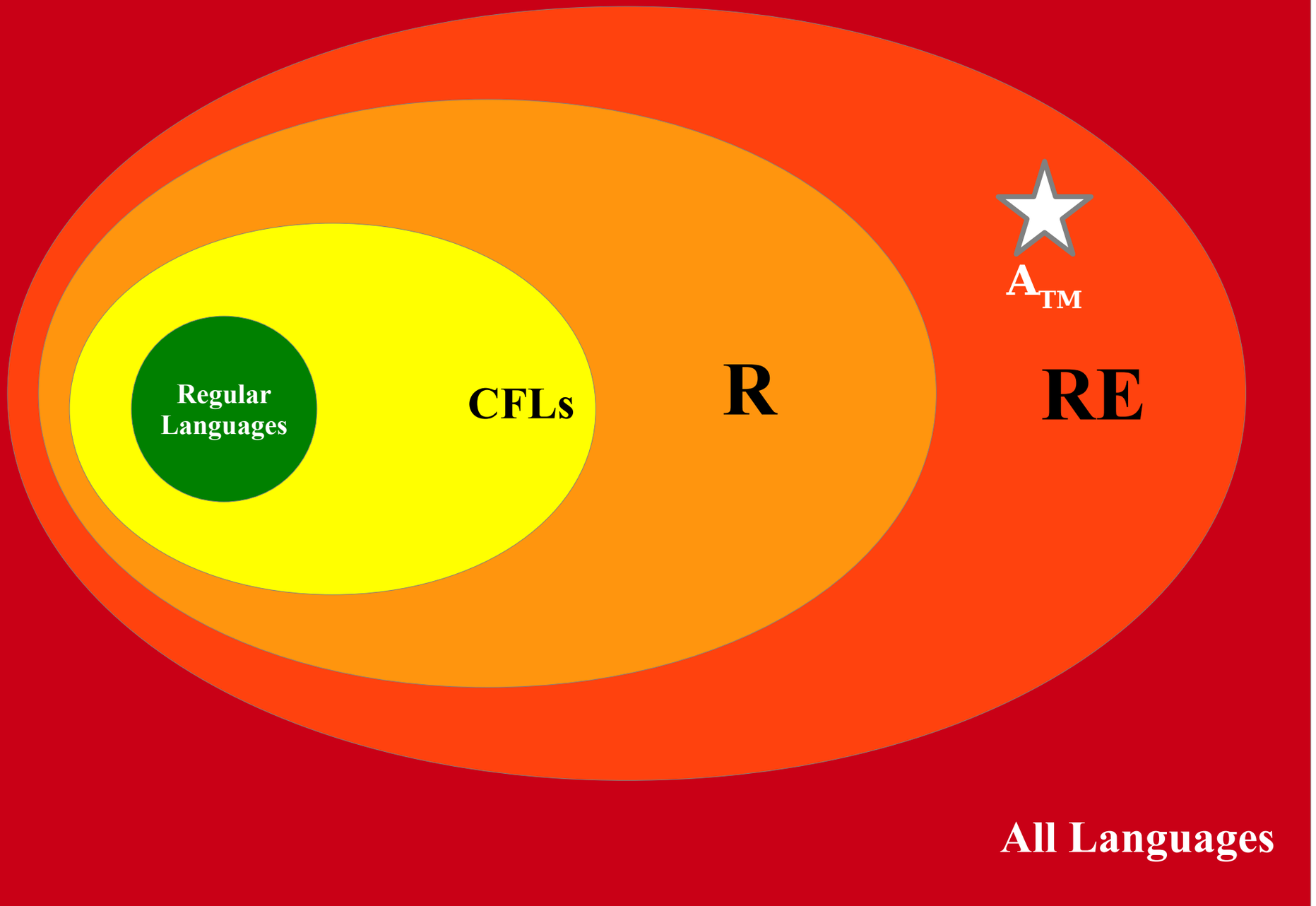
The Class RE


Regular Languages
CFLs
R
RE
A_{TM}
All Languages

More Undecidability Results

The Halting Problem

- The most famous undecidable problem is the ***halting problem***, which asks:

  Given a TM *M* and a string *w*,
  will *M* halt* when run on *w*?

- As a formal language, this problem would be expressed as

  ***HALT* = { ⟨*M*, *w*⟩ | *M* is a TM that halts on *w* }**

- How hard is this problem to solve?

* i.e., accept or reject (“halting” execution, as opposed to infinite loop)

$HALT \in \mathbf{RE}$

- ***Claim:*** $HALT \in \mathbf{RE}$.
- ***Idea:*** We can just run TM M on w, and see if it halts. *(It might not, but that's ok, we just need a recognizer TM to show $HALT \in RE$.)*

```
bool checkHalt(TM M, string w) {
   // This might infinite loop, and if it does, we will
   // not reach the "if" code below this
   bool result = M(w);

   // whether result is true or false, at least M did halt,
   // so we return true
   if (result || !result) {
      return true;
   }
}
```

$HALT$ and A_{TM}

- Comparing recognizer TMs for $HALT$ and A_{TM}

```
bool checkHalt(TM M, string w) {
   // This might infinite loop
   bool result = M(w);

   // Accept both true and false
   if (result || !result) {
      return true;
   }
}
```

```
bool checkATM(TM M, string w) {
   // This might infinite loop
   bool result = M(w);

   // Accept only true
   if (result) {
      return true;
   } else {
      return false;
   }
}
```

$HALT \notin \mathbf{R}$

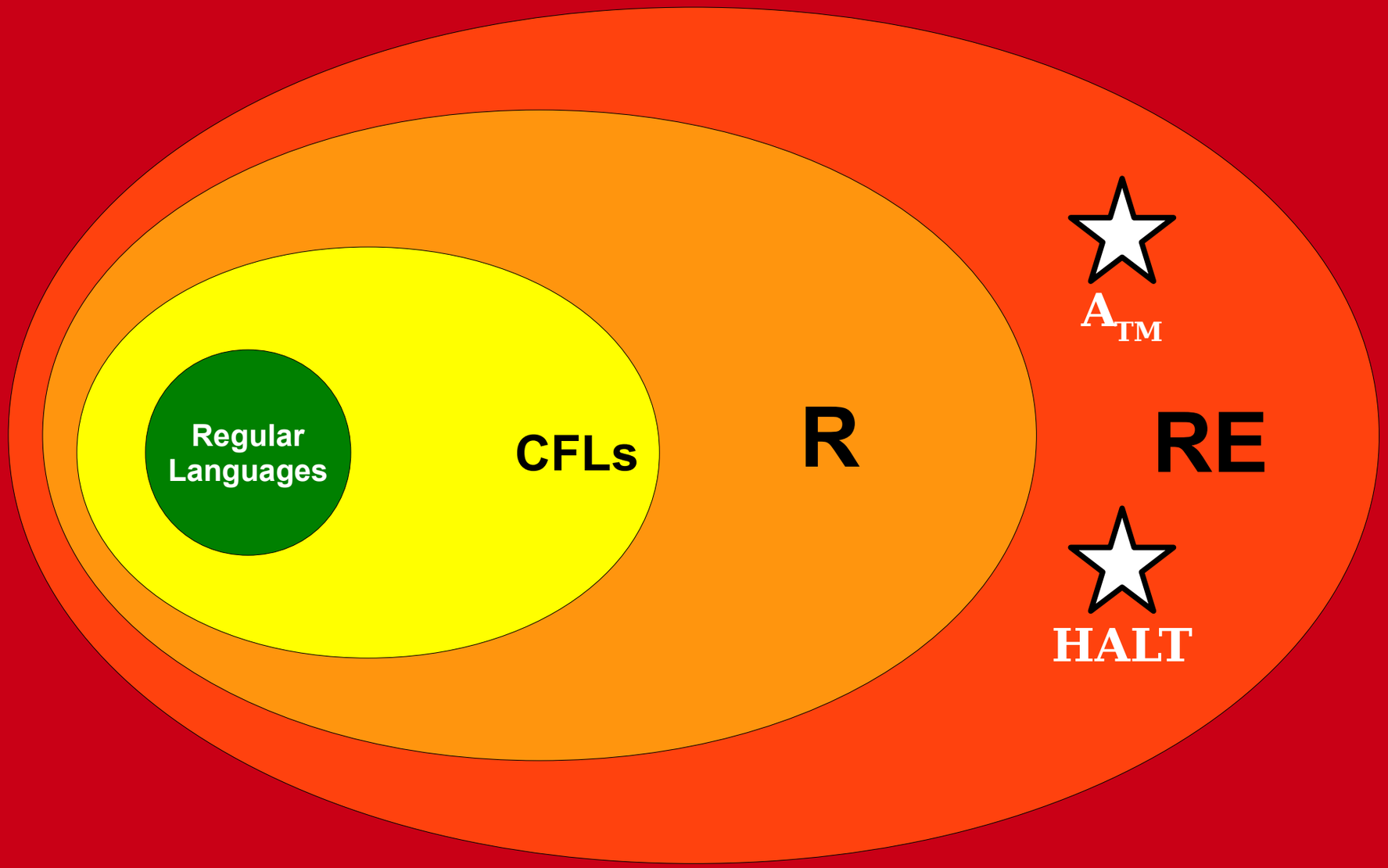
- ***Claim:*** $HALT \notin \mathbf{R}$.
- If *HALT* is decidable, there would exist some *decider* function

  ```
  bool willHalt(TM M, string w)
  ```

  that reports whether the program `M` will halt when run on the given input `w`.
- Then, we could do a trickster setup like we did for A_{TM}.

  (skipping the formal proof for the sake of time, but it follows the same template as A_{TM}, but using this version of trickster)

```
bool trickster(string input) {
   string me = mySource();
   if (willHalt(me, input)) {
      while (true) {
         // loop infinitely
      }
   } else {
      return true;
   }
}
```


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

The Class RE

- Languages L that are in RE, *but not R*, are those where:
  - We can build a TM M, where $\mathcal{L}(M) = L$
  - That TM M has the risk of getting stuck in an infinite loop for at least one input string that is not in L
- **Next:** Just like the class Regular was defined in multiple ways (DFAs, NFAs, RegExes), today we'll learn **another way** to define this class RE!

Verification

```
    if (errorObj && errorObj.stack && errorObj.stack.indexOf('dynamicAddScript') !== -1) {
      var errorMessage = {
        Message: event.message,
        LineNo: event.lineno,
        ColumnNo: event.colno,
        Stack: errorObj.stack
      }
    }

  }

  var ezDynamic = document.createElement('script');
  ezDynamic.type = 'text/javascript';
  ezDynamic.innerHTML = script;
  document.head.appendChild(ezDynamic);

  window.removeEventListener('error', errorHandler);
  if (window.ezFinishedStatic !== true || typeof window.ezstaticerrors !== 'undefined') {
      d.Script = script;
      d.ErrorMessages = (errorMessages || '') + (window.ezstaticerrors || '');
      var dataTxt = JSON.stringify(d);
      if (dataTxt.length > 0) {
          var logXHR = new XMLHttpRequest()
          logXHR.open('POST','/ezais/log?cb=1', true);
          logXHR.setRequestHeader('Content-Type','application/json');
          logXHR.send(dataTxt);
```

```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input “abbababababbbb”?

.SOURCE file open in Microsoft Visual Studio Code 1.85. © FileInfo.com

Verification

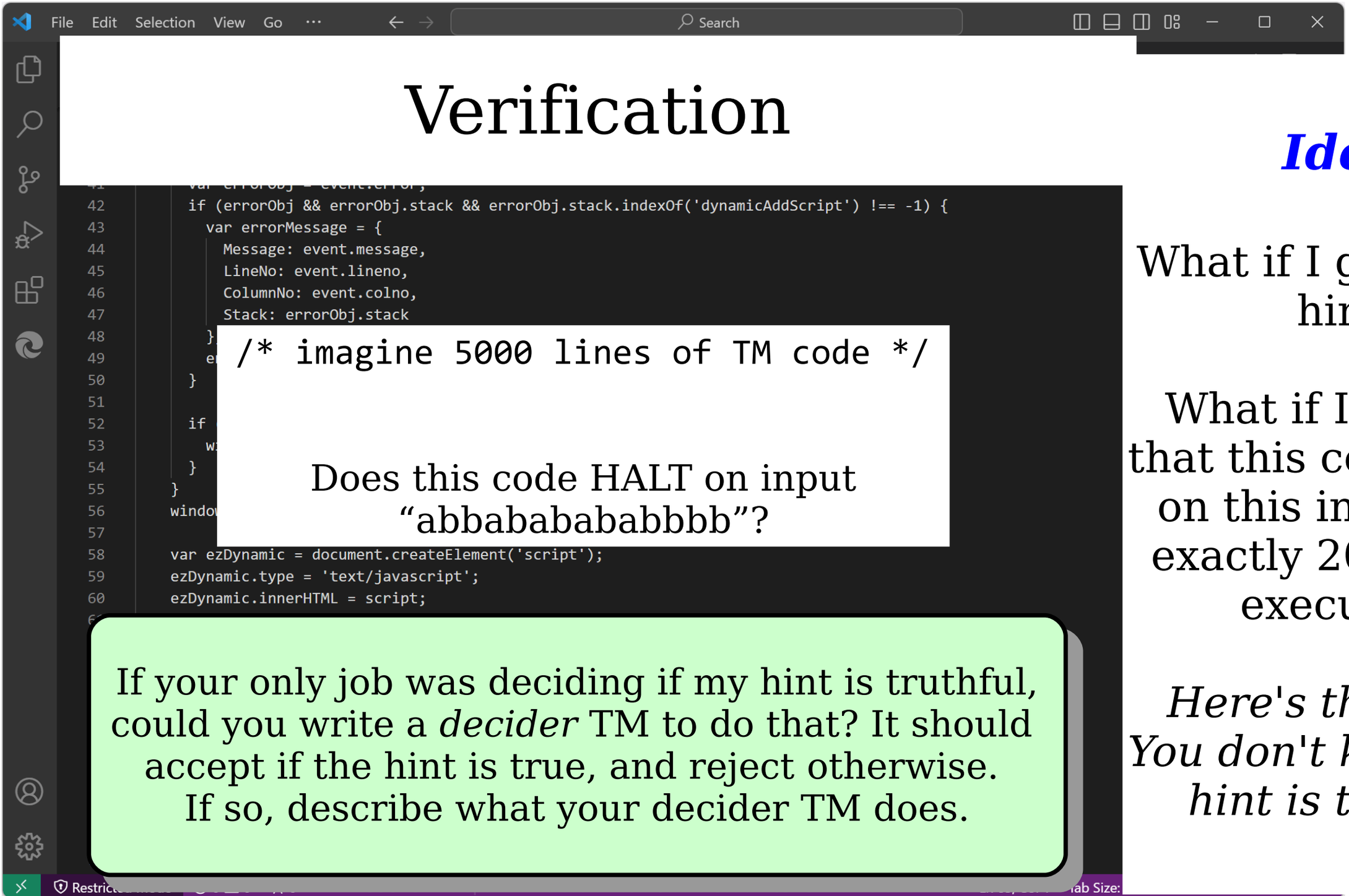
```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input “abbababababbbb”?

