形式语言与计算复杂性

第 2 章 Automata and Languages

2.2 Context-free Languages

黄文超

https://faculty.ustc.edu.cn/huangwenchao

$\longrightarrow$ 教学课程 $\longrightarrow$ 形式语言与计算复杂性

2.2 Context-Free Languages

Outline

2.2 Context-Free Languages

Outline

2.2 Context-Free Languages

Introduction

回顾: Finite Automata, e.g., M - Regular Languages, e.g., $L(M)$

- M recognizes $L(M)$
  - M has *extremely limited amount of memory*
  - $L(M)$ is equivalent to a *regular expression*
  - So M can be regarded as a *lexical analyzer*
- M *cannot* recognize Non-regular Languages, e.g., $\{0^n 1^n \mid n \geq 0\}$

Now, we introduce

- A *method* of describing languages: *context-free grammars*
  - syntax
- A class of *languages*: *Context-free languages*
  - a parser
- A class of *machines*: *Pushdown Automata*
  - extra component: stack

2.2 Context-Free Languages

Introduction

回顾: Finite Automata, e.g., M - Regular Languages, e.g., $L(M)$

- M recognizes $L(M)$
  - M has *extremely limited amount of memory*
  - $L(M)$ is equivalent to a *regular expression*
  - So M can be regarded as a *lexical analyzer*
- M *cannot* recognize Non-regular Languages, e.g., $\{0^n 1^n \mid n \geq 0\}$

Now, we introduce

- A *method* of describing languages: *context-free grammars*
  - syntax
- A class of *languages*: *Context-free languages*
  - a parser
- A class of *machines*: *Pushdown Automata*
  - extra component: stack

2.2 Context-Free Languages

Outline

2.2 Context-Free Languages

Outline

问: What are *context-free grammars*?

例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$
  - One variable is designated as the *start variable*, e.g., A

问: What are *context-free grammars*?
例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$
  - One variable is designated as the *start variable*, e.g., A

问: What are *context-free grammars*?
例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$
  - One variable is designated as the *start variable*, e.g., A

问: What are *context-free grammars*?
例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$
  - One variable is designated as the *start variable*, e.g., A

问: What are *context-free grammars*?
例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
    The variable symbols often are represented by capital letters.
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$
  - One variable is designated as the *start variable*, e.g., A

问: What are *context-free grammars*?

例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$

    The terminals are analogous to the input alphabet and often are represented by lowercase letters, numbers, or special symbols.
  - One variable is designated as the *start variable*, e.g., A

问: What are *context-free grammars*?

例: G_1

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

定义: rule, symbol, variable, terminal, start variable

- A *grammar* consists of a collection of *substitution rules*, also called *productions*
- *Each rule* appears as a line in the grammar, comprising a *symbol* and a *string* separated by an arrow
  - The symbol is called a *variable*, e.g., A, B
  - The other symbols in the string are called *terminals*, e.g., $0, 1, \#$
  - One variable is designated as the *start variable*, e.g., A

    It usually occurs on the left-hand side of the topmost rule.

问: What are *context-free grammars*?

例: G_1

$$A \to 0A1$$

$$A \to B$$

$$B \to \#$$

问: What are *context-free grammars*?

例: G_1

$$A \to 0A1$$

$$A \to B$$

$$B \to \#$$

问: How to *use* a *grammar* to *describe* a *language*?

答: By *generating each string* of that language in the following manner

1. Write down the *start variable*.
2. *Find a variable* that is written down and *a rule* that *starts* with that *variable*.
3. *Repeat* step 2 until *no variables* remain.

问: What are *context-free grammars*?

例: G_1

$$A \to 0A1$$

$$A \to B$$

$$B \to \#$$

问: How to *use* a *grammar* to *describe* a *language*?

答: By *generating each string* of that language in the following manner

1. Write down the *start variable*.
2. *Find a variable* that is written down and *a rule* that *starts* with that *variable*.
3. *Repeat* step 2 until *no variables* remain.

例: G_1 generates $000\#111$

解析: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

问: What are *context-free grammars*?

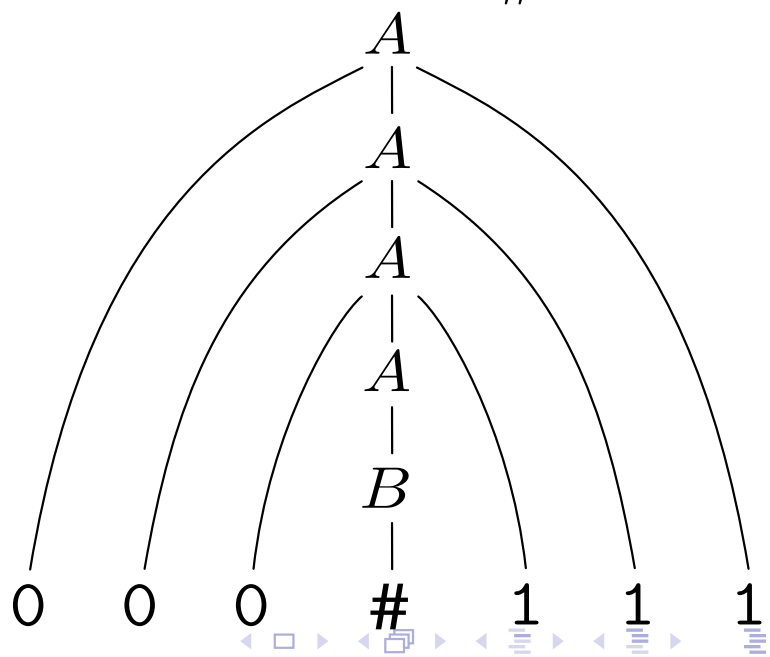
例: G_1

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

例: G_1 generates $000\#111$

解析: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

Parse Tree:



问: What are *context-free grammars*?

例: G_2, which describes a fragment of the English language

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\to \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\to \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\to \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\to \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\to \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\to \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\to \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\to \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\to \texttt{with}
\end{aligned}
$$

问: How to *use* a *grammar* to *describe* a *language*?

- a boy sees
- the boy sees a flower
- a girl with a flower likes the boy

问: What are *context-free grammars*?

例: G_2, which describes a fragment of the English language

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\to \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\to \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\to \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\to \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\to \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\to \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\to \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\to \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\to \texttt{with}
\end{aligned}
$$

问: How to *use* a *grammar* to *describe* a *language*?

- a boy sees
- the boy sees a flower
- a girl with a flower likes the boy

2.2 Context-Free Languages

Outline

定义: Context-free Grammars

A context-free grammar is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, *disjoint from* V, called the terminals,
3. R is a finite set of rules, with each rule being a *variable* and a *string* of *variables* and *terminals*
4. $S \in V$ is the start variable.

定义: Yield, Derive, Languages of Context-free Grammars

If u, v, and w are *strings*, and $A \to w$ is a *rule* of the grammar,

- uAv *yields* uwv, written $uAv \Rightarrow uwv$
- u *derives* v, written $u \overset{*}{\Rightarrow} v$
- The *language* of the grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$

定义: Context-free Grammars

A context-free grammar is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, *disjoint from* V, called the terminals,
3. R is a finite set of rules, with each rule being a *variable* and a *string* of *variables* and *terminals*
4. $S \in V$ is the start variable.

定义: Yield, Derive, Languages of Context-free Grammars

If u, v, and w are *strings*, and $A \to w$ is a *rule* of the grammar,

- uAv *yields* uwv, written $uAv \Rightarrow uwv$
- u *derives* v, written $u \overset{*}{\Rightarrow} v$
- The *language* of the grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$

定义: Context-free Grammars

A context-free grammar is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, *disjoint from* V, called the terminals,
3. R is a finite set of rules, with each rule being a *variable* and a *string* of *variables* and *terminals*
4. $S \in V$ is the start variable.

定义: Yield, Derive, Languages of Context-free Grammars

If u, v, and w are *strings*, and $A \to w$ is a *rule* of the grammar,

- uAv *yields* uwv, written $uAv \Rightarrow uwv$
- u *derives* v, written $u \overset{*}{\Rightarrow} v$
- The *language* of the grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$

定义: Context-free Grammars

A context-free grammar is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, *disjoint from* V, called the terminals,
3. R is a finite set of rules, with each rule being a *variable* and a *string* of *variables* and *terminals*
4. $S \in V$ is the start variable.

定义: Yield, Derive, Languages of Context-free Grammars

If u, v, and w are *strings*, and $A \to w$ is a *rule* of the grammar,

- uAv *yields* uwv, written $uAv \Rightarrow uwv$
- u *derives* v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$
- The *language* of the grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$

定义: Context-free Grammars

A context-free grammar is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the variables,
2. Σ is a finite set, *disjoint from* V, called the terminals,
3. R is a finite set of rules, with each rule being a *variable* and a *string* of *variables* and *terminals*
4. $S \in V$ is the start variable.

