Turing Machines

Agenda

- Turing Machines
  - Alan Turing
  - Motivation
    - Church-Turing Thesis
  - Definitions
    - Computation
    - TM Configuration
    - Recognizers vs. Deciders

Alan Turing

Alan Turing was one of the founding fathers of CS.

- His computer model –the Turing Machine– was inspiration/premonition of the electronic computer that came two decades later
- Was instrumental in cracking the Nazi Enigma cryptosystem in WWII
- Invented the “Turing Test” used in AI
- Legacy: The Turing Award. Pre-eminent award in Theoretical CS

A Thinking Machine

First Goal of Turing's Machine: A model that can compute anything that a human can compute. Before invention of electronic computers the term "computer" actually referred to a *person* who's line of work is to calculate numerical quantities!

As this is a philosophical endeavor, it can't really be proved.

Turing's Thesis: Any "algorithm" can be carried out by one of his machines

A Thinking Machine

Second Goal of Turing's Machine: A model that's so simple, that can actually prove interesting epistemological results. Eyed Hilbert's 10^{th} problem, as well as a computational analog of Gödel's Incompleteness Theorem in Logic.

Philosophy notwithstanding, Turing's programs for cracking the Enigma cryptosystem prove that he really was a true hacker! Turing's machine is actually easily programmable, if you really get into it. Not practically useful, though…

A Thinking Machine

Imagine a super-organized, obsessive-compulsive human computer. The computer wants to avoid mistakes so everything written down is completely specified one letter/number at a time. The computer follows a finite set of rules which are referred to every time another symbol is written down. Rules are such that at any given time, only one rule is active so no ambiguity can arise. Each rule activates another rule depending on what letter/number is currently read, EG:

A Thinking Machine
EG Successor Program

Sample Rules:

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

Let's see how they are carried out on a piece of paper that contains the *reverse* binary representation of 47:

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

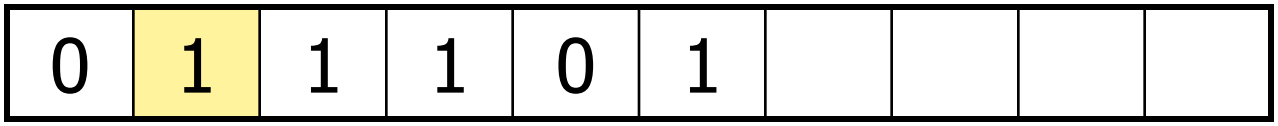
1	1	1	1	0	1				

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

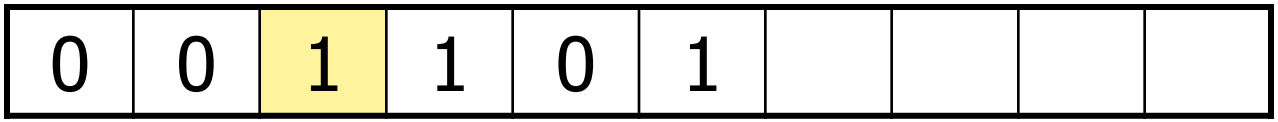


A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

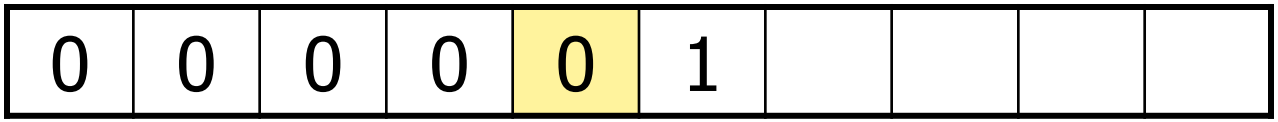
0	0	0	1	0	1				

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

0	0	0	0	1	1				

A Thinking Machine
EG Successor Program

So the successor's output on 111101 was 000011 which is the reverse binary representation of 48.