Reflection:

Hard for humans! We know that TMs are also not able to answer this in the general case without the possibility of looping (*HALT* in RE, not R).

Verification



If your only job was deciding if my hint is truthful, could you write a *decider* TM to do that? It should accept if the hint is true, and reject otherwise. If so, describe what your decider TM does.

Idea:

What if I gave you a hint?

What if I told you that this code HALTs on this input after exactly 20 steps of execution.

Here's the catch: You don't know if my hint is truthful.

Verification

```
    if (errorObj && errorObj.stack && errorObj.stack.indexOf('dynamicAddScript') !== -1) {
      var errorMessage = {
        Message: event.message,
        LineNo: event.lineno,
        ColumnNo: event.colno,
        Stack: errorObj.stack
      }
    }

  }
  windo

var ezDynamic = document.createElement('script');
ezDynamic.type = 'text/javascript';
ezDynamic.innerHTML = script;
```

```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input "abbababababbbb"?

If your only job was deciding if my hint is truthful, could you write a *decider* TM to do that? It should accept if the hint is true, and reject otherwise. If so, describe what your decider TM does.

Decider TM:

Run the code on the input for exactly 20 steps of execution.

If on the 20^{th} step we observe it HALT, *accept*.

Otherwise *reject* (hint was not truthful).

Verification

```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input “abbababababbbb”?

Example:

Someone gives us the number of steps hint “20.” We run the TM on the input “abbababababbbb” and observe the TM **accepts** the input on step 20.

This $\langle M, w \rangle$ is in the language *HALT*.

Verification

```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input “abbababababbbb”?

Is this $\langle M, w \rangle$ in the language HALT?

Example:

Someone gives us the number of steps hint “20.” We run the TM on the input “abbababababbbb” and observe the TM **rejects** the input on step 20.

Verification

```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input “abbababababbbb”?

Is this $\langle M, w \rangle$ in the language HALT?

Example:

Someone gives us the number of steps hint “20.” We run the TM on the input “abbababababbbb” and observe the TM **rejects** the input on step 20.

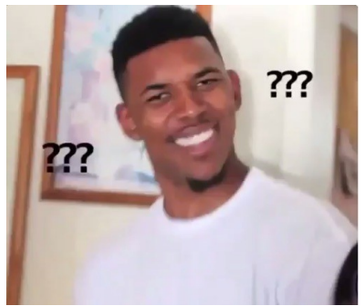
This $\langle M, w \rangle$ is in the language *HALT*.

Verification

```
/* imagine 5000 lines of TM code */
```

Does this code HALT on input “abbababababbbb”?

Is this $\langle M, w \rangle$ in the language HALT?



Example:

Someone gives us the number of steps hint “20.” We run the TM on the input “abbababababbbb” and observe the TM **has neither accepted nor rejected** the input after 20 steps (*M* would keep running if we let it, but we stop after 20 steps).

Is this $\langle M, w \rangle$ in the language *HALT*?

Verification

- In each of the preceding cases, we were given some TM M and some input string w, and some evidence/hint supporting the claim that M halts on w.
- Given a truthful evidence/hint, we can write a decider TM to see that the answer is indeed “yes,” M halts on w.
- Given false evidence, **we aren't immediately sure whether M halts on w.**
  - Maybe there's *no* evidence saying that the answer is “yes,” because the answer is no, M doesn't halt on w!
  - Or maybe there is some evidence, but just not the evidence we were given.
- Let's formalize this idea.

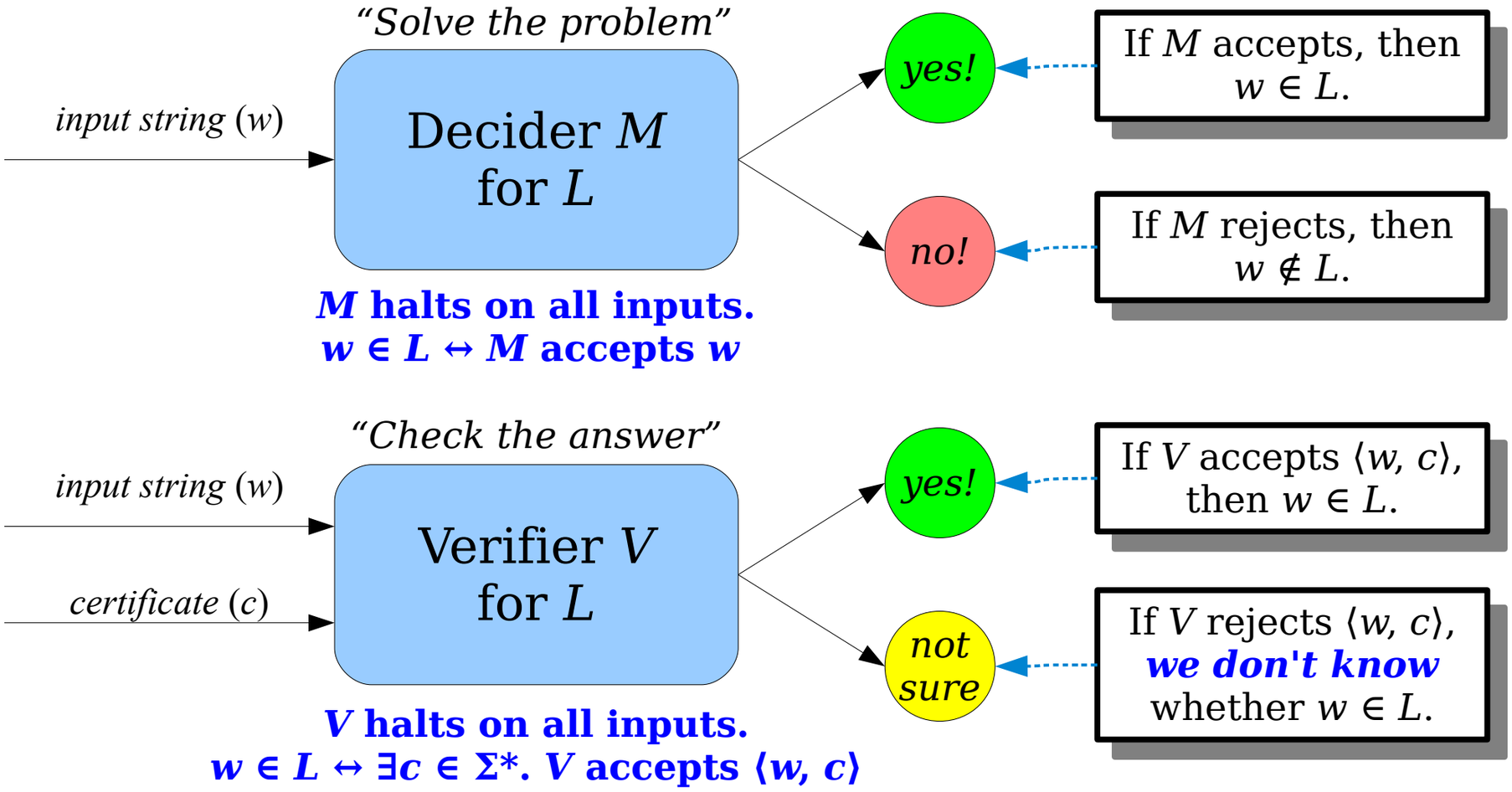
Verifiers

- 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:

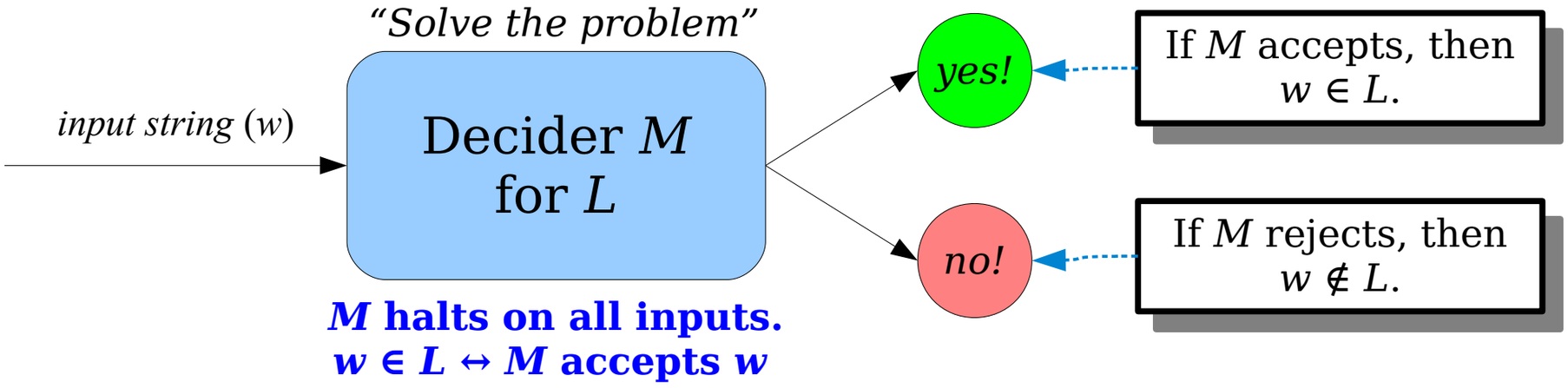
$$\boldsymbol{w \in L \iff \exists c \in \Sigma^*.\ V \text{ accepts } \langle w, c \rangle}$$

