オートマトンと計算理論

第2部 文脈自由文法とプッシュダウンオートマトン

火曜5・6限目　必修科目

尾張　正樹

居室：J2415 (情報2号館4階)

masakiowari@inf.shizuoka.ac.jp

講義資料：¥¥fs.inf.in.shizuoka.ac.jp¥share¥class¥2018オートマトン

第２部
文脈自由文法と
プッシュダウンオートマトン

第三章：文脈自由文法

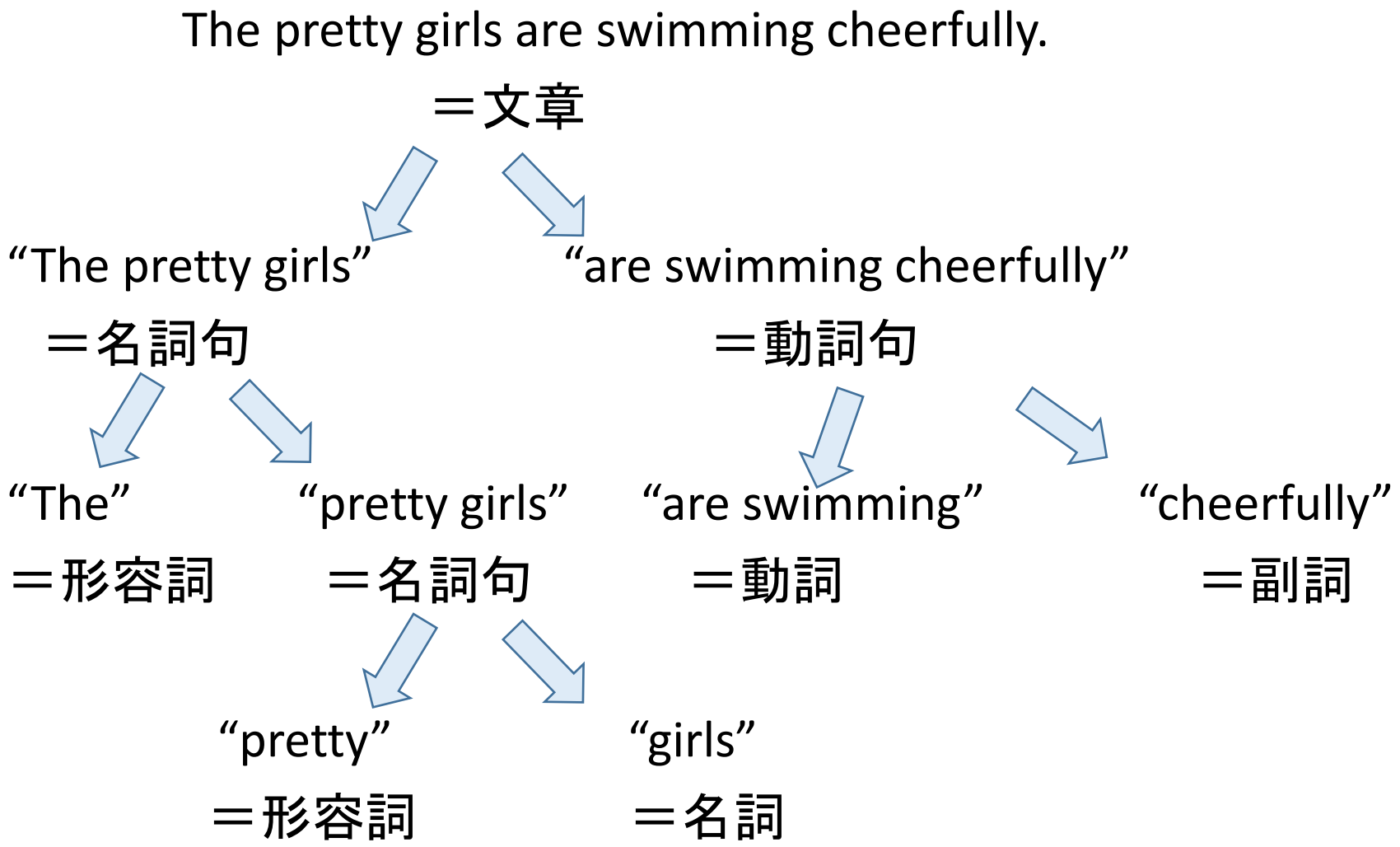
正規表現より表現力の強い『文法』

0. 生成文法
1. 文脈自由文法の重要性
2. 文脈自由文法
3. 文脈自由文法と正規言語
4. 文脈自由文法の標準形
5. 文脈自由文法で生成できない言語

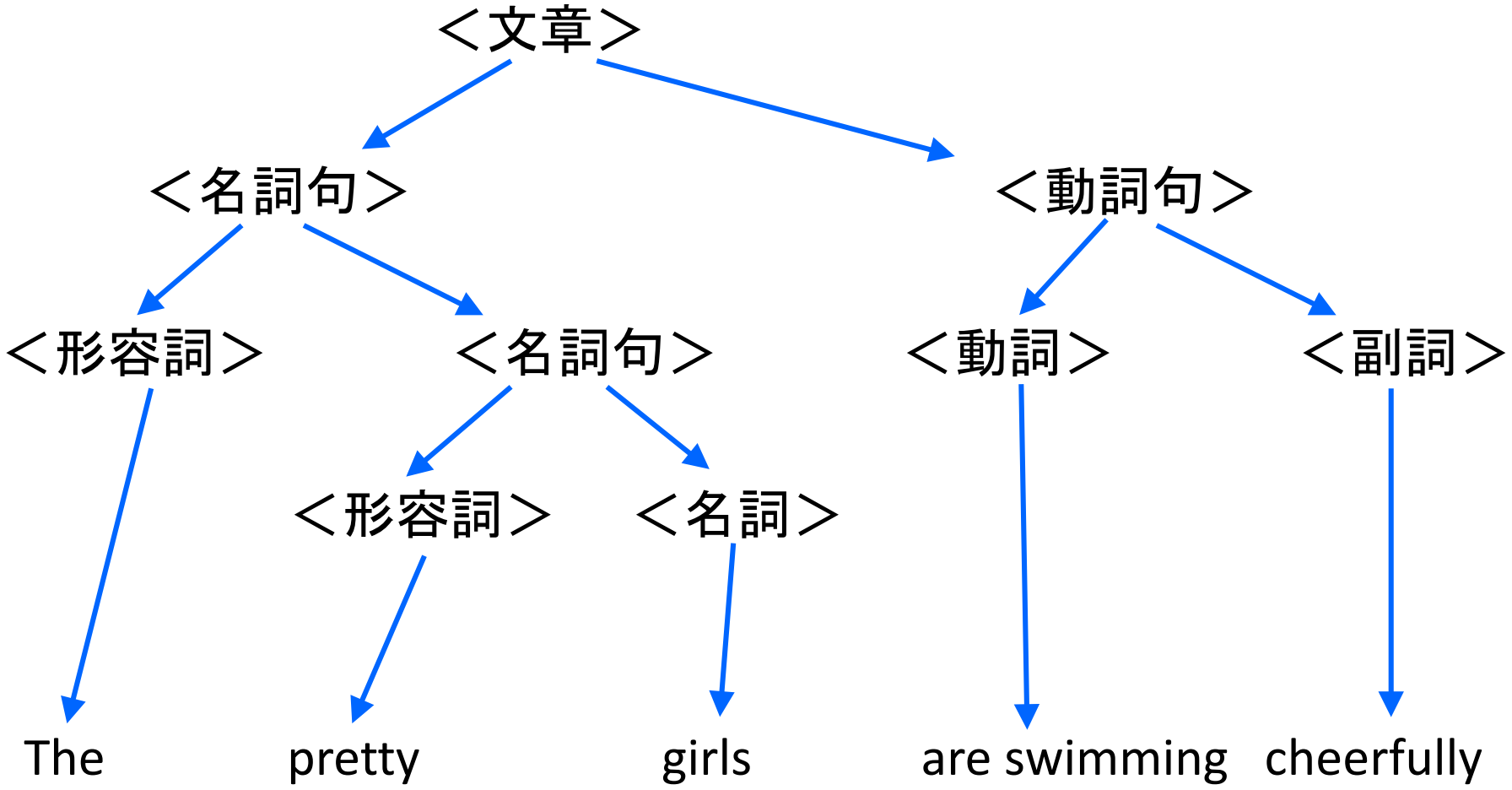
0.生成文法（1）

- 1950年代中盤に、ノーム＝チョムスキー (1928 ― 現在)により、自然言語の理論として作られた。
- 言語を生成する抽象化されたシステム
- 開始記号(例えば＜文章＞)から初めて、記号列を別の記号列で書き換る規則を適応していくことで、文字列(文章)を生成する。

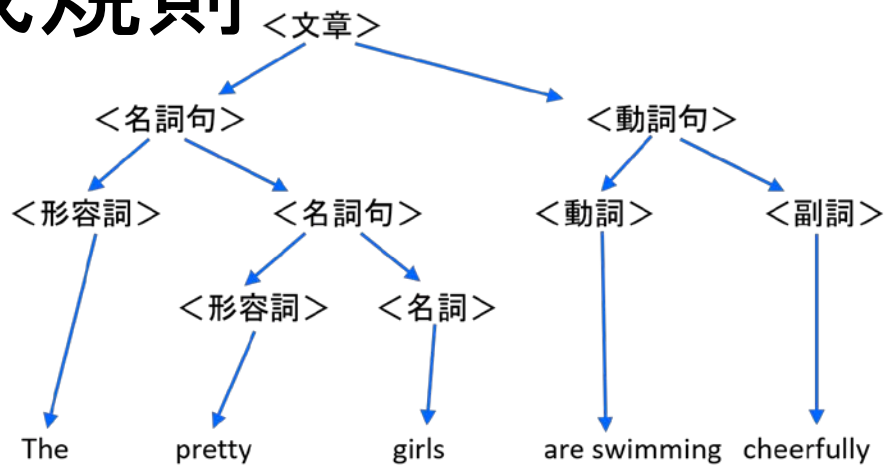
0.生成文法（2）構文解析


The pretty girls are swimming cheerfully.
＝文章
“The pretty girls”
＝名詞句
“are swimming cheerfully”
＝動詞句
“The”
＝形容詞
“pretty girls”
＝名詞句
“are swimming”
＝動詞
“cheerfully”
＝副詞
“pretty”
＝形容詞
“girls”
＝名詞

0.生成文法(3) 構文解析


＜文章＞
＜名詞句＞
＜動詞句＞
＜形容詞＞
＜名詞句＞
＜動詞＞
＜副詞＞
＜形容詞＞
＜名詞＞
The
pretty
girls
are swimming
cheerfully

0.生成文法(4)生成規則



構文解析では、
以下の規則を用いている:

＜文章＞ → ＜名詞句＞＜動詞句＞
＜名詞句＞ → ＜形容詞＞＜名詞句＞
＜名詞句＞ → ＜形容詞＞＜名詞＞
＜動詞句＞ → ＜動詞＞＜副詞＞

＜形容詞＞ →The,　＜形容詞＞ →pretty,
＜名詞＞ →girls,　＜動詞＞ →are swimming,
＜副詞＞ →cheerfully,

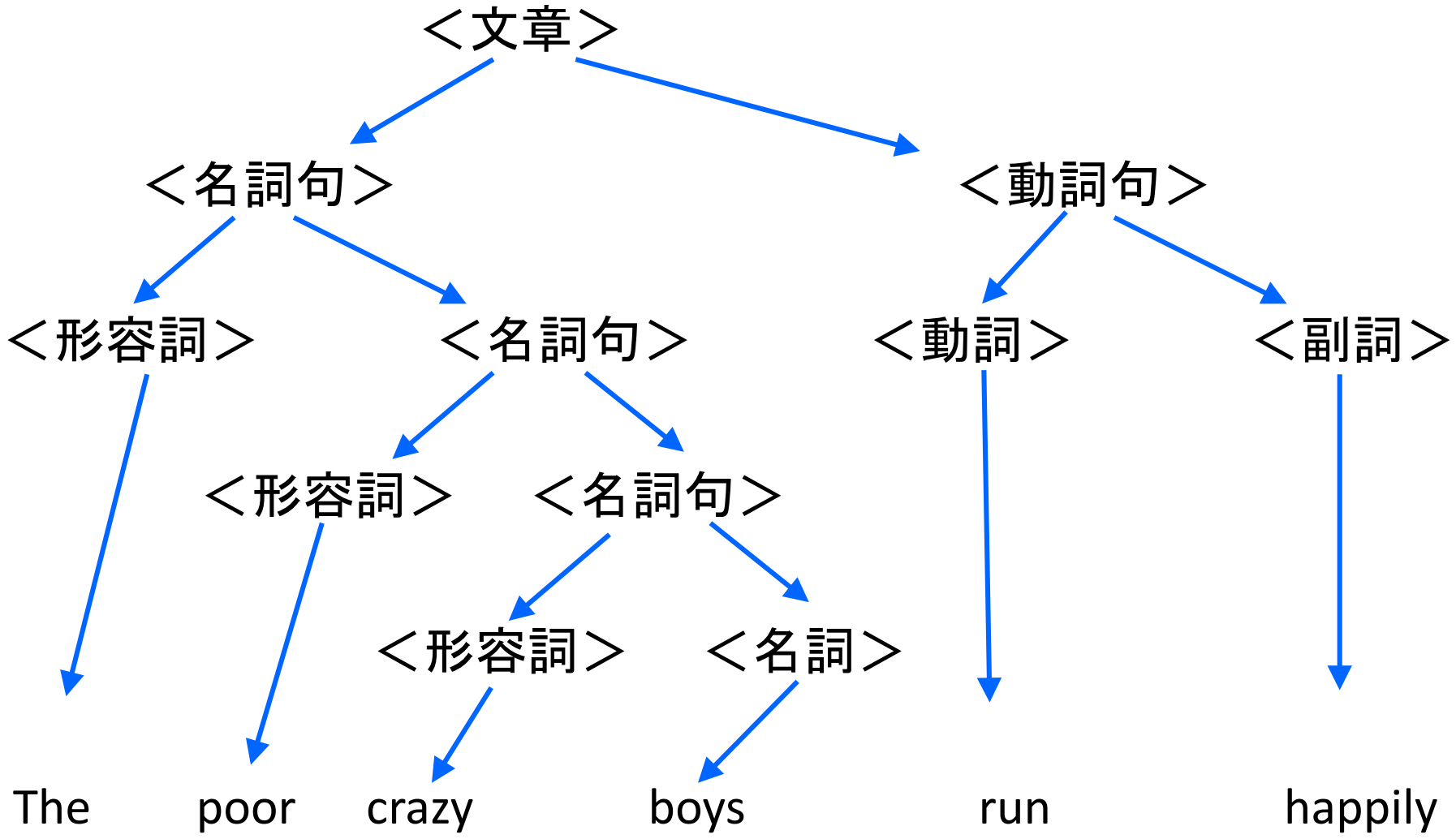
0.生成文法(5)生成規則

逆に、規則を用いて文章を作れる:

＜文章＞ → ＜名詞句＞＜動詞句＞
＜名詞句＞ → ＜形容詞＞＜名詞句＞| ＜形容詞＞＜名詞＞
＜動詞句＞ → ＜動詞＞＜副詞＞

＜形容詞＞ →The ｜ pretty ｜crazy | poor
＜名詞＞ →girls | boys | dogs |birds
＜動詞＞ →are swimming ｜fly | run | cry | sleep
＜副詞＞ →cheerfully | rapidly | silently | happily

0.生成文法(6) 構文解析


＜文章＞
＜名詞句＞
＜動詞句＞
＜形容詞＞
＜名詞句＞
＜動詞＞
＜副詞＞
＜形容詞＞
＜名詞句＞
＜形容詞＞
＜名詞＞
The
poor
crazy
boys
run
happily

1.文脈自由文法の重要性

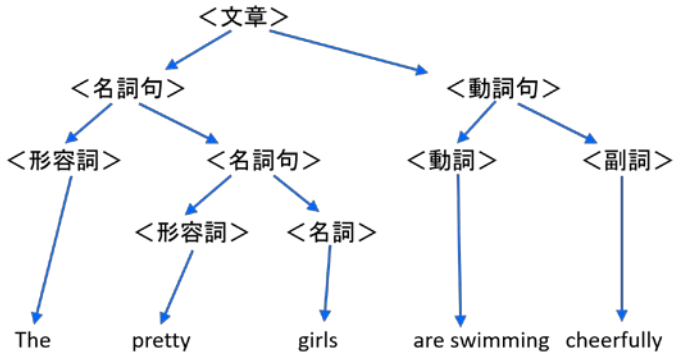
① 文脈自由文法は文法の典型的なクラス
形式言語の世界で「文法」といえば文脈自由文法

② 文脈自由文法で表現できる言語(文脈自由言語)が重要
例：多くのプログラミング言語
その他多くの人口的な言語

③ プッシュダウンオートマトンと等価(次章で説明)

④ 文脈自由文法は、我々が知っている『文法』と良く合致する。
文→名詞句・動詞句、名詞句→ 冠詞・名詞、
動詞句→ 動詞・形容詞　といった構造に非常に似ている。
「文法とはこういうものか」と理解できる。

2.文脈自由文法 (1)



＝言語に属する列を『生成する』システム

文脈自由文法(cfg) $G = (V, \Sigma, P, S)$

V: 変数 （有限集合）
Σ: 終端記号 (有限集合)　　(空列ϵも終端記号として扱う)
$S \in V$: 開始記号 、 特別な変数

P : 生成規則の有限集合
　　生成規則は、変数Aと列$\alpha \in (V \cup \Sigma)^*$に対して、
　　$A \to \alpha$ の形をしている。

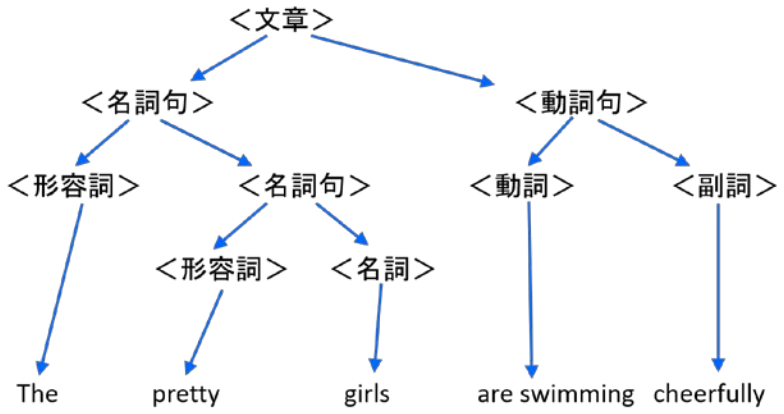
2.文脈自由文法 (1’)

さきほどの例

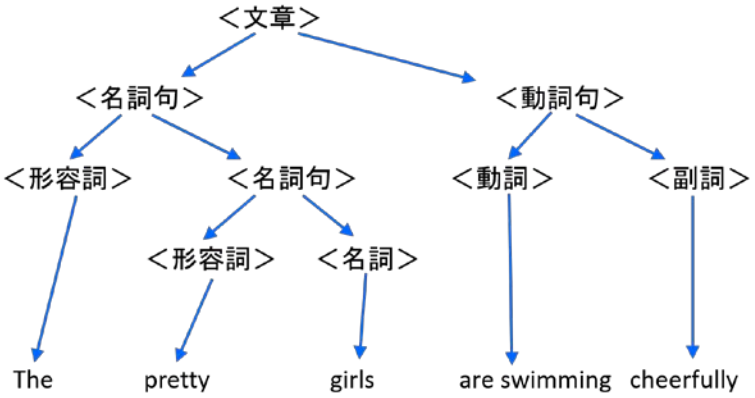
変数　V = {< 文章 >, < 名詞句 >, < 名詞 >, < 動詞句 >,
< 動詞 >, < 副詞 >, < 形容詞 >}

終端記号 Σ = {the, pretty, crazy, poor, girls, boys, birds, dogs,
are swimming, fly, cry, run, sleep,
cheerfully, rapidly, silently, happily}

開始記号 S =< 文章 >



2.文脈自由文法 (1’’)



さきほどの例

生成規則　P:

＜文章＞ → ＜名詞句＞＜動詞句＞

＜名詞句＞ → ＜形容詞＞＜名詞句＞| ＜形容詞＞＜名詞＞

＜動詞句＞ → ＜動詞＞＜副詞＞

＜形容詞＞ →The ｜ pretty ｜crazy | poor

＜名詞＞ →girls | boys | dogs |birds

＜動詞＞ →are swimming ｜fly | run | cry | sleep

＜副詞＞ →cheerfully | rapidly | silently | happily

ただし、$A \to \alpha|\beta|\gamma$ は、$A \to \alpha, A \to \beta, A \to \gamma$の略記

2.文脈自由文法 (2)

例: cfg $G_1 = (\{S\},\ \{0,1\},\ \{S \to \epsilon, S \to 0S1\},\ S)$

G_1 は導出という操作により、G_1 が表現する言語($L(G_1)$ と書く)を導く。

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000\epsilon111 = 000111$$

導出: 開始記号から初めて、生成規則をどんどん適用する。
S から初めて、$S \to \epsilon$, と $S \to 0S1$をどんどん適用する。

・変数を含まない列が得られれば、その列は$L(G_1)$に入っている。

・生成規則は、どんな順番で適用してもよい。

・変数が複数ある時は、どの変数に生成規則を適用してもよい。

2.文脈自由文法 (3)

定義： 列$u, v \in (V \cup \Sigma)^*$に対して、

$\exists \alpha, \gamma \in (V \cup \Sigma)^*$とある生成規則$A \to \beta \in P$が存在して、

$u = \alpha A \gamma$かつ$v = \alpha \beta \gamma$のとき、
vはuから1ステップで導出される（$u \Rightarrow v$）と呼ぶ。

$u_0, u_1, \cdots, u_n \in (V \cup \Sigma)^*$が、

$u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n$のとき、
u_nはu_0から導出されると呼び$u_0 \overset{*}{\Rightarrow} u_n$と書く。

$u \overset{*}{\Rightarrow} u$は常に成立する。

2.文脈自由文法 (4)

定義： 文法Gの開始記号をSとしたとき、変数を含まない列$x \in \Sigma^*$がSから導出されるとき、xは文法Gによって生成されるという。

Gが生成する言語($L(G)$)＝ Gによって生成される列全体の集合

文脈自由文法によって生成される言語を文脈自由言語と呼ぶ。

例： cfg $G_1 = (\{S\},\ \{0,1\},\ \{S \to \epsilon, S \to 0S1\},\ S)$

$000111 \in L(G_1)$

$00001111 \in L(G_1)$ などから、

$L(G_1) = \{0^n 1^n \mid n \geq 0\}$ であることが分かる。

2.文脈自由文法 (4')

定義： 文法Gの開始記号をSとしたとき、変数を含まない列$x \in \Sigma^*$がSから導出されるとき、xは文法Gによって生成されるという。

Gが生成する言語($L(G)$)＝ Gによって生成される列全体の集合

例: cfg $G_1 = (\{S\},\ \{0,1\},\ \{S \to 0S1\},\ S)$

$S \to 0S1$しか生成規則がないので、

どうやっても、Sを消すことができない。

つまり、終端記号だけの列が導出されることがない。

$L(G_1) = \emptyset$ (空集合) であることが分かる。

2.文脈自由文法 (5)

