Formal languages and Genshin

形式语言与自动机

Chapter1.2 Three basic concepts

Beijing University of Posts and Telecommunications

北京邮电大学

刘咏彬 liuyb@bupt.edu.cn

2023.9

Contents

Concepts

- TERMS
  - **Alphabet（字母表）**: An alphabet is a finite, nonempty set Σ of **symbols（符号）**
  - **Strings（符号串）**:Strings are finite sequences of symbols from the alphabet
- EXAMPLE
  - if the alphabet Σ = {a, b}, then abab and $\mathrm{aaabbba}$ are strings on Σ
- Symbolic conventions
  - Lowercase letters $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$ for elements of Σ
  - Lowercase letters $\mathrm{u}, \mathrm{v}, \mathrm{w}, \ldots$ for string names.
  - $\mathrm{w} = \mathrm{abaaa}$

Concepts

- TERMS

① The **concatenation(连接)** of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1a_2 \cdots a_n$$

$$v = b_1b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1a_2 \cdots a_nb_1b_2 \cdots b_m$$

② The **reverse(逆)** of a string is obtained by writing the symbols in reverse order;if w is a string as shown above, then its reverse wR is

$$w^R = a_n \cdots a_2a_1.$$

Concepts

- TERMS
  - **length（长度）**， denoted by $|w|$, is the number of symbols inthe string.
  - **empty string（空串）**, a string with no symbols at all. It will be denoted by λ.
  - For all w:

$$|\lambda| = 0,$$
$$\lambda w = w\lambda = w$$

  - Obviously:
  - |uv| = |u| + |v| .

Concepts

● TERMS

Any string of consecutive symbols in some w is said to be a **substring（子串）** of w. If

$$w = vu,$$

then the substrings v and u are said to be a **prefix（前缀）** and a **suffix（后缀）** of w,respectively.

●EXAMPLE

if $w = abbab$,

then $\{\lambda, a, ab, abb, abba, abbab\}$ is the set of all prefixes of w,

while bab, ab, b are some of its suffixes.

Concepts

- TERMS
  - **Power(幂)**: w^n stands for repeating w n times.

    for all w :

    $$w^0 = \lambda,$$
  - **$\Sigma^{\square}$** denote the set of strings obtained by concatenating zero or more symbols from Σ.
  - **Σ^{+}** The set $\Sigma^{\square}$ always contains λ. To exclude the empty string， we define

    $$\Sigma^{+} = \Sigma^{\square} - \{\lambda\} .$$

    $\Sigma^{\square}$ and Σ^{+} are always infinite

Concepts

- TERMS
  - **Language**: a subset of $\Sigma^{\square}$.
  - A string in a language L will be called a **sentence** of L.
  - A language is a set of any number of sentences.

  - The set with no elements, called the **empty set** or the **null set**, is denoted by Φ.

EXAMPLE 1.9

Let $\Sigma = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, ...\}.$$

The set

$$\{a, aa, aab\}$$

is a language on Σ. Because it has a finite number of sentences, we call it a finite language. The set

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

is also a language on Σ. The strings $aabb$ and $aaaabbbb$ are in the language L, but the string abb is not in L. This language is infinite. Most interesting languages are infinite.

Language Operations

- Complement(补):

$$\overline{L} = \Sigma^* - L$$

- Reverse(逆)

$$L^R = \{w^R : w \in L\}$$

- Concatenation(连接):

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}$$

$$L^0 = \{\lambda\}$$

$$L^1 = L$$

$$L^n = L^{n-1}L$$

- Star-Closure(星闭包):

$$L^* = L^0 \cup L^1 \cup L^2 \cdots$$

- Positive-Closure(正闭包):

$$L^+ = L^1 \cup L^2 \cdots$$

EXAMPLE 1.10

If

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

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Note that n and m in the above are unrelated; the string $aabbaaabbb$ is in L^2.

The reverse of L is easily described in set notation as

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

but it is considerably harder to describe $\overline{L}$ or L^* this way. A few tries will quickly convince you of the limitation of set notation for the specification of complicated languages.

