

有限オートマトンでの計算可能性問題

有限オートマトンで認識できる

\iff “待ち” が有限種類

$\ell_w : \Sigma^* \longrightarrow \Sigma^* : \text{“左平行移動”}$

$$v \longmapsto wv$$

言語 $L \in \mathcal{P}(\Sigma^*)$ に対し、

$S_L : \Sigma^* \longrightarrow \mathcal{P}(\Sigma^*) : \text{“待ち” の集合}$

$$w \longmapsto \{v \in \Sigma^* \mid wv \in L\} = \ell_w^{-1}(L)$$

$$\#\text{Im}S_L < \infty \iff \exists M : L = L(M)$$

有限オートマトンでの計算可能性問題

非決定性有限オートマトンで認識できない

言語が存在する!!

(\iff 正規でない言語が存在する)

例: $A = \{a^n b^n \mid n \geq 0\}$ (a と b との個数が同じ)

実際 $w_n = a^n b$ に対する $S_L(w_n)$ が全て異なる

一般には、証明には部屋割り論法

(の一種の **pumping lemma**)

を利用することが多い

Pumping Lemma (注入補題・反復補題)

正規言語 A 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) $\forall k \geq 0 : xy^kz \in A$

有限オートマトンで認識できる / ない言語の例

$$\Sigma = \{a, b\}$$

- a と b との個数が同じ
- a が幾つか続いた後に b が幾つか続いたもの
- a で始まり a, b が交互に並んで b で終わる
- 同じ文字列 2 回の繰返しから成る
- 回文 (palindrome)

などなど

このうちで、

有限オートマトンで認識できる言語は？

有限オートマトンで認識できない言語が存在する



より強力な計算モデルが必要



- 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



有限オートマトンより強力な計算モデル



正規言語より広い範囲の言語を扱う



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

→ 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) E から出発して、
次の規則のいずれかを
“非決定的に” 適用して得られるもののみ

- $E \rightarrow A$
- $E \rightarrow EBE$
- $E \rightarrow (E)$
- $A \rightarrow$ 変数名のどれか
- $B \rightarrow$ 演算子のどれか
- 変数名・演算子・ (\cdot) は
それ以上書換えない (終端記号)

→ 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

→ 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： $\{a^{2n}b^{2m+1} \mid n, m \geq 0\}$

(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、 $(aa)^*b(bb)^*$

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

まとめて次のようにも書く

- $S \rightarrow aaS \mid bB$
- $B \rightarrow bbB \mid \varepsilon$

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：

- $uAv \rightarrow uww$

$u, v \in \Sigma^*$: 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 $v \rightarrow w$ 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 $A \rightarrow w$ ($A \in V$) 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 $A \rightarrow w$ を表す

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

- $S \rightarrow aSb \mid \varepsilon$

従って、

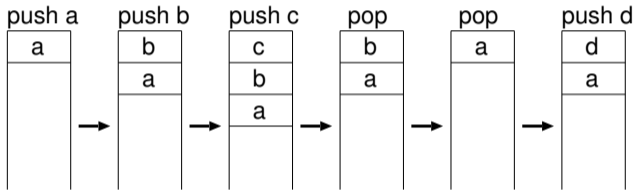
文脈自由言語は正規言語より真に広い!!

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
… 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

… 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 … 状態の集合
- Σ : 有限集合 … **alphabet**
- Γ : 有限集合 … **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) … 可能な遷移先全体
- $s \in Q$ … 初期状態
- $F \subset Q$ … 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)
- $x = y$ は書き換え無し
- $x = \varepsilon$ は **push** のみ
- $y = \varepsilon$ は **pop** のみ
- $a = \varepsilon$ は入力を読まずに遷移