Turing Machines

Part Three

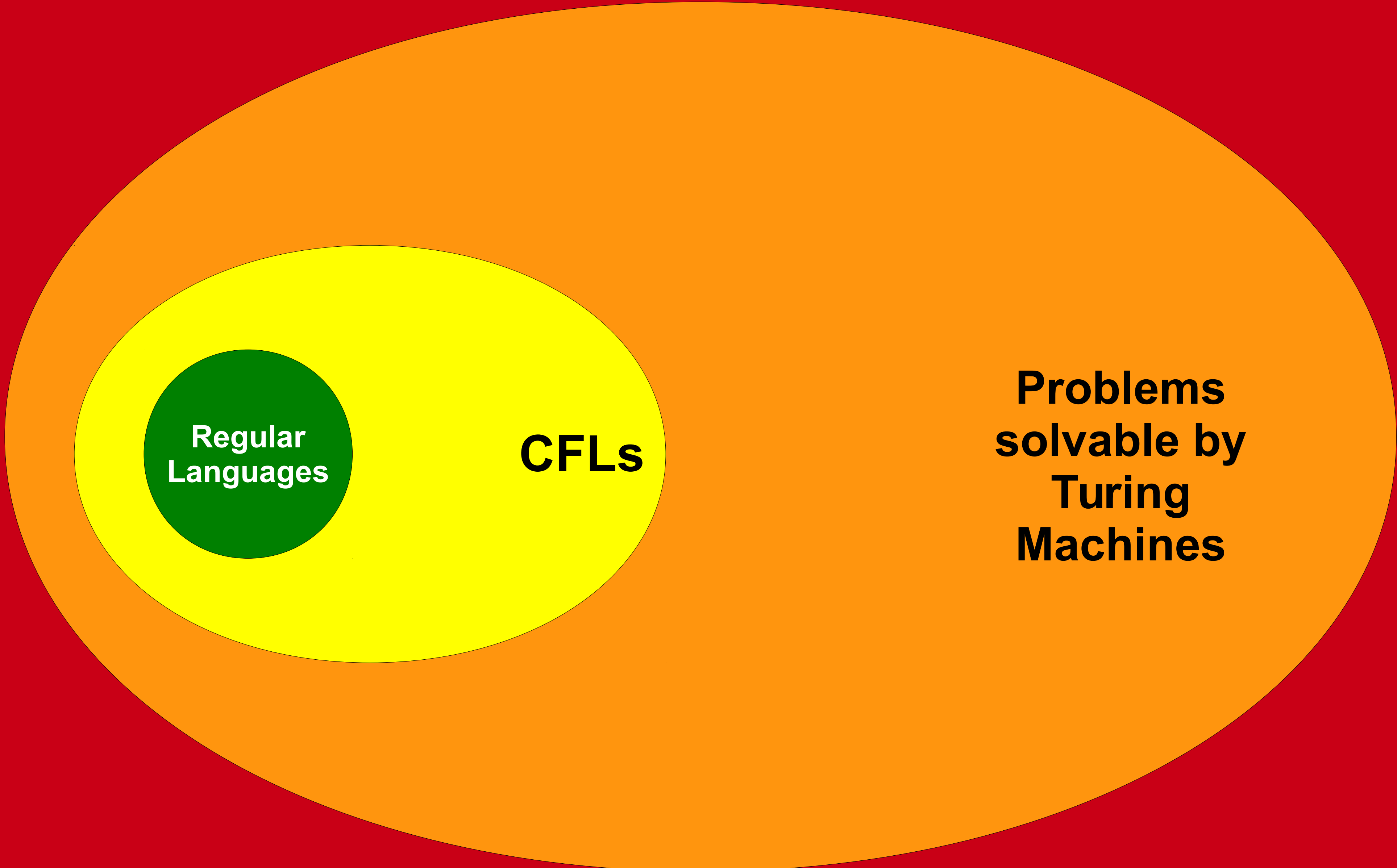
Last Time: **How powerful are Turing machines?**

The ***Church-Turing Thesis*** claims that

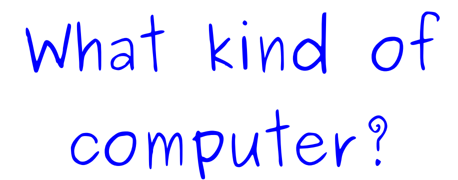
every effective method of computation is either equivalent to or weaker than a Turing machine.

"This is not a theorem – it is a falsifiable scientific hypothesis. And it has been thoroughly tested!"

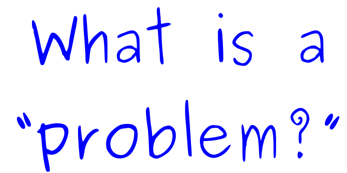
- Ryan Williams


Regular Languages
CFLs
Problems solvable by Turing Machines
All Languages

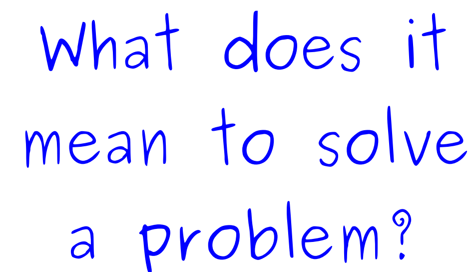
What problems can we solve with a computer?



What problems can we solve with a computer?

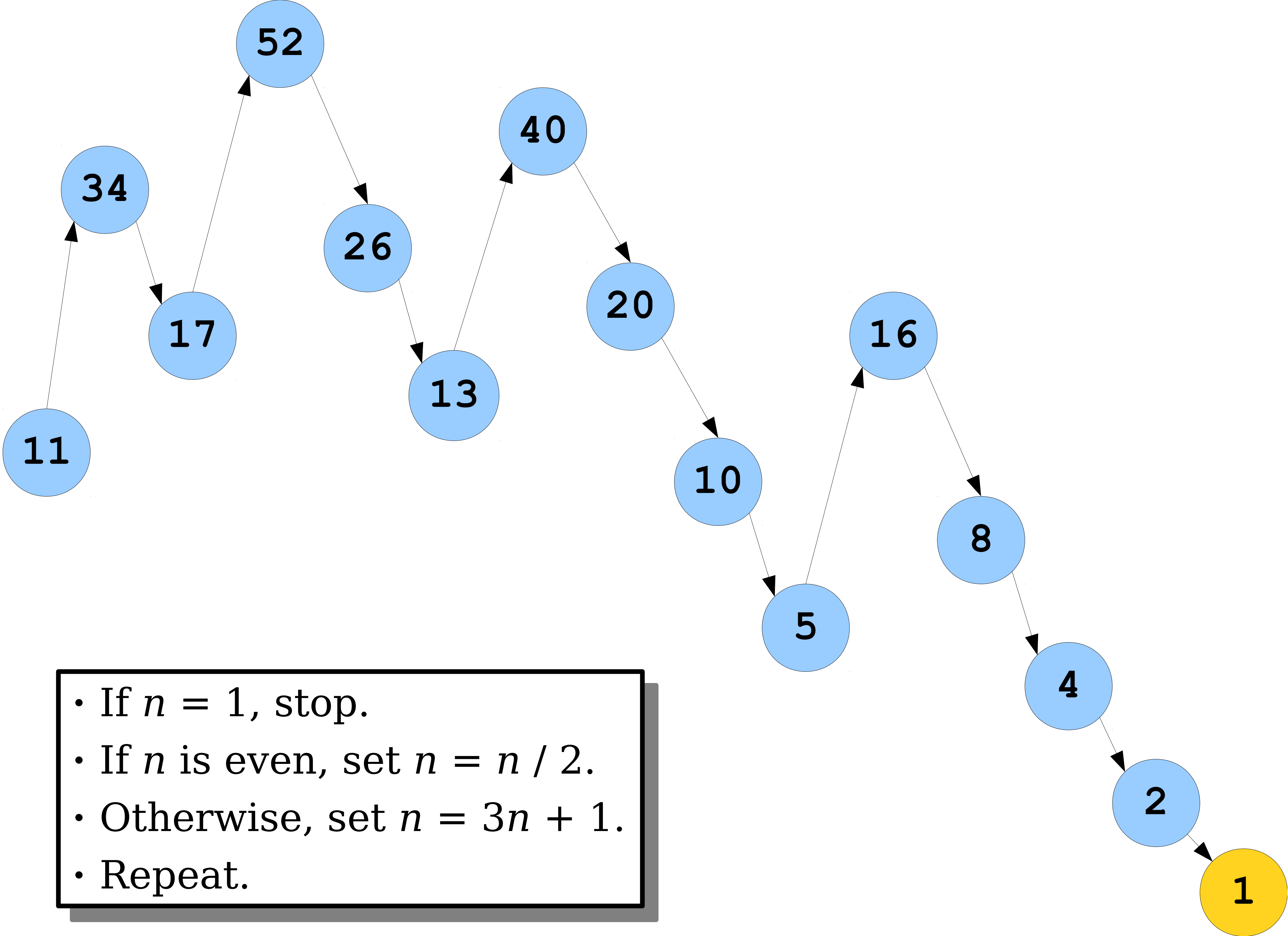


What problems can we solve with a computer?



The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
  - If $n = 1$, you are done.
  - If n is even, set $n = n / 2$.
  - Otherwise, set $n = 3n + 1$.
  - Repeat.
- **Question:** Given a number n, does this process terminate?


11
34
17
52
26
13
40
20
10
5
16
8
4
2
1
• If $n = 1$, stop.
• If n is even, set $n = n / 2$.
• Otherwise, set $n = 3n + 1$.
• Repeat.

The Hailstone Sequence

- Let $\Sigma = \{\texttt{1}\}$ and consider the language

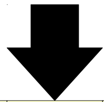
$$L = \{ \texttt{1}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}.$$

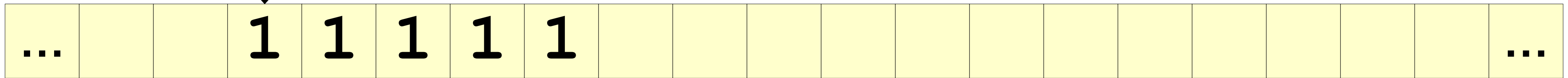
- Could we build a TM for L?

The Hailstone Turing Machine

- We can build a TM that works as follows:
  - If the input is ε, reject.
  - While the string is not `1`:
    - If the input has even length, halve the length of the string.
    - If the input has odd length, triple the length of the string and append a `1`.
  - Accept.

