計算の仕組み

オートマトンからラムダ計算まで

関山 太朗

国立情報学研究所 アーキテクチャ科学研究系 助教

今日のトピック

コンピュータ（計算機）が行う

計算

を数学的に扱う方法

「計算」とは?

- 与えられた「入力」に対し「出力」を返すプロセス

例1: 四則演算

入力

数と+-×÷で書ける式

3 × 4 + 6 ÷ -2 ⟹ 9

出力

+-×÷を含まない式

「計算」とは?

- 与えられた「入力」に対し「出力」を返すプロセス

例2: 文字のマッチング

入力 文字列		出力 入力がAを含めばYES 含まなければNO
Calculation	⇒	YES
Computer	⇒	NO

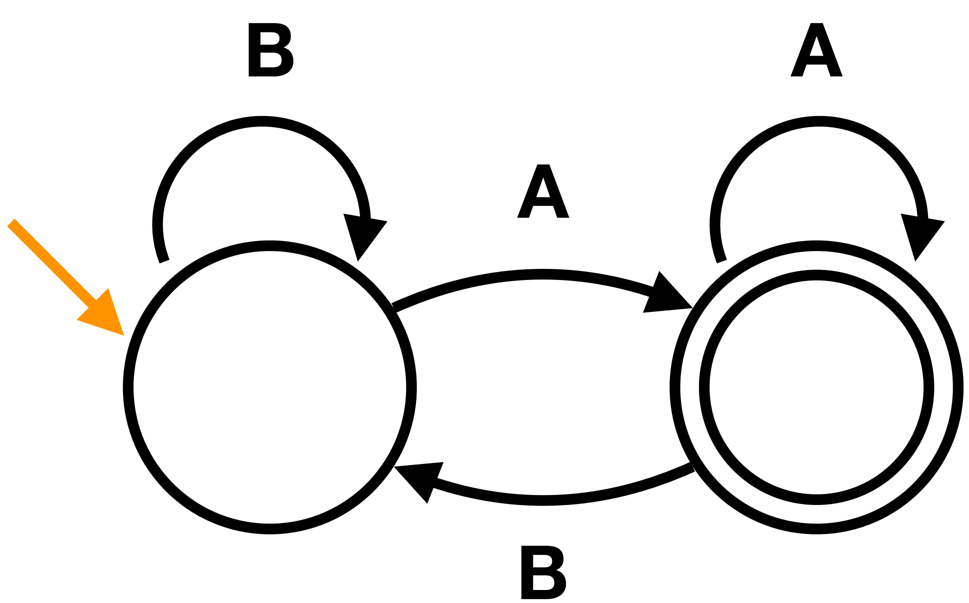
本日の内容

計算モデル（計算の数学的定義）の紹介

1. 有限オートマトン
2. チューリングマシン
3. ラムダ計算

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：A

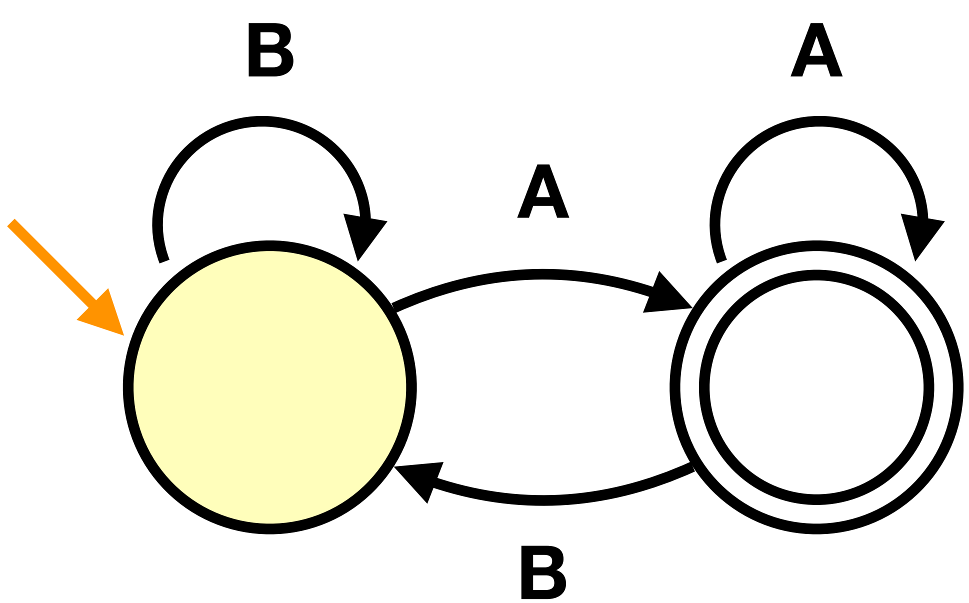
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：A

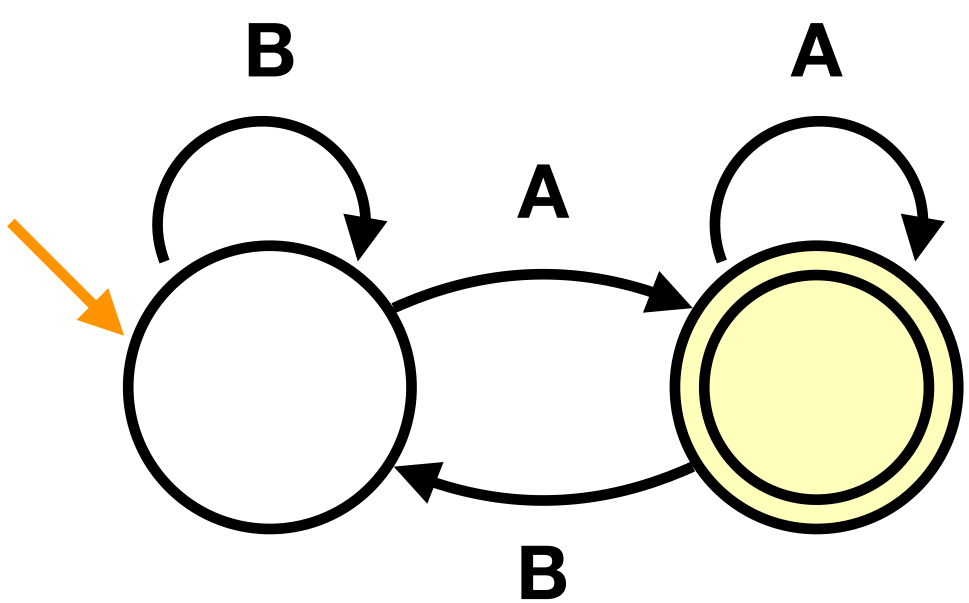
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：A
出力：YES

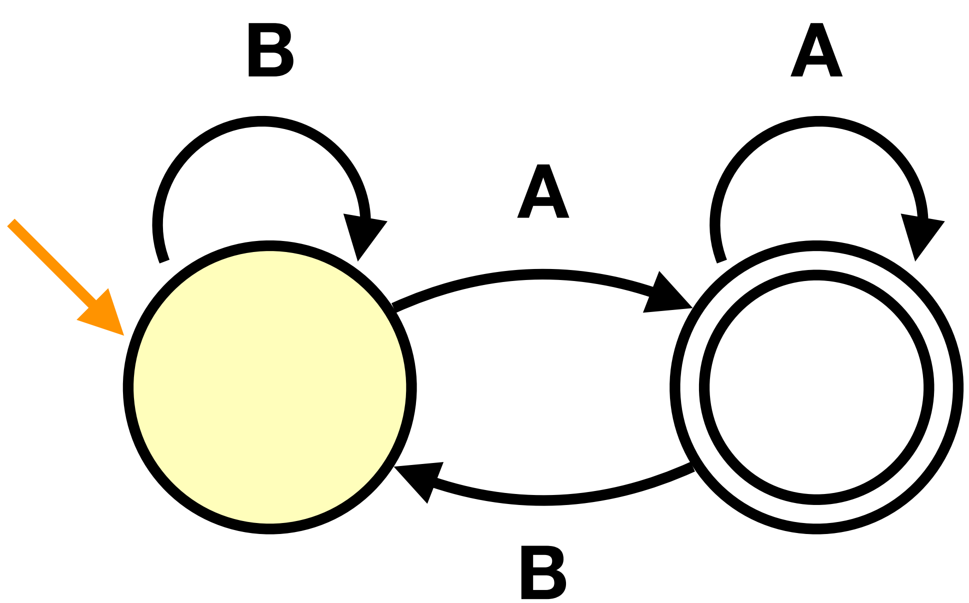
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：AB

