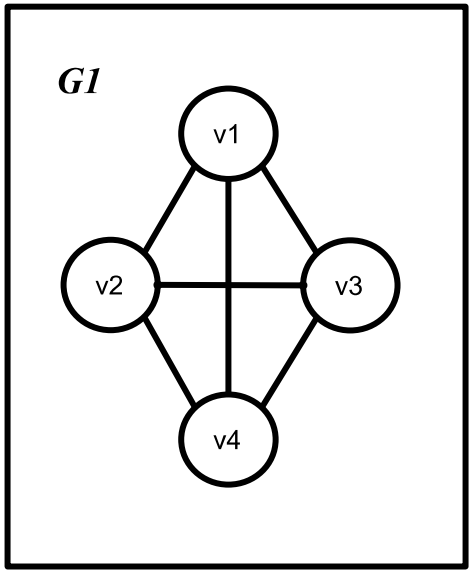
Verifiers and RE

Get ready to answer
some questions in rapid-fire style!
(about 10 seconds per question,
but I won’t close the recorded poll for a
while so don’t stress about that)

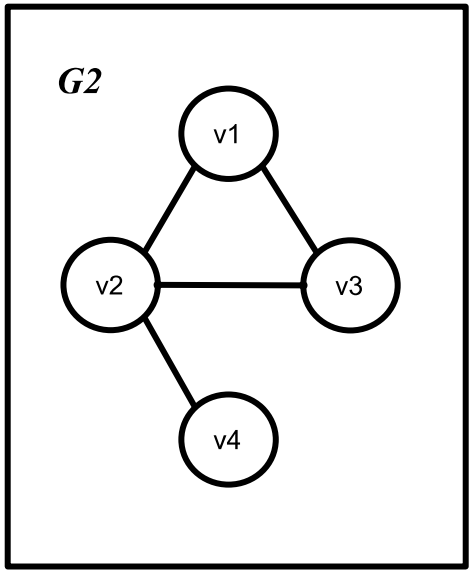
Definition:

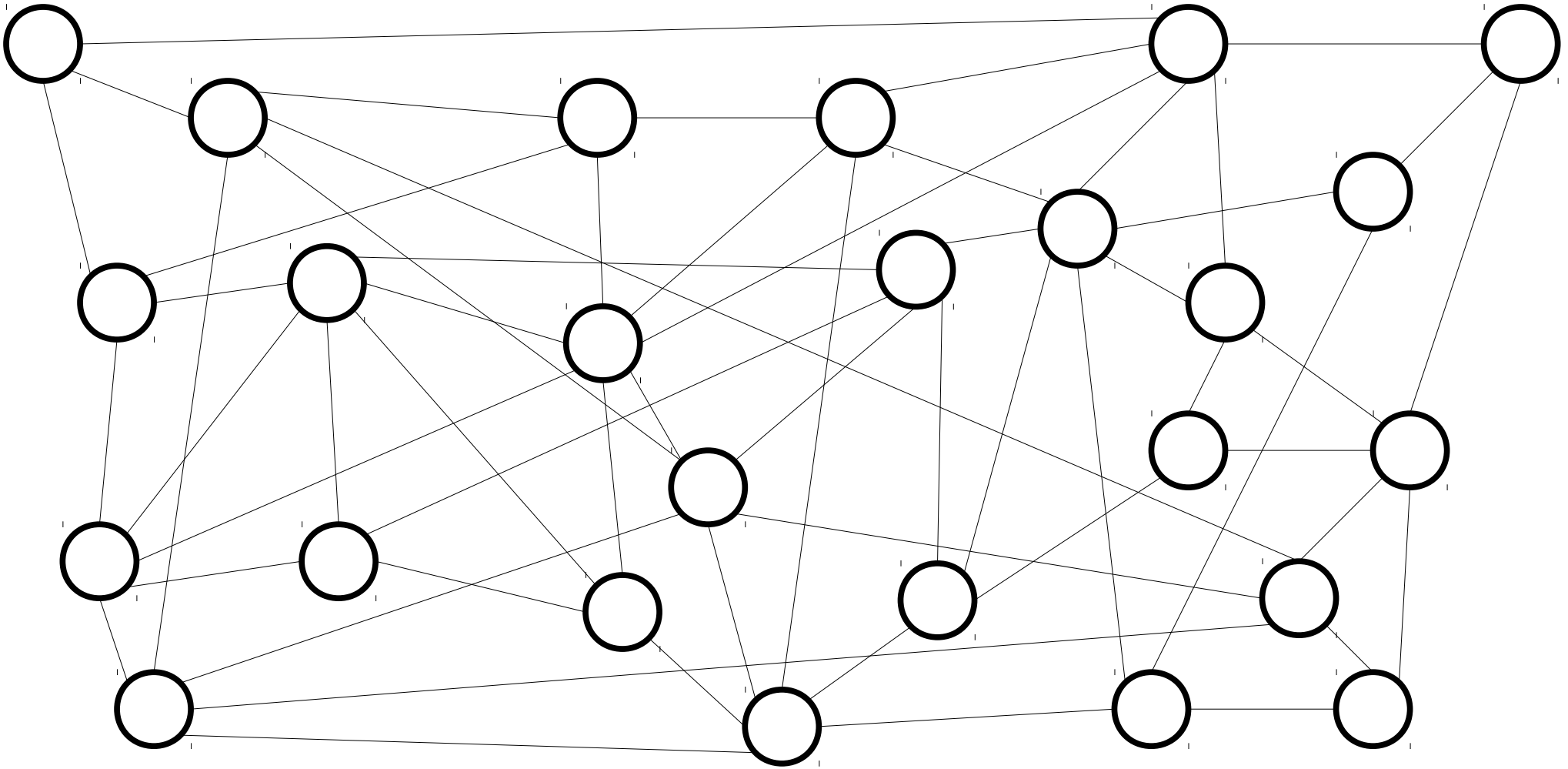
A ***k*-Clique** is a set of k vertices of a graph that are all adjacent to each other (all possible edges between those k vertices are present in the graph).

has a 4-Clique:



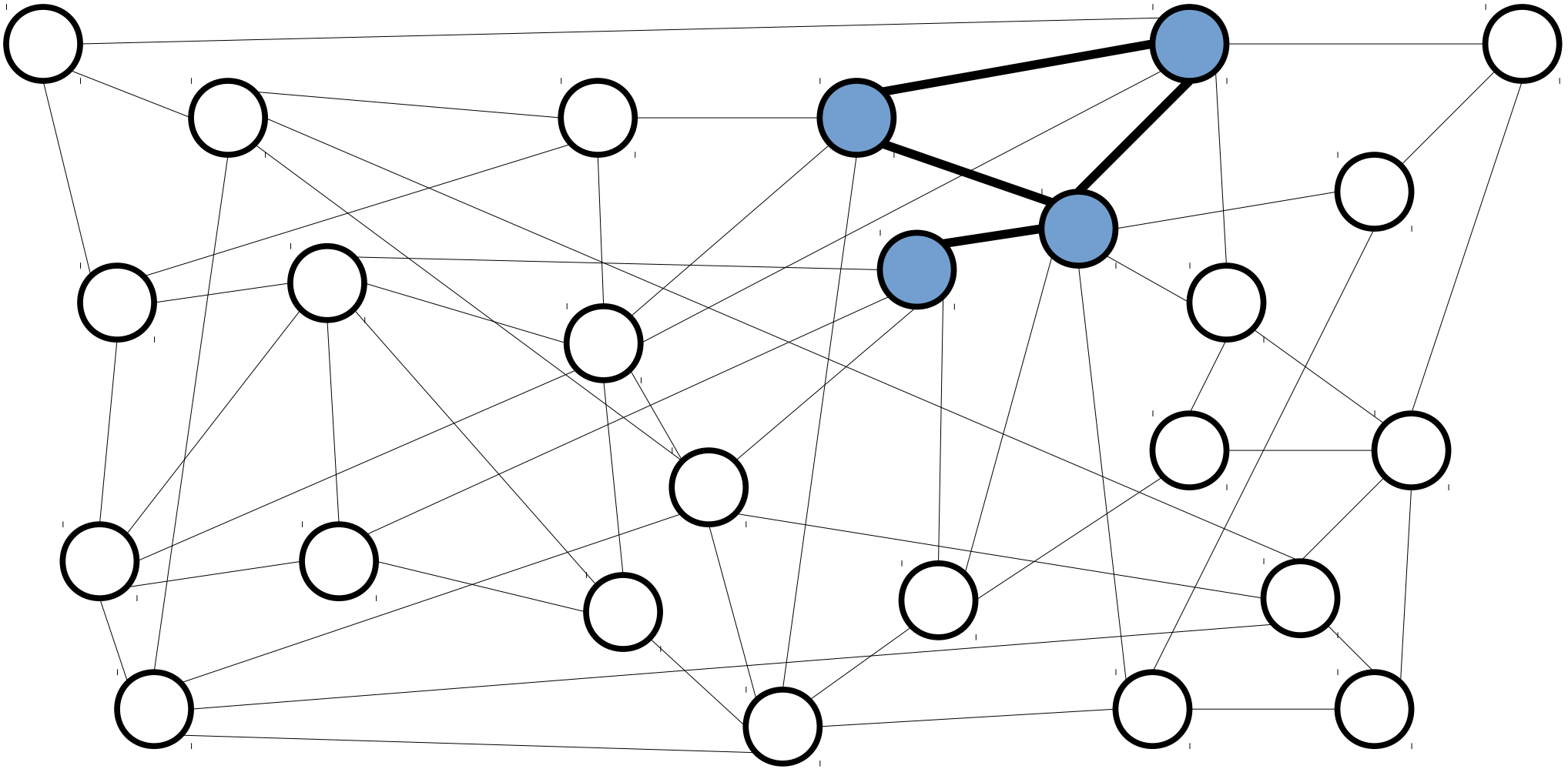
does *not* have a 4-Clique (has 3-Clique though):





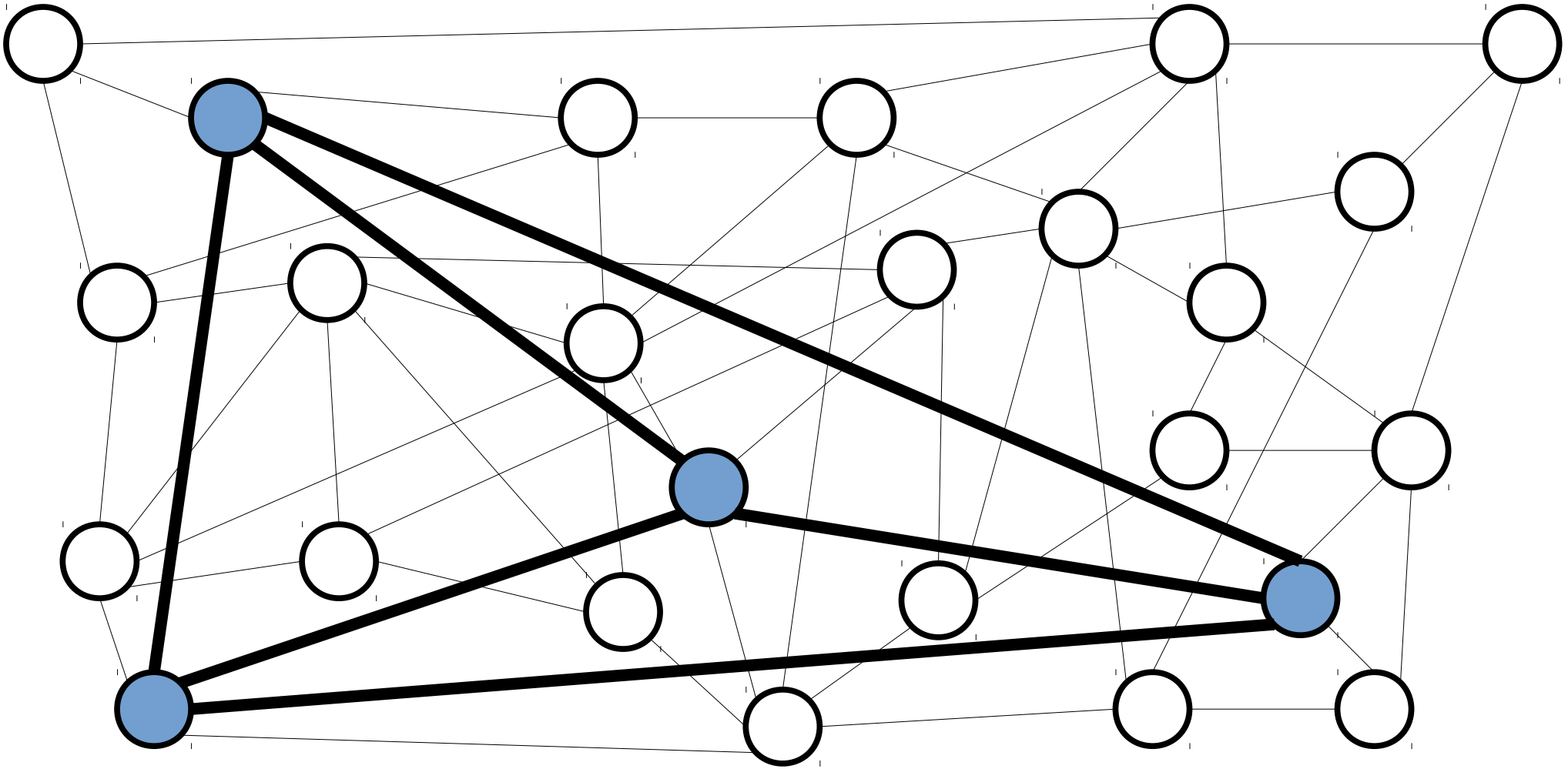
QUICK REACTION: Does this graph contain a 4-clique?

Answer at **PollEv.com/cs103** or text **CS103** to **22333**
once to join, then **Y, N,** or **? (for "I don't know")**.



WITH A HINT: Does this graph contain a 4-clique?

Answer at **PollEv.com/cs103** or text **CS103** to **22333**
once to join, then **Y, N,** or **? (for "I don't know")**.



WITH A *NEW* HINT: Does this graph contain a 4-clique?

Answer at **PollEv.com/cs103** or text **CS103** to **22333**
once to join, then **Y, N,** or **? (for "I don't know")**.

Key Intuition:

A language L is in **RE** if, for any string w, if you are *convinced* that $w \in L$, there is some piece of evidence you could provide to convince someone else.

Discussion Question:

A language L is in **RE** if, for any string w, if you are *convinced* that $w \in L$, there is some piece of evidence you could provide to convince someone else.

What about for a $w \notin L$? What would a piece of evidence for that look like?

More examples of helpful hints vs unhelpful hints

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

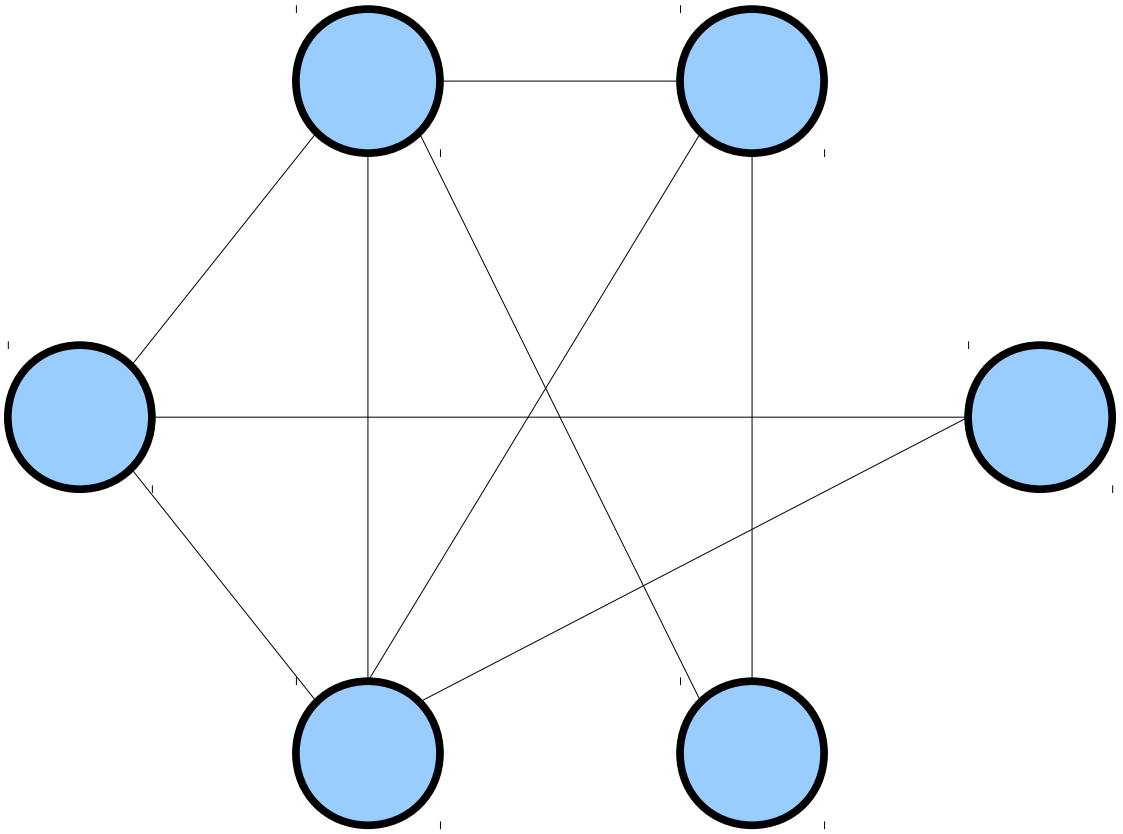
Does this Sudoku puzzle have a solution?

Verification

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

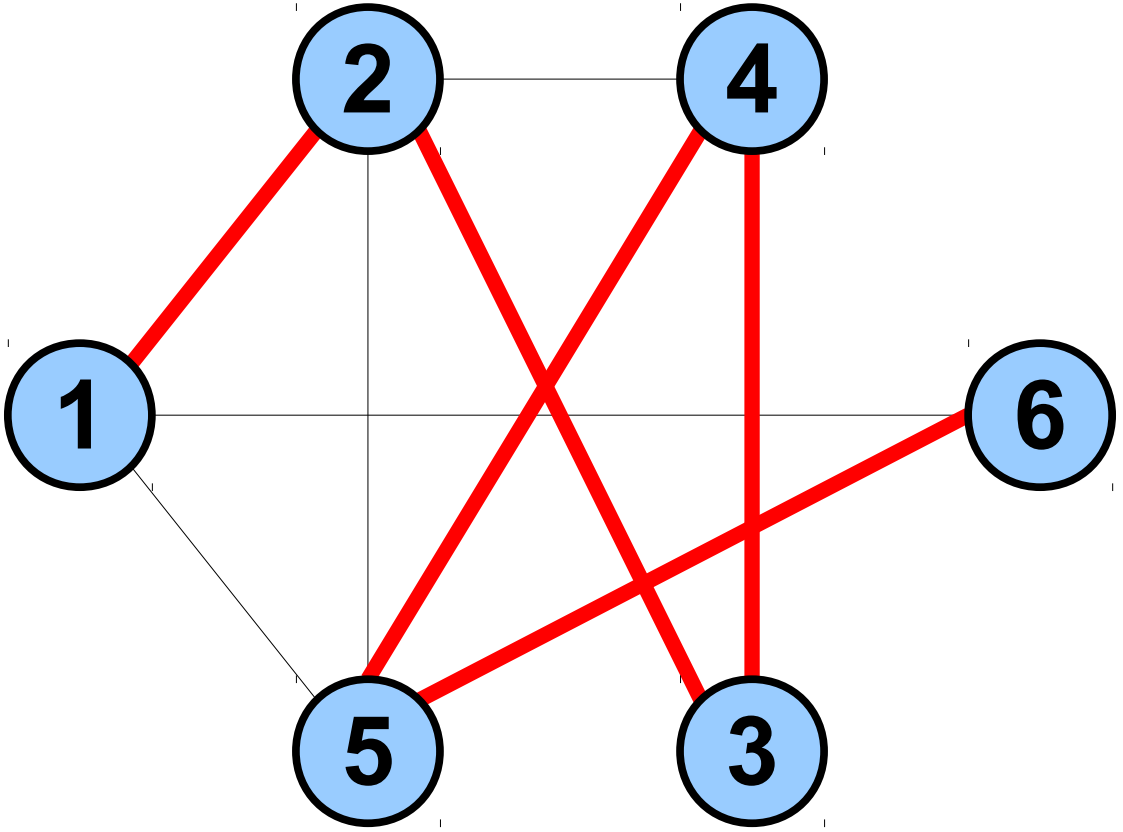
Does this Sudoku puzzle have a solution?

Verification



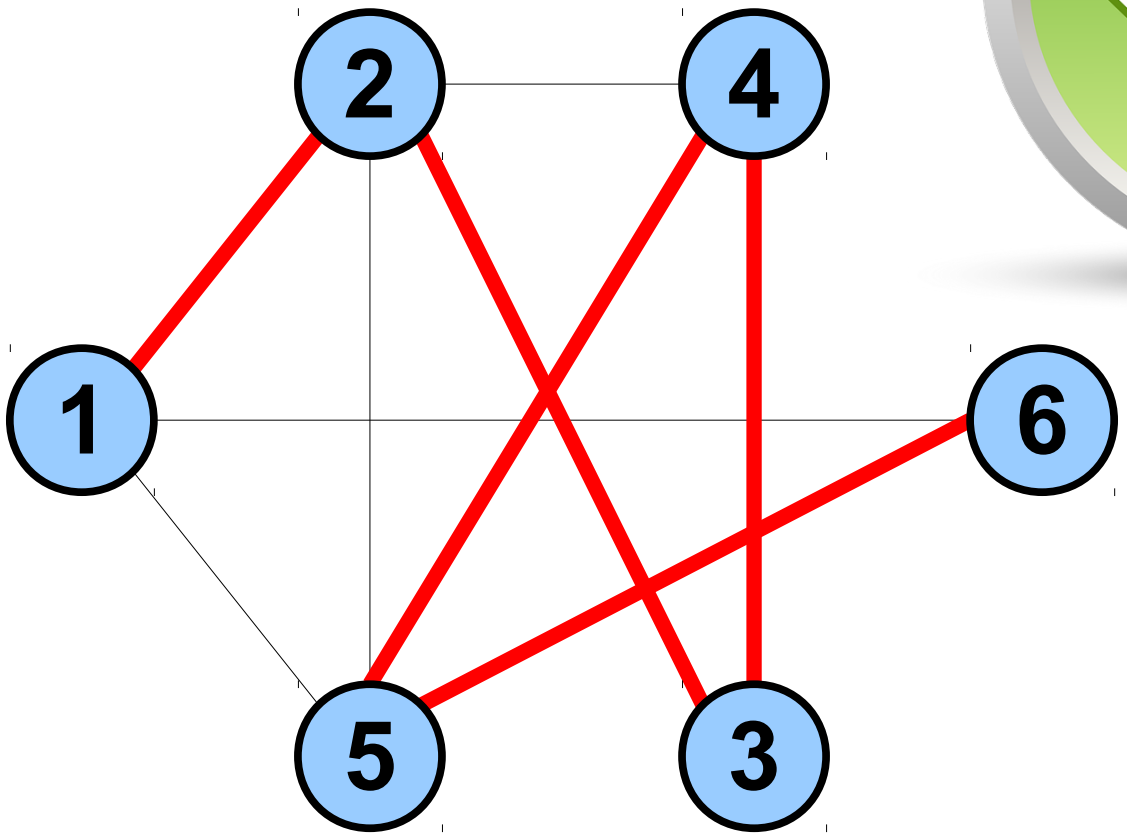
Does this graph have a ***Hamiltonian path*** (a simple path that passes through every node exactly once?)

Verification



Does this graph have a ***Hamiltonian path*** (a simple path that passes through every node exactly once?)

Verification



Does this graph have a ***Hamiltonian path*** (a simple path that passes through every node exactly once?)

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		

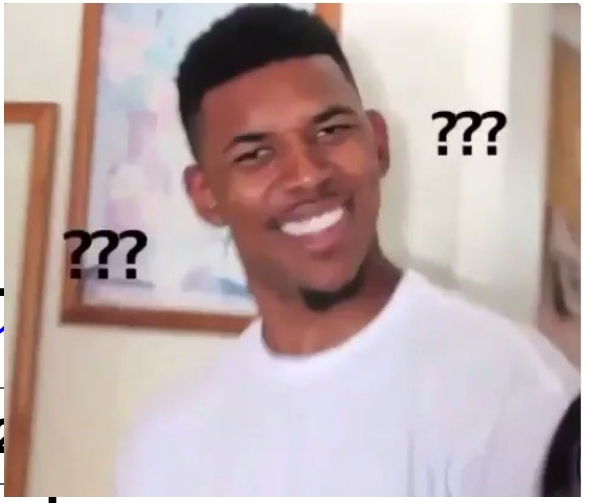
Does this Sudoku puzzle have a solution?

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?

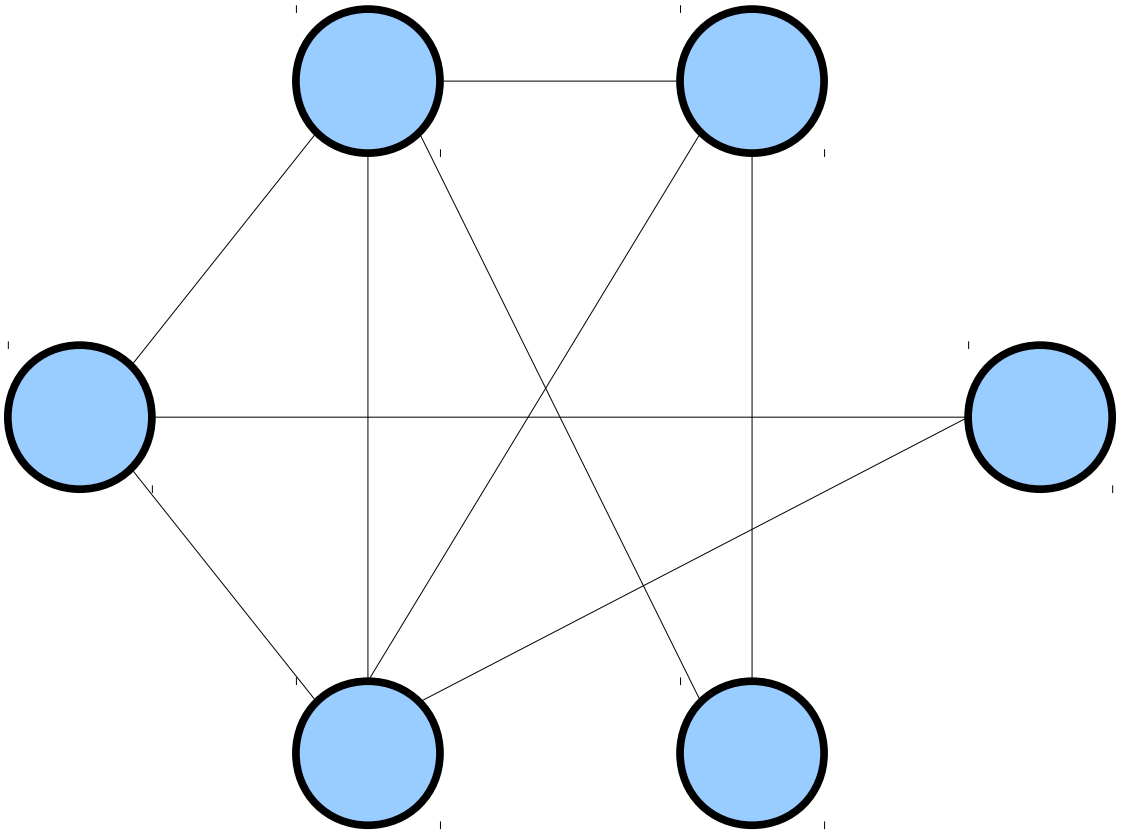
Verification



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

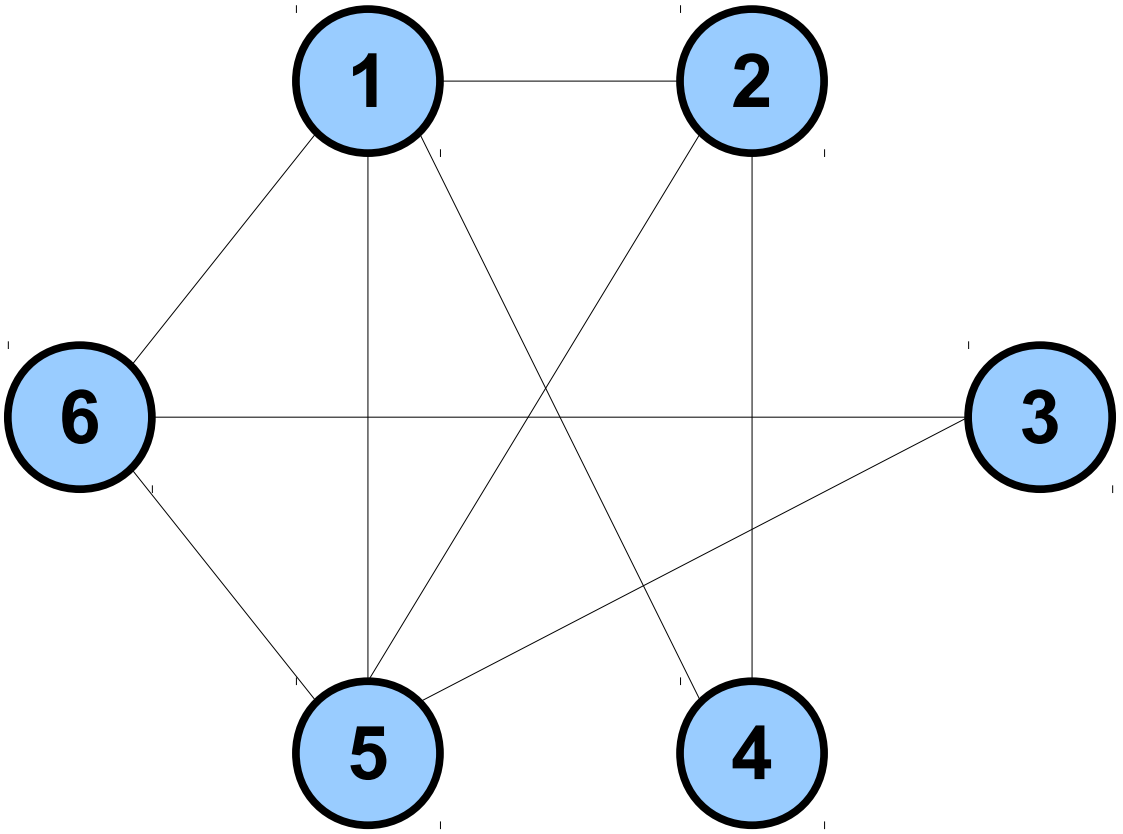
Does this Sudoku puzzle have a solution?