例題3.1　言語$\{0^n 1^m | \, n \neq m\}$を生成するcfgを求めよ。
（解答）

$G_2 = (\{S, S', A, B\}, \{0,1\}, P, S)$ 、

ただし生成規則Pは以下 :

$S \to AS' \mid S'B$
$S' \to \epsilon \mid 0S'1$

$A \to 0A \mid 0$
$B \to B1 \mid 1$

であることを証明する。

2.文脈自由文法 (5’)

$S \to AS' \mid S'B$
$S' \to \epsilon \mid 0S'1$

$A \to 0A \mid 0$
$B \to B1 \mid 1$

(証明)

$x = 0^i 1^{i+j}$は以下のように生成される:

$$S \Rightarrow S'B \overset{*}{\Rightarrow} 0^i S' 1^i B \Rightarrow 0^i 1^i B \overset{*}{\Rightarrow} 0^i 1^i B 1^{j-1} \Rightarrow 0^i 1^i 1^j.$$

$x = 0^{i+j} 1^i$も$S \Rightarrow AS'$から初めと同様に生成可能。

よって、 $x = 0^i 1^{i+j}$ or $x = 0^{i+j} 1^i \Longrightarrow x$が生成可能

2.文脈自由文法 (6)

次に逆(xが生成可能$\Longrightarrow x = 0^i 1^{i+j}$ or $x = 0^{i+j} 1^i$)を証明する。

最初に使う規則は、$S \to AS'$ か $S \to S'B$ しかない。

$S \to S'B$を使ったとする。すると、$S \Rightarrow S'B$である。

文脈自由文法において『$AB \overset{*}{\Rightarrow} z$である必要十分条件は、$z = xy$である列$x, y$が存在して、$A \overset{*}{\Rightarrow} x$ かつ $B \overset{*}{\Rightarrow} y$である。』

ここで、生成規則より $B \overset{*}{\Rightarrow} x \in \Sigma^*$ なら、$x = 1^j$しかありえない。

同様に、$S' \overset{*}{\Rightarrow} y \in \Sigma^*$なら、$y = 0^i 1^i$しかありえない。

結局、$S \Rightarrow S'B \overset{*}{\Rightarrow} z$ ならば、$z = 0^i 1^i 1^j$ である。

同様に、$S \Longrightarrow AS' \overset{*}{\Rightarrow} z$ ならば、$z = 0^j 0^i 1^i$ が言える。

2.文脈自由文法 (8)

練習問題　$G_2 = (\{S\}, \{0,1\}, \{S \to \epsilon, S \to 0S0, S \to 1S1\}, S)$　はどのような言語を生成するか？

（解答）

いくつか列を生成してみる。

$S \Rightarrow 0S0 \Rightarrow 01S10 \Rightarrow 010S010 \Rightarrow 0100S0010 \Rightarrow 01000010$

$L(G_2) = \{xx^R \mid x \in \{0,1\}^*\}$ である。

2.文脈自由文法 (9)

今後は、文法を与えるときは生成規則のみを与える。
開始記号はS、変数は大文字$(A, B, ..)$で、終端記号は小文字$(a, b, ...)$か数字$(0,1,2, \cdots)$で書くとする。

例題3.5　$G_2 = (\{S\}, \{0,1\}, \{S \to \epsilon, S \to 0S0, S \to 1S1\}, S)$　の補集合$L = \{0,1\}^* - L(G_2)$を生成する文法を求めよ。

（解答）$L(G_2) = \{xx^R \mid x \in \{0,1\}^*\}$

$x \in L = \{0,1\}^* - L(G_2)$ならば、

(i) xの長さが奇数、
または
(ii) $\exists\alpha, \exists\beta \in \{0,1\}^*$に対して、$x = \alpha 0 \beta 1 \alpha^R$ または $x = \alpha 1 \beta 0 \alpha^R$.

2.文脈自由文法 (10)

$x \in L$ならば、(i) xの長さが奇数、
または
(ii) $\exists\alpha, \exists\beta \in \{0,1\}^*$に対して、$x = \alpha 0 \beta 1 \alpha^R$ または $x = \alpha 1 \beta 0 \alpha^R$.

(i)の列を導く文法は、

$$S_1 \to 0 \mid 1 \mid 00S_1 \mid 01S_1 \mid 10S_1 \mid 11S_1$$

で与えられる。

(ii)の列を導く文法は、

$$S_2 \to 0S_2 0 \mid 1S_2 1 \mid 0A1 \mid 1A0,$$
$$A \to \epsilon \mid 0A \mid 1A$$

で与えられる。 実際、$S_2 \overset{*}{\Rightarrow} xS_2x^R \begin{cases} \Rightarrow x0A1x^R \\ \Rightarrow x1A0x^R \end{cases}$ である。

2.文脈自由文法 (11)

よって、

$x \in L$ならば、

(i) xの長さが奇数、
または
(ii) $\exists\alpha, \exists\beta \in \{0,1\}^*$に対して、$x = \alpha 0 \beta 1 \alpha^R$ または $x = \alpha 1 \beta 0 \alpha^R$.

を導く文法は、

$$S \to S_1 \mid S_2$$
$$S_1 \to 0 \mid 1 \mid 00S_1 \mid 01S_1 \mid 10S_1 \mid 11S_1$$
$$S_2 \to 0S_2 0 \mid 1S_2 1 \mid 0A1 \mid 1A0$$
$$A \to \epsilon \mid 0A \mid 1A$$

で与えられる。

2.文脈自由文法 (12)

先ほどの例題の言語は、条件(i)を満たす言語と、条件(ii)を満たす言語の和集合という形で与えられた。

(i) (ii)がどのような条件であっても、それらがcfgで導出可能であれば、同様に(i)の言語と(ii)の言語の和集合をcfgで導出可能である。

つまり、$S \to S_1$と$S \to S_2$を生成規則として、S_1とS_2をそれぞれ(i)と(ii)の言語を導出するcfgの開始記号として使うとよい。

定理3.1 言語L_1とL_2が文脈自由文法で生成されるなら、それらの和集合も文脈自由文法で生成できる。

2.文脈自由文法 (13)

導出の過程で、2つ以上の変数があれば、どの変数に生成規則を適用してもよい。

最左導出：二つ以上の変数があるとき、常に一番左の変数に生成規則を適用する導出。

実は、最左導出に限定しても一般性を失わない。

定理3.2 文法Gによって、終端記号のみからなる列xが導出されるなら、最左導出によっても導出される。

証明は後で述べる。

2.文脈自由文法 (14)

例題3.6 以下の文法G_4はどのような言語を生成するか？

$$S \to SS \mid \epsilon \mid aB \mid bA$$
$$A \to a \mid bAA,$$
$$B \to b \mid aBB$$

(解答)

$$L(G_4) = \{x \in \{a, b\}^* \mid \#_a(x) = \#_b(x)\}$$

であることを証明する。

例として$aaababbb$を最左導出で導出してみる:

$$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabaBBB$$
$$\Rightarrow aaababBB \Rightarrow aaababbB \Rightarrow aaababbb$$

最左導出の場合、目標の列xに対して導出過程が（ある程度）一意に決まる。

2.文脈自由文法 (15)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

例として$aaababbb$を最左導出で導出してみる:

$$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabaBBB$$
$$\Rightarrow aaababBB \Rightarrow aaababbB \Rightarrow aaababbb$$

最左導出の場合、目標の列xに対して(ある程度)導出過程が一意に決まる。
今の場合、$S \to SS$を使わなくて導出できたので、これは使わないとする。
すると、最初の置き換えは $S \to aB$しかありえない。
($S \to bA$だと先頭の記号がbとなってしまう！)
同様に、次の置き換えは $B \to aBB$、その次も$B \to aBB$と決まる。

結局、導出途中では常に$\#_a + \#_A = \#_b + \#_B$となっている。

2.文脈自由文法 (16)

$$S \to SS \mid \epsilon \mid aB \mid bA$$
$$A \to a \mid bAA,$$
$$B \to b \mid aBB$$

Bは『将来bで置き換える$(B \to b)$』ための変数。

$B \to aBB$は、本来bで置き換える変数からaを作ったために、

バランスをとるためにBを追加している。

$S \to aB, B \to b, B \to aBB$のみでは、$\#_a \geq \#_b$が、

$S \to bA, A \to a, A \to bAA$のみでは、$\#_a \leq \#_b$が常に成立してしまう。

よって、前から順にプレフィックスを見たときに、途中で$\#_a - \#_b$の正負が変わっているような列は、上記の方法だけでは作成できない。

そのような列を作るためには$S \to SS$を使う必要がある。

2.文脈自由文法 (17)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

例として$aabbbaab$を考える

	a	a	b	b	b	a	a	b
$\#_a - \#_b$	1	2	1	0	−1	0	1	0

表から2回$\#_a - \#_b$の正負が逆転している。
そこで最初に$S \to SS$を2回使って以下のように最左導出する。

$$S \to SS \to SSS \to aBSS \to aaBBSS \to aabBSS \to aabbSS$$
$$\to aabbbAS \to aabbbaS \to aabbbaaB \to aabbbaab$$

この最左導出で任意性があるのは、$S \to SS$をいつ使うかということのみである。

以上のことを元にすると、$L(G_4) = \{x \in \{a,b\}^* \mid \#_a(x) = \#_b(x)\}$の証明を構築することができる。証明略

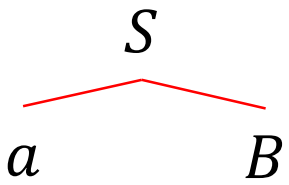
2.文脈自由文法 (18)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

導出木：文法と導出が与えられると決定される木。
　開始記号を根として、葉を左から右に読むと導出された列になる。

例：導出木の作成途中

S
$\Rightarrow aB$



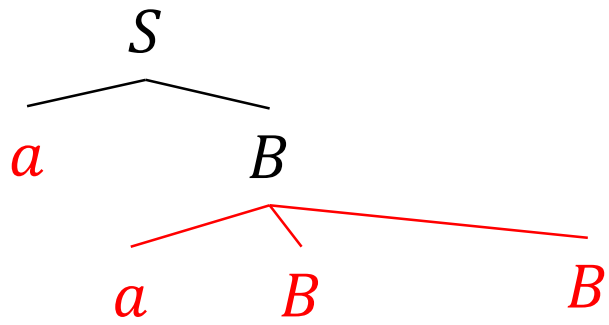
2.文脈自由文法 (18)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

導出木：文法と導出が与えられると決定される木。
　開始記号を根として、葉を左から右に読むと導出された列になる。

例：導出木の作成途中

S
$\Rightarrow aB$
$\Rightarrow aaBB$



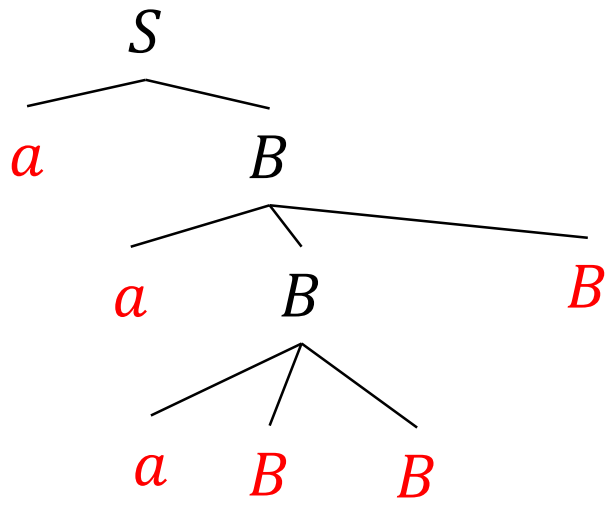
2.文脈自由文法 (18)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

導出木：文法と導出が与えられると決定される木。
　開始記号を根として、葉を左から右に読むと導出された列になる。

例：導出木の作成途中

S
$\Rightarrow aB$
$\Rightarrow aaBB$
$\Rightarrow aaaBBB$



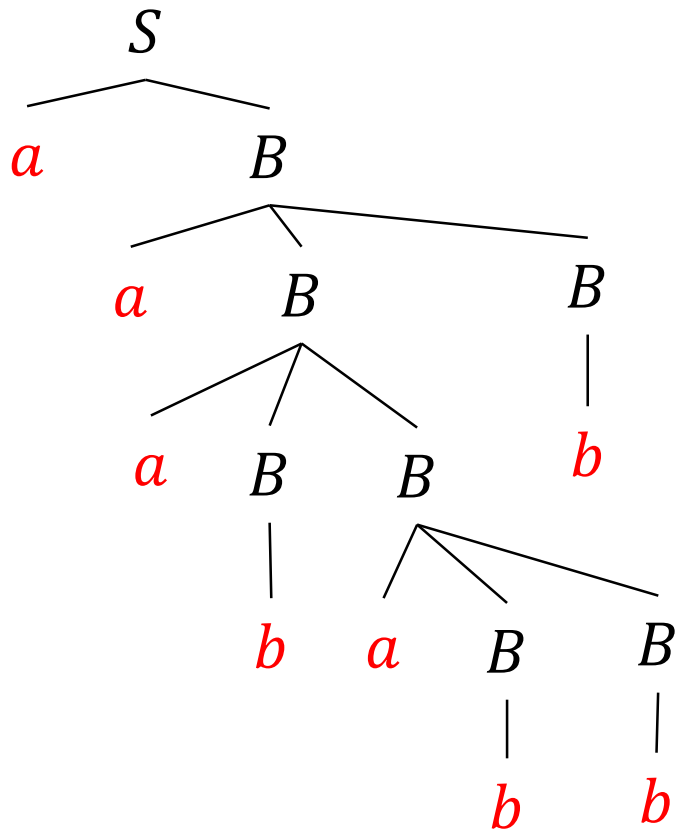
2.文脈自由文法 (18)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

導出木：文法と導出が与えられると決定される木。
　開始記号を根として、葉を左から右に読むと導出された列になる。

例：

S
$\Rightarrow aB$
$\Rightarrow aaBB$
$\Rightarrow aaaBBB$
$\Rightarrow aaabBB$
$\Rightarrow aaabaBBB$
$\Rightarrow aaababBB$
$\Rightarrow aaababbB$
$\Rightarrow aaababbb$
の導出木が完成



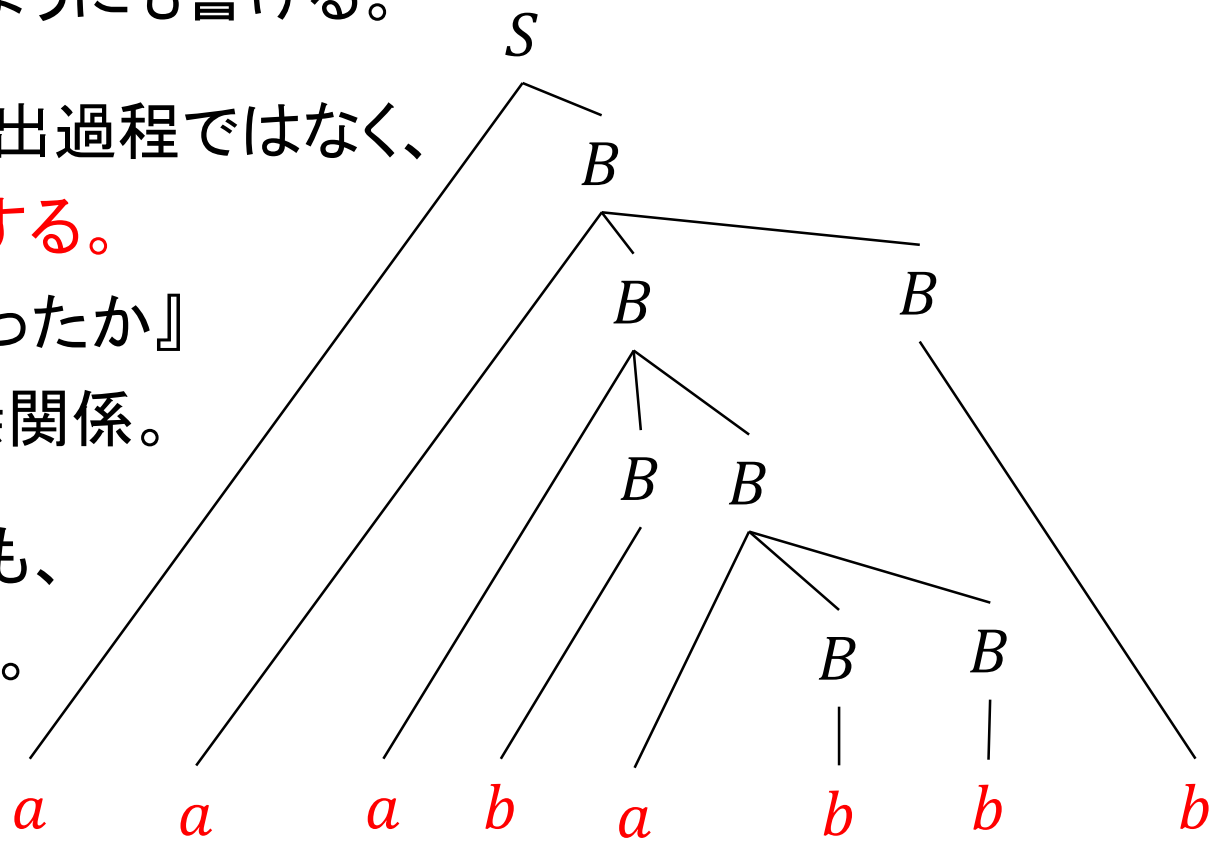
2.文脈自由文法 (19)

$S \to SS \mid \epsilon \mid aB \mid bA$
$A \to a \mid bAA,$
$B \to b \mid aBB$

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaabaBBB$
$\Rightarrow aaababBB \Rightarrow aaababbB \Rightarrow aaababbb$
の導出木は以下のようにも書ける。

導出される列は、導出過程ではなく、
導出木のみに依存する。
『どの順番で枝を作ったか』
は導出される列に無関係。

よって、最左導出でも、
同じ列が導出される。



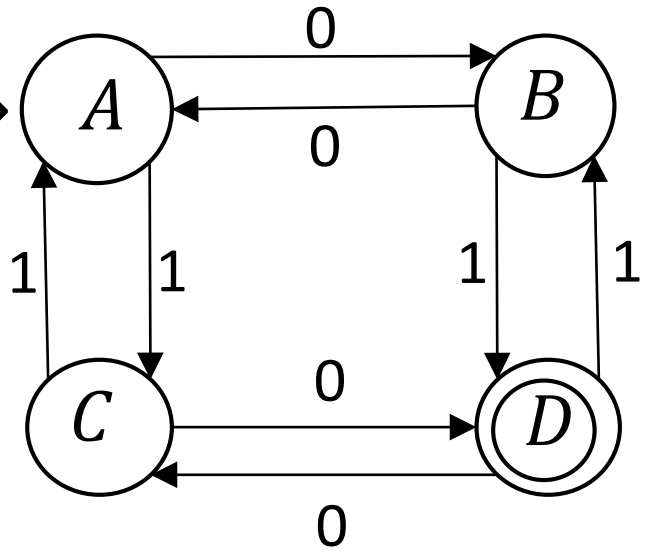
3.文脈自由文法と正規言語(1)

文脈自由文法は、正規言語より強力である。

定理3.3 Lが正規言語であるなら、Lを生成する文脈自由文法が存在する。

（証明の概要）

左の有限オートマトンを例として考える。 ⇒



開始記号をAとして、

$A \overset{0}{\rightarrow} B$に対応して、規則$A \rightarrow 0B$を、

$A \overset{1}{\rightarrow} C$に対応して、規則$A \rightarrow 1C$を、

・・・・と次々と規則を作っていく。

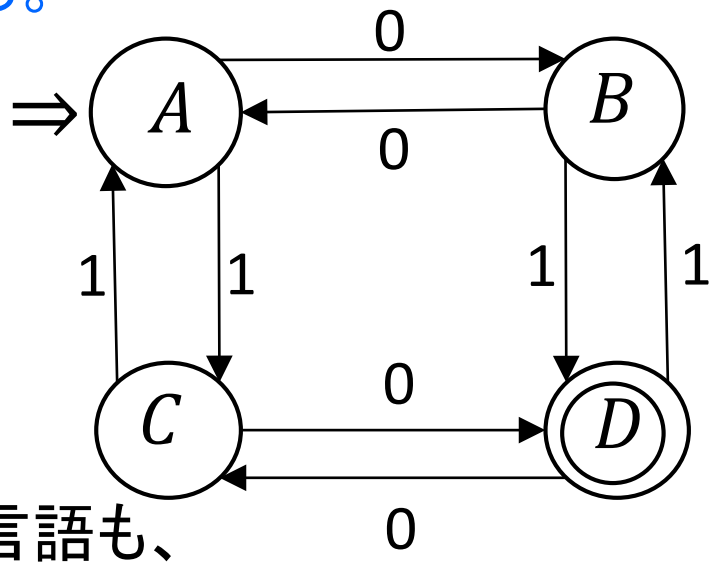
3.文脈自由文法と正規言語 (2)

$$A \to 0B \mid 1C, \qquad B \to 0A \mid 1D,$$
$$C \to 0D \mid 1A, \qquad D \to 0C \mid 1B$$

という規則が得られる。

Dが受理状態なので、$C \overset{0}{\to} D$と$B \overset{1}{\to} D$で受理される。

そこで、$B \to 1, C \to 0$を規則に付け加える。



0010の導出は、

$A \Rightarrow 0B \Rightarrow 00A \Rightarrow 001C \Rightarrow 0010$
とすればできる。

この作り方で、どのようなdfaが認識する言語も、
cfgで生成できることは、明らかであると思われる。

問題

以下の言語を生成する文脈自由文法を求めよ。アルファベットΣは$\{0,1\}$とする。

① $\{w \in \Sigma^* \mid w$は少なくとも3つの1を含む$\}$

② $\{w \in \Sigma^* \mid |w| \geq 2$ かつ、
wの先頭の文字と最後の文字は同じ$\}$

③ $\{w \in \Sigma^* \mid w$の長さは奇数$\}$

④ $\{w \in \Sigma^* \mid w$の長さは奇数で、真ん中の文字は0$\}$

⑤ $\{w \in \Sigma^* \mid w = w^R\}$　: 要するにwは回文

⑥ ∅　(空集合)

問題

以下の言語を生成する文脈自由文法を求めよ。

① $\{w \in \{a,b\}^* \mid |w| \geq 1$ かつ $\#_a(w) > \#_b(w)\}$

② $\{w \in \{a,b\}^* | \forall n \geq 0, w \neq a^n b^n\}$

③ $\{w\#x \mid w, x \in \{0,1\}^*$、かつ、$w^R$は$x$の部分列$\}$

④ $\{x_1\#x_2\#\cdots\#x_k \mid k \geq 1, x_i \in \{a,b\}^*$、かつ
$\exists i \exists j \left(x_i = x_j^R\right)\}$

④において$k = 1$のときは#は不必要とする。

4.文脈自由文法の標準形 (1)

