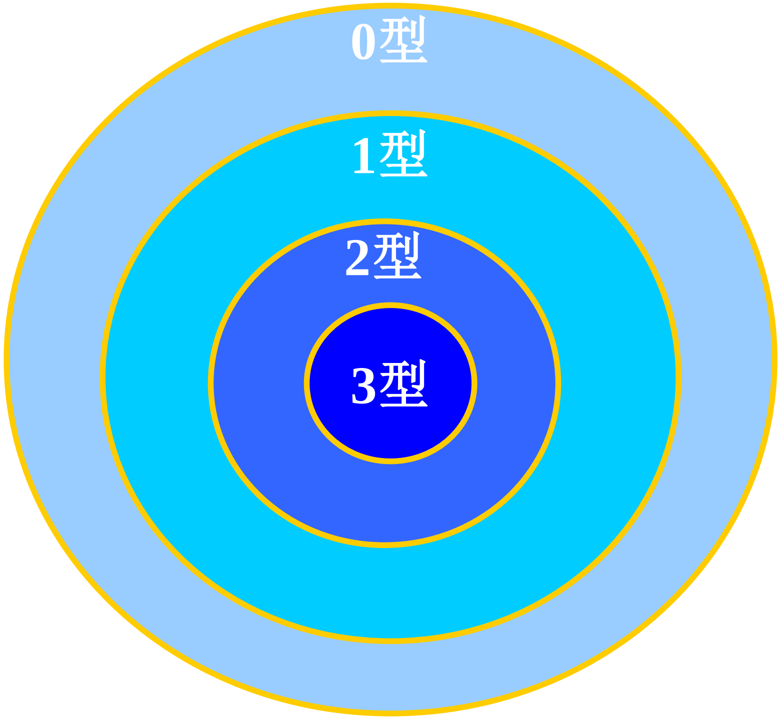
√ • 2型(上下文无关文法，非确定下推自动机)
  - 产生式形如： $A \rightarrow \beta$
  - 其中：$A \in V_N$；$\beta \in (V_T \cup V_N)^*$

√ • 3型(正规文法，有限自动机)
  - 产生式形如： $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$
  - 其中： $\alpha \in V_T^*$；$A, B \in V_N$
  - 产生式形如： $A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$
  - 其中： $\alpha \in V_T^*$；$A, B \in V_N$

右线性文法

左线性文法

四种类型文法描述能力比较



正则文法与正则表达式

- 正则表达式(a|b)*abb
- 文法G（A_0）：$A_0 \to aA_0 | bA_0 | aA_1$

$$A_1 \to bA_2$$

$$A_2 \to bA_3$$

$$A_3 \to \varepsilon$$

上下文无关文法与正则文法

- $L=\{a^nb^n|n\geq 1\}$ 不能由正规文法产生，但可由上下文无关文法产生

  G(S)：

  $S \rightarrow aSb|\ ab$

- 计算思维的典型方法--递归
  - 问题的解决又依赖于类似问题的解决，只不过后者的复杂程度或规模较原来的问题更小
  - 一旦将问题的复杂程度和规模化简到足够小时，问题的解法其实非常简单

上下文无关文法与上下文有关文法

- $L=\{a^nb^nc^n \mid n\geq 1\}$ 不能由上下文无关文法产生，但可由上下文有关文法产生

G(S)：

$$
\begin{aligned}
S &\to aSBC \mid aBC \\
CB &\to BC \\
aB &\to ab \\
bB &\to bb \\
bC &\to bc \\
cC &\to cc
\end{aligned}
$$

$$
\begin{aligned}
&S \\
&\Rightarrow aSBC \\
&\Rightarrow aaSBCBC \\
&\Rightarrow aaaBCBCBC \\
&\Rightarrow aaaBBCCBC \\
&\Rightarrow aaaBBCBCC \\
&\Rightarrow aaaBBBCCC \\
&\Rightarrow aaabBBCCC \\
&\Rightarrow aaabbBCCC \\
&\Rightarrow aaabbbCCC \\
&\Rightarrow aaabbbcCC \\
&\Rightarrow aaabbbccC \\
&\Rightarrow aaabbbccc
\end{aligned}
$$

