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 new approach to problem-solving.
- ***Beyond RE***
  - Monstrously hard problems!

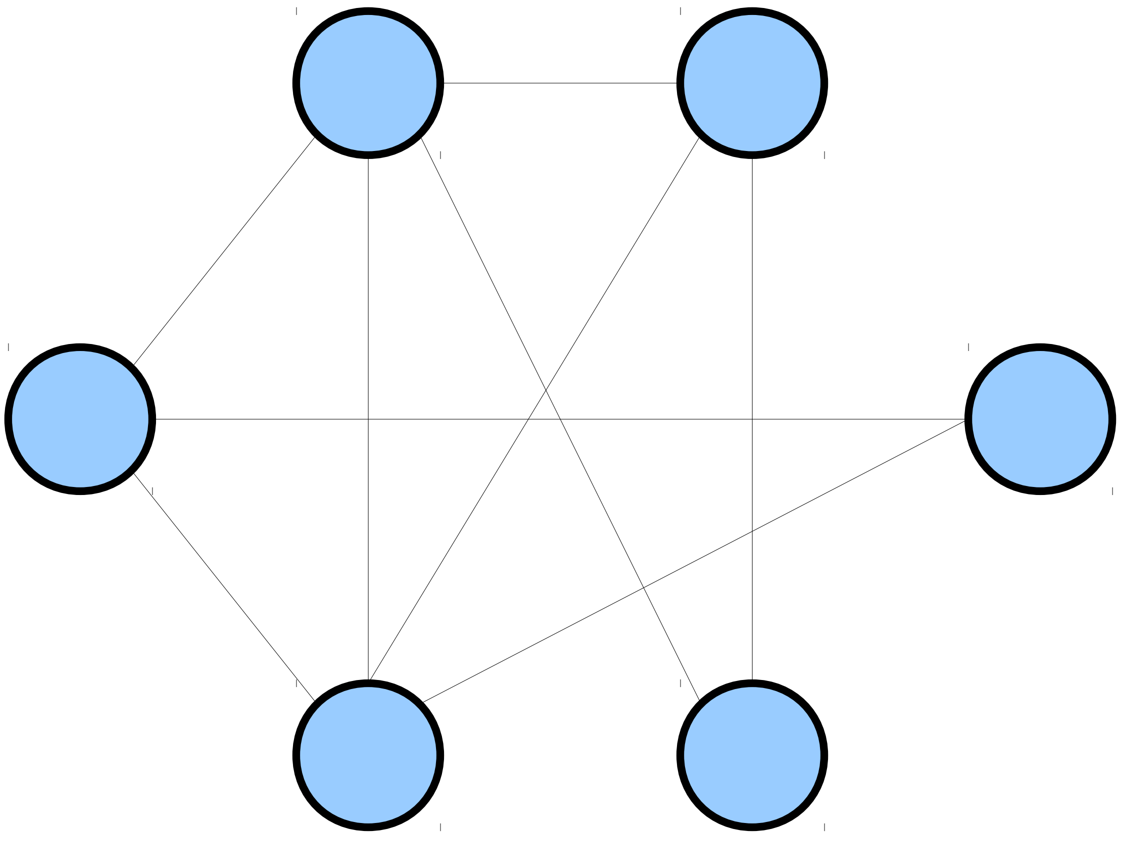
Verifiers

- 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:

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

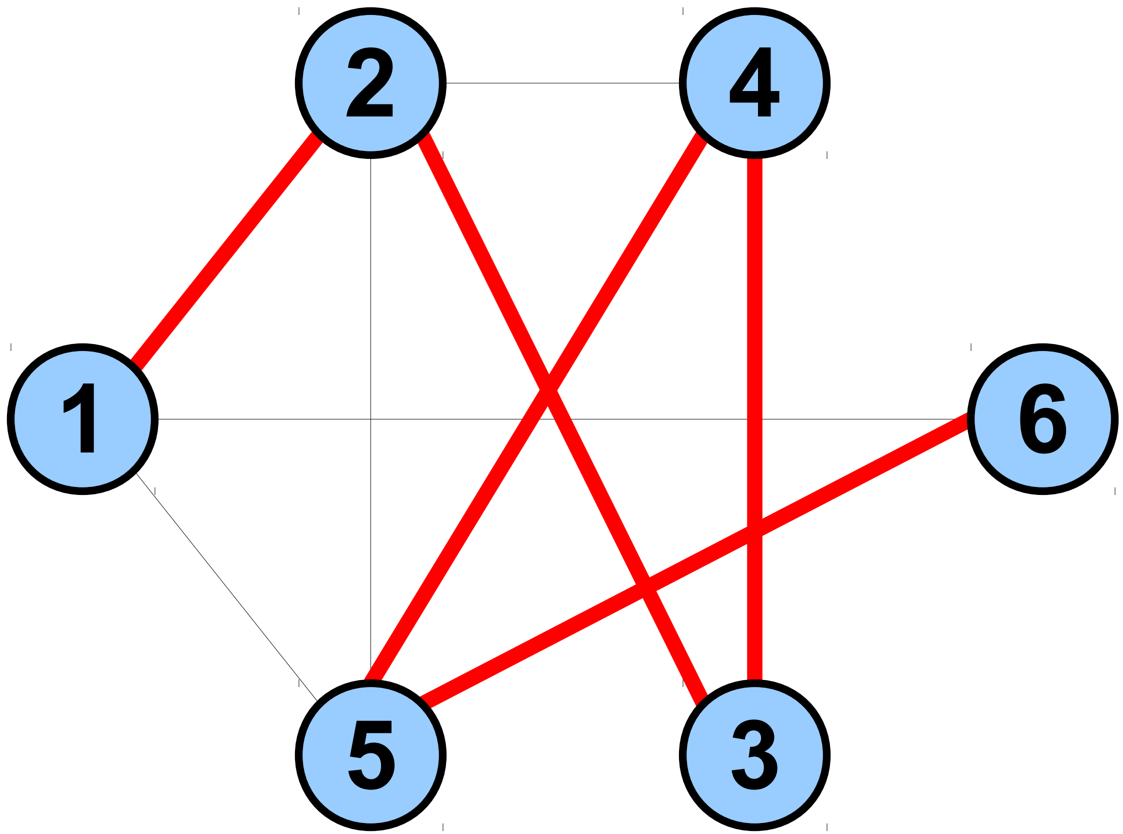
- A string c where V accepts $\langle w, c \rangle$ is called a ***certificate*** for w.
- Intuitively, what does this mean?

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?