Similarly, the successor of 127 should be 128:

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

1	1	1	1	1	1	1			

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

0	1	1	1	1	1	1			

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

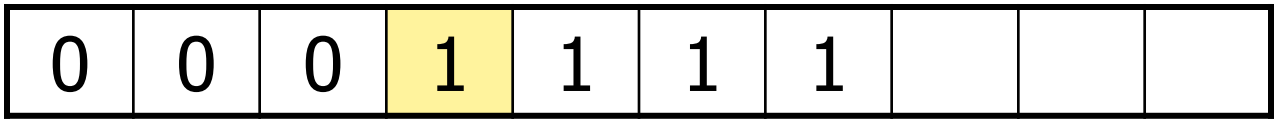
0	0	1	1	1	1	1			

A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

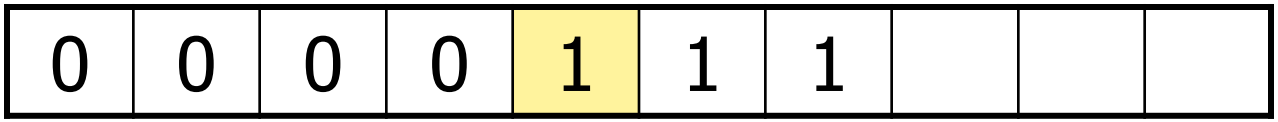


A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

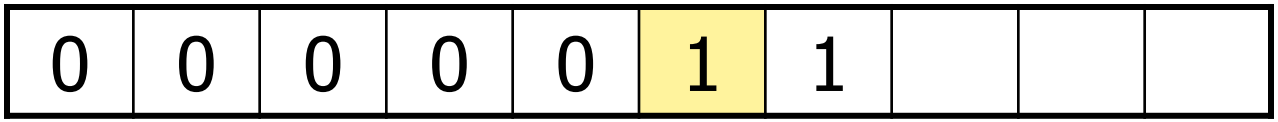


A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

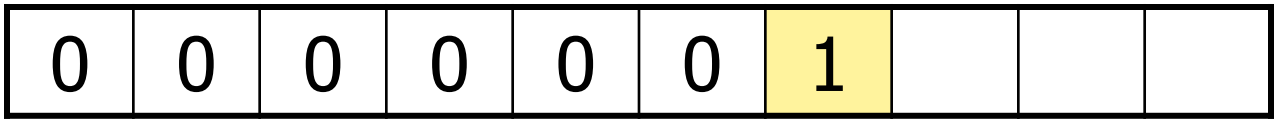


A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

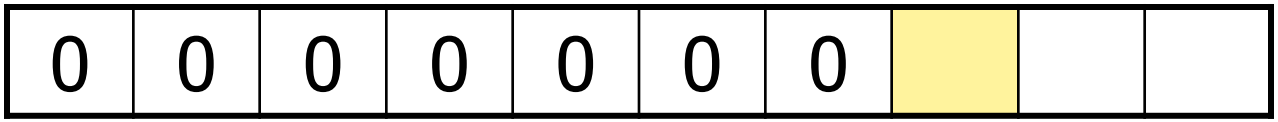


A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!



A Thinking Machine
EG Successor Program

If read 1, write 0, go right, repeat.

If read 0, write 1, HALT!

If read ; write 1, HALT!

0	0	0	0	0	0	0	1		

A Thinking Machine

It was hard for the ancients to believe that *any* algorithm could be carried out on such a device. For us, it's much easier to believe, especially if you've programmed in assembly!

However, ancients did finally believe Turing when Church's lambda-calculus paradigm (on which lisp programming is based) proved equivalent!

Turing Machines

A Turing Machine (**TM**) is a device with a finite amount of *read-only* "*hard*" memory (states), and an unbounded[1] amount of read/write tape-memory. There is no separate input. Rather, the input is assumed to reside on the tape at the time when the TM starts running.

Just as with Automata, TM's can either be input/output machines (compare with Finite State Transducers), or yes/no decision machines. Start with yes/no machines.

Comparison with Previous Models

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA			
PDA			
TM			

Comparison with Previous Models

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA			
TM			

Comparison with Previous Models

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM			

Comparison with Previous Models

Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM	No	1-way infinite tape. 1 cell access per step.	Yes (but will also allow crashes)

Turing Machine
Decision Machine Example

First example (adding 1 bit to reverse binary string) was basically something that a Finite Transducer could have achieved (except when there's overflow). Let's give an example from next step up in language hierarchy.

