

チューリングマシン

専門演習D

チューリングマシンとは

- 1936年にアラン・チューリングによって発表された、計算機の模型の1つ。
- 現在の計算機をシンプルに考えた時に、チューリングマシンと同じになるといわれている。

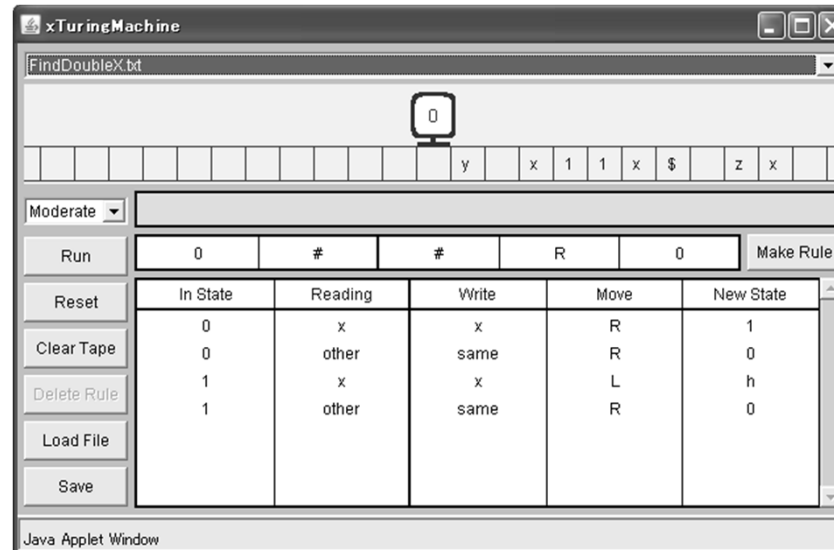
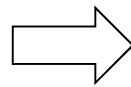


チューリングマシンの模式図
(出典: Wikipedia, チューリングマシン)

xTuringMachine

- チューリングマシンのアプレット
 - <http://math.hws.edu/TMCM/java/labs/xTuringMachineLab.html>
- 様々なサンプルがあるので、参考にしてください。

ここを変更してサンプル
プログラムを切り替える



xTuringMachineの概要

The screenshot shows the xTuringMachine Java applet interface. The window title is "xTuringMachine". The file name is "Change01toXY.bt". The tape contains the sequence: x z 0 x 1 x 1 0 0 z z. The current state is 0, and the current reading is #. The current rule is: In State 0, Reading #, Write #, Move R, New State 0. The interface includes a "Run" button, a "Step" button, a "Clear Tape" button, a "Delete Rule" button, a "Load File" button, and a "Save" button. The "Directions" are set to "L" and "R".

チューリングマシーン

テープ

プログラムの実行

プログラムの1行実行

テープの全消去

選択しているルールの消去

プログラム、テープの読み込み

プログラム、テープの保存

ルール(プログラム)

In State	Reading	Write	Move	New State
0	#	#	R	0
0	\$	\$	R	h
0	0	x	R	0
0	1	y	R	0
0	x	x	R	0
0	y	y	R	0
0	z	z	R	0

チューリングマシンの動作

チューリングマシンの状態

チューリングマシンが読み取る記号

状態0で記号xを読み込んだ時のルール

In State: 現在状態

Reading: 読んだ記号

ルールは、

Write: 現在のテープの場所に書き込む記号

Move: 次に移動する方向 (R or L)

New State: 次のチューリングマシンの状態

Directions: L R

In State	Reading	Write	Move	New State
0	#	#	R	0
0	\$	\$	R	h
0	0	x	R	0
0	1	y	R	0
0	x	x	R	0
0	y	y	R	0
0	z	z	R	0

状態h は、Halt(ハルト)と呼び、プログラムの停止を表す。

サンプルプログラムの概要 (1)

- CountInBinary (2進数のインクリメント)
 - 2進数の数字の右端からスタートする。
 - 0を読み込んだ時は1を書き出し、1を読み込んだ時は0を書き出すことで、2進数のインクリメントを実現している。
 - このプログラムは停止しない。