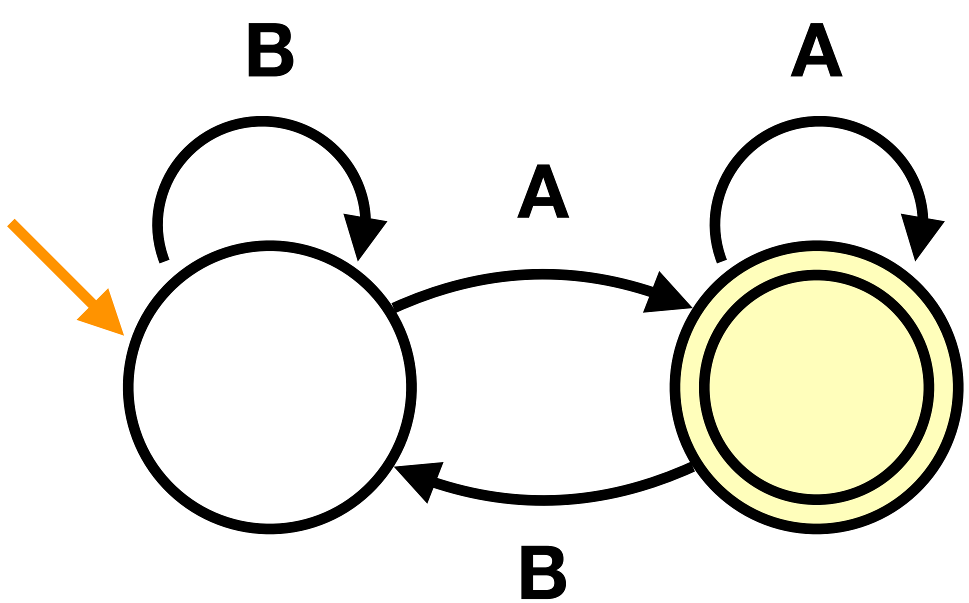
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：AB

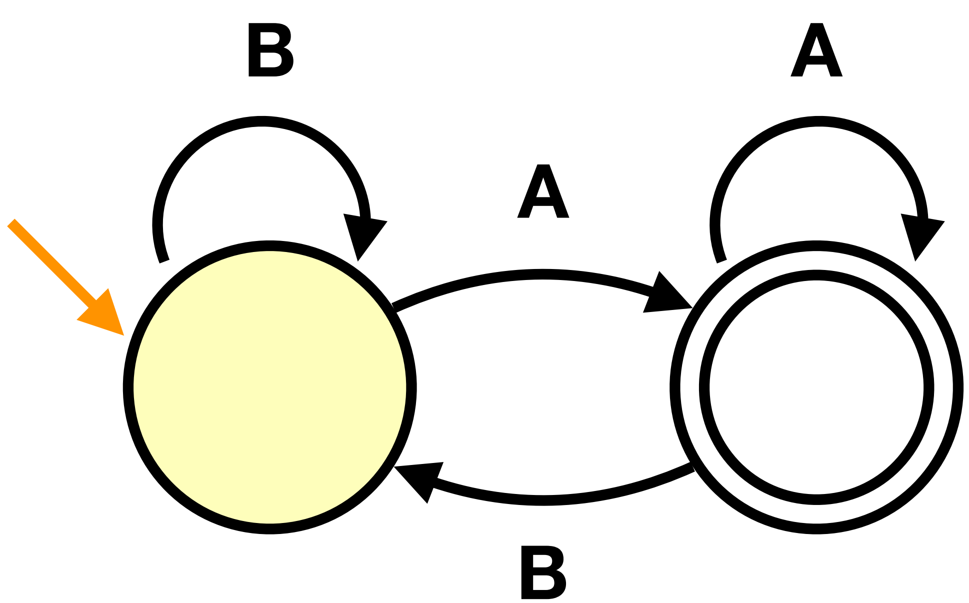
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：AB

出力：NO

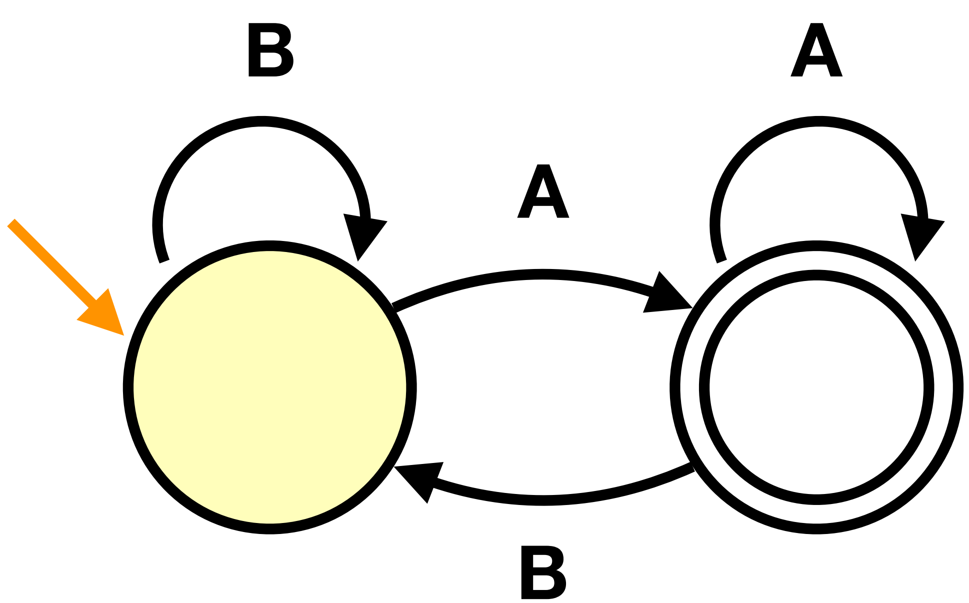
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：ABA

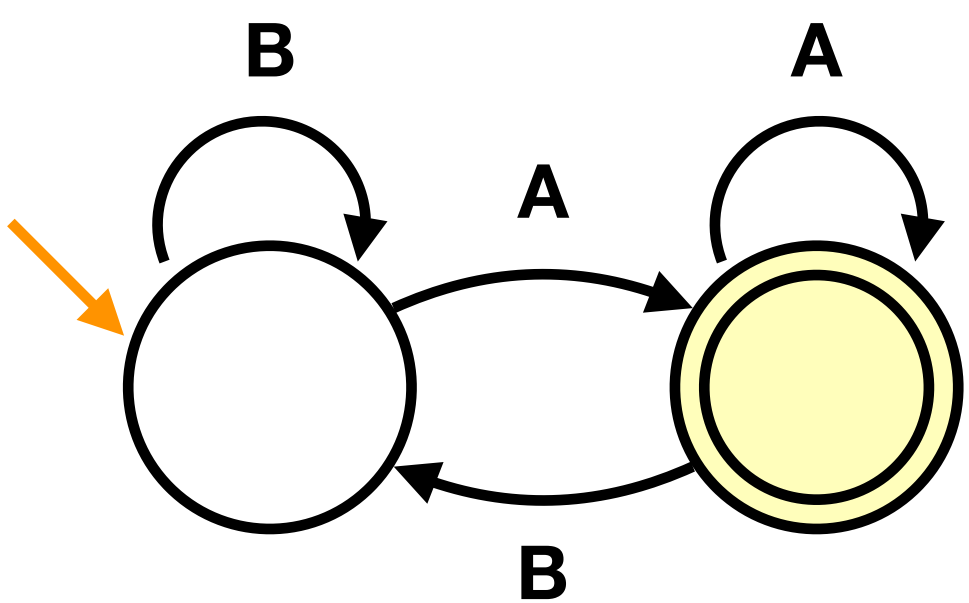
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力：ABA

