Franklin's 13 virtues and Puritan ethics

Temperance
Silence
Order
Resolution

(5) Frugality: Make no expense but to do good to others or yourself; i.e., waste nothing.

节俭：不要做无益的花销，杜绝不必要的浪费。

静以修身，俭以养德

(6)Industry: Lose no time; be always employed in something useful; cut off all unnecessary actions.

勤勉：不浪费时间，时时刻刻都要用于正经事，戒除一切不必要的行动。

Lost wealth may be replaced by industry, lost knowledge by study, lost health by temperance or medicine, but lost time is gone forever.

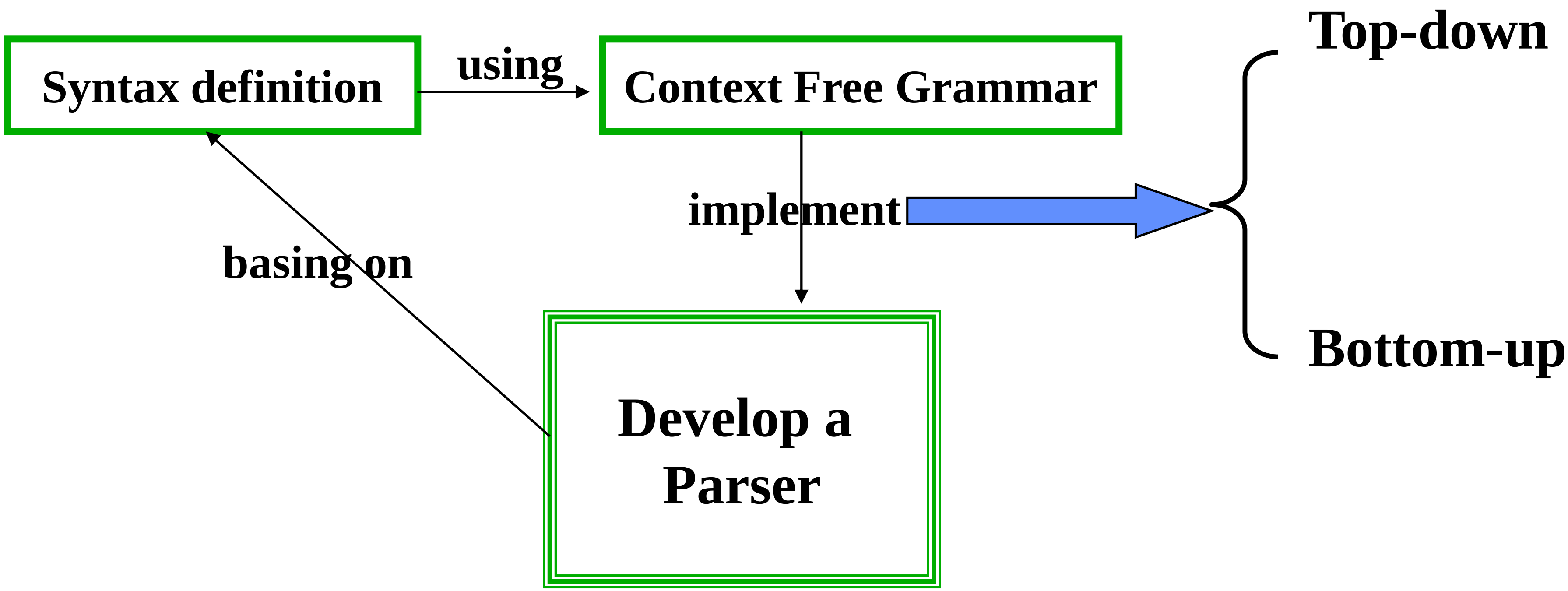
Information Science and Technology College of
Northeast Normal University

Compiling and Running of Program

Dr. Zheng Xiaojuan
Professor

October. 2019

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 Sample Language （简单语言的语法）

§ 4 Top-down Parsing

4.1 Overview of Top-down Parsing

4.2 Three Important Sets

4.3 Left Recursion Removal & Left Factoring

4.4 Recursive-Descent Parsing

4.5 LL(1) Parsing

4.1 Overview of Top-down Parsing

- **Problem:**
  - **Given CFG definition of the Syntax;**
  - **Check whether a program is of well defined structure;**
  - **Generate parse tree of the program;**
- **Main Idea**
  - **Read sequence of tokens (source program);**
  - **Start from the start symbol;**
  - **Try to find a proper sequence of derivations, resulting in the sequence of token of the source program;**

Example

P:

(1) $Z \rightarrow aBd$
(2) $B \rightarrow d$
(3) $B \rightarrow c$
(4) $B \rightarrow bB$

a	b	c	d

句型	输入	动作
Z	abcd	Derive (1) （指针所指）
aBd	abcd	Match （指针后移）
Bd	bcd	Derive(4)
bBd	bcd	Match
Bd	cd	Derive(3)
cd	cd	Match
d	d	Match
		Succeed

4.1 Overview of Top-down Parsing

- **Key problem:（derivation、match）**
  - **According to rest token sequence and the first non-terminal symbol, how to select a right production to make next derivation?**
- **Look ahead how many tokens?**
  - **Current token (look ahead one token)**
- **Calculate *<u>Predict set</u>* for a production $A \rightarrow \gamma$**
  - **$\{a \mid S \Rightarrow^{+} \alpha A \beta \Rightarrow \alpha \gamma \beta \Rightarrow^{+} \alpha a \beta'\}$, where $\alpha \in V_T^*$, S is start symbol.**

How to calculate the predict set for each production?