文脈自由文法の生成規則には、左辺が1個の変数であること以外に制限はない。しかし、右辺にも制限を加えた標準形がいくつか知られている。

（重要な仮定） ここでは、cfgは、$A \to \epsilon$の形の生成規則を含まないとする。

・言語にϵが含まれなければ、右辺がϵの生成規則は必要ない

・言語にϵが含まれていたとしても、ϵが含まれる規則は、$S \to \epsilon$のみで、しかもSはどの規則の右辺にも表れないような文法で生成できる。

ということが成立するので、

このように制限しても一般性を失わない。

4.文脈自由文法の標準形 (2)

チョムスキーの標準形:
右辺が、ただ1個の終端記号か、2個の変数のみの生成規則しか含まない。

つまり、

$$A \to a_1 \mid a_2 \mid \cdots \mid A_1A_2 \mid A_3A_4 \mid \cdots$$
$$B \to b_1 \mid b_2 \mid \cdots \mid B_1B_2 \mid B_3B_4 \mid \cdots ,$$
$$C \to c_1 \mid c_2 \mid \cdots \mid C_1C_2 \mid C_3C_4 \mid \cdots ,$$

のような形をしている。

見かけは良いが、実は重要な応用が少ない。

4.文脈自由文法の標準形 (3)

チョムスキーの標準形:
右辺が、ただ1個の終端記号か、2個の変数のみの生成規則

一般的な証明は省略して、具体的に構築する方法を例示する。

例: 以下のcfgをチョムスキーの標準形に変形する。

$$A \to B|C, \qquad B \to b|D,$$
$$C \to F|ABB, \qquad D \to E, \qquad E \to B|ADa,$$

①最初に単一規則(右辺が変数1個の規則)を取り除く
まず、Aから始まる単一規則を取り除く

1-1: 単一規則だけを使って、Aから到達できる変数を調べる。

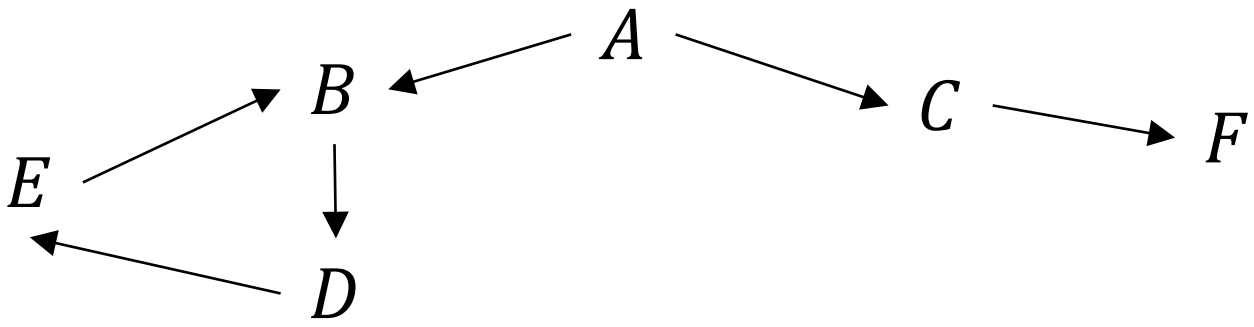
4.文脈自由文法の標準形 (4)

生成規則を略記せずに、すべて書き下すと:

$$A \to B, A \to C, B \to D, D \to E, E \to B, C \to F$$
$$B \to b, E \to ADa, C \to ABB$$

①最初に単一規則(右辺が変数1個の規則)を取り除く
まず、Aから始まる単一規則を取り除く

1-1: 単一規則だけを使って、Aから到達できる変数を調べる。



単一規則のみでの変換は上図のようになっており、Aから単一規則のみで到達できるのは、C, D, E, B, Fと全ての変数になっている。

4.文脈自由文法の標準形 (5)

例：以下のcfgをチョムスキーの標準形に変形する。

$$A \to B, A \to C, B \to D, D \to E, E \to B, C \to F$$
$$B \to b, E \to ADa, C \to ABB$$

①最初に単一規則(右辺が変数1個の規則)を取り除く
まず、Aから始まる単一規則を取り除く

1-1: 単一規則だけを使って、Aから到達できる変数を調べる。

1-2: 単一規則で$A \overset{*}{\Rightarrow} Z$ならば、$Z \to \alpha$ (αは何でもよい)という形の非単一規則すべてに対して、$A \to \alpha$ を新たに追加する。

Aからすべての変数に単一規則で到達できたので、

$A \to b, A \to ADa, A \to ABB$　を追加する。

1-3: Aから始まる単一規則を削除する。

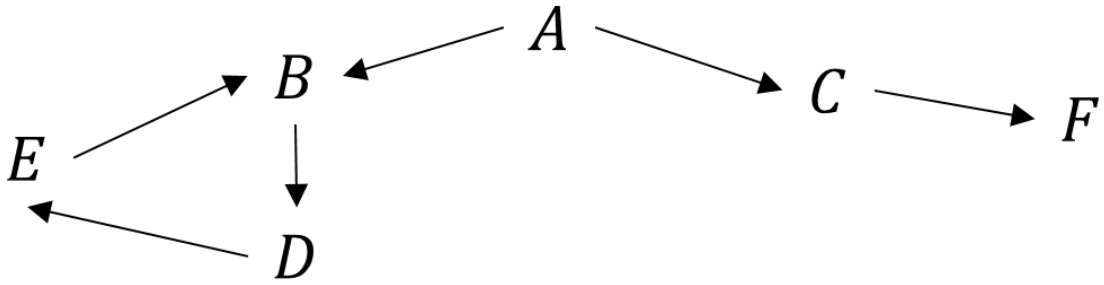
4.文脈自由文法の標準形 (6)

$B \to D, D \to E, E \to B, C \to F$
$B \to b, E \to ADa, C \to ABB$
$A \to b, A \to ADa, A \to ABB$

次に、Bから始まる単一規則を取り除く

1-1: 単一規則だけを使って、Bから到達できる変数を調べる。

Bから到達できるのは、
E, Dである。 ⇒



1-2: 単一規則で$B \overset{*}{\Rightarrow} Z$ならば、$Z \to \alpha$ (αは何でもよい)という形の非単一規則すべてに対して、$B \to \alpha$ を新たに追加する。

よって、$B \to ADa$が追加される。

1-3: Bから始まる単一規則を削除する。

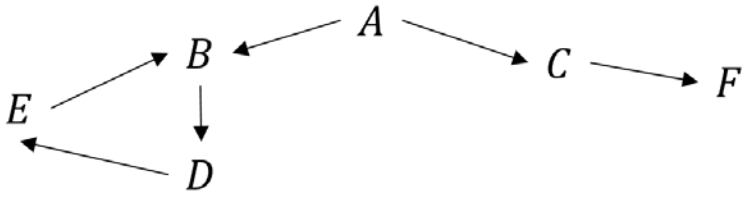
4.文脈自由文法の標準形 (7)

同様のことを、C, D, E, Fに対しても行うと、以下の規則が得られる:

$B \to b, E \to ADa, C \to ABB$

$A \to b, A \to ADa, A \to ABB$

$B \to ADa, D \to ADa, D \to b, E \to b$



次に、

② 右辺に終端記号が一つだけという規則以外の規則に対して、右辺の終端記号を変数で置き換えることで、右辺の終端記号を消去する。

例えば、$A \to ADa, B \to ADa, D \to ADa, E \to ADa,$ の右辺にaがあるが、変数X_aと規則 $X_a \to a$を導入して、

$$X_a \to a, A \to ADX_a, B \to ADX_a, D \to ADX_a, E \to ADX_a,$$

と規則を変形する。

4.文脈自由文法の標準形 (8)

$$A \to b, B \to b, D \to b, E \to b, X_a \to a$$
$$A \to ADX_a, A \to ABB, B \to ADX_a,$$
$$C \to ABB, D \to ADX_a, E \to ADX_a,$$

最後に

③ 新しい変数を導入することにより、右辺に3つ以上の変数がある規則を、複数の右辺に2つの変数しかない規則で置き換える。

具体的には、$A \to ADX_a$に対して、変数Y_1を導入して、
$A \to AY_1, Y_1 \to DX_a$とする。

この操作を続けると結局、以下のチョムスキーの標準形を得る:

$$A \to b, B \to b, D \to b, E \to b, X_a \to a,$$
$$A \to AY_1, Y_1 \to DX_a, A \to AY_2, Y_2 \to BB, B \to AY_3, Y_3 \to DX_a,$$
$$C \to AY_4, Y_4 \to BB, D \to AY_5, Y_5 \to DX_a, E \to AY_6, Y_6 \to DX_a,$$

4.文脈自由文法の標準形 (8)

この操作を続けると結局、以下のチョムスキーの標準形を得る:

$A \to b, B \to b, D \to b, E \to b, X_a \to a,$
$A \to AY_1, Y_1 \to DX_a, A \to AY_2, Y_2 \to BB, B \to AY_3, Y_3 \to DX_a,$
$C \to AY_4, Y_4 \to BB, D \to AY_5, Y_5 \to DX_a, E \to AY_6, Y_6 \to DX_a,$

同じ変数の規則を略記すると、以下になる:

$A \to b|AY_1|AY_2|,$
$B \to b|AY_3,$ $C \to AY_4,$ $D \to b|AY_5,$ $E \to b|AY_6,$
$X_a \to a,$
$Y_1 \to DX_a,$ $Y_2 \to BB,$ $Y_3 \to DX_a,$
$Y_4 \to BB,$ $Y_5 \to DX_a,$ $Y_6 \to DX_a,$

4.文脈自由文法の標準形 (8)

まとめると、チョムスキーの標準形を求める方法は:

①最初に単一規則(右辺が変数1個の規則)を取り除く。

② 右辺に終端記号が一つだけという規則以外の規則に対して、右辺の終端記号を変数で置き換えることで、右辺の終端記号を消去する。

③　新しい変数を導入することにより、右辺に3つ以上の変数がある規則を、複数の右辺に2つの変数しかない規則で置き換える。

という3ステップで行うことができる。

この方法は、どのような生成規則に対しても適応することができる。

4.文脈自由文法の標準形 (9)

グライバッハの標準形:
規則の右辺が左端の1個の終端記号とそれに続く0個以上の変数

つまり、すべての生成規則は、$A \to a\gamma$ 、
ただしaは終端記号で、γは変数記号だけからなる列

生成規則全体は、

$A \to a_{10} \mid a_{20} \mid \cdots \mid a_{11}A_{11} \mid a_{21}A_{21} \mid \cdots \mid a_{12}A_{12}A_{22} \mid a_{22}A_{32}A_{33} \mid \cdots$
$B \to b_{10} \mid b_{20} \mid \cdots \mid b_{11}B_{11} \mid b_{21}B_{21} \mid \cdots$
$\cdots$

のような形をしている。

4.文脈自由文法の標準形 (10)

練習問題　Gを以下のグライバッハの標準形で与える:

$$S \to 0SB \mid 2A$$
$$A \to 2$$
$$B \to 1$$

Gで、$x = 00221$が導出可能かどうかを確かめよ。

ヒント:最左導出に限ってよい。

すると、規則を適応する順番が決まってくる。

4.文脈自由文法の標準形 (11)

練習問題　Gを以下のグライバッハの標準形で与える:

$$S \to 0SB \mid 2A, \qquad A \to 2, \qquad B \to 1$$

Gで、$x = 00221$が導出可能かどうかを確かめよ

(解答例)　終端記号は左端にしかないことと、
xは0から始まることから、最初は$S \to 0SB$を使うしかない。
次も0なので、$S \to 0SB$、ここまでで、$S \Rightarrow 0SB \Rightarrow 00SBB$となる。
次は2なので、$S \to 2A$しかない。

このようにしていくと、以下の導出以外できないことが分かる:

$$S \Rightarrow 0SB \Rightarrow 00SBB \Rightarrow 002ABB \Rightarrow 0022BB \Rightarrow 00221B$$

しかし、まだBが残っている。
よって、00221はこの文法では導出できない。

４.文脈自由文法の標準形 (１２)

グライバッハの標準形は、右辺の最初が終端記号で始まっているので、目標とする列が分かっていると、
適応すべき生成規則の選択の幅が限られる。

⇒目標とする列が生成可能かどうかの判断がつきやすい。

生成規則が正確に1個の終端記号を含むので、
長さnの列を導出するなら規則の適用回数もちょうどn回

4.文脈自由文法の標準形 (13)

定理3.5 Lが文脈自由言語であるなら、Lを生成するグライバッハの標準形が存在する。

証明は非常に難しいので省略する。

以下のチョムスキーの標準形をグライバッハの標準形に直してみる:

$$S \to AX, S \to CC, X \to SB, A \to 0, B \to 1, C \to 2$$

後ろ3つの規則はグライバッハの標準形の条件を満たしている。

前の3つを終端記号から始まるように直そうとしてみる。

4.文脈自由文法の標準形 (14)

$$S \to AX, S \to CC, X \to SB, A \to 0, B \to 1, C \to 2$$

$X \to SB$を終端記号から始まるように変形してみる。

$S \overset{*}{\Rightarrow} \alpha X \gamma$で、$\alpha$は終端記号の列とする。
$X \to SB$　を適用すると、$S \overset{*}{\Rightarrow} \alpha X \gamma \Rightarrow \alpha SB\gamma$となる。
次は、$S \to AX,$ か$S \to CC$を適用できる。
$S \to AX$の場合、$S \overset{*}{\Rightarrow} \alpha X \gamma \Rightarrow \alpha SB\gamma \Rightarrow \alpha AXB\gamma \Rightarrow \alpha 0XB\gamma$
$S \to CC$の場合、$S \overset{*}{\Rightarrow} \alpha X \gamma \Rightarrow \alpha SB\gamma \Rightarrow \alpha CCB\gamma \Rightarrow \alpha 2CB\gamma$
しかありえない。
結局、$X \to SB$の代わりに、$X \to 0XB$と$X \to 2CB$を追加すればよい。

4.文脈自由文法の標準形 (15)

$$S \to AX, S \to CC, X \to SB, A \to 0, B \to 1, C \to 2$$

$X \to SB$の代わりに、$X \to 0XB$と$X \to 2CB$を追加すればよい。

同様の議論を$S \to AX$と$S \to CC$に行うと、これらをそれぞれ、$S \to 0X$と$S \to 2C$で置き換えれることが分かる。

このようにして、グライバッハの標準形が得られる:

$$S \to 0X, S \to 2C, X \to 0XB, X \to 2CB, A \to 0, B \to 1, C \to 2$$

しかし、以上の手法がどのようなcfgにも適応できるとは限らない。
上記の例では、$X \to 0XB$と$X \to 2CB$で良かったが、
例えば、Xが再度先頭に来てしまうcfgも存在したりする。

一般の場合のグライバッハの標準形の作り方は省略する。

5.文脈自由文法で生成できない言語 (1)

文脈自由文法は、正規表現より真に能力が高い。
しかし、それでも生成できない言語が存在する。

一般に、文脈自由文法で生成できないことの証明は難しいので、
以下の$uvwxy$定理を使う:

定理3.6
Lを有限でない文脈自由言語とする。すると、Lによって決まるある定数Kが存在して、Lに属す長さK以上の任意の列zに対し、
以下の3条件を満たす5個の列u, v, w, x, yが存在する。
(i) $z = uvwxy$と書ける.
(ii) $vx \neq \epsilon$.
(iii) 任意の$i \geq 1$に対して$uv^i wx^i y \in L$.

5.文脈自由文法で生成できない言語 (2)

定理3.6 Lを有限でない文脈自由言語とする。すると、Lによって決まるある定数Kが存在して、Lに属す長さK以上の任意の列zに対し、以下の3条件を満たす5個の列u, v, w, x, yが存在する。
(i) $z = uvwxy$と書ける. (ii) $vx \neq \epsilon$.
(iii) 任意の$i \geq 1$に対して$uv^i wx^i y \in L$.

例題3.8 $L = \{a^n b^n c^n \mid n \geq 1\}$ が文脈自由言語でないことを示せ。

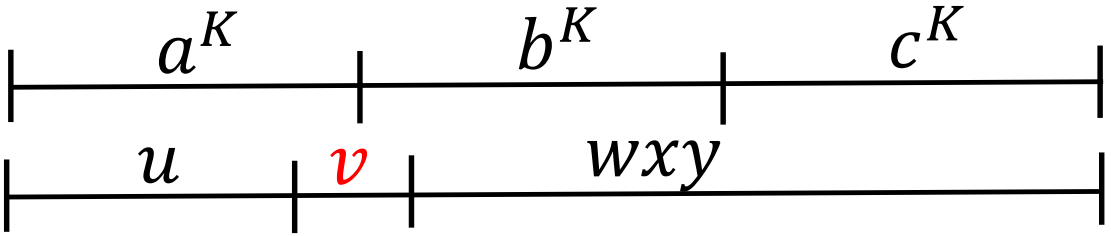
（証明）Lが文脈自由言語であると仮定する。すると、定理の条件を満たすKが存在する。
$a^K b^K c^K \in L$は長さK以上なので、$a^K b^K c^K = uvwxy$で、(ii)と(iii)の条件を満たす列u, v, w, x, yが存在する。
(ii)より、$vx \neq \epsilon$である。 まず、$v \neq \epsilon$の場合を考える。
vが$a^K b^K c^K$のどの部分に来るかで3つに場合分けする。

5.文脈自由文法で生成できない言語 (4)

$a^K b^K c^K = uvwxy$で、(ii)と(iii)の条件を満たす列u, v, w, x, yが存在する。(ii)より、$vx \neq \epsilon$である。 まず、$v \neq \epsilon$の場合を考える。
vが$a^K b^K c^K$のどの部分に来るかで場合分けする。

(場合1)$v = a^i b^j c^l, i \neq 0, j \neq 0$ （vがaとbの境界をまたぐ）のとき。
uv^2wx^2yは、$\cdots a \cdots b \cdots a \cdots b \cdots$という形になり$a^n b^n c^n$と表せないので$L$に入らない。よって、このような分解は不可能。

vがbとcの境界をまたぐ場合も、同様にありえないことが示せる。

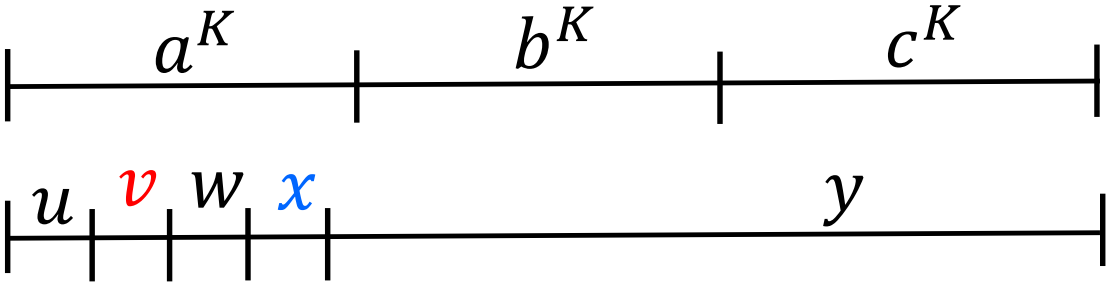


5.文脈自由文法で生成できない言語 (5)

(場合2)$v = a^i, i \neq 0$　（vがa^Kの内部）のとき。

次にxをどこに取るか考える。aとbもしくはbとcの境界をまたぐと(場合1)と同様に矛盾が生じる。よって、xはa^Kもしくはb^Kもしくはc^Kの内部である。

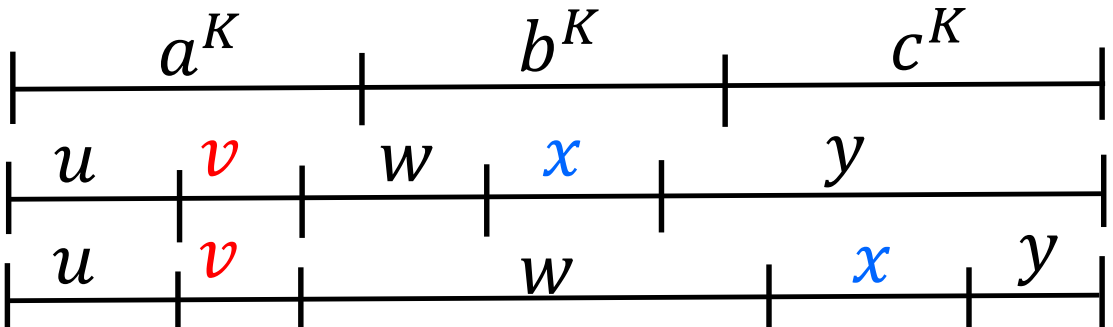
今、xをa^Kの内部に取るとする。このとき、uv^2wx^2yを考えると、aの数は増えるのに、bとcの数は増えない。よって、$a^n b^n c^n$の形には書けない。よって、uv^2wx^2yはLに入らず、このような分解はできない。



5.文脈自由文法で生成できない言語 (5')

次に、xをb^Kの内部に取るとする。このとき、uv^2wx^2yを考えると、aとbの数は増えるのに、cの数は増えない。よって、$a^nb^nc^n$の形には書けない。よって、uv^2wx^2yはLに入らず、このような分解はできない。

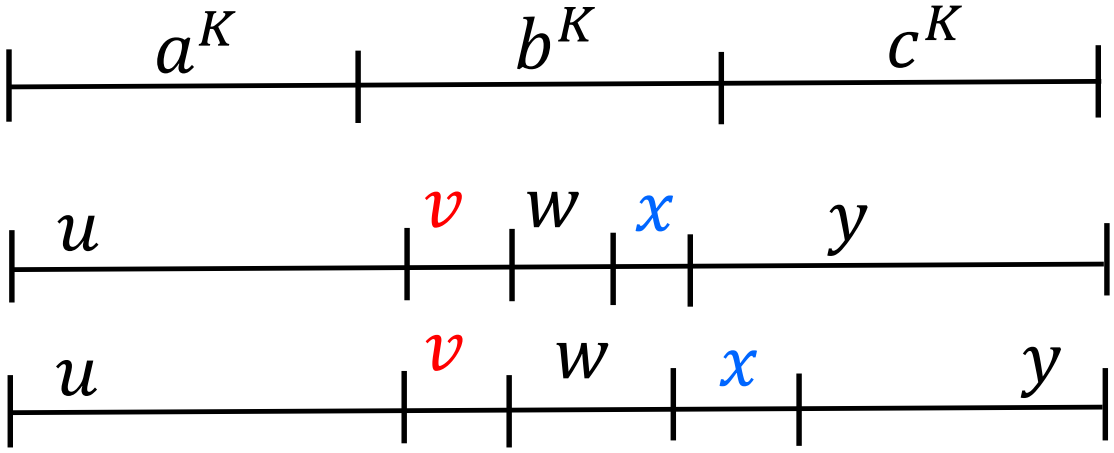
同様に、xをc^Kの内部に取った場合も、a, b, cの数のバランスがおかしくなる(bが増えない)ことから、矛盾が生じる。



5.文脈自由文法で生成できない言語 (6)

(場合3)$v = b^i, i \neq 0$　（vがb^Kの内部）のとき。

a^Kはuの部分列である。uv^2wx^2yを考えると、bは増えるのに、aは増えない。よって、aとbが同じ数にならないので、$a^n b^n c^n$の形には書けない。



5.文脈自由文法で生成できない言語 (6)

(場合4)$v = c^i, i \neq 0$　（vがc^Kの内部）のとき。

$a^K b^K$はuの部分列である。uv^2wx^2yを考えると、cは増えるのに、aとbは増えない。よって、a, bとcが同じ数にならないので、$a^n b^n c^n$の形には書けない。

以上は$v \neq \epsilon$の場合であるが、$x \neq \epsilon$の場合もxに関して同様の場合分けをすることで、同じ議論が成立し、全ての場合において矛盾を示せる。

結局、$uvwxy$定理が保証している分解を$a^K b^K c^K$に施すことができないので、$L = \{a^n b^n c^n \mid n \geq 1\}$ は文脈自由言語ではない。 ■

5.文脈自由文法で生成できない言語 (7)

定理3.6 Lを有限でない文脈自由言語とする。すると、Lによって決まるある定数Kが存在して、Lに属す長さK以上の任意の列zに対し、以下の3条件を満たす5個の列u, v, w, x, yが存在する。
(i) $z = uvwxy$と書ける. (ii) $vx \neq \epsilon$.
(iii) 任意の$i \geq 1$に対して$uv^i wx^i y \in L$.

