Principles of Compiler Construction

warning：语法上接受
error：不接受

递归-》非递归，关键在于堆栈

Prof. Wen-jun LI

School of Computer Science and Engineering

lnslwj@mail.sysu.edu.cn

Lecture 4. Top-Down Parsing

1. Introduction to Parsing
2. Top-Down Parsing
3. Rewriting Grammars
4. Top-Down Parser with Backtracking
5. Recursive Descent Predictive Parser

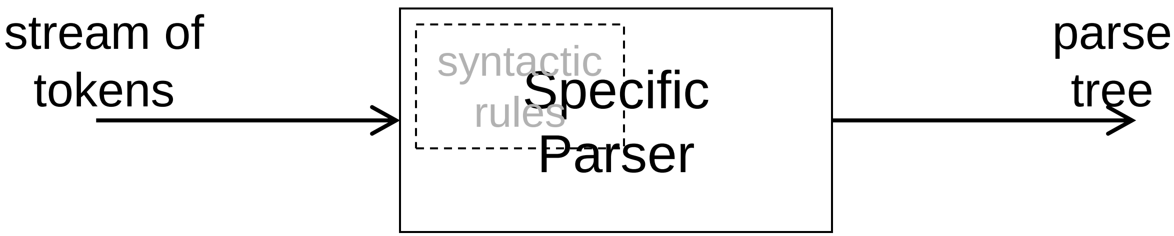
1. Introduction to Parsing 语法分析

- Parser: input, process, and output (IPO)

Two questions must be answered.

Structure of a Parser

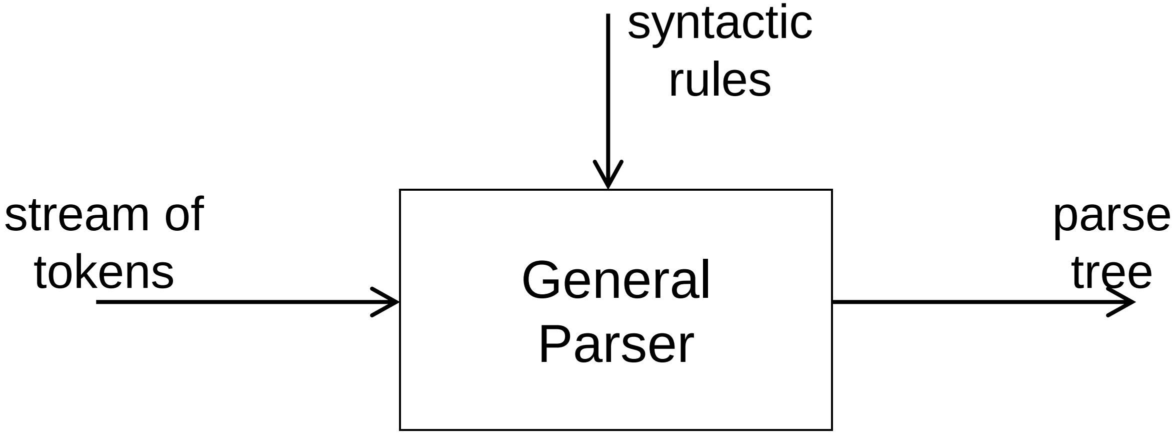
- Implicit syntactic rules
  含在代码里



Specific to some predefined language.

Structure of a Parser (cont')

- Explicit syntactic rules (interpretation model)

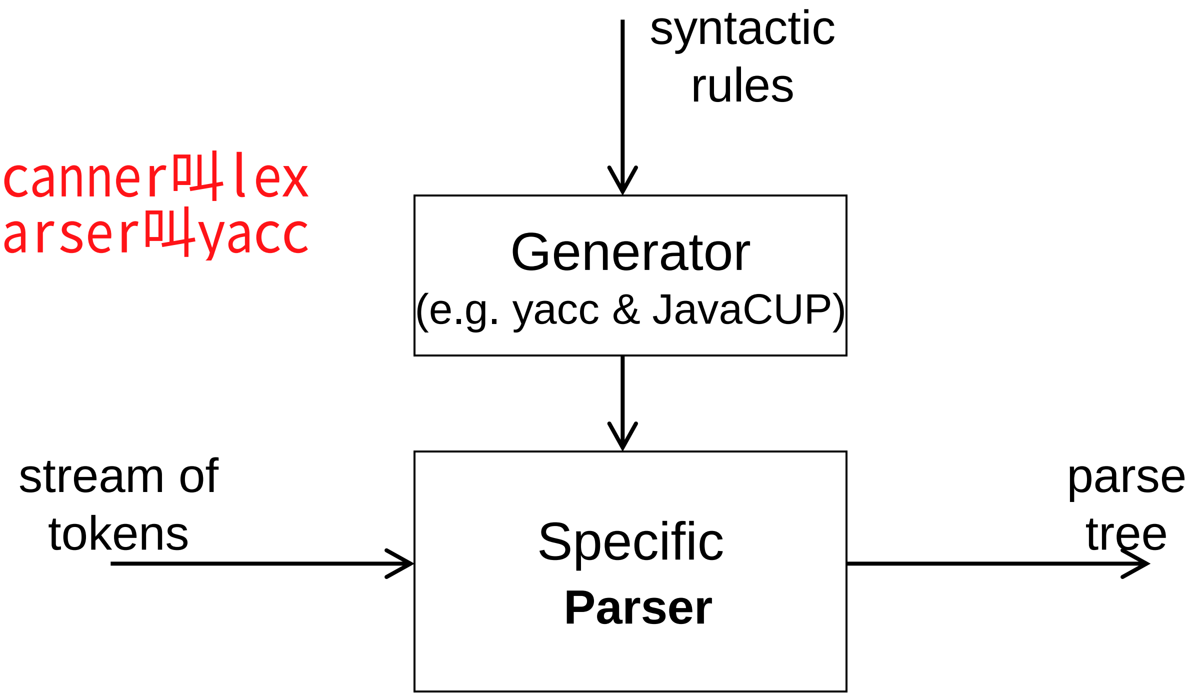


No hard-coding of language-specific code.

Structure of a Parser (cont')

- Explicit syntactic rules (compilation model)

scanner叫lex
parser叫yacc



Review

逻辑上

- The pipe between a parser and a scanner
  - Method invocation (procedure call)
  - Logical vs. physical
- Context-free grammars
  - Parse tree, derivation, and reduction
  - Ambiguity

Capability of Context-Freedom

- What languages it can generate?
  - $L_0 = \{a^n b^n \mid n \geq 1\}$
    - Abstraction of some problem in practice
- What languages it can not generate?
  - $L_1 = \{\omega c \omega \mid \omega \in (a \mid b)^* \wedge a, b, c \in \Sigma\}$ 变量先声明后使用
    - Abstraction of some problem in practice
    - How to solve the problem?
  - $L_2 = \{a^n b^m c^n d^m \mid n \geq 1 \wedge m \geq 1\}$ 函数参数个数
    - Abstraction of some problem in practice
    - How to solve the problem?

2. Top-Down Parsing

- Parsing strategies
  - Top-down parsing
    - How to choose a unique production in multiple candidates?
  - Bottom-up parsing
    - How to find the handle in a sentential form?

Top-Down Parsing: Motivation

- A motivating example 都是tokens

$$
\begin{array}{lcl}
\mathit{type} & \rightarrow & \mathit{simple} \\
 & | & \uparrow\ \mathbf{id} \\
 & | & \mathbf{array}\ [\ \mathit{simple}\ ]\ \mathbf{of}\ \mathit{type} \\
\mathit{simple} & \rightarrow & \mathbf{integer} \\
 & | & \mathbf{char} \\
 & | & \mathbf{num}\ \underset{..}{\mathbf{dotdot}}\ \mathbf{num}
\end{array}
$$

枚举

  - **array** [1..10] **of** **integer**

Parsing Process (Initial)

*Derive with "type → **array [** simple **] of** type"*

type

$$
\begin{aligned}
type &\rightarrow simple \\
&\mid \ \hat{}\ \mathbf{id} \\
&\mid \ \mathbf{array}\ [\ simple\ ]\ \mathbf{of}\ type \\
simple &\rightarrow \mathbf{integer} \\
&\mid \ \mathbf{char} \\
&\mid \ \mathbf{num}\ \mathbf{dotdot}\ \mathbf{num}
\end{aligned}
$$