Grammars

- A grammar for the English language

$$\text{<sentence>} \rightarrow \text{<noun phrase>}\text{<predicate>}$$

$$\text{<noun phrase>} \rightarrow \text{<article>}\text{<noun>}$$

$$\text{<predicate>} \rightarrow \text{<verb>}$$

$$\text{<article>} \rightarrow \text{a} \mid \text{the}$$

$$\text{<noun>} \rightarrow \text{boy} \mid \text{dog}$$

$$\text{<verb>} \rightarrow \text{runs} \mid \text{walks}$$

$\text{<sentence>} \Rightarrow^{*}$ "a boy runs"

$\text{<sentence>} \Rightarrow^{*}$ "the dog walks"

Grammar -DEFINITION 1.1

A grammar G is defined as a quadruple

$$G = (V, T, S, P),$$

where V is a finite set of objects called **variables(变量)**,

T is a finite set of objects called **terminal symbols（终极符）**，

S $\in$ V is a special symbol called the **start variable（开始变量）**，

P is a finite set of **productions（产生式）**．

It will be assumed without further mention that the sets V and T are non-empty and disjoint.

Symbolic conventions

- Uppercase letters $A, B, C, \ldots$ for elements of V
- Lowercase letters $a, b, c, \ldots$ for elements of T.
- Lowercase letters $x, y, \ldots$ for elements of $(V \cup T)^*$

Examples of Grammars

(1) $(\{A\}, \{0, 1\}, A, \{A\to 01, A\to 0A1, A\to 1A0\})$

(2) $(\{A\}, \{0, 1\}, A, \{A\to 0, A\to 0A\})$

(3) $(\{A, B\}, \{0, 1\}, A, \{A\to 01, A\to 0A1, A\to 1A0, B\to AB,$
$B\to 0\})$

(4) $(\{A, B\}, \{0, 1\}, A, \{A\to 0, A\to 1, A\to 0A, A\to 1A\})$

(5) $(\{S, A, B, C, D\}, \{a, b, c, d, \#\}, S, \{S\to ABCD,$
$S\to abc\#, A\to aaA, AB\to aabbB, BC\to bbccC, cC\to cccC,$
$CD\to ccd\#, CD\to d\#, CD\to \#d\})$

(6) $(\{S\}, \{0, 1\}, S, \{S\to 00S, S\to 11S, S\to 00, S\to 11\})$

Grammar -production rules

- **Production**:

$$x \rightarrow y$$

where $x \in (V \cup T)^{+}$ and $y \in (V \cup T)^{*}$

补充：x中至少含1个V中的变量。

- Given a string w of the form, $w = uxv$

we may use it to **replace** x with y:

$$z = uyv$$

$$w \Rightarrow z$$

We say that w **derives** z or that z is derived from w.

- 补充：推导的逆过程称为归约(reduction)

Successive derivation

- Successive strings are derived by applying the productions of the grammar in arbitrary order.

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

- The □ indicates that an unspecified number of steps (including zero).

$$w_1 \overset{*}{\Rightarrow} w_n$$

- The + indicates at least one step derivation

$$w_1 \overset{+}{\Rightarrow} w_n$$

- We can also write n to indicate n-step derivation

$$w_1 \overset{n}{\Rightarrow} w_n$$

Sentence derivation

- Let G = (V, T, S, P) be a grammar. Then the set

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

  is the language generated by G.

- If $\mathrm{w} \in \mathrm{L(G)}$, then the sequence

$$\mathrm{S} \Rightarrow \mathrm{w}_1 \Rightarrow \mathrm{w}_2 \Rightarrow \cdots \Rightarrow \mathrm{w}_n \Rightarrow \mathrm{w}$$

  is a **derivation(推导)** of the sentence w.

  The strings S, w_1, w_2, ..., w_n, which contain variables as well as terminals, are called **sentential forms(句型)** of the derivation.

EXAMPLE 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \rightarrow aSb,$$
$$S \rightarrow \lambda.$$

