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？

$\longrightarrow$ 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!!
($\Longleftrightarrow$ 正規でない言語が存在する)

有限オートマトンで認識できない言語が存在する
($\Longleftrightarrow$ 正規でない言語が存在する)

$\Downarrow$

正規言語より広い範囲の言語を考えることが必要

$\Downarrow$

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

$\longrightarrow$ **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：

- $u\mathit{A}v \to uwv$

$u, v \in \Sigma^*$: **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、

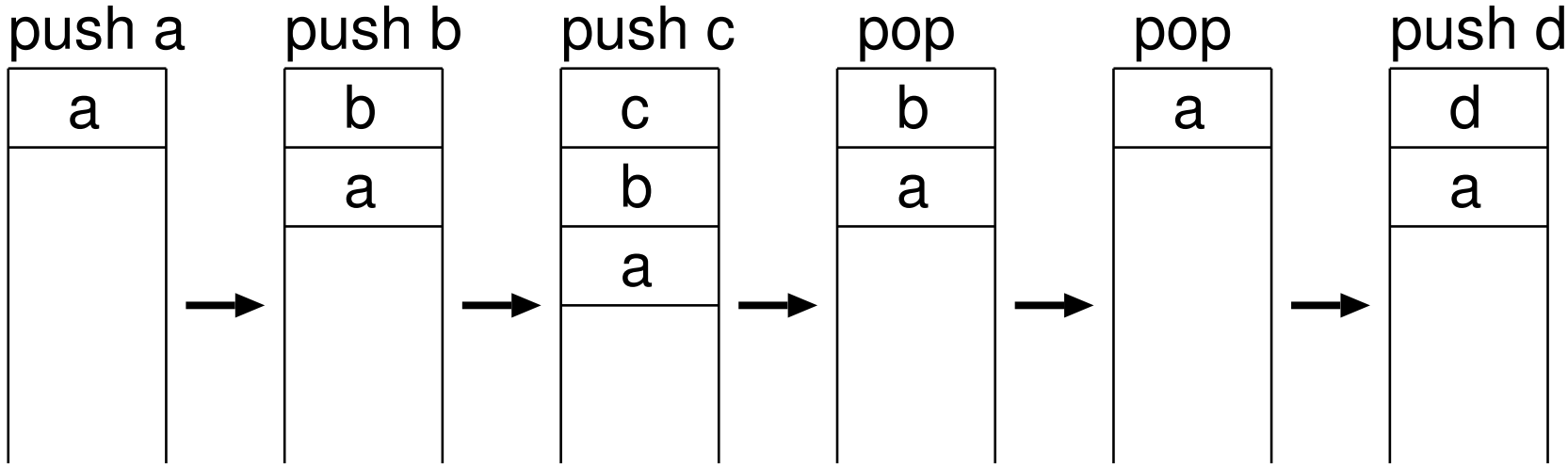
文脈自由言語は正規言語より真に広い!!

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
· · · 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**