Some Verifiers

- 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;
}
```

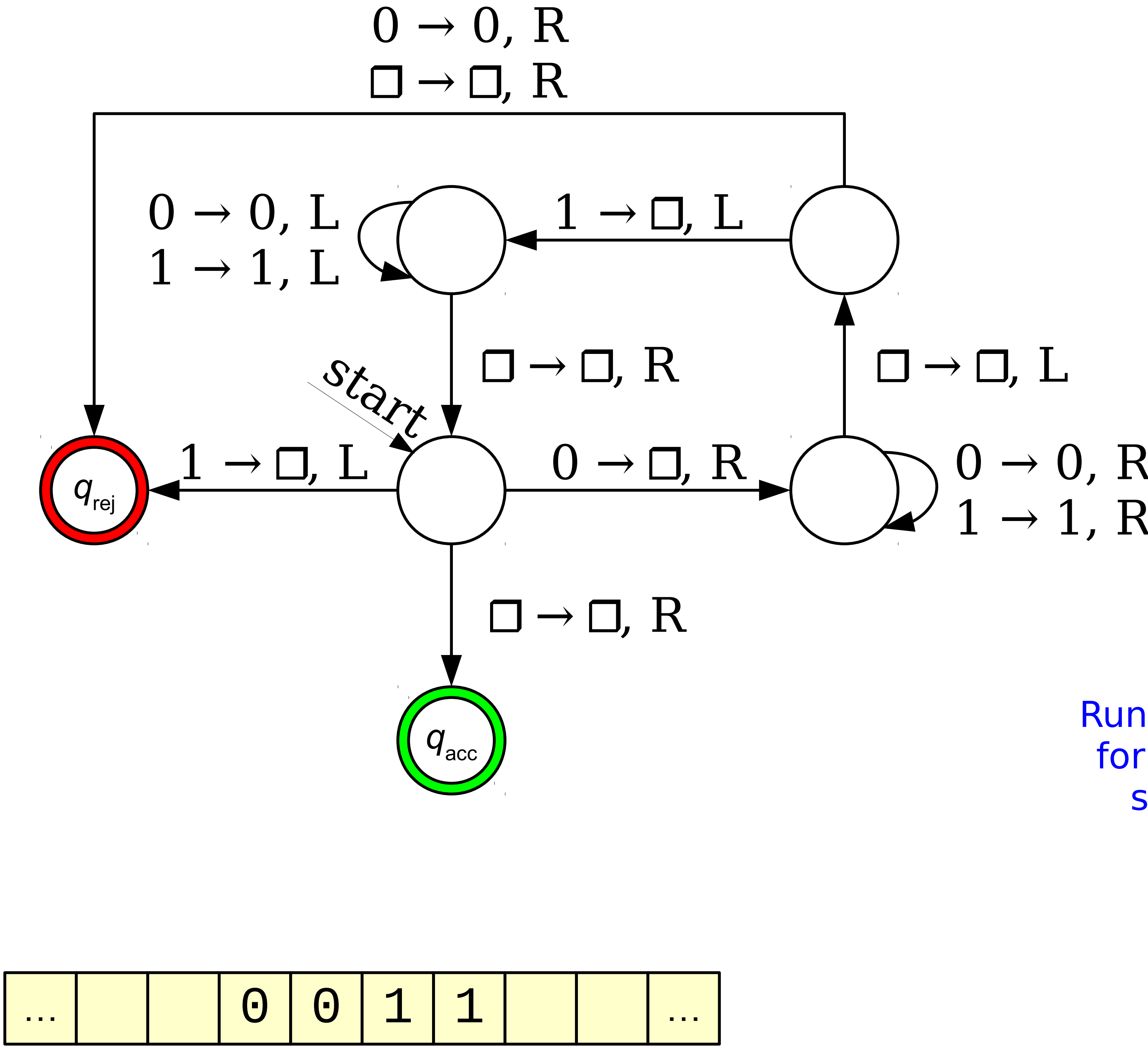
- Do you see why $\langle G \rangle \in L$ iff there is a c where `checkHamiltonian(G, c)` returns true?
- Do you see why `checkHamiltonian` always halts?

Some Verifiers

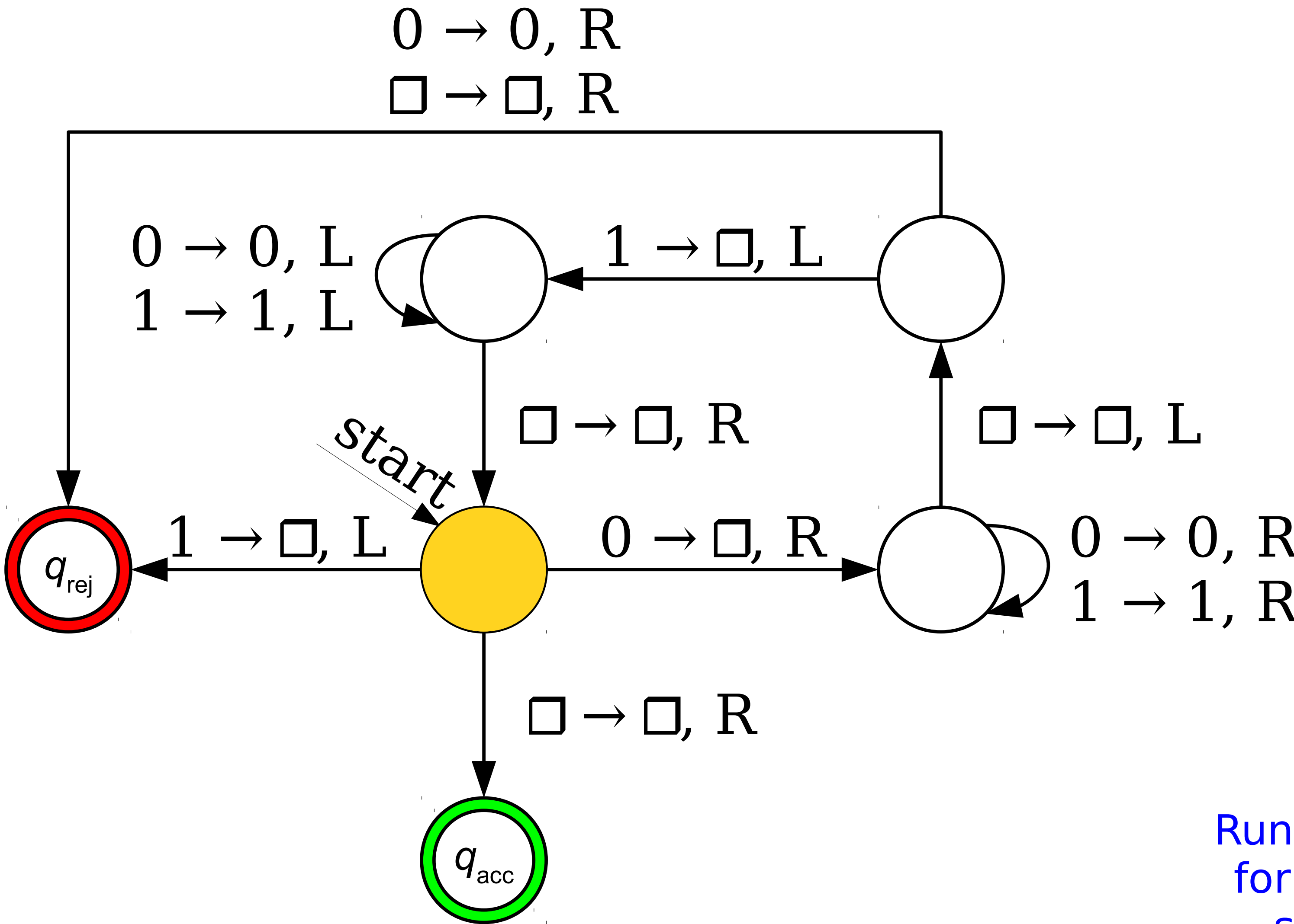
- Consider A_{TM}:

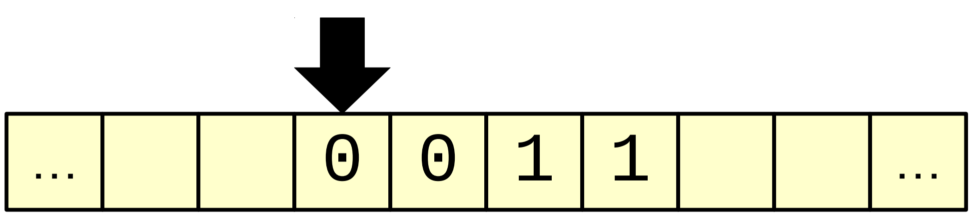
  $A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}.$

- 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!

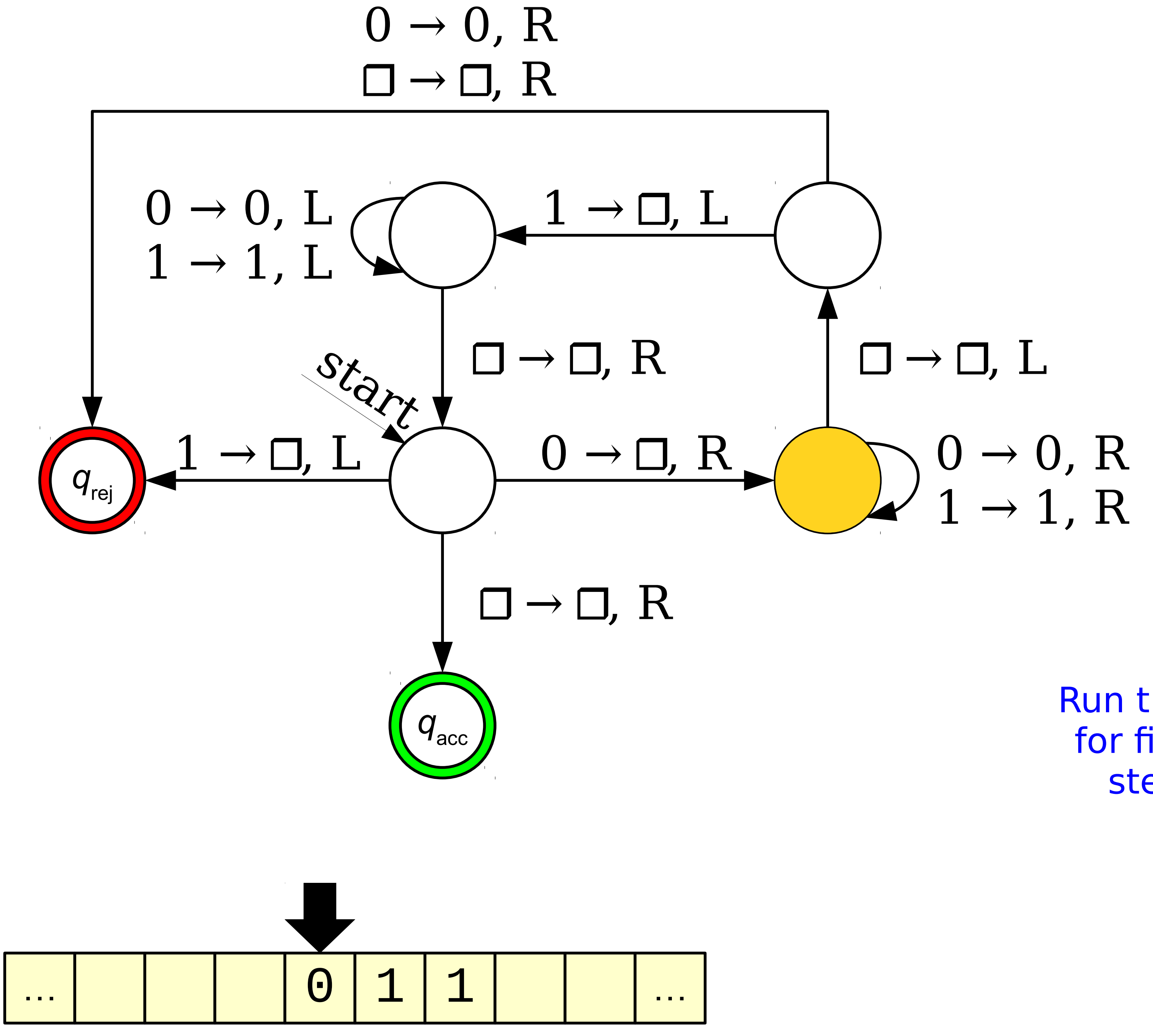


Run this TM for fifteen steps.

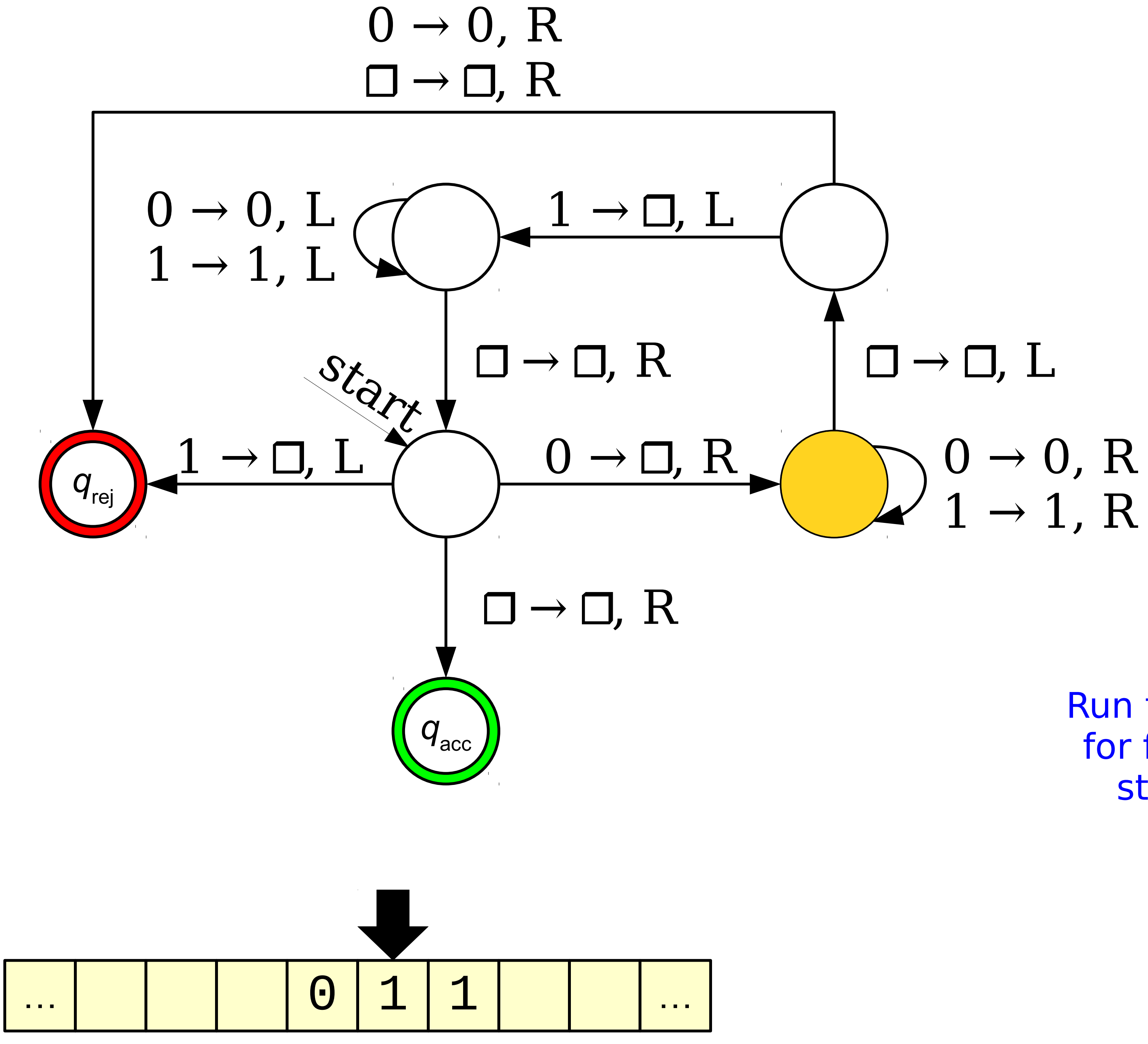




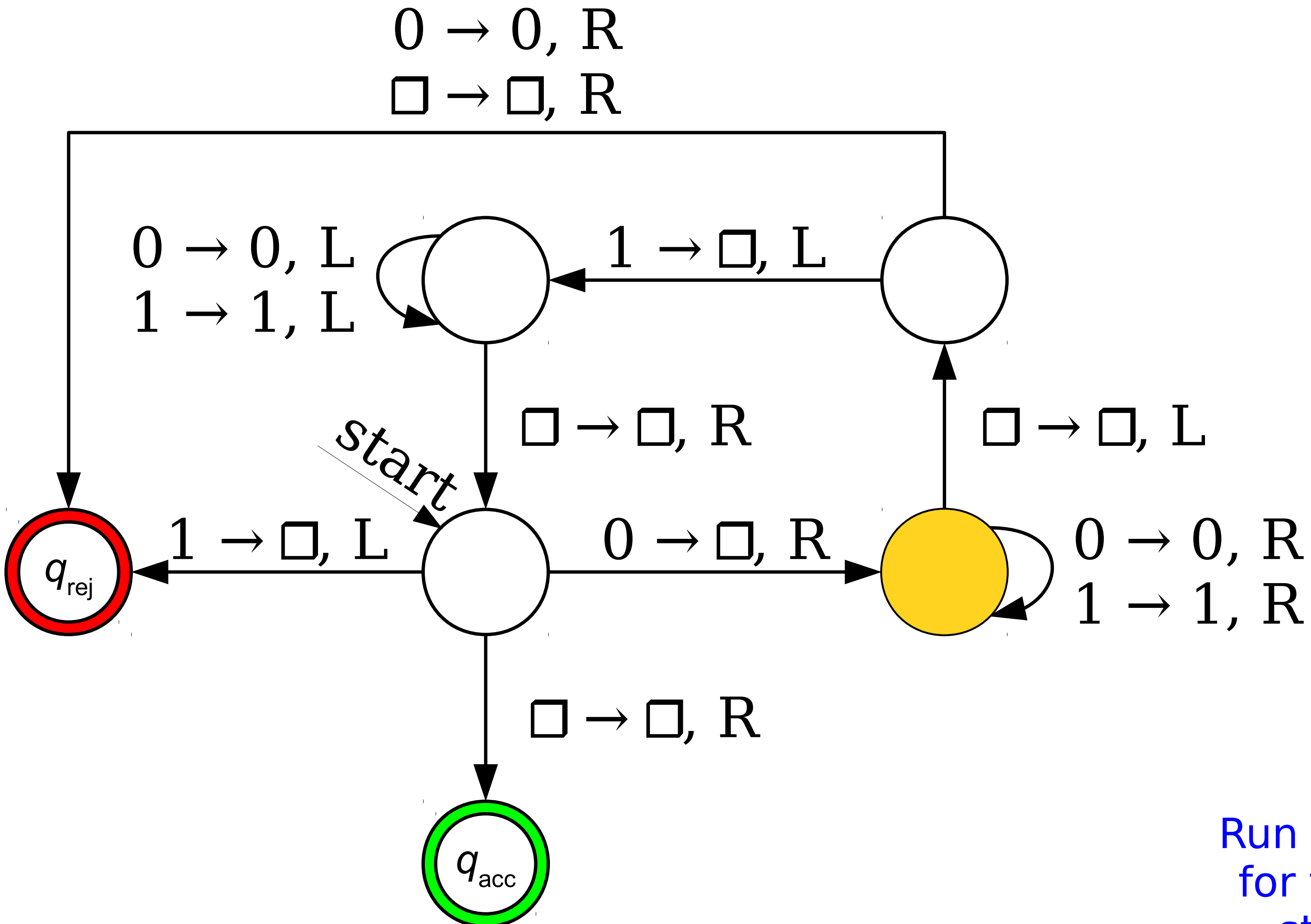
Run this TM for fifteen steps.



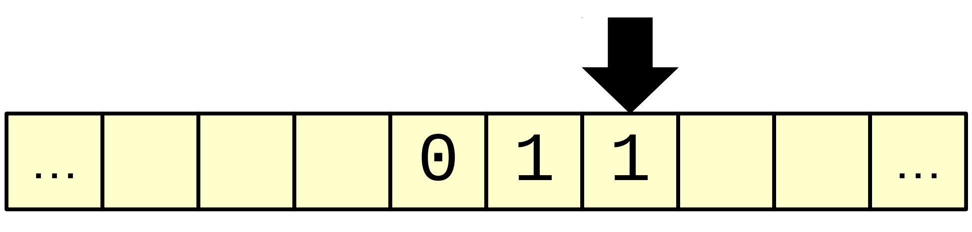
Run this TM
for fifteen
steps.

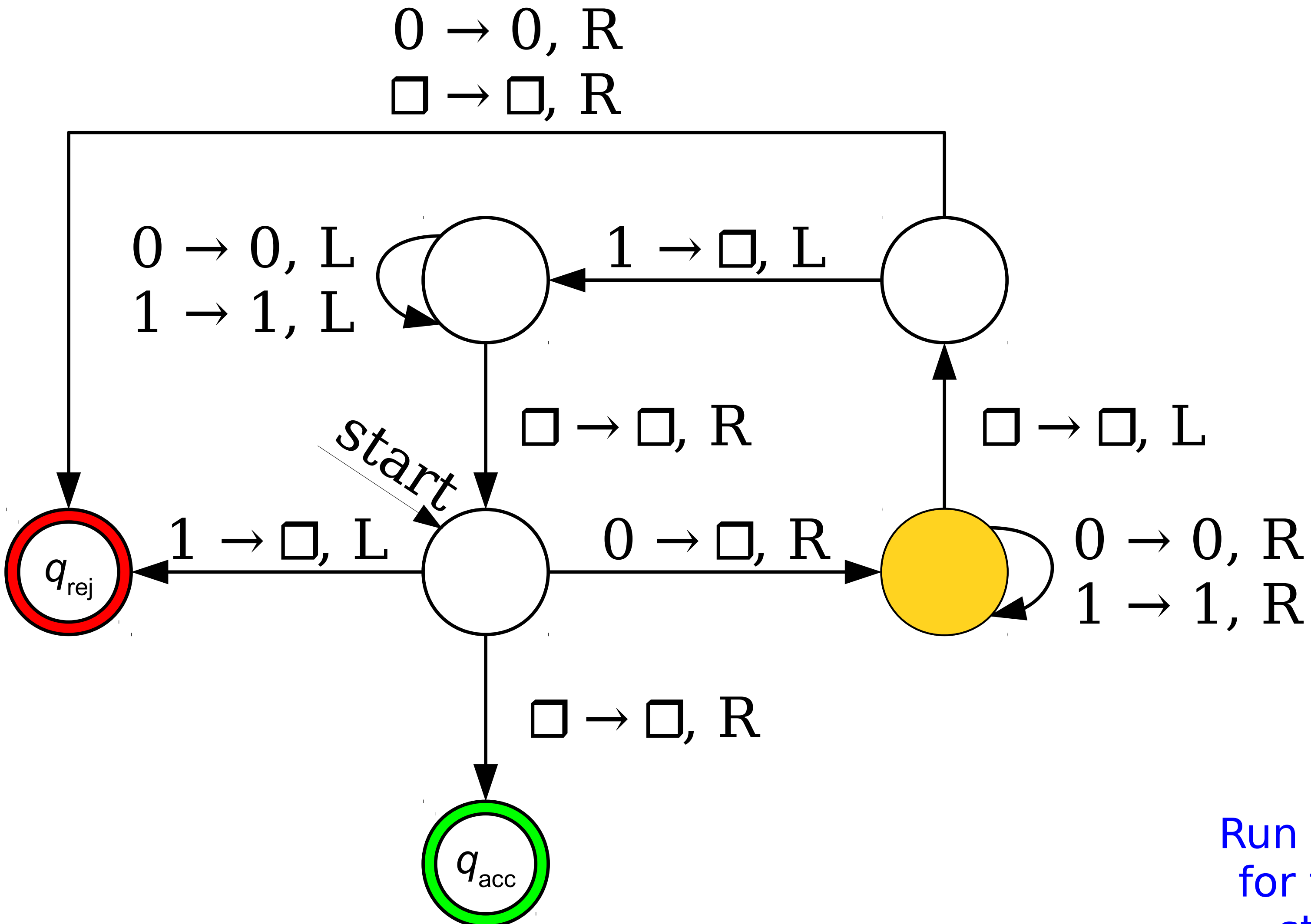


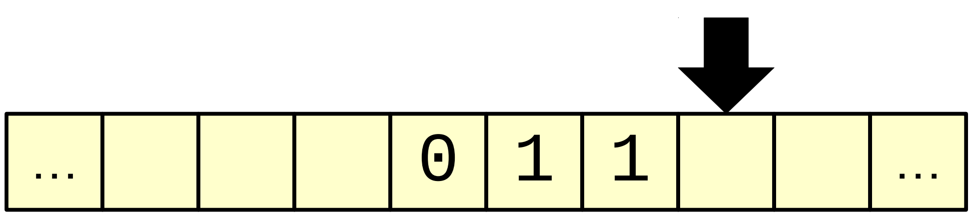
Run this TM for fifteen steps.



Run this TM
for fifteen
steps.

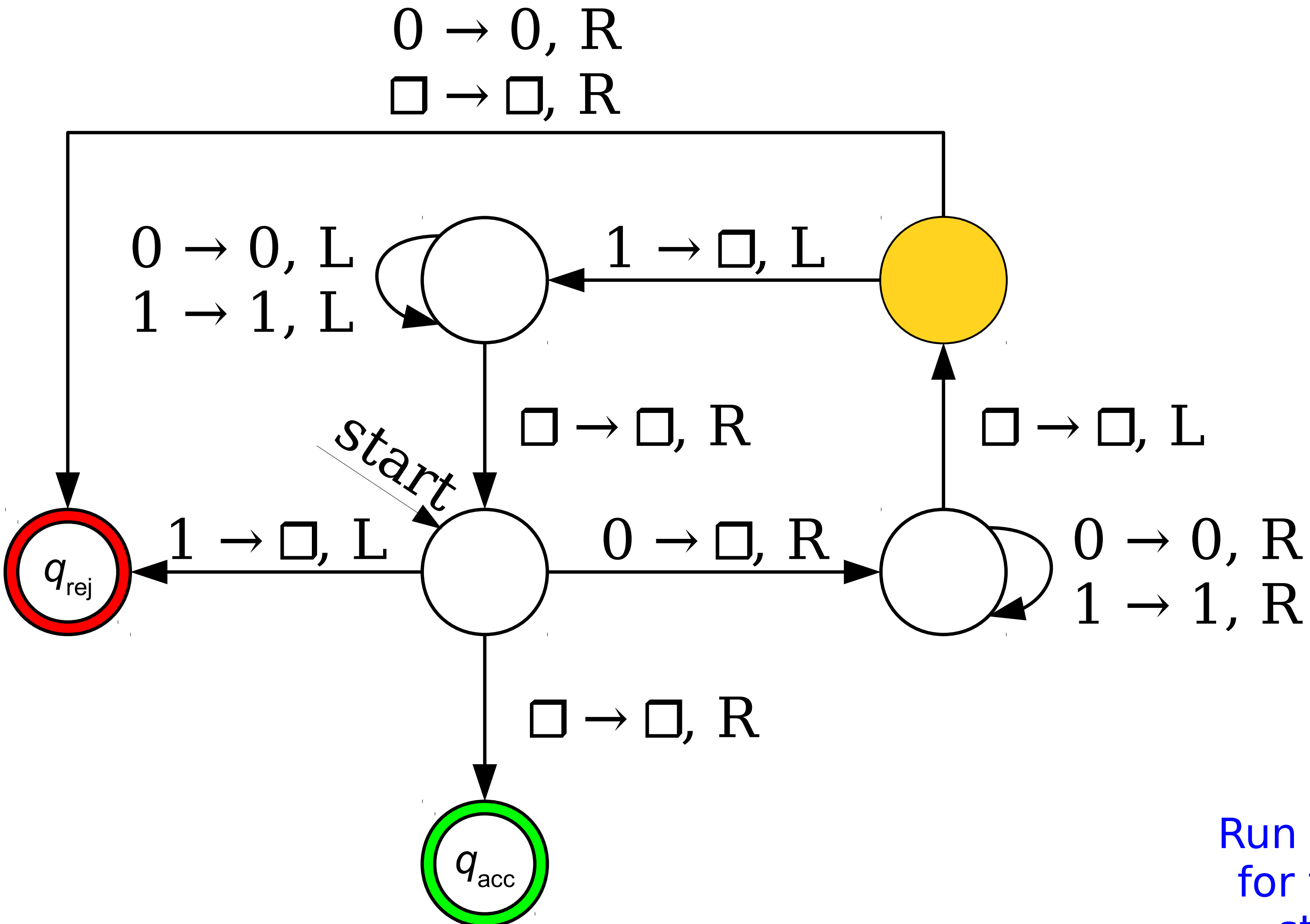


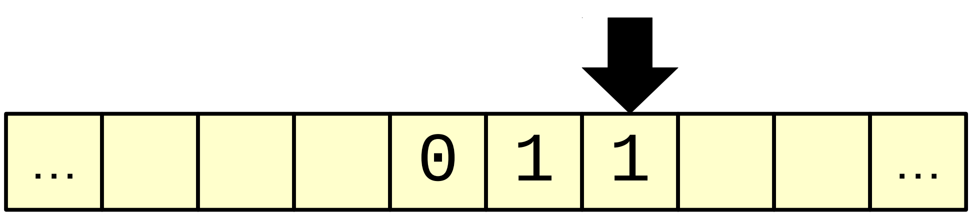


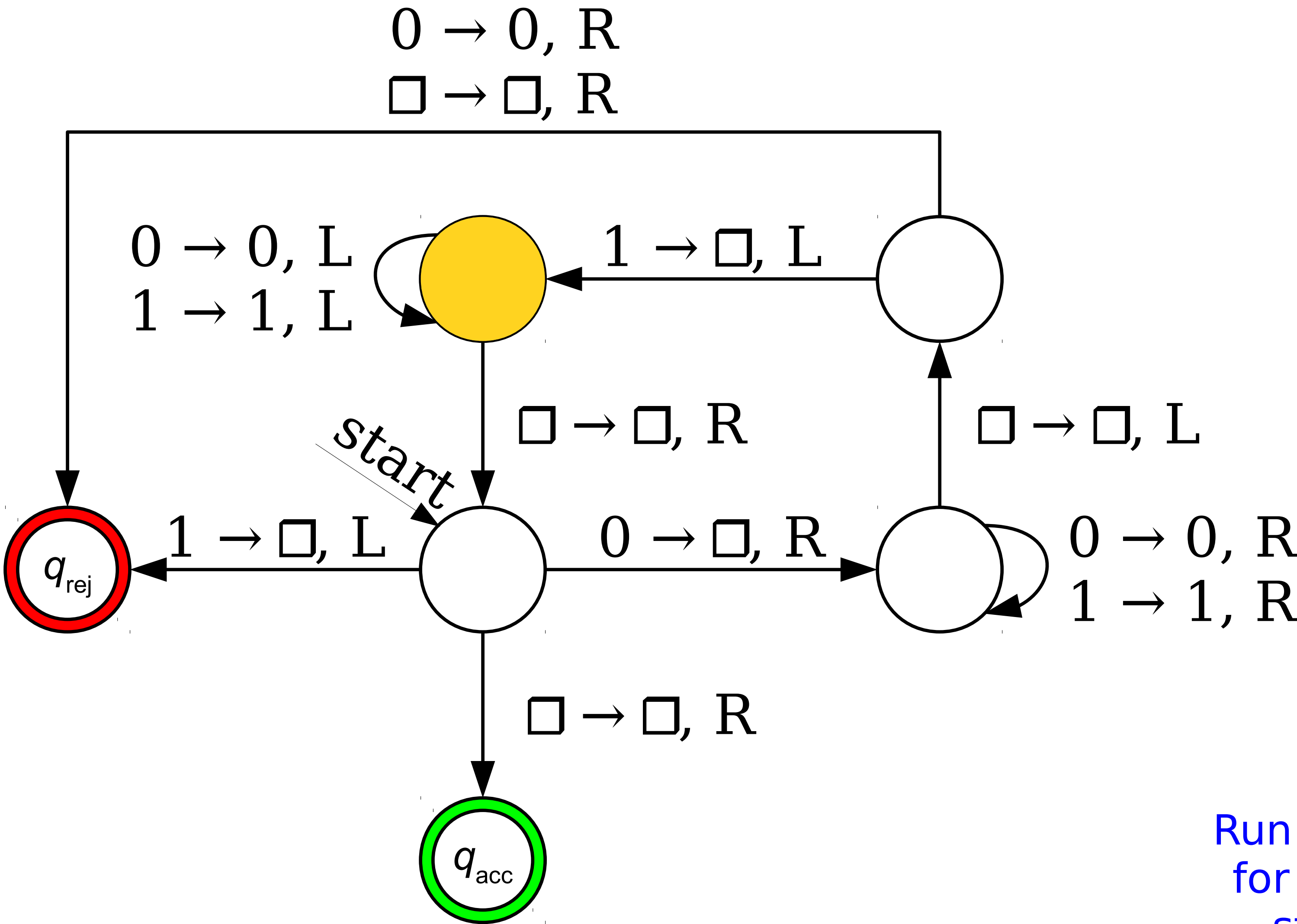


Run this TM
for fifteen
steps.

