Turing Machines

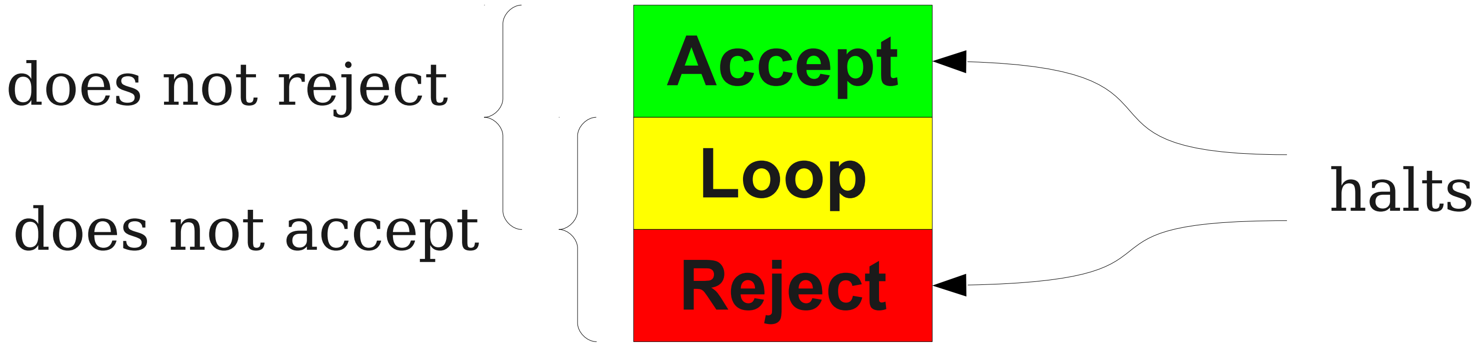
Part III

Announcements

- Problem Set 6 due now.
- Problem Set 7 out, due Monday, March 4.
    - Play around with Turing machines, their powers, and their limits.
    - Some problems require Wednesday's lecture; if you want to get a jump on that, look at Sipser, Chapter 4.
    - We *strongly suggest* starting this one early.
- Problem Set 5 graded; will be returned at end of lecture.

