Unsolvable Problems

Part Two

Outline for Today

- ***Recap from Last Time***
  - Where are we, again?
- ***A Different Perspective on RE***
  - What exactly does “recognizability” mean?
- ***Verifiers***
  - A different perspective on problem-solving.
- ***Beyond RE***
  - Monster problems!

Recap from Last Time

Self-Referential Programs

- ***Claim:*** Any program can be augmented to include a method called `mySource()` that returns a string representation of its source code.
- ***Theorem:*** It it possible to build Turing machines that get their own encodings and perform arbitrary computations on them.

What does this program do?

```
bool willAccept(string program, string input) {
   /* … some implementation … */
}

int main() {
   string me = mySource();
   string input = getInput();

   if (willAccept(me, input)) {
      reject();
   } else {
      accept();
   }
}
```

What happens if...

... this program accepts its input?
It rejects the input!

... this program doesn't accept its input?
It accepts the input!

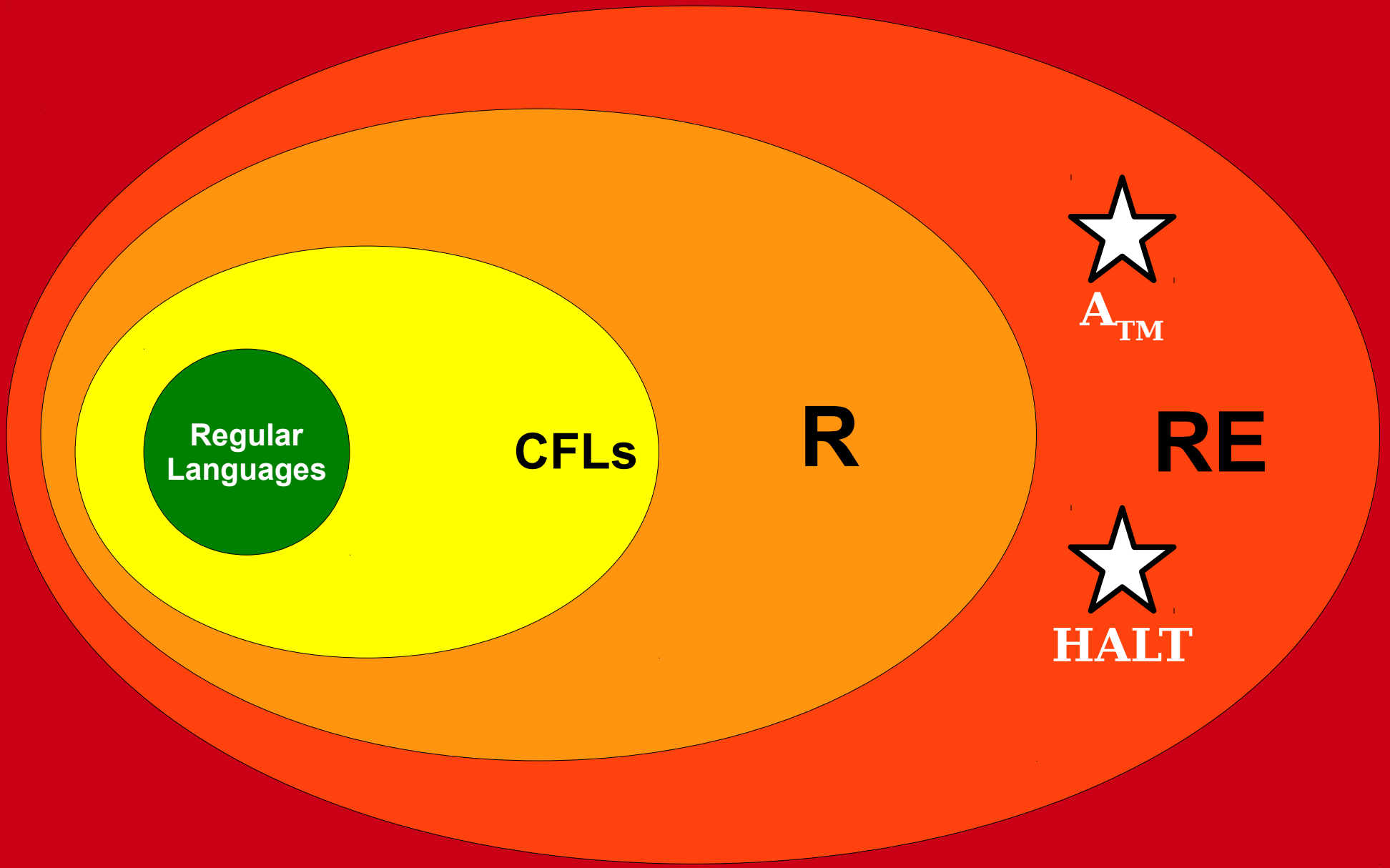
What does this program do?

```
bool willHalt(string program, string input) {
    /* … some implementation … */
}

int main() {
    string me = mySource();
    string input = getInput();

    if (willHalt(me, input)) {
        while (true) {
            // loop infinitely
        }
    } else {
        accept();
    }
}
```

What happens if...

… this program halts on this input?
It loops on the input!

… this program loops on this input?
It halts on the input!


Regular Languages
CFLs
R
RE
A_{TM}
HALT
All Languages

New Stuff!

Beyond **R** and **RE**

Beyond **R** and **RE**

- We've now seen how to use self-reference as a tool for showing undecidability (finding languages not in **R**).
- We still have not broken out of **RE** yet, though.
- To do so, we will need to build up a better intuition for the class **RE**.

What exactly is the class **RE**?

RE, Formally

- Recall that the class **RE** is the class of all recognizable languages:

  $$\mathbf{RE} = \{ L \mid \text{there is a TM } M \text{ where } \mathscr{L}(M) = L \}$$

- Since **R** ≠ **RE**, there is no general way to "solve" problems in the class **RE**, if by "solve" you mean "make a computer program that can always tell you the correct answer."
- So what exactly *are* the sorts of languages in **RE**?

Key Intuition:

A language L is in **RE** if, for any string w, if you are *convinced* that $w \in L$, there is some way you could prove that to someone else.

Verification

- ***Recall:*** When focusing on the **RE** languages, we need to abandon the idea that we can “solve” the problems we're looking at.
- Rather than *solving problems*, we can think about *checking answers*.

Verification

		7		6		1		
					3		5	2
3			1		5	9		7
6		5		3		8		9
	1						2	
8		2		1		5		4
1		3	2		7			8
5	7		4					
		4		8		7		

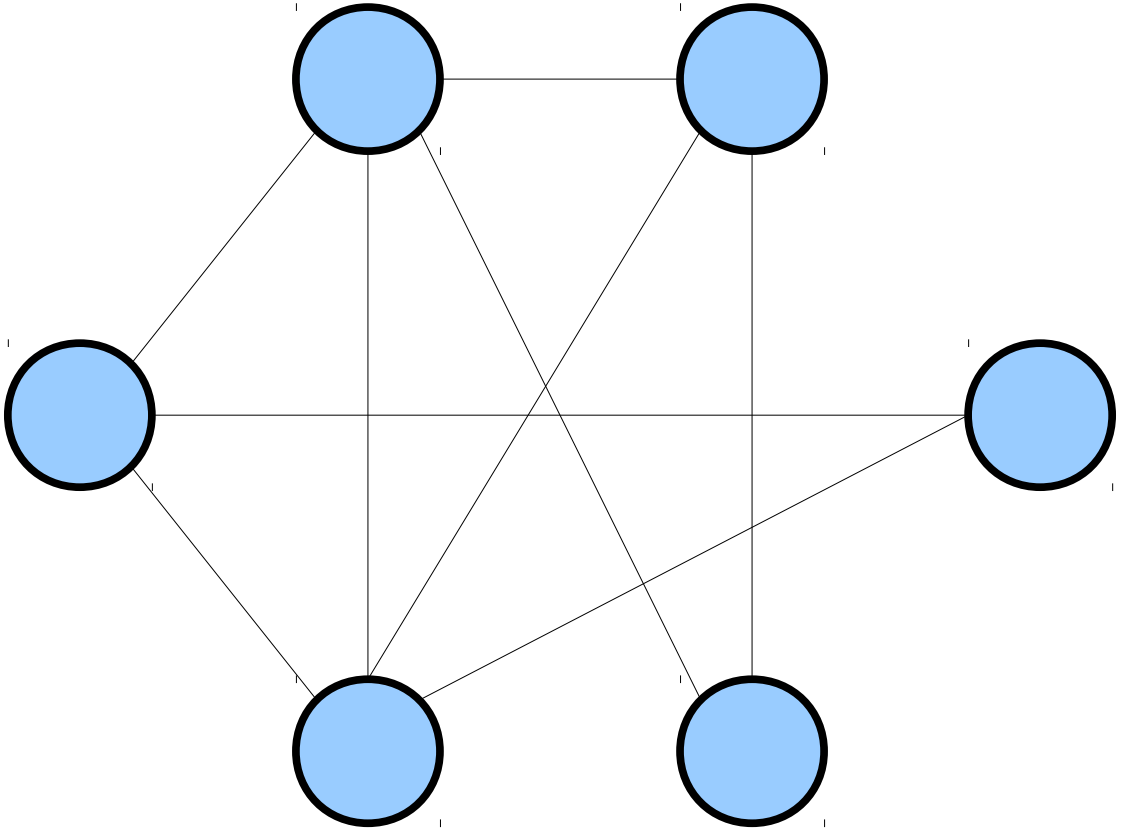
Does this Sudoku puzzle have a solution?

Verification

2	5	7	9	6	4	1	8	3
4	9	1	8	7	3	6	5	2
3	8	6	1	2	5	9	4	7
6	4	5	7	3	2	8	1	9
7	1	9	5	4	8	3	2	6
8	3	2	6	1	9	5	7	4
1	6	3	2	5	7	4	9	8
5	7	8	4	9	6	2	3	1
9	2	4	3	8	1	7	6	5

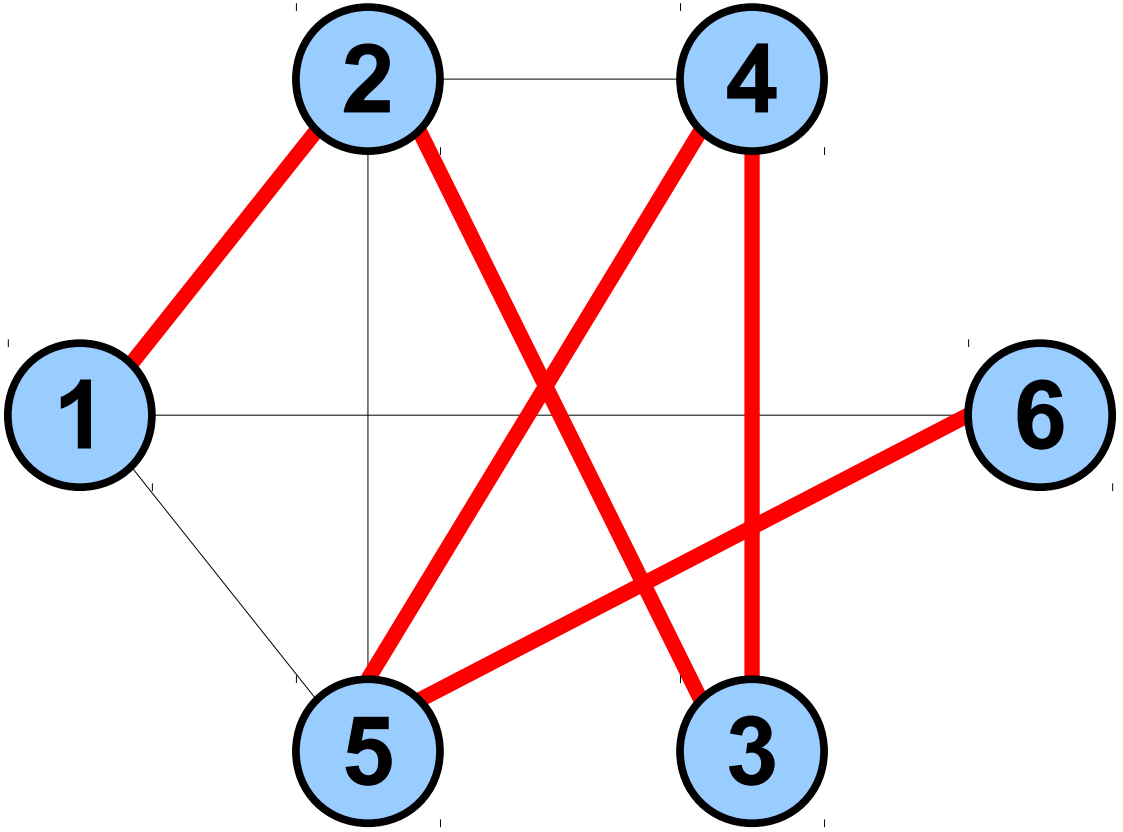
Does this Sudoku puzzle have a solution?

Verification



Is there a simple path that goes through every node exactly once?

Verification



Is there a simple path that goes through every node exactly once?

Verification

11

Does the hailstone sequence
terminate for this number?

Verification

11

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

34

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

17

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

52

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

26

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

13

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

40

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

20

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

10

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

5

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

16

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

8

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

4

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

2

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

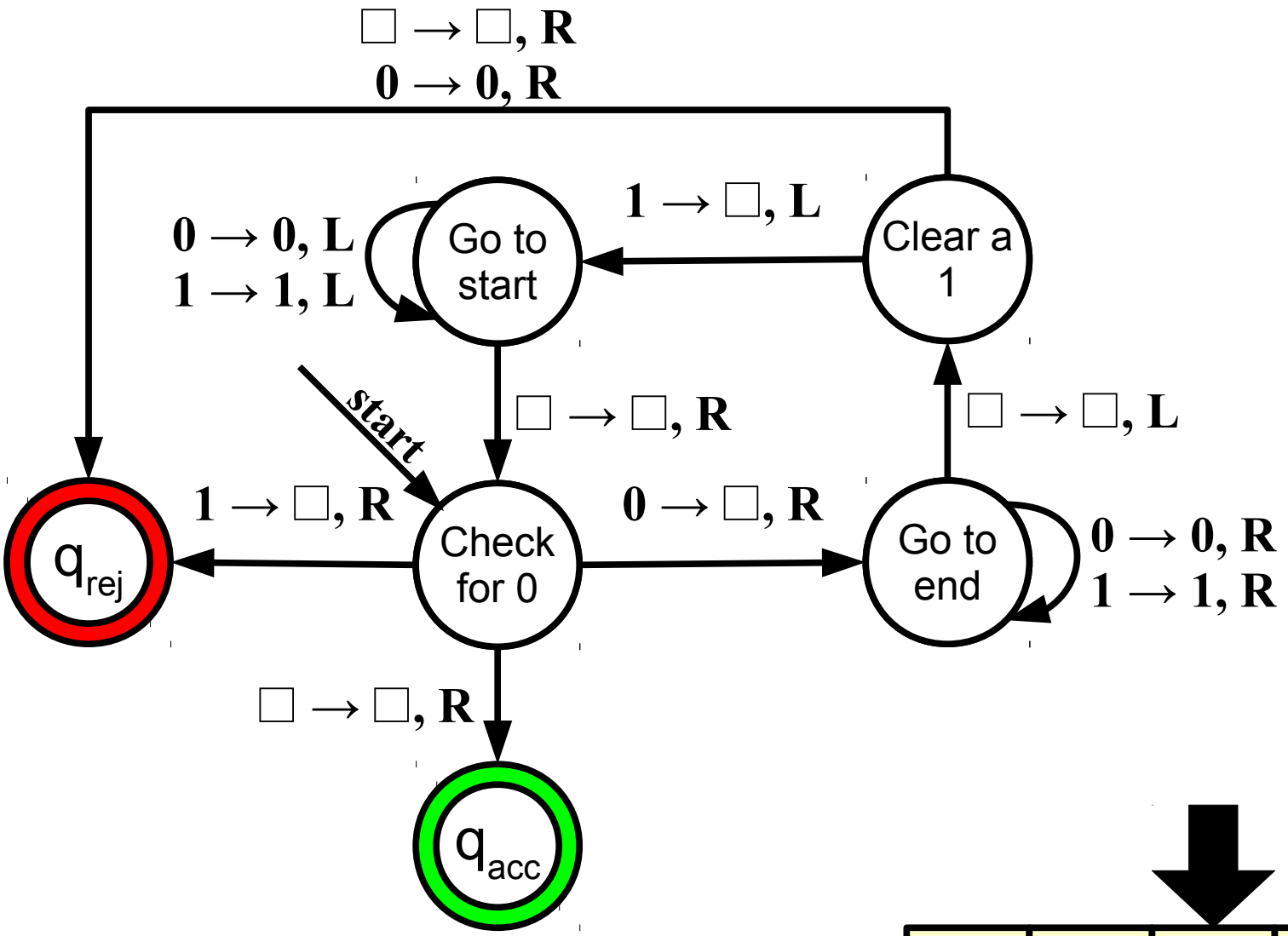
Verification

1

Try running fourteen steps of the Hailstone sequence.

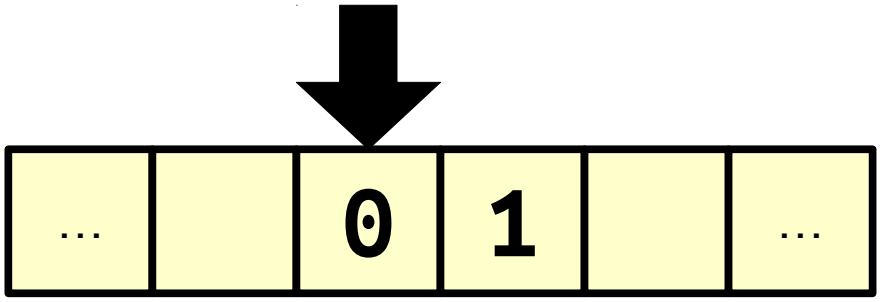
Does the hailstone sequence
terminate for this number?

Verification

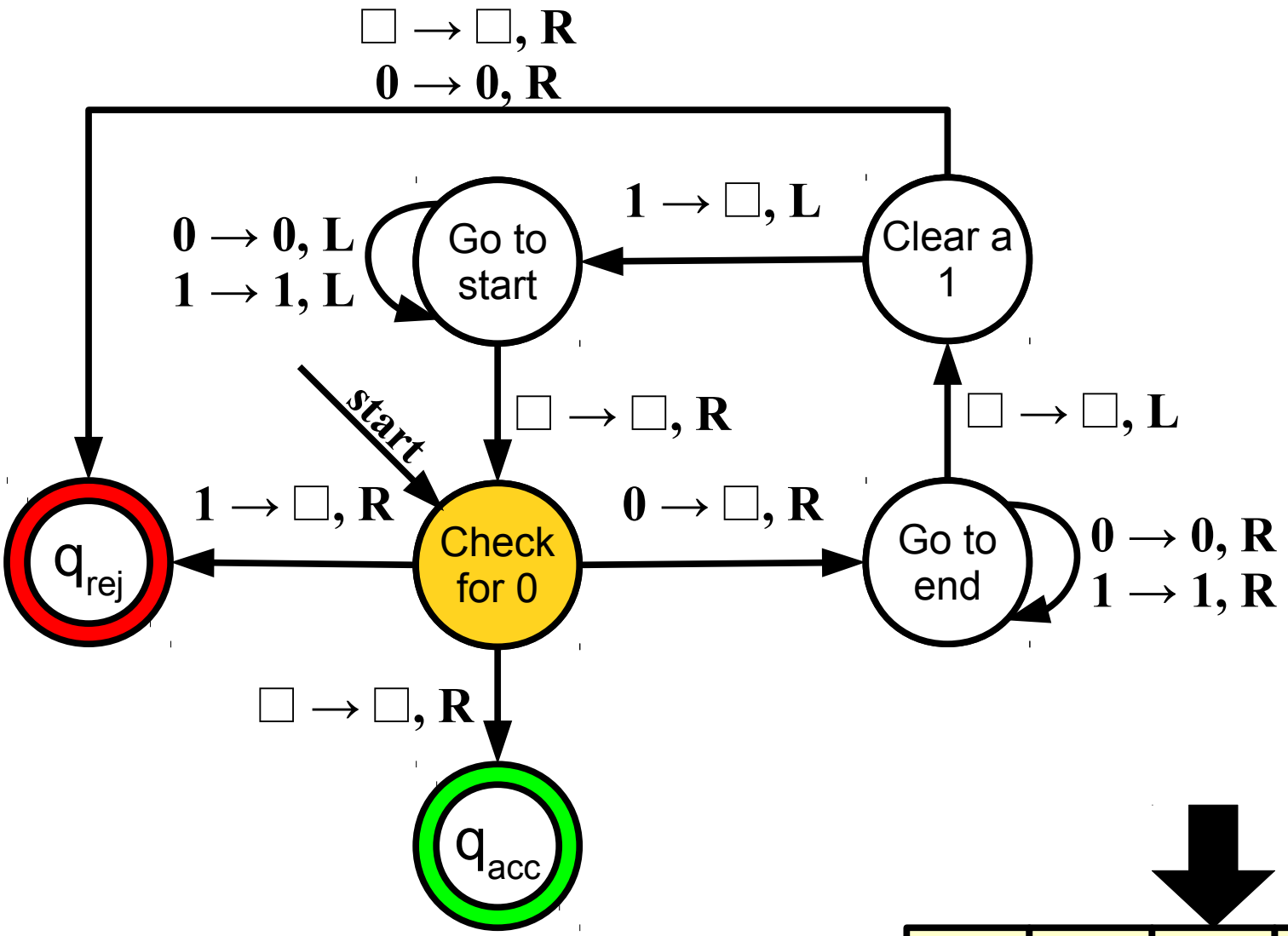


Try running it for six steps.

Does this TM halt on this input?

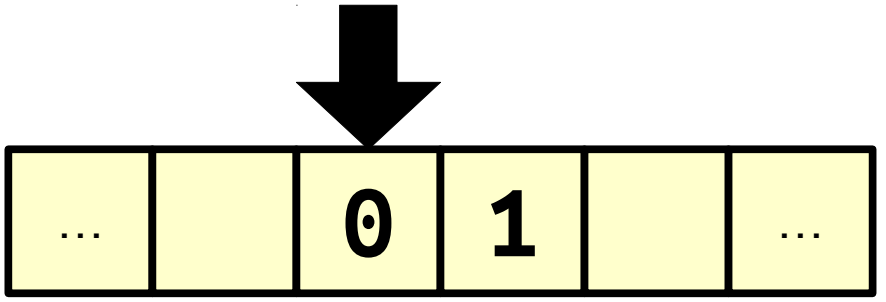


Verification

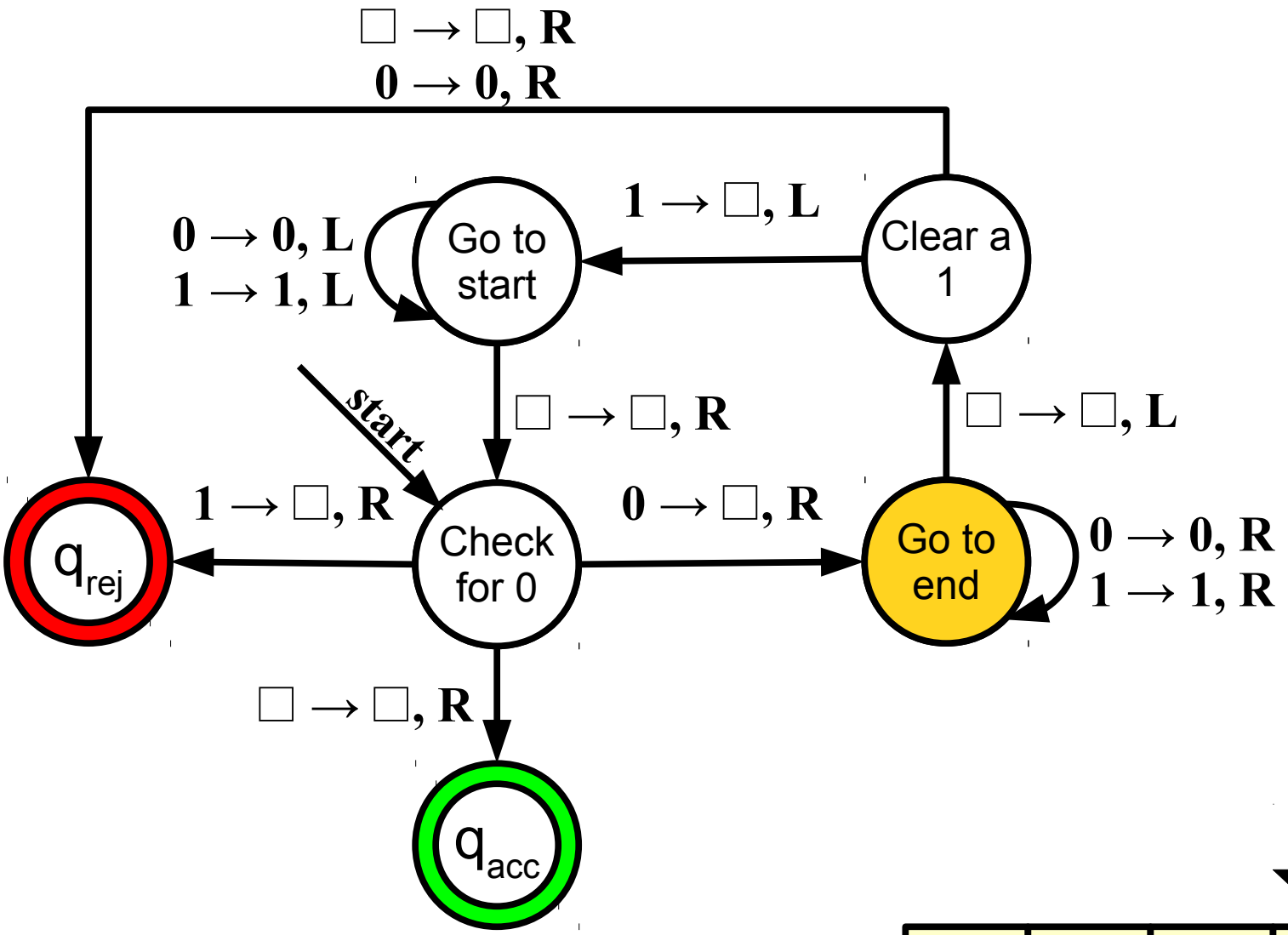


Try running it for six steps.

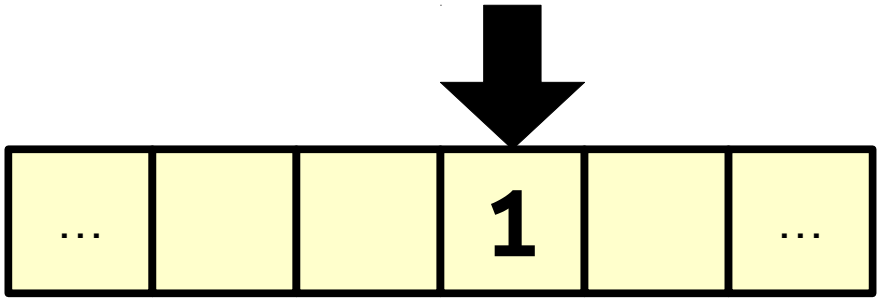
Does this TM halt on this input?



Verification

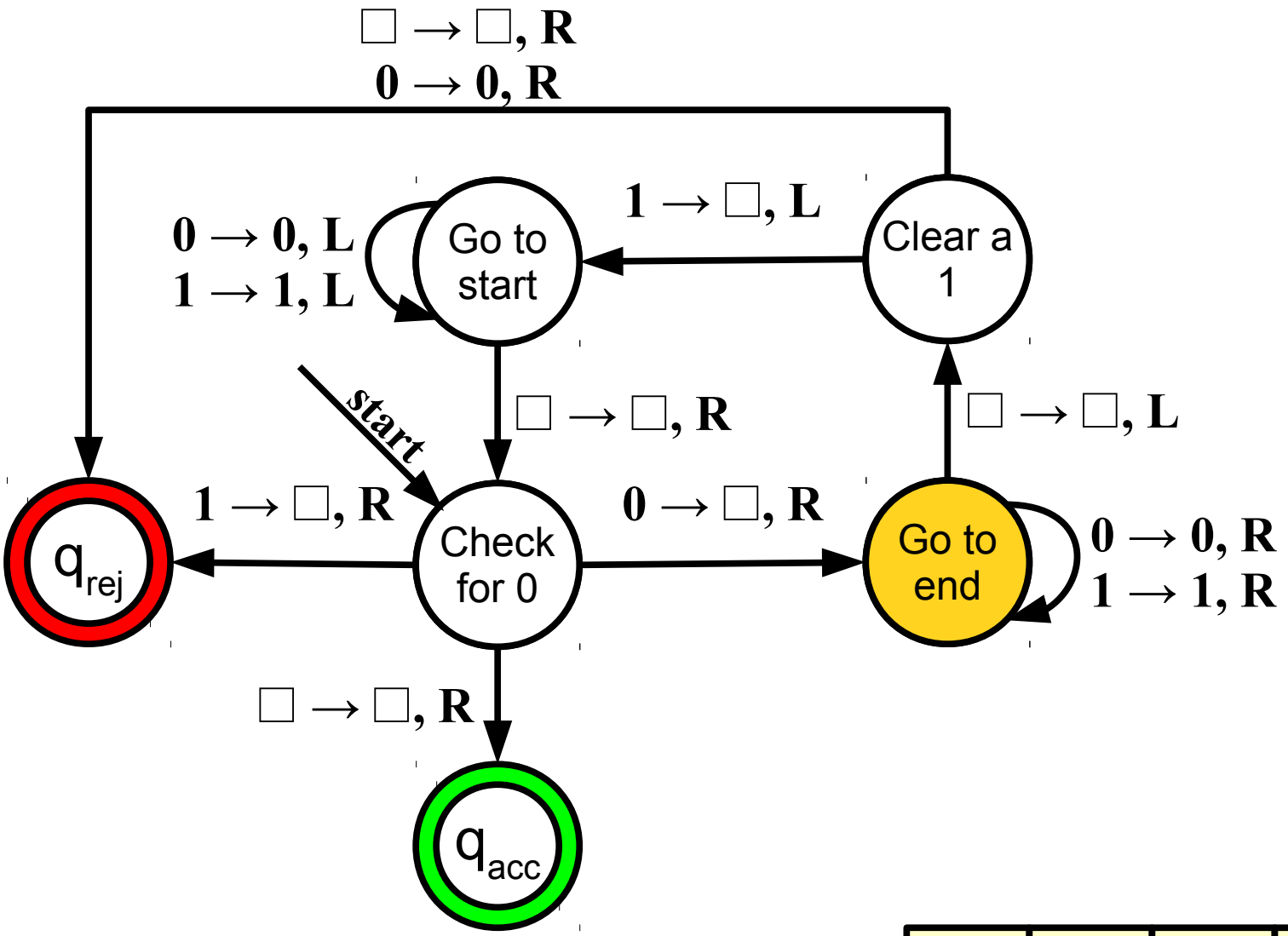


Try running it for six steps.

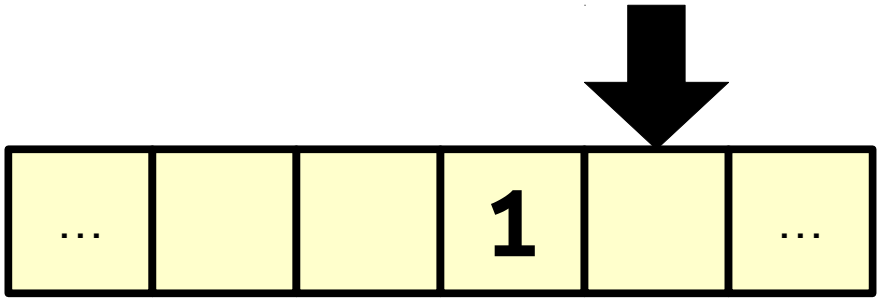


Does this TM halt on this input?