出力：YES

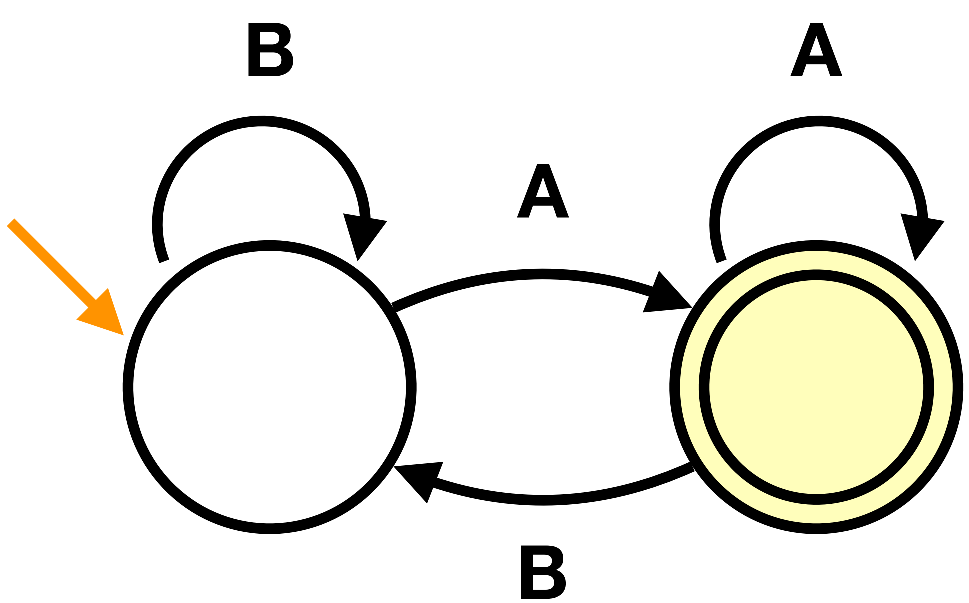
- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

  受理状態 ◎ であれば **YES を出力**

  その他の状態 ○ であれば **NO を出力**

有限オートマトン

- 文字列を扱うための計算モデル
- **入力：文字列** から **出力：YES または NO** を計算する機械



入力がAで終わる ⇨ **出力YES**

それ以外 ⇨ **出力NO**

- 入力文字列に従って計算機の状態が変化する
- 状態は有限個の ○ か ◎ で表現
- 入力を最初から読んでいき、読み終わったときの状態が

受理状態 ◎ であれば **YES を出力**

その他の状態 ○ であれば **NO を出力**

単純だからこその便利さ

- 構造が単純なのでいろいろな解析が可能
- 次の問題を解くための手順（アルゴリズム）が知られている

受理判定 有限オートマトンのある文字列に対する出力を判定する

等価性判定 二つの有限オートマトンが全ての文字列に対して同じ出力を返すかを判定する

空判定 与えられた有限オートマトンが YES と出力するような文字列が存在するかを判定する

何に使う?

- 文字列処理
  - 計算機の分野では最も基本的な処理の一つ
  - 入力文字列があるパターンに沿ったものであるかを判定できる
- プログラム検証
  - プログラムを有限オートマトンとして抽象化し、先に述べたアルゴリズムを通してプログラムの性質を調べる
- 様々な計算モデルのベースとなる
  - 時間オートマトン、Büchi オートマトン、YES・NO以外の出力ができるオートマトンなどが拡張として知られている

何ができない?

- 次のような有限オートマトンは構築できないことが**証明できる**

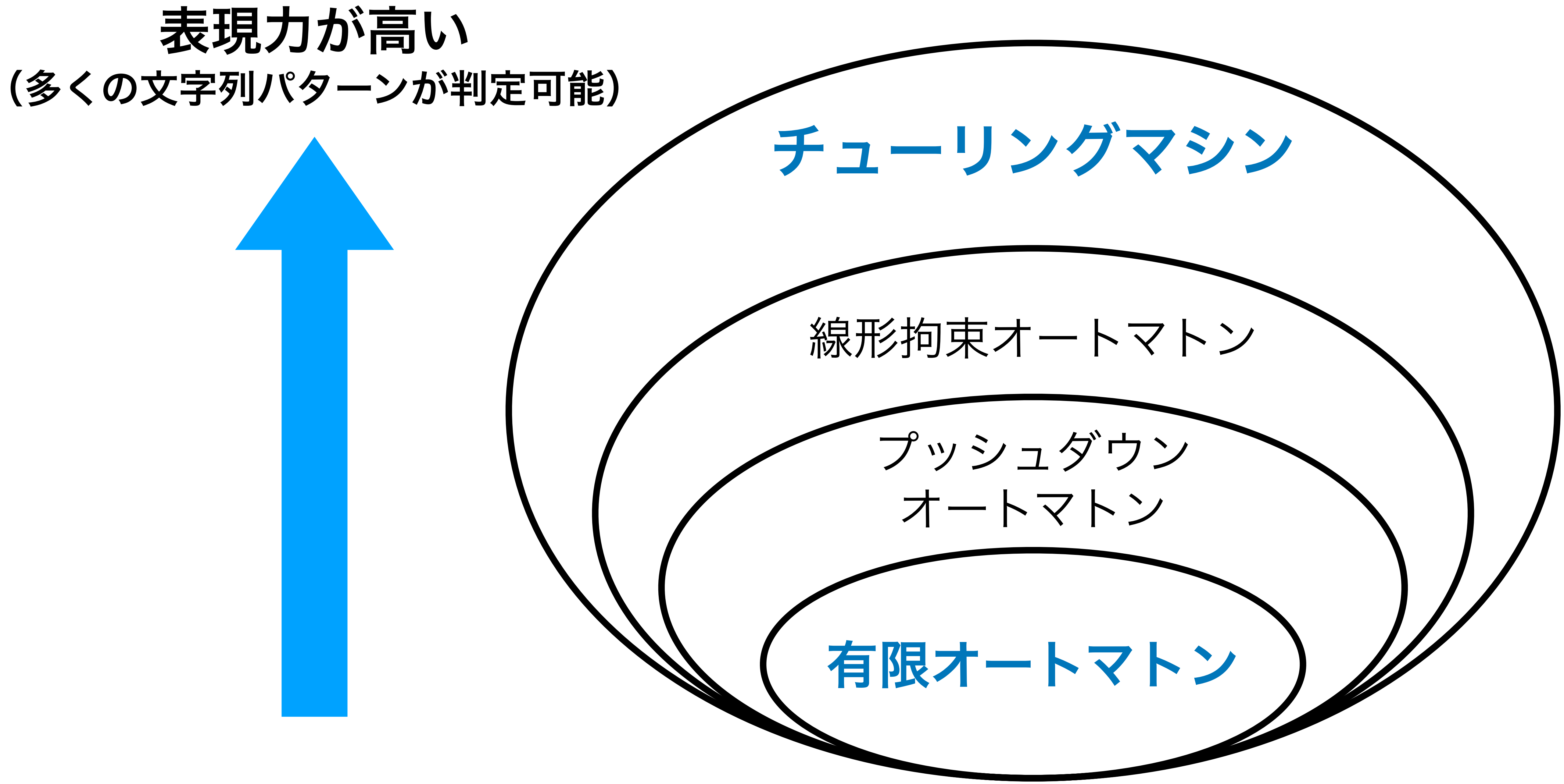
$A^n B^n$ の文字列

$\left\{ \begin{array}{c} \text{AB, AABB,} \\ \text{AAABBB,} \\ \dots \end{array} \right\}$ ⟹ YES