枚举

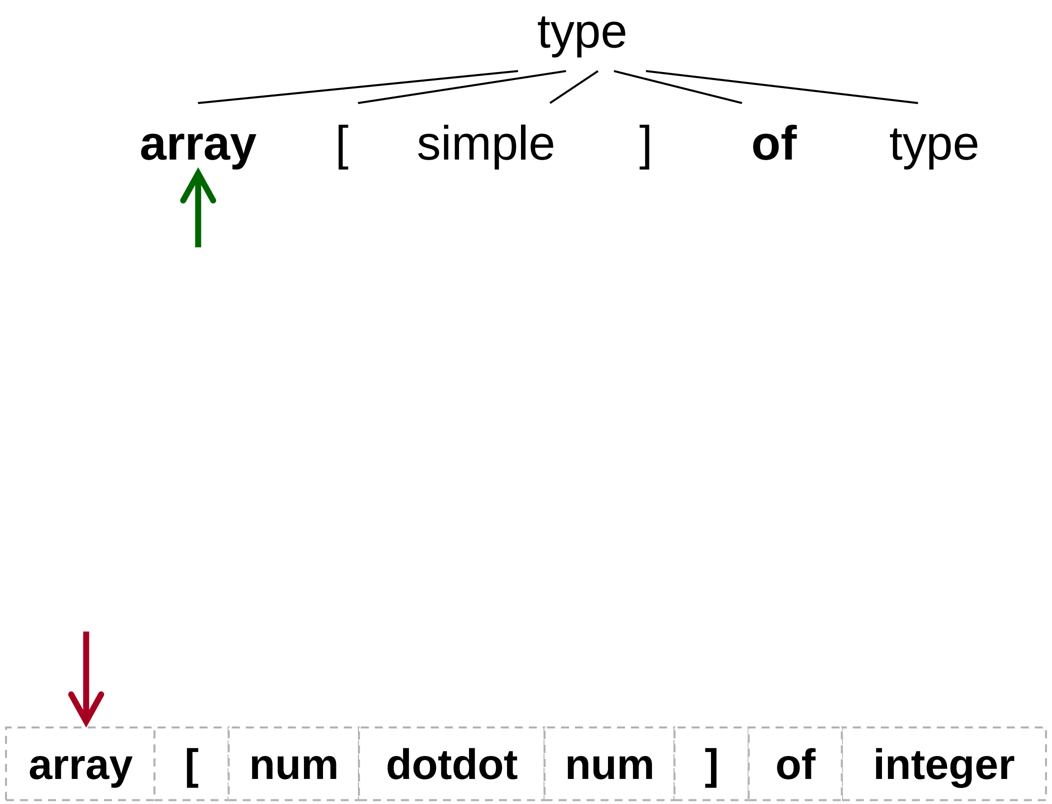
只找到唯一一个推导

lookahead

array	[	num	dotdot	num	]	of	integer

Parsing Process (Action 1)

Match !

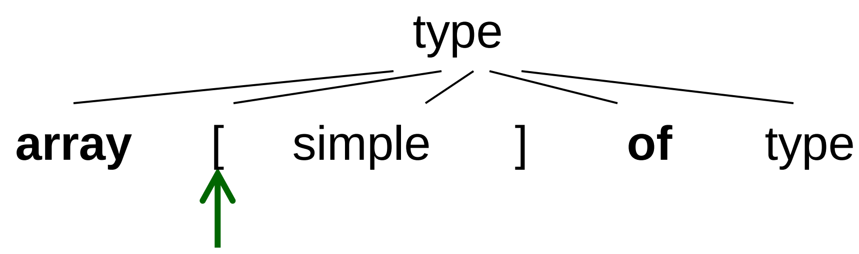


Parsing Process (Action 2)

Match !

acc error均有的操作

终结符不match就报错



array	**[**	**num**	**dotdot**	**num**	**]**	**of**	**integer**

Parsing Process (Action 3)

*Derive with "simple → **num dotdot num**"*

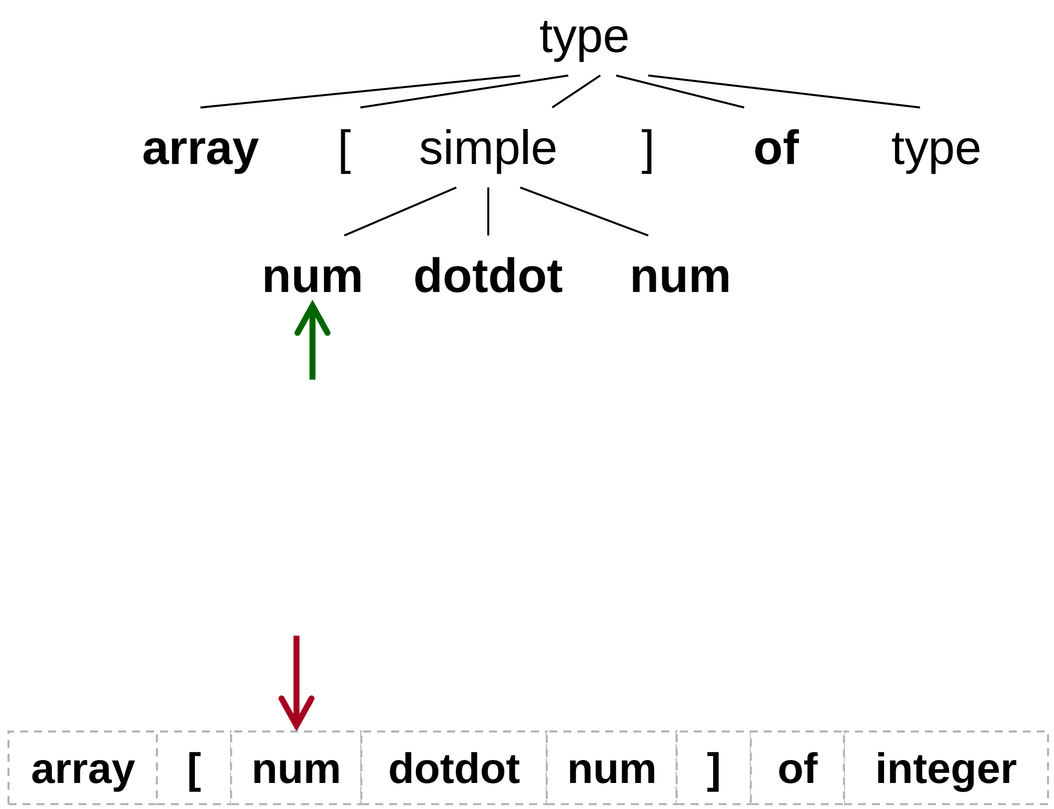
type

array [simple] **of** type

array	**[**	**num**	**dotdot**	**num**	**]**	**of**	**integer**

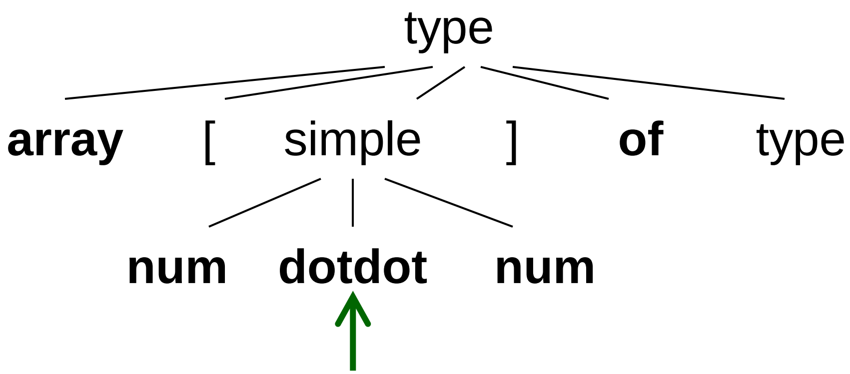
Parsing Process (Action 4)

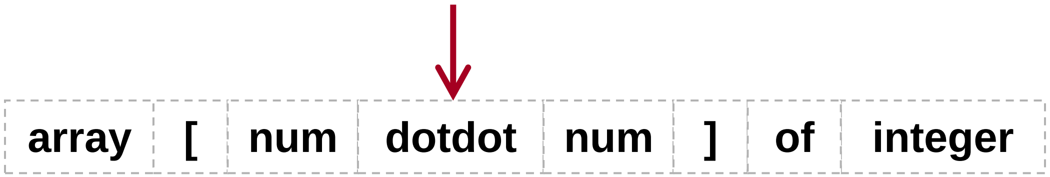
Match !



Parsing Process (Action 5)

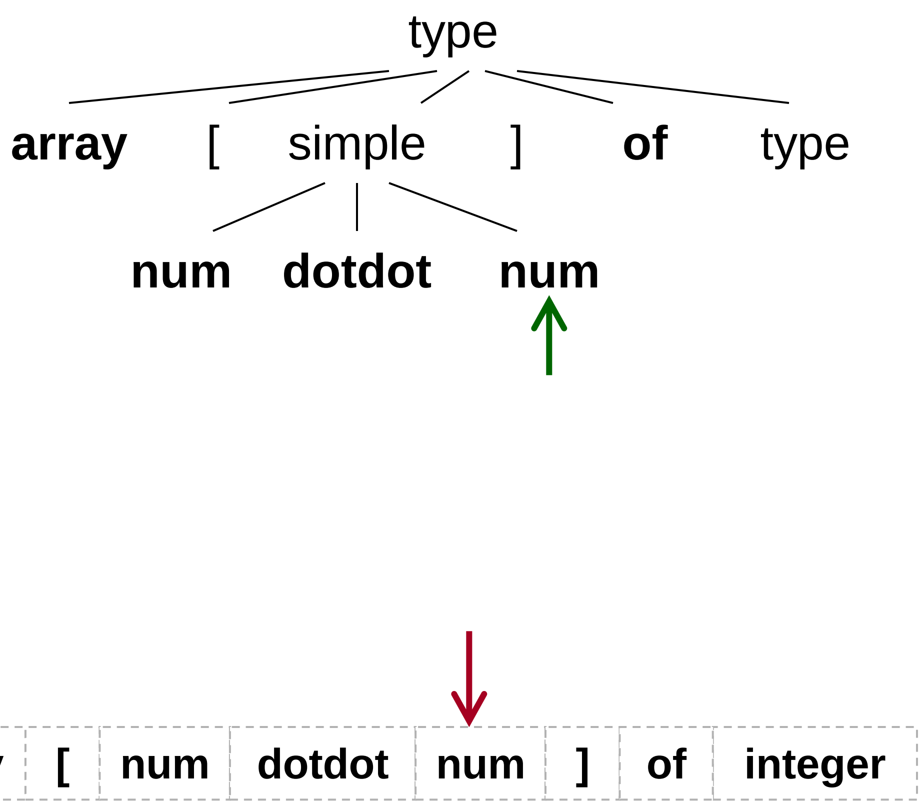
Match !