Verification

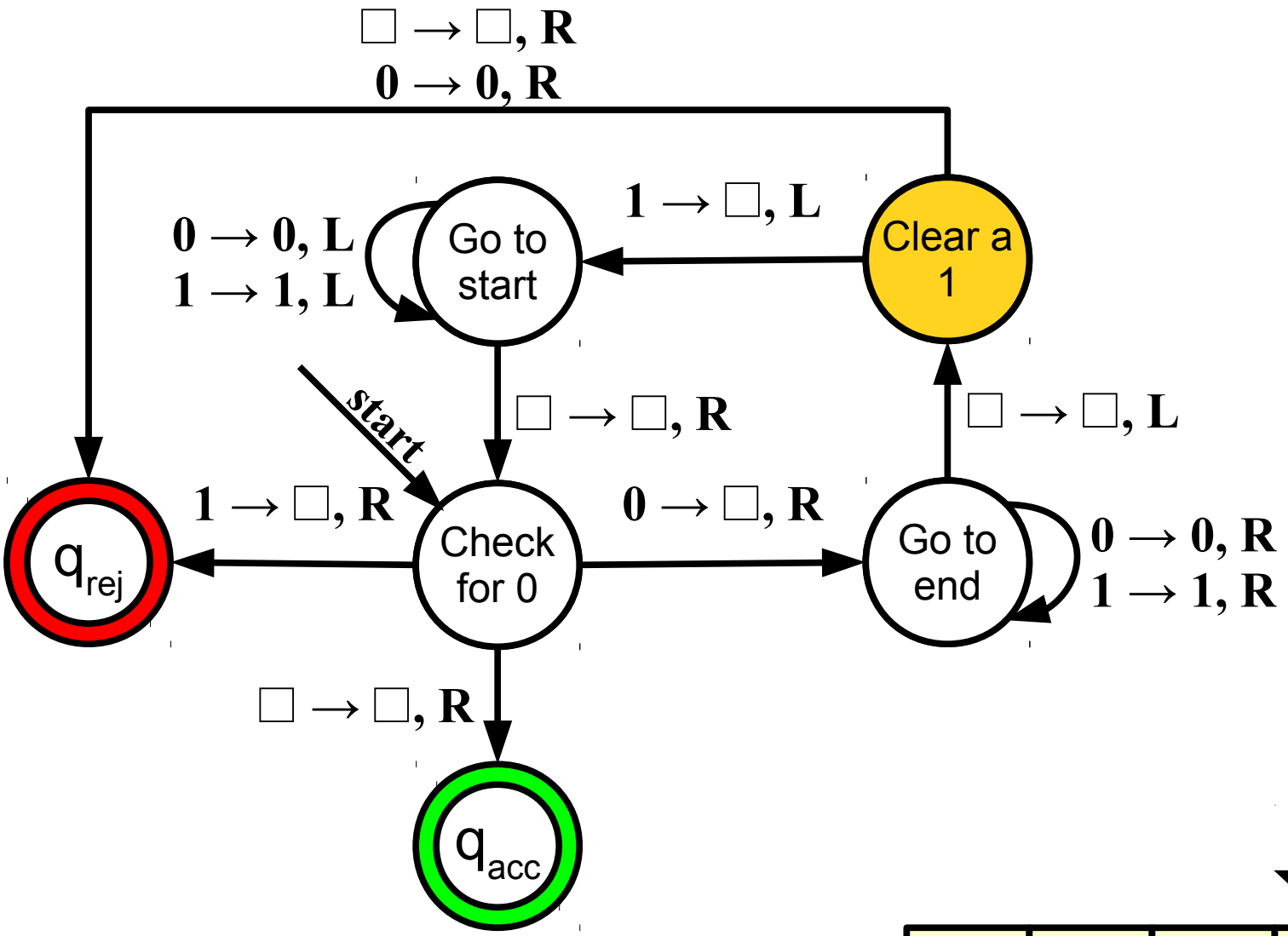


Try running it for six steps.

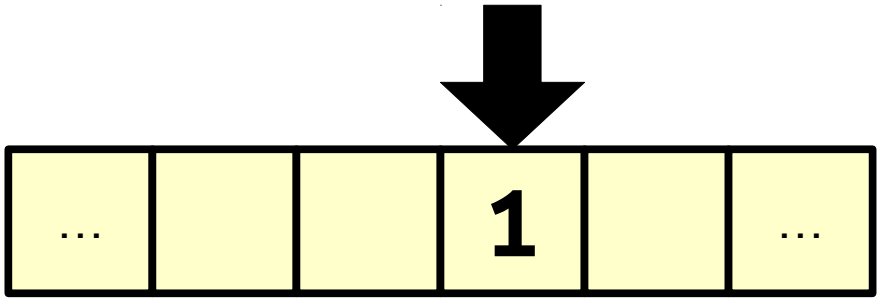


Does this TM halt on this input?

Verification

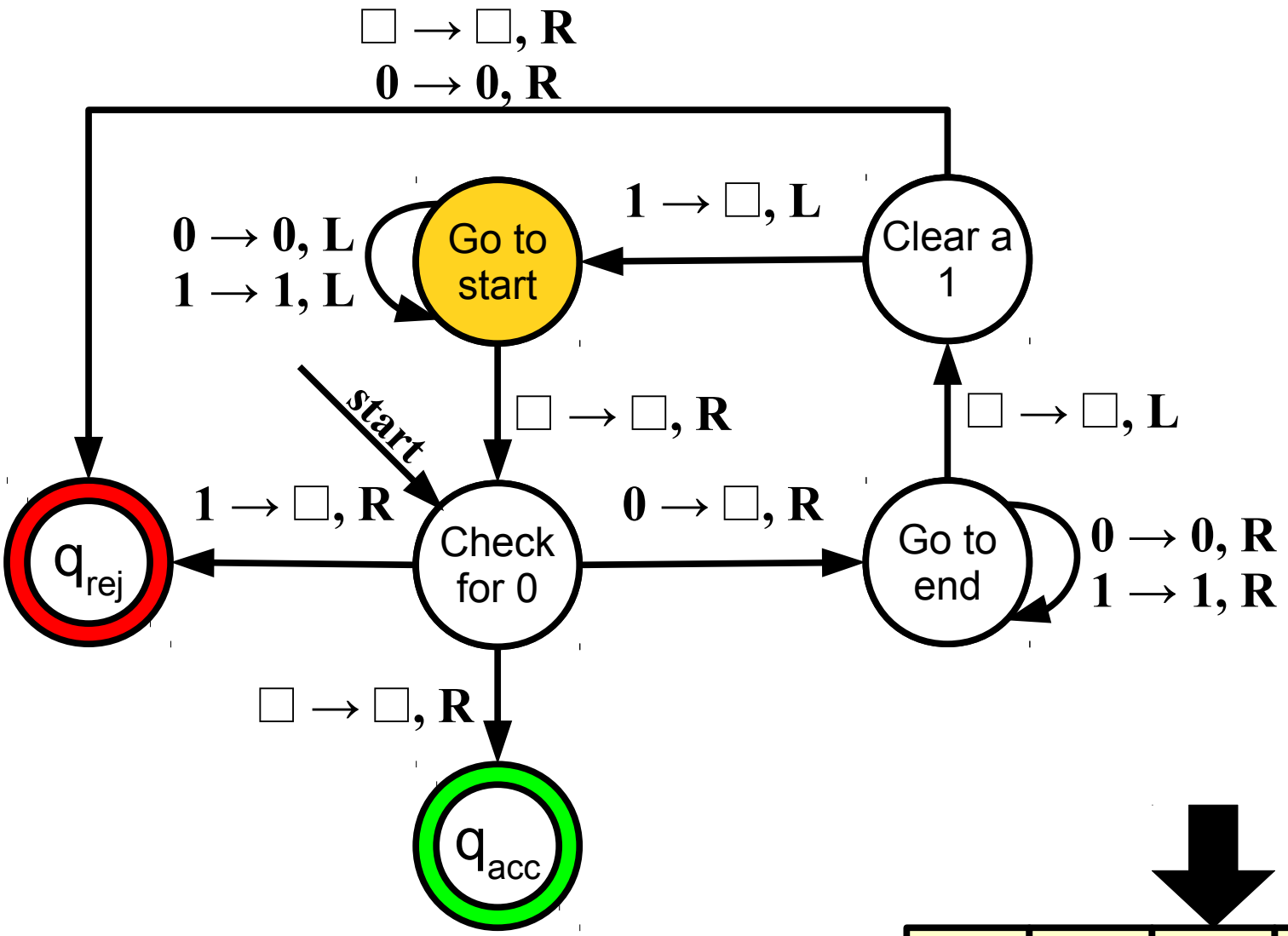


Try running it for six steps.

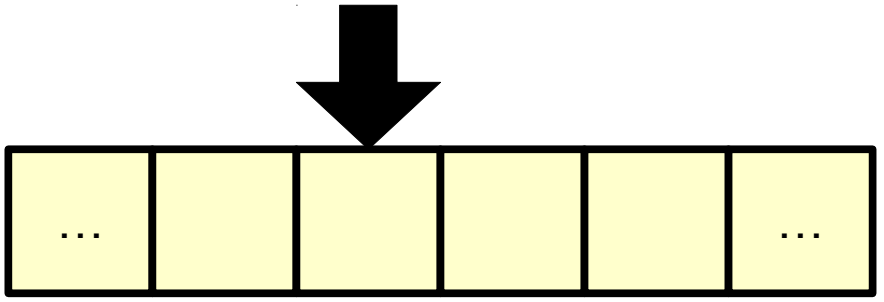


Does this TM halt on this input?

Verification

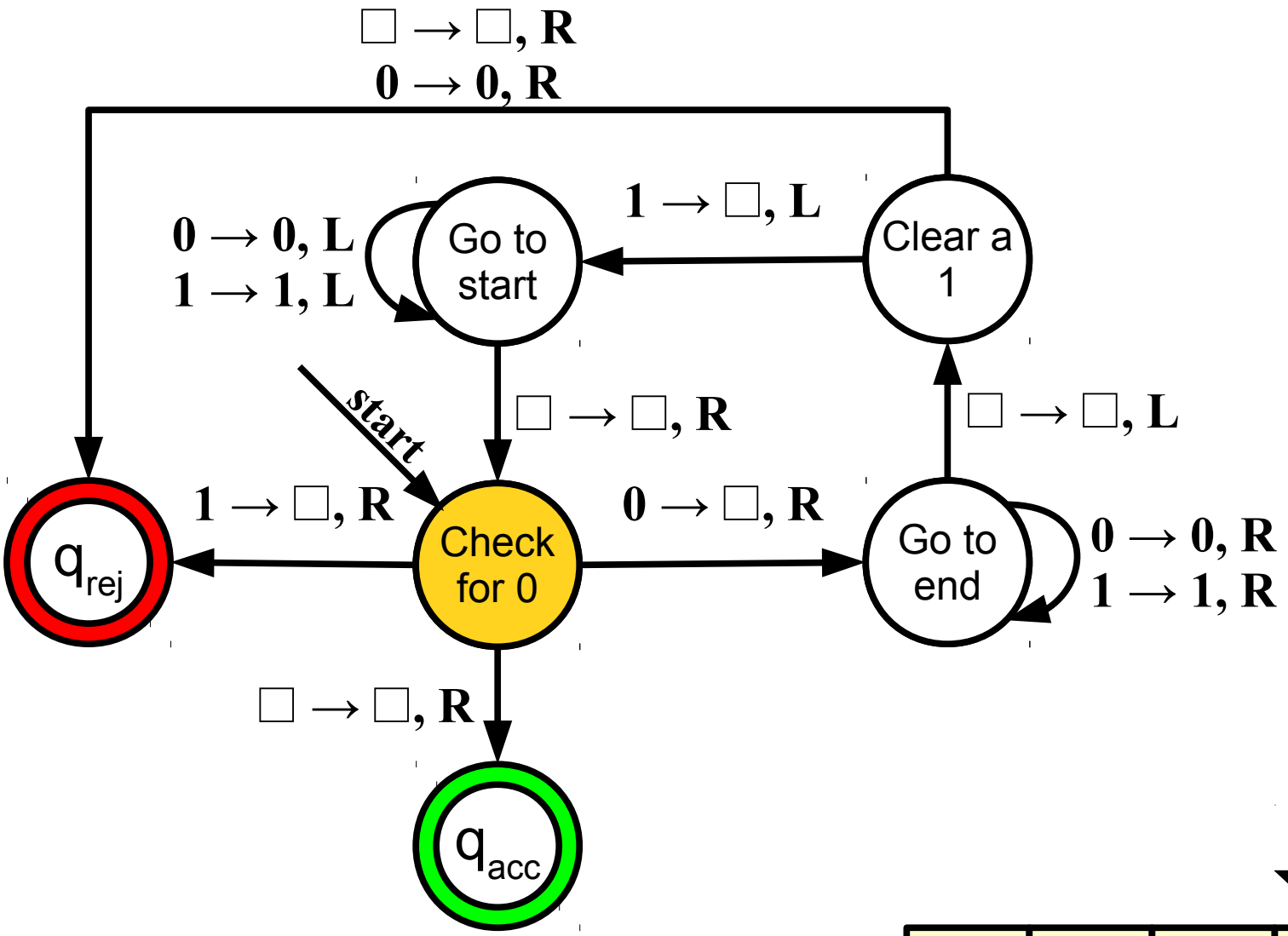


Try running it for six steps.

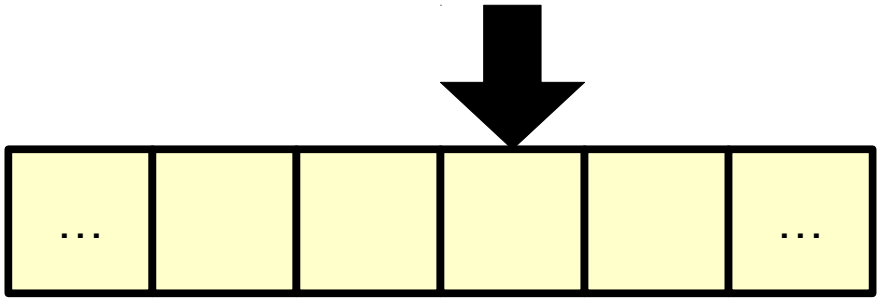


Does this TM halt on this input?

Verification

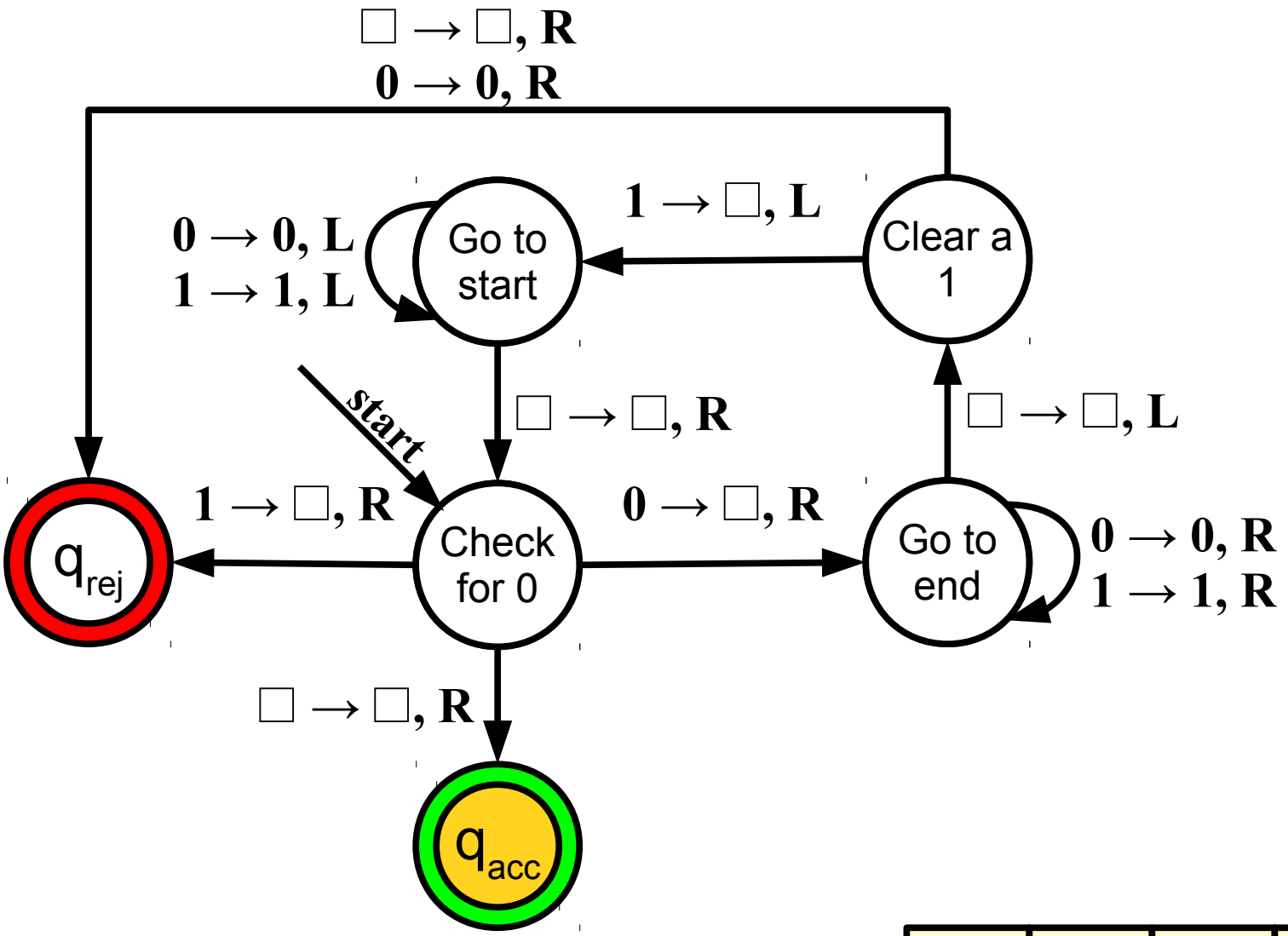


Try running it for six steps.



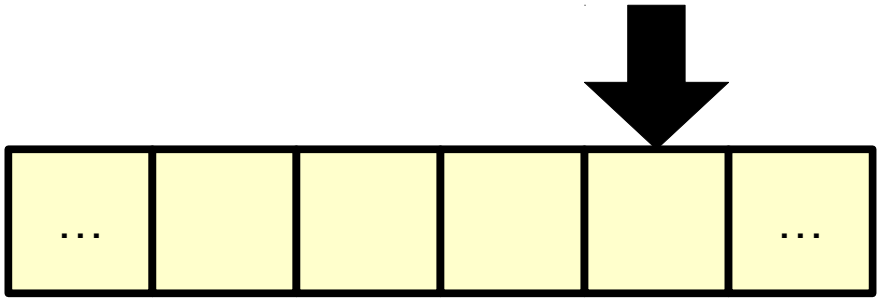
Does this TM halt on this input?

Verification



Try running it for six steps.

Does this TM halt on this input?



Verification

		7		6		1		
					3		5	2
3			1		5	9		7
6		5		3		8		9
	1						2	
8		2		1		5		4
1		3	2		7			8
5	7		4					
		4		8		7		

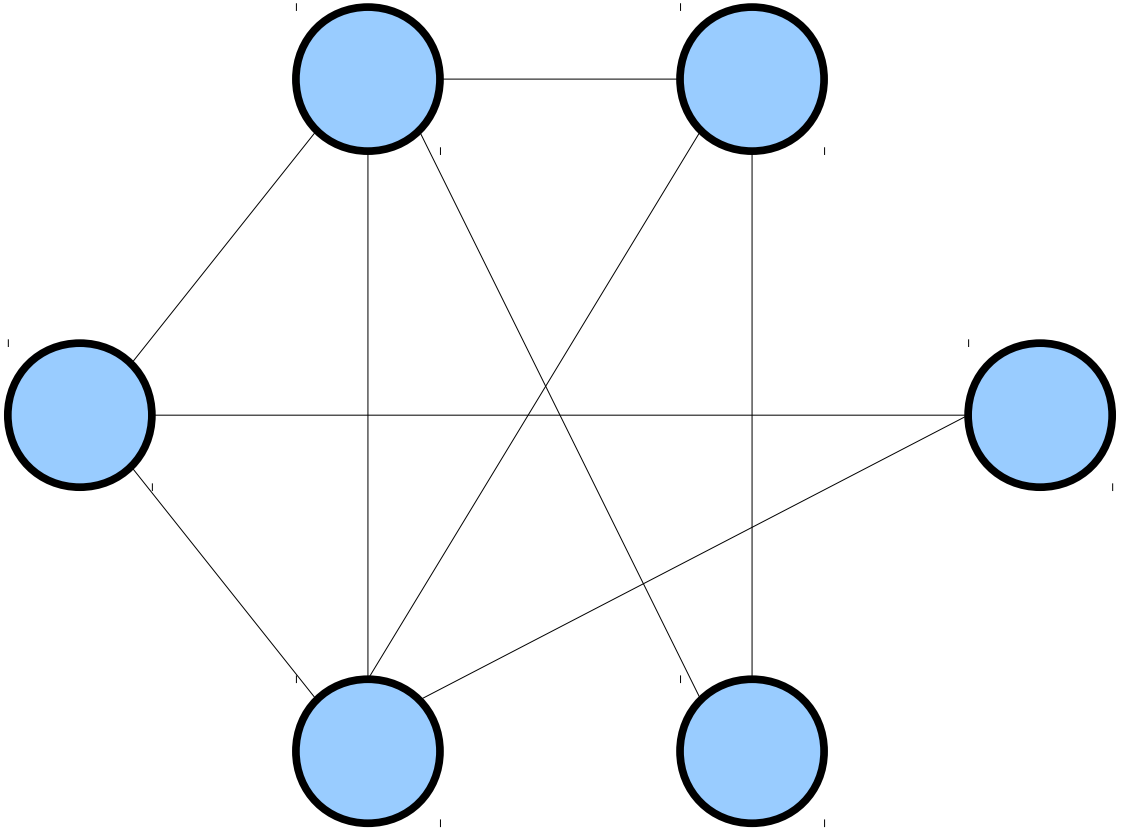
Does this Sudoku puzzle have a solution?

Verification

1	1	7	1	6	1	1	1	1
1	1	1	1	1	3	1	5	2
3	1	1	1	1	5	9	1	7
6	1	5	1	3	1	8	1	9
1	1	1	1	4	1	1	2	1
8	1	2	1	1	1	5	1	4
1	1	3	2	1	7	1	1	8
5	7	1	4	1	1	1	1	1
1	1	4	1	8	1	7	1	1

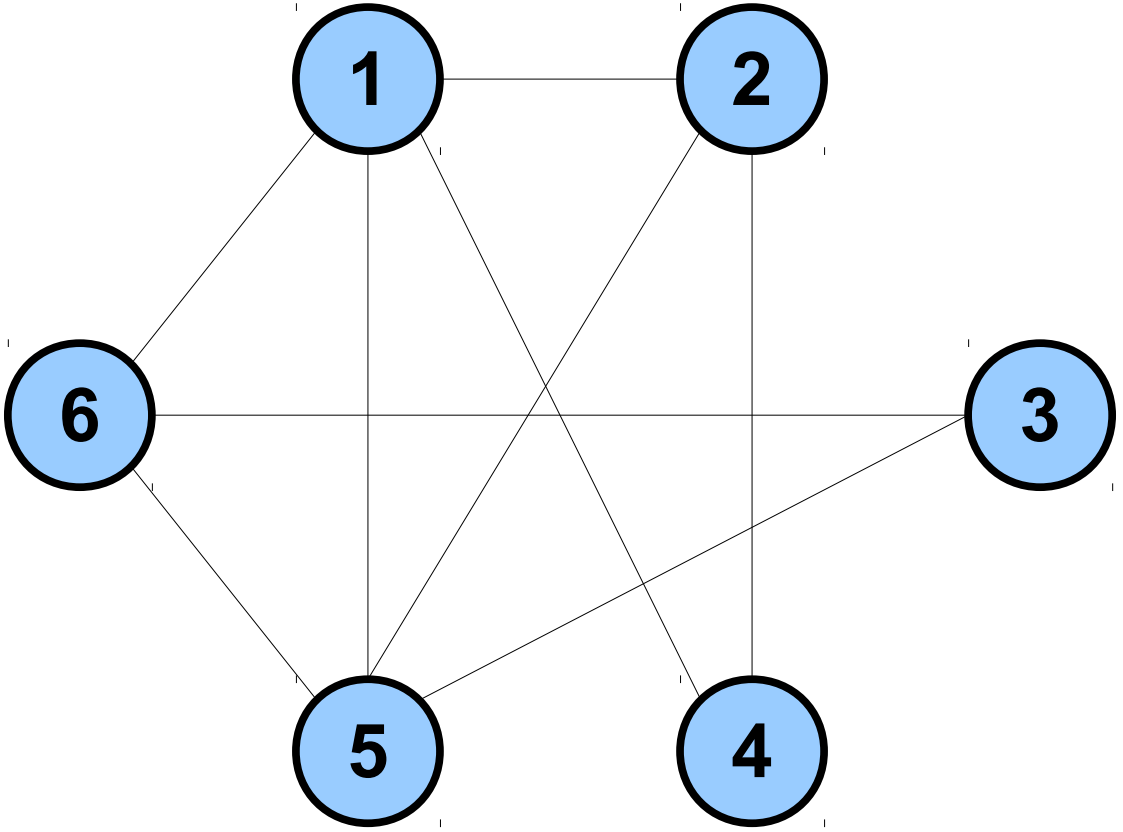
Does this Sudoku puzzle have a solution?

Verification



Is there a simple path that goes through every node exactly once?

Verification



Is there a simple path that goes through every node exactly once?

Verification

11

Does the hailstone sequence
terminate for this number?

Verification

11

Try running five steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

34

Try running five steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

17

Try running five steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

52

Try running five steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

26

Try running five steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

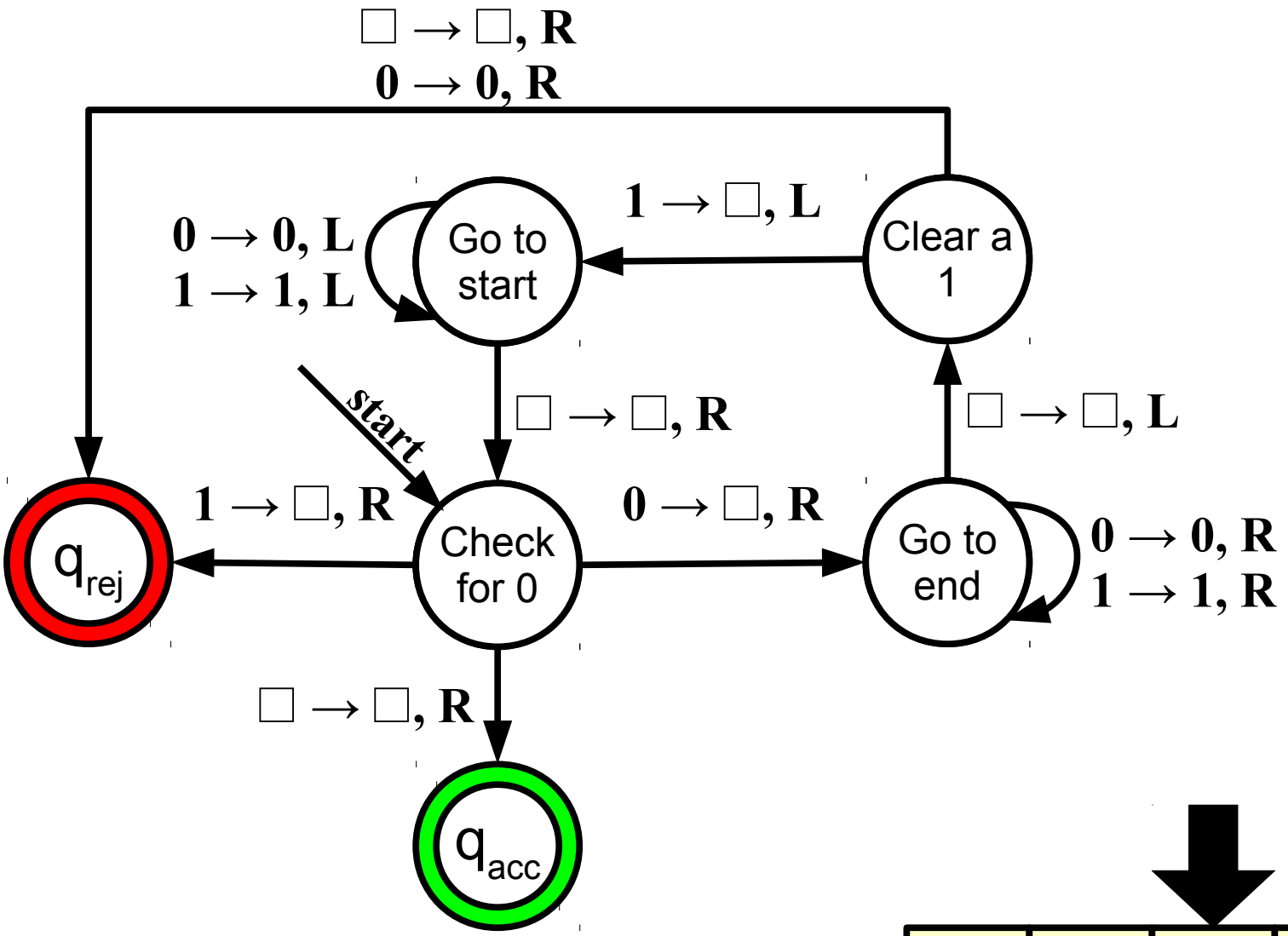
Verification

13

Try running five steps of the Hailstone sequence.

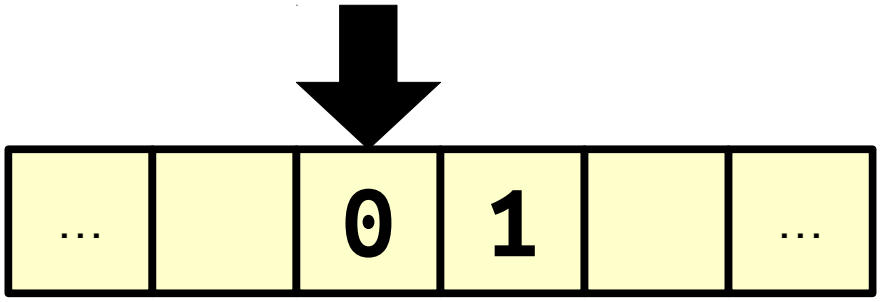
Does the hailstone sequence
terminate for this number?