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

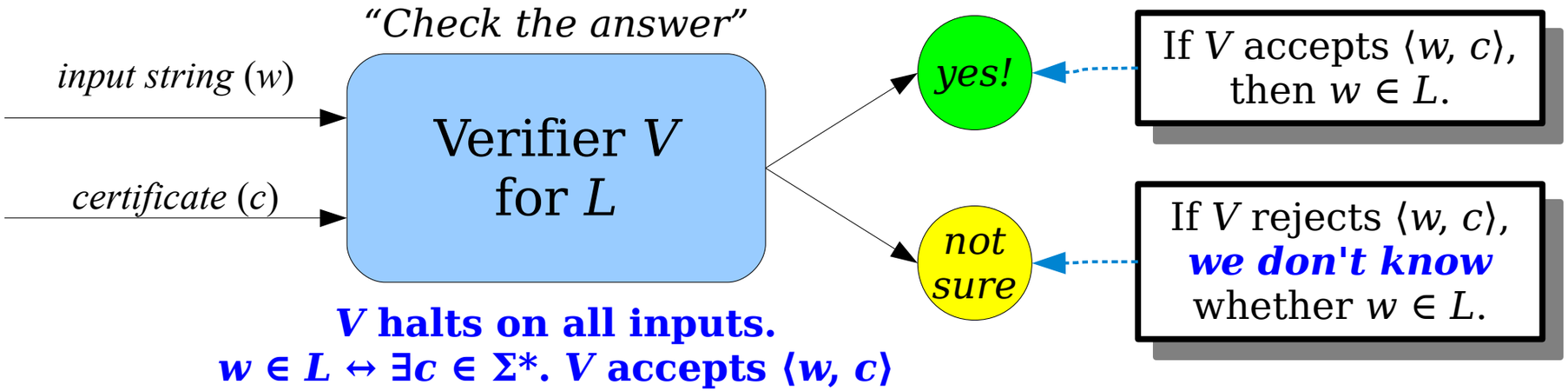
Deciders and Verifiers



Deciders and Verifiers



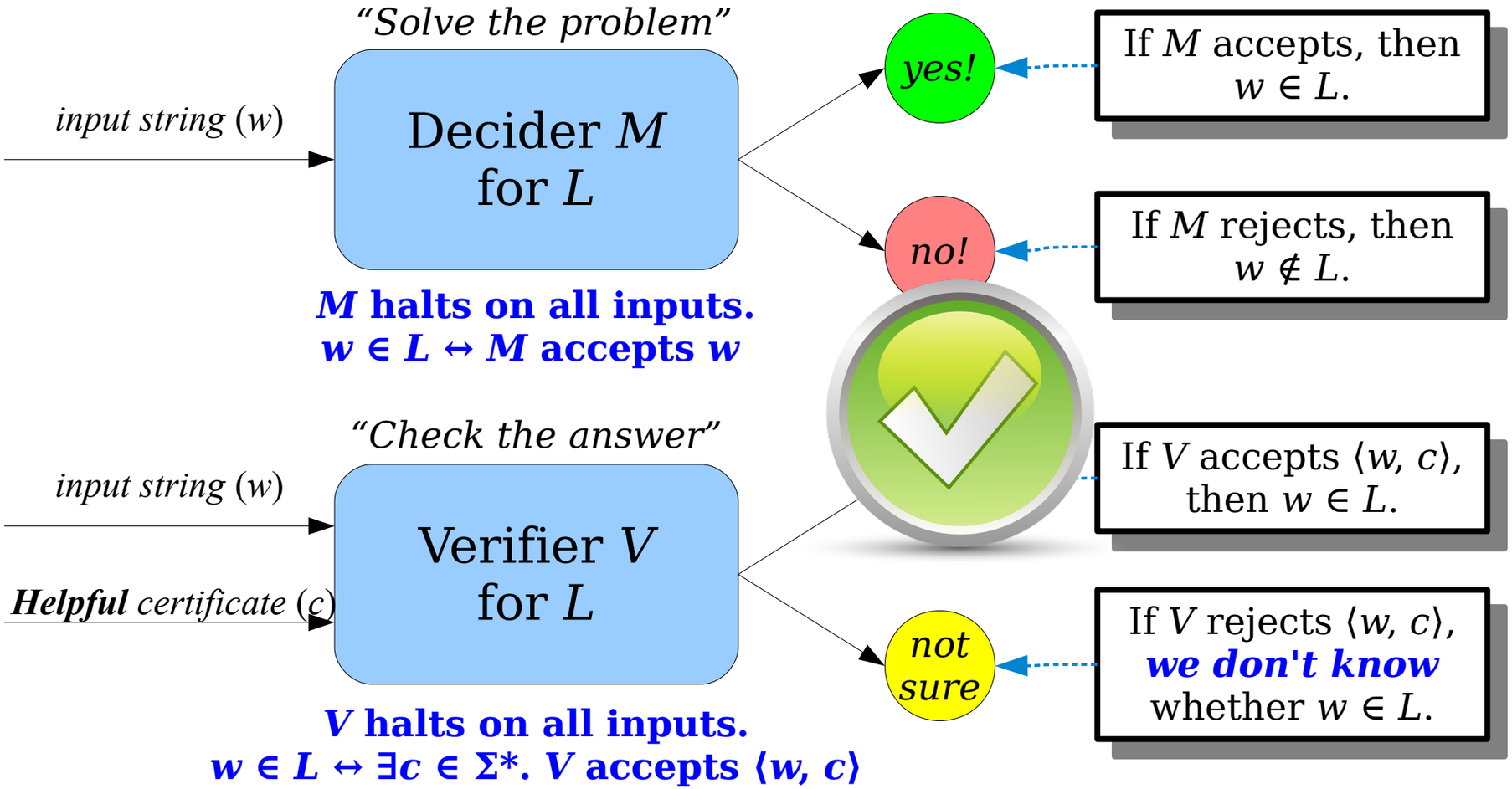
***M* halts on all inputs.**
w* ∈ *L* ↔ *M* accepts *w



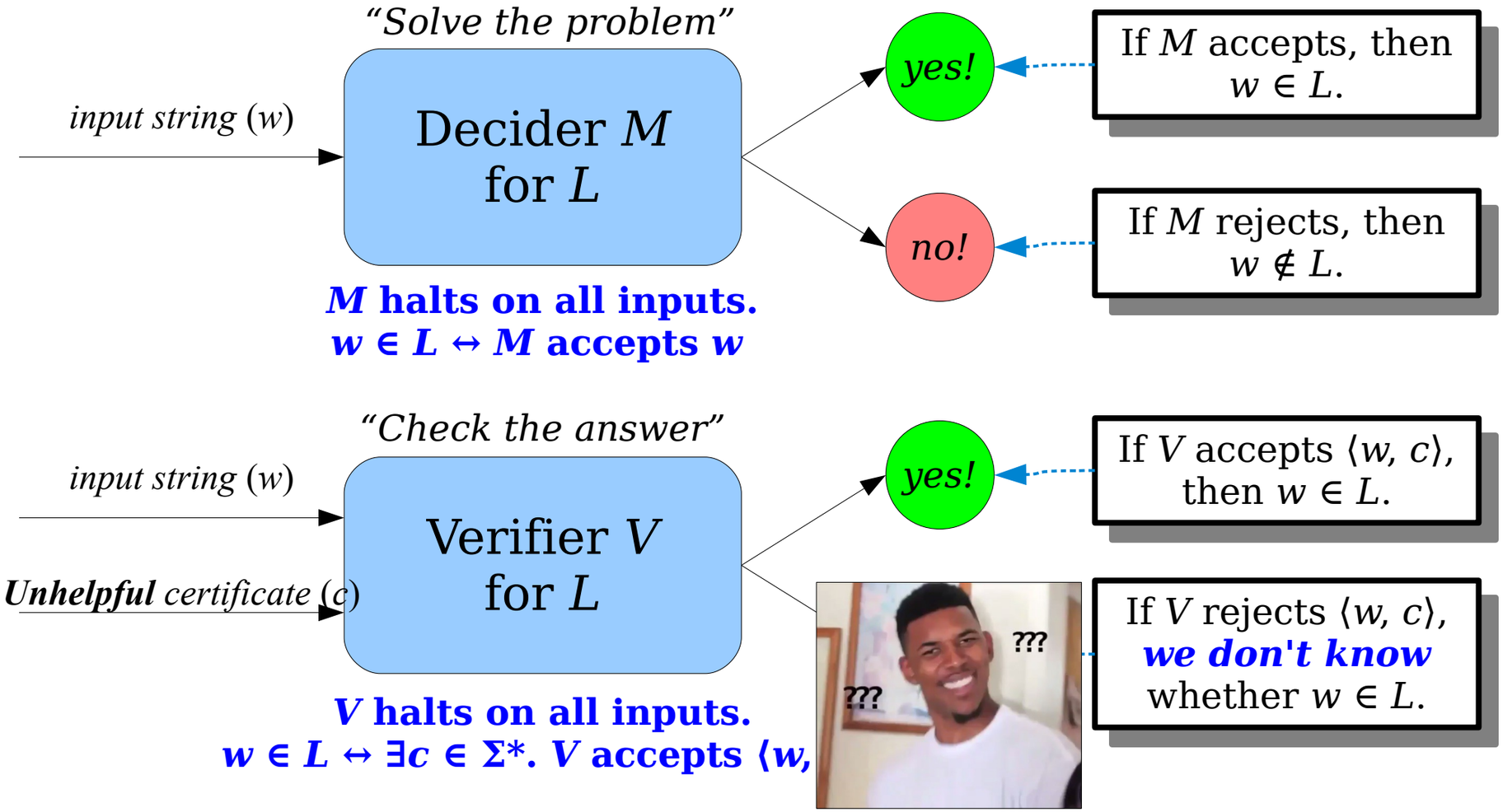
***V* halts on all inputs.**
***w* ∈ *L* ↔ ∃*c* ∈ Σ*. *V* accepts ⟨*w*, *c*⟩**

"certificate" is the official term for a "hint"

Deciders and Verifiers



Deciders and Verifiers



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:

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

- 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 $\mathcal{L}(V) \neq L$. *(Good question to hold on to for a second: what <u>is</u> $\mathcal{L}(V)$?)*
  - 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:

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

- 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;
}
```

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;
}
```

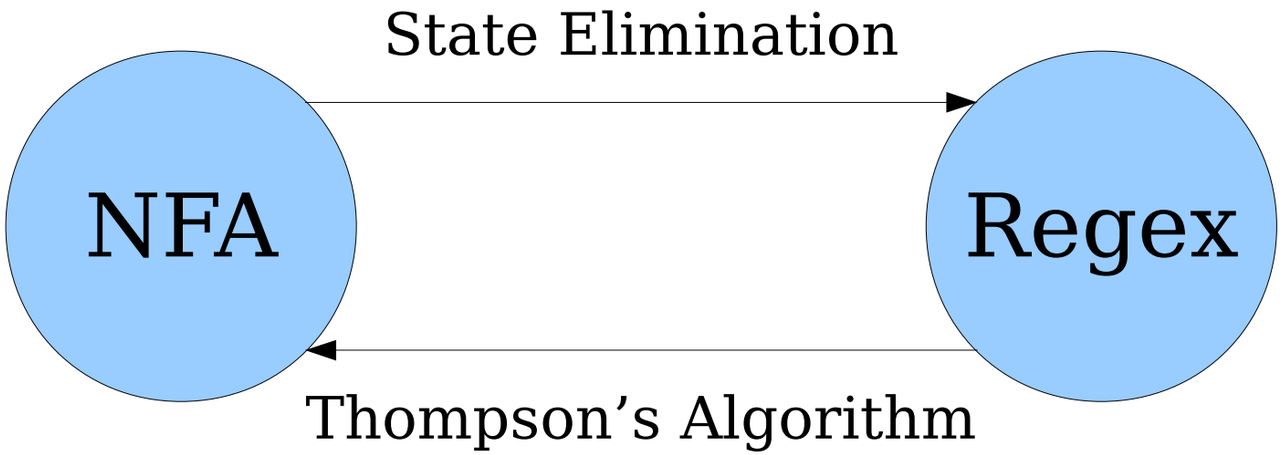
How many of these statements are true of $\mathcal{L}(V)$?

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

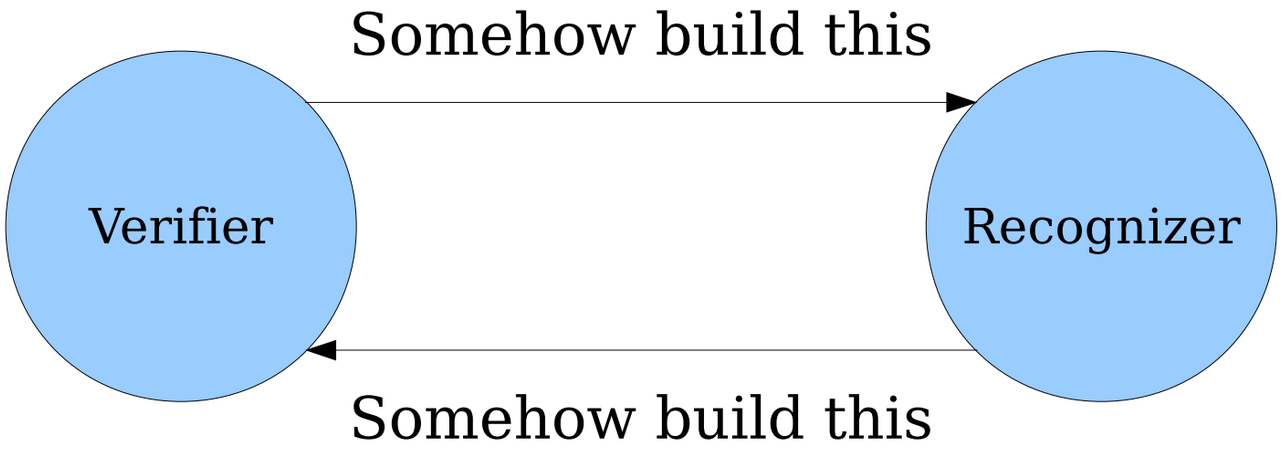
$L = \{ \langle n \rangle \mid n \in \mathbb{N}$ and the hailstone sequence terminates for $n \}$

```
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;
}
```

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 our standard 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!