（証明のアイデア）

具体例で考える:

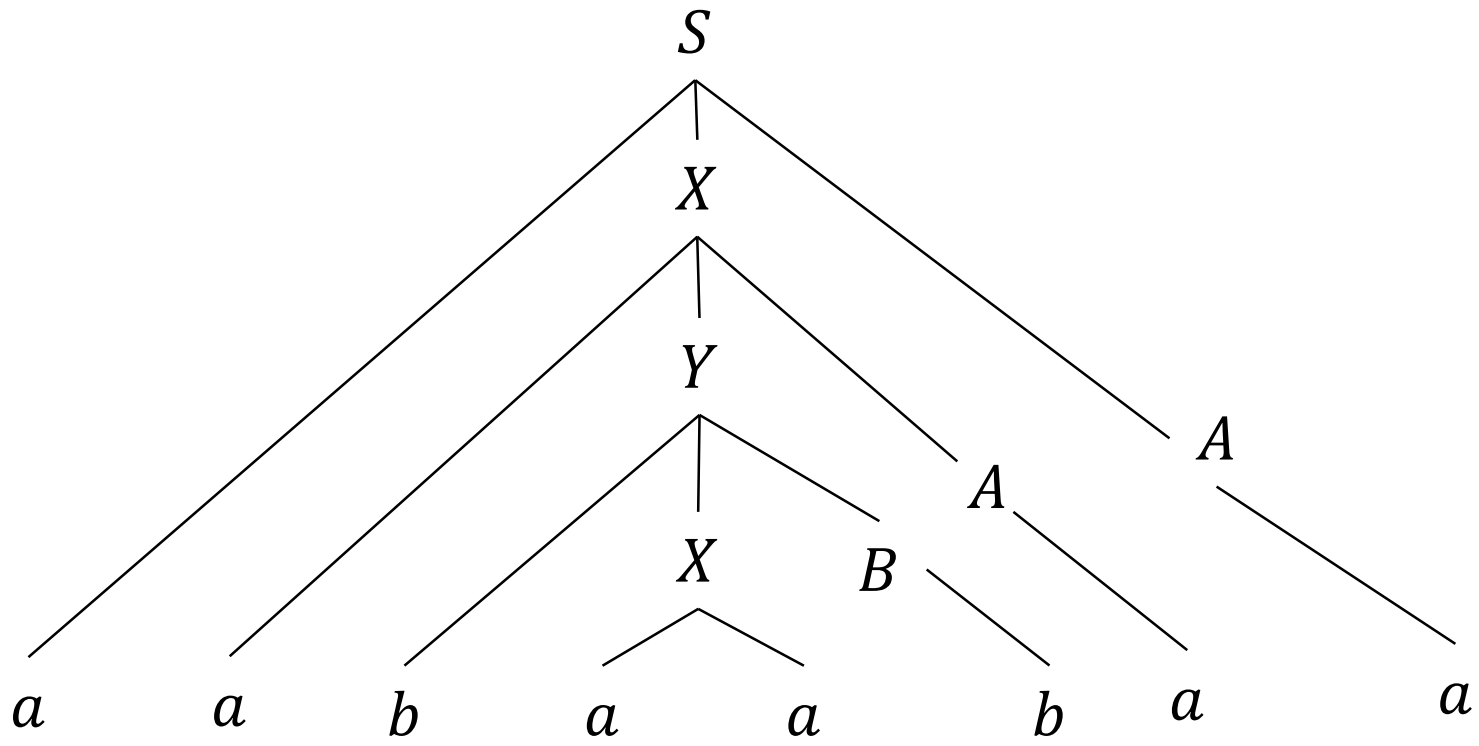
$$S \to aXA \mid bXB,$$
$$X \to aa|bb|aYA| bYB,$$
$$Y \to aa|bb|aXA| bXB,$$
$$A \to a, B \to b,$$

この規則で、$aabaabaa$を導出する導出木を作ってみる。

5.文脈自由文法で生成できない言語 (7)

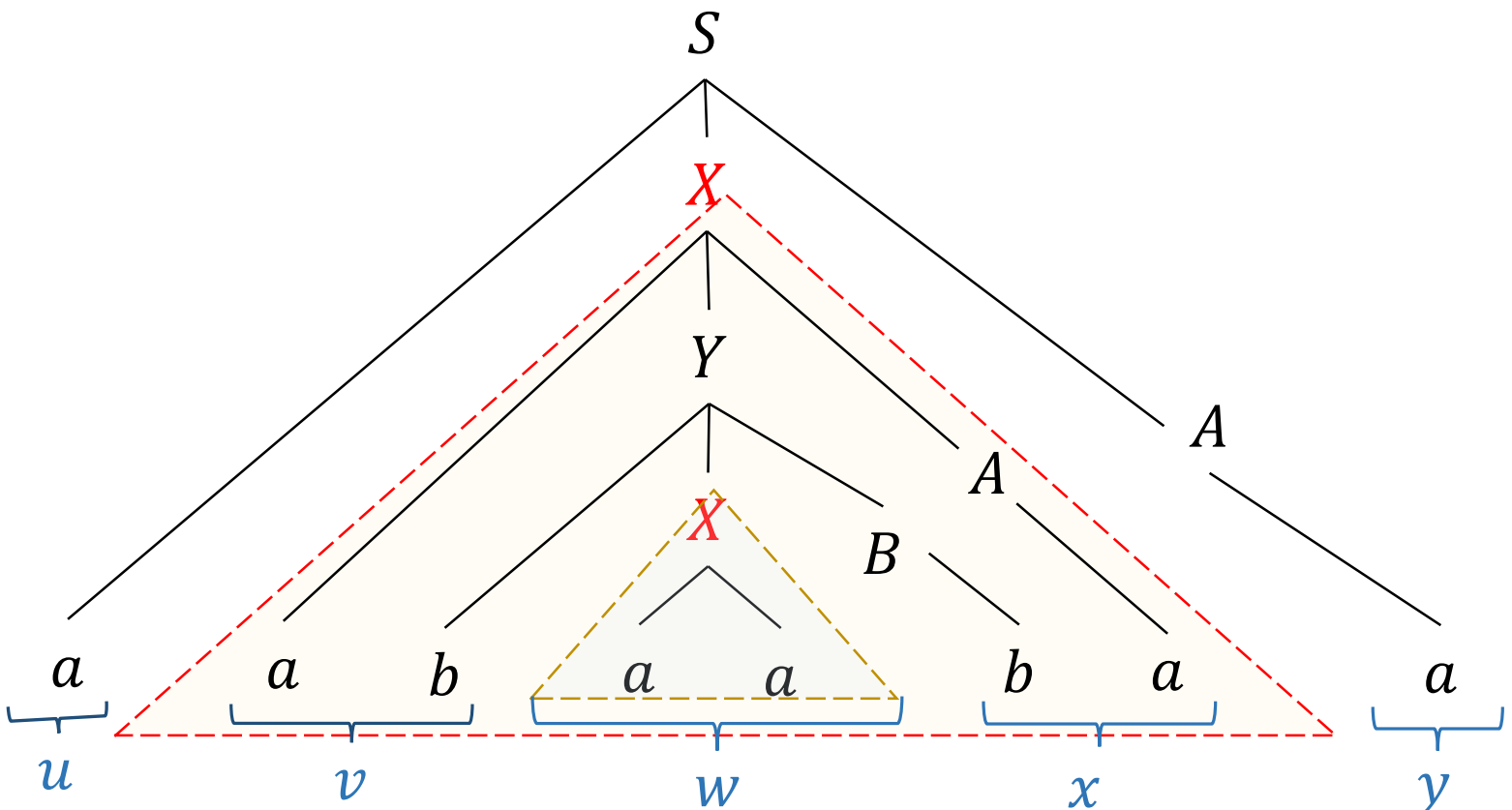
$S \to aXA \mid bXB, \qquad A \to a, B \to b,$
$X \to aa|bb|aYA| \ bYB, \qquad Y \to aa|bb|aXA| \ bXB,$

この規則で、$aabaabaa$を導出する導出木を作ってみる。



5.文脈自由文法で生成できない言語 (8)

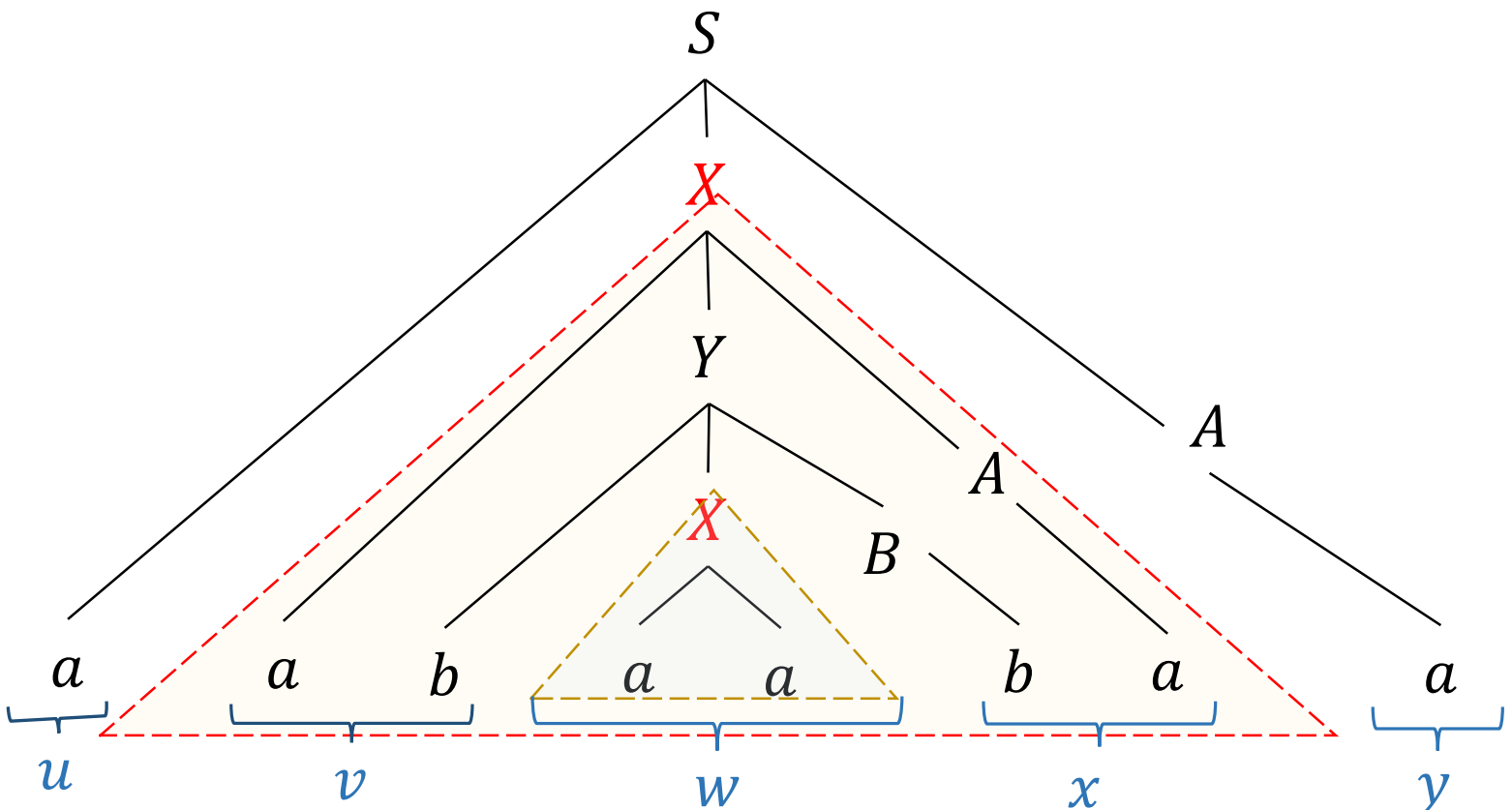
この規則で、$aabaabaa$を導出する導出木を作ってみる。
根から葉までの一番長いパスには、Xが2回現れる。
以下のようにu, v, w, x, yを定義する。



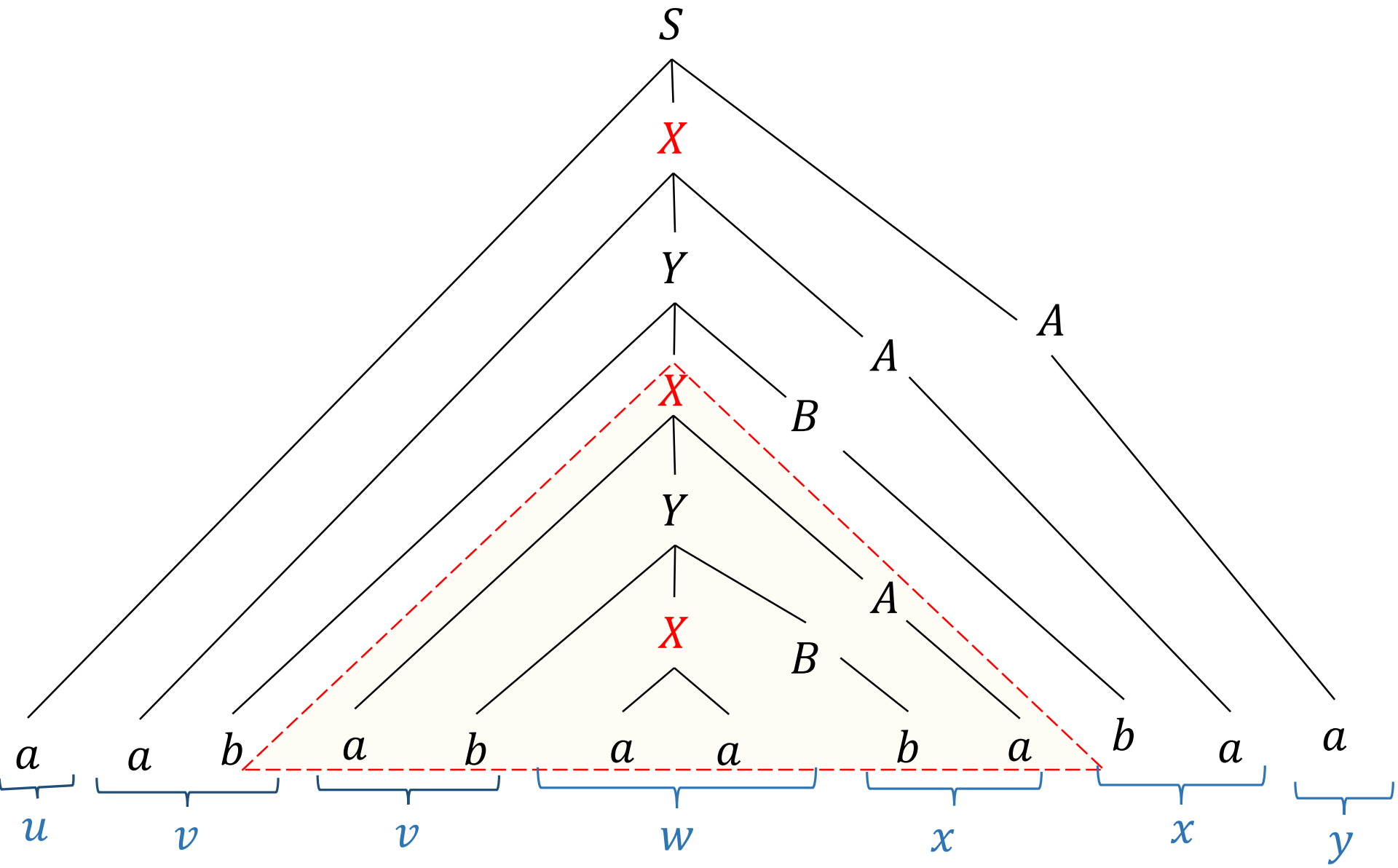
5.文脈自由文法で生成できない言語 (9)

小さい方の三角(2回目のXより下の部分木)を

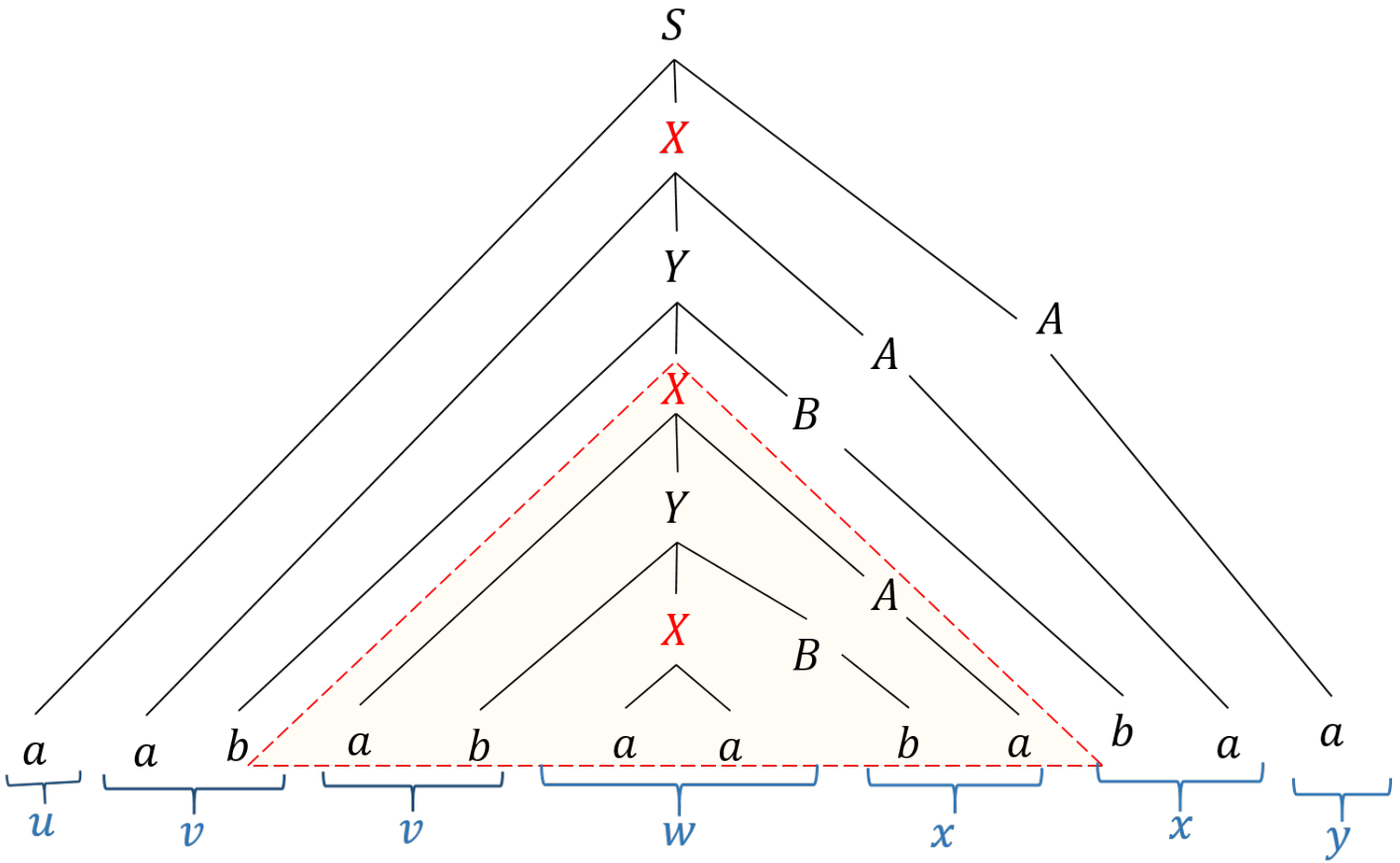
大きい方の三角(1回目のXより下の部分木)で置き換えても、導出される列はLに入っているはずである。



5.文脈自由文法で生成できない言語 (10)



5.文脈自由文法で生成できない言語 (11)

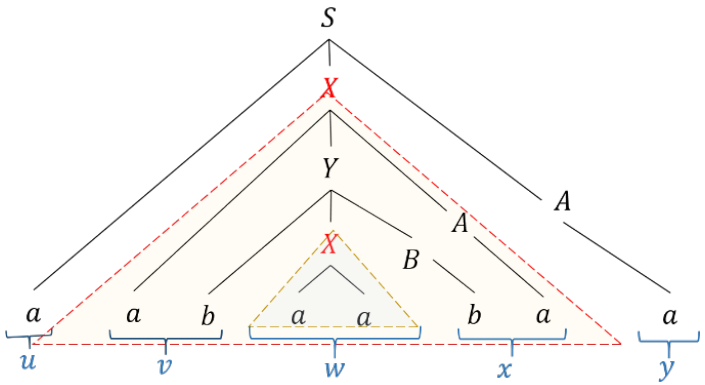


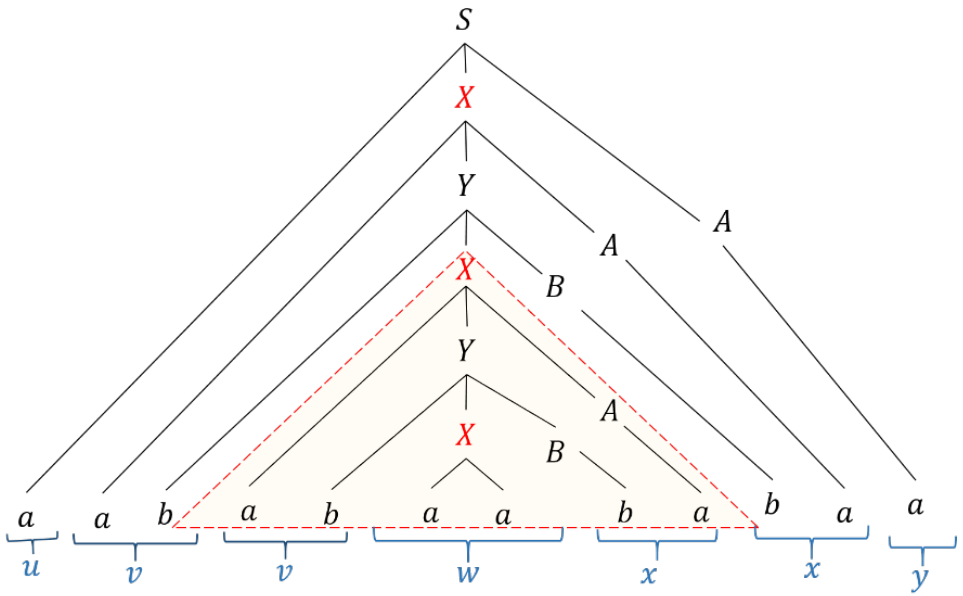
この導出木から、uv^2wx^2yが導出可能であることが分かる。
この木の3番目のXより下を、赤い三角でさらに置き換えることで、uv^3wx^3yを得る。このようにして、任意のi に対して、uv^iwx^iyが導出可能であることが分かる。