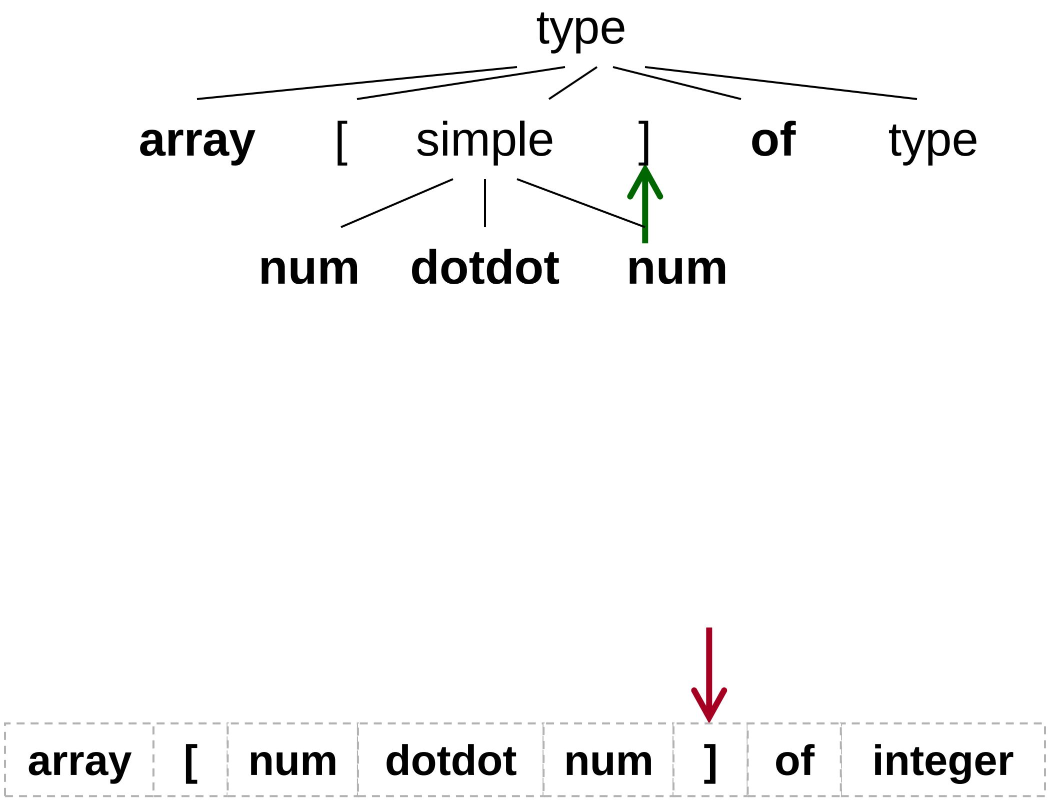
Parsing Process (Action 6)

Match !



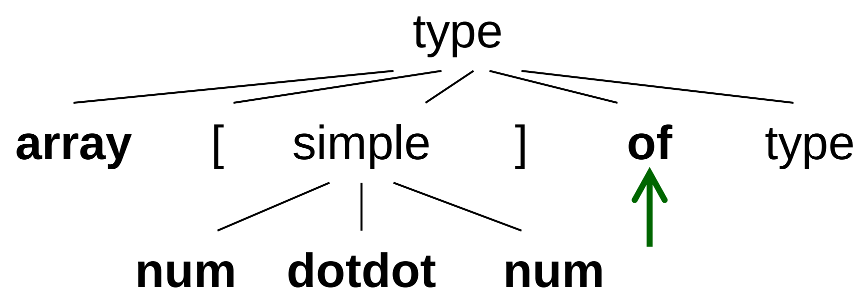
Parsing Process (Action 7)

Match !



Parsing Process (Action 8)

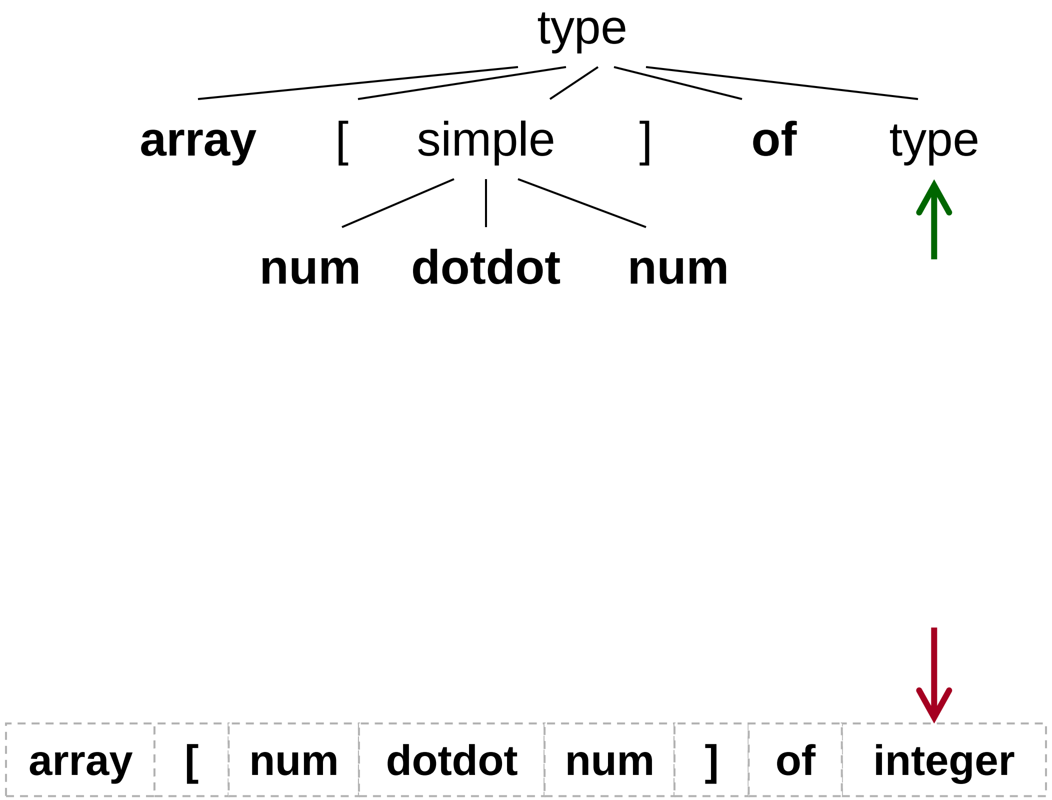
Match !



array	**[**	**num**	**dotdot**	**num**	**]**	**of**	**integer**

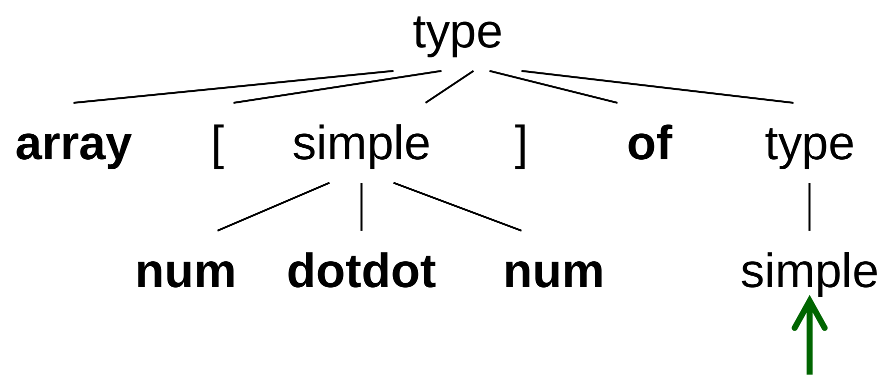
Parsing Process (Action 9)

Derive with "type → simple"



Parsing Process (Action 10)