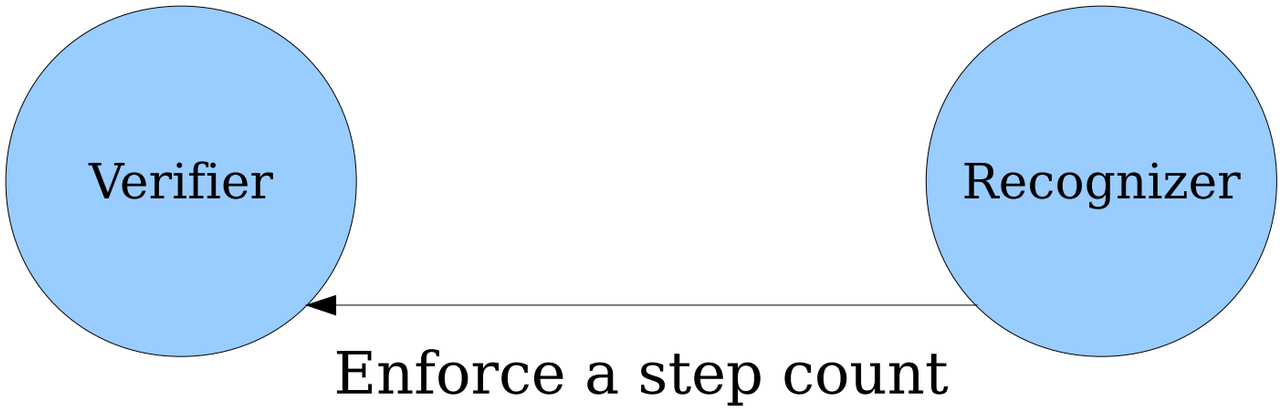
A Verifier for A_{TM}

- 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



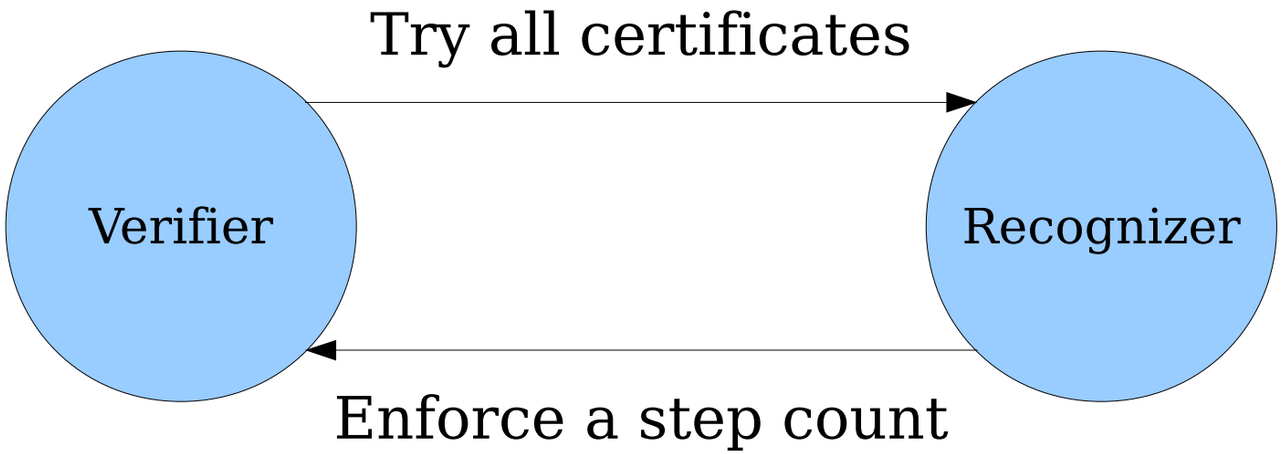
What languages are verifiable?

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?

Equivalence of Verifiers and Recognizers



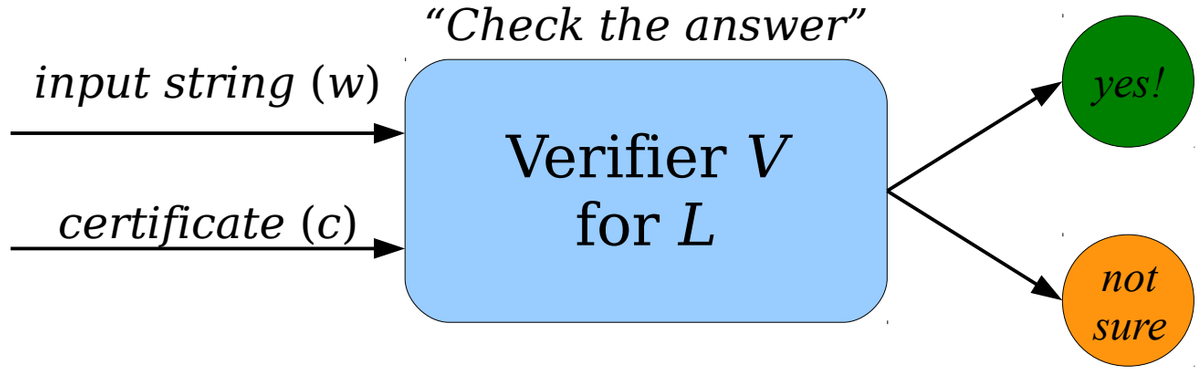
Theorem: If L is a language, then there is a verifier for L if and only if $L \in \mathbf{RE}$.

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.

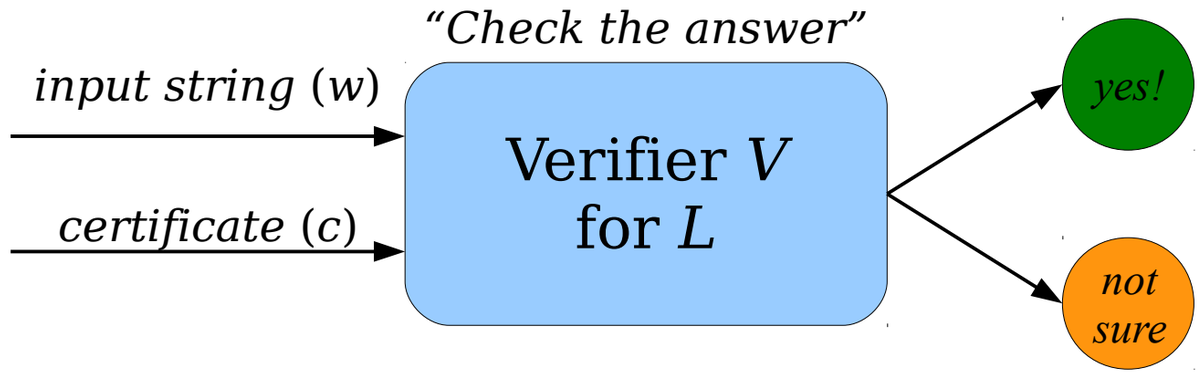
Verifiers and **RE**

- ***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.

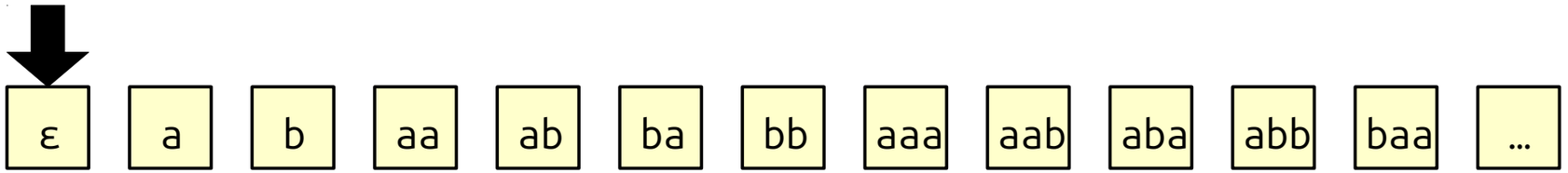


Verifiers and **RE**

- ***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.

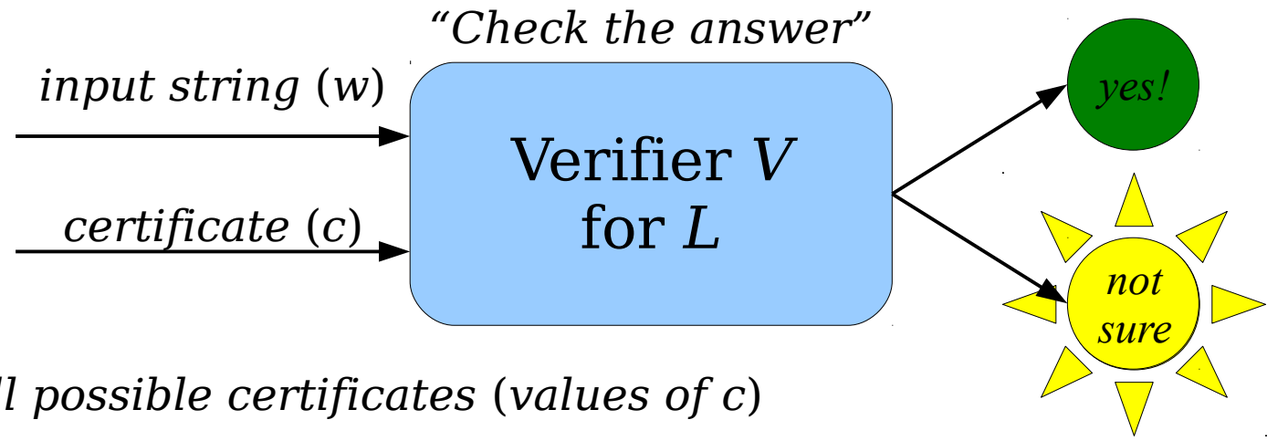


We will try all possible certificates (*values of c*)

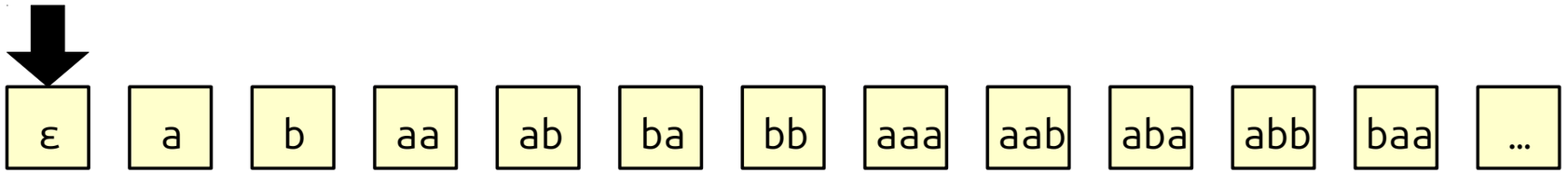


Verifiers and **RE**

- ***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.

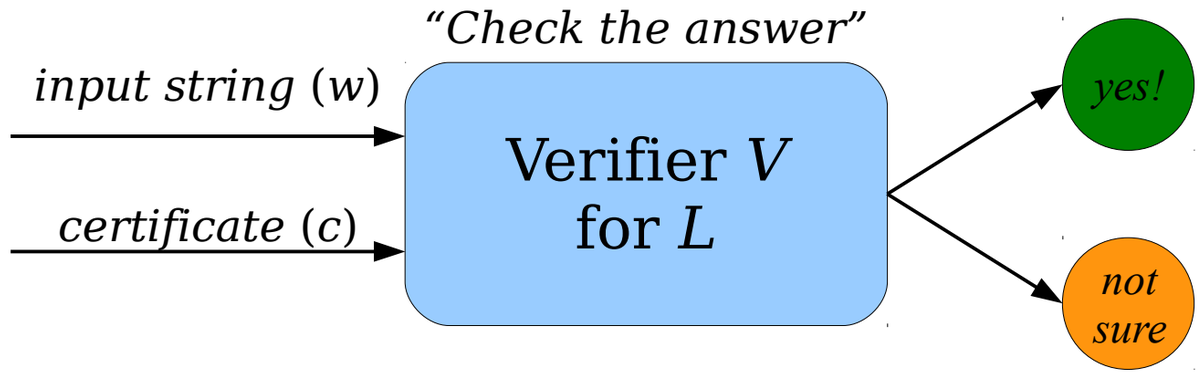


We will try all possible certificates (*values of c*)

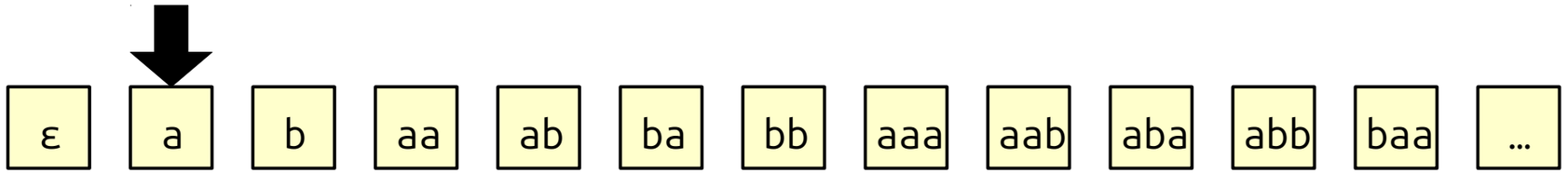


Verifiers and **RE**

- ***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.