Then

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb,$$

so we can write

$$S \overset{*}{\Rightarrow} aabb.$$

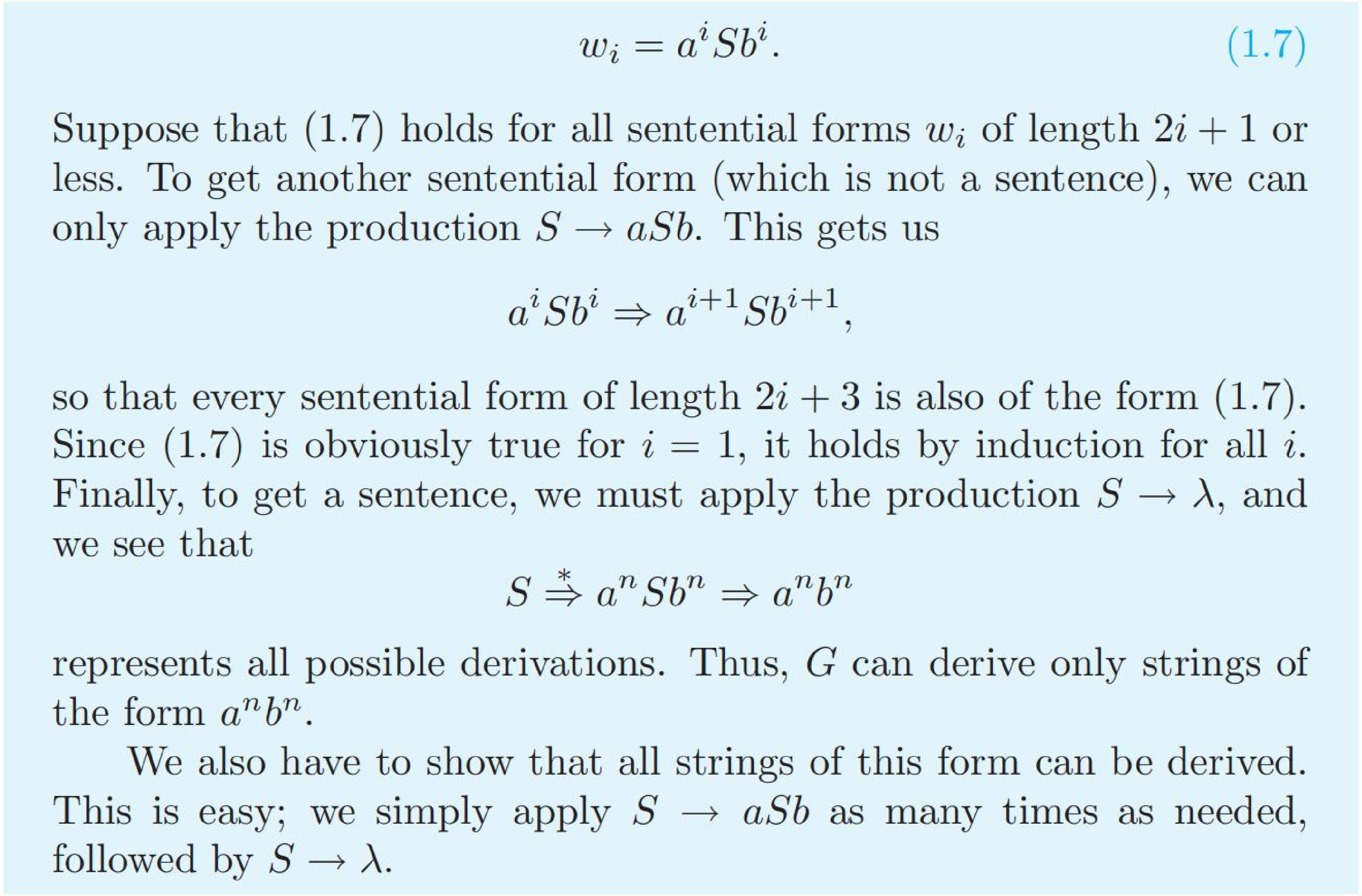
The string $aabb$ is a sentence in the language generated by G, while $aaSbb$ is a sentential form.

A grammar G completely defines $L(G)$, but it may not be easy to get a very explicit description of the language from the grammar. Here, however, the answer is fairly clear. It is not hard to conjecture that

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

Example 1.11‘s proof

归纳法证明（见书1.1.4）



$$w_i = a^i S b^i. \tag{1.7}$$

Suppose that (1.7) holds for all sentential forms w_i of length $2i + 1$ or less. To get another sentential form (which is not a sentence), we can only apply the production $S \to aSb$. This gets us

$$a^i S b^i \Rightarrow a^{i+1} S b^{i+1},$$

so that every sentential form of length $2i + 3$ is also of the form (1.7). Since (1.7) is obviously true for $i = 1$, it holds by induction for all i. Finally, to get a sentence, we must apply the production $S \to \lambda$, and we see that

$$S \overset{*}{\Rightarrow} a^n S b^n \Rightarrow a^n b^n$$

represents all possible derivations. Thus, G can derive only strings of the form $a^n b^n$.

We also have to show that all strings of this form can be derived. This is easy; we simply apply $S \to aSb$ as many times as needed, followed by $S \to \lambda$.