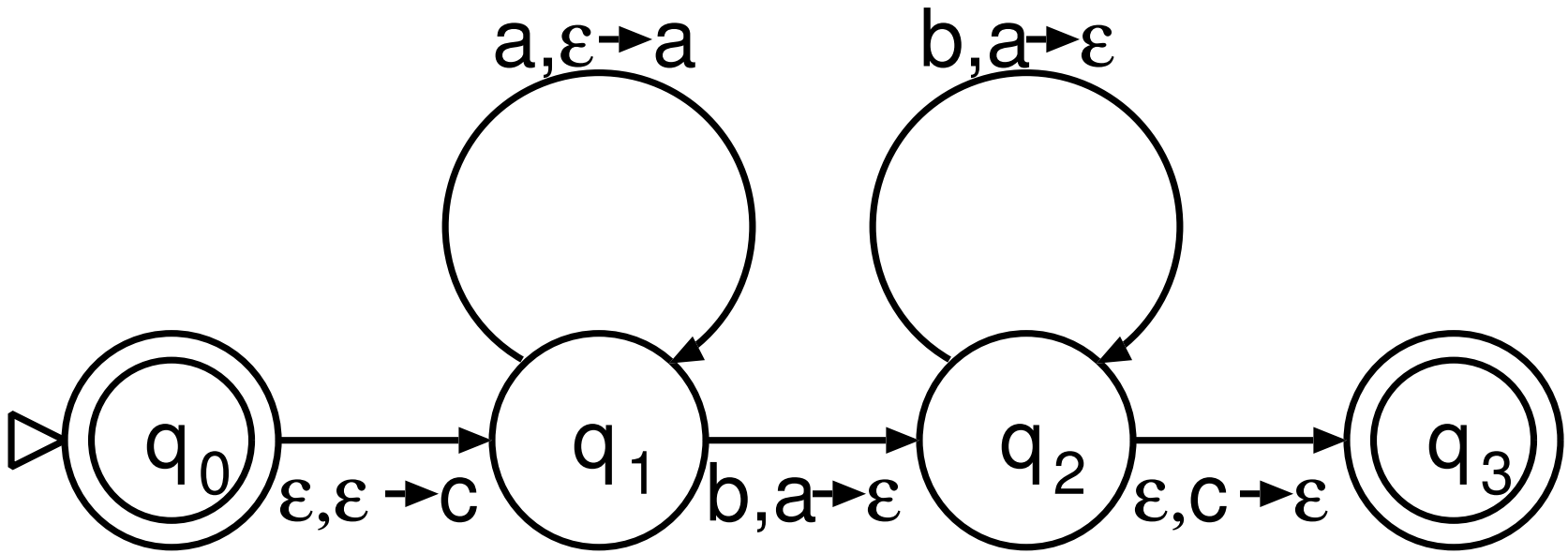
$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$$

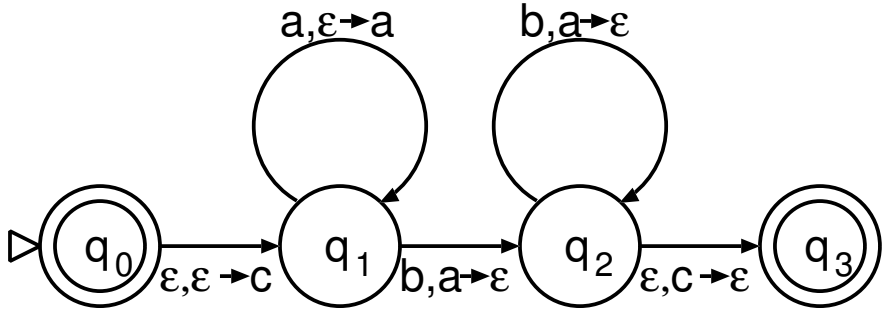
- $(r, y) \in \delta(q, a, x)$ **とは、**
  「入力 a **を読んだとき、**
  状態 q **でスタックの先頭が** x **なら、**
  スタックの先頭を y **に書換えて、**
  状態 r **に移って良い」**
  ということ (pop; push y)