それ以外 ⟹ NO

アイデア：オートマトンの状態数は有限なので
状態数を超える文字列の情報を覚えておくことはできない！

チョムスキー階層


表現力が高い
（多くの文字列パターンが判定可能）
チューリングマシン
線形拘束オートマトン
プッシュダウン
オートマトン
有限オートマトン

チューリングマシン

Alan Turing

https://en.wikipedia.org/wiki/Alan_Turing

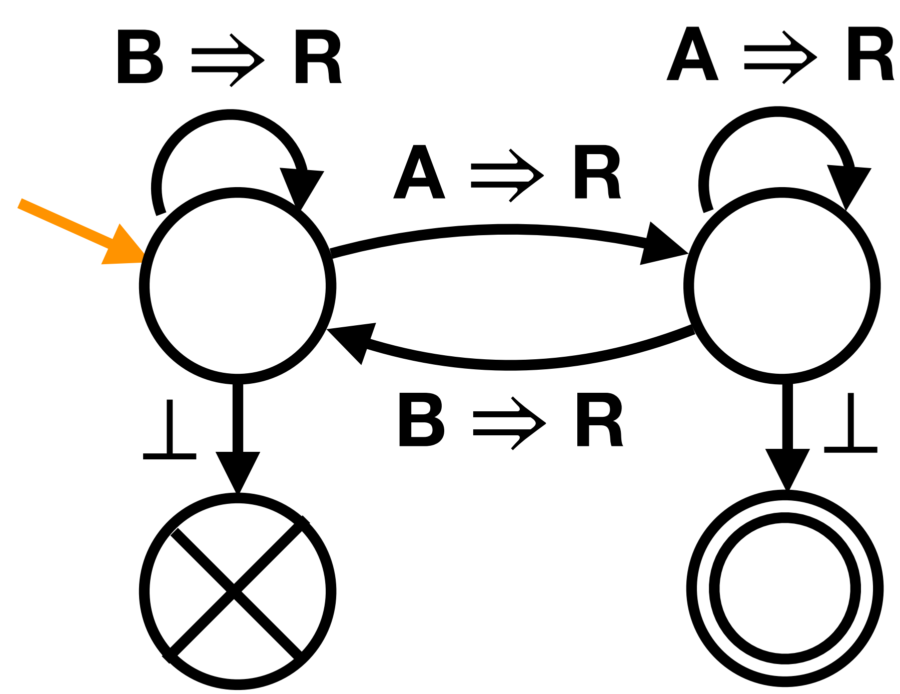
文字情報を**テープ**に書いて保存できる計算モデル

- テープの長さは**無制限**、
  有限長の文字列なら保存可能
- テープの読み書き位置を
  ヘッド⬇で指す
- 計算は**状態遷移系**で表現
  現在の状態とヘッドの文字から
  1. 次の状態
  2. ヘッドへ書き込む文字
  3. ヘッドを動かす方向 (**左L**か**右R**)

  を決める
- 受理状態◎に到達すれば **YES**
  拒否状態⊗に到達すれば **NO** を出力

入力ABAに対する初期テープ

Aで入力に**YESを出力**



チューリングマシン

Alan Turing

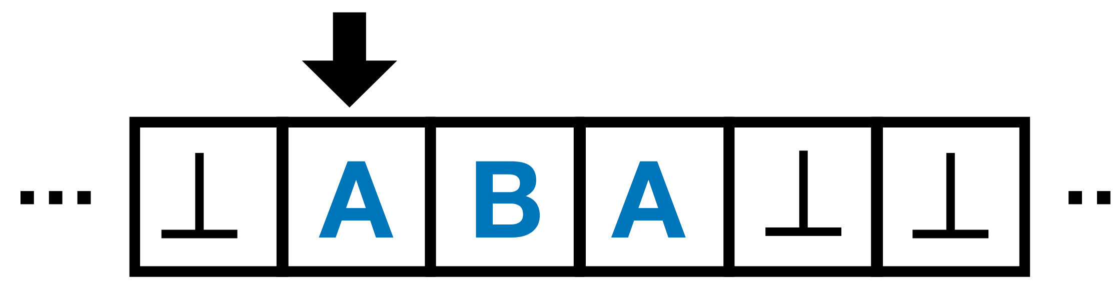
https://en.wikipedia.org/wiki/Alan_Turing

文字情報を**テープ**に書いて保存できる計算モデル

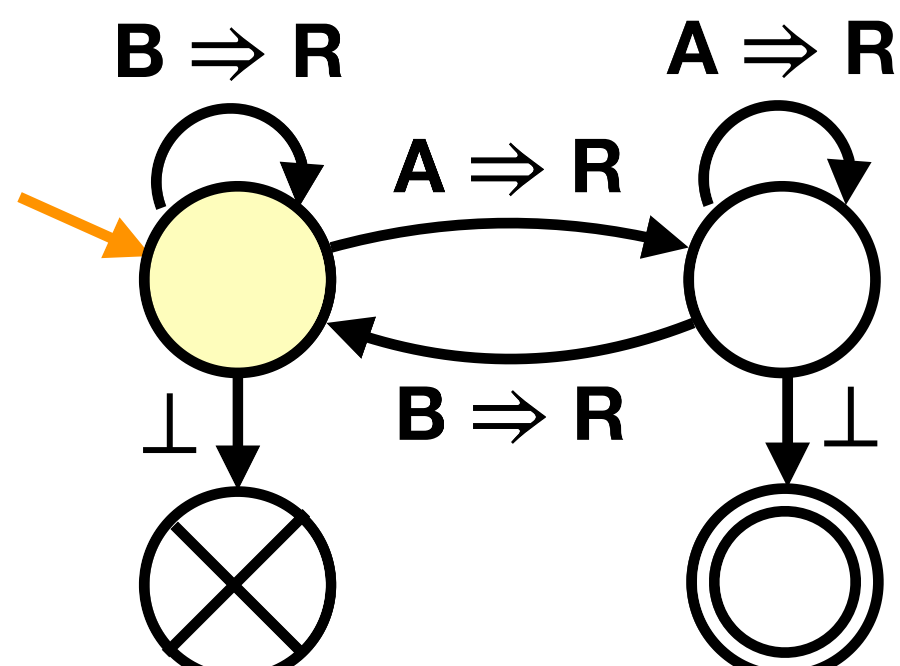
- テープの長さは**無制限**、
  有限長の文字列なら保存可能
- テープの読み書き位置を
  ヘッド⬇で指す
- 計算は**状態遷移系**で表現
  現在の状態とヘッドの文字から
  1. 次の状態
  2. ヘッドへ書き込む文字
  3. ヘッドを動かす方向 (**左Lか右R**)