4.2 Three Important Sets

- ***First Set(first集)*** **for a string α with non-terminal and terminal symbols;**
  - **First(α)(non-terminal的*first集*）**
- ***Follow Set(follow集)*** **for a non-terminal symbol A;**
  - **Follow(A)**
- ***Predict Set(预测集)*** **for a production;**
  - **Predict(A→ α)**

First Set (first集)

- **Definition:**
  - **$First(\alpha) = \{a \mid \alpha \Rightarrow^* a\beta,\ a \in V_T\}$,**
  - **if $\alpha \Rightarrow^* \varepsilon$ then $First(\alpha) = First(\alpha) \cup \{\varepsilon\}$**

- **How to calculate First(α)?** （α是产生式右部）
  - **$\alpha = \varepsilon$,** **$First(\alpha) = \{\varepsilon\}$**
  - **$\alpha = a,\ a \in V_T$,** **$First(\alpha) = \{a\}$**
  - **$\alpha = A\beta$，$A \in V_N$,** **$First(\alpha) = First(A)$**
  - **$\alpha = A_1 A_2 \ldots\ldots A_{i-1} A_i \ldots\ldots A_n$，$A_1 \Rightarrow^* \varepsilon$**

$\alpha = A_1 A_2 \ldots\ldots A_{i-1} A_i \ldots\ldots A_n, \quad A_1 \Rightarrow^* \varepsilon$

For each symbol A, calculate First(A)

- $S_\varepsilon = \{A \mid A \Rightarrow^* \varepsilon , A \in V_N\}$
- $V_N=\{A_1, \ldots, A_n\}$, calculate First(A_i)

(1)初始化, First(A_i) ={};

(2)for i =1 to n 对于A_i的每个产生式,

- if $A_i \rightarrow \varepsilon$, First($A_i$) = First($A_i$) $\cup$ {ε};
- if $A_i \rightarrow Y_1 \ldots Y_m$, $\{Y_1 ,\ldots ,Y_m\} \subseteq S_\varepsilon$,
  First(A_i) = First(A_i) $\cup$ First(Y_1) $\cup$ ….. $\cup$ First(Y_m)
- if $A_i \rightarrow Y_1 \ldots Y_m$, $\{Y_1 ,\ldots ,Y_{j-1}\} \subseteq S_\varepsilon$, $Y_j \notin S_\varepsilon$
  First(A_i) = {First(A_i) $\cup$ {First(Y_1) $\cup$ ….. $\cup$ First(Y_{j-1})-{ε}} $\cup$ First(Y_j)

(3) Repeat (2) until 每个First(A_i)没有变化(收敛).

Example

$S\varepsilon = \{E', T'\}$

P:

(1) $E \rightarrow TE'$

(2) $E' \rightarrow + TE'$

(3) $E' \rightarrow \varepsilon$

(4) $T \rightarrow FT'$

(5) $T' \rightarrow * F T'$

(6) $T' \rightarrow \varepsilon$

(7) $F \rightarrow (E)$

(8) $F \rightarrow i$

(9) $F \rightarrow n$

E	{i, n , (}
E'	{ + , ε }
T	{ i, n , (}
T'	{ *, ε }
F	{ i, n , (}

Example

P:

(1) E → TE’
(2) E’ → + TE’
(3) E’ → ε
(4) T → FT’
(5) T’ → * F T’
(6) T’ → ε
(7) F → (E)
(8) F → i
(9) F → n

E	{i, n , (}
E’	{ + , ε }
T	{ i, n , (}
T’	{ *, ε }
F	{ i, n , (}

First(E’T’E) =

First(T’E’) =

4.2 Three Important Sets

- ***First Set(first集)*** **for a string α with non-terminal and terminal symbols;**
  - **First(α)**
- ***Follow Set(follow集)*** **for a non-terminal symbol A;**
  - **Follow(A)**
- ***Predict Set(预测集)*** **for a production;**
  - **Predict(A→ α)**

Follow Set (follow集)

- **Definition:**
  - **Follow(A) = $\{a \mid S \Rightarrow^{+} \ldots Aa \ldots,\ a \in V_T\}$,**
  - **if $S \Rightarrow^{+} \ldots A$, then Follow(A)= Follow(A) $\cup$ {#}**
  - **# represents the end of the string**

(0) Z → cAb

(1) A →a的First(A)？

(2) A → ε的First(A)？

Z → cAb ⇒ cab?通过求产生式（1）First(A) =a确定选择产生式（1）

Z → cAb ⇒ cb? 通过求产生式（2） Follow (A) =b确定选择产生式（2）

Follow Set (follow集)

- **How to calculate Follow(A), $A \in V_N$**

  (1)初始化, $\forall A \in V_N$, Follow(A) = { }

  (2)Follow(S) = {#}

  (3)对于每个产生式$A \to \alpha$

  - **If there is no non-terminal symbol in α, skip;**
  - **If $\alpha = \beta B \gamma$, $B \in V_N$, Follow(B) = Follow(B) $\cup$ (First(γ)-{ε})**
  - **If $\varepsilon \in$ First(γ), Follow(B) = Follow(B) $\cup$ Follow(A)**
  - **If $\alpha = \beta B$, Follow(B) = Follow(B) $\cup$ Follow(A)**

  (4) Repeat (3) until all follow sets do not change any more;

$Z \Rightarrow cAb$

$\Rightarrow c\,\beta B b$

$A \to \beta B$

Example

P:

(1) E → TE’

(2) E’ → + TE’

(3) E’ → ε

(4) T → FT’

(5) T’ → * F T’

(6) T’ → ε

(7) F → (E)

(8) F → i

(9) F → n

E	{i, n , (}
E’	{ + , ε }
T	{ i, n , (}
T’	{ *, ε }
F	{ i, n , (}

E	{#,)}
E’	{#,)}
T	{+,), #}
T’	{+,), #}
F	{*, +,), #}

4.2 Three Important Sets

- ***First Set(first集)*** **for a string α with non-terminal and terminal symbols;**
  - **First(α)**
- ***Follow Set(follow集)*** **for a non-terminal symbol A;**
  - **Follow(A)**
- ***Predict Set(预测集)*** **for a production;**
  - **Predict(A→ α)**

Predict Set (predict集)

- **Definition:**
  - **Predict(A → α) = first(α), if ε ∉ first(α);**
  - **Predict(A → α) = (first(α)- ε) ∪ follow(A),**
    if ε ∈ first(α);
  - **Predict(A → ε) = follow(A),**

Example

P:

(1) E → TE' → First(TE')={i, n,(}

(2) E' → + TE' → First(+TE')={+}

(3) E' → ε → Follow(E')={#,)}

(4) T → FT' → First(FT')={i,n,(}

(5) T' →* F T' → First(*FT')={*}

(6) T' → ε → Follow(T')={),+, # }

(7) F → (E) → First((E))={ (}

(8) F → i → First(i)={i}

(9) F → n → First(n)={n}

first集

E	{i, n , (}
E'	{ + , ε }
T	{ i, n , (}
T'	{ *, ε }
F	{ i, n , (}

follow集

E	{#,)}
E'	{#,)}
T	{+,), #}
T'	{+,), #}
F	{*, +,), #}

Top-down parsing

- **Precondition for Top-down parsing (look ahead one symbol) 自顶向下语法分析方法的前提条件**
  - **$G = (V_T, V_N, S, P)$**
  - **For any $A \in V_N$,**
  - **For any two productions of A,**
  - **$Predict(A \to \alpha 1) \cap Predict(A \to \alpha 2) = \Phi$**