定义: Yield, Derive, Languages of Context-free Grammars

If u, v, and w are *strings*, and $A \to w$ is a *rule* of the grammar,

- uAv *yields* uwv, written $uAv \Rightarrow uwv$
- u *derives* v, written $u \overset{*}{\Rightarrow} v$
- The *language* of the grammar is $\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$

2.2 Context-Free Languages

Outline

例: Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$. The set of rules, R, is

$$S \to aSb \mid SS \mid \varepsilon$$

解析:

- This grammar generates strings such as $abab$, $aaabbb$, and $aababb$
- $L(G_3)$ is the language of all strings of properly *nested parentheses*.
  - $a \sim ($
  - $b \sim)$

例: Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$. The set of rules, R, is

$$S \to aSb \mid SS \mid \varepsilon$$

解析:

- This grammar generates strings such as $abab$, $aaabbb$, and $aababb$
- $L(G_3)$ is the language of all strings of properly *nested parentheses*.
  - $a \sim ($
  - $b \sim)$

例: Consider grammar $G_3 = (\{S\}, \{a, b\}, R, S)$. The set of rules, R, is

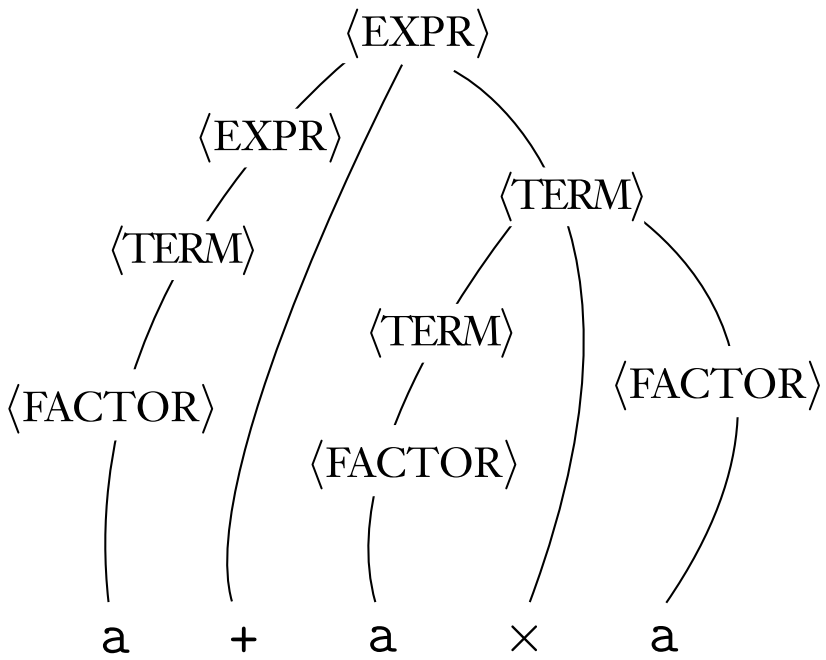
$$S \to aSb \mid SS \mid \varepsilon$$

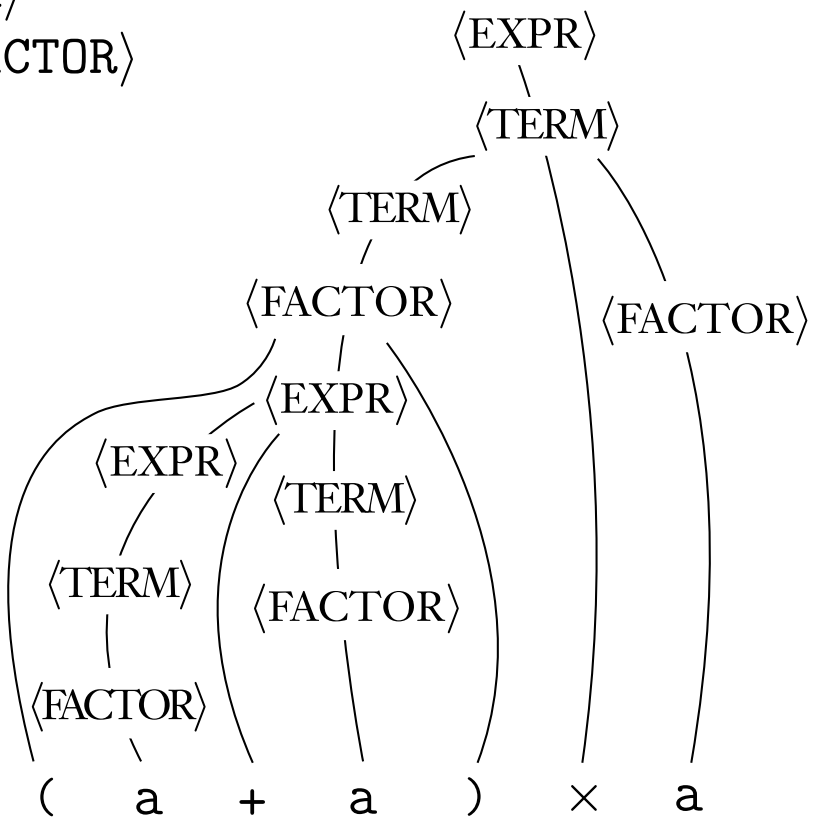
解析:

- This grammar generates strings such as $abab$, $aaabbb$, and $aababb$
- $L(G_3)$ is the language of all strings of properly *nested parentheses*.
  - $a \sim ($
  - $b \sim)$

例: Consider grammar $G_4 = (V, \Sigma, R, \langle \texttt{EXPR} \rangle)$

- V is $\{\langle \texttt{EXPR} \rangle, \langle \texttt{TERM} \rangle, \langle \texttt{FACTOR} \rangle\}$
- Σ is $\{a, +, \times, (,)\}$
- The rules are
  - $\langle \texttt{EXPR} \rangle \to \langle \texttt{EXPR} \rangle + \langle \texttt{TERM} \rangle \mid \langle \texttt{TERM} \rangle$
  - $\langle \texttt{TERM} \rangle \to \langle \texttt{TERM} \rangle \times \langle \texttt{FACTOR} \rangle \mid \langle \texttt{FACTOR} \rangle$
  - $\langle \texttt{FACTOR} \rangle \to (\langle \texttt{EXPR} \rangle) \mid a$





2.2 Context-Free Languages

Outline

问: Is there any problem for Context-free Grammars?
答: Ambiguity

问: What is Ambiguity?
答:

- If a grammar generates the *same string* in *several different ways*, we say that the string is derived *ambiguously* in that grammar.
- If a grammar generates some string *ambiguously*, we say that the *grammar* is *ambiguous*.

问: Is there any problem for Context-free Grammars?
答: Ambiguity

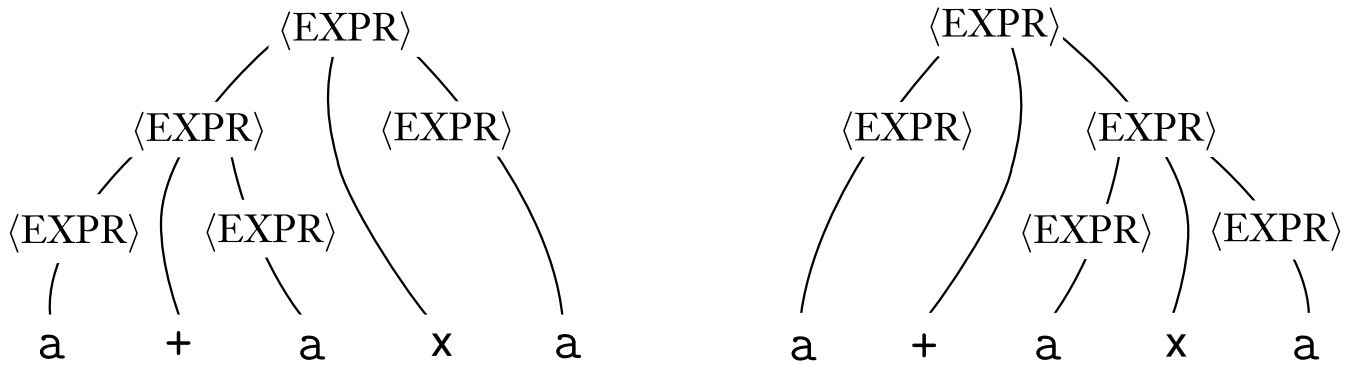
问: What is Ambiguity?
答:

- If a grammar generates the *same string* in *several different ways*, we say that the string is derived *ambiguously* in that grammar.
- If a grammar generates some string *ambiguously*, we say that the *grammar* is *ambiguous*.

问: What is Ambiguity?

例: Consider the grammar G_5

$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \mid (\langle \text{EXPR} \rangle) \mid a$$



问: What is Ambiguity?
Recall G_2

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\to \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\to \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\to \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\to \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\to \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\to \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\to \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\to \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\to \texttt{with}
\end{aligned}
$$

the girl touches the boy with the flower

问: 如何准确定义 Ambiguity?
答: 先定义 leftmost derivation:

准备: leftmost derivation

- A *derivation* of a string w in a grammar G is a *leftmost derivation*, if at *every step* the *leftmost remaining* variable is the one *replaced*.

$$\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a}\ \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB} \rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}$$

问: 如何准确定义 Ambiguity?
答: 先定义 leftmost derivation:

准备: leftmost derivation

- A *derivation* of a string w in a grammar G is a *leftmost derivation*, if at *every step* the *leftmost remaining* variable is the one *replaced*.

$$\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a}\ \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB} \rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}$$

问: 如何准确定义 Ambiguity?
答: 先定义 leftmost derivation:

准备: leftmost derivation

- A *derivation* of a string w in a grammar G is a *leftmost derivation*, if at *every step* the *leftmost remaining* variable is the one *replaced*.