Verification

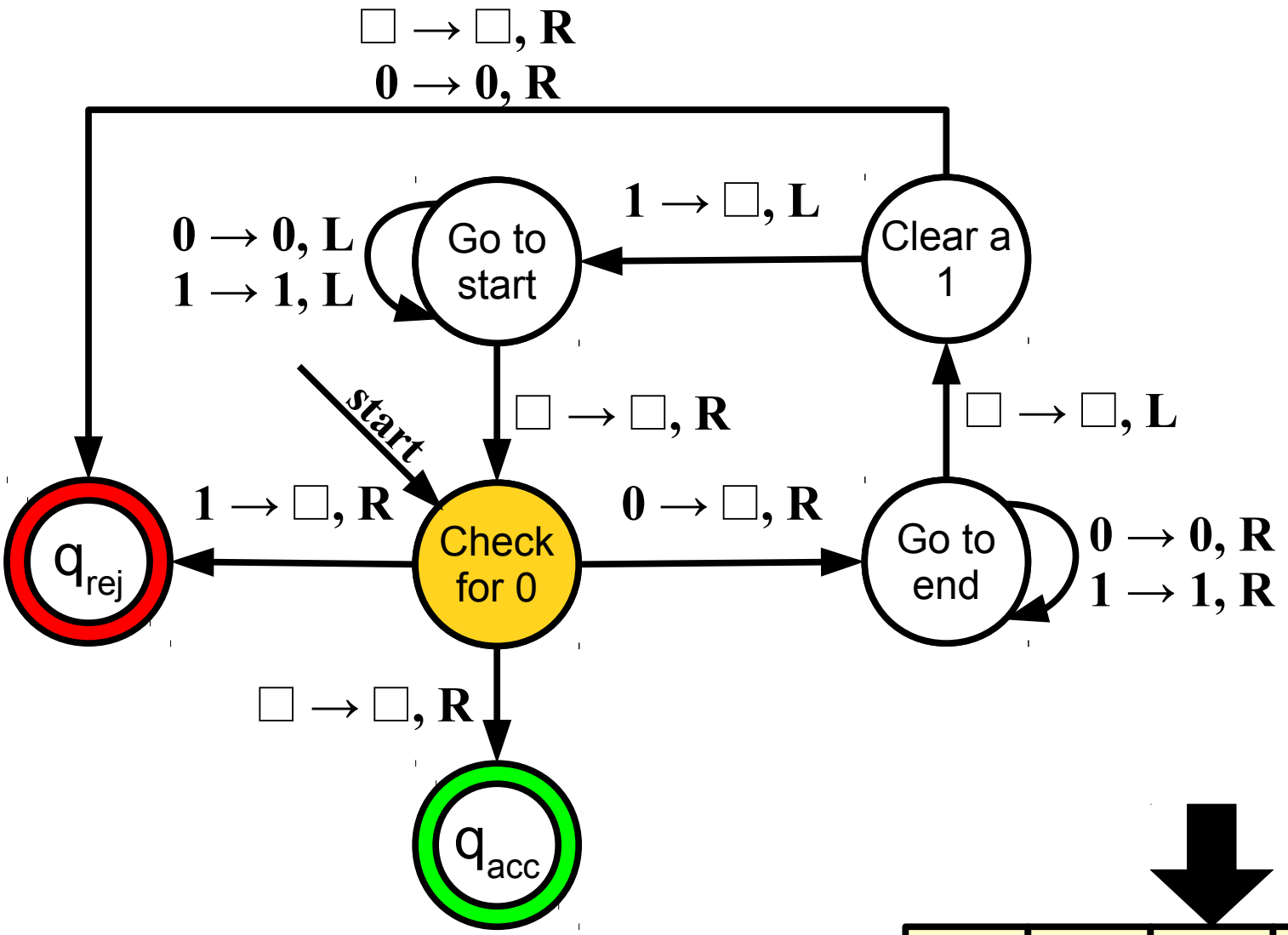


Try running it for four steps.

Does this TM halt on this input?

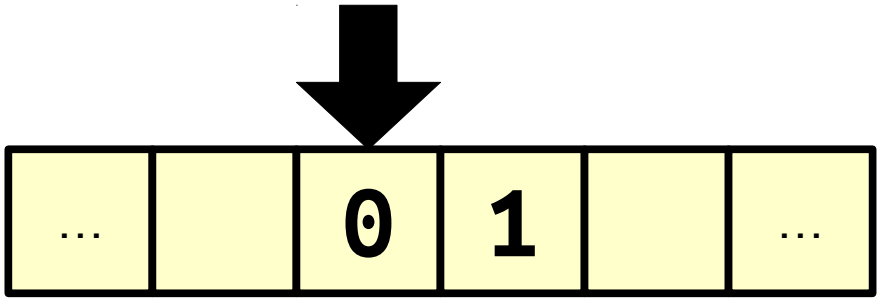


Verification

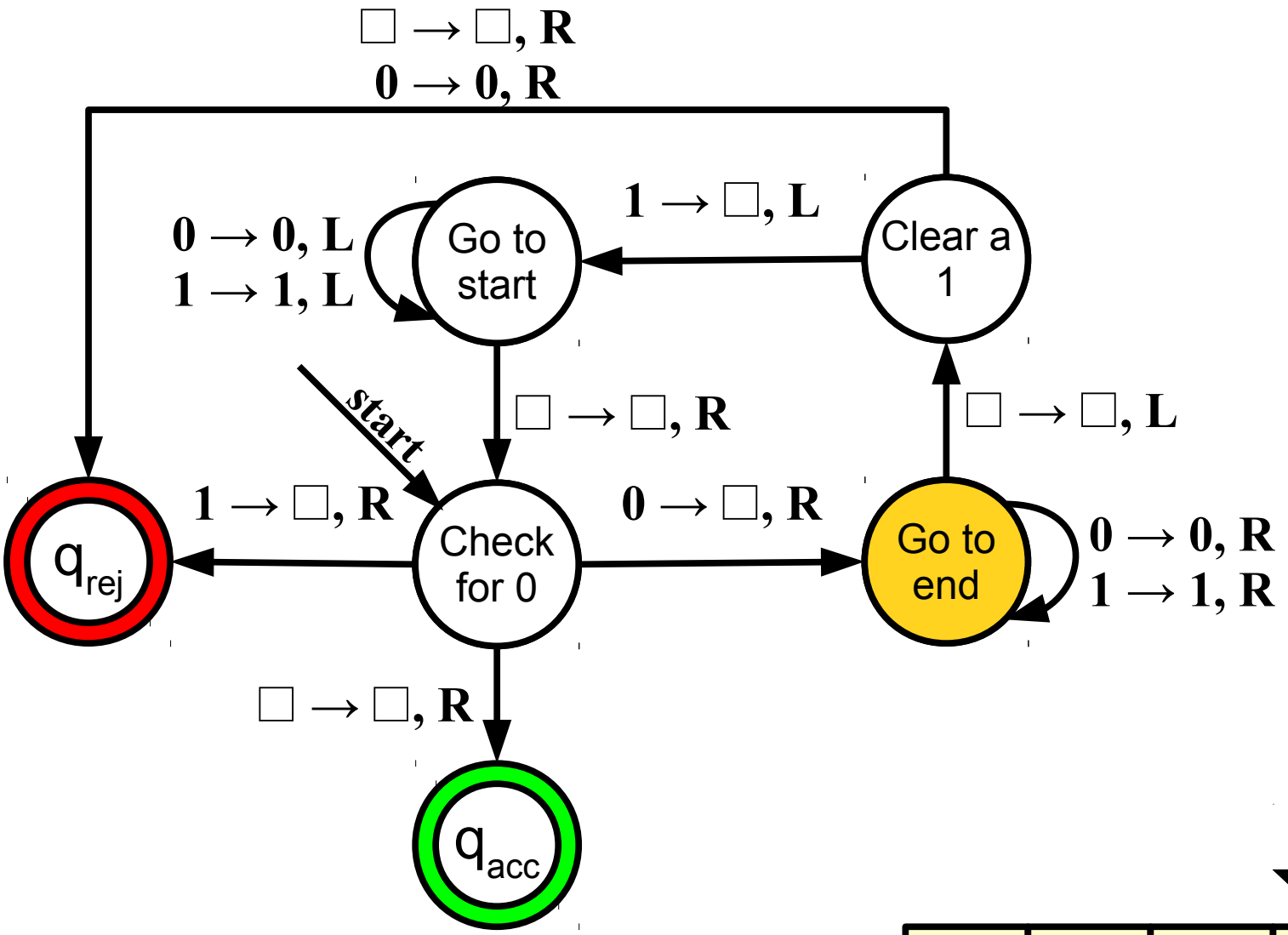


Try running it for four steps.

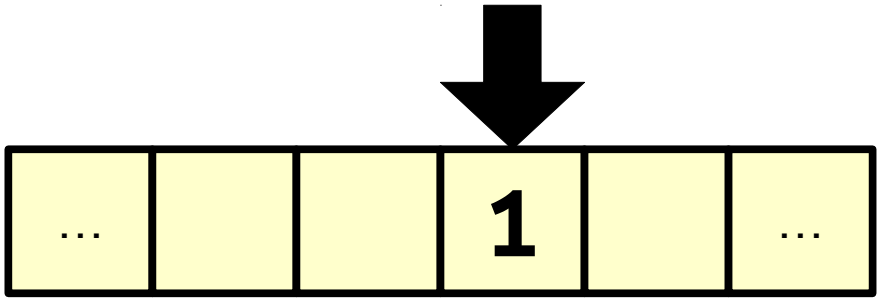
Does this TM halt on this input?



Verification

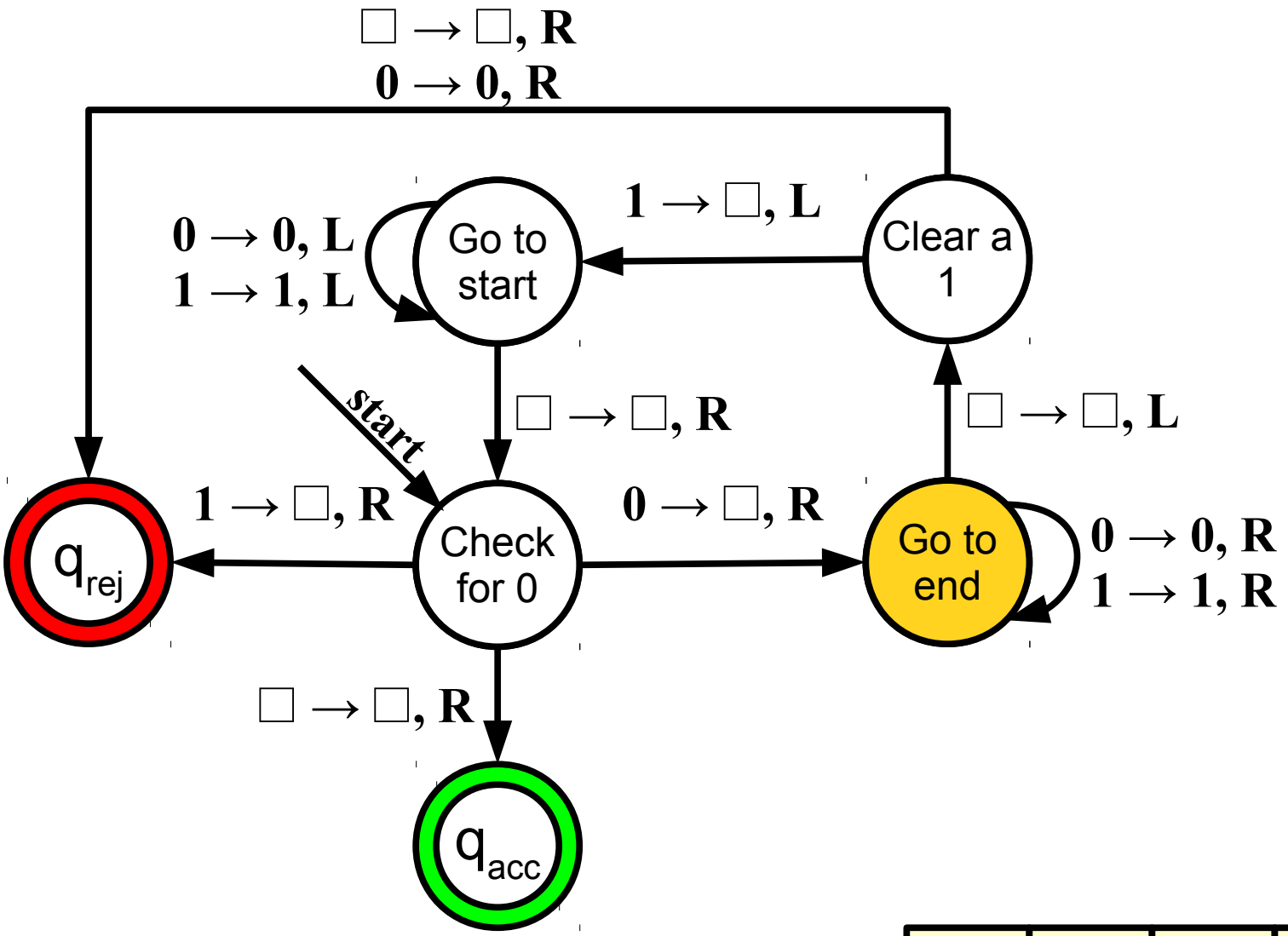


Try running it for four steps.

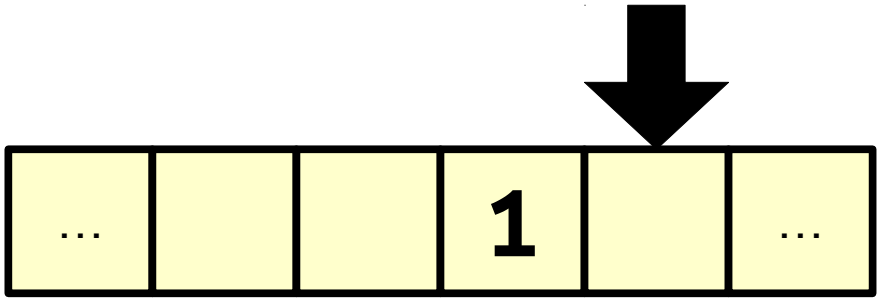


Does this TM halt on this input?

Verification

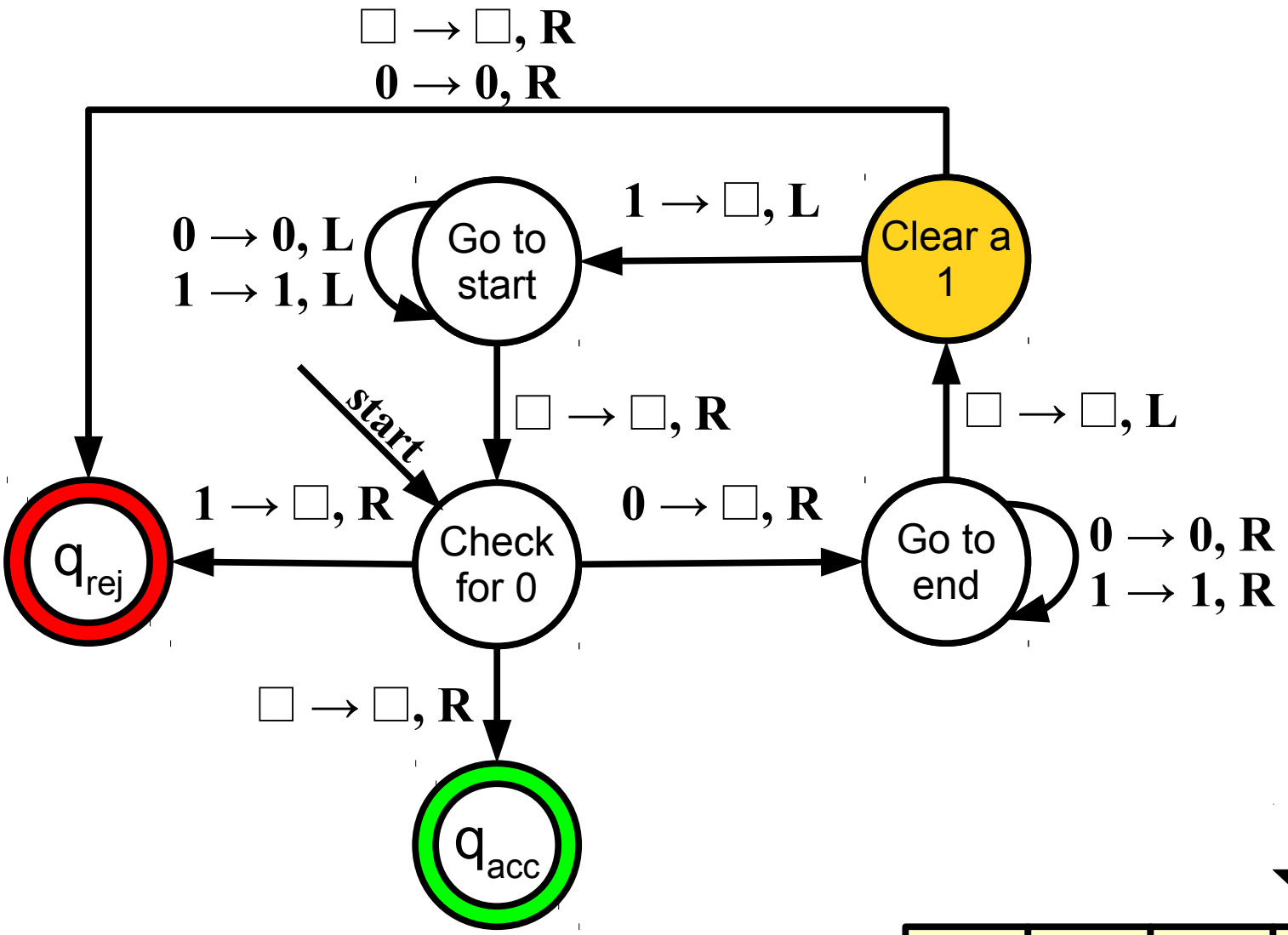


Try running it for four steps.

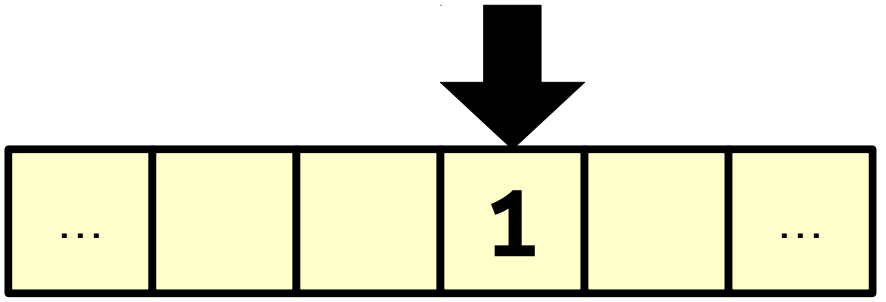


Does this TM halt on this input?

Verification

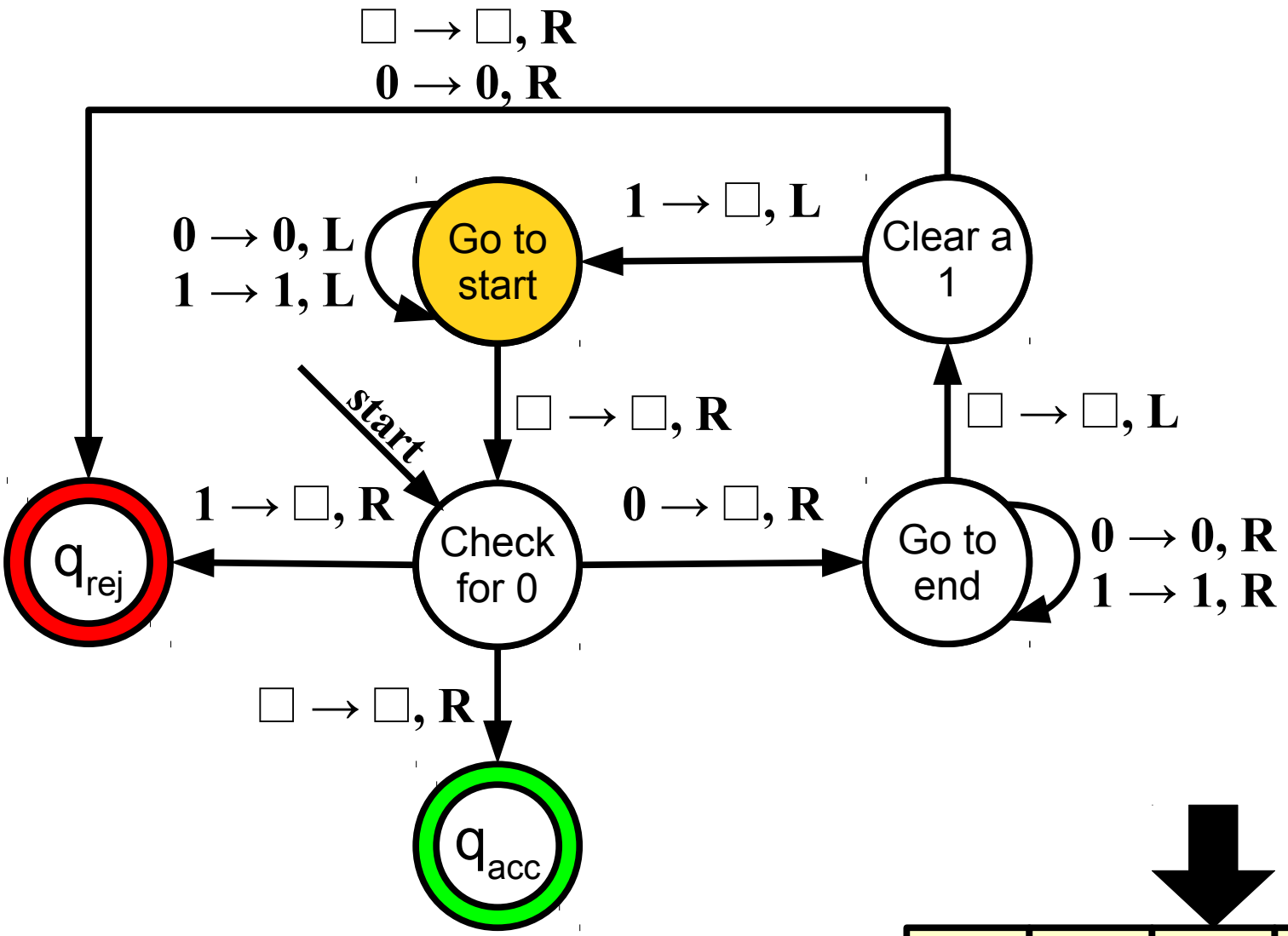


Try running it for four steps.

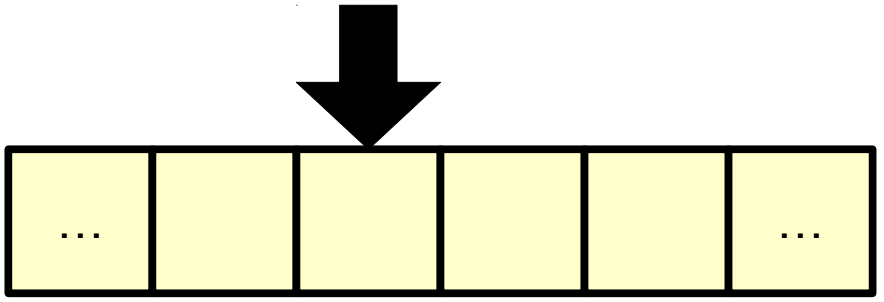


Does this TM halt on this input?

Verification

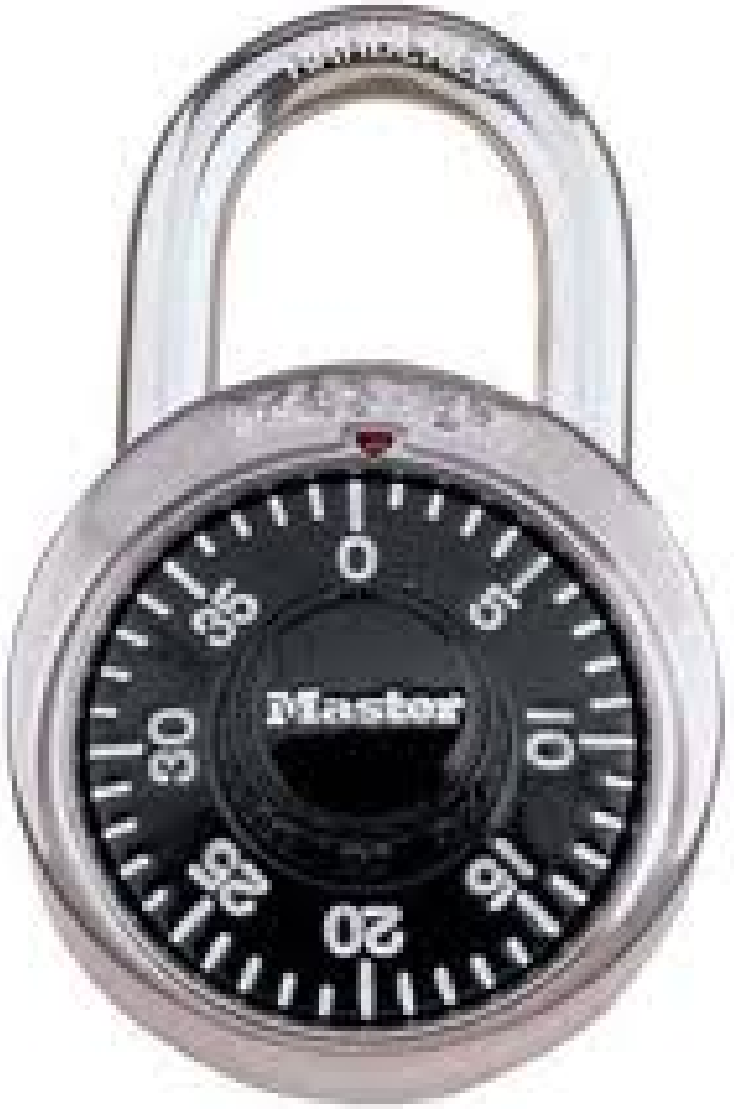


Try running it for four steps.



Does this TM halt on this input?

Verification



Question:
Can this lock be opened?

Verification

- In each of the preceding cases, we were given some problem and some evidence supporting the claim that the answer is “yes.”
- Given the correct evidence, we can be certain that the answer is indeed “yes.”
- Given incorrect evidence, we aren't sure whether the answer is “yes.”
  - Maybe there's *no* evidence saying that the answer is “yes,” or maybe there is some evidence, but just not the evidence we were given.
- Let's formalize this idea.

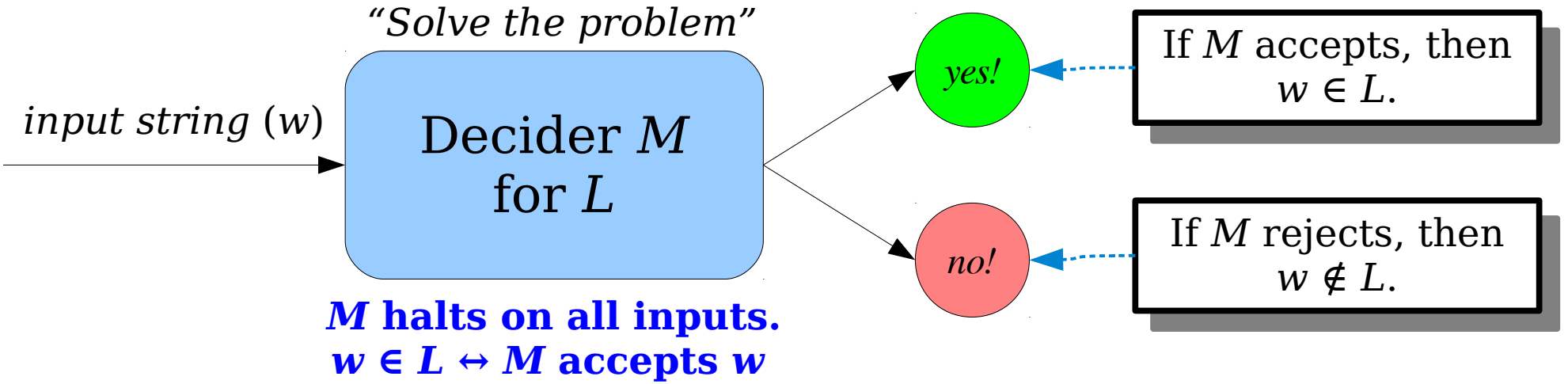
Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

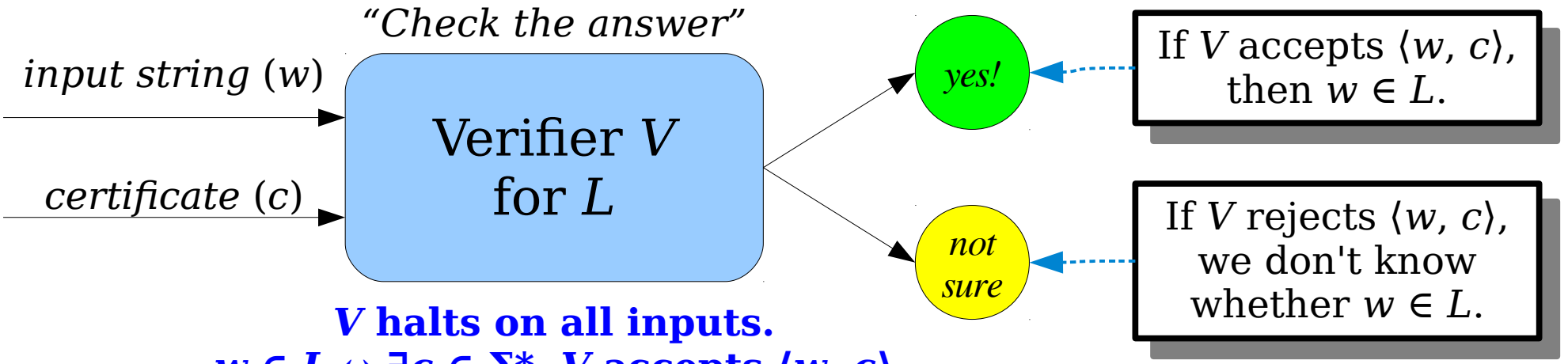
$$w \in L \leftrightarrow \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle$$

- A string c where V accepts $\langle w, c \rangle$ is called a ***certificate*** for w.
- Intuitively, what does this mean?

Deciders and Verifiers



***M* halts on all inputs.**
w* ∈ *L* ↔ *M* accepts *w



***V* halts on all inputs.**
***w* ∈ *L* ↔ ∃*c* ∈ Σ*. *V* accepts ⟨*w*, *c*⟩**

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Some notes about V:
  - If V accepts $\langle w, c \rangle$, then we're guaranteed $w \in L$.
  - If V does not accept $\langle w, c \rangle$, then either
    - $w \in L$, but you gave the wrong c, or
    - $w \notin L$, so no possible c will work.

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Some notes about V:
  - Notice that c is existentially quantified. Any string $w \in L$ must have at least one c that causes V to accept, and possibly more.
  - V is required to halt, so given any potential certificate c for w, you can check whether the certificate is correct.

Verifiers

- A ***verifier*** for a language L is a TM V with the following properties:
  - V halts on all inputs.
  - For any string $w \in \Sigma^*$, the following is true:

$$\boldsymbol{w \in L \quad \leftrightarrow \quad \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

- Some notes about V:
  - Notice that $\mathscr{L}(V) \neq L$. *(Good question: what <u>is</u> $\mathscr{L}(V)$?)*
  - The job of V is just to check certificates, not to decide membership in L.

Some Verifiers

- Let L be the following language:

$$L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$$

- Let's see how to build a verifier for L.

Verification

11

Does the hailstone sequence
terminate for this number?

Verification

11

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

34

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

17

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

52

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

26

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

13

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

40

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

20

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

10

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

5

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

16

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

8

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

4

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

2

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Verification

1

Try running fourteen steps of the Hailstone sequence.

Does the hailstone sequence
terminate for this number?

Some Verifiers

- Let L be the following language:

$$L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n \}$$

```
private boolean checkHailstone(int n, int k) {
   for (int i = 0; i < k; i++) {
       if (n % 2 == 0) n /= 2;
       else n = 3*n + 1;
   }
   return n == 1;
}
```

- Do you see why $\langle n \rangle \in L$ iff there is some k such that `checkHailstone(n, k)` returns true?
- Do you see why `checkHailstone` always halts?