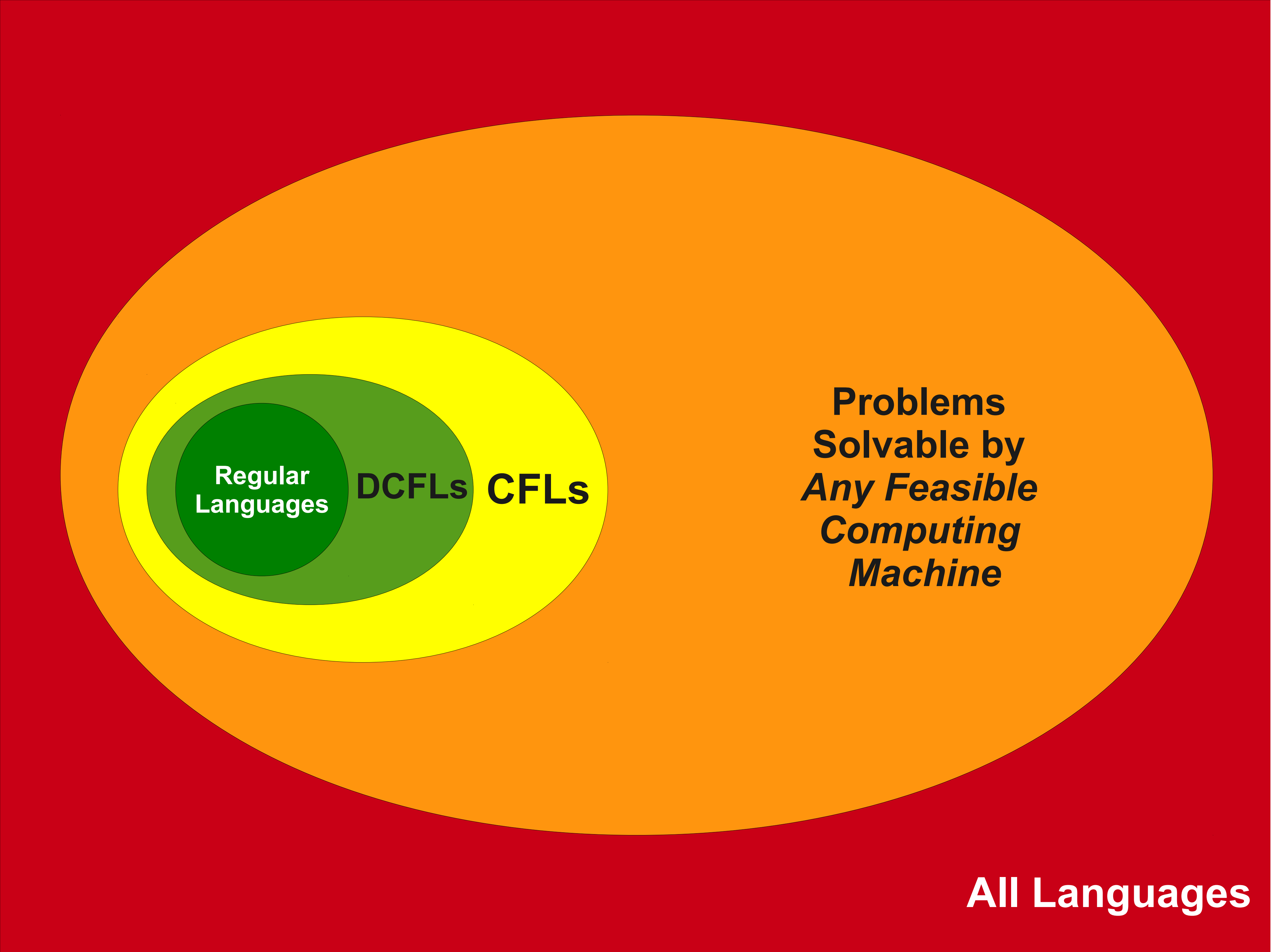
Recap from Last Time

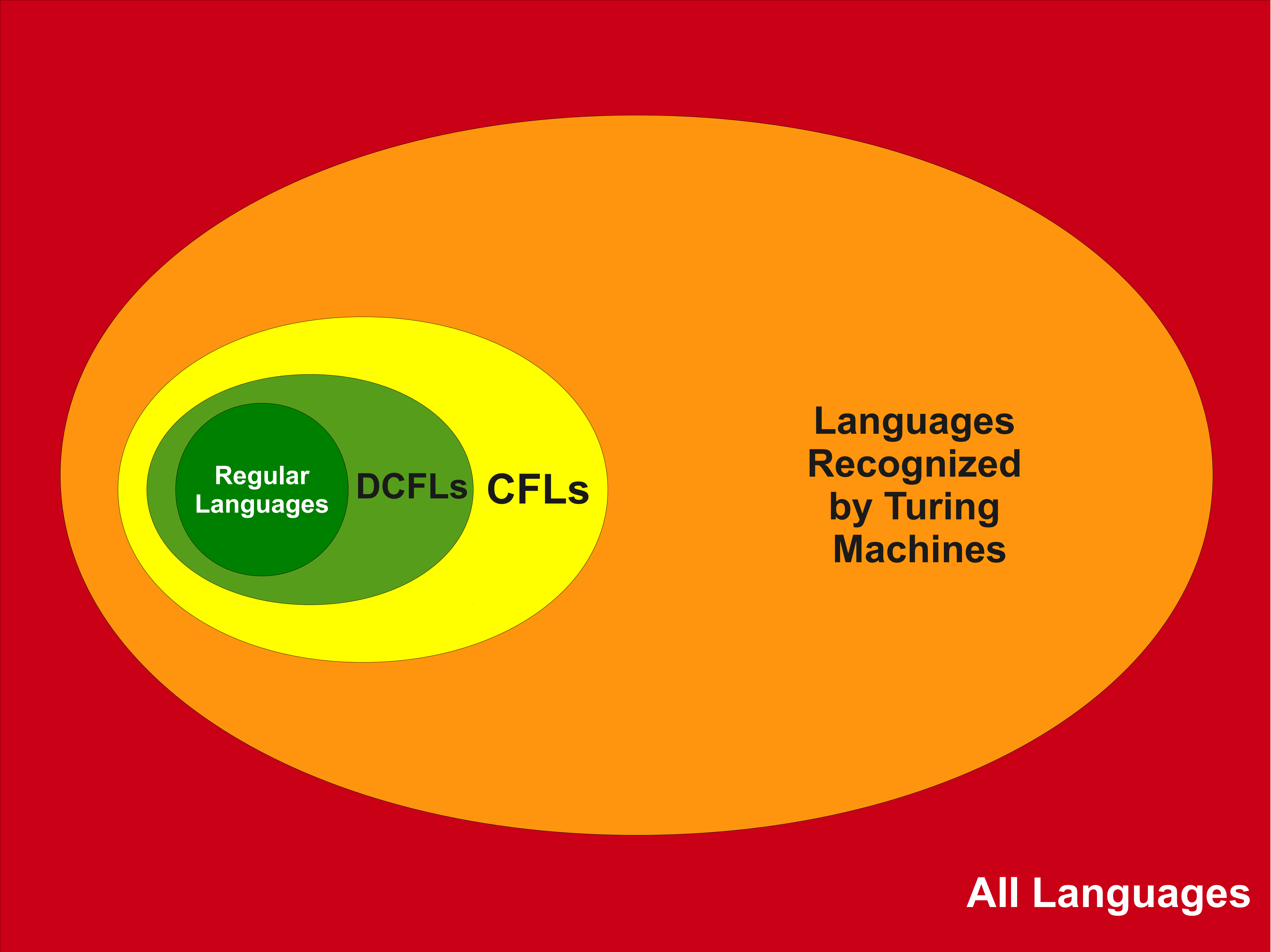
- Let M be a Turing machine.
- M **accepts** a string w if it enters the accept state when run on w.
- M **rejects** a string w if it enters the reject state when run on w.
- M **loops infinitely** (or just **loops**) on a string w if when run on w it enters neither the accept or reject state.
- M **does 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.