  (同一个非终极符的任意两个产生式的predict集合互不相交)
- **这个条件保证:针对当前的符号和当前的非终极符,可以选择唯一的产生式来进行推导;**

Information Science and Technology College of
Northeast Normal University

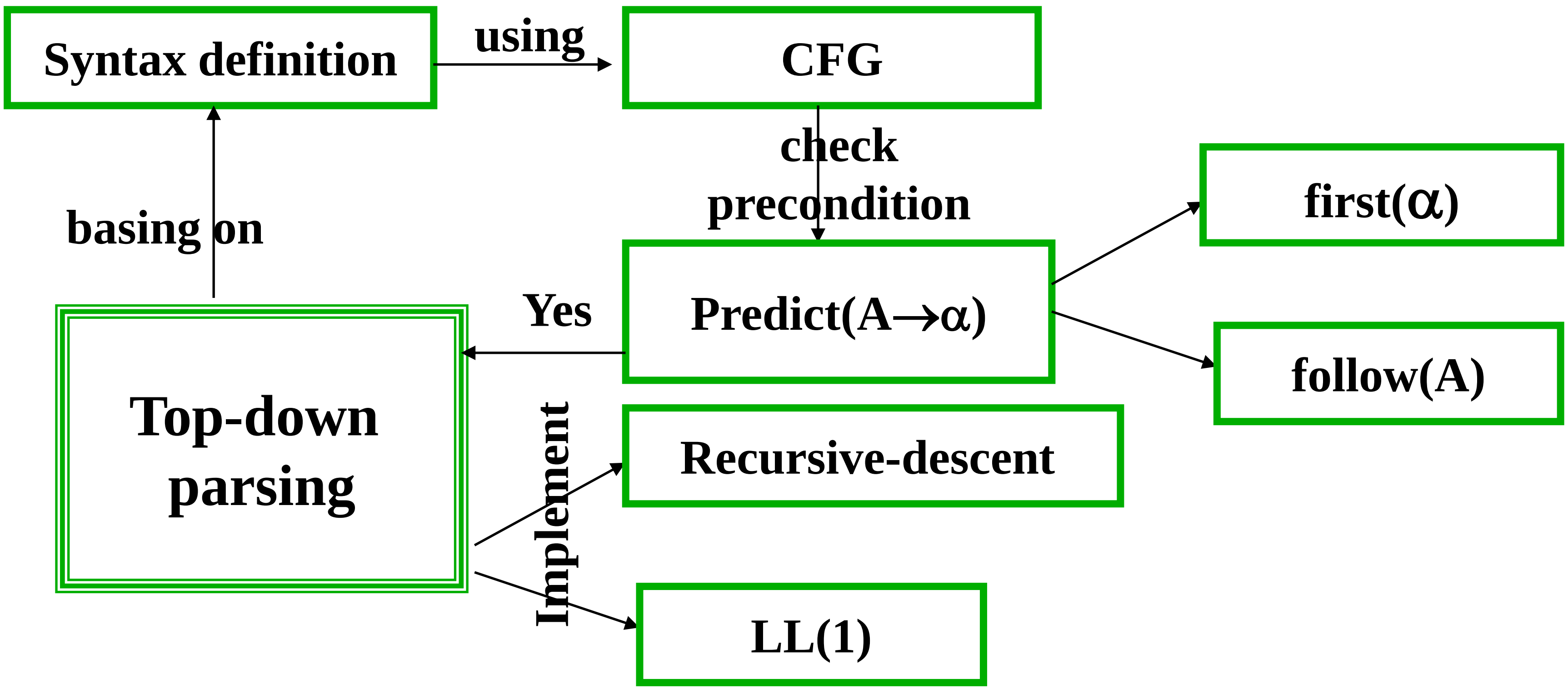
Example

P:	
(1) Z → aBd	{a}
(2) B → d	{d}
(3) B → c	{c}
(4) B → bB	{b}

a	b	c	d

句型	输入	动作
Z	abcd	Derive (1)
aBd	abcd	Match
Bd	bcd	Derive(4)
bBd	bcd	Match
Bd	cd	Derive(3)
cd	cd	Match
d	d	Match
		Succeed

Knowledge Relation Graph



Information Science and Technology College of
Northeast Normal University

4.3 Left Recursion Removal & Left Factoring

- 消除左递归
  - 直接左递归
  - 间接左递归
- 消除公共前缀
  - 提取公共前缀

4.4 Recursive-Descent Parsing

- **Main Idea:**
  - **For each non-terminal symbol, generate one parsing sub-routine(语法分析子过程）according to its productions;**
- **递归下降法:**
  - **针对满足*分析条件*的CFG;**
  - **“递归”:由于产生式的递归导致分析子程序的递归;**
  - **“下降”：自项向下分析;**
- **Disadvantages:**
  - **Restriction on CFG is too strict;**
  - **inefficient because of too many function calls;**

4.4 Recursive-Descent Parsing

- **The goal of parsing**
  - ***Check* whether the input string belongs to the language of CFG;**
- **Two actions**
  - **match(a): to check current symbol, if match, read next symbol;**
  - **Derivation: select the production**

Example

P:	
(1) Z → aBd	{a}
(2) B → d	{d}
(3) B → c	{c}
(4) B → bB	{b}

a	b	c	d

```
Z ( )
{
if token = a
 {match(a);
  B( );
  match(d);
  }
else error( );
}
```

```
B ( )
{
 case token of
  d: match(d);break;
  c: match(c); break;
  b:{ match(b);
        B( ); break;}
  other: error( );
}
```

```
main( ){ read(token); Z( )}
```

General Process

- **$G = (V_T, V_N, S, P)$**
- **Predefined function:**

  void match(a: V_T)
- **Global variable:**

  token: V_T
- **Input string: str**

- **For each $A \in V_N$,**
- **$A \rightarrow \alpha_1 | \ldots\ldots | \alpha_n$**

```
A( )
{ case token of
    Predict(A → α1):  SubR(α1)  ; break;
    ……
    Predict(A → αn): SubR(αn) ; break;
    other: error;
}
```

General Process

```
void match(a: V_T)
{
 if token == a
     token = readNext(str);
  else error();
}
```

SubR(α):

$\alpha = X_1 X_2 \ldots\ldots X_n$

If $X_i \in V_T$, match(X_i)
If $X_i \in V_N$, X_i();

SubR(α): $\alpha = \varepsilon$	{ }

Some Notes

- **Not real parsing program, but algorithm;**
- **More detailed issues need to be considered**
  - **Dealing with errors**
  - **Building parse tree**

How to deal with these issues?