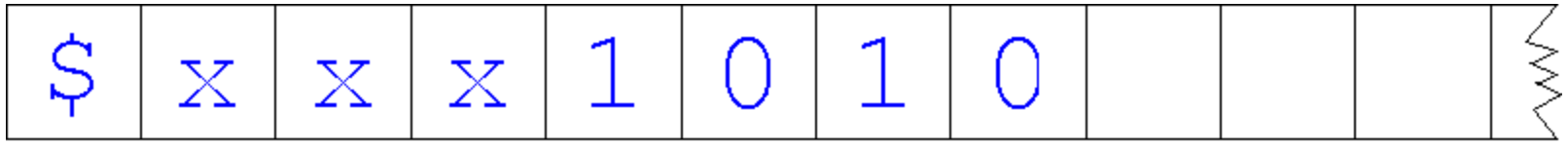
{bit-strings with same number of 0's as 1's}

–a context free language:

Turing Machine Decision Machine Example

This is a “true” Turing machine as:

- Tape is semi-infinite (indicated by torn cell):



- Input is prepared at beginning of tape
- No intrinsic way to detect left tape end
  - similar to empty stack detection problem for PDA’s
  - similar trick used –introduce $ as the end symbol
- All rules must include a move direction (R/L)
- Situations that can’t happen aren’t dealt with (technically under-deterministic)

Turing Machine
Decision Machine Example

{bit-strings with same number of 0's as 1's}:

Pseudocode:

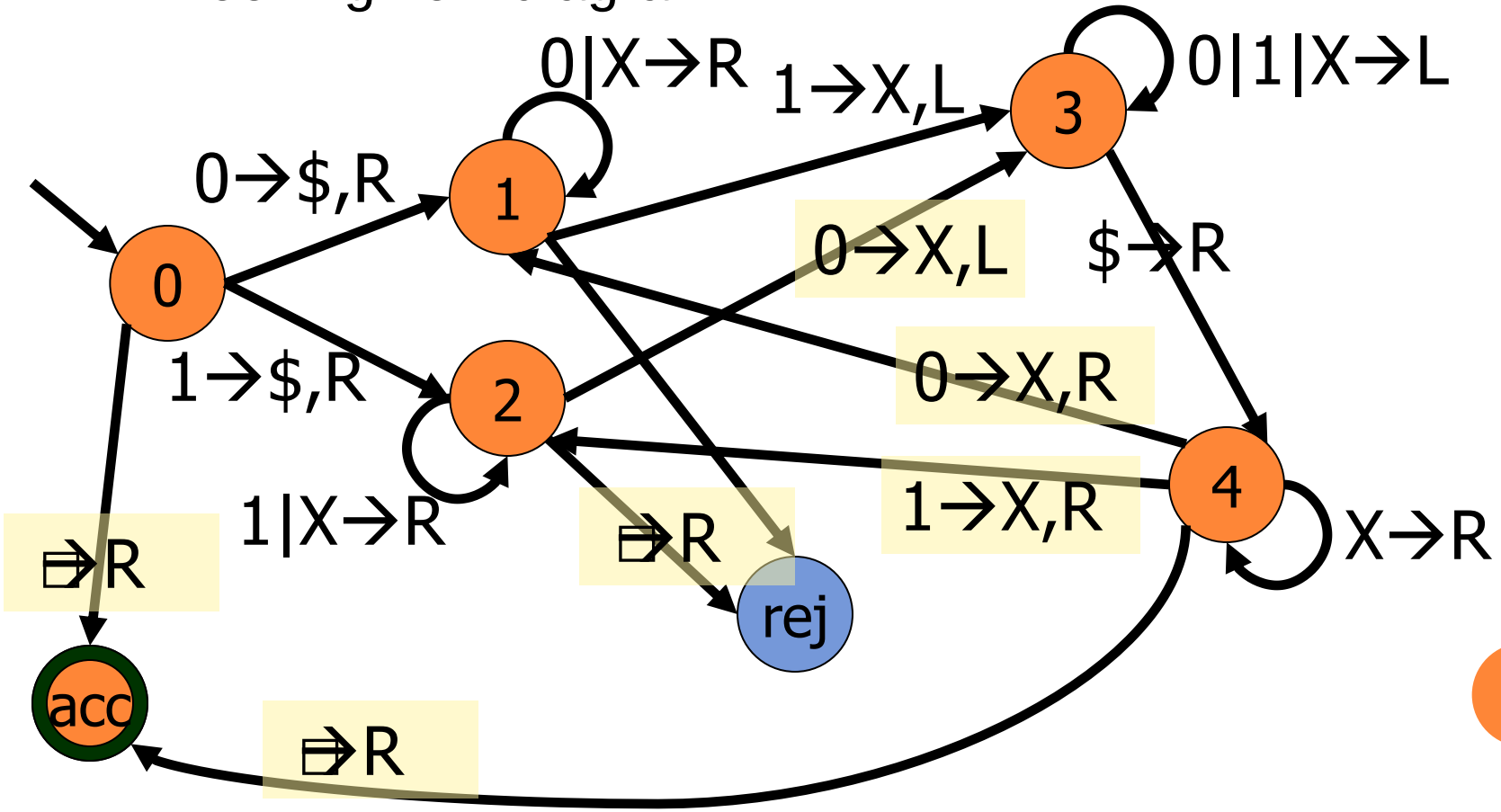
```
while (there is a 0 and a 1)
  cross these out
if (everything crossed out)
  accept
else
  reject
```

