Get Ready...

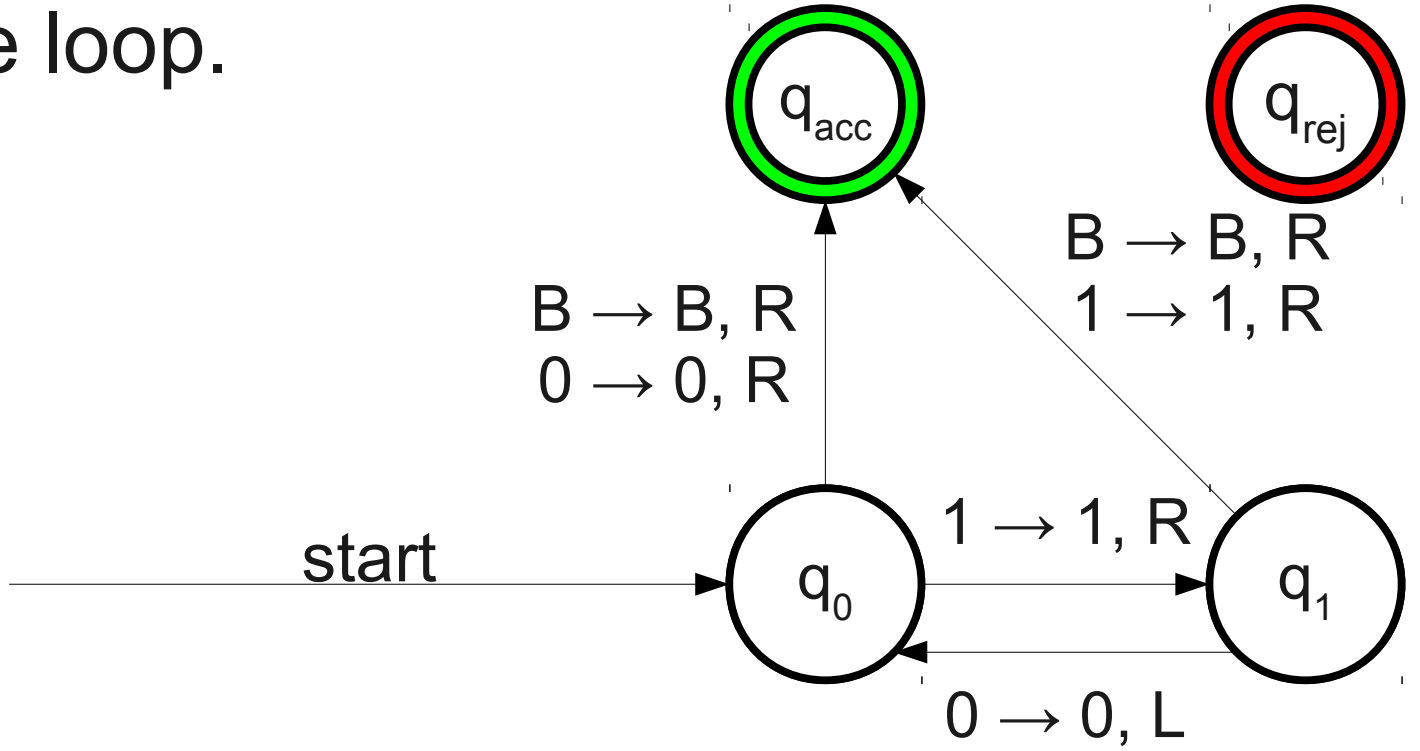
Announcements

- Problem Set 7 due Friday at 2:15 PM.
  - Stop by OH with questions!
  - Email **cs103@cs.stanford.edu** with questions!
  - Can submit using a late day before this Saturday at 2:15PM.

The **Limits** of Turing Machines

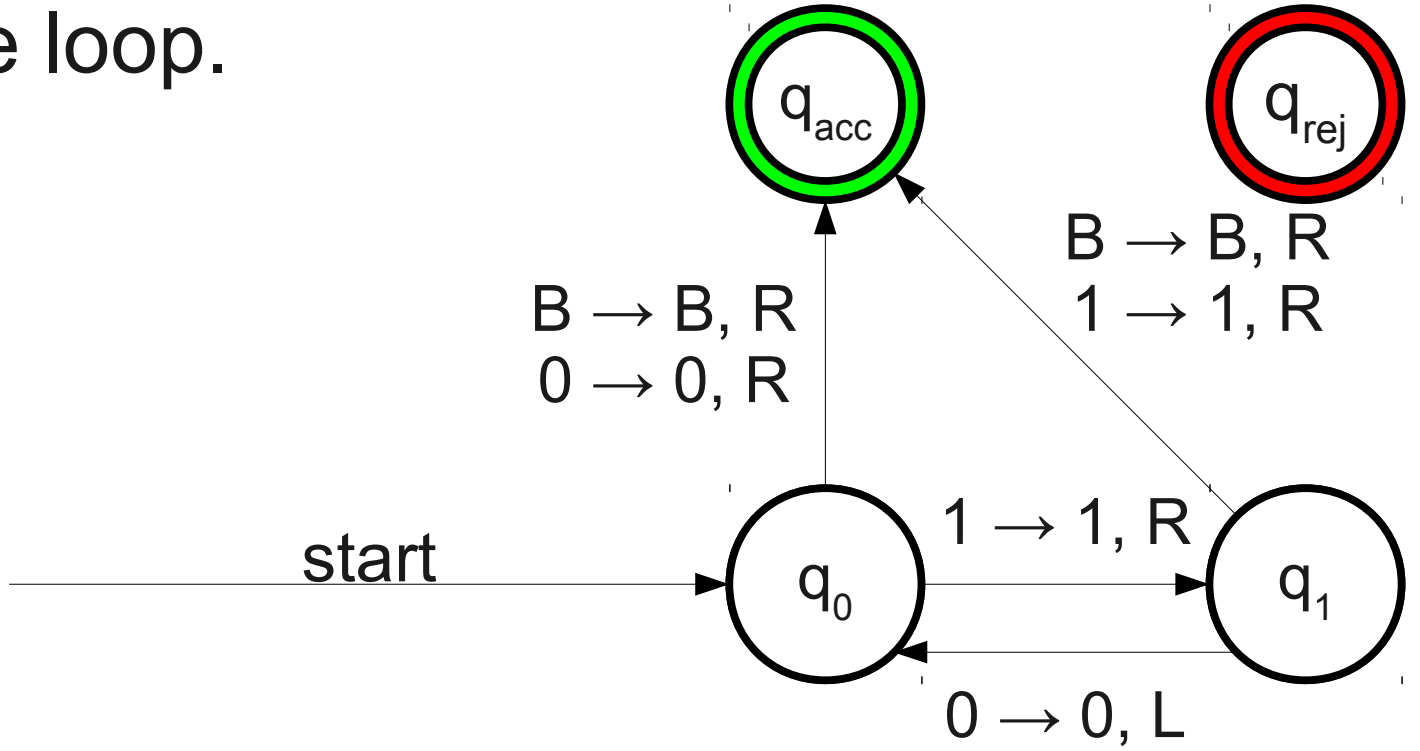
An Infinite Loop

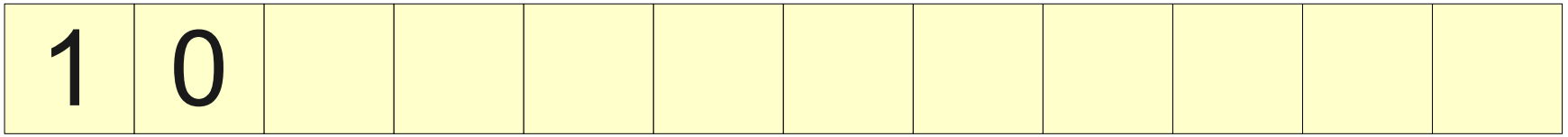
- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



An Infinite Loop

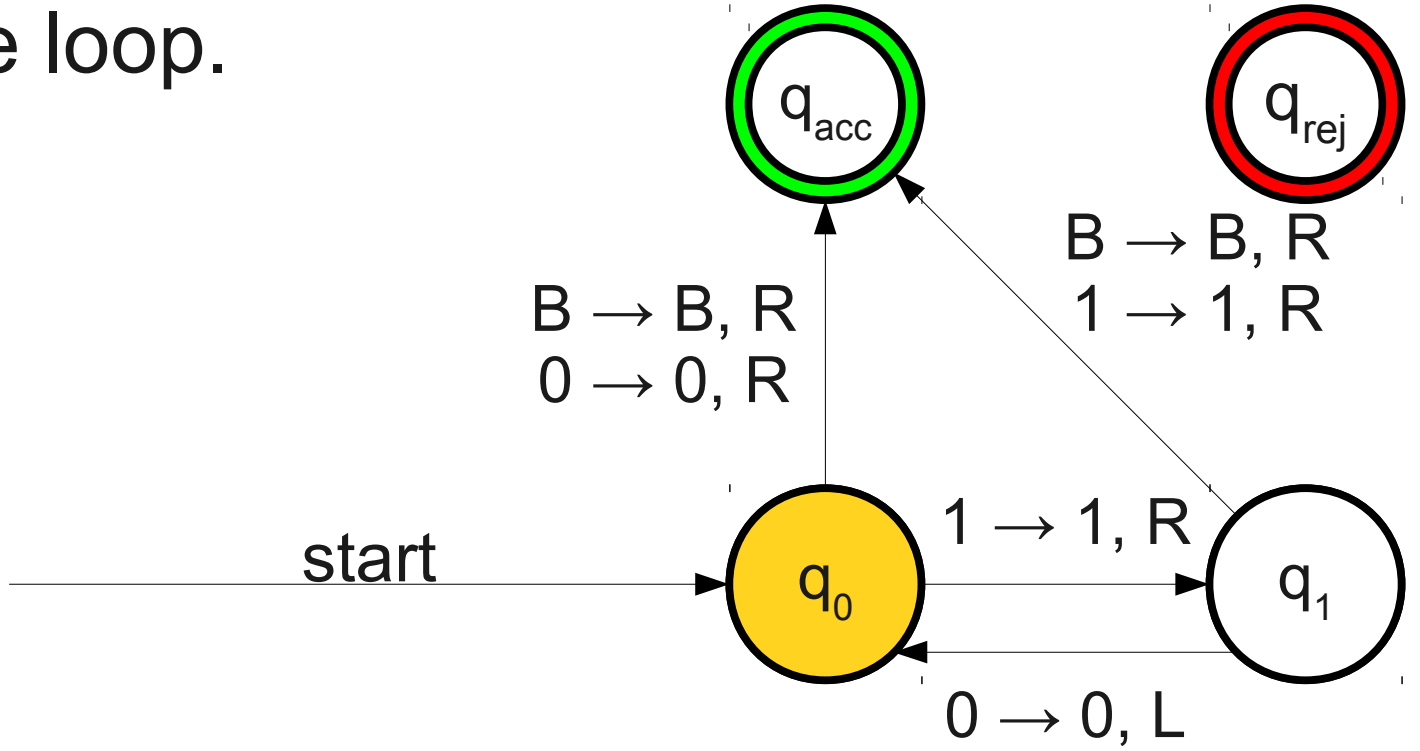
- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.





An Infinite Loop

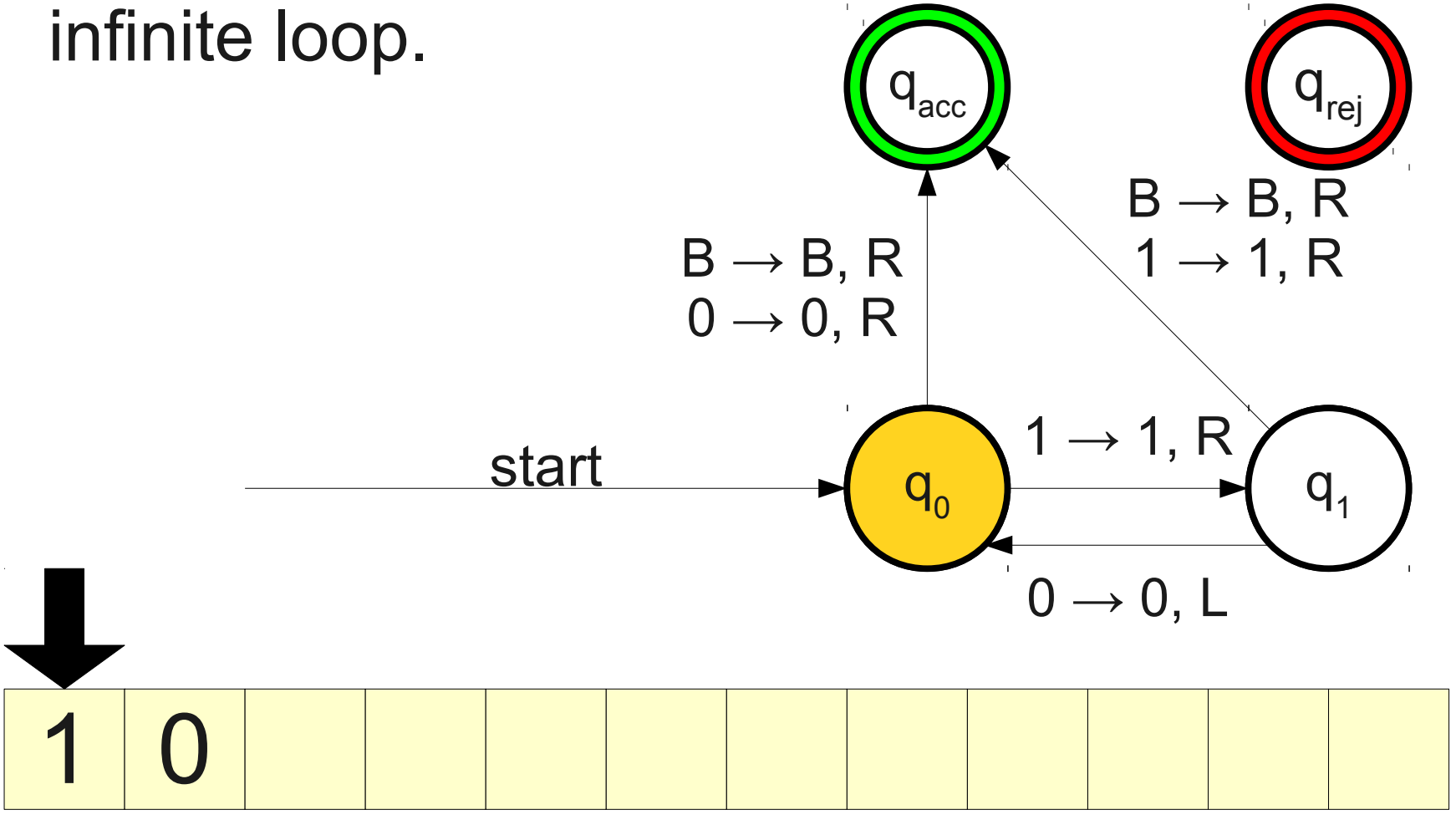
- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



1	0										

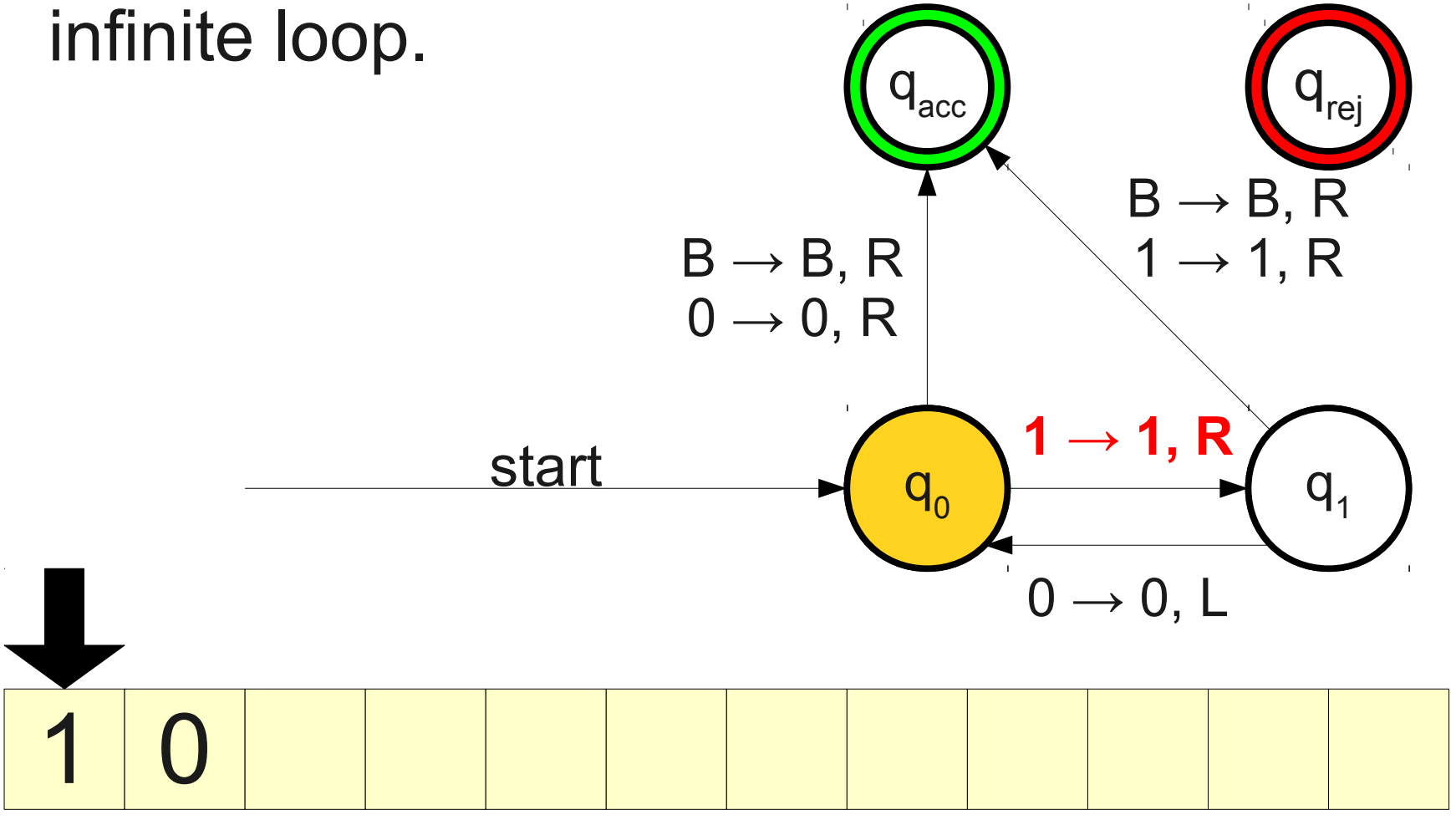
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



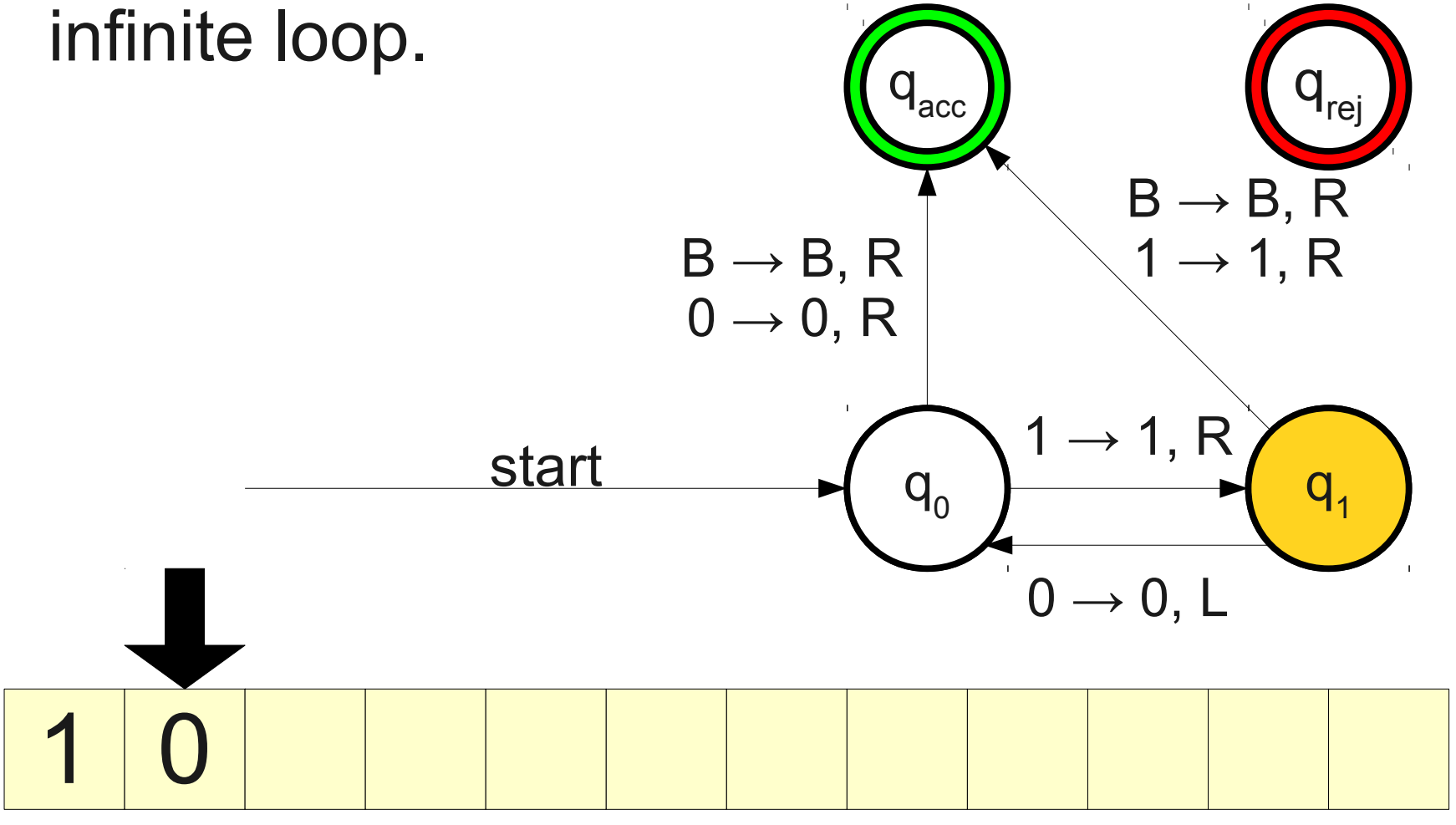
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



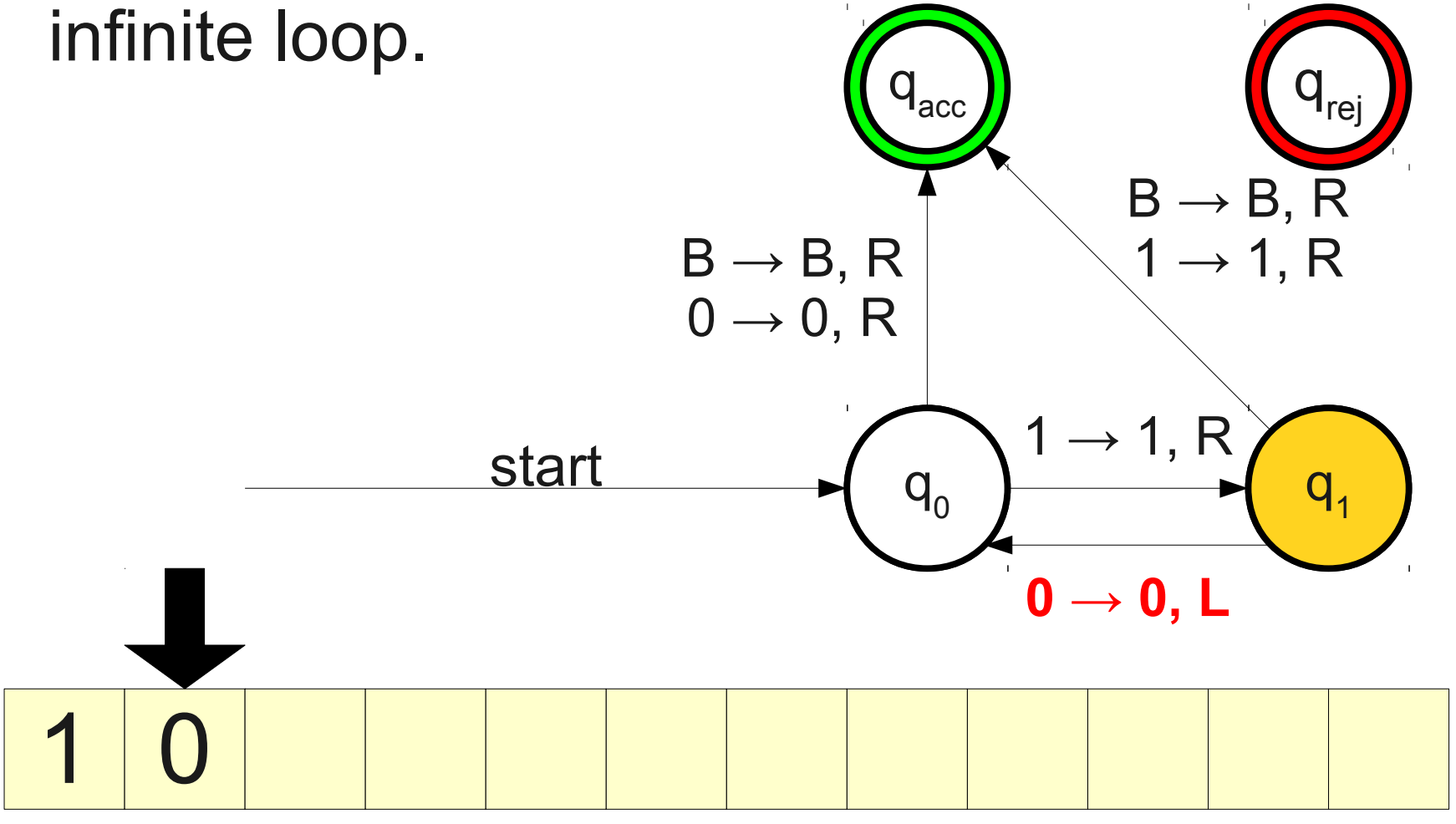
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



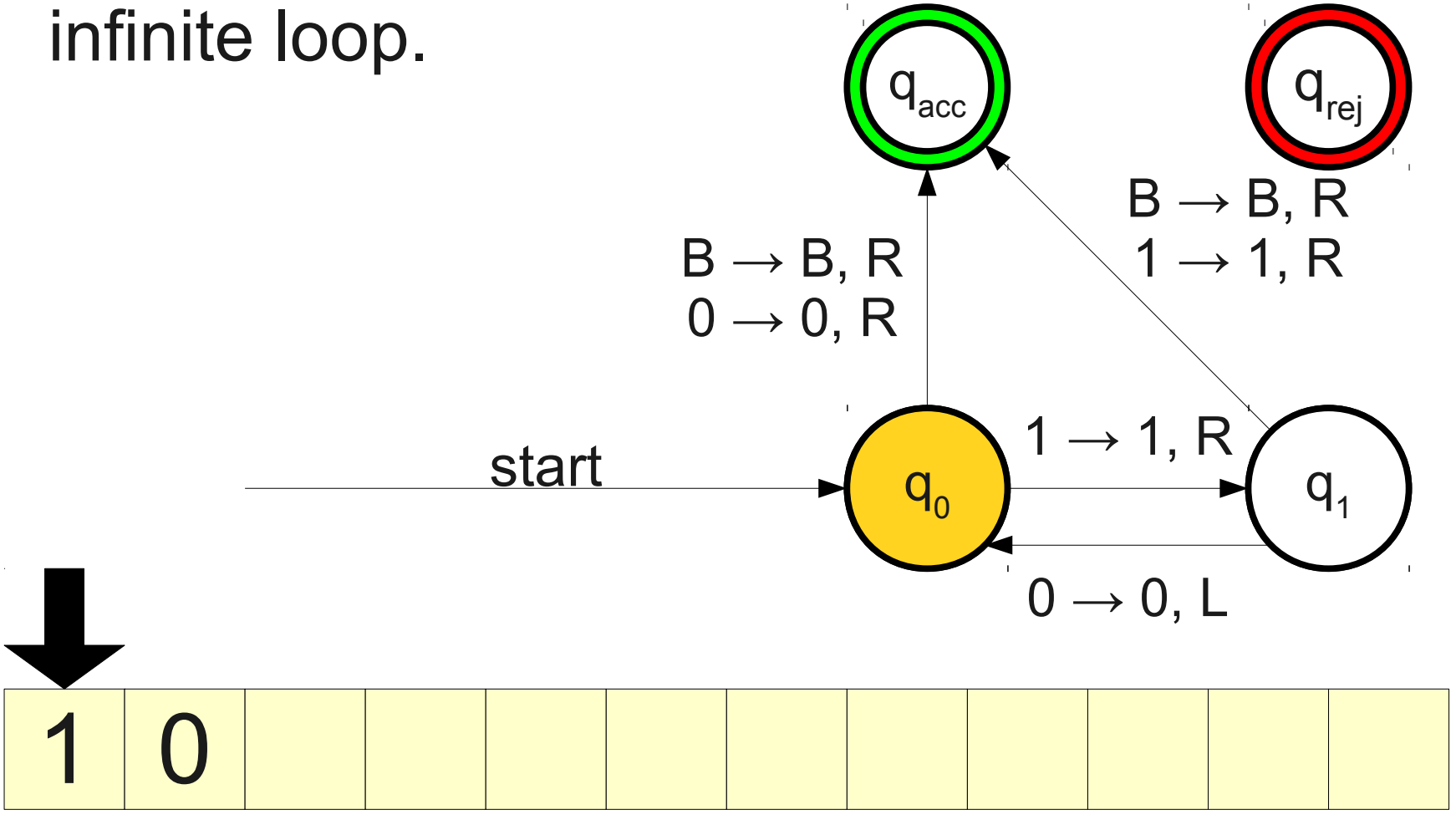
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



