Context-Free Language

齐建鹏 2019-09-24

本节目标

- 掌握CFL与CFG特点及解析过程
- 歧义性
- 文法标准化

Context-Free Language的直观概念

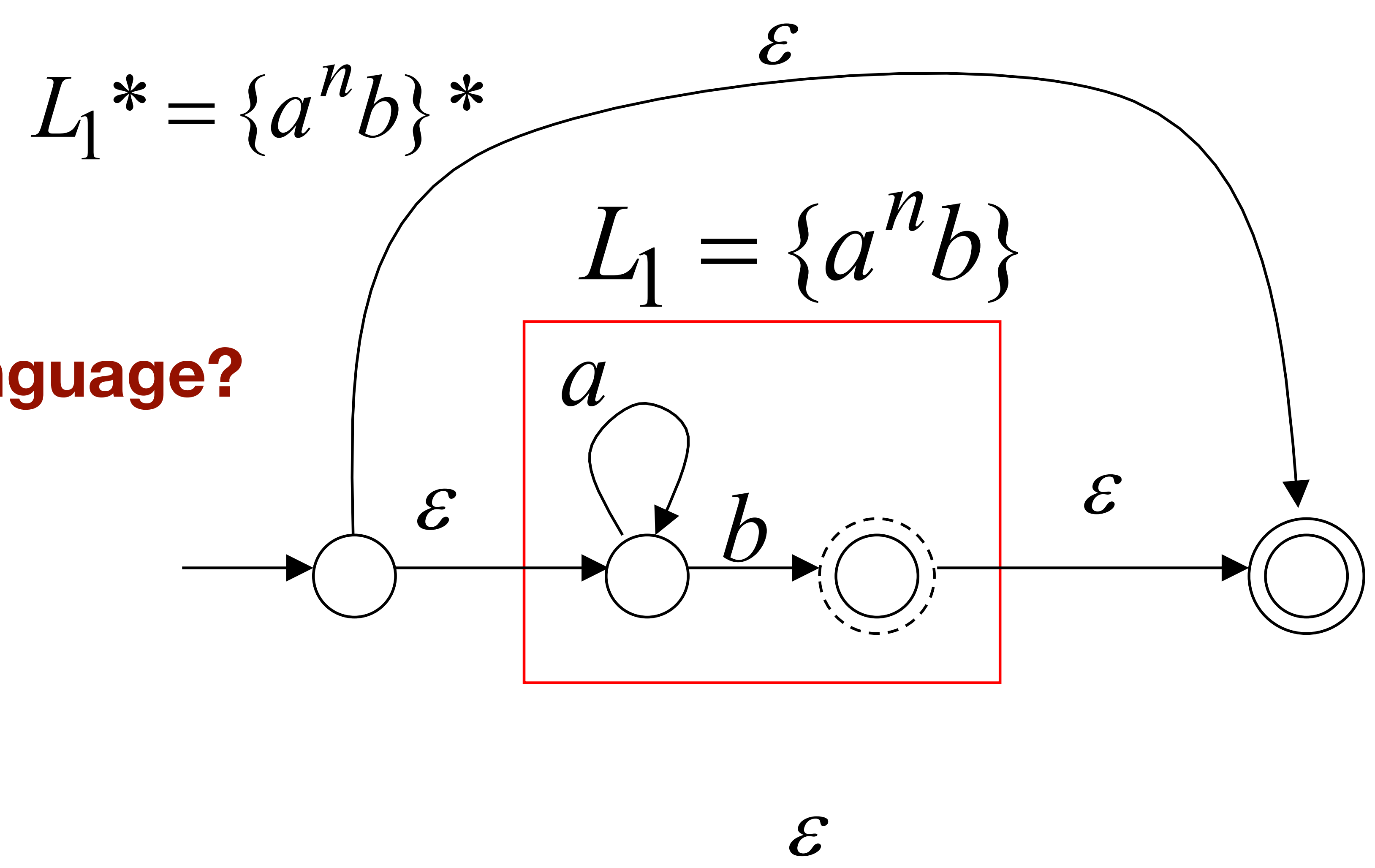
- 括号配对: (((( ))))

- XML中tag配对 <root><child></child></root>

General idea:
CFLs are languages that can be recognized by automata that have **one single stack**:

- { 0^n1^n | n≥0} is a CFL
- { $0^n1^n0^n$ | n≥0} is not a CFL

Regular Language?



Context-Sensitive Language的直观概念

- P="Capital of china" 如何理解?

中国的首都(北京） 或 瓷都（景德镇)

有歧义(ambiguous)……

aSb -> aaSb
aSc ->aaSc

X china has The Great Wall -> x 中国 …

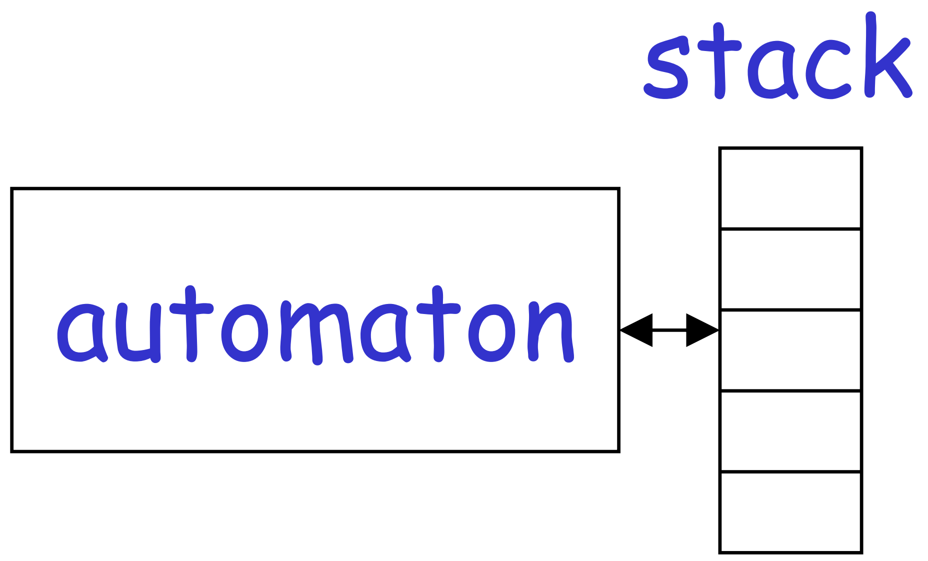
U china burns…-> X 瓷器 burns…

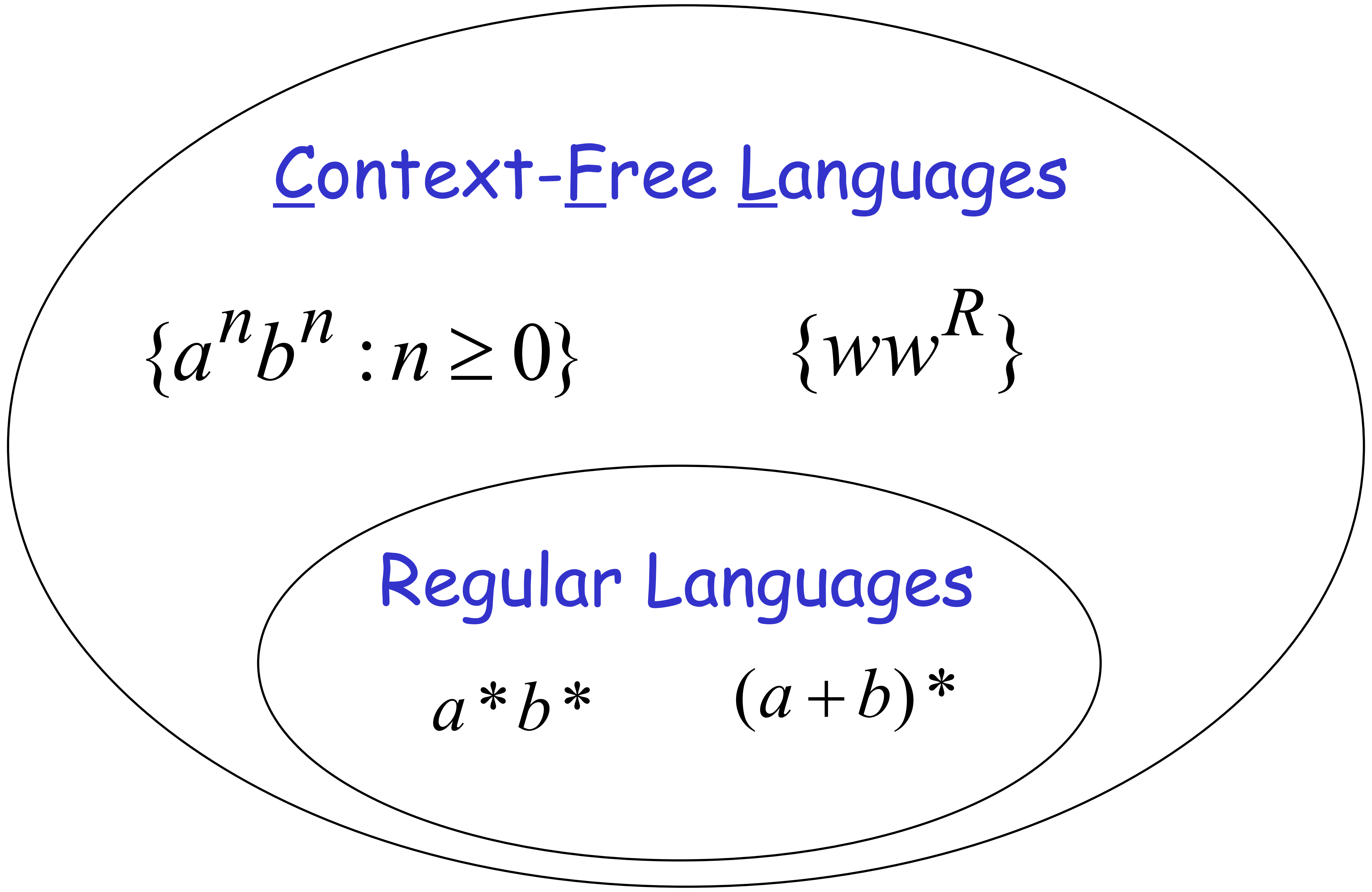
人类语言太丰富，是CSL，比较复杂， 以后讨论。诡辩术"断章取义" 的语言学本质：前后相关语言中，去掉W的前后文，在W的释义集合中，选择有利于自己的意义

Context-Free Languages

Context-Free Grammars

Pushdown Automata





Context-Free Grammars(Inf.)