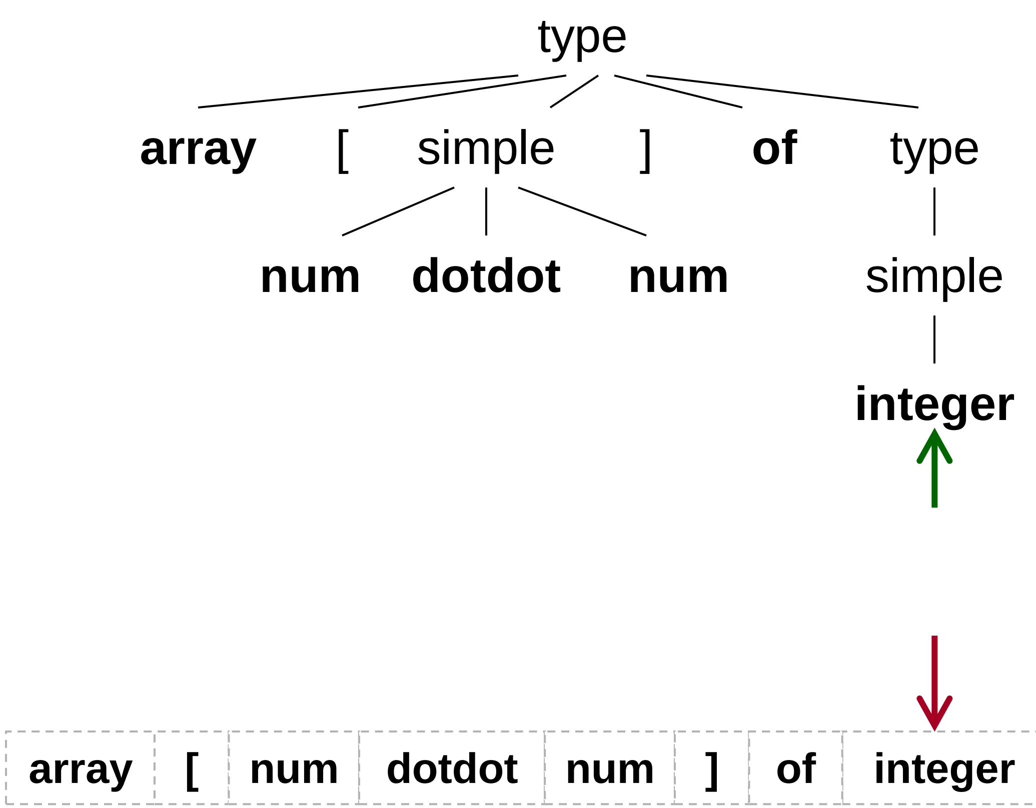
*Derive with "simple → **integer**"*



array	**[**	**num**	**dotdot**	**num**	**]**	**of**	**integer**

Parsing Process (Action 11)

Match and Accept !



4个动作：derive、match、acc、err
match-derive parser

自底向上的4个动作shift、reduce、acc、err

Top-Down Parsers: Perspectives

- Perspective 1:
  parsing capability vs. efficiency trade off
  - Top-down parser with backtracking 回溯
  - Predictive parser 预测

Two **orthogonal** perspectives

orthogonal
英 [ɔːˈθɒɡən(ə)l] 美 [ɔrˈθɑɡənəl]
adj. [数] 正交的；直角的
n. 正交直线

Top-Down Parsers: Perspectives (cont')

- Perspective 2:
  parser implementation
  - Recursive descent parser 递归程序 向下
    - A parser with backtracking
    - A predictive parser
  - Table-driven parser 表驱动
    - Non-recursive programs with an explicit 显式的堆栈 stack and a parsing table.
    - An approach to automation (usually predictive)

lex - 生成状态转化表

3. Rewriting Grammars

Grammar Transformation

- Resolving ambiguities 二义性
  - Trade-off and consequence
- Elimination of ε-productions
  - Systematic elimination of left recursions
- Elimination of left recursions 左递归 A->Aa 无限循环 改为右递归
  - Avoid infinite loop in top-down parsing
- Left-factoring 提取公共左因子 为了消除回溯
  - Avoid backtracking in top-down parsing

目的在于变得predicted

☑ Resolving Ambiguities

- Review: ambiguities in practice
  - Expression
  - Dangling-else
- Resolving ambiguities
  - Ad hoc constraints
  - Trade-off and consequence

Ambiguous Expressions

- Ambiguous grammar

$$
\begin{aligned}
\mathit{expr} \;&\rightarrow\; \mathit{expr} + \mathit{expr} \\
&\;\;|\;\; \mathit{expr}\; \mathbf{*}\; \mathit{expr} \\
&\;\;|\;\; \mathbf{(}\; \mathit{expr}\; \mathbf{)} \;|\; \mathbf{n}
\end{aligned}
$$

- Unambiguous grammar

$$
\begin{aligned}
\mathit{expr} \;&\rightarrow\; \mathit{expr} + \mathit{term} \;|\; \mathit{term} \\
\mathit{term} \;&\rightarrow\; \mathit{term}\; \mathbf{*}\; \mathit{factor} \;|\; \mathit{factor} \\
\mathit{factor} \;&\rightarrow\; \mathbf{(}\; \mathit{expr}\; \mathbf{)} \;|\; \mathbf{n}
\end{aligned}
$$

优先级

Rewriting Rules

- Ad hoc but heuristic rewriting rules
  - Rules for precedence
  - Rules for associativity

Dangling-**else** Problem

- Grammar

$$stmt \rightarrow \textbf{if}\ expr\ \textbf{then}\ stmt \mid \textbf{if}\ expr\ \textbf{then}\ stmt\ \textbf{else}\ stmt \mid \textbf{other}$$

- Example

if E_1 **then** S_1 **else** $\underline{\textbf{if}\ E_2\ \textbf{then}\ S_2\ \textbf{else}\ S_3}$

- Ambiguity

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

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

Unambiguous Grammar

强加约束，二义性更简洁

- Additional disambiguation rule
  - *Each **else** is matched with the closest unmatched **then**.*
- Unambiguous grammar

$$
\begin{array}{lcl}
stmt & \rightarrow & matched\_stmt \mid open\_stmt \\
matched\_stmt & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ matched\_stmt\ \textbf{else}\ matched\_stmt \\
 & \mid & \textbf{other} \\
open\_stmt & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ stmt \\
 & \mid & \textbf{if}\ expr\ \boxed{\textbf{then}\ matched\_stmt}\ \textbf{else}\ open\_stmt
\end{array}
$$

crux

- Example

  if E_1 **then** <u>**if** E_2 **then** S_1 **else** S_2</u>

☑ Eliminating ε-Productions

Some textbook defines ε–free grammar with only the first restriction

- An ε-free grammar
  - No production body is ε (ε-production), or
  - The only ε-production is $S \rightarrow \varepsilon$, and S does not appear in the body of any productions.
- Elimination algorithm
  - For every production $A \rightarrow X_1\ X_2\ \ldots\ X_n$, where $X_i \in \Sigma \cup N$, $1 \leq i \leq n$
  - Add new productions $A \rightarrow a_1\ a_2\ \ldots\ a_n$, where
    - $\neg\ (X_i \Rightarrow^* \varepsilon) \Rightarrow (a_i = X_i)$
    - $(X_i \Rightarrow^* \varepsilon) \Rightarrow (a_i = X_i \vee a_i = \varepsilon)$
    - $\exists 1 \leq i \leq n.\ a_i \neq \varepsilon$

Eliminating ε-Productions: Example 1

- Original grammar

$$S \rightarrow \boxed{A\,\mathbf{a}} \mid \mathbf{b}$$
$$A \rightarrow A\,\mathbf{c} \mid S\,\mathbf{d} \mid \boxed{\varepsilon}$$

- Rewriting grammar

$$S \rightarrow A\,\mathbf{a} \mid \boxed{\mathbf{a}} \mid \mathbf{b}$$
$$A \rightarrow A\,\mathbf{c} \mid \boxed{\mathbf{c}} \mid S\,\mathbf{d}$$

消除空符：
涉及的变换，不涉及照搬

Eliminating ε-Productions: Example 2

- Original grammar

$$S \rightarrow \mathbf{a}\,S\,\mathbf{b}\,S \mid \mathbf{b}\,S\,\mathbf{a}\,S \mid \varepsilon$$

- Equivalent ε-free grammar

$$S' \rightarrow S \mid \varepsilon$$

$$S \rightarrow \mathbf{a}\,S\,\mathbf{b}\,S \mid \mathbf{a}\,\mathbf{b}\,S \mid \mathbf{a}\,S\,\mathbf{b} \mid \mathbf{a}\,\mathbf{b}$$

$$\mid \mathbf{b}\,S\,\mathbf{a}\,S \mid \mathbf{b}\,\mathbf{a}\,S \mid \mathbf{b}\,S\,\mathbf{a} \mid \mathbf{b}\,\mathbf{a}$$

Augmented grammar

找个不存在非终结符，加入其中

Theorem on ε–Free Grammars

- Given any context-free grammar G, there exists an ε-free grammar G', so that $L(G) - \{\varepsilon\} = L(G')$
  - $\varepsilon \notin L(G) \Rightarrow L(G) = L(G')$
  - The only difference between G and G' is the productions.

☑ Eliminating Left Recursions

- Simple immediate left recursion
  - $A \rightarrow A\alpha \mid \beta$ 左递归：非终结符在左边
  - $A \rightarrow \beta A'$
    $A' \rightarrow \alpha A' \mid \varepsilon$

From left recursion to right recursion

- Elimination algorithm
  - $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m$
    $\mid \quad \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$
  - $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$
    $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

Immediate Left Recursion: Example

- Original grammar

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \mathbf{n}$$

- Equivalent grammar without left recursions

$$E \rightarrow T\, E'$$
$$E' \rightarrow + T\, E' \mid \varepsilon$$
$$T \rightarrow F\, T'$$
$$T' \rightarrow * F\, T' \mid \varepsilon$$
$$F \rightarrow (E) \mid \mathbf{n}$$

Systematic Elimination

- Preconditions
  - No cycles, e.g. $A \Rightarrow^+ A$
  - No ε-productions, e.g. $A \to \varepsilon$

Why?