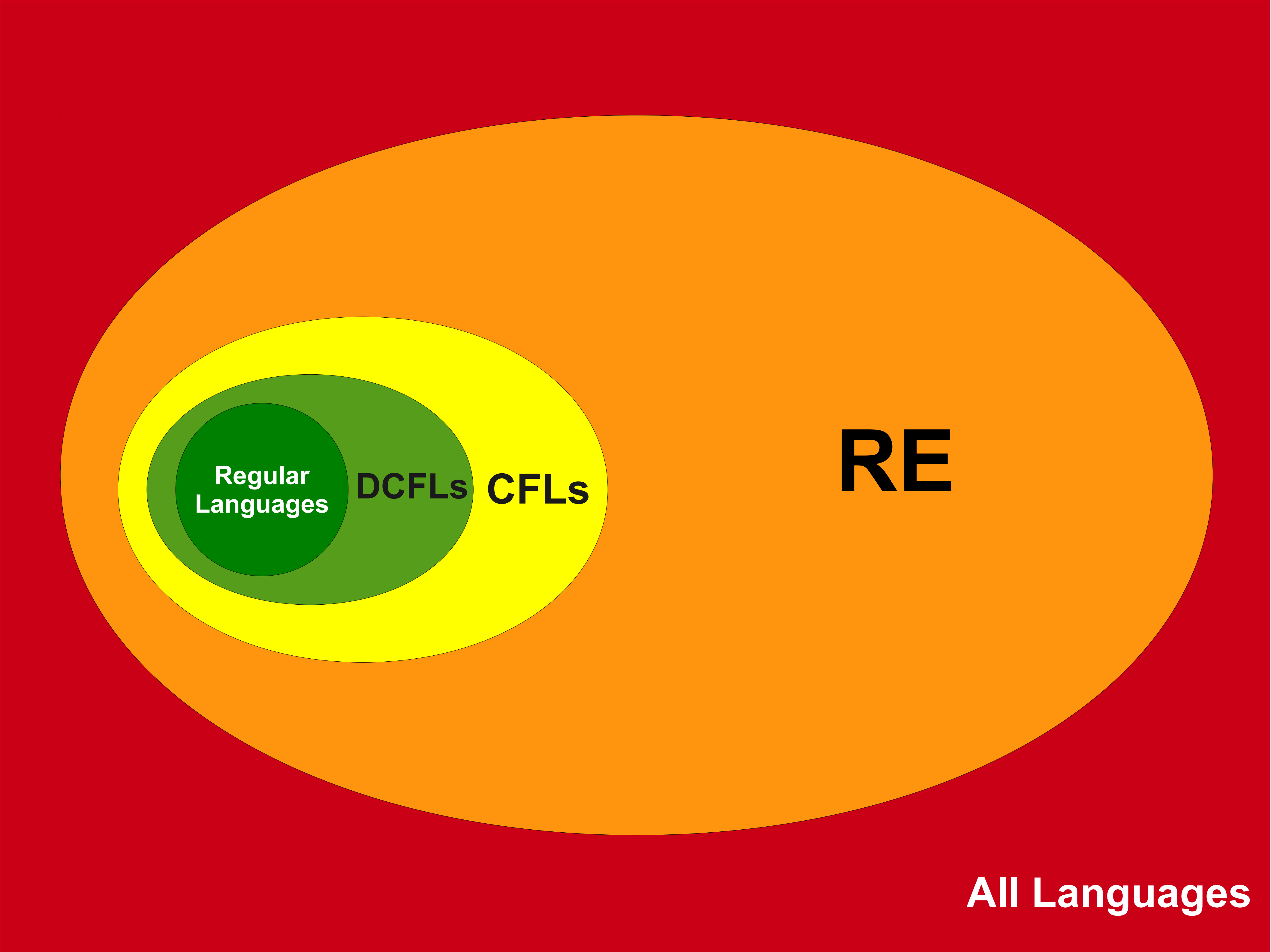


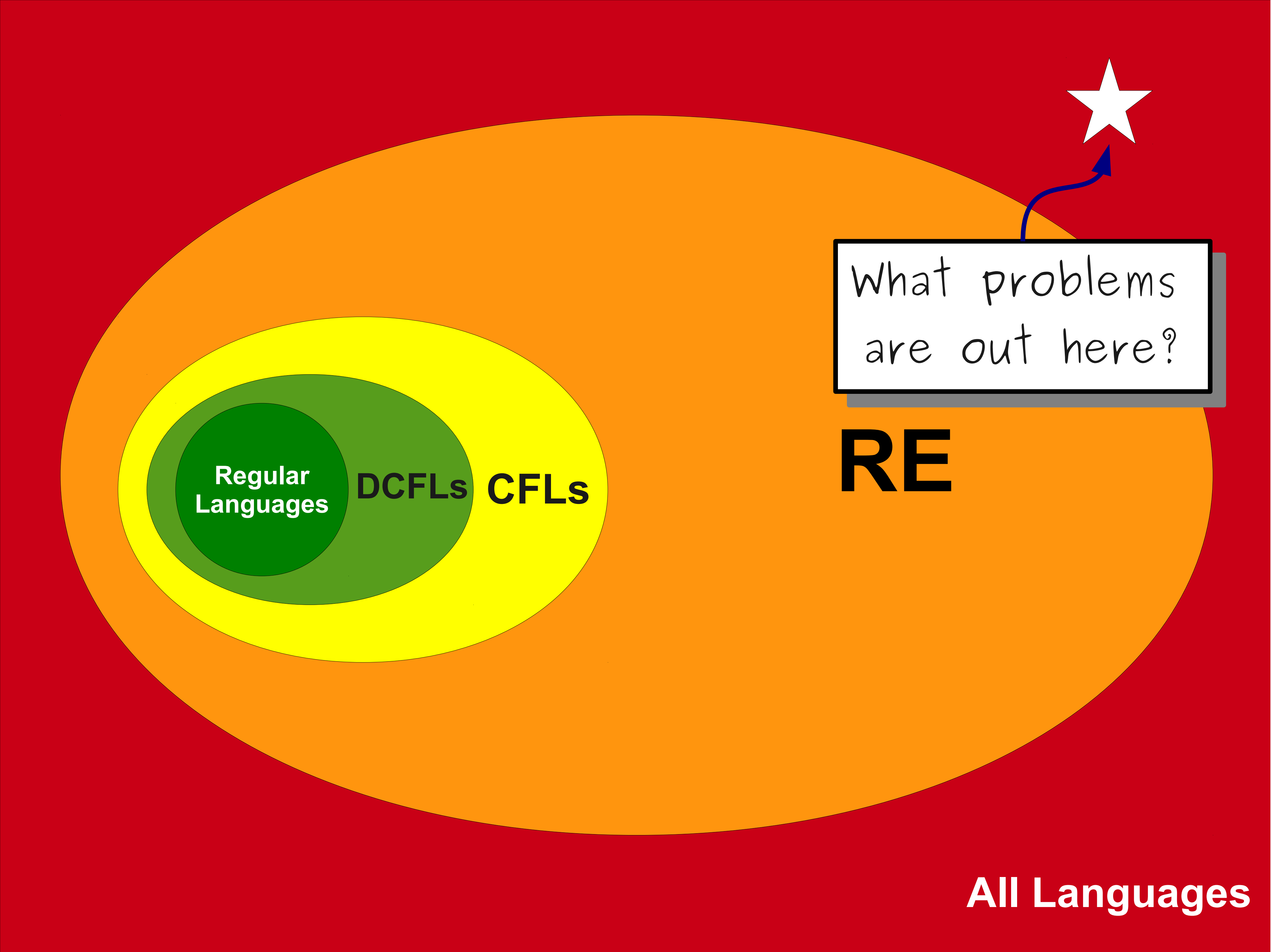
Major Ideas from Last Time

- **Worklists**
  - Turing machines can recognize many languages by exhaustively searching over all possible options until a solution is found.
  - If no answer is found, the machine might loop infinitely.
- **Nondeterministic Turing Machines**
  - NTMs can nondeterministically guess information that might be useful in solving a problem.
  - NTMs are equivalent in power to deterministic TMs (DTMs).
- **The Church-Turing Thesis**
  - Conjecture: All effective methods of computation are equal to or weaker than a TM.
  - Cannot be proven, but widely suspected to be true.


Regular Languages
DCFLs
CFLs
Problems Solvable by *Any Feasible Computing Machine*
All Languages


Regular Languages
DCFLs
CFLs
Languages Recognized by Turing Machines
All Languages


Regular Languages
DCFLs
CFLs
RE
All Languages


What problems are out here?
RE
Regular Languages
DCFLs
CFLs
All Languages

Outline for Today

- **High-Level Descriptions**
  - Given the Church-Turing thesis, how rigorous do we need to be in our description of TMs?
- **Encodings**
  - How do TMs compute over discrete structures?
- **The Universal Turing Machine**
  - Can TMs simulate other TMs?
- **Unsolvable Problems**
  - What problems cannot be solved by TMs?

High-Level Descriptions

The Church-Turing Thesis

- The Church-Turing thesis states that all effective models of computation are equivalent to or weaker than a Turing machine.
- As as a result, we can start to be less precise with our TM descriptions.

High-Level Descriptions

- A **high-level description** of a Turing machine is a description of the form

```
M = "On input x:
        Do something with x."
```

- Example:

```
M = "On input x:
        Repeat the following:
            If |x| ≤ 1, accept.
            If the first and last symbols of x aren't
                the same, reject.
            Remove the first and last characters of x."
```

High-Level Descriptions

- A **high-level description** of a Turing machine is a description of the form

```
M = "On input x:
        Do something with x."
```

- Example:

```
M = "On input x:
        Construct y, the reverse of x.
        If x = y, accept.
        Otherwise, reject."
```

High-Level Descriptions

- A **high-level description** of a Turing machine is a description of the form

  M = "On input x:
          Do something with x."

- Example:

  M = "On input x:

          If x is a palindrome, accept.

          Otherwise, reject."