Verification



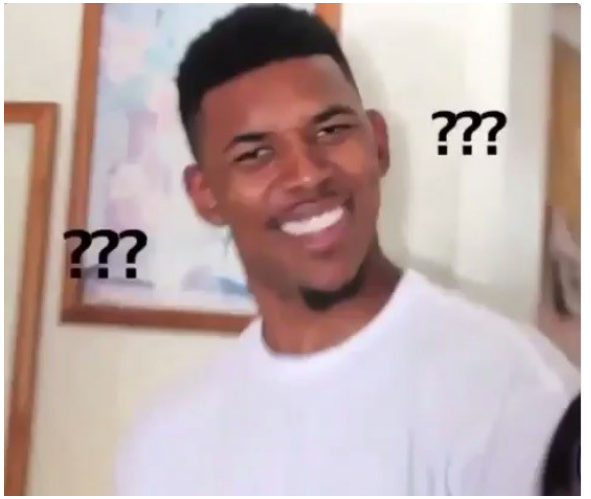
Does this graph have a ***Hamiltonian path*** (a simple path that passes through every node exactly once?)

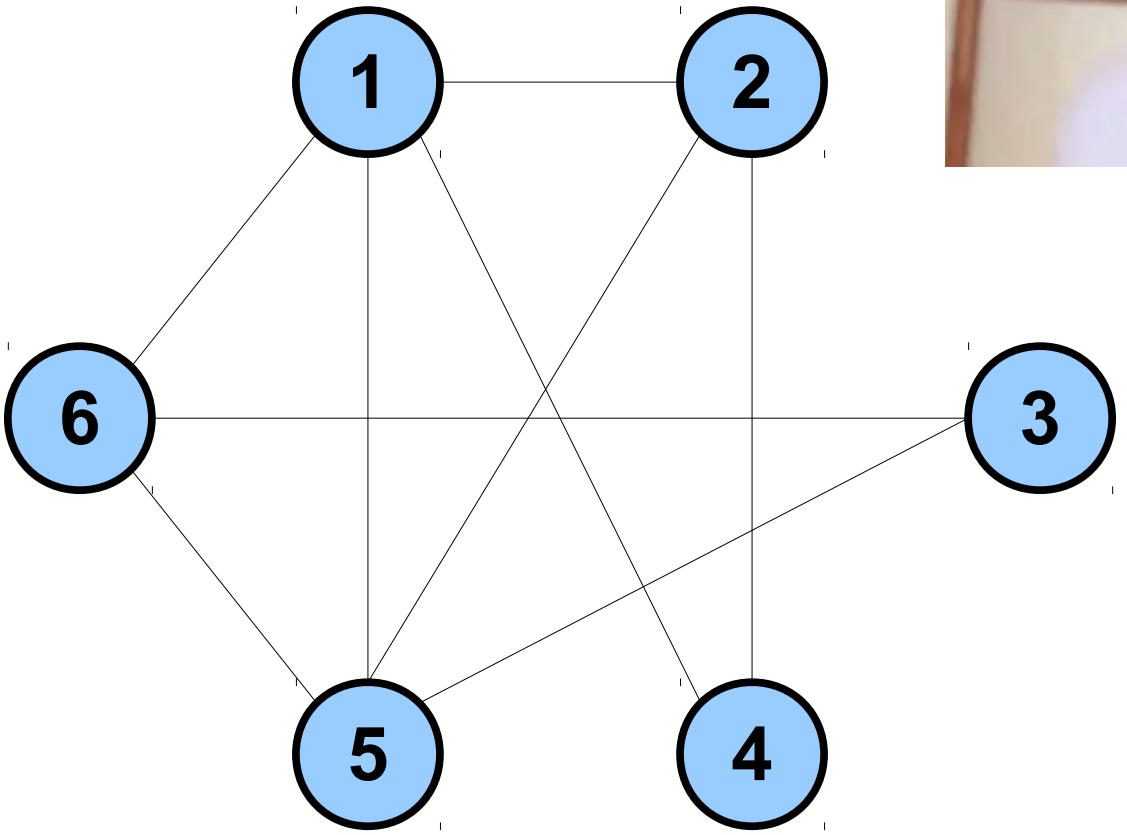
Verification



Does this graph have a ***Hamiltonian path*** (a simple path that passes through every node exactly once?)

Verification





Does this graph have a ***Hamiltonian path*** (a simple path that passes through every node exactly once?)

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,” because the answer is no!
  - Or maybe there is some evidence, but just not the evidence we were given.
- Let's formalize this idea.

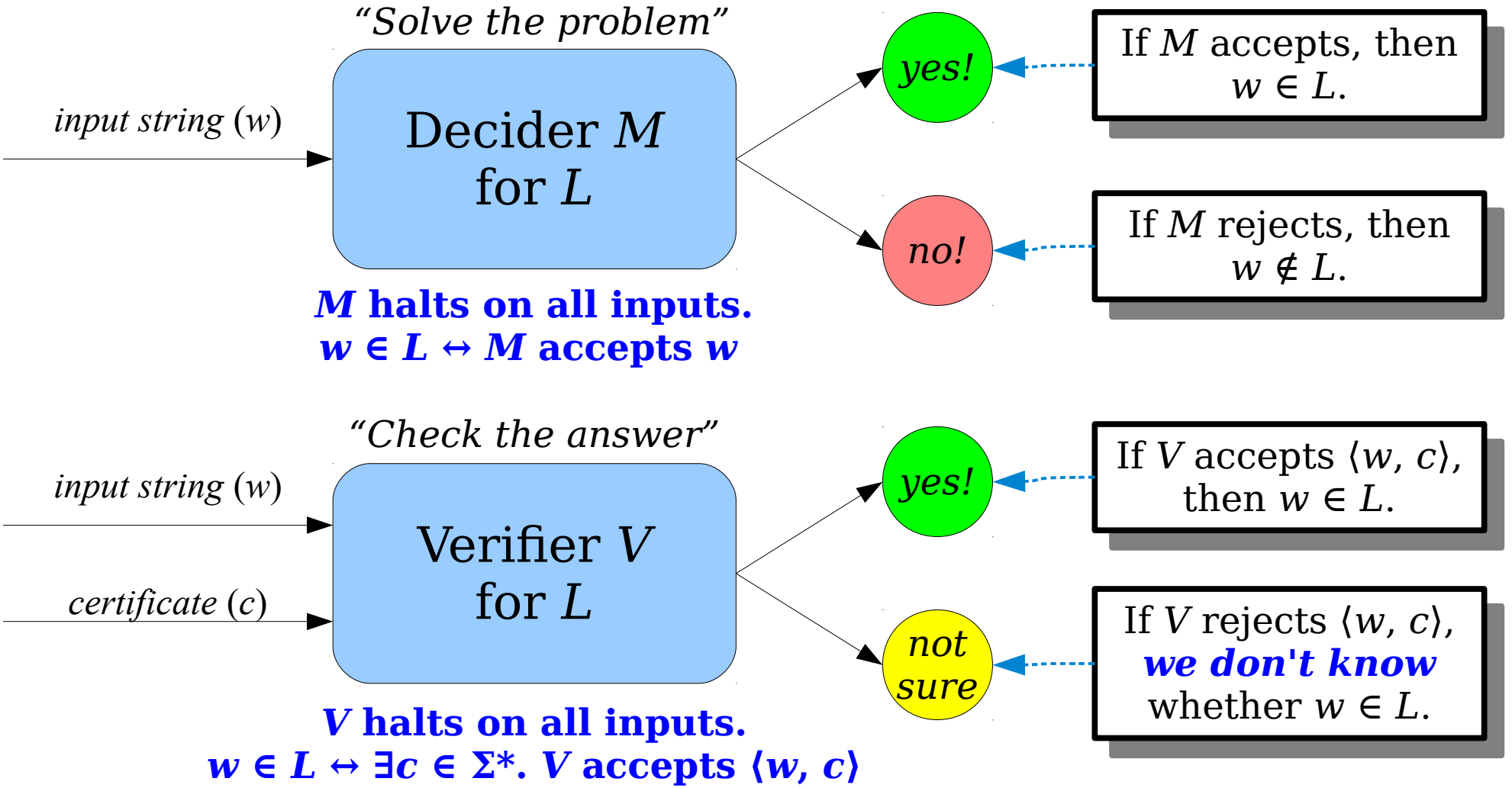
Verifiers

- 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:

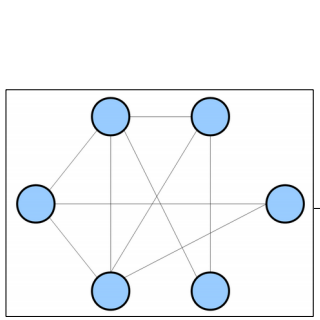
$$w \in L \iff \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle$$

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

Deciders and Verifiers



Deciders and Verifiers



(w)

"Solve the problem"

Decider M for L

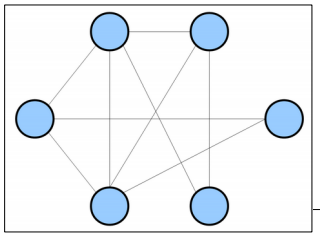
yes!

no!

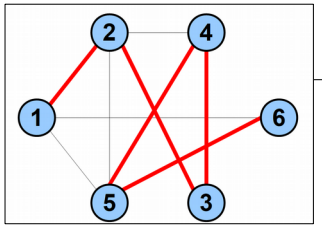
If M accepts, then $w \in L$.

If M rejects, then $w \notin L$.

M halts on all inputs.
$w \in L \leftrightarrow M$ accepts w



(w)



(c)

"Check the answer"

Verifier V for L

yes!

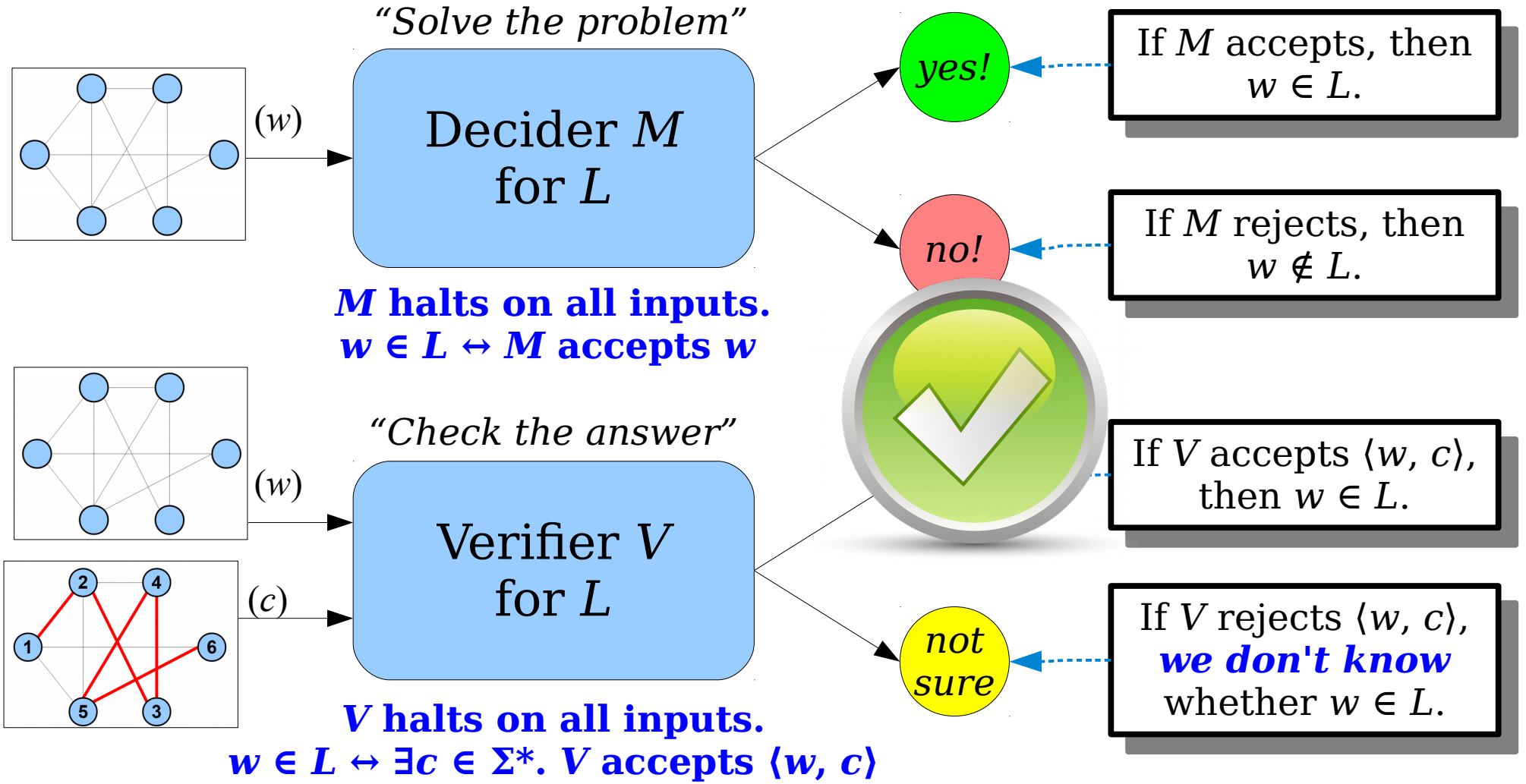
not sure

If V accepts $\langle w, c \rangle$, then $w \in L$.

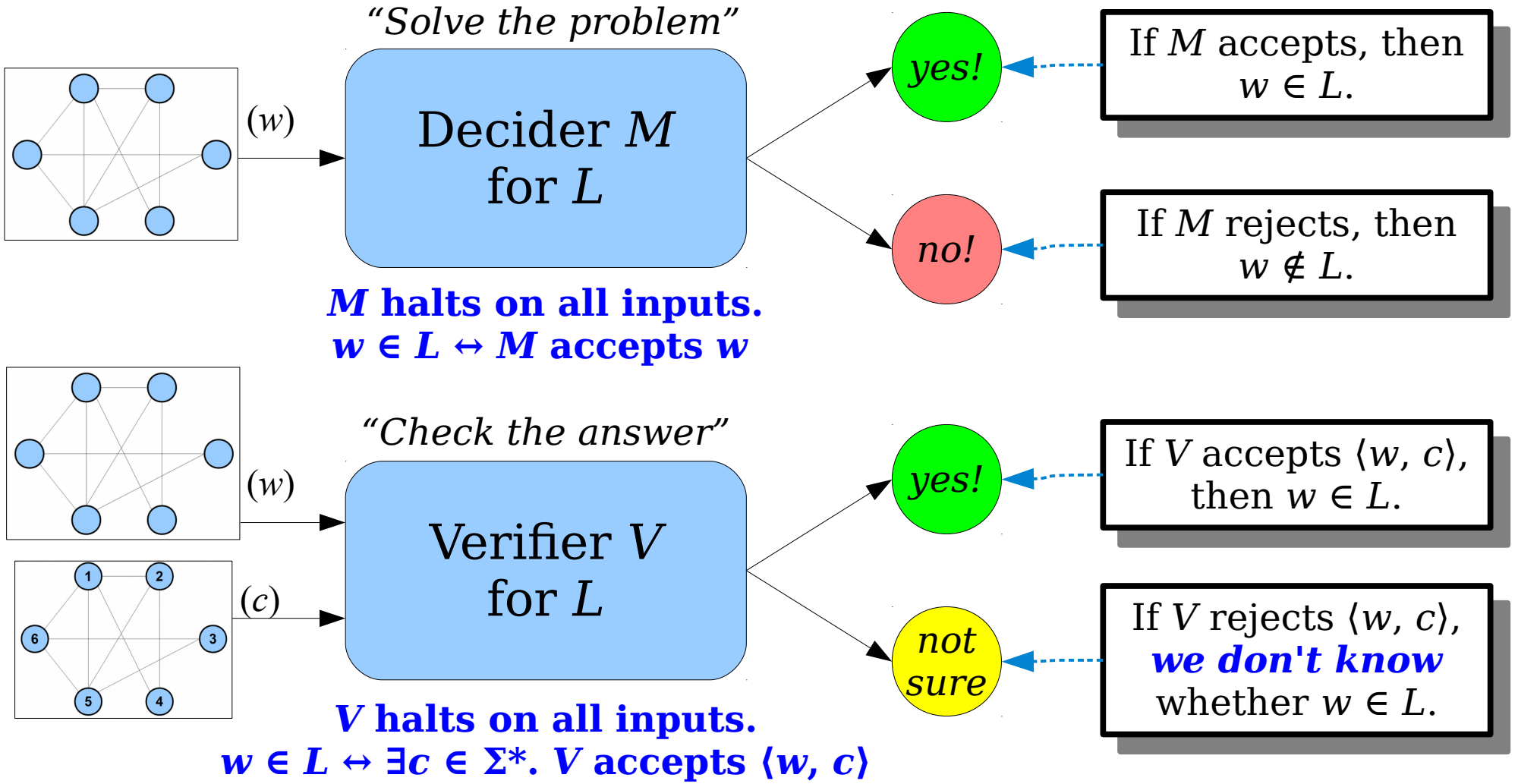
If V rejects $\langle w, c \rangle$, ***we don't know*** whether $w \in L$.

V halts on all inputs.
$w \in L \leftrightarrow \exists c \in \Sigma^*$. V accepts $\langle w, c \rangle$

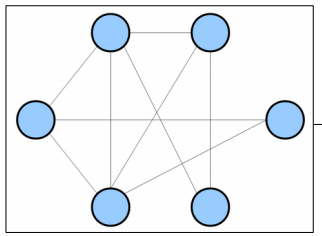
Deciders and Verifiers



Deciders and Verifiers



Deciders and Verifiers



(w)

"Solve the problem"

Decider M for L

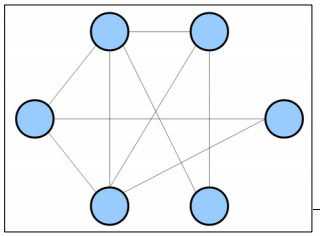
yes!

If M accepts, then $w \in L$.

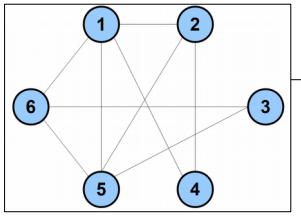
no!

If M rejects, then $w \notin L$.

M halts on all inputs.
$w \in L \leftrightarrow M$ accepts w



(w)



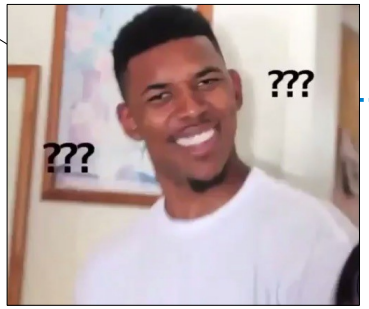
(c)

"Check the answer"

Verifier V for L

yes!

If V accepts $\langle w, c \rangle$, then $w \in L$.



If V rejects $\langle w, c \rangle$, ***we don't know*** whether $w \in L$.

V halts on all inputs.
$w \in L \leftrightarrow \exists c \in \Sigma^*$. V accepts $\langle w,$

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:

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

- More notes about V:
  - Notice that c is existentially quantified.
  - Notice V is required to halt *always* (like a decider).

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:

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

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

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:

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

- A note about c:
  - Figuring out what would make a good certificate (should it be a number of steps to take, an equation-solving variable assignment, a set of graph nodes, an array of numbers to fill in a whole Sudoku board?) is custom work to do for each different language L.

Some Verifiers

- Let L be the following language:

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

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

Some Verifiers

Does this always halt?

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

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

Answer at **PollEv.com/cs103** or text **CS103** to **22333** once to join, then **Y or N.**

Some Verifiers

For one given $\langle n \rangle \in L$ (say 11), how many different values of c will work to cause the verifier to accept?

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

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

Answer at **PollEv.com/cs103** or text **CS103** to **22333** once to join, then **a number.**

How many of these statements are true of $\mathscr{L}(V)$?

- $\mathscr{L}(V) = L$
- $\mathscr{L}(V) \subseteq L$
- $L \subseteq \mathscr{L}(V)$

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

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

Answer at **PollEv.com/cs103** or text **CS103** to **22333** once to join, then **a number.**

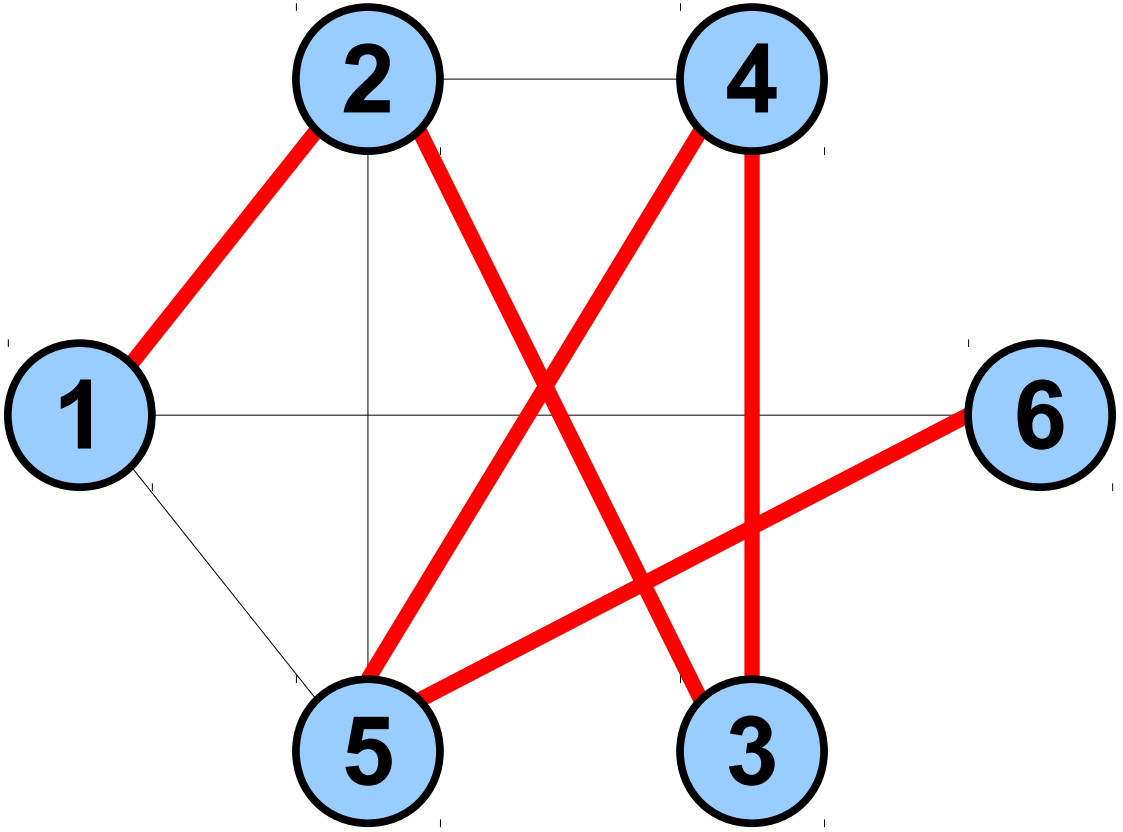
Some Verifiers

- Let L be the following language:

$$L = \{ \langle G \rangle \mid G \text{ is a graph and } G \text{ has a Hamiltonian path} \}$$

- (A Hamiltonian path is a simple path that visits every node in the graph.)
- Let's see how to build a verifier for L.

Verification



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

Verifier Example: Hamiltonian Path

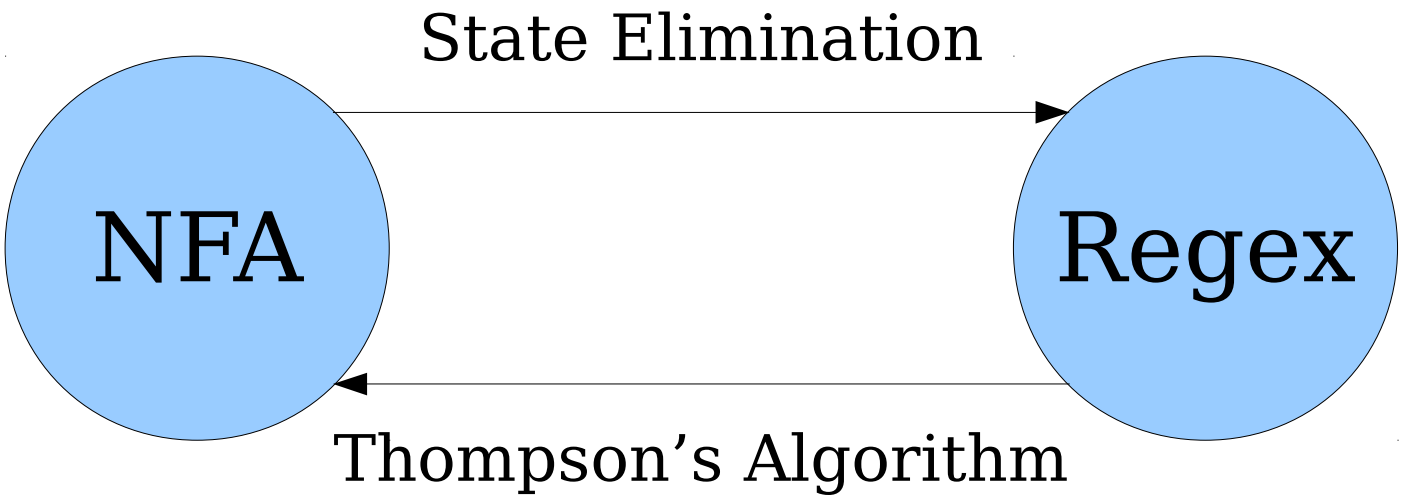
- Let L be the following language:

$$L = \{ \langle G \rangle \mid G \text{ is a graph with a Hamiltonian path } \}$$

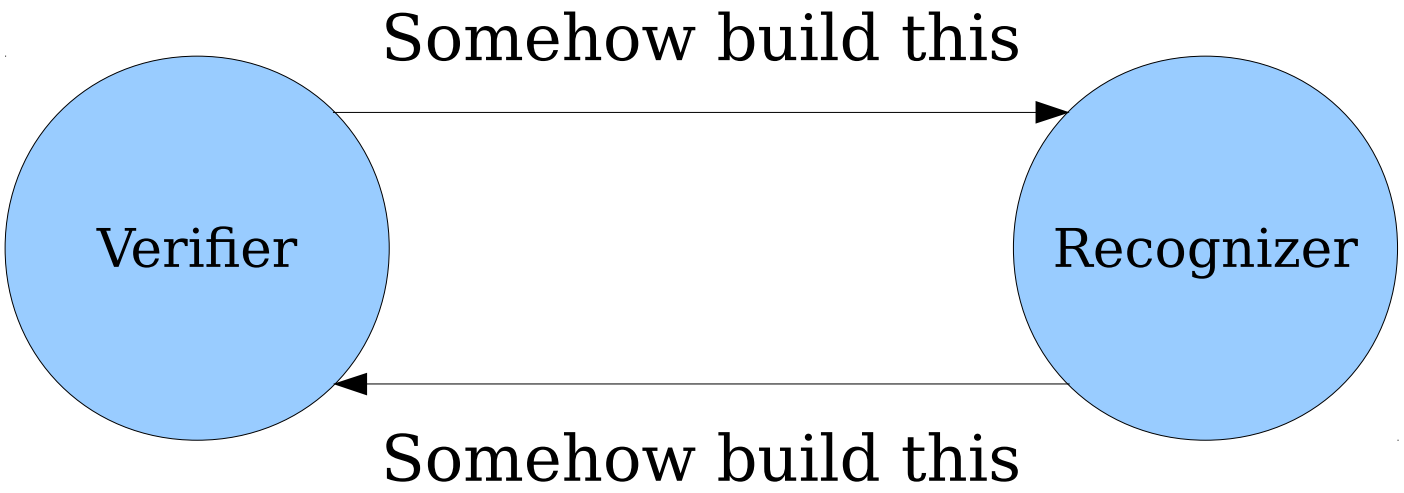
```
bool checkHamiltonian(Graph G, vector<Node> c) {
   if (c.size() != G.numNodes()) return false;
   if (containsDuplicate(c)) return false;

   for (size_t i = 0; i < c.size() - 1; i++) {
       if (!G.hasEdge(c[i], c[i+1])) return false;
   }
   return true;
}
```

- Do you see why $\langle G \rangle \in L$ iff there is a c where `checkHamiltonian(G, c)` returns true?
- Do you see why `checkHamiltonian` always halts?