Building Parse Tree

- **Data structure**
  - **ParseTree**

符号	儿子个数	儿子指针 →

- **Operations**
  - **ParseTree BuildRoot(symbol: V_T);**
  - **ParseTree BuildOneNode(symbol: $V_T \cup V_N$)**
  - **AddOneSon(father:*ParseTree, son:*ParseTree)**
  - **SetNum(Node:*ParseTree, n:int)**

Example

```
   *ParseTree        Z ( )
{     if token = a {
      T = BuildRoot(Z);    SetNum(T, 3);
      match(a);   A = BuildOneSon(a); AddOneSon(T, A);

       BB= B( );    AddOneSon(T, BB);

      match(d);  D = BuildOneSon(d); AddOneSon(T, D);
      return T;
      }  else  {error(); return nil;}
}
```

Example

```
 B ( )
{
 case token of
    d: match(d);break;
    c: match(c); break;
    b:{ match(b);
           B( ); break;}
    other: error();
}
```

You can do it yourself!

Assignment

- $G = \{V_T, V_N, S, P\}$
- $V_T = \{-\ ,\ (,\ ),\ id\}$
- $V_N = \{E, ET, V, VT\}$
- $S = E$
- $P = \{\ E \rightarrow -E$

$$E \rightarrow (E)$$
$$E \rightarrow V\ \ ET$$
$$ET \rightarrow -E$$
$$ET \rightarrow \varepsilon$$
$$V \rightarrow id\ VT$$
$$VT \rightarrow (E)$$
$$VT \rightarrow \varepsilon\quad \}$$

Write recursive-descent parsing program for G.

Assignment

- **According to the CFG of C0, write down recursive-desce parsing program.**

  (根据C0语言的上下文无关文法的定义，写出递归下降法程序.）

  - **(1) calculate predict sets for each production;**
  - **(2) check whether the grammar meets the precondition;**
  - **(3) if not, rewrite grammar;**
  - **(4) write function for each non-terminal symbol;**

Build parse tree!

§ 4 Top-down Parsing

4.1 Overview of Top-down Parsing

4.2 Three Important Sets

4.3 Left Recursion Removal & Left Factoring

4.4 Recursive-Descent Parsing

4.5 LL(1) Parsing

4.5 LL(1) Parsing

- **Main idea of LL(1) Parsing Method**
- **LL(1) Grammar (LL(1)文法)**
- **LL(1) Parsing Table (LL(1)分析表)**
- **LL(1) Parsing Engine (LL(1)分析驱动程序)**
- **LL(1) Parsing Process (LL(1)分析过程)**
- **LL(1) Parser Generator – LLGen、JavaCC (LL(1)分析程序的自动生成器)**

Main idea of LL(1) Parsing Method

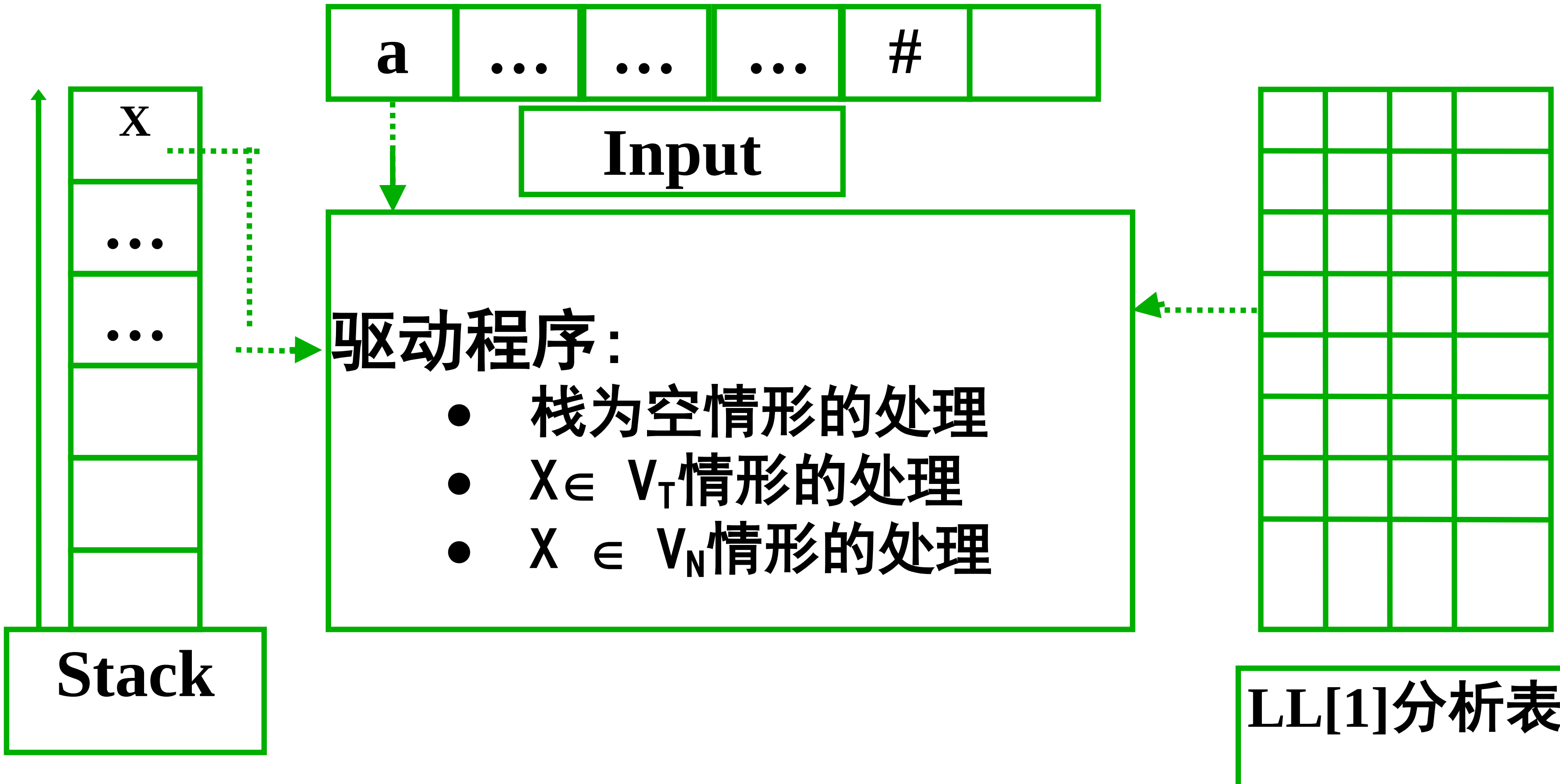
- **LL(1) Parsing**
  - **Left-to-right parsing, Left-most derivation, 1-symbol look ahead ;**
  - **Requires the same precondition(和递归下降法要求相同的前提条件)**
    - **$G = (V_T, V_N, S, P)$**
    - **For any $A \in V_N$,**
    - **For any two productions of A,**
    - **$Predict(A \rightarrow \alpha 1) \cap Predict(A \rightarrow \alpha 2) = \Phi$**