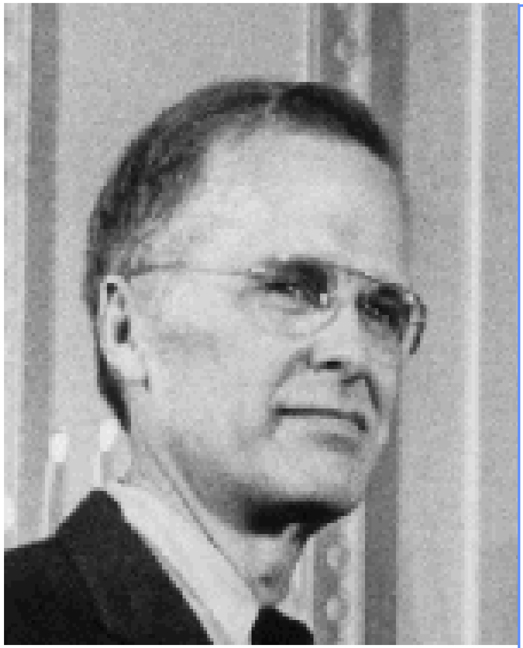
程序设计语言采用上下文无关文法

- 程序设计语言不是上下文无关语言，甚至不是上下文有关语言
- $L=\{\alpha c\alpha \mid \alpha\in\{a, b\}^*\}$ 不能由上下文无关文法产生，甚至连上下文有关文法也不能产生，只能由0型文法产生
  - 标识符引用
  - 过程调用过程中，“形–实参数的对应性"(如个数，顺序和类型一致性)
- 对于现今程序设计语言，在编译程序中，仍然采用上下文无关文法来描述其语言结构

Context Free Grammar

上下文无关文法

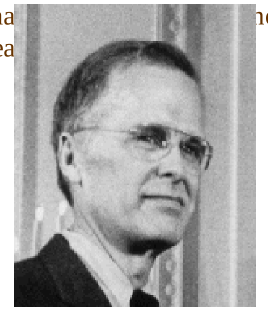
- 巴科斯范式(BNF)
  - “ → ”用“::="表示



John W. Backus

For profound, influential, and lasting contributions to the design of practical high-level programming systems, notably through his work on FORTRAN, and for seminal publication of formal procedures for the specification of programming languages.

巴科斯范式(BNF)

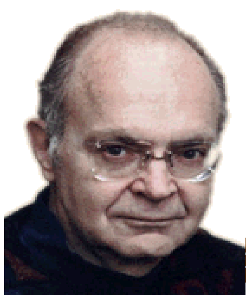


John W. Backus

首次在ALGOL 58中使用这种记号系统描述语法

Peter Naur

在ALGOL 60中发展并简化
命名Backus Normal Form



Donald E. Knuth

主张称为巴斯科-诺尔范式(Backus－Naur Form)
认为它不算是一种正规形式(Normal Form)

Context Free Grammar (CFG)

- 定义为四元组(V_T, V_N, S, P)
  - V_T是有限的终极符集合
  - V_N是有限的非终极符集合
  - S是开始符，$S \in V_N$
  - P是产生式的集合，且具有下面的形式：

$$A \rightarrow X_1X_2\ldots X_n$$

其中$A \in V_N$，$X_i \in (V_T \cup V_N)$，右部可空。

Example

V_T = {i, n, +, *, (,)}

V_N = {E, T, F }

S = E

P:

$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow F$

$T \rightarrow T * F$

$F \rightarrow (E)$

$F \rightarrow i$

$F \rightarrow n$

Context Free Grammar

Some Concepts

Some General Notations(Variables)

- **Normally**
  - **{A, B, …., Z} are used to represent non-terminal symbols;**
  - **{a, b, …., z} are used to represent terminal symbols;**
  - **{α, β, γ, … } are used to represent strings;**
  - **ε represents empty string;**