  を決める
- 受理状態◎に到達すれば **YES**
  拒否状態⊗に到達すれば **NO** を出力

入力ABAに対する初期テープ



Aで入力に**YESを出力**



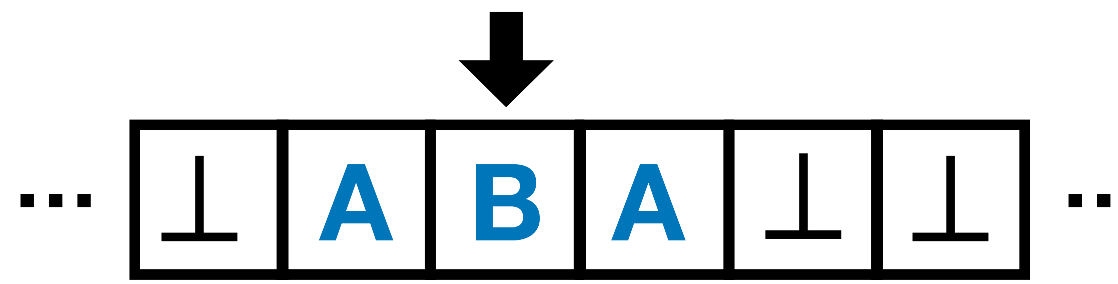
チューリングマシン

Alan Turing
https://en.wikipedia.org/wiki/Alan_Turing

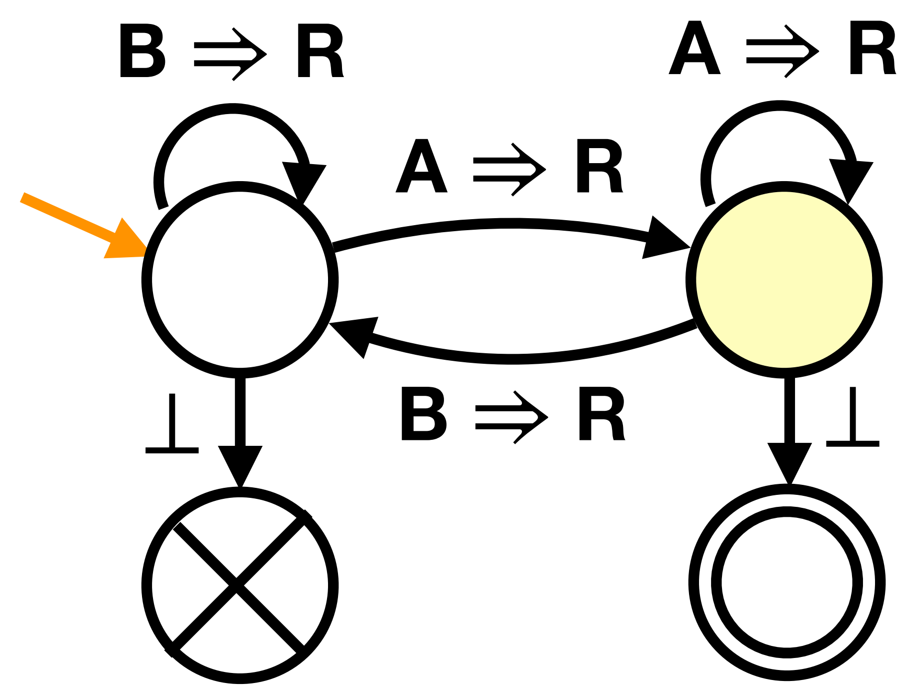
文字情報を**テープ**に書いて保存できる計算モデル

- テープの長さは**無制限**、
  有限長の文字列なら保存可能
- テープの読み書き位置を
  ヘッド⬇で指す
- 計算は**状態遷移系**で表現
  現在の状態とヘッドの文字から
  1. 次の状態
  2. ヘッドへ書き込む文字
  3. ヘッドを動かす方向 (**左L**か**右R**)

  を決める
- 受理状態◎に到達すれば **YES**
  拒否状態⊗に到達すれば **NO** を出力



Aで入力にYESを出力



チューリングマシン

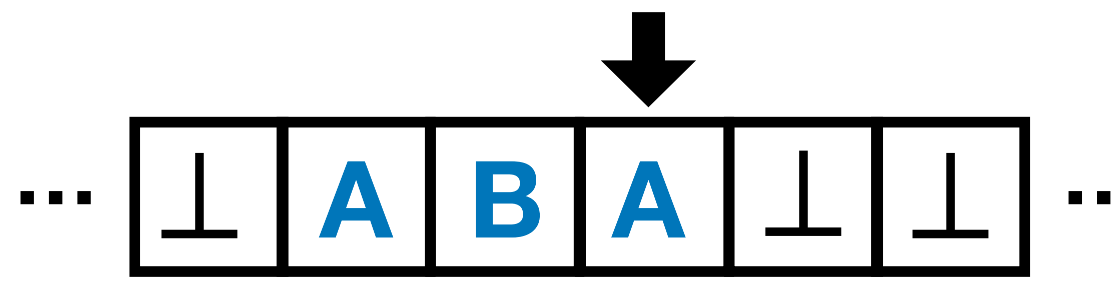
Alan Turing

https://en.wikipedia.org/wiki/Alan_Turing

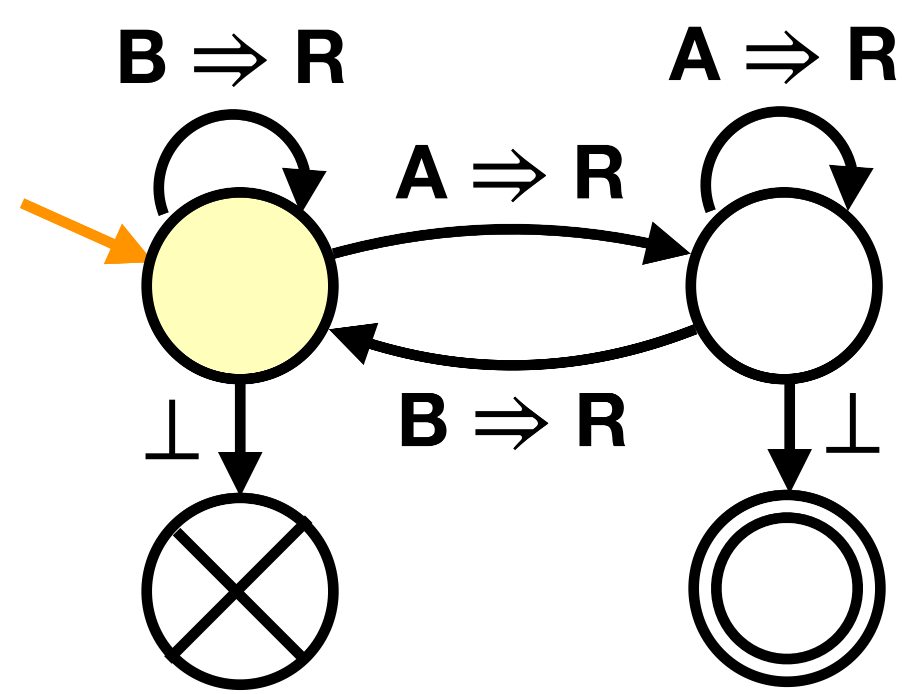
文字情報を**テープ**に書いて保存できる計算モデル

- テープの長さは**無制限**、
  有限長の文字列なら保存可能
- テープの読み書き位置を
  ヘッド⬇で指す
- 計算は**状態遷移系**で表現
  現在の状態とヘッドの文字から
  1. 次の状態
  2. ヘッドへ書き込む文字
  3. ヘッドを動かす方向 (**左L**か**右R**)