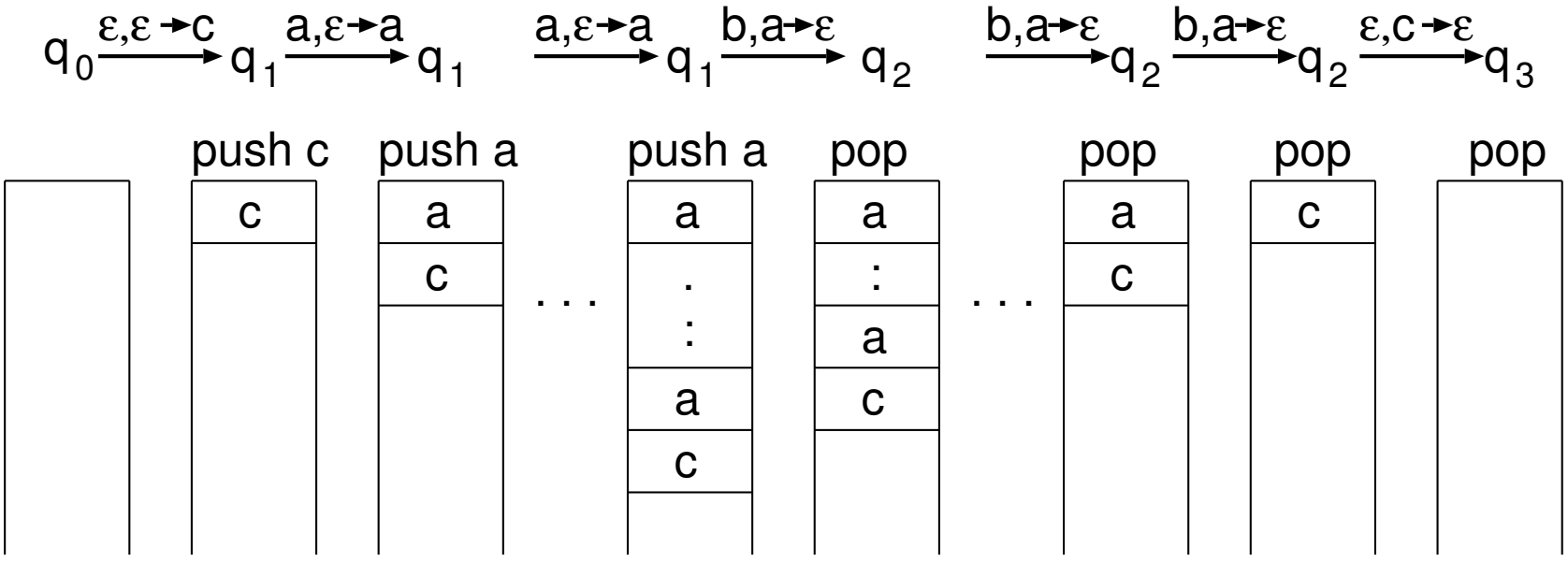
- $x = y$ **は書き換え無し**
- $x = \varepsilon$ **は push のみ**
- $y = \varepsilon$ **は pop のみ**
- $a = \varepsilon$ **は入力を読まずに遷移**

例: 言語 $A = \{a^n b^n | n \geq 0\}$ 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