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\Rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{CMPLX-NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a}\ \langle \text{NOUN} \rangle \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB-PHRASE} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{CMPLX-VERB} \rangle \\
&\Rightarrow \texttt{a boy}\ \langle \text{VERB} \rangle \\
&\Rightarrow \texttt{a boy sees}
\end{aligned}
$$

定义: Ambiguity

- A string w is derived *ambiguously* in context-free grammar G if it has two or more different *leftmost derivations*.
- Grammar G is *ambiguous* if it generates some string *ambiguously*.

回顾: Consider the grammar G_5

$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \mid (\langle \text{EXPR} \rangle) \mid a$$

定义: Ambiguity

- A string w is derived *ambiguously* in context-free grammar G if it has two or more different *leftmost derivations*.
- Grammar G is *ambiguous* if it generates some string *ambiguously*.

回顾: Consider the grammar G_5

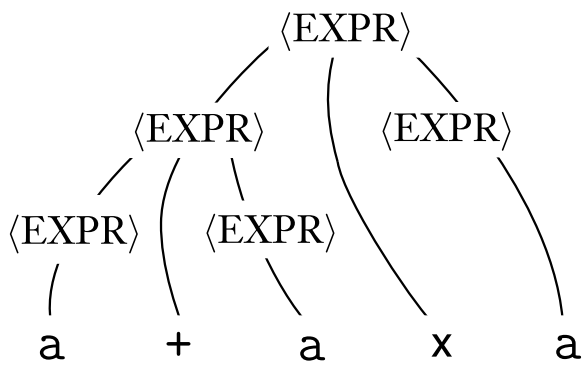
$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \mid (\langle \text{EXPR} \rangle) \mid a$$

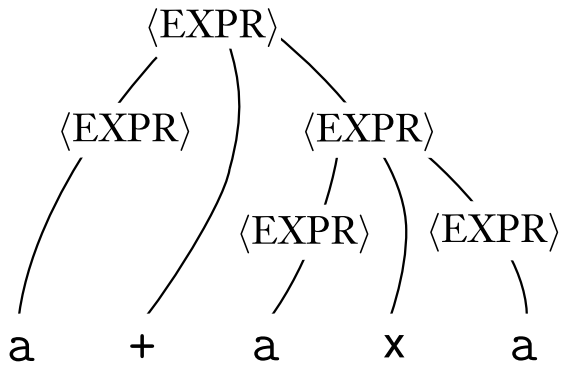
定义: Ambiguity

- A string w is derived *ambiguously* in context-free grammar G if it has two or more different *leftmost derivations*.
- Grammar G is *ambiguous* if it generates some string *ambiguously*.

回顾: Consider the grammar G_5

$$\langle \texttt{EXPR} \rangle \to \langle \texttt{EXPR} \rangle + \langle \texttt{EXPR} \rangle \mid \langle \texttt{EXPR} \rangle \times \langle \texttt{EXPR} \rangle \mid (\langle \texttt{EXPR} \rangle) \mid a$$





2.2 Context-Free Languages

Outline

问: Can context-free grammars be simplified?
答: Chomsky Normal Form, one of the *simplest* and *most useful* forms

定义: Chomsky Normal Form

A context-free grammar is in *Chomsky normal form* if every rule is of the form

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B, and C are *any* variables —except that B and C *may not* be the start variable.
In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

问: Can context-free grammars be simplified?
答: Chomsky Normal Form, one of the *simplest* and *most useful* forms

定义: Chomsky Normal Form

A context-free grammar is in *Chomsky normal form* if every rule is of the form

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B, and C are *any* variables —except that B and C *may not* be the start variable.
In addition, we permit the rule $S \to \varepsilon$, where S is the start variable.

定理

Any context-free language is *generated* by a context-free grammar in *Chomsky normal form*.

证明思路

1. add a new start variable
2. eliminate all ε-rules of form $A \to \varepsilon$ not involving the start variable
3. eliminate all unit rules of the form $A \to B$
4. convert the remaining rules into the proper form

定理

Any context-free language is *generated* by a context-free grammar in *Chomsky normal form*.

证明思路

1. add a new start variable
2. eliminate all ε-rules of form $A \to \varepsilon$ not involving the start variable
3. eliminate all unit rules of the form $A \to B$
4. convert the remaining rules into the proper form

定理

Any context-free language is *generated* by a context-free grammar in *Chomsky normal form*.

证明

1. add a new start variable

   add a new start variable S_0 and the rule $S_0 \to S$, where S was the original start variable

   Result: This change guarantees that the *start variable doesn't occur* on the *right-hand side* of a rule. See Def.

定理

Any context-free language is *generated* by a context-free grammar in *Chomsky normal form*.

证明

2. eliminate all ε-rules of the form $A \to \varepsilon$ not involving the start variable
   - *remove* an ε-rule $A \to \varepsilon$, where A is not the start variable
   - if $R \to uAv$ is a rule in which u and v are strings of variables and terminals, we *add* rule $R \to uv$.
   - if $R \to A$ is a rule, we *add* $R \to \varepsilon$ unless we had previously removed the rule $R \to \varepsilon$.
   - repeat

定理

Any context-free language is *generated* by a context-free grammar in *Chomsky normal form*.

证明

3. eliminate all unit rules of the form $A \to B$
   - remove a unit rule $A \to B$
   - whenever a rule $B \to u$ appears, we add the rule $A \to u$ unless this was a unit rule previously removed
   - repeat

定理

Any context-free language is *generated* by a context-free grammar in *Chomsky normal form*.

证明

4. convert the remaining rules into the proper form
   - replace each rule $A \to u_1 u_2 \cdots u_k$, where $k \geq 3$ and each u_i is a variable or terminal symbol, with the rules $A \to u_1 A_1$, $A_1 \to u_2 A_2$, $A_2 \to u_3 A_3, \ldots,$ and $A_{k-2} \to u_{k-1} u_k$
   - We replace any terminal u_i in the preceding rule(s) with the new variable U_i and add the rule $U_i \to u_i$.

例: Transform the following CFG into Chomsky Normal Form ▸ Proof

$$
\begin{aligned}
S &\to ASA \mid \mathtt{a}B \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon
\end{aligned}
$$

例: Transform the following CFG into Chomsky Normal Form ▸ Proof

$$
\begin{aligned}
S &\to ASA \mid \mathtt{a}B \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon
\end{aligned}
$$

Step 1:

$$
\begin{aligned}
S &\to ASA \mid \mathtt{a}B \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{S_0} &\to \boldsymbol{S} \\
S &\to ASA \mid \mathtt{a}B \\
A &\to B \mid S \\
B &\to \mathtt{b} \mid \varepsilon
\end{aligned}
$$

例: Transform the following CFG into Chomsky Normal Form ▸ Proof

Step 1:

$$\begin{aligned} S &\to ASA \mid \mathtt{a}B \\ A &\to B \mid S \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned}$$

$$\begin{aligned} \boldsymbol{S_0} &\to \boldsymbol{S} \\ S &\to ASA \mid \mathtt{a}B \\ A &\to B \mid S \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned}$$

Step 2:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathbf{a} \\ A &\to B \mid S \mid \boldsymbol{\varepsilon} \\ B &\to \mathtt{b} \mid \varepsilon \end{aligned}$$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\ A &\to B \mid S \mid \varepsilon \\ B &\to \mathtt{b} \end{aligned}$$

例: Transform the following CFG into Chomsky Normal Form ▸ Proof

Step 2:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathbf{a} \\ A &\to B \mid S \mid \boldsymbol{\varepsilon} \\ B &\to \mathtt{b} \mid \boldsymbol{\varepsilon} \end{aligned}$$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\ A &\to B \mid S \mid \boldsymbol{\varepsilon} \\ B &\to \mathtt{b} \end{aligned}$$

Step 3a:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \mid S \\ A &\to B \mid S \\ B &\to \mathtt{b} \end{aligned}$$

$$\begin{aligned} S_0 &\to S \mid \boldsymbol{ASA} \mid \mathbf{a}\boldsymbol{B} \mid \mathbf{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to B \mid S \\ B &\to \mathtt{b} \end{aligned}$$

例: Transform the following CFG into Chomsky Normal Form ▸ Proof

Step 3a:

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \mid S \\ A &\to B \mid S \\ B &\to \mathtt{b} \end{aligned}$$

$$\begin{aligned} S_0 &\to S \mid \boldsymbol{ASA} \mid \mathbf{a}\boldsymbol{B} \mid \mathbf{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to B \mid S \\ B &\to \mathtt{b} \end{aligned}$$

Step 3b:

$$\begin{aligned} S_0 &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to B \mid S \mid \mathbf{b} \\ B &\to \mathtt{b} \end{aligned}$$

$$\begin{aligned} S_0 &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to S \mid \mathtt{b} \mid \boldsymbol{ASA} \mid \mathbf{a}\boldsymbol{B} \mid \mathbf{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ B &\to \mathtt{b} \end{aligned}$$

例: Transform the following CFG into Chomsky Normal Form ▸ Proof

Step 3b:

$$\begin{aligned} S_0 &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to B \mid S \mid \mathtt{b} \\ B &\to \mathtt{b} \end{aligned}$$

$$\begin{aligned} S_0 &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S &\to ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A &\to S \mid \mathtt{b} \mid \boldsymbol{ASA} \mid \mathtt{a}\boldsymbol{B} \mid \mathtt{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ B &\to \mathtt{b} \end{aligned}$$