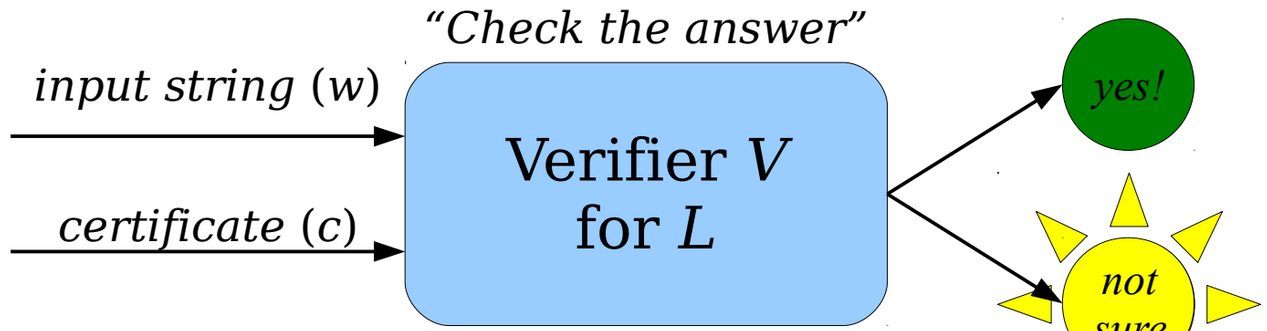


We will try all possible certificates (*values of c*)

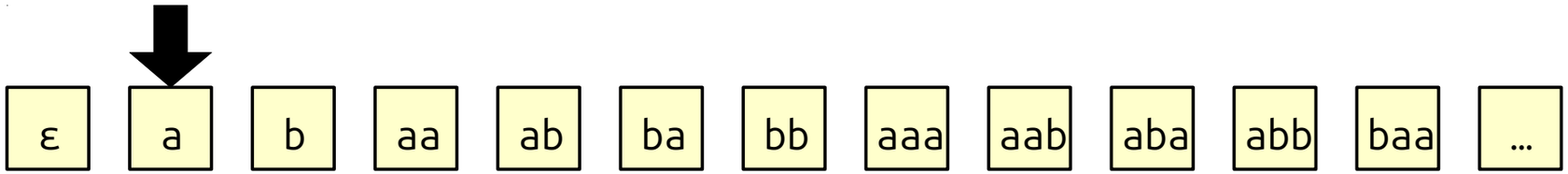


Verifiers and **RE**

- ***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.

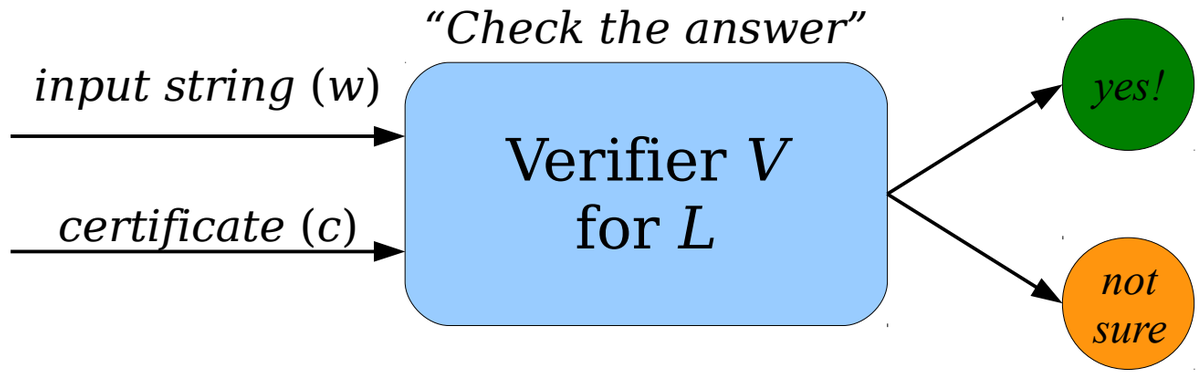


We will try all possible certificates (*values of c*)

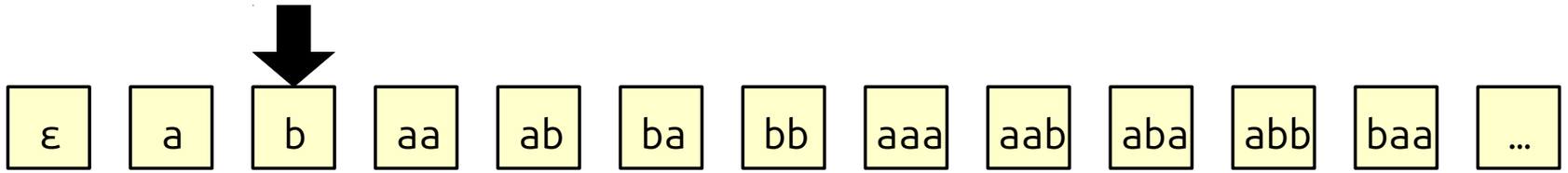


Verifiers and **RE**

- ***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.

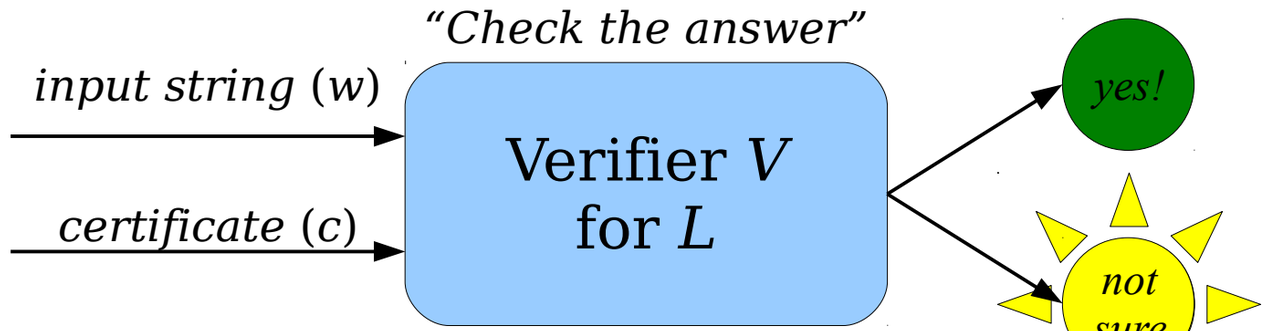


We will try all possible certificates (*values of c*)

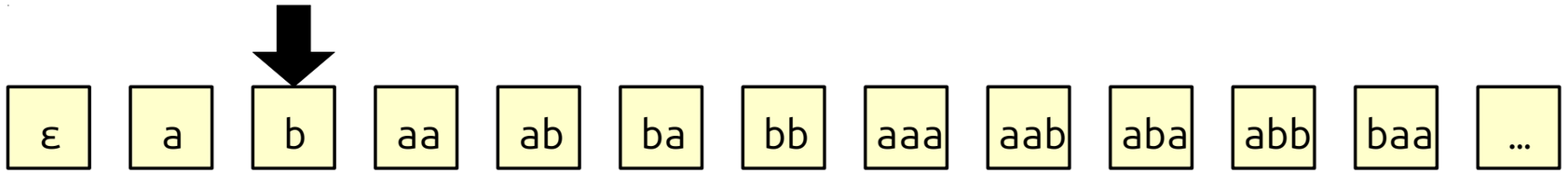


Verifiers and **RE**

- ***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.

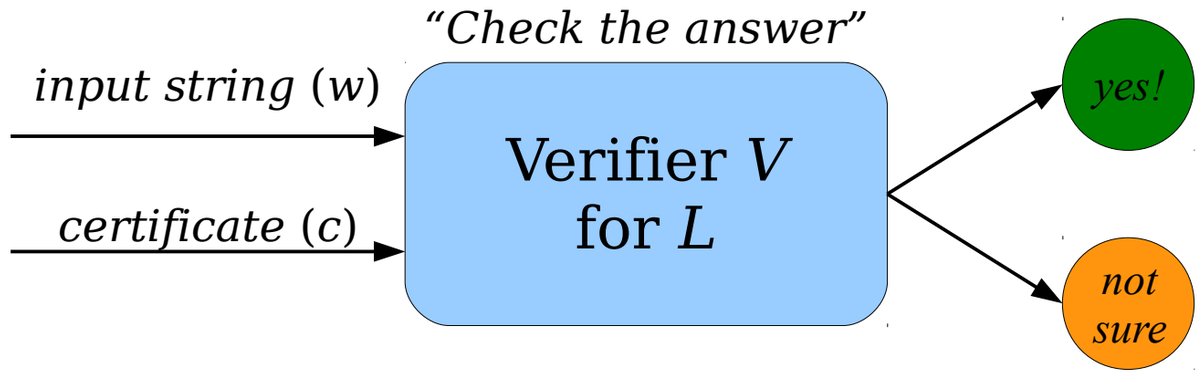


We will try all possible certificates (*values of c*)

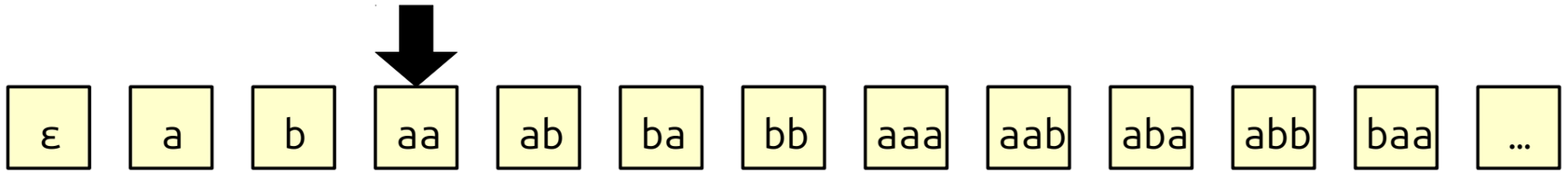


Verifiers and **RE**

- ***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.

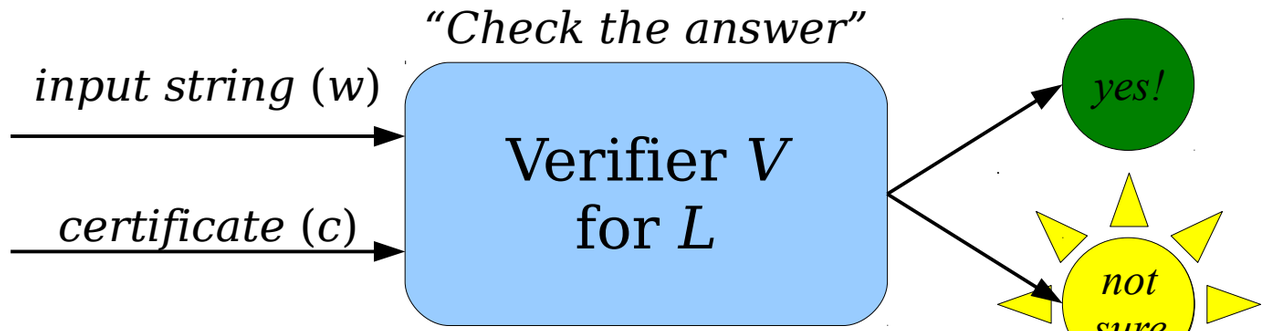


We will try all possible certificates (*values of c*)

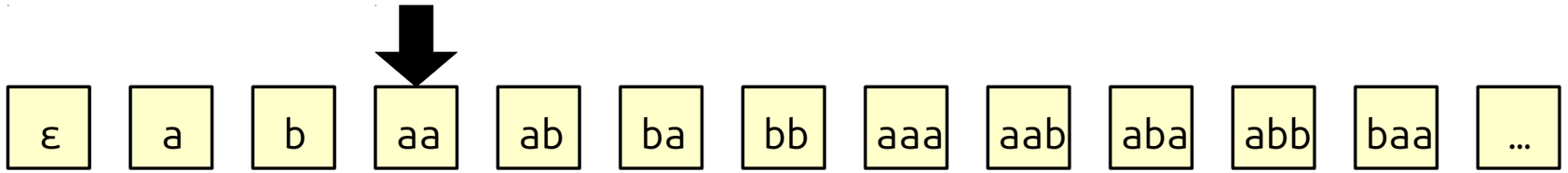


Verifiers and **RE**

- ***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.

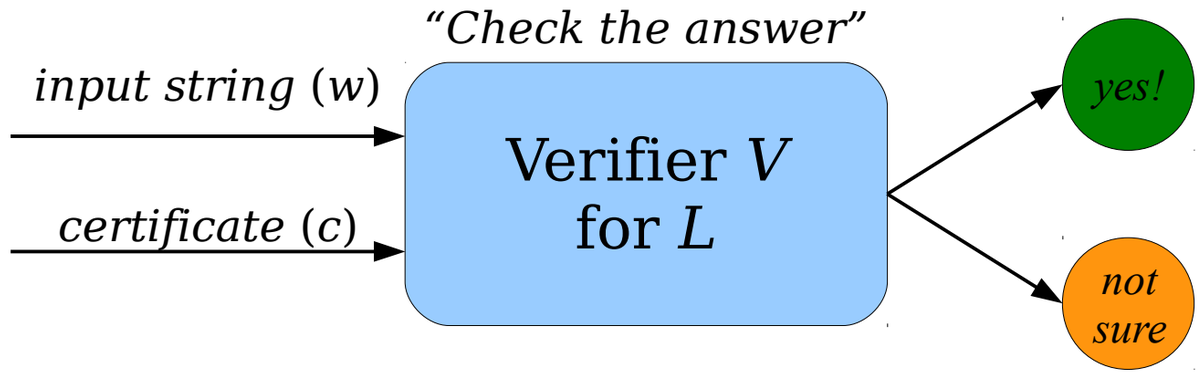


We will try all possible certificates (*values of c*)

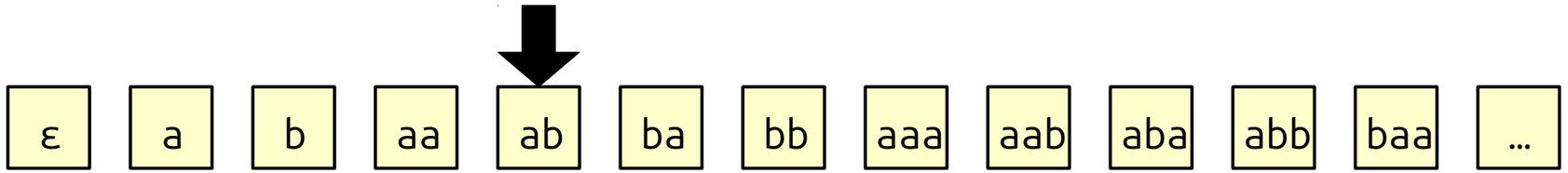


Verifiers and **RE**

- ***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.