EXAMPLE 1.12

Find a grammar that generates

$$L = \left\{ a^n b^{n+1} : n \geq 0 \right\}.$$

The idea behind the previous example can be extended to this case. All we need to do is generate an extra b. This can be done with a production $S \rightarrow Ab$, with other productions chosen so that A can derive the language in the previous example. Reasoning in this fashion, we get the grammar $G = (\{S, A\}, \{a, b\}, S, P)$, with productions

$$\begin{aligned} S &\rightarrow Ab, \\ A &\rightarrow aAb, \\ A &\rightarrow \lambda. \end{aligned}$$

Derive a few specific sentences to convince yourself that this works.

EXAMPLE 1.13

Take $\Sigma = \{a, b\}$, and let $n_a(w)$ and $n_b(w)$ denote the number of a's and b's in the string w, respectively. Then the grammar G with productions

$$
\begin{aligned}
S &\rightarrow SS,\\
S &\rightarrow \lambda,\\
S &\rightarrow aSb,\\
S &\rightarrow bSa
\end{aligned}
$$

generates the language

$$L = \{w : n_a(w) = n_b(w)\}.$$

Analysis of Example 1.13

First, it is clear that every sentential form of G has an equal number of a's and b's, since the only productions that generate an a, namely $S \rightarrow aSb$ and $S \rightarrow bSa$, simultaneously generate a b. Therefore, every element of $L(G)$ is in L. It is a little harder to see that every string in L can be derived with G.

Let us begin by looking at the problem in outline, considering the various forms $w \in L$ can have. Suppose w starts with a and ends with b. Then it has the form

$$w = aw_1b,$$

where w_1 is also in L. We can think of this case as being derived starting with

$$S \Rightarrow aSb$$

if S does indeed derive any string in L. A similar argument can be made if w starts with b and ends with a. But this does not take care of all cases, since a string in L can begin and end with the same symbol.

