Pumping Lemma（注入補題・反復補題）

正規言語 $A \subset \Sigma^*$ に対し、

$\exists n \in \mathbb{N}:$

$\forall w \in A, |w| \geq n:$

$\exists x, y, z \in \Sigma^* : w = xyz$

(1) $y \neq \varepsilon$
(2) $|xy| \leq n$
(3) $\forall k \geq 0 : xy^k z \in A$

Pumping Lemma（注入補題・反復補題）

対偶を取ると、

言語 $A \subset \Sigma^*$ **に対し、**

次が成り立つなら、A **は正規ではない：**

$$
\boxed{
\begin{array}{l}
\forall n \in \mathbb{N} : \\
\exists w \in A, |w| \geq n : \\
\forall x, y, z \in \Sigma^* : \boxed{
\begin{array}{l}
\quad \textbf{(1)}\ \ w = xyz \\
\quad \textbf{(2)}\ \ y \neq \varepsilon \\
\quad \textbf{(3)}\ \ |xy| \leq n \\
\Longrightarrow \exists k \geq 0 : xy^k z \notin A
\end{array}
}
\end{array}
}
$$

演習問題 3：

$\Sigma = \{a, b\}$ とする。

次で定まる言語を何らかの方法で書き表した上で、有限オートマトンで認識できるかできないかを答えよ。認識できるなら（決定性または非決定性の）有限オートマトンを構成し、認識できないならそのことを（pumping lemma を用いて）示せ。

(1) a と b との個数が同じ
(2) 同じ文字が隣り合わない
(3) 同じ文字列 2 回の繰返しから成る

有限オートマトンで認識できない言語が存在する

$\Downarrow$

より強力な計算モデルが必要

$\Downarrow$

- プッシュダウンオートマトン

- チューリングマシン

有限オートマトンで認識できない言語が存在する

$\Downarrow$

有限オートマトンより強力な計算モデル

$\Downarrow$

正規言語より広い範囲の言語を扱う

$\Downarrow$

生成規則による言語の記述（生成文法）

例：“文法に適っている” 数式とは
どのようなものか？

簡単のため二項演算子のみ考えることにすれば、

- 単独の文字（変数名）は式
- 式と式とを演算子で繋いだものは式
- 式を括弧で括ったものは式
- それだけ

$\longrightarrow$ これは式を作り出す規則とも考えられる

“文法に適っている” 数式

初期記号（開始変数）$\mathcal{E}$ から出発して、
次の規則のいづれかを
“非決定的に” 適用して得られるもののみ

- $\mathcal{E} \to \mathcal{A}$
- $\mathcal{E} \to \mathcal{E}\mathcal{B}\mathcal{E}$
- $\mathcal{E} \to (\mathcal{E})$
- $\mathcal{A} \to$ **変数名のどれか**
- $\mathcal{B} \to$ **演算子のどれか**
- **変数名・演算子・$($・$)$ は それ以上書換えない（終端記号）**

$\longrightarrow$ **生成規則（書換規則）**

生成規則・生成文法

生成規則を与えることでも
言語を定めることが出来る

$\longrightarrow$ **生成文法 (generative grammar)**

生成規則による “文法に適っている” 語の生成

- 初期変数を書く
- 今ある文字列中の或る変数を
  生成規則のどれかで書換える
- 変数がなくなったら終わり

例：$\{a^{2n}b^{2m+1} | n, m \geq 0\}$

（a **が偶数個（**0 **個も可）続いた後に、**
b が奇数個続く）

正規表現で表すと、$(aa)^*b(bb)^*$

- $S \to aaS$
- $S \to bB$
- $B \to bbB$
- $B \to \varepsilon$

まとめて次のようにも書く

- $S \to aaS | bB$
- $B \to bbB | \varepsilon$

生成規則・生成文法

実際の（自然言語を含めた）“文法” では、
或る特定の状況で現われた場合だけ
適用できる規則もあるだろう

そのような生成規則は例えば次の形：

- $uAv \to uwv$

$u, v \in \Sigma^*$：**文脈 (context)**

変数 A が uAv の形で現われたら、
語 $w \in \Sigma^*$ で書換えることが出来る

生成文法の形式的定義

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

- V：**有限集合（変数の集合）**
- Σ：**有限集合（終端記号の集合）**
  ここに $V \cap \Sigma = \emptyset$
- R：**有限集合** $\subset (V \cup \Sigma)^* \times (V \cup \Sigma)^*$
  （規則の集合）
- $S \in V$：**開始変数**