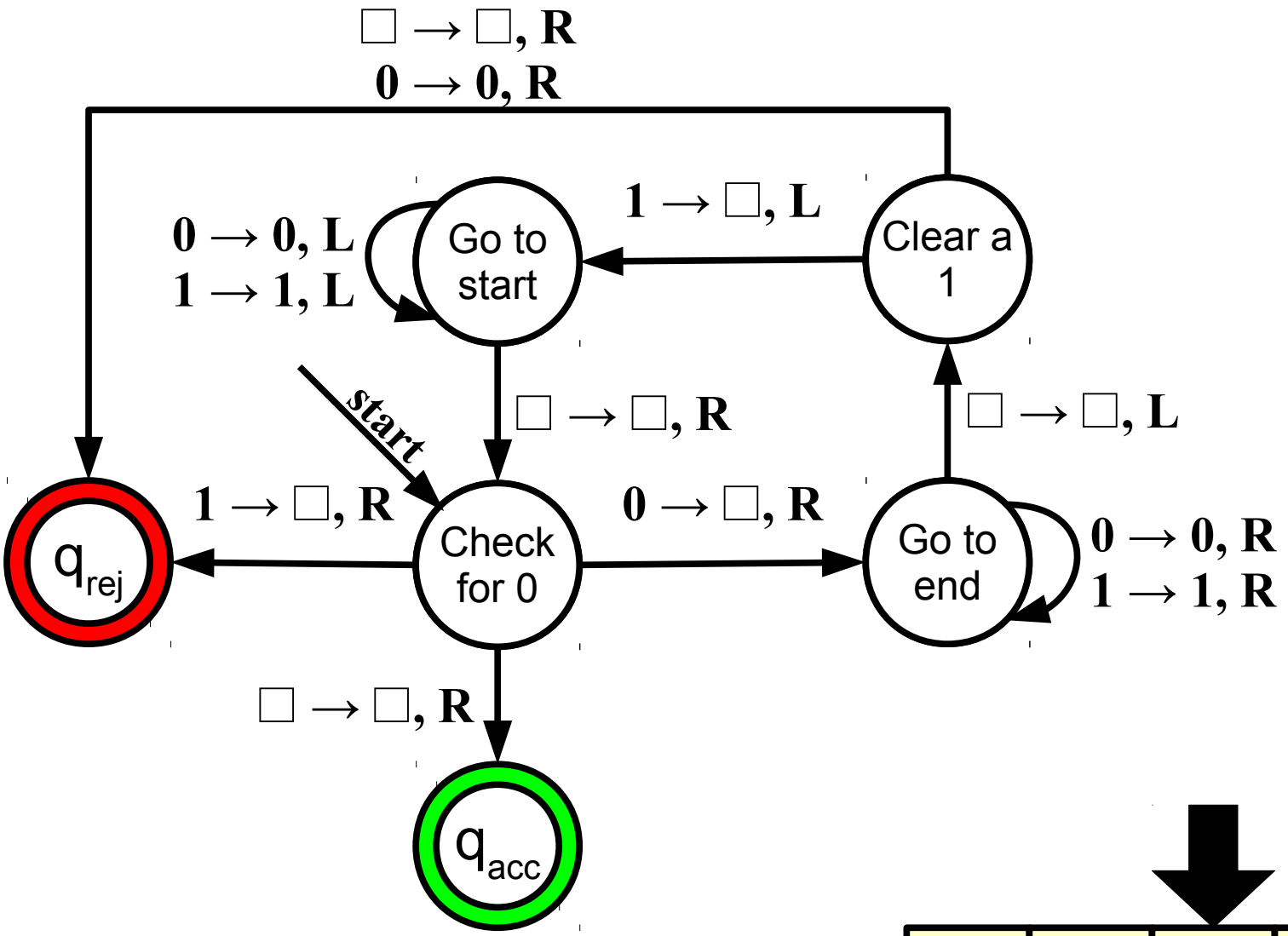
Some Verifiers

- Consider *HALT*:

  $HALT = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on } w \}$

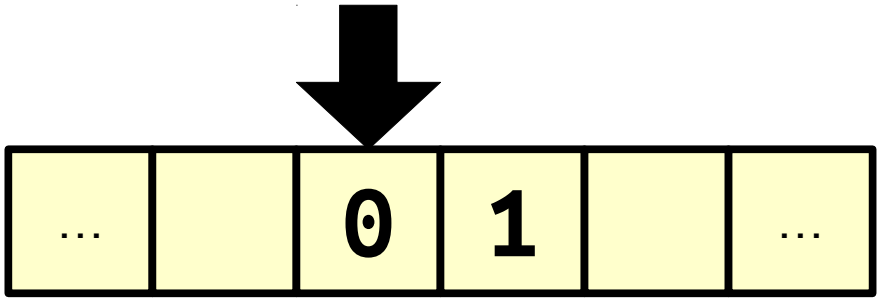
- Let's see how to build a verifier for *HALT*.

Verification

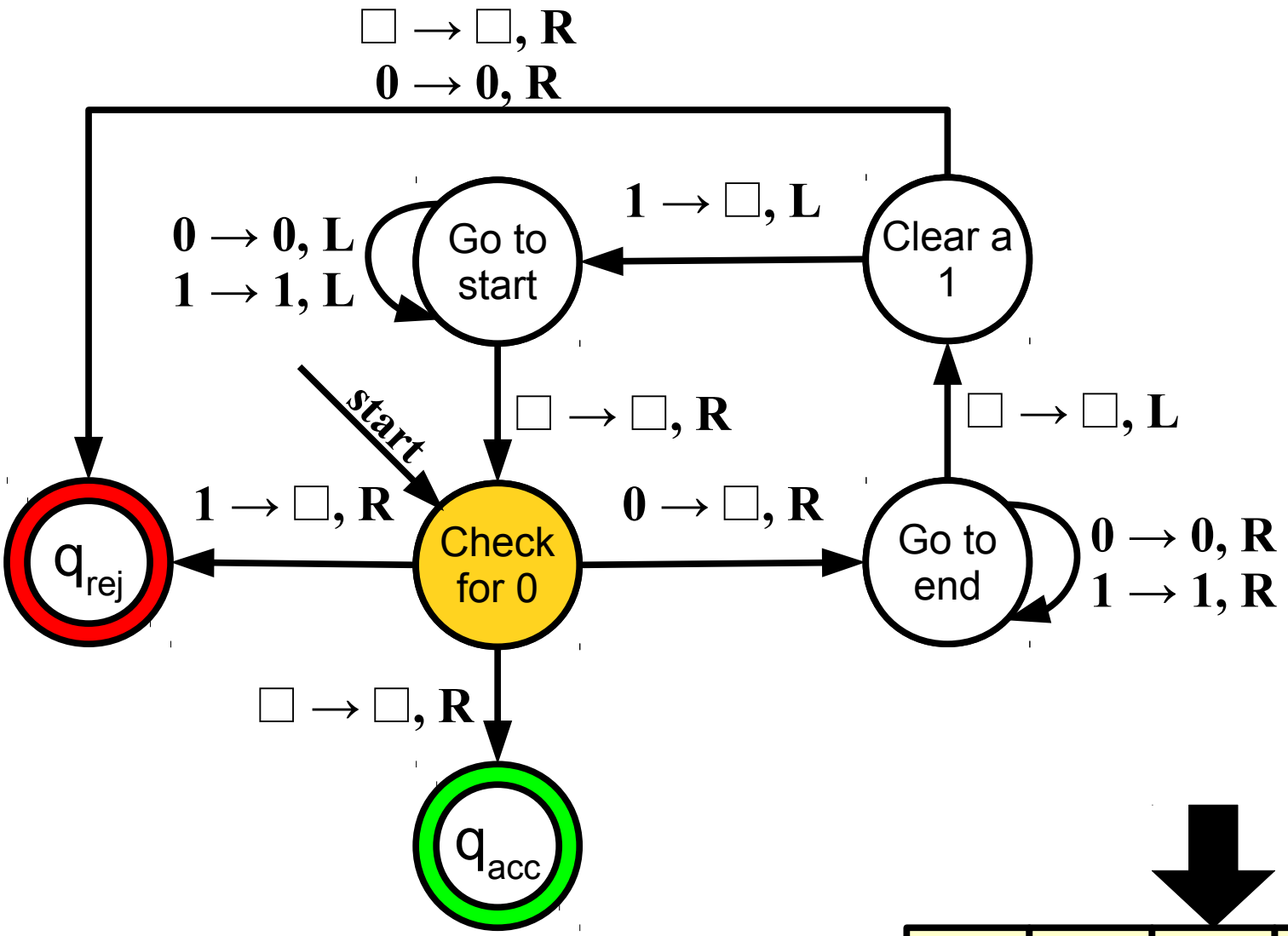


Try running it for six steps.

Does this TM halt on this input?

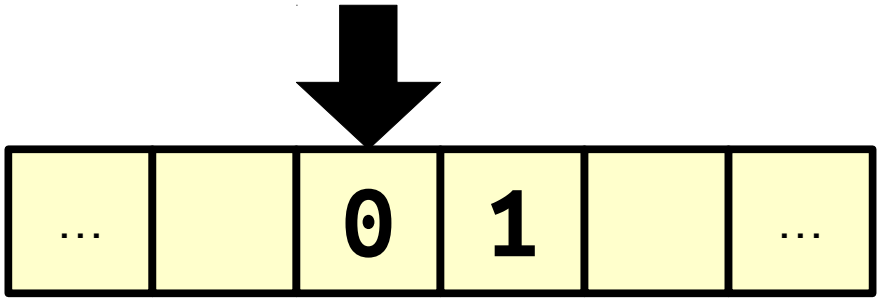


Verification

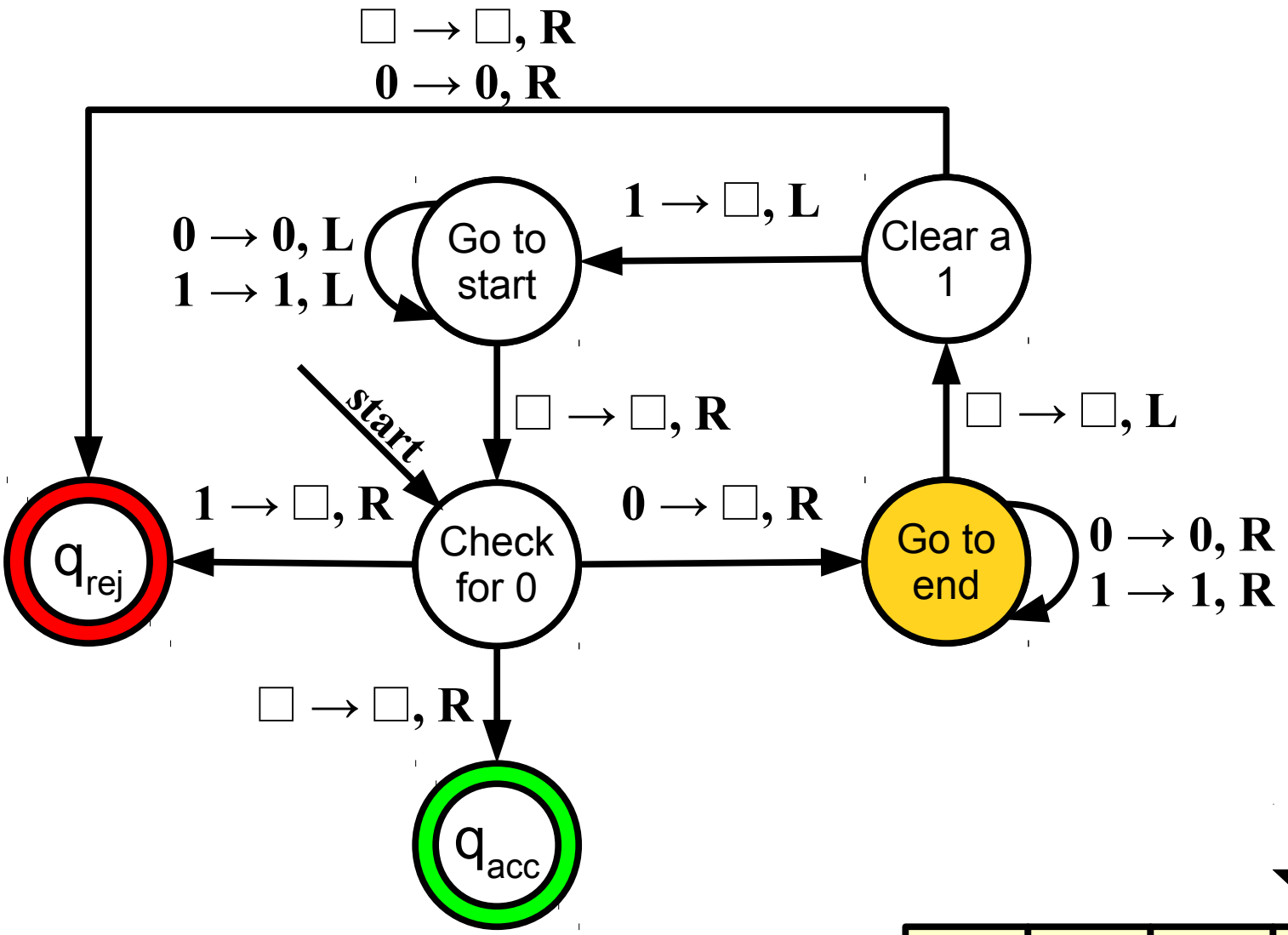


Try running it for six steps.

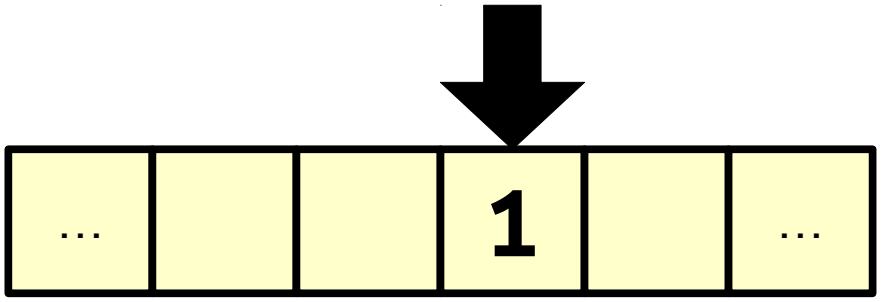
Does this TM halt on this input?



Verification

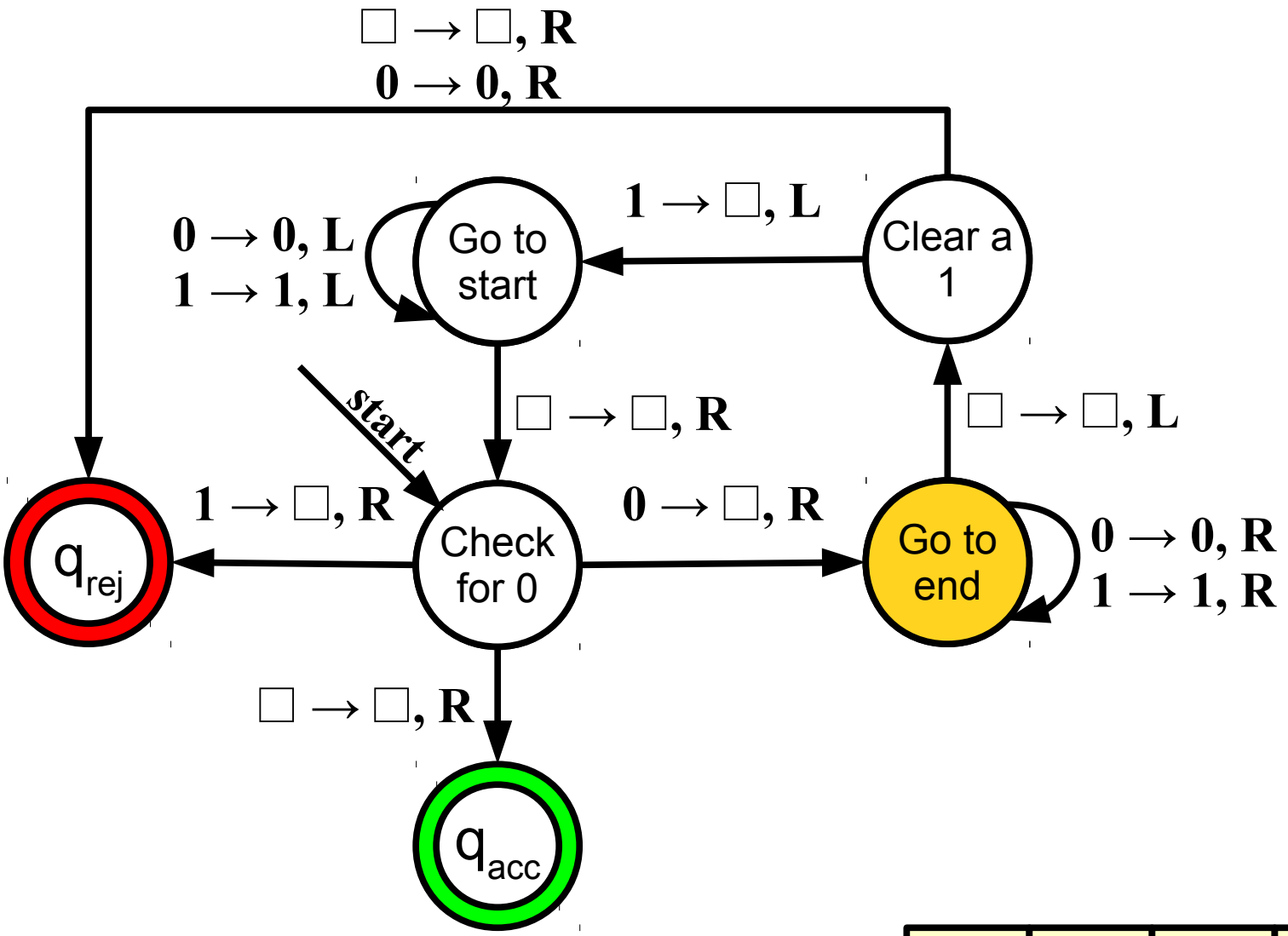


Try running it for six steps.

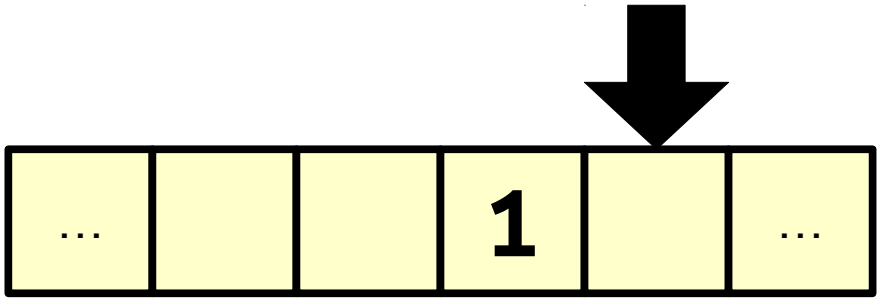


Does this TM halt on this input?

Verification

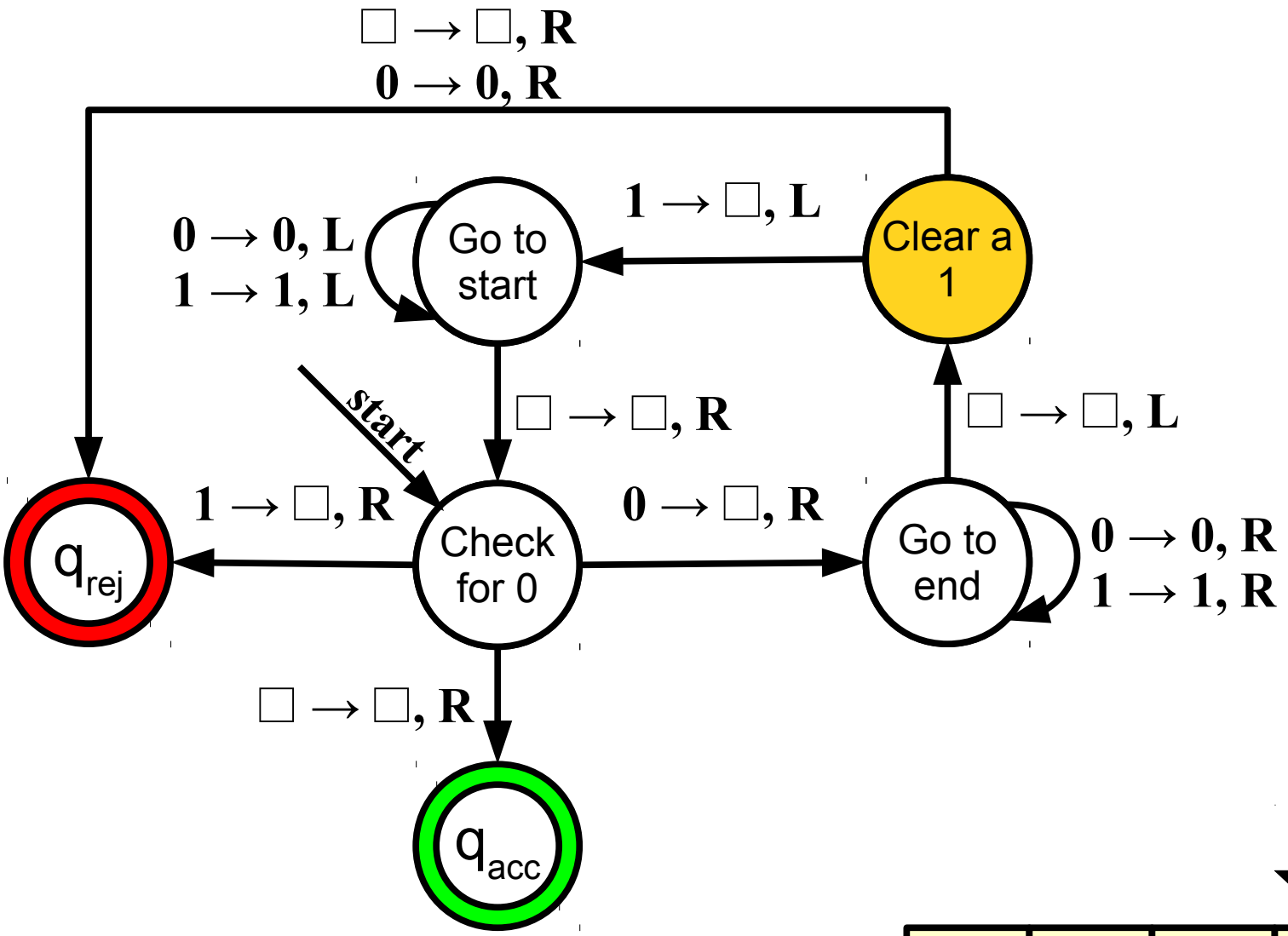


Try running it for six steps.

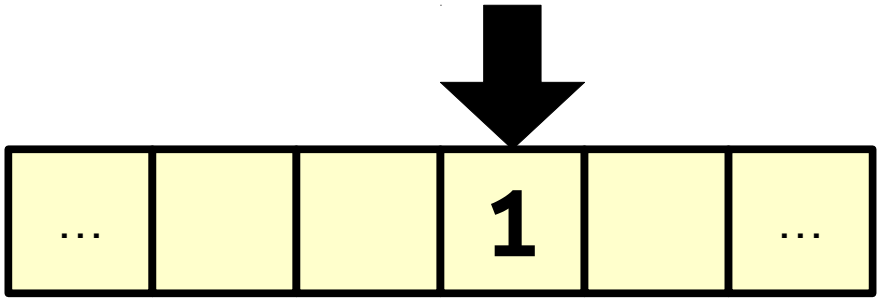


Does this TM halt on this input?

Verification

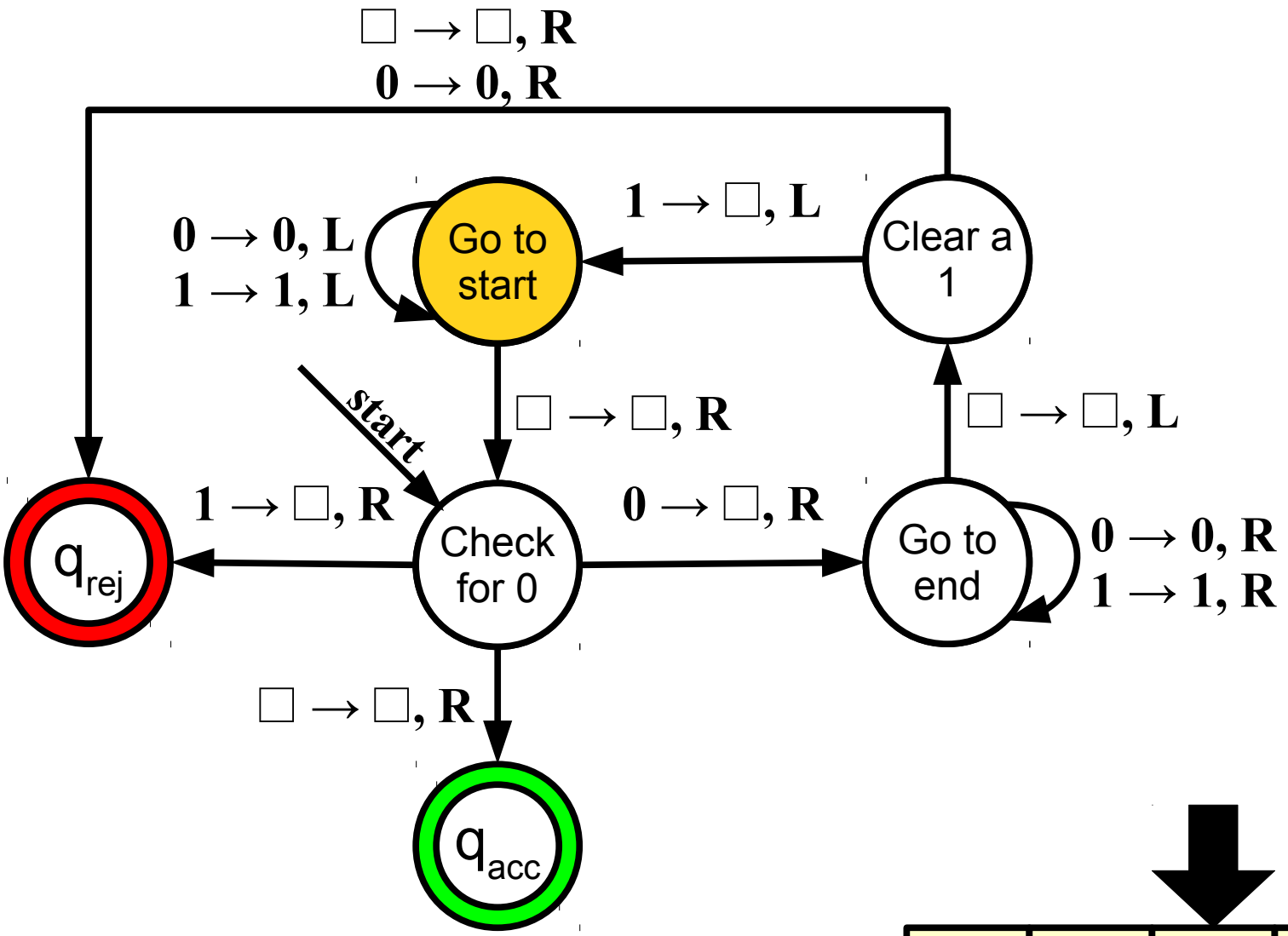


Try running it for six steps.

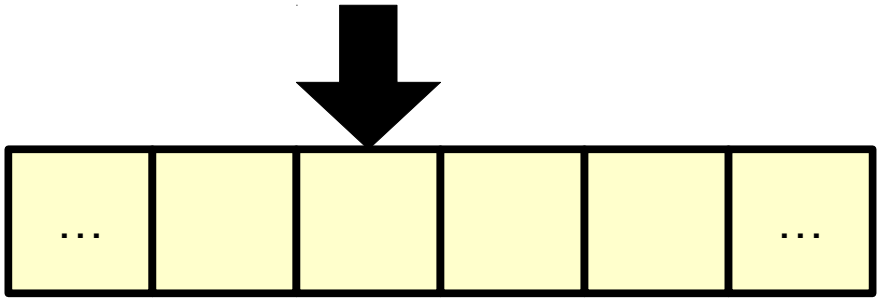


Does this TM halt on this input?

Verification

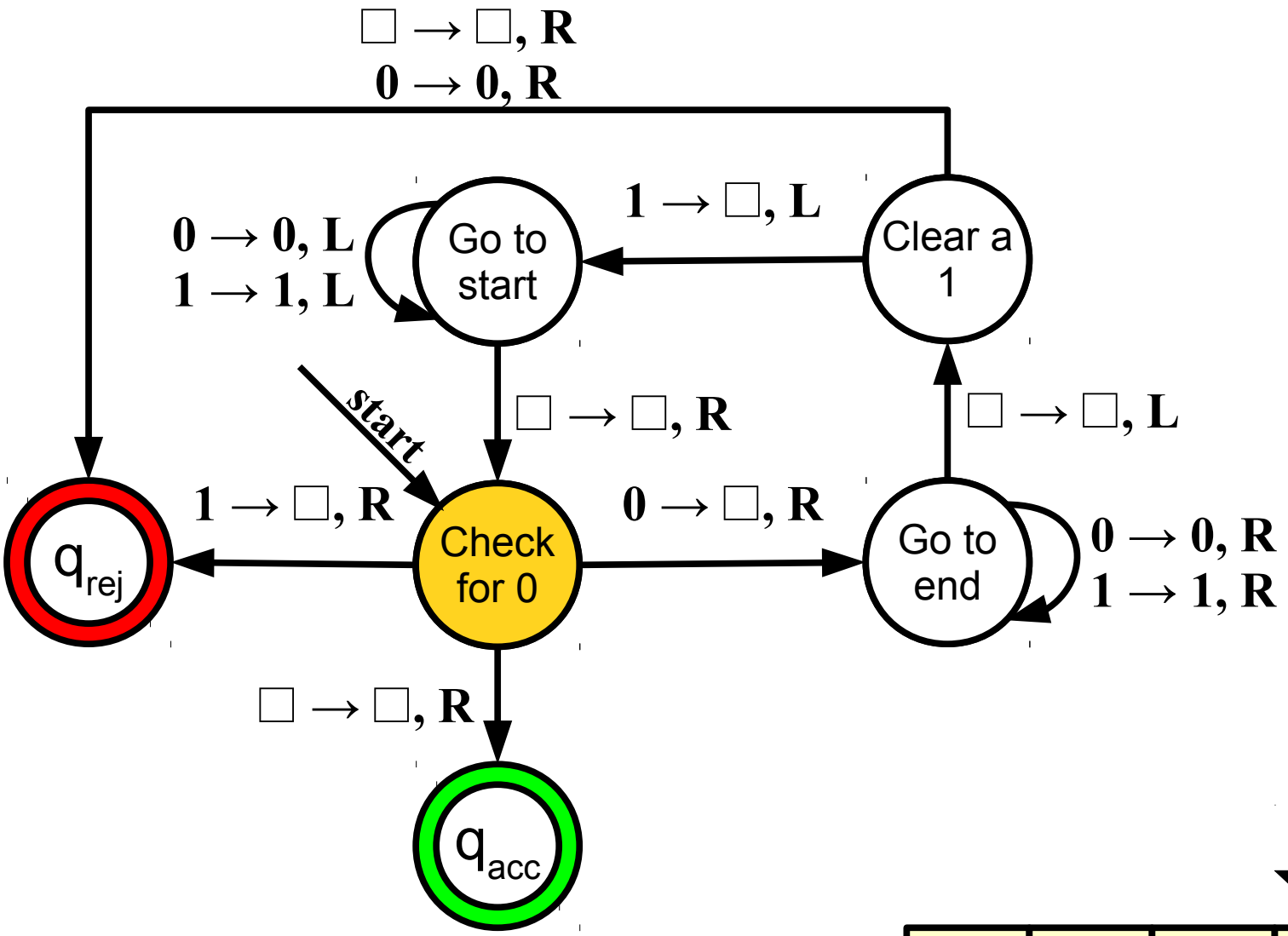


Try running it for six steps.

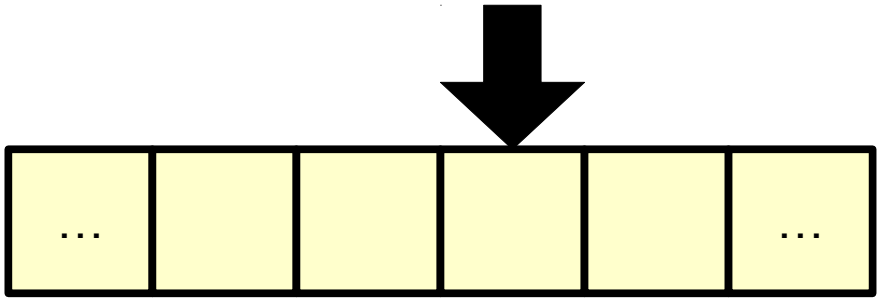


Does this TM halt on this input?