  を決める
- 受理状態◎に到達すれば **YES**
  拒否状態⊗に到達すれば **NO** を出力



Aで入力にYESを出力



チューリングマシン

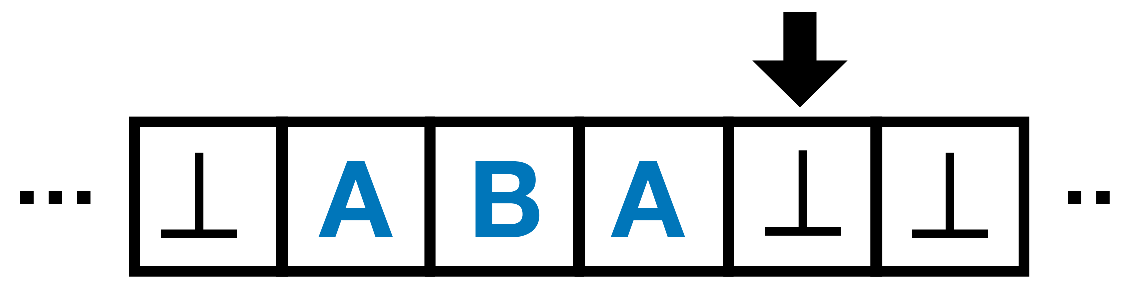
Alan Turing

https://en.wikipedia.org/wiki/Alan_Turing

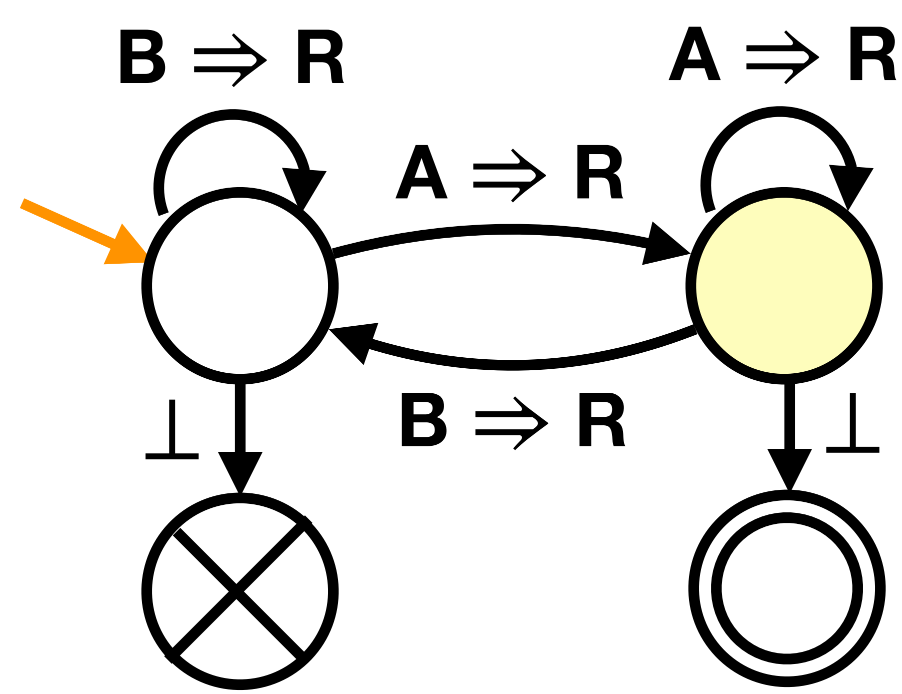
文字情報を**テープ**に書いて保存できる計算モデル

- テープの長さは**無制限**、
  有限長の文字列なら保存可能
- テープの読み書き位置を
  ヘッド⬇で指す
- 計算は**状態遷移系**で表現
  現在の状態とヘッドの文字から
  1. 次の状態
  2. ヘッドへ書き込む文字
  3. ヘッドを動かす方向 (**左L**か**右R**)

  を決める
- 受理状態◎に到達すれば **YES**
  拒否状態⊗に到達すれば **NO** を出力



Aで入力にYESを出力



チューリングマシン

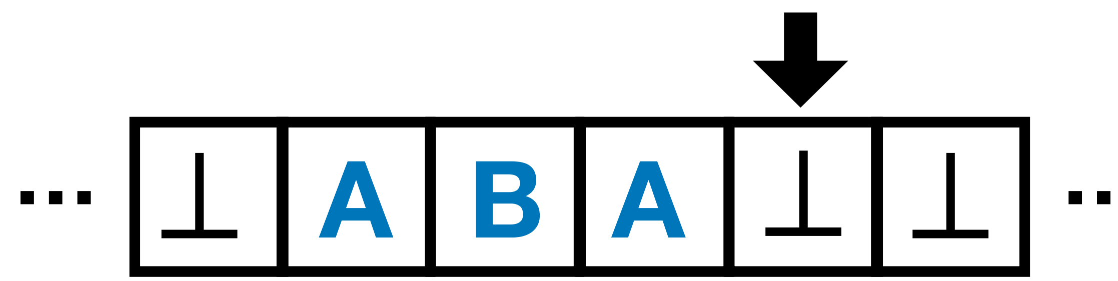
Alan Turing

https://en.wikipedia.org/wiki/Alan_Turing

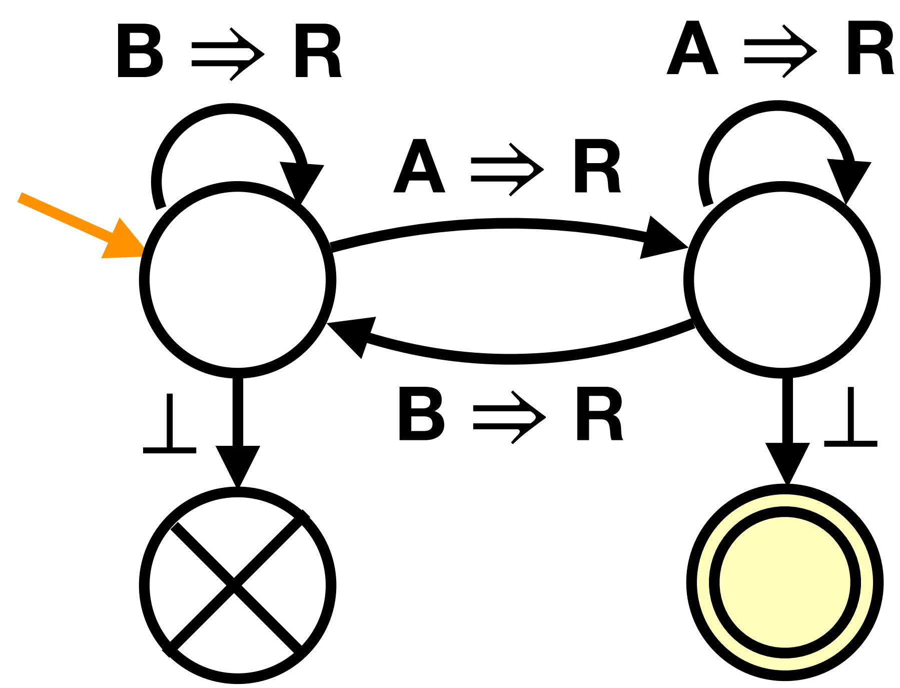
文字情報を**テープ**に書いて保存できる計算モデル

- テープの長さは**無制限**、
  有限長の文字列なら保存可能
- テープの読み書き位置を
  ヘッド⬇で指す
- 計算は**状態遷移系**で表現
  現在の状態とヘッドの文字から
  1. 次の状態
  2. ヘッドへ書き込む文字
  3. ヘッドを動かす方向 (**左L**か**右R**)

  を決める
- 受理状態◎に到達すれば **YES**
  拒否状態⊗に到達すれば **NO** を出力



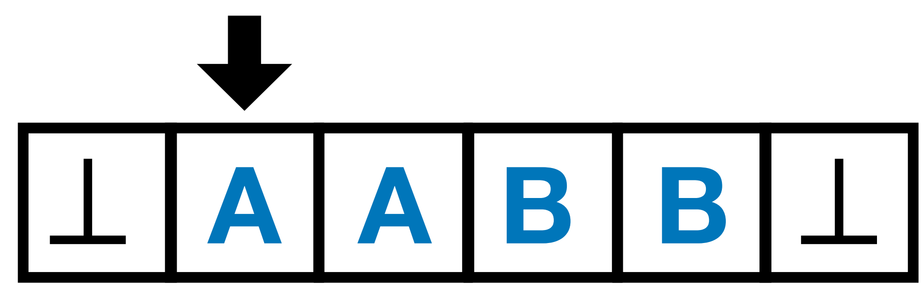
Aで入力にYESを出力



$A^n B^n$ を判定する チューリングマシン

アイデア　調べた入力文字A,Bを X でマーキング

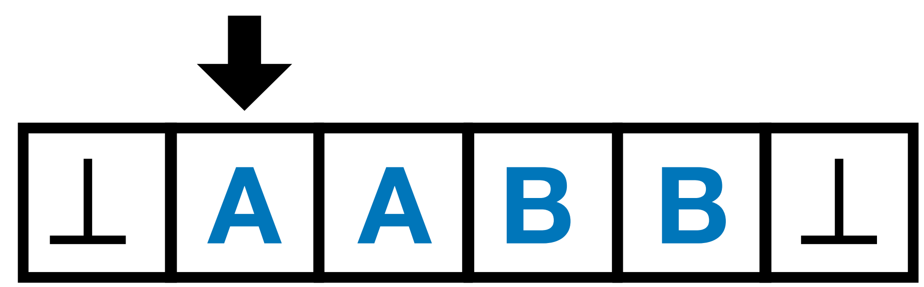
1. 右に B を探す（なければ NO を出力）
2. X を書き込み、X 以外の文字が見つかるまで左に移動
   - A 以外が見つかった ⇒ NO を出力
   - A が見つかった ⇒ X を書き込み、X 以外の文字が見つかるまで右に移動
     - A ⇒ NO を出力
     - B が見つかった ⇒ もう一度 2. を行う
     - ⊥ が見つかった ⇒ 左が全て X なら YES、そうでなければ NO を出力