The Hailstone Turing Machine





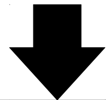
If the input is ε, reject.

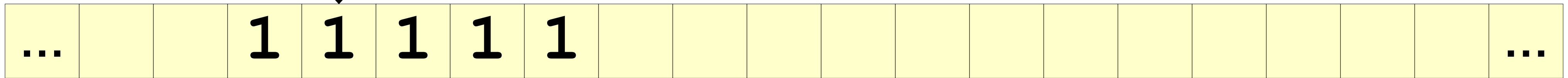
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





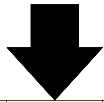
If the input is ε, reject.

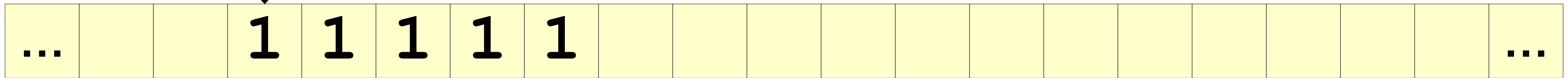
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





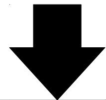
If the input is ε, reject.

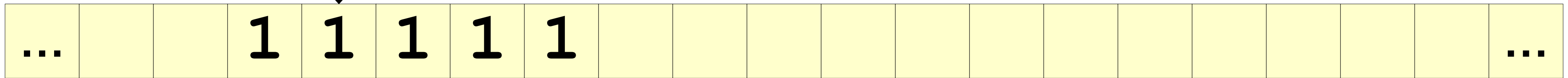
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





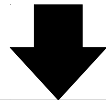
If the input is ε, reject.

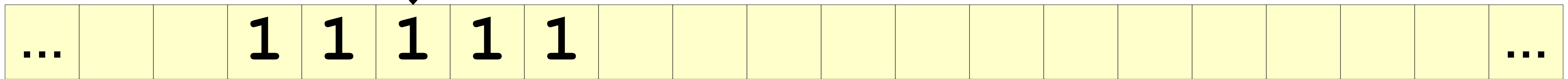
While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine





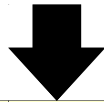
If the input is ε, reject.

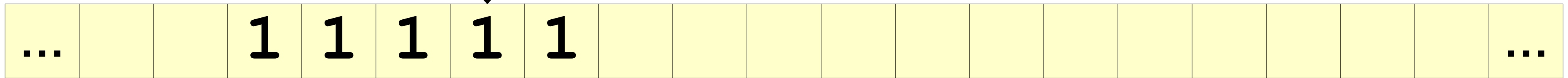
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





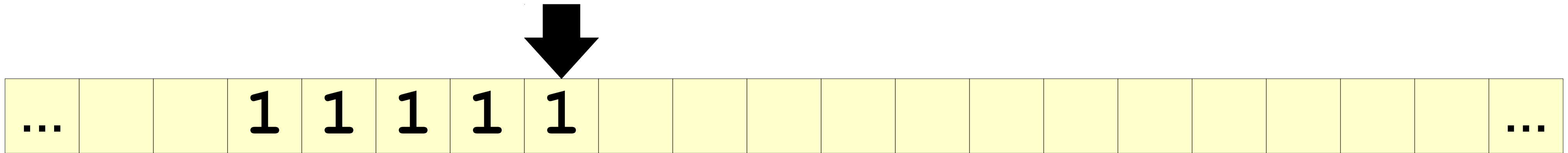
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



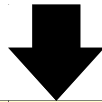
If the input is ε, reject.

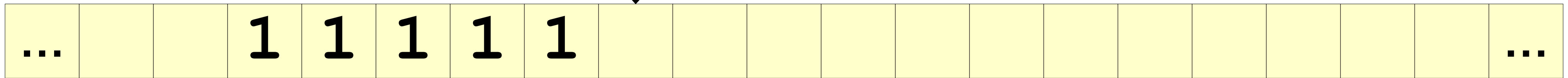
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





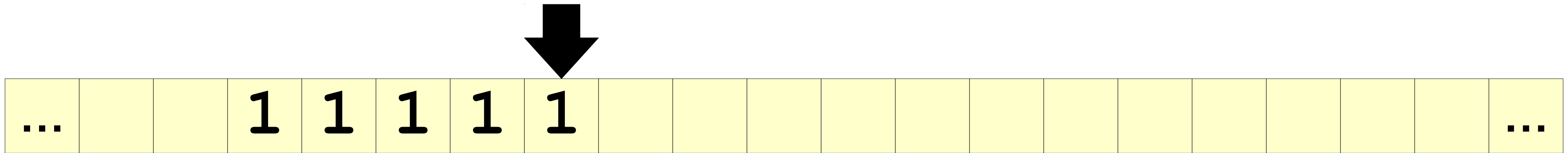
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



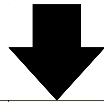
If the input is ε, reject.

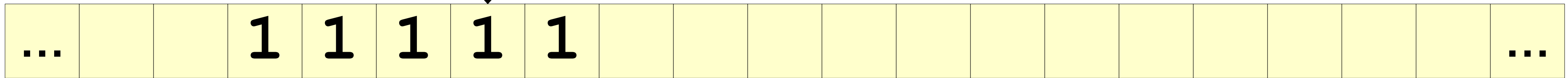
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





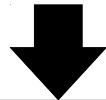
If the input is ε, reject.

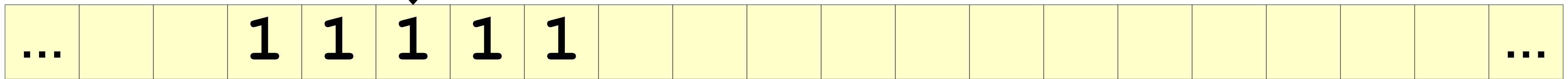
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





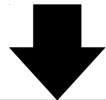
If the input is ε, reject.

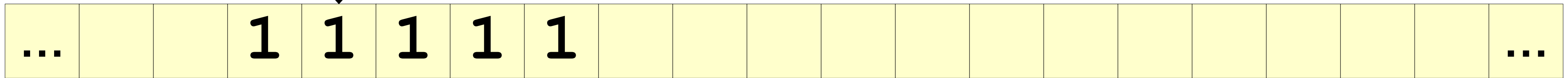
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





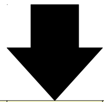
If the input is ε, reject.

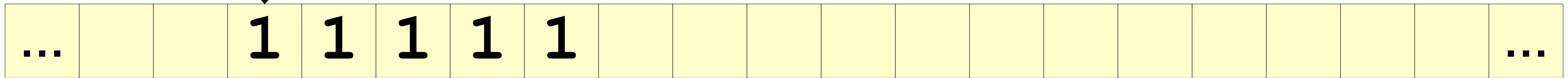
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





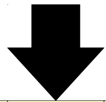
If the input is ε, reject.

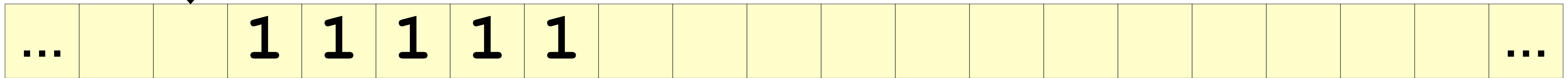
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





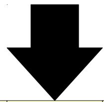
If the input is ε, reject.

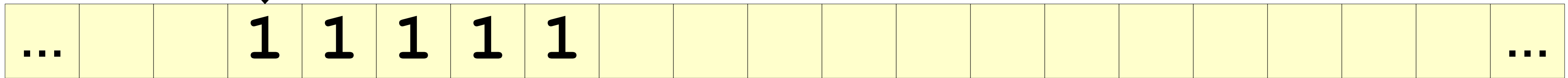
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





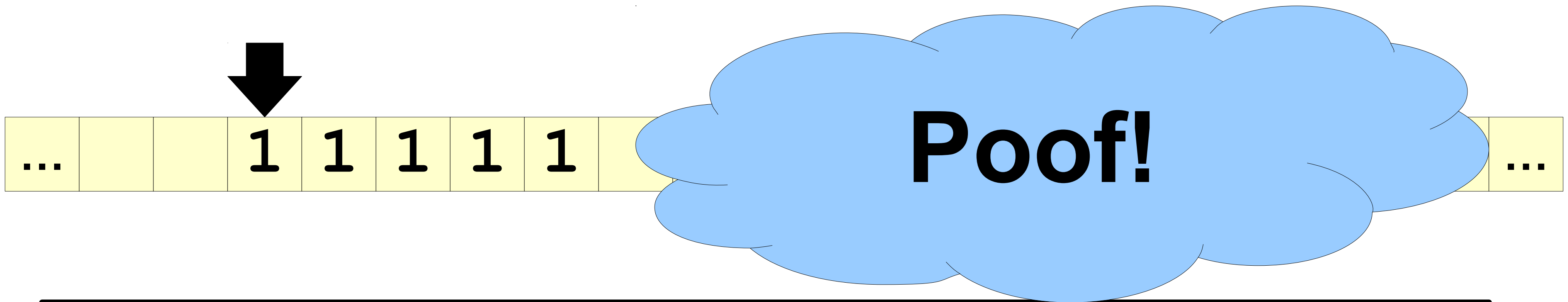
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



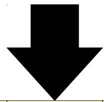
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

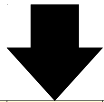
If the input is ε, reject.

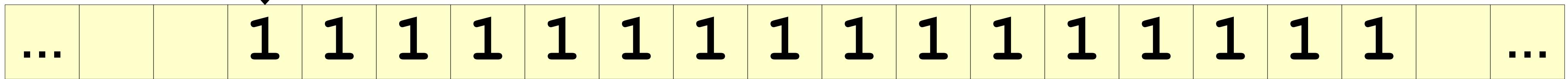
While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine





If the input is ε, reject.

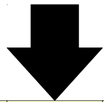
While the input is not 1:

- If the input has even leng
  of the string.
- If the input has odd leng
  string and append a 1.

Accept.

Interesting problem: Build a TM that, starting with n 1s on its tape, ends with $3n + 1$ 1s on its tape.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

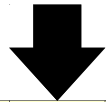
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

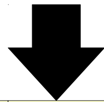
If the input is ε, reject.