- Elimination algorithm

  Arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$.

  for $i = 1$ **to** n **do begin**

      for $j = 1$ **to** $i - 1$ **do begin**

          Replace each production of the form $A_i \to A_j\gamma$

          by the production $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$,

          where $A_j \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ *are all current* A_j*-productions*

      end

      Eliminate the immediate left recursion among the A_i*-productions*

  end

Systematic Elimination: Example 1

- Original grammar

  $S \rightarrow A\,\mathbf{a} \mid \mathbf{b}$

  $A \rightarrow A\,\mathbf{c} \mid S\,\mathbf{d} \mid \varepsilon$

- Rewriting grammar
  - Eliminating ε-productions:

    $S \rightarrow A\,\mathbf{a} \mid \mathbf{a} \mid \mathbf{b}$

    $A \rightarrow A\,\mathbf{c} \mid \mathbf{c} \mid S\,\mathbf{d}$

  - Eliminating left recursions, ordered by S, A:

    $S \rightarrow A\,\mathbf{a} \mid \mathbf{a} \mid \mathbf{b}$

    ~~$A \rightarrow A\,\mathbf{c} \mid \mathbf{c} \mid A\,\mathbf{a}\,\mathbf{d} \mid \mathbf{a}\,\mathbf{d} \mid \mathbf{b}\,\mathbf{d}$~~

    $A \rightarrow \mathbf{c}\,A' \mid \mathbf{a}\,\mathbf{d}\,A' \mid \mathbf{b}\,\mathbf{d}\,A'$

    $A' \rightarrow \mathbf{c}\,A' \mid \mathbf{a}\,\mathbf{d}\,A' \mid \varepsilon$

去除左递归 总结：A后面的东西统统写到A',后面加上A'和空串
不含A的后面加上A’，加到A后头

Systematic Elimination: Example 2

- Original grammar

  S → A **c** | **c**
  A → B **b** | **b**
  B → S **a** | **a**

- Rewriting with different order

Ordered by **S**, **A**, **B**:

S → A **c** | **c**
A → B **b** | **b**
B → A **c a** | **c a** | **a**
B → B **b c a** | **b c a** | **c a** | **a**
B → **b c a** B' | **c a** B' | **a** B'
B' → **b c a** B' | ε

Ordered by **B**, **A**, **S**:

B → S **a** | **a**
A → S **a b** | **a b** | **b**
S → S **a b c** | **a b c** | **b c** | **c**
S → **a b c** S' | **b c** S' | **c** S'
S' → **a b c** S' | ε

Equivalent

☑ Left Factoring

- Simple left factoring
  - $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$
  - $A \rightarrow \alpha A'$
    $A' \rightarrow \beta_1 \mid \beta_2$
- Left factoring algorithm
  - $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$
  - $A \rightarrow \alpha A' \mid \gamma$
    $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

Left Factoring: Example

- Original grammar

$$S \rightarrow \textbf{if}\ E\ \textbf{then}\ S \mid \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S \mid \textbf{other}$$

$$E \rightarrow \textbf{bool}$$

- After left factoring

$$S \rightarrow \textbf{if}\ E\ \textbf{then}\ S\ S' \mid \textbf{other}$$

$$S' \rightarrow \textbf{else}\ S \mid \varepsilon$$

$$E \rightarrow \textbf{bool}$$

Conclusions: Why Rewriting?

对于二义性，可以用规则进行规定约束

- For all parsing techniques
  - Resolving ambiguities: why?
- Only for top-down parsing

未教消除别消除

  - Eliminating ε-productions: why? for
  - Eliminating left recursions: why? 不断循环
  - Left factoring: why? 回溯

重视原因

4. Top-down Parser with Backtracking

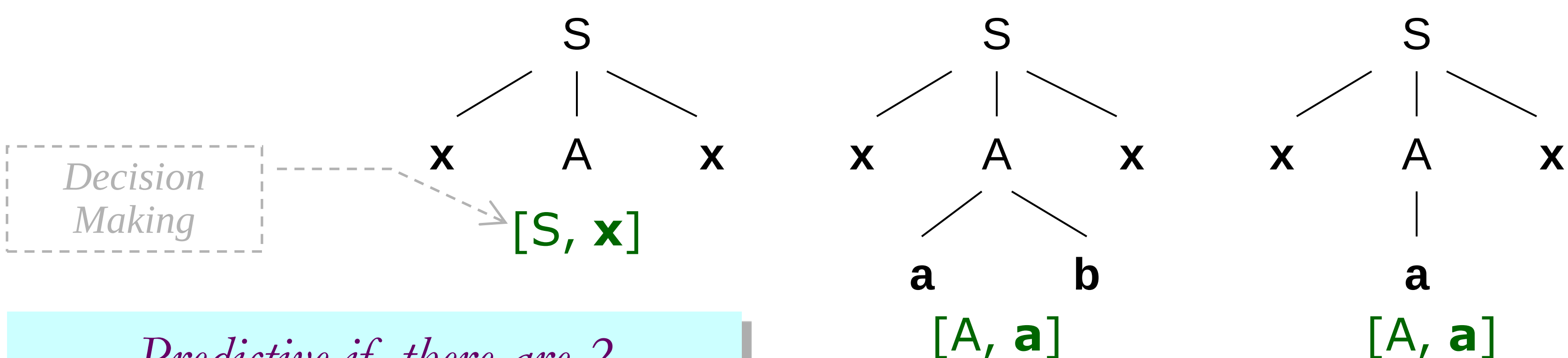
- Trade-off and consequence
  - Pros: powerful to handle most CFG.
  - Cons: complex, and low efficiency.
- Only used to demonstrate the idea of top-down parsing
  - Why a left recursion leads to infinite loop?
  - The meaning of the first symbol that a nonterminal can derive.

Making Decisions on Actions

- Given the following grammar

$$S \rightarrow \mathbf{x}\,A\,\mathbf{x}$$
$$A \rightarrow \mathbf{a}\,\mathbf{b} \mid \mathbf{a} \mid \mathbf{b}$$

- Parsing with only one lookahead
  - Sentence: **`xax`**



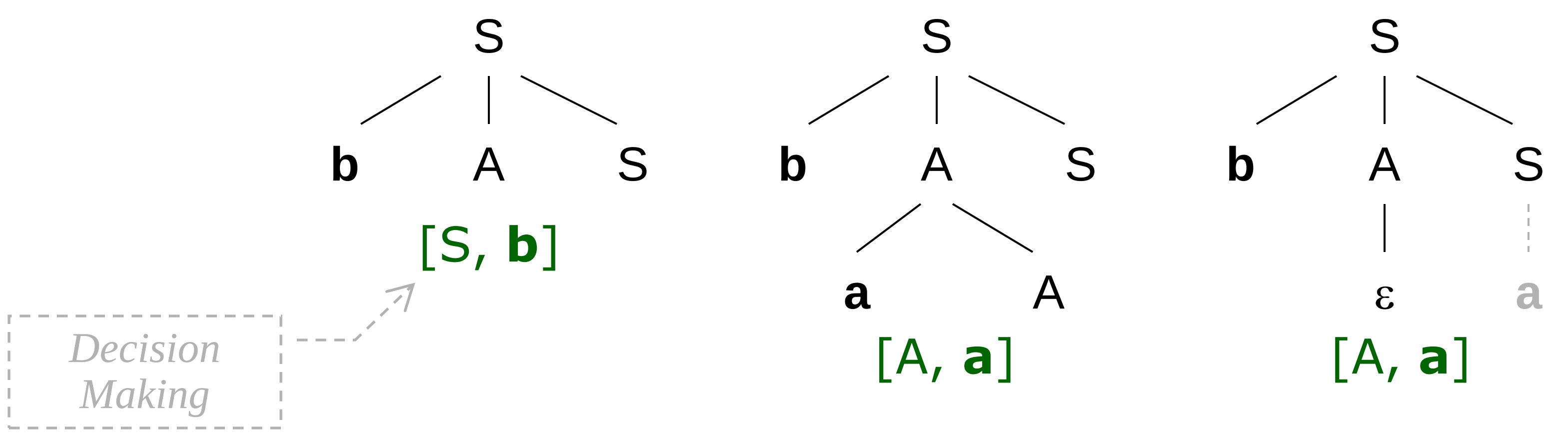
Predictive if there are 2 lookaheads

Decision on Actions (cont')

- Given the following grammar

$$S \rightarrow \mathbf{b}\,A\,S \mid \mathbf{a}$$
$$A \rightarrow \mathbf{a}A \mid \varepsilon$$

- Backtracking caused by an ε-production
  - Sentence: **ba**



Four Actions in Top-Down Parsing

- General actions
  - Accept
  - Error
- Actions specific to top-down parsing
  - match
  - derive

5. Recursive Descent Predictive Parser

- How can a parser be predictive?
  - Sufficient and necessary conditions for the grammar.
  - How to select a unique production candidate with regard to the lookahead.
- An approach suitable for manual development of a top-down parser.