- **Derivation (直接推导):**

if there is a production $A \rightarrow \beta$, we can have $\alpha A \gamma \Rightarrow \alpha\beta\gamma$, where $\Rightarrow$ represents one step derivation (用 $A \rightarrow \beta$一步推导). We can say that **$\alpha\beta\gamma$** is <u>*derived*</u> from **$\alpha A \gamma$;**

- **$\Rightarrow$的含义是，使用一条规则，代替$\Rightarrow$左边的某个符号，产生$\Rightarrow$右端的符号串**

- $\alpha \Rightarrow^{+} \beta$: represents one or more steps derivation (α通过一步或多步可推导出 β)

- $\alpha \Rightarrow^{*} \beta$： 表示α 通过0步或多步可推导出β

Example

P:

(1) $E \rightarrow T$
(2) $E \rightarrow E + T$
(3) $T \rightarrow F$
(4) $T \rightarrow T * F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow i$
(7) $F \rightarrow n$

- $(E) \Rightarrow (E+T)$
- $E \Rightarrow^{+} (i+n)$
- $E+E+T \Rightarrow^{*} E+i+F$

- $G = (V_T, V_N, S, P)$
- 句型：if $S \Rightarrow^* \beta$， 则称符号串β为G的*句型*。我们用SF(G)表示文法G的所有句型的集合；
- Sentence(句子)：只包含终极符的句型被称为G的*句子*
- Language(语言):

  $L(G) = \{ u \mid S \Rightarrow^+ u, u \in V_T^* \}$

  the set of all sentences of G;

Example

P:
(1) $E \rightarrow T$
(2) $E \rightarrow E + T$
(3) $T \rightarrow F$
(4) $T \rightarrow T * F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow i$
(7) $F \rightarrow n$

- **句型: E, T, E+T, F, T*F, i, n, (E),**

  …….
- **句子: i, n, (i), (n), i+i, i+n, ……**
- **语言:{i, n, (i), (n), i+i, i+n , ……}**

- Leftmost(rightmost) derivation 最左（右）推导：如果进行推导时选择的是句型中的最左(右）非终极符，则称这种推导为最左(右)推导，并用符号$\Rightarrow_{lm}$（$\Rightarrow_{rm}$）表示最左（右）推导。
- 左（右）句型：

  用最左推导方式导出的句型，称为*<u>左句型</u>*，

  用最右推导方式导出的句型，称为*<u>右句型</u>*(*<u>规范句型</u>*)。

- conclusion:

each sentence has its rightmost or leftmost derivation（但对句型此结论不成立）

Why ???

Example

P:

(1) $E \rightarrow T$
(2) $E \rightarrow E + T$
(3) $T \rightarrow F$
(4) $T \rightarrow T * F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow i$
(7) $F \rightarrow n$

- 最左推导:

  $i+T*n+F \Rightarrow_{lm} i+F*n+F$

- 最右推导:

  $i+T*n+F \Rightarrow_{lm} i+T*n+i$

- 左句型: $i+i*F$
- 右句型: $E+(i)$
- 特例: $i+(T*n)$

i + i * n

P:

(1) E → T

(2) E → E + T

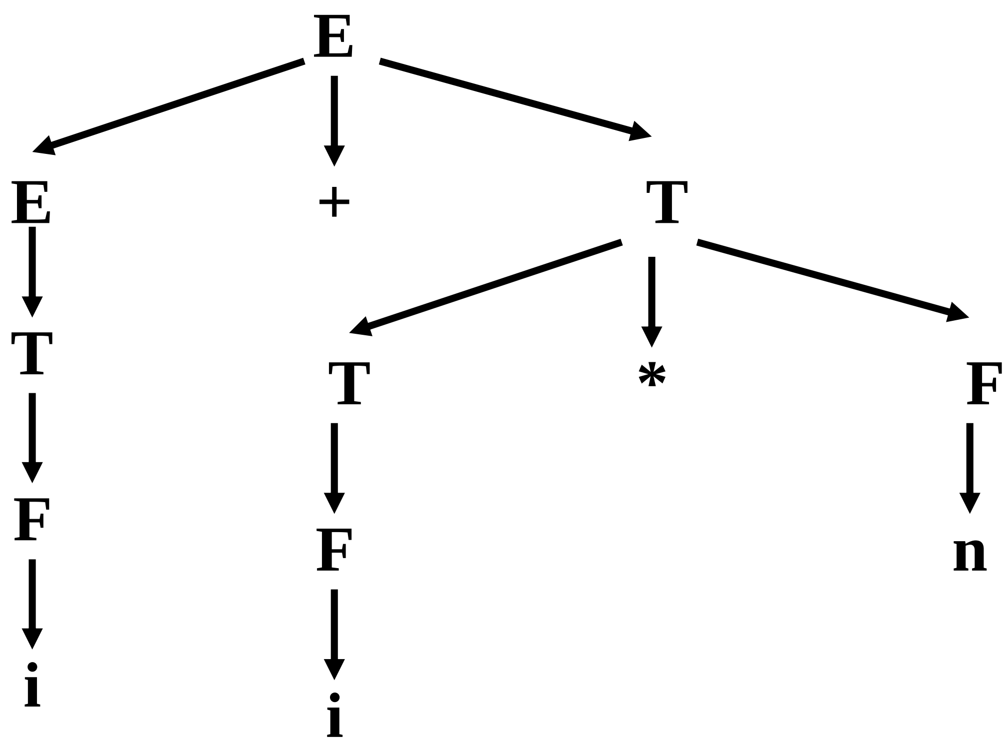
(3) T → F

(4) T → T * F

(5) F → (E)

(6) F → i

(7) F → n



P:

(1) $E \rightarrow T$

(2) $E \rightarrow E + T$

(3) $T \rightarrow F$

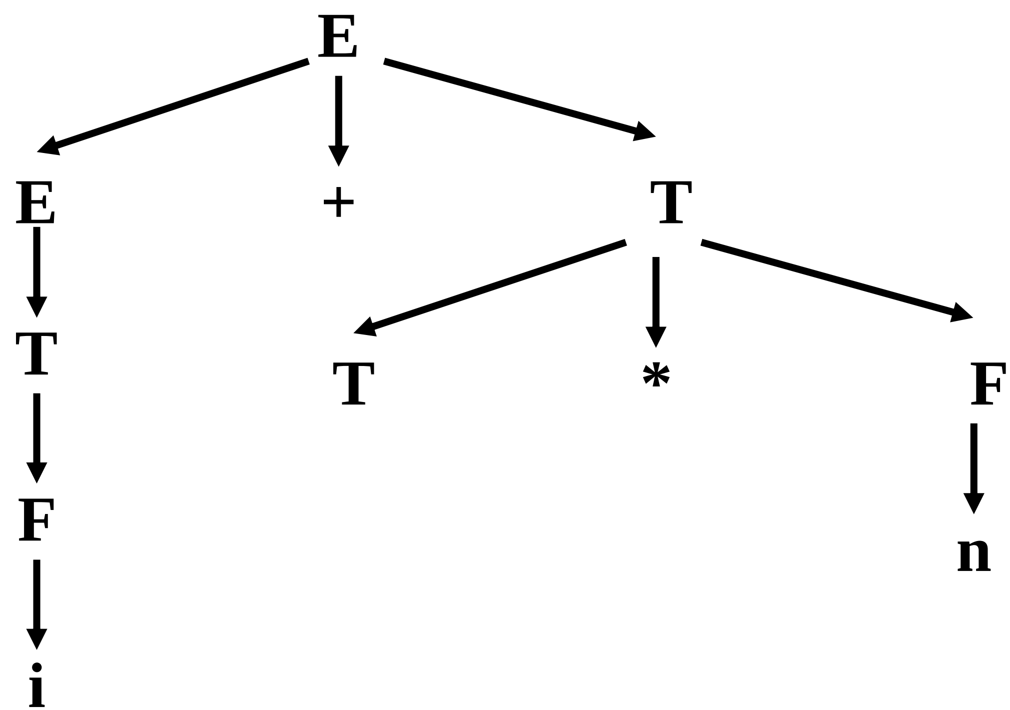
(4) $T \rightarrow T * F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow i$