High-Level Descriptions

- A **high-level description** of a Turing machine is a description of the form

```
M = "On input x:
        Do something with x."
```

- Example:

```
M = "On input x:
        Check that x has the form 0ⁿ1ᵐ2ᵖ.
        If not, reject.
        If nm = p, accept.
        Otherwise, reject."
```

High-Level Descriptions

- A **high-level description** of a Turing machine is a description of the form

```
$M$ = "On input $x$:
          Do something with $x$."
```

- Example:

```
$M$ = "On input $x$:
        Check that $x$ has the form $p?t$,
            where $p, t \in \{0, 1\}^*$.
        If not, reject.
        If so, if $p$ is a substring of $t$, accept.
        Otherwise, reject."
```

Formatted Input

- Many languages require the input to be in some particular format.
    - (Think about *ADD* or *SEARCH* from the problem sets).
- We can encode this directly into our TMs:

  M = "On input p`?`t, where p, $t \in \{$`0`, `1`$\}^*$

  If p is a substring of t, accept.

  Otherwise, reject."
- Machines of this form implicitly reject any inputs that don't have the right format.

Formatted Input

- Many languages require the input to be in some particular format.
  - (Think about *ADD* or *SEARCH* from the problem sets).
- We can encode this directly into our TMs:

```
M = “On input $0^m1^n2^p$:
        If $mn = p$, accept.
        Otherwise, reject.”
```

- Machines of this form implicitly reject any inputs that don't have the right format.

What's Allowed?

- Rule of thumb:

  You can include *anything* in a high-level description, as long as you could write a computer program for it.

- A few exceptions: can't get input from the user, make purely random decisions, etc.
- Unsure about what you can do? Try building the TM explicitly, or ask the course staff!

Encodings

Computing over Objects

- Turing machines always compute over strings.
- We have seen examples of automata that can *essentially* compute over other objects:
  - There and Back Again: Compute over paths.
  - Multiplication: Compute over numbers.
  - IDs: Compute over TM configurations.
- We have always said how we will encode objects:
  - e.g. $\{ \mathtt{1}^m\mathtt{×}\mathtt{1}^n\mathtt{=}\mathtt{1}^{mn} \mid m, n \in \mathbb{N} \}$

A Multitude of Encodings

- There can be many ways of encoding the same thing.
- Example: the natural number 13:
  - In unary: `1111111111111`
  - In binary: `1101`
  - In decimal: `13`
  - In skew binary: `120`
  - In Roman numerals: `XIII`
  - …
- **Claim:** Turing machines are sufficiently powerful to transform any one of these representations into any other of these representations.

An Abstract Idea of Encodings

- For simplicity, from this point forward we will make the following assumption:

For any finite, discrete object *O*, it is always possible to find some way of encoding *O* as a string.

- When working with Turing machines, it really doesn't matter *how* we do the encoding. A TM can convert any reasonable encoding scheme into any other encoding scheme.

Notation for Encodings

- For any object O, we will denote a string encoding of O by writing O in angle brackets: O is encoded as $\boldsymbol{\langle O \rangle}$.
- This makes it much easier to specify languages.
- Examples:

$$\{ \langle p \rangle \mid p \text{ is a path in 2D space that ends where it starts.} \}$$

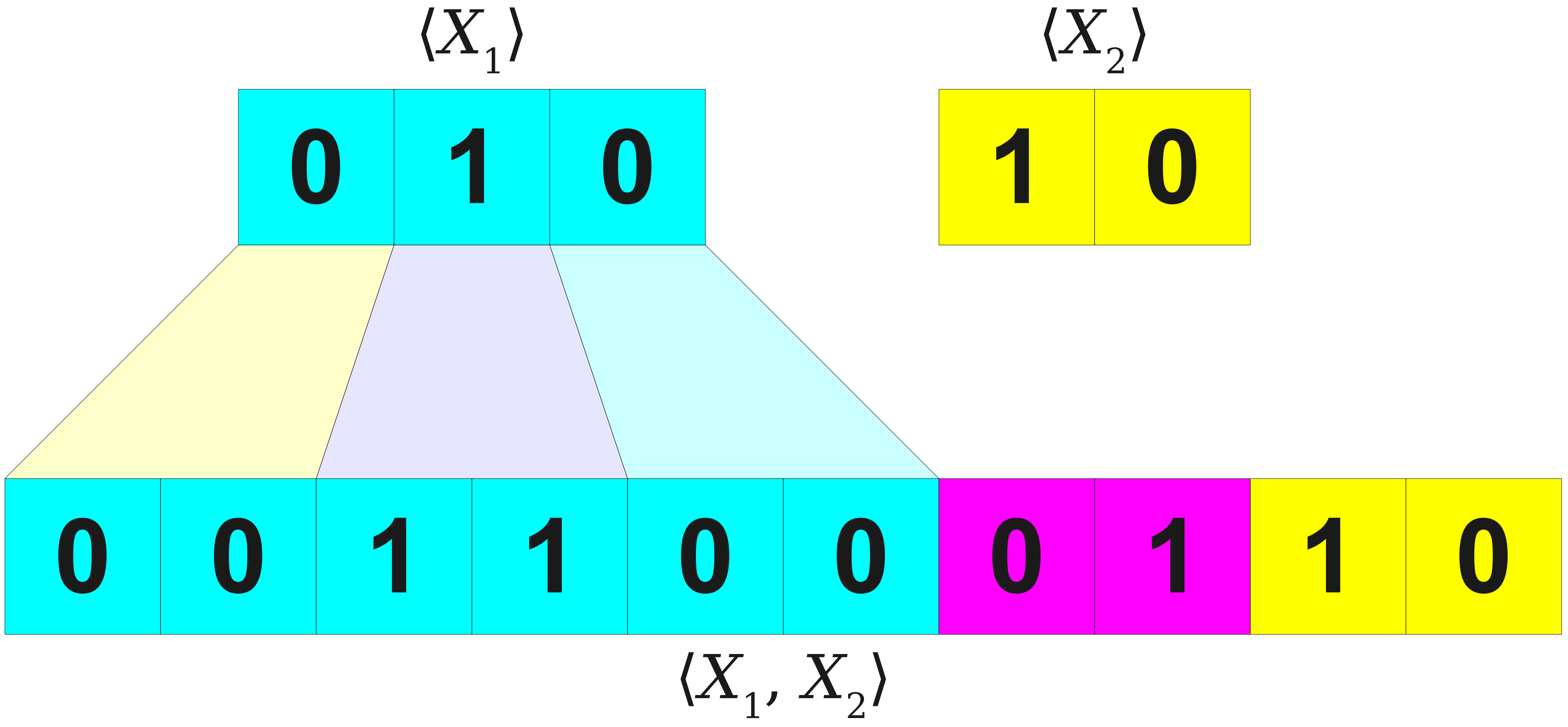
$$\{ \langle n \rangle \mid n \in \mathbb{N} \text{ and the hailstone sequence terminates for } n.\}$$