5.文脈自由文法で生成できない言語 (12)

今の例では、Xが導出木に2回現れたことが、uv^iwx^iyを作るポイントであった。しかし、どのようなcfgでも、変数の数は有限なので、十分に長い列の導出木には、2回以上現れる変数が存在する。

よって、どのようなcfgでも同様の議論が成立することになり、定理が正しいことが分かる。





5.文脈自由文法で生成できない言語 (13)

$uvwxy$定理を使うと以下のような言語が、文脈自由文法で生成できないことが分かる。証明は参考書を参照。

$$L = \{zz \mid z \in \{a, b\}^*\}$$

$$L = \left\{0^n 1^{n^2} \mid n \geq 0\right\}$$
$$L = \left\{0^{2^n} \mid n \geq 0\right\}$$

一般に、記号の数を数えるnが「非線形」である言語は、cfgでは生成できない。

第四章：
プッシュダウンオートマトン

スタックを使用できるオートマトン．
文脈自由文法と同等の表現力を持つ。

1. プッシュダウンオートマトン
2. プッシュダウンオートマトンの設計
3. 文脈自由文法との等価性

1.プッシュダウンオートマトン(1)

有限オートマトンの能力の限界は、記憶状態が有限であることが原因であった。しかし、状態を無限個作ると、状態遷移図が書けないので困る。

そこで、状態を無限個作る代わりに、入出力動作が制限されている『補助記憶装置』を導入する。

補助記憶装置としてプッシュダウンスタック(『スタック』と呼ぶ)を用いるのが、プッシュダウンオートマトン(pdaと略する)である。

1.プッシュダウンオートマトン (2)

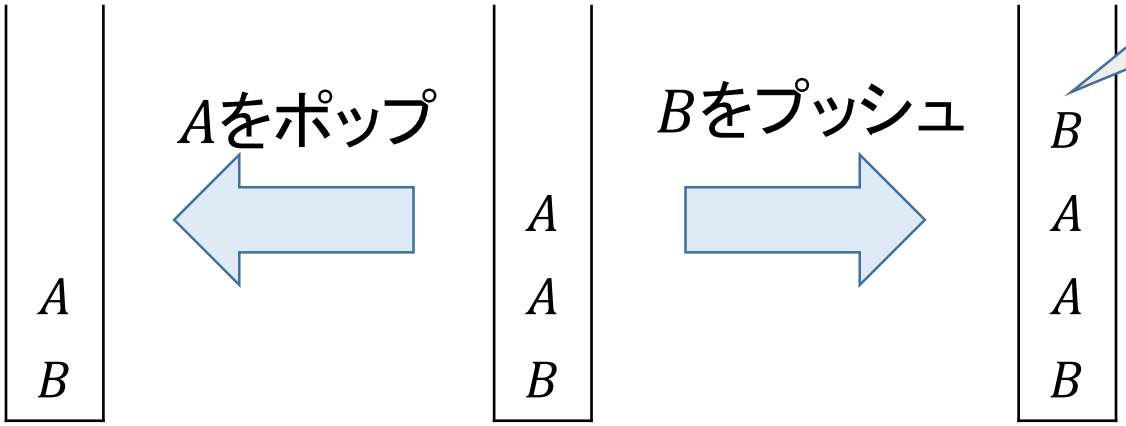
スタックには、4種類の動作が許されている:

(i) スタックの先頭の記号を読む。

(ii) スタックの先頭の記号を取り去る(ポップ)。
先頭の記号を取り去った後は、2番目の記号が現れる。

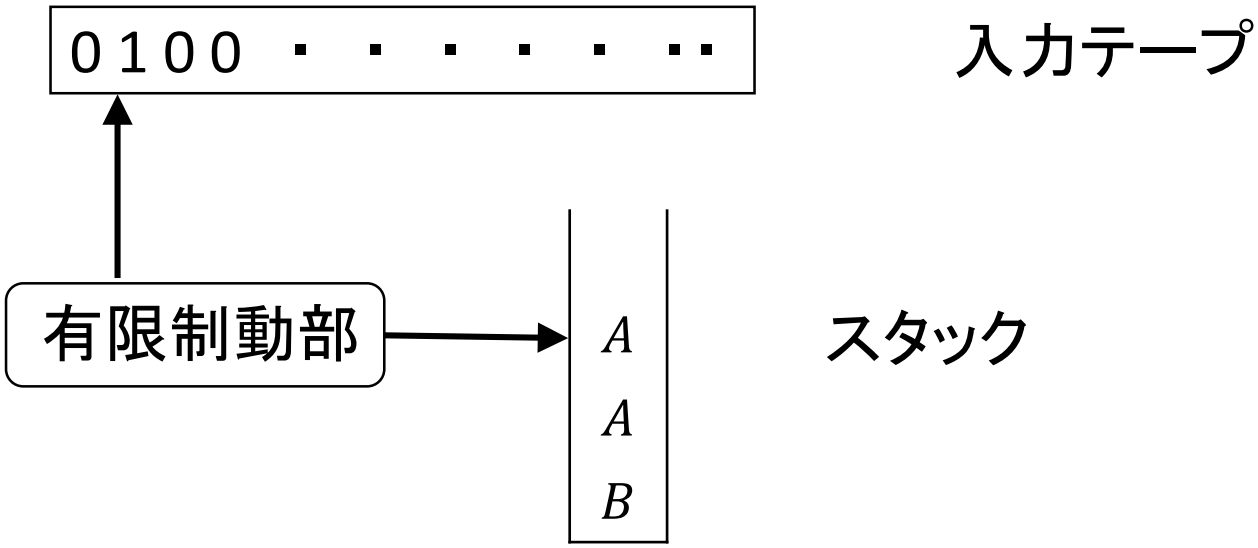
(iii) 新たな記号をスタックの先頭に詰め込む(プッシュ)。

(iv) スタックの内容には手を付けない。

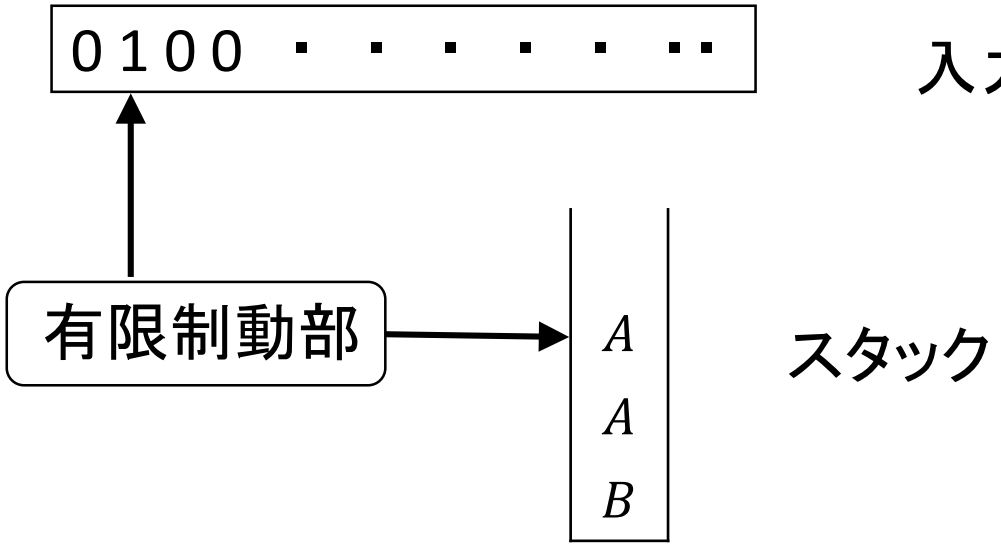


1.プッシュダウンオートマトン (3)

プッシュダウンオートマトンは、
入力テープ、有限制動部とスタックから構成される。



1.プッシュダウンオートマトン (4)



pdaの1ステップの動作:

(i) ヘッドの下の入力記号とスタックの先頭記号を読んで入力ヘッドを1コマ右に移動する。

(ii) 読んだ記号と現在の状態で決まるスタック演算(ポップ、プッシュ、もしくは何もしない)を実行する。

(iii) 読んだ記号で決定される次の状態に遷移する。

1.プッシュダウンオートマトン (5)

$L = \{0^n 1^n \mid n \geq 0\}$を認識するpda Mは以下で与えられる:

状態1: 記号0を読むと、スタックにZをプッシュ。
　　記号1を読むと、スタックをポップし、状態2に遷移。

状態2: 記号1を読むと、スタックの先頭の記号をポップ。

☆上記に書いていない全ての場合で、入力を拒否。

読んだ0の個数と1の個数が同一の場合は、入力列を読み終わったときに、
ちょうどスタックが1個の記号もない状態になる(空になる。)

0の個数の方が多いと、入力列が読み終わってもスタックが空にならず。

1の個数の方が多いと、入力列が読み終わる前にスタックが空になる。
⇒入力列を読み終わったときに、スタックが空になればOK！

1.プッシュダウンオートマトン (7)

定義 (非決定性)プッシュダウンオートマトン(pda)は、
$M = (K, \Sigma, \Gamma, \delta, s_0, Z_0)$で与えられる。

K: 状態　　（有限集合）
Σ: 入力記号　　（有限集合）
Γ: スタック記号　（有限集合）
$s_0 \in K$: 初期状態
$Z_0 \in \Gamma$: スタック初期記号

δ: $K \times \Sigma \times \Gamma \to 2^{K \times \Gamma^*}$　状態遷移関数

δがややこしいので、詳細に解説する。

1.プッシュダウンオートマトン (8)

$M = (K, \Sigma, \Gamma, \delta, s_0, Z_0)$
K: 状態、Σ: 入力記号、 Γ: スタック記号、
$s_0 \in K$: 初期状態、$Z_0 \in \Gamma$: スタック初期記号

δ: $K \times \Sigma \times \Gamma \to 2^{K \times \Gamma^*}$　状態遷移関数

もし決定性ならδ: $K \times \Sigma \times \Gamma \to K \times \Gamma^*$となっているはず。この場合、

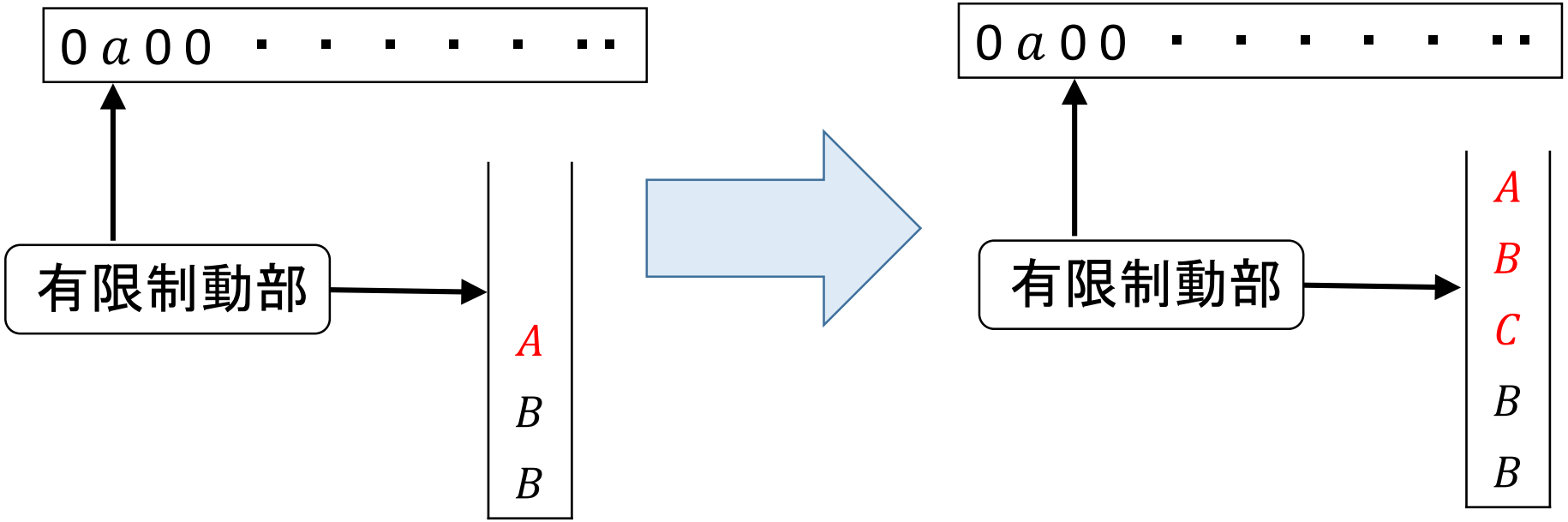
$$\delta(s, a, A) = (s', ABC)$$

は、現在の状態がsで、入力記号としてaを読み、スタックの先頭の記号がAならば、
次の状態はs'で、スタックの先頭の記号（今はA）をスタック記号の列ABCで置き換えるという動作を指定する。

1.プッシュダウンオートマトン (9)

$$\delta(s, a, A) = (s', ABC)$$

は、現在の状態がsで、入力記号としてaを読み、スタックの先頭の記号がAならば、
次の状態はs'で、スタックの先頭の記号（今はA）をスタック記号の列ABCで置き換えるという動作を指定する。



1.プッシュダウンオートマトン(10)

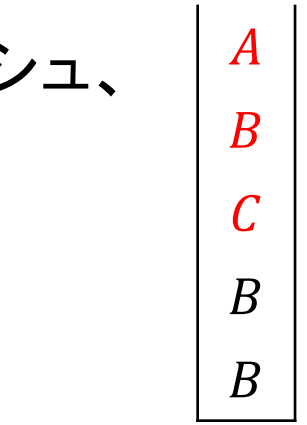
$$\delta(s, a, A) = (s', ABC)$$

は、現在の状態がsで、入力記号としてaを読み、スタックの先頭の記号がAならば、
次の状態はs'で、スタックの先頭の記号(今はA)をスタック記号の列ABCで置き換えるという動作を指定する。

以前の約束では、
① Aをポップ、②Cをプッシュ、③Bをプッシュ、④Aをプッシュ、という4ステップの動作になるところを、

① Aをポップ、②ABCをプッシュ、という

1ステップの動作で実行できる定義になっている。



1.プッシュダウンオートマトン(11)

この定義では、
『Aをポップして状態をs''にする』は、

$$\delta(s, a, A) = (s'', \epsilon)$$

『スタックに何もせずに状態をs'''にする』は、

$$\delta(s, a, A) = (s''', A)$$

と書ける。

今は、非決定で定義されているので、$\delta\colon K \times \Sigma \times \Gamma \to 2^{K \times \Gamma^*}$

$$\delta(s, a, A) = \{(s', ABC), (s'', \epsilon)\}$$

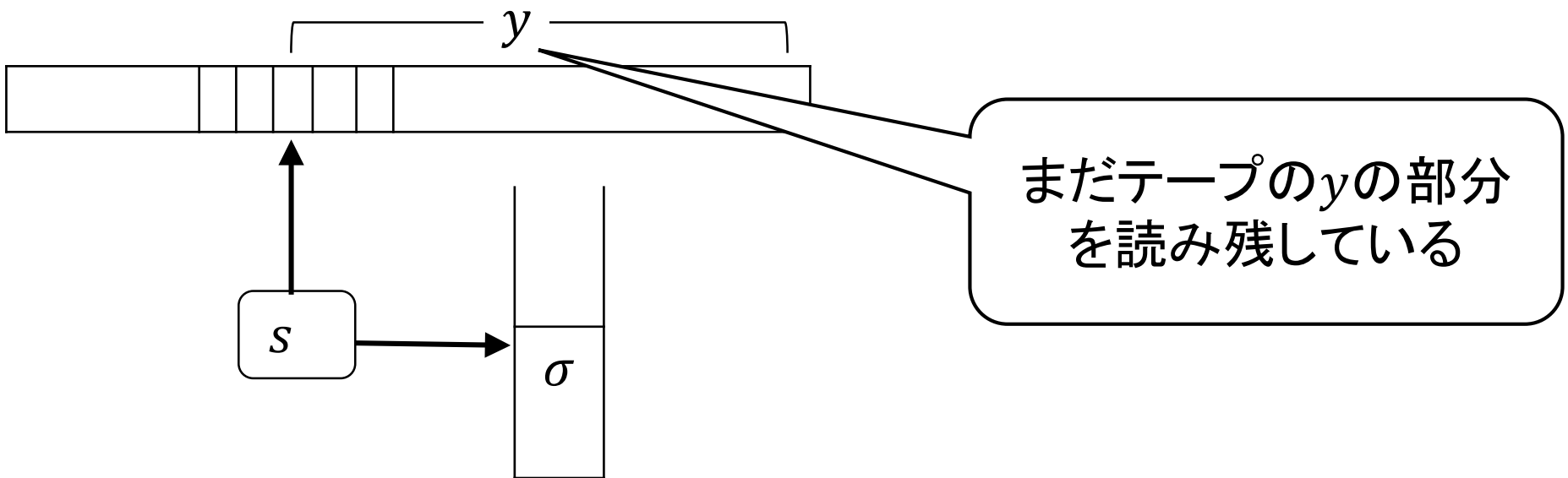
のようになる。
これは、『AをABCで置き換え、状態をs'にする』ことも『Aをポップして、状態をs''にする』ことも可能であると解釈する。

1.プッシュダウンオートマトン(12)

定義： 状態$s \in K$、入力列$y \in \Sigma^*$、スタック記号の列$\sigma \in \Gamma^*$としたとき、三つ組(s, y, σ)を様相という。

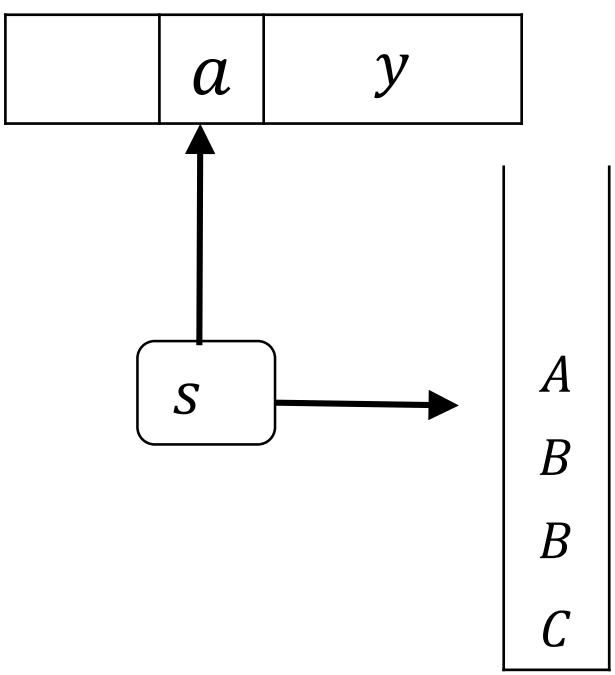
Mの様相が(s, y, σ)である。

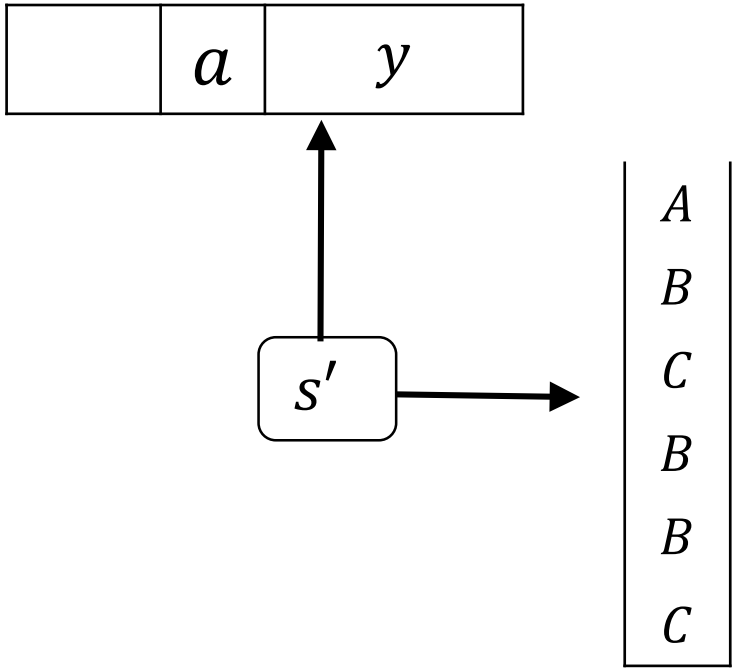
⇔ 状態はsで、入力列yを読み残し、スタックの内容は列σである。



1.プッシュダウンオートマトン (13)

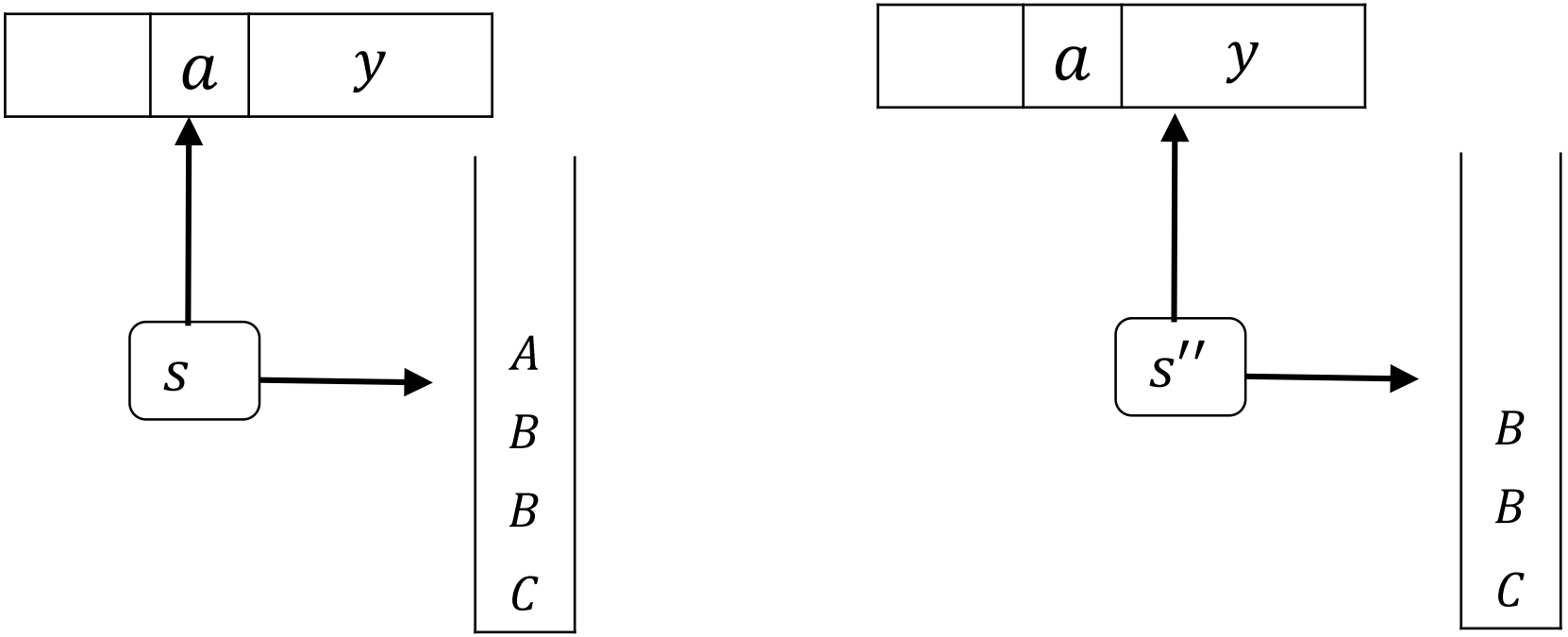
$\delta(s, a, A) = \{(s', ABC), (s'', \epsilon)\}$とすると、
様相$(s, ay, ABBC)$から様相$(s', y, ABCBBC)$に1ステップで変りうる。