(7) $F \rightarrow n$

i + T* n



Some Notes

- CFG (V_T,V_N, S, P) will be used to define syntactic structure of a programming language;
- Normally V_T will be set of tokens of the programming language;
- one terminal symbol might be one token, one token type, or one symbol representing certain structure;
- Non-terminal symbols act as intermediate representation of certain structure;
- Productions are rules on how to derive syntactic structure;

句型、句子和语言练习

- 设文法 G(A)：

  $A \rightarrow c \mid Ab$

  $G_1(A)$产生的语言是什么?

$$
\begin{aligned}
A &\Rightarrow c \\
A &\Rightarrow Ab \\
&\Rightarrow cb \\
A &\Rightarrow Ab \\
&\Rightarrow Abb \\
&\Rightarrow Abbb \\
&\Rightarrow ... \\
&\Rightarrow Ab...b \\
&\Rightarrow cb...b
\end{aligned}
$$

- 以c开头，后继若干个b
- $L(G_1)=\{c，cb，cbb，...\}$

句型、句子和语言练习

- 设文法G(S)：

$$S \rightarrow AB$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow bB \mid b$$

G(S)产生的语言是什么?

$L(G)=\{a^m b^n \mid m, n>0\}$

$$S \Rightarrow AB$$
$$A \Rightarrow a$$
$$A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow ... \Rightarrow a...aA \Rightarrow a...aa$$
$$B \Rightarrow b$$
$$B \Rightarrow bB \Rightarrow bbB \Rightarrow bbbB \Rightarrow ... \Rightarrow b...bB \Rightarrow b...bb$$

Example(1)

- **Arithmetic Expressions**

V_T = {*id*, *num*, *(*, *)*, +, -, *, /}

V_N = {Exp}

S = Exp

P:
Exp → Exp + Exp
Exp → Exp – Exp
Exp → Exp * Exp
Exp → Exp / Exp
Exp → *(*Exp*)*
Exp → *id*
Exp → *num*

Example(2)

- **Program**

V_T = {*VarDec, TypeDec, ConstDec, MainFun, FunDec* }

V_N = {Program, Dec, Decs}

S = Program

P: Program → Decs *MainFun* Decs

Dec→ ε

Dec → *VarDec* **Dec → *ConstDec***

Dec → *FunDec* **Dec→ *TypeDec***

Decs → Dec **Decs → Decs; Dec**

Extended BNF (扩展巴克斯范式)

- **Extended Notations**
  - **Optional** |

    $A \to \alpha \mid \beta \mid \gamma$

    $A \to \alpha \qquad A \to \beta \qquad A \to \gamma$
  - **Repetition** * or {}

    $A \to A\alpha \mid \beta$ (left recursive) $A \to \beta\,\alpha\,*$

    $A \to \alpha\,A \mid \beta$ (right recursive) $A \to \alpha\,*\,\beta$

Some Algorithm on CFG Transformation

Some algorithms on Grammar Transformation

- **文法等价变化: L(G1) = L(G2)**
- **增补文法(增广文法): the start symbol does not appear in the right part of any productions;**
  - **$Z \rightarrow S$**

Some algorithms on Grammar Transformation

- 消除公共前缀(left factoring)
- 公共前缀
  - $A \rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_m$
- 提取公因子
  - $A \rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m$
  - $A' \rightarrow \beta_1 \mid \ldots \mid \beta_n$

Some algorithms on Grammar Transformation

- 消除左递归(left recursion)
  - 直接左递归: $A \to A(\alpha_1 \mid \ldots \mid \alpha_n) \mid \beta_1 \mid \ldots \mid \beta_m$
  - 消除方法:

    $A \to (\beta_1 \mid \ldots \mid \beta_m) A'$

    $A' \to (\alpha_1 \mid \ldots \mid \alpha_n) A' \mid \varepsilon$

Example

P:

(1) $E \rightarrow T$

(2) $E \rightarrow E + T$

(3) $T \rightarrow F$

(4) $T \rightarrow T * F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow i$