Which simple machine produces the non-regular language: $\{0^n 1^n \mid n \in N\}$

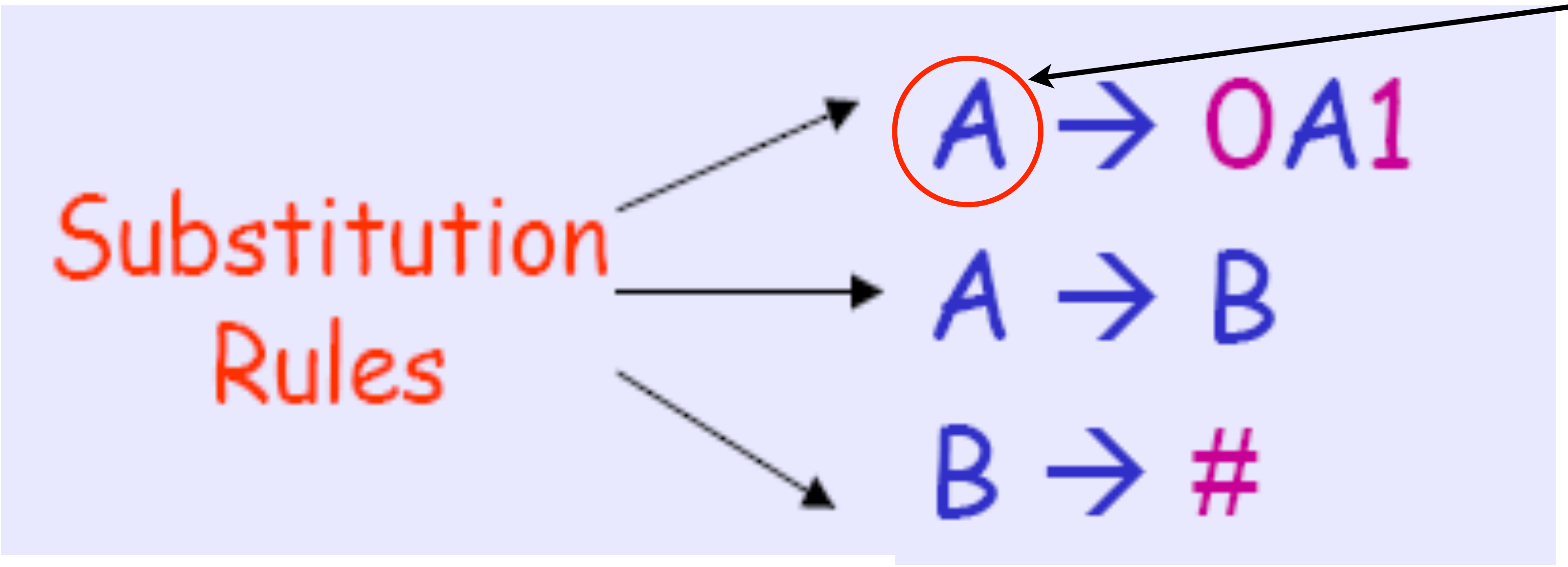
Start symbol S with rewrite rules:

1. $S \rightarrow 0S1$
2. $S \rightarrow \epsilon$

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow \ldots \Rightarrow 0^n S 1^n \Rightarrow 0^n 1^n$$

Context-Free Grammars

起始变元(*start variable*)



变元(Variable): **A,B**

终结符(Terminals): 输入符号，用小写字母、数字或特殊符号表示： 0, 1, #

字符串(变元 + 终结符)

生成式(production)： 替换规则

⚠️ CFG中替换规则左侧有且只有一个变元，与CSG不同

CFG 形式定义

$\boxed{S \to aSb \mid \varepsilon}$

productions

$R = \{S \to aSb,\ S \to \varepsilon\}$

$G = (V, \Sigma, S, R)$

$V = \{S\}$
variables

$T = \{a, b\}$

start variable

上下文无关文法是一个4元组(V, Σ, R, S)：

1. V 是一个有穷集合，称作变元集；
2. Σ是一个与V不相交的有穷集合，称作终结符集；
3. R是一个有穷的规则集，每一条规则是一个变元和一个由变元和终结符组成的字符串；
4. $S \in V$是起始变元。

⚠️ 起始变元是第一条规则左边的变元

How does CFG generate strings?

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$

1. Write down the start symbol;
2. Find a variable that is written down, and a rule that starts with that variable; Then, replace the variable with the rule;
3. Repeat the above step until no variable is left

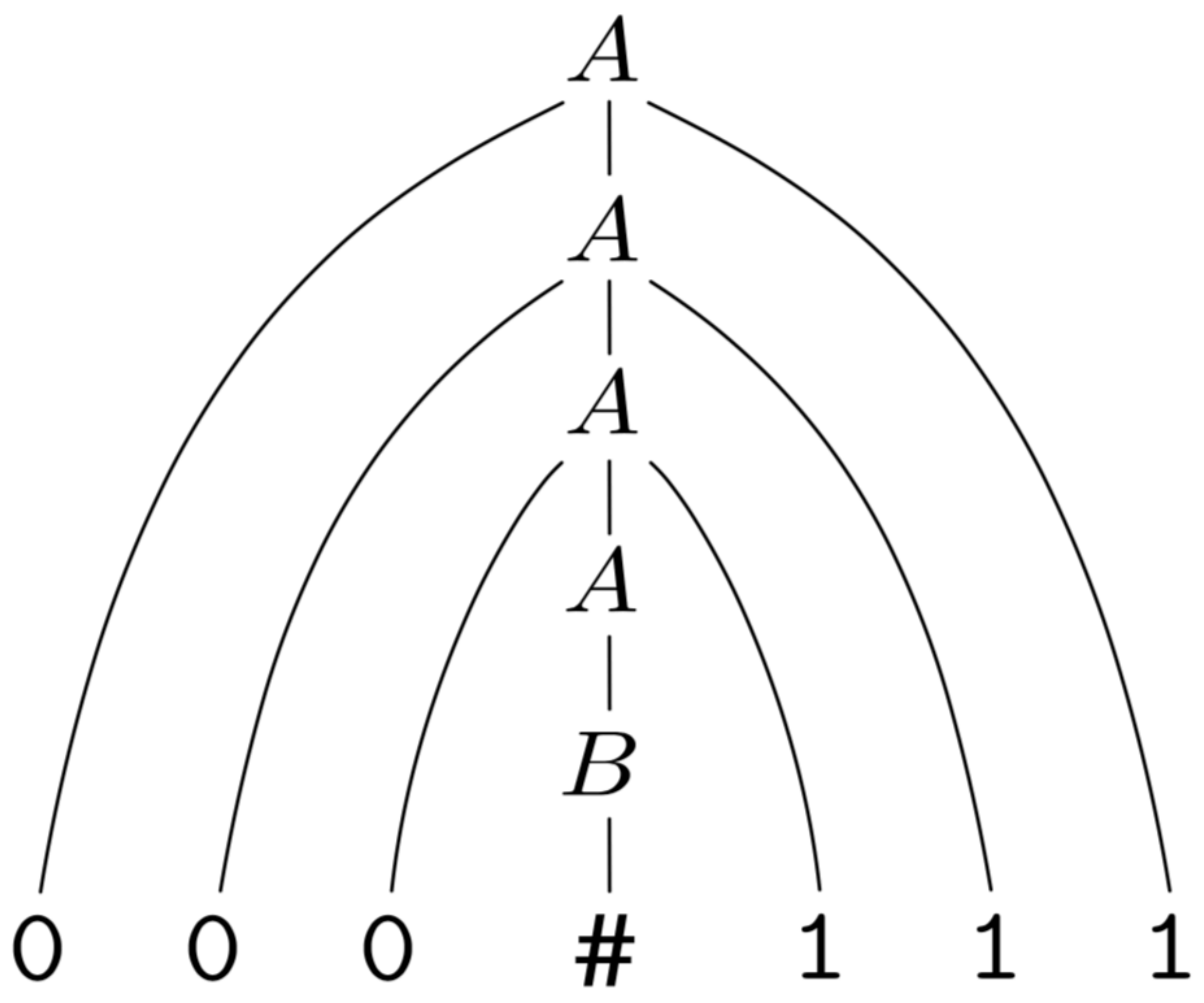
How does CFG generate strings?

文法G

$A \to 0A1$
$A \to B$
$B \to \#$

Step 1. A (write down the start variable)
Step 2. $0A1$ (find a rule and replace)
Step 3. $00A11$ (find a rule and replace)
Step 4. $00B11$ (find a rule and replace)
Step 5. $00\#11$ (find a rule and replace)

parse tree:



多步派生

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow \qquad 000\#111$$

派生（derivation）：获取一个字符串的替换序列

Derivation $\stackrel{*}{\Rightarrow}$ 派生，生成

- If u, v, and w are strings of variables and terminals, and $A \to w$ is a rule of the grammar, we say that uAv ***yields*** uwv, written $uAv \Rightarrow uwv$.

- Say that u ***derives*** v, **written $u \stackrel{*}{\Rightarrow} v$**, if $u = v$ or if a sequence $u_1, u_2, u_3, \ldots, u_k$ exists for $k \geq 0$ and $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$.

Context-Free Language

通过派生可得到CFL，For a grammar G with start variable S:

$$L(G) = \{w: \quad S \overset{*}{\Rightarrow} w, \quad w \in T^*\}$$

String of terminals or ϵ

CFG $\boxed{S \rightarrow aSb \mid \varepsilon}$

CFL $L(G) = \{a^n b^n : \quad n \geq 0\}$

Context-Free Language判定

A language L is context-free, if there is a context-free grammar G

with $L = L(G)$

$L = \{a^n b^n : \quad n \geq 0\}$

$\updownarrow$

$\boxed{S \to aSb \mid \varepsilon}$

$L(G) = \{ww^R : \quad w \in \{a,b\}^*\}$

$\updownarrow$

$\boxed{S \to aSa \mid bSb \mid \varepsilon}$

Quick Quiz

实现加法(+), 乘法(*), 括号() 运算文法 G_4, $G_4 = (V, \Sigma, R, \langle EXPR \rangle)$.