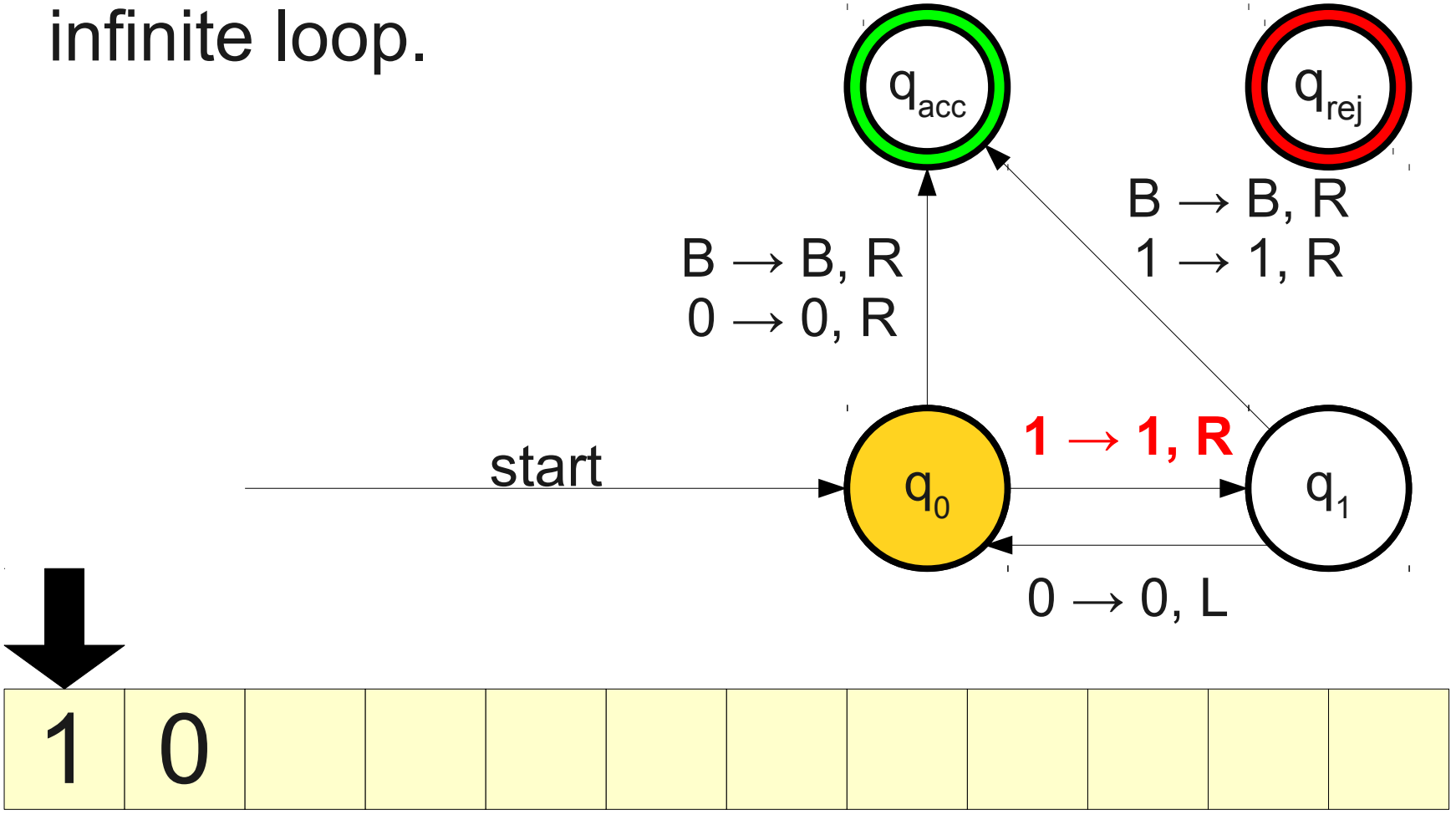
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



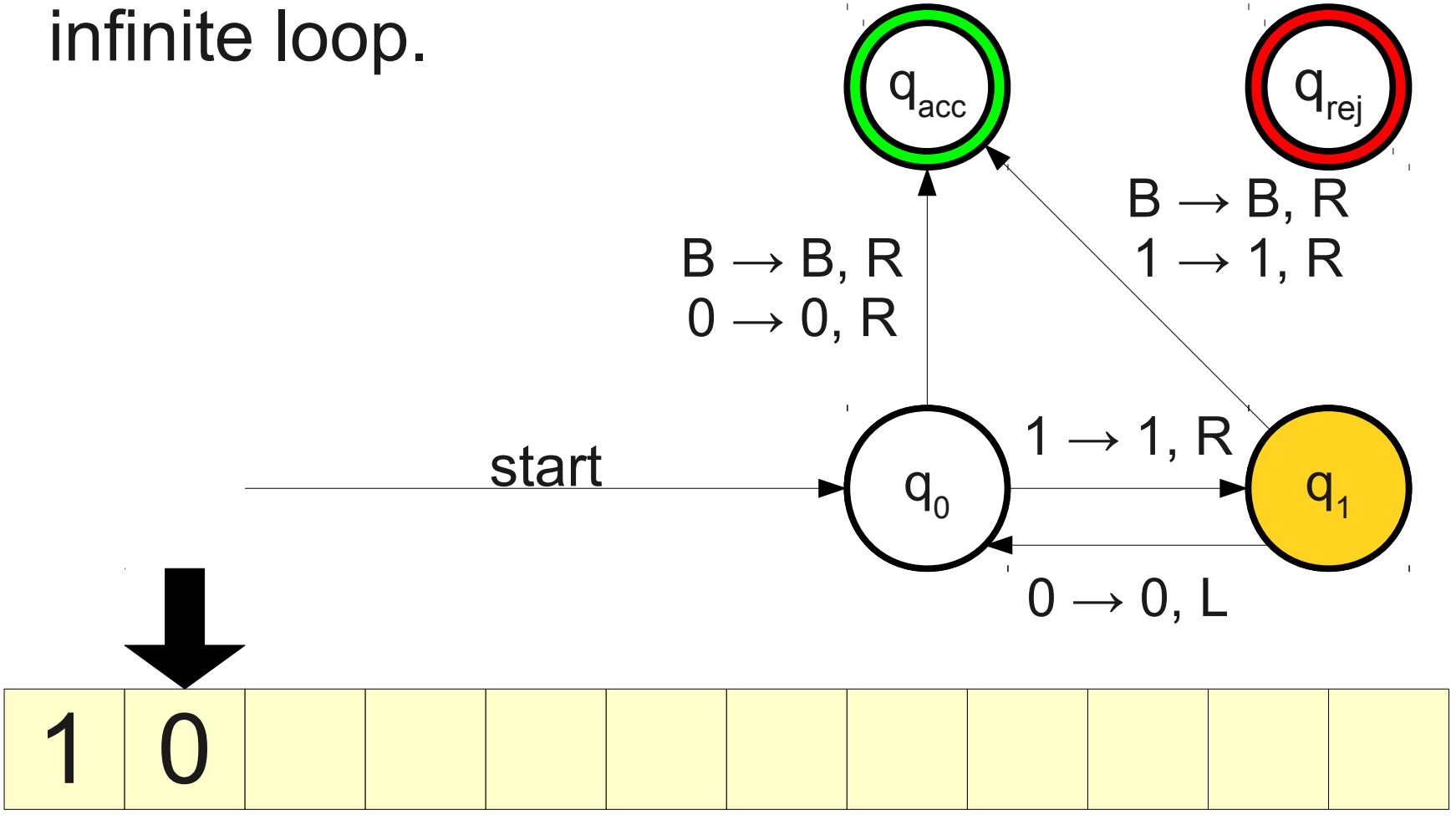
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



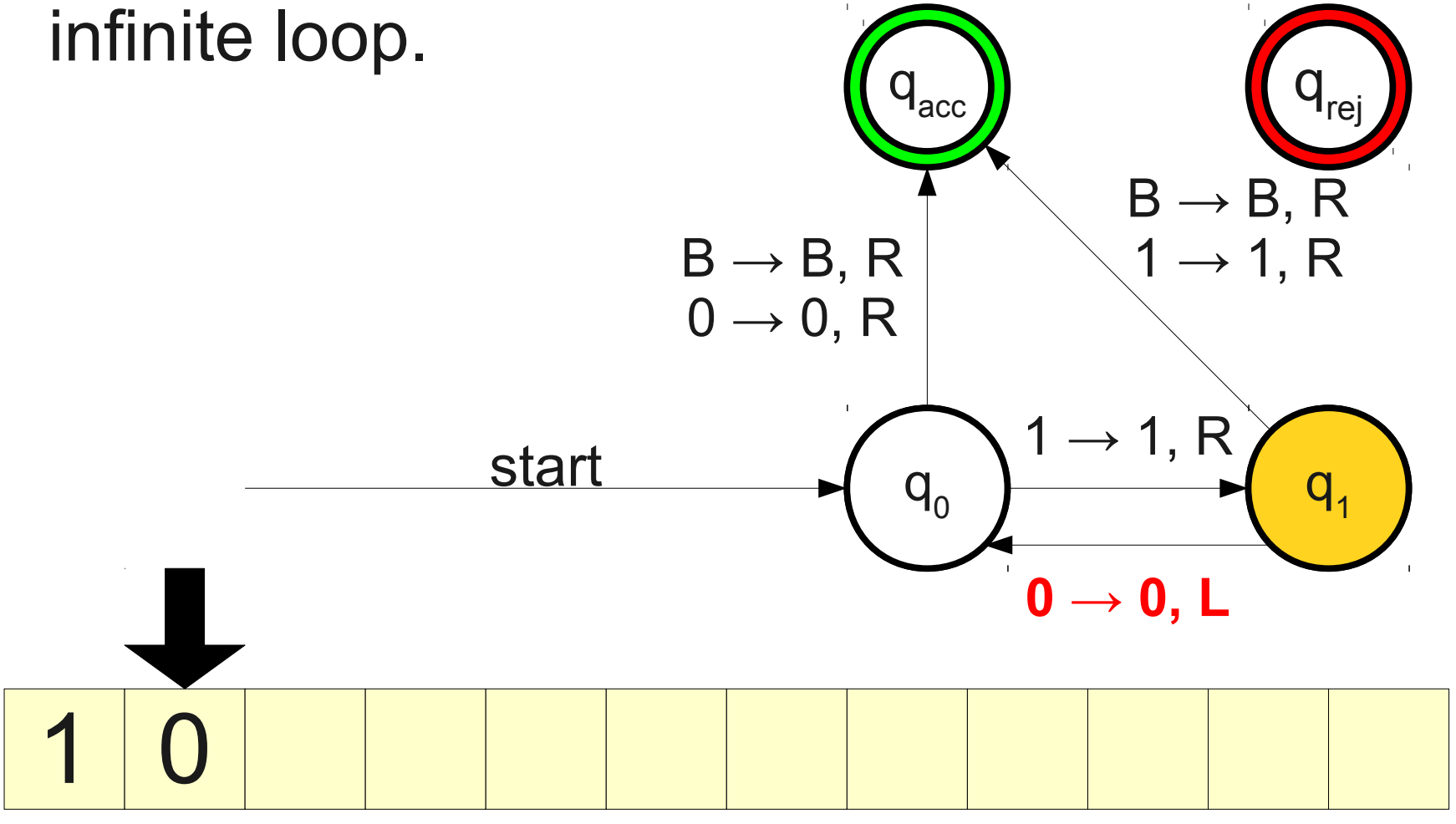
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



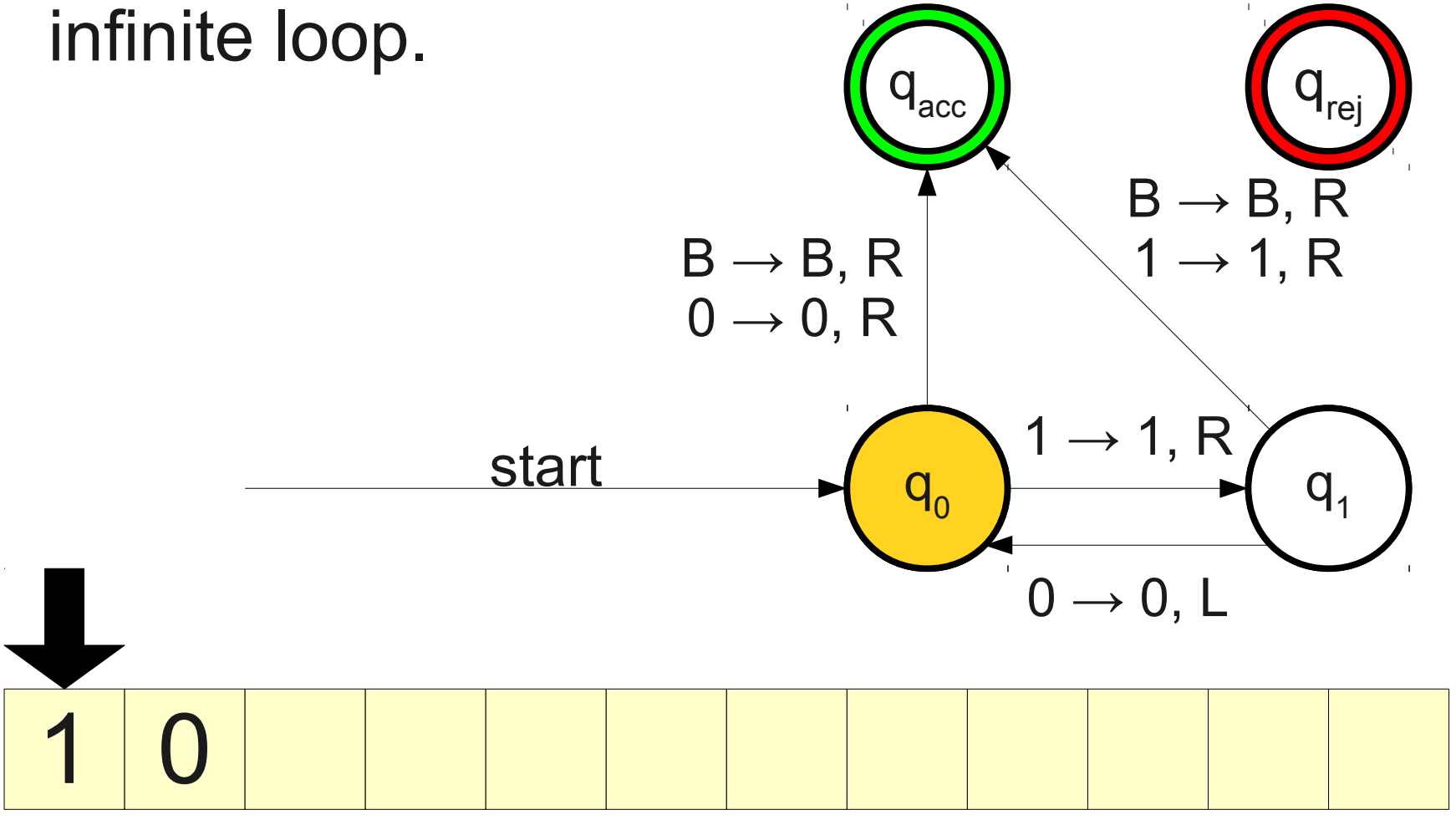
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



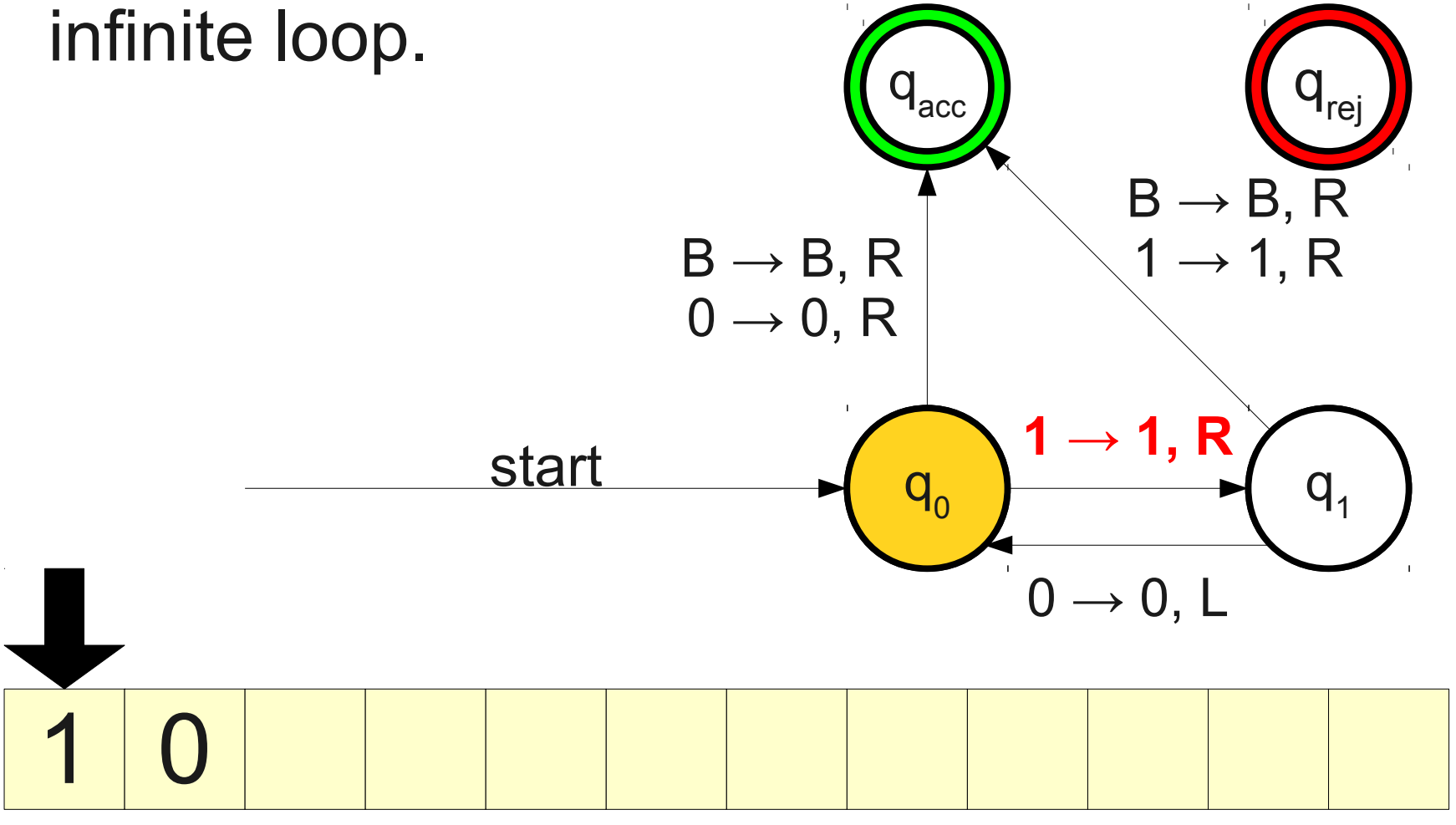
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



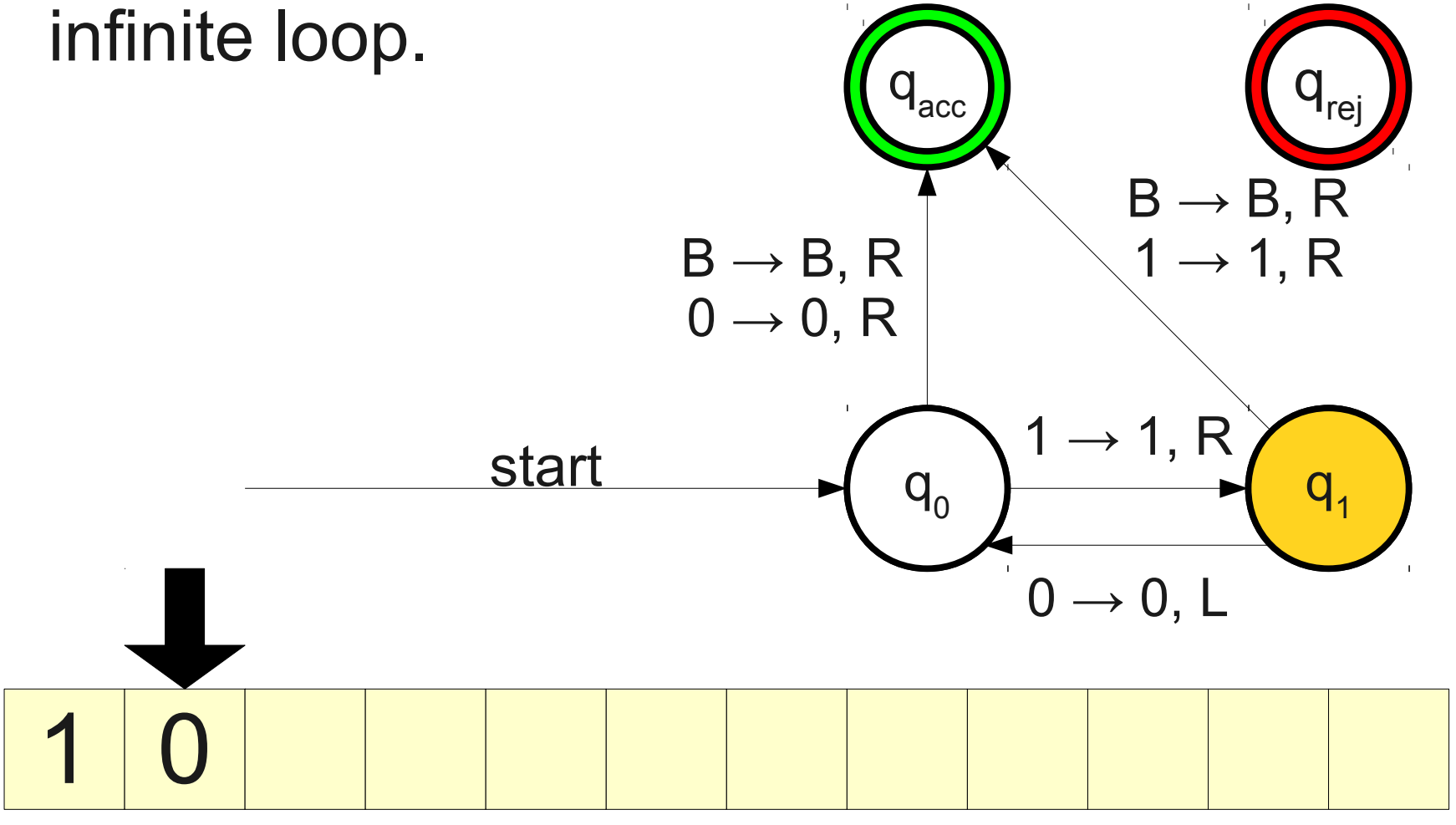
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



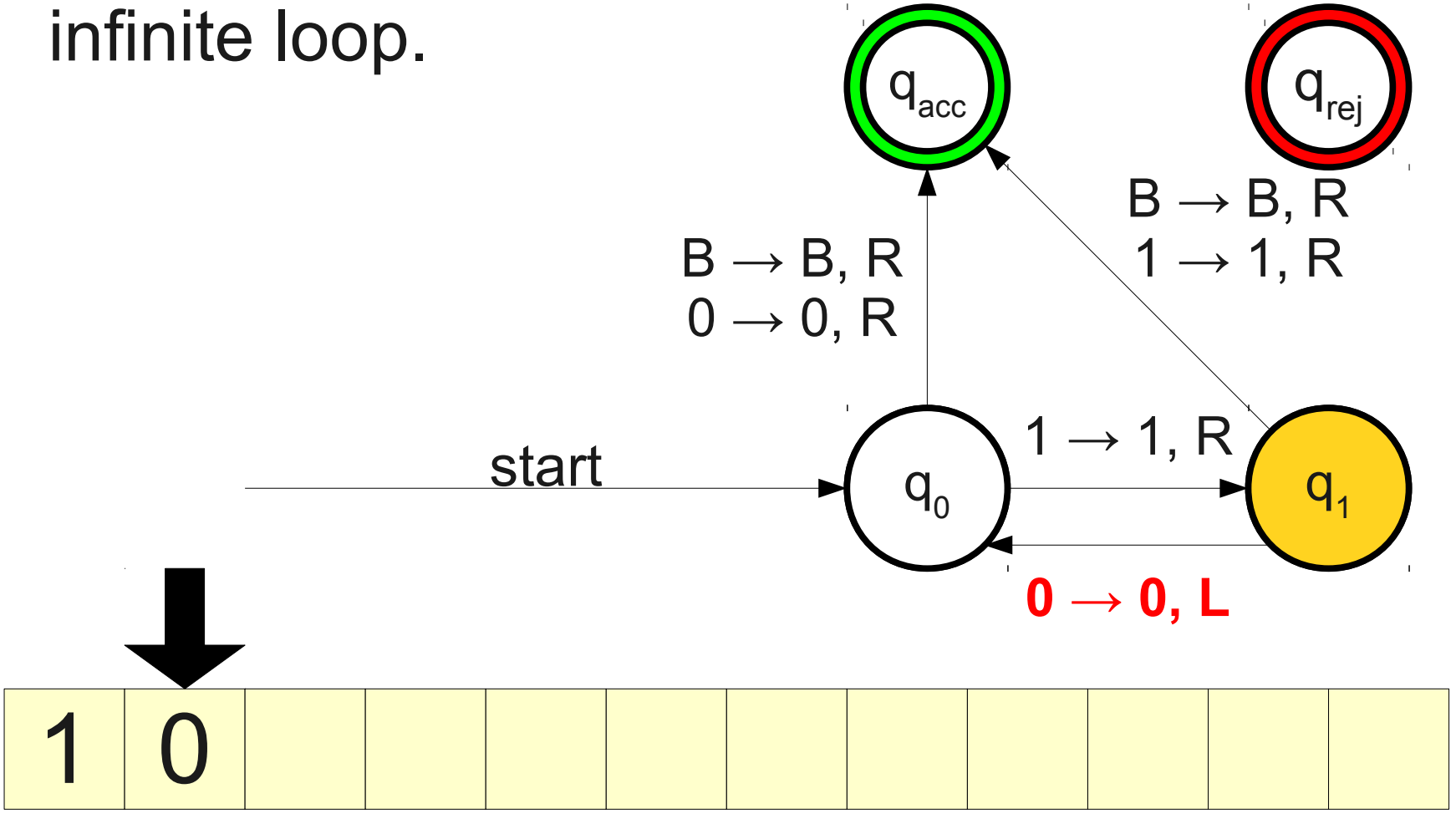
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



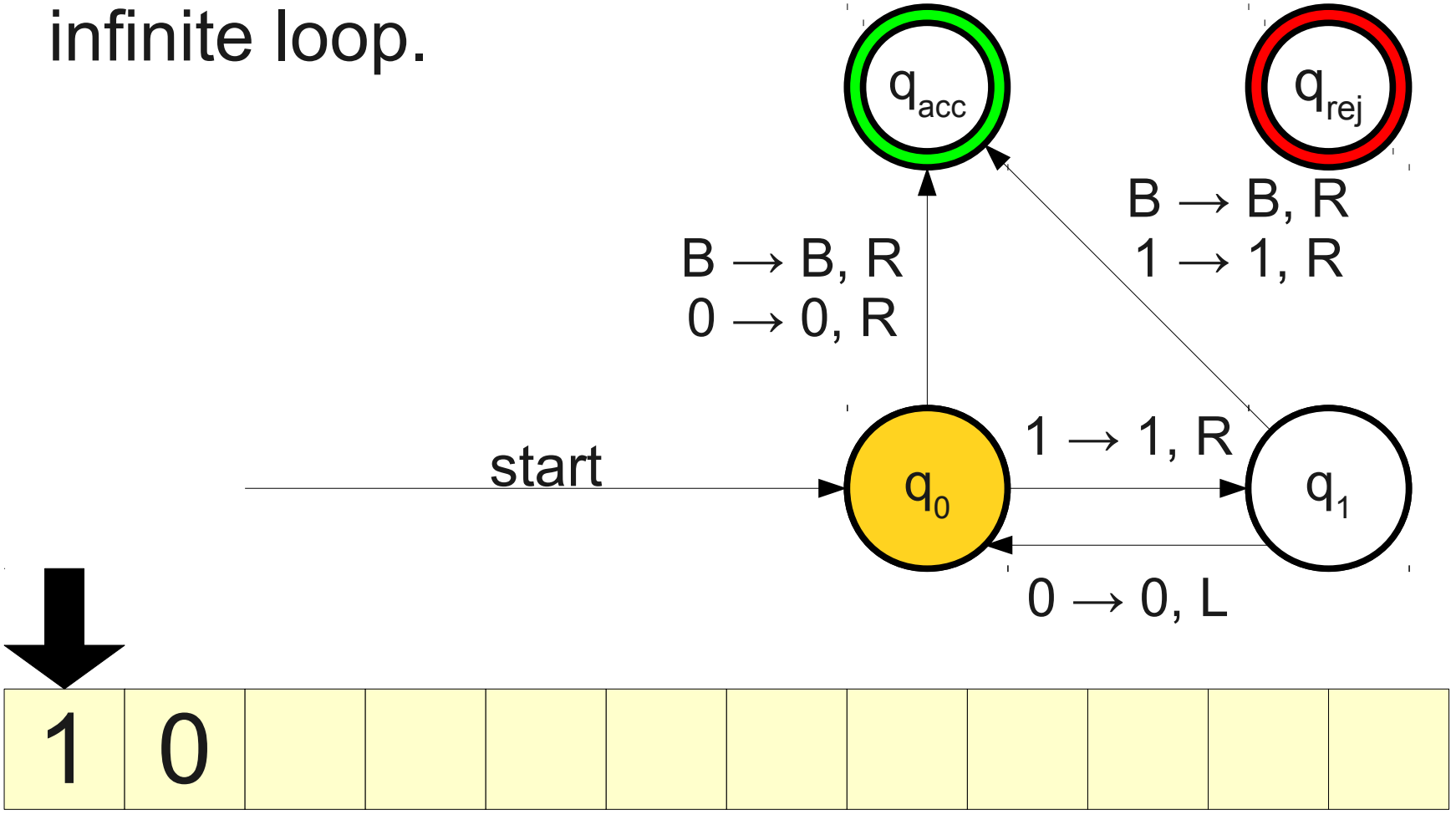
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



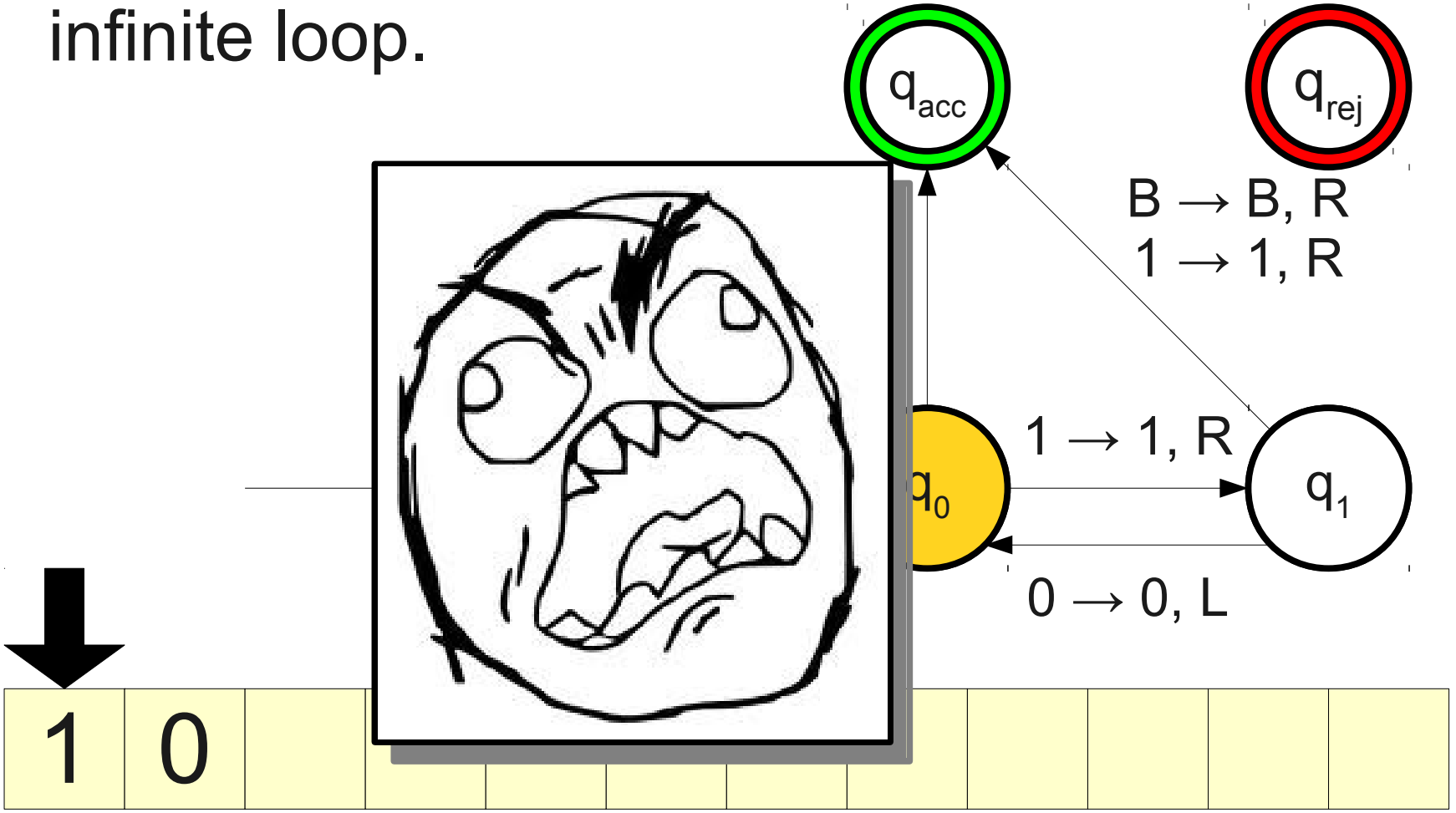
An Infinite Loop

- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



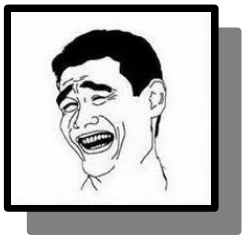
An Infinite Loop

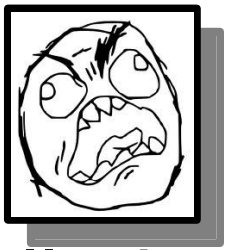
- Unlike the other automata we've seen so far, Turing machines can sometimes enter an infinite loop.