Verification

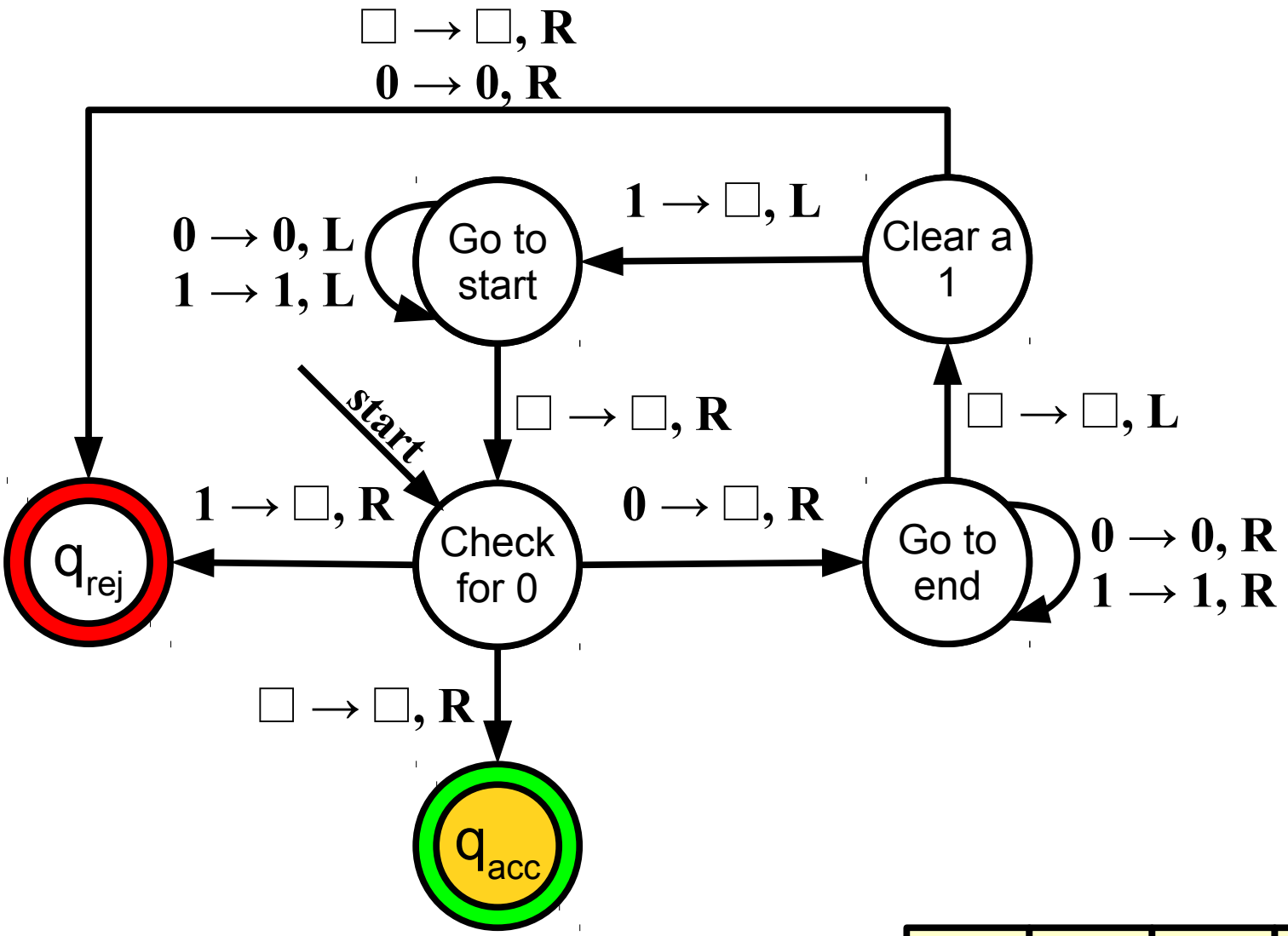


Try running it for six steps.



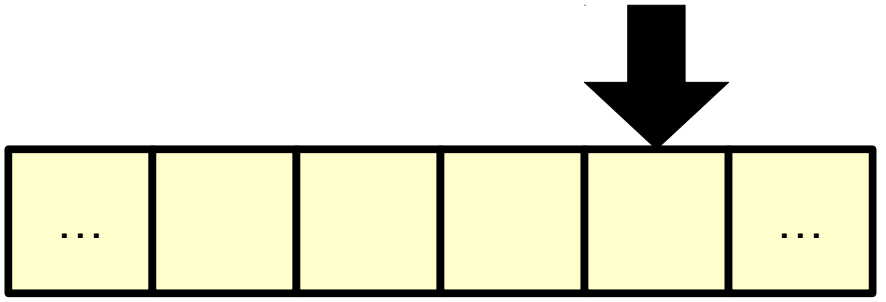
Does this TM halt on this input?

Verification



Try running it for six steps.

Does this TM halt on this input?



Some Verifiers

- Consider *HALT*:

  $HALT = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on } w \}$

```
private boolean checkHalt(TM M, string w, int k) {
    simulate M running on w for k steps.
    if (M is in an accepting state) return true;
    if (M is in a rejecting state) return true;
    return false;
}
```

- Do you see why M halts on w iff there is some k such that `checkHalt(M, w, k)` returns true?
- Do you see why `checkHalt` always halts?

What languages are verifiable?

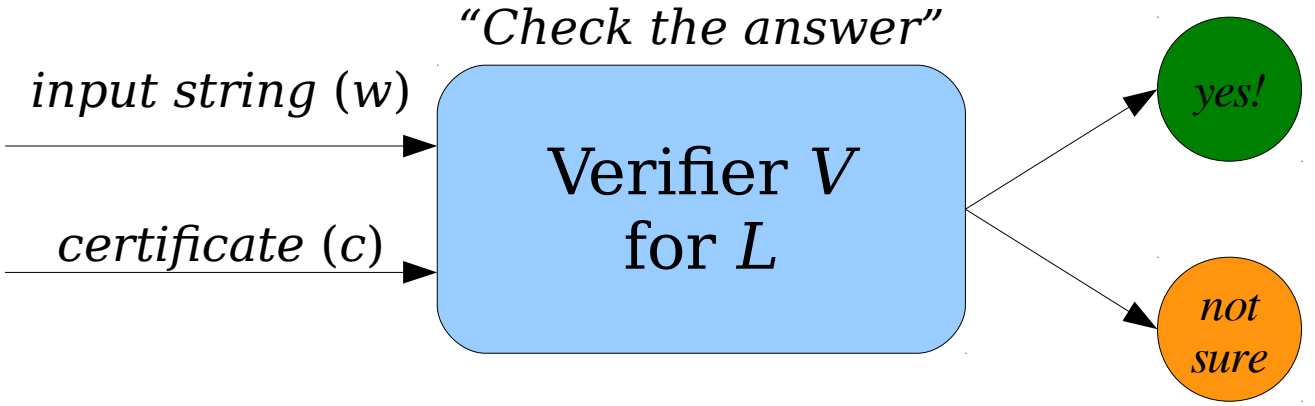
Theorem: If L is a language, then there is a verifier for L if and only if $L \in \mathbf{RE}$.

Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

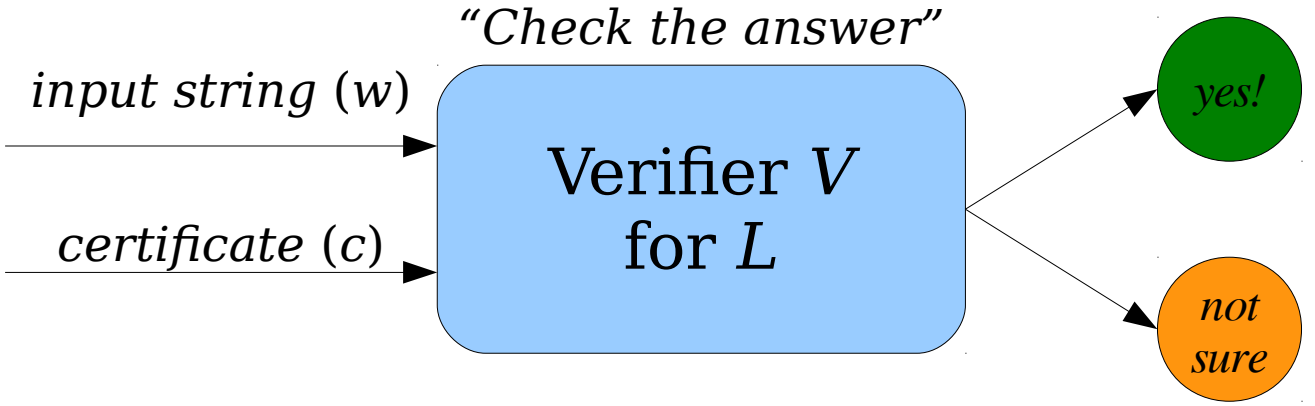
Verifiers and **RE**

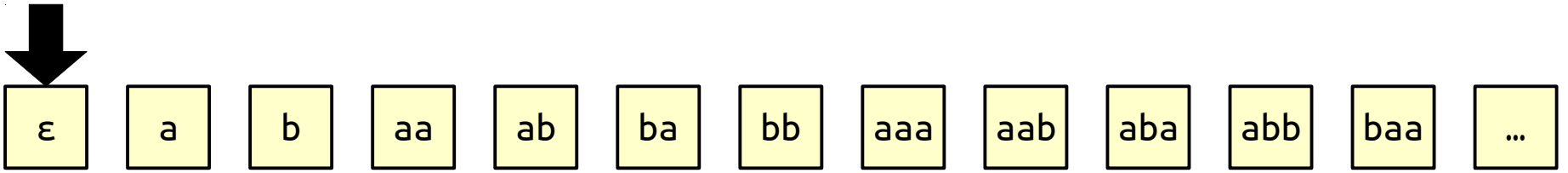
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



Verifiers and **RE**

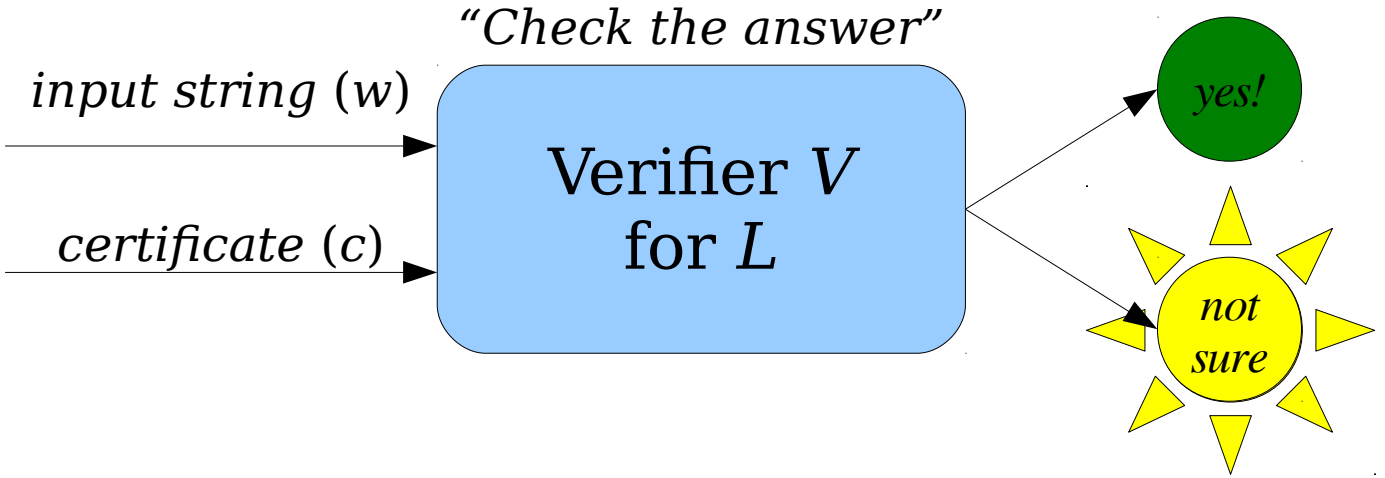
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

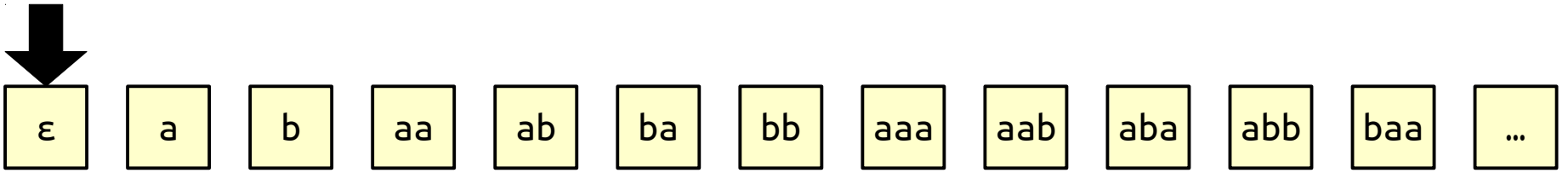




Verifiers and **RE**

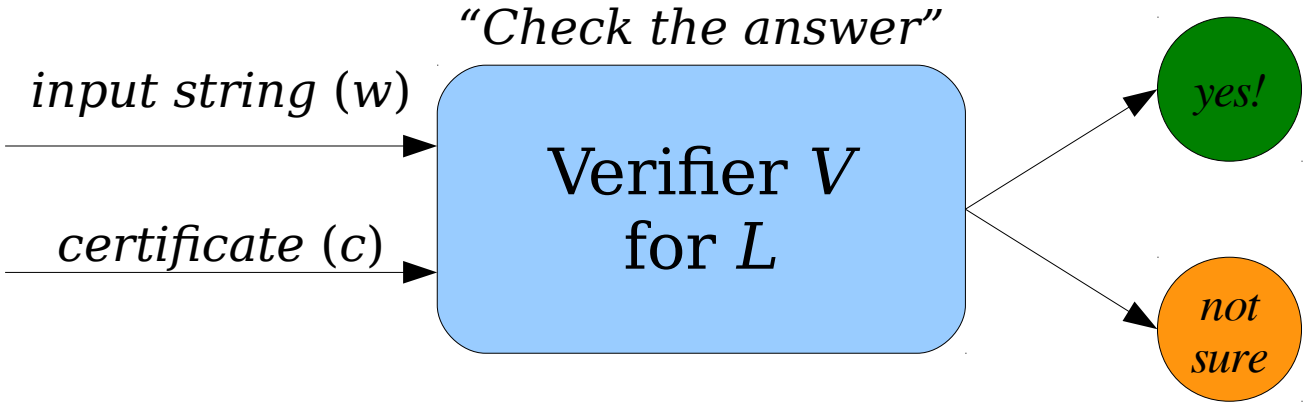
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

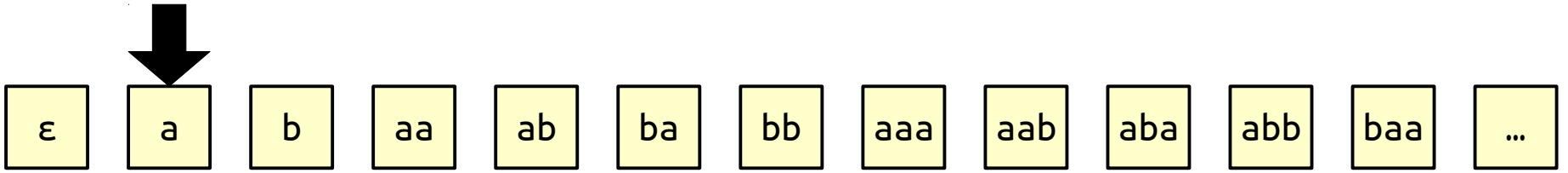




Verifiers and **RE**

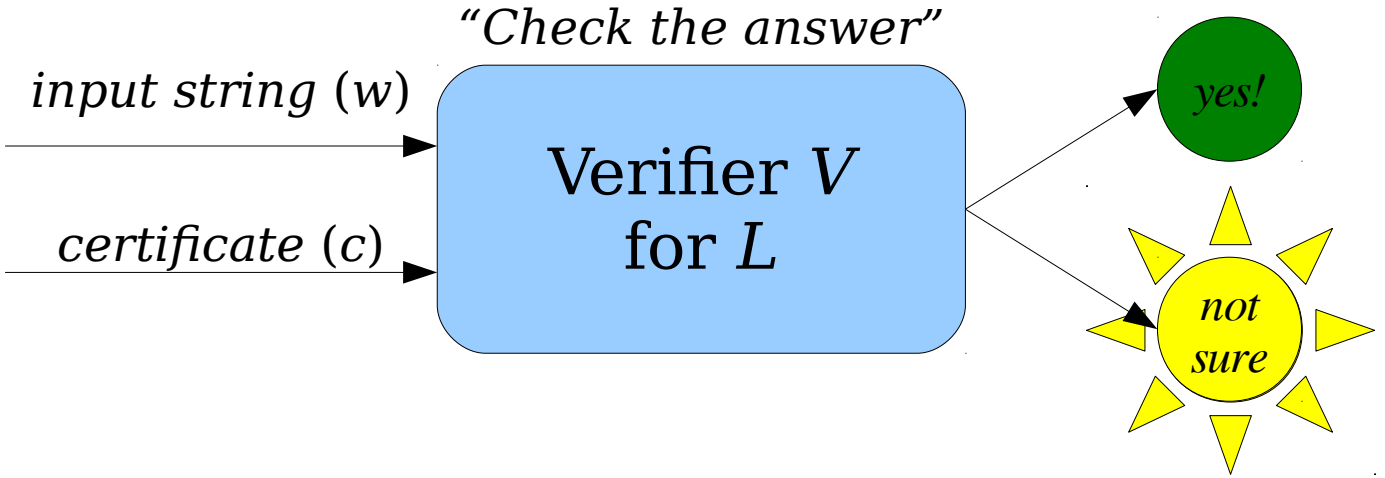
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

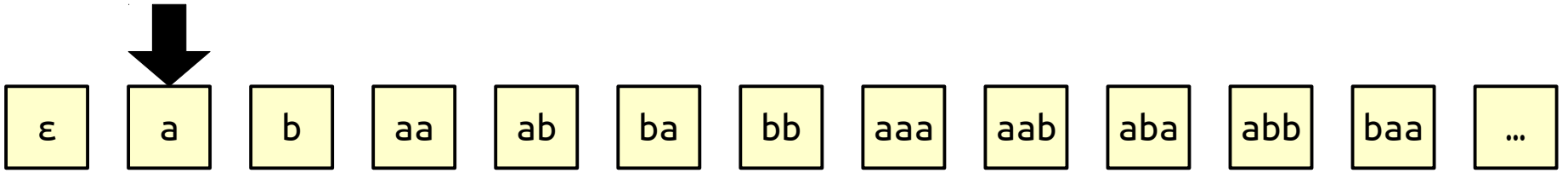




Verifiers and **RE**

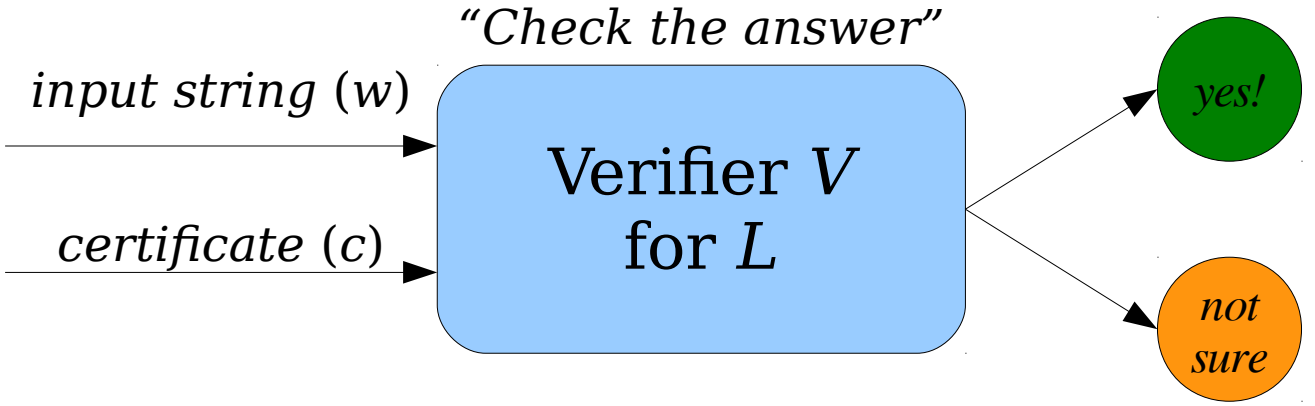
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

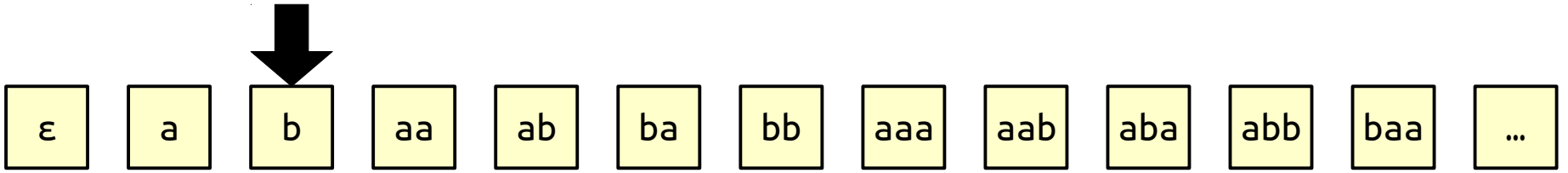




Verifiers and **RE**

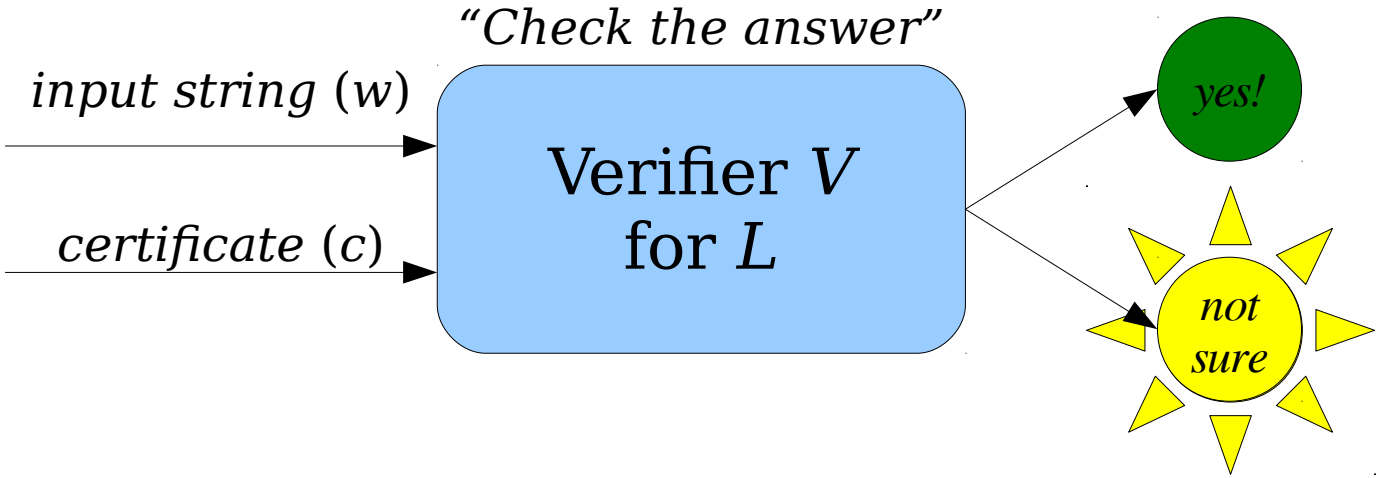
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

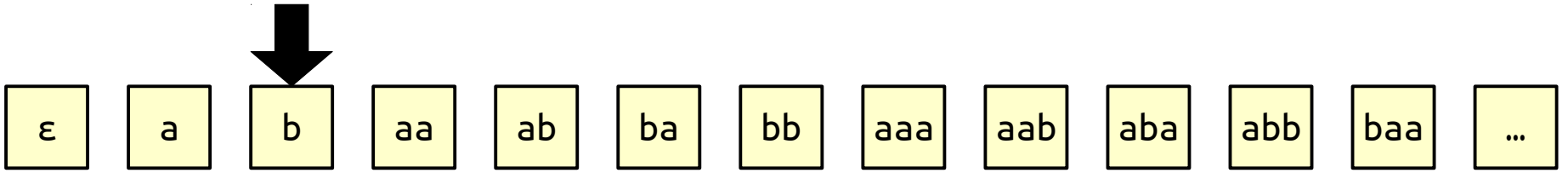




Verifiers and **RE**

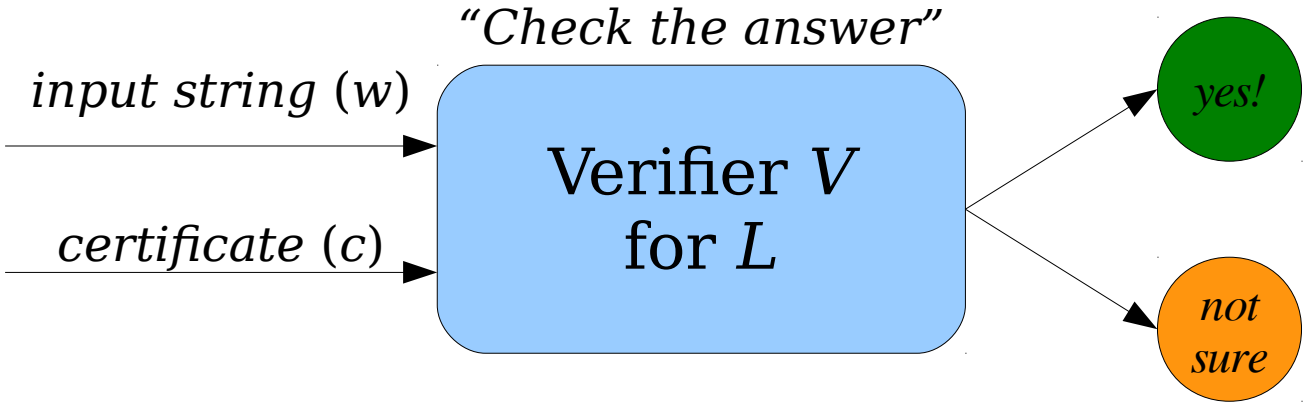
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

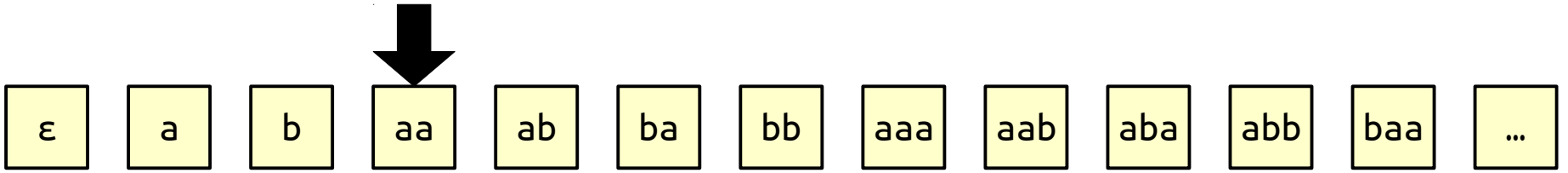




Verifiers and **RE**

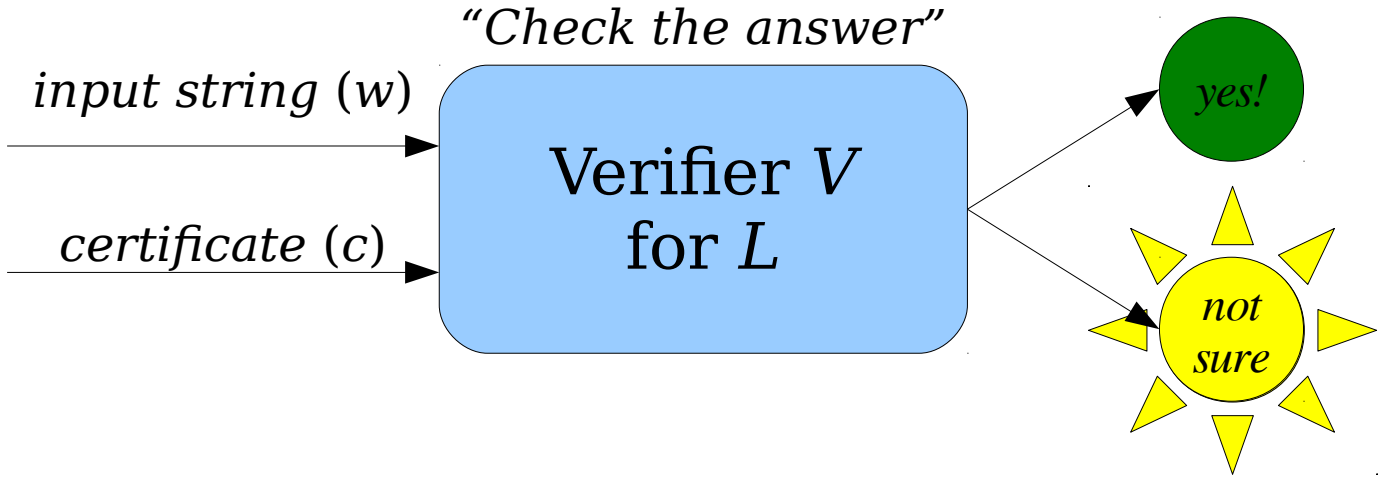
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

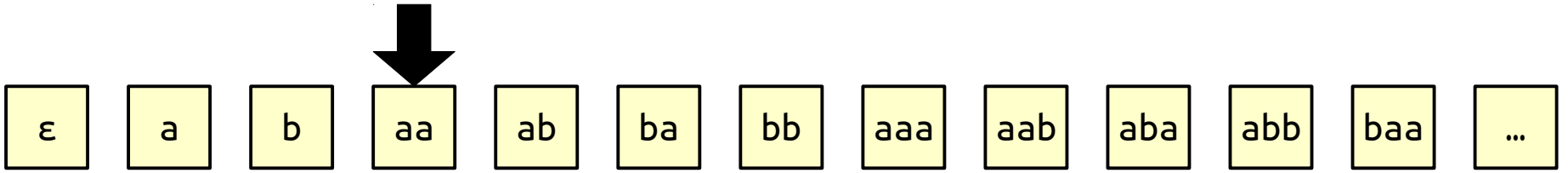




Verifiers and **RE**

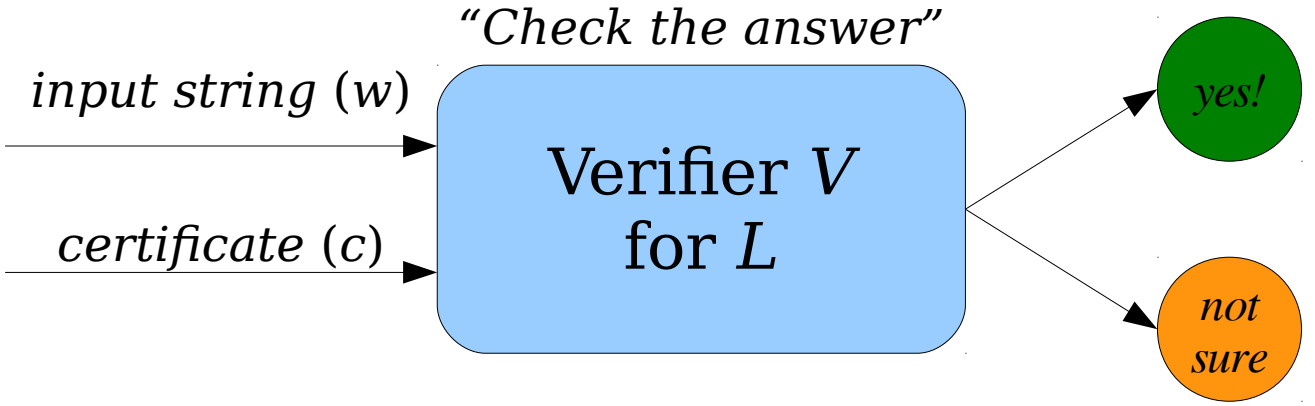
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