Step 4:

$$\begin{aligned} S_0 &\to AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ S &\to AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ A &\to \mathtt{b} \mid AA_1 \mid UB \mid \mathtt{a} \mid SA \mid AS \\ A_1 &\to SA \\ U &\to \mathtt{a} \\ B &\to \mathtt{b} \end{aligned}$$

2.2 Context-Free Languages

Outline

Now, we introduce See Introduction

- A class of *machines*: *Pushdown Automata*
  - extra component: stack

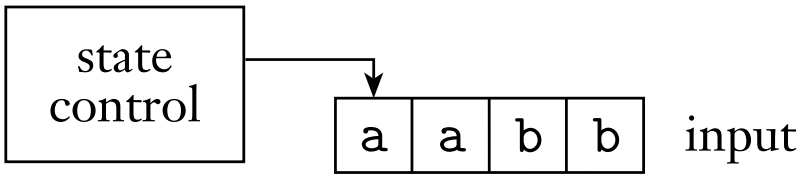


图: Finite Automata

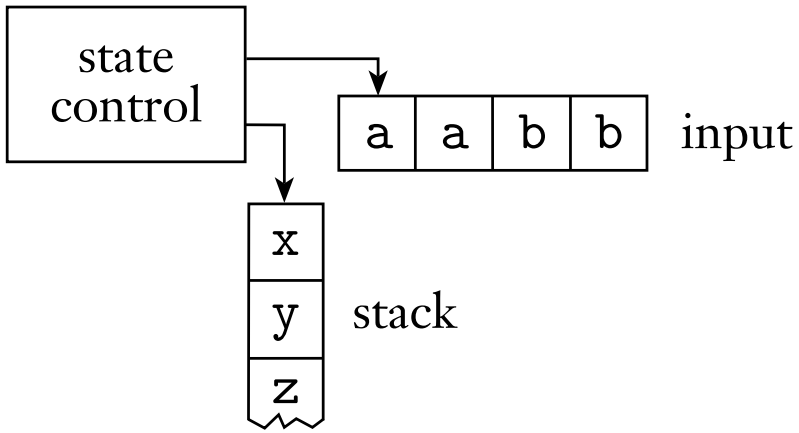


图: Pushdown Automata

Outline

定义: Pushdown Automata (PDA)

A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states
2. Σ is the *input* alphabet
3. Γ is the *stack* alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function
5. $q_0 \in Q$ is the *start state*, and
6. $F \subseteq Q$ is the set of *accept states*

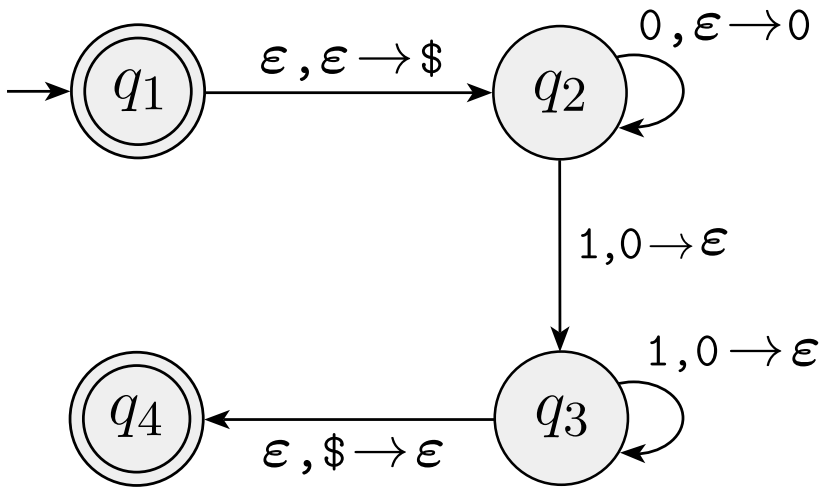
例 1: PDA M_1 recognizes the language $\{0^n 1^n \mid n \geq 0\}$.

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, \$\}$
- $F = \{q_1, q_4\}$
- δ is given by the following table, wherein blank entries signify $\emptyset$

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

例 1: PDA M_1 recognizes the language $\{0^n 1^n \mid n \geq 0\}$.

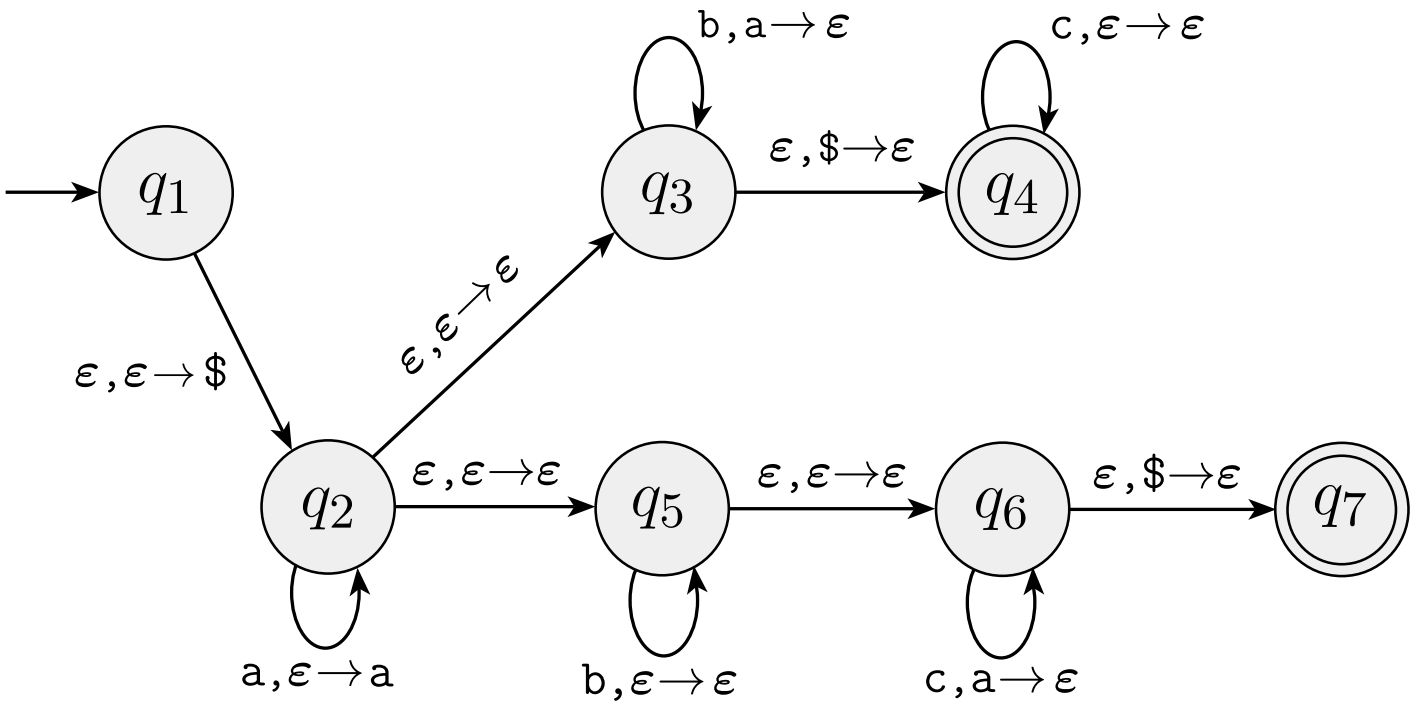
- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, \$\}$
- $F = \{q_1, q_4\}$
- δ is given by the following table, wherein blank entries signify $\emptyset$



Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$			$\{(q_4, \varepsilon)\}$		
q_4									

$a, b \to c$: when the machine is reading an a from the input, it may replace the symbol b on the top of the stack with a c

例 2: PDA M_2 recognizes $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$



Outline

定理

A language is context free if and only if some pushdown automaton recognizes it

引理 1

If a language is context free, then some pushdown automaton recognizes it

引理 2

If a pushdown automaton recognizes some language, then it is context free

引理 1

If a language is context free, then some pushdown automaton recognizes it

证明思路 ▸ Example

- Place the marker symbol \$ and the start variable on the stack.
- Repeat the following steps forever
  - If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the *string* on the right-hand side of the rule
  - If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a
    - If they match, repeat
    - If they do not match, reject on this branch of the nondeterminism
  - If the top of stack is the *symbol \$*, enter the *accept state*. Doing so accepts the input if it has all been read

引理 1

If a language is context free, then some pushdown automaton recognizes it

准备: 证明过程中的问题: How to substitute a variable?
答: For instance, PDA may read the a and pop the s, then push the string xyz onto the stack and go on to the state r.