Decidability and Recognizability

- When a TM M is run on a string *w*, it can do one of three things:
  - **accept** by entering q_{acc},
  - **reject** by entering q_{rej}, or
  - **loop forever**, never accepting or rejecting.

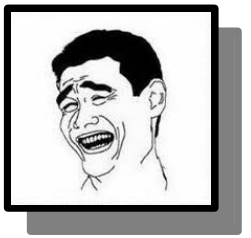


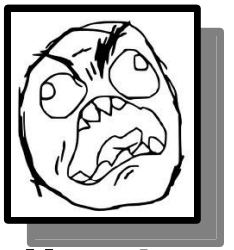


- A TM that always accepts or rejects is called a **decider**.

Decidability and Recognizability

- When a TM M is run on a string *w*, it can do one of three things:
  - **accept** by entering q_{acc},
  - **reject** by entering q_{rej}, or
  - **loop forever**, never accepting or rejecting.





- A TM that always accepts or rejects is called a **decider**.

An Unrecognizable Language

Computing over Strings

- The decision problems we have considered so far are all **language questions** of deciding whether a string is contained within some language.
- Strings can be used to encode other objects:
  - Numbers
  - Paths in graphs
  - Objects being distributed
- Can we encode **Turing machines** as strings?

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - its set of states,
  - its alphabet,
  - its tape alphabet,
  - its transition table,
  - its start state,
  - its accepting state,
  - its rejecting state,
  - the blank symbol

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - its set of states,
  - **its alphabet,**
  - **its tape alphabet,**
  - its transition table,
  - its start state,
  - its accepting state,
  - its rejecting state,
  - **the blank symbol**

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - its set of states,
  - **its alphabet,**
  - **its tape alphabet,**
  - its transition table,
  - its start state,
  - its accepting state,
  - its rejecting state,
  - **the blank symbol**

Let

$\Sigma = \{ s_1, s_2, s_3, \ldots, s_n \}$

Let

$\Gamma = \{ s_1, s_2, \ldots, s_n, s_{n+1}, \ldots, s_{n+m} \}$

Such that $B = s_{n+m}$

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - its set of states,
  - its transition table,
  - its start state,
  - its accepting state,
  - its rejecting state,

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - **its set of states,**
  - its transition table,
  - **its start state,**
  - **its accepting state,**
  - **its rejecting state,**

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - **its set of states,**
  - its transition table,
  - **its start state,**
  - **its accepting state,**
  - **its rejecting state,**

We can number the states

$Q = \{q_0, q_1, q_2, \ldots, q_n\}$

Such that

the start state is q_0,
the accepting state is q_{n-1},
and the rejecting state is q_n

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - its transition table

Encoding a Turing Machine

- To encode a Turing machine, we can provide an encoding of
  - its transition table

Encoding the Transition Table

	1			B		
q_0	B	R	q_1	B	R	q_{acc}
q_1	B	R	q_0	B	R	q_{rej}

Encoding the Transition Table

	1			B		
q_0	B	R	q_1	B	R	q_{acc}
q_1	B	R	q_0	B	R	q_{rej}

$(q_0, 1, B, R, q_1)(q_0, B, B, R, q_{acc})$

Encoding the Transition Table

	1			B		
q_0	B	R	q_1	B	R	q_{acc}
q_1	B	R	q_0	B	R	q_{rej}

$(q_0, 1, B, R, q_1)(q_0, B, B, R, q_{acc})$

$(q_1, 1, B, R, q_o)(q_1, B, B, R, q_{rej})$

Encoding the Transition Table

	1			B		
q_0	B	R	q_1	B	R	q_{acc}
q_1	B	R	q_0	B	R	q_{rej}

$(q_0, 1, B, R, q_1)(q_0, B, B, R, q_{acc})$

$(q_1, 1, B, R, q_o)(q_1, B, B, R, q_{rej})$

$(q_0, 1, B, R, q_1)(q_0, B, B, R, q_{acc})(q_1, 1, B, R, q_0)(q_1, B, B, R, q_{rej})$

Encoding Turing Machines

- Using some (scary!) mathematical hackery, we can convert this encoding of a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}, B)$ into a string in Σ^* for **any** choice of Σ.
  - Details are tricky. Talk to me after lecture if you want to see how to do this.
- If M is a TM, let $\langle M \rangle$ be the encoding of M using its own alphabet.
  - More generally, if O is some object, then $\langle O \rangle$ is a string encoding of that object.
- **This allows TMs to compute over TMs.**

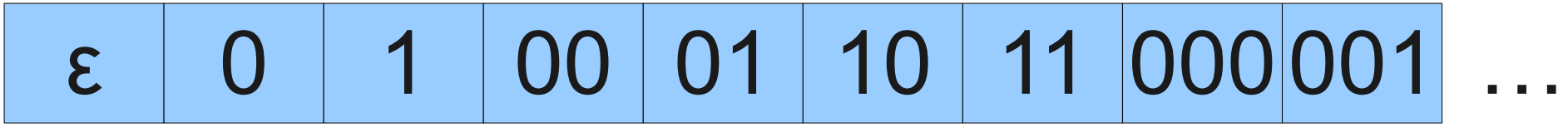
Enumerating Strings

- We can list off all strings over some alphabet in order by sorting by length, then alphabetically.
- For example:

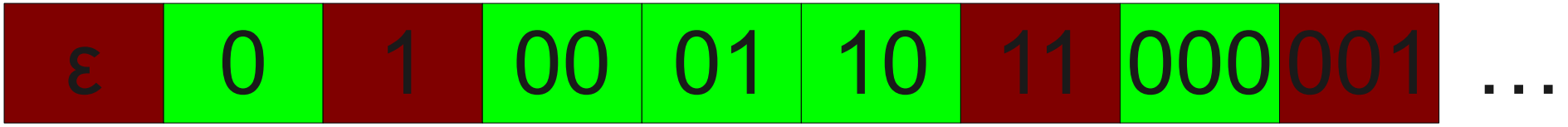
  ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, …

- Some strings encode TMs; others do not.
- Every TM has an encoding.

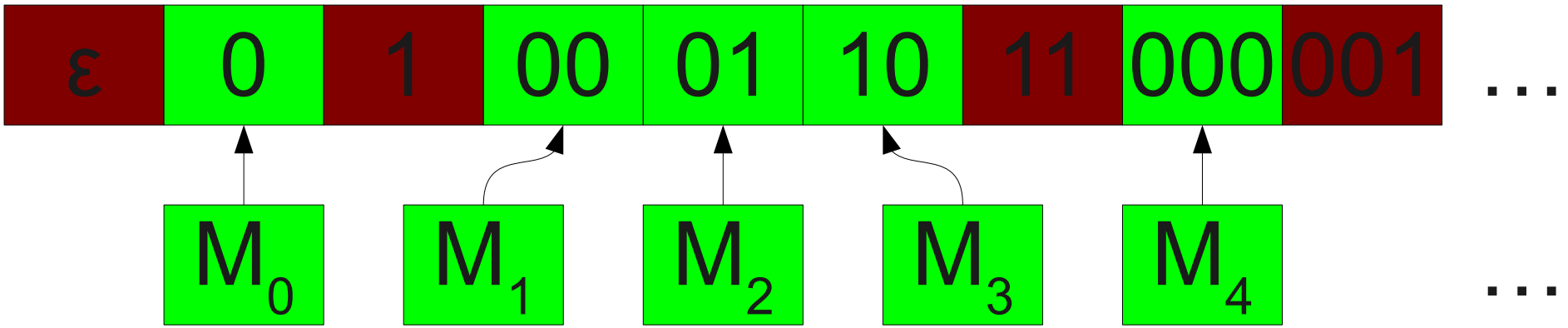
Enumerating Turing Machines



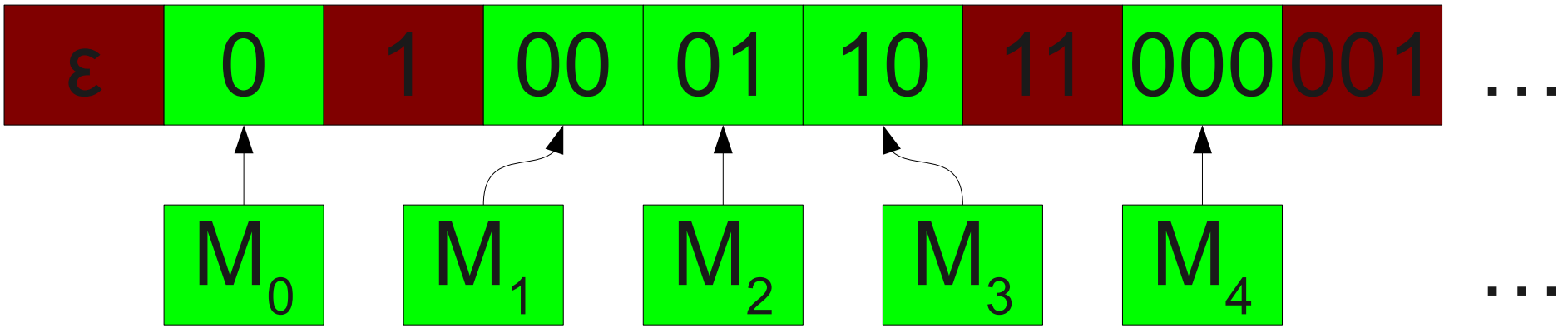
Enumerating Turing Machines



Enumerating Turing Machines



Enumerating Turing Machines



Every Turing machine appears somewhere in this sequence.

Enumerating Strings

- We can list off all strings over some alphabet in order by sorting by length, then alphabetically.
- For example:

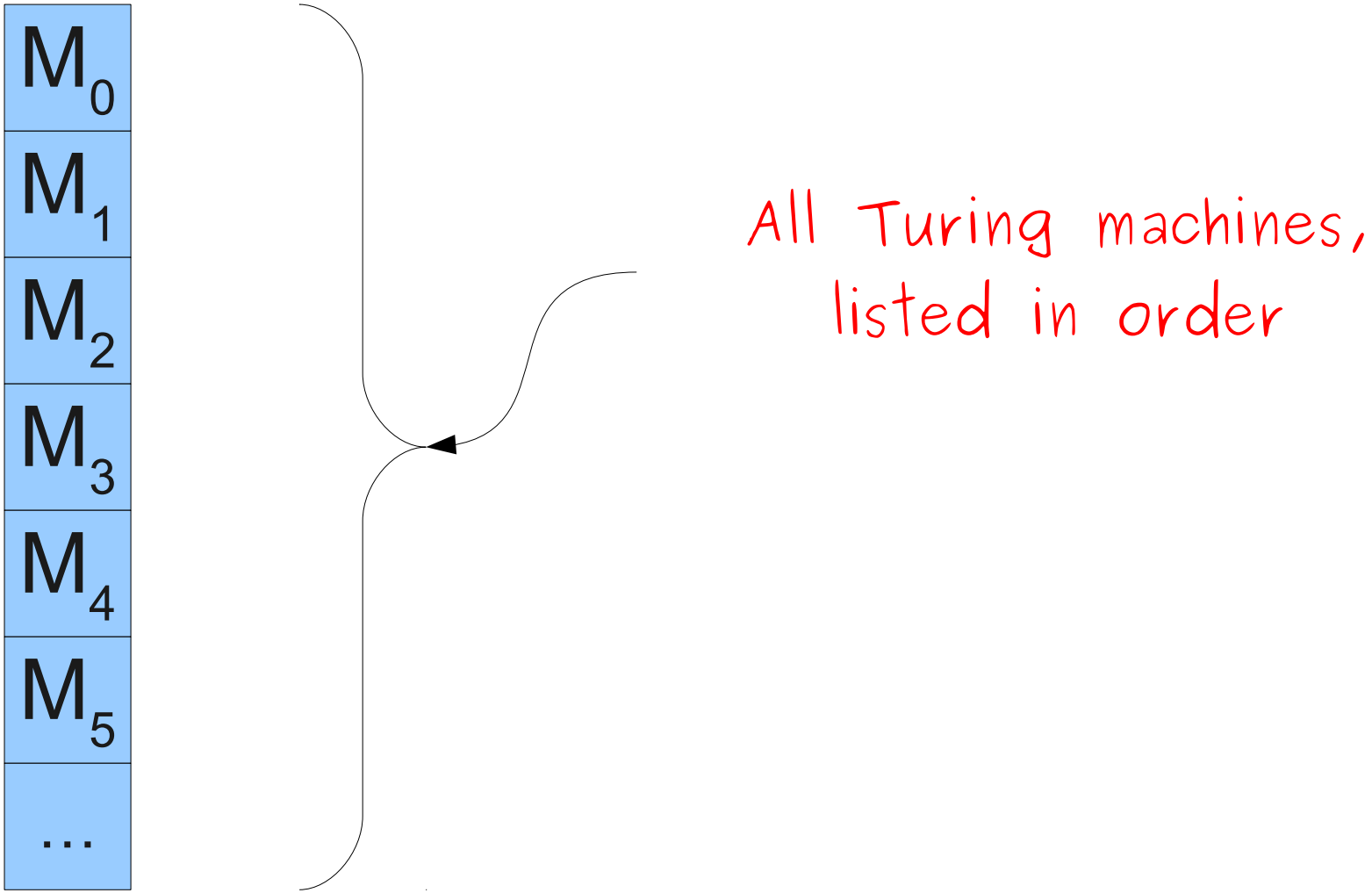
  ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, …

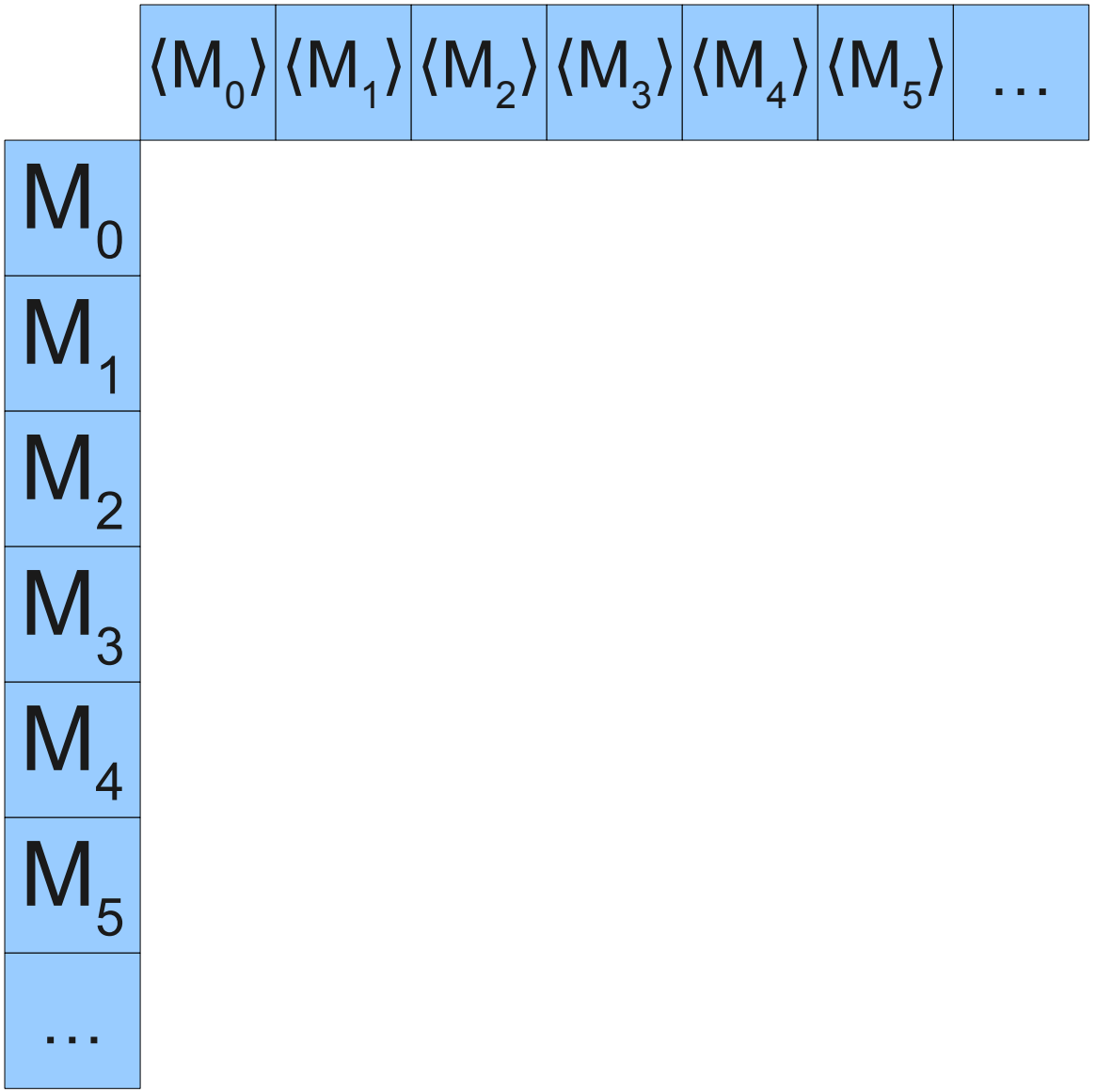
- Some strings encode TMs; others do not.
- Every TM has an encoding.
- We can list off all TMs as M_0, M_1, M_2, … as follows:
  - M_0 is the TM encoded by first string that encodes a TM.
  - M_{n+1} is the TM encoded by the first string after the one encoding M_n that encodes a TM.

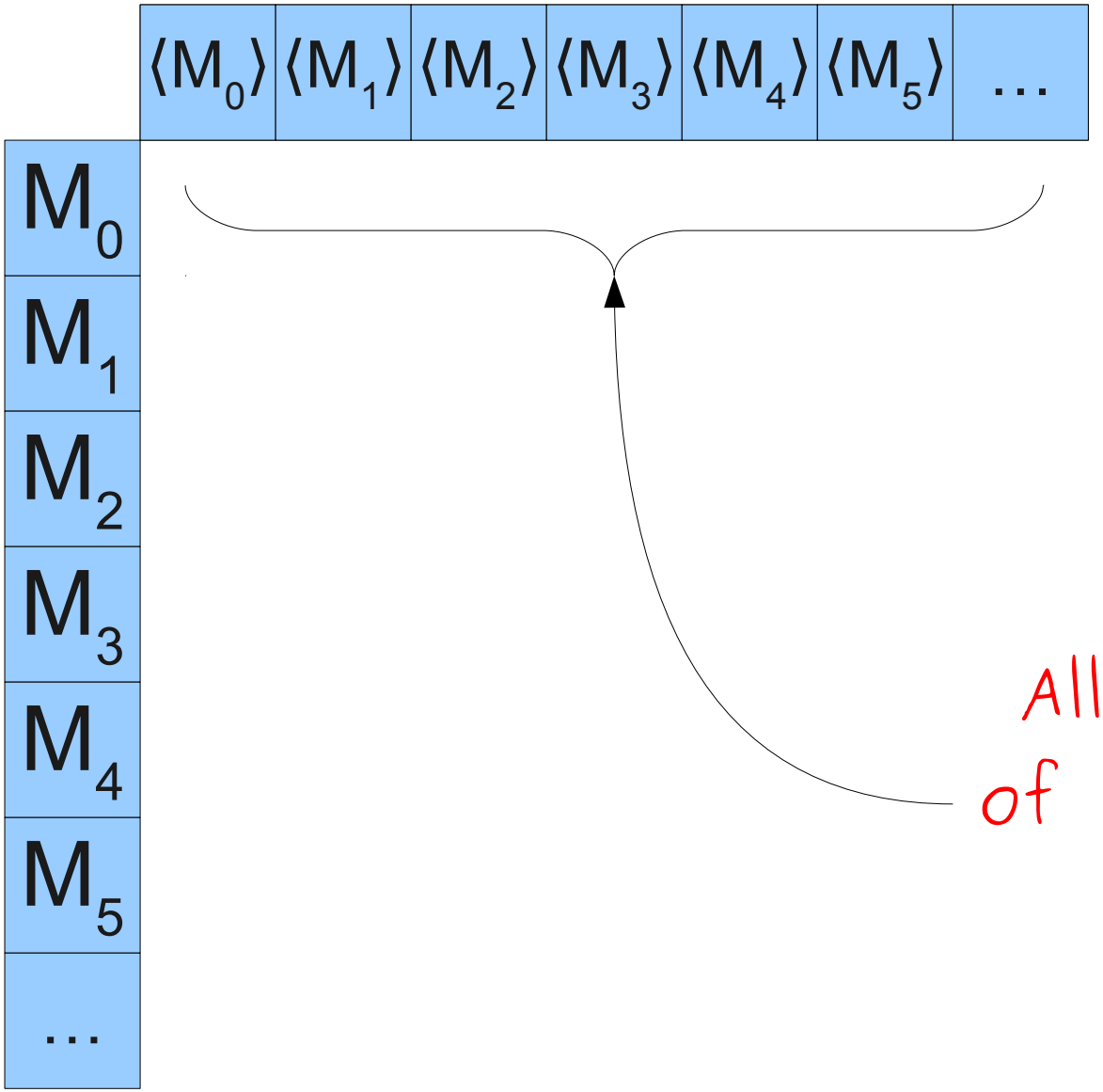
A Critical Observation

Every TM M either accepts ⟨M⟩ or it does not
(it either rejects or loops infinitely).


M_0
M_1
M_2
M_3
M_4
M_5
...


M_0
M_1
M_2
M_3
M_4
M_5
...
All Turing machines,
listed in order


⟨M0⟩
⟨M1⟩
⟨M2⟩
⟨M3⟩
⟨M4⟩
⟨M5⟩
…
M0
M1
M2
M3
M4
M5
…



All descriptions of TMs, listed in order.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1							
M_2							
M_3							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2							
M_3							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

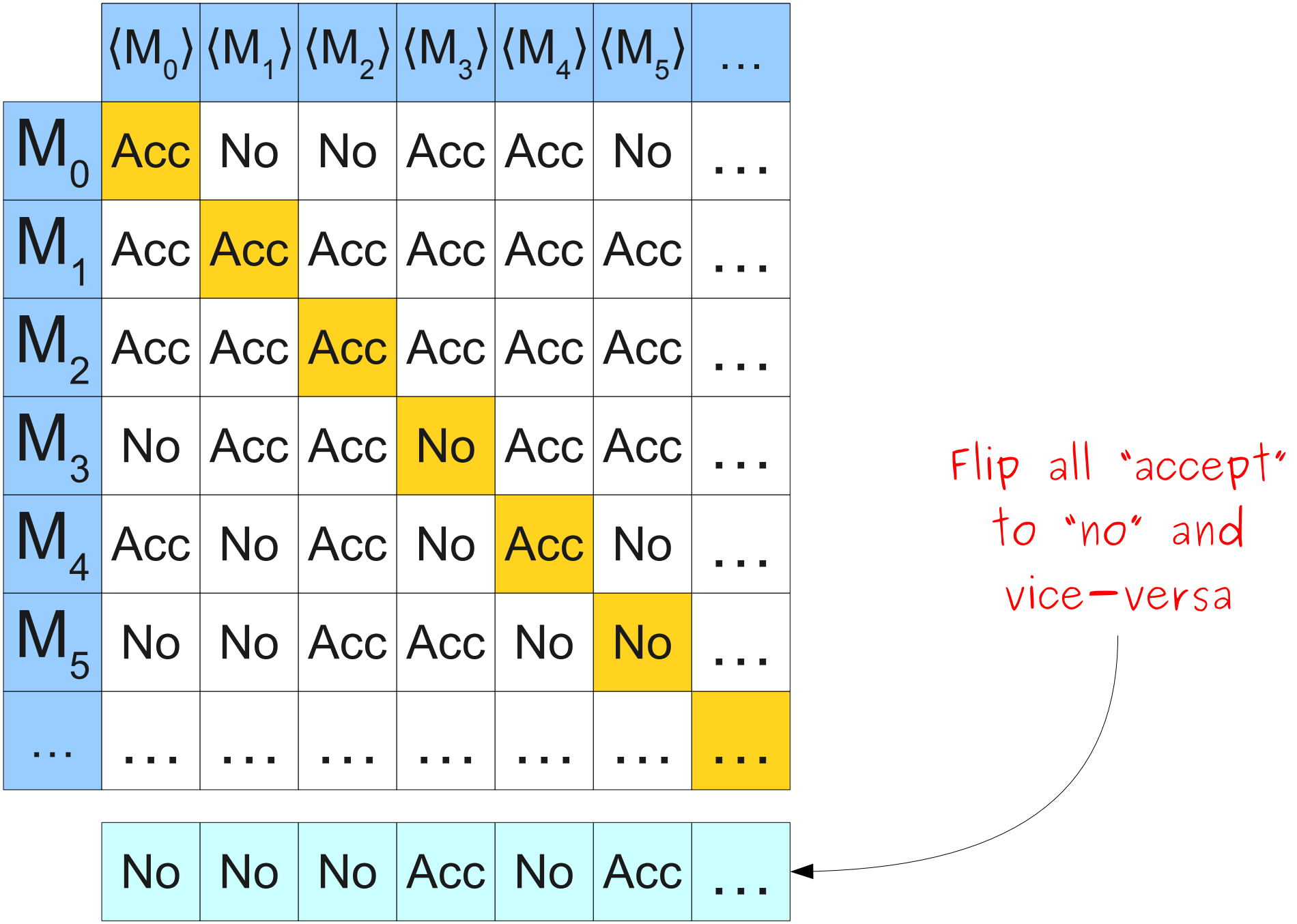
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

Acc	Acc	Acc	No	Acc	No	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	No	No	No	Acc	No	Acc	…


⟨M0⟩ ⟨M1⟩ ⟨M2⟩ ⟨M3⟩ ⟨M4⟩ ⟨M5⟩ …
M0 Acc No No Acc Acc No …
M1 Acc Acc Acc Acc Acc Acc …
M2 Acc Acc Acc Acc Acc Acc …
M3 No Acc Acc No Acc Acc …
M4 Acc No Acc No Acc No …
M5 No No Acc Acc No No …
… … … … … … … …
No No No Acc No Acc …
Flip all "accept" to "no" and vice-versa

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

What TM has this behavior?