TM Example
Instructions Set

0. if read ␣ go right (*dummy move*), ACCEPT
   if read 0, write $, go right, goto 1 // $ detects start of tape
   if read 1, write $, go right, goto 2
1. if read ␣ go right, REJECT
   if read 0 or X, go right, repeat (= *goto* 1) // look for a 1
   if read 1, write X, go left, goto 3
2. if read ␣ go right, REJECT
   if read 1 or X, go right, repeat // look for a 0
   if read 0, write X, go left, goto 3
3. if read $, go right, goto 4 // look for start of tape
   else, go left, repeat
4. if read 0, write X, go right, goto 1 // similar to step 0
   if read 1, write X, go right, goto 2
   if read X, go right, repeat
   if read ␣ go right, ACCEPT

TM Example
State Diagram

These instructions can be expressed by a familiar looking flow diagram:



TM Transition Notation

An edge from the state *p* to the state *q* labeled by …

- *a*→*b*,D means if in state *p* and tape head reading *a*, replace *a* by *b* and move in the direction D, and into state *q*
- *a*→D means if in state *p* and tape head reading *a*, don't change *a* and move in the direction D, and into state *q*
- *a*|*b*|…|*z* → … means that given that the tape head is reading any of the pipe separated symbols, take same action on any of the symbols

TM Configuration Notation

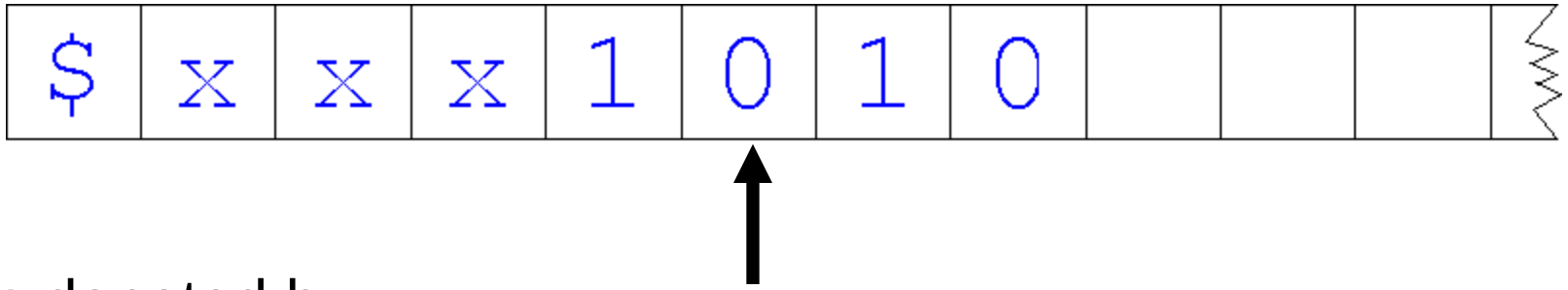
A TM's next action is completely determined by current state and symbol read, so can predict all of future actions if know:

1. current state
2. current tape contents
3. current position of TM's reading "head"

Handy notation lists all of these in a single string. A symbol representing current state, is sandwiched between content of tape to left of head, and content of tape to right (including tape head). The part of tape which is blank ad-infinitum is ignored.