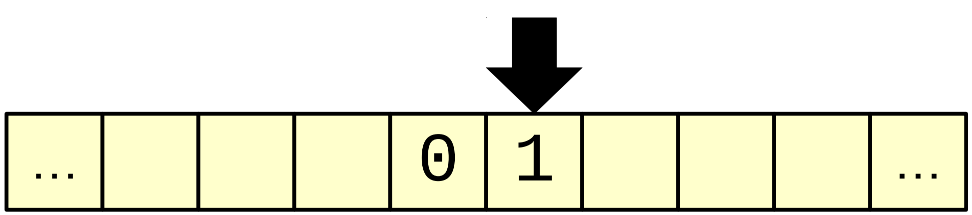
Run this TM for fifteen steps.

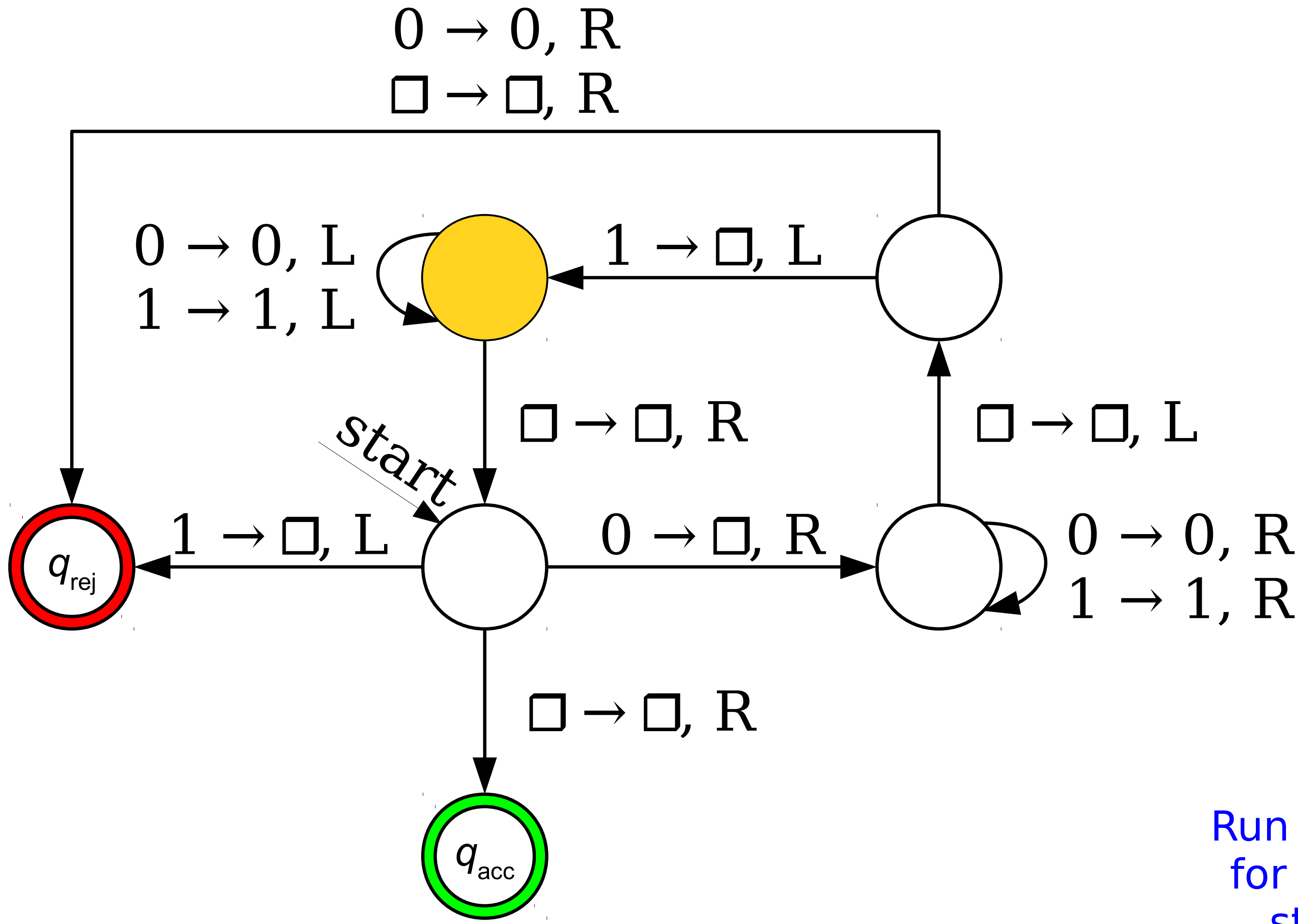




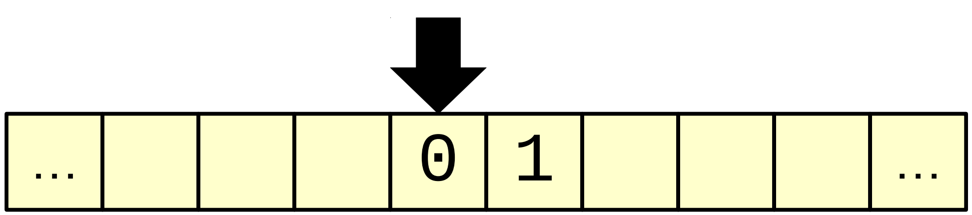


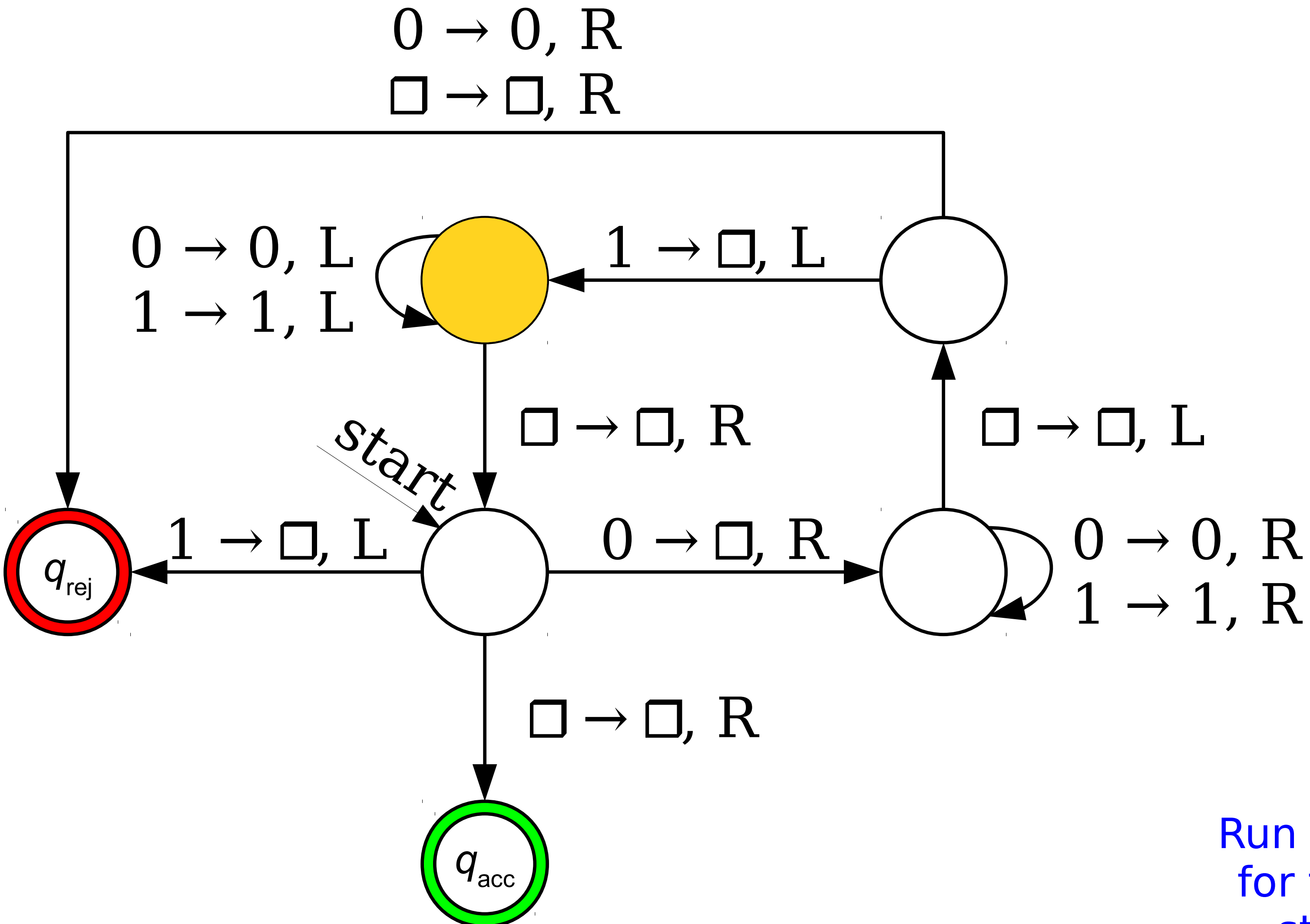
Run this TM for fifteen steps.



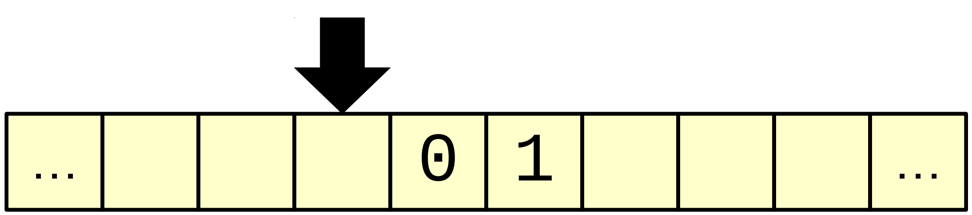


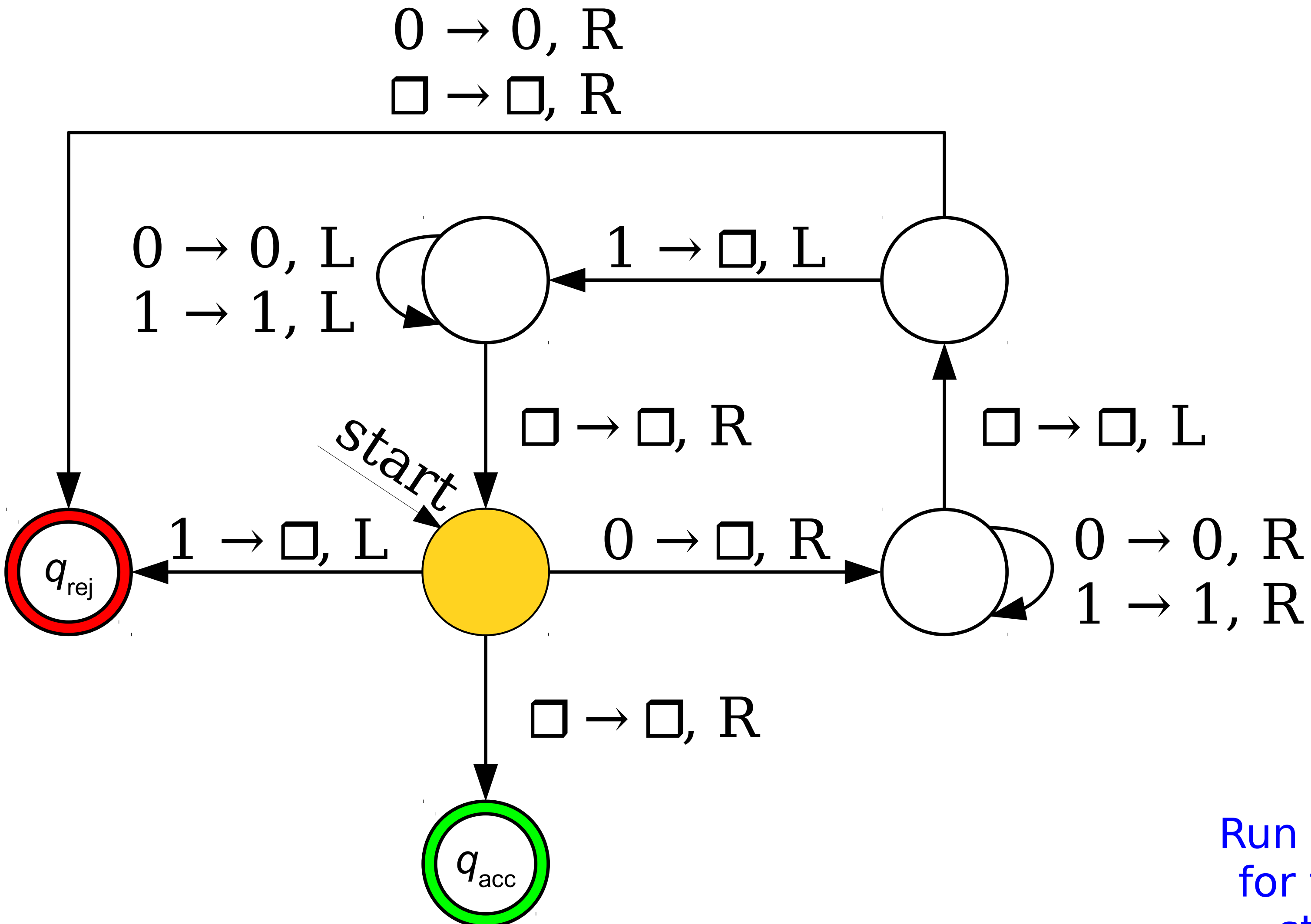
Run this TM for fifteen steps.



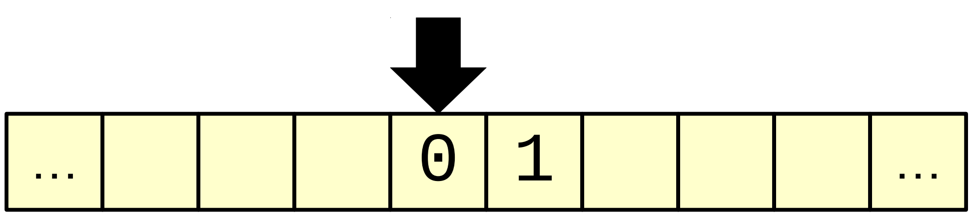


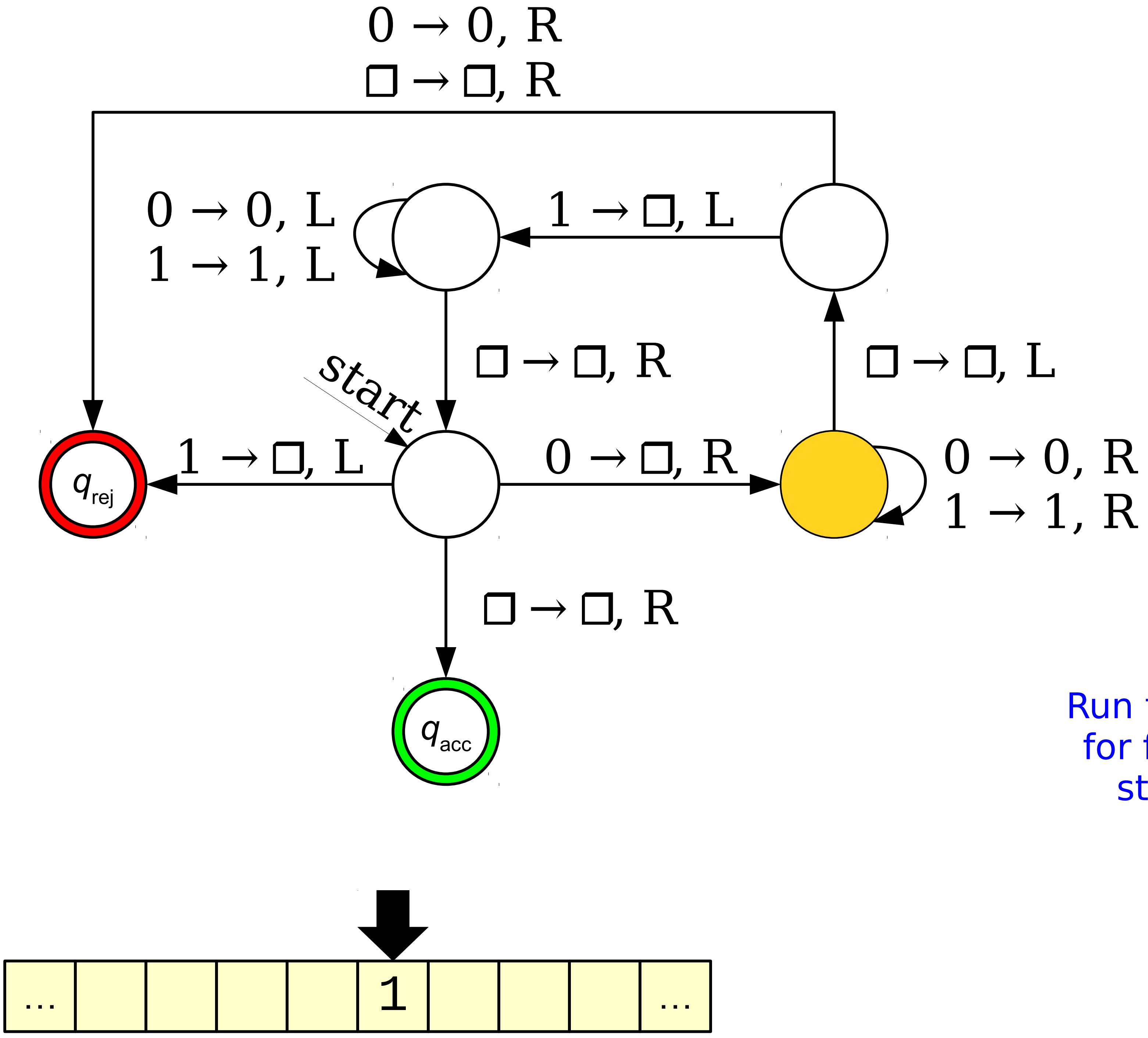
Run this TM for fifteen steps.



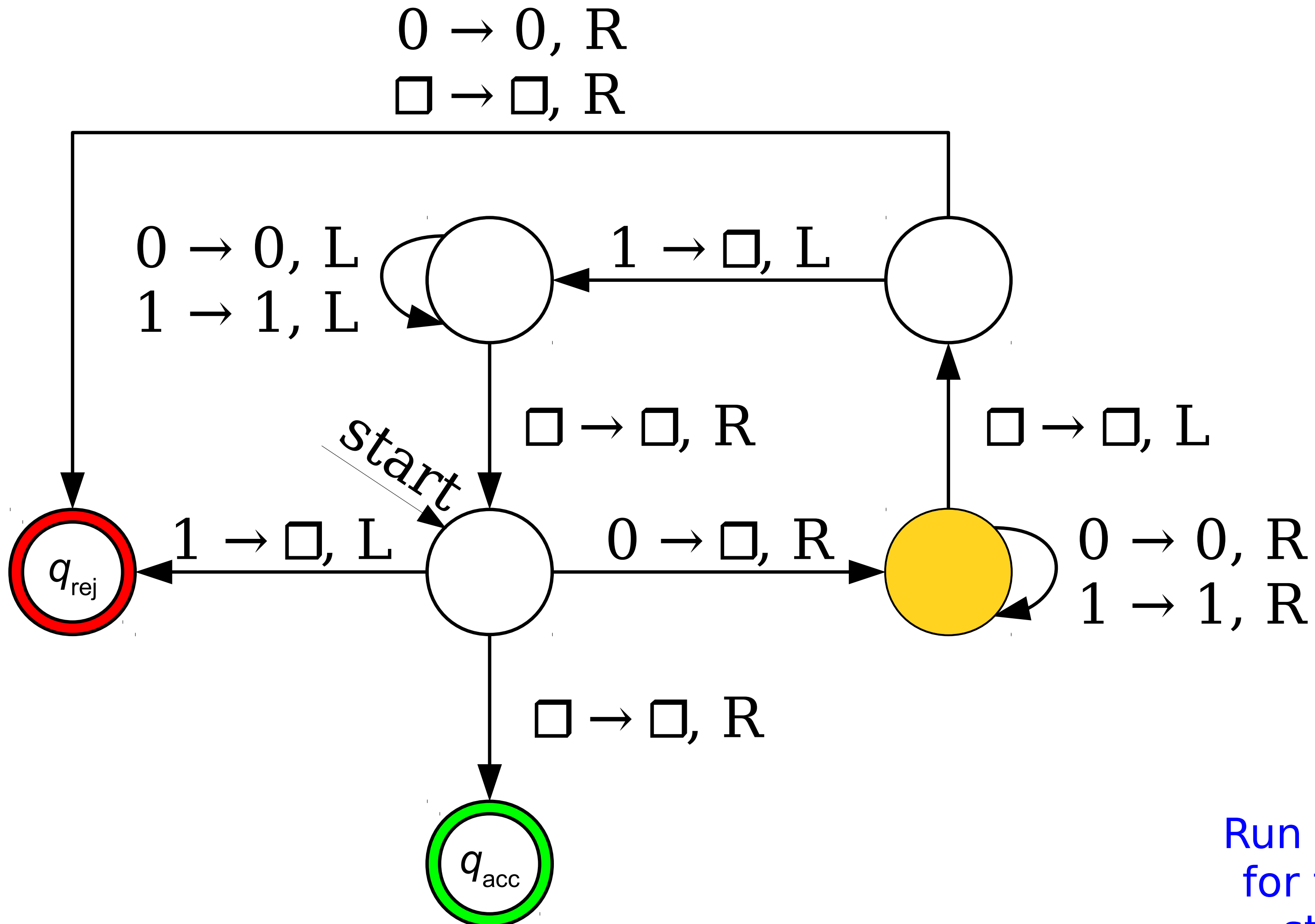


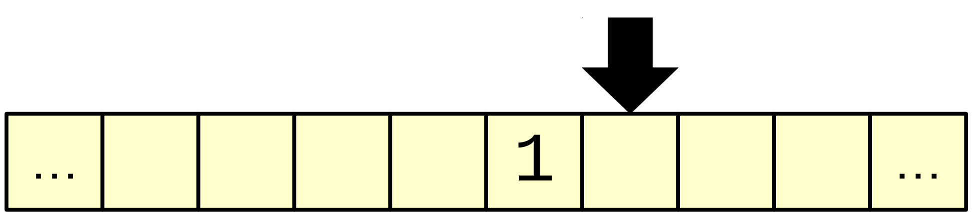
Run this TM for fifteen steps.



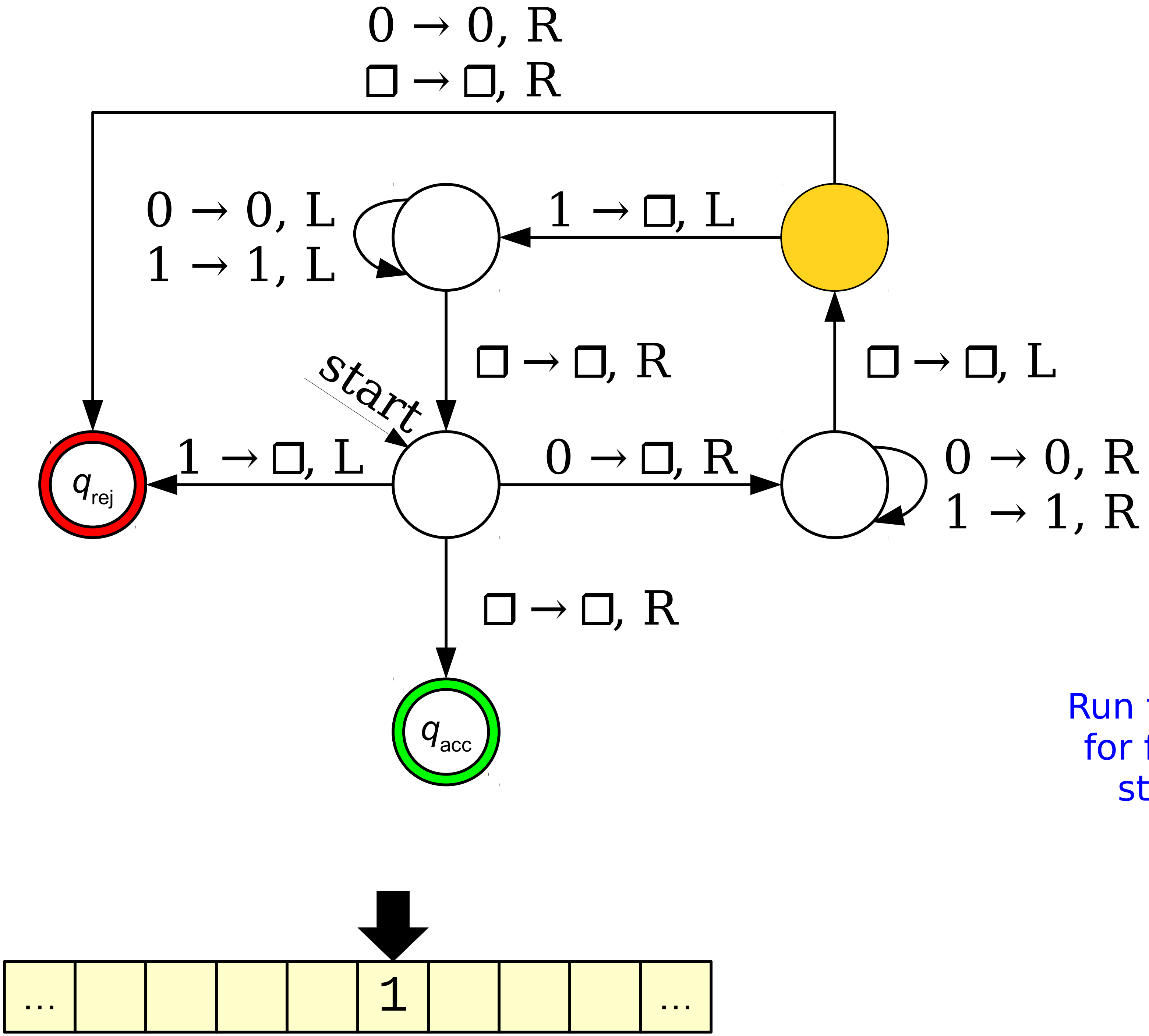


Run this TM for fifteen steps.

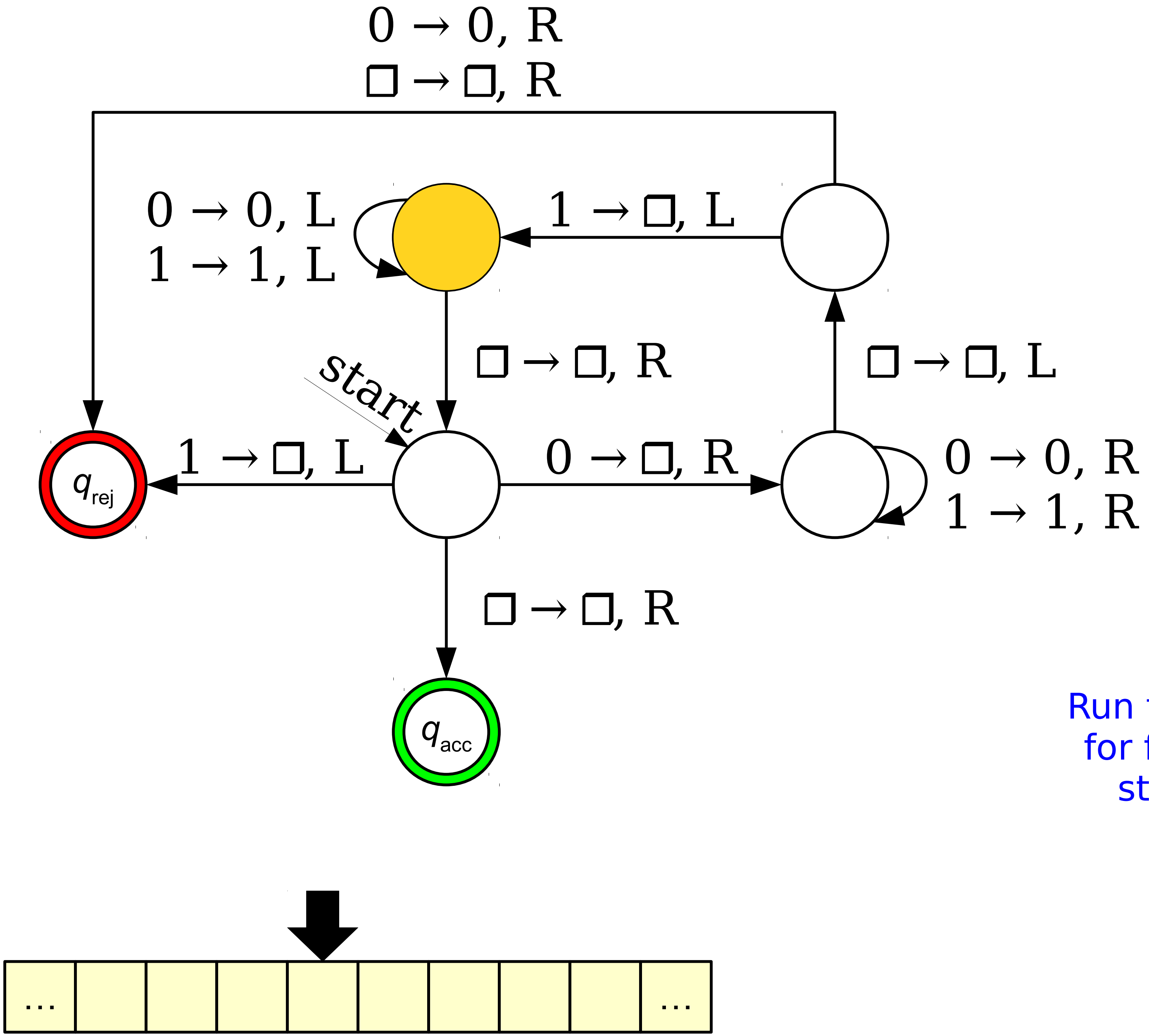




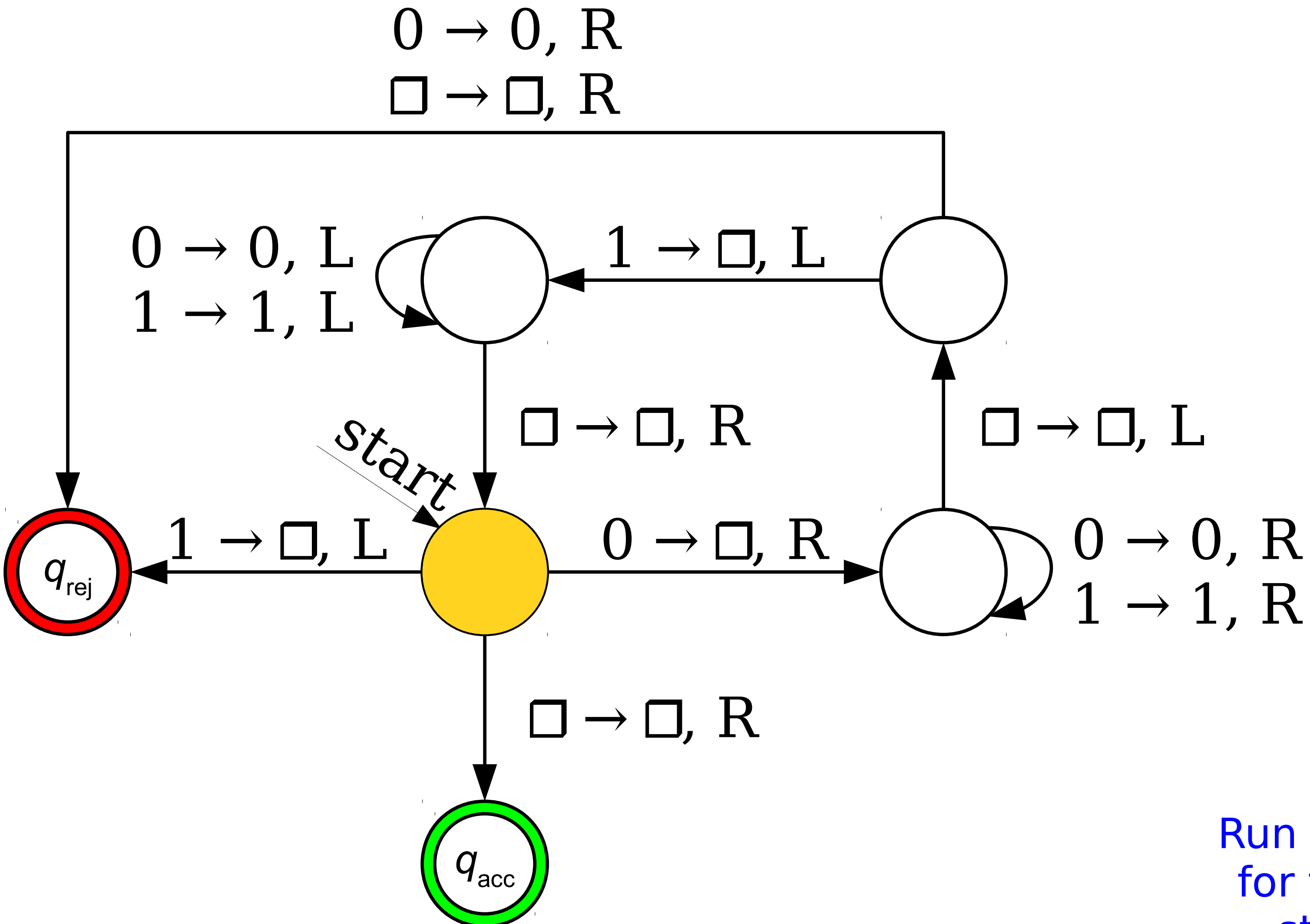
Run this TM
for fifteen
steps.



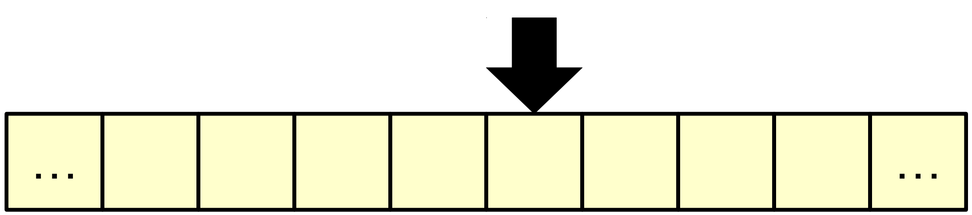
Run this TM for fifteen steps.

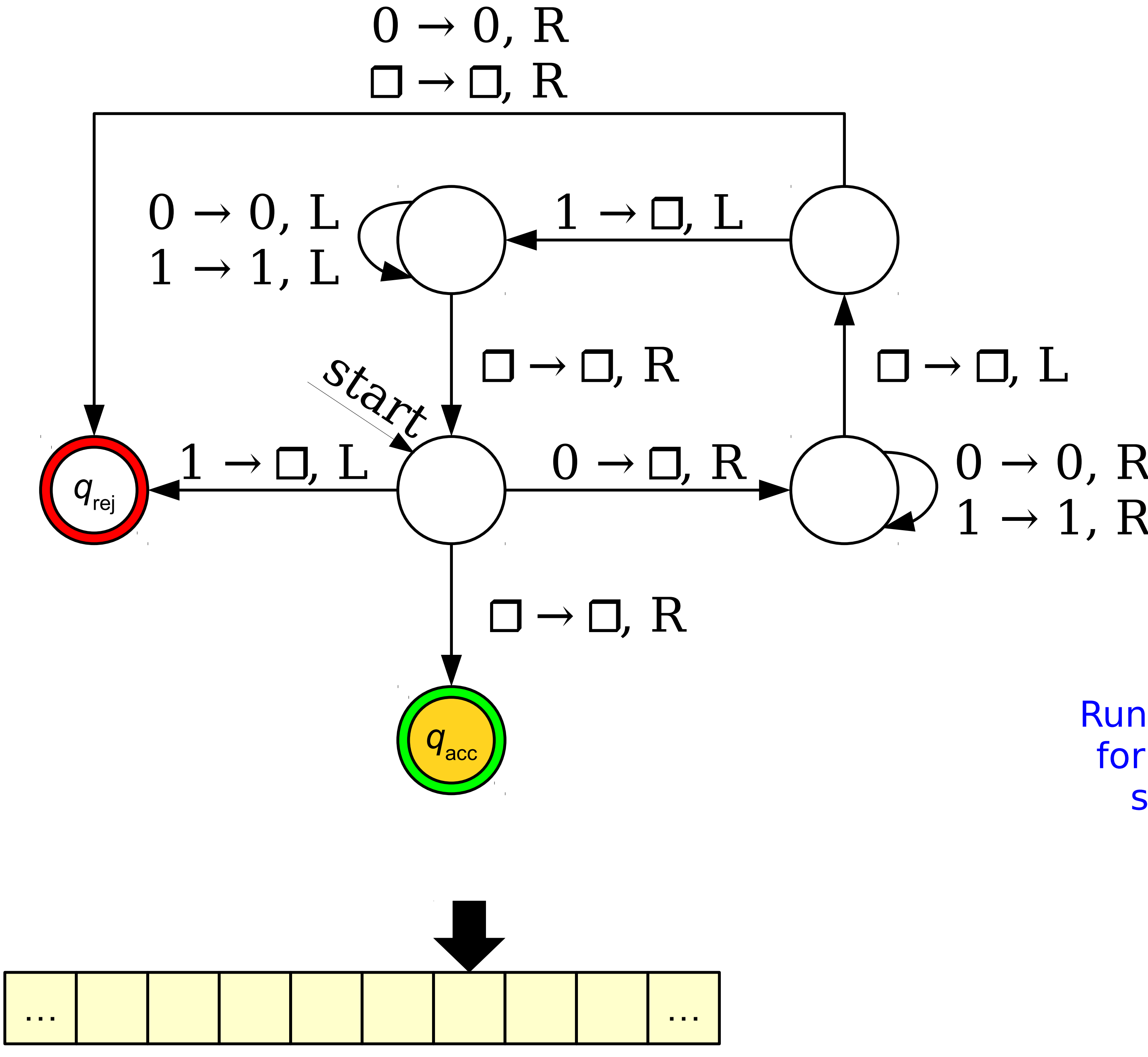


Run this TM for fifteen steps.



Run this TM for fifteen steps.





Run this TM
for fifteen
steps.

Some Verifiers

- Consider A_{TM}:

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