Main idea of LL(1) Parsing Method

- **LL(1) Parsing**
  - **LL(1) parsing table to record predict sets for each production; (LL(1)分析表)**
  - **A general engine(一个通用的驱动程序)**

Information Science and Technology College of
Northeast Normal University

LL(1) Parsing Mechanism



4.5 LL(1) Parsing

- **Main idea of LL(1) Parsing Method**
- **LL(1) Grammar (LL(1)文法)**
- **LL(1) Parsing Table (LL(1)分析表)**
- **LL(1) Parsing Engine (LL(1)分析驱动程序)**
- **LL(1) Parsing Process (LL(1)分析过程)**
- **LL(1) Parser Generator – LLGen、JavaCC (LL(1)分析程序的自动生成器)**

LL(1) Grammar

- **If a CFG *G* meets the precondition below, we will call *G* is a LL(1) Grammar;**

$G = (V_T, V_N, S, P)$
For any $A \in V_N$,
For any two productions of A,
$Predict(A \to \alpha_1) \cap Predict(A \to \alpha_2) = \Phi$

4.5 LL(1) Parsing

- **Main idea of LL(1) Parsing Method**
- **LL(1) Grammar (LL(1)文法)**
- **LL(1) Parsing Table (LL(1)分析表)**
- **LL(1) Parsing Engine (LL(1)分析驱动程序)**
- **LL(1) Parsing Process (LL(1)分析过程)**
- **LL(1) Parser Generator – LLGen (LL(1)分析程序的自动生成器)**

LL(1) Parsing Table (LL(1)分析表)

- **The purpose of LL(1) Parsing Table**
  - **According to <current non-terminal symbol, current input symbol>, decide that which production of current non-terminal symbol can be used to derive;**
  - **根据当前的非终极符和当前的输入符（输入流）决定用哪一个产生式来进行推导;**
- **If current non-terminal symbol is X, current input symbol is a, we can use $X\rightarrow\alpha$ if and only if $a\in$predict($X\rightarrow\alpha$);**

LL(1) Parsing Table (LL(1)分析表)

- **How to build LL(1) Parsing Table for a LL(1) Grammar?**
  - **For a LL(1) Grammar G = (V_T, V_N, S, P)**
  - **$V_T = \{a_1, \ldots, a_n\}$**
  - **$V_N = \{A_1, \ldots, A_n\}$**
  - **$LL(A_i, a_i) = [A_i \rightarrow \alpha]$, if $a_i \in predict(A_i \rightarrow \alpha)$**
  - **$LL(A_i, a_i)$ = error, if a_i not belong to the predict set of any production of A_i**

	a_1	…	a_n	#
A_1				
…	….	….	…	
A_n				

Example

P:

(1) Z → aBd
(2) B → d
(3) B → c
(4) B → bB

产生式	Predict集
(1)	{a}
(2)	{d}
(3)	{c}
(4)	{b}

	a	b	c	d	#
Z	(1)				
B		(4)	(3)	(2)	

Example

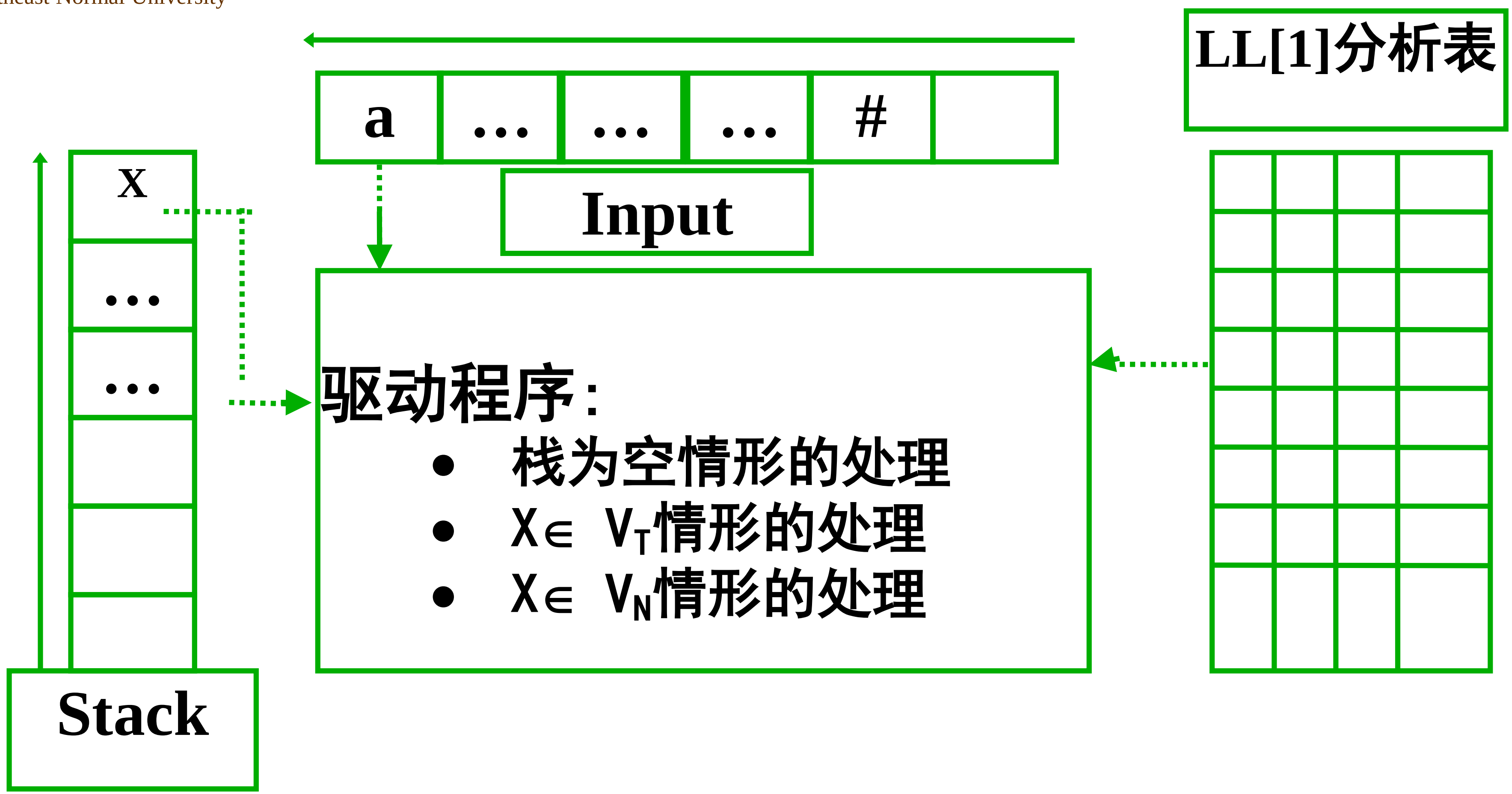
(1) E $\rightarrow$ TE’	{ i, n, (}
(2) E’ $\rightarrow$ + TE’	{+}
(3) E’ $\rightarrow$ ε	{#,)}
(4) T $\rightarrow$ FT’	{i,n,(}
(5) T’ $\rightarrow$ * F T’	{*}
(6) T’ $\rightarrow$ ε	{),+, # }
(7) F $\rightarrow$ (E)	{ (}
(8) F $\rightarrow$ i	{i}
(9) F $\rightarrow$ n	{n}