サンプルプログラムの概要 (2)

- CopyXYZ (xyz文字列のコピー)
 - xyz文字の左側からチューリングマシンは実行する。
 - 1文字を読み込んで、右側に移動し、最初にある文字の右側に読み込んだ文字を書きだす。
 - すべての文字に対して実行すると、最初にある文字と同じ文字列が右側に出来上がる。

サンプルプログラムの概要 (3)

- BinaryAddition (2進数の足し算)
 - 2進数の2つの数字の右端からスタートする。
 - 右側の数値の右端を読み込み、左側の数値の右端に足す。その結果、1になるところはy、0になるところはxと書き込む(すでに演算済みの桁を保持するため)。このときに、右側の読み込んだ数字は消す。
 - 右側で足すときに、桁上げする場合は、先のCountInBinaryサンプルのように1つインクリメントするプログラムを実施する。
 - 右側の数値がすべてなくなると、左側の数値のxを0、yを1に変えて2進数の数字に直す。

アプレットにあるサンプルプログラムの概要

- Change01toXY
 - 0, 1をx, yに書き換える。
- FindDoubleX
 - 連続するxを見つける。
- CopyXYZ
 - xyz文字列をコピーする。
- Increment
 - 2進数の数値に1を加える。
- AddBinaryNumbers
 - 2進数の足し算を行う。
- MultiplyByAdding
 - 2進数の掛け算を行う。
左の数値を右の数値の回数だけ足し合わせる
ことによって掛け算を
実現している。

ルールの作り方

ここでルールを作成し、「Make Rule」ボタンを押すと下に登録される。

States: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24					
0	#	#	R	0	Make Rule
In State	Reading	Write	Move	New State	

ルールのそれぞれの箇所は、左から、
In State: 現在の状態
Reading: 読み取った記号
Write: 書き込む記号
Move: 移動方向
New State: 次の状態

ルールの変更

条件部分の変更

変更したい条件を
セットし、実行部分
を作成する。

The screenshot shows a rule editor interface. At the top, there is a 'Symbols:' row with characters: #, \$, 0, 1, x, y, z, and a note '(#= blank)'. Below this is a row of five input fields containing '0', '#', '0', 'R', and '0'. To the right of these fields is a 'Replace' button. Below the input fields is a table with five columns: 'In State', 'Reading', 'Write', 'Move', and 'New State'. The first row of the table contains '0', '#', '#', 'R', and '0'. The second row contains '0', '\$', '\$', 'R', and '0'. A thick black arrow points from the left to the '0' in the first input field. Another thick black arrow points from the bottom to the 'Replace' button.

Replaceボタンを押
すと変更される。

実行部分の変更

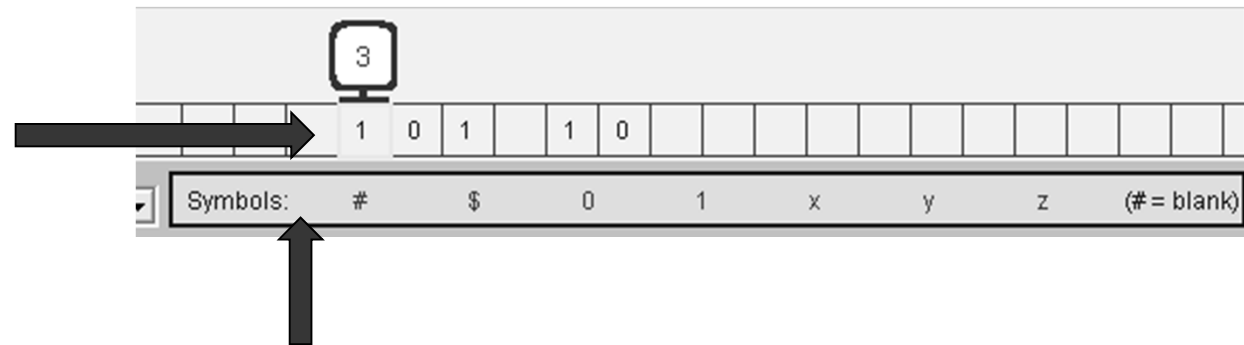
変更したい記号を
クリックする。

This screenshot shows the same rule editor interface as above. The 'Make Rule' button is now visible to the right of the input fields. The 'Write' field in the table below contains '#'. A thick black arrow points from the left to the '0' in the first input field. Another thick black arrow points from the bottom to the '#' in the 'Write' field.