```
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?

What languages are verifiable?

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

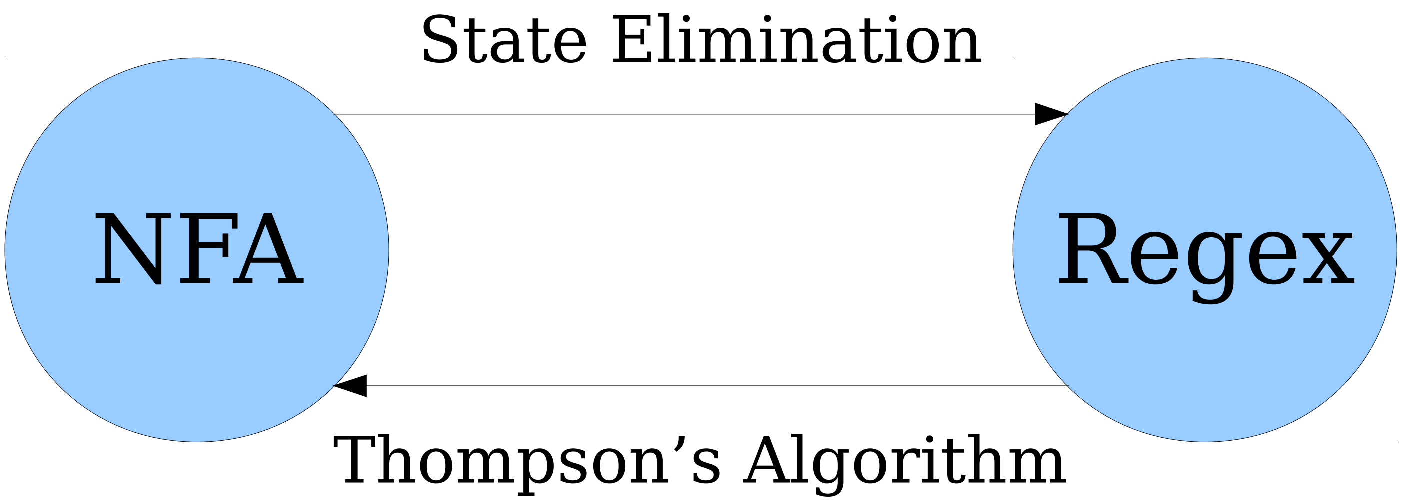
```
bool mysteryFunction(string w) {
    int i = 0;
    while (true) {
        for (each string c of length i) {
            if (V accepts ⟨w, c⟩) return true;
        }
        i++;
    }
}
```

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

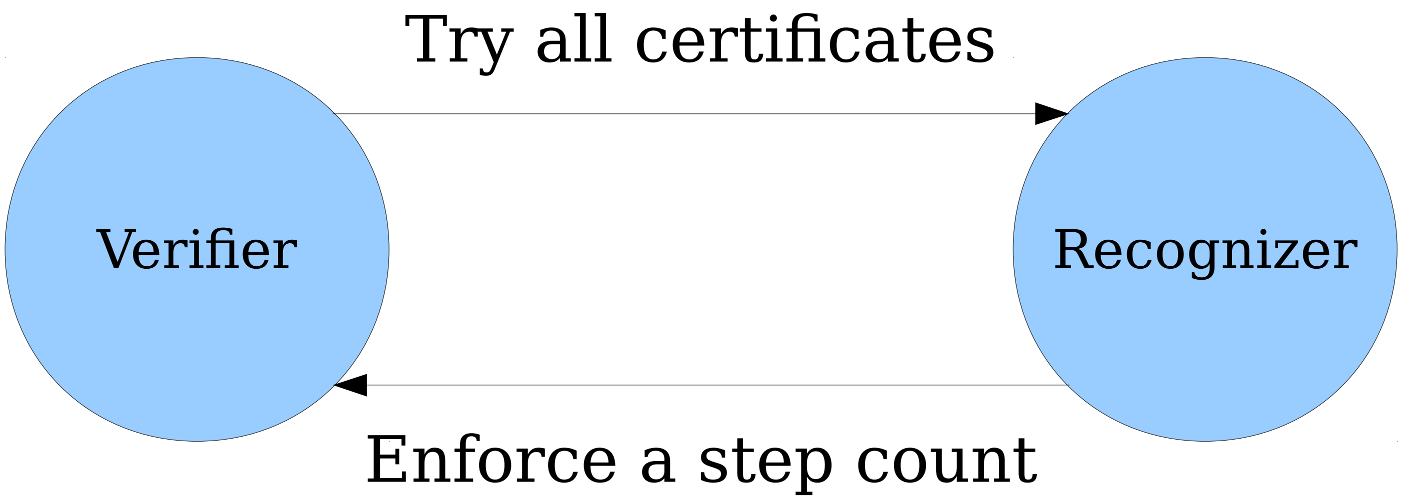
Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then **your answer**.

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

Where We've Been



Where We're Going

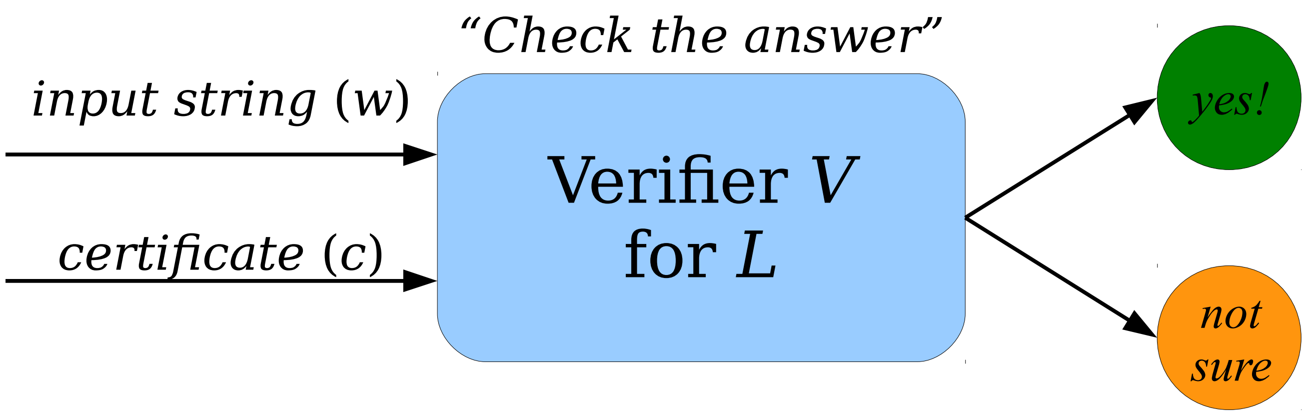


Verifiers and **RE**

- ***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.