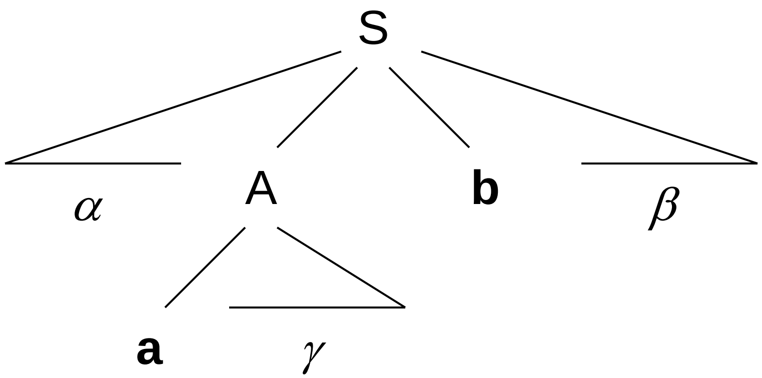
Development Phrases

- Development of a recursive descent predictive parser
  1. Resolve ambiguities in the grammar.
  2. Eliminate left recursions in the grammar.
  3. Left factor the grammar.
  4. Construct transition diagrams from the grammar. 转换图
  5. Reduce the transition diagrams (ad hoc and optional).
  6. Write the parser using the transition diagrams as blue print.

diagram
英 [ˈdaɪəɡræm] 美 [ˈdaɪəɡræm]
n. 图表，图解；几何图形
v. 以图表示

FIRST() and FOLLOW()

- Purposes of these two functions
  - **a** $\in$ FIRST(A) and **b** $\in$ FOLLOW(A)



Function FIRST()

- Intent: while choosing one of $A \to \alpha \mid \beta$ with regard to the lookahead **x**, we must have $\mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta) = \varnothing$.
- Syntax (signature)
  - $\mathrm{FIRST}: (\Sigma \cup N)^* \to 2^{\Sigma \cup \{\varepsilon\}}$

2表示该元素的子集

映射结果

- Semantics
  - $\{\boldsymbol{x} \mid \alpha \Rightarrow^* \boldsymbol{x}\ldots, \boldsymbol{x} \in \Sigma\} \subseteq \mathrm{FIRST}(\alpha)$
  - $\alpha \Rightarrow^* \varepsilon \ \Rightarrow \ \varepsilon \in \mathrm{FIRST}(\alpha)$
    - Why do we permit ε in $\mathrm{FIRST}(\alpha)$?

signature
英 [ˈsɪɡnətʃə(r)] 美 [ˈsɪɡnətʃər]
n. 签名，署名；签字，签署；鲜明特色，明显特征；

Function FIRST() (cont')

- Given $X \in \Sigma \cup N$, we have FIRST(X)
  - $X \in \Sigma \Rightarrow \mathrm{FIRST}(X) = \{X\}$
  - $X \in N \wedge X \rightarrow Y_1 Y_2 \ldots Y_n \Rightarrow$
    - $\exists 1 \leq i \leq n.\ \mathbf{a} \in \mathrm{FIRST}(Y_i)$
      $\wedge \forall 1 \leq j \leq i-1.\ \varepsilon \in \mathrm{FIRST}(Y_j)$
      $\Rightarrow \mathbf{a} \in \mathrm{FIRST}(X)$
    - $\forall 1 \leq k \leq n.\ \varepsilon \in \mathrm{FIRST}(Y_k)$
      $\Rightarrow \varepsilon \in \mathrm{FIRST}(X)$
      - So we have $X \rightarrow \varepsilon \Rightarrow \varepsilon \in \mathrm{FIRST}(X)$

Function FIRST() (cont')

- Given $\alpha = X_1X_2\ldots X_n \in (\Sigma \cup N)^*$, we have $\mathrm{FIRST}(\alpha)$
  - $\mathrm{FIRST}(X_1) - \{\varepsilon\} \subseteq \mathrm{FIRST}(\alpha)$
  - $\forall 2 \le i \le n.\ \forall 1 \le j \le i-1.\ \varepsilon \in \mathrm{FIRST}(X_j)$ $\Rightarrow \mathrm{FIRST}(X_i) - \{\varepsilon\} \subseteq \mathrm{FIRST}(\alpha)$
  - $\forall 1 \le i \le n.\ \varepsilon \in \mathrm{FIRST}(X_i) \Rightarrow \varepsilon \in \mathrm{FIRST}(\alpha)$

Overloading

Function FOLLOW()

- Syntax (signature)
  - FIRST: $(\Sigma \cup N)^* \rightarrow 2^{\Sigma \cup \{\varepsilon\}}$

2表示该元素的子集

映射结果

- Semantics
  - $\{x \mid \alpha \Rightarrow^* x..., x \in \Sigma\} \subseteq \mathrm{FIRST}(\alpha)$
  - $\alpha \Rightarrow^* \varepsilon \;\Rightarrow\; \varepsilon \in \mathrm{FIRST}(\alpha)$

- Intent: while choosing one of $A \rightarrow \alpha \mid \varepsilon$ with regard to the lookahead **x**, we must have $\mathrm{FIRST}(\alpha) \cap \mathrm{FOLLOW}(A) = \varnothing$.
- Syntax (signature)

  非终结符 2表示该元素的子集

  - FOLLOW: $N \rightarrow 2^{\Sigma \cup \{\$\}}$, *where a special symbol* **$** *indicates the end of input token stream.*

  end of file

- Semantics
  - $\{\boldsymbol{x} \mid S \Rightarrow^* \alpha A \boldsymbol{x} \beta \wedge \boldsymbol{x} \in \Sigma\} \subseteq \mathrm{FOLLOW}(A)$
  - $S \Rightarrow^* \alpha A \;\Rightarrow\; \boldsymbol{\$} \in \mathrm{FOLLOW}(A)$

Function FOLLOW() (cont')

- Given $X \in N$, we have FOLLOW(X)
  - $X = S \Rightarrow \mathbf{\$} \in \mathrm{FOLLOW}(X)$
  - $A \rightarrow \alpha X \beta \Rightarrow \mathrm{FIRST}(\beta) - \{\varepsilon\} \subseteq \mathrm{FOLLOW}(X)$
  - $(A \rightarrow \alpha X) \vee (A \rightarrow \alpha X \beta \wedge \varepsilon \in \mathrm{FIRST}(\beta))$
    $\Rightarrow \mathrm{FOLLOW}(A) \subseteq \mathrm{FOLLOW}(X)$

top-down 消除左递归
First定义域为串，逐个字符求完取并集

Example 1

FIRST总结：
看左部，从底向上看
如果推出终结符，就加入
如果推出非终结符，就除空串外全部加入
若可以空串：全空或者加后面的first or可以直接退出

- Given the following grammar

$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +\,T\,E' \mid \varepsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \varepsilon \\
F &\rightarrow (\,E\,) \mid \mathbf{n}
\end{aligned}
$$

FIRST是+，FOLLOW是=

- We have
  - FIRST(F) = FIRST(T) = FIRST(E) = {**(**, **n**}
  - FIRST(T') = {*****, ε}
  - FIRST(E') = {**+**, ε}
  - FOLLOW(E) = FOLLOW(E') = {**)**, **$**}
  - FOLLOW(T) = FOLLOW(T') = {**+**, **)**, **$**}
  - FOLLOW(F) = {**+**, *****, **)**, **$**}

记住使用这种写法更好，不然需要不断执行直到没有更新了

Follow总结：
按左部的顺序看右部，从上向下
开始符号后面要加$
后面跟着什么就加什么，除了空串
如果可以作为最后一个字符，那么左部的follow就是他的follow

20分，极重要

Example 2

- Given the following grammar

$$A \rightarrow B\ C$$
$$B \rightarrow A\ \mathbf{x} \mid \mathbf{x} \mid \varepsilon$$
$$C \rightarrow \mathbf{y}\ C \mid \mathbf{y}$$

- We have

Symbol	FIRST	FOLLOW
A	x y	x $
B	+A的 x y ε	y
C	y	x $

Heuristics in manual calculation:
Left vs. **right**
Top-down vs. **bottom-up**

Conditions for Predictive Parsing

- Sufficient and necessary conditions
- CFG G is LL(1) iif whenever $A \rightarrow \alpha \mid \beta$ are two distinct productions of G, the following conditions hold

充要条件

  - $\mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta) = \varnothing$
  - $\varepsilon \in \mathrm{FIRST}(\beta) \Rightarrow \mathrm{FIRST}(\alpha) \cap \mathrm{FOLLOW}(A) = \varnothing$

$LL(1)$文法

- 文法G是$LL(1)$的，当且仅当G的任意两个具有相同左部的产生式$A \rightarrow \alpha \mid \beta$满足下面的条件：
  - 如果α和β均不能推导出ε，则$FIRST(\alpha) \cap FIRST(\beta) = \Phi$

为准

  - α和β至多有一个能推导出ε

SELECT (A→β) = (FIRST (β) − {ε}) ∪ FOLLOW (A)

SELECT (A→α)

  - 如果$\beta \Rightarrow^{*} \varepsilon$，则$FIRST(\alpha) \cap FOLLOW(A) = \Phi$；
    如果$\alpha \Rightarrow^{*} \varepsilon$，则$FIRST(\beta) \cap FOLLOW(A) = \Phi$；

aim 同一非终结符的各个产生式的可选集互不相交

可以为$LL(1)$文法构造预测分析器

Discussions

- What benefits from the LL(1) conditions?
  - Decision of the derive/error action in the parser with regard to the current nonterminal and a single lookahead.
  - Only 0 or 1 production will be chosen.
- They are not predictive (why?)
  - Ambiguous grammars
  - Grammars with left recursions
  - Grammars with left factors

