

有限オートマトンで認識できない言語が存在する



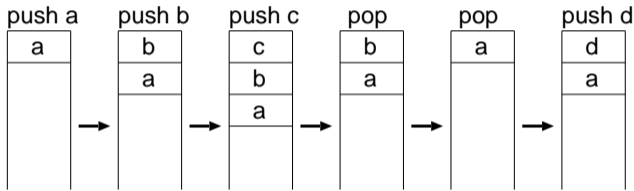
より強力な計算モデルが必要



- プッシュダウンオートマトン
- チューリングマシン

# 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)

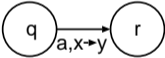
## プッシュダウンオートマトンの形式的定義

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

- $Q$  : 有限集合 … 状態の集合
- $\Sigma$  : 有限集合 … **alphabet**
- $\Gamma$  : 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  $y$ )

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

## スタックマシン

このように

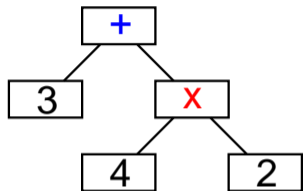
記憶場所としてプッシュダウンスタックを備えた  
計算モデルや仮想機械・処理系を

一般に**スタックマシン**という

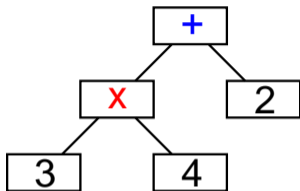
例：

- 逆ポーランド電卓
- **PostScript**

## 式と演算木



$$3+4 \times 2$$



$$3 \times 4 + 2$$

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  $\implies$  **push**
- 演算子  $\implies$  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  $\implies$  **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 × 項
- 因子 :: 定数 || 変数 || (式)

(+ と × 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 : 正規言語



L が或る有限オートマトンで認識される

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

本質的な違いは？

文脈自由言語は再帰 (recursion) を記述できる