Where We've Been



Where We're Going Today



Verifier for A_{TM}?

- Consider A_{TM}:

  $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$.

- This is a ***canonical*** example of an undecidable language. There's no way, in general, to tell whether a TM M will accept a string w.
- Although this language is undecidable, it's an **RE** language, and it's possible to build a verifier for it!

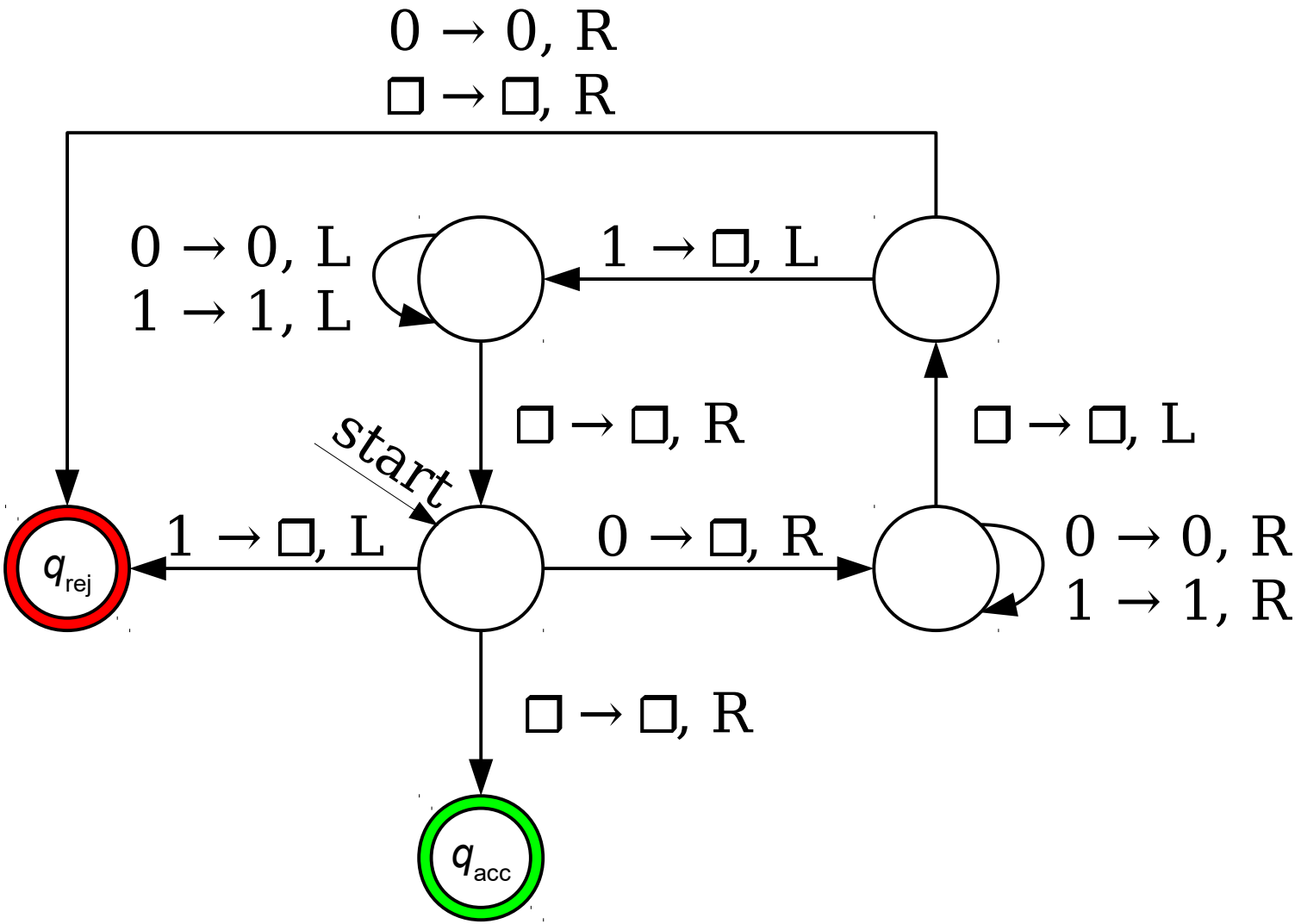
What would make a good certificate for a verifier for A_{TM}?

- Consider A_{TM}:

  $$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}.$$

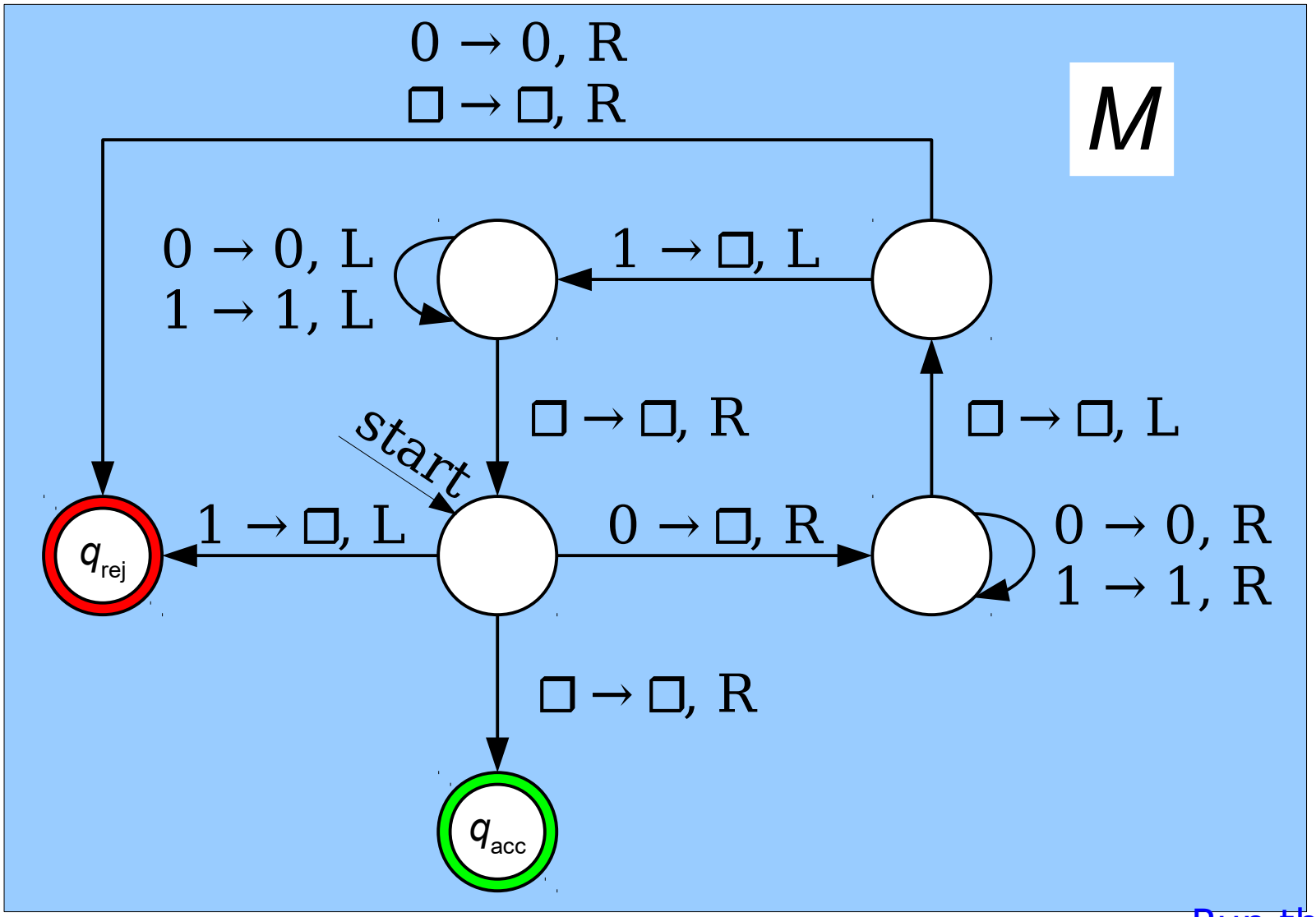
- This is a ***canonical*** example of an undecidable language. There's no way, in general, to tell whether a TM M will accept a string w.
- Although this language is undecidable, it's an **RE** language, and it's possible to build a verifier for it!

Answer at **PollEv.com/cs103** or text **CS103** to **22333** once to join, then **an idea**



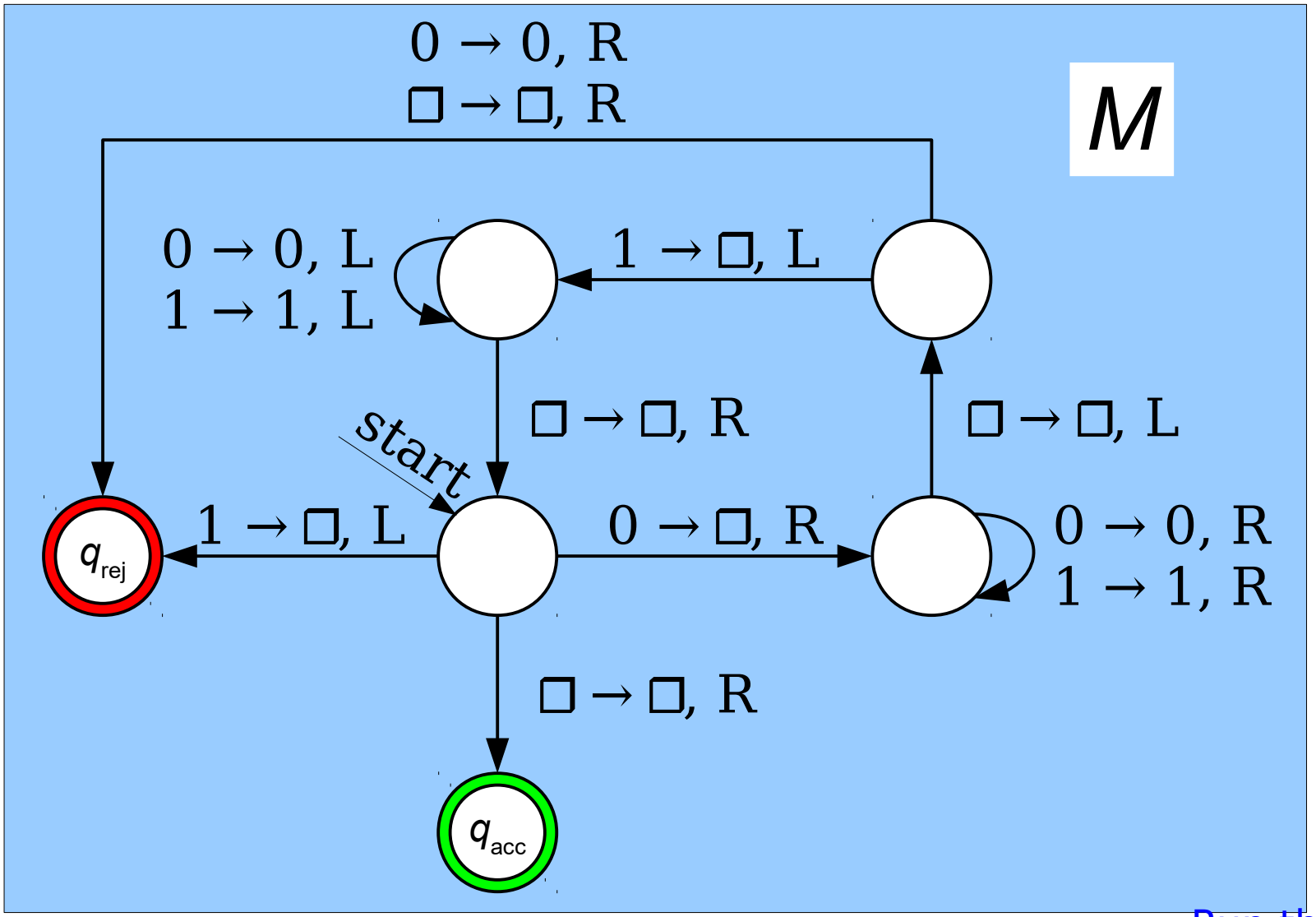
Run this TM for fifteen steps.

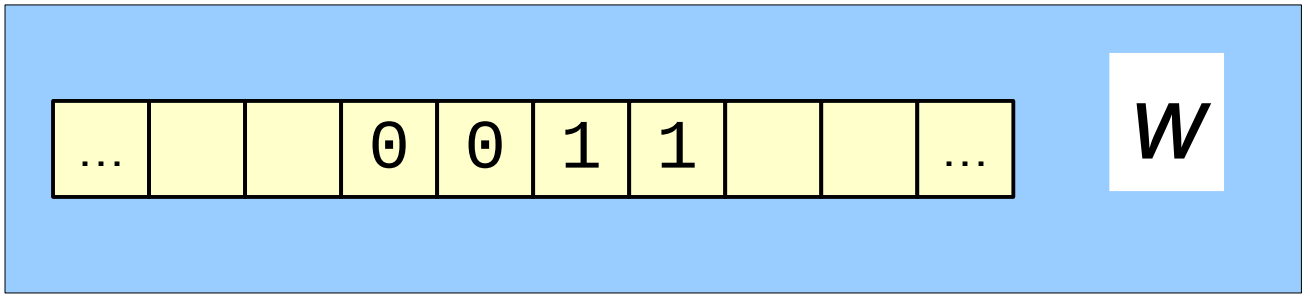
…			0	0	1	1			…



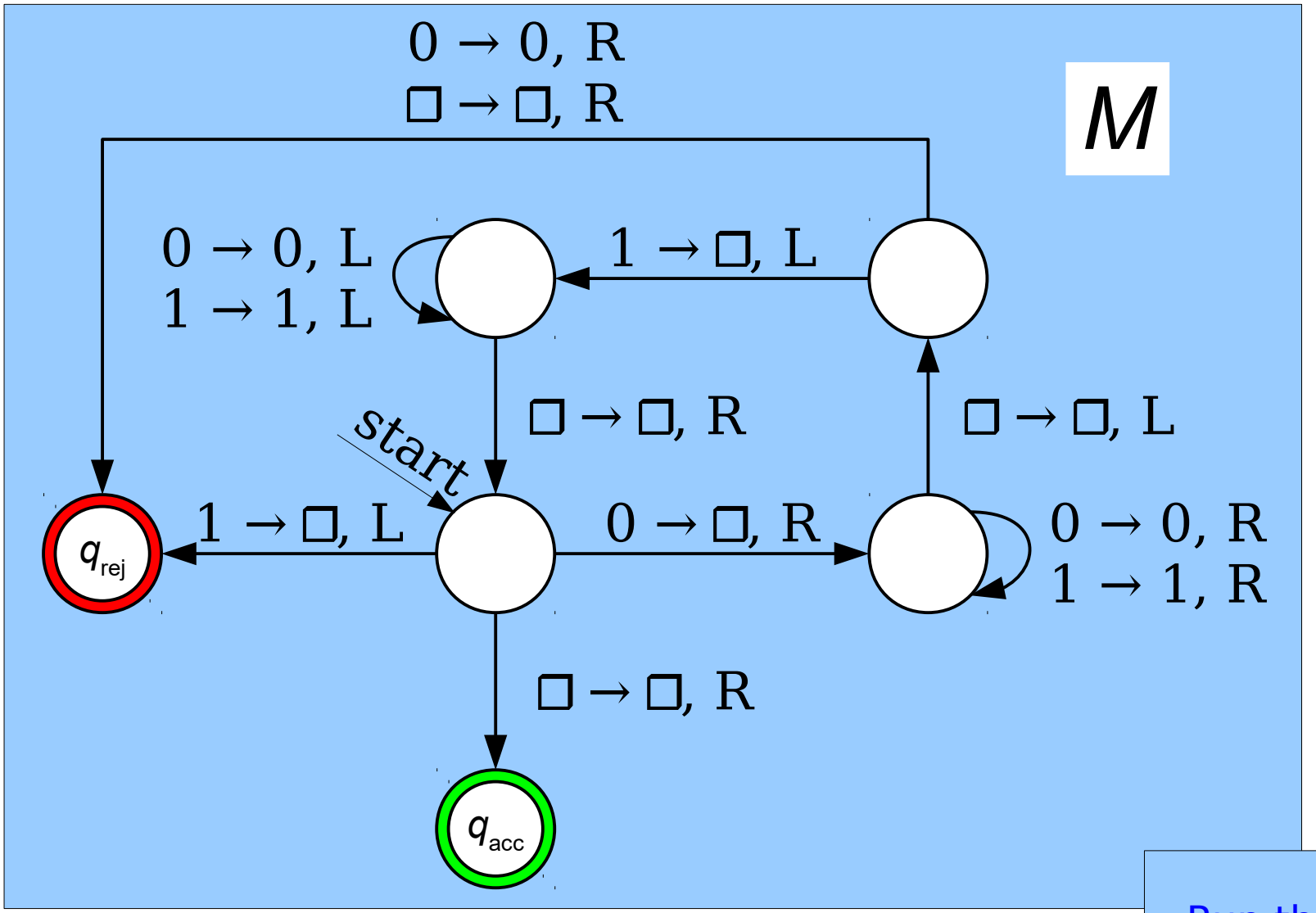
Run this TM for fifteen steps.

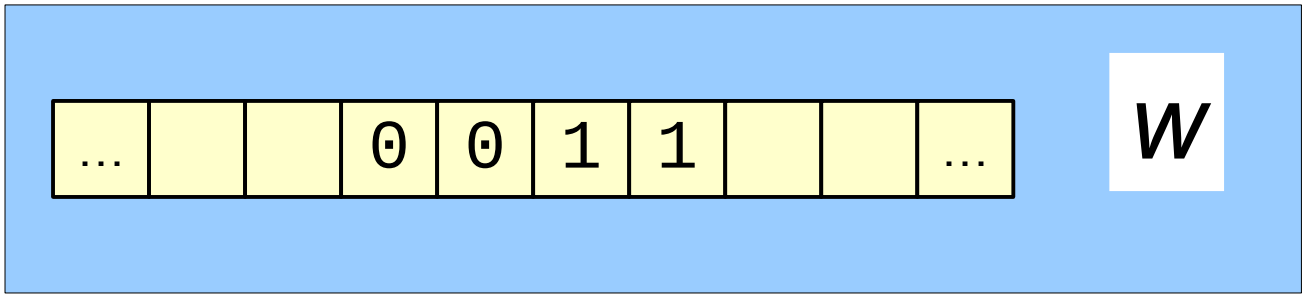
…			0	0	1	1			…





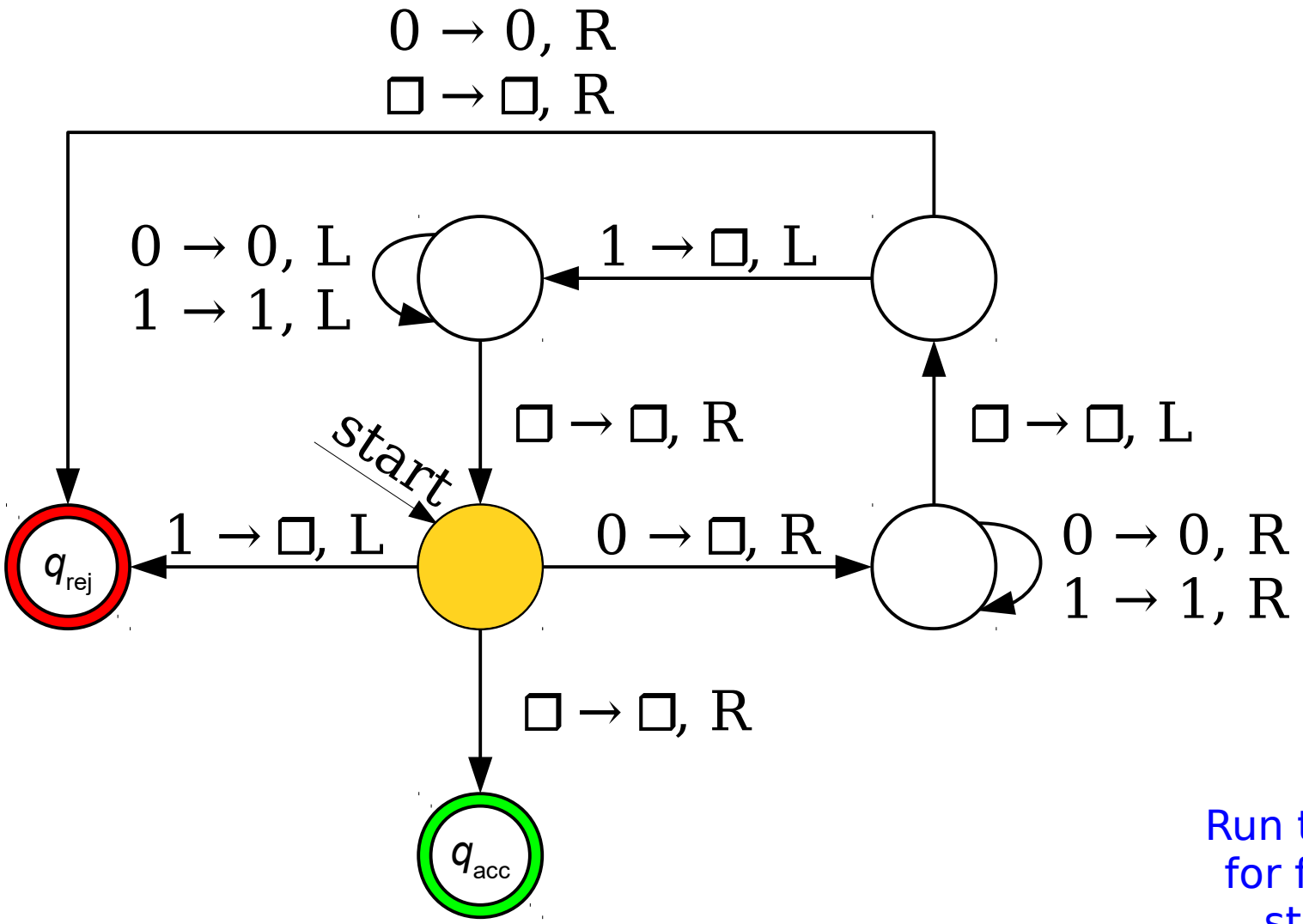
Run this TM for fifteen steps.



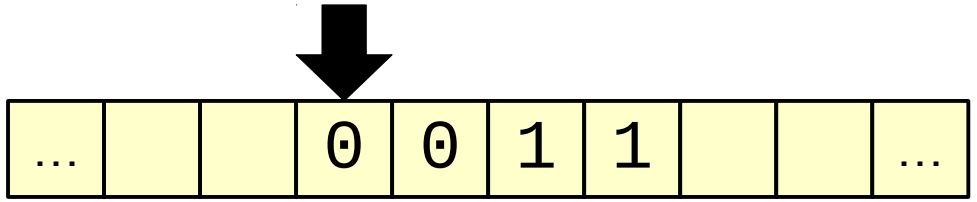


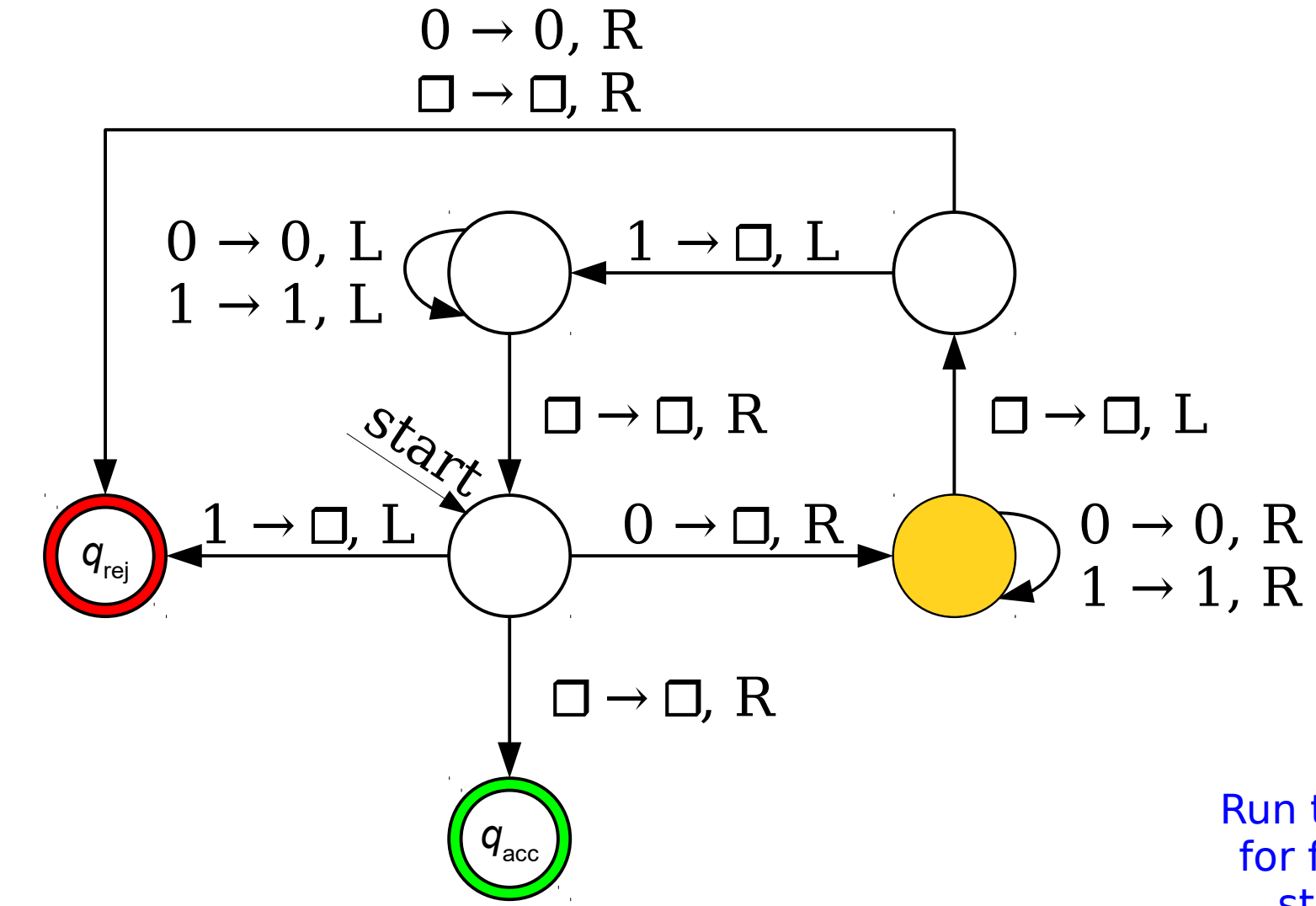
Run this TM for fifteen steps.

c

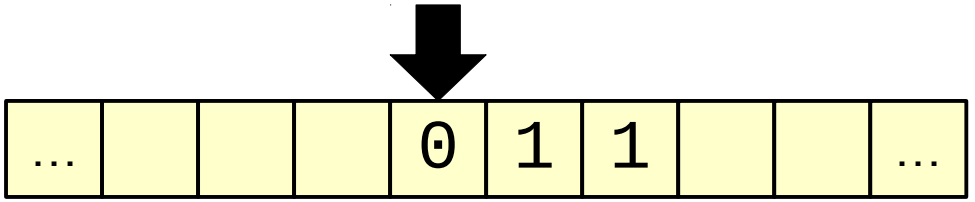


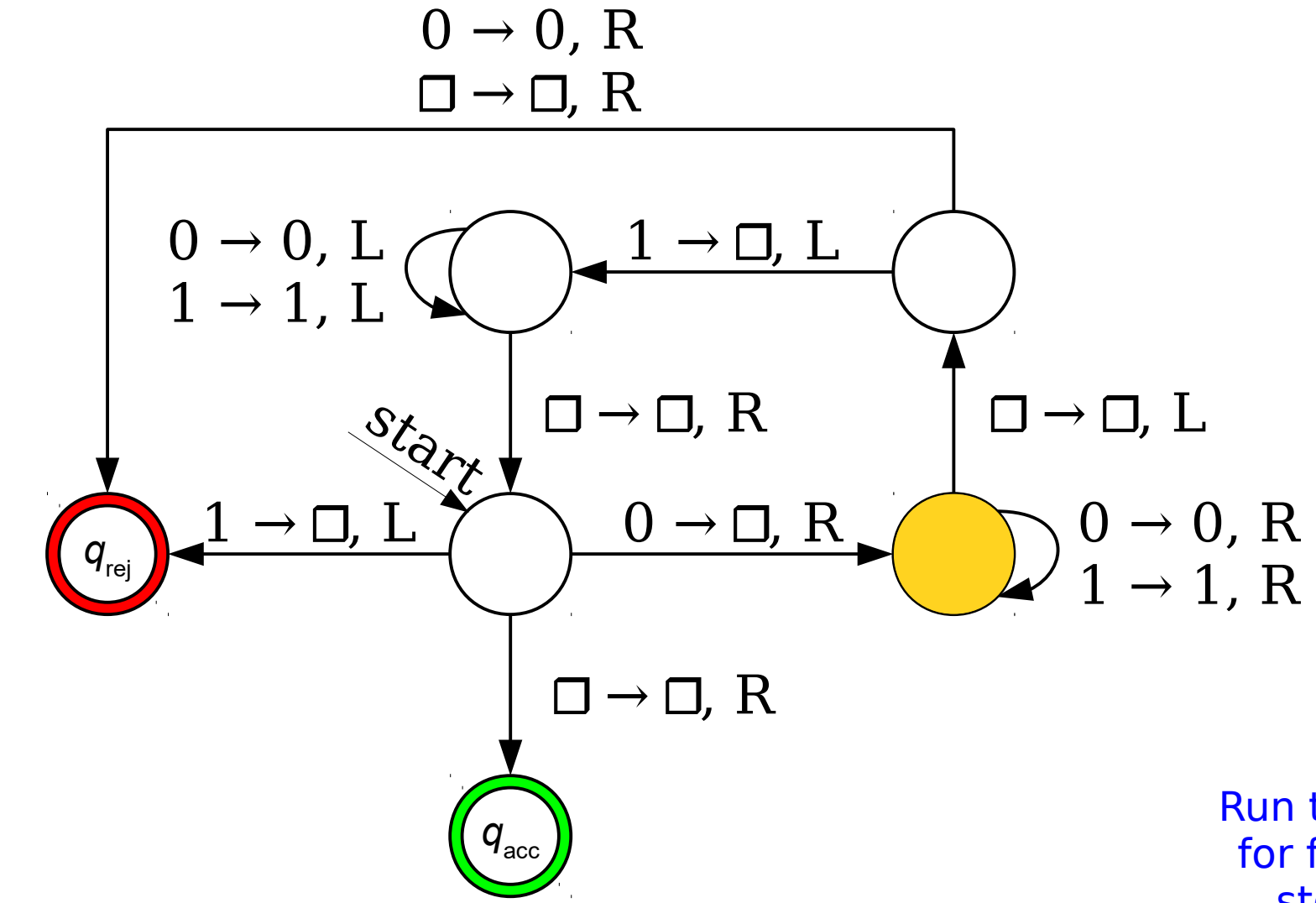
Run this TM for fifteen steps.