(7) $F \rightarrow n$

$E \rightarrow E + T \mid T$

$\alpha_1 = + T$

$\beta_1 = T$

$E \rightarrow T E'$

$E' \rightarrow +T E' \mid \varepsilon$

$A \rightarrow A(\alpha 1 \mid \ldots \mid \alpha\ n) \mid \beta 1 \mid \ldots \mid \beta m$

$A \rightarrow (\beta 1 \mid \ldots \mid \beta m)A'$

$A' \rightarrow (\alpha 1 \mid \ldots \mid \alpha n)A' \mid \varepsilon$

Some algorithms on Grammar Transformation

- 消除左递归(left recursion)
  - 间接左递归:

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

1:S
2:A

$A \rightarrow Aba \mid b$

$A \rightarrow bA'$
$A' \rightarrow baA' \mid \varepsilon$

  - 消除方法:
    - Pre-conditions
    - Algorithm

Some Notes on these Algorithms

- **Applying one transformation algorithm might introduce other problems;**
- **All these algorithms need to be used together to get satisfactory result.**

§ 3 Context Free Grammar & Parsing

3.1 The Parsing Process （语法分析过程）

3.2 Context-free Grammars （上下文无关文法）

3.3 Parse Trees and Abstract Syntax Tree
（语法分析树和抽象语法树）

3.4 Ambiguous （二义性）

3.5 Syntax of Sample Language （简单语言的语法）

3.3 Parse Trees & Abstract Syntax Tree

- **<u>Problem</u>: Derivation(推导) is a way to construct a sentence from the start symbol;**
  - **Many derivation for the same sentence;**
  - **Not uniquely represent the structure of the sentence**
- ***<u>Parse tree</u>*: one way to represent the structure of a sentence;**

Example

P:

(1) E → T
(2) E → E + T
(3) T → F
(4) T → T * F
(5) F → (E)
(6) F → i
(7) F → n

sentence : i + i * n

Several derivations for it

Parse Tree

Parse Tree

- **A *<u>labeled tree</u>* for a CFG**
- **The *<u>root</u>* must be labeled with the start symbol;**
- **Each *<u>node</u>* has a symbol associated with it;**
- **Each *<u>leaf</u>* must be labeled with a terminal symbol ;**
- **For each node which is associated with a non-terminal symbol *<u>A</u>*, has n sons, from left to right they are associated with symbols B1, …, Bn, then there must be a production**

A → B1 … Bn

Abstract Syntax Tree

- **Problem with Parse tree**
  - **Includes much more than necessary nodes**
- **Abstract Syntax Tree**
  - **contains only those nodes necessary for compilation**

sentence : i + i * n

Information Science and Technology College of
Northeast Normal University

§ 3 Context Free Grammar & Parsing

3.1 The Parsing Process （语法分析过程）

3.2 Context-free Grammars （上下文无关文法）

3.3 Parse Trees and Abstract Syntax Tree

（语法分析树和抽象语法树）

3.4 Ambiguous （二义性）

3.5 Syntax of Sample Language （简单语言的语法）

3.4 Ambiguous Grammar

- **二义性文法**
  - **For a Grammar G, if there exists one sentence which has more than one *parse tree*, G is called *ambiguous Grammar*;**

语言的二义性

语言的二义性：一个语言是二义的，如果对它不存在无二义的文法

对于语言L，可能存在G和G’，使得L(G)=L(G’)=L，有可能其中一个文法为二义的，另一个为无二义的

John saw Mary in a boat.

Example

P:

Exp → Exp + Exp
Exp → Exp – Exp
Exp → Exp * Exp
Exp → Exp / Exp
Exp → (Exp)
Exp → id
Exp → num

id + id * id

Example

V_T = {*if*, *then*, *else*,*Exp*, *others*}
V_N = {If-stm, Stms}
S = Stms

P:
If-stm $\rightarrow$ *if Exp then* Stms
If-stm $\rightarrow$ *if Exp then* Stms *else* Stms
Stms $\rightarrow$ If-Stm | *others* | Stms ; Stms

sentence: *if Exp then if Exp then others else others*

Removing Ambiguity(消除二义性)