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

No	No	No	Acc	No	Acc	…

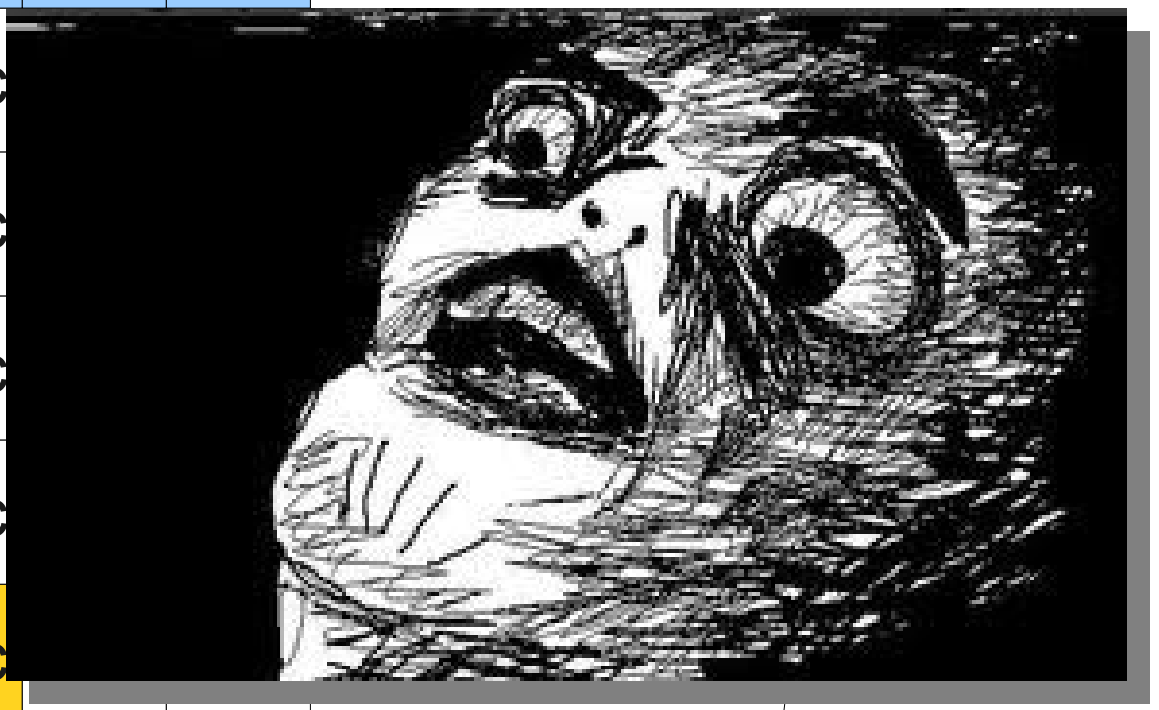
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…

	No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…
	No	No	No	Acc	No	Acc	…

No TM has this behavior!

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc		
M_1	Acc	Acc	Acc	Acc	Acc		
M_2	Acc	Acc	Acc	Acc	Acc		
M_3	No	Acc	Acc	No	Acc		
M_4	Acc	No	Acc	No	Acc		
M_5	No	No	Acc	Acc	No	No	…
…	…	…	…	…	…	…	…



No	No	No	Acc	No	Acc	…

A Deviously Tricky Problem

- The **diagonalization language** L_D is defined as

$$L_D = \{ w \mid w = \langle M \rangle \text{ for some TM } M, \text{ and } M \text{ does not accept } \langle M \rangle \}$$

- We can alternatively write this as

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

- That is, L_D is the set of descriptions of Turing machines that do not accept themselves.
- Is L_D Turing-recognizable?

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

Case 2: H does not accept $\langle H \rangle$.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

Case 2: H does not accept $\langle H \rangle$. Then $\langle H \rangle \notin L_D$, so $\langle H \rangle$ either is not an encoding of a TM or M accepts $\langle H \rangle$.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

Case 2: H does not accept $\langle H \rangle$. Then $\langle H \rangle \notin L_D$, so $\langle H \rangle$ either is not an encoding of a TM or M accepts $\langle H \rangle$. Since $\langle H \rangle$ is indeed an encoding of H, this means that M accepts $\langle H \rangle$, which is impossible because H does not accept $\langle H \rangle$.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

Case 2: H does not accept $\langle H \rangle$. Then $\langle H \rangle \notin L_D$, so $\langle H \rangle$ either is not an encoding of a TM or M accepts $\langle H \rangle$. Since $\langle H \rangle$ is indeed an encoding of H, this means that M accepts $\langle H \rangle$, which is impossible because H does not accept $\langle H \rangle$.

In either case we reach a contradiction.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

Theorem: L_D is not Turing-recognizable.

Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

Case 2: H does not accept $\langle H \rangle$. Then $\langle H \rangle \notin L_D$, so $\langle H \rangle$ either is not an encoding of a TM or M accepts $\langle H \rangle$. Since $\langle H \rangle$ is indeed an encoding of H, this means that M accepts $\langle H \rangle$, which is impossible because H does not accept $\langle H \rangle$.

In either case we reach a contradiction. Therefore, our assumption was wrong and L_D is not Turing-recognizable.

$$L_D = \{ \langle M \rangle \mid M \text{ does not accept } \langle M \rangle \}$$

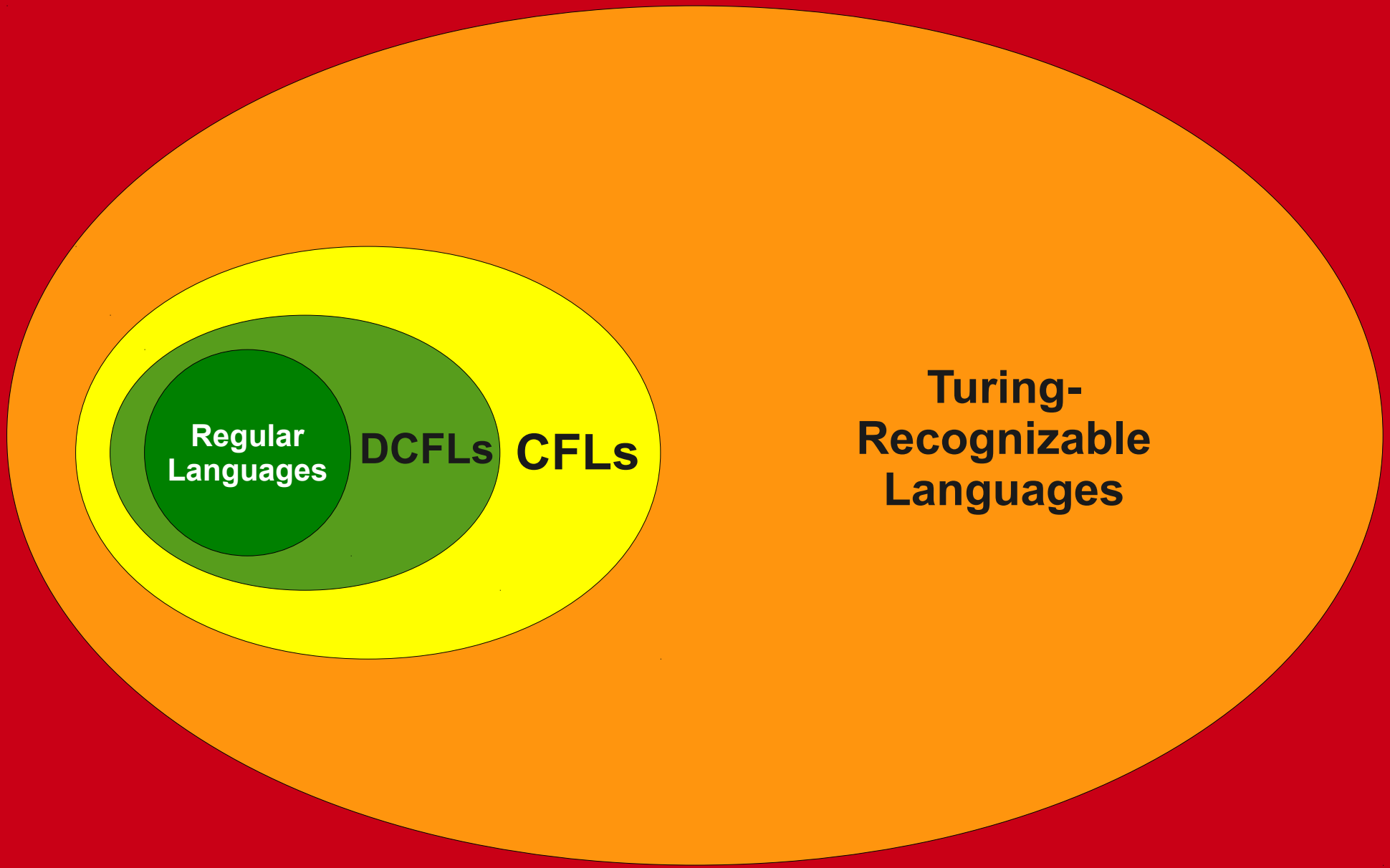
Theorem: L_D is not Turing-recognizable.

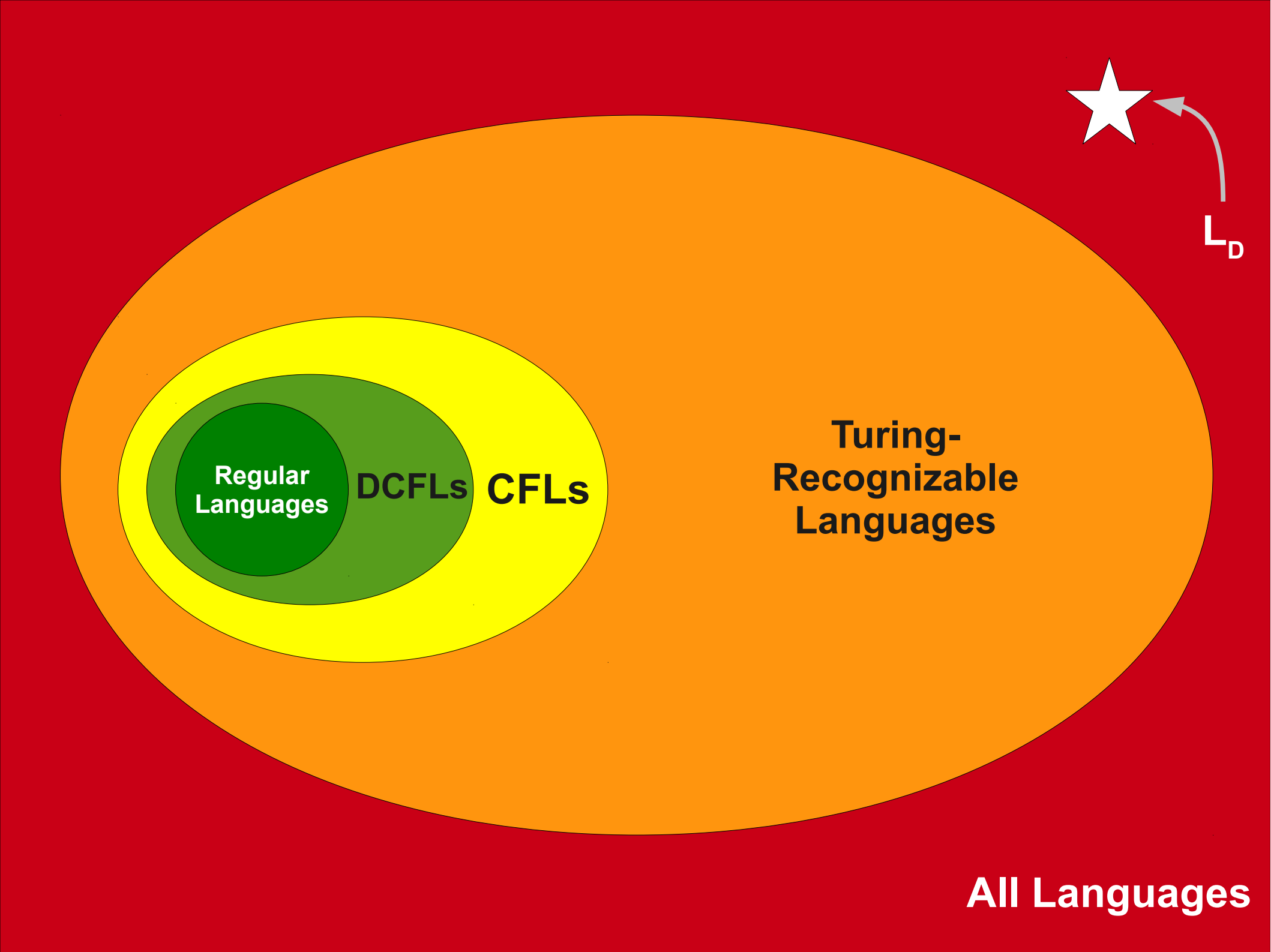
Proof: By contradiction; assume that L_D is Turing-recognizable. Then there must be a TM H such that $L(H) = L_D$, so H accepts *w* iff it is contained in L_D. Now, consider what happens when we run H on $\langle H \rangle$. Either H accepts $\langle H \rangle$ or it does not. We consider two cases:

Case 1: H accepts $\langle H \rangle$. Then $\langle H \rangle \in L_D$, so H does not accept $\langle H \rangle$. This is impossible, since H does indeed accept $\langle H \rangle$.

Case 2: H does not accept $\langle H \rangle$. Then $\langle H \rangle \notin L_D$, so $\langle H \rangle$ either is not an encoding of a TM or M accepts $\langle H \rangle$. Since $\langle H \rangle$ is indeed an encoding of H, this means that M accepts $\langle H \rangle$, which is impossible because H does not accept $\langle H \rangle$.

In either case we reach a contradiction. Therefore, our assumption was wrong and L_D is not Turing-recognizable. ■


Regular Languages
DCFLs
CFLs
Turing-Recognizable Languages
All Languages


L_D
Regular Languages
DCFLs
CFLs
Turing-Recognizable Languages
All Languages

What Just Happened?

- What is it about L_D that makes it impossible to solve with a Turing machine?
- **Indirect self-reference.**
- Because TMs can be encoded as strings, TMs that compute over other TMs can be forced to compute some property of themselves **without realizing it**.
- The language L_D self-destructs given a Turing machine that recognizes L_D by stating “this machine accepts itself if and only if it does not accept itself.”

Diagonalization Revisited

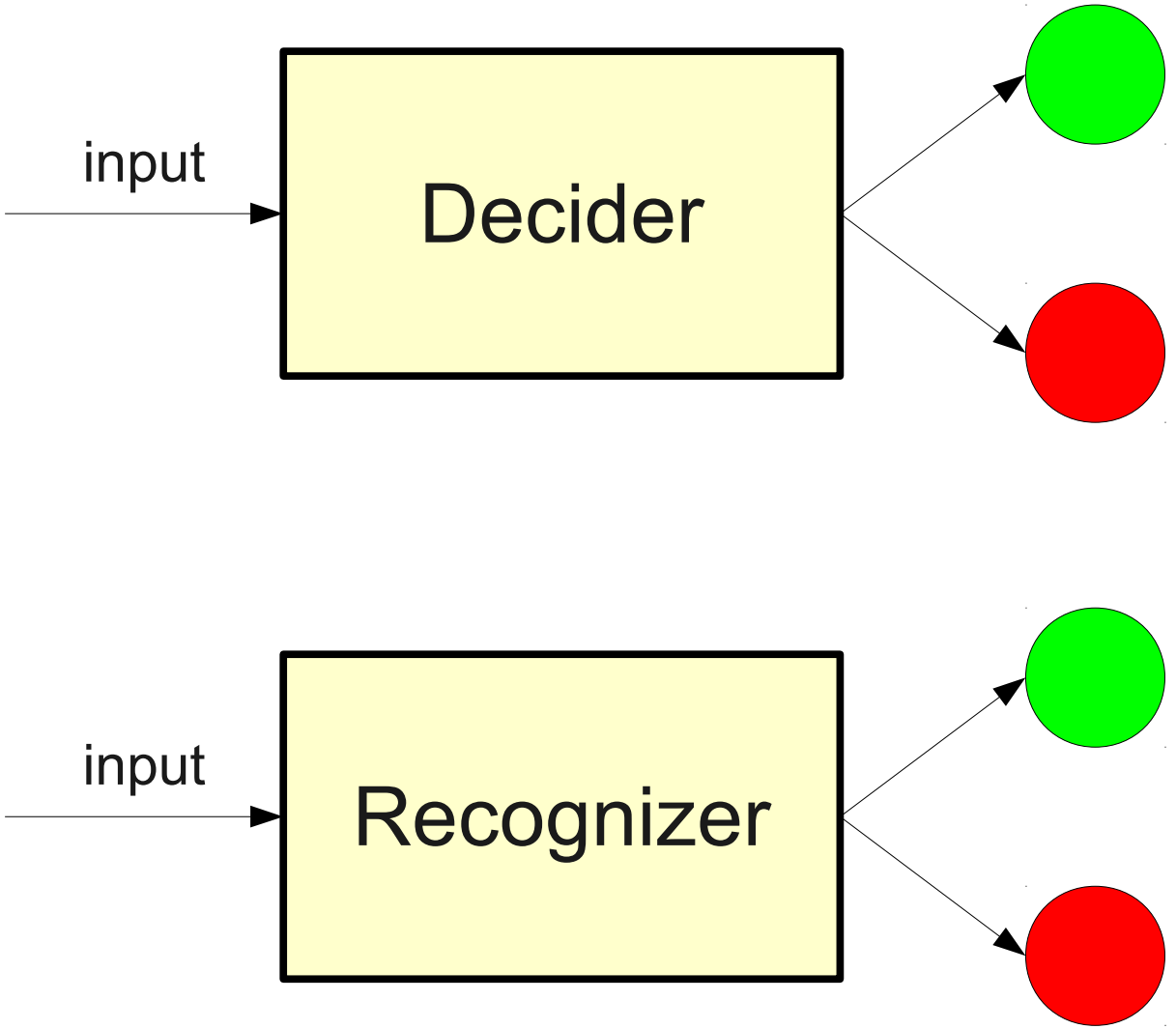
- At our first lecture, we saw Cantor's diagonal argument to show that |S| < |P(S)| for any choice of S.
- The proof that L_D is undecidable is almost **exactly** the same proof.
  - The language of a TM may contain descriptions of TMs.
  - We can draw a matrix showing which TMs are accepted by which TMs.
  - The complemented diagonal cannot be accepted by any TM, because each TM disagrees with itself.
  - So there is some language not recognized by any TM.

An Undecidable Problem

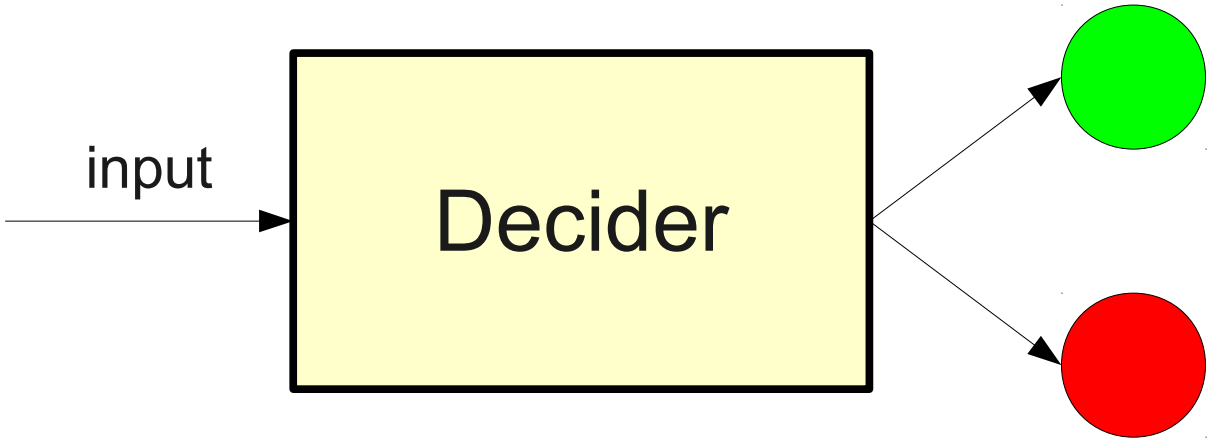
Decidability and Recognizability

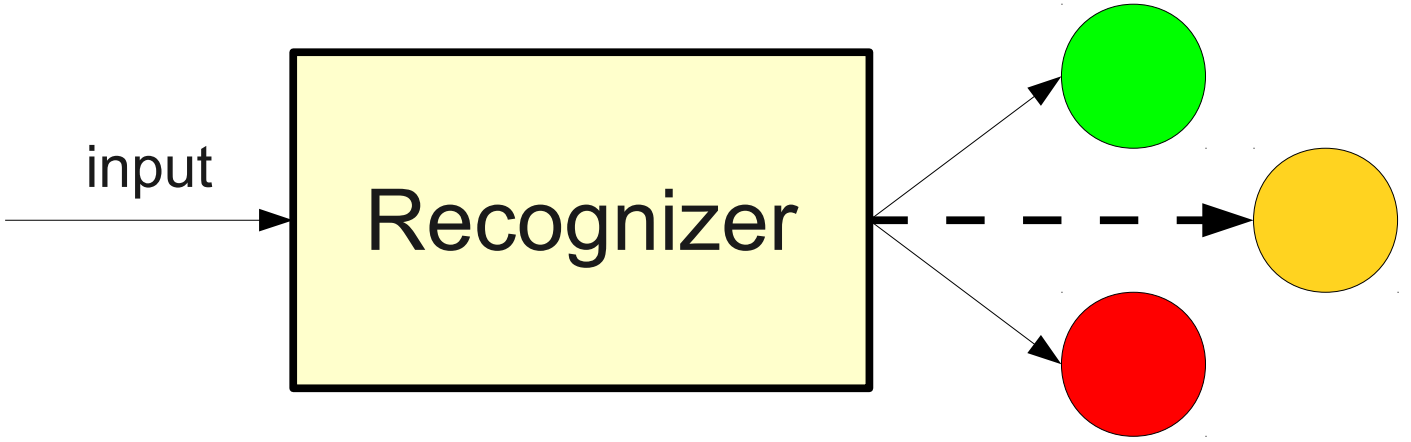
- L is **Turing-recognizable** if there is a Turing machine M that accepts every string in L.
  - It might loop forever on strings not in L.
- L is **Turing-decidable** (if there is a Turing machine M that accepts every string in L and rejects every string not in L.
  - It never enters an infinite loop.
  - These languages are also sometimes called **computable**, **decidable**, or **recursive**.

Decidability and Recognizability



Decidability and Recognizability





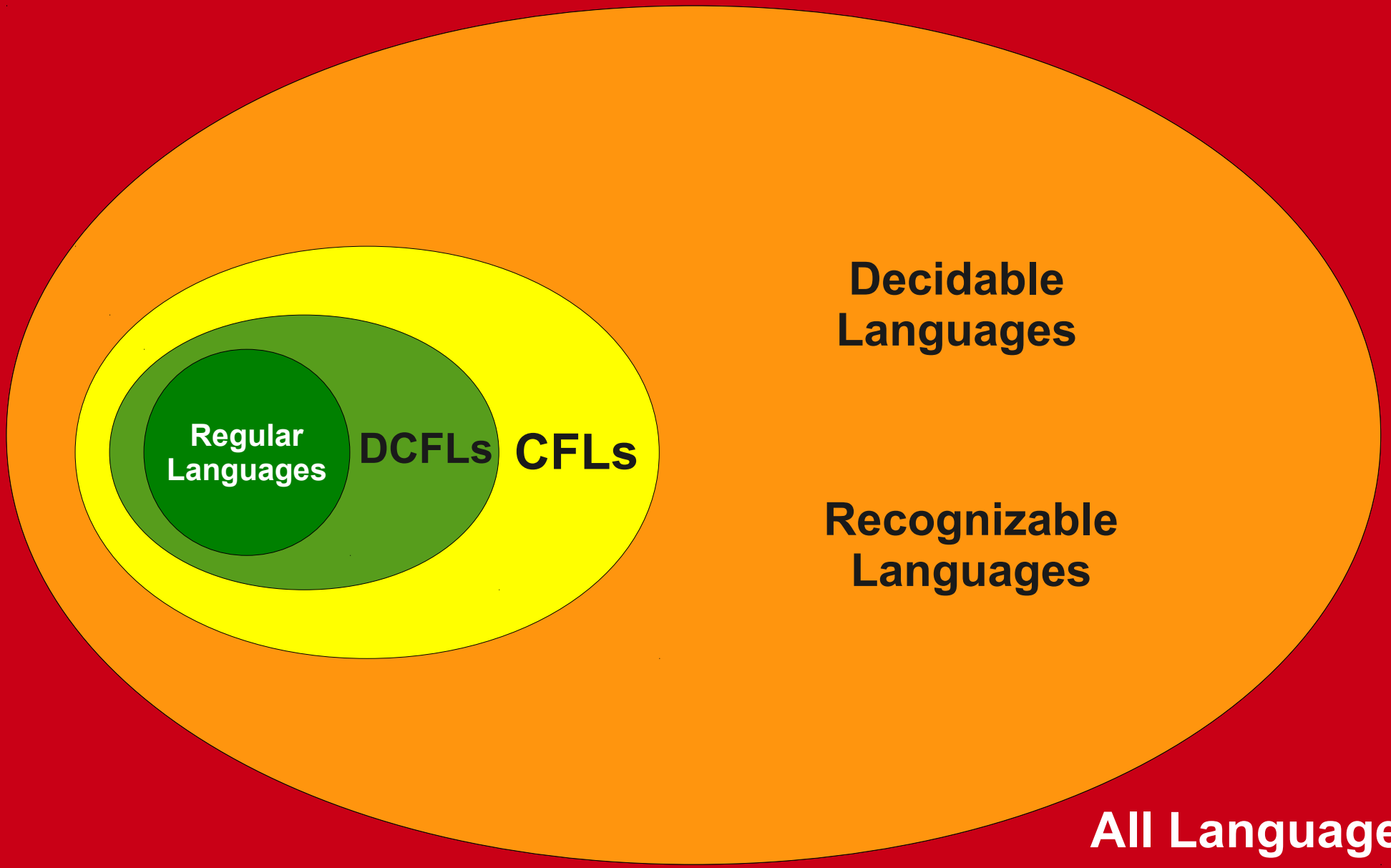
Decidable Problems

- The Church-Turing thesis asserts that any feasible model of computation is no more powerful than a Turing machine.
- The Turing-decidable languages are, therefore, problems for which there is some computer that can always produce a yes or no answer.
- A problem is decidable precisely when there is some algorithm to solve it.
- **Decidability formalizes the definition of an algorithm.**

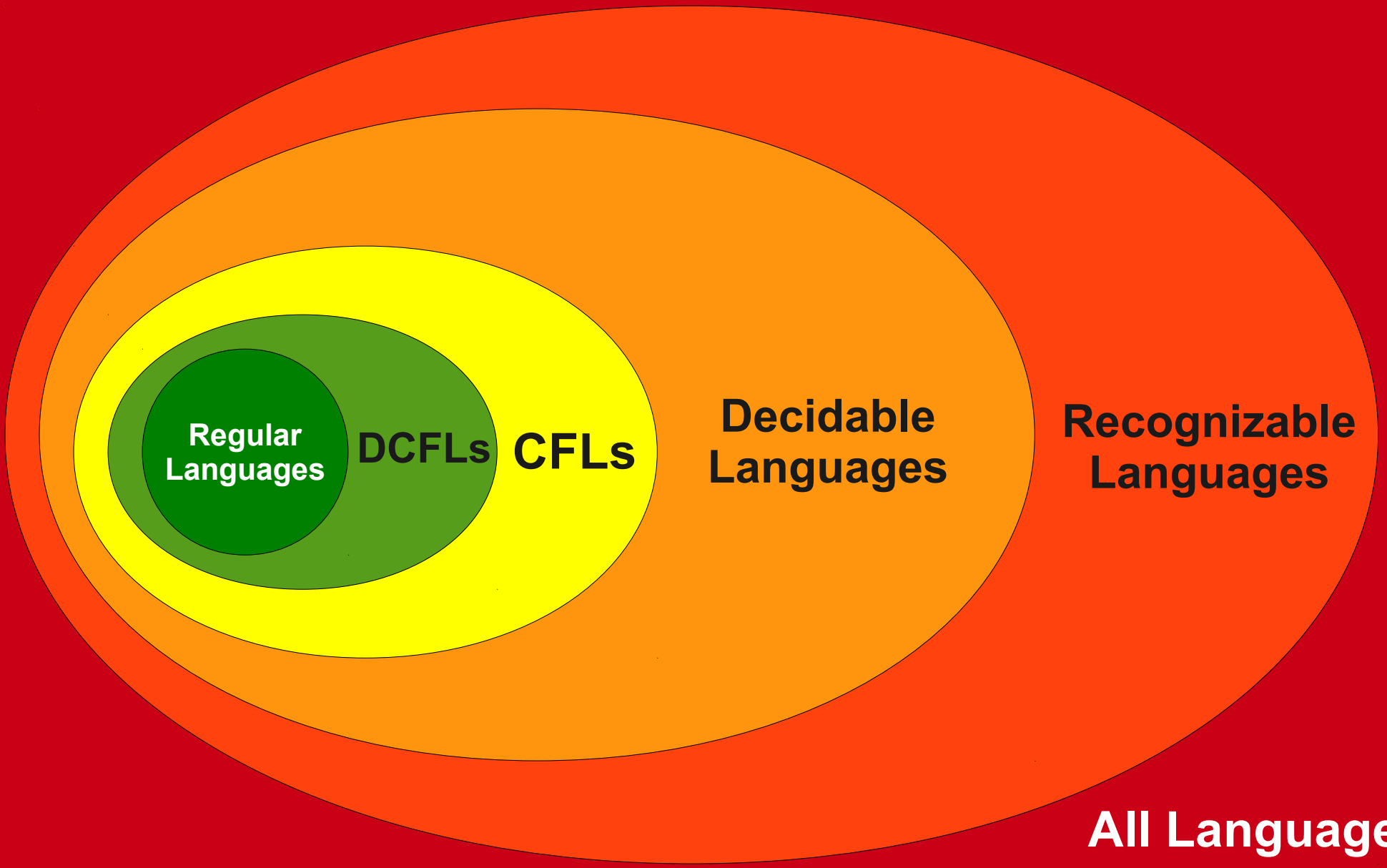
Decidable Problems

- Any regular language is decidable.
  - Obtain a DFA for L.
  - Convert L into a TM by adding accept and reject states.
  - Make one pass over the input, then accept if in an accepting state and reject otherwise.
- Any DCFL is decidable.
  - Obtain a DPDA for L.
  - Use a multitape TM to simulate consuming the input and maintaining the stack.
- Any CFL is decidable.
  - A trickier proof; see **Earley's algorithm** for details on how to do this.

