チューリングマシンのシミュレータ

コンピュータの数学的モデルであるチューリングマシンのシミュレータを構築してみよう.

1. コンピュータの本質はチューリングマシンである

人工知能を作ろうという願望は, からくり人形の時代からあったが, からくりというツールでは全く素朴すぎて歯が立たなかった. コンピュータの出現ではじめて人工知能を作る試みは現実的に意味を持つようになった.

コンピュータの出現は, 20 世紀の前半にさかのぼる. このころ, エルブラン, ゲーデル, チャーチ, クリーネ, ポストなど「計算」に興味を持つ数学者たちが現れた. この成果を集大成したのがチューリングであり, 1936 年前後に計算概念を数学的チューリングマシンとして定式化した.

チューリングマシンは, 入力テープと状態遷移機械から構成されている「抽象的な機械」である. チューリングマシンの一例を図 1 に示す. 入力テープは, 左右に連なる無限個のマス目から構成される. マス目は空白ないし 1 個の「アルファベット」が書き込まれる. 図 1 では, 0, 1 がアルファベットであり, b は空白(blank)を表している. 一方, 状態遷移機械は, ヘッドをもち入力テープのある位置をスキャンしている. 状態遷移機械に従って, ヘッドのある位置でアルファベットを読み書きしたり, ヘッドを左右に動かしたりできる. この状態遷移機械が入力テープへの操作を示している. 図 1 の場合でいうと, チューリングマシンをスタートさせるとまず開始状態 q_0 に入る. この状態から状態 q_1 に向かうリンクに書かれている 0/R は, 「そのときのヘッドが読み取っている入力テープのマス目に 0 と書かれていると右(Right)のマス目にヘッドを移せ」という指令を意味している. 状態 q_0 にはこれ以外にリンクはつながっていない. これ以外のことが起きる, すなわち, ヘッドがいま読んでいるマス目に 0 以外のアルファベットが書かれていたら, それに対応する指令はないので, チューリングマシンはそこで異常停止する. *x*/*y* となっているところで, $\underline{y}$ が L か R 以外の場合は, ヘッドがあるマス目に書かれているアルファベットが *x* ならば, それを *y* に書き換えることを意味している. また, 状態が二重丸表記されている q_5 という状態はこれが停止状態と呼ばれるものであり, その状態に達すると, チューリングマシンは「正常終了」する.

練習問題　図 1 のチューリングマシンは, 入力テープ上に空白にはさまれた 0^n1^n （ただし, $1 \leq n$）の左端にヘッドがあれば, 正常状態に至って停止することを確認せよ.

チューリングマシンに与えられる問題は, 上のような YES/NO 判定問題（入力テープに書かれた問題がある命題を満たす角か判定せよという問題）, 計算問題（入力テープ状のデータを有限ステップで別のデータに書き換えよという問題）である. チューリングマシンは理論的なものであり, ある問題を解くためのチューリングマシンがどのようなものであるかということよりも, 与えられた問題を解くチューリングマシンが構成可能かどうかで計算可能性を判断するための理論的な装置として使われる. すなわち, 判定問題の正解が YES であるとき, 確かに YES という停止状態に遷移して有限ステップで計算を終えるチューリングマシンを構成可能でなければ, その判定問題は計算不能, 構成可能ならば（少なくとも）弱計算可能であるという. さらに, 判定問題に対して必ず NO という停止状態に入って停止するようなチューリングマシンを構成可能ならば, その判定問題は強判定可能であるという. 計算問題の場合は, 計算不能, 弱計算可能, 強計算可能であると言われる.

チューリングマシンが偉大である理由は, 次のような点にある.

(1) 種々のバージョン:テープが複数か否か, 動作が決定的か非決定的か…などの違いがあっても計算能力（計算可能性）は変わらない.