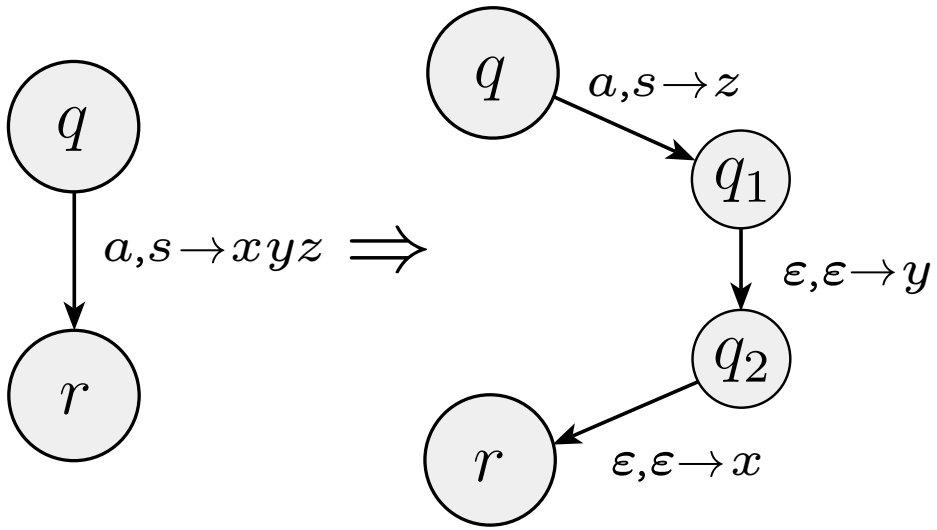
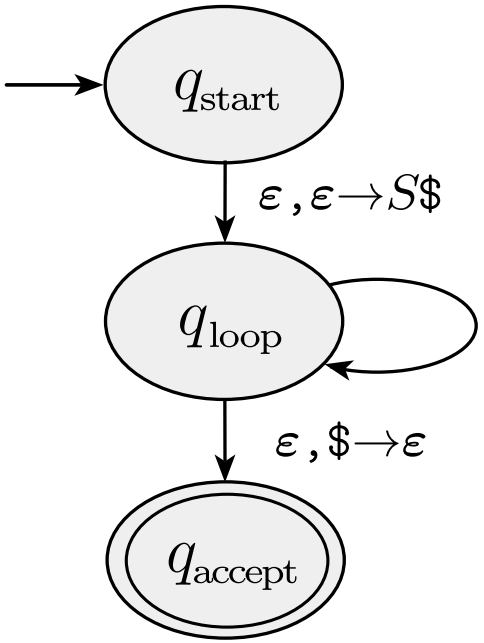


图: Implementing the shorthand $(r, xyz) \in \delta(q, a, s)$

引理 1

If a language is context free, then some pushdown automaton recognizes it

证明过程图例: ▸ Proving process of Lemma1



$\varepsilon, A \to w$ for rule $A \to w$

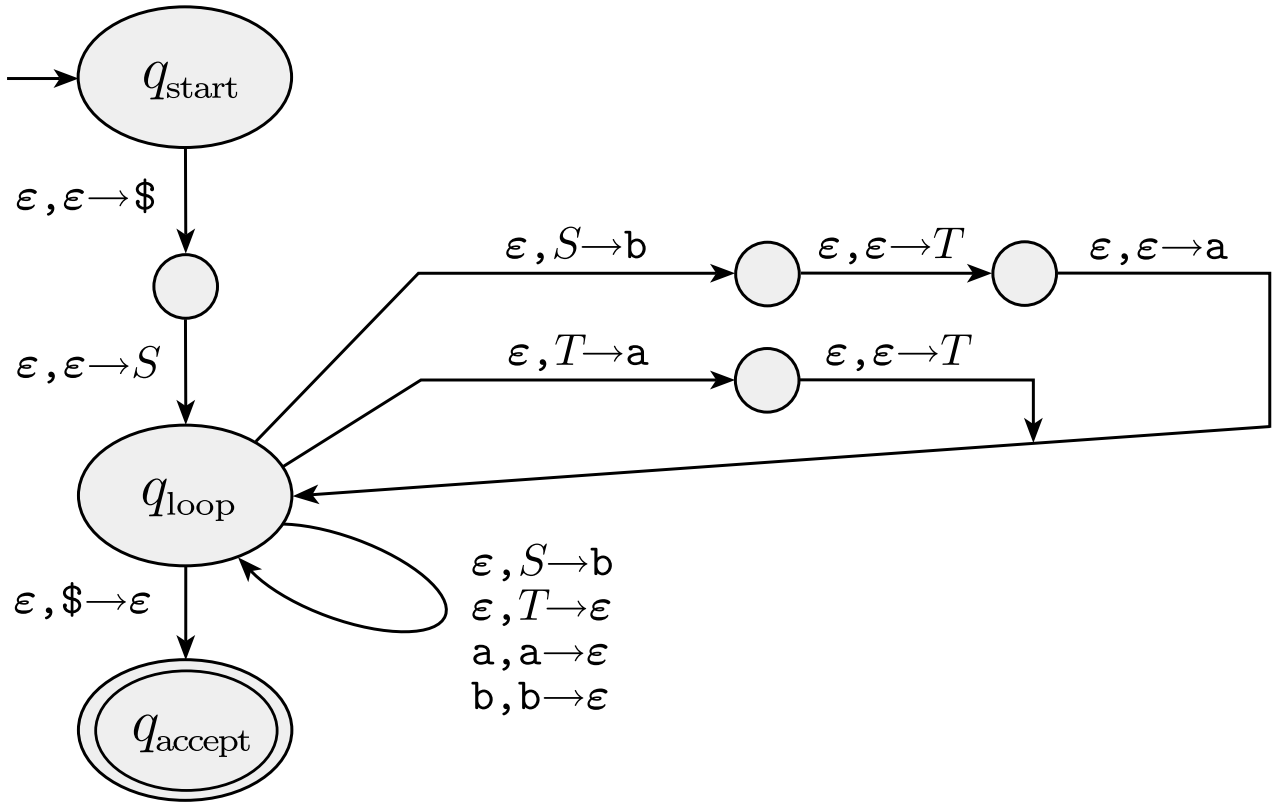
$\mathtt{a}, \mathtt{a} \to \varepsilon$ for terminal a

引理 1

If a language is context free, then some pushdown automaton recognizes it

例: Construct PDA from the following CFG:

$$S \to aTb \mid b$$
$$T \to Ta \mid \varepsilon$$



引理 2

If a pushdown automaton recognizes some language, then it is context free

证明思路

0. Design a grammar which has a variable A_{pq} for *each pair* of states p and q in P
   - This variable *generates all the strings* that *can* take PDA P from p with an *empty stack (note: assumption)* to q with an *empty stack (note: conclusion)*.
     - Observation: *such strings can also* take P from p to q, regardless of the stack contents at p, leaving the *stack* at q in the *same condition* as it was at p
1. Modify P slightly
2. Consider two cases occurring during P*'s computation on* x

引理 2

If a pushdown automaton recognizes some language, then it is context free

证明思路

0. Design a grammar which has a variable A_{pq} for *each pair* of states p and q in P
1. Modify P slightly
   - It has a single accept state, q_{accept}
   - It empties its stack before accepting
   - Each transition *either pushes* a symbol onto the stack (a push move) *or pops* one off the stack (a pop move), but it does not do both at the same time
2. Consider two cases occurring during *P's computation on x*

引理 2

If a pushdown automaton recognizes some language, then it is context free

证明思路

0. Design a grammar which has a variable A_{pq} for *each pair* of states p and q in P
1. Modify P slightly
2. Consider two cases occurring during *P's computation on x*
   - Case 1: the symbol *popped* at the *end* is the symbol that was *pushed* at the *beginning*
     - the stack could be empty only at the beginning and end of P's computation on x
     - Design *rule* $A_{pq} \to aA_{rs}b$, where a is the input read at the first move, b is the input read at the last move

引理 2

If a pushdown automaton recognizes some language, then it is context free

证明思路

0. Design a grammar which has a variable A_{pq} for *each pair* of states p and q in P
1. Modify P slightly
2. Consider two cases occurring during *P's computation on x*
   - Case 2: the symbol *popped* at the *end* is *not* the symbol that was *pushed* at the *beginning*
     - Design *rule* $A_{pq} \to A_{pr}A_{rq}$, where r is the state when the stack becomes empty

引理 2

If a pushdown automaton recognizes some language, then it is context free

证明 (Proof by construction)

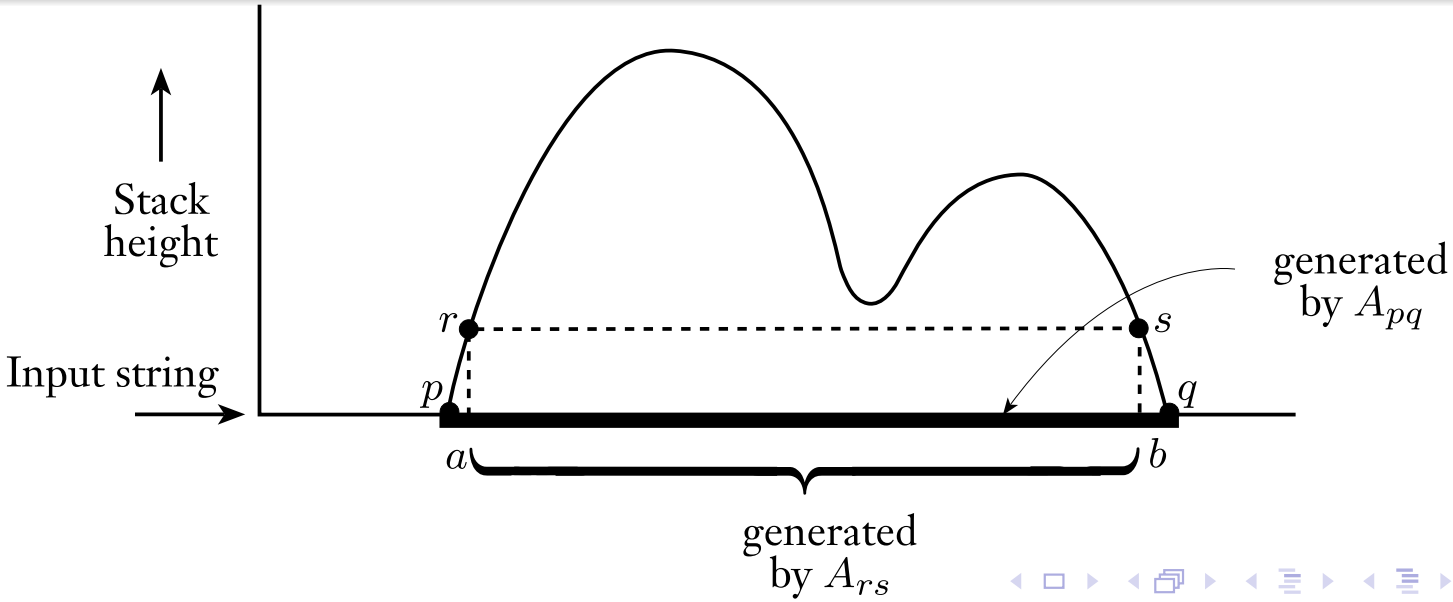
Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$ and *construct* G. The variables of G are $\{A_{pq} \mid p, q \in Q\}$. The *start variable* is $A_{q_0, q_{\text{accept}}}$.