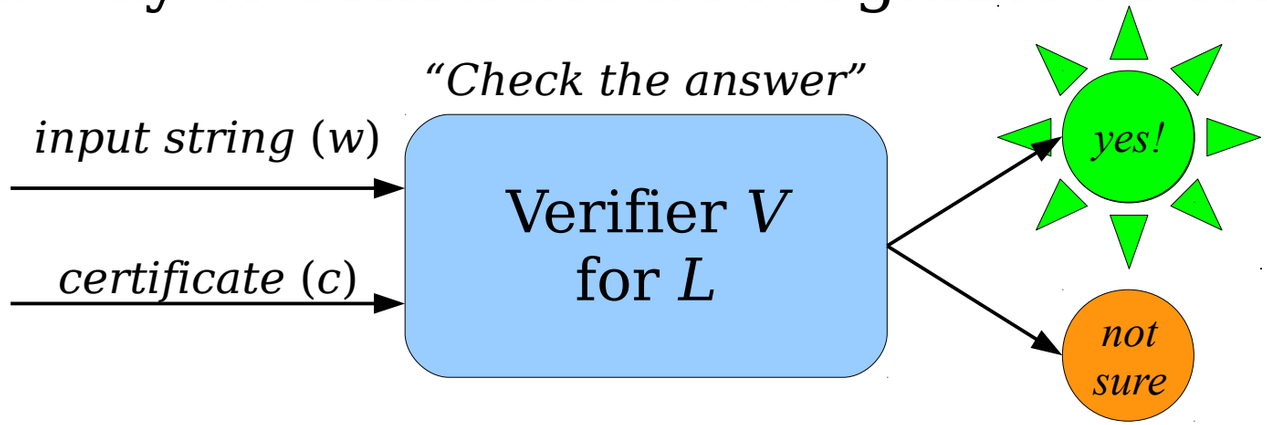


We will try all possible certificates (*values of c*)

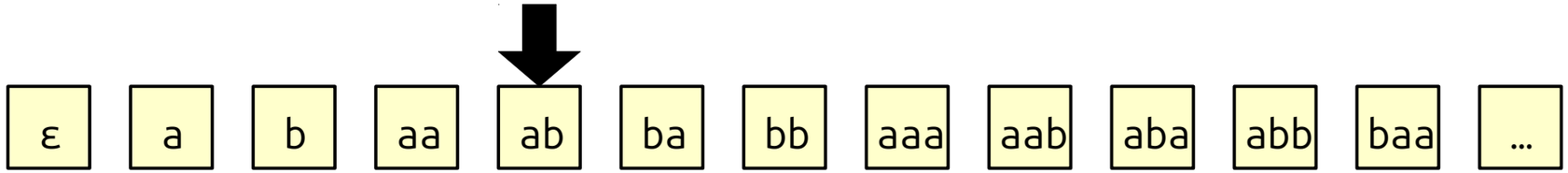


Verifiers and **RE**

- ***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.

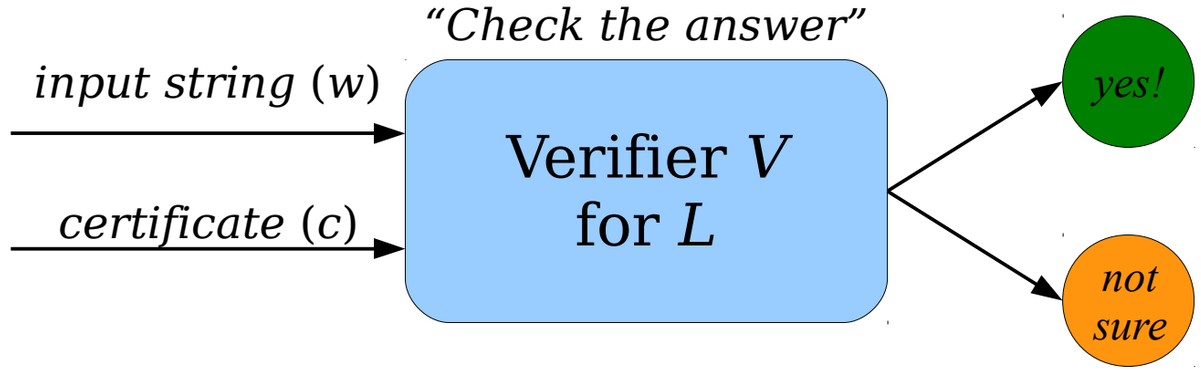


We will try all possible certificates (*values of c*)



Verifiers and **RE**

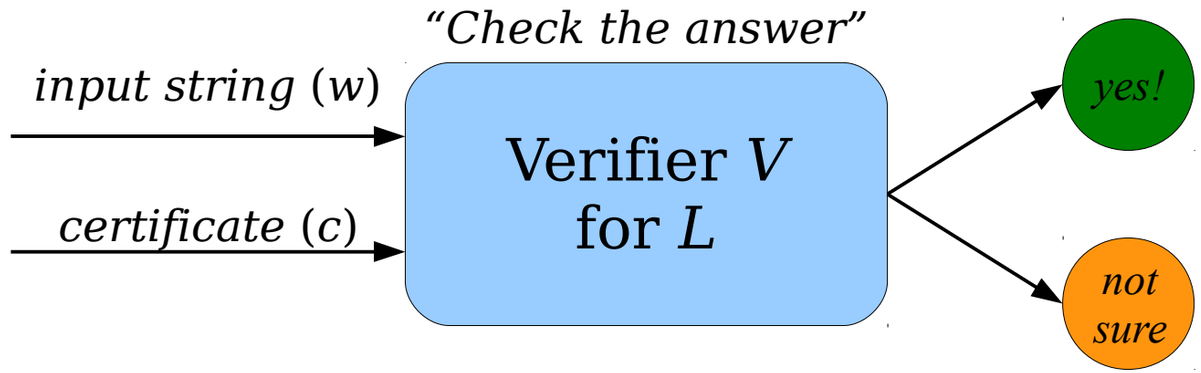
- ***Theorem:*** If there is a verifier V for a language L, then $L \in \mathbf{RE}$.
- ***Proof goal:*** Given a verifier V for a language L, find a way to construct a recognizer M for L.



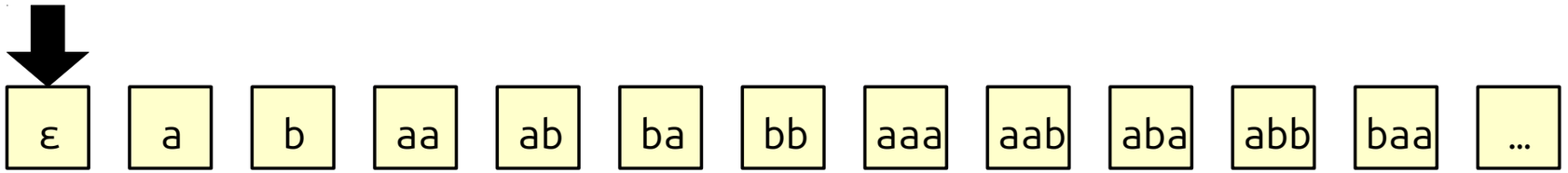
ε	a	b	aa	ab	ba	bb	aaa	aab	aba	abb	baa	…

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.

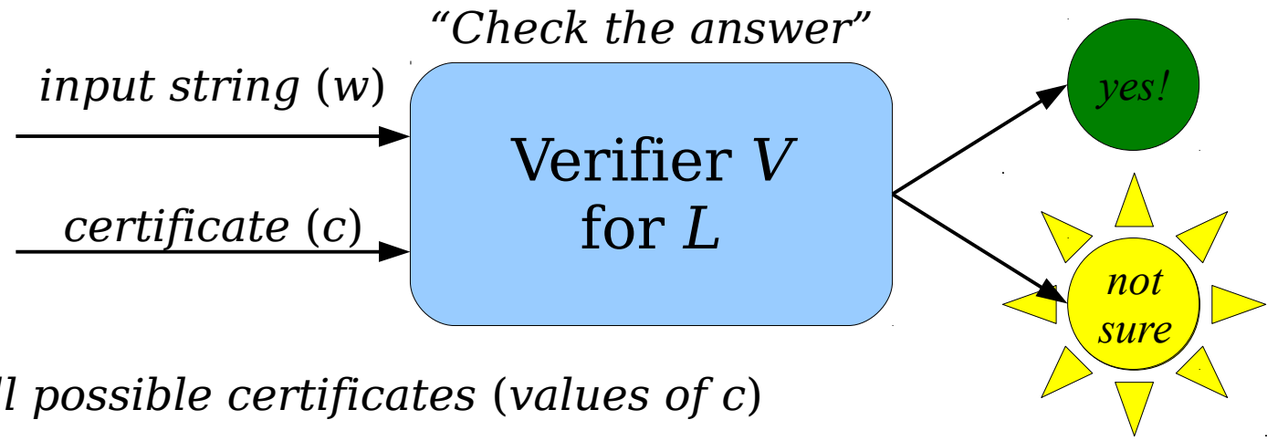


We will try all possible certificates (*values of c*)

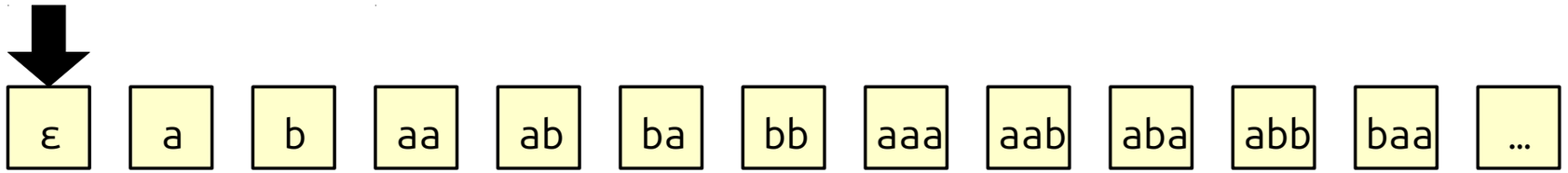


Verifiers and **RE**

- ***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.

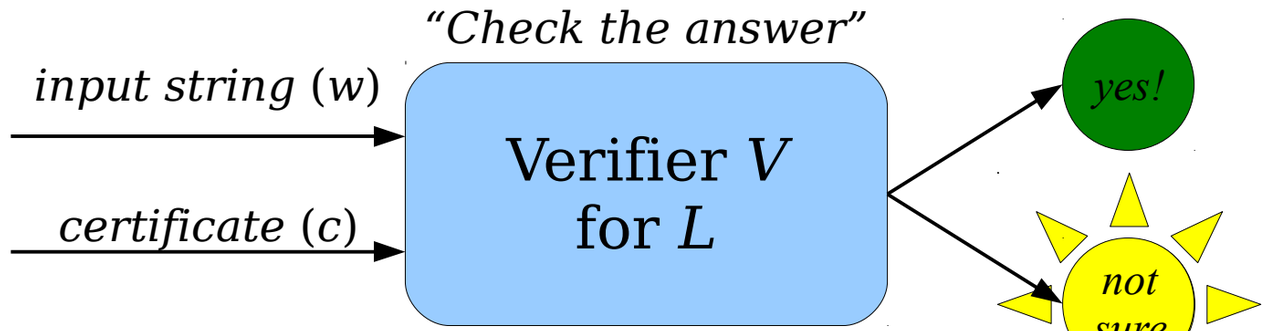


We will try all possible certificates (*values of c*)

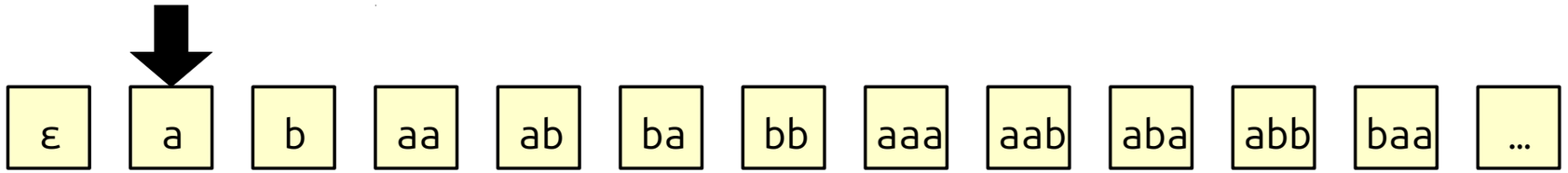


Verifiers and **RE**

- ***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.