Which Picture is Correct?



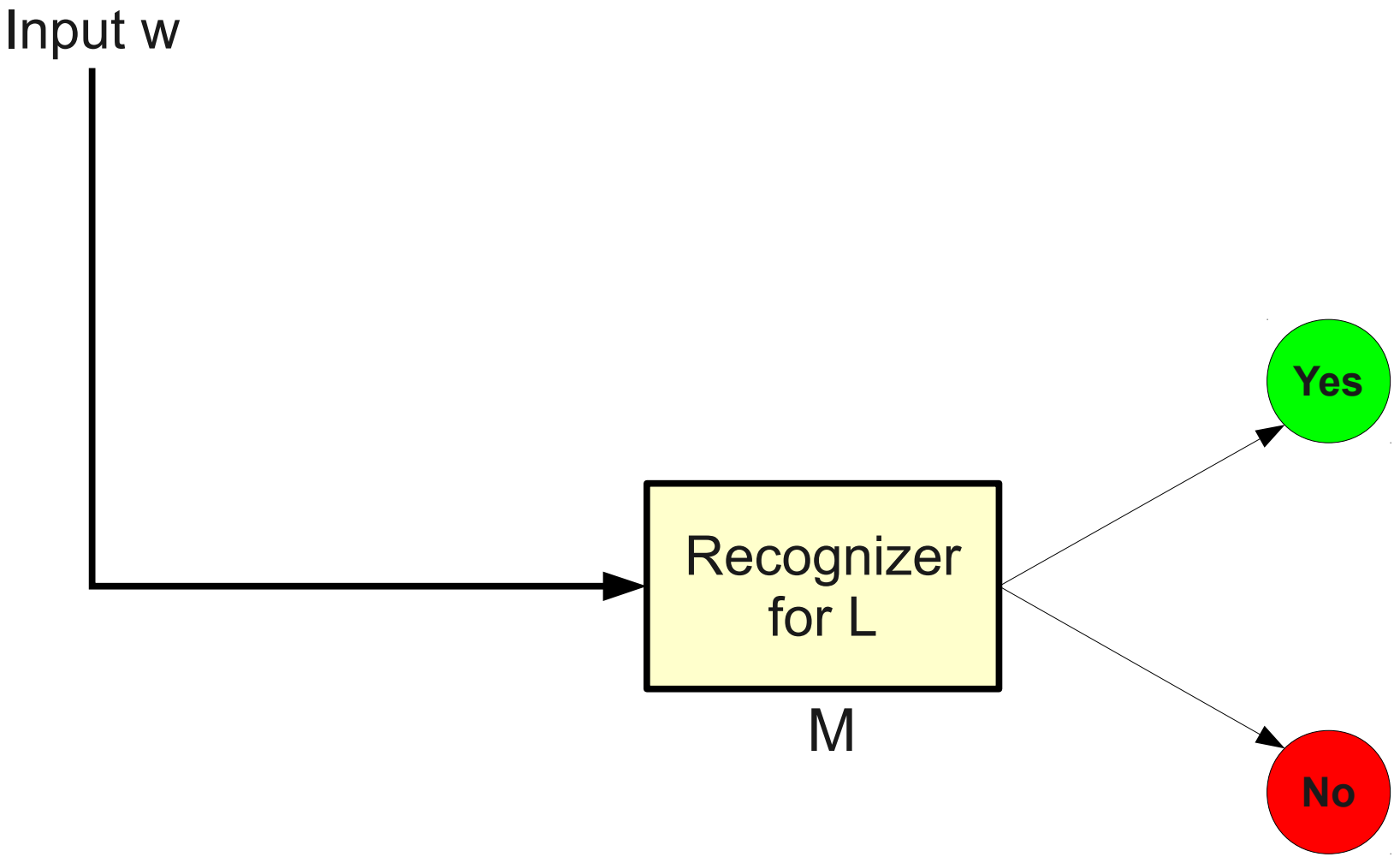
Which Picture is Correct?



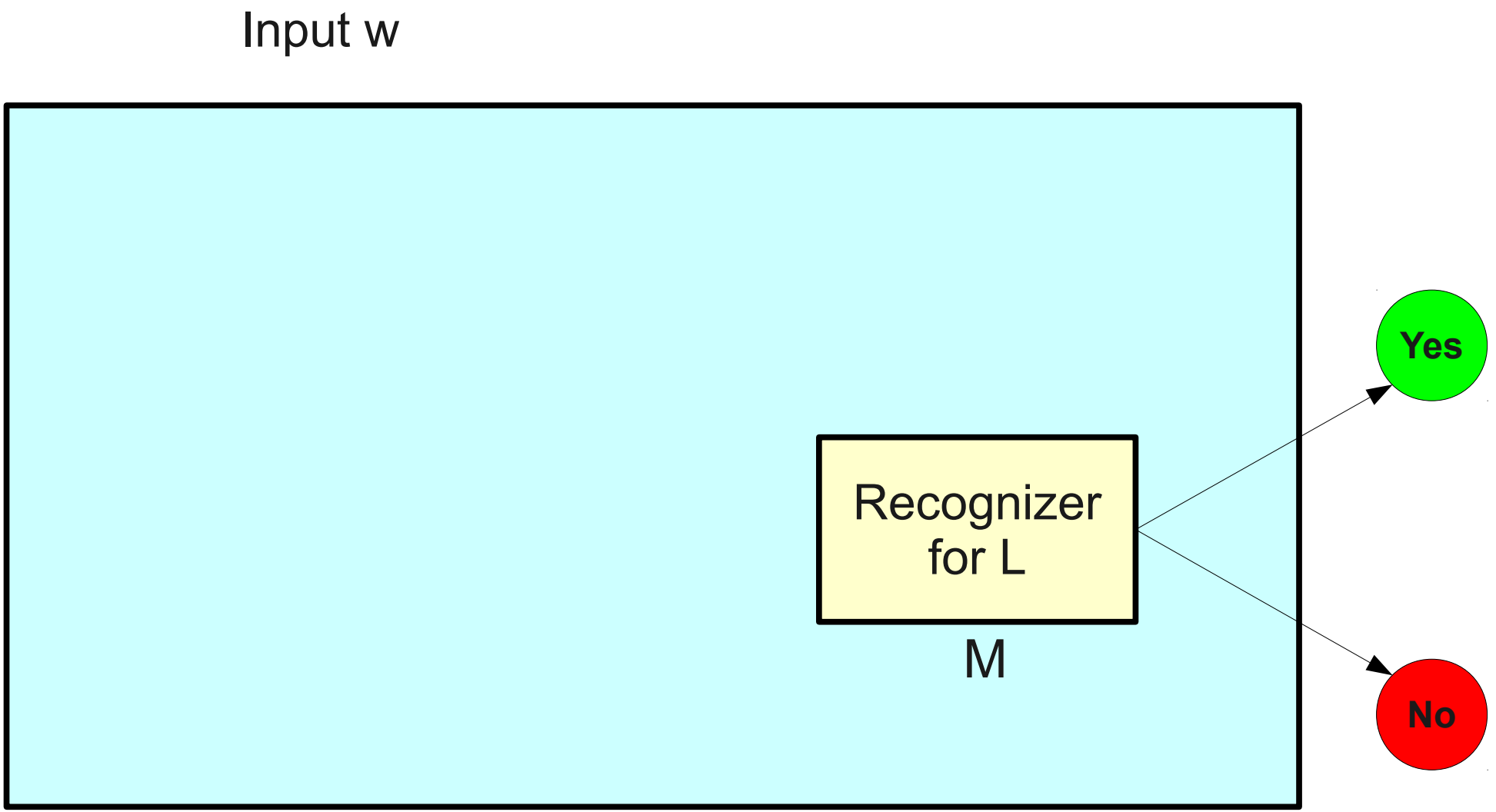
From Recognizers to Deciders

- Recall: When running a TM on an input *w*, the TM will do exactly one of the following:
  - **Accept** w,
  - **Reject** w, or
  - **Loop forever**.
- If we could detect whether the TM could loop forever, then we could build a decider for it.
  - Check whether the TM will eventually halt.
  - If so, run the TM and output its result.
  - If not, immediately reject.

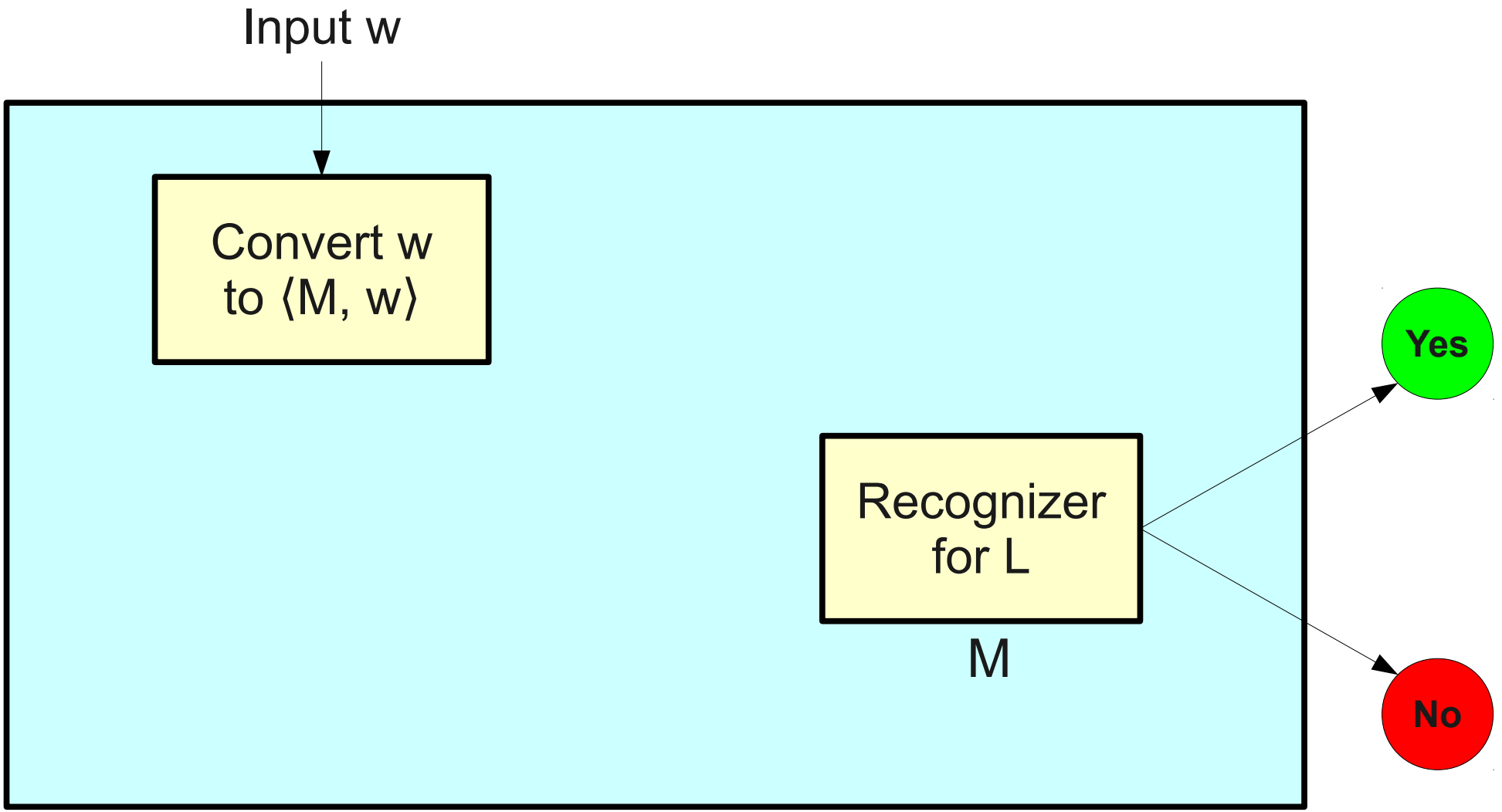
From Recognizers to Deciders



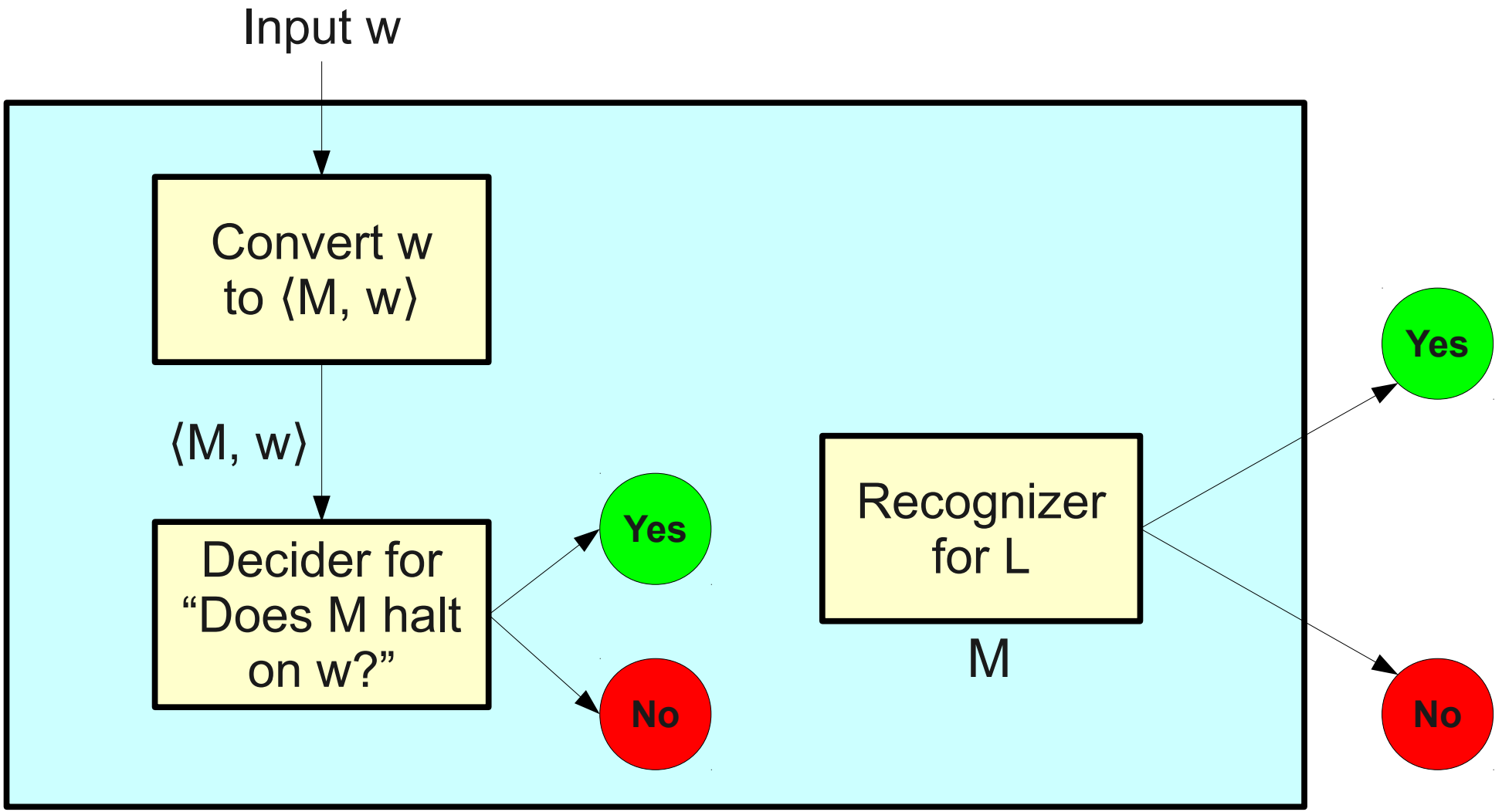
From Recognizers to Deciders



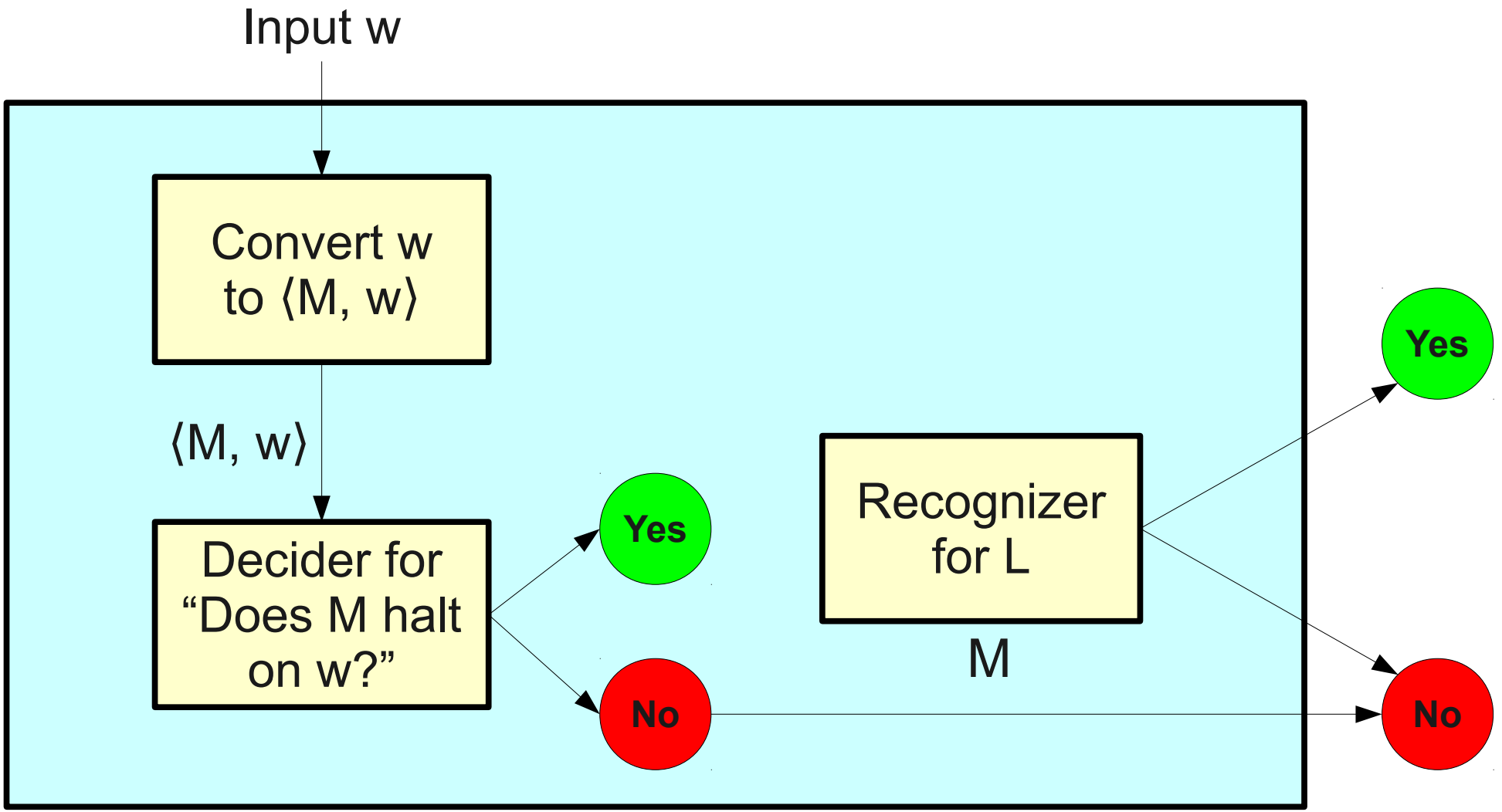
From Recognizers to Deciders



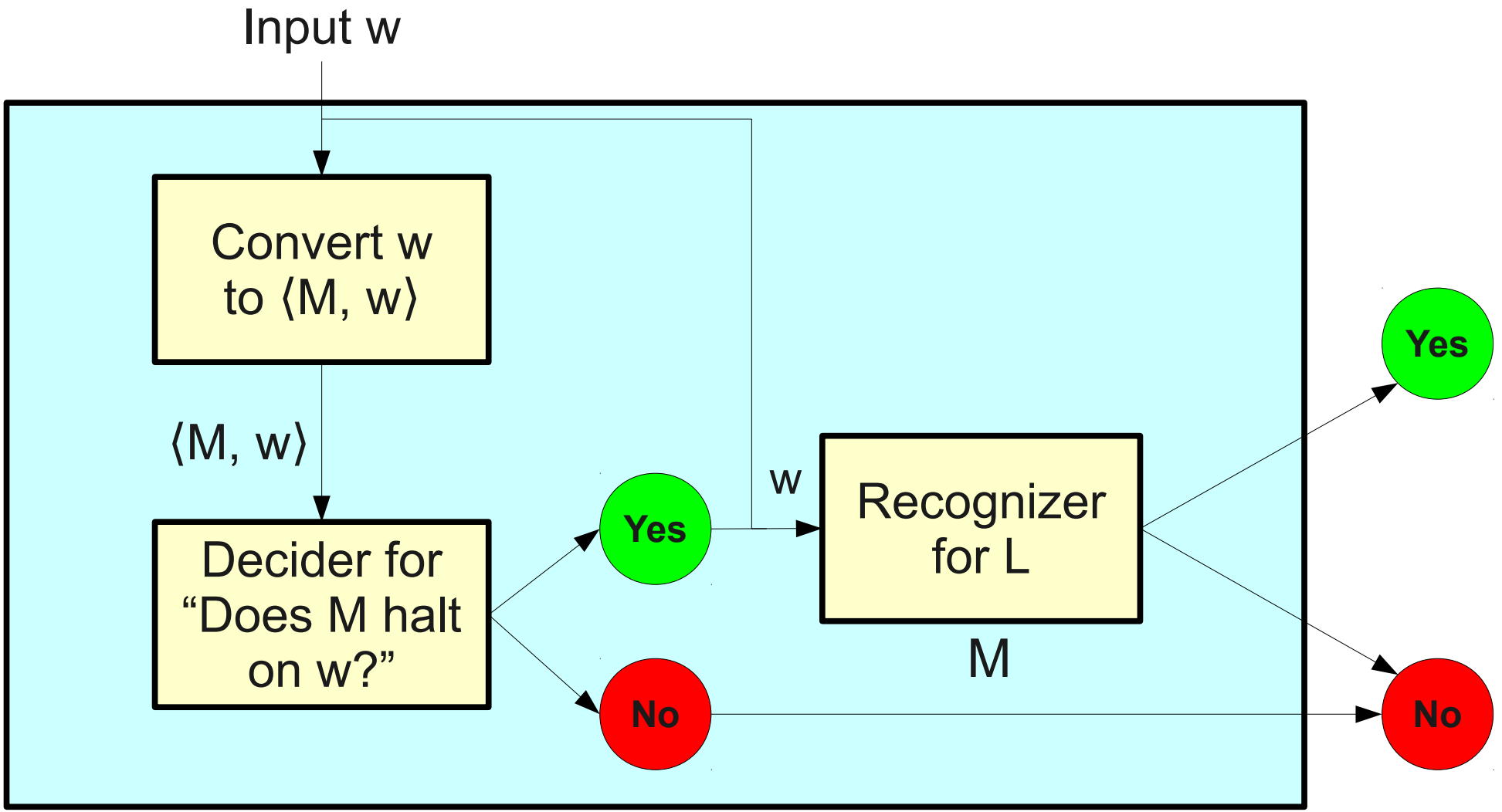
From Recognizers to Deciders



From Recognizers to Deciders



From Recognizers to Deciders



What This Would Mean

- The previous construction relies on the following language being decidable:

  $HALT = \{ \langle M, w \rangle \mid M \text{ halts when run on } w \}$

- This is called the **halting problem**.
- If this is a decidable language, this has **huge** implications.

If *HALT* is Decidable...

- Suppose we have a TM H that decides *HALT*.
- We can check whether some (computable) property P(n) is true for all natural numbers n.
  - Build a TM M that ignores its input and starts checking P(0), P(1), P(2), …
    - M terminates and rejects if for some n, P(n) is false.
    - M loops infinitely otherwise.
  - If *H* halts on ⟨M, ε⟩, then P(n) is not true for all n.
  - Otherwise, P(n) is true for all n.

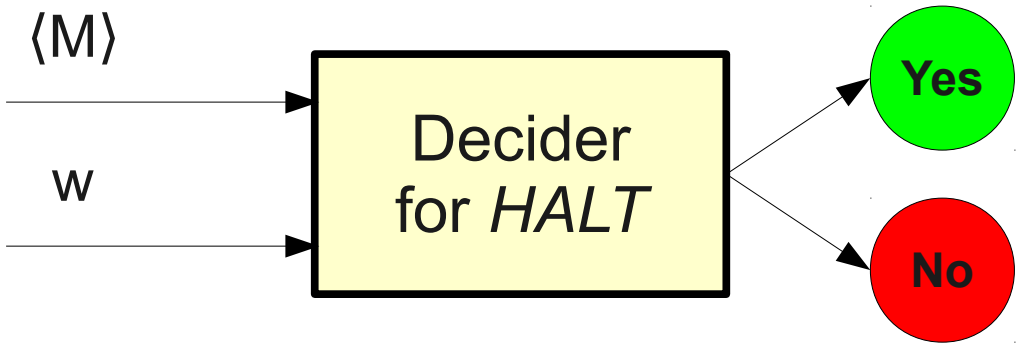
If *HALT* is Decidable...

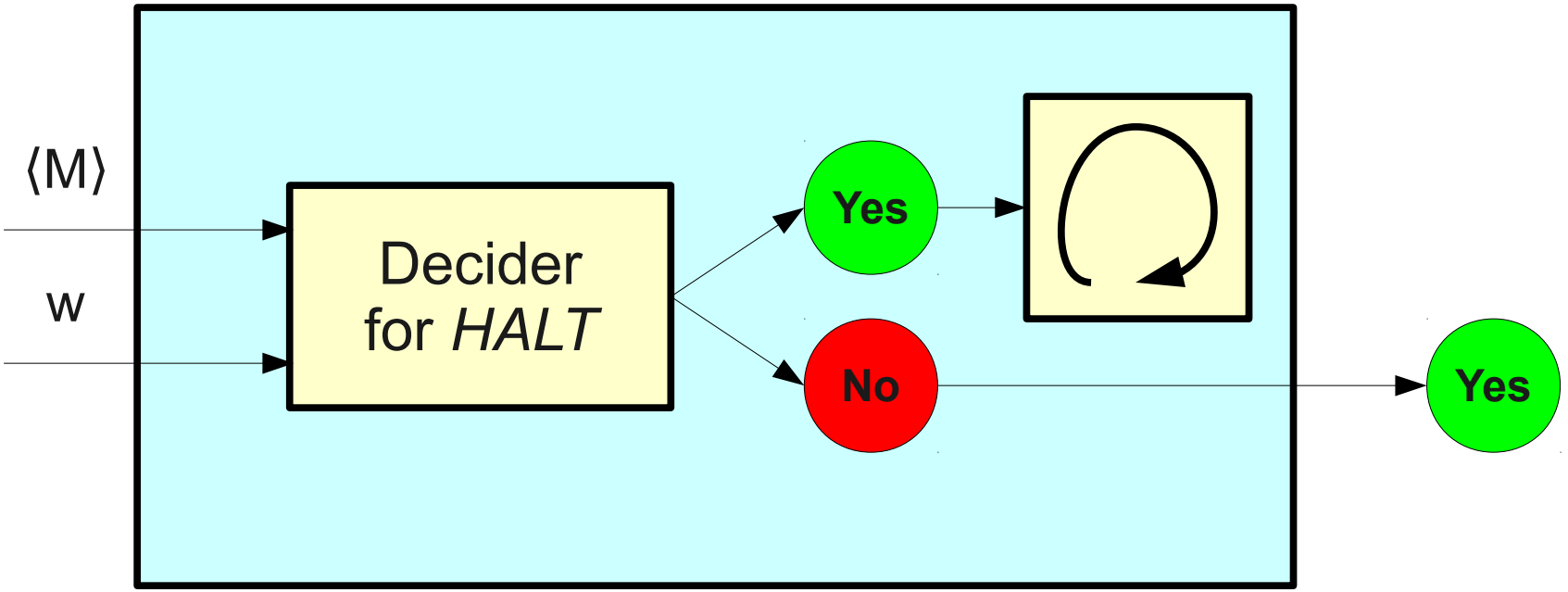
- Suppose we have a TM H that decides *HALT*.
- We can build a check whether or not a statement φ in first-order logic is true.
  - Build a TM M that begins listing off all true FOL statements one at a time.
    - Begin with a set of **axioms**.
    - Keep applying **inference rules** one at a time.
  - If φ is true, M halts and accepts.
  - If φ is false, M loops forever.
  - If H halts on ⟨M, φ⟩, then φ is true.
  - Otherwise, φ is not true.