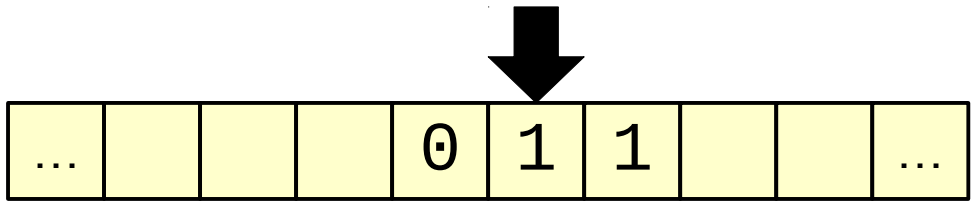


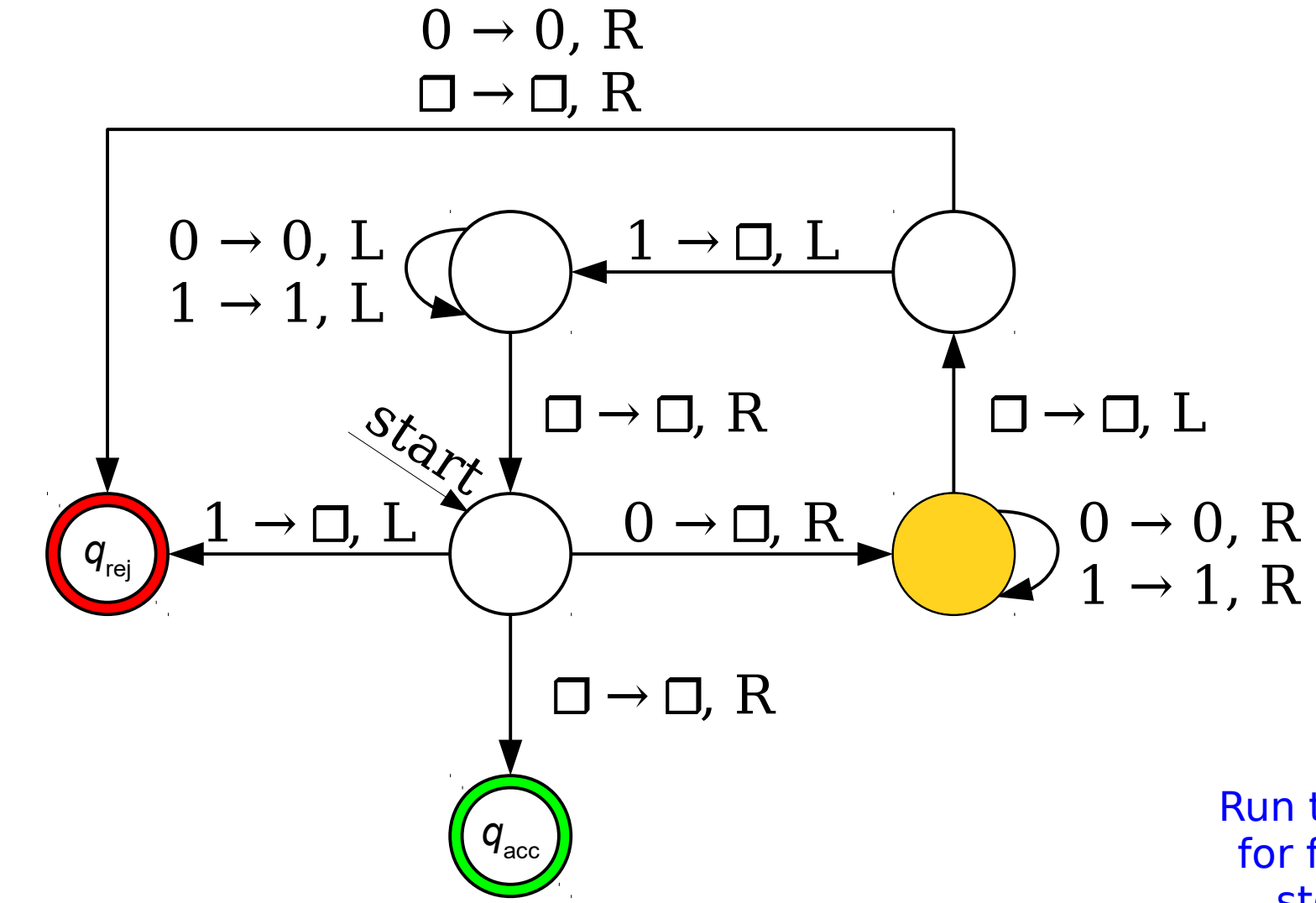
Run this TM for fifteen steps.

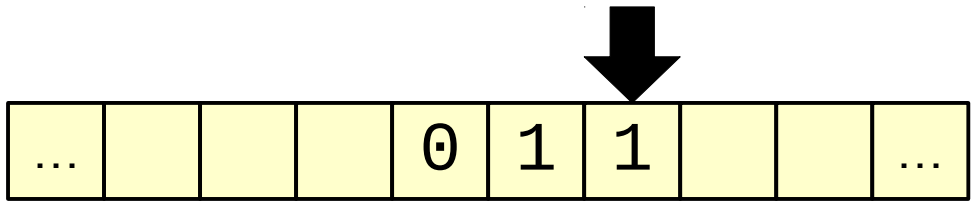




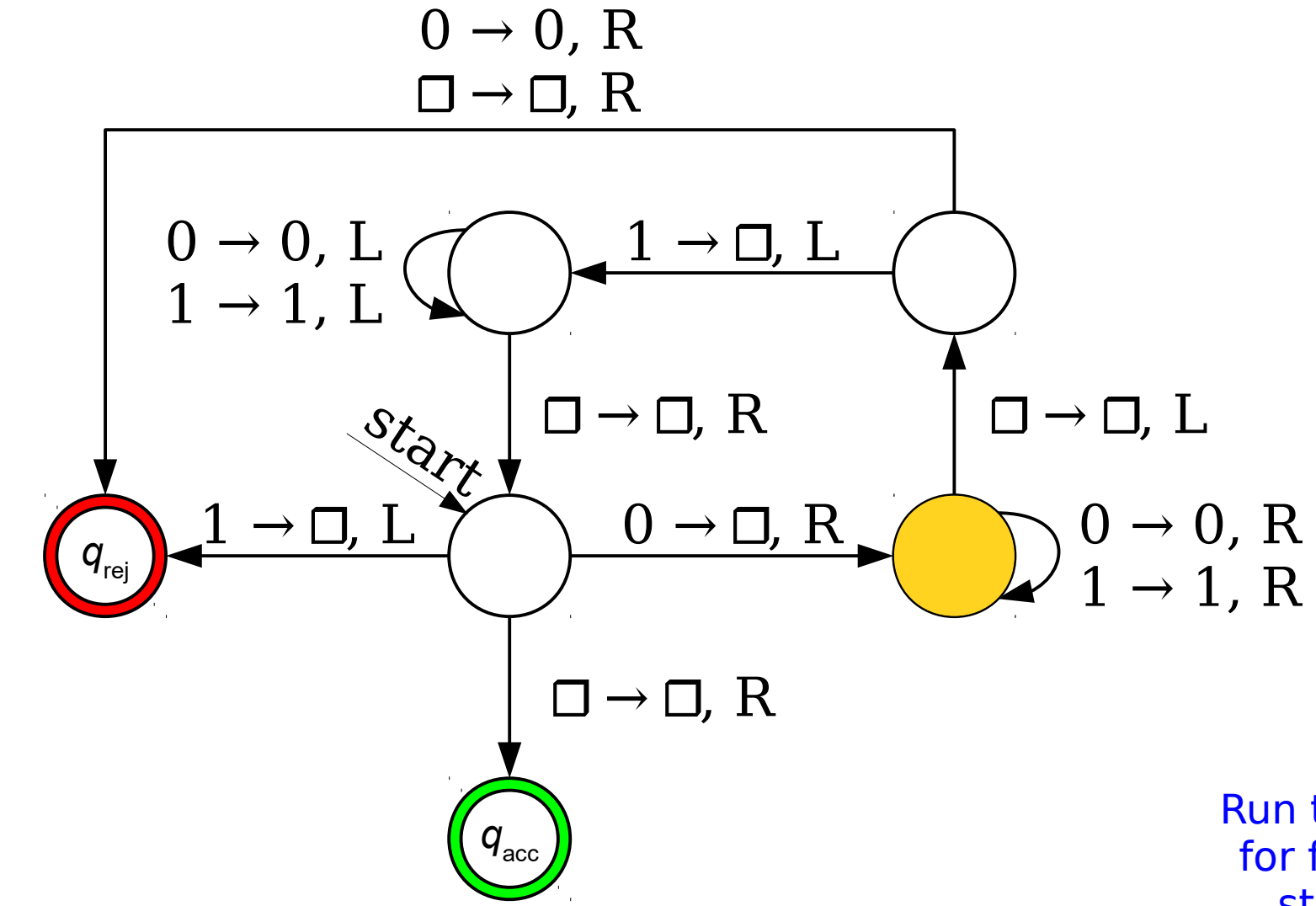
Run this TM for fifteen steps.

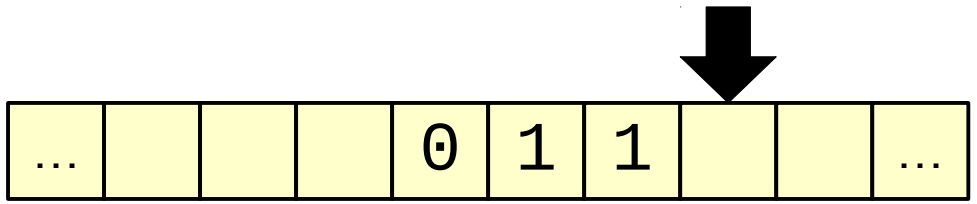




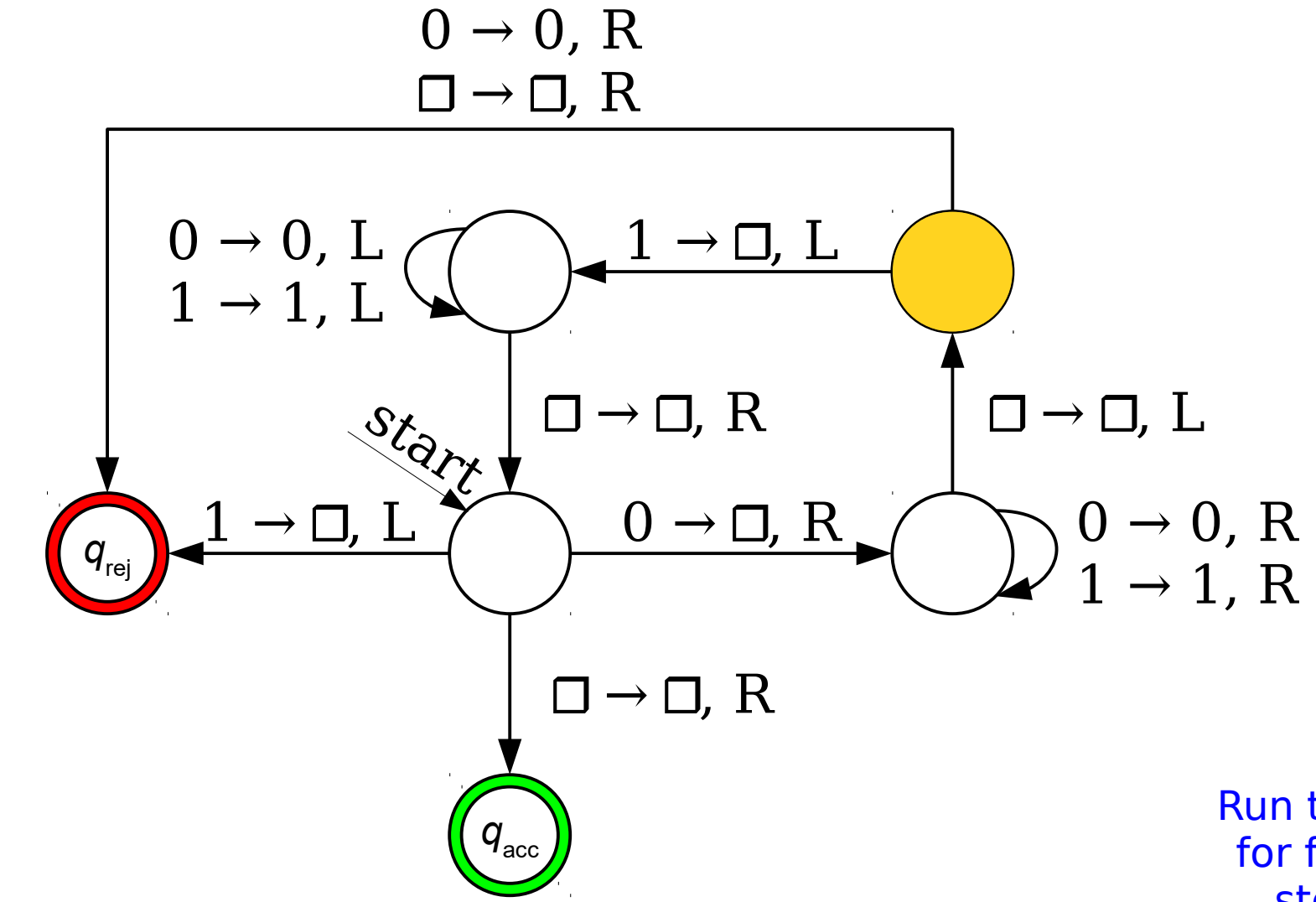


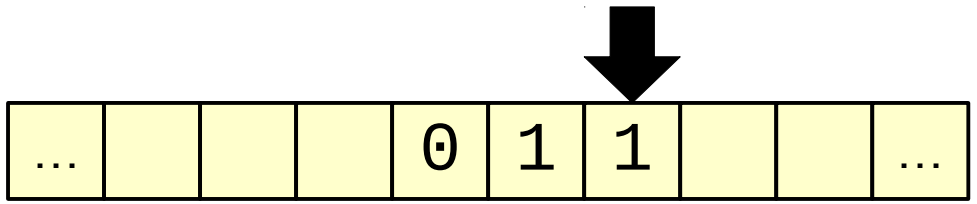
Run this TM for fifteen steps.



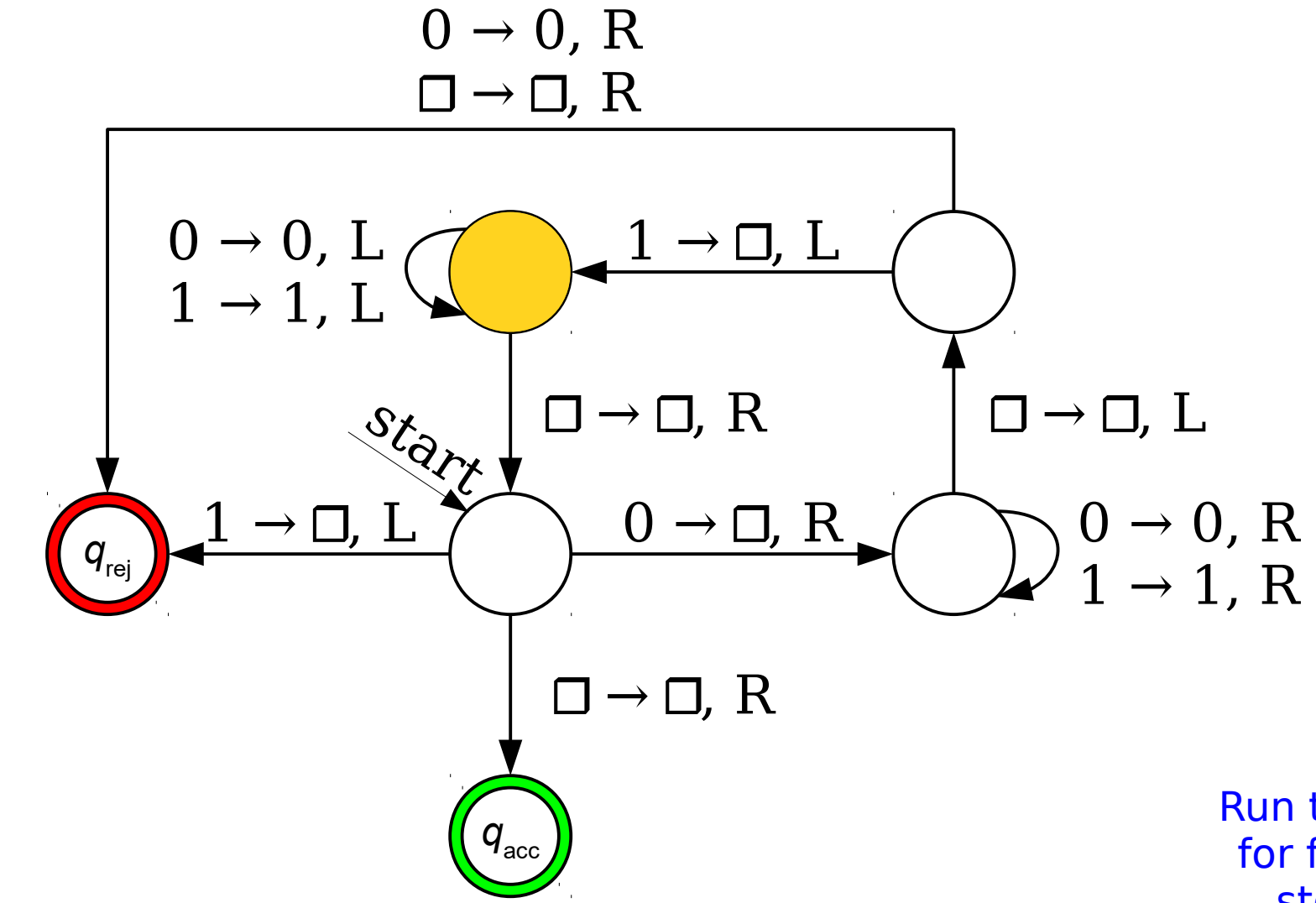


Run this TM for fifteen steps.

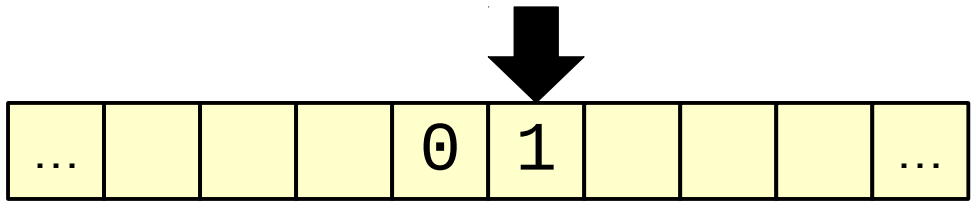


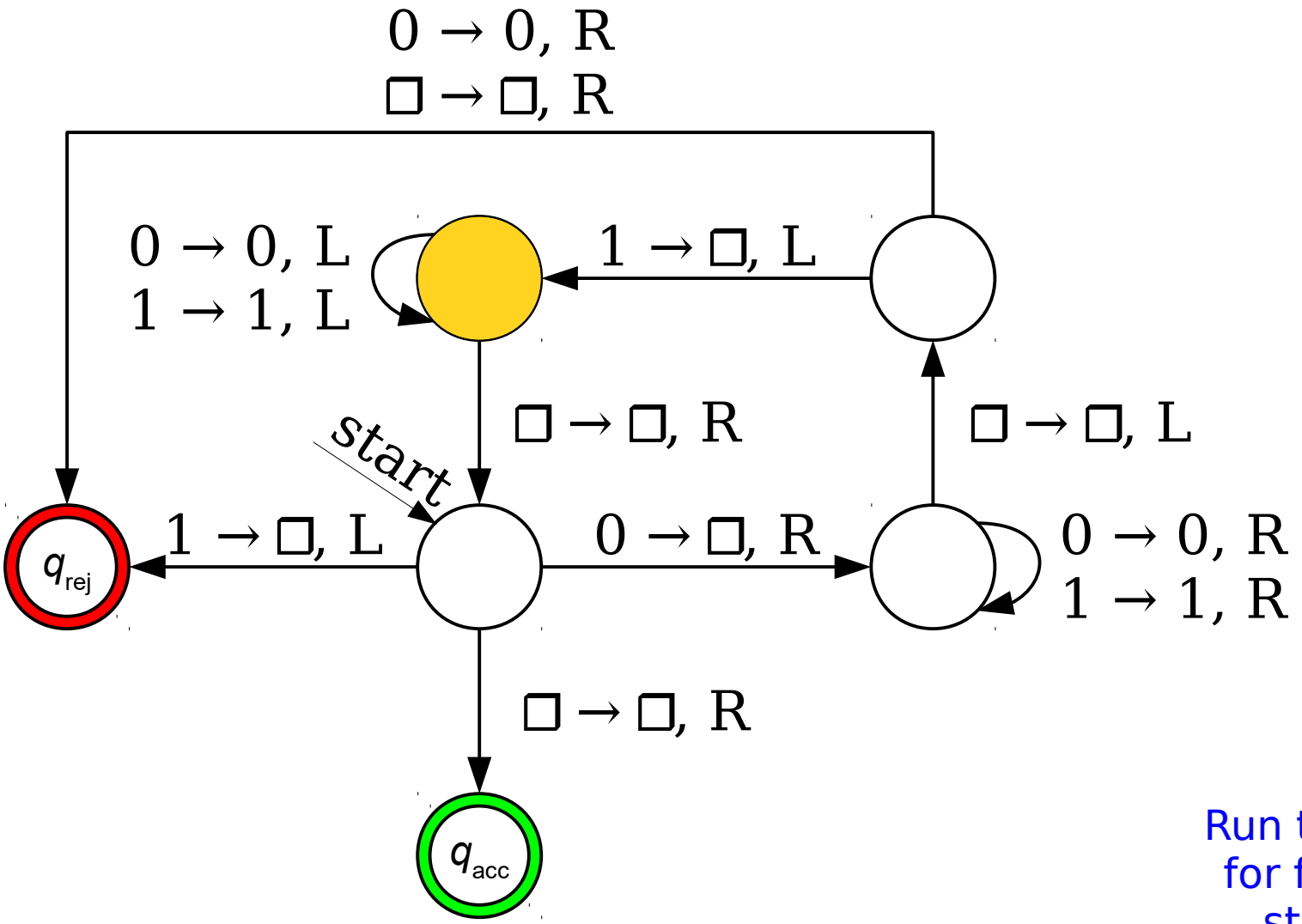


Run this TM for fifteen steps.

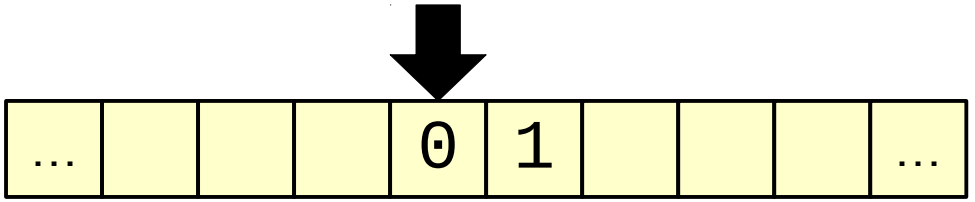


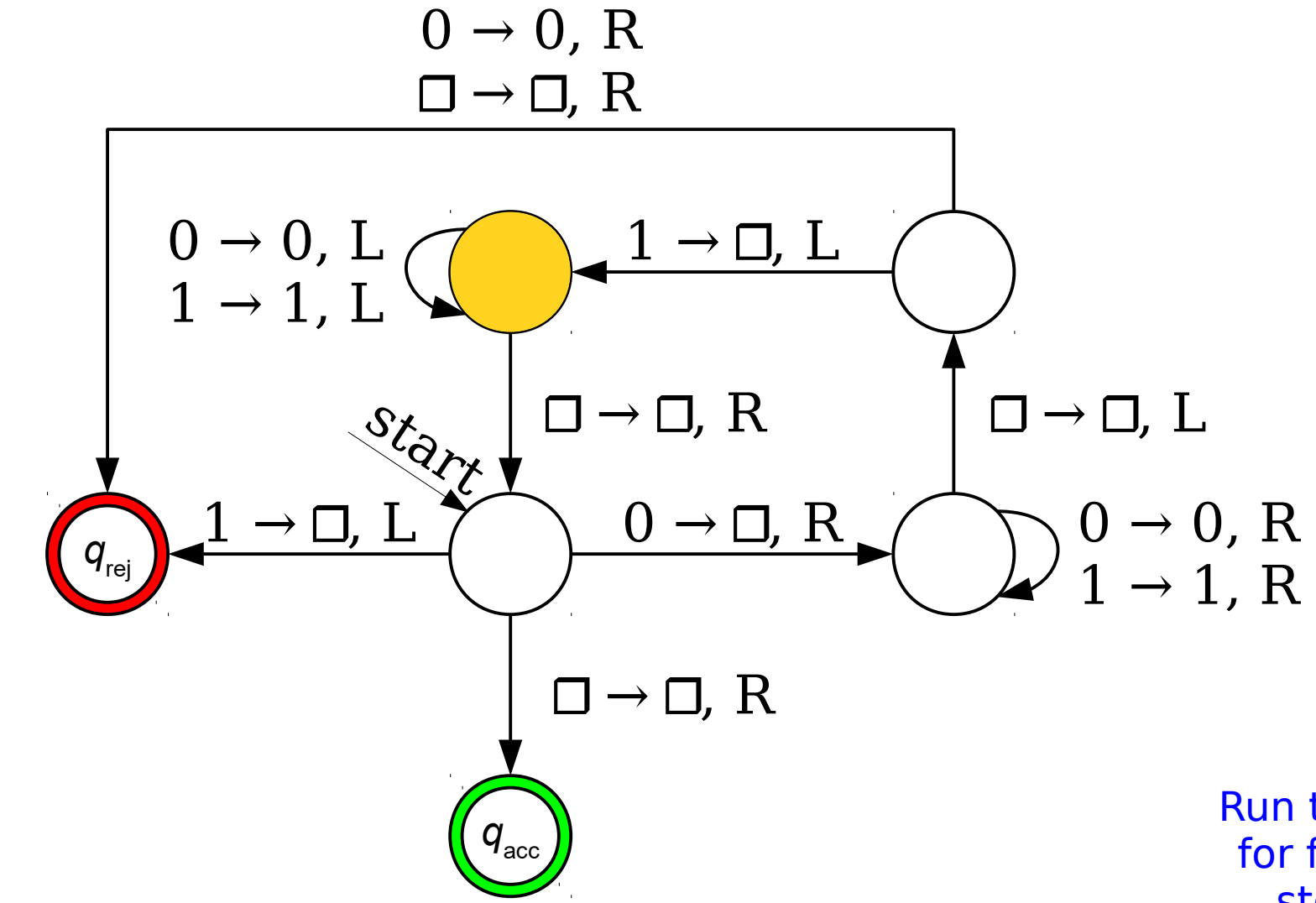
Run this TM for fifteen steps.