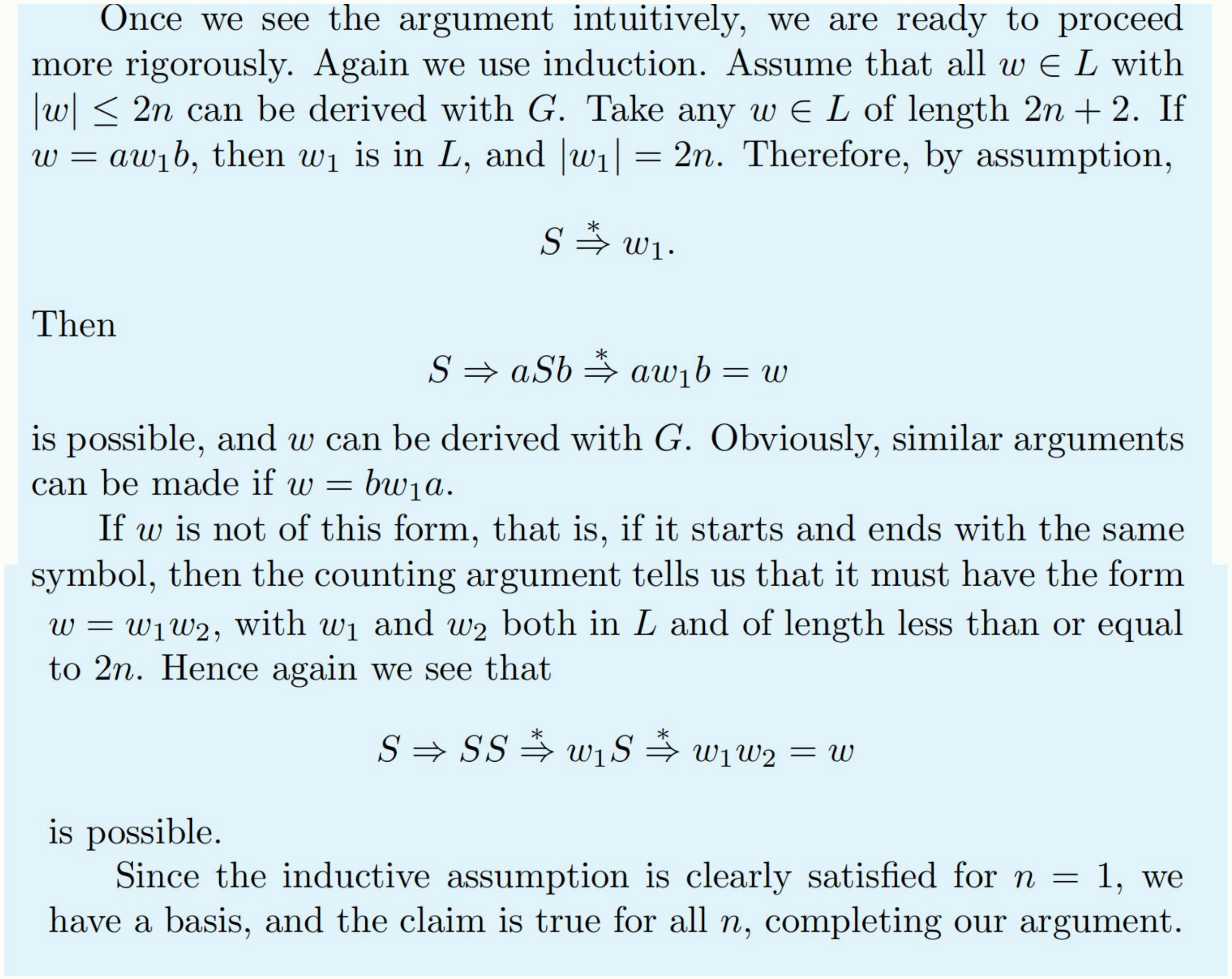
Analysis of Example 1.13

If we write down a string of this type, say $aabbba$, we see that it can be considered as the concatenation of two shorter strings $aabb$ and ba, both of which are in L. Is this true in general? To show that this is indeed so, we can use the following argument: Suppose that, starting at the left end of the string, we count $+1$ for an a and -1 for a b. If a string w starts and ends with a, then the count will be $+1$ after the leftmost symbol and -1 immediately before the rightmost one. Therefore, the count has to go through zero somewhere in the middle of the string, indicating that such a string must have the form

$$w = w_1 w_2,$$

where both w_1 and w_2 are in L. This case can be taken care of by the production $S \rightarrow SS$.

Proof of Example 1.13



Once we see the argument intuitively, we are ready to proceed more rigorously. Again we use induction. Assume that all $w \in L$ with $|w| \leq 2n$ can be derived with G. Take any $w \in L$ of length $2n + 2$. If $w = aw_1b$, then w_1 is in L, and $|w_1| = 2n$. Therefore, by assumption,

$$S \overset{*}{\Rightarrow} w_1.$$

Then

$$S \Rightarrow aSb \overset{*}{\Rightarrow} aw_1b = w$$

is possible, and w can be derived with G. Obviously, similar arguments can be made if $w = bw_1a$.

If w is not of this form, that is, if it starts and ends with the same symbol, then the counting argument tells us that it must have the form $w = w_1w_2$, with w_1 and w_2 both in L and of length less than or equal to $2n$. Hence again we see that

$$S \Rightarrow SS \overset{*}{\Rightarrow} w_1S \overset{*}{\Rightarrow} w_1w_2 = w$$

is possible.