	+	*	(	)	i	n	#
E			(1)		(1)	(1)	
E’	(2)			(3)			(3)
T			(4)		(4)	(4)	
T’	(6)	(5)		(6)			(6)
F			(7)		(8)	(9)	

4.5 LL(1) Parsing

- **Main idea of LL(1) Parsing Method**
- **LL(1) Grammar (LL(1)文法)**
- **LL(1) Parsing Table (LL(1)分析表)**
- **LL(1) Parsing Engine (LL(1)分析驱动程序)**
- **LL(1) Parsing Process (LL(1)分析过程)**
- **LL(1) Parser Generator – LLGen 、JavaCC (LL(1)分析程序的自动生成器)**

LL(1) Parsing Engine 分析驱动程序



LL(1) Parsing Engine

- **Configuration(格局): <stack, input>**
- **可能的情形**
  - **栈为空情形的处理< , a> : succeed, if a=#, else error;**
  - **X$\in V_T$情形的处理: <a, a>, match; else error;**
  - **X $\in V_N$情形的处理: <X, a>, derive if there is a production X$\to\alpha$ such that a$\in$predict(X$\to\alpha$), else error;**

LL(1) Parsing Engine

[1] 初始化： Stack := [ ]；Push(S)；

[2] 读下一个输入符： Read(a)；

[3] 若当前格局是(, #)，则成功结束；否则转下；

[4] 设当前格局为（..... X, a.....)，则

- 若 $X\in V_T$ & X= a，则 { Pop(1)； Read(a)；goto [3] }
- 若 $X\in V_T$ & X≠ a，则 Error；
- 若 $X \in V_N$，则：

if LL(X, a)=$X\rightarrow Y_1\ Y_2 \ldots\ldots Y_n$

then { Pop(1)；Push(Y_n,.....,Y_1)；goto[3] }

else Error

Building Parse Tree During LL(1)

[1] 初始化： Stack := [ ];

root=BuildOneNode(S); Push(S, root)；

[2] 读下一个输入符： Read(a)；

[3] 若当前格局是(, #)，则成功结束；否则转下；

[4] 设当前格局为（..... X, a.....)，则

- 若X∈ V_T & X= a ，则 { Pop(1)； Read(a)；goto [3] }
- 若X∈ V_T & X≠ a ，则 Error；
- 若X ∈ V_N ，则：

if LL(X, a)=X→Y_1 Y_2 ...... Y_n

then { (X, ptr) = Pop(1);

for i=n to 1 { p[i] = BuildOneNode(Yi), Push(Y_i, p[i])； }

AddSons(ptr, p, n);

goto[3] }

else Error

4.5 LL(1) Parsing

- Main idea of LL(1) Parsing Method
- LL(1) Grammar (LL(1)文法)
- LL(1) Parsing Table (LL(1)分析表)
- LL(1) Parsing Engine (LL(1)分析驱动程序)
- LL(1) Parsing Process (LL(1)分析过程)
- LL(1) Parser Generator – LLGen、JavaCC (LL(1)分析程序的自动生成器)

LL(1) Parsing Process

P:
(1) Z → aBd
(2) B → d
(3) B → c
(4) B → bB

	a	b	c	d	#
Z	(1)				
B		(4)	(3)	(2)	

a	b	c	d

a	c	c	d

Dealing with If-Then-Else

- **CFG for If-Then-Else**

V_T = {if, then, else, other, ;, exp}

V_N = {S, Stm, ElsePart}

P:

(1) S → Stm ;

(2) Stm → if exp then Stm ElsePart

(3) Stm → other

(4) ElsePart → else Stm

(5) ElsePart → ε

Predict set:

(1)	{if, other}
(2)	{if}
(3)	{other}
(4)	{else}
(5)	{;, else}

Dealing with If-Then-Else

- LL(1) parsing table

	if	then	else	other	exp	;	#
S	(1)			(1)			
Stm	(2)			(3)			
ElsePart			(4) (5)			(5)	

每个else与其前面未匹配的最近的then相匹配!

	if	then	else	other	exp	;	#
S	(1)			(1)			
Stm	(2)			(3)			
ElsePart			(4)			(5)	