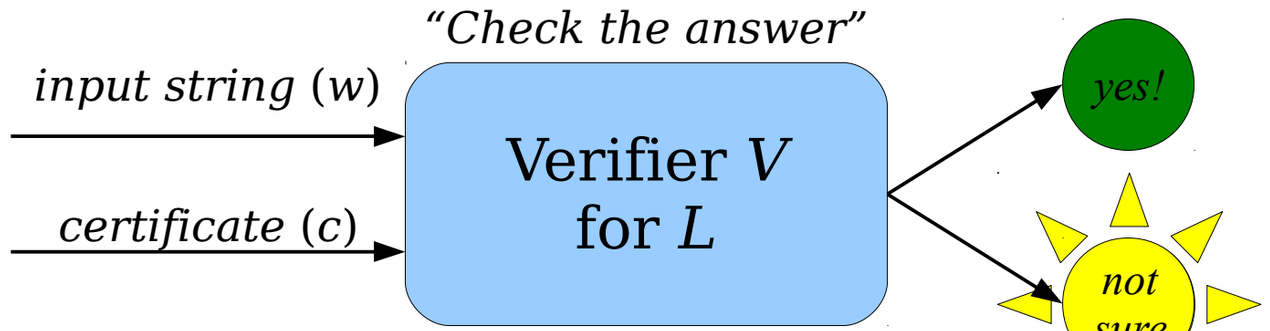


We will try all possible certificates (*values of c*)

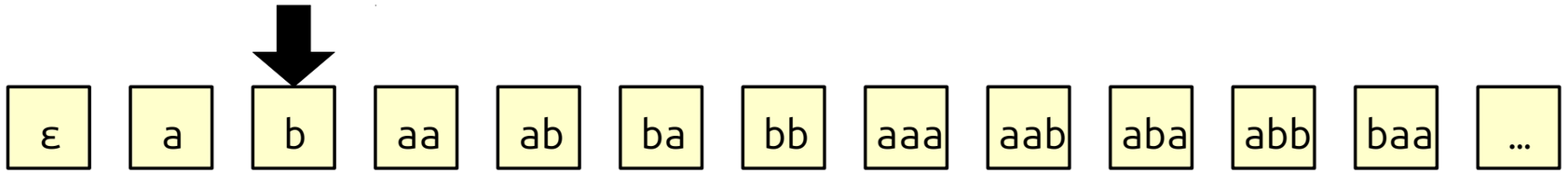


Verifiers and **RE**

- ***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.

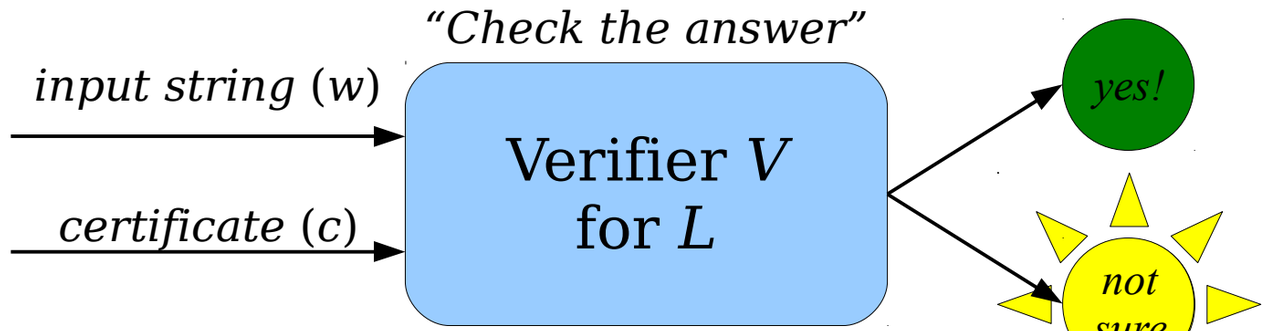


We will try all possible certificates (*values of c*)

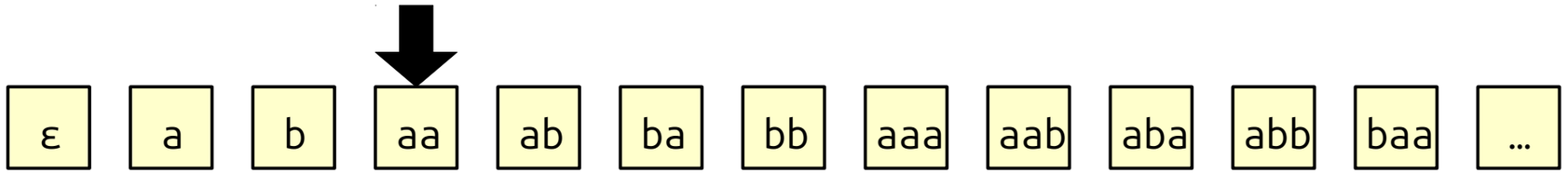


Verifiers and **RE**

- ***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.

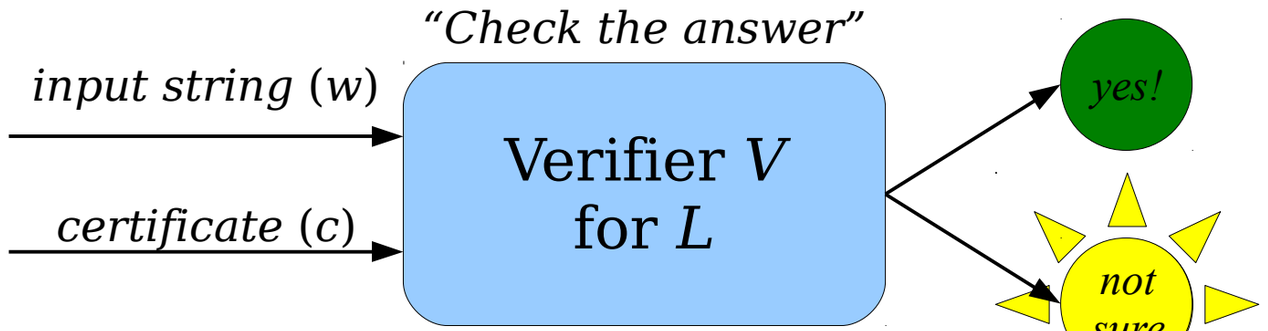


We will try all possible certificates (*values of c*)

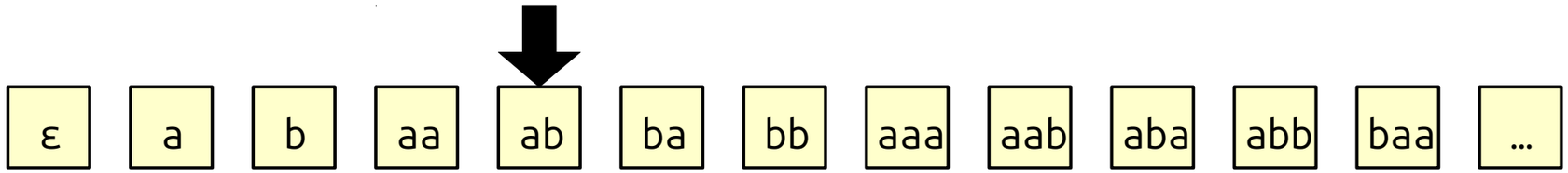


Verifiers and **RE**

- ***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.

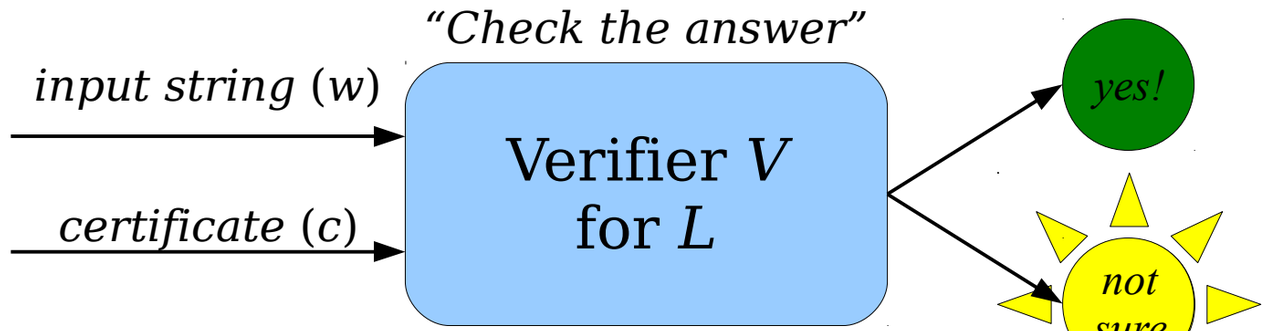


We will try all possible certificates (*values of c*)

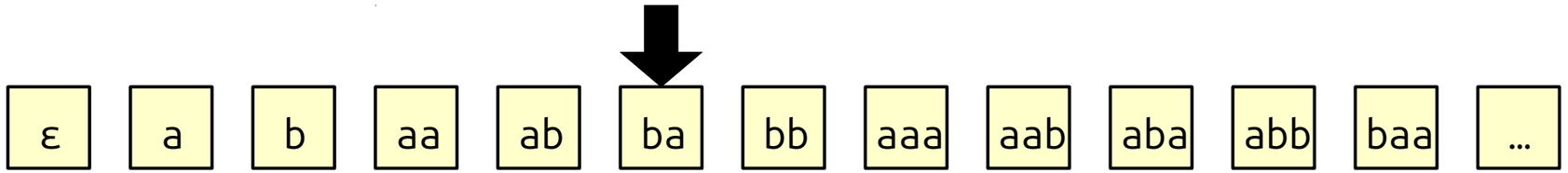


Verifiers and **RE**

- ***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.

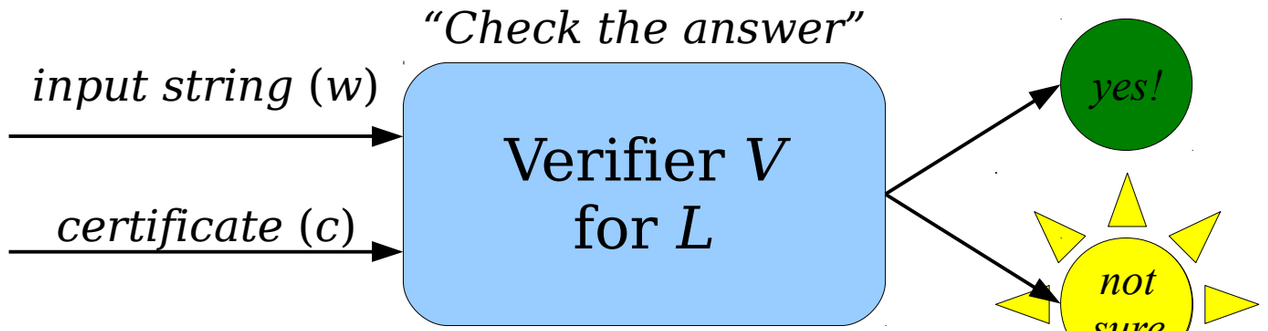


We will try all possible certificates (values of c)

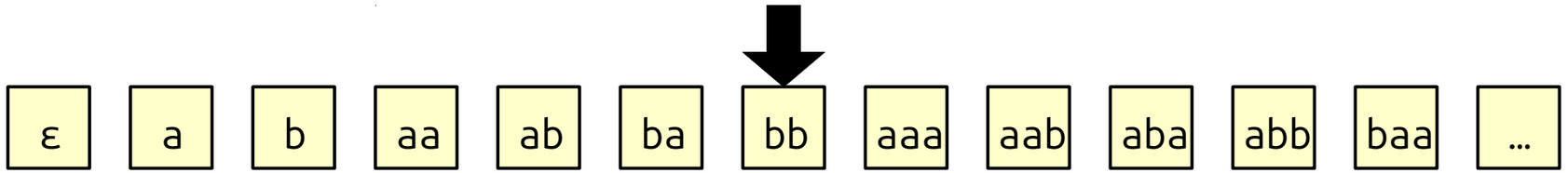


Verifiers and **RE**

- ***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.

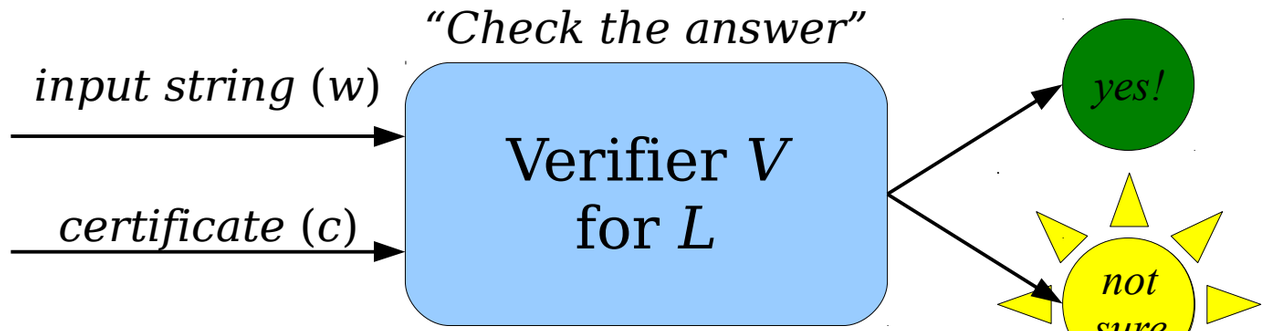


We will try all possible certificates (*values of c*)

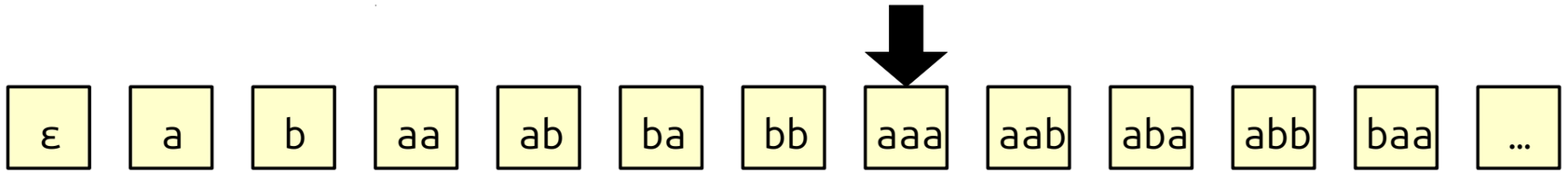


Verifiers and **RE**

- ***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.