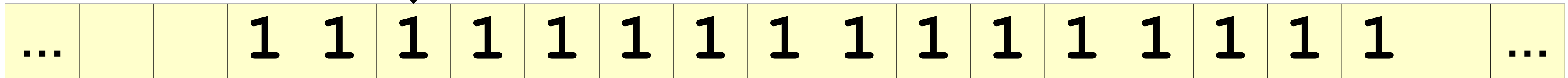
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





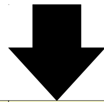
If the input is ε, reject.

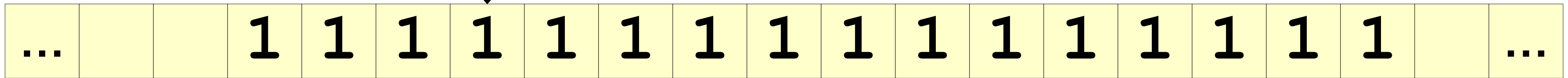
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





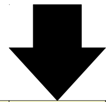
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

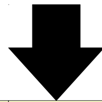
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

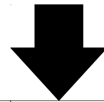
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

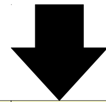
If the input is ε, reject.

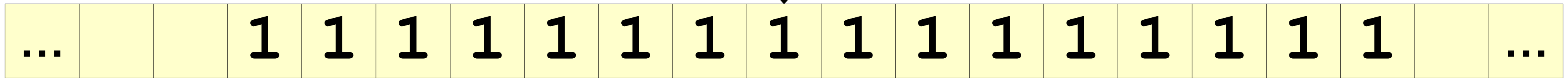
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





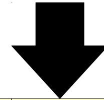
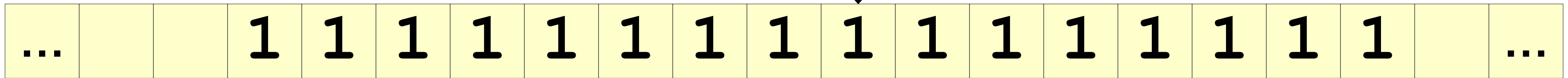
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine

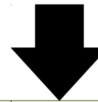
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

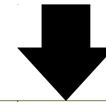
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

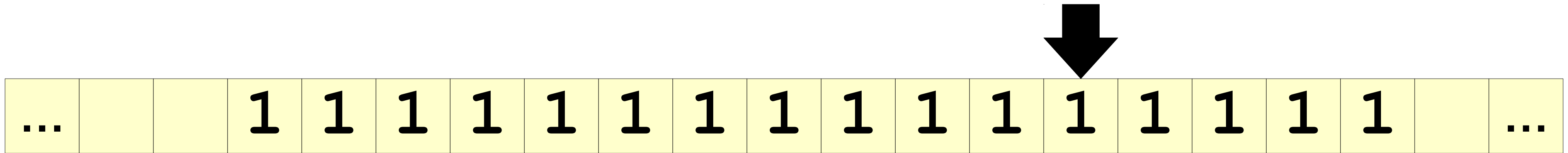
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



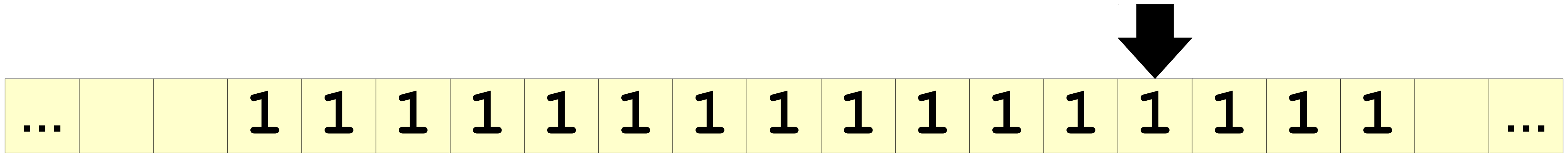
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



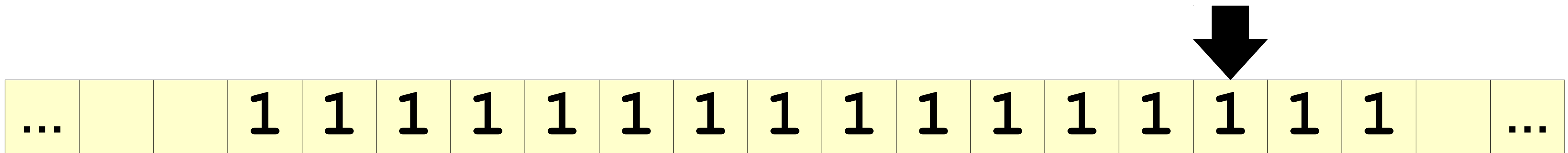
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



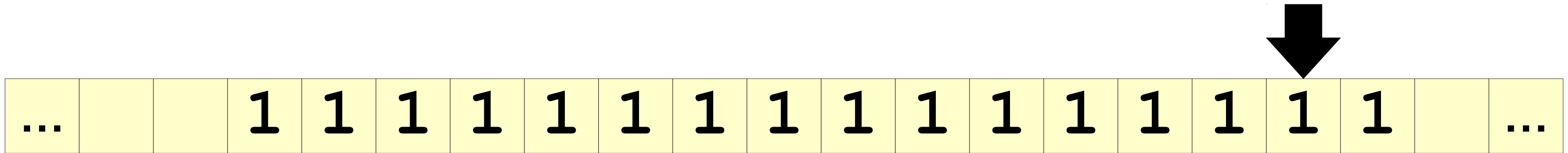
If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine



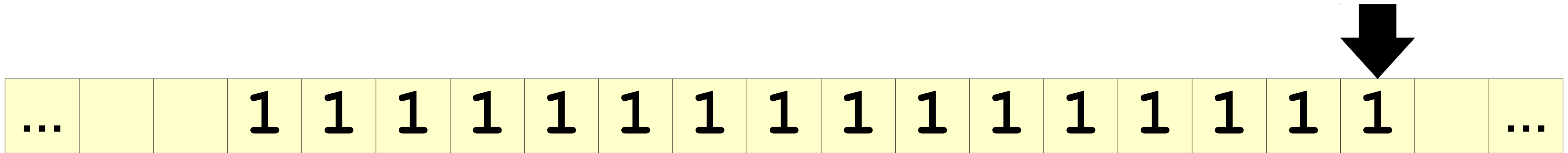
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine

...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

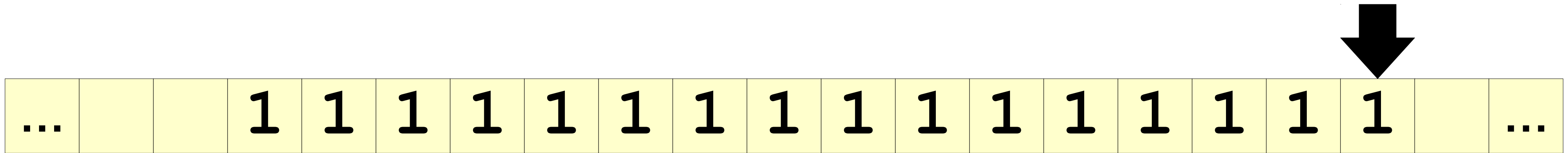
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



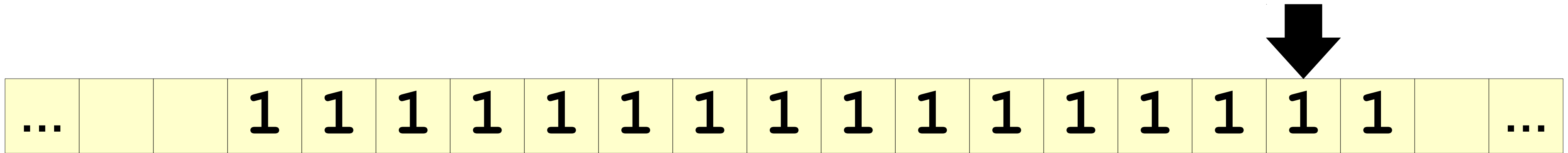
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



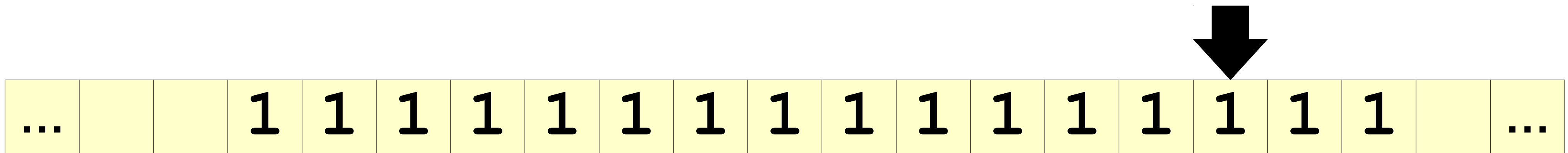
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



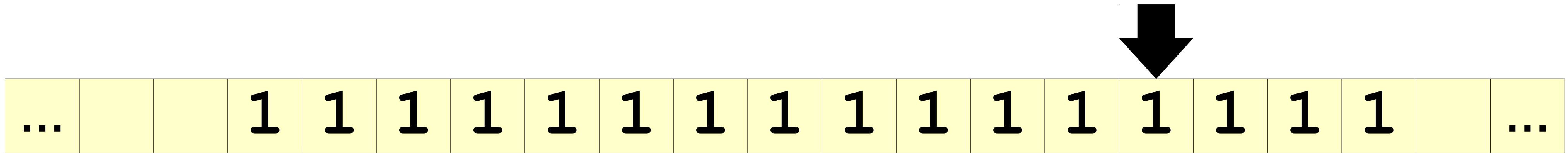
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



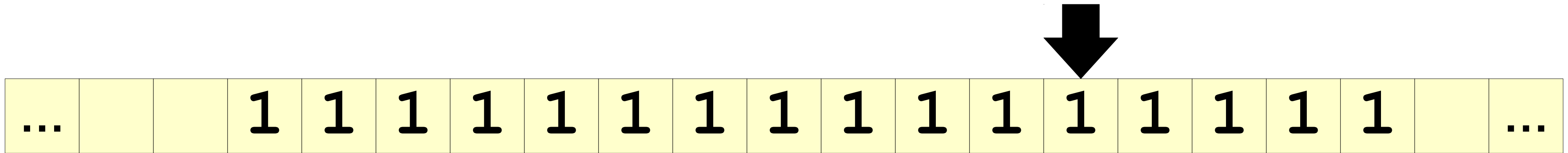
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



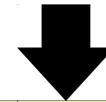
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

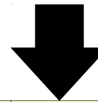
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

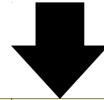
If the input is ε, reject.