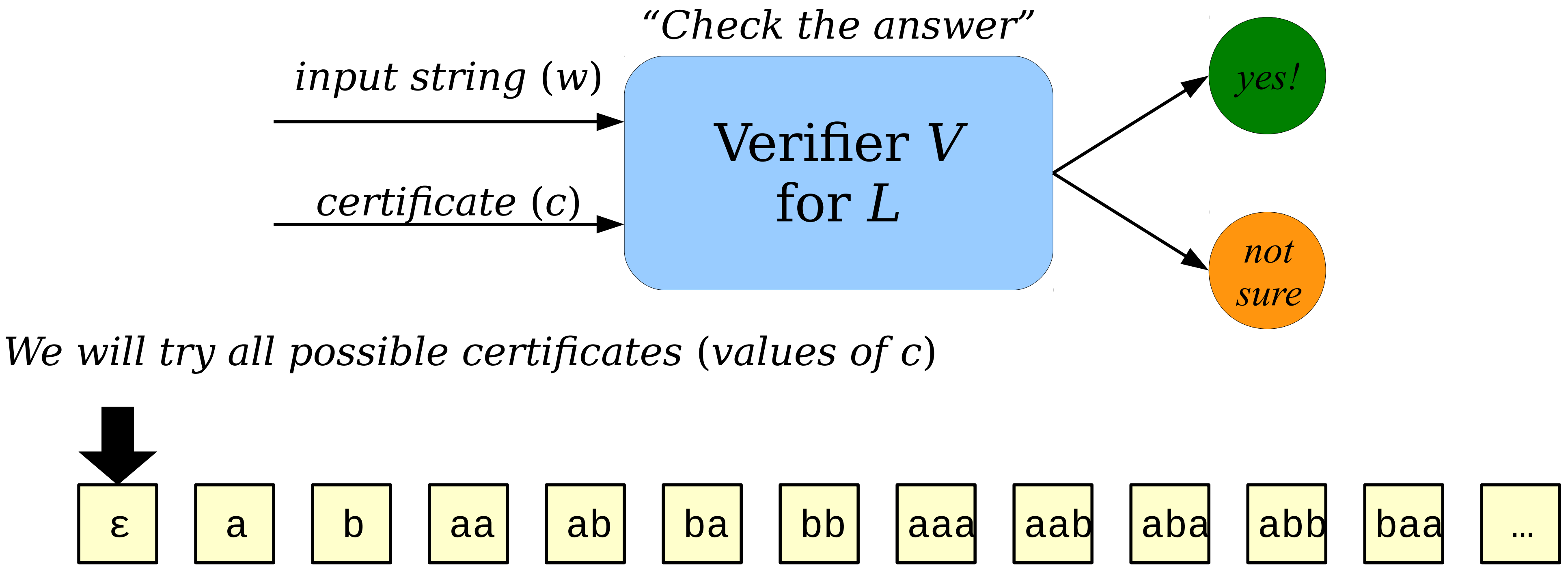
Verifiers and **RE**

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



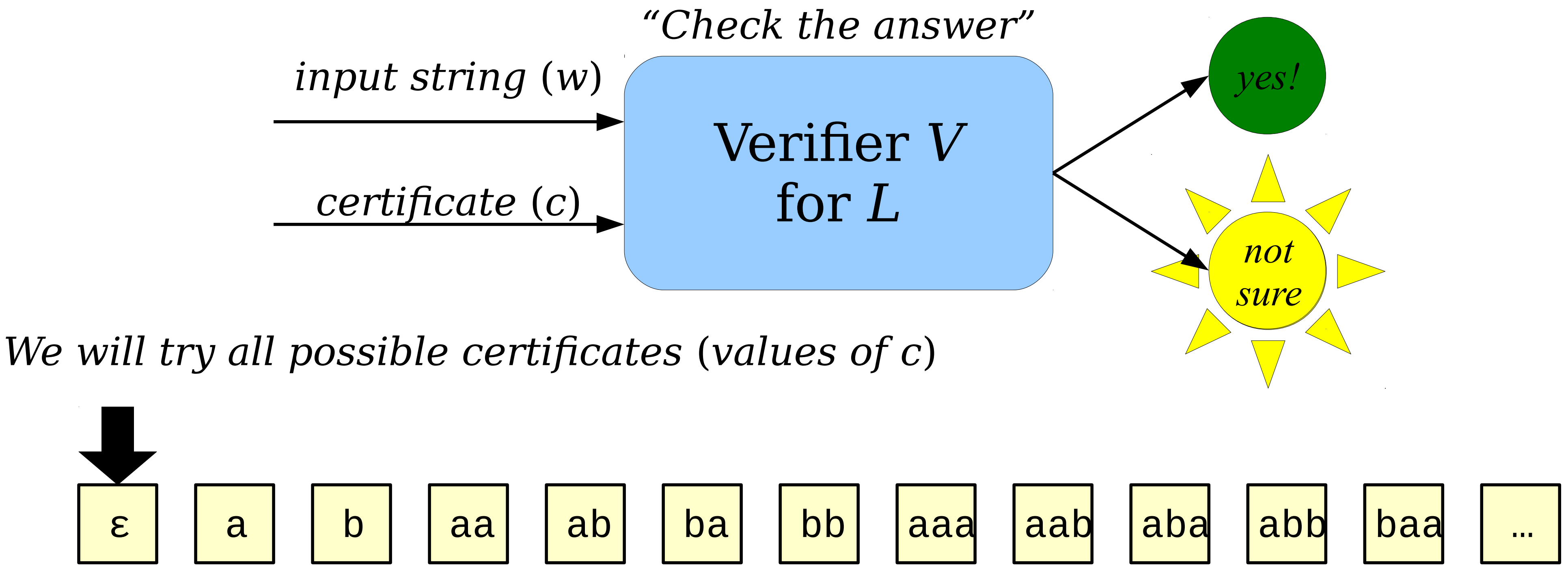
Verifiers and **RE**

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



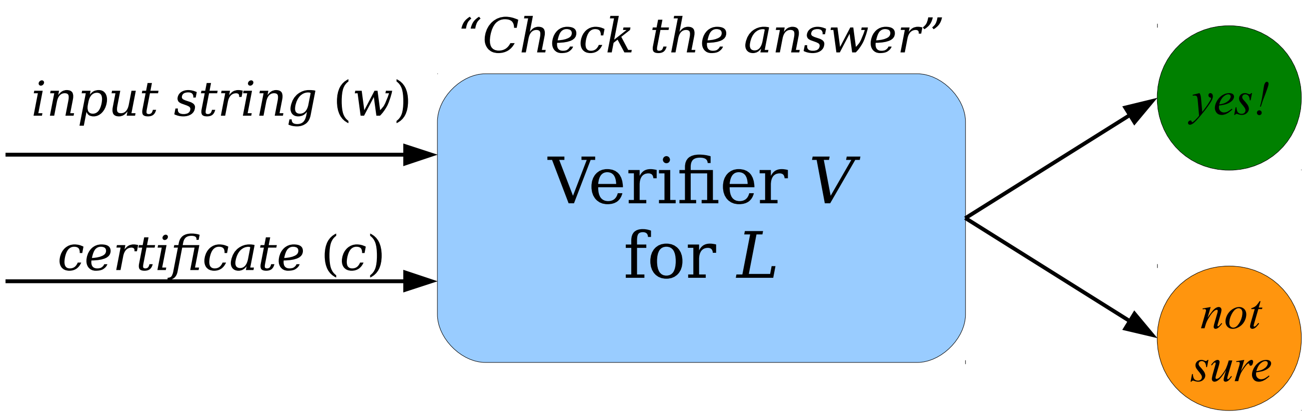
Verifiers and **RE**

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

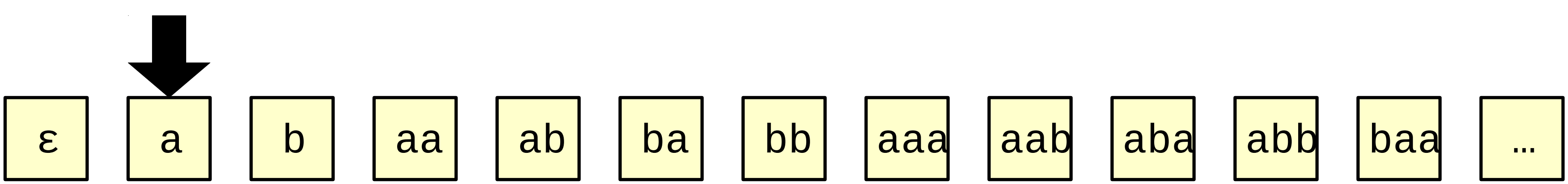


Verifiers and **RE**

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

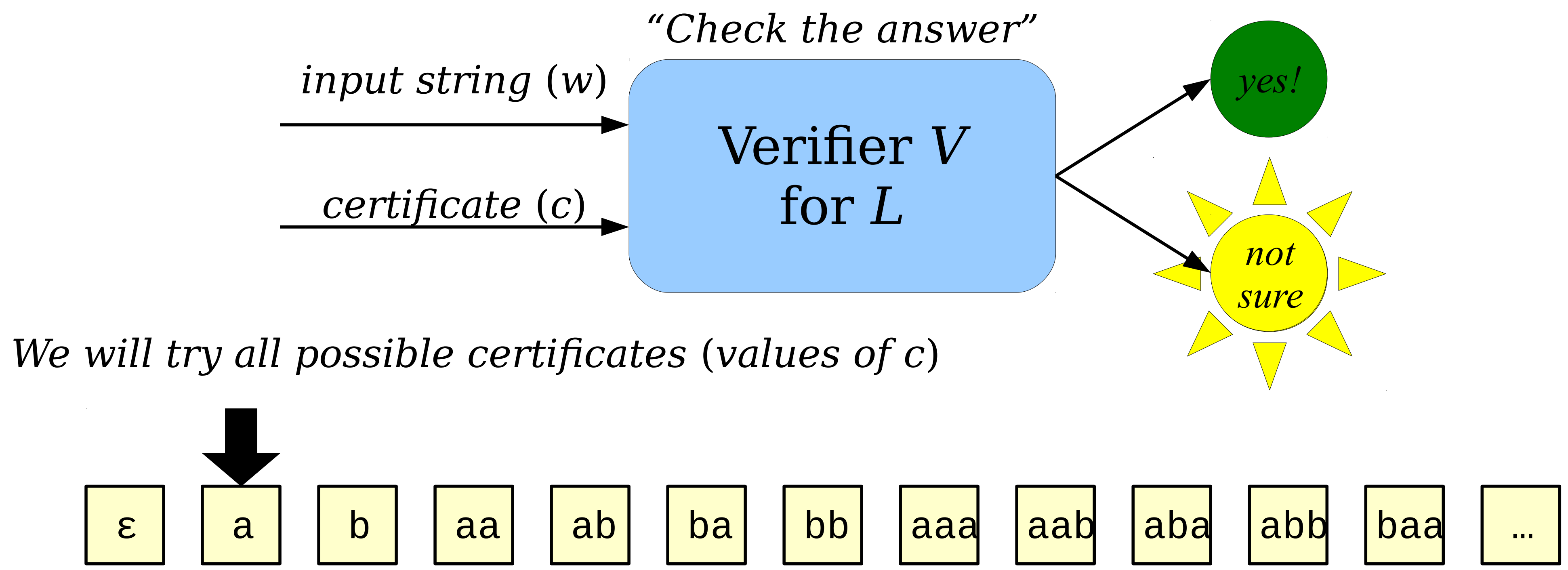


We will try all possible certificates (values of c)



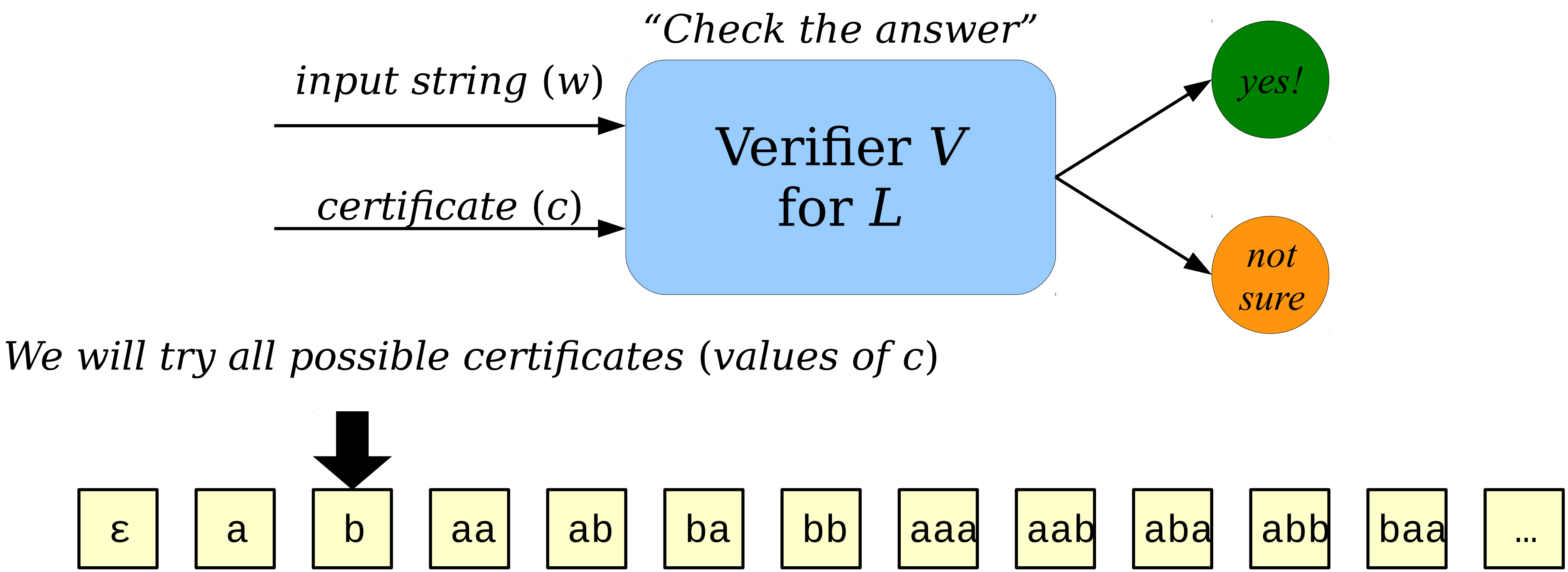
Verifiers and **RE**

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



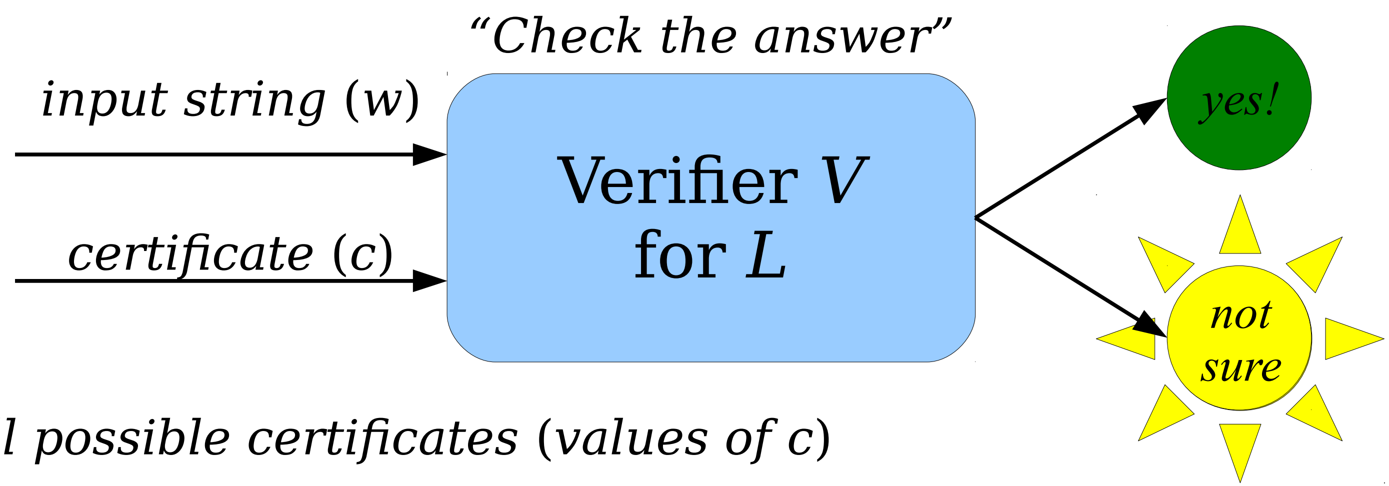
Verifiers and **RE**

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

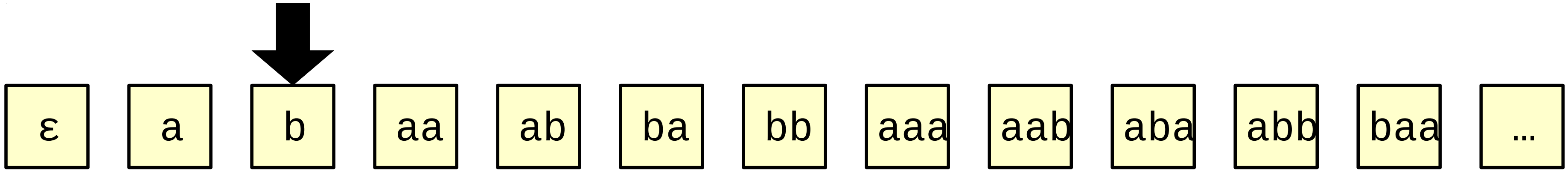


Verifiers and **RE**

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

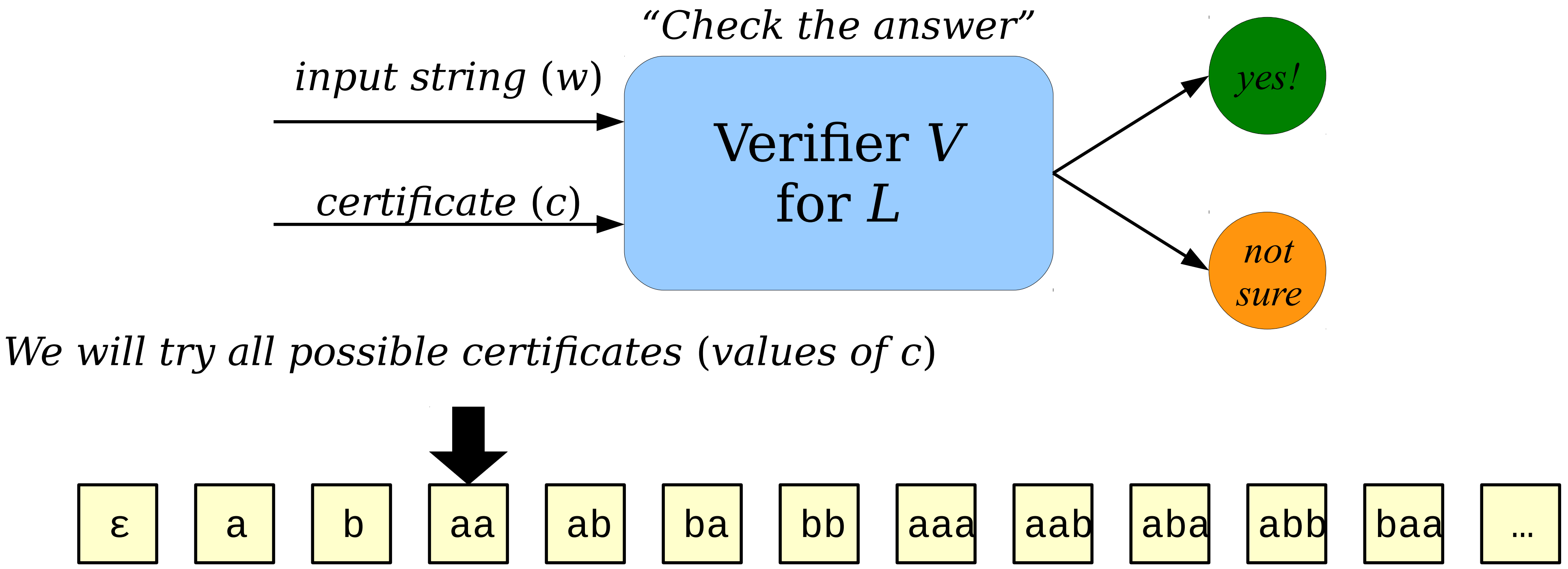


We will try all possible certificates (values of c)



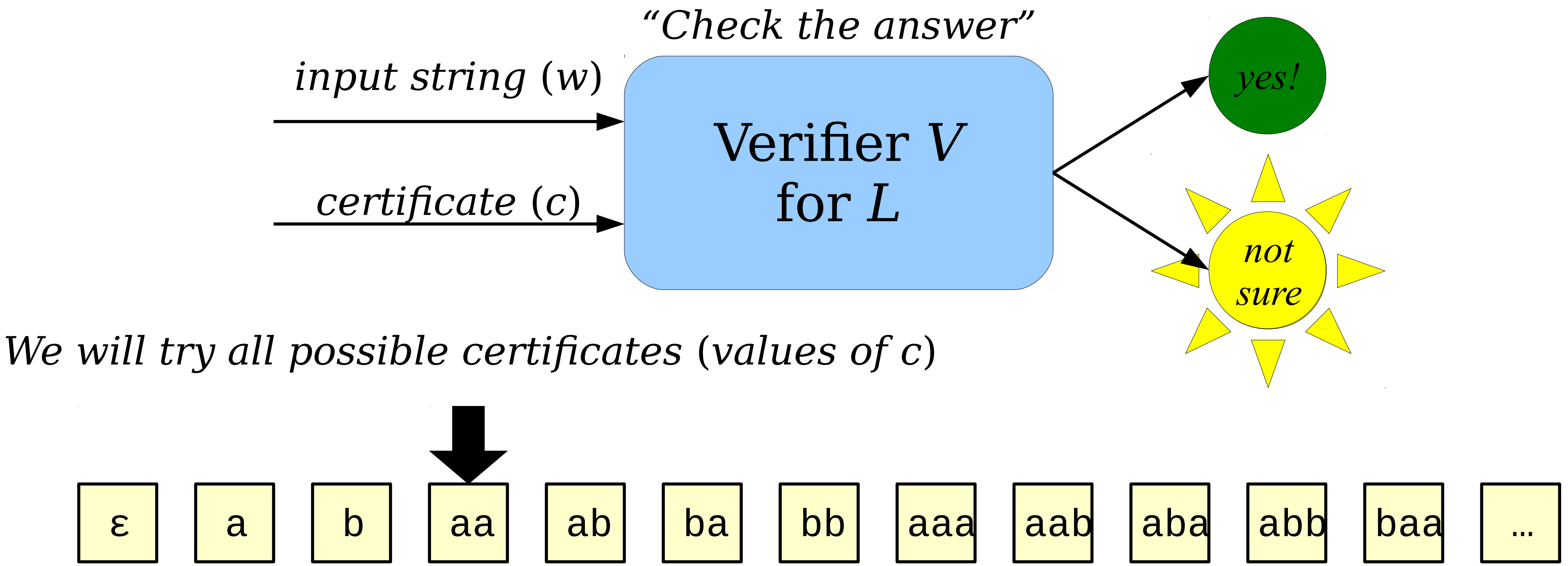
Verifiers and **RE**

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



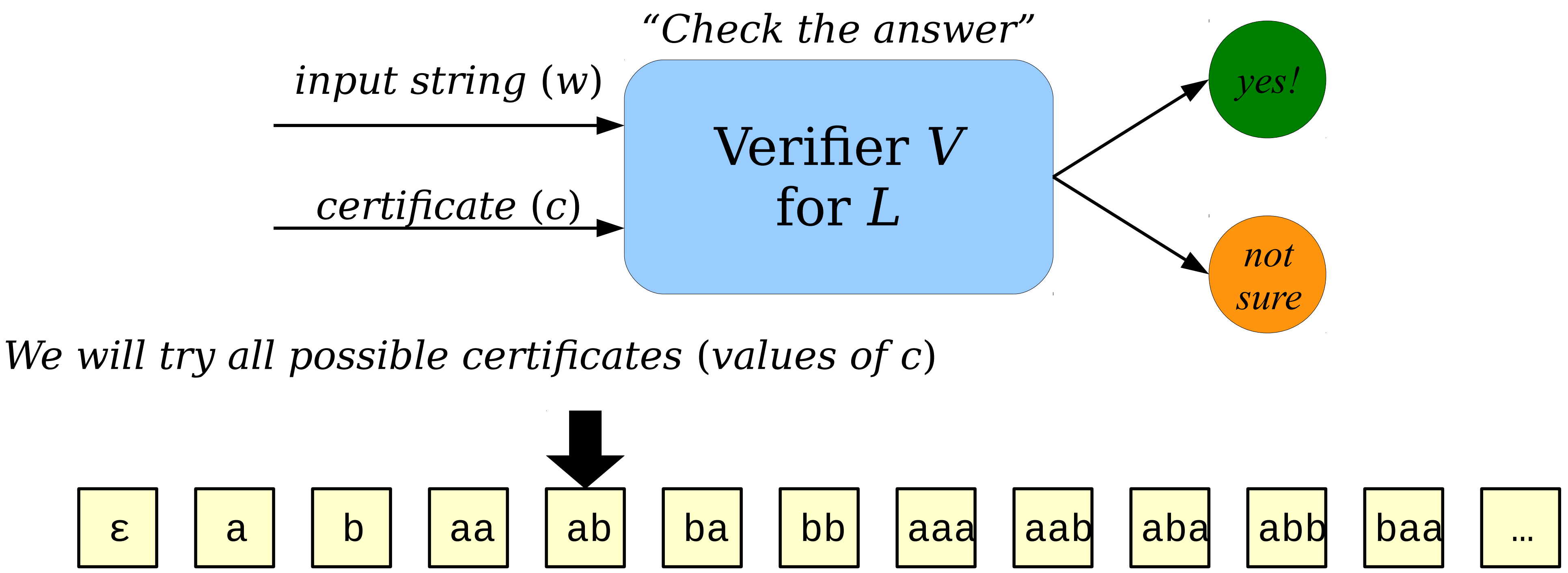
Verifiers and **RE**

- ***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.