引理 2

If a pushdown automaton recognizes some language, then it is context free

证明 (Proof by construction)

- 1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \to aA_{rs}b$ in G.

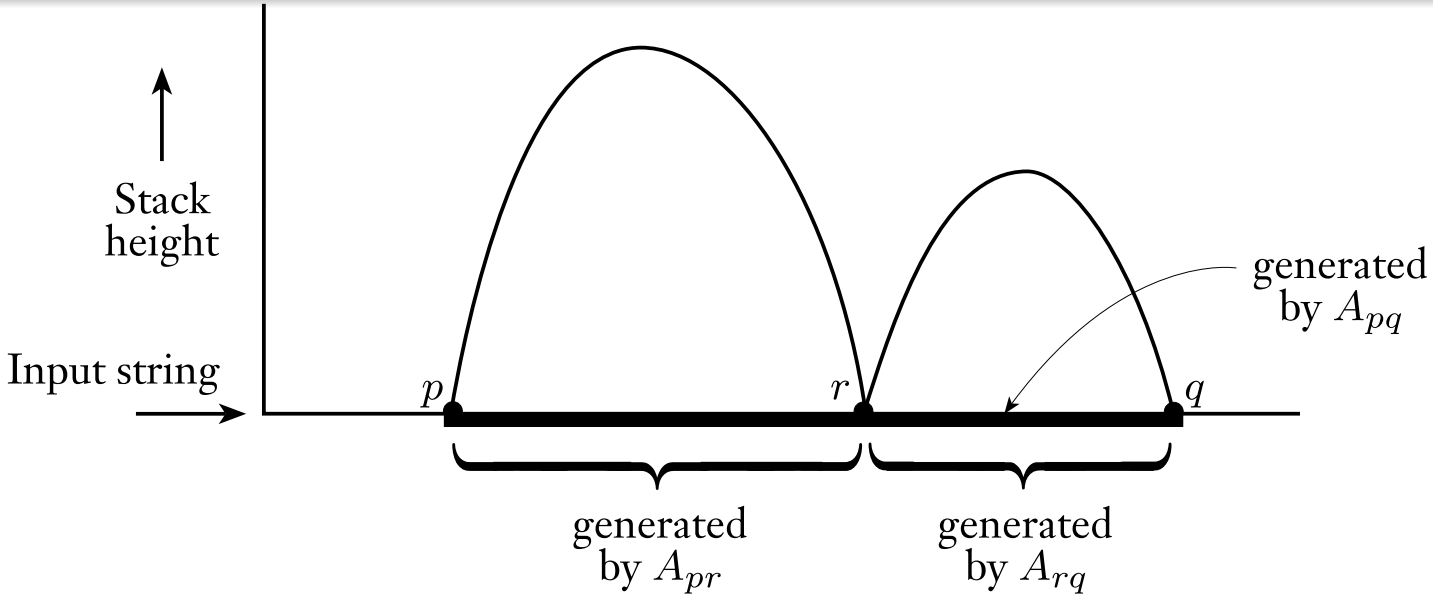


引理 2

If a pushdown automaton recognizes some language, then it is context free

证明 (Proof by construction)

- 2. For each $p, q, r \in Q$, put the rule $A_{pq} \to A_{pr} A_{rq}$ in G



引理 2

If a pushdown automaton recognizes some language, then it is context free

证明 (Proof by construction)

- 3. Finally, for each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G

引理 2

If a pushdown automaton recognizes some language, then it is context free

Claim 1

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

Claim 2

If x can bring P from p with empty stack to q with empty stack, A_{pq} generates x.

If claim 1 and 2 are proved, then P recognizes $A_{q_0,q_{\text{accept}}}$ and $A_{q_0,q_{\text{accept}}}$ is context free.

问题: How to prove claim 1 and 2?

答: 见下页

引理 2

If a pushdown automaton recognizes some language, then it is context free

Claim 1

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

Claim 2

If x can bring P from p with empty stack to q with empty stack, A_{pq} generates x.

If claim 1 and 2 are proved, then P recognizes $A_{q_0,q_{\text{accept}}}$ and $A_{q_0,q_{\text{accept}}}$ is context free.

问题: How to prove claim 1 and 2?

答: 见下页

Claim 1

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

证明: (Prove by Induction)

思路: Induction on the number of steps in the derivation of x from A_{pq}.

Basis: The derivation has 1 step.

- A derivation with a single step *must use a rule* whose *right*-hand side contains *no variables*.
- The *only rules* in G where no variables occur on the right-hand side are $A_{pp} \to \varepsilon$.
- Clearly, *input* ε takes P from p with empty stack to p with empty stack so the basis is proved.

Claim 1

If $\underline{A_{pq}}$ generates x, then x can bring $\underline{P}$ from p with empty stack to q with empty stack.

证明: (Prove by Induction)

思路: Induction on the number of steps in the derivation of x from A_{pq}.

Basis: The derivation has 1 step.

- A derivation with a single step *must use a rule* whose *right*-hand side contains *no variables*.
- The *only rules* in G where no variables occur on the right-hand side are $\underline{A_{pp} \to \varepsilon}$.
- Clearly, *input* ε takes P from p with empty stack to p with empty stack so the basis is proved.

Claim 1

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

证明: (Prove by Induction)

思路: Induction on the number of steps in the derivation of x from A_{pq}.

Induction step: Assume true for derivations of length at most k, where $k \geq 1$, and prove true for derivations of length $k + 1$

Claim 1

If $\underline{A_{pq}}$ generates x, then x can bring $\underline{P}$ from p with empty stack to q with empty stack.

证明: (Prove by Induction)

Induction step: Suppose that $A_{pq} \stackrel{*}{\Rightarrow} x$ with $k+1$ steps.

Claim 1

If $\underline{A_{pq}}$ generates x, then x can bring $\underline{P}$ from p with empty stack to q with empty stack.

证明: (Prove by Induction)

Induction step: Suppose that $A_{pq} \stackrel{*}{\Rightarrow} x$ with $k+1$ steps.

The first step in this derivation is either $A_{pq} \Rightarrow aA_{rs}b$ or $A_{pq} \Rightarrow A_{pr}A_{rq}$.

Claim 1

If $\underline{A_{pq}}$ generates x, then x can bring $\underline{P}$ from p with empty stack to q with empty stack.

证明: (Prove by Induction)

Induction step: Suppose that $A_{pq} \stackrel{*}{\Rightarrow} x$ with $k+1$ steps.

Case 1: The first step in this derivation is $\underline{A_{pq} \Rightarrow aA_{rs}b}$.

- Consider the portion y of x that A_{rs} *generates*, so $x = ayb$.
- Because $A_{rs} \stackrel{*}{\Rightarrow} y$ with k steps, P can go from r on empty stack to s on empty stack
- Process:
  - From p to r: Read a, and push an element, e.g., u
  - From r to s: Read y
  - From s to q: Read b, and pop u

Claim 1

If $\underline{A_{pq}}$ generates x, then x can bring $\underline{P}$ from p with empty stack to q with empty stack.

证明: (Prove by Induction)

Induction step: Suppose that $A_{pq} \overset{*}{\Rightarrow} x$ with $k+1$ steps.

Case 2: The first step in this derivation is $\underline{A_{pq} \Rightarrow A_{pr}A_{rq}}$.

- Consider the portions y *and* z of x that A_{pr} *and* A_{rq} respectively generate, so $x = yz$
- Because $A_{pr} \overset{*}{\Rightarrow} y$, $A_{rq} \overset{*}{\Rightarrow} z$ in k steps, y can bring P from p to r, and z can bring P from r to q, with empty stacks at the beginning and end.
- Process: similar to case 1.

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

思路: Induction on the number of steps in the computation of P that goes from p to q with *empty stacks* on input x.

Basis: The computation has 0 steps

- If a computation has 0 steps, it starts and ends at the same state, say, p
- In 0 steps, P cannot read any characters, so $x = \varepsilon$.
- By construction, G has the rule $\underline{A_{pp} \to \varepsilon}$
- So $A_{pp} \stackrel{*}{\Rightarrow} \varepsilon$

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

思路: Induction on the number of steps in the computation of P that goes from p to q with *empty stacks* on input x.