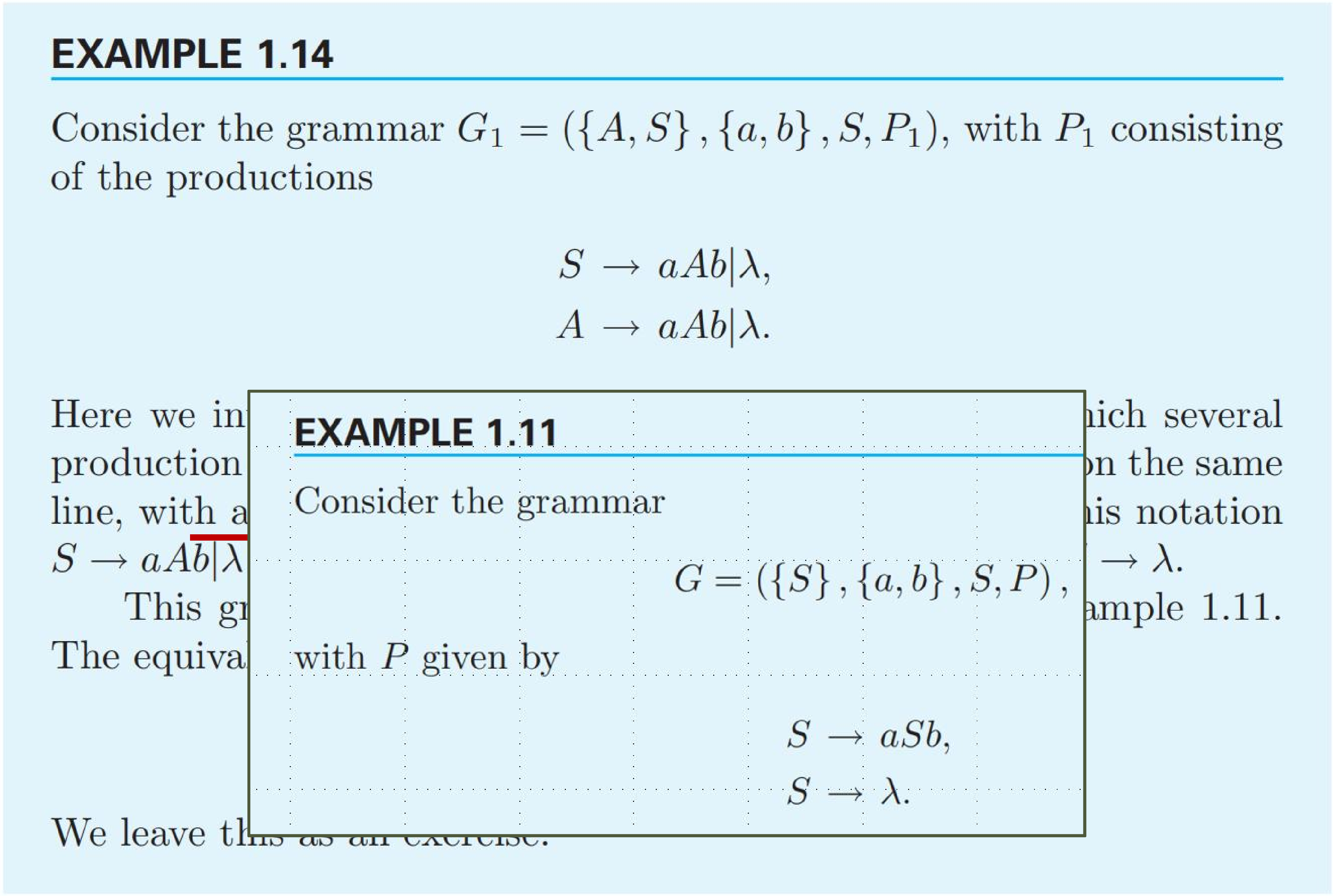
Since the inductive assumption is clearly satisfied for $n = 1$, we have a basis, and the claim is true for all n, completing our argument.

Grammar equivalence

- Normally, a given language has many grammars that generate it. Eventhough these grammars are different, they are **equivalent** in some sense.
- Two grammars G_1 and G_2 are **equivalent(等价的)** if they generate the same language, that is, if

$$L(G_1) = L(G_2)$$

Grammar equivalence



EXAMPLE 1.14

Consider the grammar $G_1 = (\{A, S\}, \{a, b\}, S, P_1)$, with P_1 consisting of the productions

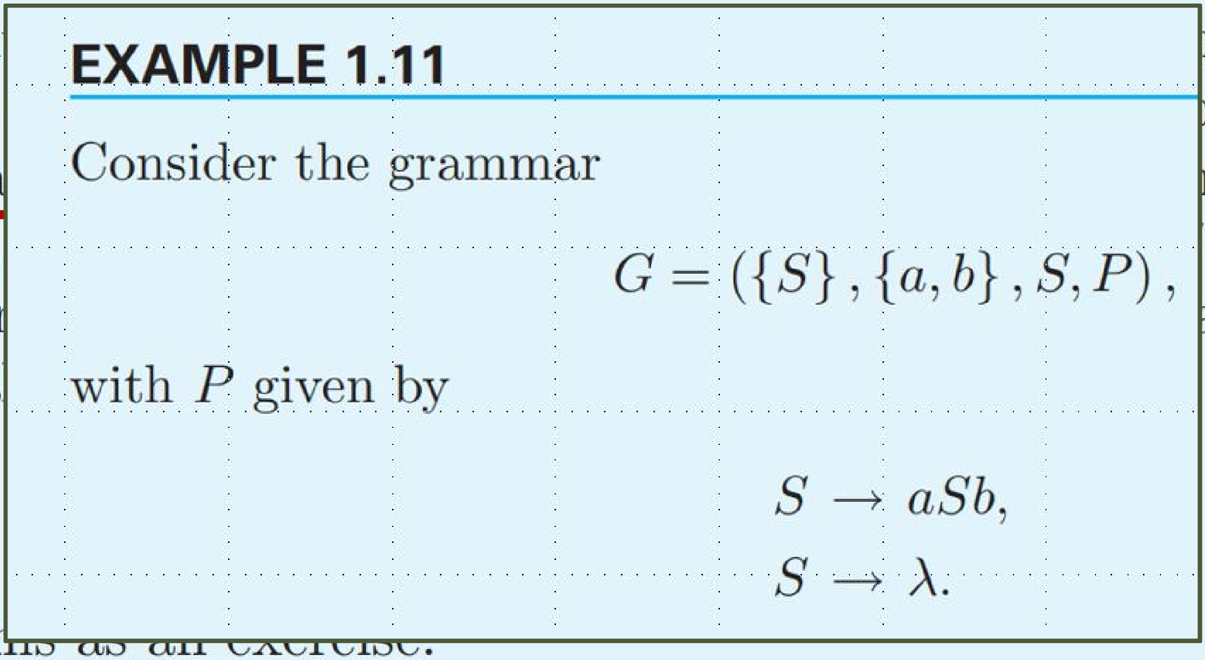
$$S \rightarrow aAb|\lambda,$$
$$A \rightarrow aAb|\lambda.$$