TM Configuration Notation

For example



Is denoted by:

Reading rule 3

$xxx1$q_3$010

TM Example
Crazy Web-Page

The following link shows how the example machine accepts 01101010 and how the tape configuration notation changes step by step.

TM Formal Definition
Static Picture

DEF: A ***Turing machine*** (TM) consists of a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$. Q, Σ, and q_0, are the same as for an FA. Γ is the ***tape alphabet*** which necessarily contains the blank symbol ␣ as well as the input alphabet Σ. δ is as follows:

$$\delta : (Q - \{q_{acc}, q_{rej}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

Therefore given a non-halt state p, and a tape symbol x, $\delta(p,x) = (q,y,D)$ means that TM goes into state q, replaces x by y, and the tape head moves in direction D.

TM Dynamic Picture

A string *x* is ***accepted*** by *M* if after being put on the tape with the Turing machine head set to the left-most position, and letting *M* run, *M* eventually enters the accept state. In this case *w* is an element of *L*(*M*) –the language accepted by *M.*

We can formalize this notion as follows:

TM Formal Definition Dynamic Picture

Suppose TM's configuration at time *t* is given by *uapxv* where *p* is the current state, *ua* is what's to the left of the head, *x* is what's being read, and *v* is what's to the right of the head.

If $\delta(p,x) = (q,y,\mathrm{R})$ then write:

$$uapxv \Rightarrow uaypv$$

With resulting configuration *uaypv* at time *t*+1. If, $\delta(p,x) = (q,y,\mathrm{L})$ instead, then write:

$$uapxv \Rightarrow upayv$$

There are also two special cases:

- head is forging new ground –pad with the blank symbol •
- head is stuck at left end –by def. head stays put (only case)

"⇒" is read as "***yields***"

TM Formal Definition Dynamic Picture

As with context free grammars, one can consider the reflexive, transitive closure "$\Rightarrow^*$" of "$\Rightarrow$". I.e. this is the relation between strings recursively defined by:

- if $u = v$ then $u \Rightarrow^* v$
- if $u \Rightarrow v$ then $u \Rightarrow^* v$
- if $u \Rightarrow^* v$ and $v \Rightarrow^* w$, then $u \Rightarrow^* w$

"$\Rightarrow^*$" is read as "***computes to***"

A string x is said to be ***accepted*** by M if the *start configuration* $q_0\, x$ computes to some *accepting configuration* y –i.e., a configuration containing q_{acc}.

The ***language accepted by*** M is the set of all accepted strings. I.e:

$$L(M) = \{\, x \in \Sigma^* \mid \exists \text{ accepting config. } y,\ q_0\, x \Rightarrow^* y \,\}$$