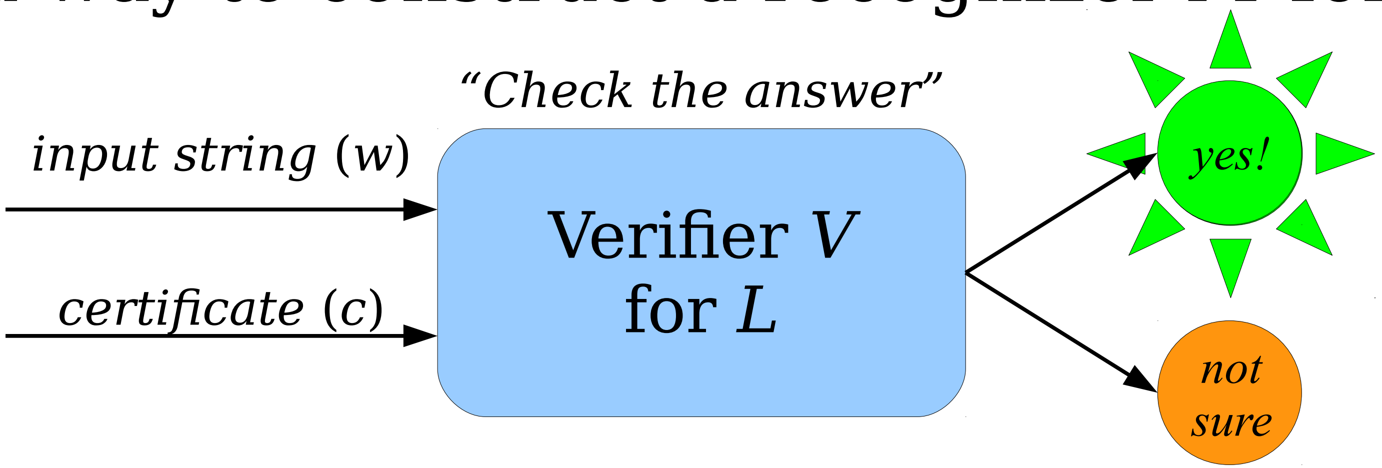
Verifiers and **RE**

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

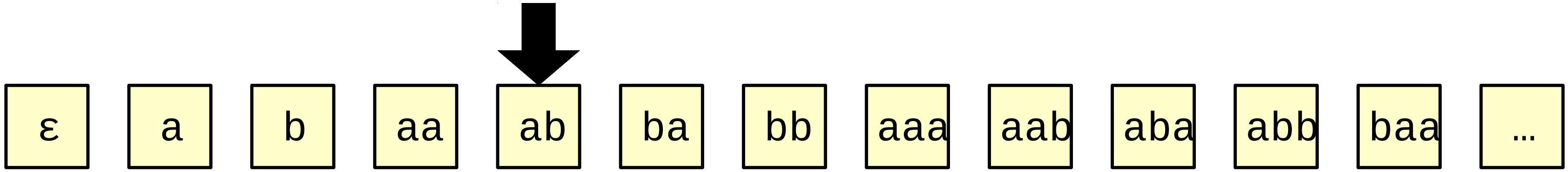


Verifiers and **RE**

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

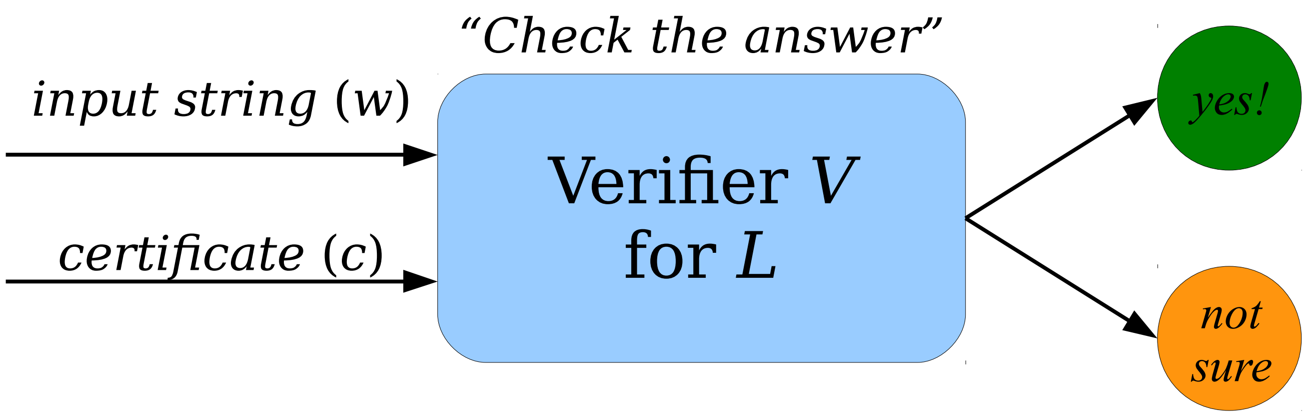


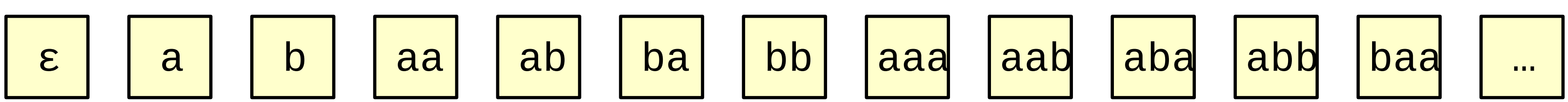
We will try all possible certificates (values of c)



Verifiers and **RE**

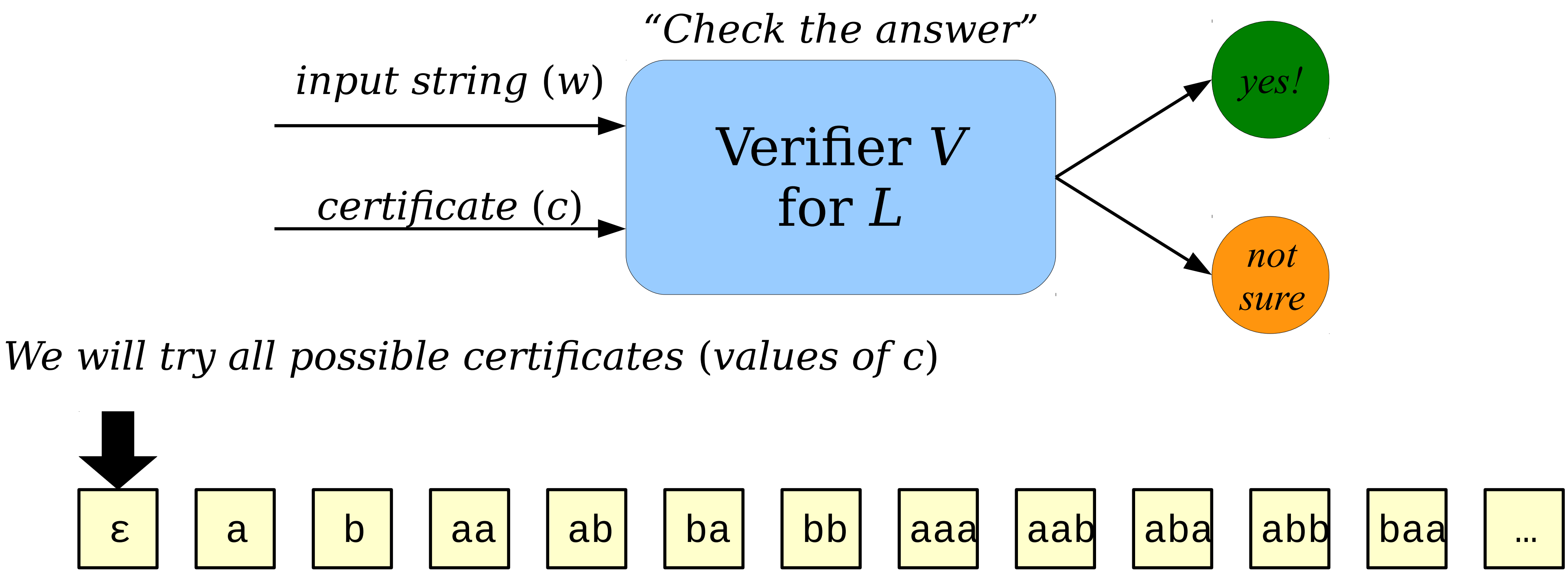
- ***Theorem:*** If there is a verifier V for a language L, then $L \in$ **RE**.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.