Basis: The computation has 0 steps

- If a computation has 0 steps, it starts and ends at the same state, say, p
- In 0 steps, P cannot read any characters, so $x = \varepsilon$.
- By construction, G has the rule $\underline{A_{pp} \to \varepsilon}$
- So $A_{pp} \stackrel{*}{\Rightarrow} \varepsilon$

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

思路: Induction on the number of steps in the computation of P that goes from p to q with *empty stacks* on input x.

Induction step: Assume true for computations of length at most k, where $k \geq 0$, and prove true for computations of length $k + 1$

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Either the stack is empty *only* at the beginning and end of this computation, or it becomes empty *elsewhere*, too.

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Case 1: the stack is empty *only* at the beginning and end of computation

- the symbol that is *pushed* at the *first move* must be the same as the symbol that is *popped* at the *last move*. Call this symbol u.

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Case 1: the stack is empty *only* at the beginning and end of computation

- the symbol that is *pushed* at the *first move* must be the same as the symbol that is *popped* at the *last move*. Call this symbol u.
- Let a be the input read in the *first move*, b be the input read in the *last move*, r be the state after the *first move*, and s be the state before the *last move*. So G has $\underline{A_{pq} \Rightarrow aA_{rs}b}$

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Case 1: the stack is empty *only* at the beginning and end of computation

- Let a be the input read in the *first move*, b be the input read in the *last move*, r be the state after the *first move*, and s be the state before the *last move*. So G has $\underline{A_{pq} \Rightarrow aA_{rs}b}$
- Let y be the portion of x without a and b, so $x = ayb$.
  - the computation on y has $(k+1)-2 = k-1$ steps
  - So $A_{rs} \overset{*}{\Rightarrow} y$

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Case 1: the stack is empty *only* at the beginning and end of computation

- Let y be the portion of x without a and b, so $x = ayb$.
  - the computation on y has $(k+1)-2 = k-1$ steps
  - So $A_{rs} \stackrel{*}{\Rightarrow} y$
- So $A_{pq} \stackrel{*}{\Rightarrow} x$

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, $\underline{A_{pq}}$ generates x.

证明: (Prove by Induction)

Induction step: Suppose that P has a computation wherein x brings p to q with empty stacks in $k+1$ steps.

Case 2: the stack is empty *not only* at the beginning and the end

- let r be a state where the stack becomes empty other than at the beginning or end of the computation on x
- Process
  - $\leq k$ steps from p to r: assume y as input, so $A_{pr} \stackrel{*}{\Rightarrow} y$
  - $\leq k$ steps from r to q: assume z as input, so $A_{rq} \stackrel{*}{\Rightarrow} z$
  - $x = yz$, so by $\underline{A_{pq} \Rightarrow A_{pr}A_{rq}}$, $A_{pq} \stackrel{*}{\Rightarrow} x$

引理 2

If a *pushdown automaton* recognizes some language, then it is *context free*

Claim 1

If A_{pq} generates x, then x can bring $\underline{P}$ from p with empty stack to q with empty stack.

Claim 2

If x can bring $\underline{P}$ from p with empty stack to q with empty stack, A_{pq} generates x.

All proved

定理

A language is *context free* if and only if some *pushdown automaton* recognizes it

引理 1

If a language is *context free*, then some *pushdown automaton* recognizes it

引理 2

If a *pushdown automaton* recognizes some language, then it is *context free*

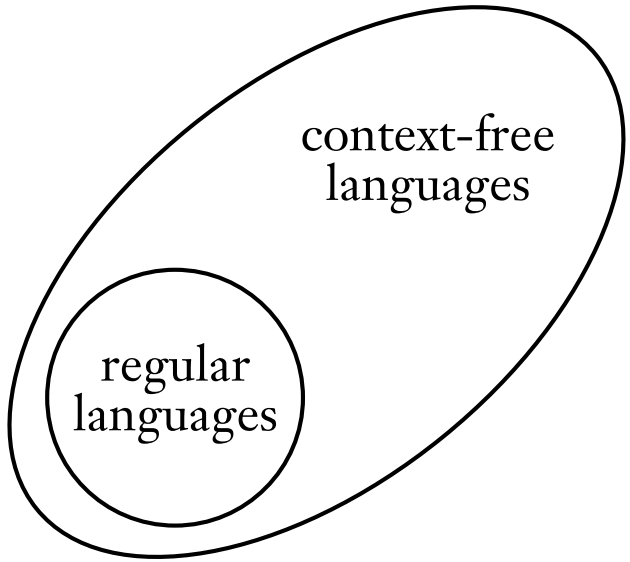
All proved

推论

Every *regular language* is *context free*

证明思路

Every *finite automaton* is automatically a *pushdown automaton* that simply *ignores its stack*



2.2 Context-Free Languages

Outline

回顾: 问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Can a PDA recognize *all* languages?
答: Not either.

问: How to name the class of languages?
答: *Non-Context-Free* languages.

问: How to prove that a language is *Non-Context-Free*
答: 准备: *pumping lemma*: A theorem, stating that all regular languages have a *special property*.

回顾: 问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Can a PDA recognize *all* languages?
答: Not either.

问: How to name the class of languages?
答: *Non-Context-Free* languages.

问: How to prove that a language is *Non-Context-Free*
答: 准备: *pumping lemma*: A theorem, stating that all regular languages have a *special property*.

回顾: 问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Can a PDA recognize *all* languages?
答: Not either.

问: How to name the class of languages?
答: *Non-Context-Free* languages.

问: How to prove that a language is *Non-Context-Free*
答: 准备: *pumping lemma*: A theorem, stating that all regular languages have a *special property*.

回顾: 问: Can a finite automaton recognize *all* languages?
答: Of course not.

问: Can a PDA recognize *all* languages?
答: Not either.

问: How to name the class of languages?
答: *Non-Context-Free* languages.

问: How to prove that a language is *Non-Context-Free*
答: 准备: *pumping lemma*: A theorem, stating that all regular languages have a *special property*.

回顾: 定理: Pumping Lemma for regular languages

If A is a *regular language*, then there is a *number* p (the *pumping length*), where if s is any string in A of *length at least* p, then s may be *divided* into *three pieces*, $s = xyz$, satisfying the following conditions:
(1) for each $i \geq 0$, $xy^iz \in A$, (2) $|y| > 0$, (3) $|xy| \leq p$

定理: Pumping Lemma for context-free languages

If A is a *context-free language*, then there is a *number* p (the *pumping length*) where, if s is any string in A of *length at least* p, then s may be *divided* into *five pieces* $s = uvxyz$ satisfying the conditions:

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$
3. $|vxy| \leq p$

回顾: 定理: Pumping Lemma for regular languages

If A is a *regular language*, then there is a *number* p (the *pumping length*), where if s is any string in A of *length at least* p, then s may be *divided* into *three pieces*, $s = xyz$, satisfying the following conditions:
(1) for each $i \geq 0$, $xy^iz \in A$, (2) $|y| > 0$, (3) $|xy| \leq p$

定理: Pumping Lemma for context-free languages

If A is a *context-free language*, then there is a *number* p (the *pumping length*) where, if s is any string in A of *length at least* p, then s may be *divided* into *five pieces* $s = uvxyz$ satisfying the conditions:

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$
3. $|vxy| \leq p$

定理: Pumping Lemma for context-free languages

If A is a *context-free language*, then there is a *number* p where, if s is any string in A of *length at least* p, then s may be *divided* into *five pieces* $s = uvxyz$ satisfying the conditions:

(1) for each $i \geq 0$, $uv^ixy^iz \in A$, (2) $|vy| > 0$, (3) $|vxy| \leq p$

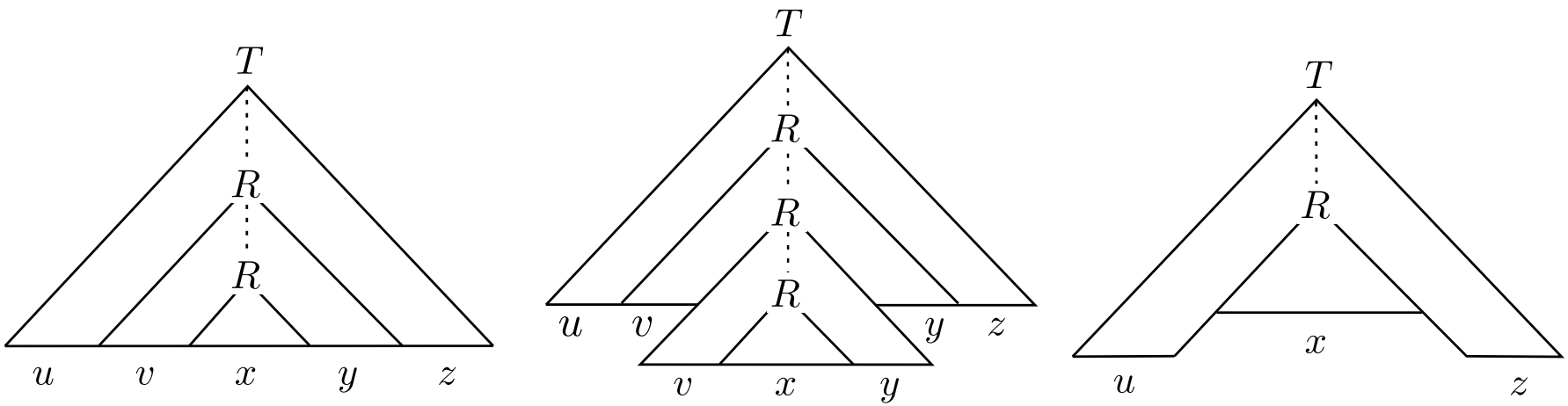
证明思路

- Let A be a CFL and let G be a CFG that generates A
- Prove that any sufficiently long string s in A can be *pumped* and *remain in* A
  - Because s is in A, it is derivable from G and so has a parse tree
  - Some variable symbol R must *repeat*

