

有限オートマトンでの計算可能性問題

- 言語 $A \subset \Sigma^*$ に対し、
A を認識する有限オートマトン M
が存在するか？
- 有限オートマトンによって
認識可能な言語はどのようなものか？

→ 正規言語・正規表現

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)

有限オートマトンより強力な計算モデルが必要



正規言語より広い範囲の言語を考えることが必要



生成規則による言語の記述 (生成文法)

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:

- $uAv \rightarrow uwwv$

$u, v \in \Sigma^*$: 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、

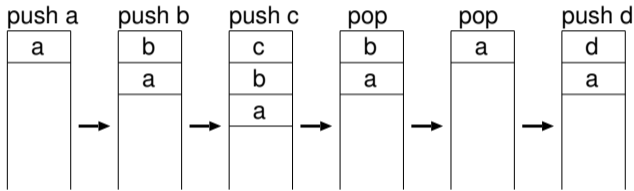
文脈自由言語は正規言語より真に広い!!

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

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

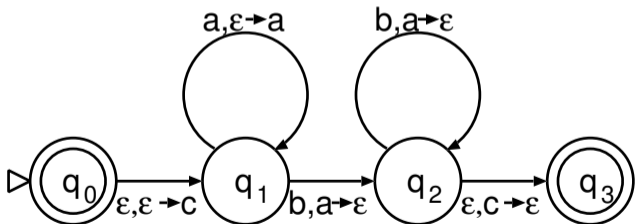
- Q : 有限集合 … 状態の集合
- Σ : 有限集合 … **alphabet**
- Γ : 有限集合 … **stack alphabet**
 $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon := \Gamma \cup \{\varepsilon\}$ と置く
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$
: 遷移関数 … 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

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

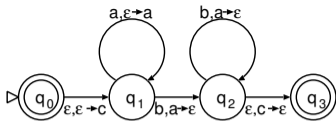
- $(r, y) \in \delta(q, a, x)$ とは、
「入力 a を読んだとき、
状態 q でスタックの先頭が x なら、
スタックの先頭を y に書換えて、
状態 r に移って良い」
ということ (pop; push y)
- $x = y$ は書き換え無し
- $x = \varepsilon$ は **push** のみ
- $y = \varepsilon$ は **pop** のみ
- $a = \varepsilon$ は入力を読まずに遷移

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

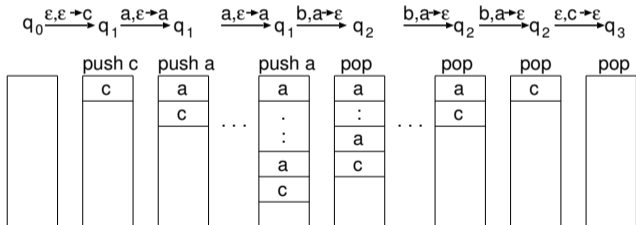
$\Sigma = \{a, b\}, \quad \Gamma = \{a, b, c\}$



PDA



による
文字列 $a^n b^n$ の受理



定理:

L : 文脈自由言語



L が或るプッシュダウンオートマトンで
認識される

文脈自由言語の例

回文全体の成す言語は文脈自由

- $S \rightarrow aSa|bSb|a|b|\epsilon$

問: 回文全体の成す言語を認識する

プッシュダウンオートマトンを構成せよ

文脈自由言語の例

回文全体の成す言語は文脈自由

- $S \rightarrow aSa | bSb | a | b | \varepsilon$

問: 回文全体の成す言語を認識する

プッシュダウンオートマトンを構成せよ

文脈自由言語の例

回文全体の成す言語は文脈自由

- $S \rightarrow aSa | bSb | a | b | \varepsilon$

問: 回文全体の成す言語を認識する

プッシュダウンオートマトンを構成せよ

プッシュダウンオートマトンでは

認識できない言語の例

同じ文字列 2 回の繰返しから成る文字列全体

$$A = \{ww \mid w \in \Sigma^*\}$$

入力を読み直せないのが弱点

→ 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では

認識できない言語の例

同じ文字列 2 回の繰返しから成る文字列全体

$$A = \{ww \mid w \in \Sigma^*\}$$

入力を読み直せないのが弱点

→ 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では

認識できない言語の例

同じ文字列 2 回の繰返しから成る文字列全体

$$A = \{ww \mid w \in \Sigma^*\}$$

入力を読み直せないのが弱点

→ より強力な計算モデルが必要

一つの方法としては、

入力を覚えておくために

プッシュダウンスタックをもう一つ

使えることにする

実際これで真により強い計算モデルが得られる

しかし、通常はこれと同等な

次のような計算モデルを考える

… チューリングマシン

一つの方法としては、

入力を覚えておくために

プッシュダウンスタックをもう一つ

使えることにする

実際これで真により強い計算モデルが得られる

しかし、通常はこれと同等な

次のような計算モデルを考える

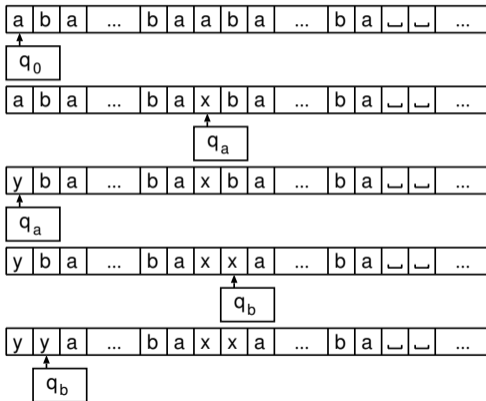
… チューリングマシン

チューリングマシン

- 有限個の内部状態を持つ
- 入力データはテープ上に一区画一文字ずつ書き込まれて与えられる
- データを読み書きするヘッドがテープ上を動く
- 遷移関数は次の形：
内部状態とヘッドが今いる場所の文字とによって、その場所の文字を書き換え、次の内部状態に移り、ヘッドを左か右かに動かす
- 受理状態または拒否状態に達したら停止するが、停止しないこともある

(非決定性) チューリングマシンによる

言語 $A = \{ww \mid w \in \Sigma^*\}$ の認識



チューリングマシンによる言語の認識

チューリングマシン T が言語 A を認識する



$$A = \left\{ w \in \Sigma^* \mid \begin{array}{l} \text{入力 } w \text{ に対し、} \\ \text{受理状態で停止する} \\ \text{遷移が存在} \end{array} \right\}$$



$w \in A \iff$ 入力 w に対し、
受理状態で停止する遷移が存在

チューリングマシンによる言語の判定

チューリングマシン T が言語 A を判定する



T は A を認識し、
かつ、全ての入力に対し必ず停止する



$w \in A \iff$ 入力 w に対し、
受理状態で停止する遷移が存在
かつ

$w \notin A \iff$ 入力 w に対し、
拒否状態で停止する遷移が存在

Church-Turing の提唱

「全てのアルゴリズム (計算手順) は、
チューリングマシンで実装できる」

(アルゴリズムと呼べるのは
チューリングマシンで実装できるものだけ)

… 「アルゴリズム」の定式化

Church-Turing の提唱

「全てのアルゴリズム (計算手順) は、
チューリングマシンで実装できる」

(アルゴリズムと呼べるのは
チューリングマシンで実装できるものだけ)

… 「アルゴリズム」の定式化

Church-Turing の提唱

「全てのアルゴリズム (計算手順) は、
チューリングマシンで実装できる」

(アルゴリズムと呼べるのは
チューリングマシンで実装できるものだけ)

… 「アルゴリズム」の定式化

何故「チューリングマシン」なのか?

- およそ計算機で実行したいことは模倣可能
(無限のメモリにランダムアクセスできる
計算機モデル)
- 多少モデルを変更しても強さが同じ
(モデルの頑強性)
 - ★ テープが両方に無限に伸びているか
 - ★ ヘッドが動かないことがあっても良いか
 - ★ 複数テープチューリングマシン
 - ★ 決定性 / 非決定性 などなど

プログラム内蔵方式 (von Neumann 型)

… プログラムもデータとして保持

→ 一つの機械で様々な計算を柔軟に実現

同様の働きをするチューリングマシン U が
構成できる

… 万能チューリングマシン
(universal Turing machine)

プログラム内蔵方式 (von Neumann 型)

… プログラムもデータとして保持

→ 一つの機械で様々な計算を柔軟に実現

同様の働きをするチューリングマシン U が
構成できる

… 万能チューリングマシン
(universal Turing machine)

プログラム内蔵方式 (von Neumann 型)

… プログラムもデータとして保持

→ 一つの機械で様々な計算を柔軟に実現

同様の働きをするチューリングマシン U が
構成できる

… 万能チューリングマシン
(universal Turing machine)

万能チューリングマシン

全てのチューリングマシンの動作を模倣する

- 入力 : $(\langle M \rangle, w)$
 - ★ $\langle M \rangle$: 機械 M の符号化 (プログラムに相当)
 - ★ w : M に与える入力データ

- 出力 : 機械 M が入力 w を受理するかどうか

定理

言語

$$A_{\text{TM}} = \left\{ (\langle M \rangle, w) \mid \begin{array}{l} \langle M \rangle : \text{TM } M \text{ の符号化} \\ M \text{ が入力 } w \text{ を受理} \end{array} \right\}$$

は認識可能だが、判定可能ではない。

証明は一種の対角線論法による
(Russell のパラドックス風)

定理

言語

$$A_{\text{TM}} = \left\{ (\langle M \rangle, w) \mid \begin{array}{l} \langle M \rangle : \text{TM } M \text{ の符号化} \\ M \text{ が入力 } w \text{ を受理} \end{array} \right\}$$

は認識可能だが、判定可能ではない。

証明は一種の対角線論法による
(Russell のパラドックス風)