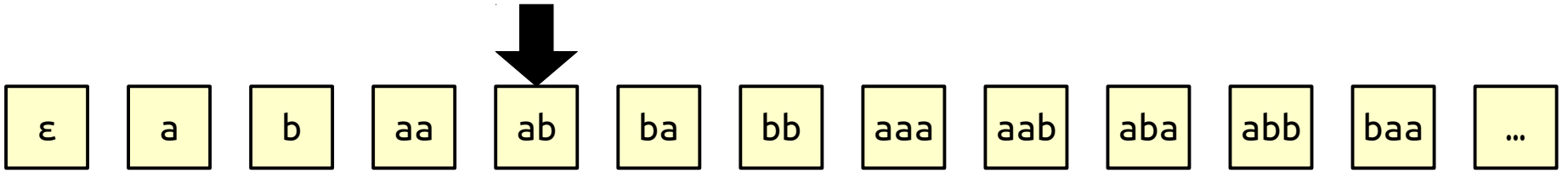




Verifiers and **RE**

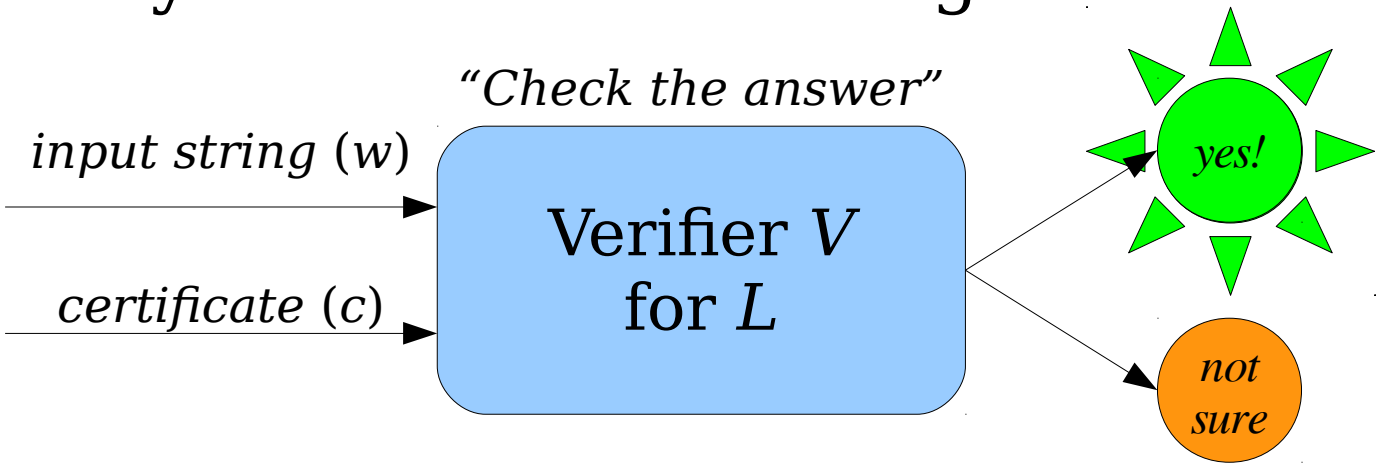
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

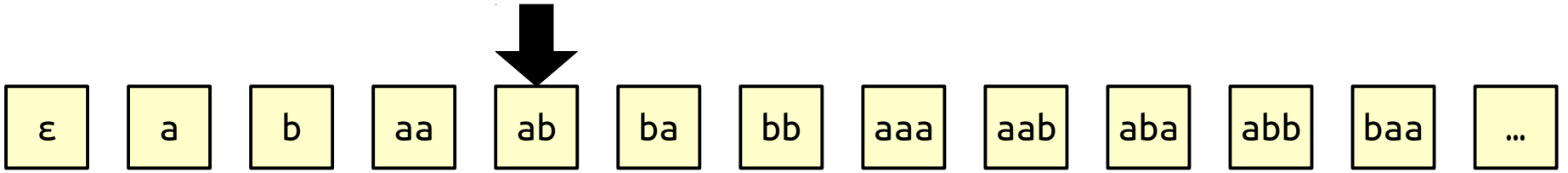




Verifiers and **RE**

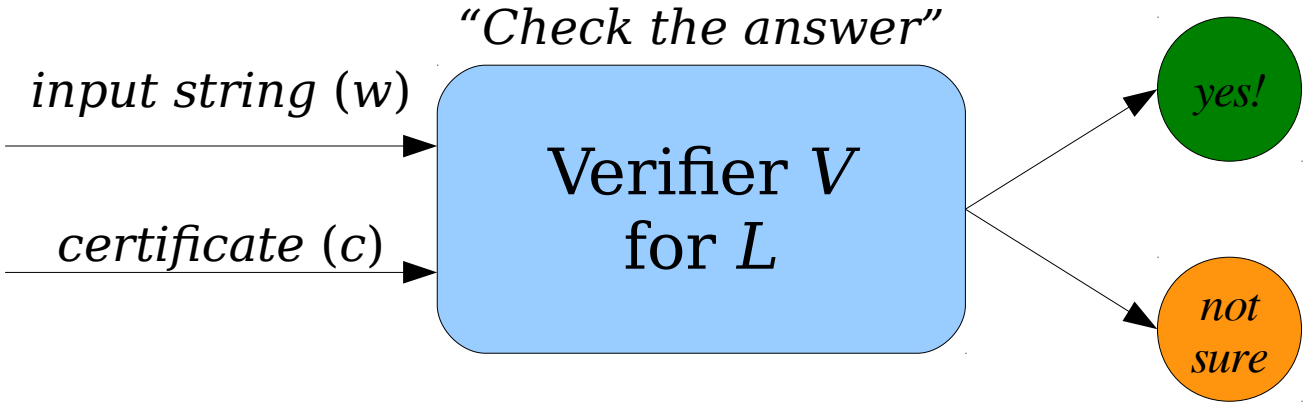
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

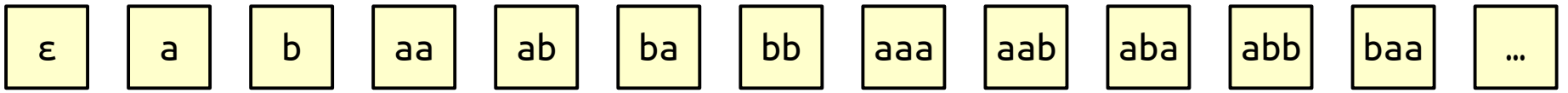




Verifiers and **RE**

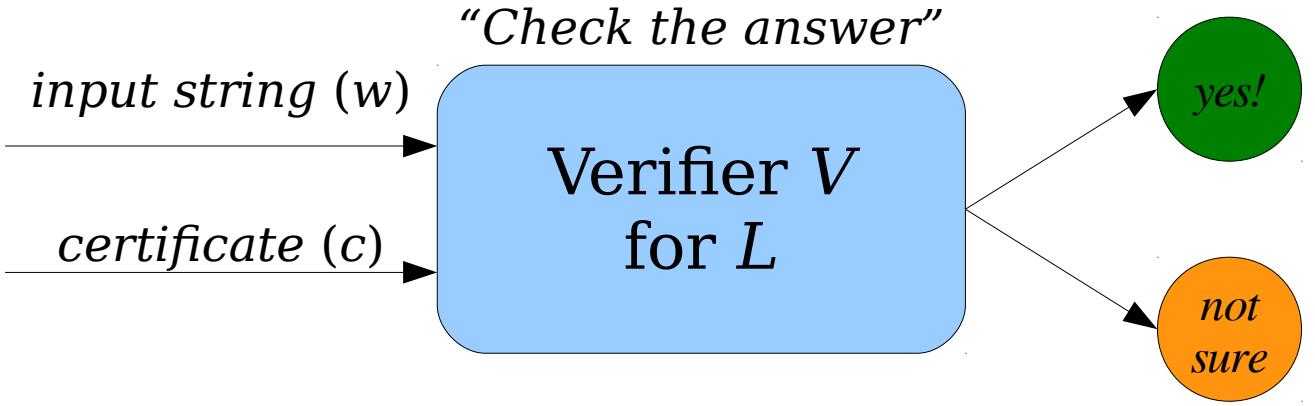
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

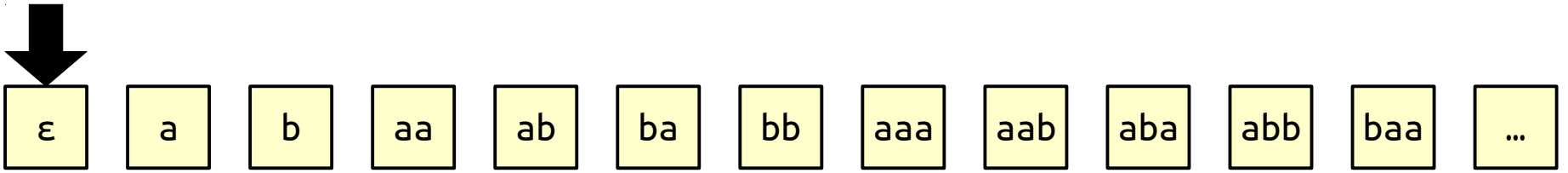




Verifiers and **RE**

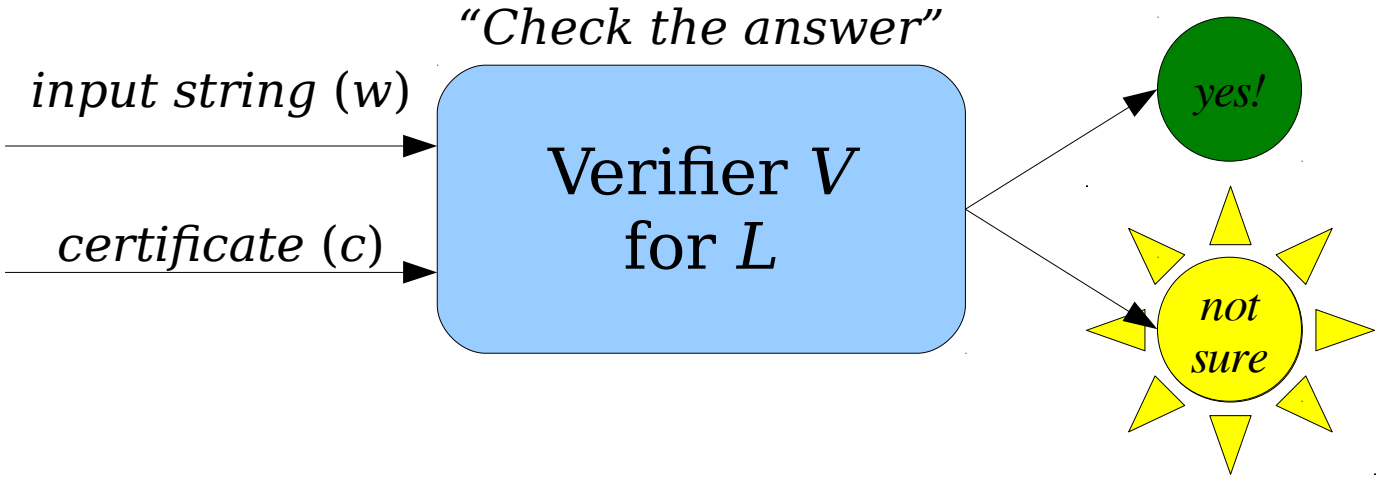
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

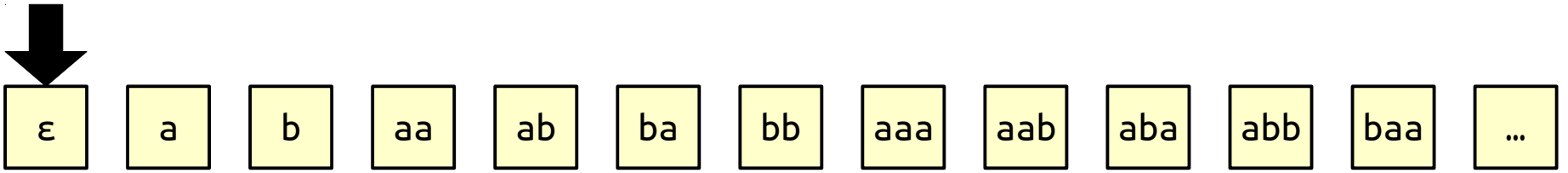




Verifiers and **RE**

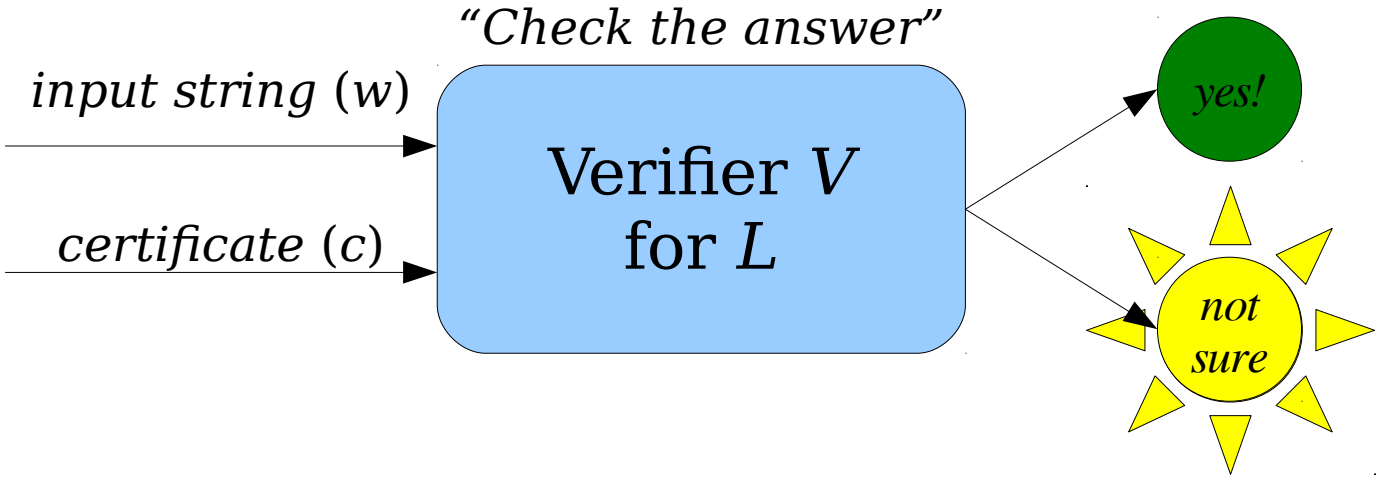
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

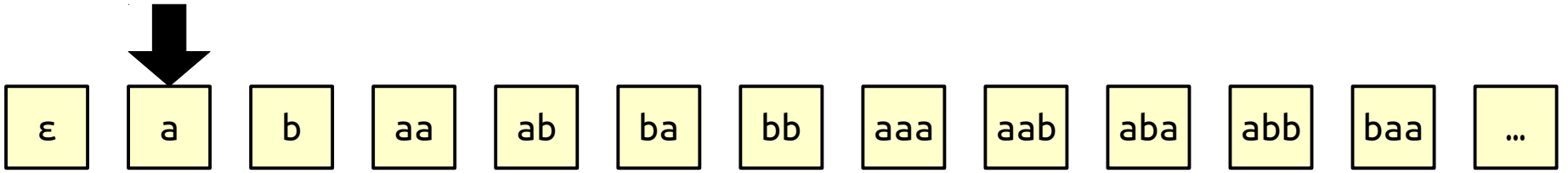




Verifiers and **RE**

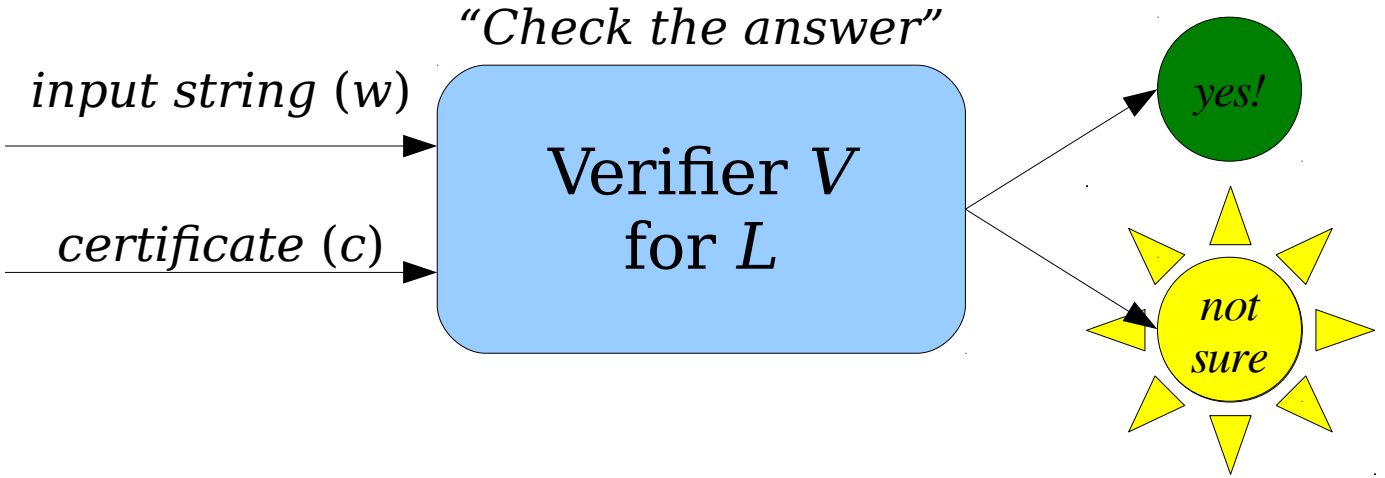
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

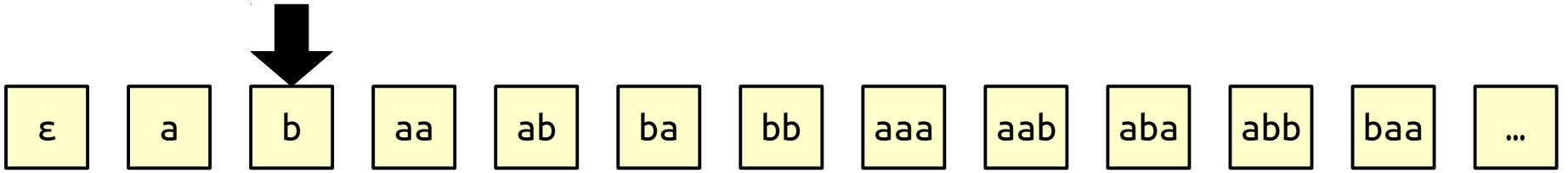




Verifiers and **RE**

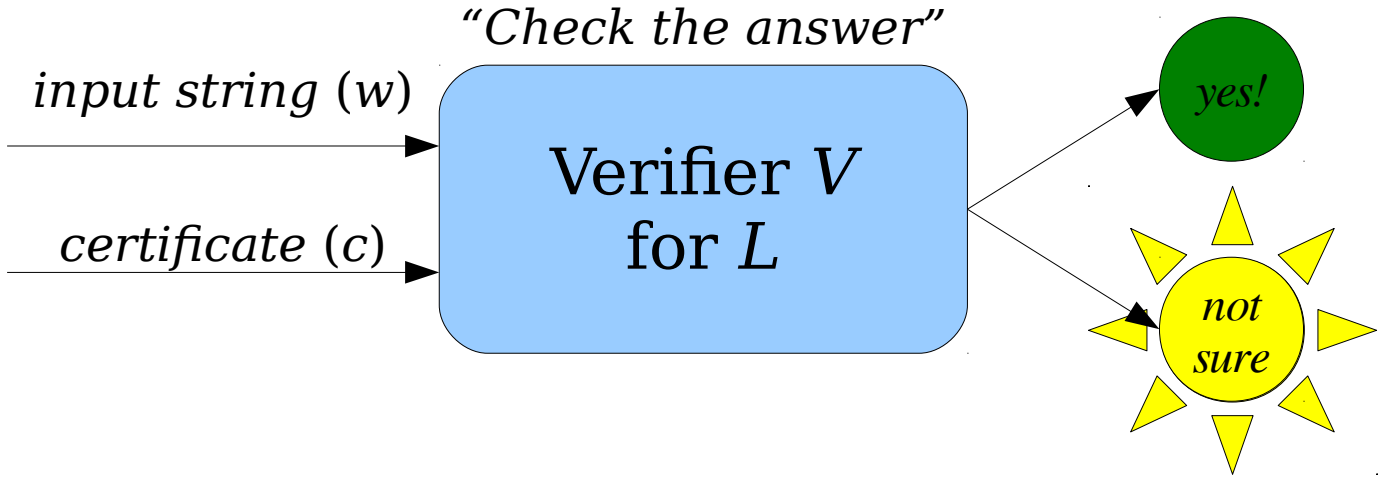
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

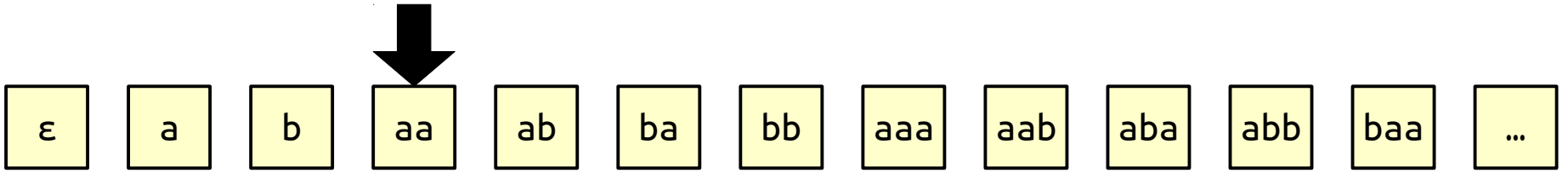




Verifiers and **RE**

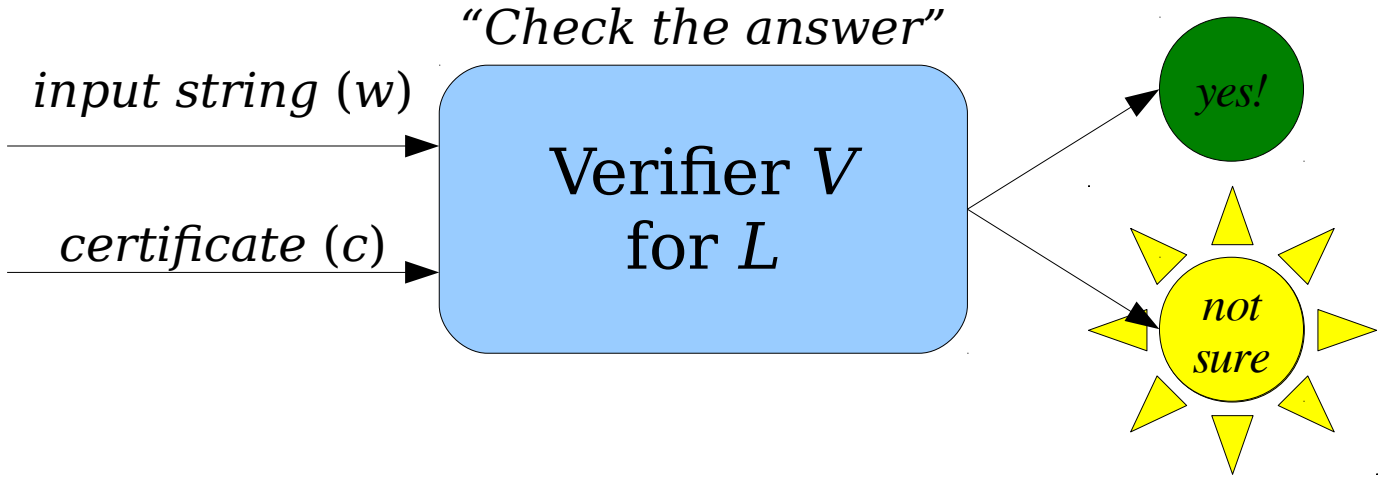
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

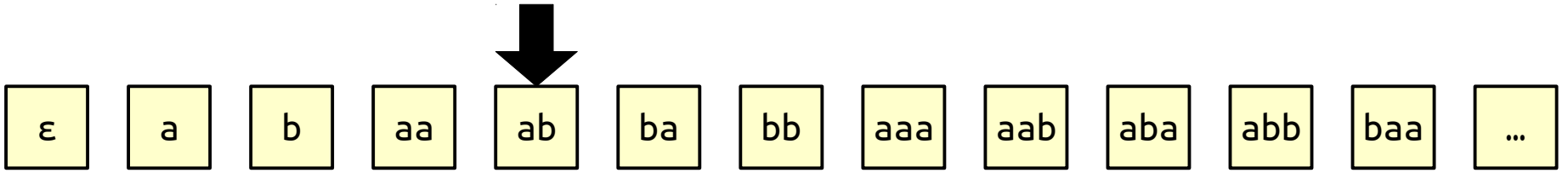




Verifiers and **RE**

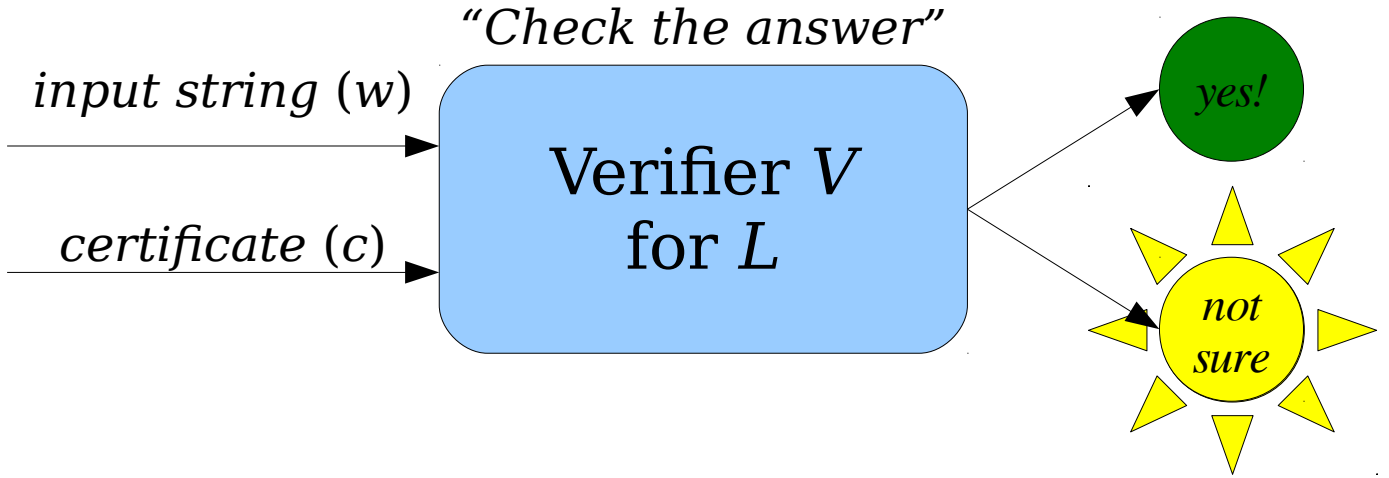
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

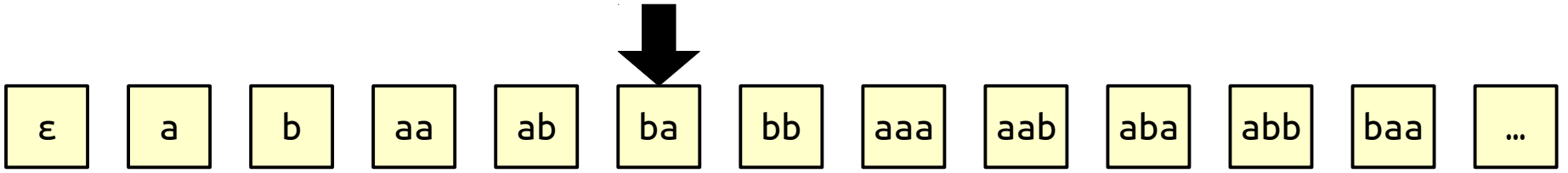




Verifiers and **RE**

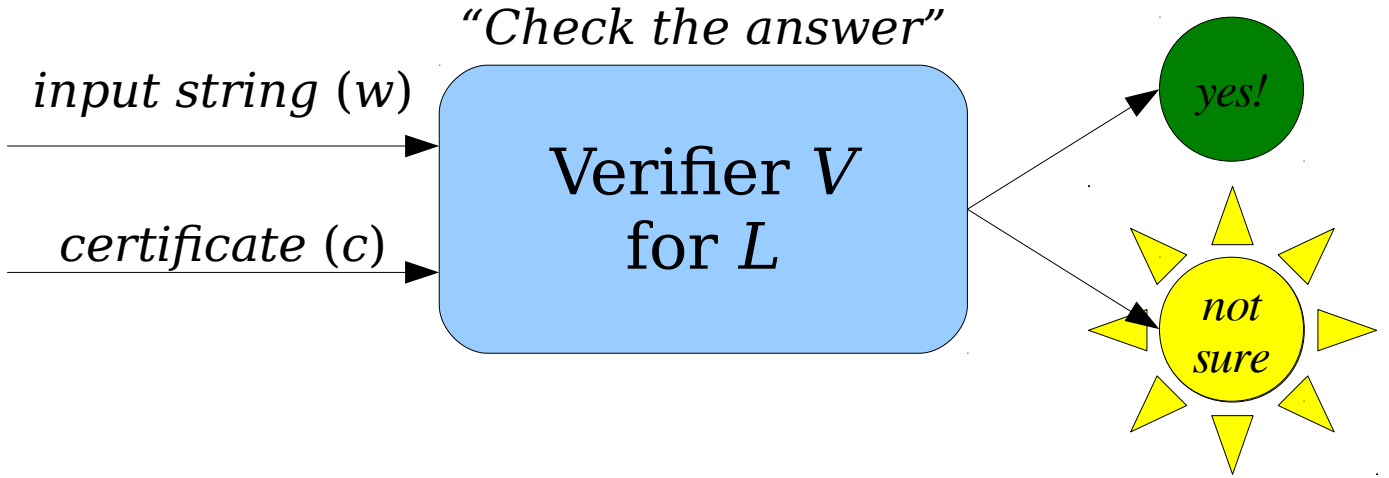
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

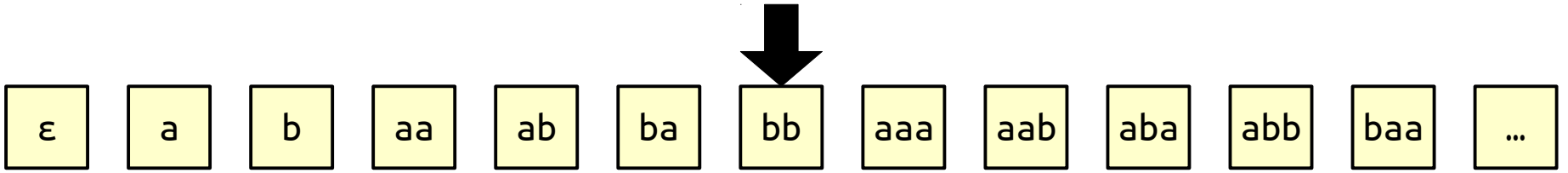




Verifiers and **RE**

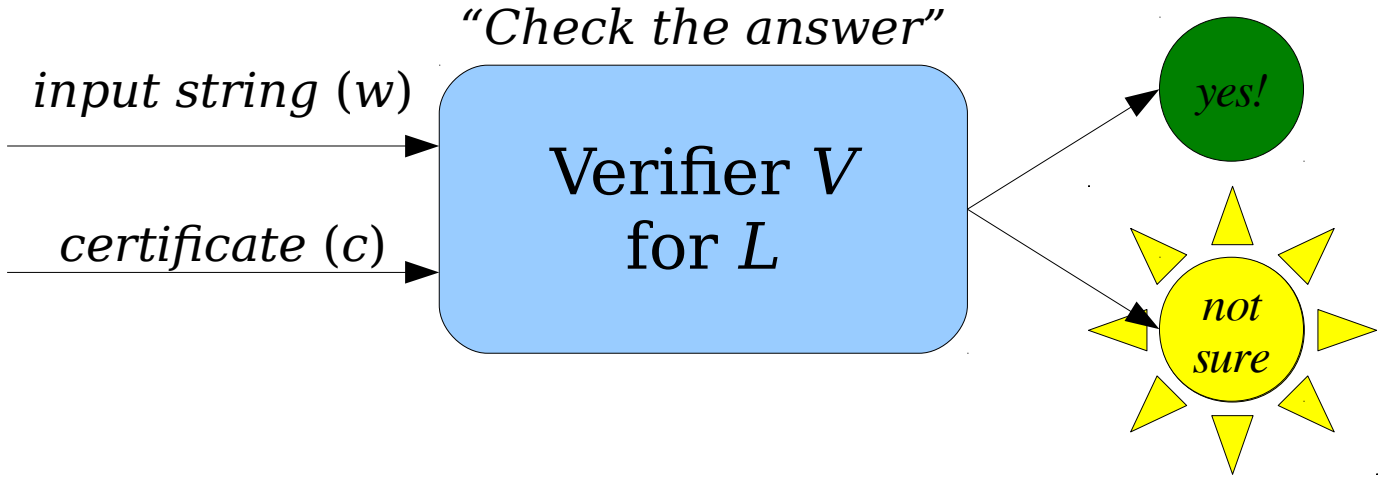
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.