- The encoding scheme can make a difference when trying to determine whether a language is regular or context-free because of the relative weakness of DFAs and PDAs.

Encoding Multiple Objects

- Suppose that we want to provide an encoding of multiple objects.
    - Two natural numbers and their product.
    - A graph and a path in the graph.
    - “I just met you” and “this is crazy.”
- We can get encodings of each individual object.
- Can we make one string encoding all of these objects?

One Encoding Scheme



Encoding Multiple Objects

- Given several different objects $O_1, \ldots, O_n$, we can represent the encoding of those n objects as $\boldsymbol{\langle O_1, O_2, \ldots, O_n \rangle}$.
- Examples:
  - $\{ \langle m, n, mn \rangle \mid m, n \in \mathbb{N} \}$
  - $\{ \langle G, w \rangle \mid G$ is a context-free grammar that generates $w \}$

Encoding Turing Machines

- **Critically important fact**: Any Turing machine can be represented as a string.
- One way to do this: encode each state and its transitions in a list.
- Stronger claim: Any TM M can be represented as a string **in *M*'s alphabet**.
- Analogy: program source code.
  - All data fed into a program is encoded using the binary alphabet $\Sigma = \{0, 1\}$.
  - A program's source code is itself represented in binary on disk.

We can now encode TMs as strings.

TMs can accept strings as input.

What can we do with this knowledge?

Universal Machines

Universal Machines and Programs

- **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.
- As a high-level description:

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

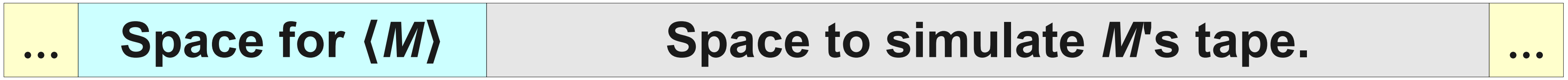
Run M on w.

If M accepts w, U_{TM} accepts $\langle M, w\rangle$.

If M rejects w, U_{TM} rejects $\langle M, w\rangle$.

If M loops on w, then U_{TM} loops as well. This is subtly but implicitly given in the description.

Building a Universal TM



```
U_TM = "On input ⟨M, w⟩:
    • Copy M to one part of the tape and w to another.
    • Place a marker in w to track M's current state and the
      position of its tape head.
    • Repeat the following:
        • If the simulated version of M has entered an accepting
          state, accept.
        • If the simulated version of M has entered a rejecting
          state, reject.
        • Otherwise:
            • Consult M's simulated tape to determine what symbol
              is currently being read.
            • Consult the encoding of M to determine how to
              simulate the transition.
            • Simulate that transition."
```

A Major Observation

- The universal Turing machine U_{TM} is capable of performing any computation that could ever be computed by any feasible computing machine.
- Why?
  - By the Church-Turing thesis, if there is a feasible computing machine, it can be converted into an equivalent TM *M*.
  - U_{TM} can then simulate the behavior of *M*.

The Language of U_{TM}

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

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

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

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

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

- For simplicity, define $\boldsymbol{A_{TM} = \mathscr{L}(U_{TM})}$. We will use the language A_{TM} extensively.

Computing on Turing Machines

Computing on Turing Machines

- Given a Turing machine *M*, we might be interested in answering some questions about *M*.
- Sample questions:
  - Does *M* accept the empty string?
  - Does *M* every accept any strings at all?
  - Does *M* accept the same set of strings as a TM *M*'?
  - Does *M* enter an infinite loop on any string?
- Each of these questions has a definitive yes or no answer.
- **Major question:** What sorts of questions about Turing machines can be answered by Turing machines?

Using U_{TM}

- Turing machines can answer many interesting questions about other Turing machines.
- Much of this is due to the existence of U_{TM}: to figure out what a TM does, we can try simulating it and seeing what happens.
- Let's see some examples of this.