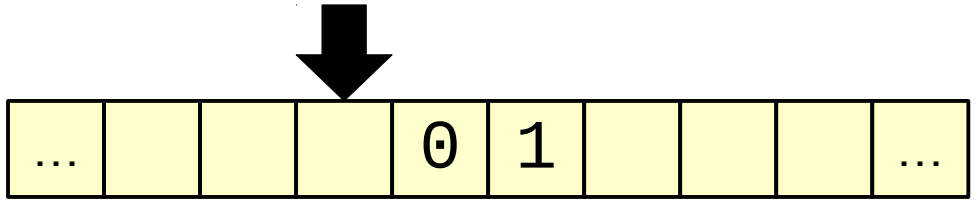


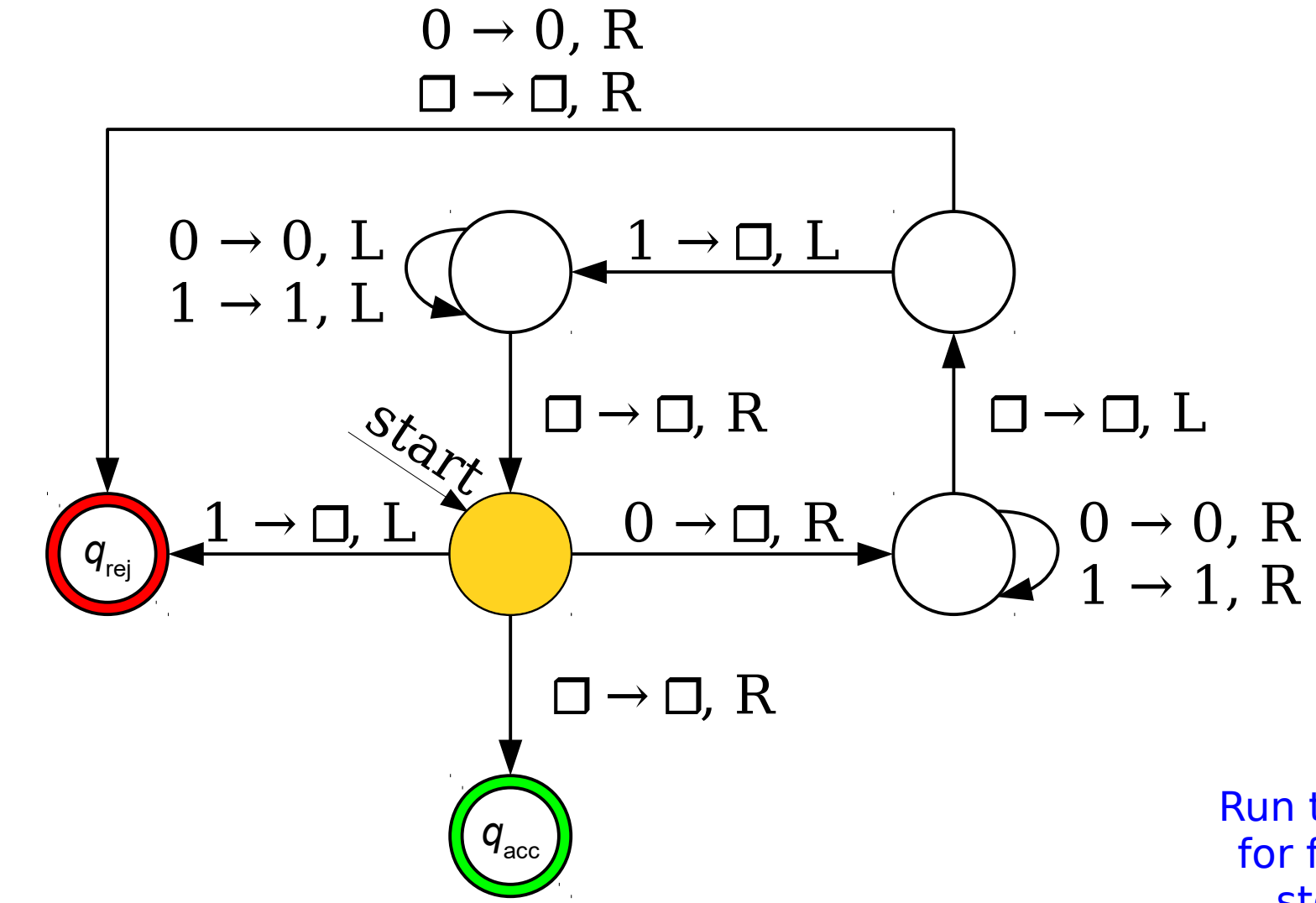
Run this TM for fifteen steps.



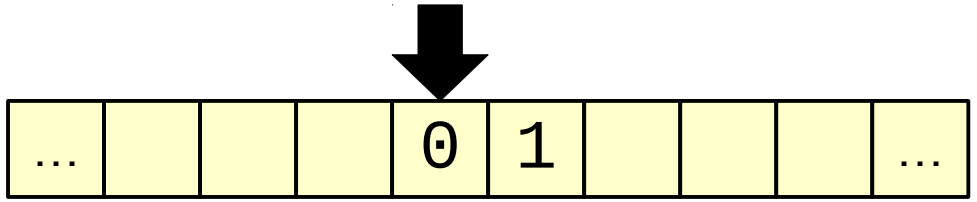


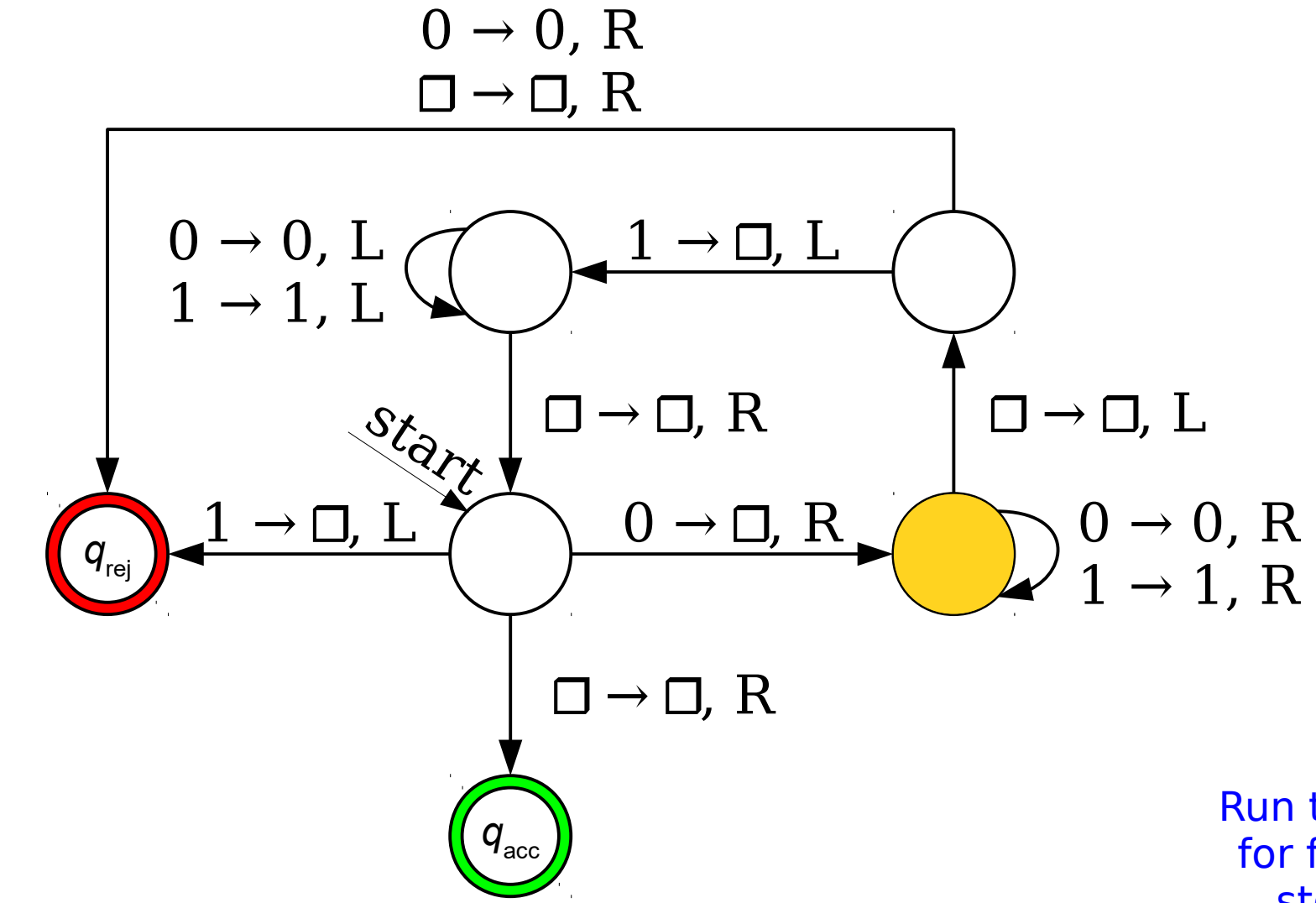
Run this TM for fifteen steps.



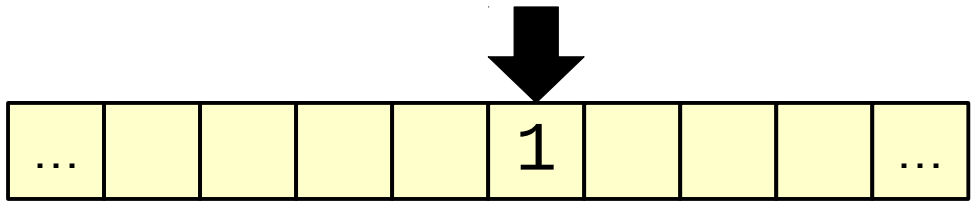


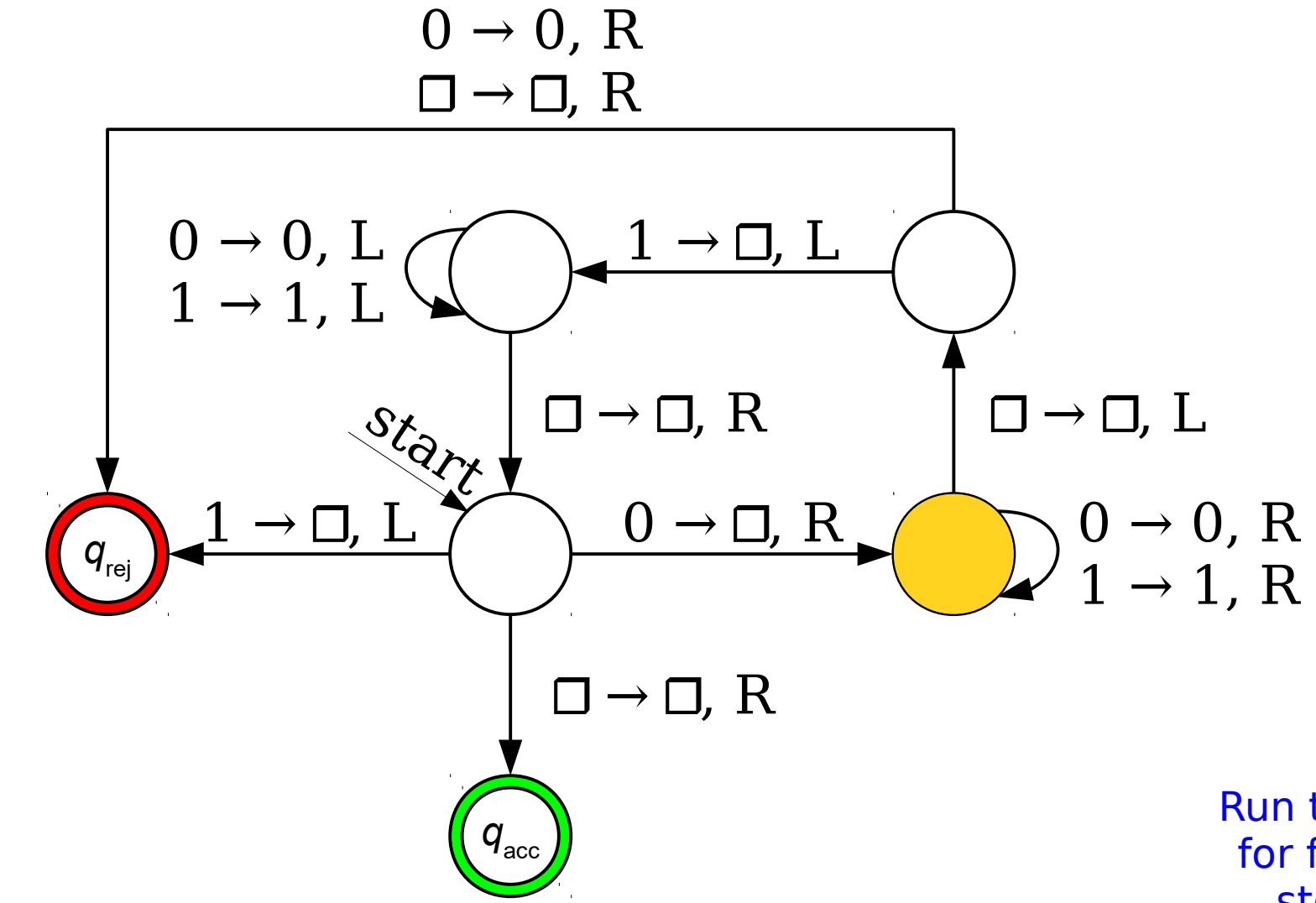
Run this TM for fifteen steps.

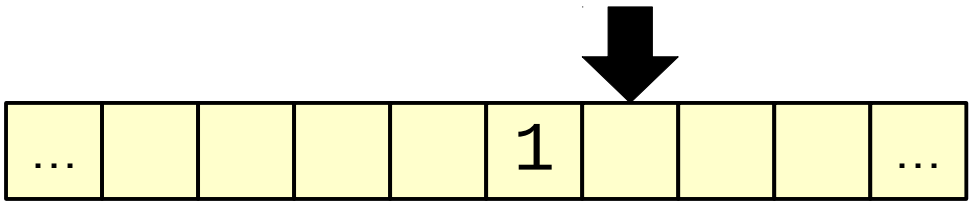




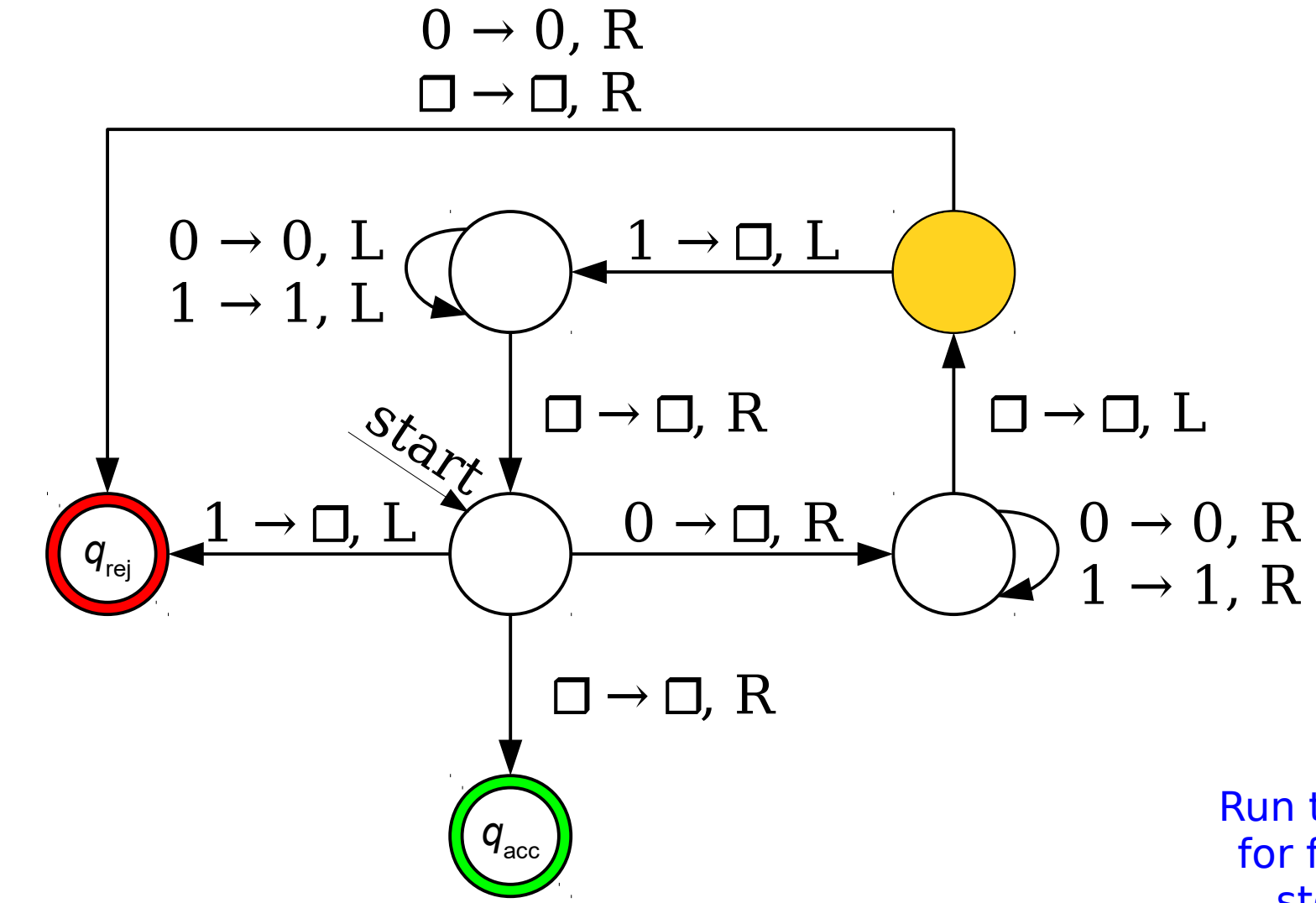
Run this TM for fifteen steps.



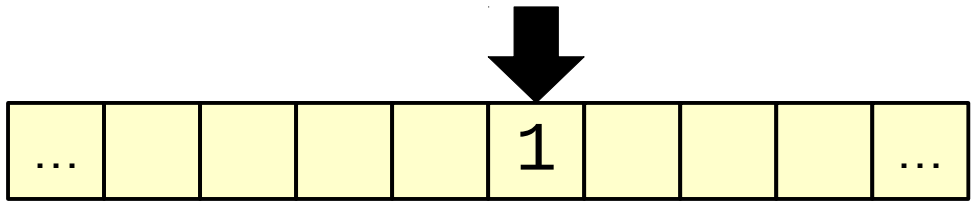


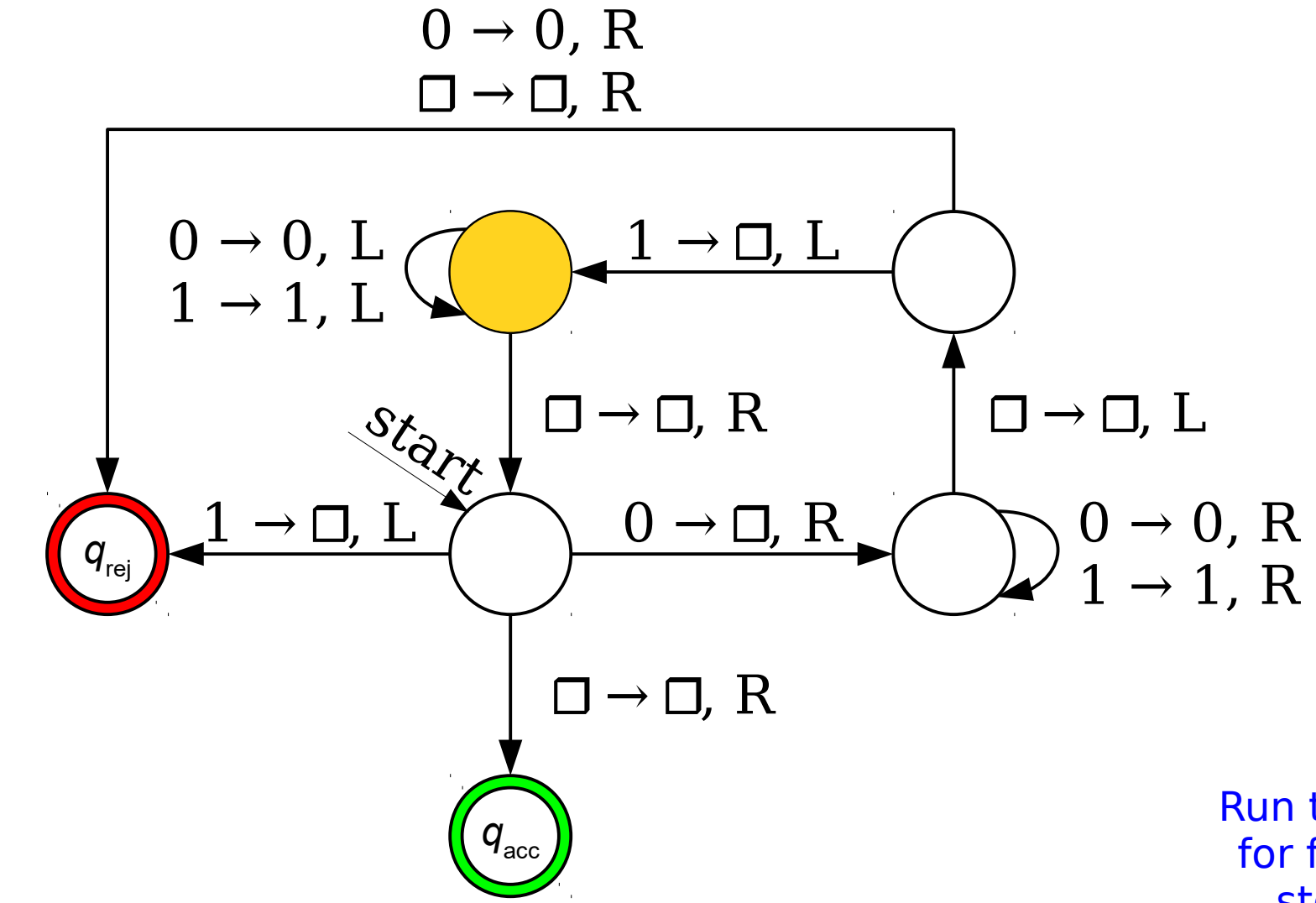


Run this TM for fifteen steps.

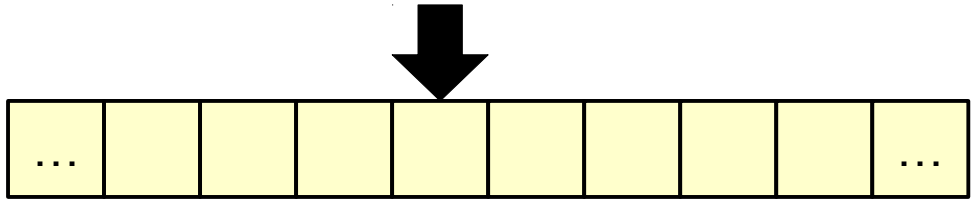


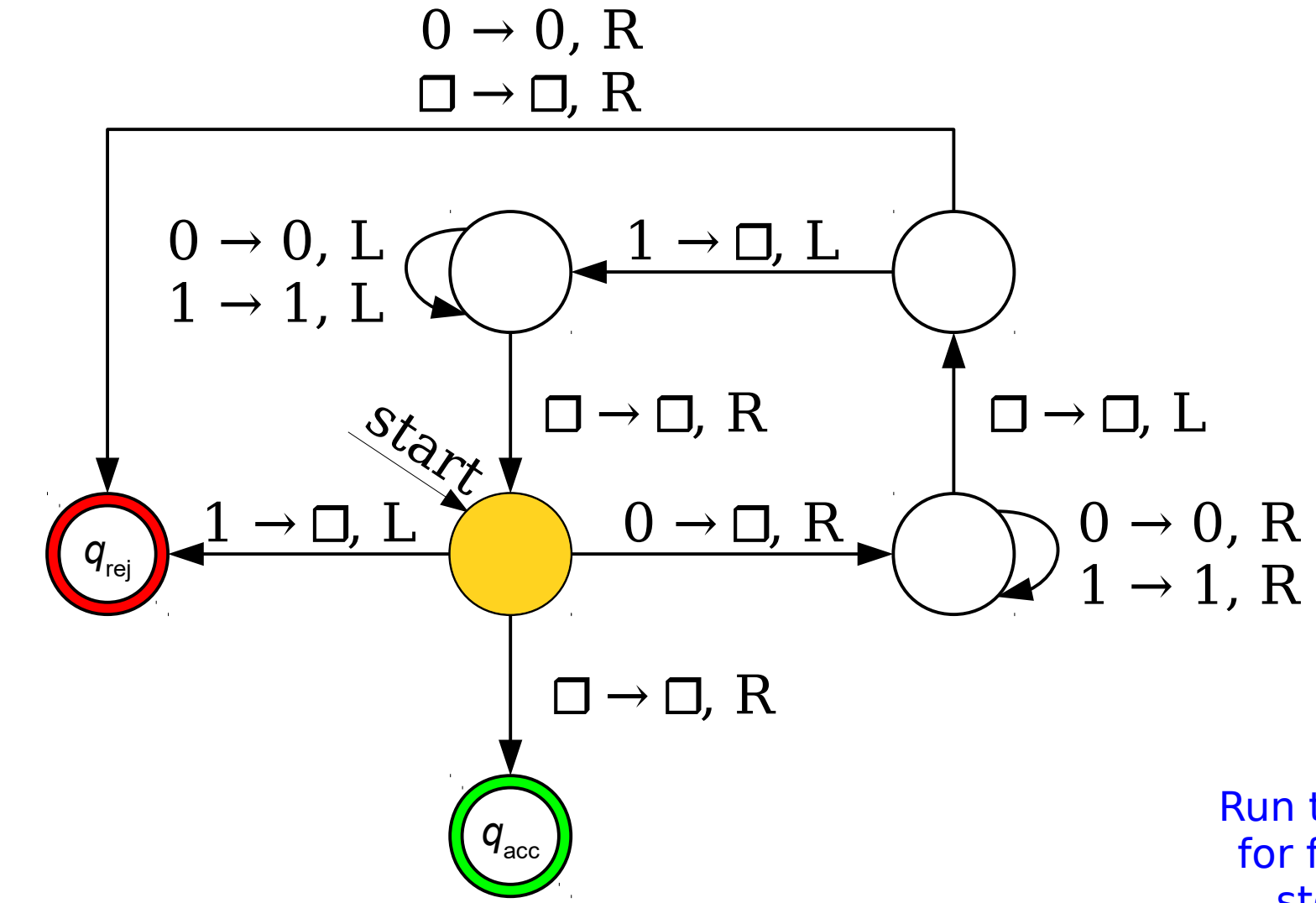
Run this TM for fifteen steps.