変更したい箇所をクリックする。

テープの作り方

変更したいテープの箇所をクリックする



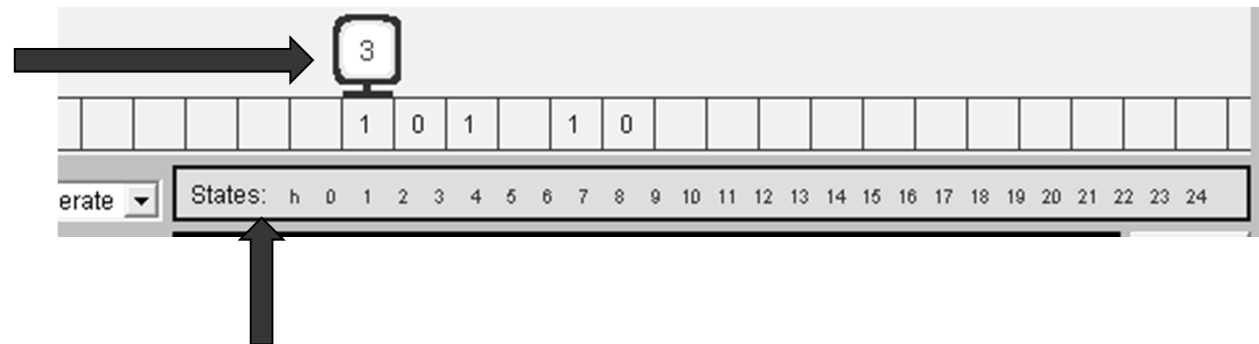
変更したい記号をクリックする。#は空白文字を表す。

テープは、矢印キーでも移動可能である。また、ドラッグ・アンド・ドロップでも左右に移動する。

実行準備

- チューリングマシンの位置、状態を決定する。

ドラッグ・アンド・ドロップで移動

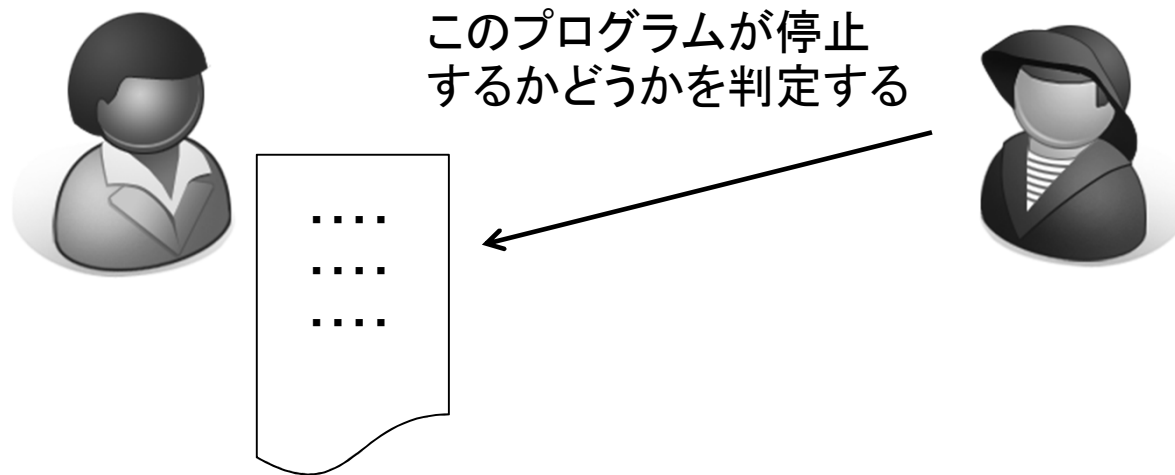


チューリングマシンをクリックして選択したあとで、
指定したい状態をクリックする。

初期位置、状態を決定したら、「Run」または「Step」で実行する。

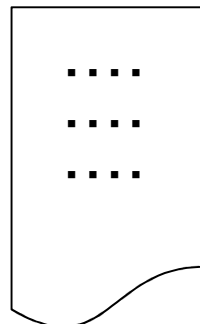
停止性について (1)