其中:

$$V = \{\langle EXPR \rangle, \langle TERM \rangle, \langle FACTOR \rangle\},$$

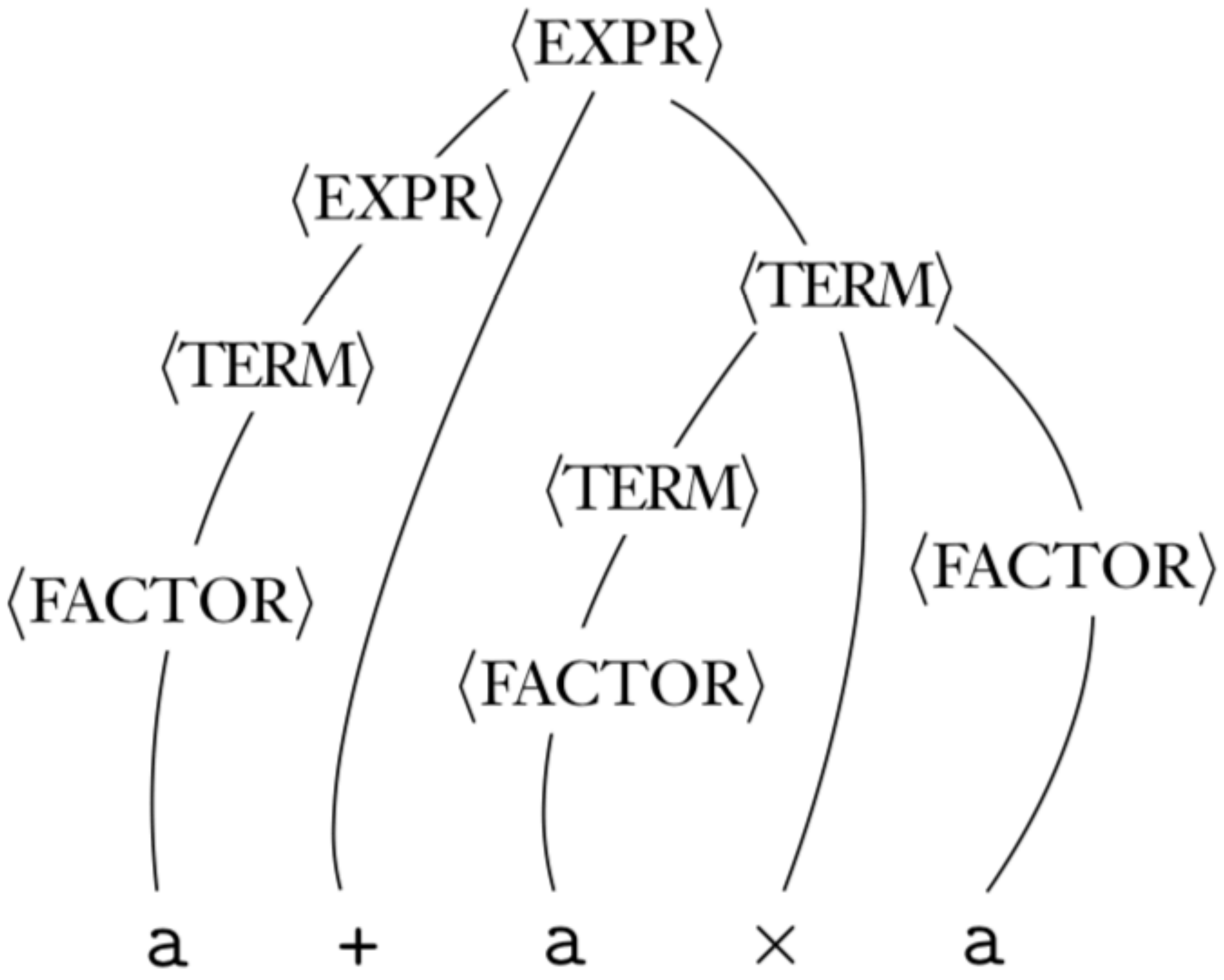
$$\Sigma = \{\, + \,, * \,, (,), [0 - 9a - z]\}$$

$$\begin{aligned}
&\langle EXPR \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle \mid \langle TERM \rangle \\
&\langle TERM \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle \mid \langle FACTOR \rangle \\
&\langle FACTOR \rangle \rightarrow (\langle EXPR \rangle) \mid CHARS \\
&CHARS \rightarrow [0 - 9a - z]+
\end{aligned}$$

? 括号匹配((((())))), 运算优先级

Quick Quiz

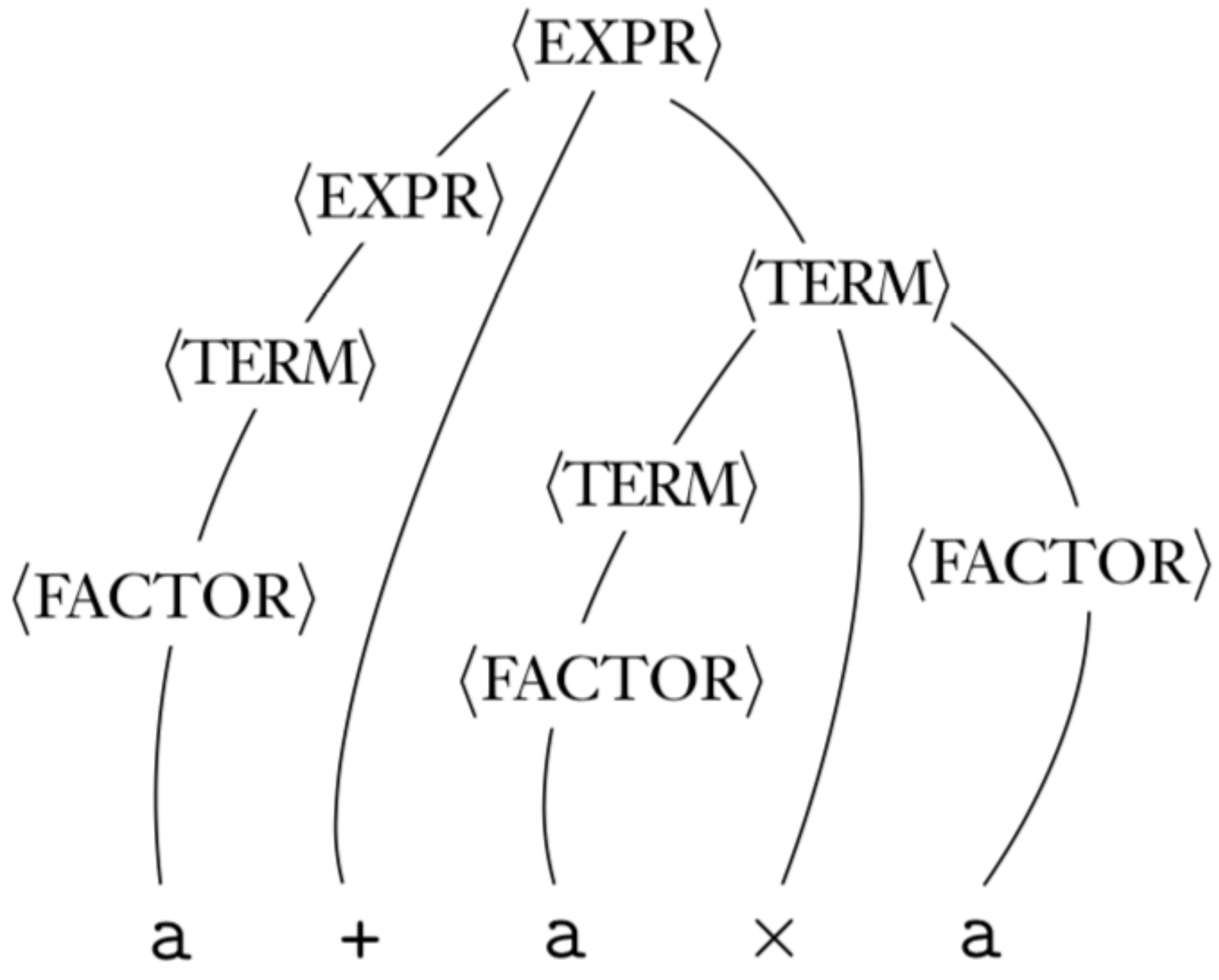
- $a + a \times a$



$$\langle EXPR \rangle \rightarrow \langle EXPR \rangle + \langle TERM \rangle \,|\, \langle TERM \rangle$$
$$\langle TERM \rangle \rightarrow \langle TERM \rangle \times \langle FACTOR \rangle \,|\, \langle FACTOR \rangle$$
$$\langle FACTOR \rangle \rightarrow (\langle EXPR \rangle) \,|\, CHARS$$
$$CHARS \rightarrow [0 - 9a - z]+$$

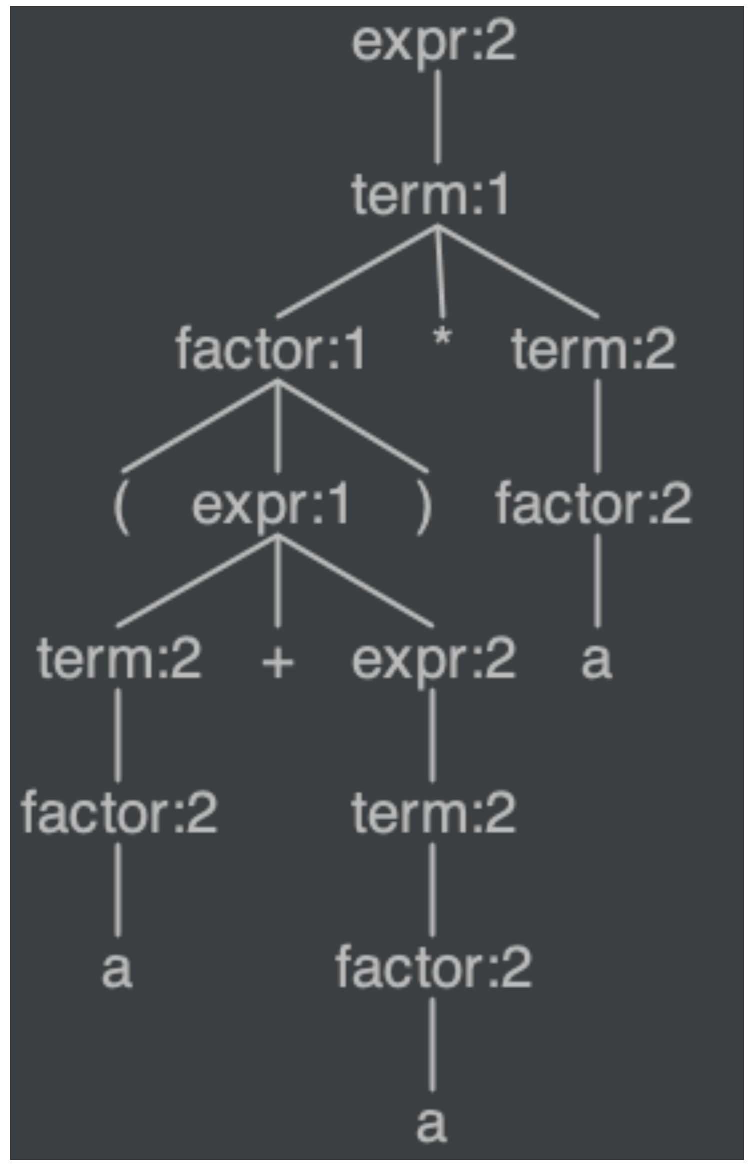
Quick Quiz

- $(a + a) \times a$



$$\langle EXPR \rangle \rightarrow \boxed{\langle EXPR \rangle} + \langle TERM \rangle \,|\, \langle TERM \rangle$$
$$\langle TERM \rangle \rightarrow \boxed{\langle TERM \rangle} \times \langle FACTOR \rangle \,|\, \langle FACTOR \rangle$$
$$\langle FACTOR \rangle \rightarrow (\langle EXPR \rangle) \,|\, CHARS$$
$$CHARS \rightarrow [0-9a-z]+$$

$$\langle EXPR \rangle \rightarrow \langle TERM \rangle + \langle EXPR \rangle \,|\, \langle TERM \rangle$$
$$\langle TERM \rangle \rightarrow \langle FACTOR \rangle \times \langle TERM \rangle \,|\, \langle FACTOR \rangle$$
$$\langle FACTOR \rangle \rightarrow (\langle EXPR \rangle) \,|\, CHARS$$
$$CHARS \rightarrow [0-9a-z]+$$