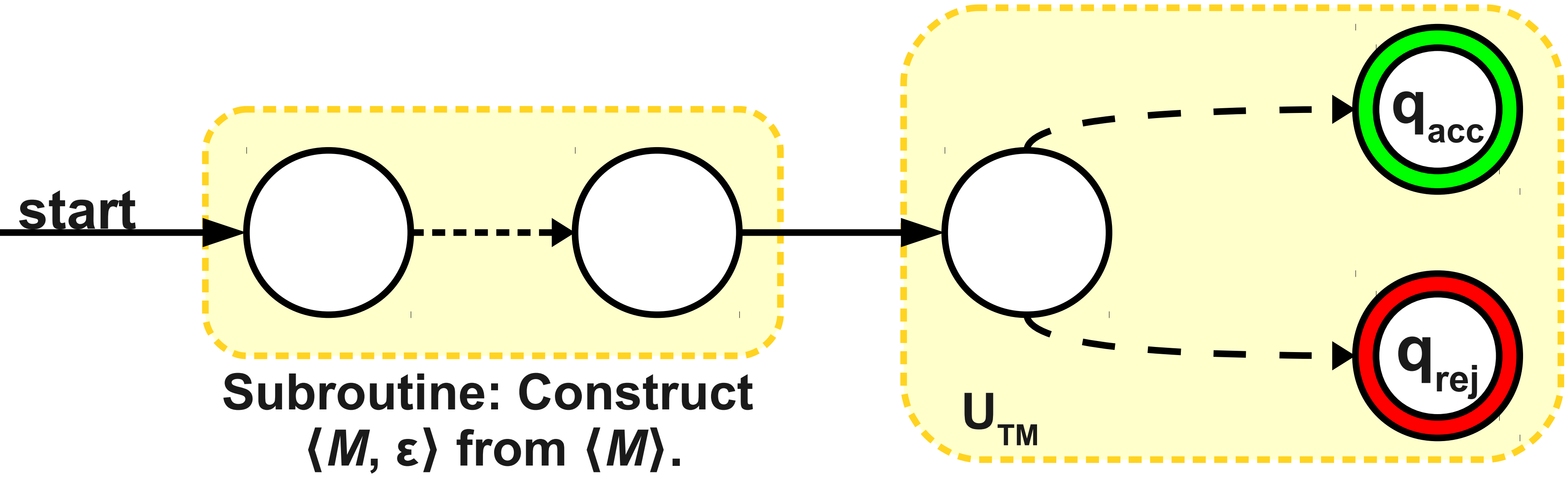
Accepting the Empty String

- Given a TM M, that TM either accepts ε or it does not accept ε.
  - Recall: M does not accept ε iff M either rejects ε or it loops on ε.
- Consider the language

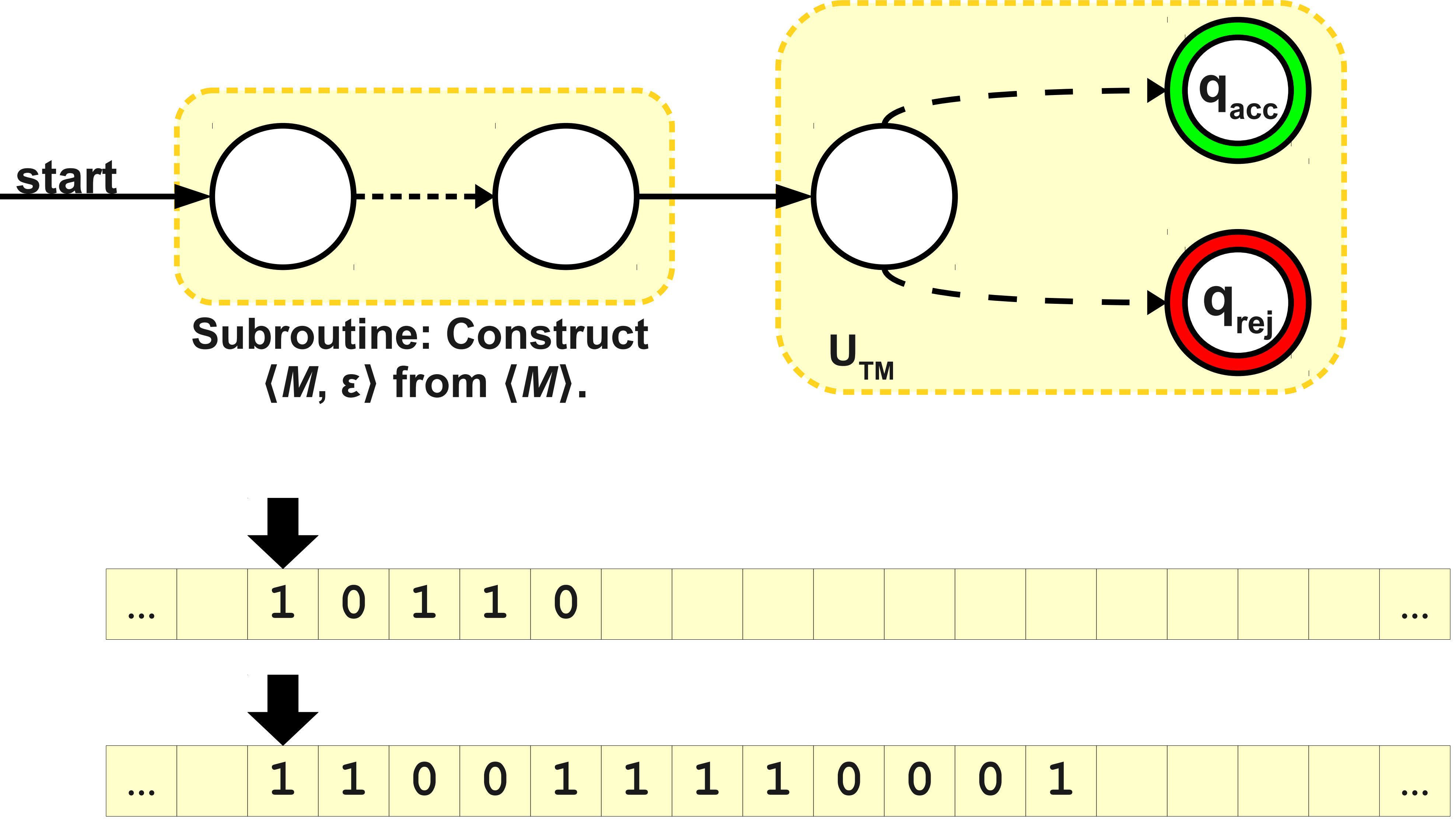
  ***L* = { ⟨*M*⟩ | *M* is a TM and *M* accepts ε }**