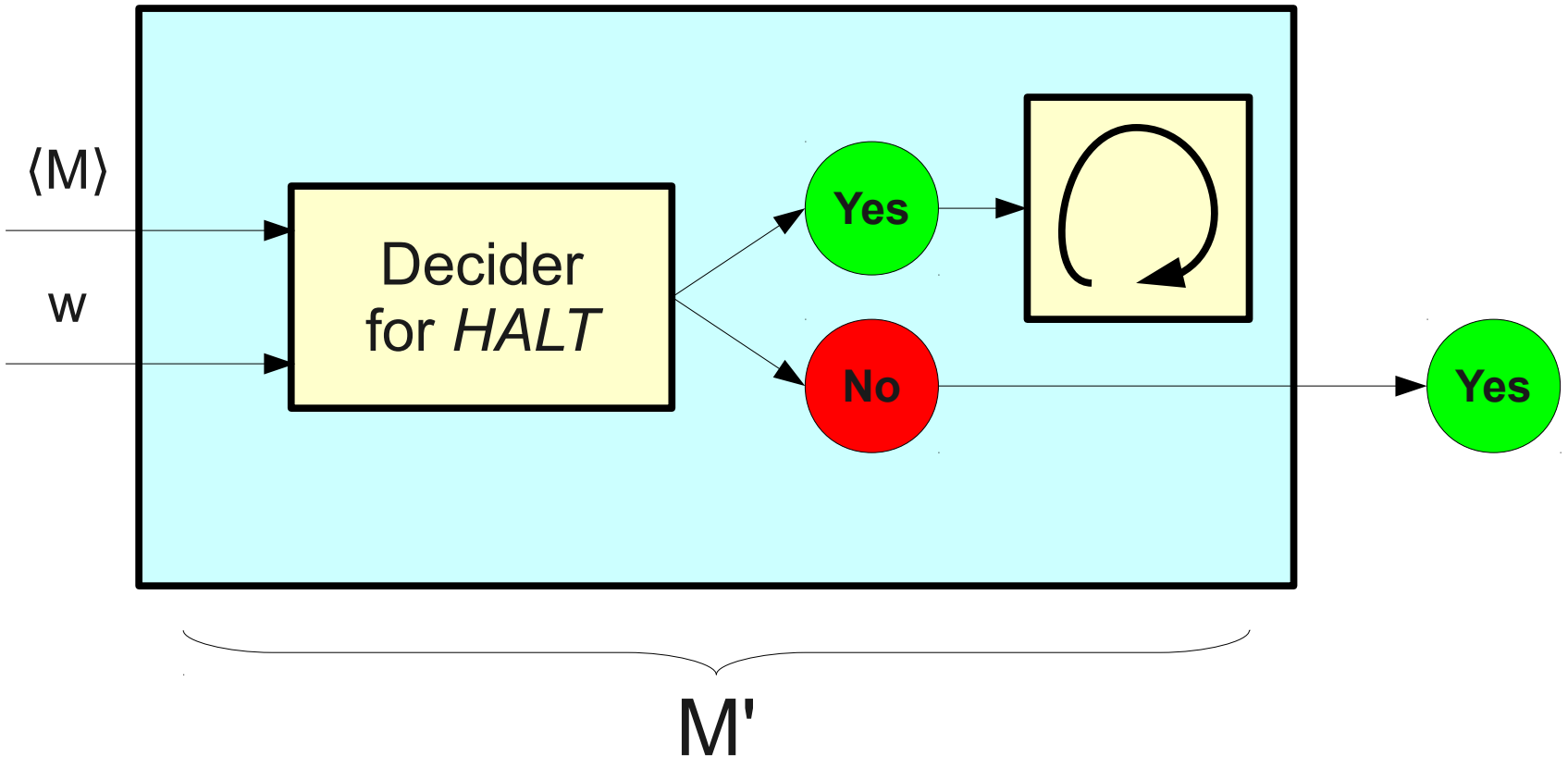
If *HALT* is decidable, then most (if not all) of mathematics could be automated.

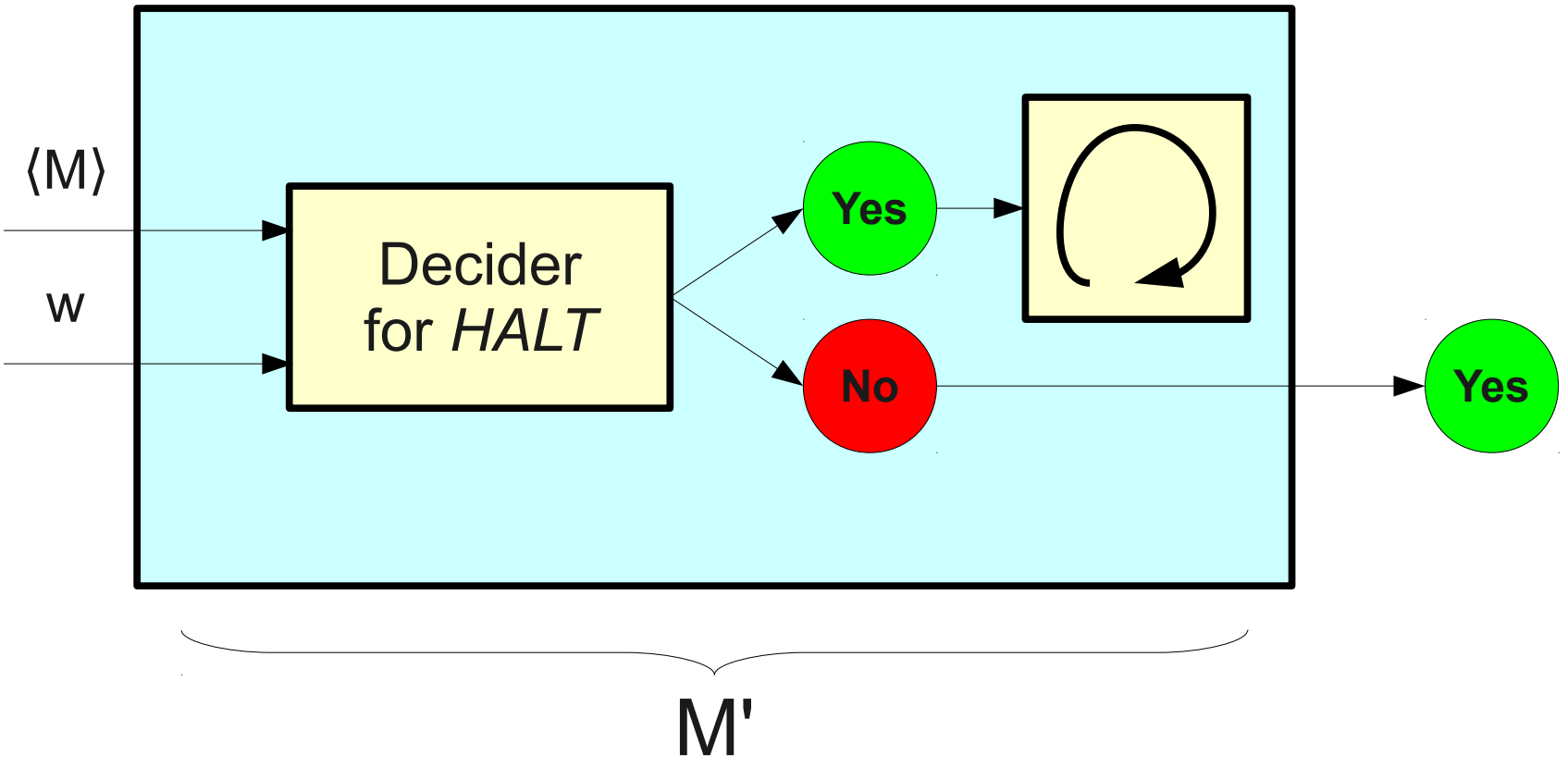
Unfortunately, *HALT* is **undecidable**.

Assume, for the sake
of contradiction, that
HALT is decidable.

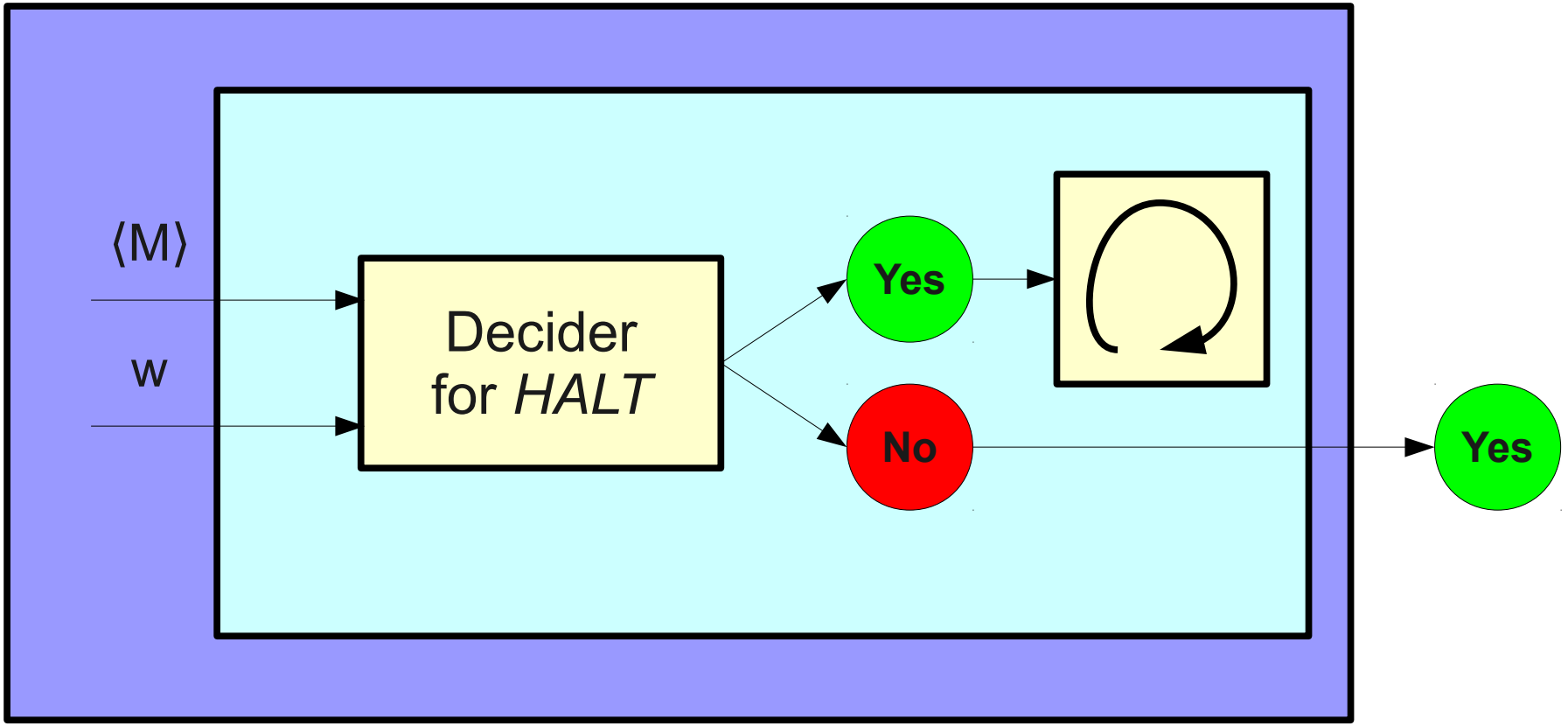

⟨M⟩
w
Decider
for *HALT*
Yes
No

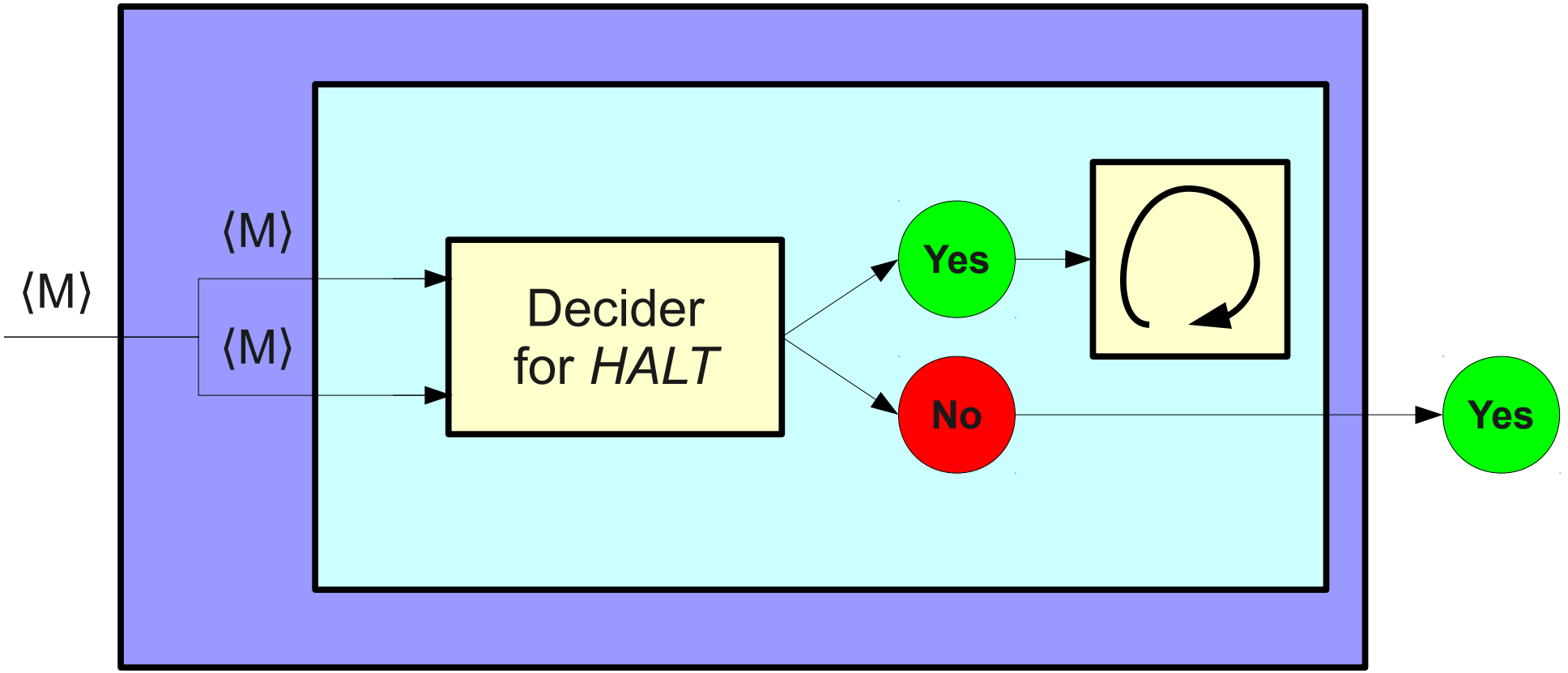

⟨M⟩
w
Decider
for *HALT*
Yes
No
Yes

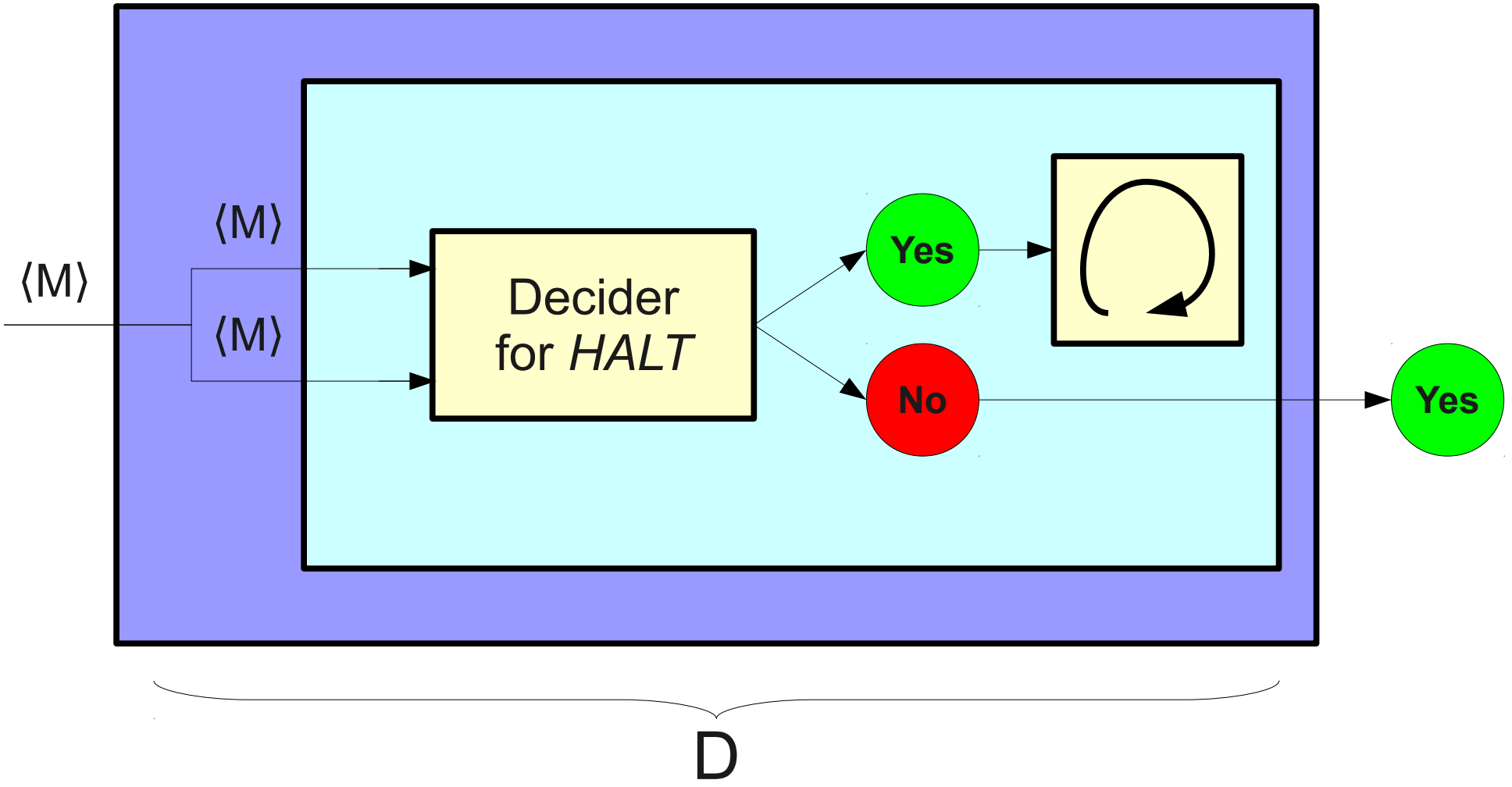

⟨M⟩
w
Decider
for *HALT*
Yes
No
Yes
M'

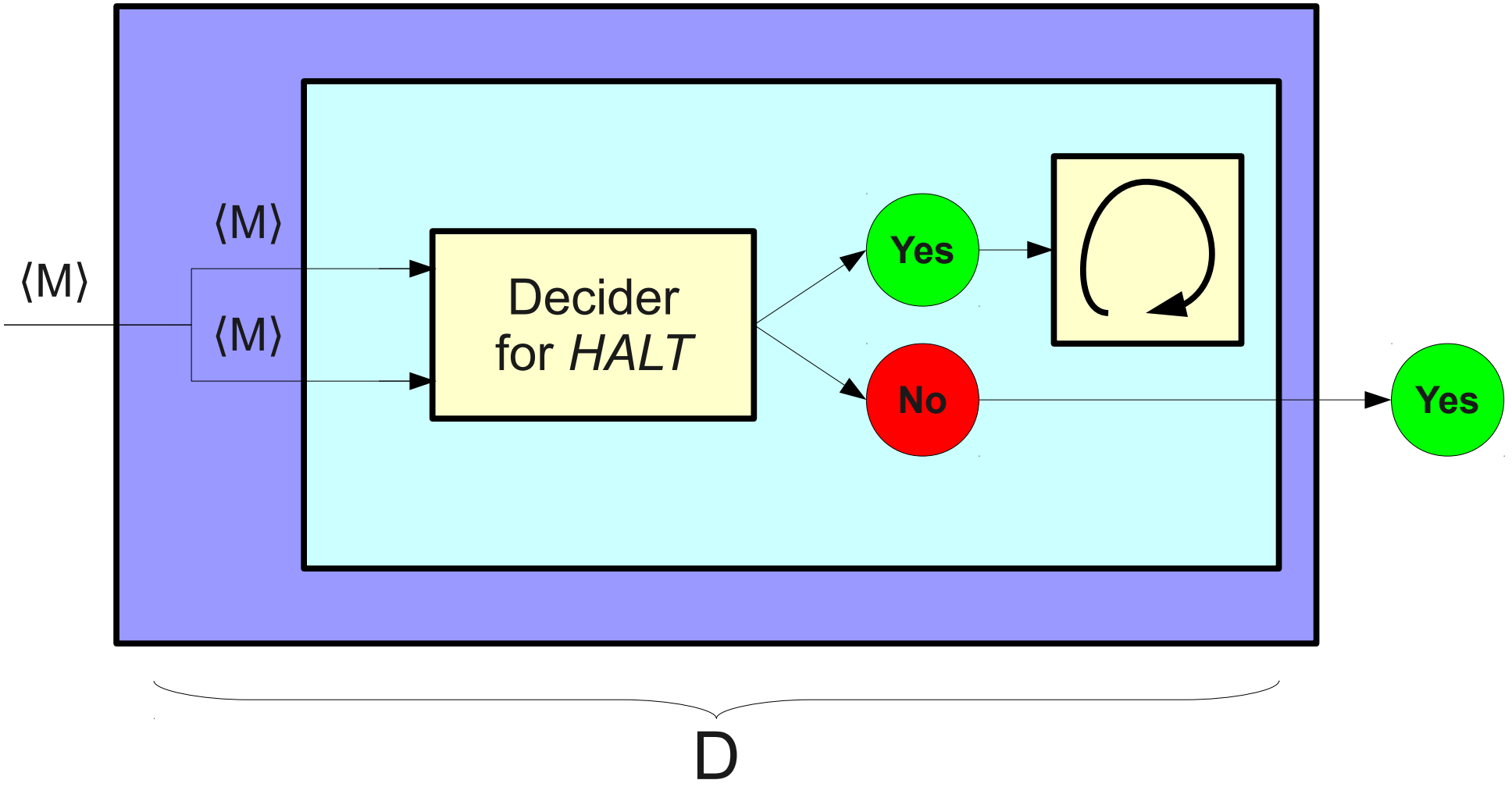


M' = “On input ⟨M, w⟩,
If M halts on w, loop infinitely.
If M loops on w, accept.

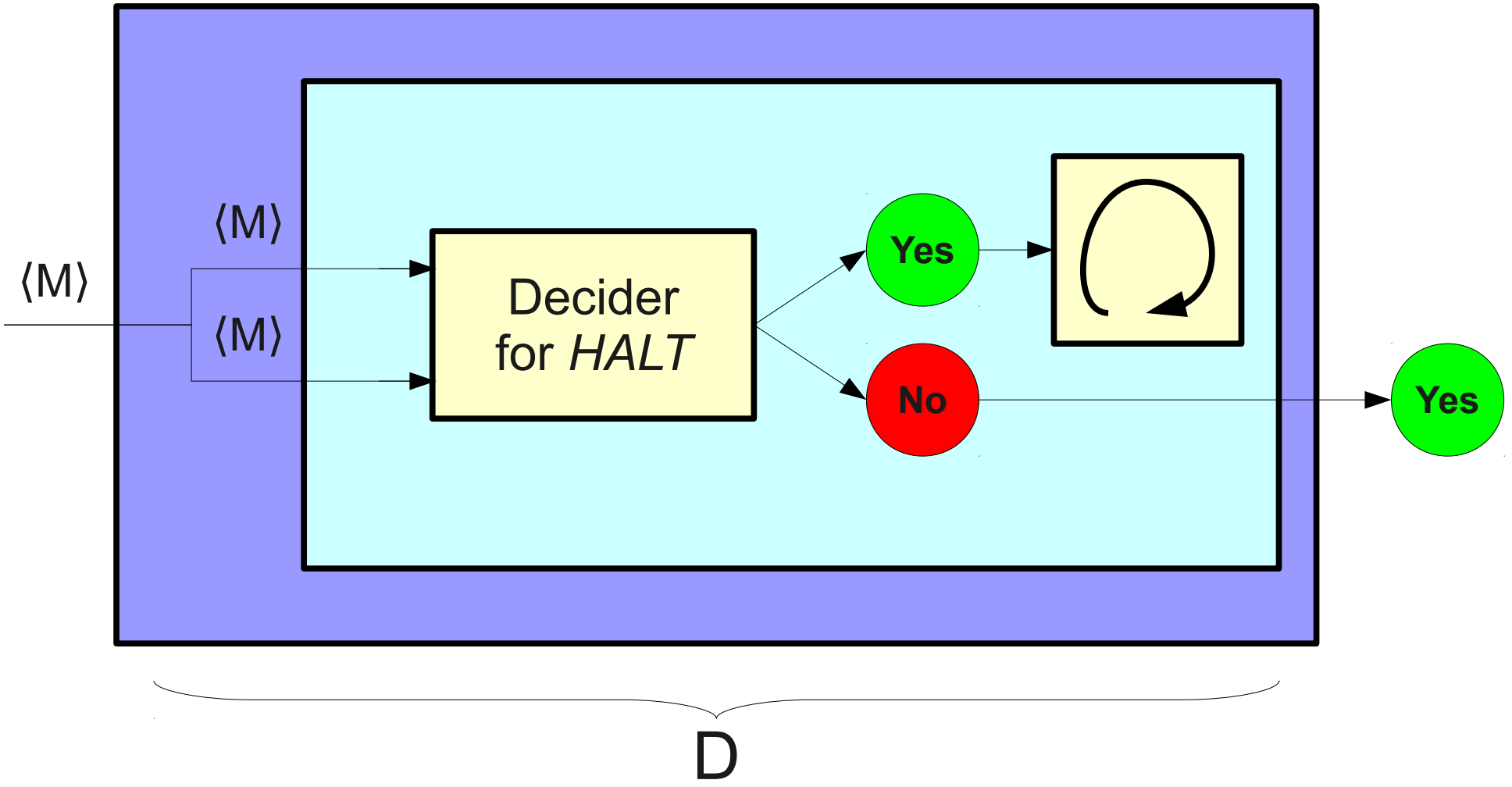

⟨M⟩
w
Decider
for *HALT*
Yes
No
Yes


⟨M⟩
⟨M⟩
⟨M⟩
Decider
for *HALT*
Yes
No
Yes

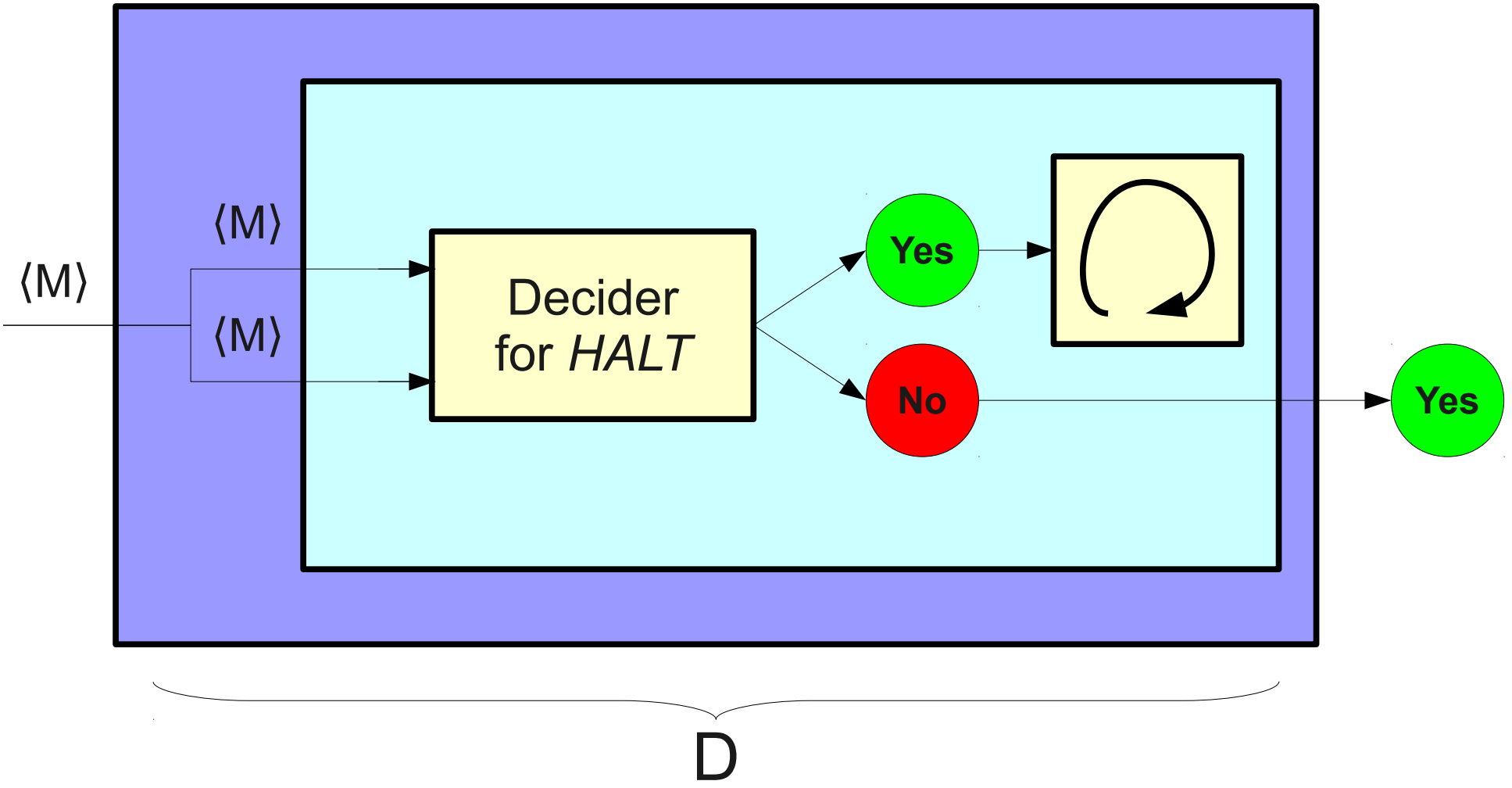

⟨M⟩
⟨M⟩
⟨M⟩
Decider
for HALT
Yes
No
Yes
D



D = “On input ⟨M⟩,



D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.