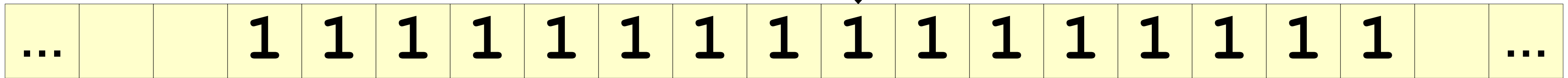
While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine





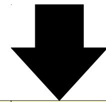
If the input is ε, reject.

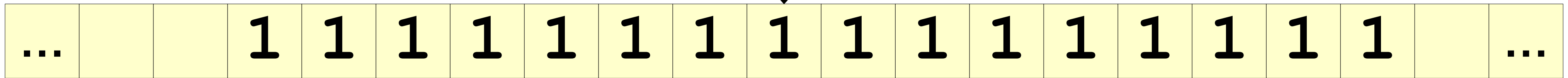
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





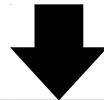
If the input is ε, reject.

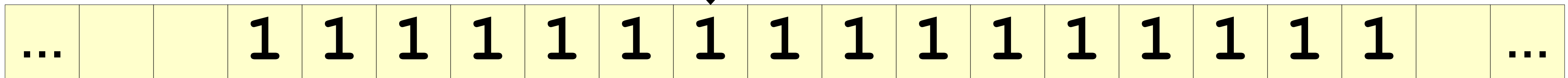
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





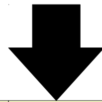
If the input is ε, reject.

While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

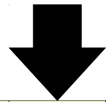
If the input is ε, reject.

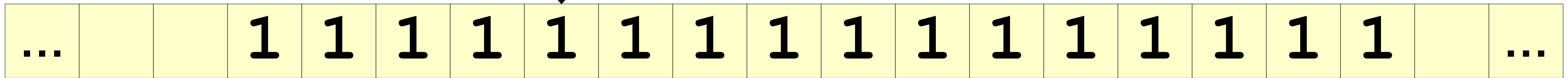
While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine





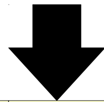
If the input is ε, reject.

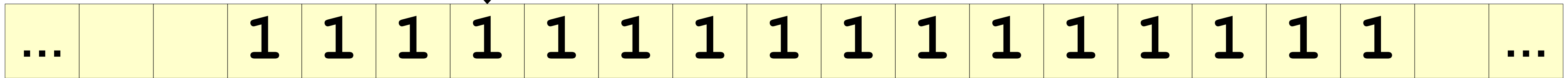
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





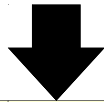
If the input is ε, reject.

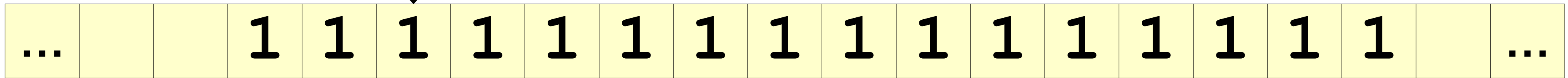
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





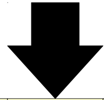
If the input is ε, reject.

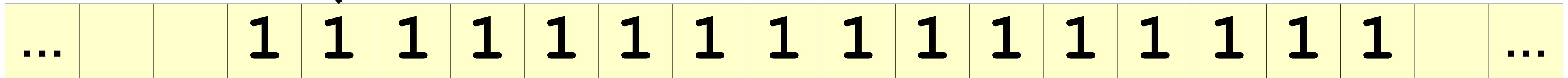
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





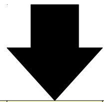
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

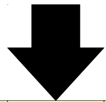
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



...			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		...

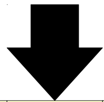
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



…			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		…

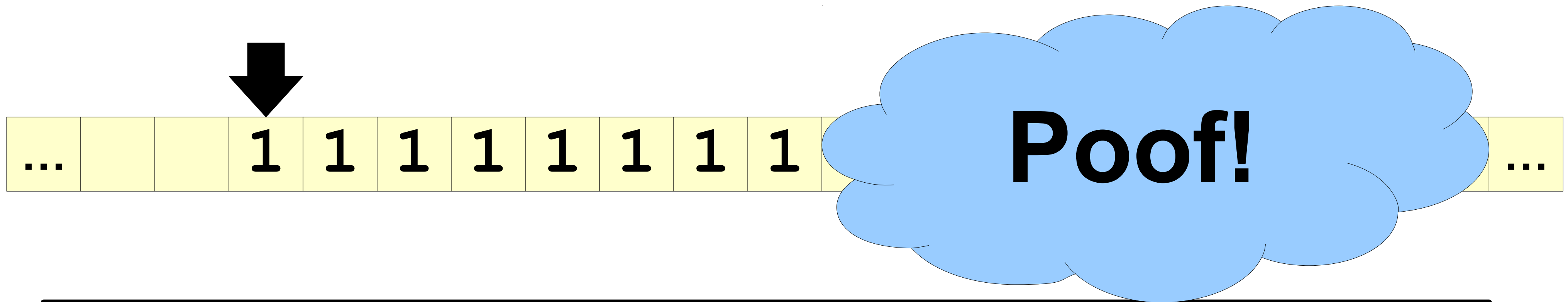
If the input is ε, reject.

While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine



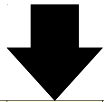
If the input is ε, reject.

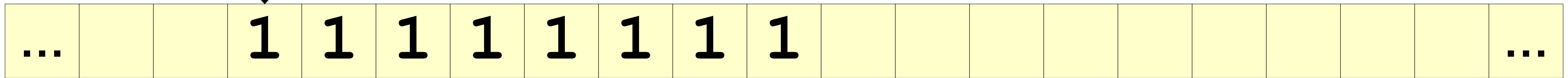
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





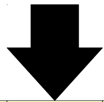
If the input is ε, reject.

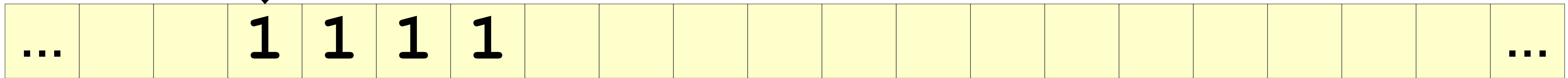
While the input is not 1:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a 1.

Accept.

The Hailstone Turing Machine





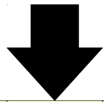
If the input is ε, reject.

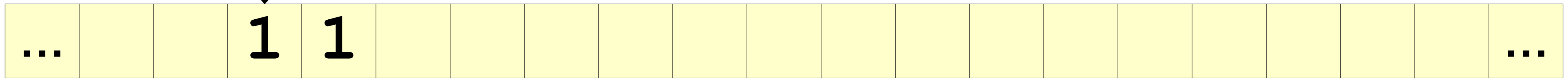
While the input is not `1`:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a `1`.

Accept.

The Hailstone Turing Machine





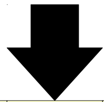
If the input is ε, reject.

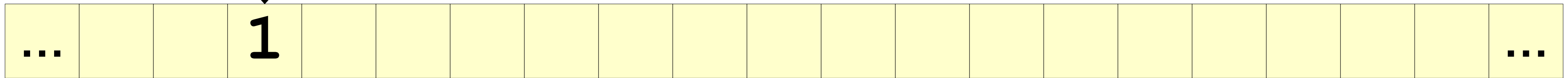
While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

The Hailstone Turing Machine





If the input is ε, reject.

While the input is not **1**:

- If the input has even length, halve the length of the string.
- If the input has odd length, triple the length of the string and append a **1**.

Accept.

Does this Turing machine always accept?

The Collatz Conjecture

- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, ***no one knows whether the TM described in the previous slides will always stop running!***
- The conjecture (unproven claim) that this always terminates is called the ***Collatz Conjecture***.

The Collatz Conjecture

"Mathematics may not be ready for such problems." - Paul Erdős

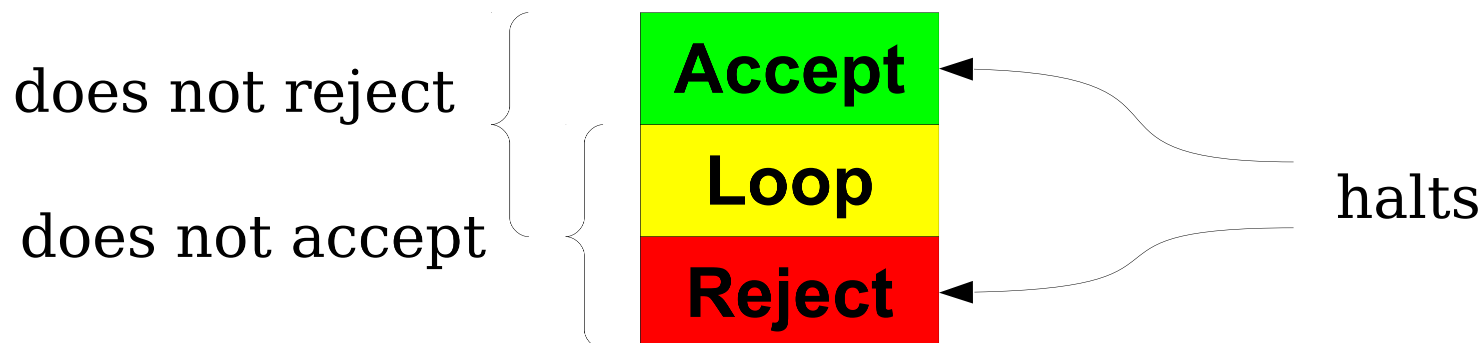
- Two years ago, some Apple employees filed a patent for a cryptographic hashing scheme based on the Collatz conjecture; see ***this link*** for details.

An Important Observation

- Unlike the other automata we've seen so far, Turing machines choose for themselves whether to accept or reject.
- It is therefore possible for a TM to run forever without accepting or rejecting.
- This leads to several important questions:
  - How do we formally define what it means to build a TM for a language?
  - What implications does this have about problem-solving?