左递归与左因子
不在LL（1）中

见PPT2

Transition Diagram

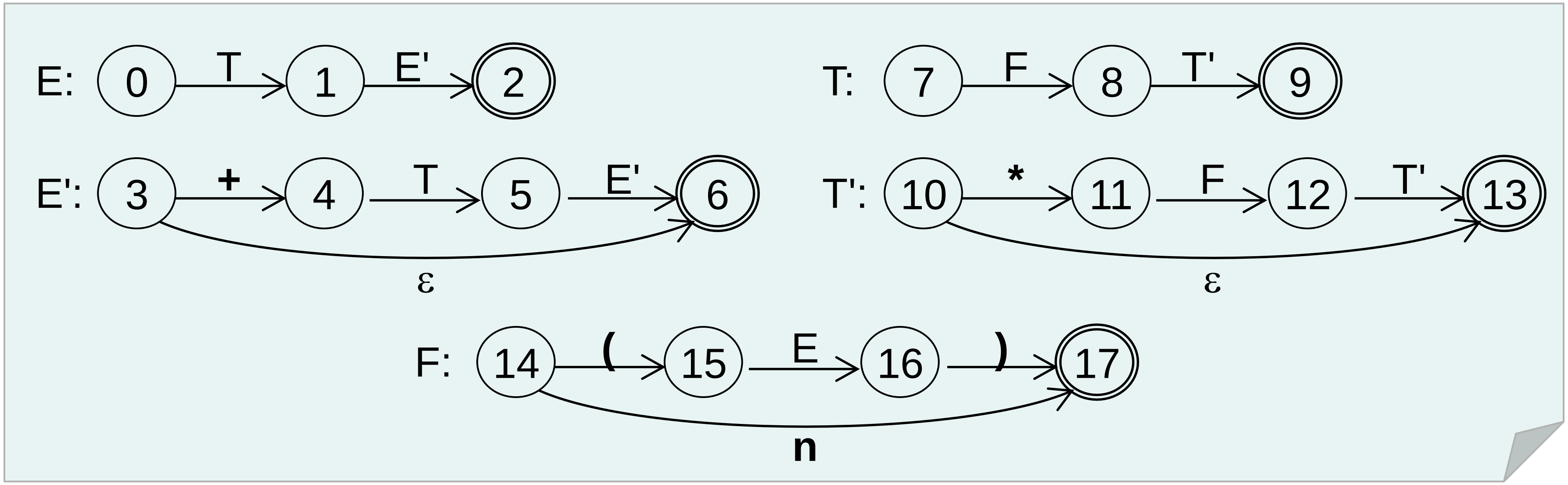
- Visualize the predictive parser
  - For each nonterminal A
    - Create an initial and final state.
    - For each $A \rightarrow X_1X_2...X_n$, create a path from the initial state to the final state, with edges labeled $X_1, X_2, ..., X_n$.
      - If $A \rightarrow \varepsilon$, the path is labeled ε.

Transition Diagram: Example

- Given the following grammar

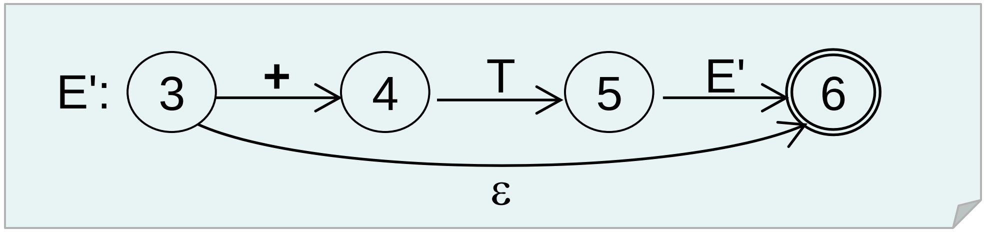
$$
\begin{array}{lll@{\qquad}lll}
E & \rightarrow & T\,E' & E' & \rightarrow & +\,T\,E' \mid \varepsilon \\
T & \rightarrow & F\,T' & T' & \rightarrow & *\,F\,T' \mid \varepsilon \\
F & \rightarrow & (\,E\,) \mid \mathbf{n} & & &
\end{array}
$$