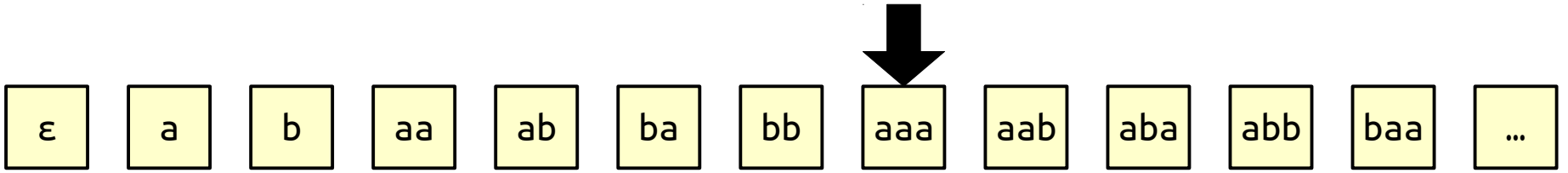




Verifiers and **RE**

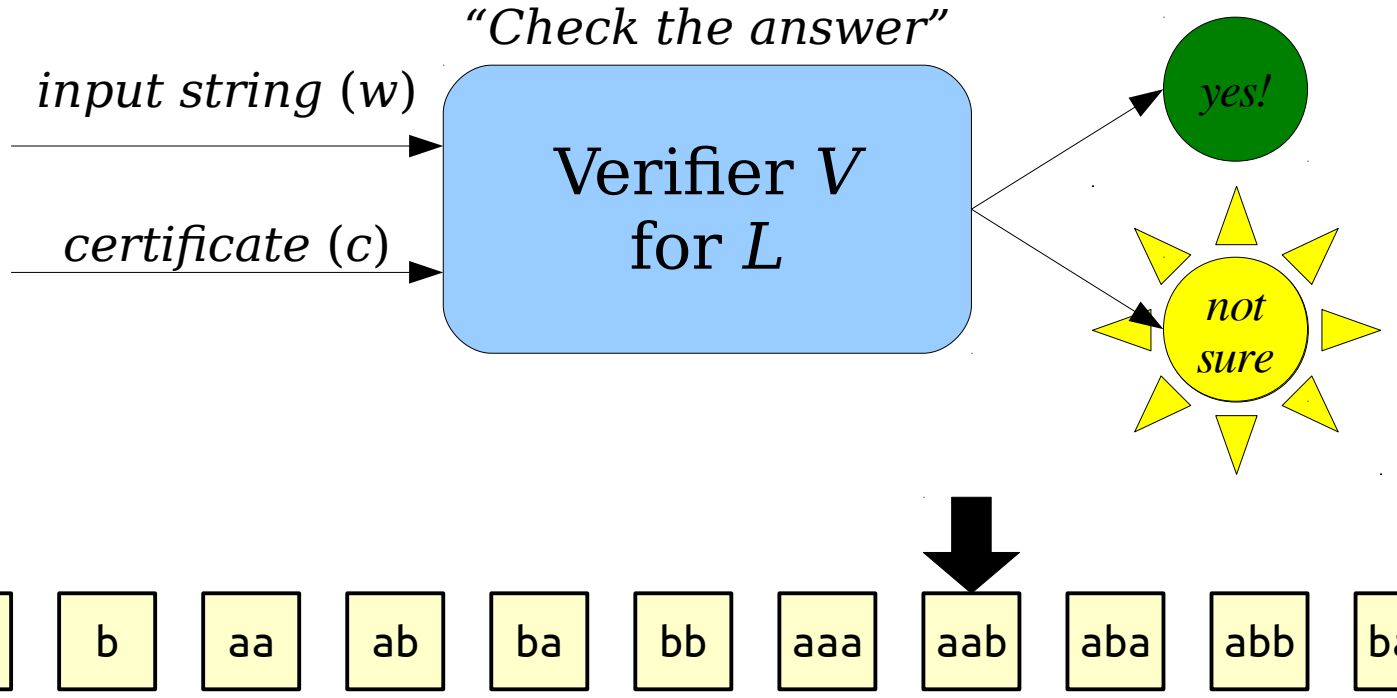
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.





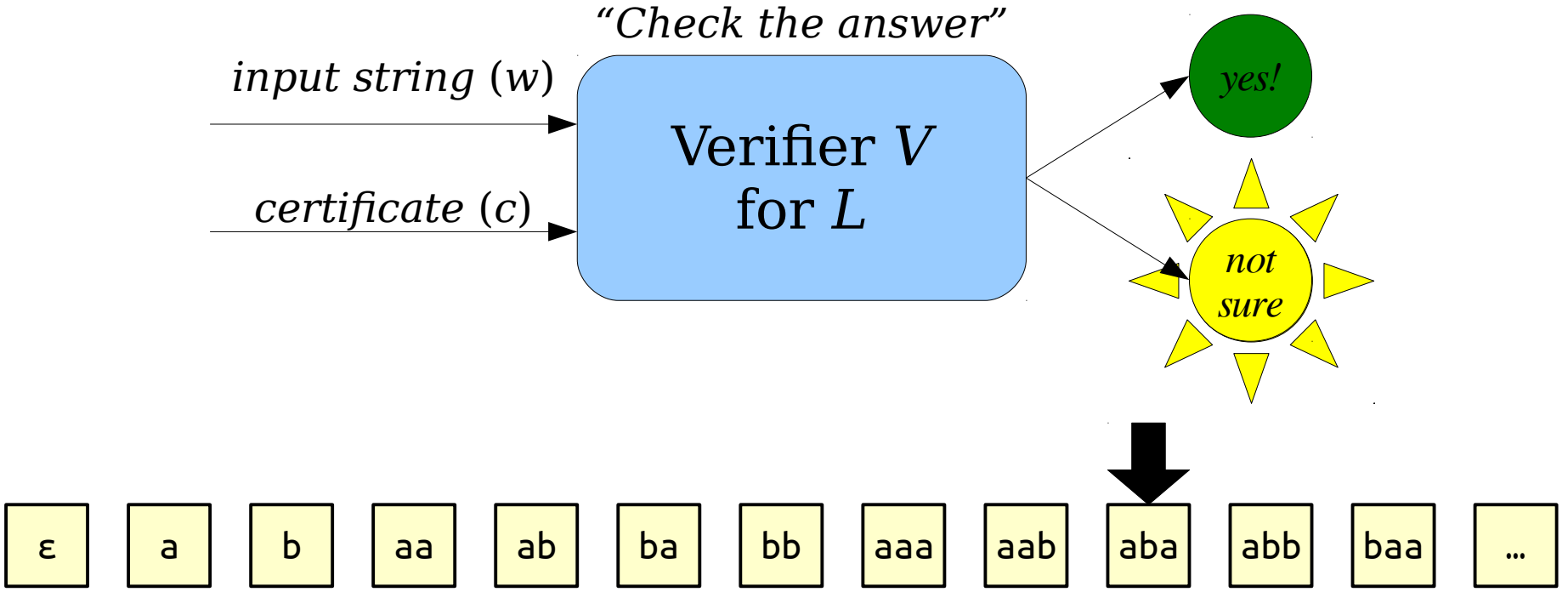
Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



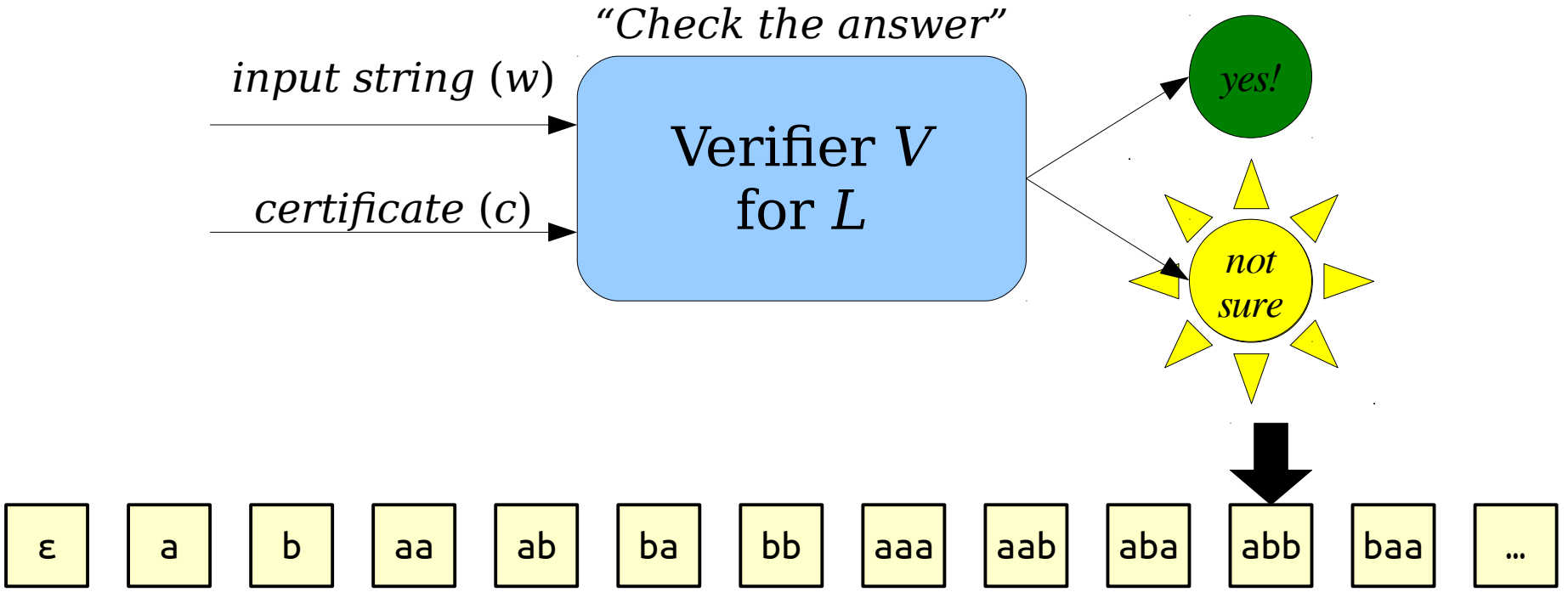
Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



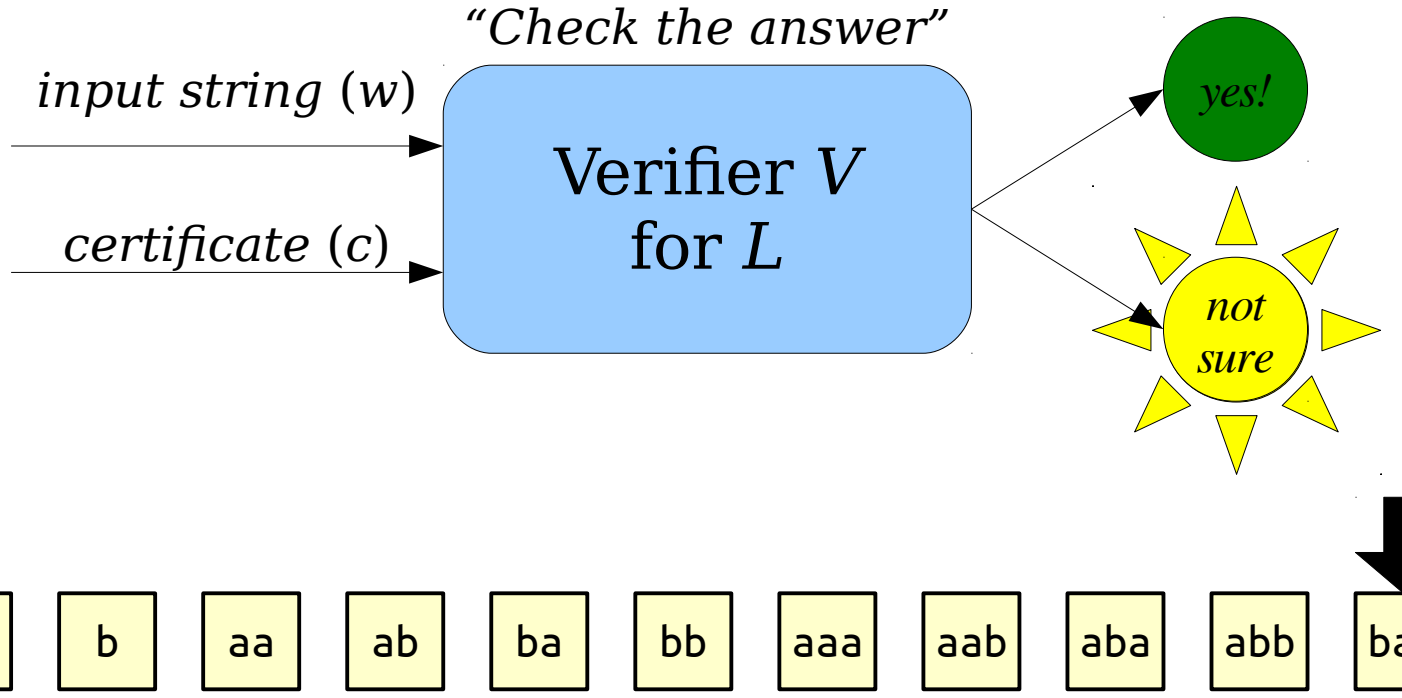
Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



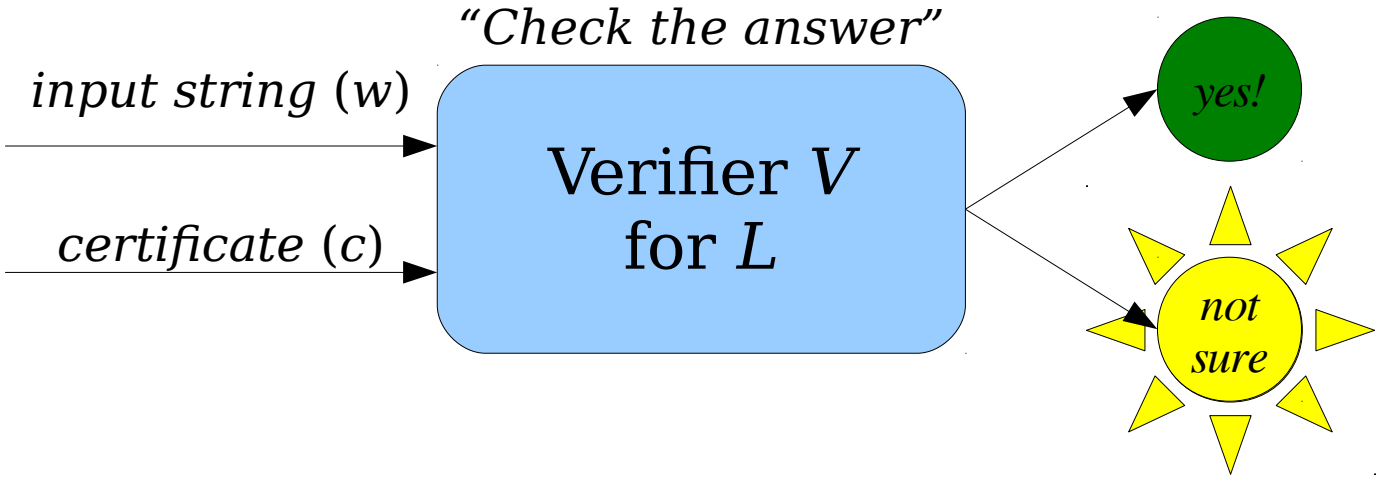
Verifiers and **RE**

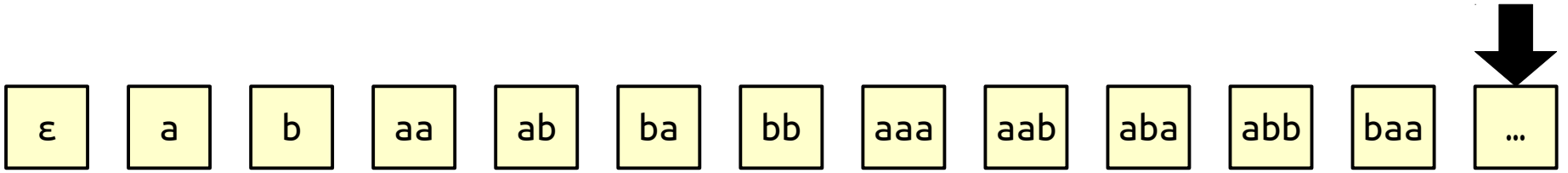
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.





Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof idea:*** Build a recognizer that tries every possible certificate to see if $w \in L$.
- ***Proof sketch:*** Consider this program:

```
private boolean isInL(string w) {
    for (i from 0 to ∞) {
        for (each string c of length i) {
            if (V accepts ⟨w, c⟩) return true;
        }
    }
}
```

If $w \in L$, then there is a $c \in \Sigma^*$ such that V accepts $\langle w, c \rangle$. This program tries all possible strings as certificates, so it eventually finds c, watches V accept $\langle w, c \rangle$, then and accepts w. If $w \notin L$, there is no $c \in \Sigma^*$ where V accepts $\langle w, c \rangle$, so this program loops on w. ■

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof goal:*** Beginning with a recognizer M for the language L, show how to construct a verifier V for L.
- The machine M will accept w if $w \in L$. If M doesn't accept w, then $w \notin L$.
- A certificate is supposed to be some sort of "proof" that the string is in the language. Since the only thing we know about L is that M is a recognizer for it, our certificate would have to tell us something about what M does.
- We need to choose a certificate with the following properties:
  - We can decide in finite time whether a certificate is right or wrong.
  - A "good" certificate proves that $w \in L$ (meaning M accepts w)
  - A "bad" certificate never proves $w \in L$.
- ***Idea:*** If M accepts w, it will do so in finitely many steps. What if our certificate is the number of steps?

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof sketch:*** Let L be an **RE** language and let M be a recognizer for it. Then show that this is a verifier for L:

```
private boolean checkInL(string w, int k) {
    run M on w for k steps;
    if (M is in an accepting state) return true;
    else return false;
}
```

RE and Proofs

- Verifiers and recognizers give two different perspectives on the “proof” intuition for **RE**.
- Verifiers are explicitly built to check proofs that strings are in the language.
  - If you know that some string w belongs to the language and you have the proof of it, you can convince someone else that $w \in L$.
- You can think of a recognizer as a device that “searches” for a proof that $w \in L$.
  - If it finds it, great!
  - If not, it might loop forever.

RE and Proofs

- If the **RE** languages represent languages where membership can be proven, what does a non-**RE** language look like?
- Intuitively, a language is *not* in **RE** if there is no general way to prove that a given string $w \in L$ actually belongs to L.
- In other words, even if you knew that a string was in the language, you may never be able to convince anyone of it!

Time-Out for Announcements!

Problem Sets

- Problem Set Eight was due at the start of class today.
  - Can extend up to Monday at 3:00PM using late days.
- Problem Set Nine goes out now. It's due next Friday at 3:00PM.
  - ***No late days may be used on this assignment***. That's a university policy – sorry about that!
  - Explore the limits of the **R** and **RE** languages and some practical applications!
  - As always, ask questions if you have them!
- We've posted two guides, the Guide to Self-Reference and the Guide to the Lava Diagram, to the course website. We strongly recommend reading over these while working on the problem set.

Final Exam Logistics

- The final exam is Monday, December 12 from 3:30PM – 6:30PM, location TBA.
  - Sorry about the timing! Blame the registrar.
- Students with OAE accommodations: if you have not yet sent us an OAE letter, you need to do so by ***Monday***.

Final Exam Logistics

- Topic coverage:
  - Cumulative exam. Covers topics from PS1 – PS9 and all of our lectures.
  - Topics from next week will be deemphasized. You'll need to know the basics, but that's about it.
  - Exam is roughly 50% topics from PS1 – PS5 (discrete math and proofs) and 50% topics from PS6 – PS9 (automata and computability theory).
- The exam is closed-book, closed-computer, and limited-note. You can bring a double-sided sheet of 8.5” × 11” with you to the exam.

Supporting Materials

- We've released solutions to EPP8, EPP9, and EPP10 on the course website.
- ***Please make a good effort to solve the problems before looking at the solutions!*** There's a ton to be gained by struggling through the problems and getting hints before arriving at the answer.
- As always, you're welcome to ask us questions in office hours or over Piazza about these problems.

Supporting Materials

- We've released ***three*** practice final exams to the course website:
  - The final exam from Winter 2016.
  - The final exam from Fall 2015, with edits.
  - The final exam from Spring 2015, with edits.
- Please, please, please, please, *please* do not just skim through these! You really should take them under semi-realistic conditions and get a sense for where you stand.
- Solutions will go up on Monday.

Your Questions

“It seems that by studying CS I'm developing a narrow set of specific skills for solving problems in the world. What sort of non-CS skills do you think engineers should develop in order to be able to do more 'good'?”

It's easy to lose perspective on just how valuable your CS skills are, though! You're going to be in a position to build consumer software, do large-scale data analysis, build back-end systems, design self-driving cars, predict crop harvests and estimate income levels, etc. It's easy to lose perspective on this when you're in giant classes with other people learning those skills, but trust me, they're rarer than you think.

To be a successful engineer, you need to be good at working in teams, communicating clearly, and designing for maintenance and growth. To really make a big impact, you need to be good at finding a need and filling it. Learning more about the world, how people act, and what problems need solving is key to doing this well.

“Keith, how did you achieve a healthy work/life balance as an undergrad?

I didn't. I had an unhealthy obsession with my grades and my classes that made me repeatedly miserable. It's only being years out from graduating and having the ability to look back at things that I see what I was doing wrong.

This is one of the reasons why I keep talking about things like this during the “ask me anything” sections of this course – it's so easy to fall into a trap and not realize it. If you're genuinely not feeling happy or feel totally overwhelmed, don't ignore those signals. They're trying to tell you something important.

“Well that was dark.
Show us something cute.”

- Totally something actually asked on Kiunei.

"Just for fun, can you relate what we've learned to Westworld? I'll also take *2001: Space Odyssey* or anything by Asimov."

Nope! But I can relate it to Dr. Seuss!

http://www.lel.ed.ac.uk/~gpullum/loopsnoop.html

Back to CS103!

Finding Non-**RE** Languages

Finding Non-**RE** Languages

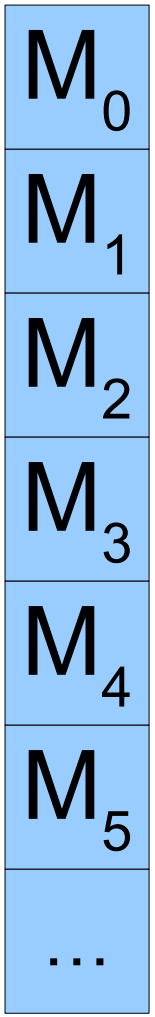
- Right now, we know that non-**RE** languages exist, but we have no idea what they look like.
- How exactly might we find one?
- The answer brings us all the way back to our very first lecture!

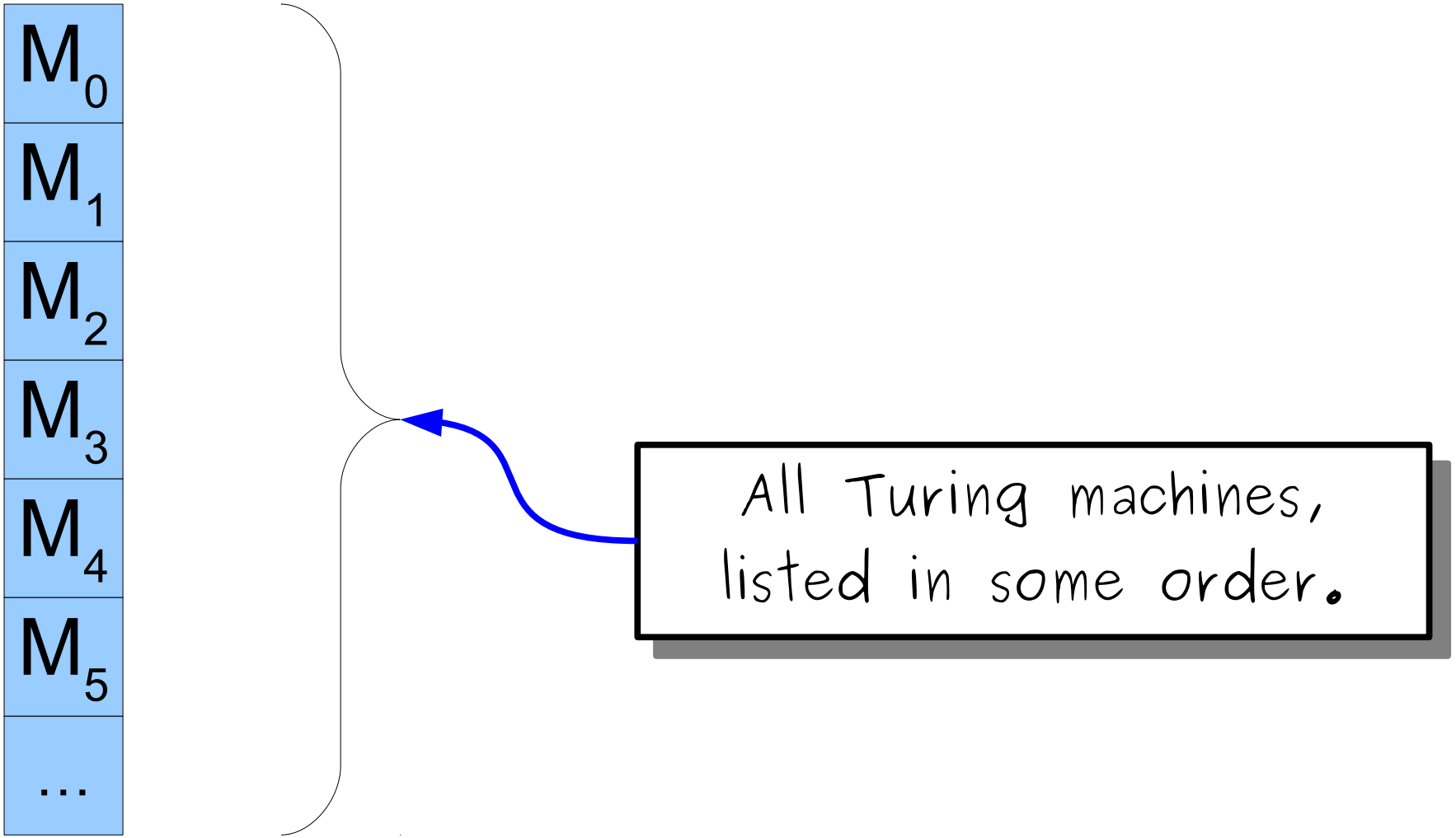
Languages, TMs, and TM Encodings

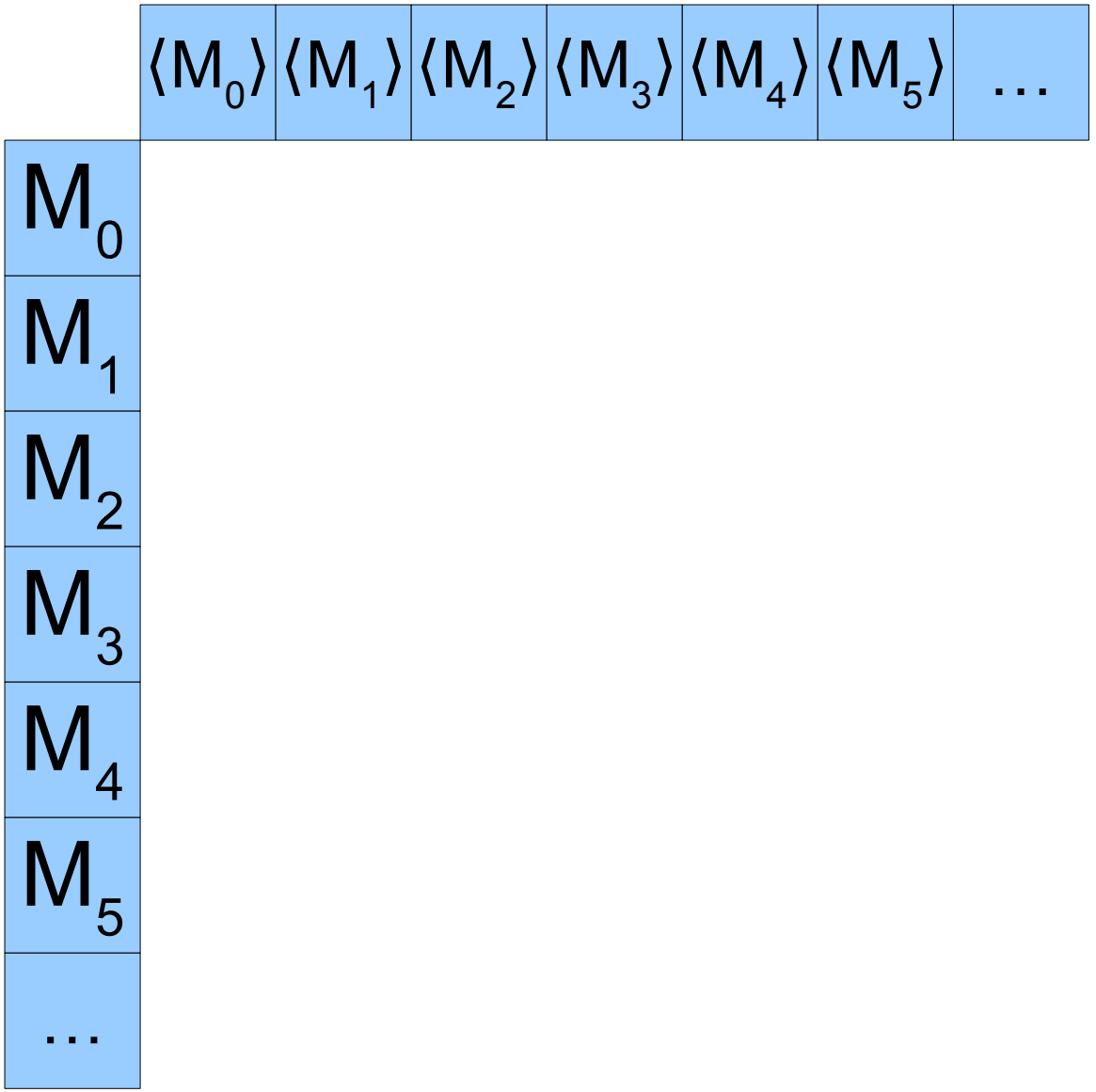
- Recall: The language of a TM M is the set

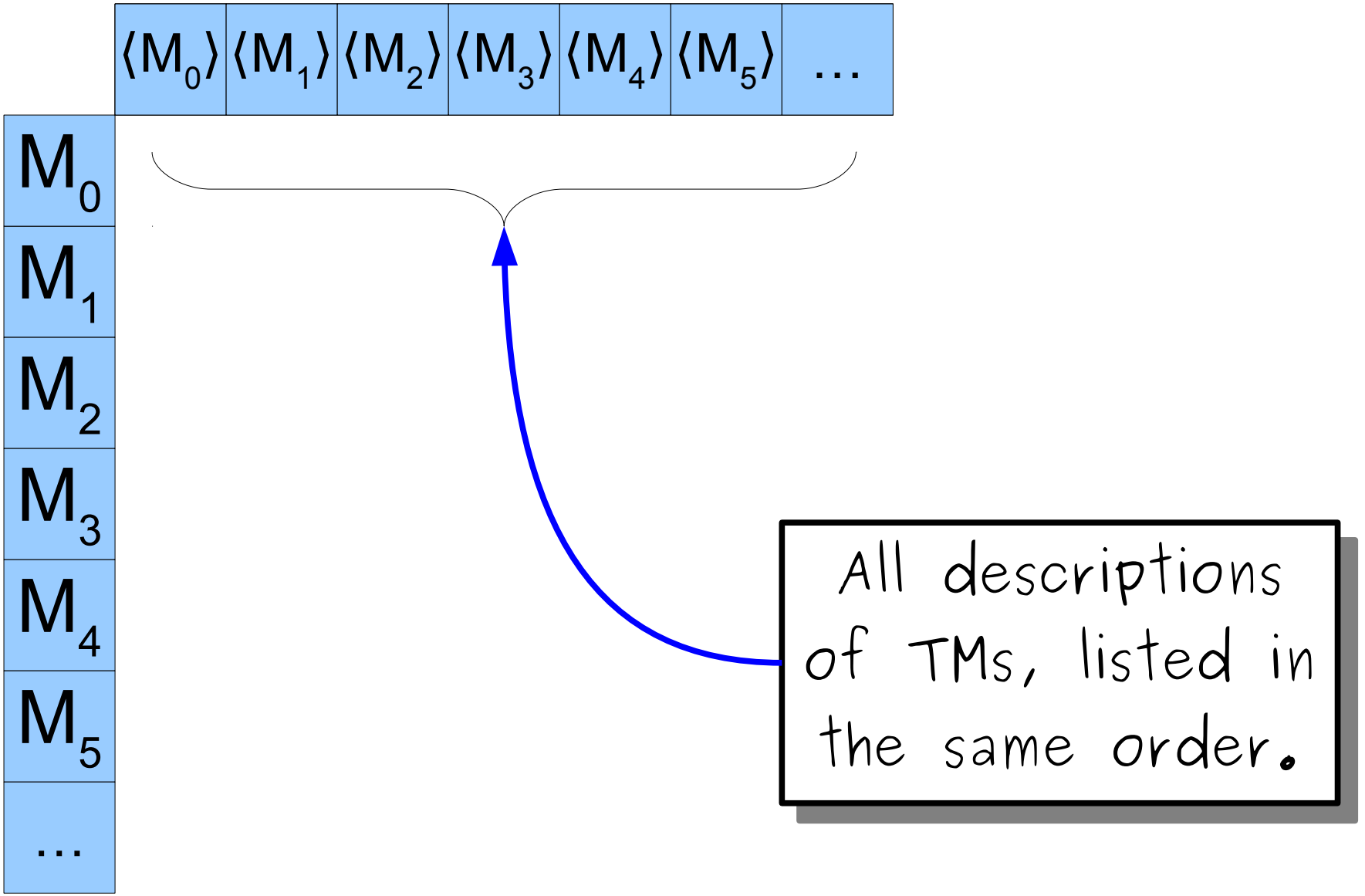
$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

- Some of the strings in this set might be descriptions of TMs.
- What happens if we just focus on the set of strings that are legal TM descriptions?