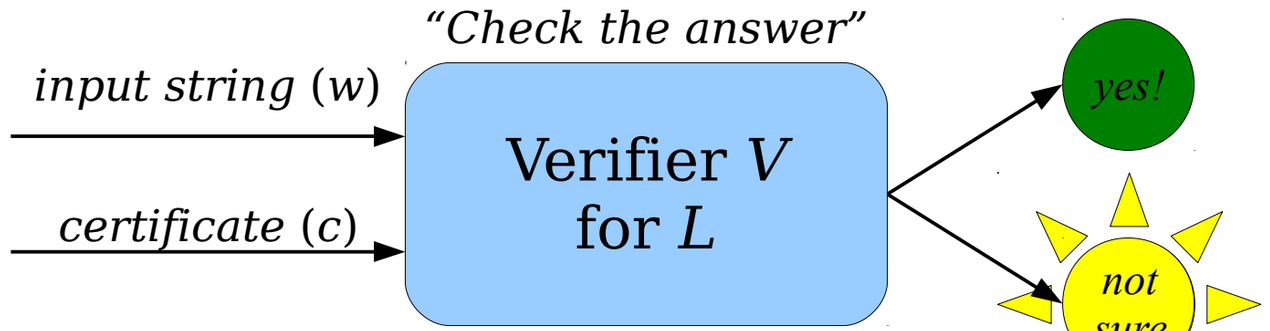


We will try all possible certificates (values of c)

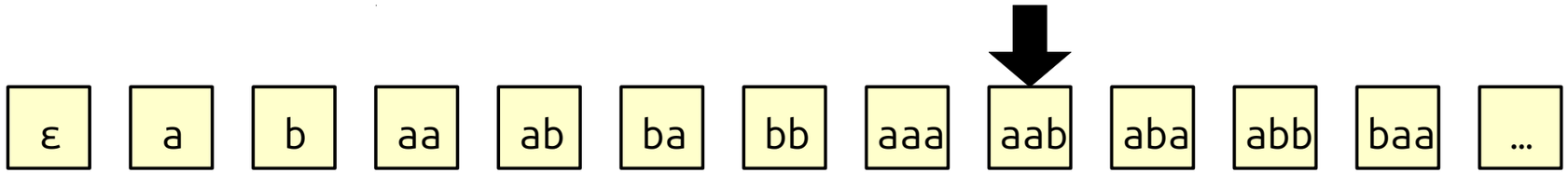


Verifiers and **RE**

- ***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.

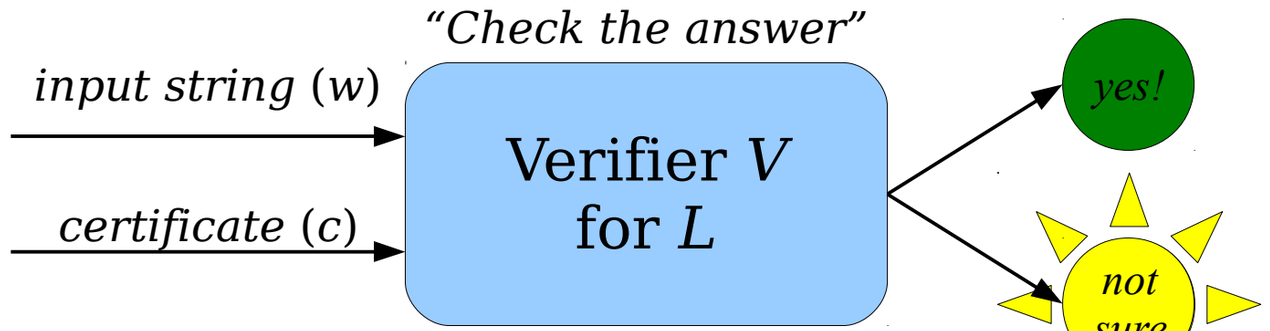


We will try all possible certificates (*values of c*)

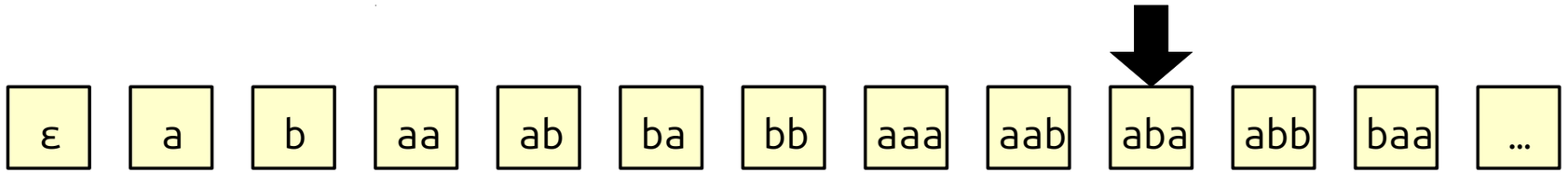


Verifiers and **RE**

- ***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.

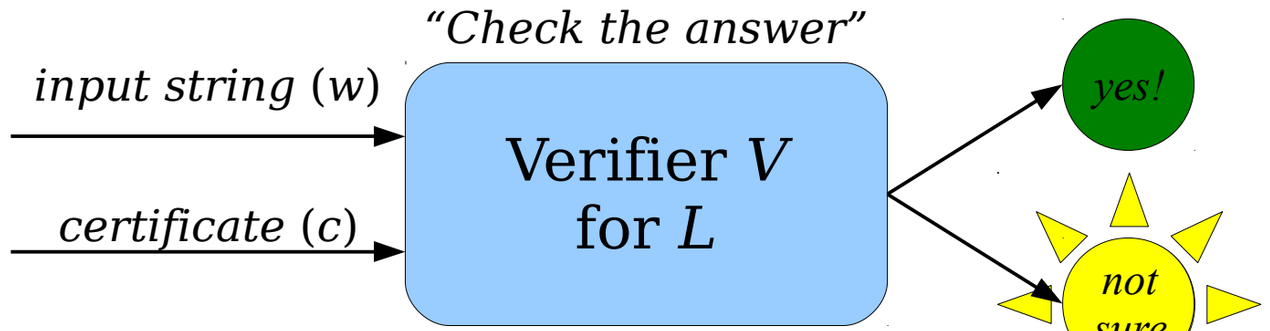


We will try all possible certificates (values of c)

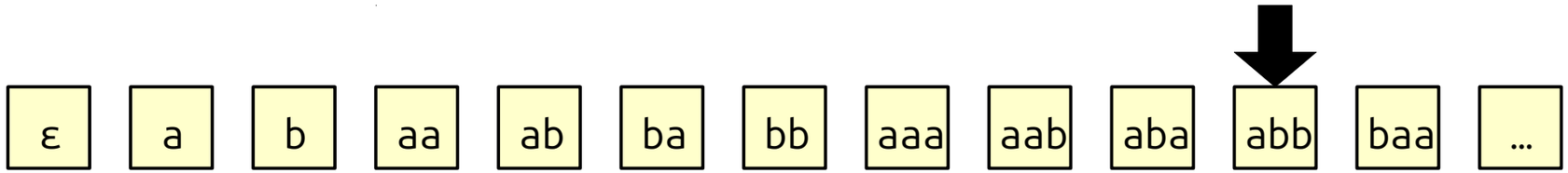


Verifiers and **RE**

- ***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.

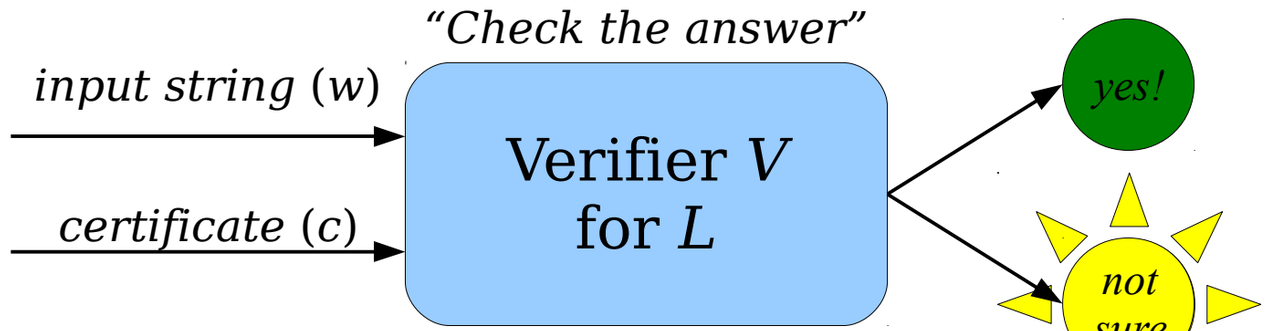


We will try all possible certificates (*values of c*)

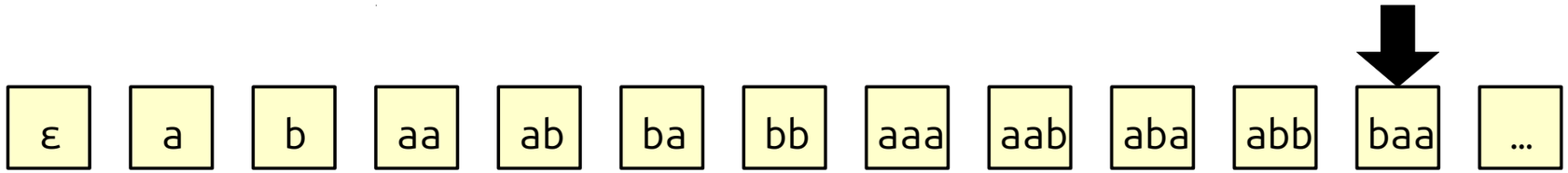


Verifiers and **RE**

- ***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.

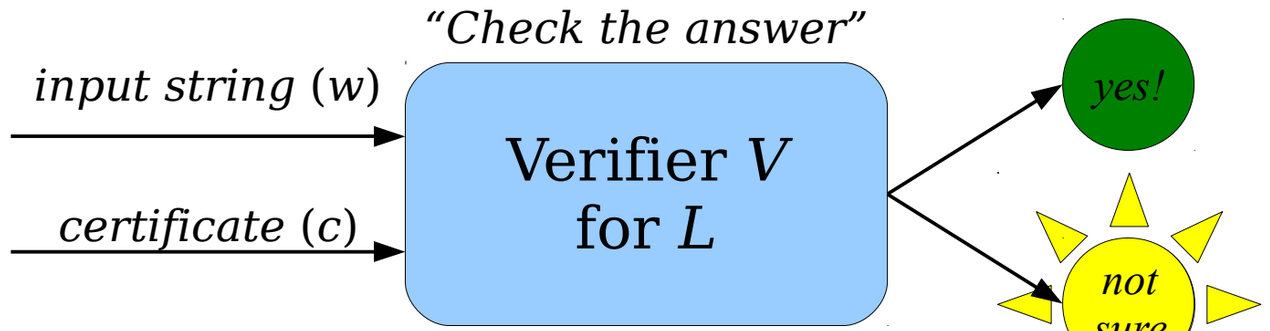


We will try all possible certificates (values of c)

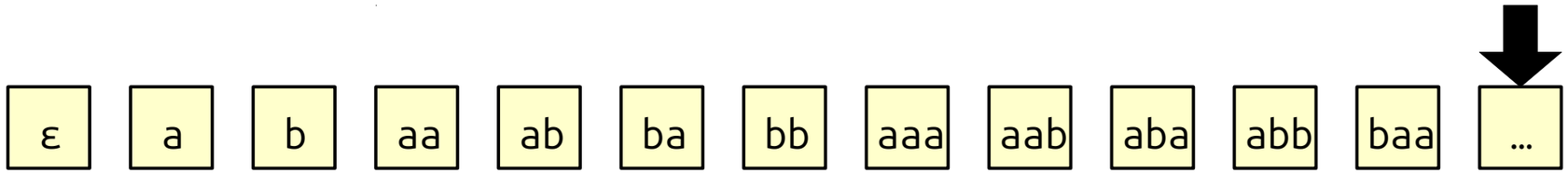


Verifiers and **RE**

- ***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 V is a verifier for L, then $L \in$ **RE**.
- ***Proo*** ng progr

```
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

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

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!