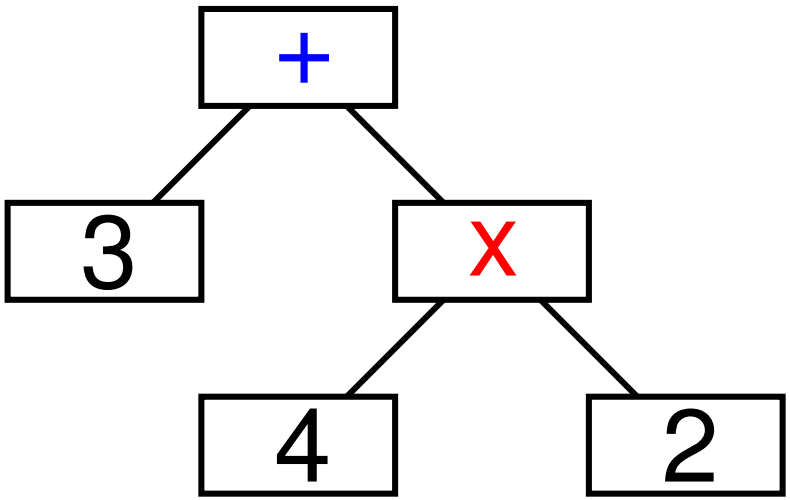
記憶場所としてプッシュダウンスタックを備えた
計算モデルや仮想機械・処理系を

一般にスタックマシンという

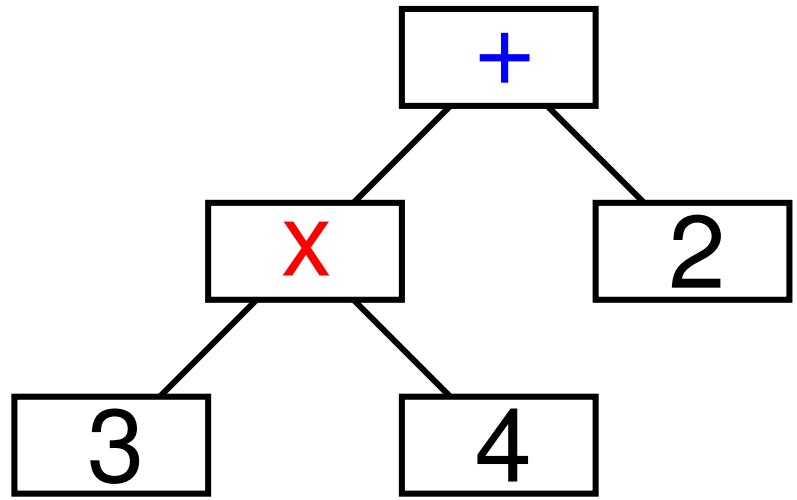
例：

- 逆ポーランド電卓
- PostScript

式と演算木



3+4x2



3x4+2

Mathematica などの
数式処理 (計算機代数) ソフトウェアでは、

通常、内部的に数式の木構造を保持

演算木の表記

演算子を置く場所により、中置・前置・後置がある

中置	前置	後置
$3 + 4 \times 2$	$+\ 3\ \times\ 4\ 2$	$3\ 4\ 2\ \times\ +$
$(3 + 4) \times 2$	$\times\ +\ 3\ 4\ 2$	$3\ 4\ +\ 2\ \times$
$3 \times 4 + 2$	$+\ \times\ 3\ 4\ 2$	$3\ 4\ \times\ 2\ +$

後置記法 (逆ポーランド記法)

後置	日本語
$3\ 4\ 2\ \times\ +$	3 に 4 に 2 を**掛**けたものを**足**したもの
$3\ 4\ +\ 2\ \times$	3 に 4 を**足**したものに 2 を**掛**けたもの
$3\ 4\ \times\ 2\ +$	3 に 4 を**掛**けたものに 2 を**足**したもの

スタックを用いた計算に便利

後置記法の演算式のスタックを用いた計算 (逆ポーランド電卓)

- 数値 $\Longrightarrow$ push
- 演算子 $\Longrightarrow$ 被演算子を (所定の個数だけ) pop
  $\longrightarrow$ 演算を施し、結果を push
- 入力終了 $\Longrightarrow$ pop
  $\longrightarrow$ スタックが丁度空になったらその値が答え

問： 後置記法 (逆ポーランド記法) の式に対し
スタックを用いて値を計算する
アルゴリズムを実装せよ

後置記法の有利性

後置記法の演算式が簡明に計算できるのは、

(各演算子に対して
被演算子の個数が決まっていれば)

括弧が必要ない (優先順位を考慮しなくてよい)

ことが大きく効いている

- **式 :: 定数 || 変数 || 式 式 二項演算子**
  (+ も × も区別なし)

中置記法と演算子の優先順位

中置記法の演算式には括弧が必要
(演算子の優先順位を定めておく必要あり)

$3 \times 4 + 2$　　$3 + 4 \times 2$

計算する際には優先順位を考慮する必要がある

- **式** :: **項** || **項** $+$ **式**
- **項** :: **因子** || **因子** $\times$ **項**
- **因子** :: **定数** || **変数** || (**式**)

($+$ と $\times$ とで純然たる区別あり)

スタックマシンの例： PostScript

ページ記述言語の一つ

- Adobe Systems が開発
- PDF (Portable Document Format) の元になった言語
- レーザプリンタなどで実装
- オープンソースなインタプリタとして Ghostscript が良く利用されている

- 図形を描いたりフォントを置いたりする
- 逆ポーランド記法

スタックマシンの例： PostScript

逆ポーランド記法

- データを push
- 命令 (演算子, operator) が
  所定数のデータ (被演算子, operand) を
  pop して処理

例： $(100, 200)$ から $(300 + 50, 400)$ へ、
引続き $(200, 600 - 50)$ へ線を引く

```
100 200 moveto
300 50 add 400 lineto
200 600 50 sub lineto
stroke
```