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 some order.


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


⟨M0⟩ ⟨M1⟩ ⟨M2⟩ ⟨M3⟩ ⟨M4⟩ ⟨M5⟩ …
M0
M1
M2
M3
M4
M5
…
All descriptions of TMs, listed in the same 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							
…							

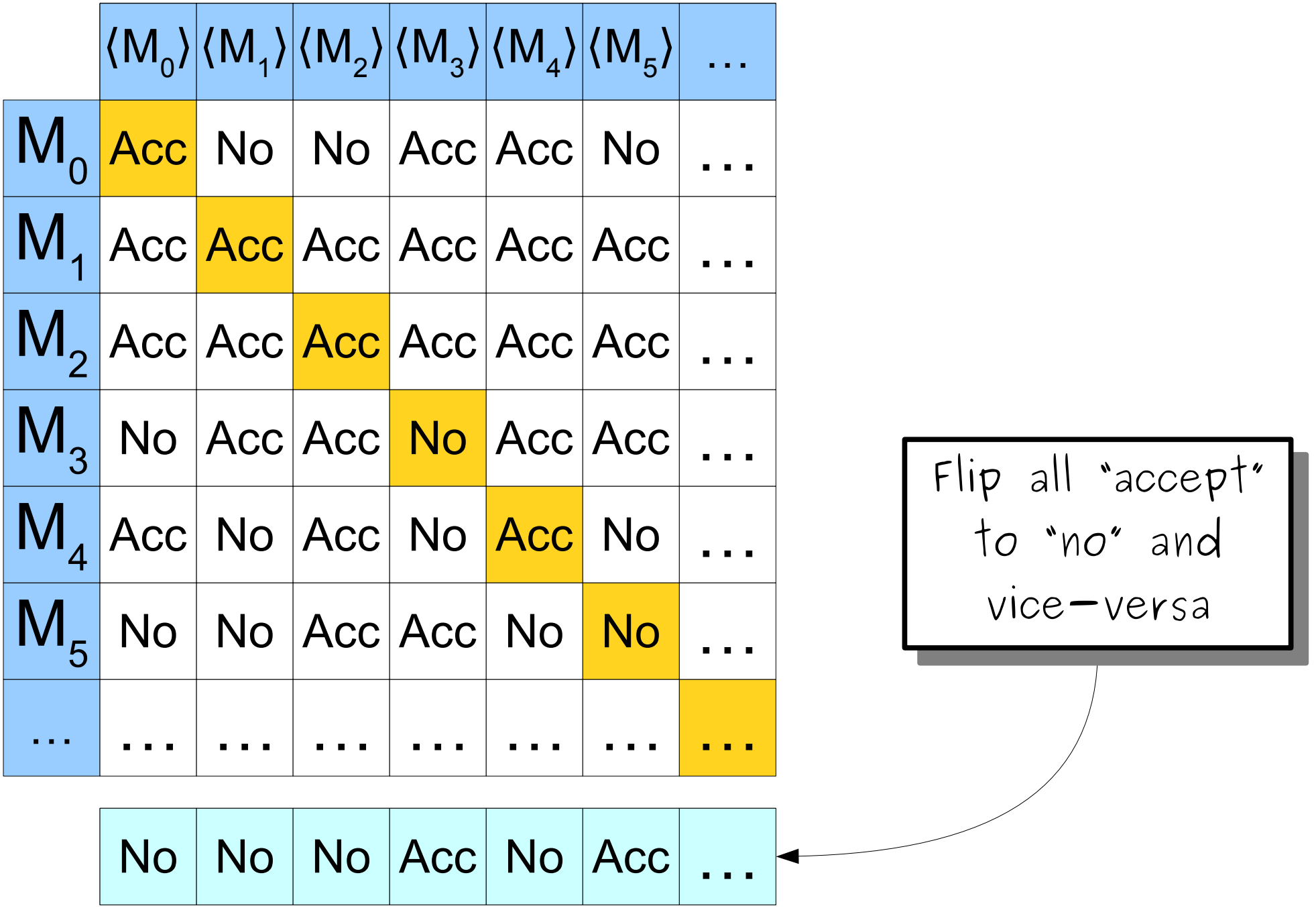
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…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

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

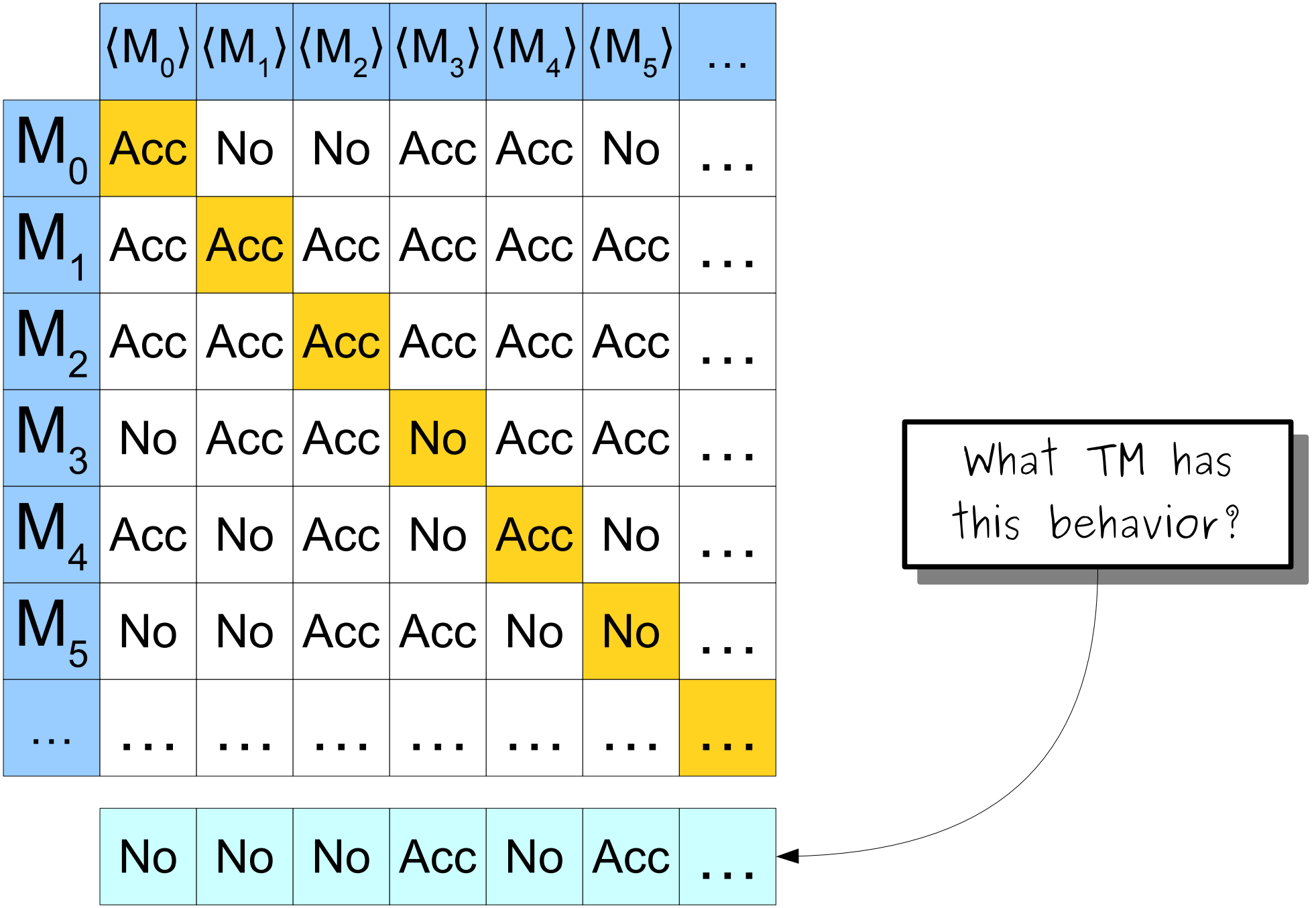
	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…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

Acc	Acc	Acc	No	Acc	No	…


⟨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	…


⟨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 …
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	…
…	…	…	…	…	…	…	…

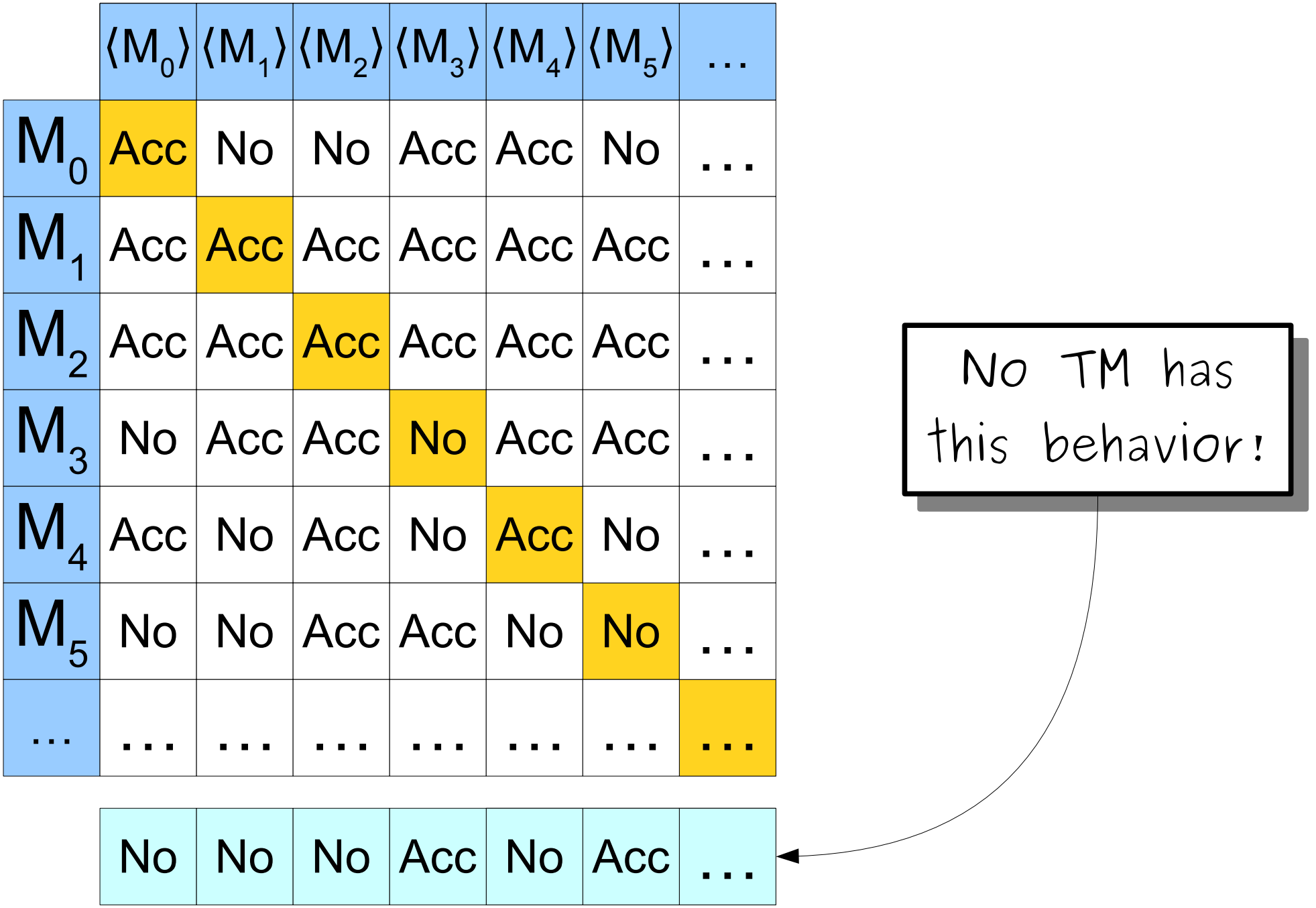
No	No	No	Acc	No	Acc	…

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


⟨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 …
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	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	…
…	…	…	…	…	…	…	…

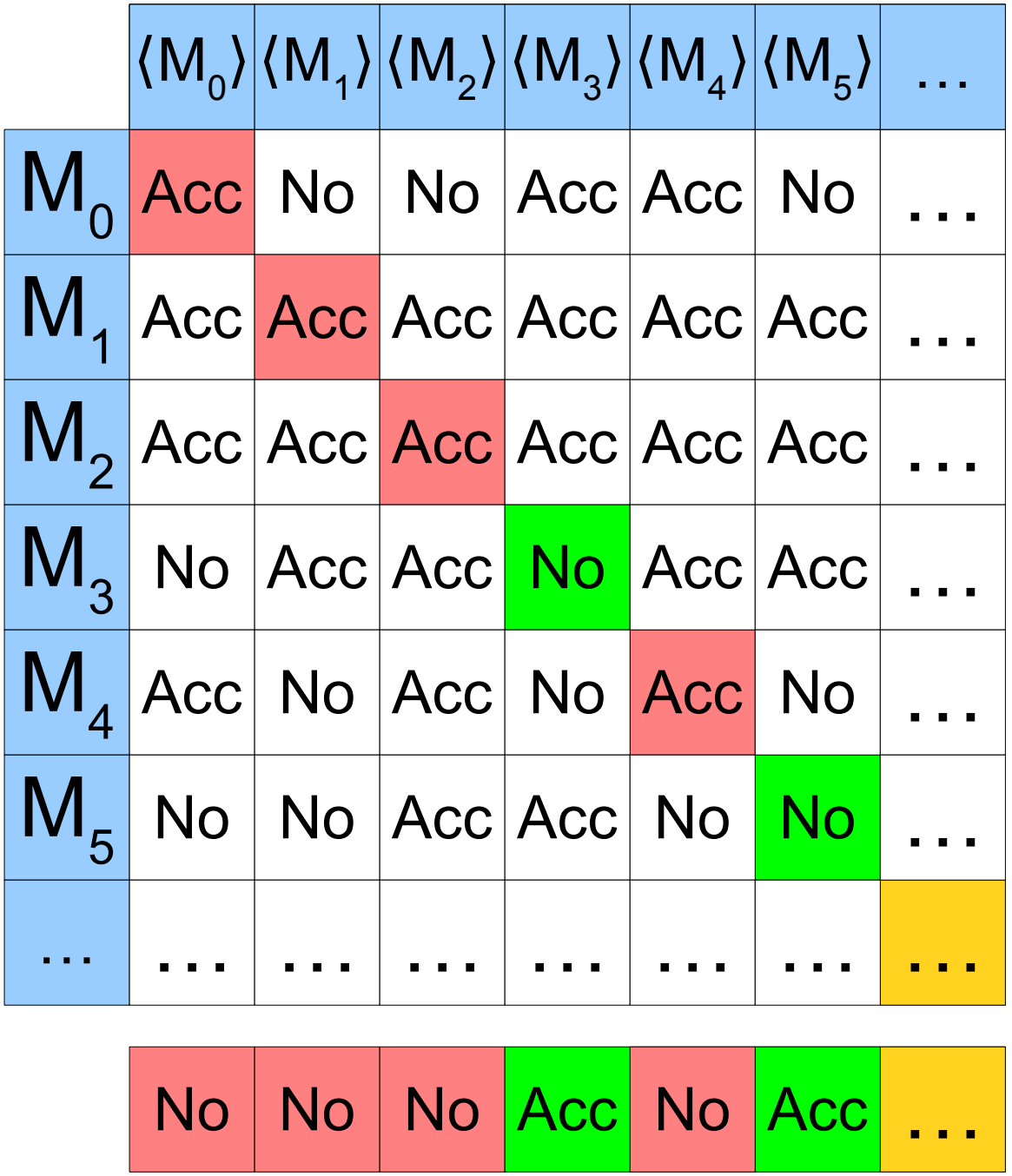
No	No	No	Acc	No	Acc	…

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

“The language of all encodings of TMs that do not accept their own descriptions.”

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

{ $\langle M \rangle$ | M is a TM that does not accept $\langle M \rangle$ }



{ ⟨*M*⟩ | *M* is a TM and ⟨*M*⟩ ∉ $\mathscr{L}$(*M*) }

Diagonalization Revisited

- The ***diagonalization language***, which we denote L_D, is defined as

  $$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$$

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

$$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$$

Theorem: $L_D \notin$ **RE**.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin$ **RE**.

Proof: By contradiction; assume that $L_D \in$ **RE**.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

$$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$$

Theorem: $L_D \notin$ **RE**.

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

Because $\mathscr{L}(R) = L_D$, we know that a string belongs to one set if and only if it belongs to the other.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

$$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

We've replaced the left-hand side of this biconditional with an equivalent statement.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$.

A nice consequence of a universally-quantified statement is that it should work in all cases.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$. If we pick $M = R$, we see that

$$\langle R \rangle \notin \mathscr{L}(R) \text{ iff } \langle R \rangle \in \mathscr{L}(R) \qquad (3)$$

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin$ **RE**.

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$. If we pick $M = R$, we see that

$$\langle R \rangle \notin \mathscr{L}(R) \text{ iff } \langle R \rangle \in \mathscr{L}(R) \qquad (3)$$

This is clearly impossible.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$. If we pick $M = R$, we see that

$$\langle R \rangle \notin \mathscr{L}(R) \text{ iff } \langle R \rangle \in \mathscr{L}(R) \qquad (3)$$

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$. If we pick $M = R$, we see that

$$\langle R \rangle \notin \mathscr{L}(R) \text{ iff } \langle R \rangle \in \mathscr{L}(R) \qquad (3)$$

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus $L_D \notin \mathbf{RE}$.

$L_D = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \mathscr{L}(M) \}$

Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. Then there must be some TM R such that $\mathscr{L}(R) = L_D$.

Since $\mathscr{L}(R) = L_D$, we know that if M is any TM, then

$$\langle M \rangle \in L_D \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (1)$$

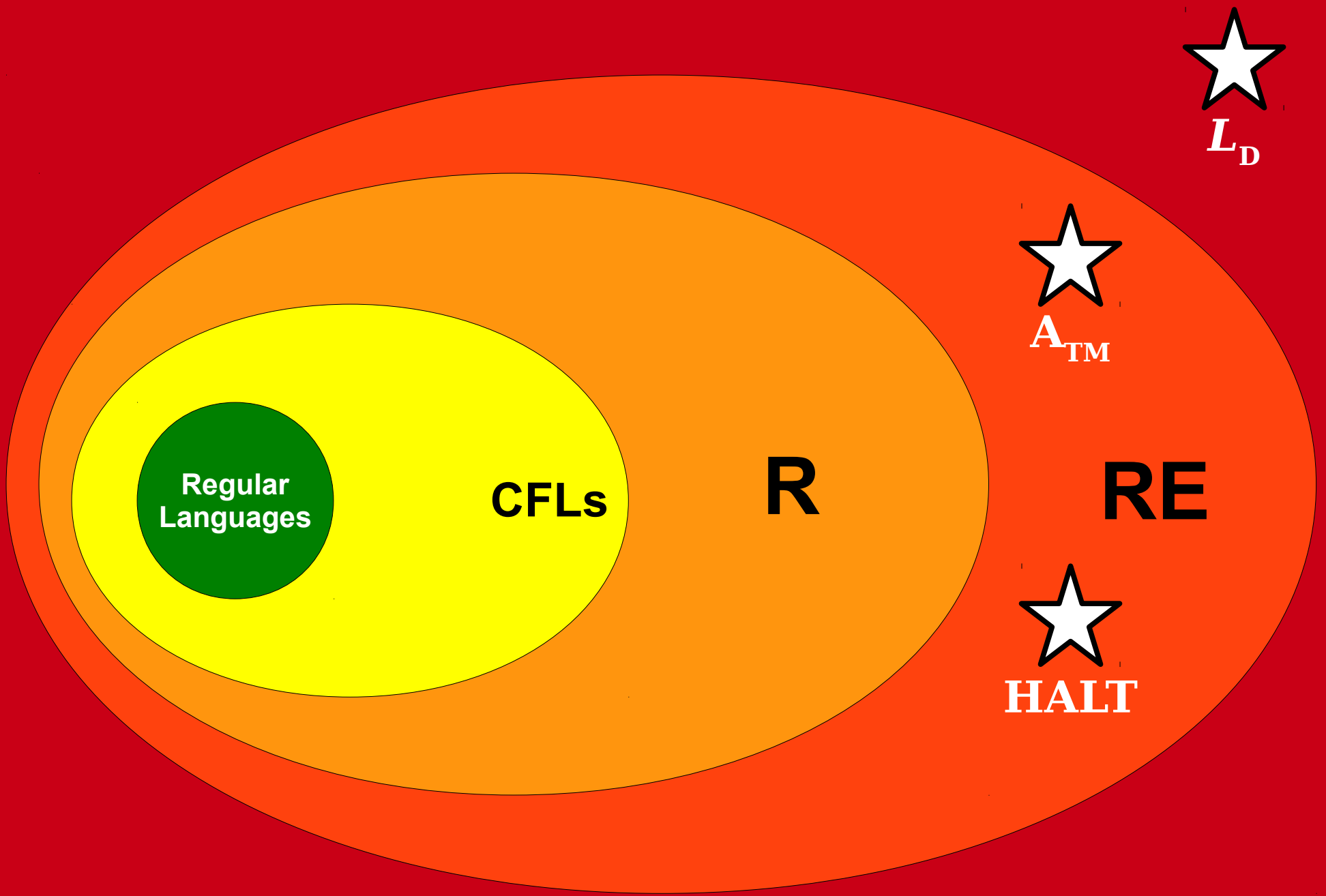
From the definition of L_D, we see that $\langle M \rangle \in L_D$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

$$\langle M \rangle \notin \mathscr{L}(M) \text{ iff } \langle M \rangle \in \mathscr{L}(R) \qquad (2)$$

Statement (2) holds for any TM M, so in particular it should hold when $M = R$. If we pick $M = R$, we see that

$$\langle R \rangle \notin \mathscr{L}(R) \text{ iff } \langle R \rangle \in \mathscr{L}(R) \qquad (3)$$

This is clearly impossible. We have reached a contradiction, so our assumption must have been wrong. Thus $L_D \notin \mathbf{RE}$. ■


L_D
A_{TM}
Regular Languages
CFLs
R
RE
HALT
All Languages

What This Means

- On a deeper philosophical level, the fact that non-**RE** languages exist supports the following claim:

There are statements that are true but not provable.

- Intuitively, given any non-**RE** language, there will be some string in the language that *cannot* be proven to be in the language.
- This result can be formalized as a result called ***Gödel's incompleteness theorem***, one of the most important mathematical results of all time.
- Want to learn more? Take Phil 152 or CS154!

What This Means

- On a more philosophical note, you could interpret the previous result in the following way:

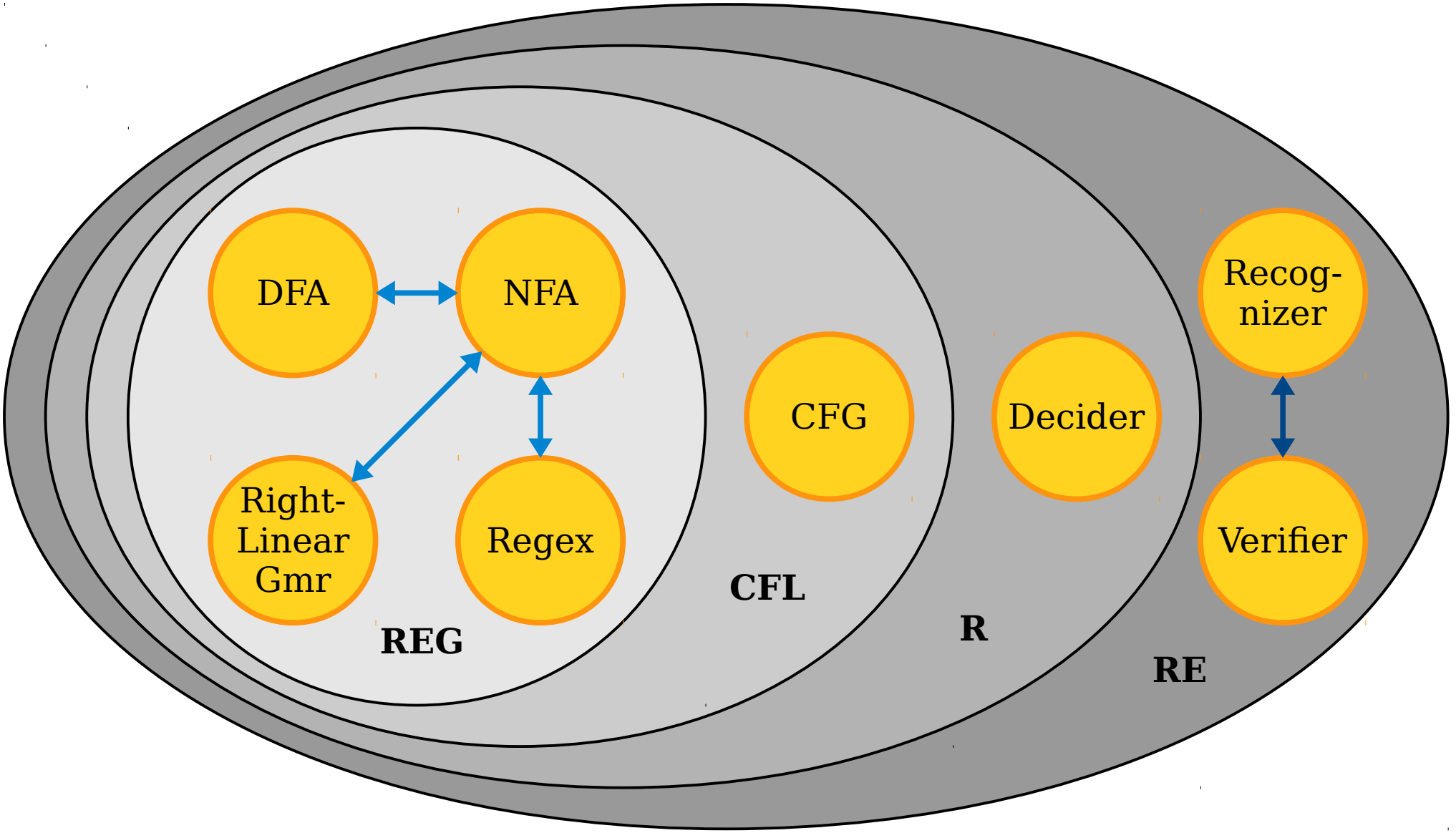
There are inherent limits about what mathematics can teach us.

- There's no automatic way to do math. There are true statements that we can't prove.
- That doesn't mean that mathematics is worthless. It just means that we need to temper our expectations about it.

Where We Stand

- We've just done a crazy, whirlwind tour of computability theory:
  - ***The Church-Turing thesis*** tells us that TMs give us a mechanism for studying computation in the abstract.
  - ***Universal computers*** – computers as we know them – are not just a stroke of luck. The existence of the universal TM ensures that such computers must exist.
  - ***Self-reference*** is an inherent consequence of computational power.
  - ***Undecidable problems*** exist partially as a consequence of the above and indicate that there are statements whose truth can't be determined by computational processes.
  - ***Unrecognizable problems*** are out there and can be discovered via diagonalization. They imply there are limits to mathematical proof.

The Big Picture



Where We've Been

- The class **R** represents problems that can be solved by a computer.
- The class **RE** represents problems where “yes” answers can be verified by a computer.

Where We're Going

- The class **P** represents problems that can be solved *efficiently* by a computer.
- The class **NP** represents problems where “yes” answers can be verified *efficiently* by a computer.

Next Time

- ***Introduction to Complexity Theory***
  - Not all decidable problems are created equal!
- ***The Classes P and NP***
  - Two fundamental and important complexity classes.
- ***The P $\stackrel{?}{=}$ NP Question***
  - A literal million-dollar question!