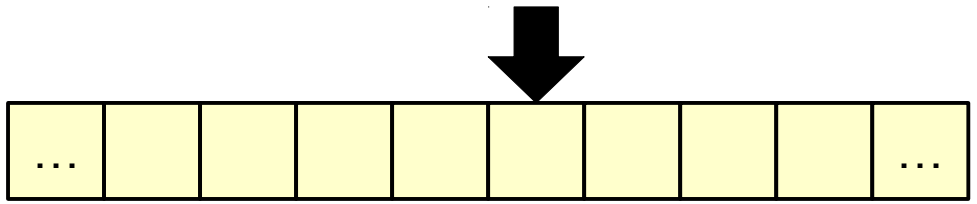


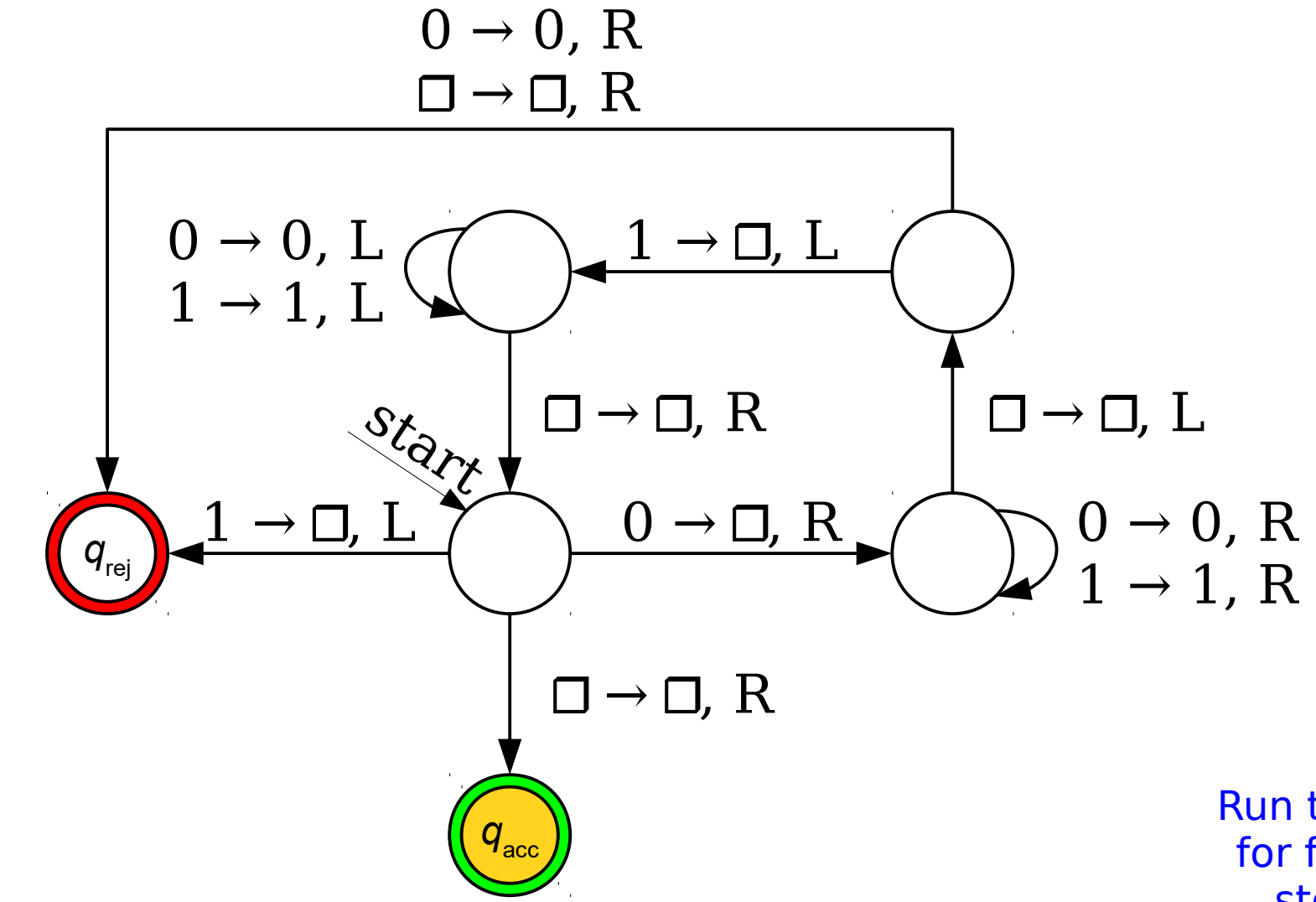
Run this TM for fifteen steps.



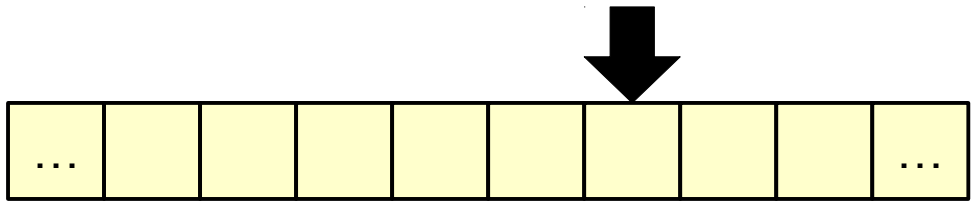


Run this TM for fifteen steps.





Run this TM for fifteen steps.



A Verifier for A_{TM}

- Recall $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$

```
bool checkWillAccept(TM M, string w, int c) {
    set up a simulation of M running on w;
    for (int i = 0; i < c; i++) {
       simulate the next step of M running on w;
    }
    return whether M is in an accepting state;
}
```

- Do you see why M accepts w iff there is some c such that `checkWillAccept(M, w, c)` returns true?
- Do you see why `checkWillAccept` always halts?

Equivalence of Verifiers and Recognizers

What languages are verifiable?

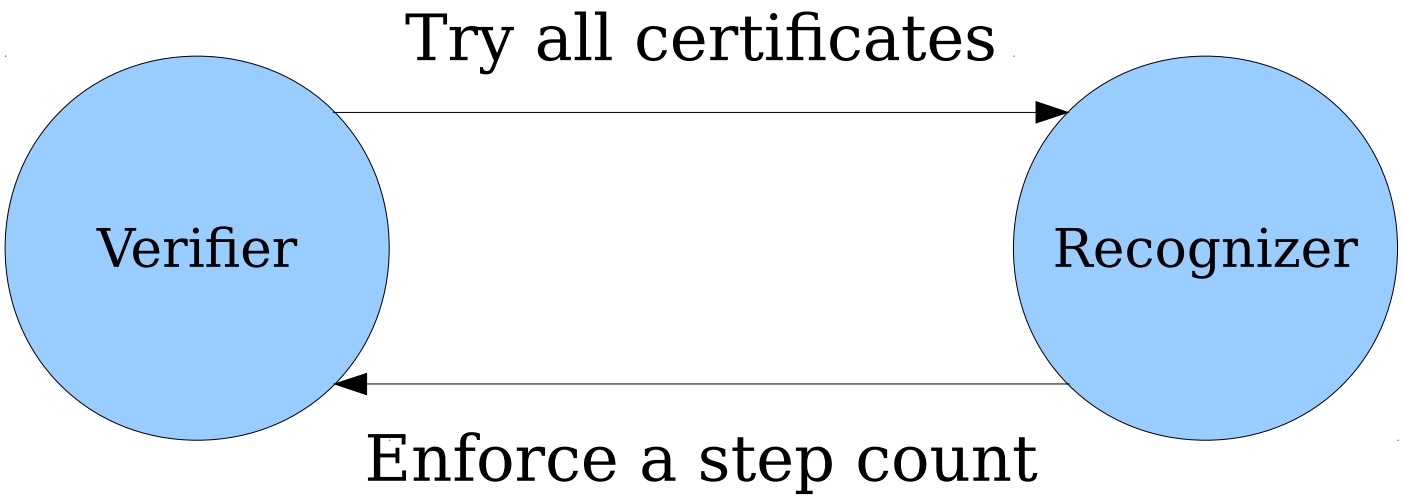
Let V be a verifier for a language L. Consider the following function given in pseudocode:

```
bool mysteryFunction(string w) {
    int i = 0;
    while (true) {
        for (each string c of length i) {
            if (V accepts ⟨w, c⟩) return true;
        }
        i++;
    }
}
```

What set of strings does `mysteryFunction` return **`true`** on?

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then **your answer**.

Equivalence of Verifiers and Recognizers



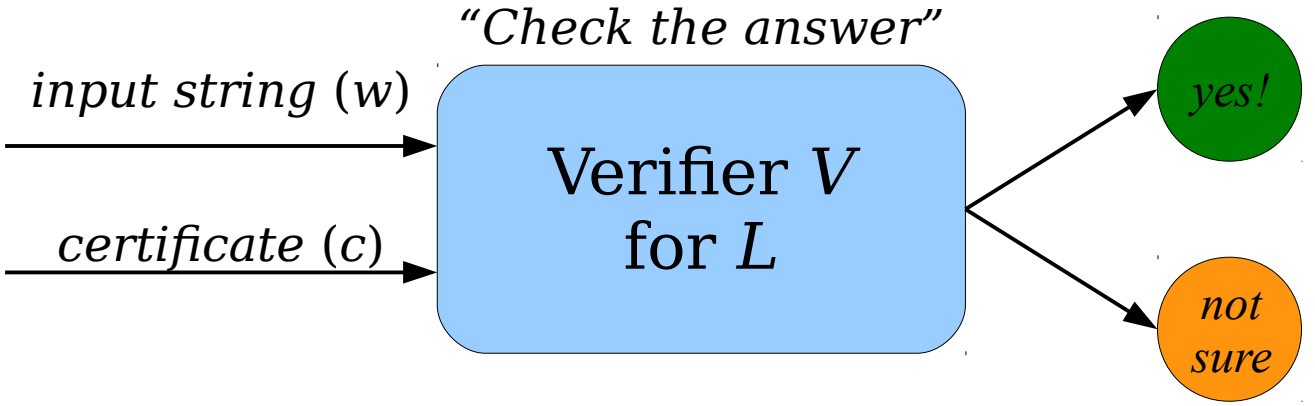
Theorem: If L is a language, then there is a verifier for L if and only if $L \in \mathbf{RE}$.

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.

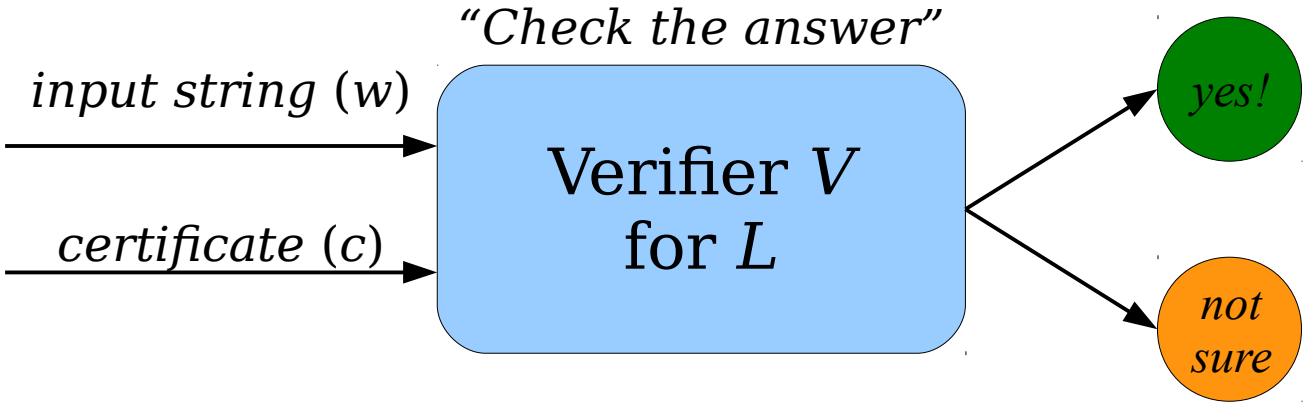
Verifiers and **RE**

- ***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.

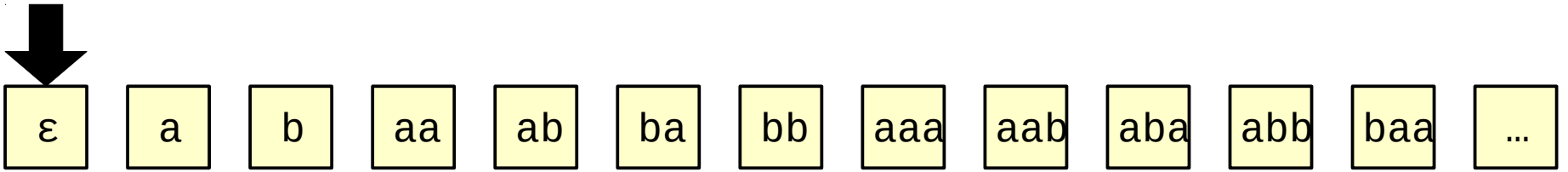


Verifiers and **RE**

- ***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.

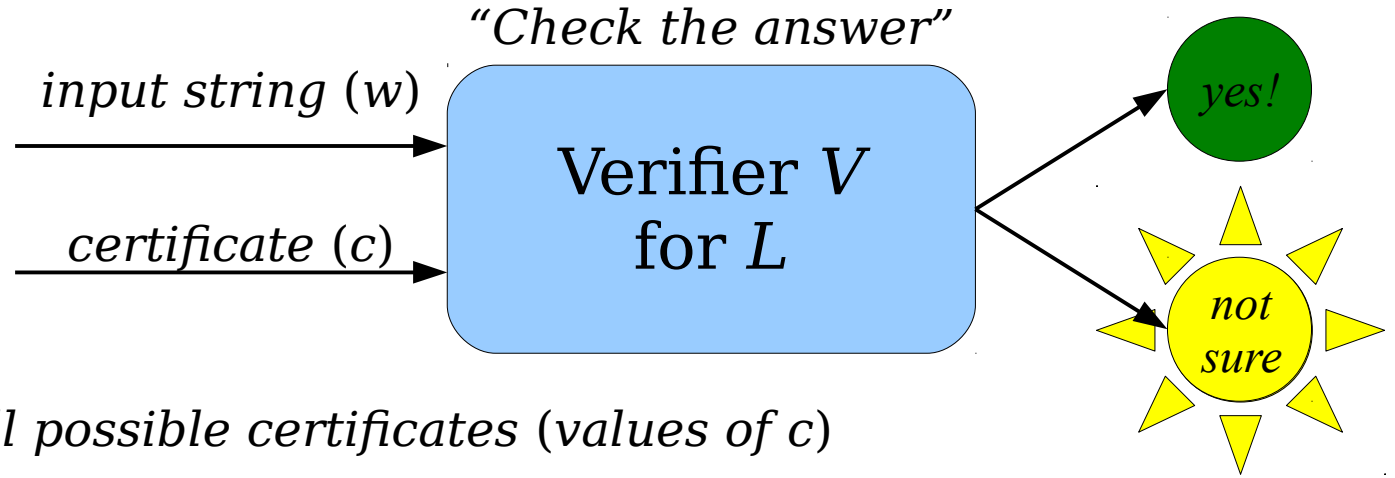


We will try all possible certificates (values of c)

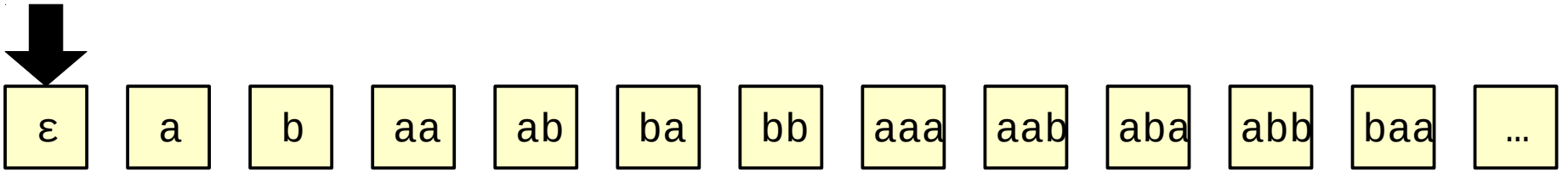


Verifiers and **RE**

- ***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.

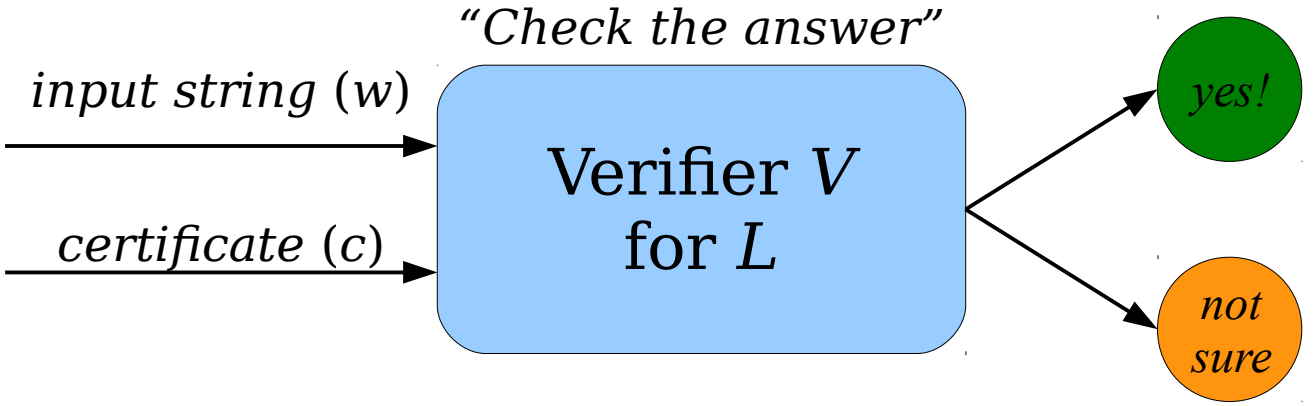


We will try all possible certificates (values of c)

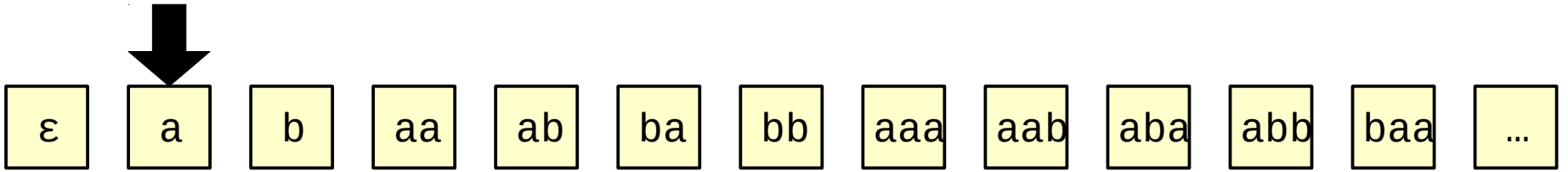


Verifiers and **RE**

- ***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.

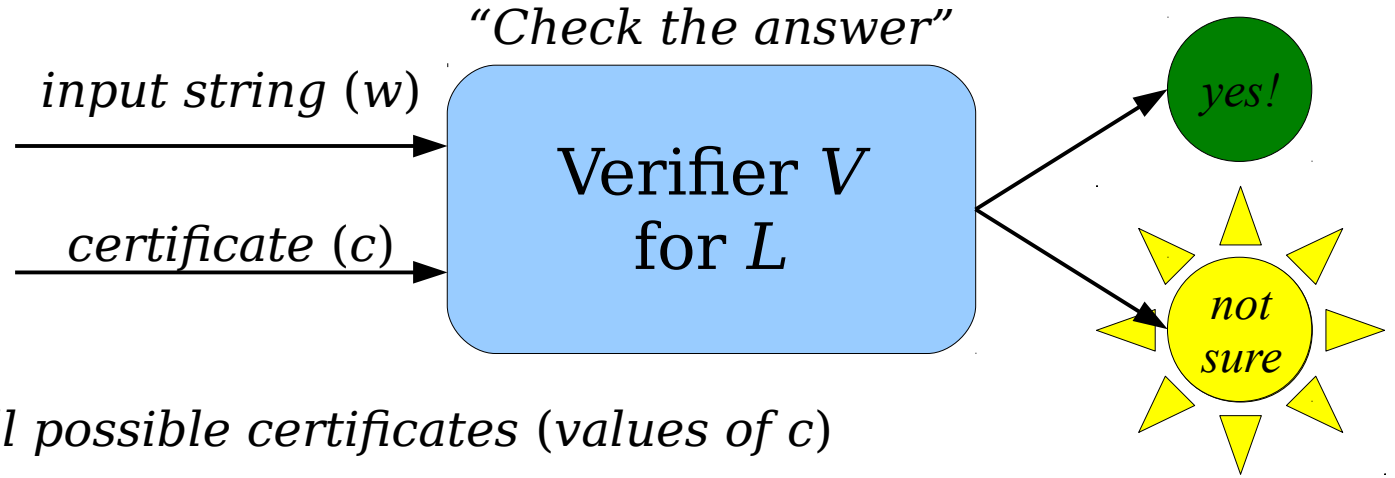


We will try all possible certificates (values of c)

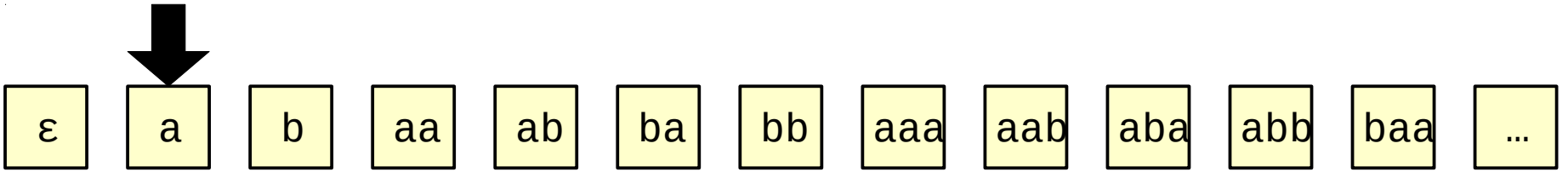


Verifiers and **RE**

- ***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.

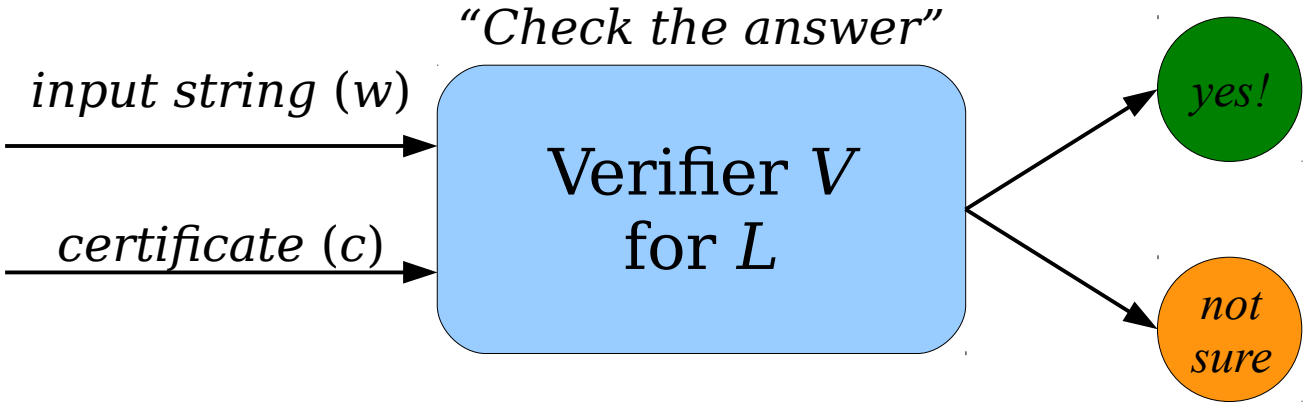


We will try all possible certificates (values of c)

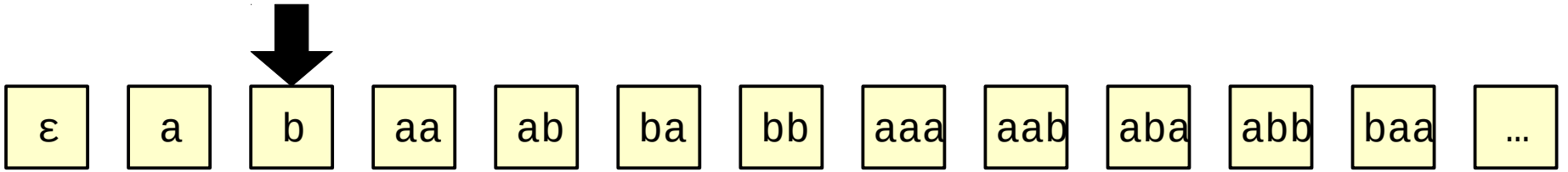


Verifiers and **RE**

- ***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.

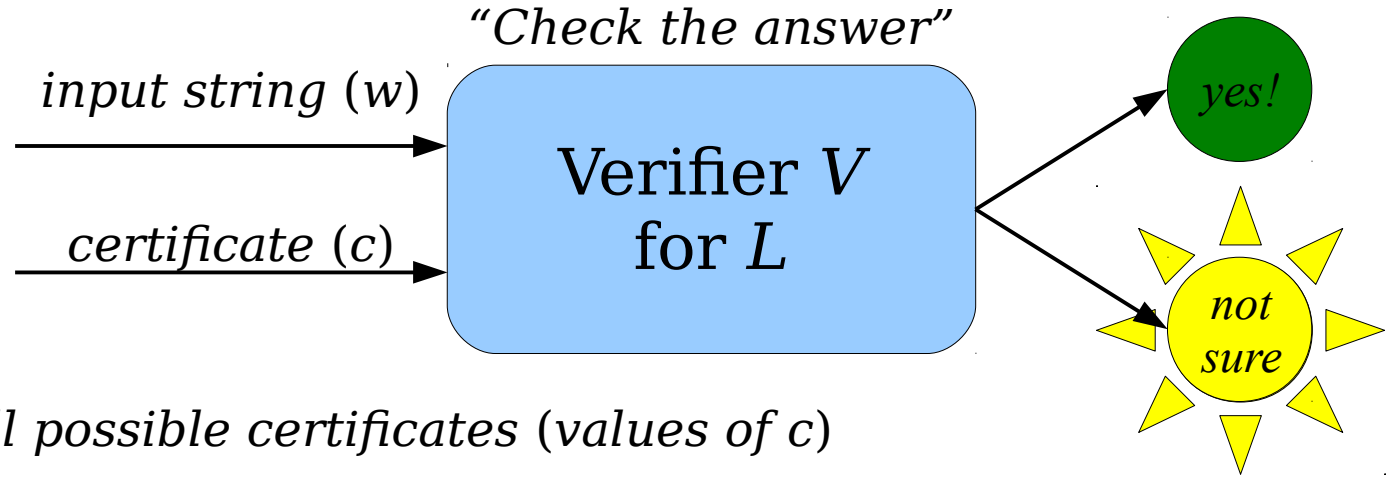


We will try all possible certificates (values of c)

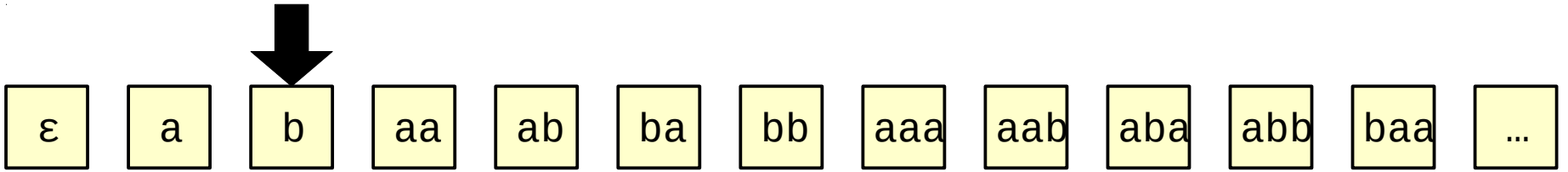


Verifiers and **RE**

- ***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.