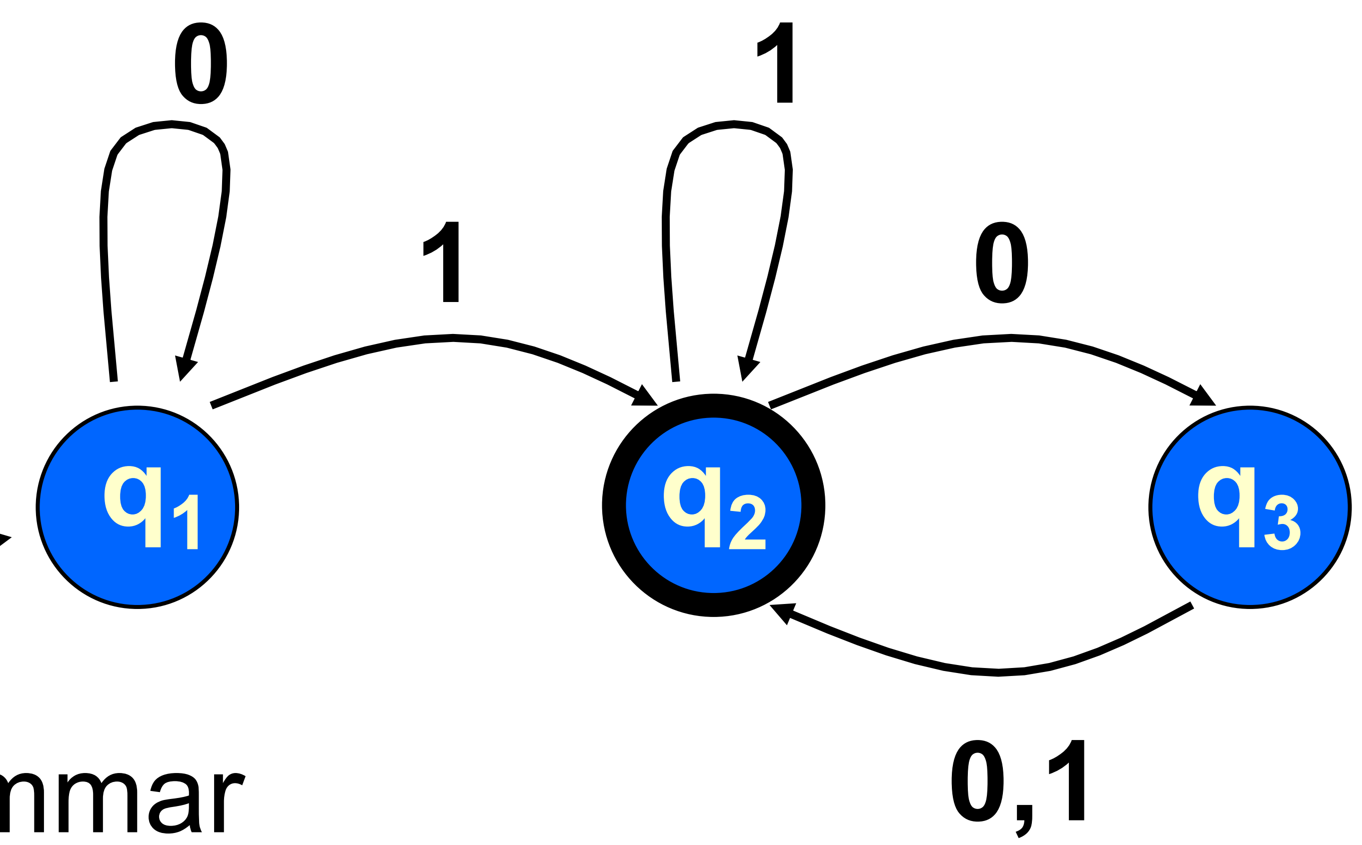
Generated by ANTLR

$RL \subseteq CFL$

Every **regular language** can be expressed by a context-free grammar.

Proof Idea: RL →**DFA**→造**CFG**, 使得结果一样

The DFA

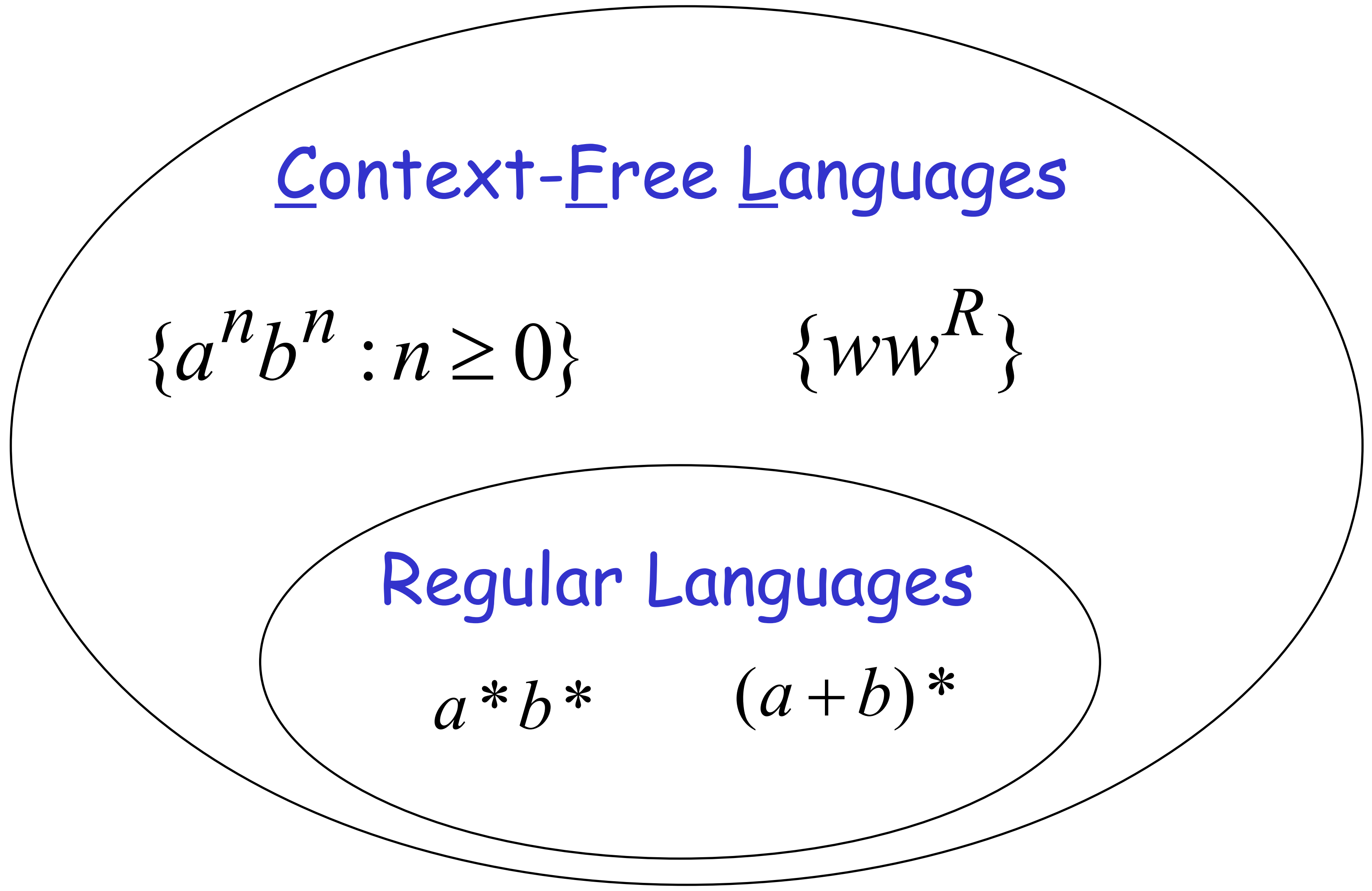


leads to the
context-free grammar
$G_M = (Q,\Sigma,R,q_1)$ with the rules

$q_1 \to 0q_1$ **q1** $\to 1q_2$

$q_2 \to 0q_3 \mid 1q_2 \mid \varepsilon$

$q_3 \to 0q_2 \mid 1q_2$



Designing CFG

首先，许多 CFG 是由几个较简单的 CFG 合并成的。如果你要为一个 CFL 构造 CFG，而这个 CFL 可以分成几个较简单的部分，那么就把它分成几部分，并且分别构造每一部分的文法。这几个文法能够很容易地合并在一起，构造出原先那个语言的文法，只需把它们的规则都放在一起，再加入新的规则 $S \to S_1 | S_2 | \cdots | S_k$，其中 S_1，S_2，…，S_k 是各个文法的起始变元。解决几个较简单的问题常常比解决一个复杂的问题容易。

