ERATO 蓮尾メタ数理システムデザインプロジェクト (ERATO MMSD)

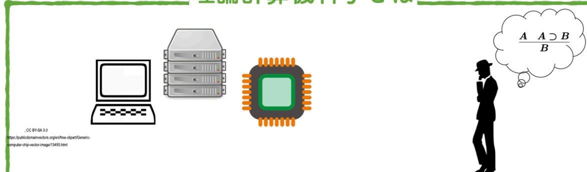
理論計算機科学入門： 有限と無限のあいだ ～ オートマトン理論を例に

連絡先: 蓮尾 一郎，ERATO 蓮尾メタ数理システムデザインプロジェクト/国立情報学研究所　http://group-mmm.org/eratommsd

はじめに

* このポスター発表は，2019年度 NII 市民講座 第3回「理論計算機科学入門 有限と無限のあいだ－数学的理論から、AI・自動運転－」のダイジェストです．
* 興味がお有りの方はぜひそちらを！
  * 市民講座ページ リンク
  * 講演ビデオ (youtube)
  * スライド, Q&A, 当日レポート

理論計算機科学とは

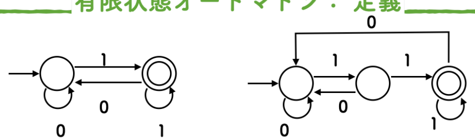


* 計算機，計算機システムの振る舞いを数学的に研究
  * 速さ（アルゴリズム，計算量理論）
  * 正しさ（「バグがないか？」．形式手法，プログラミング言語理論）
* 使う数学：
  * 論理学，代数学，グラフ理論など
  * 離散的（↔ 連続的）
  * 有限（記号の世界）と無限（イデアの世界）をはっきり区別

人間の手の届かない無限を，有限の記号列で表現

* ➜ 有限と無限のせめぎあいがテーマ

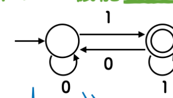
有限状態オートマトン： 定義



* 有限個の状態（左は2個，右は3個）
* 状態間の遷移（矢印）
  * 各遷移は文字でラベル付け（ここでの文字は0,1）
* 初期状態（矢印で表現）
* 状態の「色付け」．2色：
  * ○：非受理状態
  * ◎：受理状態

> * 計算モデルのうち，単純なものの一つ
>   * 計算モデル：「計算とは何か？」の数学的定義
> * さまざまな計算モデルを，能力順にならべると：
>   有限オートマトン < …
>   < プッシュダウン・オートマトン < …
>   < チューリングマシン

有限状態オートマトン： 機能



* 機能：
  * 有限長の文字列を読んで，
  * 「受理 (OK!)」あるいは「非受理 (NG!)」と言う
* 初期状態から，文字列に沿って遷移をたどっていって，最後にたどり着いた状態が ◎ なら「受理」，○ なら「非受理」
* すなわち，オートマトン M は，文字列の集合 L(M) を表現する（ひきおこす）．ここでは，
  L(M) = { w1 ： w は任意の文字列}
  = {1, 01, 11, 001, 011, 101, 111, … }
* つまり…
  * 無限の数学的実体 L(M) を
  * 有限のフォーマリズムであるオートマトンMが表現している
* 有限と無限のせめぎあい

00 は非受理

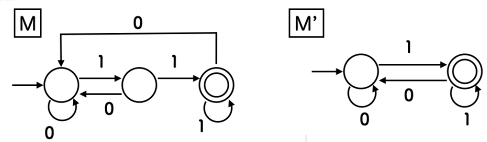
001 は受理

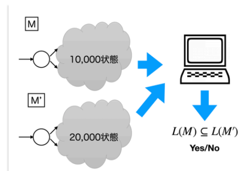
1100 は?

空列（長さ0の文字列）は？

文字列 w が受理される ⇔ w の最後の文字が 1

例題： 有限状態オートマトンの包含問題





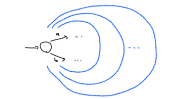
* クイズ　次は成り立つ？ $L(M) \subseteq L(M')$
* 答え Yes!
  L(M) = { 11 で終わる文字列全体 },
  L(M') = { 1 で終わる文字列全体 }
  * しかしこれは「人間の頭を使った答え」
  * 自動，アルゴリズムで解きたい ➜ 大きなオートマトンにも適用可能なように（上図）
* （間違ったアルゴリズム）
  文字列 w それぞれについて，
  「w が M に受理されるならば，w は M' にも受理される」
  ことを確かめる
  * L(M) は無限集合 ➜ いつまでたっても終わらない！
* アイデア： 有限の表現たるオートマトン M, M' を使ってがんばる

w は無限個ある．一方，人間の一生は有限

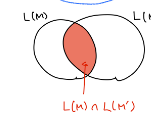
理論計算機科学のテーマ：「有限の手段をうまく使って無限の対象をどうにか操作」

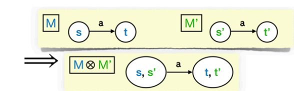
包含関係を解くアルゴリズム，3つの材料

* 材料1（空判定）
  * 入力：有限状態オートマトン M
  * 出力： $L(M) = \emptyset$ が成り立つかどうか　（∅は空集合，からっぽ）
  * アルゴリズム：
    「初期状態（→○）から辿り着ける状態」を列挙して（有限だからできる），◎が含まれないかどうかチェック

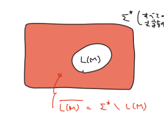


* 材料2（オートマトンの同期積）
  * 入力：有限状態オートマトン M, M'
  * 出力： $L(M \otimes M') = L(M) \cap L(M')$ となるオートマトン $M \otimes M'$
  * アルゴリズム（作り方）：
    「2つのオートマトンを両方同時に動かす」（右図）





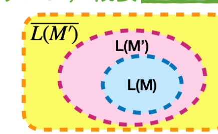
* 材料3（言語反転）
  * 入力：有限状態オートマトン M
  * 出力： $L(M^c) = \overline{L(M)} = \Sigma^* \setminus L(M)$ となるオートマトン M^c
  * アルゴリズム（作り方）：
    ○と◎を反転



包含関係を解くアルゴリズム，概要

* 次が成り立つことに注意（右図参照）：

$$L(M) \subseteq L(M')$$
$$\iff L(M) \cap \overline{L(M')} = \emptyset$$
$$\iff L(M \otimes (M')^c) = \emptyset$$



* よって，次のようにすればよい
  * $(M')^c$ を作る　（○↔◎，材料３）
  * $M \otimes (M')^c$ を作る　（オートマトンの同期積，材料２）
  * $M \otimes (M')^c$ の空判定　（◎が到達可能かどうか探索，材料１）
* … ぜひ自分で試してみてください．プログラミングの練習問題
  * かなり大きなオートマトンに対してもガシガシ動くはず