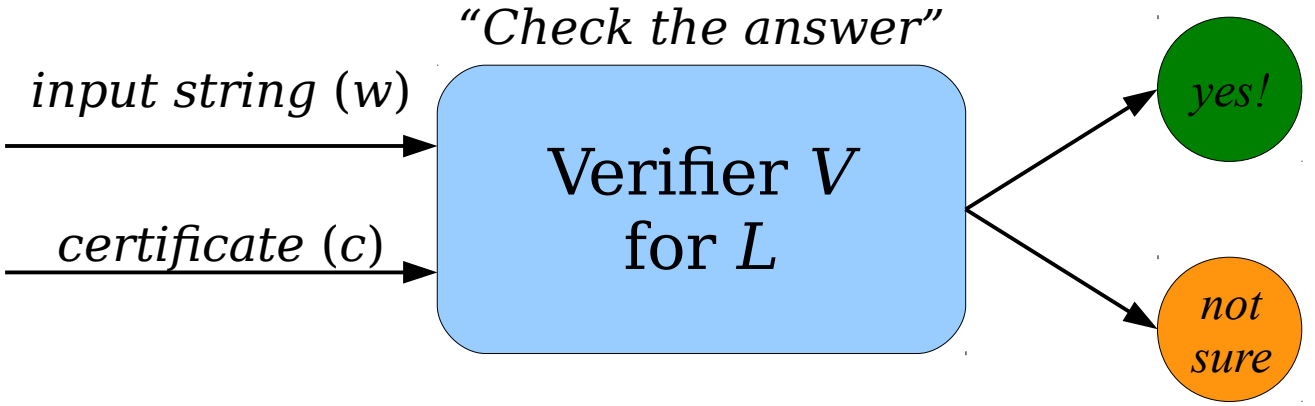


We will try all possible certificates (values of c)

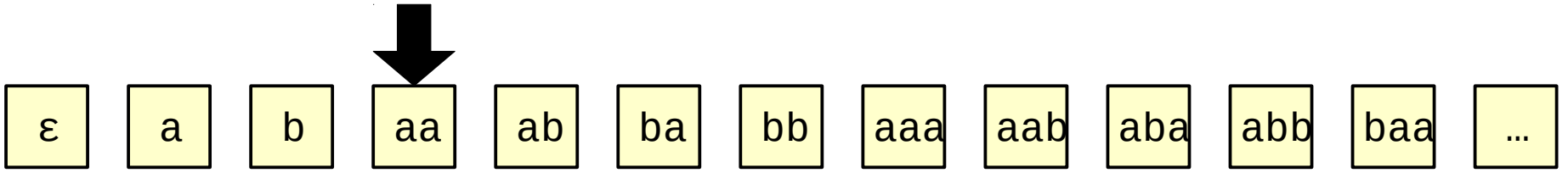


Verifiers and **RE**

- ***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.

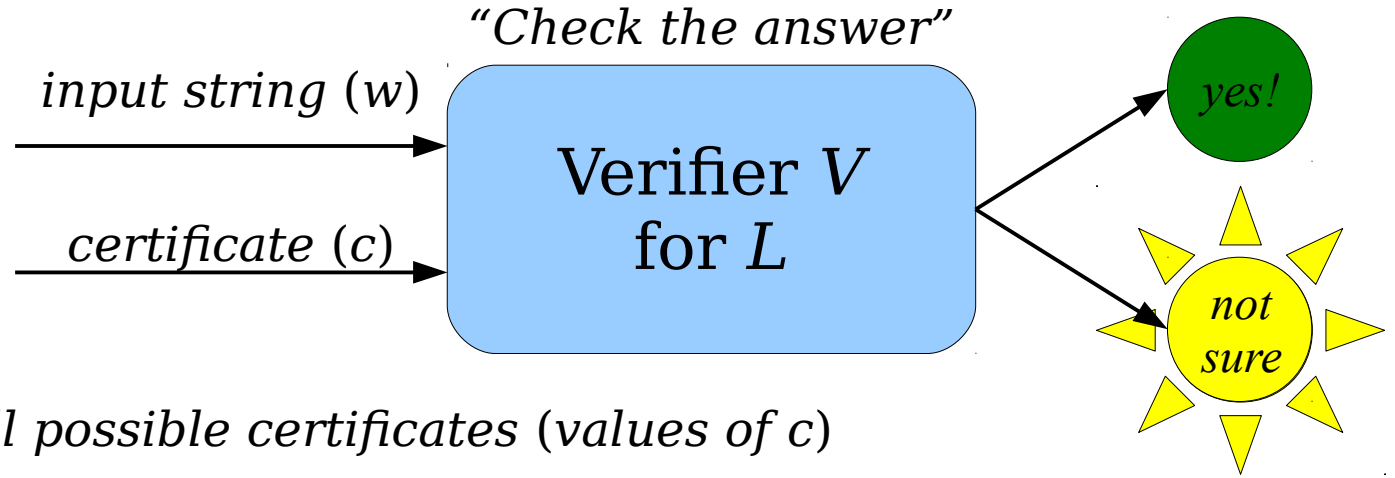


We will try all possible certificates (values of c)

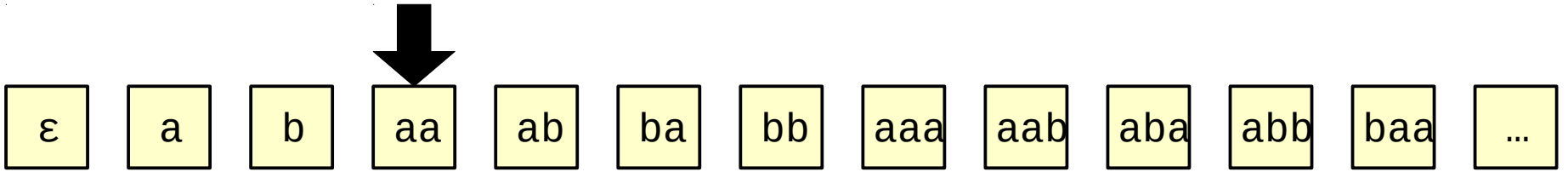


Verifiers and **RE**

- ***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.

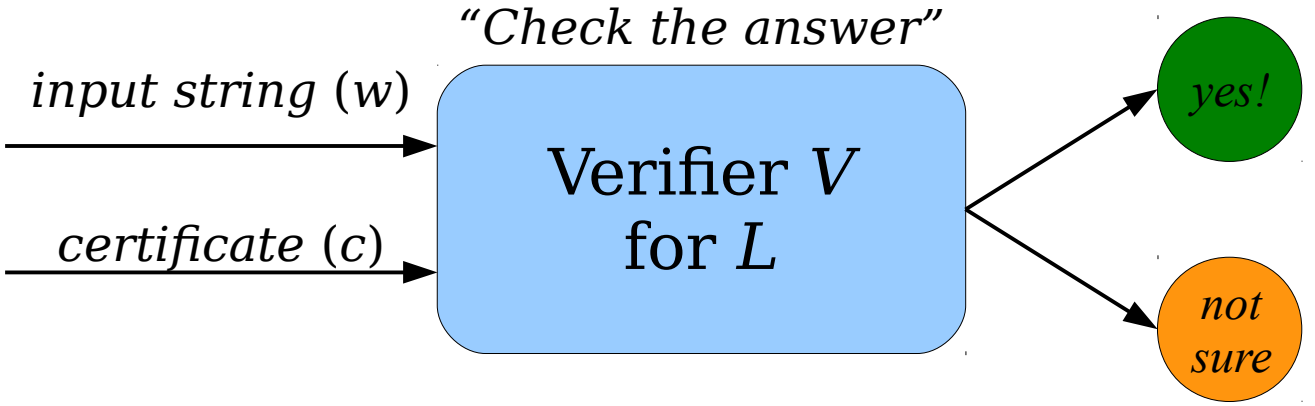


We will try all possible certificates (values of c)

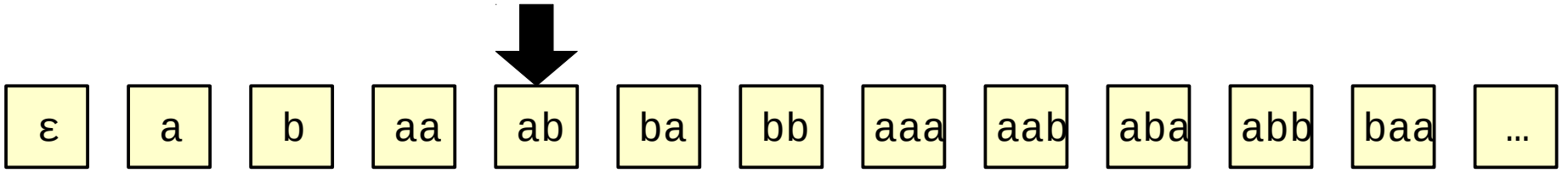


Verifiers and **RE**

- ***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.

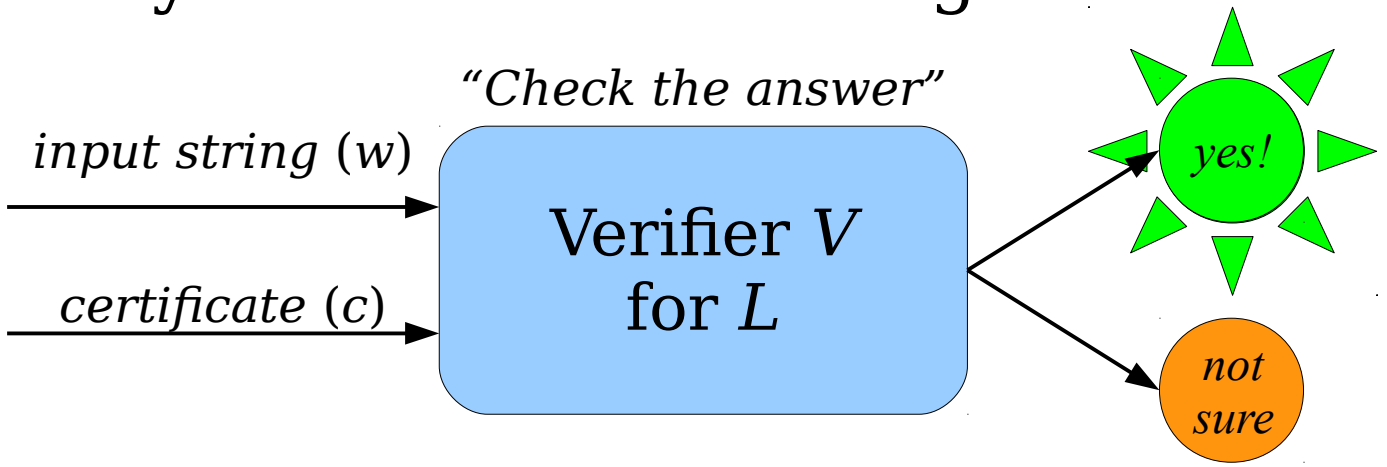


We will try all possible certificates (values of c)

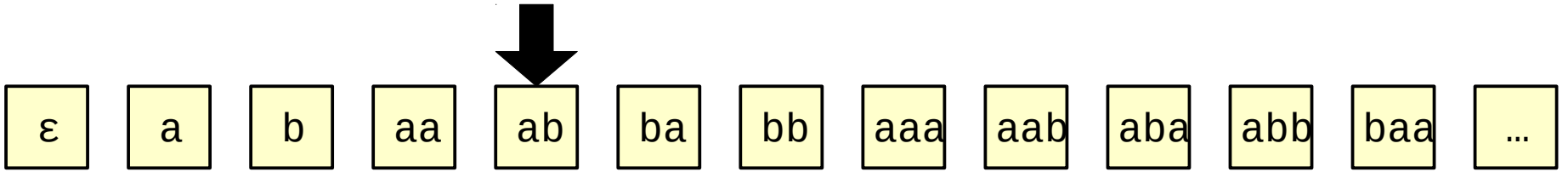


Verifiers and **RE**

- ***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.

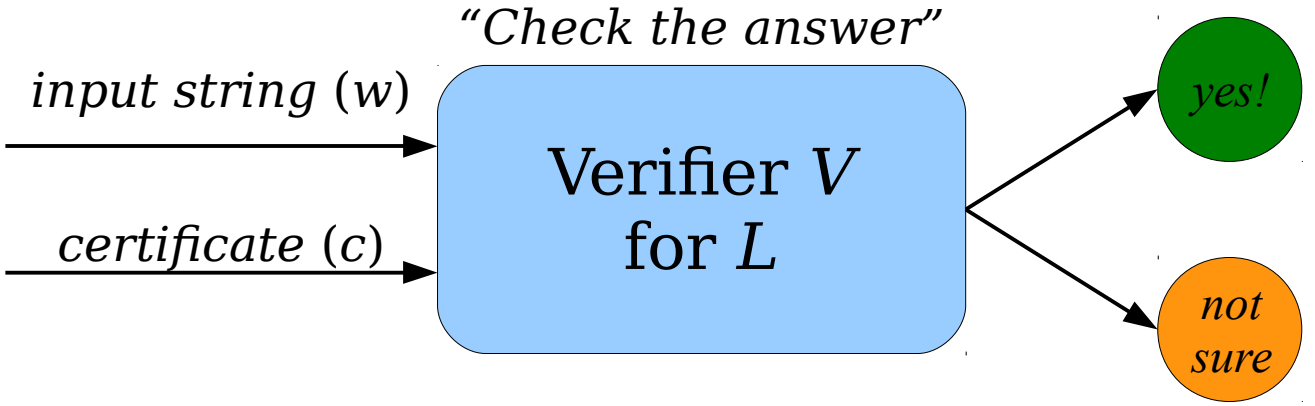


We will try all possible certificates (values of c)



Verifiers and **RE**

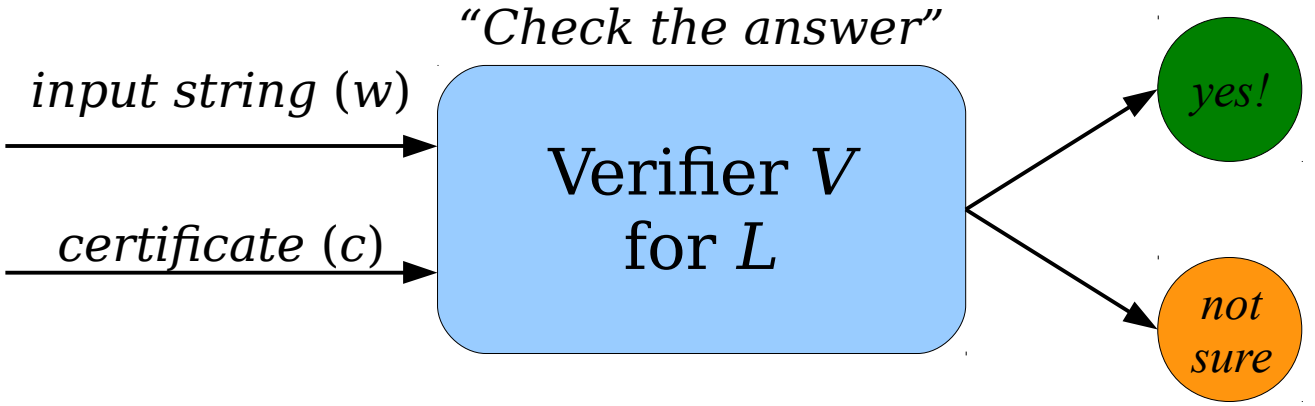
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



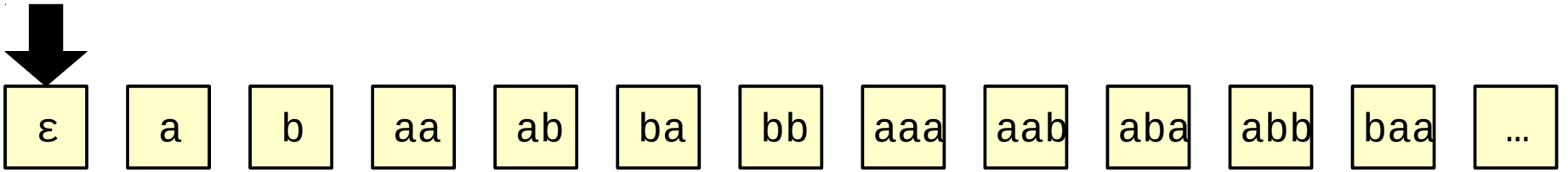
ε	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	…

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.

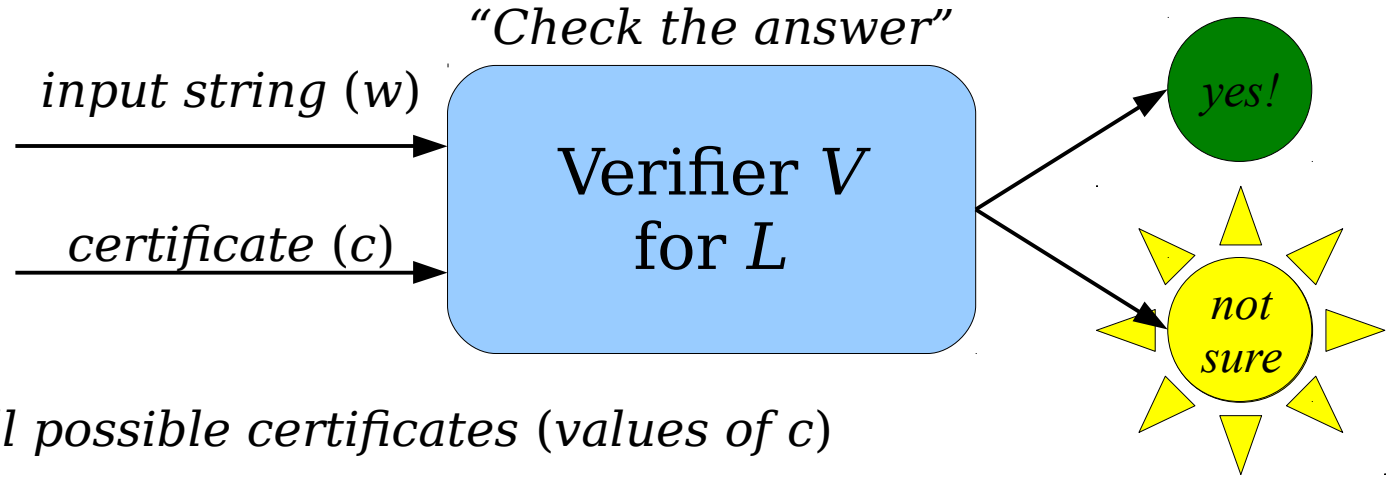


We will try all possible certificates (values of c)

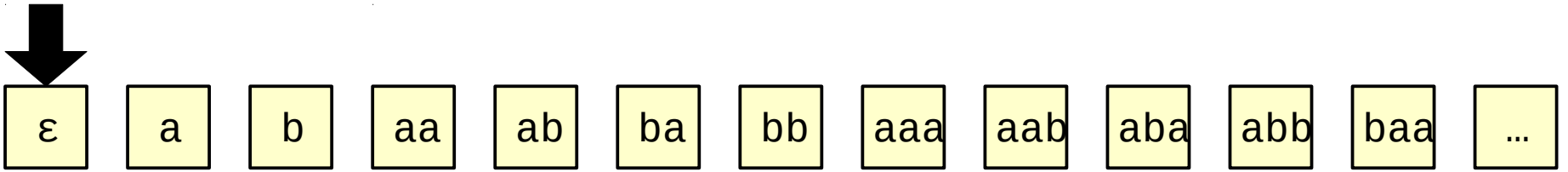


Verifiers and **RE**

- ***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.

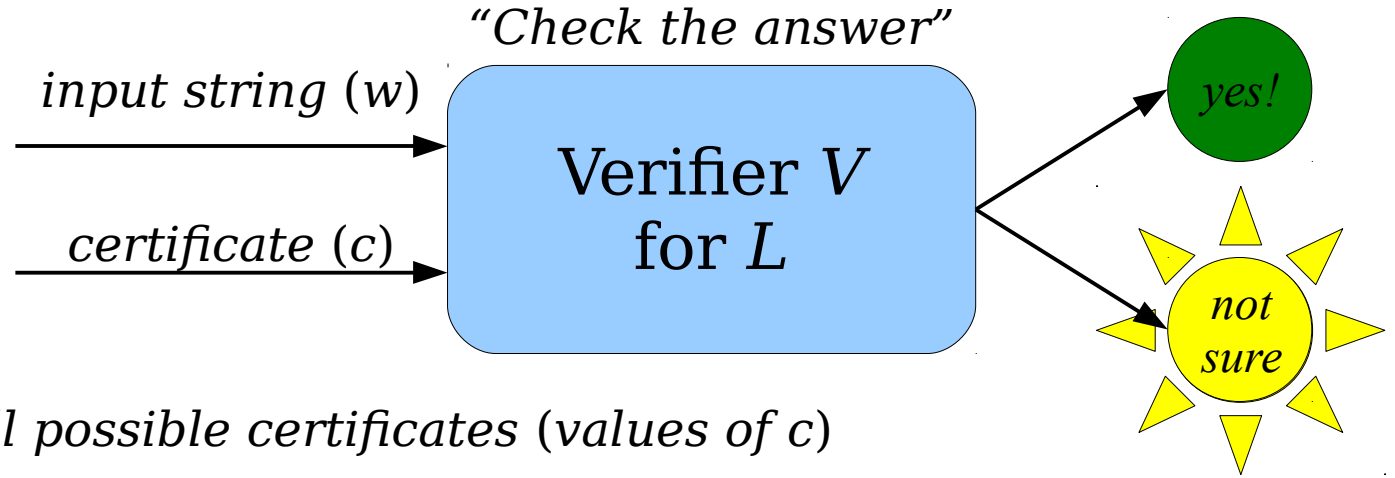


We will try all possible certificates (values of c)

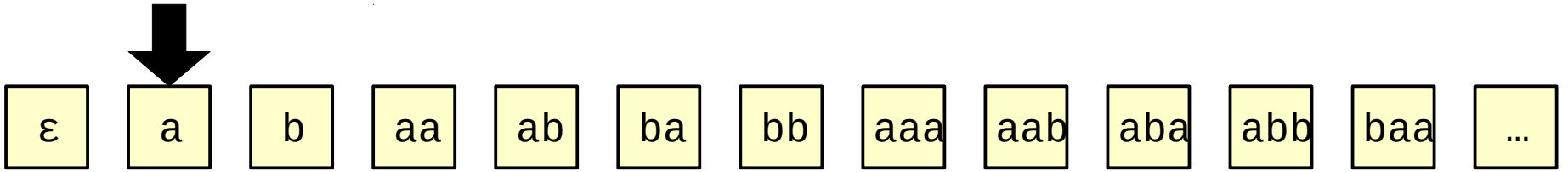


Verifiers and **RE**

- ***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.

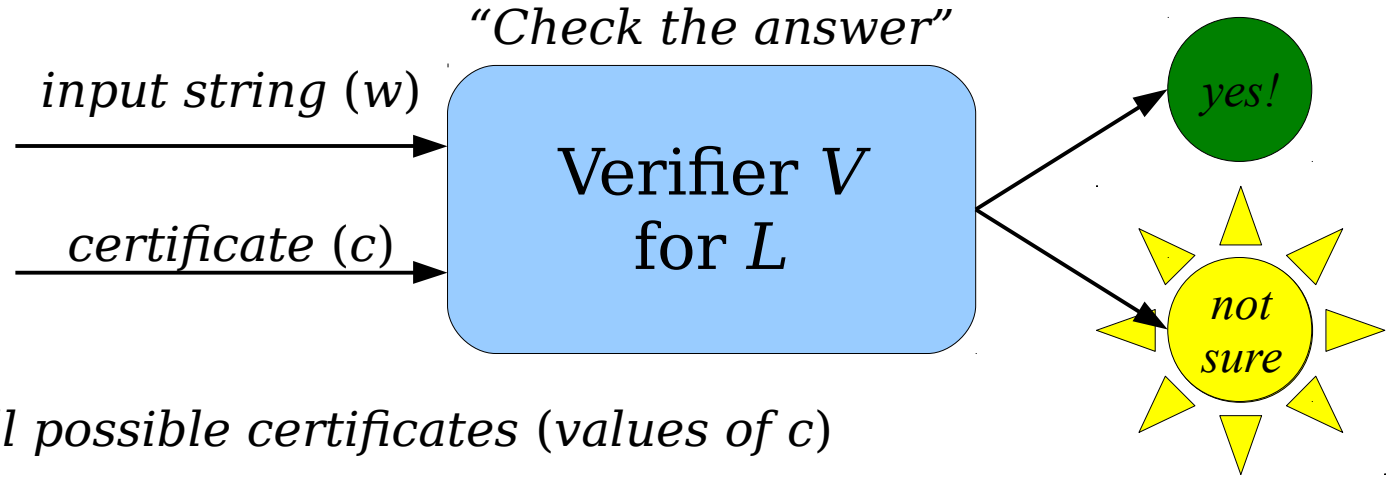


We will try all possible certificates (values of c)

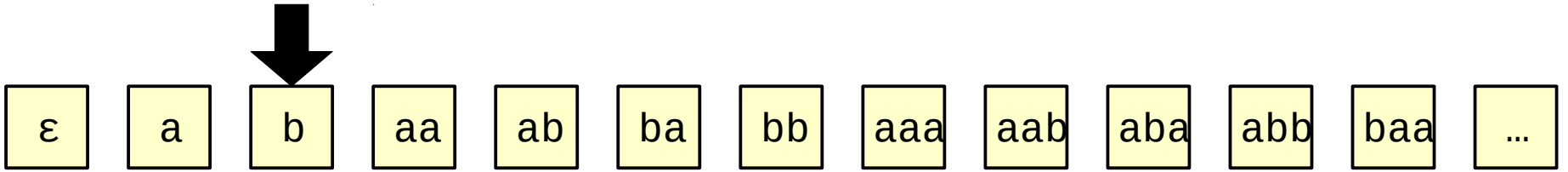


Verifiers and **RE**

- ***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.

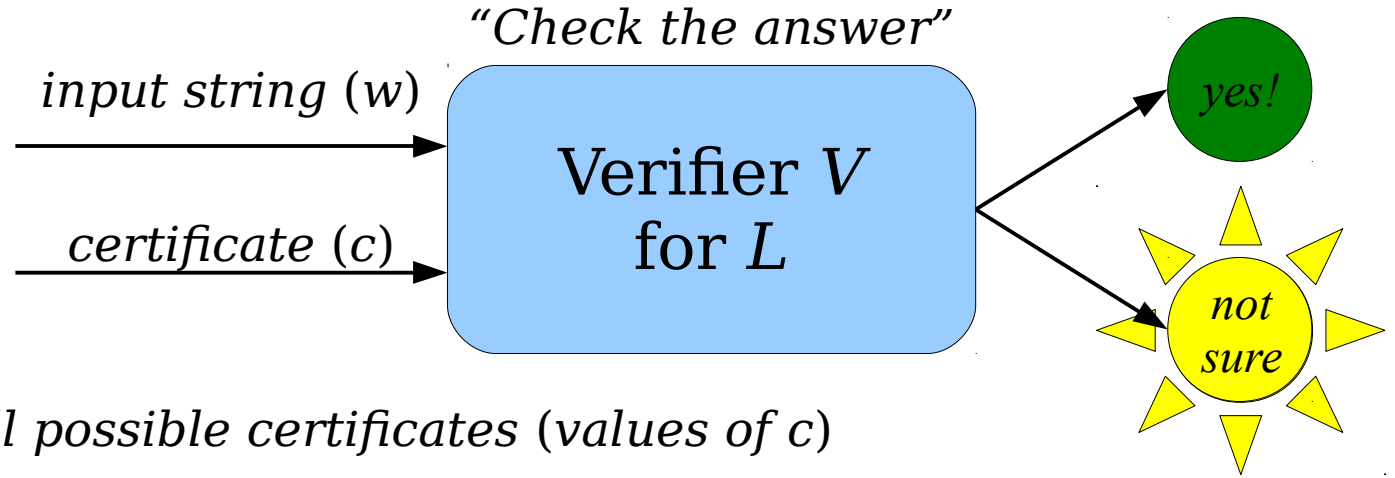


We will try all possible certificates (values of c)

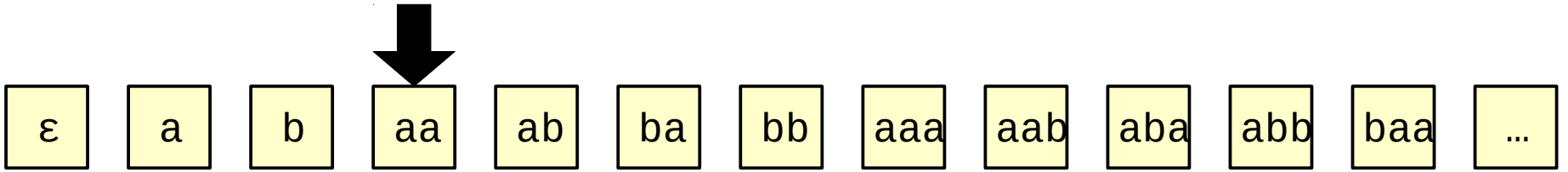


Verifiers and **RE**

- ***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.