$$\{0^n1^n | n \geq 0\} \cup \{1^n0^n | n \geq 0\}$$

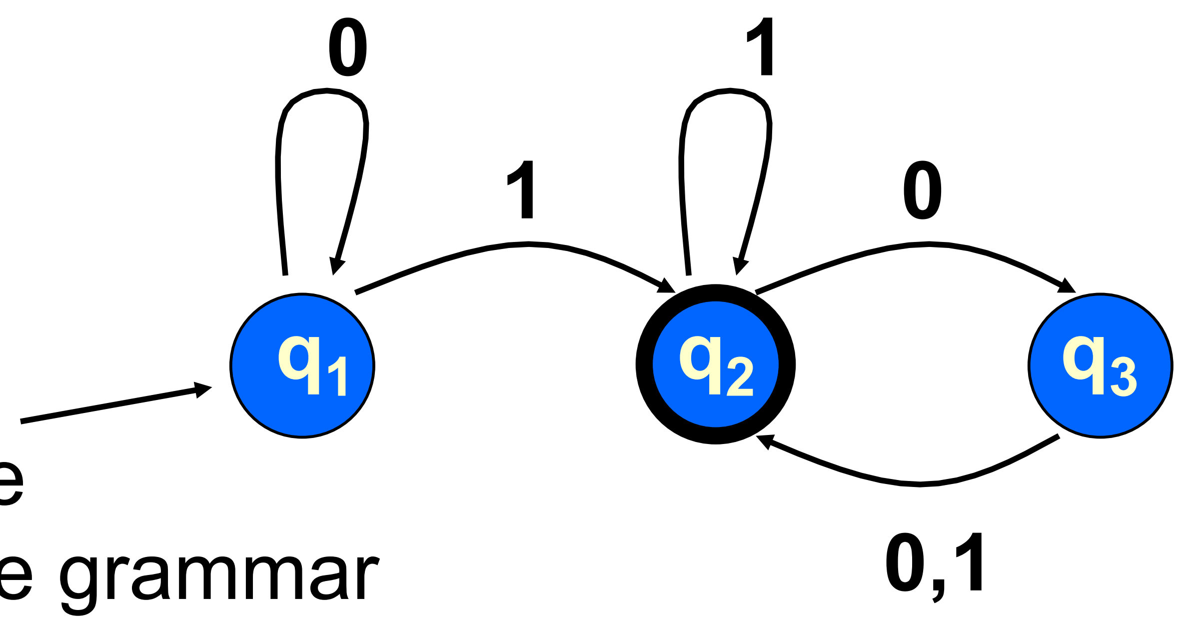
$$\begin{aligned} S_1 &\to 0S_11 \mid \varepsilon \\ S_2 &\to 1S_20 \mid \varepsilon \end{aligned}$$

$$\begin{aligned} S &\to S_1 \mid S_2 \\ S_1 &\to 0S_11 \mid \varepsilon \\ S_2 &\to 1S_20 \mid \varepsilon. \end{aligned}$$

Designing CFG

其次，如果这个语言碰巧是正则的，你可以先构造它的 DFA，然后再构造它的 CFG 就容易了。能够按下述做法把任何一台 DFA 转换成等价的 CFG。对于 DFA 的每一个状态 q_i，设一个变元 R_i。如果 $\delta(q_i, a) = q_j$ 是 DFA 中的一个转移，则把规则 $R_i \to aR_j$ 加入 CFG。如果 q_i 是 DFA 的接受状态，则把规则 $R_i \to \varepsilon$ 加入 CFG。设 q_0 是 DFA 的起始状态，则取 R_0 作为 CFG 的起始变元。能够验证所得到的 CFG 生成的语言与 DFA 识别的语言相同。

The DFA



leads to the
context-free grammar
$G_M = (Q,\Sigma,R,q_1)$ with the rules

$q_1 \to 0q_1$ **q1** $\to 1q_2$

$q_2 \to 0q_3 \mid 1q_2 \mid \varepsilon$

$q_3 \to 0q_2 \mid 1q_2$

Designing CFG

第三，某些上下文无关语言中的字符串有两个“相互联系”的子串，为了检查这两个子串中的一个是否正好对应于另一个，识别这种语言的机器需要记住关于这个子串的信息，而这个信息量是无界的。例如，在语言 $\{0^n1^n \mid n \geq 0\}$ 中就出现这种情况。为了检查字符串中 0 的个数是否等于 1 的个数，机器需要记住 0 的个数。对于这种情况，可以使用 $R \to uRv$ 形式的规则，它产生的字符串中包含 u 的部分对应包含 v 的部分。

- Can we design CFG for $\{0^{2n}1^{3n} \mid n \geq 0\}$?
- Yes, by “linking” the occurrence of 0's with the occurrence of 1's
- The desired CFG is:

$$S \to 00S111 \mid \varepsilon$$

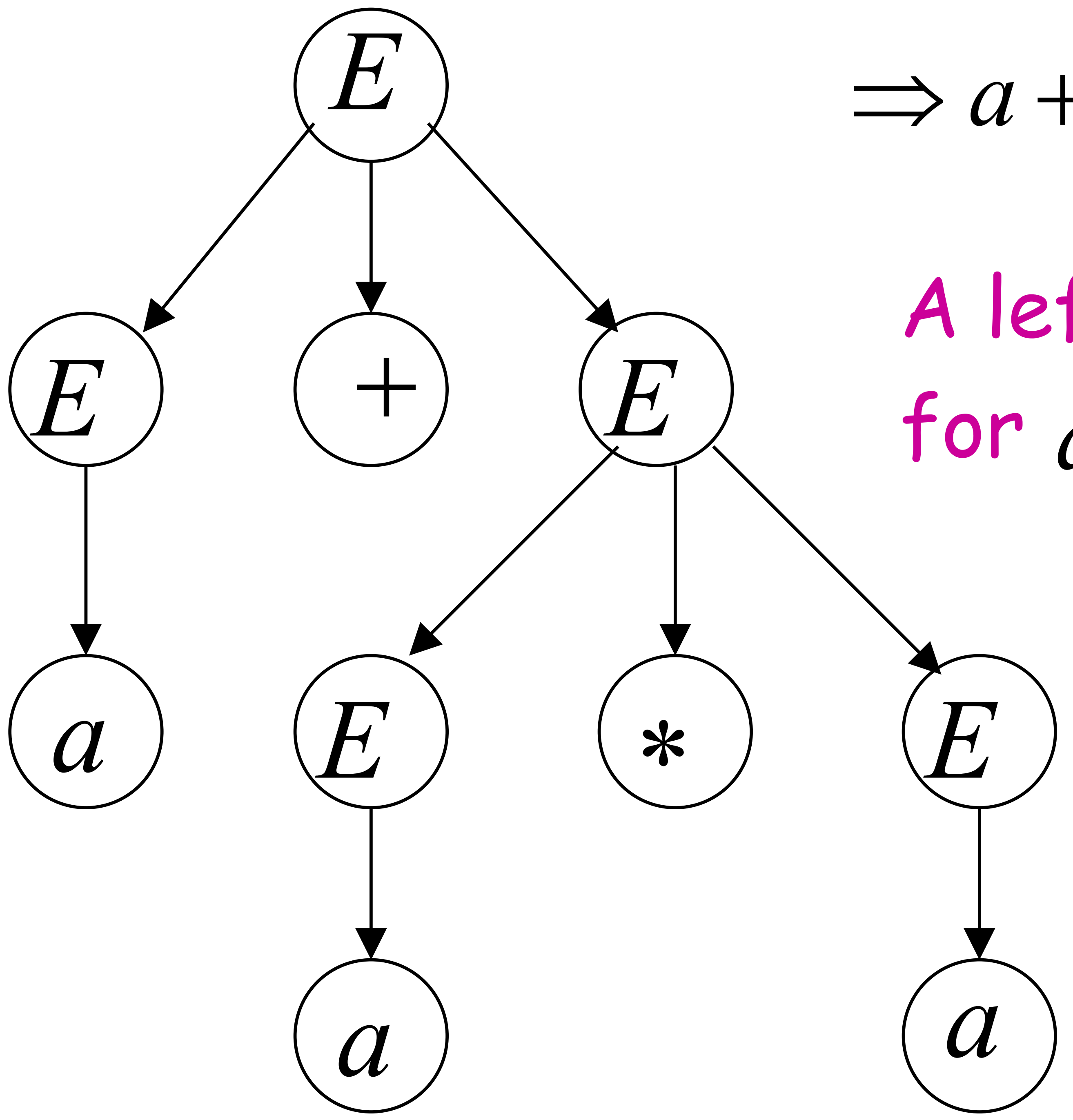
歧义性(Ambiguity)

I saw that girl with telescope

Grammar for mathematical expressions

$$\boxed{E \rightarrow E + E \ \mid E * E \mid (E) \mid a}$$

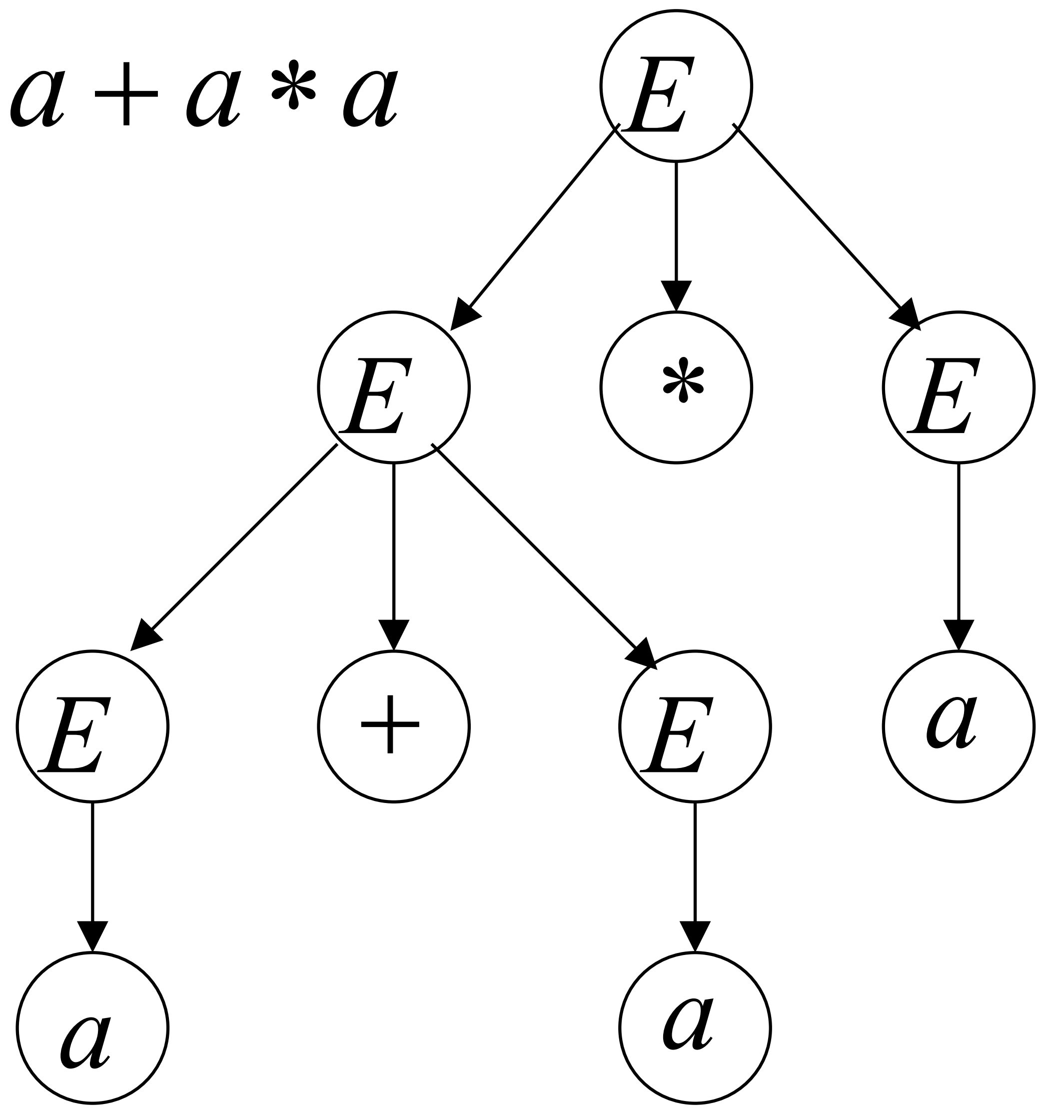
$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$