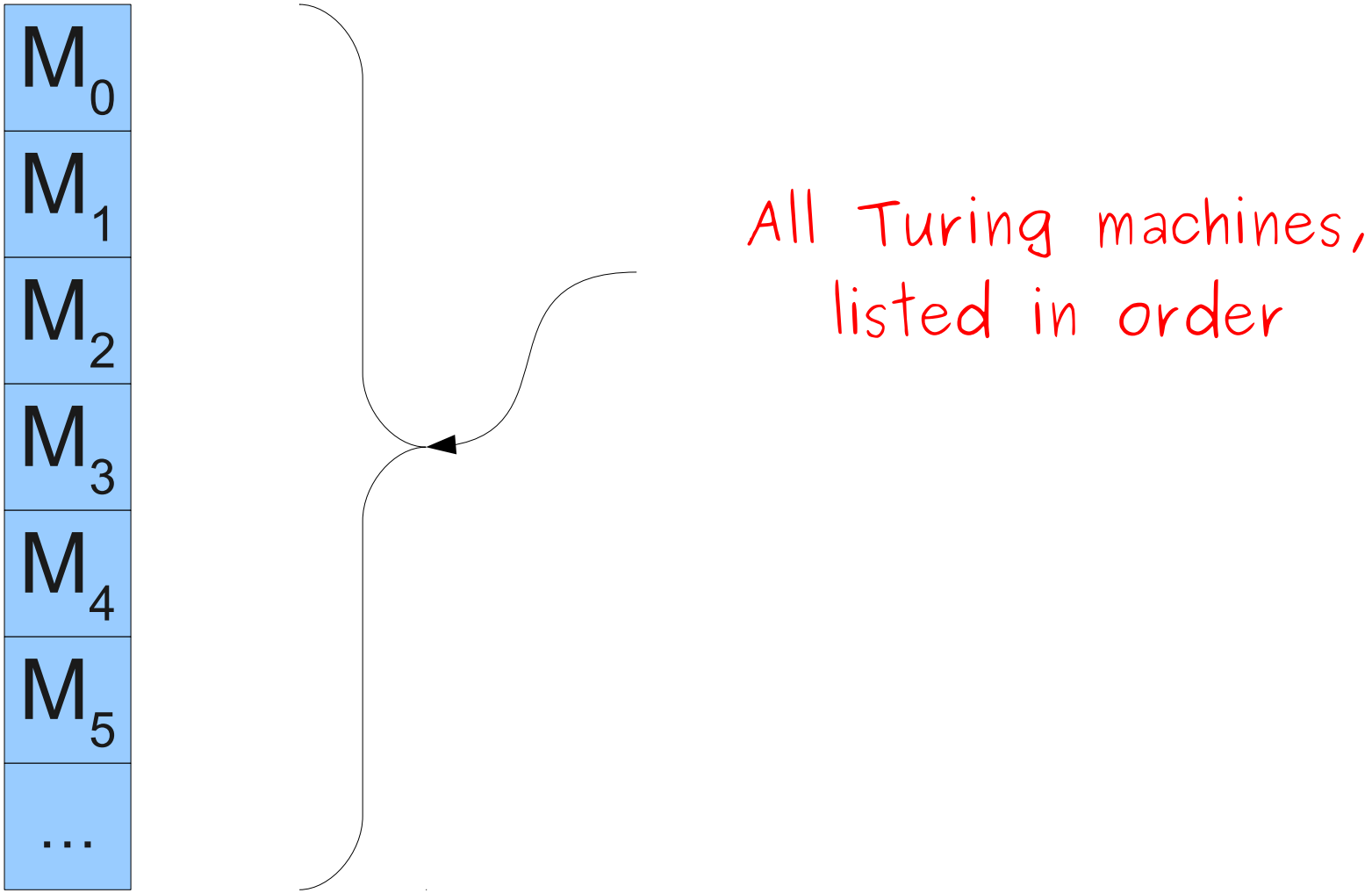


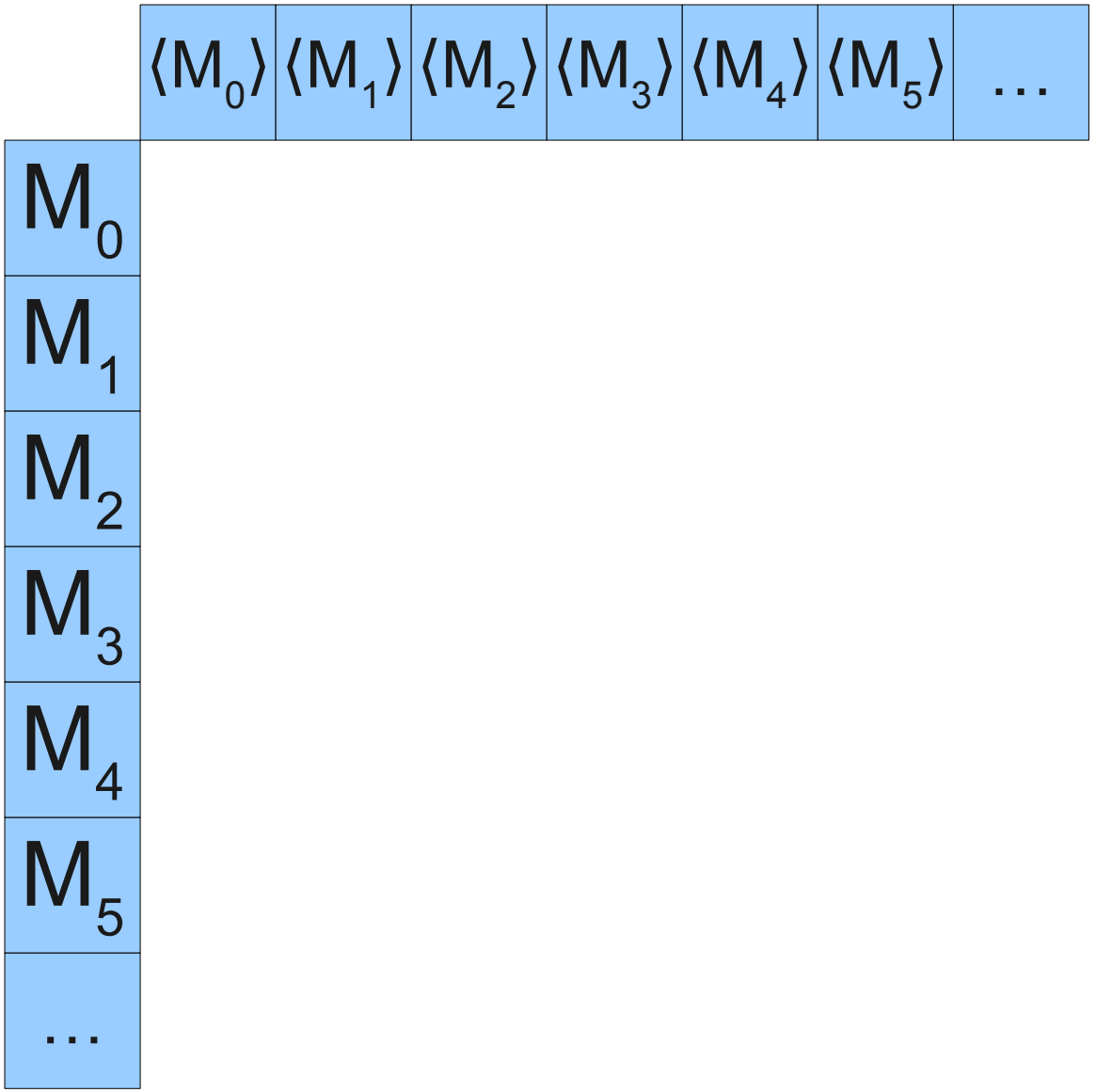
D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

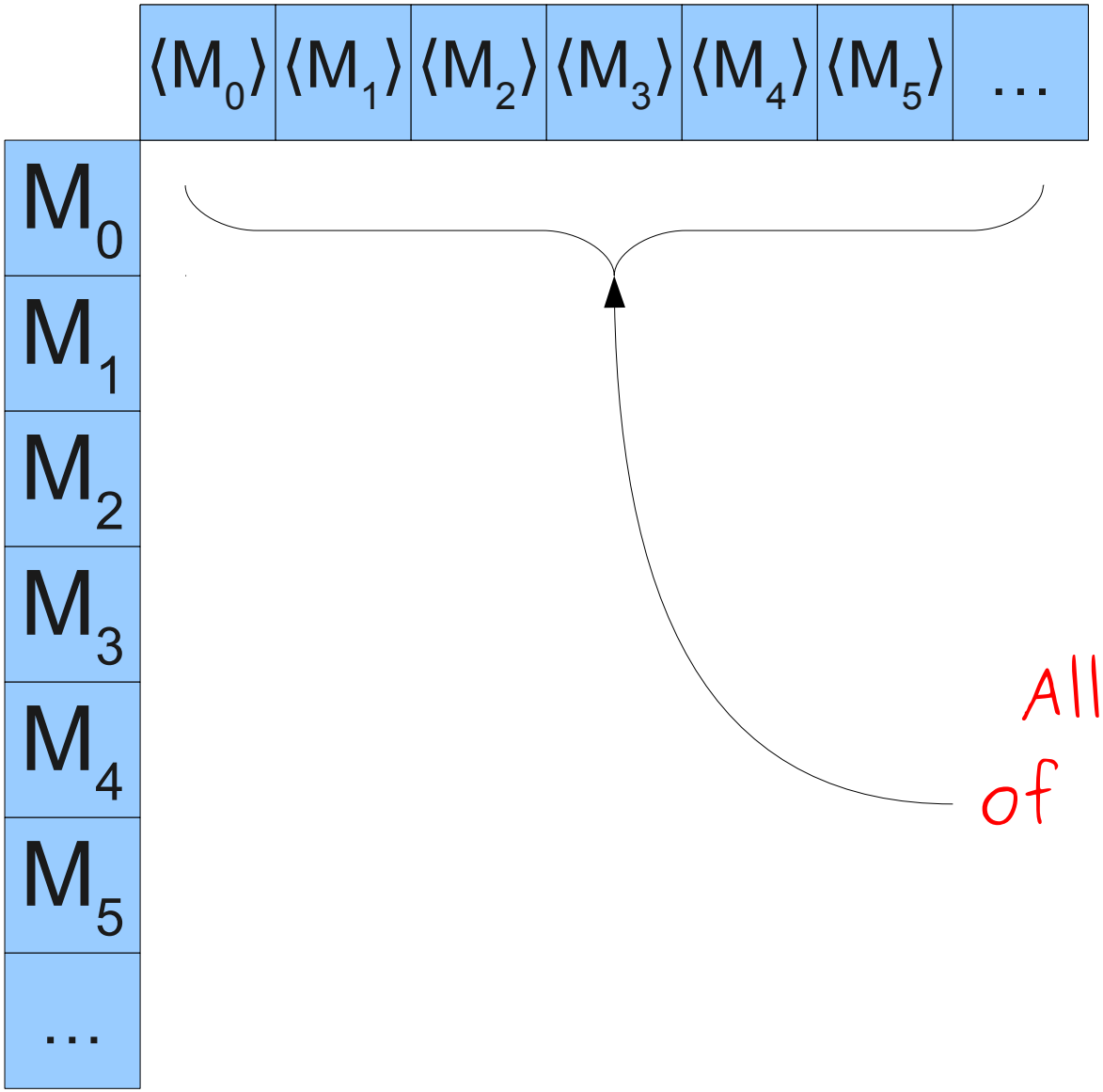
If *HALT* is decidable, we can build
a TM that does the following on input ⟨M⟩:

Accept if M loops infinitely on ⟨M⟩
Loop infinitely if M halts on ⟨M⟩


M_0
M_1
M_2
M_3
M_4
M_5
...


M_0
M_1
M_2
M_3
M_4
M_5
...
All Turing machines,
listed in order


⟨M0⟩
⟨M1⟩
⟨M2⟩
⟨M3⟩
⟨M4⟩
⟨M5⟩
…
M0
M1
M2
M3
M4
M5
…



All descriptions of TMs, listed in order.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1							
M_2							
M_3							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2							
M_3							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3							
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4							
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5							
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

H	H	H	L	H	L	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…
	H	H	H	L	H	L	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

This is the language L(D)

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

What TM has this behavior?

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…
	L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…
	L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…
	L	L	L	H	L	H	…

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

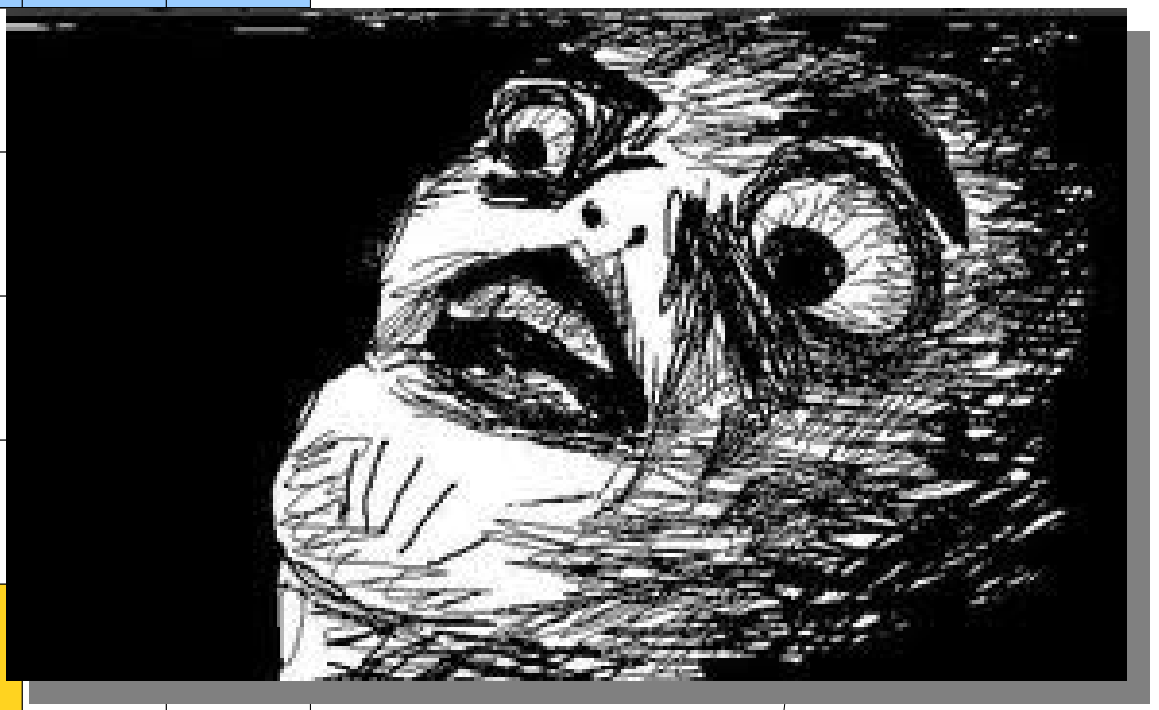
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H	L	…
M_1	H	H	H	H	H	H	…
M_2	H	H	H	H	H	H	…
M_3	L	H	H	L	H	H	…
M_4	H	L	H	L	H	L	…
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…

L	L	L	H	L	H	…

No TM has this behavior!

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	H	L	L	H	H		
M_1	H	H	H	H	H		
M_2	H	H	H	H	H		
M_3	L	H	H	L	H		
M_4	H	L	H	L	H		
M_5	L	L	H	H	L	L	…
…	…	…	…	…	…	…	…



L	L	L	H	L	H	…

HALT is Undecidable

- If *HALT* is decidable, then we can build a TM D that works as follows:

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.

HALT is Undecidable

- If *HALT* is decidable, then we can build a TM D that works as follows:

  D = “On input ⟨M⟩,
  If M halts on ⟨M⟩, loop infinitely.
  If M loops on ⟨M⟩, accept.

- What happens when we run D on ⟨D⟩?

HALT is Undecidable

- If *HALT* is decidable, then we can build a TM D that works as follows:

  D = “On input ⟨**D**⟩,
  If **D** halts on ⟨**D**⟩, loop infinitely.
  If **D** loops on ⟨**D**⟩, accept.

- What happens when we run D on ⟨D⟩?

HALT is Undecidable

- If *HALT* is decidable, then we can build a TM D that works as follows:

  D = "On input ⟨**D**⟩,
  If **D** halts on ⟨**D**⟩, loop infinitely.
  If **D** loops on ⟨**D**⟩, accept.

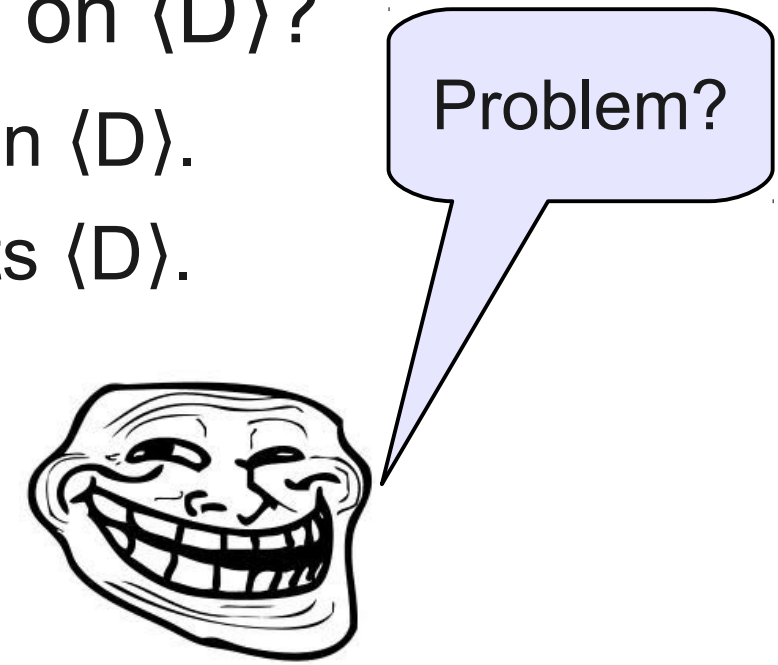
- What happens when we run D on ⟨D⟩?
  - If D halts on ⟨D⟩, then D loops on ⟨D⟩.
  - If D loops on ⟨D⟩, then D accepts ⟨D⟩.

HALT is Undecidable

- If *HALT* is decidable, then we can build a TM D that works as follows:

  D = "On input ⟨**D**⟩,
  If **D** halts on ⟨**D**⟩, loop infinitely.
  If **D** loops on ⟨**D**⟩, accept.

- What happens when we run D on ⟨D⟩?
  - If D halts on ⟨D⟩, then D loops on ⟨D⟩.
  - If D loops on ⟨D⟩, then D accepts ⟨D⟩.



Theorem: HALT is undecidable.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Then, using H', construct the machine D as follows:

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Then, using H', construct the machine D as follows:

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.”

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Then, using H', construct the machine D as follows:

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.”

Consider what happens when we run D on ⟨D⟩.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Then, using H', construct the machine D as follows:

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.”

Consider what happens when we run D on ⟨D⟩. If D accepts ⟨D⟩, then this means that D loops on ⟨D⟩, a contradiction.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Then, using H', construct the machine D as follows:

D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.”

Consider what happens when we run D on ⟨D⟩. If D accepts ⟨D⟩, then this means that D loops on ⟨D⟩, a contradiction. Otherwise, if D loops on ⟨D⟩, then this means that D halts on ⟨D⟩, a contradiction.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = "On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept."

Then, using H', construct the machine D as follows:

D = "On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept."

Consider what happens when we run D on ⟨D⟩. If D accepts ⟨D⟩, then this means that D loops on ⟨D⟩, a contradiction. Otherwise, if D loops on ⟨D⟩, then this means that D halts on ⟨D⟩, a contradiction. In each case we reach a contradiction, so our assumption was wrong.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = "On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept."

Then, using H', construct the machine D as follows:

D = "On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept."

Consider what happens when we run D on ⟨D⟩. If D accepts ⟨D⟩, then this means that D loops on ⟨D⟩, a contradiction. Otherwise, if D loops on ⟨D⟩, then this means that D halts on ⟨D⟩, a contradiction. In each case we reach a contradiction, so our assumption was wrong. Thus *HALT* is undecidable.

Theorem: HALT is undecidable.
Proof: By contradiction; assume that *HALT* is decidable. Then there is a decider H for *HALT*. Using H, construct the machine H' as follows:

H' = “On input ⟨M, w⟩:
If M halts on w, loop infinitely.
If M loops on w, accept.”

Then, using H', construct the machine D as follows:

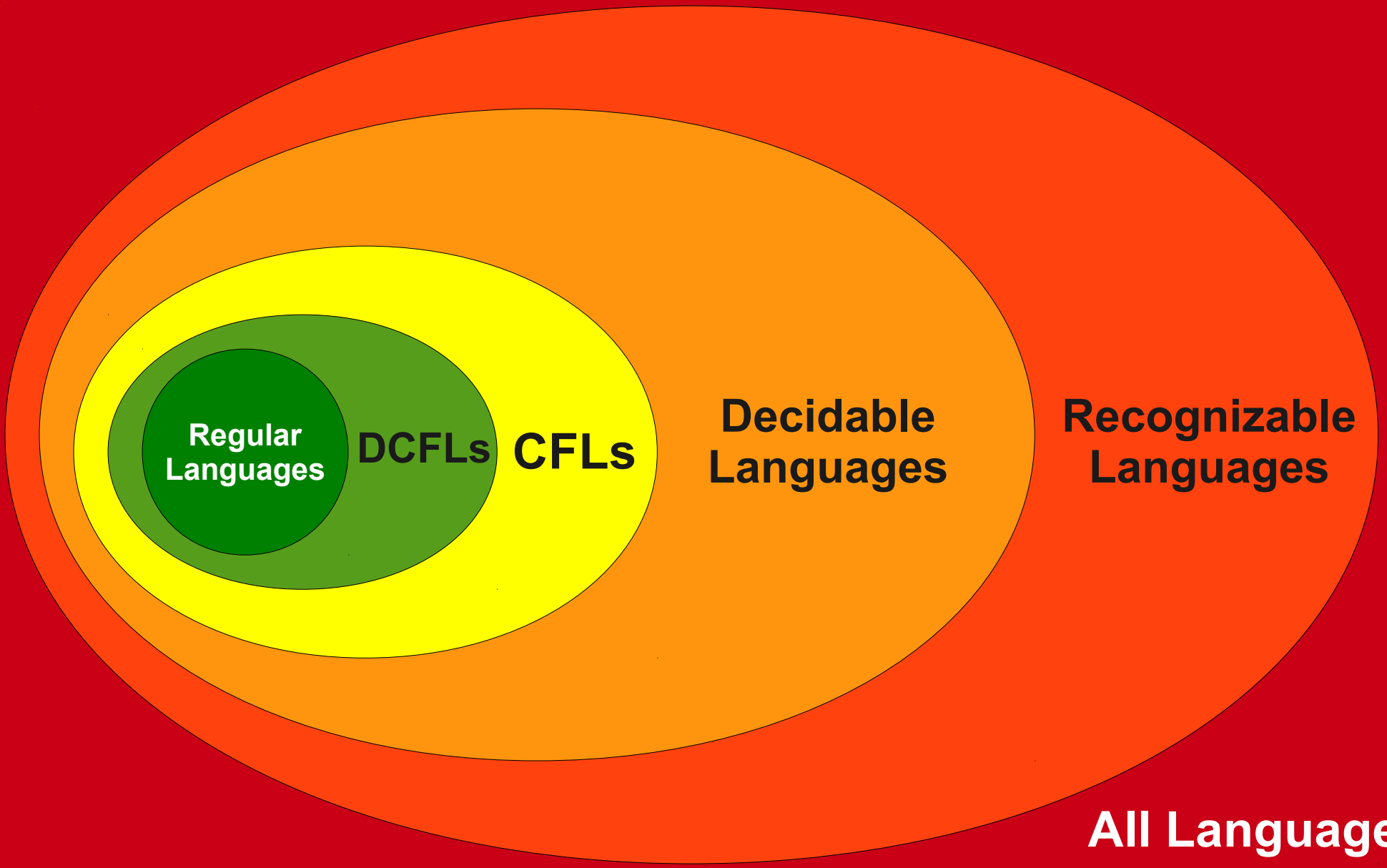
D = “On input ⟨M⟩,
If M halts on ⟨M⟩, loop infinitely.
If M loops on ⟨M⟩, accept.”

Consider what happens when we run D on ⟨D⟩. If D accepts ⟨D⟩, then this means that D loops on ⟨D⟩, a contradiction. Otherwise, if D loops on ⟨D⟩, then this means that D halts on ⟨D⟩, a contradiction. In each case we reach a contradiction, so our assumption was wrong. Thus *HALT* is undecidable. ■

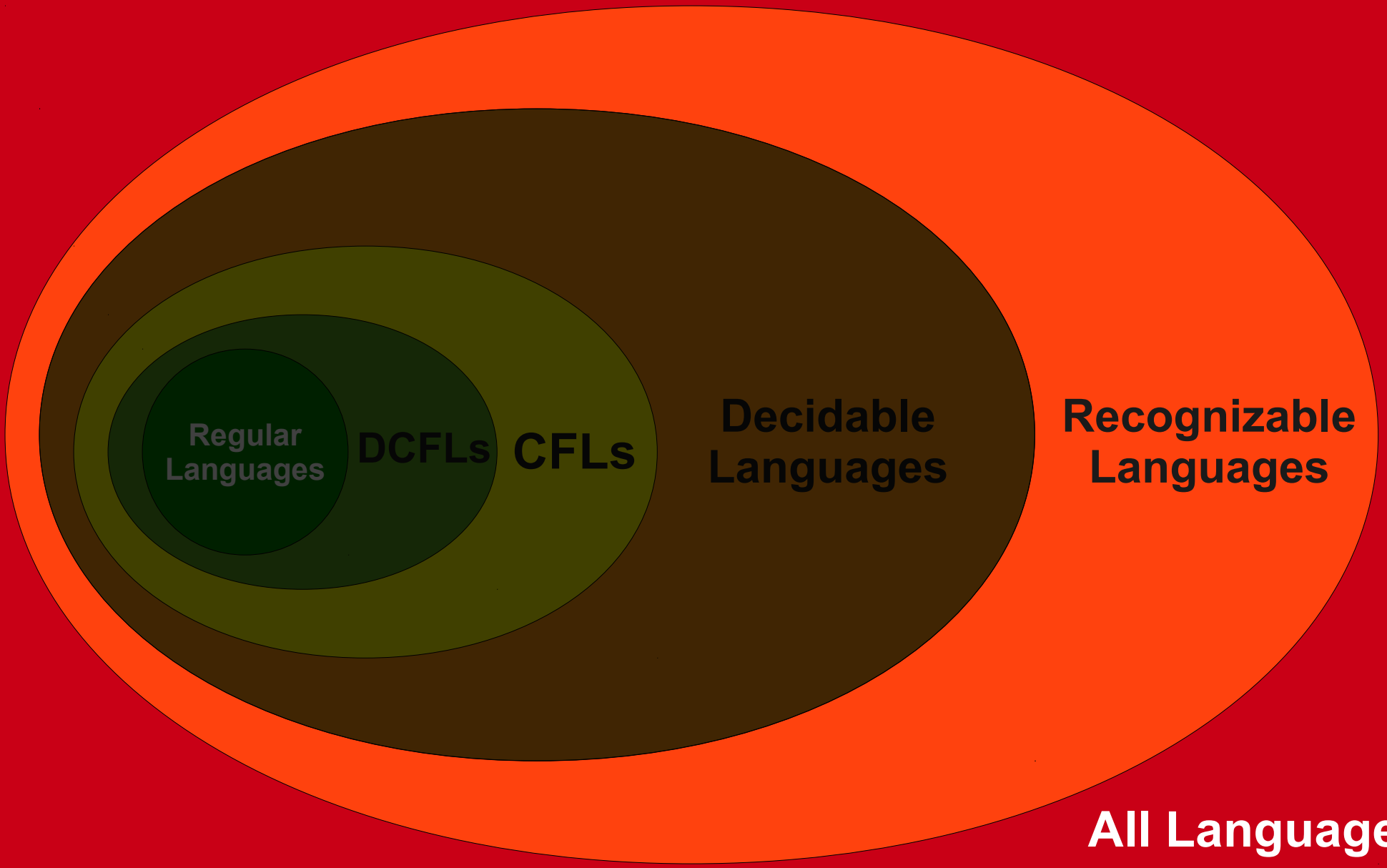
Indirect Self-Reference

- The proof that *HALT* is undecidable, while more complex than the proof that L_D is unrecognizable, uses the same basic trick.
- Assume, for the sake of contradiction, that you can build a TM that operates over descriptions of TMs.
- Feed this TM its own description.
- Derive a contradiction where the TM must have the opposite of its own behavior.

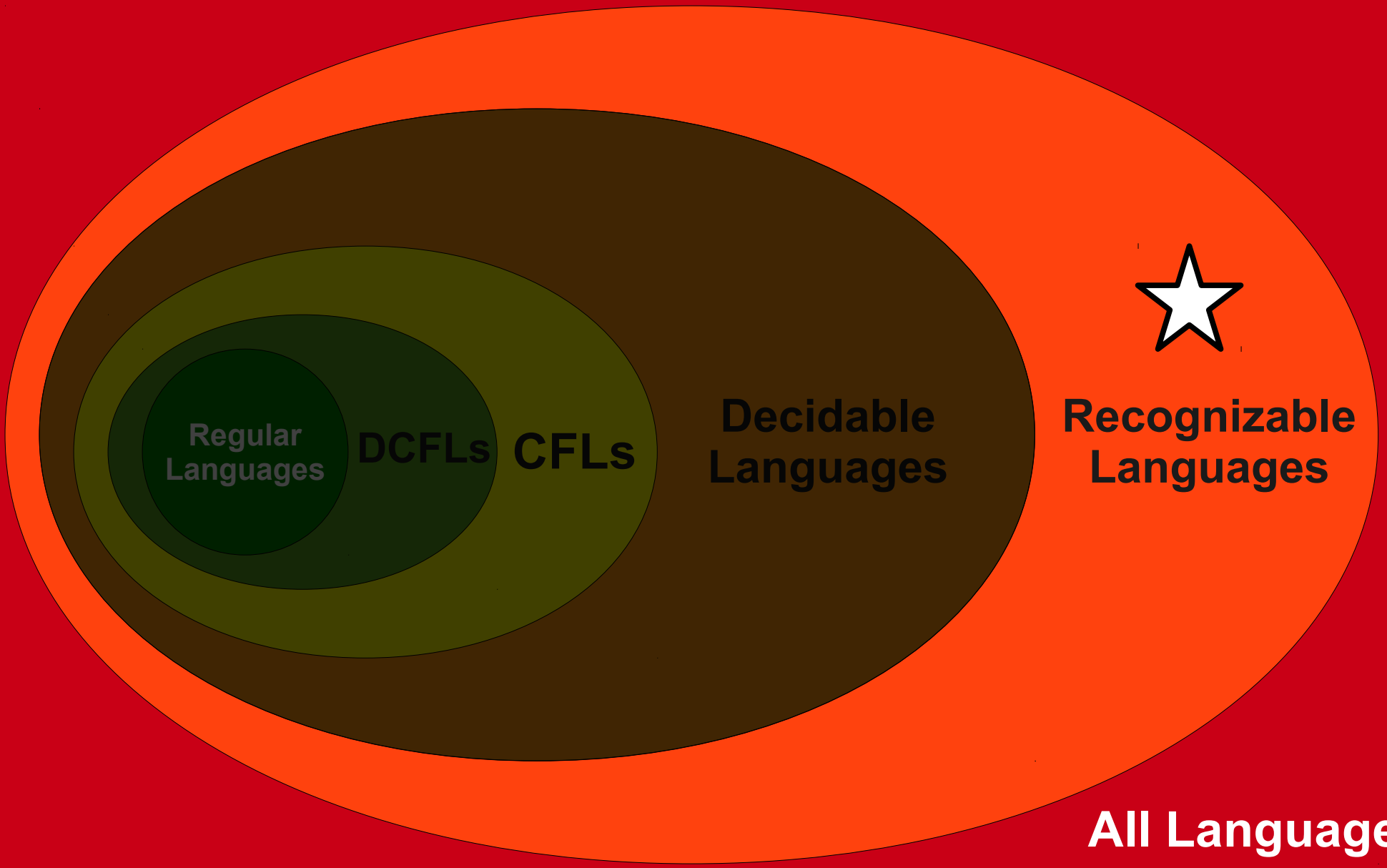
Where is *HALT*?