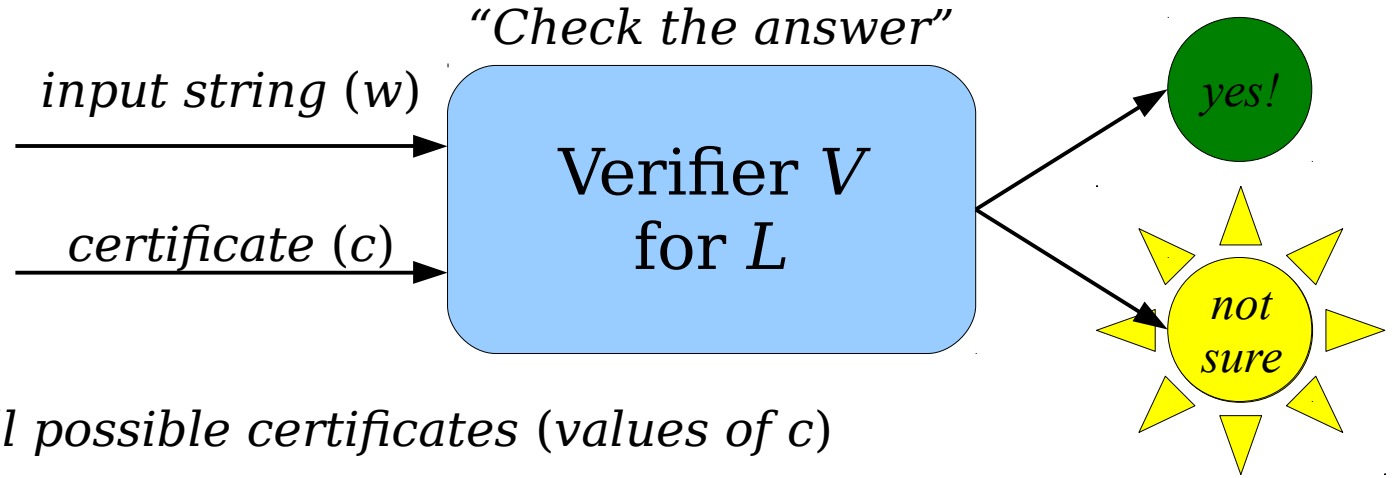


We will try all possible certificates (values of c)

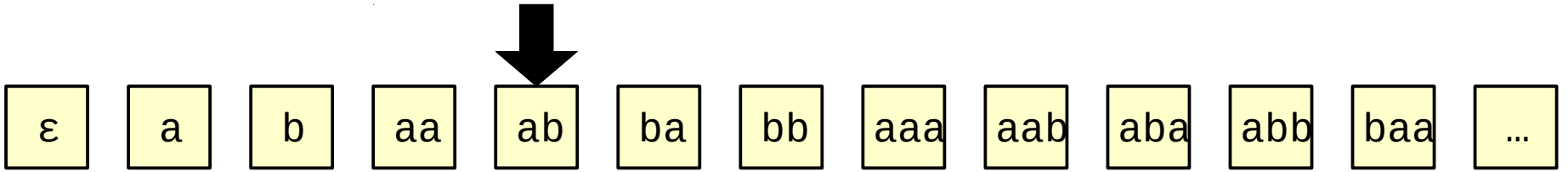


Verifiers and **RE**

- ***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.

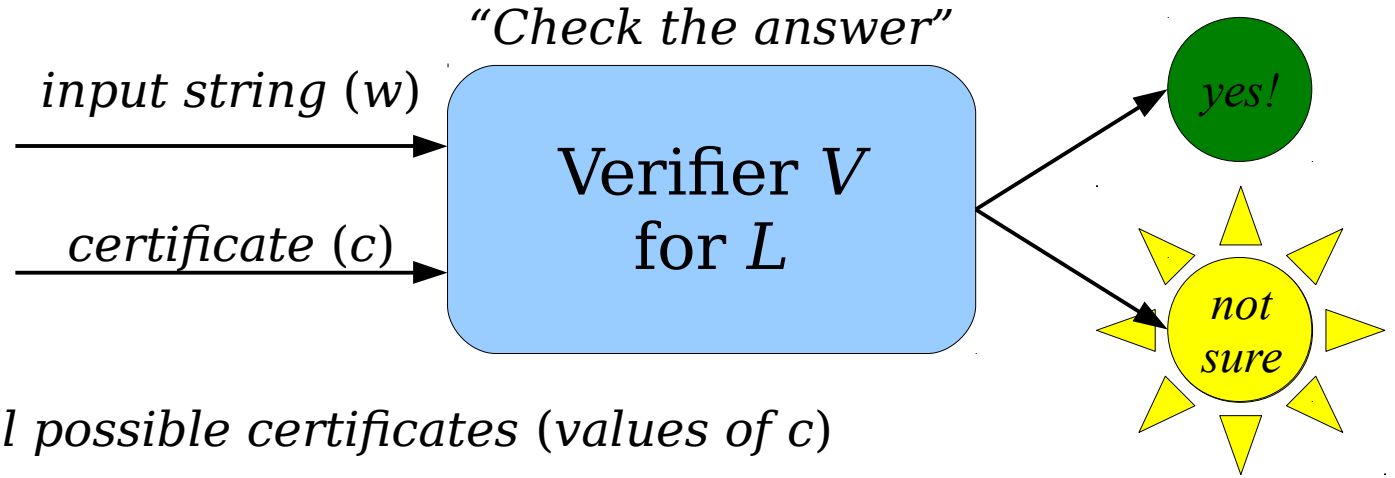


We will try all possible certificates (values of c)

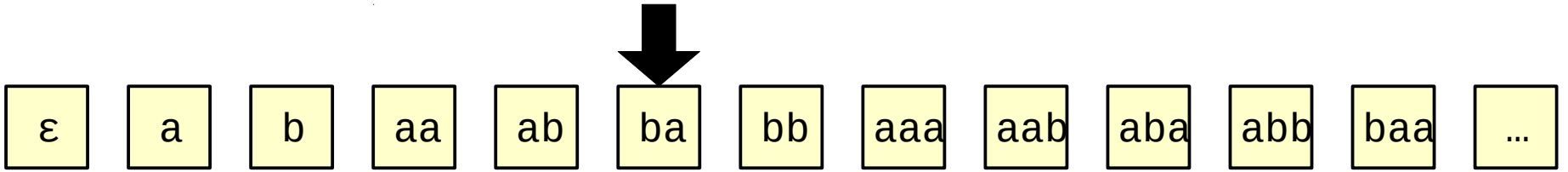


Verifiers and **RE**

- ***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.

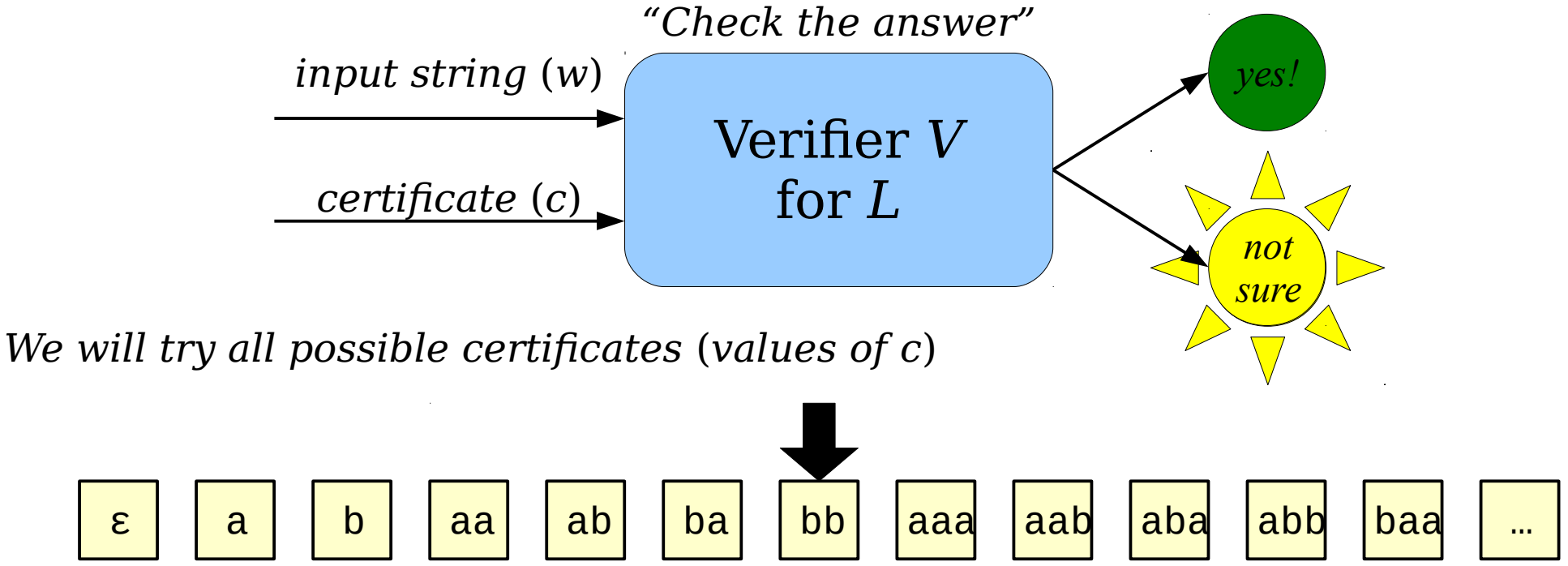


We will try all possible certificates (values of c)



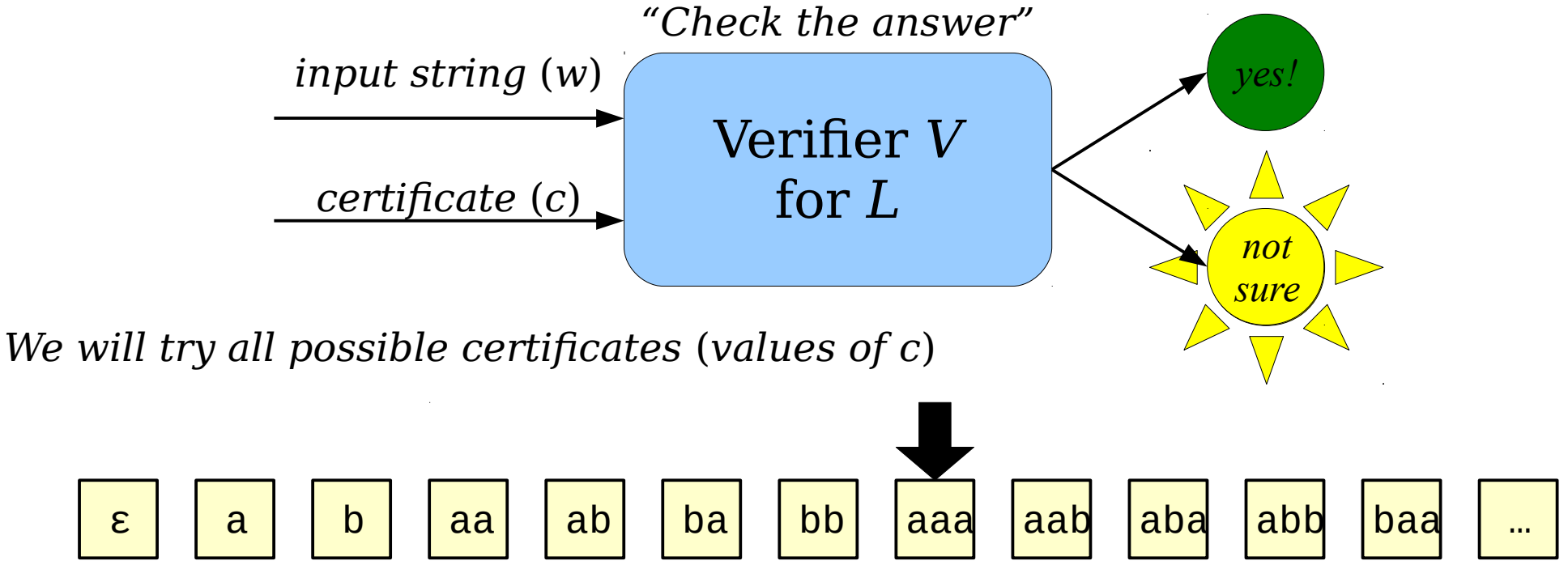
Verifiers and **RE**

- ***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.



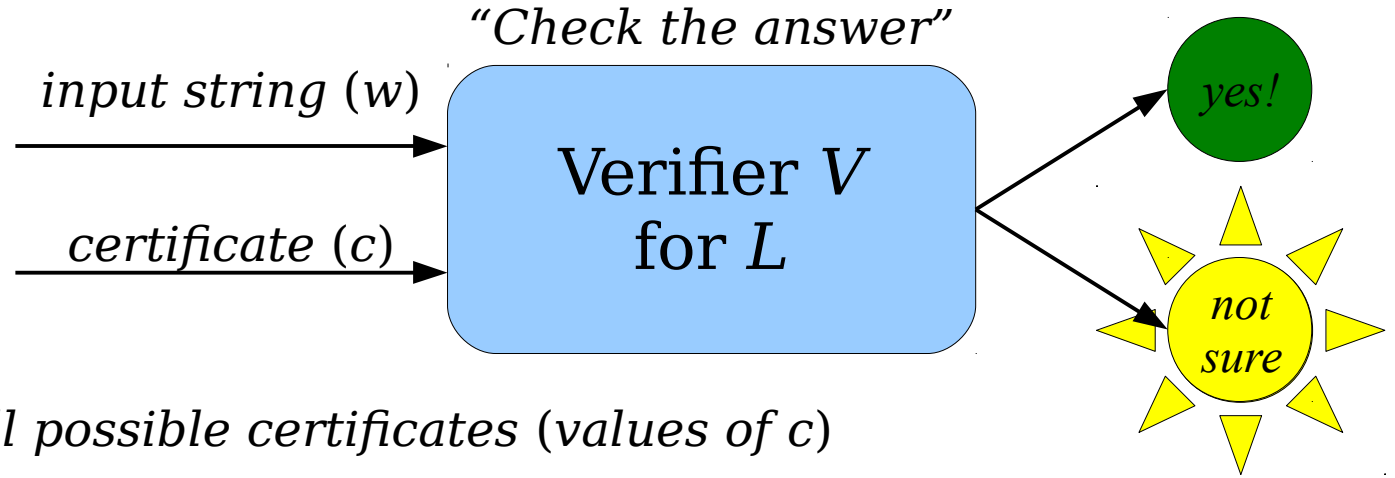
Verifiers and **RE**

- ***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.

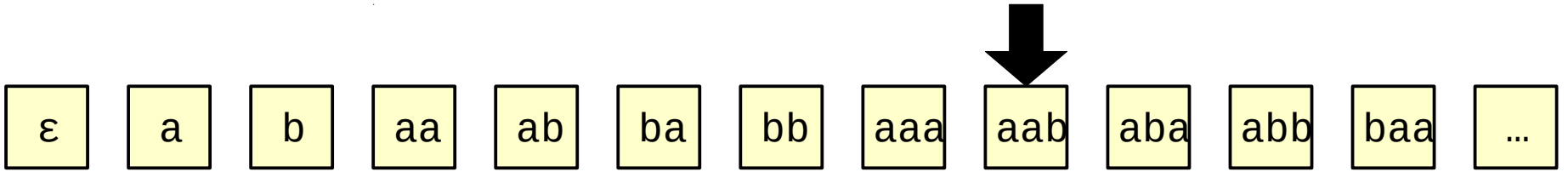


Verifiers and **RE**

- ***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.

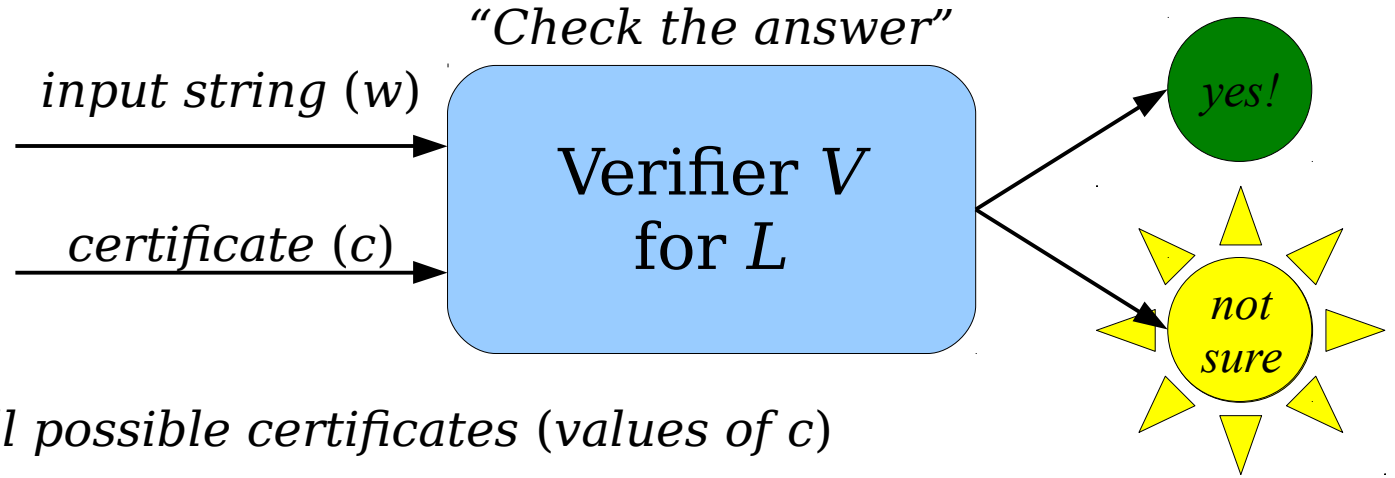


We will try all possible certificates (values of c)

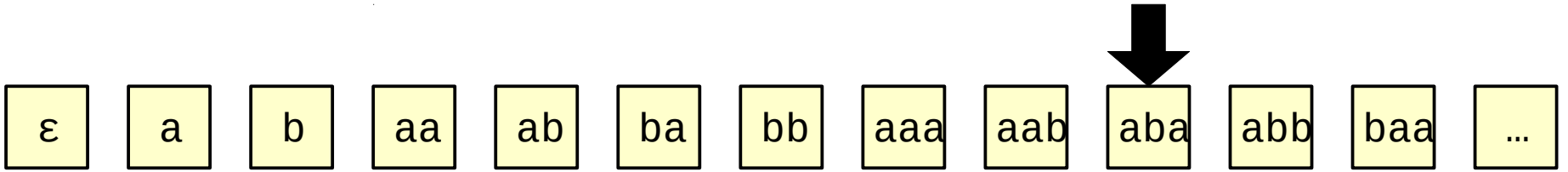


Verifiers and **RE**

- ***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.

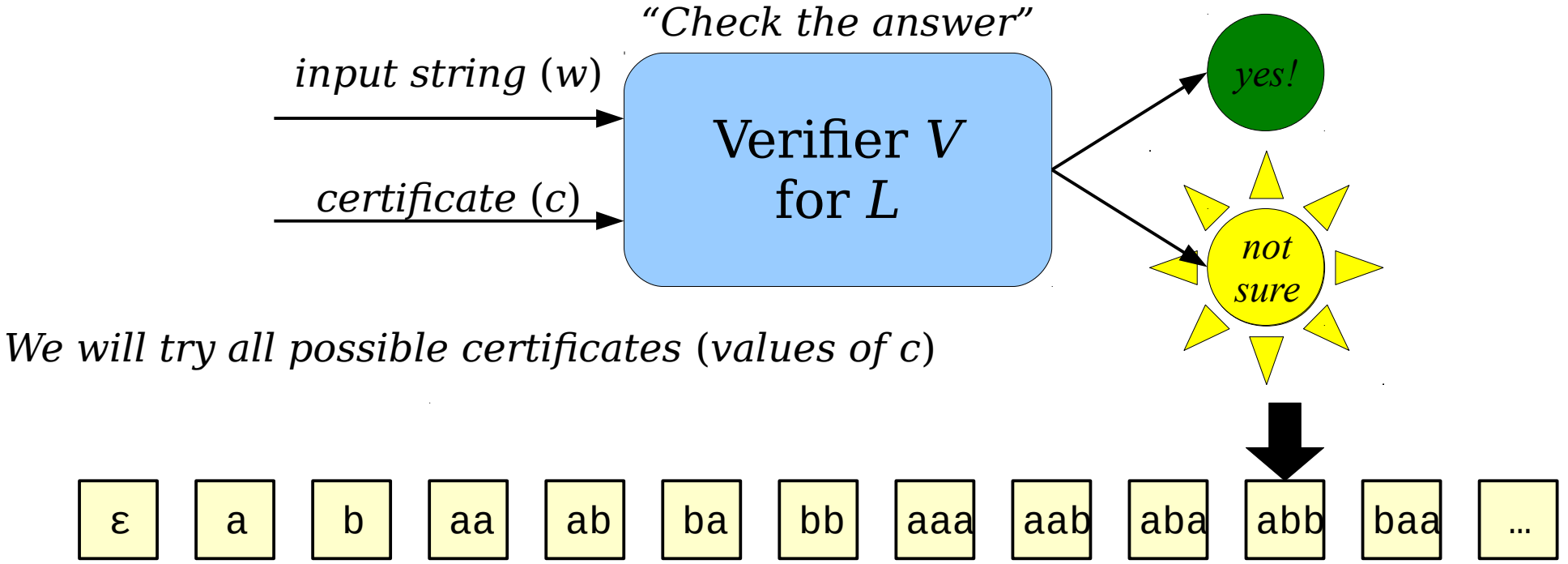


We will try all possible certificates (values of c)



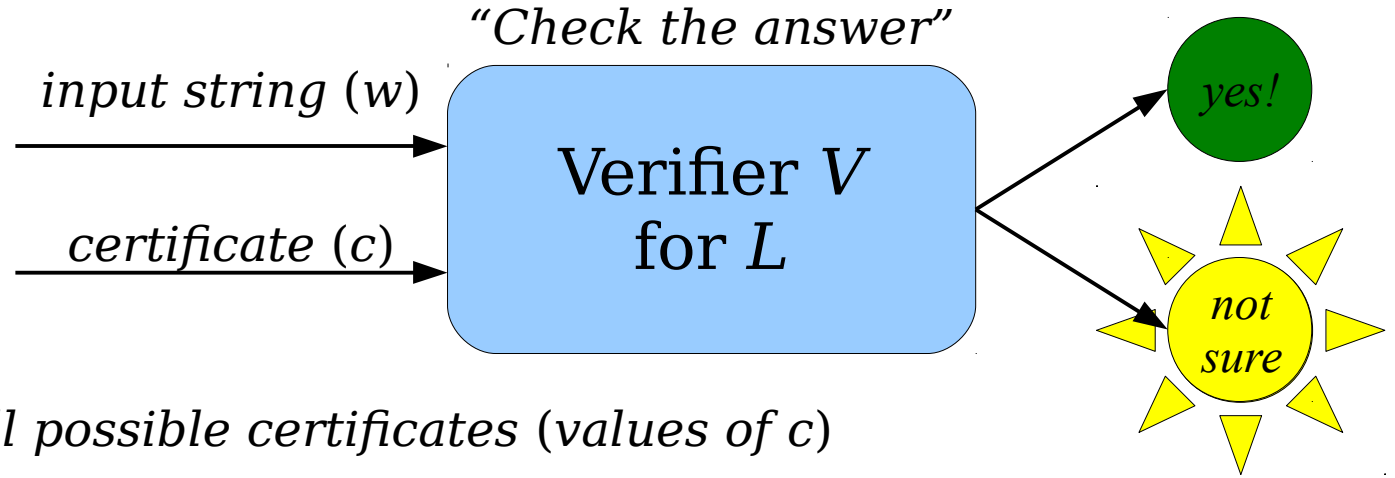
Verifiers and **RE**

- ***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.

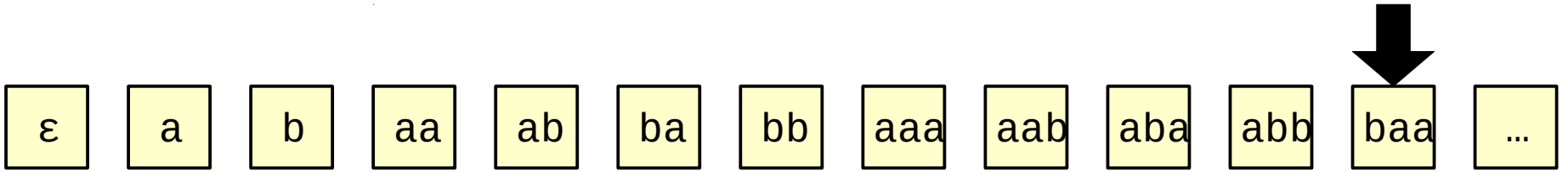


Verifiers and **RE**

- ***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.

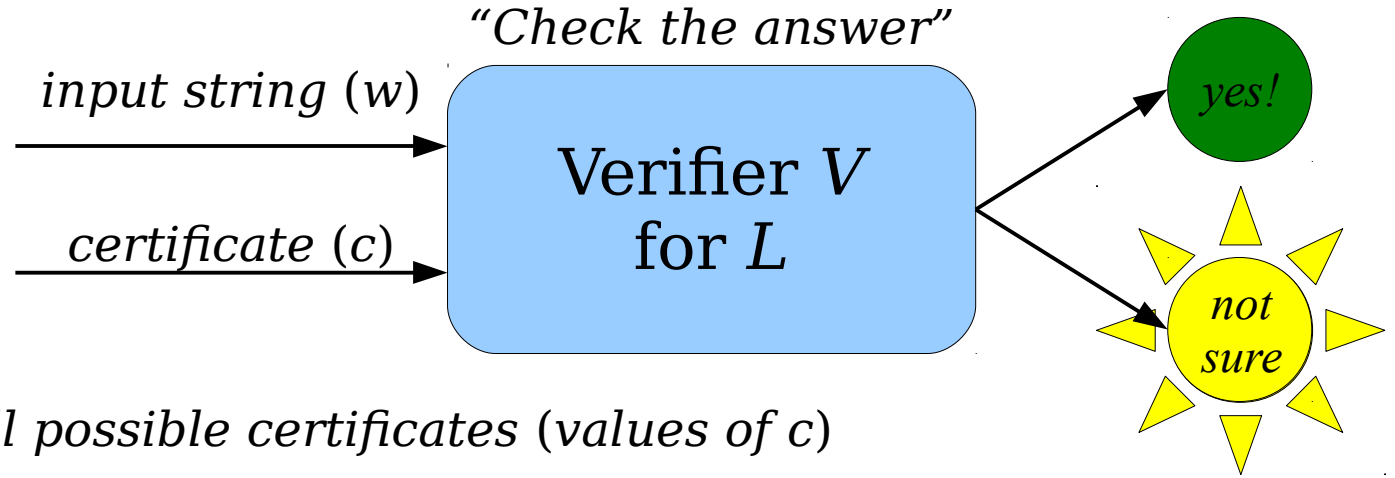


We will try all possible certificates (values of c)

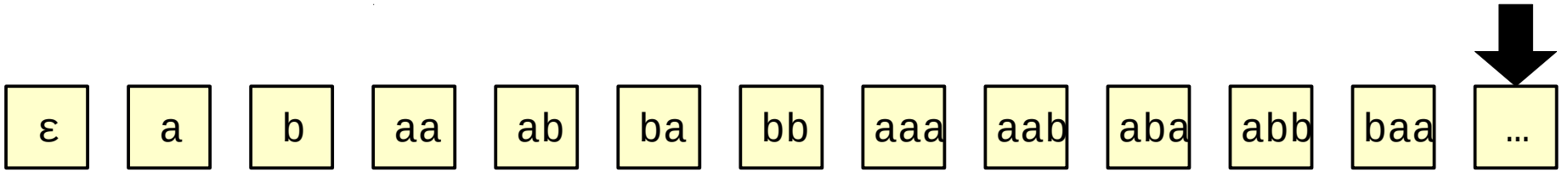


Verifiers and **RE**

- ***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.



We will try all possible certificates (values of c)



Verifiers and **RE**

- ***Theorem:*** If V is a verifier for L, then $L \in \mathbf{RE}$.
- ***Proof sketch:*** Consider the following program:

```
bool isInL(string w) {
    int i = 0;
    while (true) {
        for (each string c of length i) {
            if (V accepts ⟨w, c⟩) return true;
        }
        i++;
    }
}
```

If $w \in L$, there is some $c \in \Sigma^*$ where V accepts $\langle w, c \rangle$. The function `isInL` tries all possible strings as certificate, so it will eventually find c (or some other certificate), see V accept $\langle w, c \rangle$, then return true. Conversely, if `isInL(w)` returns true, then there was some string c such that V accepted $\langle w, c \rangle$, so $w \in L$. ■

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 challenges:
  - A recognizer M is not required to halt on all inputs. A verifier V must always halt.
  - A recognizer M takes in one single input. A verifier V takes in two inputs.
- We'll need to find a way of reconciling these requirements.

Recall: If M is a recognizer for a language L, then M accepts w iff $w \in L$.

Key insight: If M accepts a string w, it always does so in a finite number of steps.

Idea: Adapt the verifier for A_{TM} into a more general construction that turns any recognizer into a verifier by running it for a fixed number of steps.

Verifiers and **RE**

- ***Theorem:*** If $L \in$ **RE**, then there is a verifier for L.
- ***Proof sketch:*** Consider the following program:

```
bool checkIsInL(string w, int c) {
    set up a simulation of M running on w;
    for (int i = 0; i < c; i++) {
        simulate the next step of M running on W;
    }
    return whether M is in an accepting state;
}
```

Notice that `checkIsInL` always halts, since each step takes only finite time to complete. Next, notice that if there is a c where `checkIsInL(w, c)` returns true, then M accepted w after running for c steps, so $w \in L$. Conversely, if $w \in L$, then M accepts w after some number of steps (call that number c). Then `checkIsInL(w, c)` will run M on w for c steps, watch M accept w, then return true. ■

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!