Here we in ich several production n the same line, with a is notation $S \rightarrow aAb|\lambda$ $\rightarrow \lambda$.

This gr ample 1.11.

The equiva

We leave th



EXAMPLE 1.11

Consider the grammar

$$G = (\{S\}, \{a, b\}, S, P),$$

with P given by

$$S \rightarrow aSb,$$
$$S \rightarrow \lambda.$$

Automata

An automaton is an abstract model of a digital computer.

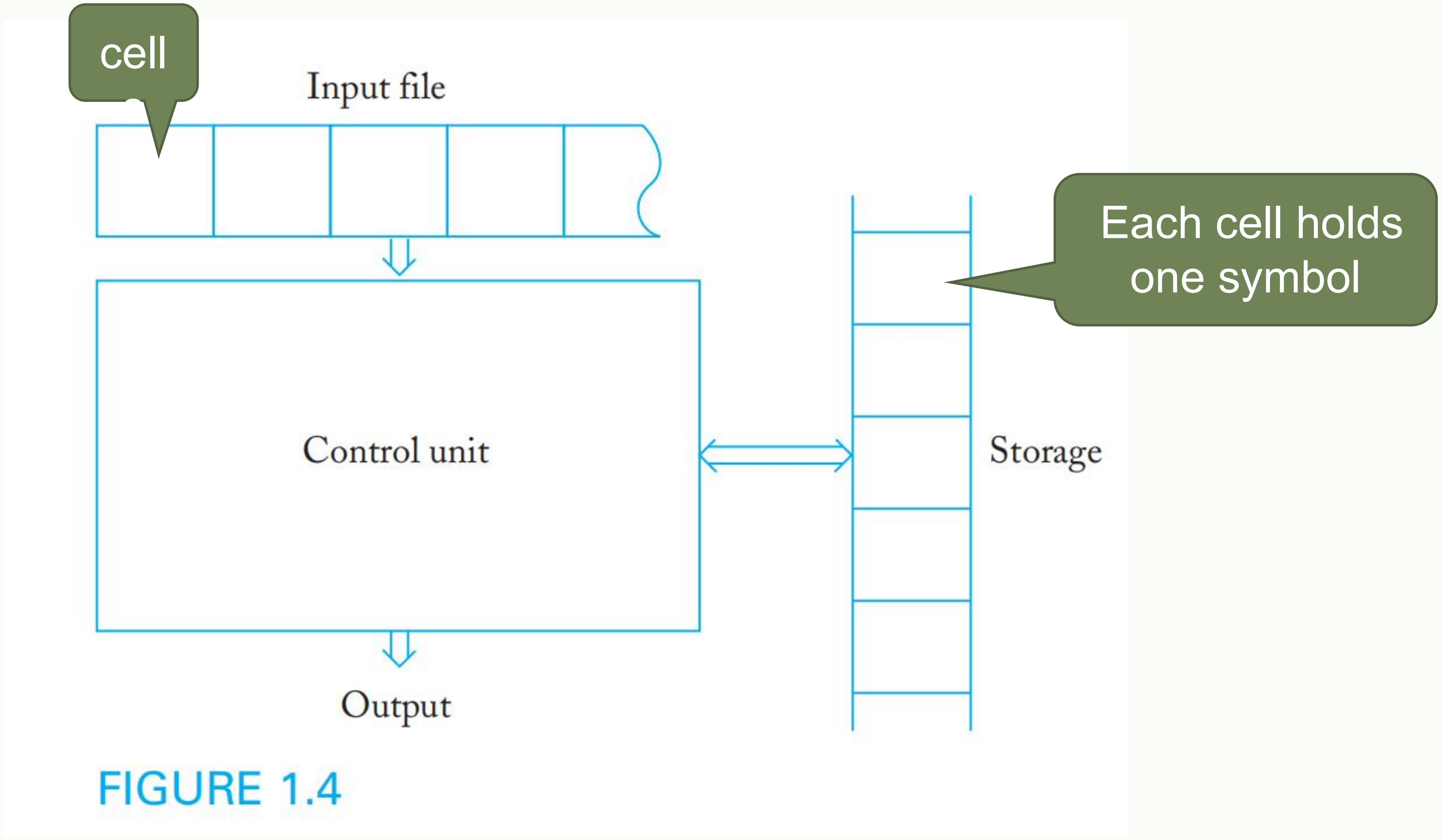


FIGURE 1.4

How does automaton work

●An automaton is assumed to operate in a discrete timeframe.

●The internal **state of the control unit(状态控制单元)** at the next time step is determined by **transition function（转移函数）**.

●This **transition function** gives the next state in terms of the **current state（当前状态）**, the **current input symbol（当前输入符号）**, and the information currently in the temporary **storage（存储区）**.

●Output may be produced or the information in the temporary storage changed.

●The term **configuration（格局）** will be used to refer to a particular state of the control unit, input file, and temporary storage.

●The transition of the automaton from one configuration to the next will be called a **move（移动）**.

Deterministic / Non-deterministic

- TERMS
  - A **deterministic automaton** is one in which each move is **uniquely determined** by the current configuration.