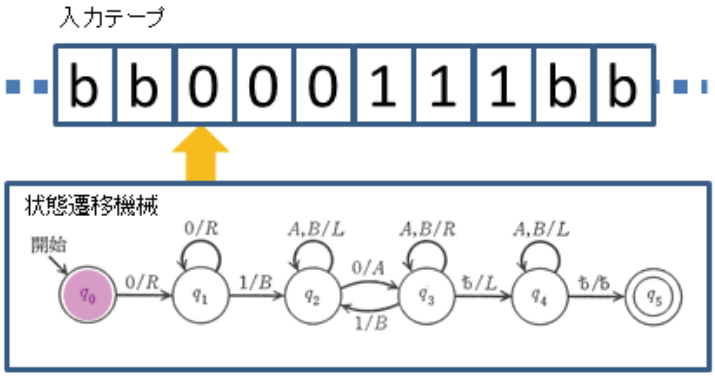


図 1: チューリングマシンの一例

(2) 言語階層とマシンの階層の対応がある.
(3) 万能チューリングマシンが存在する. 万能チューリングマシンとは, チューリングマシンの動作表と 1 対 1 対応するデータを入力テープに置いておけば, そのデータに対応するチューリングマシンの動作をシミュレートできるチューリングマシンである.
(4) アルゴリズムは, チューリングマシンの状態遷移機械の制御法に相当する.
(5) 時間計算量と空間計算量は停止までのステップ数, マス目の使用数に相当する.
(6) P≠NP?問題などさらに理論的な問題もチューリングマシンを使って定式化される.

先にも述べたように, コンピュータは本質的にはチューリングマシンを電子回路で実現したものである. 万能チューリングマシンが存在するので, その状態遷移機械の部分を電子回路さえあれば, 電子回路を変えなくてもテープの上に指定する状態遷移機械の部分を書き換えることによって, いろいろな計算を実現することができる. ただし, 実際のコンピュータは本質的にチューリングマシンと同じであっても, 見かけはかなり異なっている. コンピュータへの指示は, 状態遷移図ではなく, プログラミング言語を用いて行う.

2. Lisp インタープリタを用いたチューリングマシンのシミュレーション

チューリングマシンは論理的に定義されているので, その動きを頭の中で追ったり, 紙の上に書き出したりすることは可能であるが, 通常「計算」に要するステップ数は多く, 追跡はなかなか骨の折れる仕事である. プログラミング言語を使ってシミュレータを書くというのは, 通常の発想であるが, そのデザインには時間がかかる. Lisp はプロトタイピングに適しており, 時間をかけずに論理を追跡するのに適している.

20 世紀の人工知能研究で Lisp の果たした役割は大きい. なぜならば, Lisp は研究者が考えた人工知能のからくりの実現を容易にしただけでなく, 思考のプロセスそのものを強化したからである. 次回以降は, 今回取り上げたチューリングマシンを例にとり, いかに思考のプロセスが Lisp によって強化されるかを示す.

概ねプログラムの外部仕様を決めた後, 内部で使用するデータ構造のデザイン, アルゴリズムのデザイン, 実装, 検証の順序で進める. 大体こんなものだろうと大まかなところを決め, それを詳細化すればよい. データ構造をきちんと定義すればかなり効率的なプロトタイピングができる.

3. チューリングマシンシミュレータの外部仕様

ここで, 実現するチューリングマシンシミュレータは, チューリングマシンの仕様, ヘッドの現在位置をマークした入力テープを入力とし, シミュレーション過程を逐次出力するプログラムとする. 入

出力を気の利いたものにしようとすると，プログラムも複雑になるので，ここでは，プログラミングの平易さに重点を置き，ユーザが指定したステップ数までシミュレーションするプログラムとする．

3.1 入力テープ表現

入力テープの表現は，簡単である．ここでは，入力テープ上の左右の無限の空白列を切り落とした記号の並びを S 式で表すこととし，ヘッドが初めに置かれる位置を*で示すこととする（ヘッドはそのすぐ右隣のマス目をスキャンしている）．例えば，

(1 2 3 * 4 5 6 7)

は，ヘッドが現在スキャンしているマス目の値が 4 であり，その左のマス目にはヘッドのある位置から順に，3, 2, 1 という値が入っており，右には，4, 5, 6, 7 という値が入っていること，および，その卒側のマス目はすべて空白であることを示す．

3.2 状態遷移機械の動作表の表現

状態遷移機械の動作表を表現するためには少し複雑な S 式が必要である．その一例を図 1 に示す．これはチューリングマシンの数学的な表現に近いものであるが，これが唯一の表現方法ではなく，プログラマーが自由に設計すればよい．