定理: Pumping Lemma for context-free languages

If A is a *context-free language*, then there is a *number* p where, if s is any string in A of *length at least* p, then s may be *divided* into *five pieces* $s = uvxyz$ satisfying the conditions:

(1) for each $i \geq 0$, $uv^ixy^iz \in A$, (2) $|vy| > 0$, (3) $|vxy| \leq p$



证明思路

Therefore, we may cut s into five pieces $uvxyz$

待证问题

Prove that *a language* B is *not context free*, by using the pumping lemma

证明思路

1. *Assume* that B is *context free* in order to obtain a *contradiction*
2. Find a string s in B that has length p or greater but that *cannot be pumped*
3. Demonstrate that s cannot be pumped by *considering all ways* of dividing s into x, y, and z

例: 求证

Prove that the language $B = \{a^n b^n c^n \mid n \geq 0\}$ is *not context free*

证明: (Prove by contradiction using Pumping Lemma)

- Let p be the pumping length, select $s = a^p b^p c^p$
- We show that no matter how we divide s into $uvxyz$, one of the three conditions of the lemma is violated
- Either v or y is nonempty, according to condition 2

例: 求证

Prove that the language $B = \{a^n b^n c^n \mid n \geq 0\}$ is *not context free*

证明: (Prove by contradiction using Pumping Lemma)

- Let p be the pumping length, select $s = a^p b^p c^p$
- We show that no matter how we divide s into $uvxyz$, one of the three conditions of the lemma is violated
- Either v or y is nonempty, according to condition 2

Case 1: both v and y contain only one type of alphabet symbol

- v does not contain both a' s and b' s or both b' s and c' s, and the same holds for y
- contradiction for condition 1: the string uv^2xy^2z cannot contain equal numbers of a' s, b' s, and c' s.

例: 求证

Prove that the language $B = \{a^n b^n c^n \mid n \geq 0\}$ is *not context free*

证明: (Prove by contradiction using Pumping Lemma)

- Let p be the pumping length, select $s = a^p b^p c^p$
- We show that no matter how we divide s into $uvxyz$, one of the three conditions of the lemma is violated
- Either v or y is nonempty, according to condition 2

Case 2: Either v or y contains more than one type of symbol

- contradiction for condition 1: uv^2xy^2z may contain equal numbers of the three alphabet symbols, but not in the correct order

例: 求证

Prove that the language $B = \{a^n b^n c^n \mid n \geq 0\}$ is *not context free*

证明: (Prove by contradiction using Pumping Lemma)

- Let p be the pumping length, select $s = a^p b^p c^p$
- We show that no matter how we divide s into $uvxyz$, one of the three conditions of the lemma is violated
- Either v or y is nonempty, according to condition 2

Since one of these cases must occur, B is not context free

2.2 Context-Free Languages

Outline

问: Are PDAs deterministic or non-deterministic?
答: Non-deterministic

问: Are DFAs and NFAs equivalent in language recognition power?
答: Yes

问: Are there deterministic Pushdown Automata (*DPDAs*)?
答: Yes

问: Are DPDAs and PDAs equivalent in language recognition power?
答: No, PDAs are more powerful

问: What is the *language* of a *DPDA*?
答: Deterministic Context-Free Language, *DCFL*.

问: Why do we introduce DCFLs?
答: DCFL is relevant to *practical applications*, such as the design of *parsers* in compilers for programming languages

- Since the parsing problem is generally *easier for DCFLs* than for CFLs

问: Are PDAs deterministic or non-deterministic?
答: Non-deterministic

问: Are DFAs and NFAs equivalent in language recognition power?
答: Yes

问: Are there deterministic Pushdown Automata (*DPDAs*)?
答: Yes

问: Are DPDAs and PDAs equivalent in language recognition power?
答: No, PDAs are more powerful

问: What is the *language* of a *DPDA*?
答: Deterministic Context-Free Language, *DCFL*.

问: Why do we introduce DCFLs?
答: DCFL is relevant to *practical applications*, such as the design of *parsers* in compilers for programming languages

- Since the parsing problem is generally *easier for DCFLs* than for CFLs

问: Are PDAs deterministic or non-deterministic?
答: Non-deterministic

问: Are DFAs and NFAs equivalent in language recognition power?
答: Yes

问: Are there deterministic Pushdown Automata (*DPDAs*)?
答: Yes

问: Are DPDAs and PDAs equivalent in language recognition power?
答: No, PDAs are more powerful

问: What is the *language* of a *DPDA*?
答: Deterministic Context-Free Language, *DCFL*.

问: Why do we introduce DCFLs?
答: DCFL is relevant to *practical applications*, such as the design of *parsers* in compilers for programming languages

- Since the parsing problem is generally *easier for DCFLs* than for CFLs

问: Are PDAs deterministic or non-deterministic?
答: Non-deterministic

问: Are DFAs and NFAs equivalent in language recognition power?
答: Yes

问: Are there deterministic Pushdown Automata (*DPDAs*)?
答: Yes

问: Are DPDAs and PDAs equivalent in language recognition power?
答: No, PDAs are more powerful

问: What is the *language* of a *DPDA*?
答: Deterministic Context-Free Language, *DCFL*.

问: Why do we introduce DCFLs?
答: DCFL is relevant to *practical applications*, such as the design of *parsers* in compilers for programming languages

- Since the parsing problem is generally *easier for DCFLs* than for CFLs

问: Are PDAs deterministic or non-deterministic?
答: Non-deterministic

问: Are DFAs and NFAs equivalent in language recognition power?
答: Yes

问: Are there deterministic Pushdown Automata (*DPDAs*)?
答: Yes

问: Are DPDAs and PDAs equivalent in language recognition power?
答: No, PDAs are more powerful

问: What is the *language* of a *DPDA*?
答: Deterministic Context-Free Language, *DCFL*.

问: Why do we introduce DCFLs?
答: DCFL is relevant to *practical applications*, such as the design of *parsers* in compilers for programming languages

- Since the parsing problem is generally *easier for DCFLs* than for CFLs

问: Are PDAs deterministic or non-deterministic?
答: Non-deterministic

问: Are DFAs and NFAs equivalent in language recognition power?
答: Yes

问: Are there deterministic Pushdown Automata (*DPDAs*)?
答: Yes

问: Are DPDAs and PDAs equivalent in language recognition power?
答: No, PDAs are more powerful

问: What is the *language* of a *DPDA*?
答: Deterministic Context-Free Language, *DCFL*.

问: Why do we introduce DCFLs?
答: DCFL is relevant to *practical applications*, such as the design of *parsers* in compilers for programming languages

- Since the parsing problem is generally *easier for DCFLs* than for CFLs

定义: Deterministic Pushdown Automaton, DPDA ▸ PDA

A deterministic pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of accept states.

The transition function δ *must* satisfy the following condition:

For every $q \in Q$, $a \in \Sigma$, and $x \in \Gamma$, *exactly one* of the values

$$\delta(q, a, x), \delta(q, a, \varepsilon), \delta(q, \varepsilon, x), \delta(q, \varepsilon, \varepsilon)$$

is *not* $\emptyset$.

问: Which CFLs are DCFLs?

例: $\{0^n 1^n \mid n \geq 0\}$

问: Which CFLs are not DCFLs?

例: $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

问: Which CFLs are DCFLs?
例: $\{0^n 1^n \mid n \geq 0\}$

问: Which CFLs are not DCFLs?
例: $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Outline

总结

- 定义
  - grammar , rule , variable , terminal , start variable
  - context-free grammars (CFG) , context-free language (CFL)
  - yield , derive
  - leftmost derivation , ambiguity
  - Chomsky normal form
  - pushdown automaton (PDA)
  - Deterministic Pushdown Automata (DPDA)
- 定理
  - From CFG to Chomsky normal form
  - CFL “$\equiv$” PDA
  - Pumping lemma for CFL
- 推论
  - Regular Language $\subseteq$ CFL

作业

2.1 Recall the CFG G_4 that we gave in Example 2.4. For convenience, let's rename its variables with single letters as follows.

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T \times F \mid F \\ F &\to (E) \mid \mathtt{a} \end{aligned}$$

Give parse trees and derivations for each string.

a. a

b. a+a

c. a+a+a

d. ((a))

2.9 Give a context-free grammar that generates the language

$$A = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}.$$

Is your grammar ambiguous? Why or why not?

2.10 Give an informal description of a pushdown automaton that recognizes the language A in Exercise 2.9.

作业

2.14 Convert the following CFG into an equivalent CFG in Chomsky normal form, using the procedure given in Theorem 2.9.

$$A \to BAB \mid B \mid \varepsilon$$
$$B \to \mathtt{00} \mid \varepsilon$$

***2.33** Show that $F = \{\mathtt{a}^i\mathtt{b}^j \mid i = kj \text{ for some positive integer } k\}$ is not context free.