$(v, w) \in R$ **が生成規則** $v \to w$ **を表す**

文脈自由文法 (context-free grammar)

文脈が全て空列 ε

即ち、規則が全て $A \to w$ ($A \in V$) **の形**

文脈自由文法の形式的定義

- V：**有限集合（変数の集合）**
- Σ：**有限集合（終端記号の集合）**
  ここに $V \cap \Sigma = \emptyset$
- R：**有限集合** $\subset V \times (V \cup \Sigma)^*$ **（規則の集合）**
- $S \in V$：**開始変数**

$(A, w) \in R$ **が生成規則** $A \to w$ **を表す**

例：言語 $A=\{a^n b^n | n \geq 0\}$ は
正規言語ではないが文脈自由言語である

- $S \to aSb | \varepsilon$

従って、

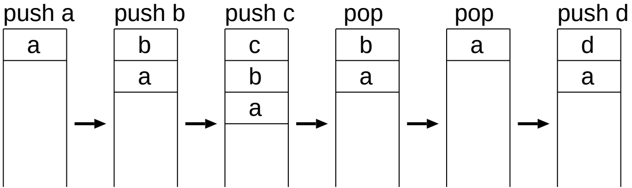
$\boxed{\text{文脈自由言語は正規言語より真に広い !!}}$

さて、正規言語を計算するモデルが
有限オートマトンであった

文脈自由言語を計算するモデル
$\cdots$ プッシュダウンオートマトン

プッシュダウンオートマトン

（非決定性）有限オートマトンに
プッシュダウンスタックを取り付けたもの



無限（非有界）の情報を保持できるが、
読み書きは先頭だけ

$\cdots$ **LIFO (Last In First Out)**

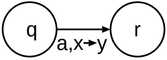
プッシュダウンオートマトンの形式的定義

$$M = (Q, \Sigma, \Gamma, \delta, s, F)$$

- Q **：有限集合** $\cdots$ **状態の集合**
- Σ **：有限集合** $\cdots$ **alphabet**
- Γ **：有限集合** $\cdots$ **stack alphabet**
  $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon := \Gamma \cup \{\varepsilon\}$ **と置く**
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$
  ：遷移関数（非決定的） $\cdots$ **可能な遷移先全体**
- $s \in Q$ $\cdots$ **初期状態**
- $F \subset Q$ $\cdots$ **受理状態の集合**

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$$

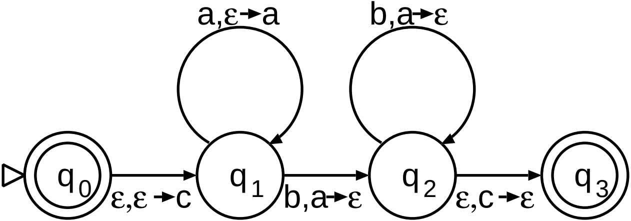
- $(r, y) \in \delta(q, a, x)$ **とは、**
  「入力 a を読んだとき、
  状態 q でスタックの先頭が x なら、
  スタックの先頭を y に書換えて、
  状態 r に移って良い」
  ということ (`pop; push y`**)**

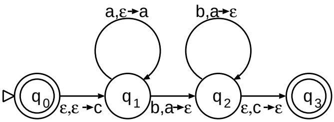
  状態遷移図では 

  で表す
- $x = y$ **は書き換え無し**
- $x = \varepsilon$ **は（スタックの先頭を見ずに）push のみ**
- $y = \varepsilon$ **は pop のみ**
- $a = \varepsilon$ **は入力を読まずに遷移**

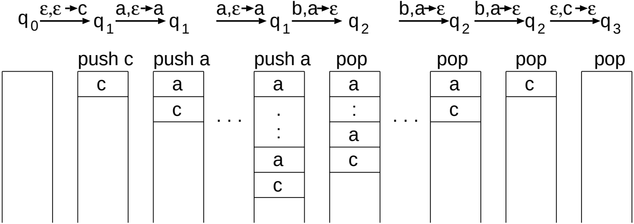
例：言語 $A = \{a^n b^n | n \geq 0\}$ を認識する
プッシュダウンオートマトン

$$\Sigma = \{a, b\}, \qquad \Gamma = \{a, b, c\}$$



PDA 

による 文字列 $a^n b^n$ の受理



スタックマシン

このように

記憶場所としてプッシュダウンスタックを備えた
計算モデルや仮想機械・処理系を

一般に**スタックマシン**という

例：

- 逆ポーランド電卓
- PostScript