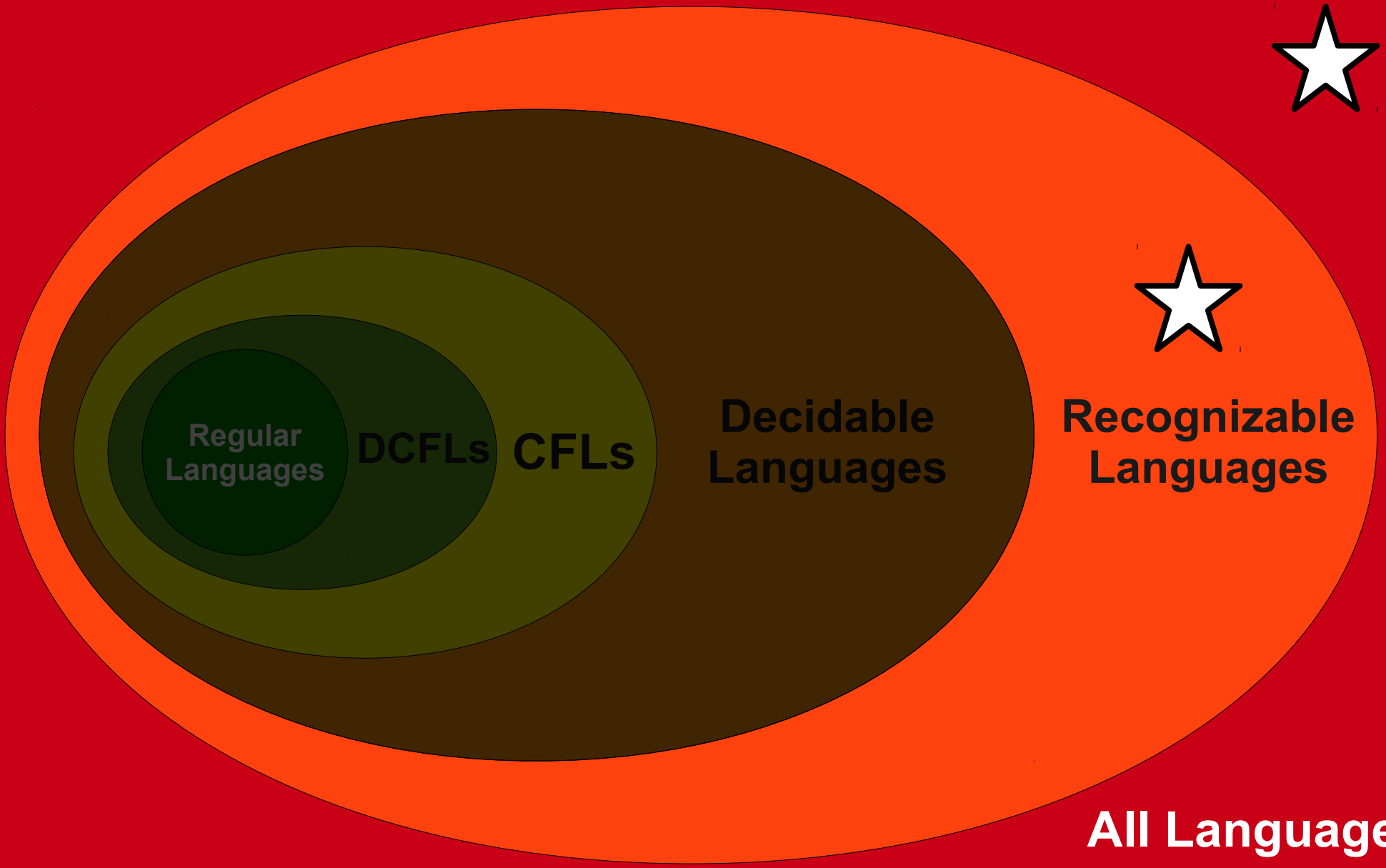
Where is *HALT*?



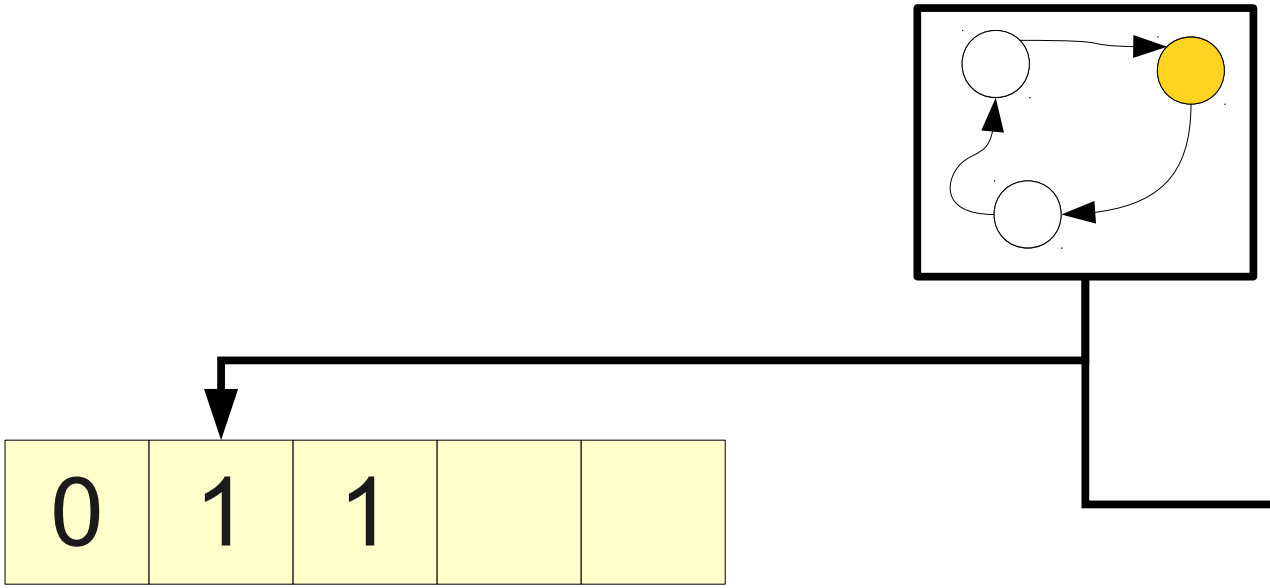
Where is *HALT*?

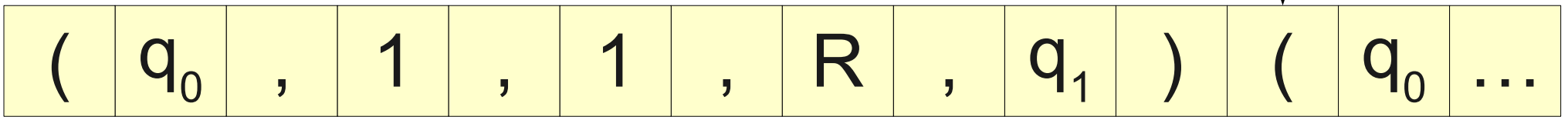


Where is *HALT*?



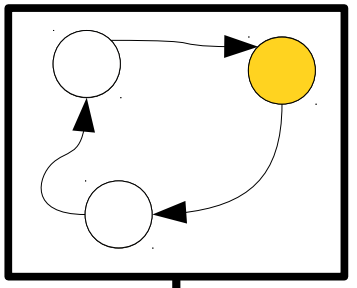
The Universal Turing Machine

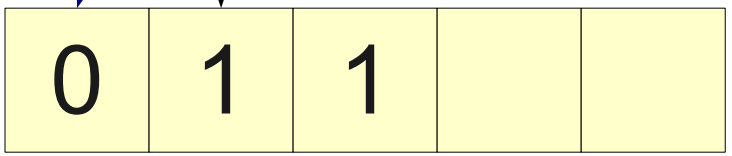


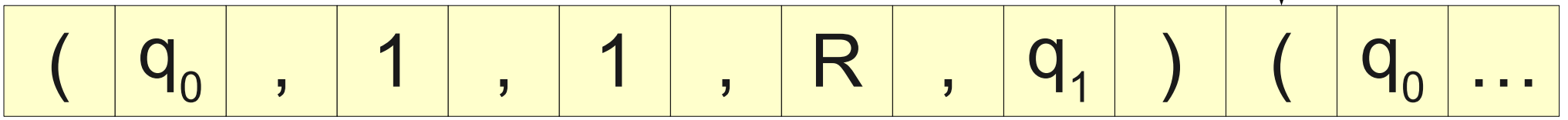


The Universal Turing Machine

The work tape holds the infinite tape from the TM being simulated.

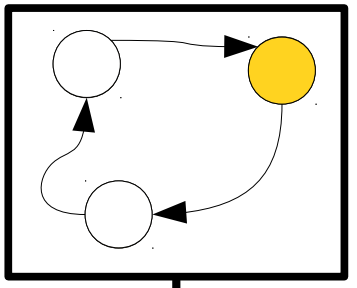






The Universal Turing Machine

The work tape holds the infinite tape from the TM being simulated.



0	1	1		

(	q_0	,	1	,	1	,	R	,	q_1	)	(	q_0	...

The program tape holds the description of the TM to simulate.

The Universal Turing Machine

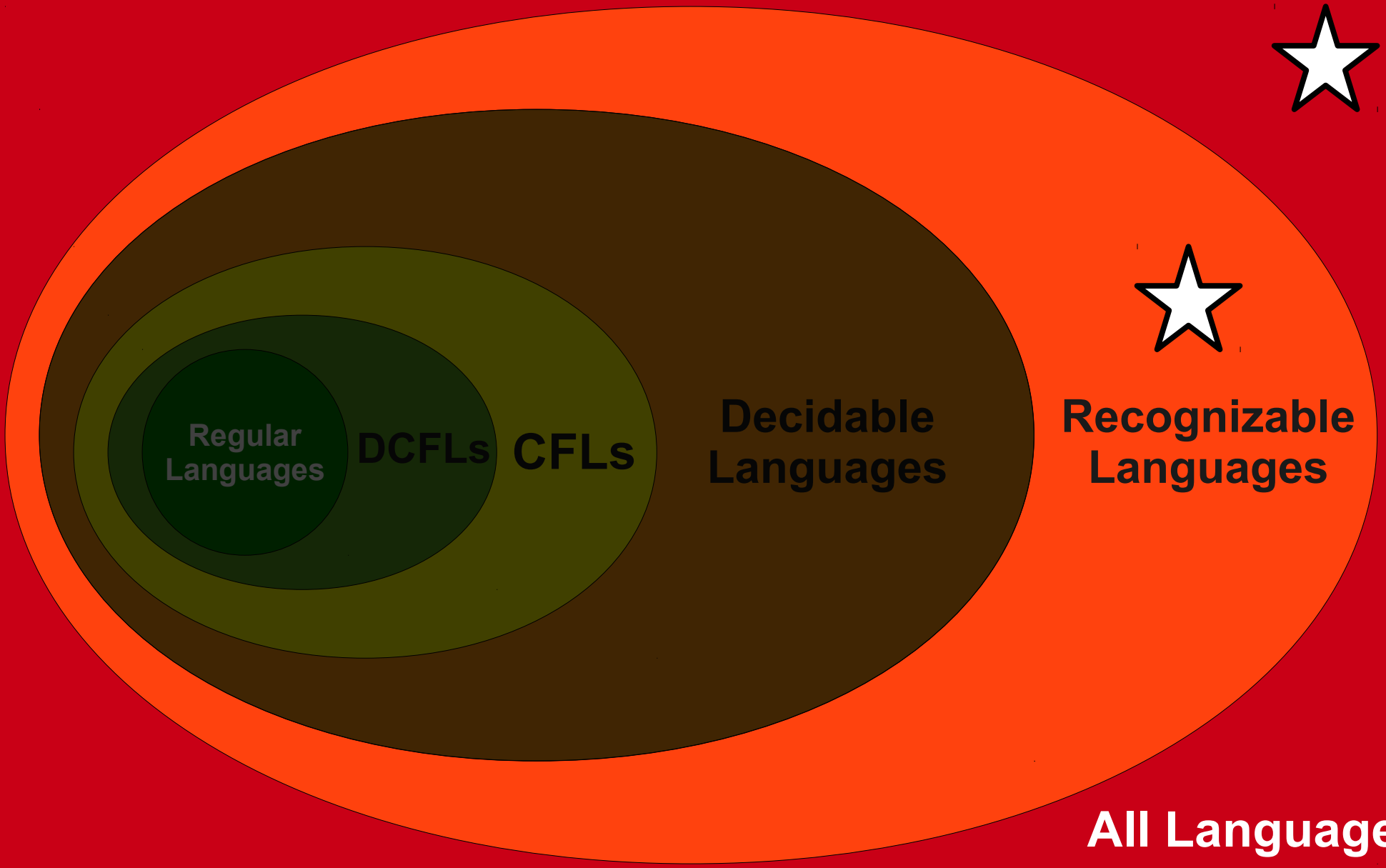
- The **universal Turing machine** U is a Turing machine that accepts ⟨M, *w*⟩ (an encoding of a machine M and string w) and simulates the behavior of M running on *w*.
- U accepts iff M accepts *w*.
- U rejects iff M rejects *w*.
- U loops iff M loops on *w*.
- The language of U is called A_{TM}:

$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ accepts } w \}$$

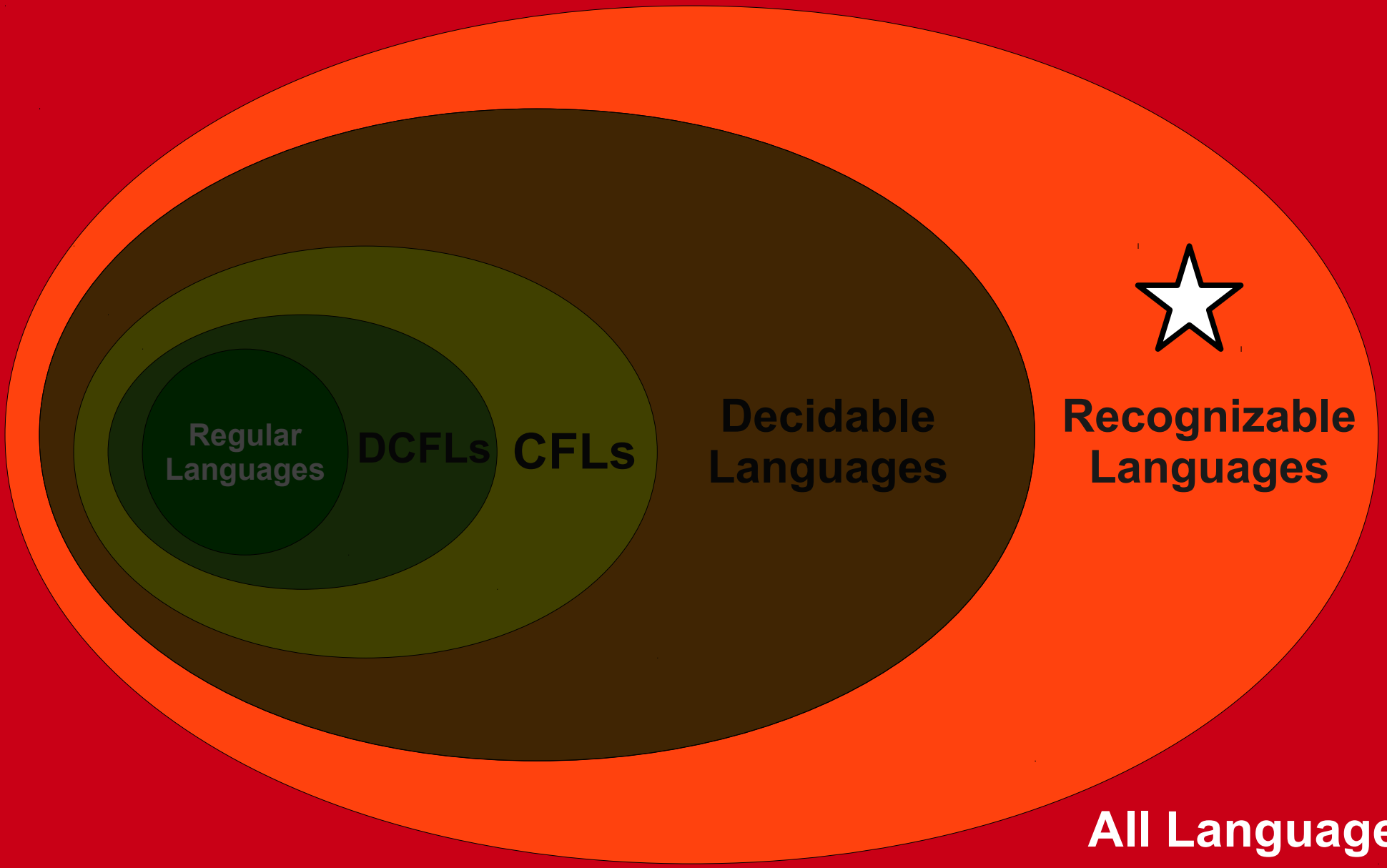
HALT is Recognizable

- We can **recognize** *HALT* by using the universal Turing machine.
- Construct a TM U' that works as follows:
  - U' = “On input ⟨M, w⟩:
    - Using U, run M on w.
    - If M accepts w, accept.
    - If M rejects w, accept.”
- Note that U' is not a decider, because M might loop forever on w.

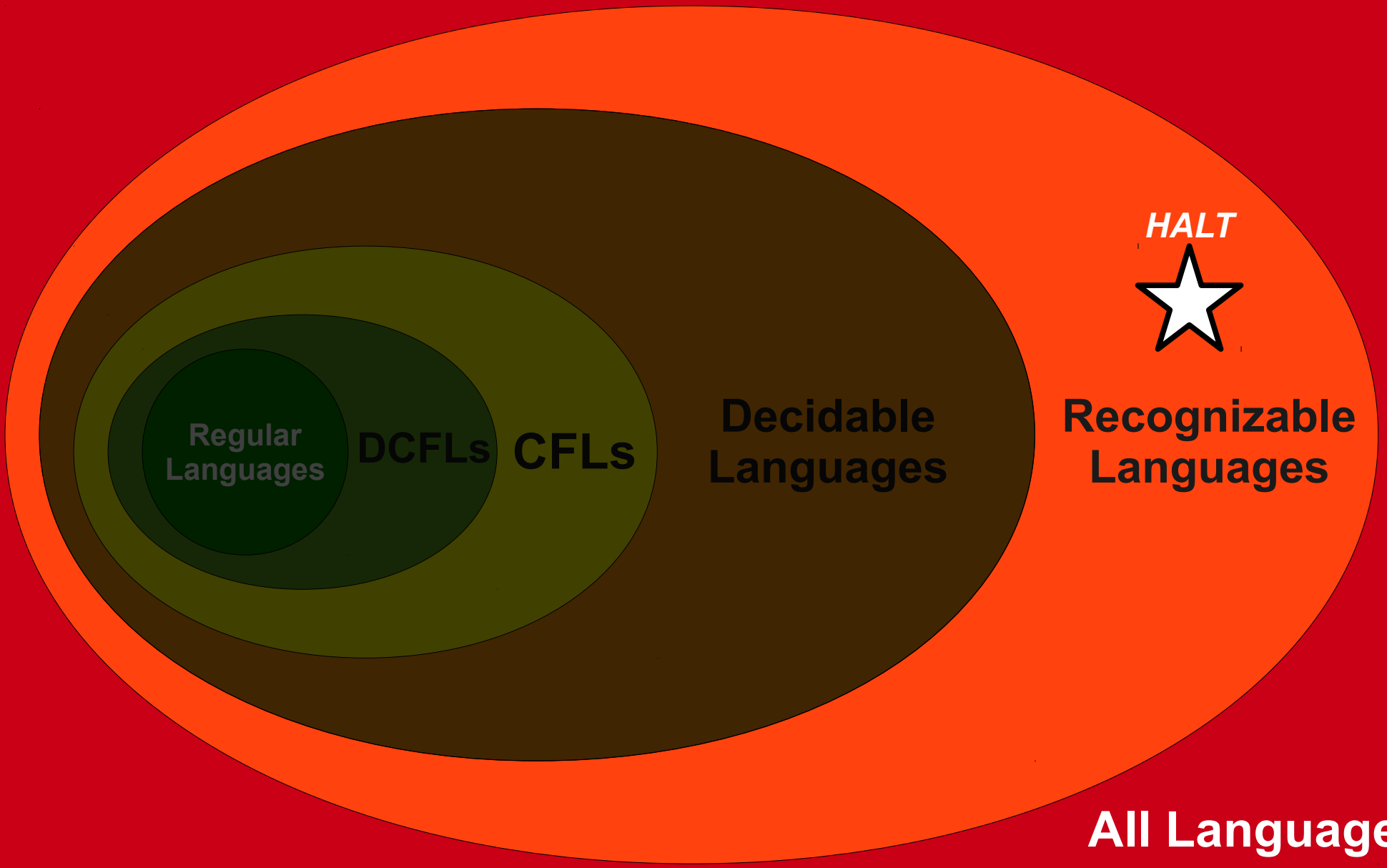
Where is *HALT*?



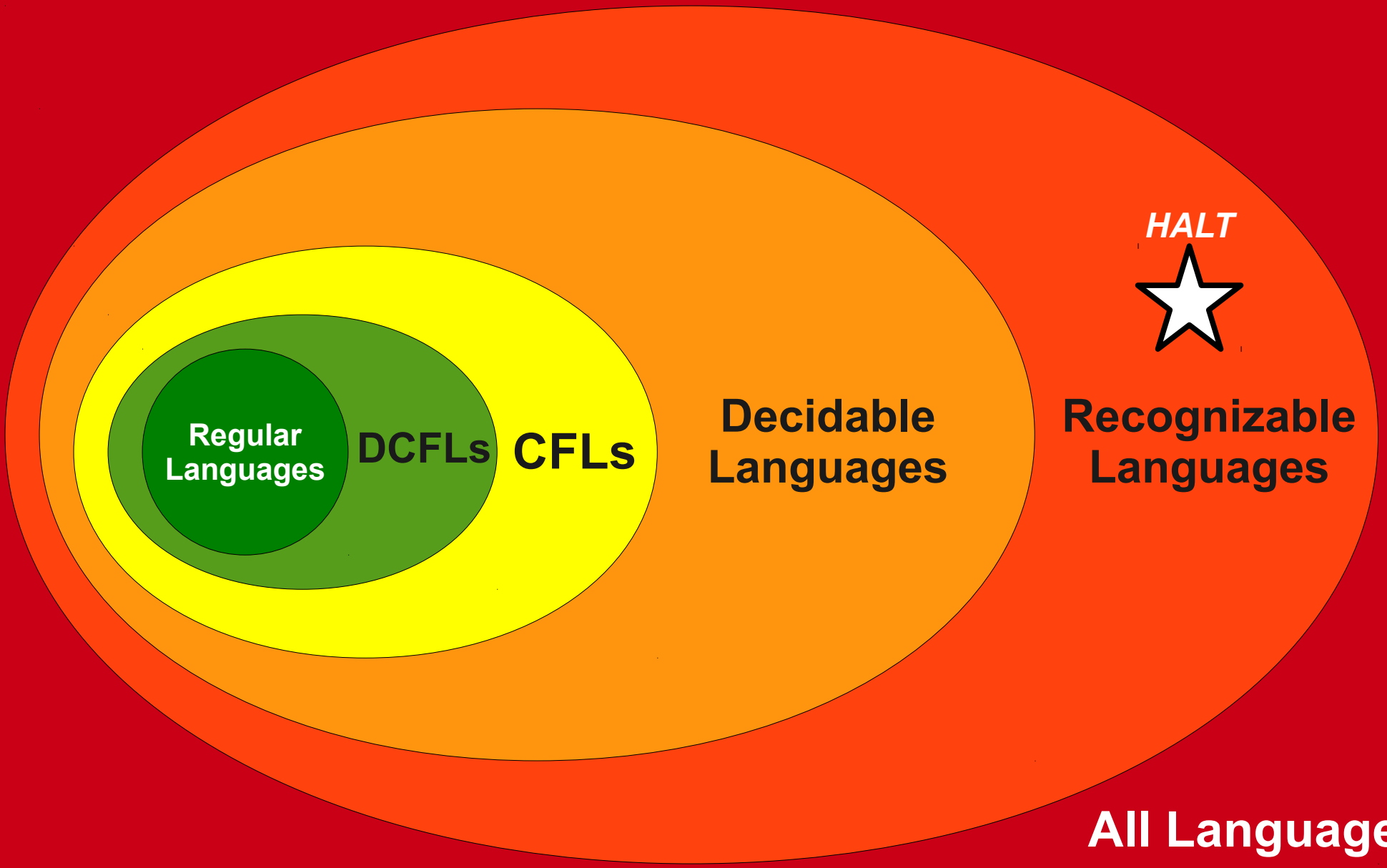
Where is *HALT*?



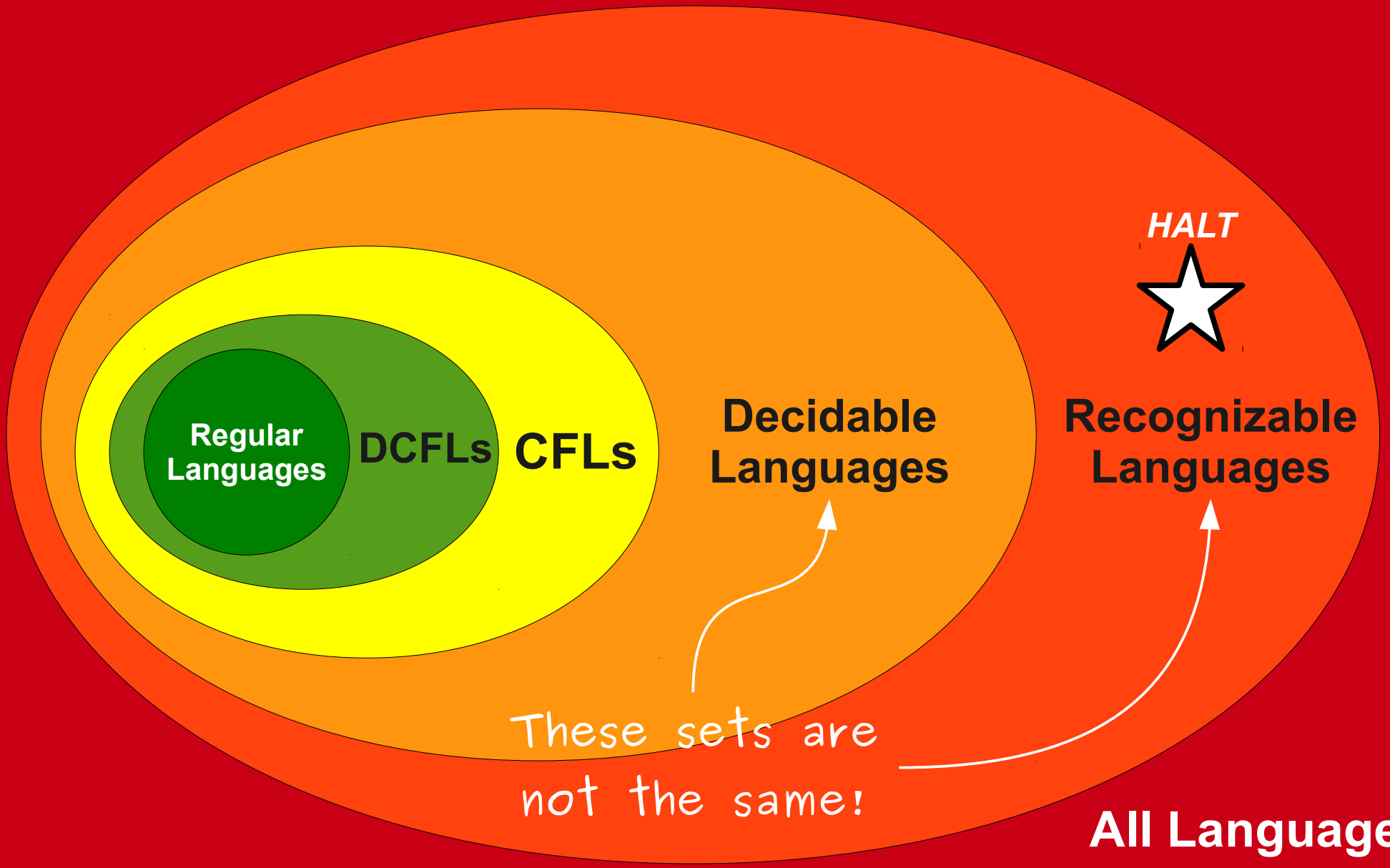
Where is *HALT*?



Where is *HALT*?



Where is *HALT*?



The Halting Problem is Undecidable

- The fact that *HALT* is undecidable immediately gives some important results.
  - There are problems that are **recognizable** but not **decidable**.
  - There is no general procedure for converting a **recognizer** into a **decider**.
  - As we will see, there is **no general algorithm** for reasoning about what a given program will or will not do.
- This is one of the most famous and important proofs in computer science.

Summary

- The **diagonalization language** L_D is not Turing-recognizable.
  - No feasible computing device is powerful enough to even *verify* that a string is in L_D!
- The **halting problem** *HALT* is Turing-recognizable, but not Turing-decidable.
  - We can confirm that a TM halts on some input by simulating it to see what happens.
  - However, we can never **decide** whether a TM will halt on some input.

Next Time

- **Finding more unsolvable problems**.
  - Closure properties of recognizable and decidable languages.
  - Mapping reductions.