$A^n B^n$ を判定する
チューリングマシン

アイデア　調べた**入力文字A,B**を **X** でマーキング

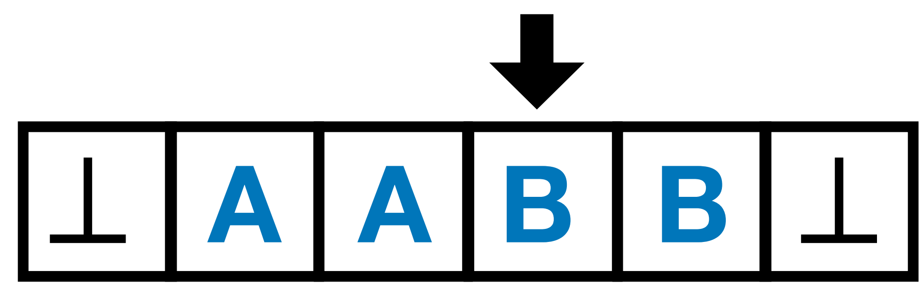
1. 右に **B** を探す（なければ **NO を出力**）
2. **X** を書き込み、**X** 以外の文字が見つかるまで左に移動
   - **A** 以外が見つかった ⇒ **NO を出力**
   - **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - **A** ⇒ **NO を出力**
     - **B** が見つかった ⇒ もう一度 2. を行う
     - ⊥ が見つかった ⇒ 左が全て **X** なら **YES**、そうでなければ **NO を出力**



A^n B^n を判定する チューリングマシン

アイデア　調べた**入力文字A,B**を **X** でマーキング

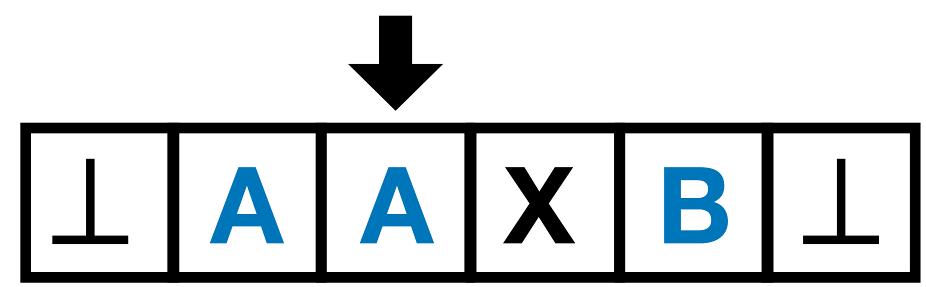
1. 右に **B** を探す（なければ **NO を出力**）
2. <u>**X** を書き込み、**X** 以外の文字が見つかるまで左に移動</u>
   - **A** 以外が見つかった ⇒ **NO を出力**
   - **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - **A** ⇒ **NO を出力**
     - **B** が見つかった ⇒ もう一度 2. を行う
     - ⊥ が見つかった ⇒ 左が全て **X** なら **YES**、そうでなければ **NO を出力**



$A^n B^n$ を判定する チューリングマシン

アイデア　調べた**入力文字A,B**を **X** でマーキング

1. 右に **B** を探す（なければ **NO を出力**）
2. **X** を書き込み、**X** 以外の文字が見つかるまで左に移動
   - □ **A** 以外が見つかった ⇒ **NO を出力**
   - □ **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - ◆ **A** ⇒ **NO を出力**
     - ◆ **B** が見つかった ⇒ もう一度 2. を行う
     - ◆ ⊥ が見つかった ⇒ 左が全て **X** なら **YES**、そうでなければ **NO を出力**



$A^n B^n$ を判定する チューリングマシン

アイデア　調べた**入力文字A,B**を **X** でマーキング

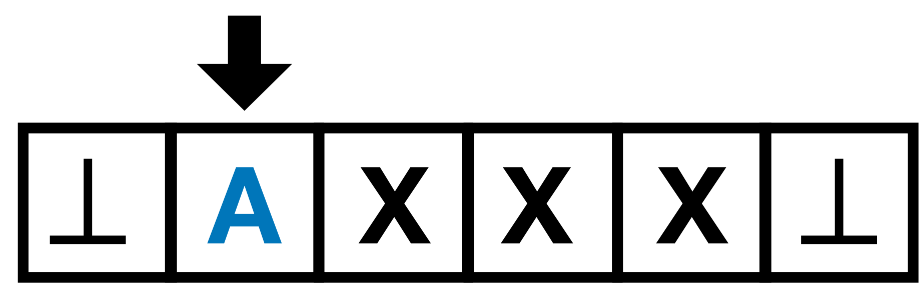
1. 右に **B** を探す（なければ **NO を出力**）
2. **X** を書き込み、**X** 以外の文字が見つかるまで左に移動
   - **A** 以外が見つかった ⇒ **NO を出力**
   - **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - **A** ⇒ **NO を出力**
     - **B** が見つかった ⇒ もう一度 2. を行う
     - ⊥ が見つかった ⇒ 左が全て **X** なら **YES**、そうでなければ **NO を出力**

⊥	**A**	**X**	**X**	**B**	⊥

$A^n B^n$ を判定する チューリングマシン

アイデア　調べた**入力文字A,B**を **X** でマーキング

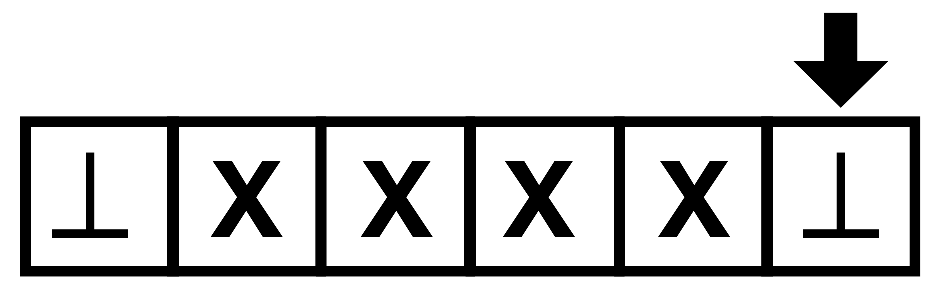
1. 右に **B** を探す（なければ **NO を出力**）
2. <u>**X** を書き込み、**X** 以外の文字が見つかるまで左に移動</u>
   - □ **A** 以外が見つかった ⇒ **NO を出力**
   - □ **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - ◆ **A** ⇒ **NO を出力**
     - ◆ **B** が見つかった ⇒ もう一度 2. を行う
     - ◆ ⊥ が見つかった ⇒ 左が全て **X** なら **YES**、そうでなければ **NO を出力**



$A^n B^n$ を判定する チューリングマシン

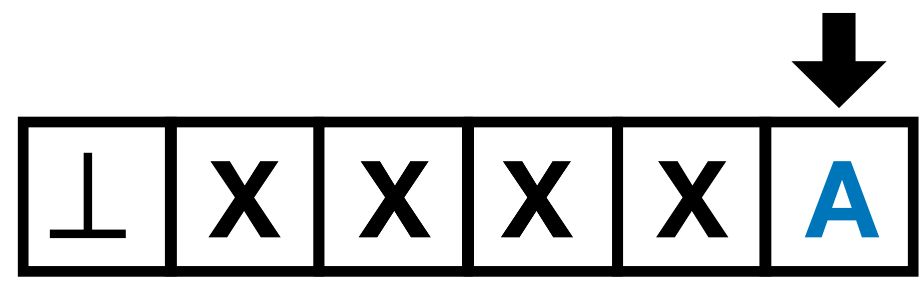
アイデア　調べた**入力文字A,B**を **X** でマーキング

1. 右に **B** を探す（なければ **NO を出力**）
2. **X** を書き込み、**X** 以外の文字が見つかるまで左に移動
   - □ **A** 以外が見つかった ⇒ **NO を出力**
   - □ **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - ◆ **A** ⇒ **NO を出力**
     - ◆ **B** が見つかった ⇒ もう一度 2. を行う
     - ◆ ⊥ が見つかった ⇒ 左が全て **X** なら **YES**、そうでなければ **NO を出力**