A leftmost derivation for $a + a * a$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

Another leftmost derivation for $a + a * a$



最左派生(*leftmost derivation*): 每一步都是替换剩下的最左边的变元，则该派生是最左派生。

Good Tree

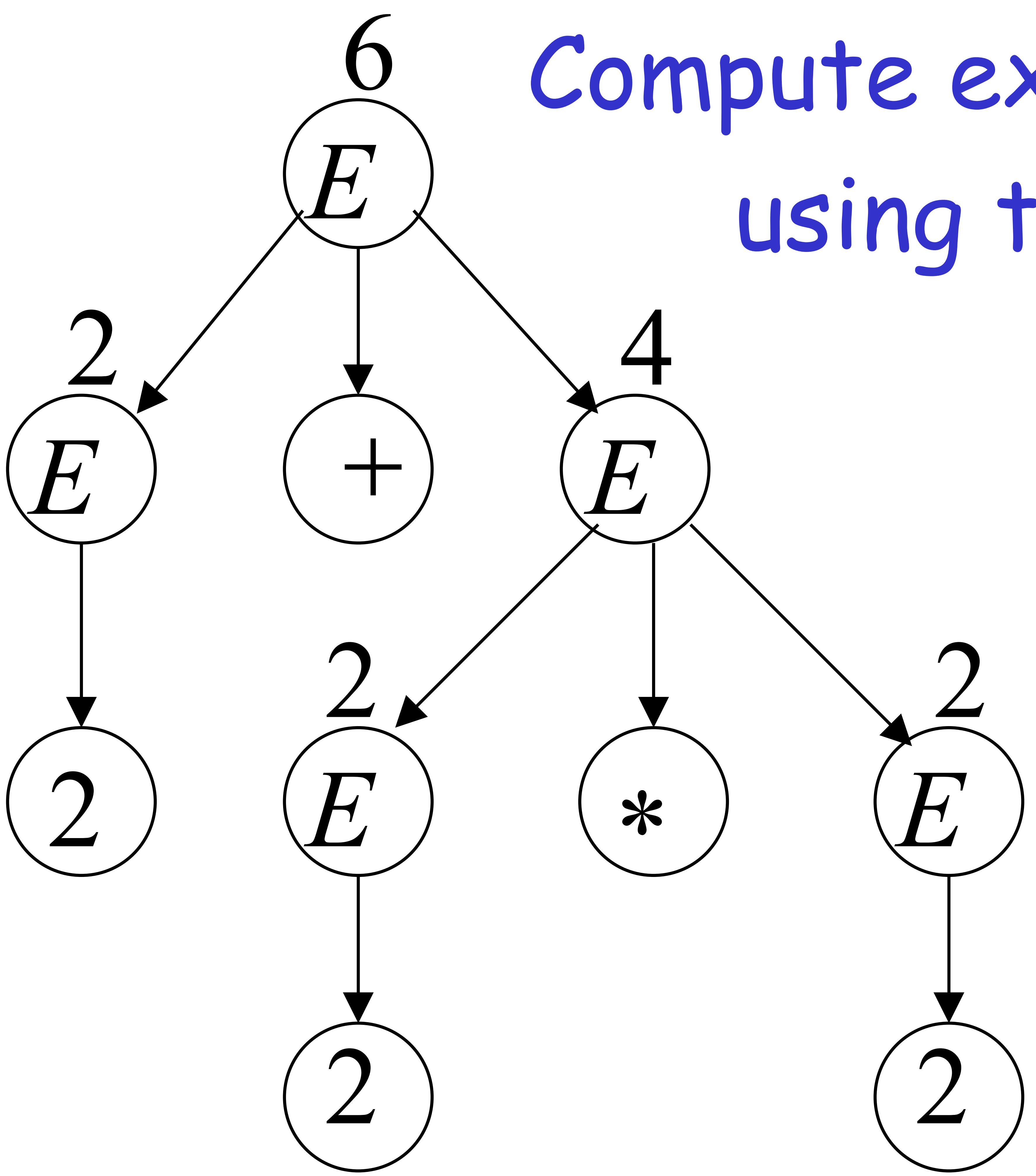
take $a = 2$

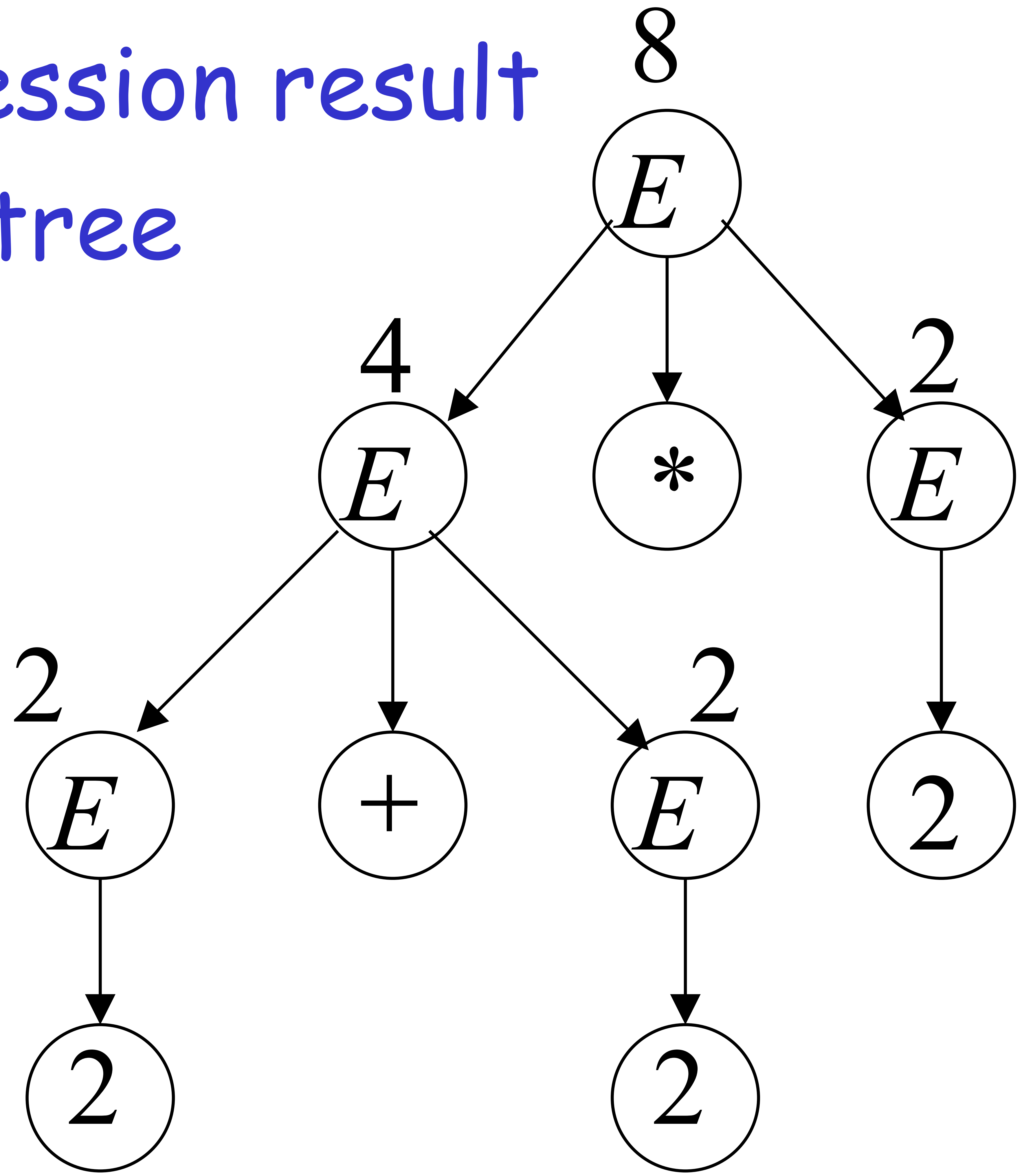
Bad Tree

$2 + 2 * 2 = 6$

$2 + 2 * 2 = 8$

Compute expression result using the tree





歧义性形式化定义

如果字符串w在上下文无关文法G中有两个或两个以上不同的最左派生，则称在G中歧义的产生字符串w。如果文法G歧义地产生某个字符串，则称G是歧义的。

上下文无关语言只能用歧义文法(Grammar)产生的称做**固有歧义的(*inherently ambiguous*)**。

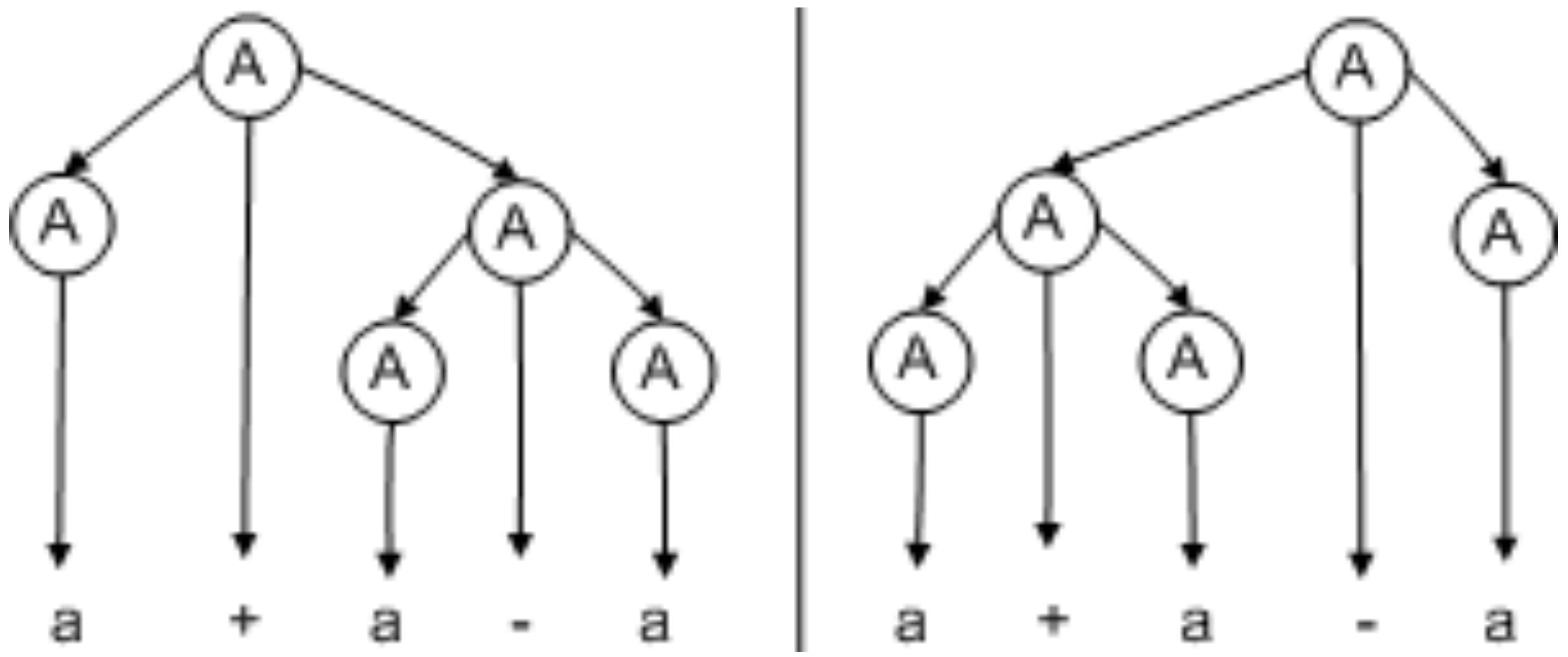
字符串 = 变元 + 终结符

固有歧义性举例

CFG: $A \to A + A \mid A - A \mid a$

is ambiguous since there are 2 leftmost derivations for the string **a+a-a**:

$$
\begin{aligned}
A &\to A + A \\
&\to a + A \\
&\to a + A - A \\
&\to a + a - A \\
&\to a + a - a
\end{aligned}
\qquad\qquad
\begin{aligned}
A &\to A - A \\
&\to A + A - A \\
&\to a + A - A \\
&\to a + a - A \\
&\to a + a - a
\end{aligned}
$$



等价CFG $A \to A + a \mid A - a \mid a$ → 非固有歧义

固有歧义性举例

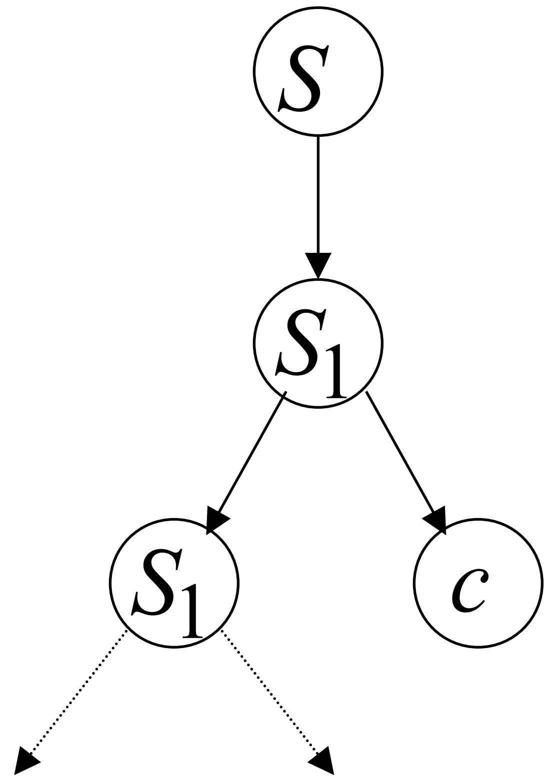
$$L = \{a^i b^j c^k \mid i = j \ or \ j = k\}$$

$$L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$$

$S \to S_1 \mid S_2$

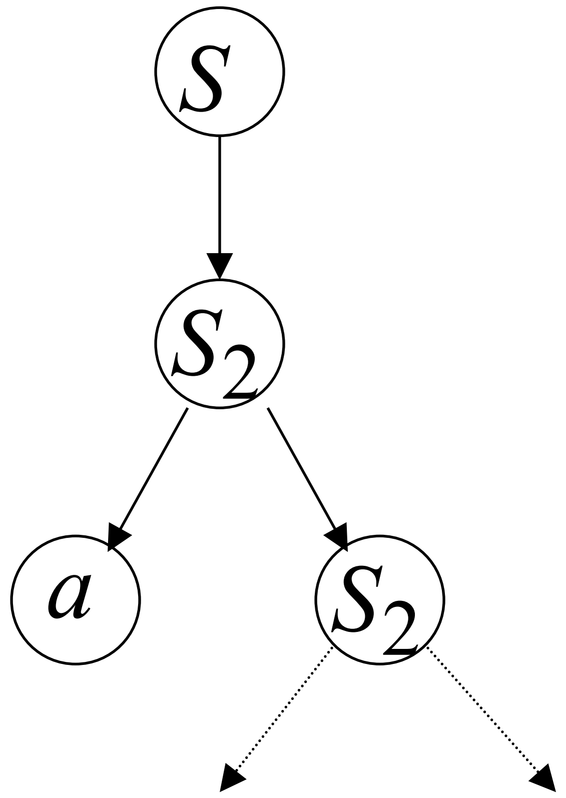
$S_1 \to S_1 c \mid A$

$A \to aAb \mid \varepsilon$



$S_2 \to aS_2 \mid B$

$B \to bBc \mid \varepsilon$



歧义性带来的问题

Two different derivation trees may cause problems in applications which use the derivation trees:

- Evaluating expressions
  - 2 + 2 * 2 = ?
- In general, in compilers for programming languages
  - **IF expr THEN stmt** ELSE stmt

Removing Ambiguity

- There is NO algorithm that can tell whether a CFG is ambiguous.
- There are techniques for eliminating ambiguity:
  - Adding non-terminals or dividing the variables into **factors(因子), terms(项), and expressions(表达式)**.

Removing Ambiguity from a + b * c

Removing Ambiguity from a + a * a

Equivalent

Ambiguous Grammar

$$E \to E + E$$
$$E \to E * E$$
$$E \to (E)$$
$$E \to a$$

Non-Ambiguous Grammar

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

generates the same language

CFG简化

$$S \to aB$$
$$A \to aaA$$
$$A \to abBc$$
$$B \to aA$$
$$B \to b$$

Substitute

$$B \to b$$

Equivalent grammar

$$S \to aB \mid ab$$
$$A \to aaA$$
$$A \to abBc \mid abbc$$
$$B \to aA$$

CFG简化

$$S \to aB \mid ab$$

$$A \to aaA$$

$$A \to abBc \mid abbc$$

$$B \to aA$$

Substitute

$$B \to aA$$

$$S \to \cancel{aB} \mid ab \mid aaA$$

$$A \to aaA$$

$$A \to \cancel{abBc} \mid abbc \mid abaAc$$

Equivalent grammar

CFG简化

In general:

$$A \rightarrow xBz$$

$$B \rightarrow y_1$$

Substitute

$$B \rightarrow y_1$$

$$A \rightarrow xBz \mid xy_1z$$

equivalent grammar

Nullable Variables