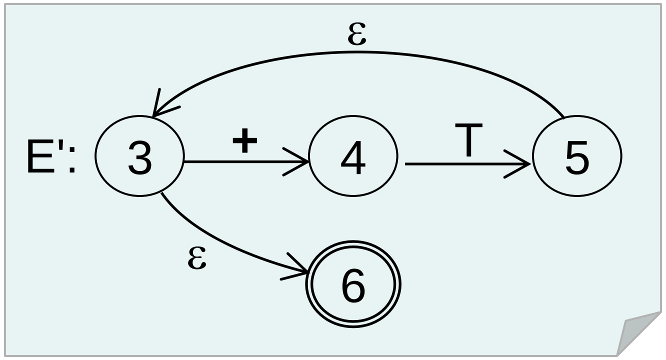
- Transform to transition diagrams

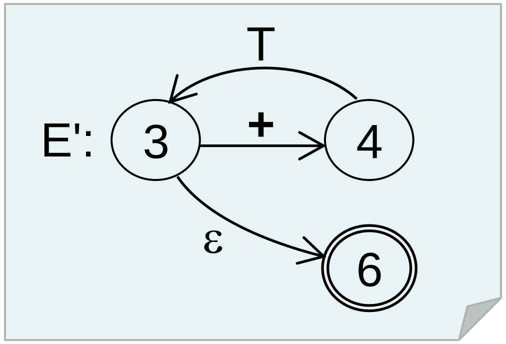


Reduction of Transition Diagrams

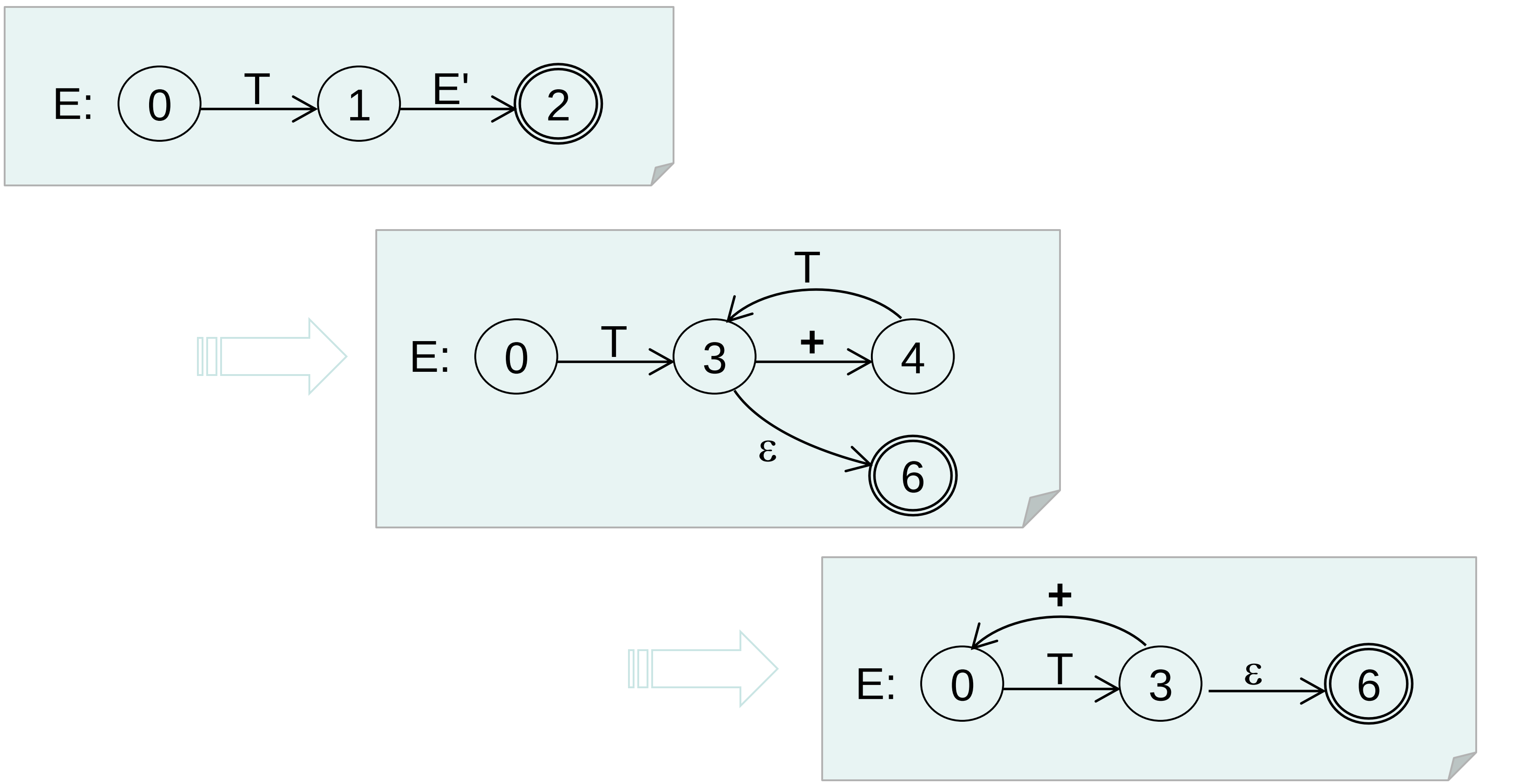
- Ad hoc rules: iterative substitution



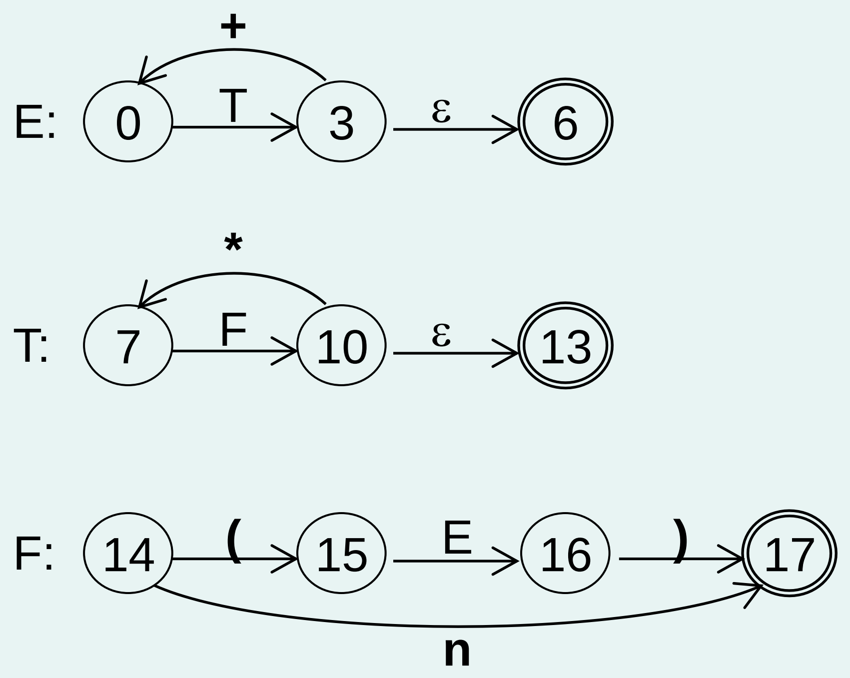




Reduction of Transition Diagrams (cont')



Reduced Transition Diagrams



Discussion

- Transition diagrams: (recursive descent predictive) parsing vs. scanning
  - Point 1
  - Point 2

Write a Recursive Predictive Parser

- Use the transition diagrams as the blue-print of the parser
  - Write a recursive procedure for each diagram.
    - The procedure for the diagram of S is the main entry.
  - Design the control flow of the procedure mimicking the paths in the diagram, with regard to the lookahead.
    - If the label of an edge in the path is a terminal, perform the **match** action.
    - If the label is a nonterminal, perform the **derive** action, that is invoking the procedure of the nonterminal (recursively).

Coding Rules

- A $\rightarrow$ a B b

```
void A() {
      match(a);
      B();
      match(b);
}
```

```
void match(Token tok) {
    if (lookahead == tok) {
        lookahead = scanner.getNextToken();
    } else  error();
}
```

Coding Rules (cont')

- $A \rightarrow a\ B\ b \mid b\ A\ B$

```
void A() {
      if (lookahead == a) {
            match(a);  B();  match(b)
      } else if (lookahead == b) {
            match(b);  A();  B()
      } else  error()
}
```

Coding Rules (cont')

- A → a B b | b A B | C

```
void A() {
    if (lookahead == a) {
        match(a);  B();  match(b)
    } else if (lookahead == b) {
        match(b);  A();  B()
    } else  C();
}
```

```
void A() {
    if (lookahead == a) {
        match(a);  B();  match(b)
    } else if (lookahead == b) {
        match(b);  A();  B()
    } else if (lookahead in FIRST(C)) {
        C()
    } else  error();
}
```

Coding Rules (cont')

- A $\rightarrow$ a B b | b A B | ε

```
void A() {
   if (lookahead == a) {
      match(a);  B();  match(b)
   } else if (lookahead == b) {
      match(b);  A();  B()
   } else  ;    // do nothing
}
```

```
void A() {
   if (lookahead == a) {
      match(a);  B();  match(b)
   } else if (lookahead == b) {
      match(b);  A();  B()
   } else if (lookahead in FOLLOW(A)) {
      // do nothing, more accurate!
   } else  error();
}
```

Example 1

type	$\rightarrow$	simple
	|	^ **id**
	|	**array** [simple] **of** type
simple	$\rightarrow$	**integer**
	|	**char**
	|	**num dotdot num**

```
void type() throws SyntacticException {
   if (lookahead.equals(new Token('^'))) {
      match(new Token('^'));
      match(new Token(Token.ID));
   } else if (lookahead.equals(new Token(Token.ARRAY))) {
      match(new Token(Token.ARRAY));
      match(new Token('['));
      simple();
      match(new Token(']'));
      match(new Token(Token.OF));
      type();
   } else  simple();
}
```

Coding is direct and intuitive while using unreduced transition diagrams as a blueprint.

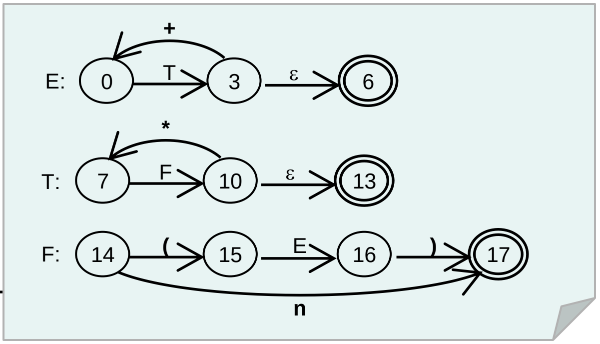
Example 1 (cont')

type → simple
  | ^ **id**
  | **array** [simple] **of** type
simple → **integer**
  | **char**
  | **num dotdot num**

```
void simple() throws SyntacticException {
   if (lookahead.equals(new Token(Token.INTEGER))) {
      match(new Token(Token.INTEGER));
   } else if (lookahead.equals(new Token(Token.CHAR))) {
      match(new Token(Token.CHAR));
   } else if (lookahead.equals(new Token(Token.NUM))) {
      match(new Token(Token.NUM));
      match(new Token(Token.DOTDOT));
      match(new Token(Token.NUM));
   } else  throw new SyntacticException();
}

void match(Token tok) throws SyntacticException {
   if (lookahead.equals(tok))
      lookahead = scanner.getNextToken();
   else  throw new SyntacticException();
}
```

Example 2

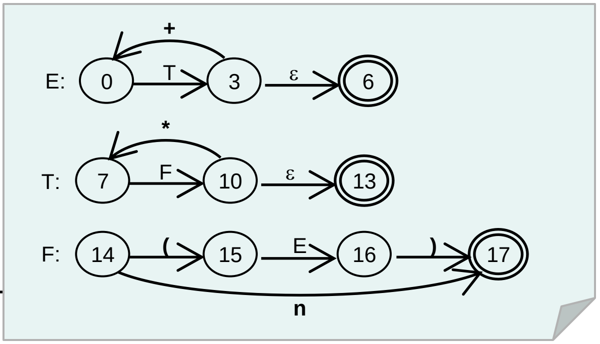



```
void expr() throws SyntacticException {
    term();
    while (lookahead.equals(new Token('+'))) {
        match(new Token('+'));
        term();
    }
}

void term() throws SyntacticException {
    factor();
    while (lookahead.equals(new Token('*'))) {
        match(new Token('*'));
        factor();
    }
}
```

Some tricks are needed while coding with reduced transition diagrams.

Example 2 (cont')




```
void factor() throws SyntacticException {
    if (lookahead.equals(new Token('('))) {
        match(new Token('('));
        expr();
        match(new Token(')'));
    } else if (lookahead.equals(new Token(Token.NUM))) {
        match(new Token(Token.NUM));
    } else {
        throw new SyntacticException();
    }
}
```

Exercise 4.1

- Given the following grammar

$$S \rightarrow (\,L\,) \mid \mathbf{a}$$

$$L \rightarrow L\,,\,S \mid S$$

  - Eliminate left recursions in the grammar.
  - Draw the transition diagrams for the grammar.
  - Write a recursive descent predictive parser.
  - Indicate the procedure call sequence for an input sentence **`(a, (a, a))`**.

Exercise 4.2

- Consider the context-free grammar

$$S \rightarrow \mathbf{a}\, S\, \mathbf{b}\, S \mid \mathbf{b}\, S\, \mathbf{a}\, S \mid \varepsilon$$

  - Can you construct a predictive parser for the grammar? and why?

Exercise 4.3

- Compute the FIRST and FOLLOW for the start symbol of the following grammar

$$S \rightarrow S\,S\,+ \mid S\,S\,\mathbf{*} \mid \mathbf{a}$$

Further Reading

- Dragon Book, 2nd Edition (DBv2)
  - Comprehensive Reading:
    - Section 2.4, 4.1.1–4.1.2 and 4.4.1 for the introduction to top-down parsing.
    - Section 4.2 and 4.3 for context-free grammar and grammar transformations.
    - Section 4.4.2 for function FIRST and FOLLOW.
  - Skip Reading:
    - Section 4.1.3–4.1.4 for error recovery in top-down parsing.

Enjoy the Course!