A^n B^n を判定する
チューリングマシン

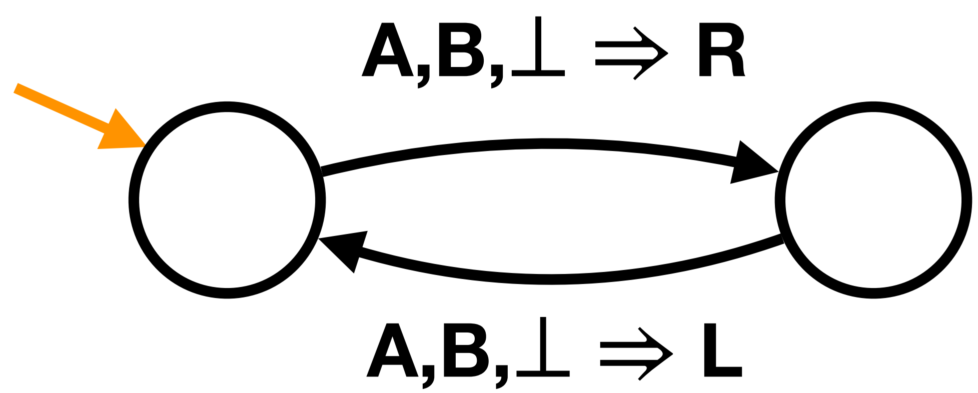
アイデア　調べた**入力文字A,B**を **X** でマーキング



1. 右に **B** を探す（なければ **NO を出力**）
2. **X** を書き込み、**X** 以外の文字が見つかるまで左に移動
   - **A** 以外が見つかった ⇒ **NO を出力**
   - **A** が見つかった ⇒ **X** を書き込み、**X** 以外の文字が見つかるまで右に移動
     - **A ⇒ NO を出力**
     - **B** が見つかった ⇒ もう一度 2. を行う
     - ⊥ が見つかった ⇒
左が全て **X** なら **YES**、
そうでなければ **NO を出力**

停止しない機械

- YES も NO も出力せず、ずっと動き続けるような チューリングマシンが構築できる



何ができない?

- 次の問題を解く一般的なアルゴリズムは存在しない

  受理判定 チューリングマシンの文字列に対する出力を判定する

  等価性判定 二つのチューリングマシンが等価であるかを判定する

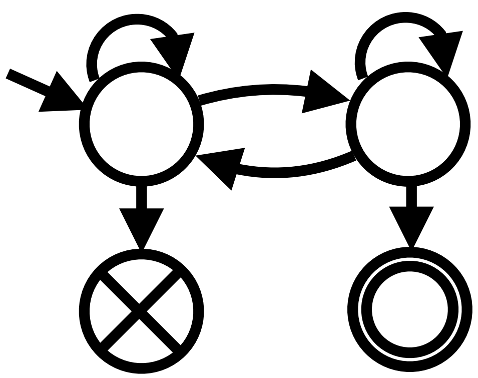
  停止性判定 チューリングマシンが停止するかを判定する

  - 証明は**対角線論法**を使って行う

- チューリングマシンで何が実現できないかを調べることも重要

他の計算モデルとの比較

チューリングマシン

帰納的関数

Jacques Herbrand

photographed by
Natascha Artin-Brunswick

Kurt Gödel

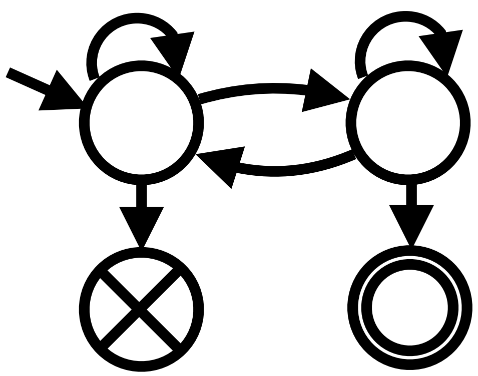
https://en.wikipedia.org/wiki/
Kurt_G%C3%B6del

ラムダ計算

Alonzo Church

他の計算モデルとの比較

チューリングマシン ＝ 帰納的関数 ＝ ラムダ計算



Jacques Herbrand

photographed by
Natascha Artin-Brunswick

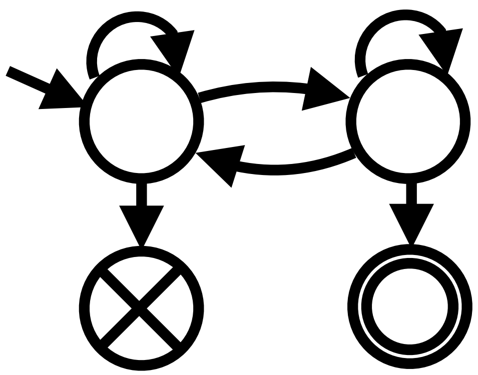
Kurt Gödel

https://en.wikipedia.org/wiki/Kurt_G%C3%B6del

Alonzo Church

他の計算モデルとの比較

チューリングマシン ＝ 帰納的関数 ＝ ラムダ計算

Jacques Herbrand

photographed by
Natascha Artin-Brunswick

Kurt Gödel

https://en.wikipedia.org/wiki/Kurt_G%C3%B6del

Alonzo Church

Church-Turing の提唱

これらのモデルで計算できるものを「**計算可能**」
と呼ぼうという提案（あるいは計算可能性の定義）

ラムダ計算

- **関数**をベースにした、**書き換えによる計算**を行うモデル
- ラムダ計算の式 **M** は三つの要素から構成される

関数	$\lambda x.M$	f(x) = M の意味
関数の呼出し	$M_1\ M_2$	f(M_2)（ただしfはM_1が表わす関数）
変数の参照	x	

- 式は次の計算規則によって書き換えられる

$$(\lambda x.M_1)\ M_2 \longrightarrow M_1[\,x := M_2\,]$$

M_1 の変数 x に
M_2 を代入した式

ラムダ計算の"計算"

書き換え規則

$$(\lambda x.M_1)\ M_2 \longrightarrow M_1\,[\,x := M_2\,]$$

$$(\lambda x.x + 1)\ 2 \longrightarrow (x + 1)\,[\,x := 2\,] = 3$$

$$(\lambda x.\lambda y.\ x + y)\ 2\ 3 \longrightarrow (\lambda y.x + y)\,[\,x := 2\,]\ 3 = (\lambda y.2 + y)\ 3$$

$$\longrightarrow (2 + y)\,[\,y := 3\,] = 2 + 3$$

$$\longrightarrow 5$$

$$(\lambda x.\lambda y.\ x)\ (\lambda z.z) \longrightarrow (\lambda y.x)\,[\,x := \lambda z.z\,] = (\lambda y.\lambda z.z)$$

停止しない式

$$(\lambda x.x\ x)\ (\lambda x.x\ x) \longrightarrow (x\ x)[\,x := \lambda x.x\ x\,]$$

$$= (\lambda x.x\ x)\ (\lambda x.x\ x)$$

$$\longrightarrow (x\ x)[\,x := \lambda x.x\ x\,]$$

$$= (\lambda x.x\ x)\ (\lambda x.x\ x)$$

$$\longrightarrow \cdots$$

計算が停止しない = 何度書き換えても、書き換え可能な式が現われる

チューリングマシン vs. ラムダ計算

- 計算可能性は同じだが**計算のプロセス・記述が全く異なる**
  - **効率性の変化**
  - **プログラミング言語への影響**

命令的言語

- チューリングマシン的
- 計算の手順を記述する
- 計算結果は記憶容量（メモリ）に保存

関数的言語

- ラムダ計算的
- 関数の入出力を記述する
- 計算結果は次の関数への引数となる

まとめ

- 三つの代表的な計算モデルについて紹介しました
  - 有限オートマトン
  - チューリングマシン
  - ラムダ計算
- 計算モデルの計算能力とその解析のしやすさには トレードオフがある
- 同じ計算能力をもつモデルでも計算プロセス・記述方法の差が実際の計算機上で動かすときに重要になる