定理：

L：文脈自由言語

$\Leftrightarrow$

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

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

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

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

入力を読み直せないのが弱点

$\longrightarrow$ 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

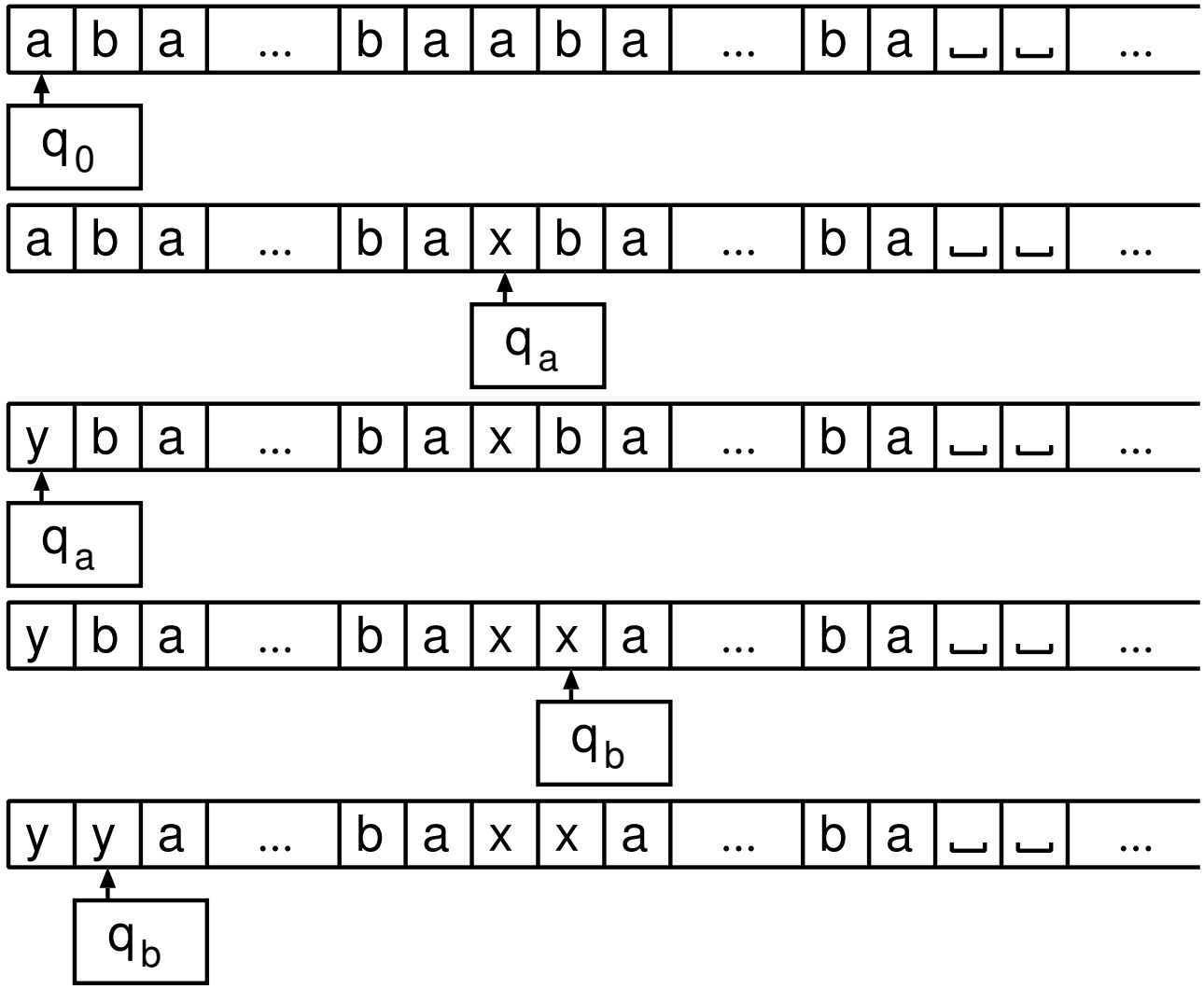
しかし、通常はこれと同等な
次のような計算モデルを考える
... チューリングマシン

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づ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 $A = \{ww | w \in \Sigma^*\}$ 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

$$\Updownarrow$$

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

$$\Updownarrow$$

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

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

$\Updownarrow$

T は A を認識し、
かつ、全ての入力に対し必ず停止する

$\Updownarrow$

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

かつ

$w \notin A \Longleftrightarrow$ **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

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