Very Important Terminology

- Let M be a Turing machine.
- M ***accepts*** a string w if it enters the accept state when run on w.
- M ***rejects*** a string w if it enters the reject state when run on w.
- M ***loops infinitely*** (or just ***loops***) on a string w if when run on w it enters neither the accept or reject state.
- M ***does not accept w*** if it either rejects w or loops infinitely on w.
- M ***does not reject w*** w if it either accepts w or loops on w.
- M ***halts on w*** if it accepts w or rejects w.



The Language of a TM

- The language of a Turing machine M, denoted $\mathscr{L}(M)$, is the set of all strings that M accepts:

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

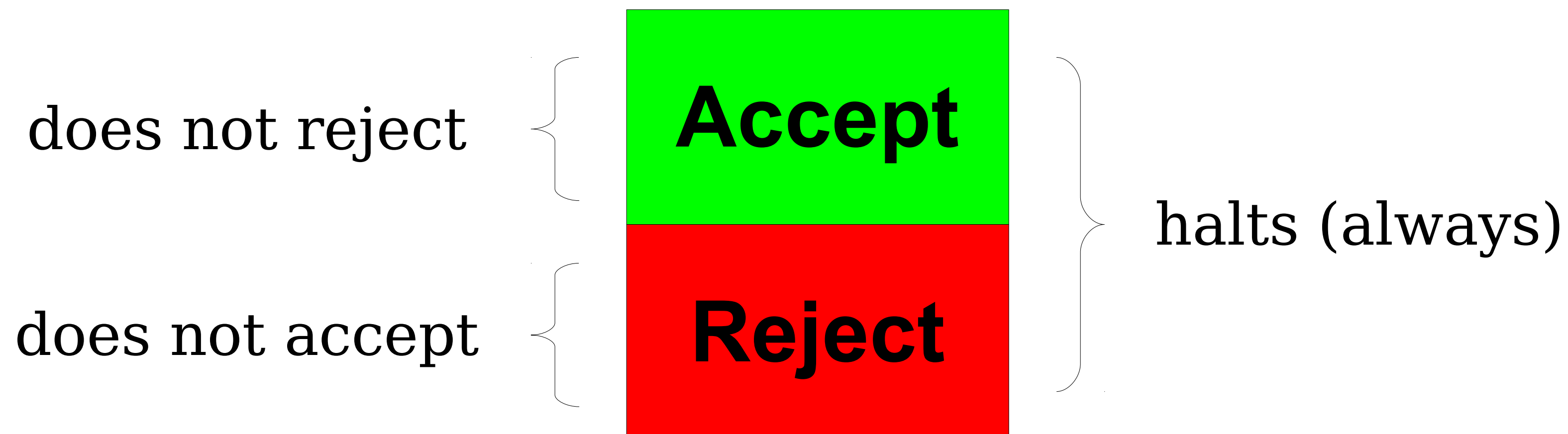
- For any $w \in \mathscr{L}(M)$, M accepts w.
- For any $w \notin \mathscr{L}(M)$, M does not accept w.
  - It might loop forever, or it might explicitly reject.
- A language is called ***recognizable*** if it is the language of some TM. A TM for a language is sometimes called a ***recognizer*** for that language.
- Notation: the class **RE** is the set of all recognizable languages.

$$L \in \textbf{RE} \quad \text{iff} \quad L \text{ is recognizable}$$

What do you think? Does that correspond to what you think it means to solve a problem?

Deciders

- Some Turing machines always halt; they never go into an infinite loop.
- If M is a TM and M halts on every possible input, then we say that M is a ***decider***.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



Decidable Languages

- A language L is called ***decidable*** if there is a decider M such that $\mathscr{L}(M) = L$.
- Equivalently, a language L is decidable if there is a TM M such that
  - If $w \in L$, then M accepts w.
  - If $w \notin L$, then M rejects w.
- The class **R** is the set of all decidable languages.

$$L \in \mathbf{R} \quad \text{iff} \quad L \text{ is decidable}$$

Examples of **R** Languages

- All regular languages are in **R**.
  - If L is regular, we can run the DFA for L on a string w and then either accept or reject w based on what state it ends in.
- $\{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$ is in **R**.
  - The TM we built on Monday is a decider.
- $\{ \mathtt{1}^n \mid n \in \mathbb{N}$ and n is composite$\}$ is in **R**.
  - The TM we built on Wednesday is a decider.
- All CFLs are in **R**.
  - Proof is tricky; check Sipser for details.
  - (This is why it's possible to build the CFG tool online!)

Why **R** Matters

- If a language is in **R**, there is an algorithm that can decide membership in that language.
  - Run the decider and see what it says.
- If there is an algorithm that can decide membership in a language, that language is in **R**.
  - By the Church-Turing thesis, any effective model of computation is equivalent in power to a Turing machine.
  - Therefore, if there is *any* algorithm for deciding membership in the language, there is a decider for it.
  - Therefore, the language is in **R**.
- ***A language is in R if and only if there is an algorithm for deciding membership in that language.***

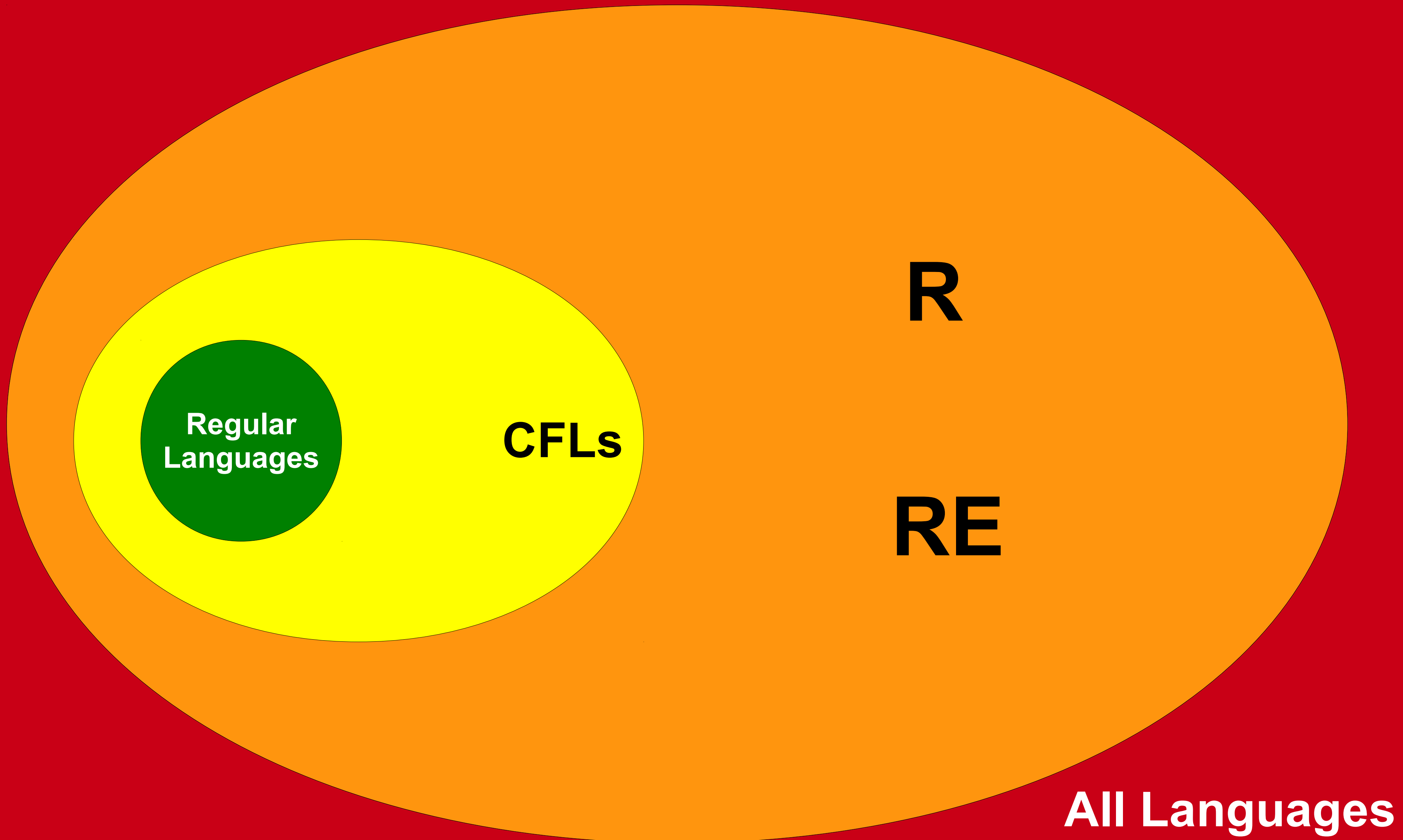
R and **RE** Languages

- Every decider is a Turing machine, but not every Turing machine is a decider.
- Thus **R** ⊆ **RE**.
- Hugely important theoretical question:

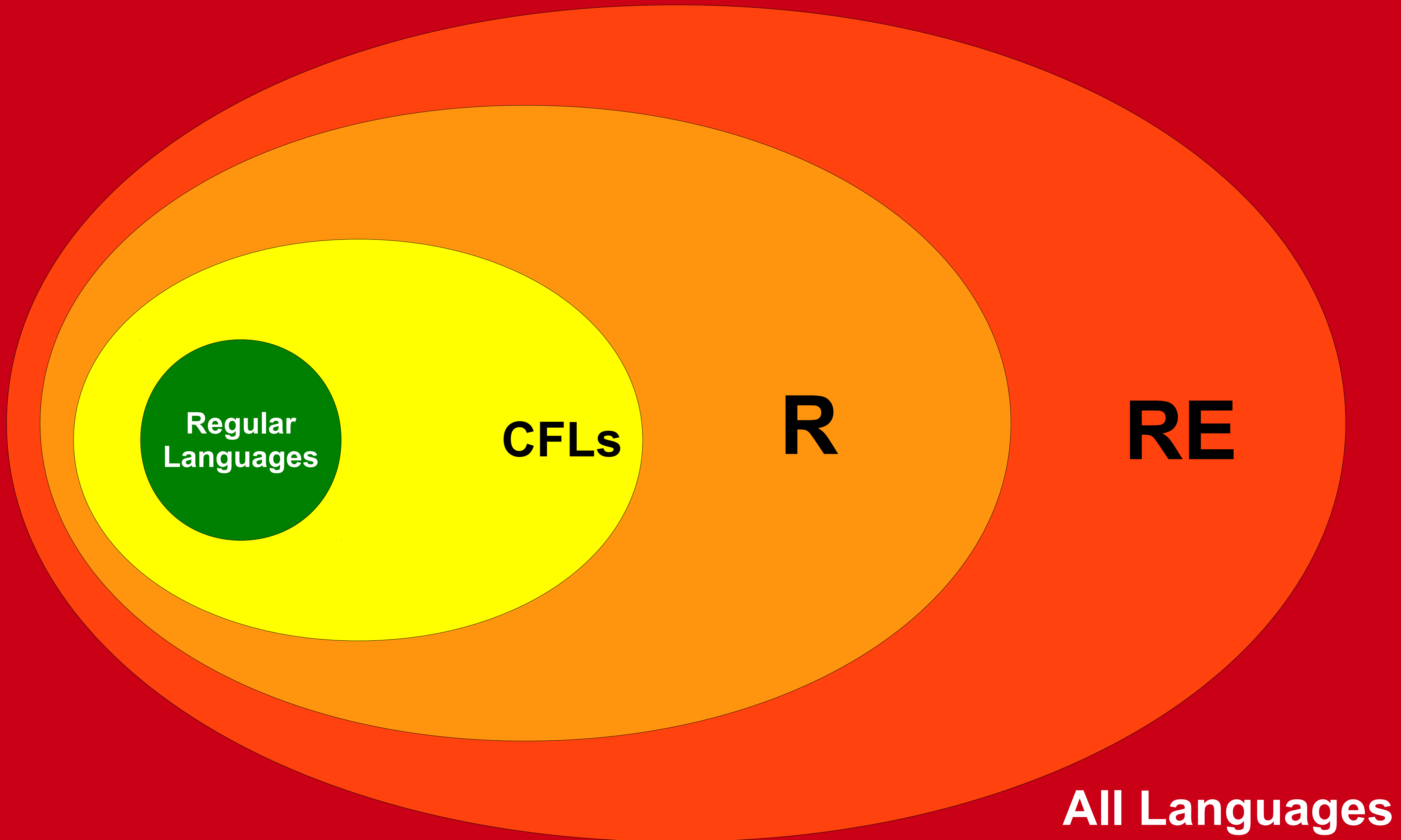
$$\mathbf{R} \stackrel{?}{=} \mathbf{RE}$$

- That is, if you can just confirm “yes” answers to a problem, can you necessarily *solve* that problem?

Which Picture is Correct?



Which Picture is Correct?



Unanswered Questions

- Why exactly is **RE** an interesting class of problems?
- What does the $\mathbf{R} \stackrel{?}{=} \mathbf{RE}$ question mean?
- Is $\mathbf{R} = \mathbf{RE}$?
- What lies beyond **R** and **RE**?
- We'll see the answers to each of these in due time.

Time-Out for Announcements!

Second Midterm Exam

- Second midterm exam is next Thursday, May 21 from 7PM – 10PM.
- Rooms divvied up by last (family) name:
  - Aba – Sow: Go to **Hewlett 200**
  - Spe – Zoc: Go to **Hewlett 201**
- Closed-book, closed-computer, open one double-sided 8.5” × 11” sheet of notes.
- Cumulative, focusing on PS4 – PS6.

Alternate Exam

- Can't make the main midterm time? We'll be holding an alternate exam from 4PM – 7PM next Thursday.
- Email us no later than Saturday at noon if you want to take the alternate exam or otherwise have a conflict.

Practice Midterm Exam

- We will be holding a practice midterm exam on Monday from 7PM – 10PM in room 320-105.
- Structure and format of practice exam is similar to that of the main exam.
- TAs will be on-hand to answer questions; we'll release solutions as well.
- Can't make it? Don't worry! We'll post the exam on the course website.

More Practice Problems

- Solutions to Extra Practice Problems 4 are available for pickup right now.
- We've released a fifth set of extra practice problems you can use to prepare for the midterm.
- Solutions and another set of practice problems will go out on Monday.

Grace Hopper

- The Grace Hopper Celebration of Women in Computing will be held in Houston from October 14 – 16.
- The CS department has funds to sponsor up to 40 students to attend this year's event.
- ***Highly recommended; this is an amazing event.***
- Interested? Fill out ***this form*** by next Friday, May 22.

Your Questions!

“Any advice for people considering a CS co-term? Would you recommend it for people who aren't CS majors? Thoughts on financial concerns if you can't do it in 4 years?”

The coterm is open to everyone regardless of major – in fact, it's specifically designed to be available to non-majors.

If you didn't do a CS undergrad degree but want to learn more CS, this is an excellent way to do so. If you did a CS undergrad and want to explore the frontiers of CS, this is also a great program.

For finances: a TAship or RAship will cover tuition up to 10 units and pay a quarterly stipend of about $8,500. Most coterms are able to take advantage of this.

Come talk to me if you have questions!

“You said that the second midterm will cover material from PS4-PS6. Will PS 5,6 be graded by then? We've only just gotten back PS4, about two weeks after it was submitted.”

We'll have PS5 graded by next Monday. I don't think it will be possible to grade PS6 – a turnaround time of three days isn't feasible. Sorry about that!

Back to CS103!

We can now encode TMs as strings.

TMs accept strings as input.

What can we do with this knowledge?

Universal Machines

The Universal Turing Machine

- ***Theorem**:* There is a Turing machine $\mathbf{U_{TM}}$ called the ***universal Turing machine*** that, when run on ⟨*M*, *w*⟩, where *M* is a Turing machine and *w* is a string, simulates *M* running on *w*.
- Conceptually:

U_{TM} = "On input ⟨*M*, *w*⟩, where *M* is a TM and $w \in \Sigma^*$:

Set up the initial configuration of *M* running on *w*.

```
while (true) {
```

If *M* accepted *w*, then U_{TM} accepts ⟨*M*, *w*⟩.

If *M* rejected *w*, then U_{TM} rejects ⟨*M*, *w*⟩.

Otherwise, simulate one more step of *M* on *w*.

```
}
```
"

The Universal Turing Machine

- ***Theorem:*** There is a Turing machine $\mathbf{U_{TM}}$ called the ***universal Turing machine*** that, when run on $\langle M, w \rangle$, where M is a Turing machine and w is a string, simulates M running on w.
- The observable behavior of U_{TM} is the following:
  - If M accepts w, then U_{TM} accepts $\langle M, w \rangle$.
  - If M rejects w, then U_{TM} rejects $\langle M, w \rangle$.
  - If M loops on w, then U_{TM} loops on $\langle M, w \rangle$.

An Intuition for U_{TM}

- You can think of U_{TM} as a general-purpose, programmable computer.
- Rather than purchasing one TM for each language, just purchase U_{TM} and program in the "software" corresponding to the TM you actually want.
- U_{TM} is a powerful machine: ***it can perform any computation that could be performed by any feasible computing device!***

The Language of U_{TM}

- Recall: For any TM M, the language of M, denoted $\mathscr{L}(M)$, is the set

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

- What is the language of U_{TM}?
- U_{TM} accepts $\langle M, w \rangle$ iff M is a TM that accepts w.
- Therefore:

$$\mathscr{L}(U_{TM}) = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$$

$$\mathscr{L}(U_{TM}) = \{ \langle M, w \rangle \mid M \text{ is a TM and } w \in \mathscr{L}(M) \}$$

- For simplicity, define $\mathbf{A_{TM}} = \mathscr{L}(\mathbf{U_{TM}})$. This is an important language and we'll see it many times.

A_{TM}

Regular Languages

CFLs

RE

All Languages

This might seem random, but trust me, it's important. We'll return back to this soon.

Self-Referential Software

Quines

Quines

- A ***Quine*** is a program that, when run, prints its own source code.
- Quines aren't allowed to just read the file containing their source code and print it out; that's cheating.
- How would you write such a program?

Writing a Quine

Self-Referential Programs

- ***Claim:*** Going forward, assume that any program can be augmented to include a method called `mySource()` that returns a string representation of its source code.
- General idea:
  - Write the initial program with `mySource()` as a placeholder.
  - Use the Quine technique we just saw to convert the program into something self-referential.
  - Now, `mySource()` magically works as intended.

The Recursion Theorem

- There is a deep result in computability theory called ***Kleene's second recursion theorem*** that, informally, states the following:

It is possible to construct TMs that perform arbitrary computations on their own descriptions.

- Intuitively, this generalizes our Quine constructions to work with arbitrary TMs.
- Want the formal statement of the theorem? Take CS154!

Resolving $\mathbf{R} \stackrel{?}{=} \mathbf{RE}$

A Recipe for Disaster

- Consider the following facts about TMs:
  - TMs can run other TMs as subroutines.
  - TMs can make decisions about what to do next based on the results of these subroutines.
- What happens when we throw these two facts into the mix?
  - TMs can accept other TMs as input.
  - TMs can get their own descriptions.

A Recipe for Disaster

- Suppose that $A_{TM} \in \mathbf{R}$.
- Formally, this means that there is a TM that decides A_{TM}.
- Intuitively, this means that there is a TM that takes as input a TM M and string w, then
  - accepts if M accepts w, and
  - rejects if M does not accept w.

A Recipe for Disaster

- To make the previous discussion more concrete, let's explore the analog for computer programs.
- If A_{TM} is decidable, we could construct a function

```
bool willAccept(string program,
                string input)
```

  that takes in as input a program and a string, then returns true if the program will accept the input and false otherwise.
- What could we do with this?

What does this program do?

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

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

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

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

… this program doesn't accept its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

… this program doesn't accept its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

… this program doesn't accept its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

… this program doesn't accept its input?

What does this program do?

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

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

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

What happens if…

… this program accepts its input?

It rejects the input!

… this program doesn't accept its input?

What does this program do?

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

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

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

What happens if…

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

… this program doesn't accept its input?

What does this program do?

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

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

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

What happens if…

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

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

What does this program do?

