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）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**

- $E \to A$
- $E \to EBE$
- $E \to (E)$
- $A \to$ **変数名のどれか**
- $B \to$ **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）**

$\longrightarrow$ **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、 $(aa)^*b(bb)^*$

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

まとめて次のようにも書く

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

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、

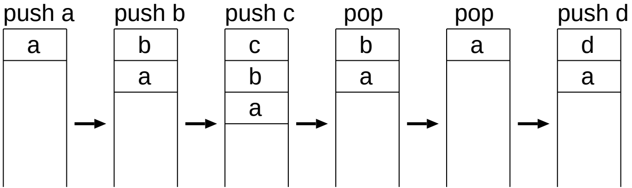
文脈自由言語は正規言語より真に広い !!

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
$\cdots$ LIFO (**L**ast **I**n **F**irst **O**ut)

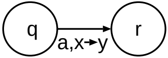
プッシュダウンオートマトンの形式的定義

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

- 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**

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

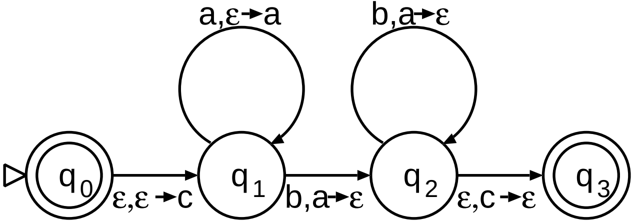
- $(r, y) \in \delta(q, a, x)$ **とは、**
  「入力 a **を読んだとき、**
  状態 q **でスタックの先頭が** x **なら、**
  スタックの先頭を y **に書換えて、**
  状態 r **に移って良い」**
  ということ (`pop; push y`**)**

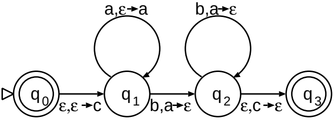
  状態遷移図では 

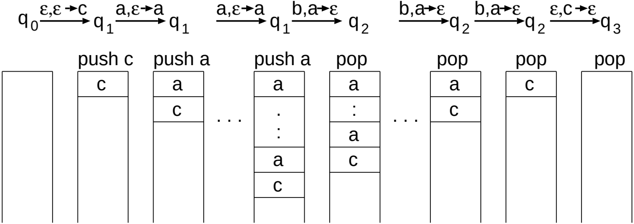
で表す
- $x = y$ **は書き換え無し**
- $x = \varepsilon$ **は（スタックの先頭を見ずに）push のみ**
- $y = \varepsilon$ **は pop のみ**
- $a = \varepsilon$ **は入力を読まずに遷移**

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

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



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

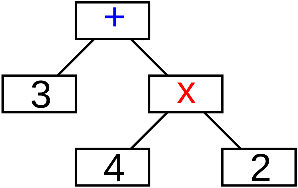
記憶場所としてプッシュダウンスタックを備えた計算モデルや仮想機械・処理系を

一般にスタックマシンという

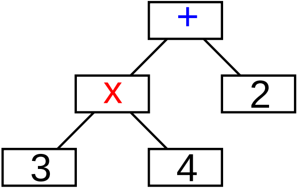
例：

- **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
(Computer Algebra System, CAS) 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

$\Downarrow$

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 $\times$ 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）

$$3 \times 4 + 2 \qquad\qquad 3 + 4 \times 2$$

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）