Dealing with If-Then-Else

Information Science and Technology College of
Northeast Normal University

stack	input	action
S	if exp then if exp then other else other ;	**(1)**
Stm ;	if exp then if exp then other else other;	**(2)**
if exp then Stm ElsePart;	**if exp then** if exp then other else other;	Match(3次)
Stm ElsePart;	if exp then other else other;	**(2)**
if exp then Stm ElsePart ElsePart;	**if exp then** other else other;	Match(3次)
Stm ElsePart ElsePart;	other else other;	**(3)**
other ElsePart ElsePart;	other else other;	Match
ElsePart ElsePart;	else other;	**(4)**

(1) S → Stm ;
(2) Stm → if exp then Stm ElsePart
(3) Stm → other
(4) ElsePart → else Stm
(5) ElsePart → ε

	if	then	else	other	exp	;	#
S	(1)			(1)			
Stm	(2)			(3)			
ElsePart			(4)			(5)	

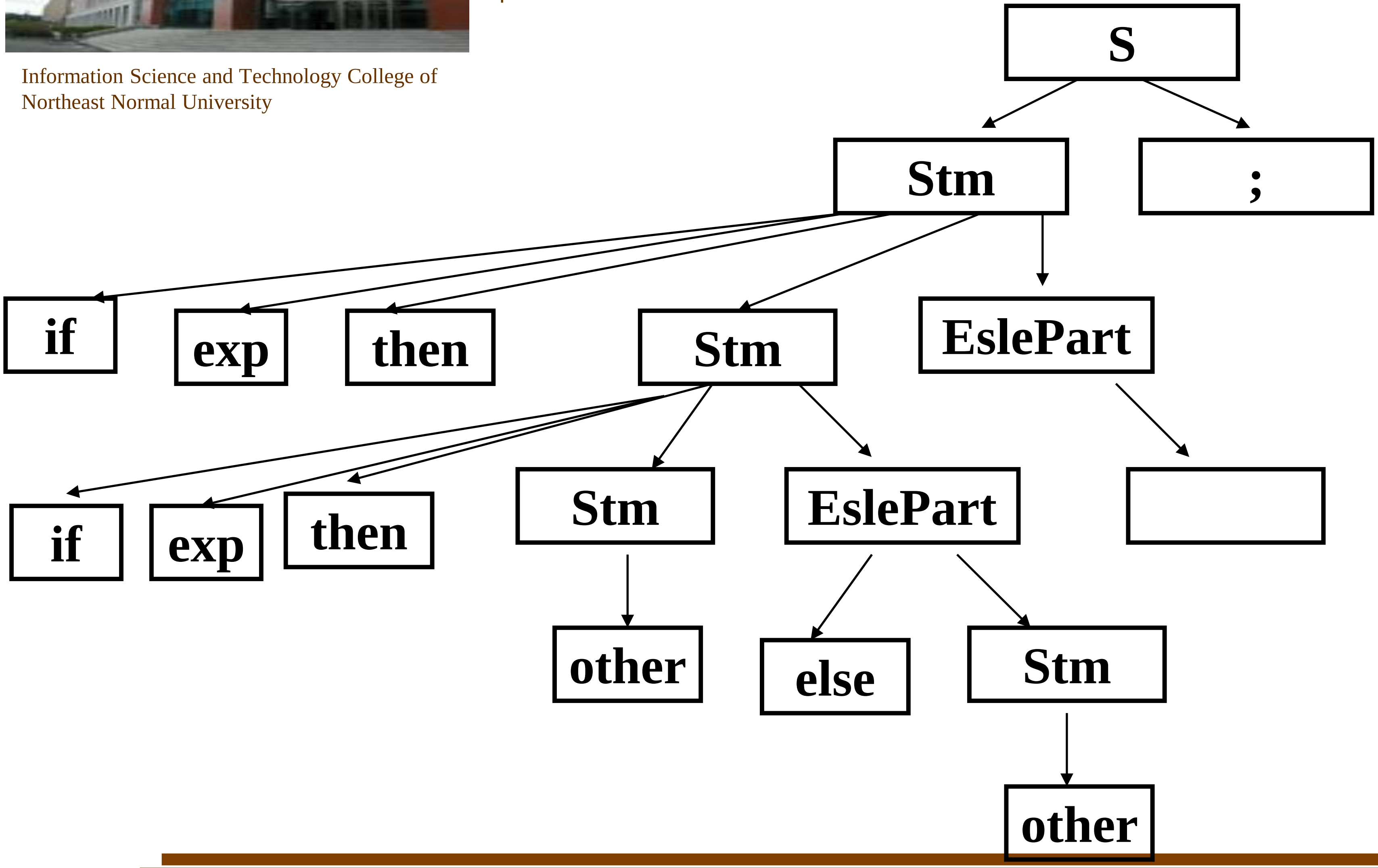
Information Science and Technology College of
Northeast Normal University

Dealing with If-Then-Else

stack	input	action
ElsePart ElsePart;	else other;	(4)
else Stm ElsePart;	**else** other;	match
Stm ElsePart;	other;	(3)
other ElsePart;	**other**;	match
ElsePart;	;	(5)
;	**;**	match
		succeed

(1) S → Stm ;
(2) Stm → if exp then Stm ElsePart
(3) Stm → other
(4) ElsePart → else Stm
(5) ElsePart → ε

Information Science and Technology College of
Northeast Normal University



4.5 LL(1) Parsing

- Main idea of LL(1) Parsing Method
- LL(1) Grammar (LL(1)文法)
- LL(1) Parsing Table (LL(1)分析表)
- LL(1) Parsing Engine (LL(1)分析驱动程序)
- LL(1) Parsing Process (LL(1)分析过程)
- LL(1) Parser Generator – LLGen 、JavaCC (LL(1)分析程序的自动生成器)

LL(1) Parser Generator – LLGen

<选择部分>

<常数定义>

<终极符定义>

<产生式定义>

LLGen

终极符/产生式个数

产生式右部长度

LL(分析表)

可导空串的符号表

文法中所有符号

非LL(1)表的冲突

JavaCC

- Java Compiler Compiler (JavaCC) – The Java Parser Generator
  - http://javacc.java.net/

JavaCC源程序 Parser.jj → JavaCC编译器 → LL(K)语法分析源程序(含词法分析) Parser.java

LL(K)语法分析源程序(含词法分析) Parser.java → Java编译器javac → LL(K)语法分析程序(含词法分析) Parser.class

输入串 → 运行LL(K)语法分析程序(含词法分析) java Parser.class → 分析结果

JavaCC

- <parser_name>.java
- <parser_name>Constants.java
- <parser_name>TokenManager.java
- ParseException.java
- SimpleCharStream.java
- Token.java
- TokenMgrError.java