$$\lambda-\text{production}: \quad X \rightarrow \lambda$$

Nullable Variable: $\quad Y \Rightarrow \ldots \Rightarrow \lambda$

Example:

$$S \rightarrow aMb$$
$$M \rightarrow aMb$$
$$M \rightarrow \lambda$$

Nullable variable

$\lambda-\text{production}$

Substitute
$M \rightarrow \lambda$

$$S \rightarrow aMb \mid ab$$
$$M \rightarrow aMb \mid ab$$

Unit-Productions

Unit Production: $X \to Y$

(a single variable in both sides)

Example:

$S \to aA$

$A \to a$

$\boxed{A \to B}$

$\boxed{B \to A}$

$B \to bb$

Unit Productions

Removal of unit productions:

$S \to aA$

$A \to a$

$\cancel{A \to B}$

$B \to A$

$B \to bb$

Substitute

$A \to B$

$S \to aA \mid aB$

$A \to a$

$B \to A \mid B$

$B \to bb$

Unit-Productions

Unit productions of form $X \to X$

can be removed immediately

$$S \to aA \mid aB$$
$$A \to a$$
$$B \to A \mid \cancel{B}$$
$$B \to bb$$

Remove

$B \to B$

$$S \to aA \mid aB$$
$$A \to a$$
$$B \to A$$
$$B \to bb$$

$$S \to aA \mid aB$$
$$A \to a$$
$$\cancel{B \to A}$$
$$B \to bb$$

Substitute

$B \to A$

$$S \to aA \mid aB \mid aA$$
$$A \to a$$
$$B \to bb$$

Unit-Productions

Remove repeated productions

$$S \to aA \mid aB \mid \cancel{aA}$$
$$A \to a$$
$$B \to bb$$

Final grammar

$$S \to aA \mid aB$$
$$A \to a$$
$$B \to bb$$

Useless Productions

$S \to aSb$

$S \to \lambda$

$S \to A$

$A \to aA$ Useless Production

Some derivations never terminate...

$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \ldots \Rightarrow aa\ldots aA \Rightarrow \ldots$

停不下来

$S \to A$

$A \to aA$

$A \to \lambda$

$B \to bA$ Useless Production

Not reachable from S

派生不到

文法标准化(NORMAL FORM)

文法标准化重要性：

1. 简化语法规则，便于解析(parseing)支撑其他文法的证明（CNF，BNF…)

   => 易于程序实现

2. 代码易于维护

3. ?

乔姆斯基范式(Chomsky Normal Form)

Each productions has form:

一分为二

$$A \to BC$$

variable variable

or

终极化

$$A \to a$$

terminal

$A \in V$ and $B, C \in V \setminus \{S\}$

$a \in \Sigma$

For the start variable S we also allow the rule

$S \to \varepsilon$

CNF-Example

$$S1 \rightarrow AS \mid a$$
$$S \rightarrow AS$$
$$S \rightarrow a$$
$$A \rightarrow SA$$
$$A \rightarrow b$$

Chomsky
Normal Form

$$S \rightarrow AS$$
$$S \rightarrow AAS$$
$$A \rightarrow SA$$
$$A \rightarrow aa$$

Not Chomsky
Normal Form

CNF-Theorem

任一CFL都可以用乔姆斯基范式的CFG产生。

$$A \to BC$$

$$A \to a$$

$$allow\ S \to \epsilon$$

Ideas: CFL -> CFG -> CNF-CFG

The only reasons for a CFG not in CNF:

1. Start variable appears on right side
2. It has ε rules, such as $A \to \varepsilon$
3. It has unit rules, such as $A \to A$, or $B \to C$
4. Some rules does not have exactly two variables or one terminal on right side

<u>Outline of Proof</u>: **or Key Points of proof**

We rewrite every CFG in Chomsky normal form.