1.プッシュダウンオートマトン (14)

同様に、$\delta(s, a, A) = \{(s', ABC), (s'', \epsilon)\}$なら
様相$(s, ay, ABBC)$から様相(s'', y, BBC)に1ステップで変りうる。



1.プッシュダウンオートマトン(15)

『1ステップで変りうる』の正式な定義:

$s_1, s_2 \in K, a \in \Sigma, y \in \Sigma^*, A \in \Gamma, \rho, \sigma \in \Gamma^*$であるとき:

$S_1 = (s_1, ay, A\rho)$から$S_2 = (s_2, y, \sigma)$へ1ステップで変りうる

$\Leftrightarrow$ ある$\alpha \in \Gamma^*$が存在して、$(s_2, \alpha) \in \delta(s_1, a, A)$かつ$\sigma = \alpha\rho$

様相S_1からS_2に1ステップに変わりうるとき、$S_1 \Rightarrow S_2$と書く

定義: $S_1 = S_n$、または$S_1 \Rightarrow S_n$、またはあるnに対して、
$S_1 \Rightarrow S_2 \Rightarrow \cdots \Rightarrow S_n$であるような$S_2, \cdots, S_{n-1}$が存在するとき、

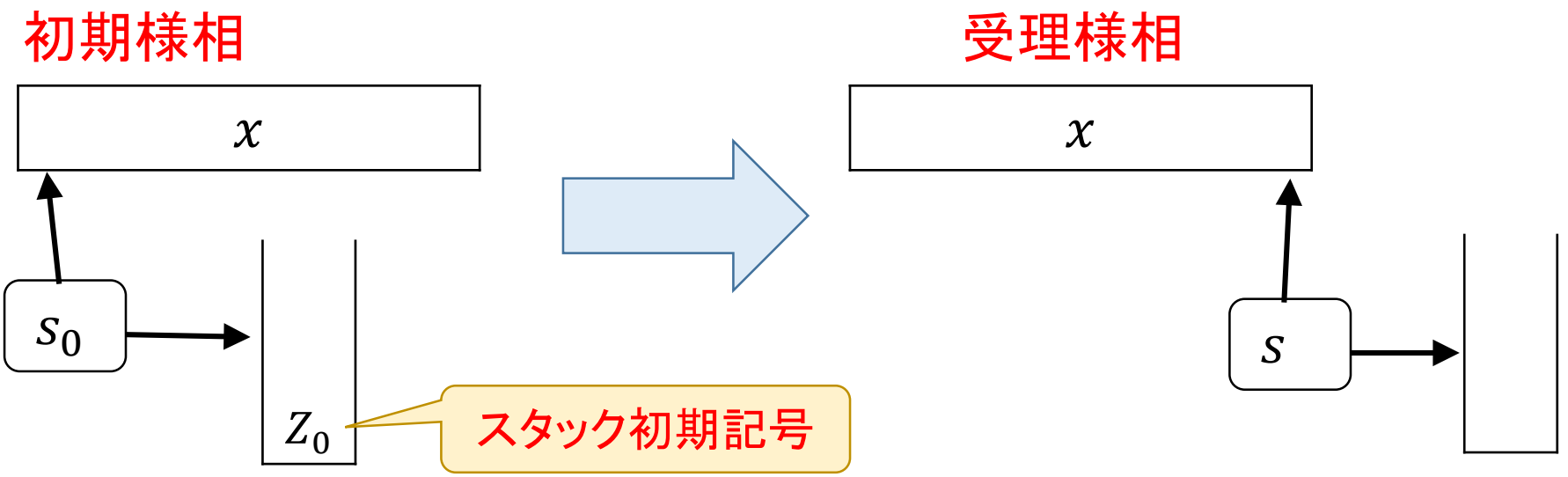
様相S_1はS_nに何ステップかで変れると呼び、$S_1 \overset{*}{\Rightarrow} S_n$と書く

1.プッシュダウンオートマトン (16)

定義： 列$x \in \Sigma^*$がある状態$s \in K$に対して条件

$$(s_0, x, Z_0) \overset{*}{\Rightarrow} (s, \epsilon, \epsilon)$$

を満たすとき、xはpda Mによって受理されるという。
受理される列全体をMで認識される言語と呼ぶ。



テープとスタックが同時に空になっているのが、受理様相の条件

1.プッシュダウンオートマトン (17)

注意1:テープだけが空の様相: (s, ϵ, σ)

と

スタックだけが空の様相:(s, y, ϵ)

は、どのような様相にも変わることができない。

つまり、テープだけ、もしくはスタックだけが空になるとpdaは止まってしまう。

理由:状態遷移関数が、δ: $K \times \Sigma \times \Gamma \to 2^{K \times \Gamma^*}$ なので、空列からの写像が定義されていないから。

1.プッシュダウンオートマトン (17')

注意2:本講義のpdaの定義を用いると空列は受理できない。

理由:入力列xがϵの時、
初期様相は(s_0, ϵ, Z_0)だが、
δ: $K \times \Sigma \times \Gamma \to 2^{K \times \Gamma^*}$ なので、
この様相からの写像は定義されていない。
従って $(s_0, x, Z_0) \overset{*}{\Rightarrow} (s, \epsilon, \epsilon)$ という遷移は不可能である。

本章では入力列はϵを含まないと仮定:
δ: $K \times \Sigma \cup \{\epsilon\} \times \Gamma \to 2^{K \times \Gamma^*}$ と状態遷移関数の定義を修正すると、ϵの受理も可能になるが、本章ではこちらの定義は用いない。

1.プッシュダウンオートマトン (18)

例題4.1 $L = \{xcx^R \mid x \in \{a, b\}^*\}$を認識するpdaを構成せよ。

解答:

状態集合: $K = \{s_0, s_1\}$ s_0は初期状態

スタック記号: $\Gamma = \{A, B, Z_0\}$ A はaに、Bはbに対応、

とする。

- 列を左から見ていき、cが現れるまで読んだ入力記号をスタックに詰め込んでいく。(積み上げモード)
  (例えば、aabを読めば、スタックは先頭からBAAとなる。)
- cが現れれば動作モードを変え、1文字ずつポップしながら、スタックの先頭の記号と一致するかをチェックする。(チェックモード)
  (上の例なら、後半の入力列がbaaならチェック成功)

1.プッシュダウンオートマトン(19)

状態遷移関数: (書かれていない組み合わせに対する値はø)

$$\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, a, A) = \{(s_0, AA)\}$$
$$\delta(s_0, b, Z_0) = \{(s_0, B)\}, \delta(s_0, b, A) = \{(s_0, BA)\},$$
$$\delta(s_0, a, B) = \{(s_0, AB)\}, \delta(s_0, b, B) = \{(s_0, BB)\}$$

s_0では入力記号をスタック記号として積んでゆく

$$\delta(s_0, c, A) = \{(s_1, A)\}, \delta(s_0, c, B) = \{(s_1, B)\}, \delta(s_0, c, Z_0) = \{(s_1, \epsilon)\}$$

cを読むとs_1に状態変化させる

$$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$$

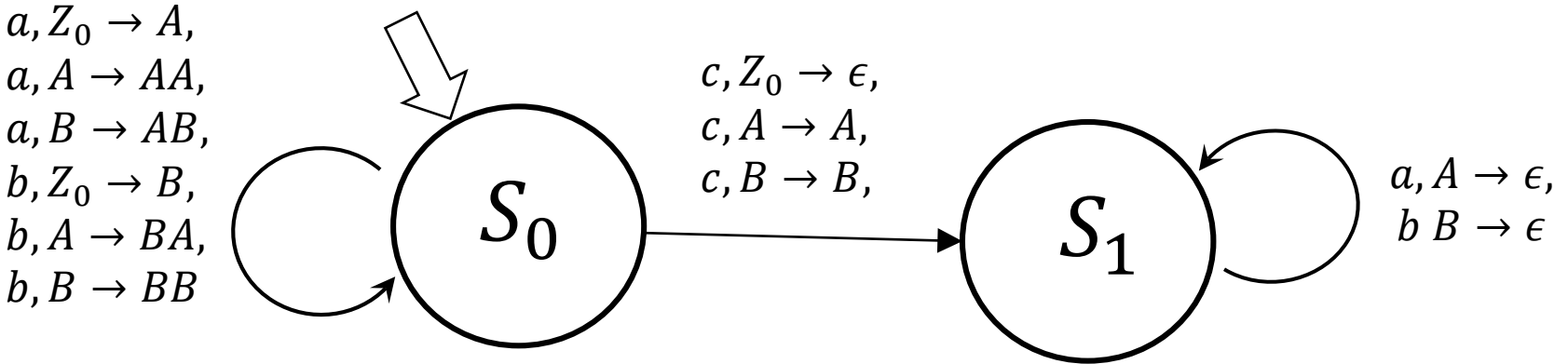
要するに決定性の動作でOK!

1.プッシュダウンオートマトン (19')

状態遷移関数: $\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, a, A) = \{(s_0, AA)\}$
$\delta(s_0, b, Z_0) = \{(s_0, B)\}, \delta(s_0, b, A) = \{(s_0, BA)\},$
$\delta(s_0, a, B) = \{(s_0, AB)\}, \delta(s_0, b, B) = \{(s_0, BB)\}$
$\delta(s_0, c, A) = \{(s_1, A)\}, \delta(s_0, c, B) = \{(s_1, B)\}, \delta(s_0, c, Z_0) = \{(s_1, \epsilon)\}$
$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$

に対応する状態遷移図は以下になる:

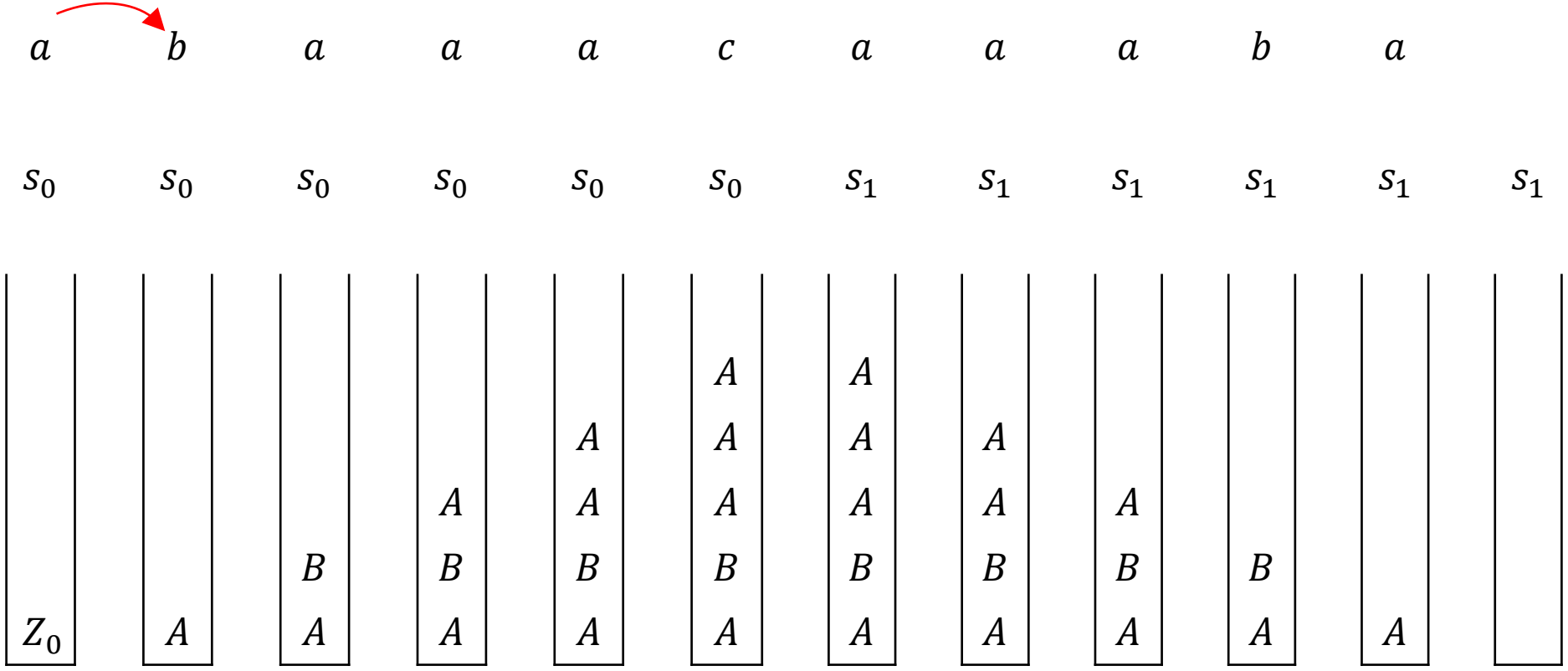
「$a, B \rightarrow AB$ 」は、「テープがaのとき、スタックのBをABにする」を意味



1.プッシュダウンオートマトン (20)

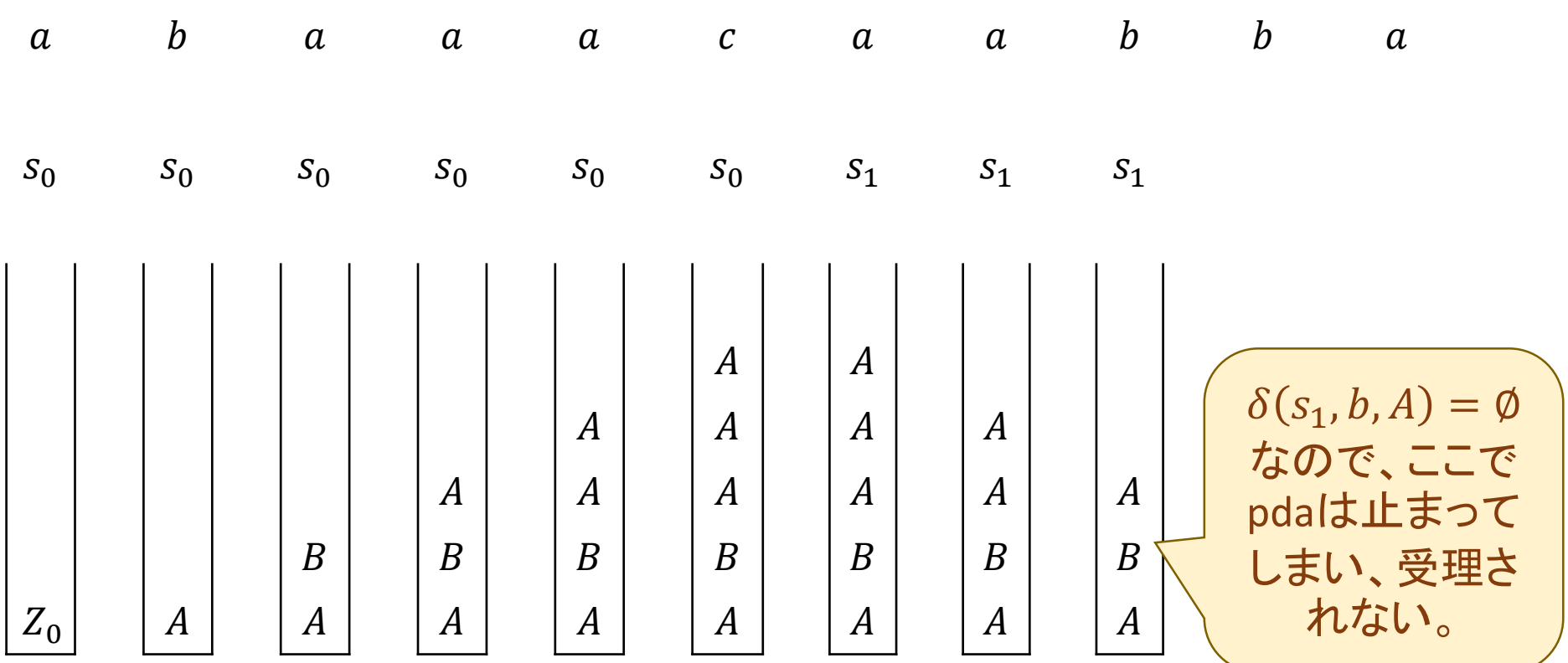
列$abaaacaaaba \in L$に対する動作例

$\delta(s_0, a, Z_0) = (s_0, A)$によって変化



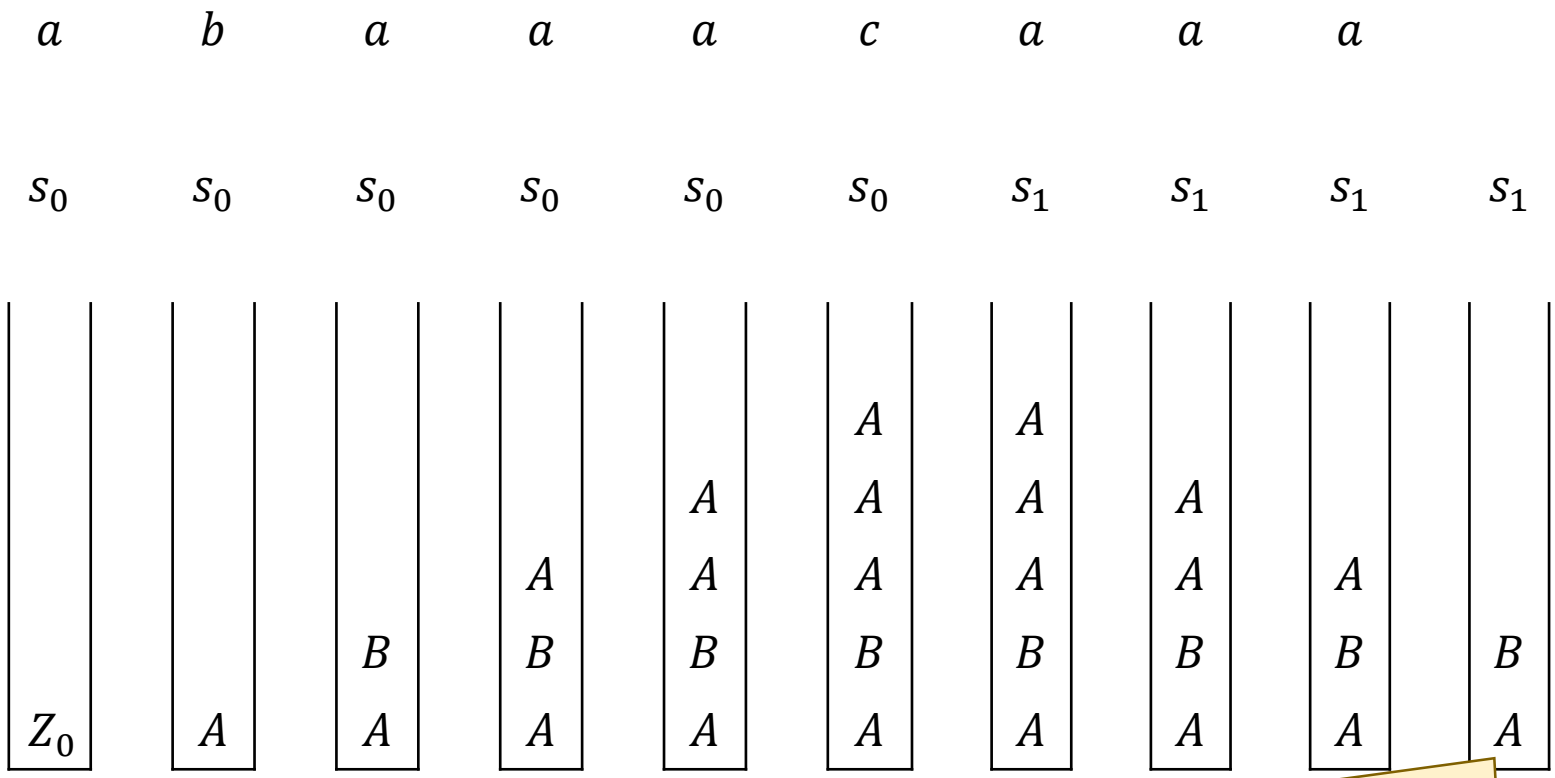
1.プッシュダウンオートマトン (21)

列$abaaacaabba \notin L$に対する動作例



1.プッシュダウンオートマトン (22)

列$abaaacaaa \notin L$に対する動作例



テープが読み終わってもスタックが空でないので、受理されない。

1.プッシュダウンオートマトン (23)

$R^+ := RR^*$

例題4.2 $L = \{xx^R \mid x \in \{a,b\}^+\}$を認識するpdaを構成せよ。

解答： 前回との違いは目印である“c”が無いこと。

- $K = \{s_0, s_1\}$ 、$\Gamma = \{A, B, Z_0\}$は同じ。
- 前半が積み上げモード、後半がチェックモードなのも同じ
- 非決定性を使って”c”が無いことをカバーする。

状態遷移関数は以下：

$$\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, b, Z_0) = \{(s_0, B)\}$$
$$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}, \delta(s_0, a, B) = \{(s_0, AB)\}$$
$$\delta(s_0, b, A) = \{(s_0, BA)\}, \delta(s_0, b, B) = \{(s_0, BB), (s_1, \epsilon)\}$$
$$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$$

積み上げモードで、入力にaaもしくはbbが来たら、そこがテープの真ん中ではないかと”Guess(推量)”する。

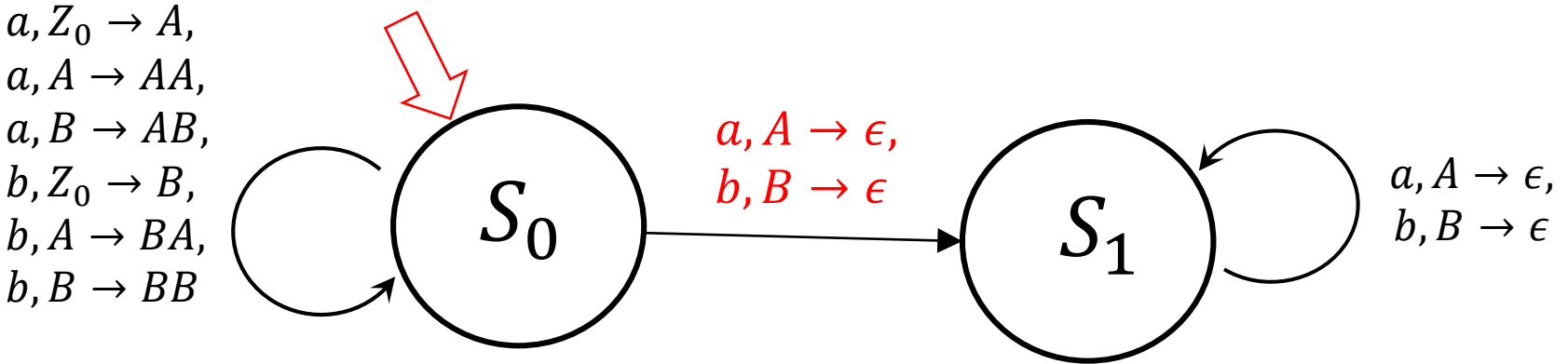
1.プッシュダウンオートマトン (23')