- 二义性文法是不可判定的;
- Solution 1:
  - Rewriting Grammar (equivalent)重写文法
- Solution 2:
  - Select one parse tree structure as preferred;

    (指定一个语法分析树为允许的)

- 计算思维的典型方法
  - 理论可实现 vs. 实际可实现
  - 理论研究重在探寻问题求解的方法，对于理论成果的研究运用又需要在能力和运用中作出权衡

§ 3 Context Free Grammar & Parsing

3.1 The Parsing Process （语法分析过程）

3.2 Context-free Grammars （上下文无关文法）

3.3 Parse Trees and Abstract Syntax Tree
（语法分析树和抽象语法树）

3.4 Ambiguous （二义性）

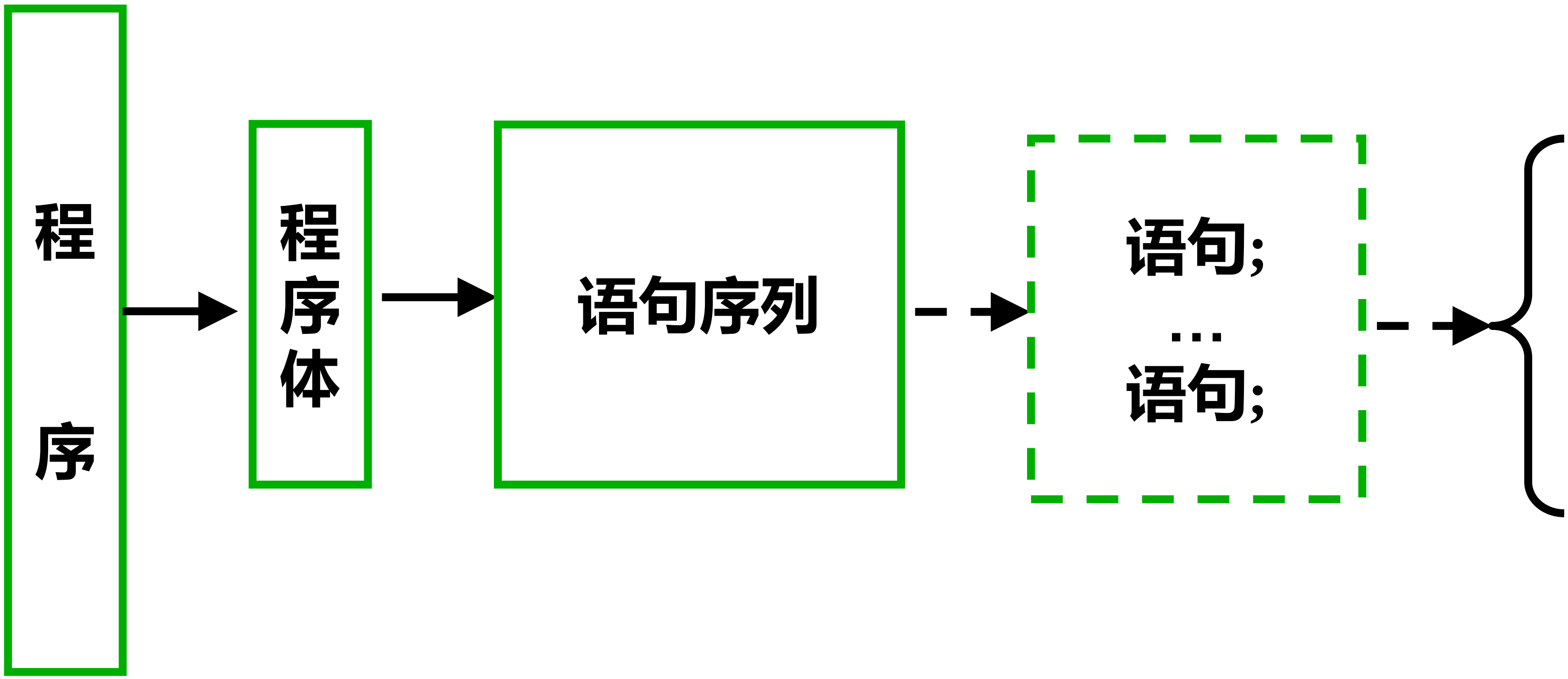
3.5 Syntax of Sample Language （简单语言的语法）

Information Science and Technology College of
Northeast Normal University

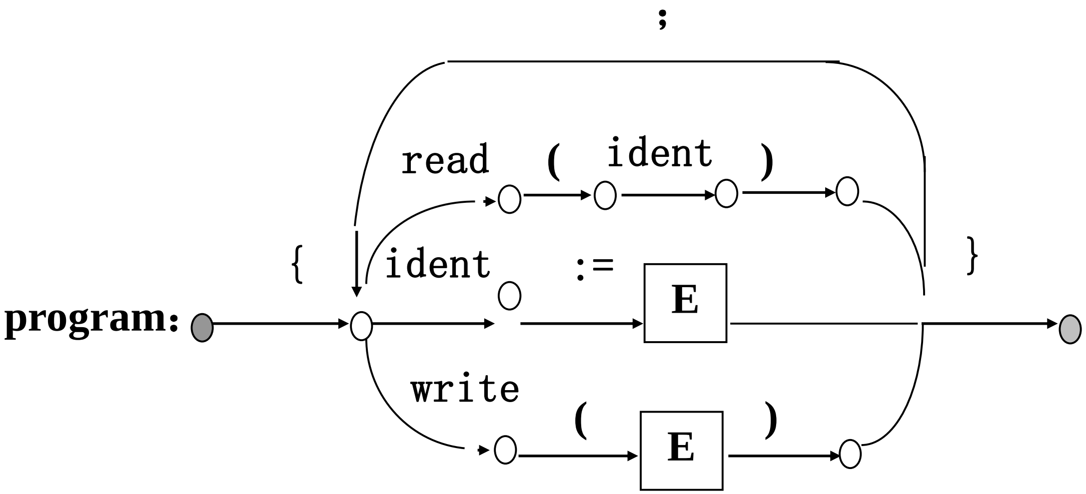
3.5 Syntax of Sample Language

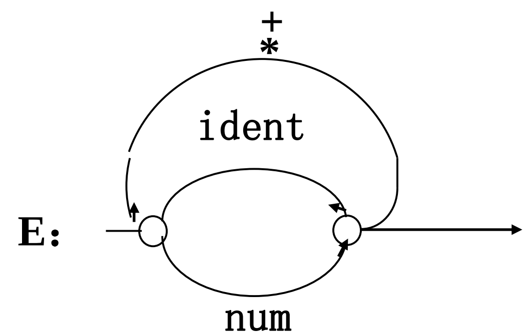
CFG for *C0* Programming Language

C0 Programming Language



The structure of program of *C0*





CFG for *C0*

V_T = { *id*, *num*, +, *, *ass*, {, *read*, *write*,(,), }}

V_N = { Prg, Stms, Stm, Assig, read-S, write-S, Exp,T,F,}

S = Prg

CFG for *C0*

Program: Prg → { Stms }

Stms → Stm;
Stm → Stm; Stms

Statement:
Stm → Assig | read-S | write-S
Assig → id *ass* Exp
read-S → *read* (id)
write-S → *write* (Exp)

CFG for *C0*

Expressions:

Exp $\rightarrow$ T

Exp $\rightarrow$ Exp+T

T $\rightarrow$ F

T $\rightarrow$ T*F

F $\rightarrow$ *id*

F $\rightarrow$ *num*

P:

(1) E $\rightarrow$ T

(2) E $\rightarrow$ E + T

(3) E $\rightarrow$ E - T

(4) T $\rightarrow$ F

(5) T $\rightarrow$ T * F

(6) T $\rightarrow$ T / F

(7) F $\rightarrow$ (E)

(8) F $\rightarrow$ i

(9) F $\rightarrow$ n

Homework

- **Define the syntax of following C statements with CFG**
  - **Variable Declaration**
  - **Assignment**
  - **If statement**
  - **While statement**
  - **Case statement**
- **Assuming that CFGs for expressions have already defined;**