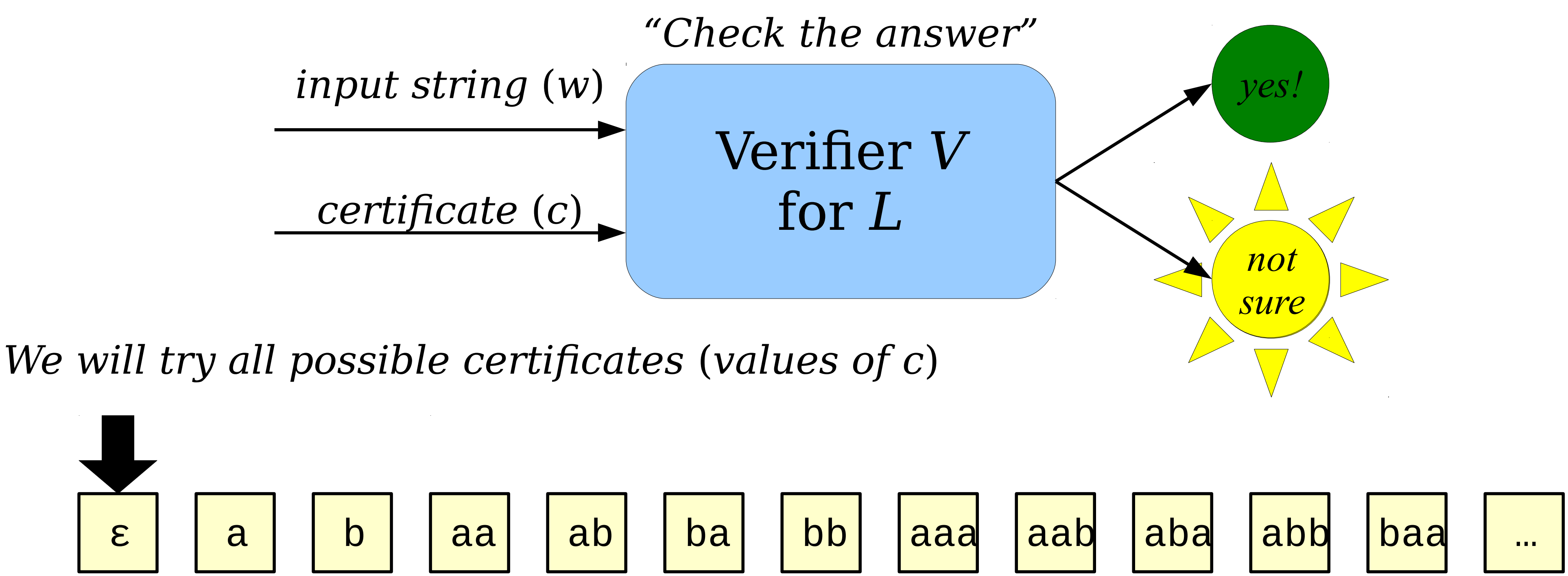
Verifiers and **RE**

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



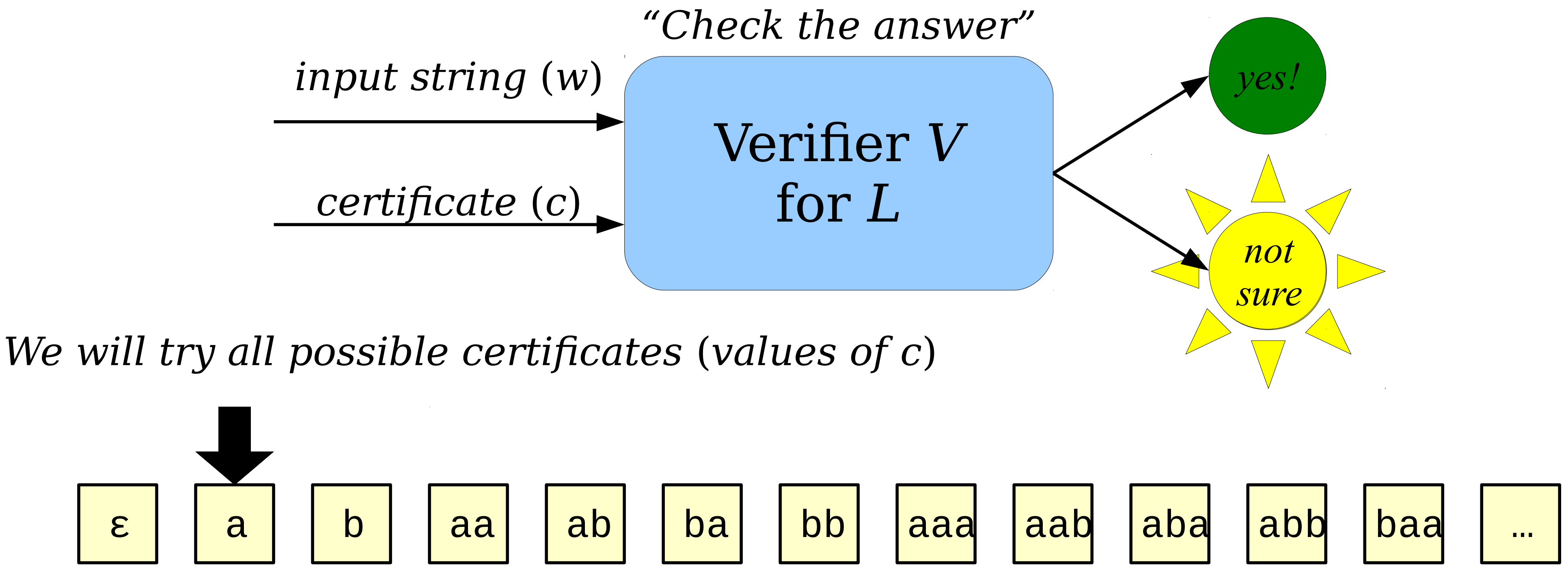
Verifiers and **RE**

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



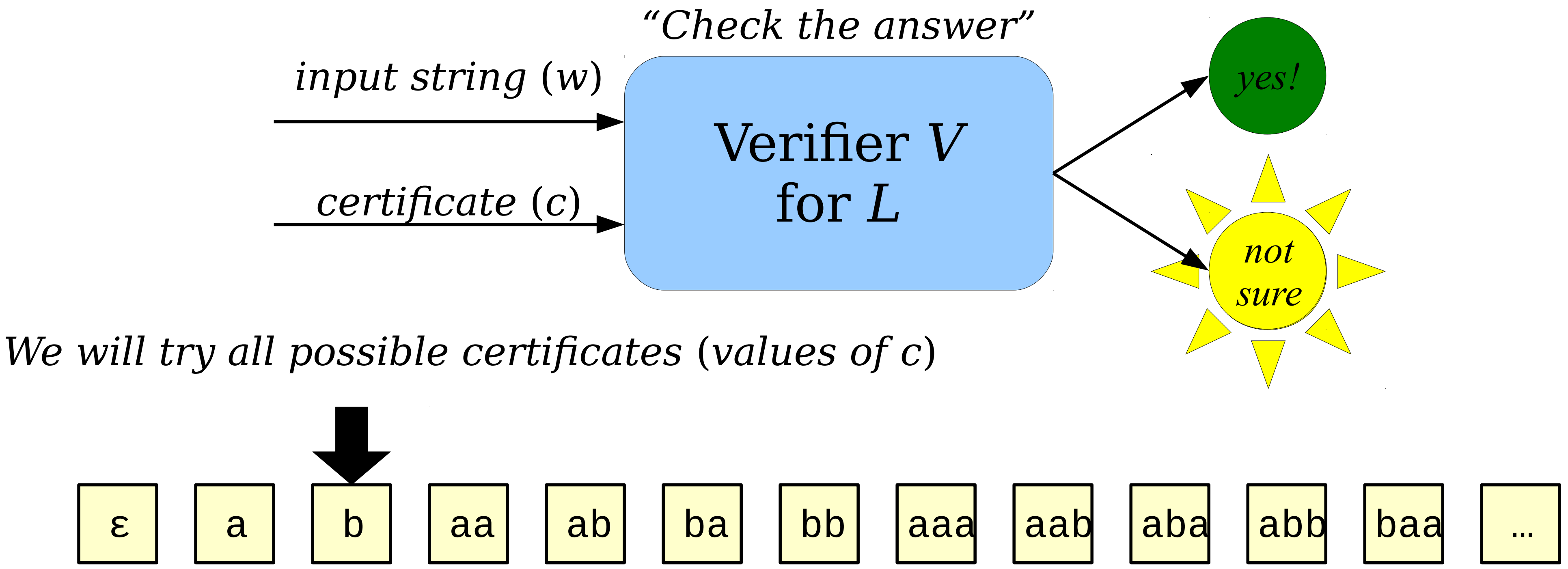
Verifiers and **RE**

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



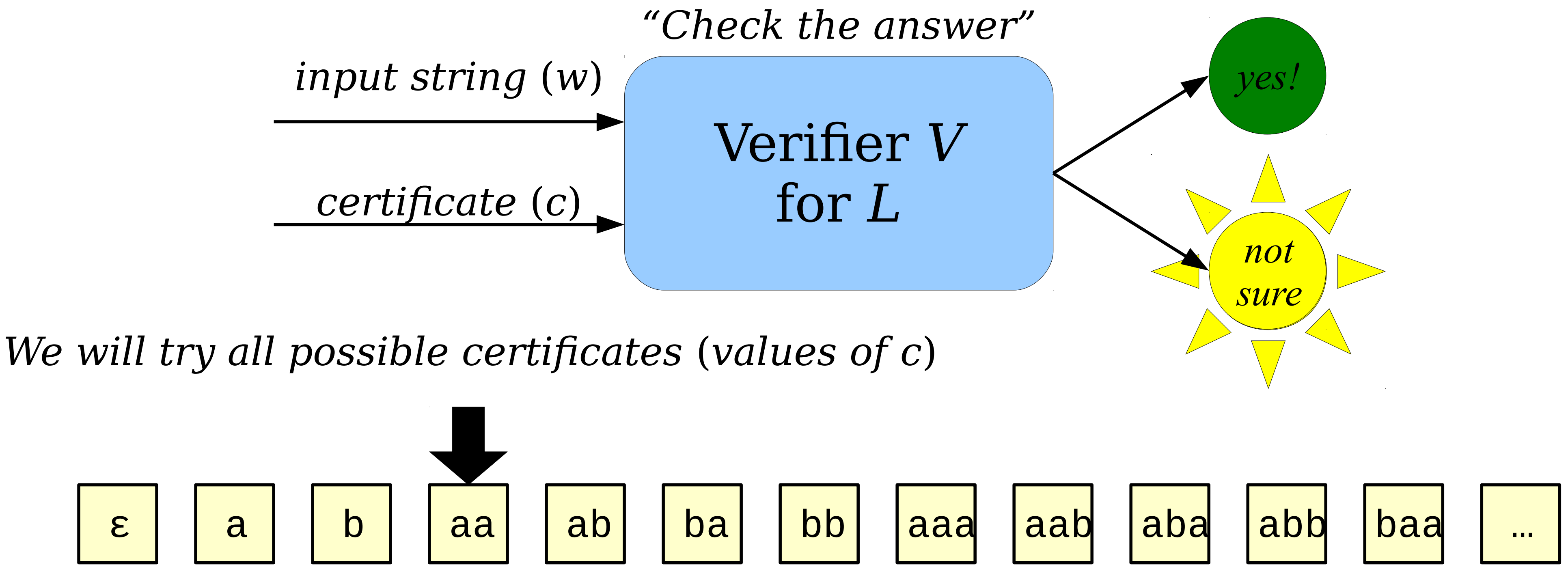
Verifiers and **RE**

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



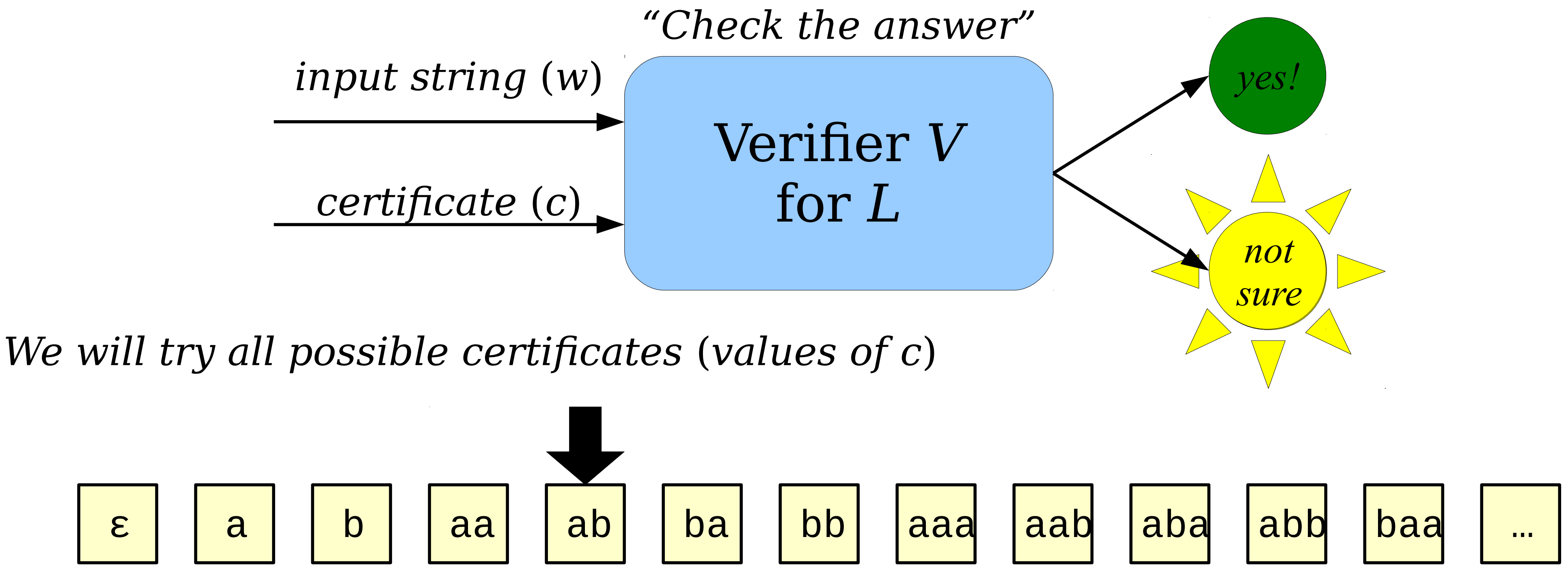
Verifiers and **RE**

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



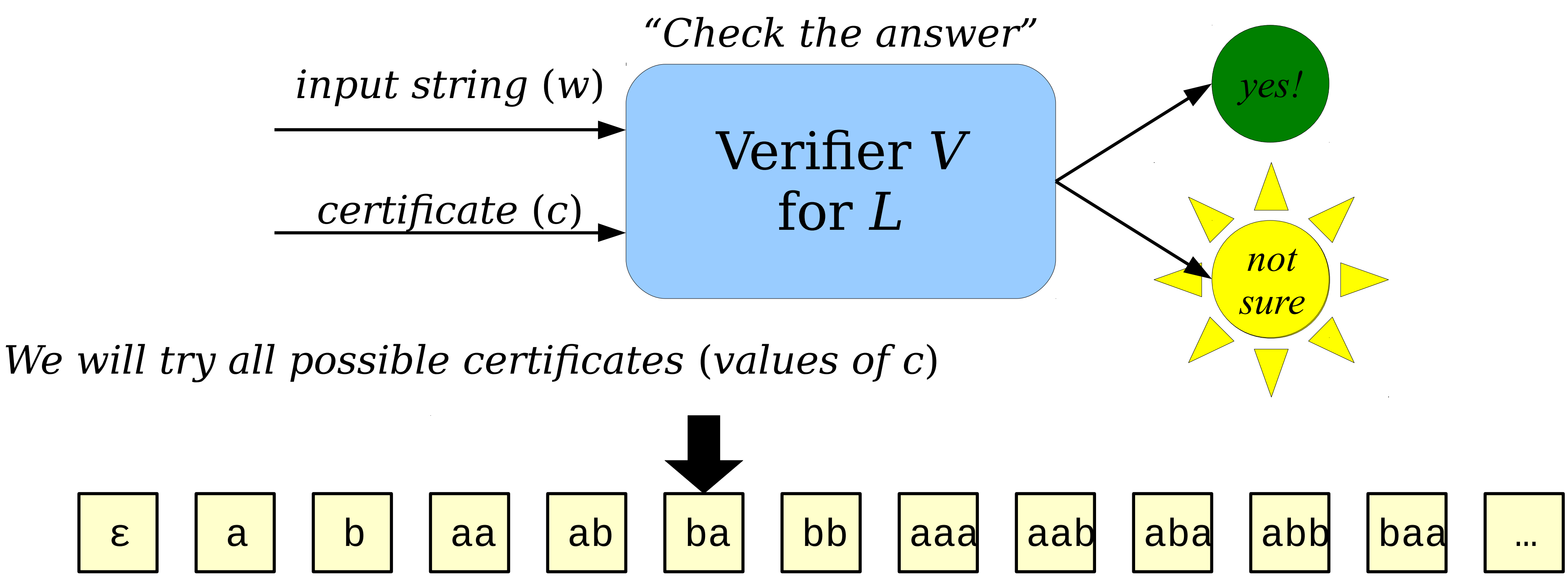
Verifiers and **RE**

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



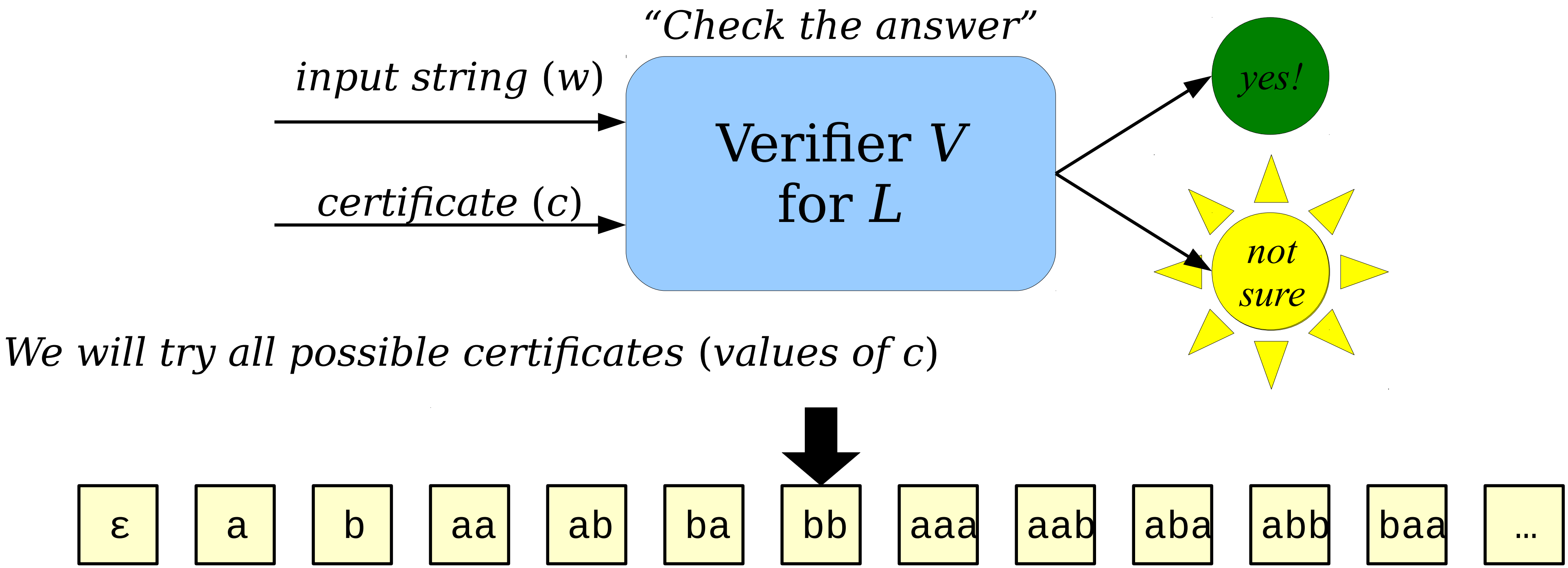
Verifiers and **RE**

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



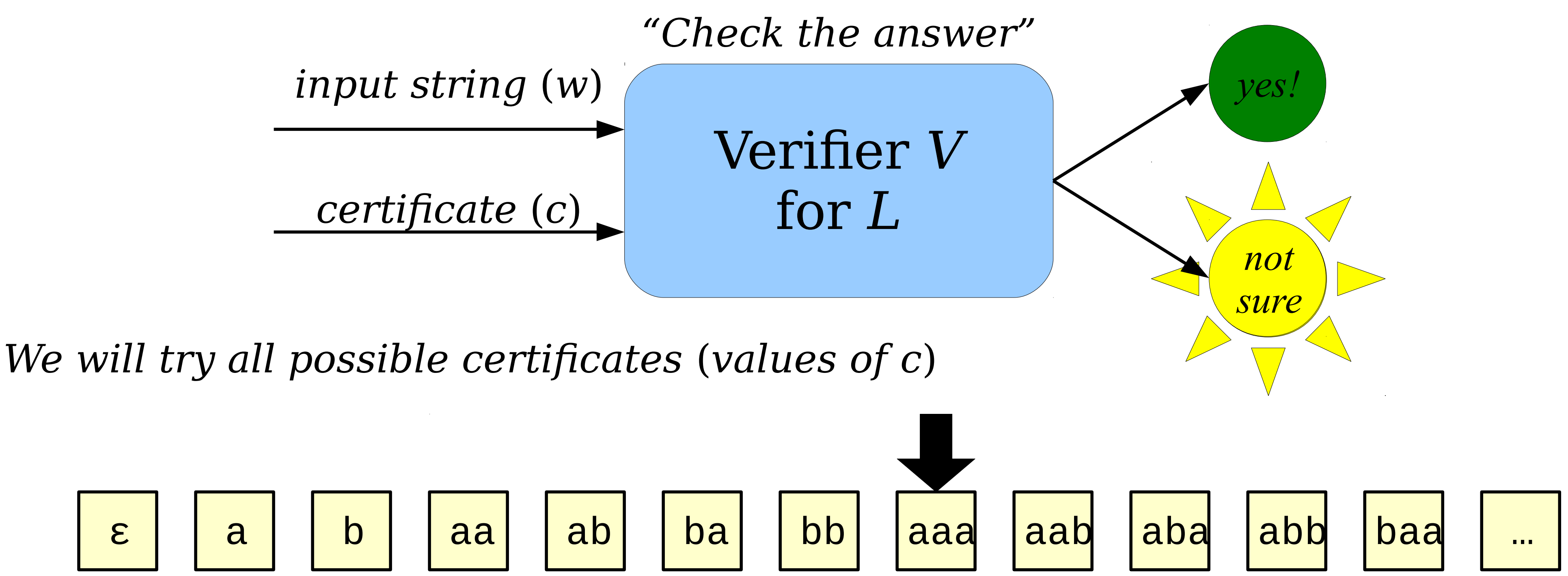
Verifiers and **RE**

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



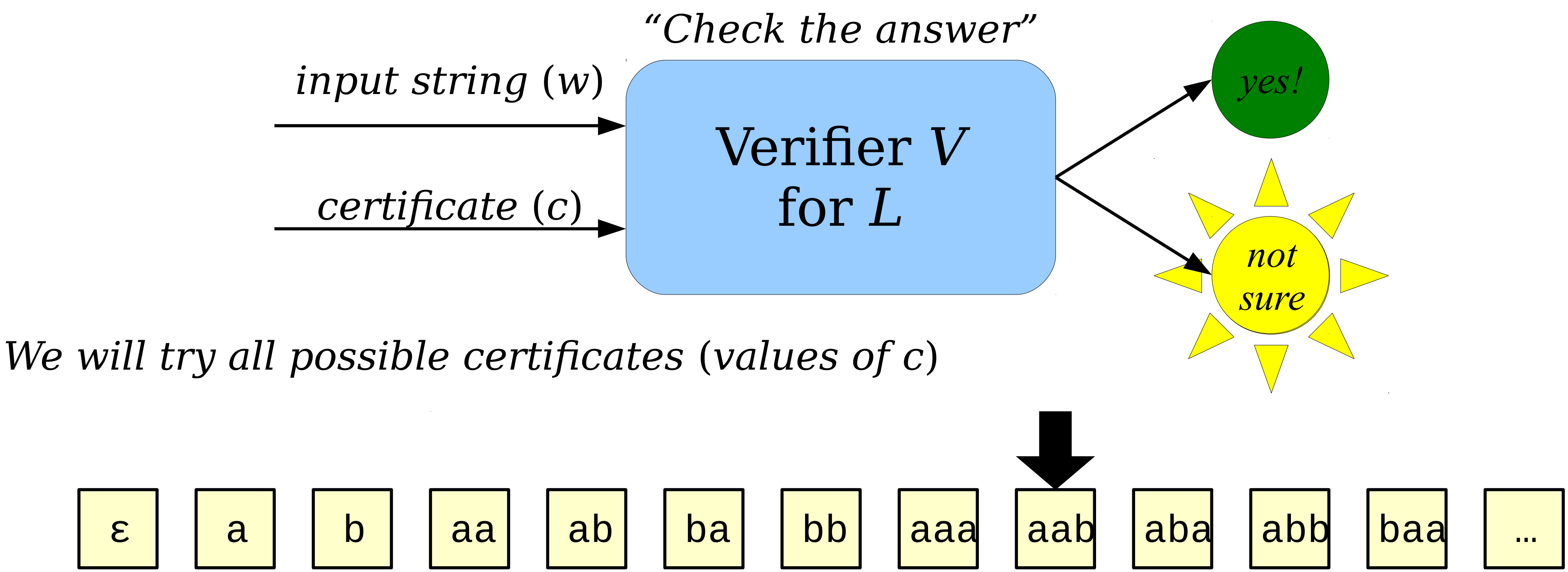
Verifiers and **RE**

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



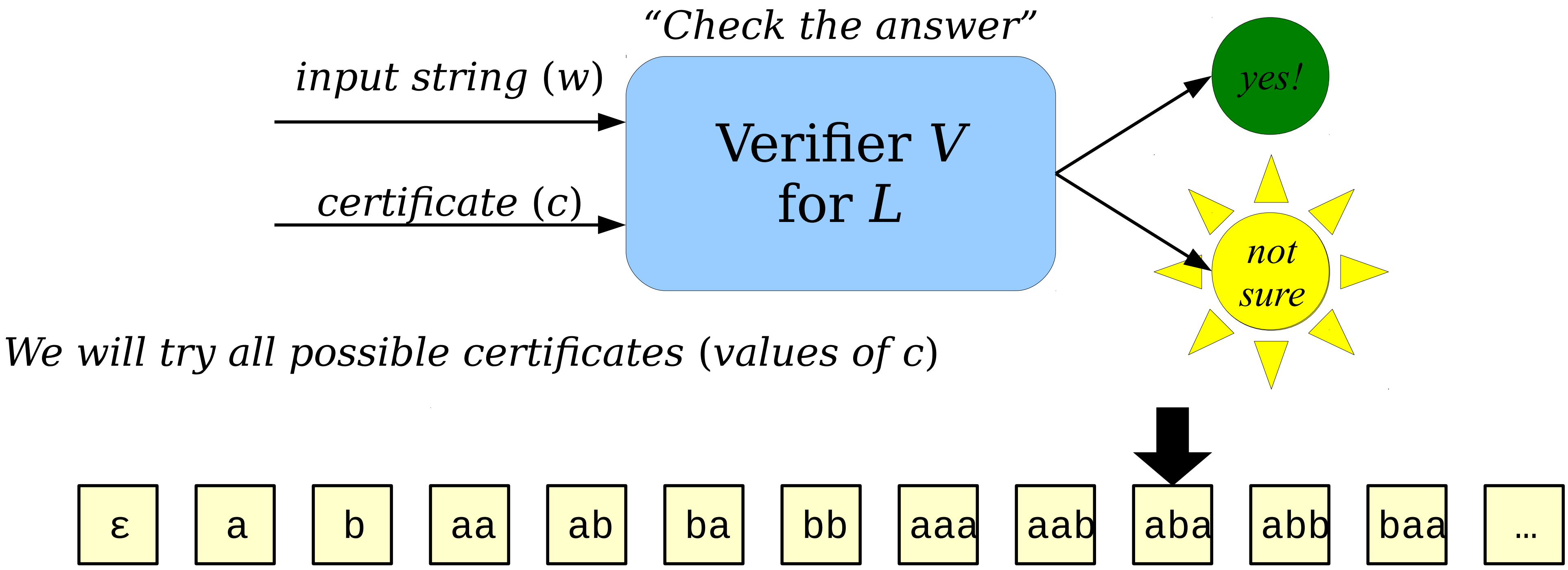
Verifiers and **RE**

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



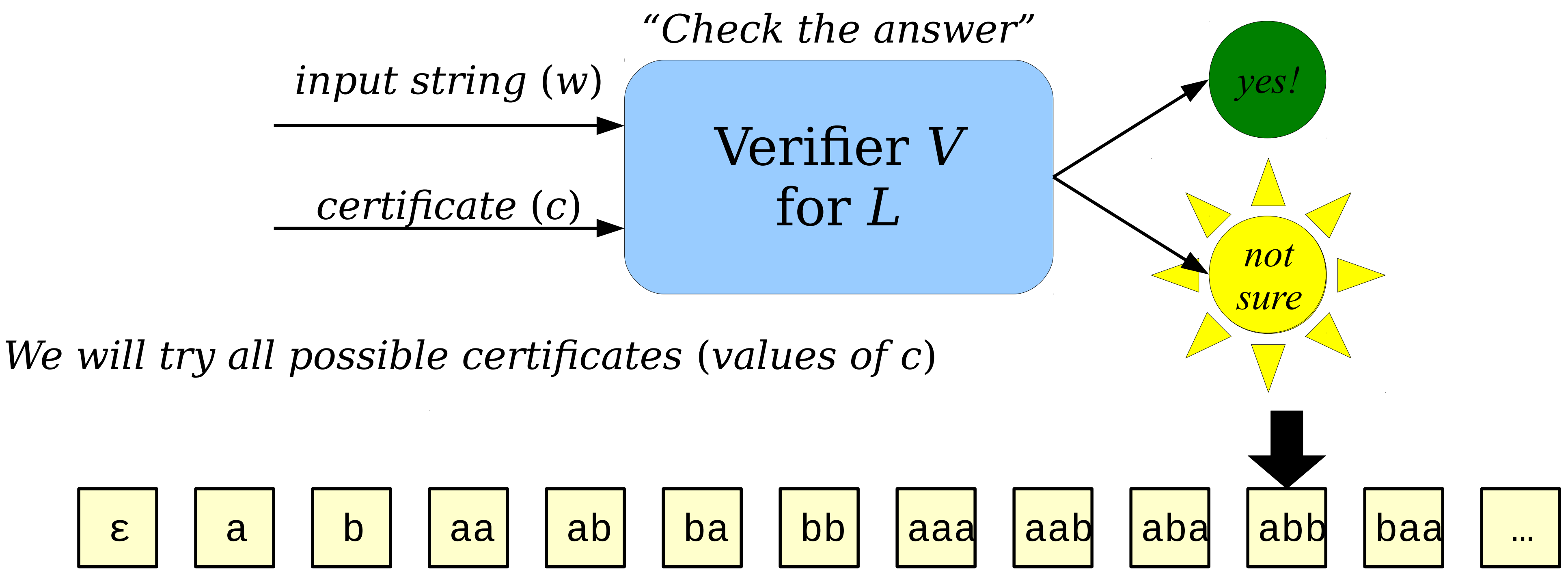
Verifiers and **RE**

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



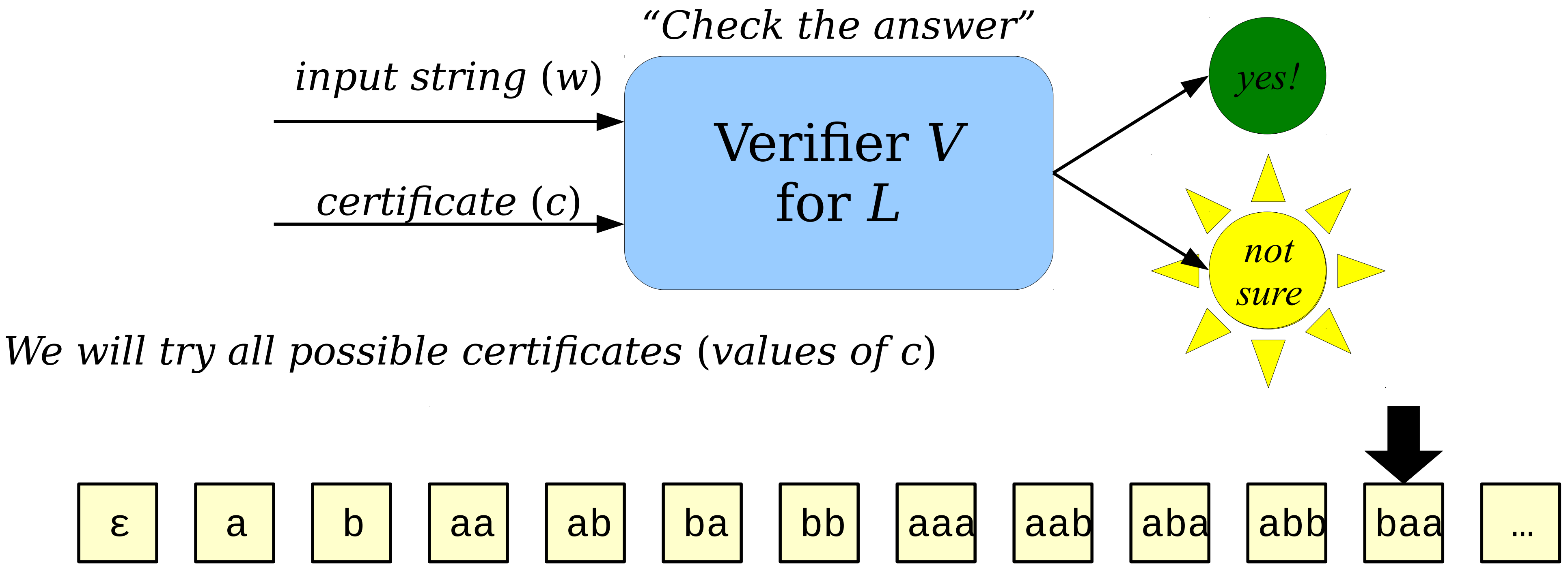
Verifiers and **RE**

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



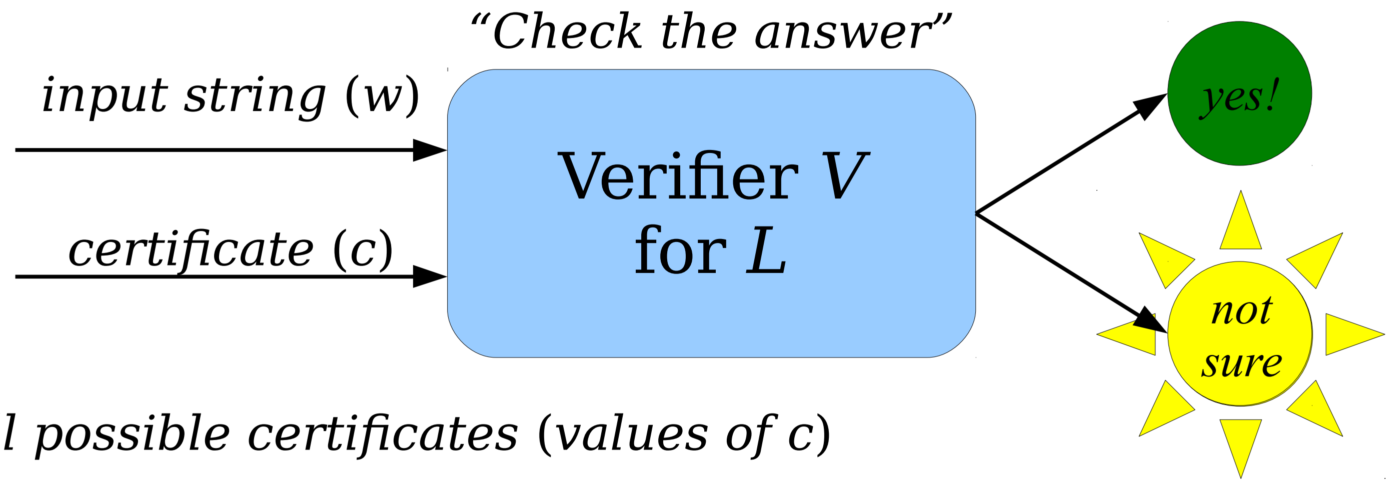
Verifiers and **RE**

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

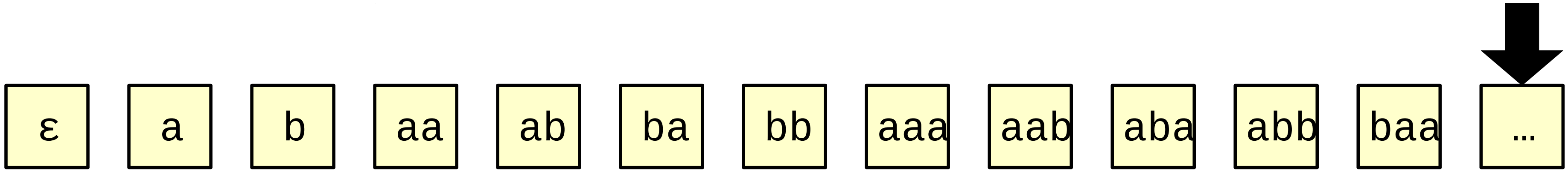


Verifiers and **RE**

- ***Theorem:*** If there is a verifier V for a language L, then $L \in$ **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 $L \in$ **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 $L \in \mathbf{RE}$, 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 $L \in \mathbf{RE}$, 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!