状態遷移関数は以下:

$$\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, b, Z_0) = \{(s_0, B)\}$$
$$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}, \delta(s_0, a, B) = \{(s_0, AB)\}$$
$$\delta(s_0, b, A) = \{(s_0, BA)\}, \delta(s_0, b, B) = \{(s_0, BB), (s_1, \epsilon)\}$$
$$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$$

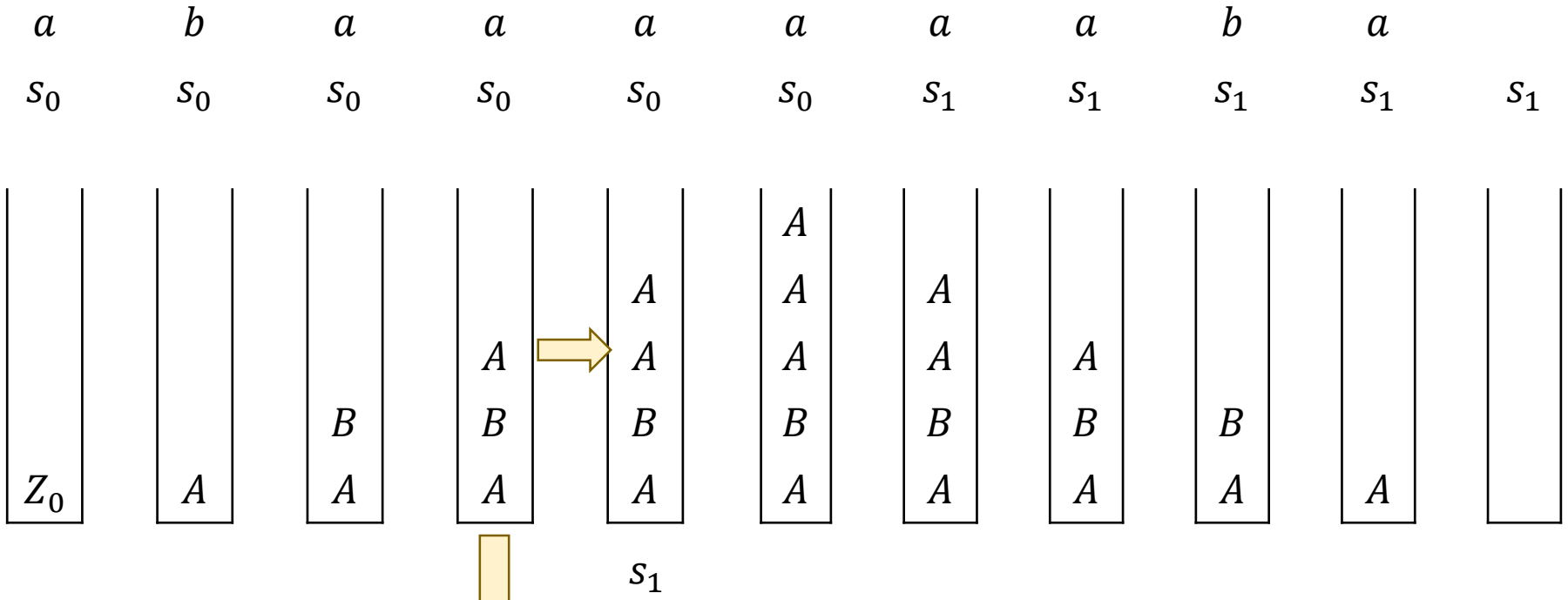
積み上げモードで、入力にaaもしくはbbが来たら、そこがテープの真ん中ではないかと"Guess(推量)"する。

状態遷移図: 真ん中の矢印の記号だけが変わった。

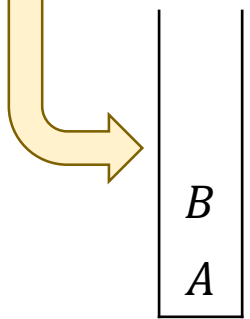


1.プッシュダウンオートマトン (24)

列$abaaaaaaba \in L$に対する動作例



チェックモードへの
切り替えのタイミングが
複数選択ができる。

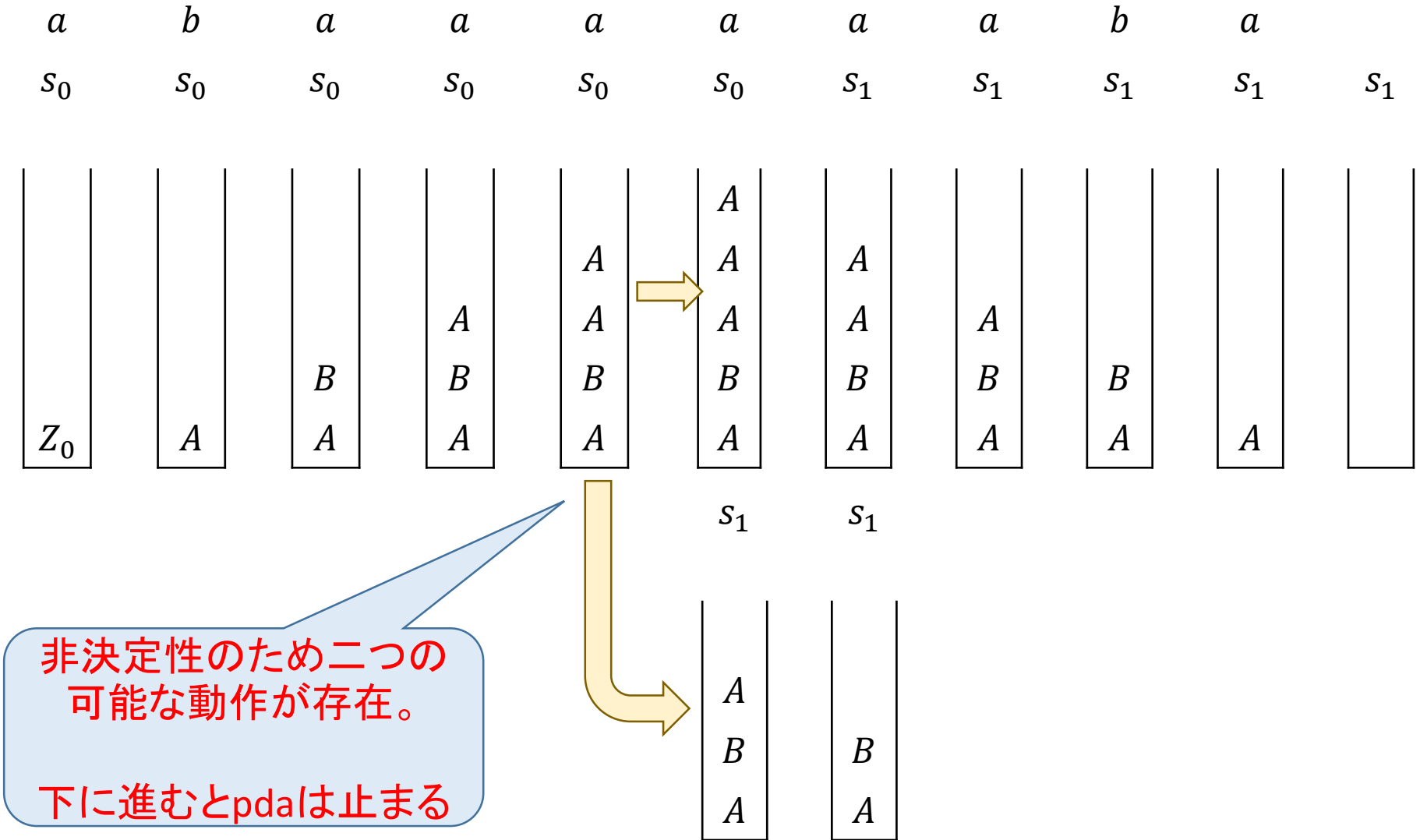


非決定性のため二つの
可能な動作が存在。

下に進むとpdaは止まる

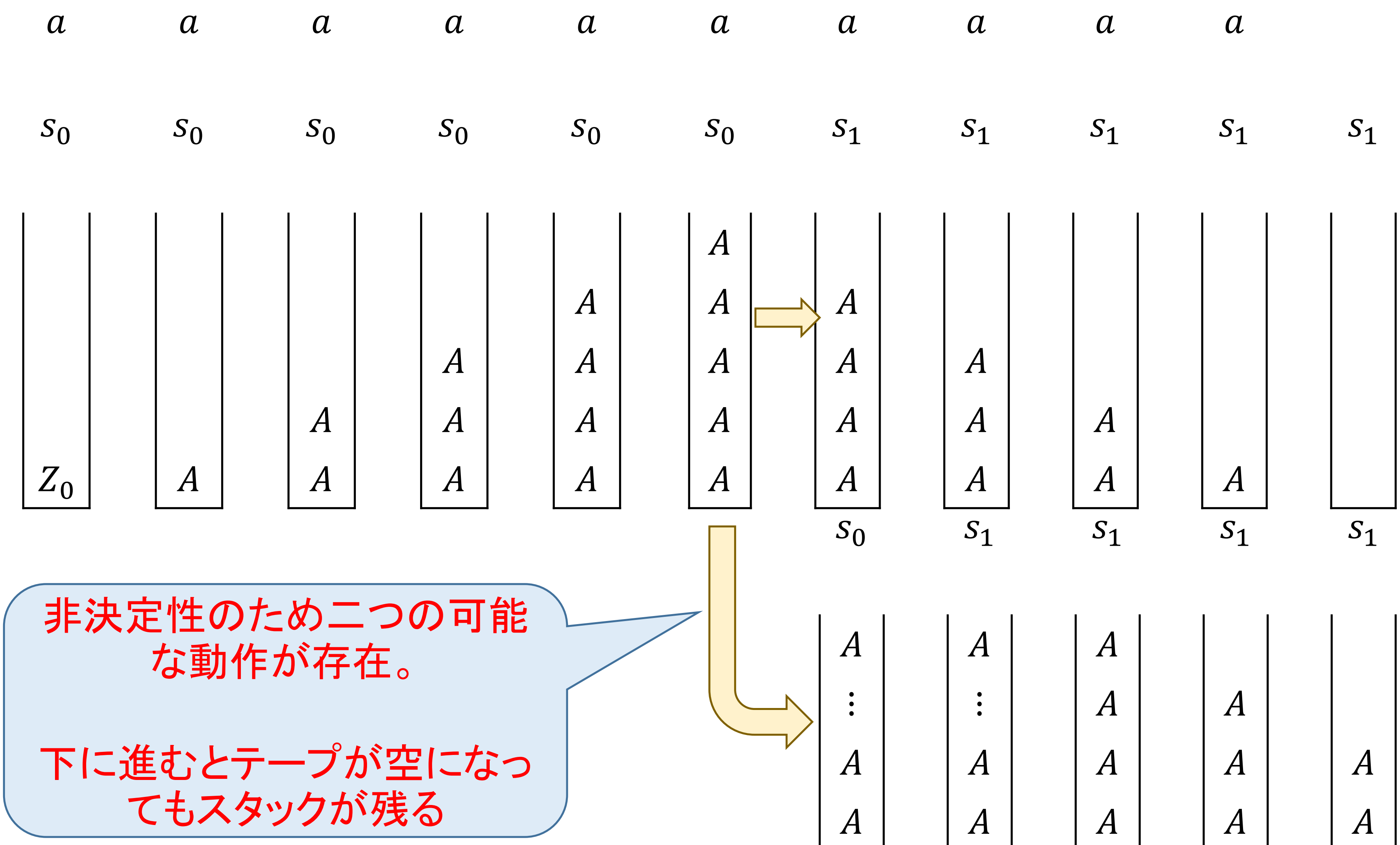
1.プッシュダウンオートマトン (25)

列$abaaaaaaba \in L$に対する動作例



1.プッシュダウンオートマトン (26)

列$aaaaaaaaaa \in L$に対する動作例



1.プッシュダウンオートマトン (27)

結局、$abaaaaaaba$の後半最初のaを読んだときに、

積み上げモードからチェックモードに切り替えないと、受理様相には到達しない。

逆にこのaで切り替えると受理様相に到達する。

$abaaaaaba$などのLに入ってない列の場合は、

どこでモードを切り替えても、決して受理様相には到達しないことが確認できる。

2.プッシュダウンオートマトンの設計(1)

pdaの状態遷移関数を書き下す説明は、簡単なpdaすらかなり複雑になる。

よって、『pdaの設計』するときは、状態遷移関数を書き下さずに、「意中のpdaの動作を上手に分かりやすく説明する」ことを目指す。

例えば、以前に説明した$L = \{xx^R \mid x \in \{a,b\}^*\}$に対するpda

$\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, b, Z_0) = \{(s_0, B)\}$

$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}, \delta(s_0, a, B) = \{(s_0, AB)\}$

$\delta(s_0, b, A) = \{(s_0, BA)\}, \delta(s_0, b, B) = \{(s_0, BB), (s_1, \epsilon)\}$

$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$

の動作を以下のように説明する ⇓

2.プッシュダウンオートマトンの設計 (2)

例えば、以前に説明した$L = \{xx^R \mid x \in \{a, b\}^+\}$に対するpdaの動作

(1) 積み上げモードの動作： 現在読んでいる記号が中央のちょうど右の記号($abaaa\underline{a}aaba$のa)かどうかをguessする。
そうでないとguessすれば、Mは読んでいる記号をスタックにプッシュし、積み上げモードを続ける。
そうであるとguessすれば、スタックの先頭記号と読んでいる入力記号が同じであるかどうかを調べ、同一ならポップし、チェックモードに移行する。もし、同一でないなら停止する。

(2) チェックモードの動作： Mは現在読んでいる記号とスタックの先頭の記号を比較し、異なれば停止。同じならポップしてチェックモードを続ける。

2.プッシュダウンオートマトンの設計 (3)

1. 積み上げモードの動作： 下内読んでいる記号が中央のちょうど右の記号($abaaa\underline{a}aaba$のa)かどうかをguessする。
   そうでないとguessすれば、Mは読んでいる記号をスタックにプッシュし、積み上げモードを続ける。
   そうであるとguessすれば、スタックの先頭記号と読んでいる入力記号が同じであるかどうかを調べ、同一ならポップし、チェックモードに移行する。もし、同一でないなら停止する。

“guess”の状態遷移関数への反映は非決定的選択を対応させる。

例えば、上の例なら、$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}$などとする。

$abaaa\underline{a}aaba$なら、1-5番目までを『中央右ではない』とguessして、6番目を中央右とguessすれば、受理様相に到達する。

『与えられた列は、受理様相に到るようなguessの系列が存在するとき、またその時に限って受理される』ようにpdaを設計する。

2.プッシュダウンオートマトンの設計 (4)

例題4.3 $L = \{0^n 1^m \mid n \neq m\}$を認識するpdaを構成せよ。

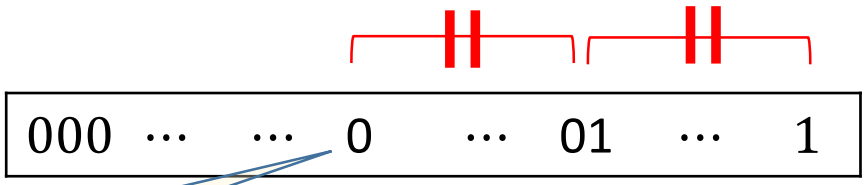
解答:

pda Mは最初に$n > m$であるか$n < m$であるかをguessする。

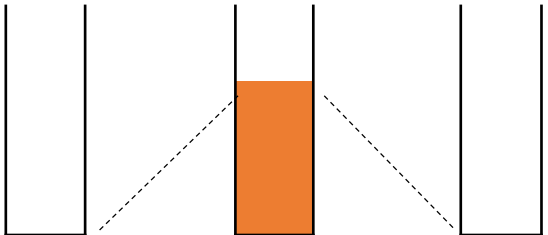
($n > m$とguessした場合)

最初の0を読み飛ばして(スタックに何もせず)、記号guessモードに入る。

<記号guessモード> 現在読んでいる0が$n - m + 1$番目の0かどうかをguessする。



この記号が左から
$n - m + 1$番目とguessする

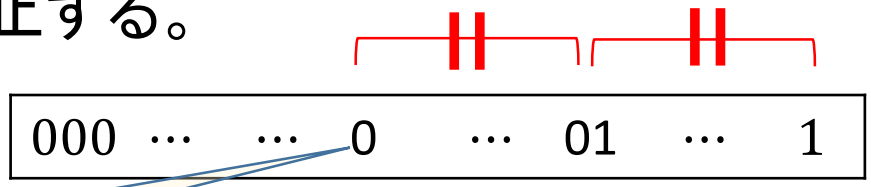


2.プッシュダウンオートマトンの設計 (5)

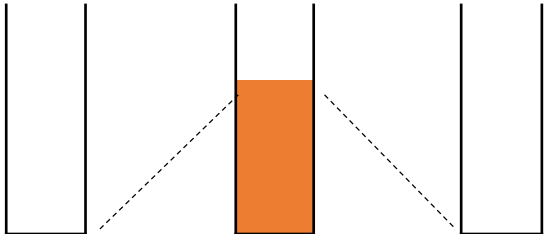
($n > m$とguessした場合)最初の0を読み飛ばして(スタックに何もせず)、記号guessモードに入る。

<記号guessモード> 現在読んでいる0が$n - m + 1$番目の0かどうかをguessする。
⇒yesとguess： チェックモードに入る
⇒noとguess: 記号guessモードを続ける。

<チェックモード> 0を読めばそれをスタックにプッシュし、1を読めばポップする。 この過程で、例えば1の後に0が来るなどが起こって列の形がおかしいとわかったら、停止する。



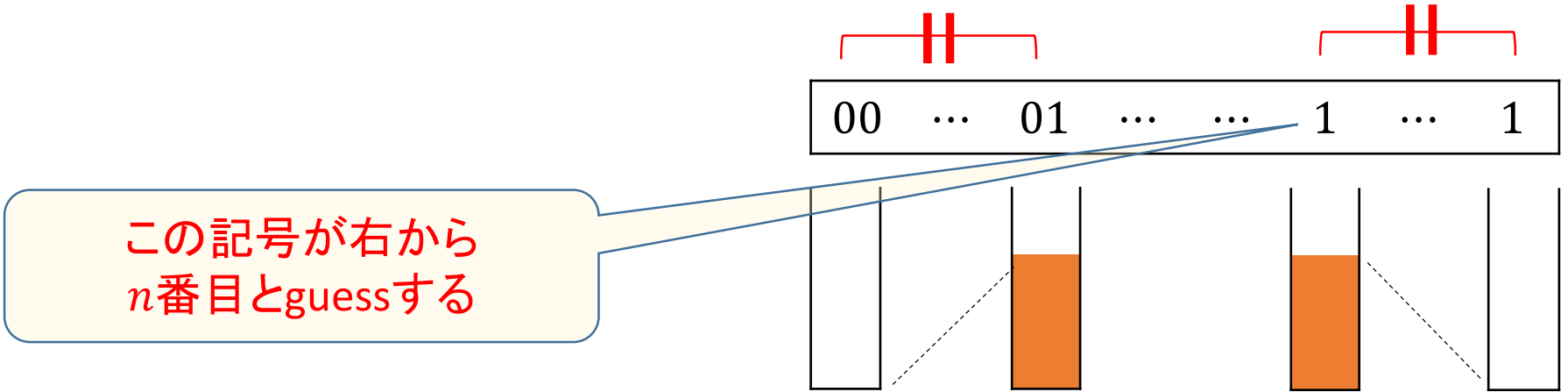
この記号が左から
$n - m + 1$番目とguessする



2.プッシュダウンオートマトンの設計 (6)

($m > n$とguessした場合)最初に積み上げモードに入って、すべての0をスタックに積み上げる。
1を読んだら、最初の1を読み飛ばして記号guessモードに入る。

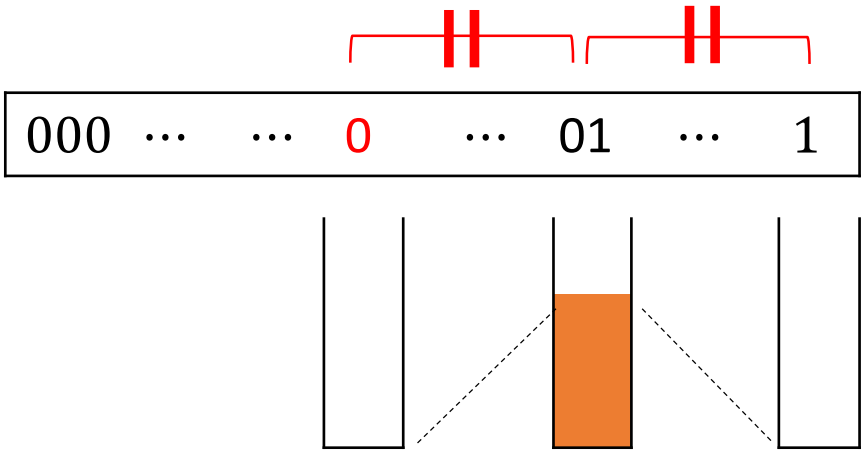
<記号guessモード> 現在読んでいる1が右からn番目の1かどうかをguessする。
⇒yesとguess:　その1からポップモードに入って、1を読むたびにスタック記号をポップする。
⇒noとguess: 記号guessモードを続ける。



2.プッシュダウンオートマトンの設計 (7)

＜正しさの検証＞

入力列$x \in L$の場合：必ず$n > m$か$m > n$なのでどちらかが当たる。
$n > m$で、かつ最初のguessが当たったとする。
すると、以下の図のような場所の0は必ず存在する。



この場所で、初めてyesとするような記号guessモードでのguessの系列が存在する。

この場合に読み終わった後で、スタックが空になっていることはほとんど明らかである。

$m > n$の場合も同様に正しさを示せる。

2.プッシュダウンオートマトンの設計 (9)

入力列$x \notin L$の場合:

列の形が0^n1^mでない場合:
明らかに途中で停止して受理しない。

(0^n1^mの形からずれると、状態遷移関数の行先が無くなる)

0^n1^mで$n = m$の場合:
最初に$n > m$とguessすると、最初に0を読み飛ばしているので、どこでチェックモードに入っても、スタックに積める0の数は、入力列の1の数より少ない。よって、1を読み終わる前にスタックが空になって受理されない。

$m > n$とguessしたときは、0の後の最初の1を読み飛ばしているので、やはり同様の議論により受理されないことが分かる。

2.プッシュダウンオートマトンの設計 (10)

pdaで有限オートマトン(dfa)を模倣する方法:

pdaはスタック付きのdfaなので、一見、そのままスタックを使わずに模倣できそうだが、問題が1つある。

問題:

pdaは、ちょうど入力を読み終わったときにスタックを空にしないと受理されない。

しかし、いま読んでいる記号がテープの右端かどうかは一般的にはわからない。

2.プッシュダウンオートマトンの設計 (10')