◆We do this by replacing, one-by-one, every rule that is not 'Chomsky'.

要点 一分为多 通过多个 一分为二 实现

Conversion to CNF

1. 添加一个新的起始变元S_0 和规则$S_0 \to S$,避免初始变元出现在规则右边

$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

$S_0 \to S$

$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

Conversion to CNF

2. 删除所有ϵ规则，删除所有的单一规则，同时添加对应改动的规则。

- After that, we remove $B \to \varepsilon$

$S_0 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

Before removing $B \to \varepsilon$

$S_0 \to S$
$S \to ASA \mid aB \mid a$
$A \to B \mid S \mid \varepsilon$
$B \to b$

After removing $B \to \varepsilon$

Conversion to CNF

2. 删除所有ϵ规则，删除所有的单一规则，同时添加对应改动的规则。

- After that, we remove $A \rightarrow \varepsilon$

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a$
$A \rightarrow B \mid S \mid \varepsilon$
$B \rightarrow b$

Before removing $A \rightarrow \varepsilon$

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S$
$A \rightarrow B \mid S$
$B \rightarrow b$

After removing $A \rightarrow \varepsilon$

Conversion to CNF

2. 删除所有ϵ规则，删除所有的单一规则，同时添加对应改动的规则。

- Then, we remove $S \rightarrow S$ and $S_0 \rightarrow S$

$S_0 \rightarrow S$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow B \mid S$
$B \rightarrow b$

After removing
$S \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow B \mid S$
$B \rightarrow b$

After removing
$S_0 \rightarrow S$

Conversion to CNF

2. 删除所有ϵ规则，删除所有的单一规则，同时添加对应改动的规则。

- Then, we remove $A \rightarrow B$

$$\begin{aligned} S_0 &\rightarrow ASA \mid aB \mid a \mid \\ &\quad SA \mid AS \\ S &\rightarrow ASA \mid aB \mid a \mid \\ &\quad SA \mid AS \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

Before removing $A \rightarrow B$

$$\begin{aligned} S_0 &\rightarrow ASA \mid aB \mid a \mid \\ &\quad SA \mid AS \\ S &\rightarrow ASA \mid aB \mid a \mid \\ &\quad SA \mid AS \\ A &\rightarrow b \mid S \\ B &\rightarrow b \end{aligned}$$

After removing $A \rightarrow B$

Conversion to CNF

2. 删除所有ϵ规则，删除所有的单一规则，同时添加对应改动的规则。

- Then, we remove $A \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow b \mid S$
$B \rightarrow b$

Before removing $A \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$
$B \rightarrow b$

After removing $A \rightarrow S$

Conversion to CNF

3. 转换规则为合适的形式

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$
$B \rightarrow b$

Before Step 4

$S_0 \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$
$S \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$
$A \rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS$
$B \rightarrow b$
$A_1 \rightarrow SA$
$U \rightarrow a$

After Step 4
Grammar is in CNF

Greinbach Normal Form

All productions have form:

$$A \to a\, V_1 V_2 \cdots V_k \qquad k \geq 0$$

symbol

variables

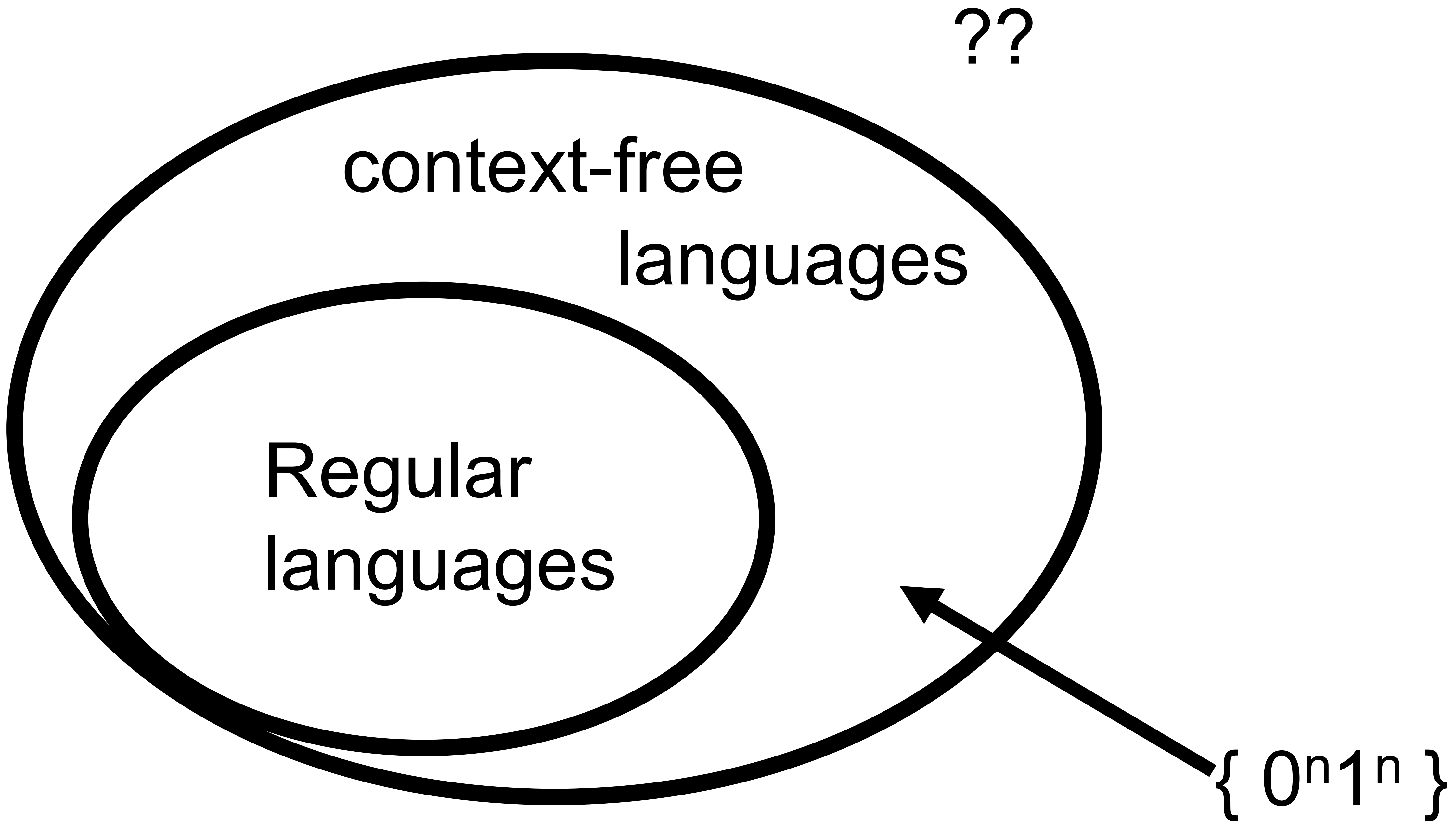
$$S \to cAB$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

Greinbach
Normal Form

$$S \to abSb$$
$$S \to aa$$

Not Greinbach
Normal Form

小结



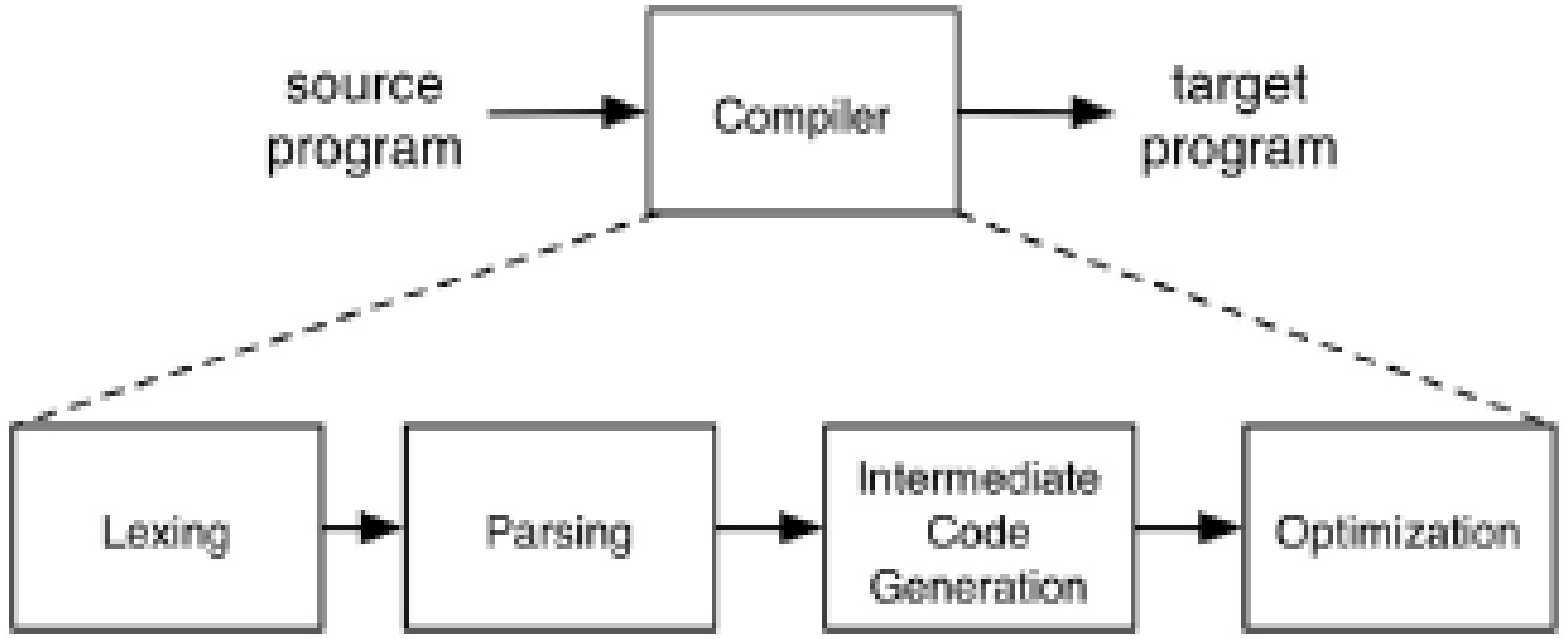
小结

REs turn raw text into a stream of tokens

- E.g., “if”, “then”, “identifier”, etc.
- This process is calling scanning or lexing
- Whitespace and comments are simply skipped
- These tokens become the input for the parser

CFGs turn tokens into parse trees

- This process is called parsing
- Parse trees become the input for the code generator



Any Questions?

加法乘法运算实现代码： *https://github.com/qijianpeng/Study/tree/master/compiler/salon_private/add_multi/Proj1_1*