Finding Non-**RE** Languages

Finding Non-**RE** Languages

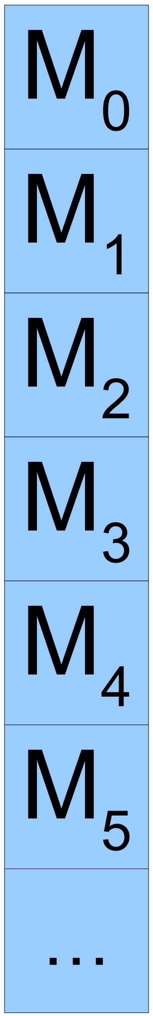
- Right now, we know that non-**RE** languages exist, but we have no idea what they look like.
- How might we find one?

Languages, TMs, and TM Encodings

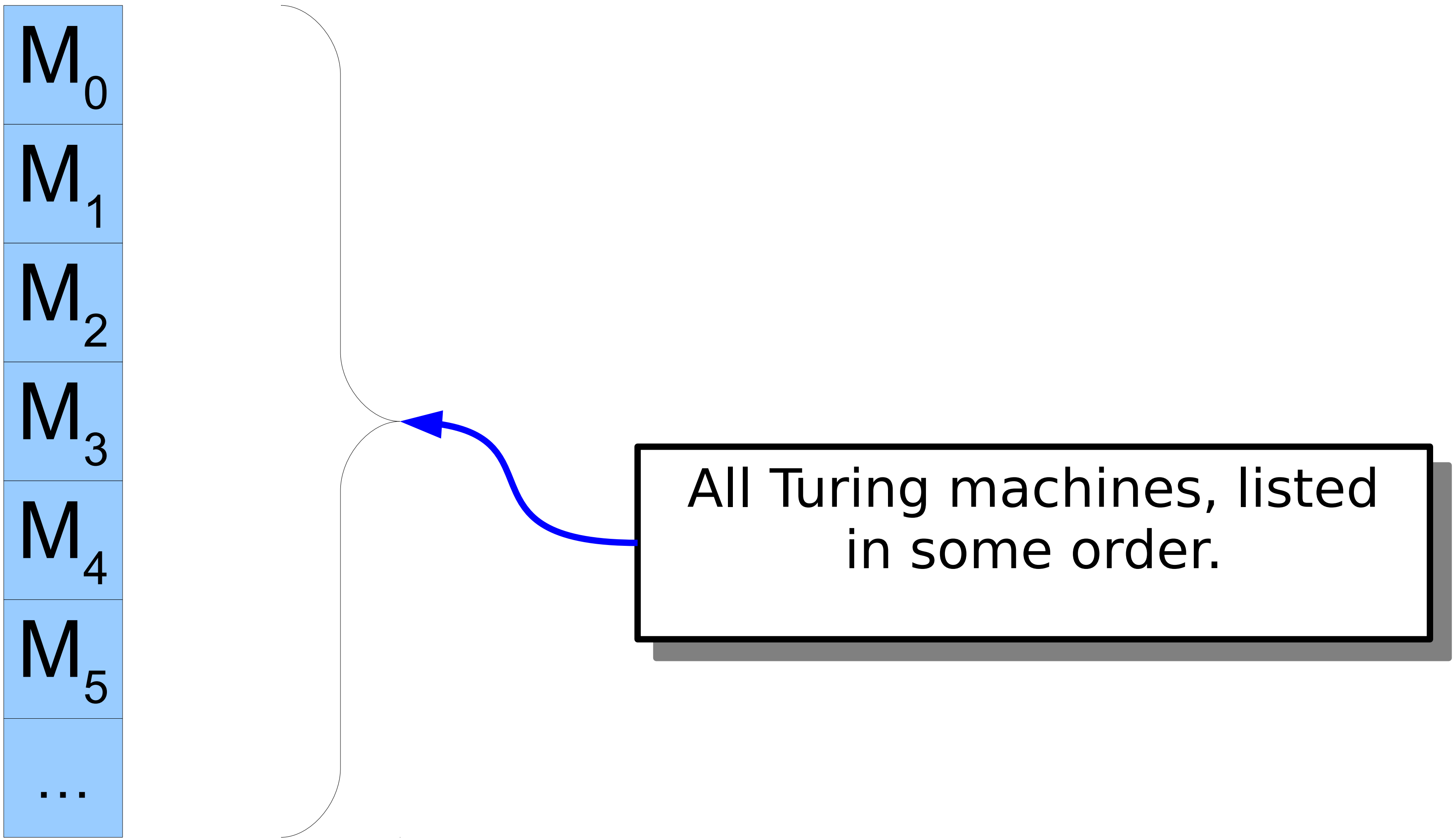
- Recall: The language of a TM M is the set

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

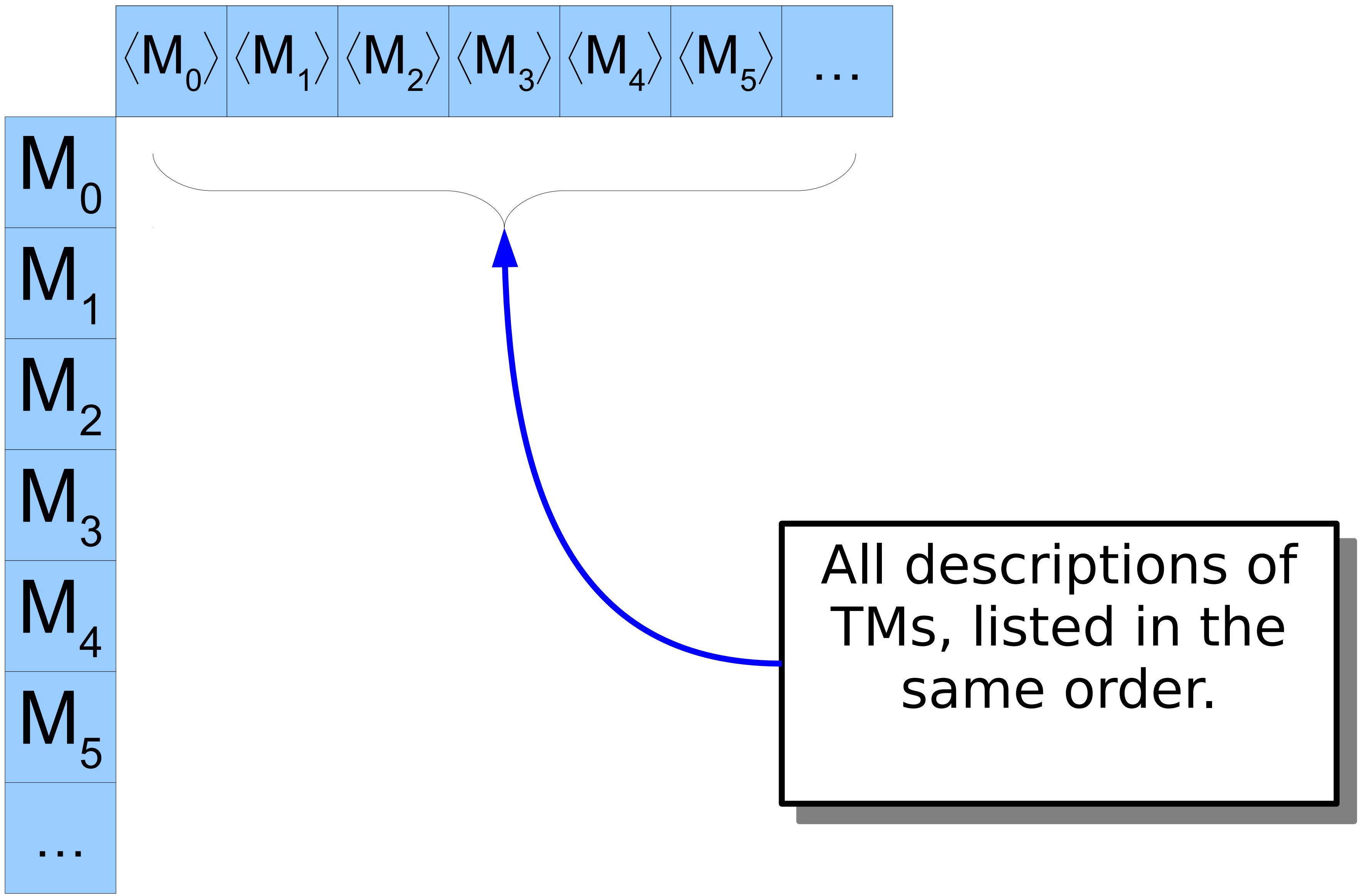
- Some of the strings in this set might be descriptions of TMs.
- What happens if we list off all Turing machines, looking at how those TMs behave given other TMs as input?



M0
M1
M2
M3
M4
M5
...


M_0
M_1
M_2
M_3
M_4
M_5
…
All Turing machines, listed in some 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							
M_1							
M_2							
M_3							
M_4							
M_5							
…							



⟨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	…
…	…	…	…	…	…	…	…

What are we going to do next?

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

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

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

Acc	Acc	Acc	No	Acc	No	…

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

No	No	No	Acc	No	Acc	…

Flip all “accept” to “no” and vice-versa

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

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

No	No	No	Acc	No	Acc	…

What TM has this behavior?

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

No TM has this behavior!

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	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 TMs that do not accept their own description."

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

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

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

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

Diagonalization Revisited

- The ***diagonalization language***, which we denote $\boldsymbol{L_D}$, is defined as

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

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

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

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

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

$L_D = \{ \langle M \rangle \mid M$ 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 recognizer R such that $\mathscr{L}(R) = L_D$.

$L_D = \{ \langle M \rangle \mid M$ 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 recognizer R such that $\mathscr{L}(R) = L_D$.

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

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

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

L_D = { ⟨M⟩ | M is a TM and ⟨M⟩ $\notin \mathscr{L}(M)$ }

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

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

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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 recognizer R such that $\mathscr{L}(R) = L_D$.

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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 recognizer R such that $\mathscr{L}(R) = L_D$.

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M.

L_D = { ⟨M⟩ | M is a TM and ⟨M⟩ $\notin \mathscr{L}(M)$ }

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

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

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M.

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 recognizer R such that $\mathscr{L}(R) = L_D$.

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M. In particular, this means that statement (2) holds for the TM R, which tells us that

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

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

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

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

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M. In particular, this means that statement (2) holds for the TM R, which tells us that

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

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 recognizer R such that $\mathscr{L}(R) = L_D$.

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M. In particular, this means that statement (2) holds for the TM R, which tells us 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.

$L_D = \{ \langle M \rangle \mid M$ 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 recognizer R such that $\mathscr{L}(R) = L_D$.

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

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

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

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M. In particular, this means that statement (2) holds for the TM R, which tells us 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 $L_D \notin \mathbf{RE}$.

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

Theorem: $L_{\mathrm{D}} \notin \mathbf{RE}$.

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

Let M be an arbitrary TM. Since $\mathscr{L}(R) = L_D$, we know that

$$\langle M \rangle \in L_{\mathrm{D}} \quad \text{iff} \quad \langle M \rangle \in \mathscr{L}(R). \qquad (1)$$

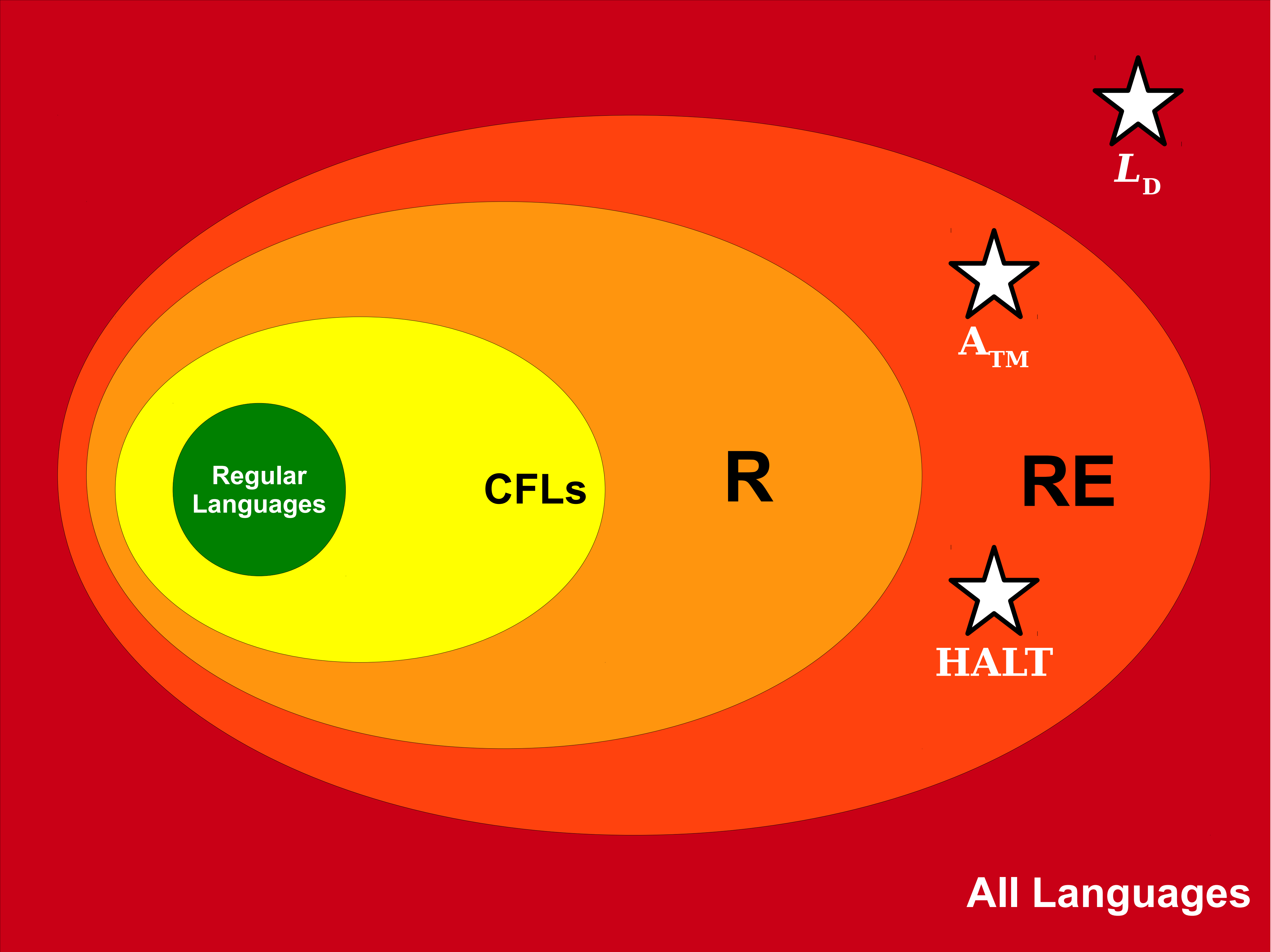
From the definition of L_{D}, we see that $\langle M \rangle \in L_{\mathrm{D}}$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Combining this with statement (1) tells us that

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

Since our choice of M was arbitrary, we see that statement (2) holds for any TM M. In particular, this means that statement (2) holds for the TM R, which tells us that

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

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



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:

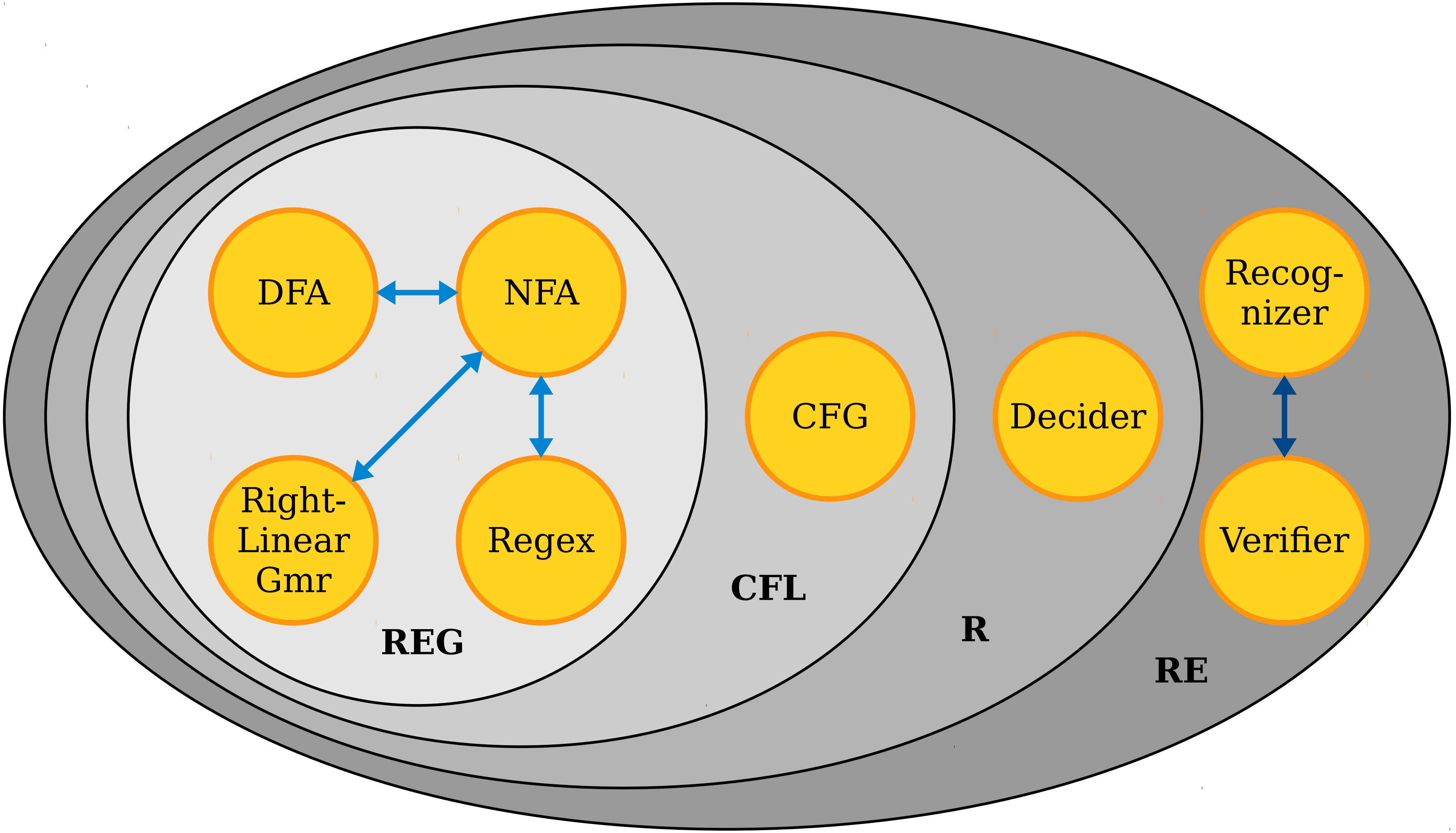
  There are inherent limits about what mathematics can teach us.

- 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!