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

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

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

What happens if…

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

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

Knowing the Future

- This TM is analogous to a classical philosophical/logical paradox:

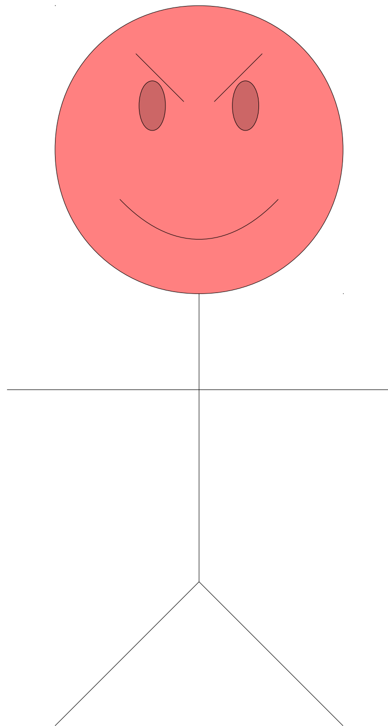
If you know what you are fated to do, can you avoid your fate?

- If A_{TM} is decidable, we can construct a TM that determines what it's going to do in the future (whether it will accept its input), then actively chooses to do the opposite.
- This leads to an impossible situation with only one resolution: **A_{TM} must not be decidable!**

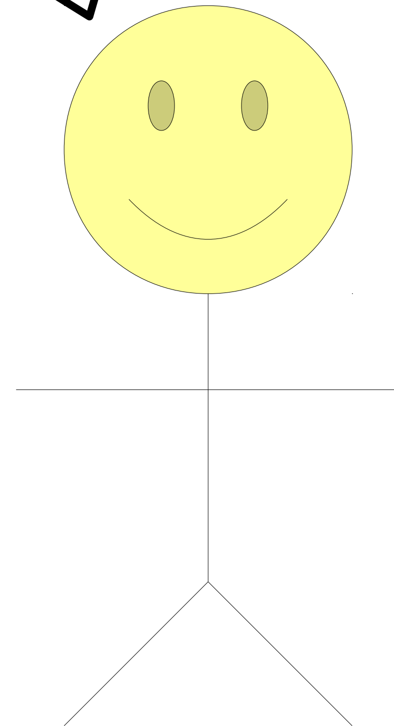
A Different Perspective

So, I hear that you can take *any* TM and *any* string and *decide* whether the TM will accept the string.

Yep! That's what I do!



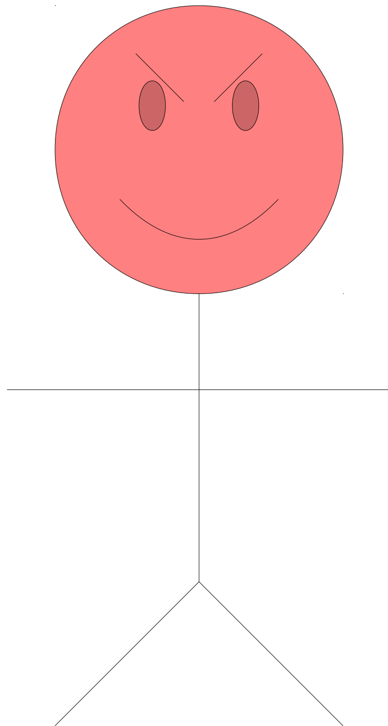
The program we just built



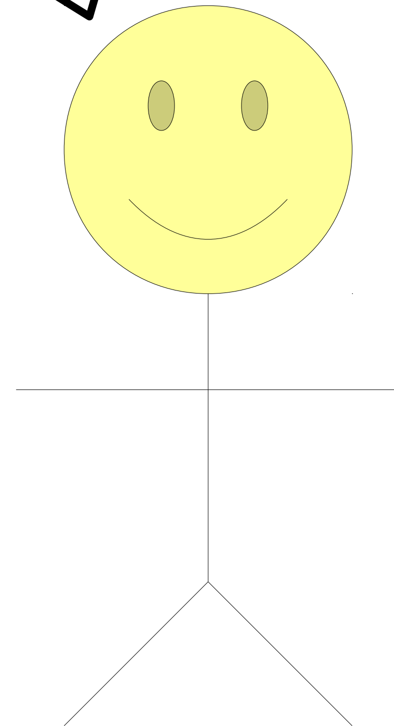
Decider for A_{TM} (willAccept method)

Oh really? *Any* TM?
Including me?

Yep! You are a TM, after all!



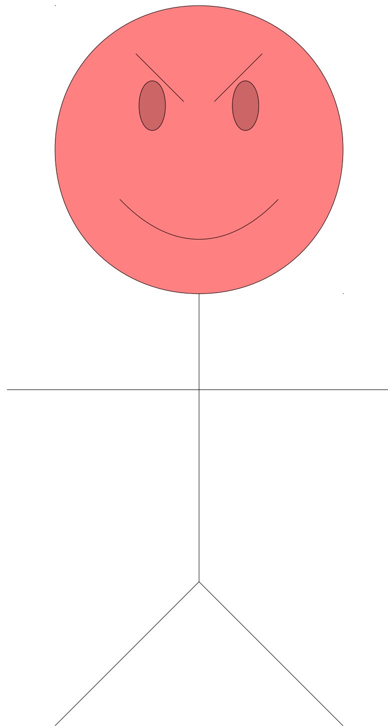
The program
we just built



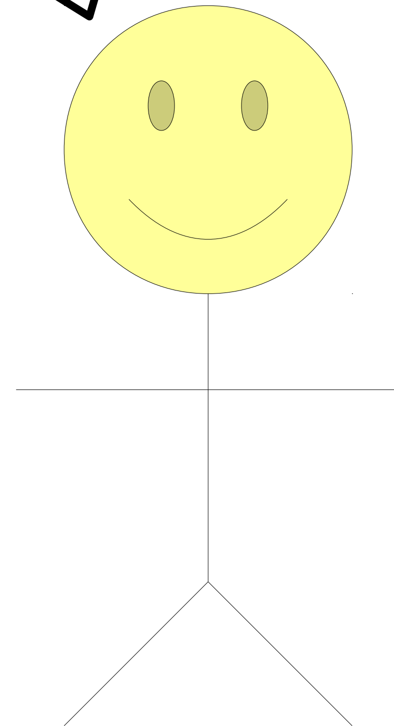
Decider for A_{TM}
(willAccept method)

Okay, well, here's what I'm going to do. If you tell me that I will accept, I'm going to choose not to!

Um, okay...



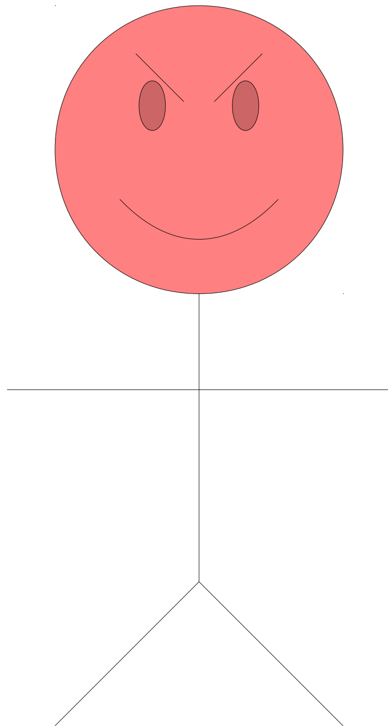
The program
we just built



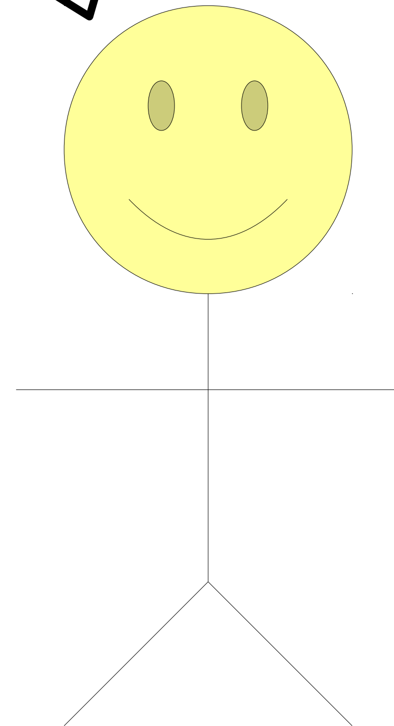
Decider for A_{TM}
(willAccept method)

And, if you tell me that I won't accept, then I *will* accept!

Uh...

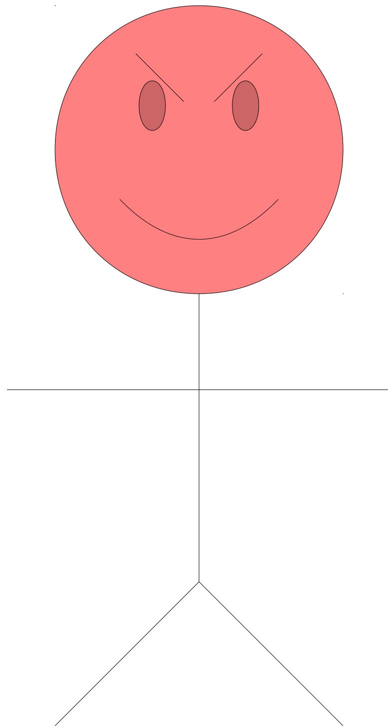


The program
we just built

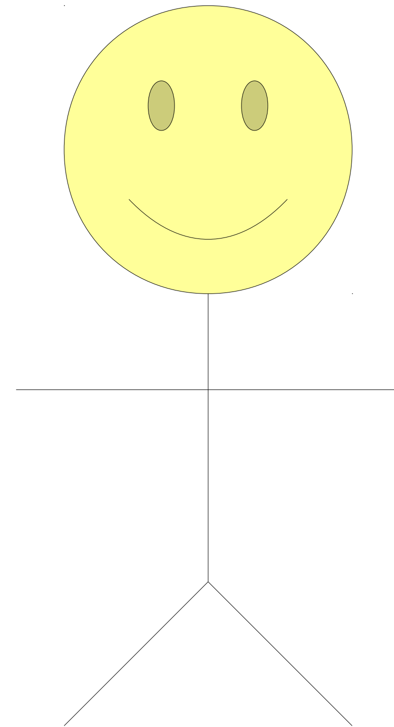


Decider for A_{TM}
(willAccept method)

So which is it? Will I accept or will I not accept?