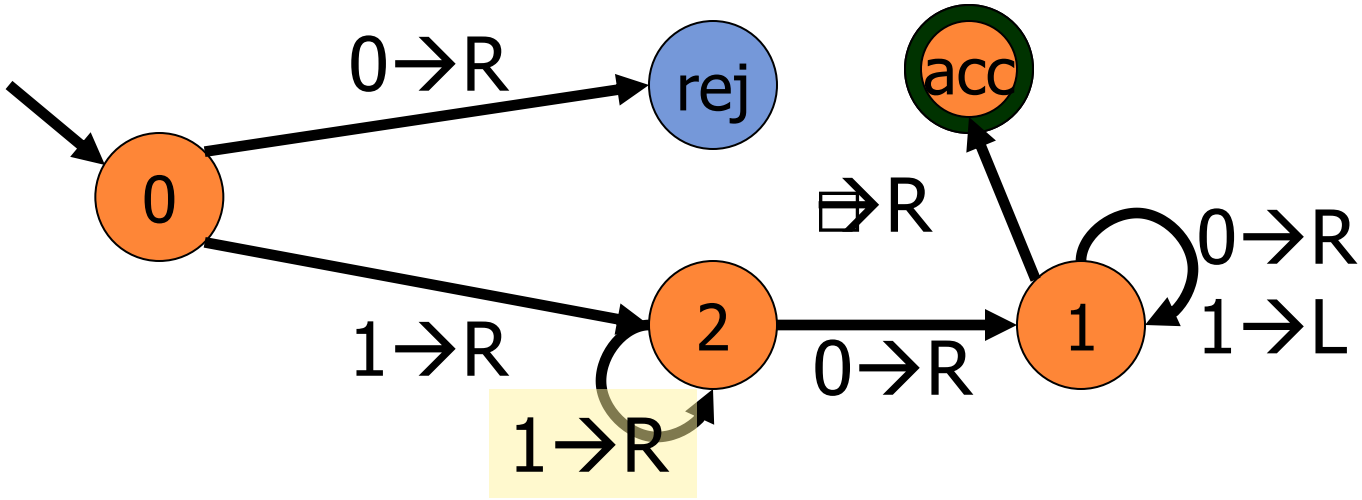
TM Acceptors vs. Deciders

Three possibilities occur on a given input w :

1. The TM M eventually enters q_{acc} and therefore halts and accepts. ($w \in L(M)$)
2. The TM M eventually enters q_{rej} *or* crashes somewhere. M ***rejects*** w . ($w \notin L(M)$)
3. Neither occurs! I.e., M never halts its computation and is caught up in an ***infinite loop***, never reaching q_{acc} or q_{rej}. In this case w is neither accepted nor rejected. However, any string not explicitly accepted is considered to be outside the accepted language. ($w \notin L(M)$)

TM Acceptors vs. Deciders

Any Turing Machines is said to be a ***recognizer*** and ***recognizes*** *L*(*M*); if in addition, *M never* enters an infinite loop, *M* is called a ***decider*** and is said to ***decide*** *L*(*M*).

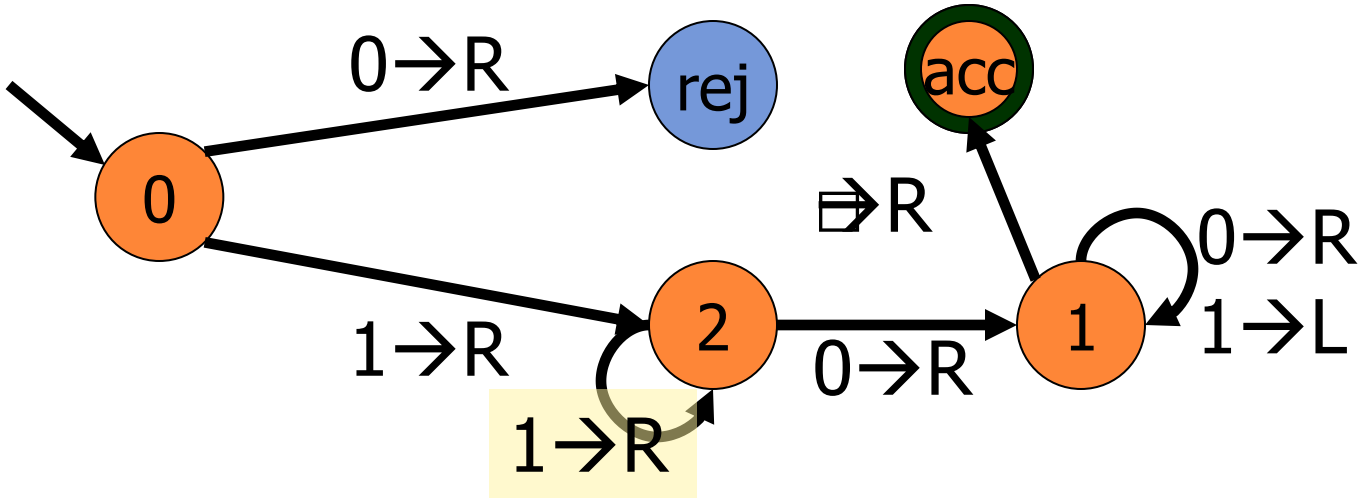


Q: Is the above *M* an recognizer? A decider? What is *L*(*M*)?

TM Acceptors vs. Deciders

A: *M* is an recognizer but not a decider because 101 causes an infinite loop.

$L(M) = 1^+ 0^+$



Q: Is *L*(*M*) decid***able*** ?

TM Acceptors vs. Deciders

A: Yes. All regular languages are decidable because can always convert a DFA into a TM without infinite loops.

Constructive Example

Here's a document showing how modular design can help you write down a TM decider for $\{a^n b^n c^n\}$. The example is non-context free.