  - In a **nondeterministic automaton**, can have multiple possible states for a ginven input symbol.
- The relation between deterministic and nondeterministic automata of various types will play a significant role in our study.

Accepter / Transducer

- TERMS
  - An automaton whose output response is limited to a simple “yes” or “no” is called an **accepter（接受器）**.
  - An automaton, capable of producing strings of symbols as output, is called a **transducer（转换器）**.

Automaton accepts C indentifiers

EXAMPLE 1.16

Figure 1.6 is an automaton that accepts all legal C identifiers. Some interpretation is necessary. We assume that initially the automaton is in State 1; we indicate this by drawing an arrow (not originating in any vertex) to this state. As always, the string to be examined is read left to right, one character at each step. When the first symbol is a letter or an underscore, the automaton goes into State 2, after which the rest of the string is immaterial. State 2 therefore represents the "yes" state of the accepter. Conversely, if the first symbol is a digit, the automaton will go into State 3, the "no" state, and remain there. In our solution, we assume that no input other than letters, digits, or underscores is possible.

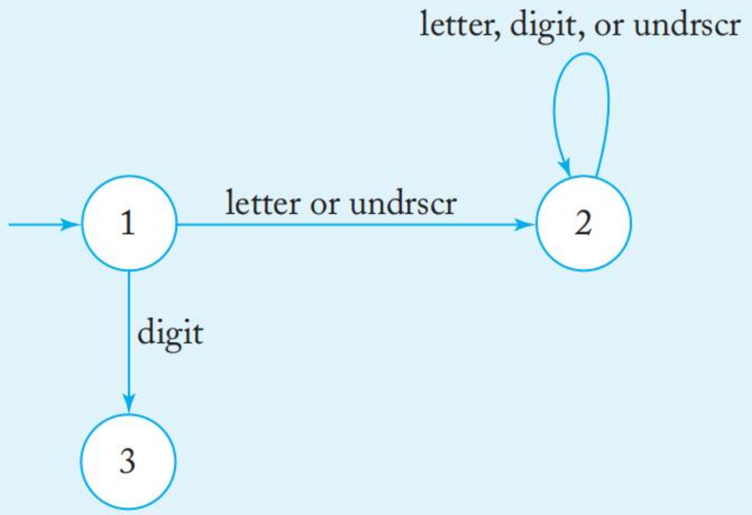


FIGURE 1.6