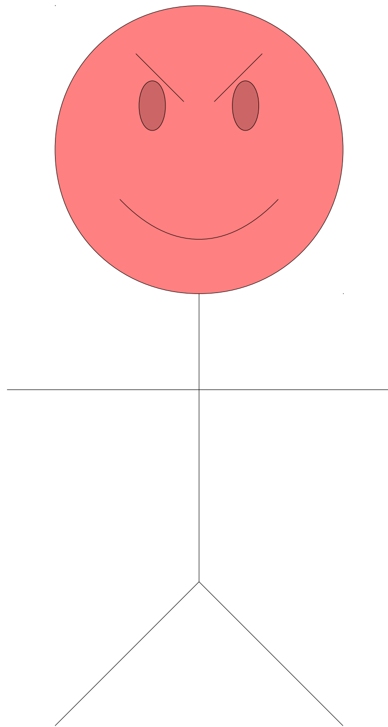


The program
we just built

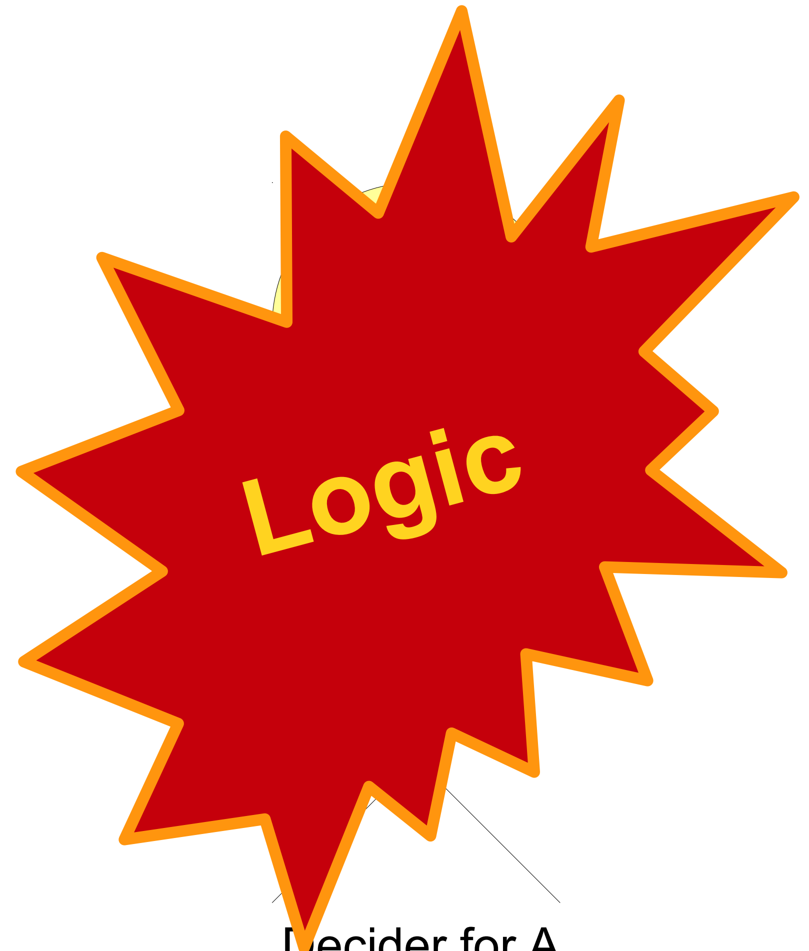


Decider for A_{TM}
(willAccept method)

So which is it? Will I accept or will I not accept?



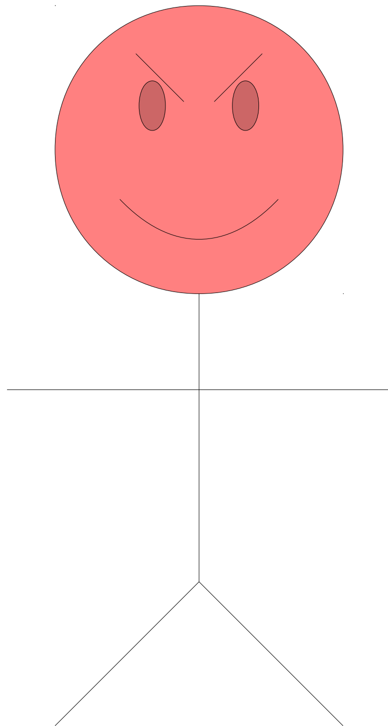
The program
we just built

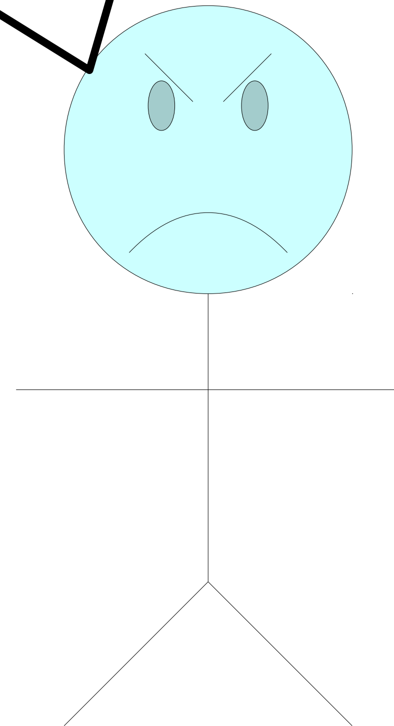


Decider for A_{TM}
(willAccept method)

Mwahahaha!

This is why we can't
have nice things!





The Impossible Bind

- We can also view the contradiction here as an impossible bind.
- We promise the decider for A_{TM} that we will behave in a way that makes it wrong no matter what it says.
- Since the decider *has* to say something, it's guaranteed to be wrong!

Theorem: $A_{TM} \notin \mathbf{R}$.

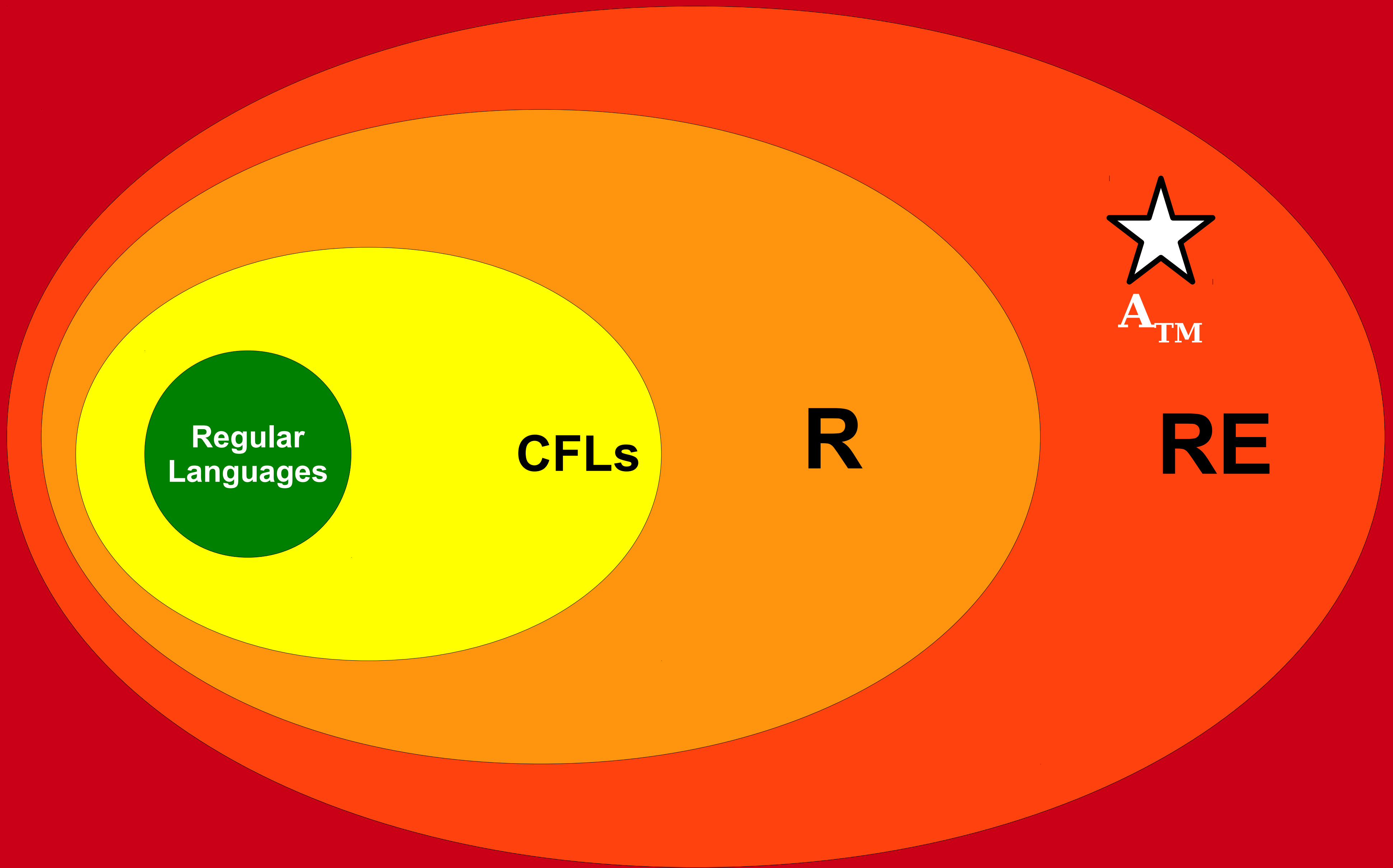
Proof: By contradiction; assume that $A_{TM} \in \mathbf{R}$. Then there is some decider D for A_{TM}. If this machine is given any TM/string pair, it will then determine whether the TM accepts the string and report back the answer.

Given this, we could then construct the following TM:

M = "On input w:
Have M obtain its own description, $\langle M \rangle$.
Run D on $\langle M, w \rangle$ and see what it says.
If D says that M will accept w, reject.
If D says that M will not accept w, accept."

Choose any string w and trace through the execution of the machine, focusing on the answer given back by machine D. If D says that M will accept w, notice that M then proceeds to reject w, contradicting what D says. Otherwise, if D says that M will not accept w, notice that M then proceeds to accept w, contradicting what D says.

In both cases we reach a contradiction, so our assumption must have been wrong. Therefore, $A_{TM} \notin \mathbf{R}$. ■



All Languages

What Does This Mean?

- In one fell swoop, we've proven that
  - **R** ≠ **RE**, because $A_{TM} \notin$ **R** but $A_{TM} \in$ **RE**.
  - A_{TM} is ***undecidable***; there is no algorithm that can determine whether a TM will accept a string.
- What do these two statements really mean? As in, why should you care?