- チューリングマシンのプログラムを入力として、そのプログラムが停止するかどうかを判定するプログラムを作成することは、できない。

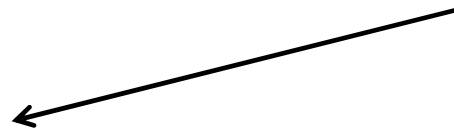


停止性について (2)

- 停止することを判定できると仮定した場合
 - 読み込んだプログラムが停止、ならば、自分は無限ループ
 - 読み込んだプログラムが停止しない、ならば、自分はプログラムが終了する
- として、自分自身を読み込むと、矛盾する。



このプログラムが停止
するかどうかを判定する



```
int main()
{
  if(プログラム == 停止){
    while(1);
  }
  return 0;
}
```

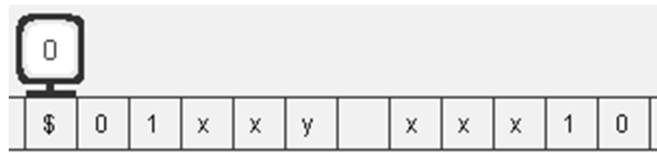
まとめ

- チューリングマシンは、現在のコンピュータ(ソフトウェア)のシンプルな模型(モデル)と呼べる。
- 実際に、低級言語のアセンブラなどでは、チューリングマシンと同じようなプログラミングが必要になる。
- Java, Cなどの高級言語であっても、プログラムを作成するための考え方は、同じである。

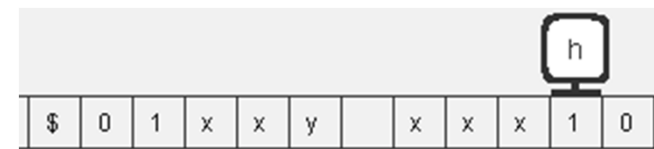
課題題

[課題2] 連続する3つのxを発見する

- テープの左端から移動を開始し, 3連続するx文字を発見するチューリングマシンを作成してください。



実行前(例)



実行後(例)

[課題3] \$文字を3倍にする

- テープには\$以外の文字は書かれていないという前提で、\$文字を3倍にするチューリングマシンを作成してください。例えば、\$\$とテープに書かれていた場合、\$\$\$\$\$\$\$(6個)とテープに書かれた状態でHaltすればよいです。



実行前(例)



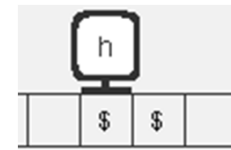
実行後(例)

[課題4] \$文字を3で割る

- テープには\$以外の文字は書かれていないという前提で、\$文字を3分の1にするチューリングマシンを作成してください。3で割り切れない時は、あまりは無視してください。例えば、\$\$\$\$\$\$\$(7個)とテープに書かれていた場合、\$\$ (2個)とテープに書かれた状態でHaltすればよいです。結果の\$は、最初の\$とテープの同じ場所にある必要はありません。



実行前(例)



実行後(例)

[課題5] 数字分の\$を書く

- テープには2進数の数値が書かれています。チューリングマシンはその右端の上にあります。そして、その数値分の\$を書くルールを作成してください。例えば、101とテープに書かれていた場合、\$\$\$\$\$(5個)とテープに書かれた状態でHaltすればよいです。



実行前(例)



実行後(例)

[課題6] xとyを分離する

- テープにはxとyのみで構成された文字列が書かれています。チューリングマシンはその左端の上にあります。そこからxとyの文字を分離するルールを作成してください。例えば、xyxyyyxyxxという文字がテープにある場合、xxxxxyyyyという文字が作成されてHaltすればよいです。



実行前(例)



実行後(例)