4. 主要な内部変数，その意味，データ構造

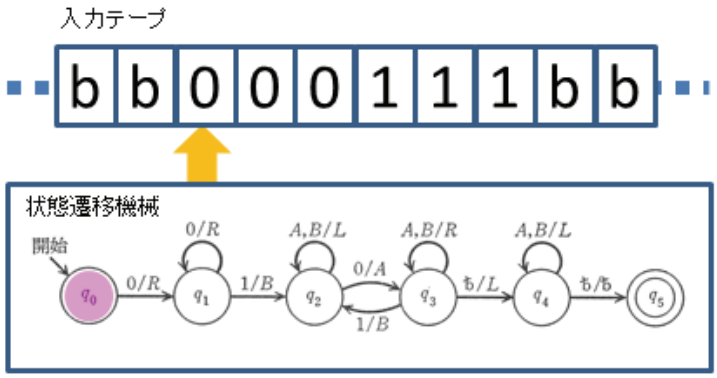
チューリングマシンの各ステップは，そのときの入力テープのマス目の内容，スキャン位置，状態遷移機械の状態によって表現される．ここで示すプログラムでは，

current-state: 現在の状態（リテラルアトム）
left-stack: 入力テープのヘッドよりも左にあるマス目の内容（右→左，リスト）
current-cell: 現在ヘッドがスキャンしているマス目の内容（リテラルアトム）
right-stack: 入力テープのヘッドよりも右にあるマス目の内容（左→右，リスト）

の 4 つの変数を用いてチューリングマシンの実行状態を管理している．また，左右に無限に続く空白については，非空白記号およびそれらに挟まれた空白記号（これは，_というリテラルアトムで表す）だけをリストに載せておくこととする．

5. シミュレーションアルゴリズム

シミュレータの本体を構成するシミュレーションアルゴリズムについては，与えられた入力を内部データ構造に変換したのちに，プログラムの核心部分，すなわち，「『チューリングマシンのヘッドがスキャンしている入力テープ上の記号と，状態遷移機械の状態が与えられたとき，状態遷移機械の動作表に基づいて入力テープに対する操作，ヘッドの動き，次状態を決定する』という作業を停止状態になるまで繰り返す」を実行すればよい．



(a) チューリングマシンの例

```
((start q0)        … 初期状態が q0 である
 (terminal q5)     … 停止状態は q5 である.
 (states           … 以下では, 各状態における状態遷移規則を表す
  (q0 (0 (Right) q1))   … 状態 q0 で, 0 を見たらヘッドを右に移動し, 状態 q1 に遷移
  (q1 (0 (Right) q1)    … 状態 q1 で, 0 を見たらヘッドを右に移動し, 状態 q1 に遷移
      (1 B q2))         … 状態 q1 で, 1 を見たら B と書き換え, 状態 q2 に遷移
  (q2 (A (Left) q2)     … 状態 q2 で, A を見たらヘッドを左に移動し, 状態 q2 に遷移
      (B (Left) q2)     … 状態 q2 で, B を見たら B と書き換え, 状態 q2 に遷移
      (0 A q3))         … 状態 q2 で, 0 を見たら A と書き換え, 状態 q3 に遷移
  (q3 (1 B q2)          … 状態 q3 で, 1 を見たら B と書き換え, 状態 q2 に遷移
      (A (Right) q3)    … 状態 q3 で, A を見たらヘッドを右に移動し, 状態 q3 に遷移
      (B (Right) q3)    … 状態 q3 で, B を見たらヘッドを右に移動し, 状態 q3 に遷移
      (_ (Left) q4))    … 状態 q3 で, 空白_を見たらヘッドを左に移動し, 状態 q4 に遷移
  (q4 (A (Left) q4)     … 状態 q4 で, A を見たらヘッドを左に移動し, 状態 q4 に遷移
      (B (Left) q4)     … 状態 q4 で, B を見たらヘッドを左に移動し, 状態 q4 に遷移
      (_ _ q5))))       … 状態 q4 で, 空白_を見たら, 状態 q5 に遷移
```

(b) 上の示したチューリングマシンの状態遷移機械の S 式表現

図 1: チューリングマシンの状態遷移機械の S 式表示例

プログラムを書き始める前に, 手作業でこの作業をたどってみよう. シミュレーションは, 入力テープに 000111 という記号が置かれ, チューリングマシンは初期状態 q0 にあり, ヘッドは左端のゼロをスキャンしているという状態から始まる. これを下のように図示する.

```
000111
↑
q0
```

動作表によれば, このとき, ヘッドを 1 コマ右に進め, 状態 q1 に遷移するので, 次は

```
000111
 ↑
```

```
q1
```

となる. 同様の作業を進めていくと,

```
000B11        0AABB1        _AAABBB
  ↑     …       ↑     …     ↑
  q2            q3          q5
```

となり, 停止状態 q5 に入って停止することがわかる. q5 は予め指定された停止状態であるので, これは「受理」停止, すなわち, このチューリングマシンが入力テープ上にある記号列は 0^n1^n というパターンをしていることを知らせてくれたことになる.

具体的なイメージができたら後は楽にプログラミングできるはずだ.