解決法:

dfa Mが与えられたとする、Mを模倣するpda TはMの状態遷移を、スタックに一切手を付けずに模倣しながら、

現在の記号がテープの右端かどうかをguessする。

⇒yesとguess: その記号を読んだ行先がMの受理状態なら、スタックからZ_0をポップする。

⇒noとguess: そのまま模倣を続ける。

2.プッシュダウンオートマトンの設計 (11)

例題4.4 $L = \{x \mid x \in \{a, b\}^*$ かつ $x = yy$と書けない$\}$を認識するpdaを構成せよ。

解答

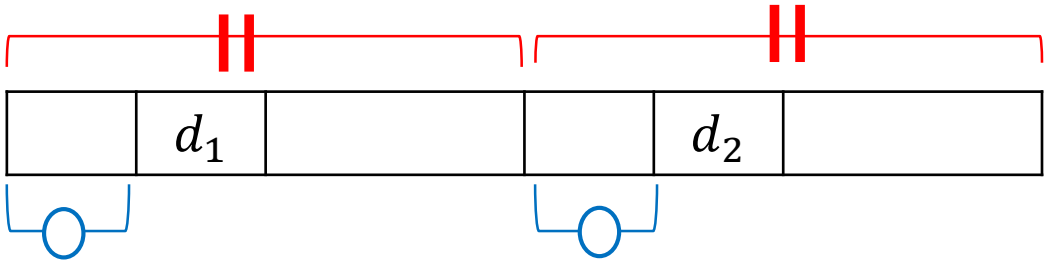
言語に入っている列xは、以下の(i)か(ii)の何れかを満たす:

(i) 長さが奇数

(ii) 長さが偶数で、

$$x = a_1 a_2 \cdots a_n a_{n+1} \cdots a_{2n}$$

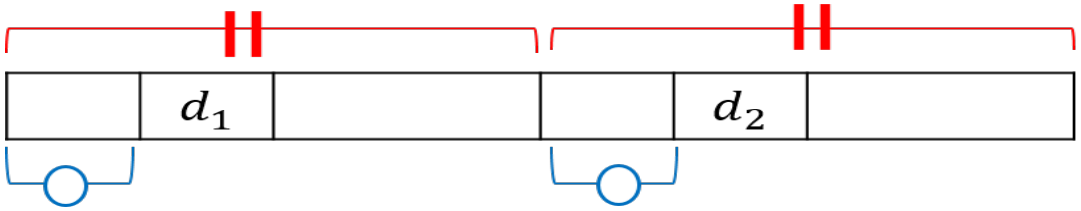
とすると、あるiに対して$a_i \neq a_{n+i}$



このように異なった2個の記号をd_1とd_2とする。

$$d_1 \neq d_2$$

2.プッシュダウンオートマトンの設計 (12)



(i) 長さが奇数

(ii) 長さが偶数で、

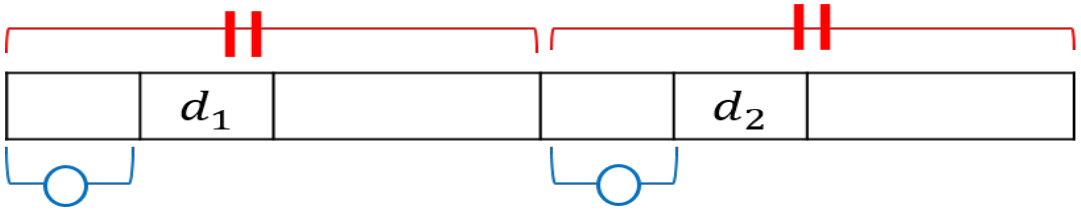
$$x = a_1 a_2 \cdots a_n a_{n+1} \cdots a_{2n}$$

とすると、あるiに対して$a_i \neq a_{n+i}$

間違った構成:

(1) 最初に(i)または(ii)のいずれが満たされるのかguessする。
(i)とguessした場合は、その後はチェックモードに入り、dfaの模倣により長さが奇数かどうかをチェックする。奇数なら受理する。

2.プッシュダウンオートマトンの設計 (12')



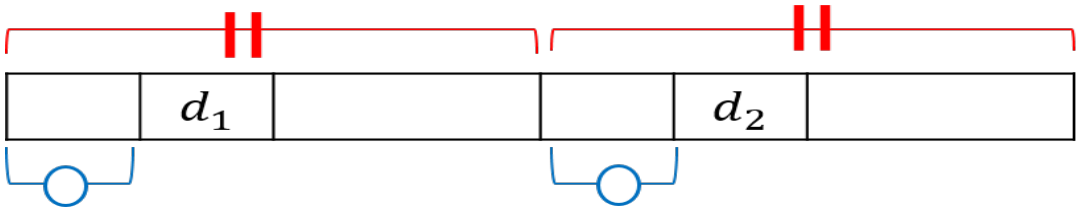
(2) (ii)とguessした場合

① まず、記号d_1をguessする。
テープの最初からguessしたd_1までの距離(記号数)をスタックに蓄える。d_1がaかbかを状態を使って記憶する。
($s_1 \rightarrow (s_1, a)$のように状態を変化させるとよい。)

② 次に、ヘッドを右に動かしながら、スタックはそのままで、テープの中央をguessする。
guessされた中央からは、スタックを1個ずつポップしていくとd_2の場所が分かる。
d_2が記憶しておいたd_1と等しいかチェック
⇒no. 残りの記号を読み飛ばして受理。
⇒yes. その場で停止して受理しない。

2.プッシュダウンオートマトンの設計 (13)

間違った構成:



＜d_2の決め方の詳細＞

guessしたd_1が先頭から5番目だったとする。

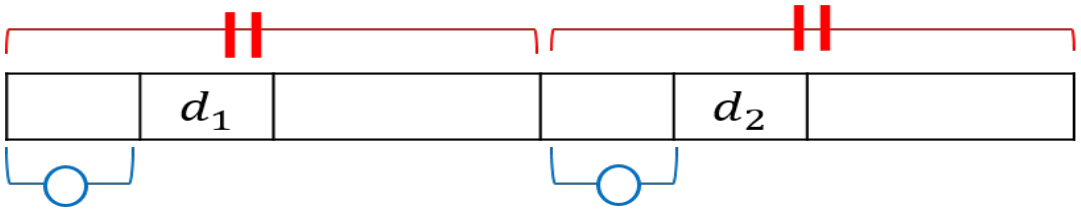
d_1より左の4個の記号については、1個読むたびAをプッシュする。

d_1を読み終わった段階で、スタックは$AAAAZ_0$となっている。

テープが(guessした)中央を超えた後は、1個の入力記号を読むたびAを1個ポップする。Z_0がスタックの先頭に現れたときのヘッドの下の記号がd_2である。

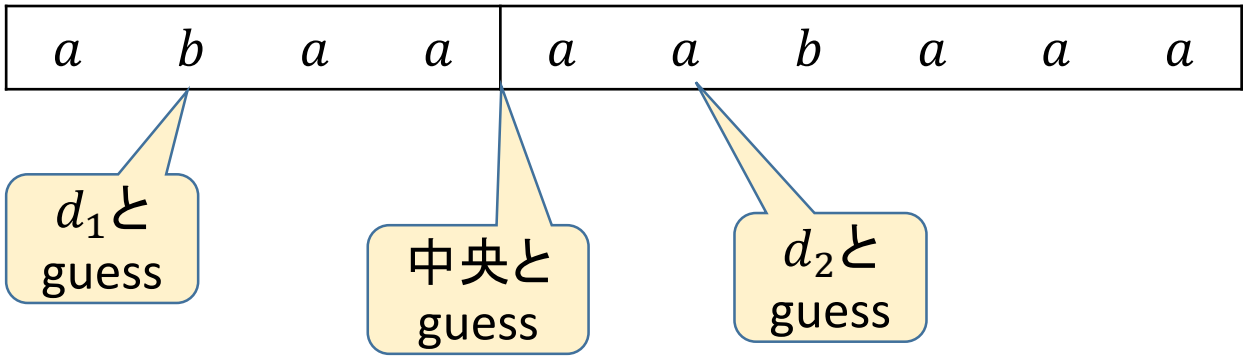
2.プッシュダウンオートマトンの設計 (14)

間違った構成:



（間違っている理由）

以上の構成では、$abaaaabaaa$を受理してしまう。



失敗の原因:
全てのguessが成功したときのみ受理され、1つでもguessが外れれば不受理にならなければならない。

『中央とguess』が外れたことを、この構成ではチェックできない。

2.プッシュダウンオートマトンの設計 (15)

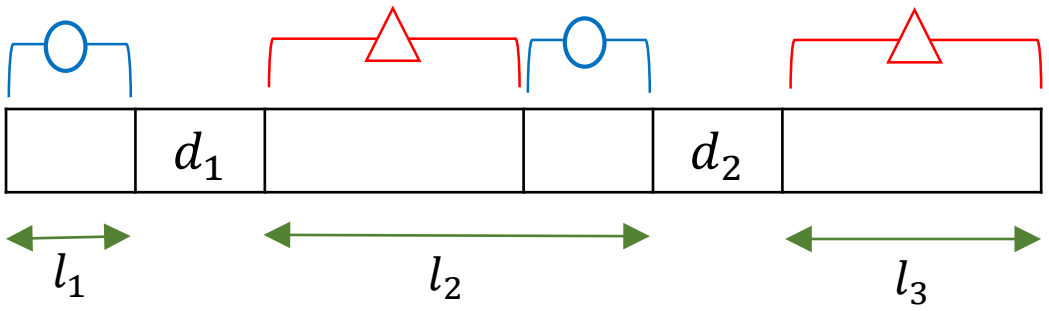
中央をguessできたことをチェックすることは難しい。

⇒実は、中央のguessは必要ない。

記号d_1とd_2が正しい位置にあるための必要十分条件は、d_1の前、d_1とd_2の間、d_2のあとの記号をそれぞれl_1, l_2, l_3とすると、

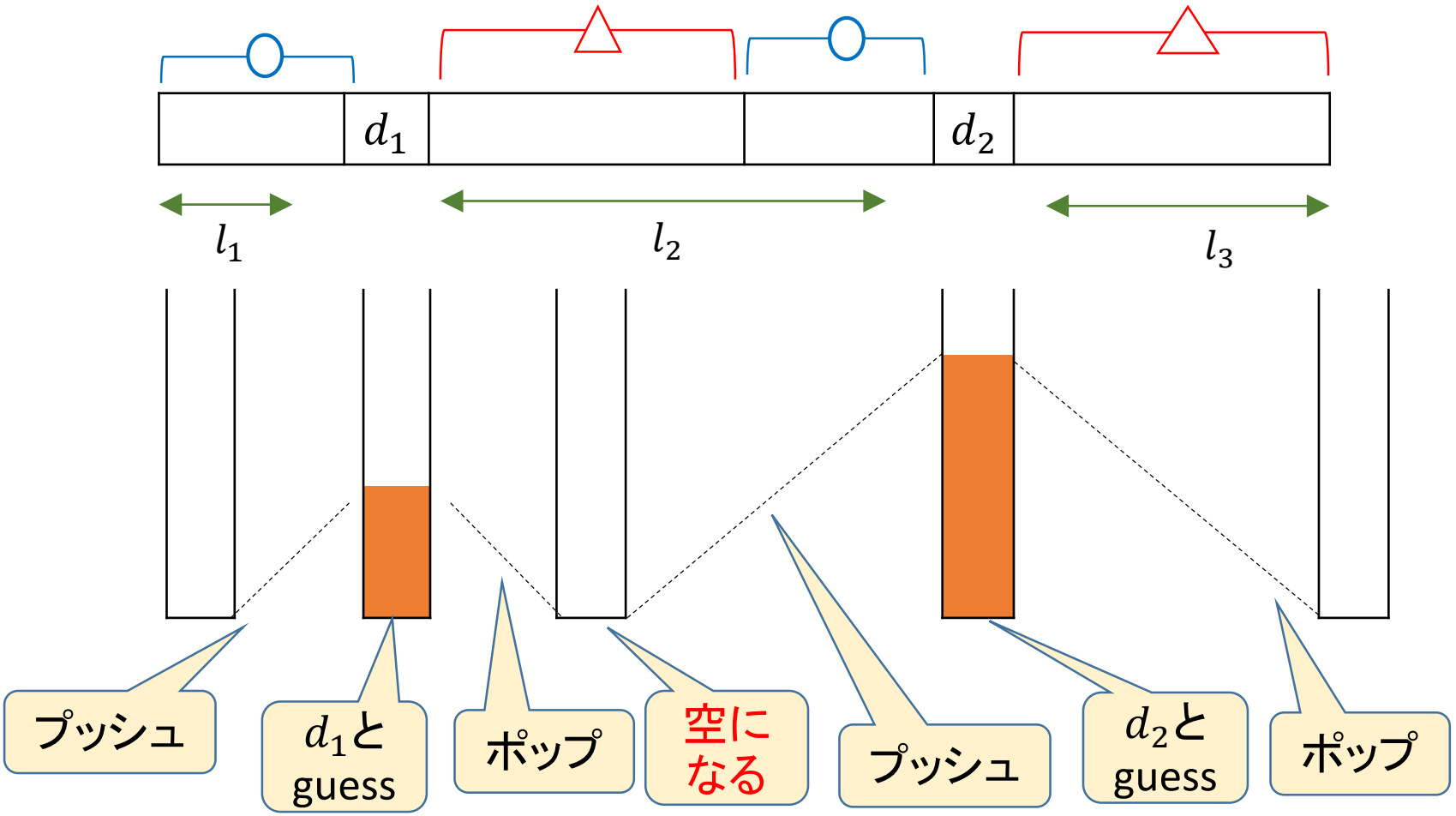
$$l_2 = l_1 + l_3$$

が満たされることである。



2.プッシュダウンオートマトンの設計 (16)

結局、(ii)とguessした場合は以下のようにすればよい。



2.プッシュダウンオートマトンの設計 (17)

正しい構成: (1)は以前の構成のまま、(2)を以下で置き換える。

(ii)とguessした場合。現在の記号がd_1かどうかをguessする。

⇒noとguess. 1個の記号をプッシュして次に進む。

⇒yesとguess. その記号がaかbかを有限状態を使って記憶し、$d_2$guessモードに入る。

<$d_2$guessモード> スタックの記号を1個ずつポップしながら入力記号を読み飛ばす。スタックが空になったら(本当に空になると停止するので、スタック初期記号Z_0を底を示す記号として利用する)、記号を1個ずつプッシュしながらd_2をguessする。

⇒ noとguess. $d_2$guessモードを続ける。

⇒yesとguess. 記憶しているd_1と異なるか確かめる。もし同じなら停止、異なっていれば、排出モードに入る。

<排出モード>記号を1個読むごとにスタック記号を1個ポップする。

文脈自由文法との等価性(1)

pdaの重要性は、文脈自由文法との等価性にある。

等価性の鍵となるのは、下記の定理:

定理4.1　与えられたpdaに対して、同じ言語を認識する状態数1のpdaが構成できる。

要するに、有限制動部の状態は実はpdaには必要ない。
異なる状態は、スタックをうまく使うことで『模倣』できる。

一般的な、証明は難しいので具体例で見てみる。

文脈自由文法との等価性(2)

以前、例4.2で取り上げた $L = \{xx^R \mid x \in \{a,b\}^+\}$ で考える:

積み上げモード(状態s_0):

$$\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, b, Z_0) = \{(s_0, B)\}$$
$$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}, \delta(s_0, a, B) = \{(s_0, AB)\}$$
$$\delta(s_0, b, A) = \{(s_0, BA)\}, \delta(s_0, b, B) = \{(s_0, BB), (s_1, \epsilon)\}$$

チェックモード(状態s_1):

$$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$$

スタックの先頭記号を読むことで、今の状態を認識できるようにする。
積み上げモードでは、先頭の記号を$A\ or\ B$から$A'\ or\ B'$にする。
2番目以降のスタックの記号はそのまま、$A\ or\ B$とする。
すると、先頭の記号をポップすることがモードの変更になる。

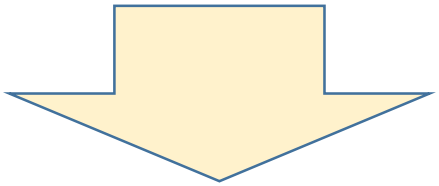
文脈自由文法との等価性(3)

積み上げモード（状態s_0）:

$$\delta(s_0, a, Z_0) = \{(s_0, A)\}, \delta(s_0, b, Z_0) = \{(s_0, B)\}$$
$$\delta(s_0, a, A) = \{(s_0, AA), (s_1, \epsilon)\}, \delta(s_0, a, B) = \{(s_0, AB)\}$$
$$\delta(s_0, b, A) = \{(s_0, BA)\}, \delta(s_0, b, B) = \{(s_0, BB), (s_1, \epsilon)\}$$

チェックモード（状態s_1）:

$$\delta(s_1, a, A) = \{(s_1, \epsilon)\}, \delta(s_1, b, B) = \{(s_1, \epsilon)\}$$



積み上げモード（スタックの先頭が、Z_0 or A' or B'）:

$$\delta(s_0, a, Z_0) = \{(s_0, A')\}, \delta(s_0, b, Z_0) = \{(s_0, B')\}$$
$$\delta(s_0, a, A') = \{(s_0, A'A), (s_0, \epsilon)\}, \delta(s_0, a, B') = \{(s_0, A'B)\}$$
$$\delta(s_0, b, A') = \{(s_0, B'A)\}, \delta(s_0, b, B') = \{(s_0, B'B), (s_0, \epsilon)\}$$

チェックモード（スタックの先頭が、A or B ）:

$$\delta(s_0, a, A) = \{(s_0, \epsilon)\}, \delta(s_0, b, B) = \{(s_0, \epsilon)\}$$

文脈自由文法との等価性(4)

列$abaaaaaaba \in L$に対する動作例

文脈自由文法との等価性(5)

状態が1つになれば文脈自由文法への変換は容易である:

$\delta(s_0, a, Z_0) = \{(s_0, A')\}$ を $Z_0 \to aA'$、
$\delta(s_0, a, A') = \{(s_0, A'A), (s_0, \epsilon)\}$ を $A' \to aA'A$, と$A' \to a$
のように変換していく。すると、

$$\delta(s_0, a, Z_0) = \{(s_0, A')\}, \delta(s_0, b, Z_0) = \{(s_0, B')\}$$
$$\delta(s_0, a, A') = \{(s_0, A'A), (s_0, \epsilon)\}, \delta(s_0, a, B') = \{(s_0, A'B)\}$$
$$\delta(s_0, b, A') = \{(s_0, B'A)\}, \delta(s_0, b, B') = \{(s_0, B'B), (s_0, \epsilon)\}$$

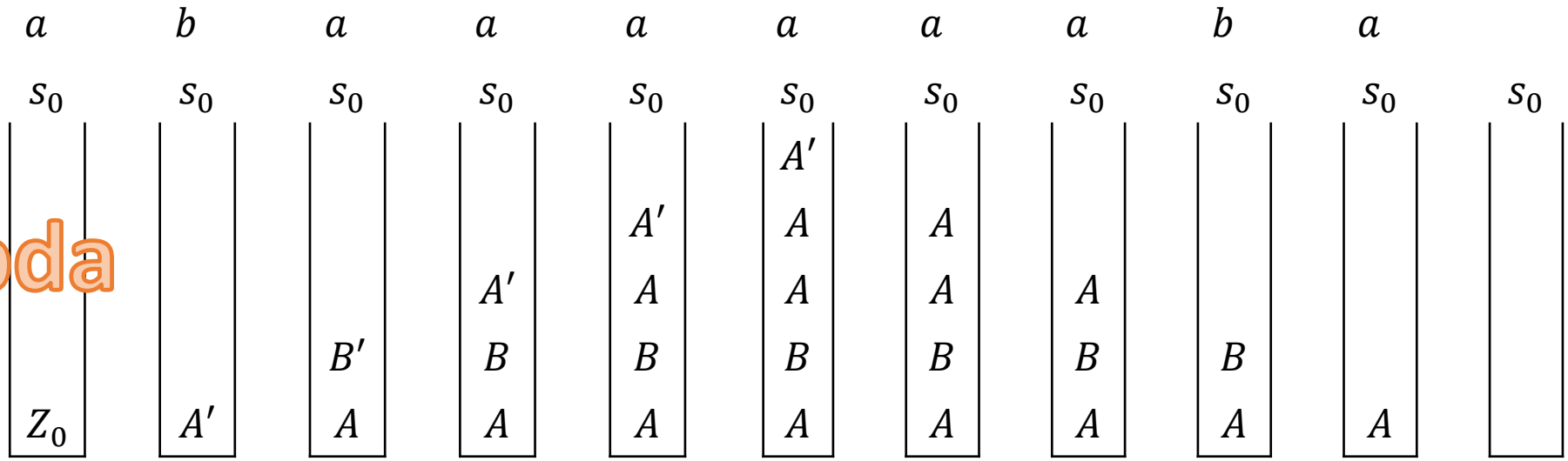
$$\delta(s_0, a, A) = \{(s_0, \epsilon)\}, \delta(s_0, b, B) = \{(s_0, \epsilon)\}$$

より、以下のグライバッハの標準形を得る:

$$Z_0 \to aA', \quad Z_0 \to bB'$$
$$A' \to aA'A, A' \to a, \quad B' \to aA'B$$
$$A' \to bB'A, \quad B' \to bB'B, B' \to b$$
$$A \to a, \quad B \to b$$

列$abaaaaaaba \in L$に対する動作例

pda



cfg

																					a
																				b	b
																			a	a	a
																		b	a	a	a
																a	a	a	a	a	a
														b	a	A'	a	a	a	a	a
						a	a	A'	A	A	a	a	a	a	a	a	a	a	a	a	a
						b	A'	A	A	A	A	a	a	a	a	a	a	a	a	a	a
			a		a	B'	B	B	B	B	B	B	b	b	b	b	b	b	b	b	b
Z_0	$\Rightarrow$	A'	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	A	$\Rightarrow$	a

文脈自由文法との等価性(7)

定理4.1　与えられたpdaに対して、同じ言語を認識する状態数1のpdaが構成できる。

一般の場合は、実はもっと複雑な方法を使わないと、

複数の状態を持つpdaを状態1つのみのpdaに書き直すことはできない。（参考書 page88-90を参照）

この定理を用いるとpdaとcfgの等価性を示せる。

定理4.2 文脈自由文法とプッシュダウンオートマトンは等価である。

文脈自由文法との等価性(8)

（証明の概要）

<pdaからcfgを構成する方法>

まず与えられたpdaを1状態のpdaに直す。

その後は、先ほどの練習問題と同様に、

$(s, \alpha) \in \delta(s, a, A) \quad \Leftrightarrow \quad A \to a\alpha$ ---(∗)

を対応させる形でグライバッハの標準形の生成規則を導く。

すると、pdaを最左導出で模倣できることが分かる。

<cfgからpdaを構成する方法>

まず与えられたcfgをグライバッハの標準形に直す。

次に、上記(∗)の対応を用いて状態遷移関数を書き下す。

すると、最左導出がこの状態遷移関数で模倣できることが分かる。