§ 4 Top-down Parsing

4.1 Overview of Top-down Parsing

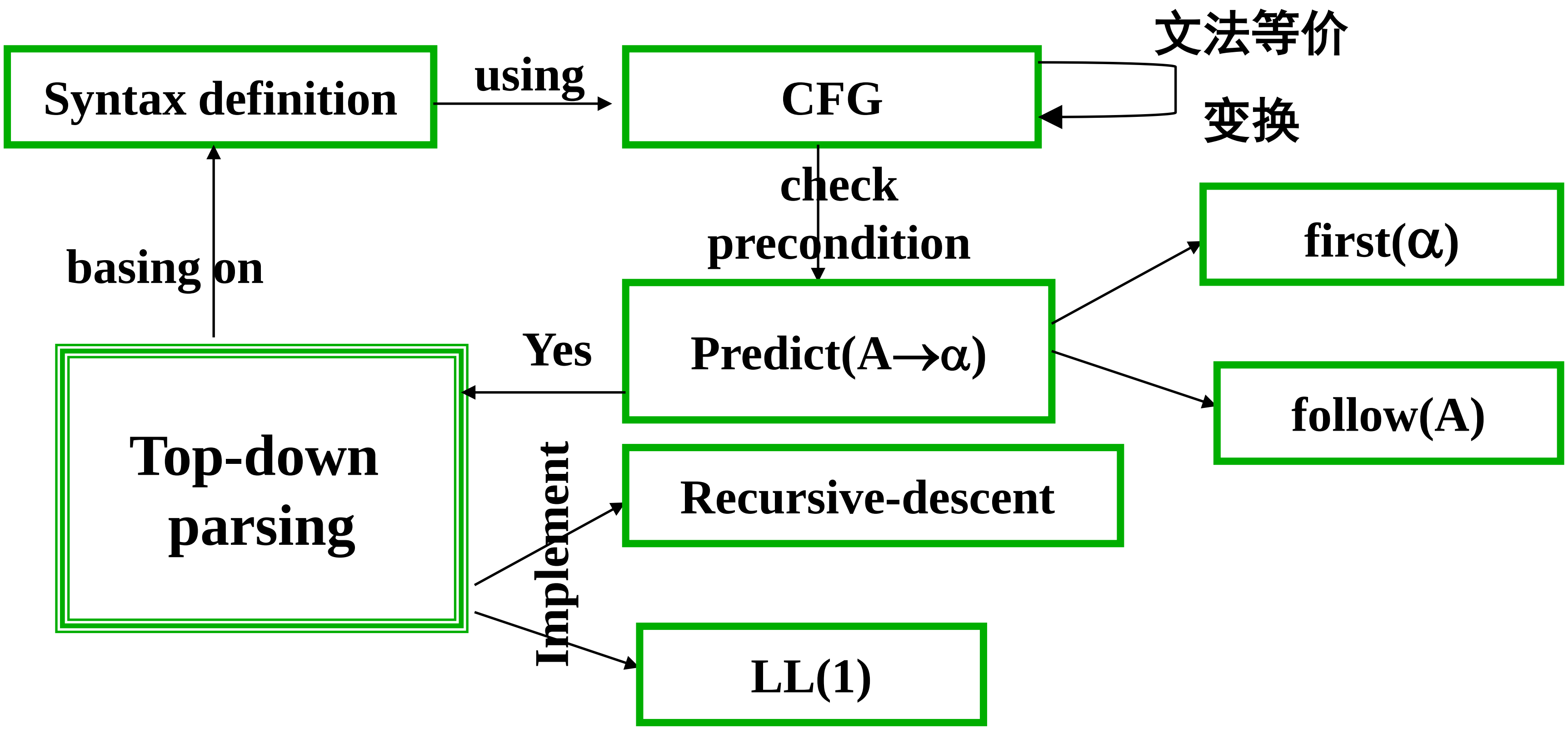
4.2 Three Important Sets

4.3 Left Recursion Removal & Left Factoring

4.4 Recursive-Descent Parsing

4.5 LL(1) Parsing

Knowledge Relation Graph



Summary

- **What is the *<u>main idea of Top-down parsing</u>*?**
- **What is the *<u>precondition</u>* of recursive-descent and LL(1) parsing?**
- **The *<u>definitions of three sets</u>* ?**
- **For a given CFG, *<u>calculate three sets</u>*?※**
- **The *<u>main idea of recursive-descent parsing</u>*?**

Summary

- **For a given CFG, *develop recursive-descent parser*?**
  - ***Calculate predict set* for each production;**
  - ***Check* whether meets the *precondition*;**
  - **If yes, *develop* function for each non-terminal symbol;**
  - ***Build parse tree* during parsing;**

Summary

- **The main idea of LL(1) parsing?**
- **LL(1) Grammar?**
- **The mechanism of LL(1) parsing?**
- **For a given CFG, develop LL(1) parser?**
  - **Calculate predict set for each production;**
  - **Check whether meets the precondition;**
  - **If yes, generate LL(1) parsing table;**
  - **LL(1) parser = LL(1) parsing table + LL(1) parsing engine**
- **Give the LL(1) parsing process?**

Summary

- **The *main idea of LL(1) parsing*?**
- ***LL(1) Grammar*?**
- **The *mechanism of LL(1) parsing*?**
- **For a given CFG, develop LL(1) parser?**
  - **Calculate predict set for each production;**
  - **Check whether meets the precondition;**
  - **If yes, generate LL(1) parsing table;**
  - **LL(1) parser = LL(1) parsing table + LL(1) *parsing engine***
  - ***Build parse tree*;**
- **Give the LL(1) parsing process?**

Assignment

- $G = \{V_T, V_N, S, P\}$
- $V_T = \{-\ ,\ (,\ ),\ id\}$
- $V_N = \{E, ET, V, VT\}$
- $S = E$
- $P = \{\ E \rightarrow -E \qquad ET \rightarrow \varepsilon$

  $E \rightarrow (E) \qquad V \rightarrow id\ VT$

  $E \rightarrow V\ \ ET \qquad VT \rightarrow (E)$

  $ET \rightarrow -E \qquad VT \rightarrow \varepsilon\ \}$

(1)Generate LL(1) Parsing table;

(2) “-(id--id)”的分析过程;