- Is $L \in$ **RE**? That is, is there a Turing machine M such that $\mathscr{L}(M) = L$?

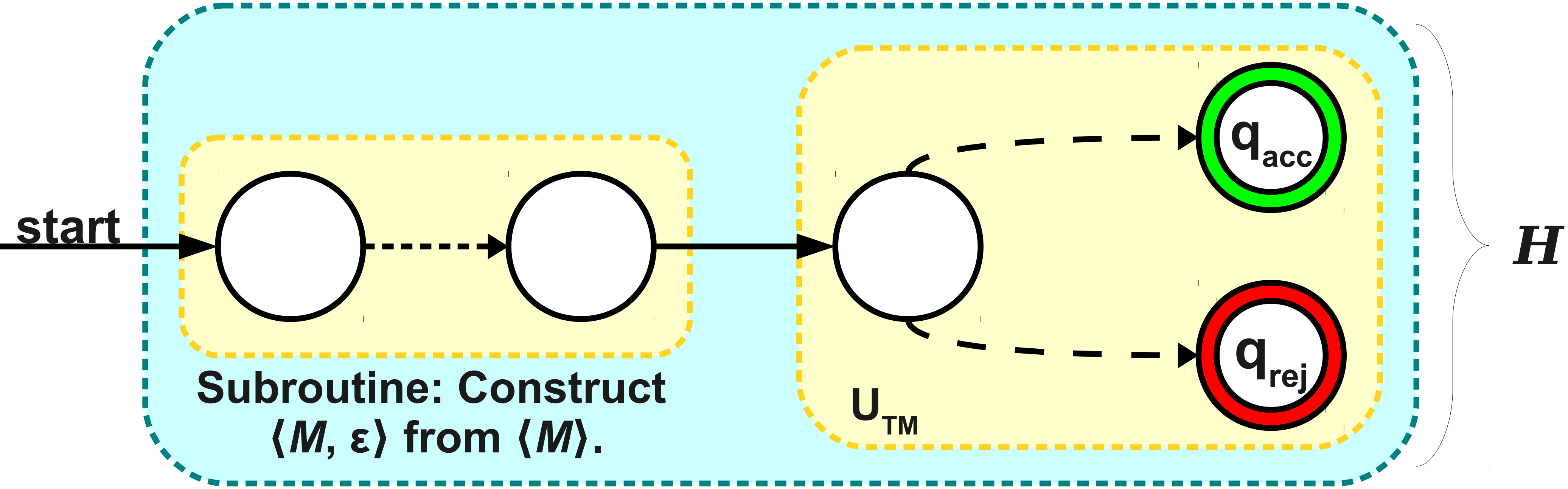
$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$



$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$



$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$



H = "On input ⟨*M*⟩, where *M* is a Turing machine:

- Construct the string ⟨*M*, ε⟩.
- Using U_{TM} as a subroutine, run U_{TM} on ⟨*M*, ε⟩.
- If U_{TM} accepts ⟨*M*, ε⟩, *H* accepts ⟨*M*⟩.
- If U_{TM} rejects ⟨*M*, ε⟩, *H* rejects ⟨*M*⟩."

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts ε }

H = "On input $\langle M \rangle$, where M is a Turing machine:

- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts $\varepsilon \}$

```
H = "On input ⟨M⟩, where M is a Turing machine:
        · Construct the string ⟨M, ε⟩.
        · Using U_TM as a subroutine, run U_TM on ⟨M, ε⟩.
        · If U_TM accepts ⟨M, ε⟩, H accepts ⟨M⟩.
        · If U_TM rejects ⟨M, ε⟩, H rejects ⟨M⟩."
```

What does H do on input $\langle M \rangle$ if M accepts ε?

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts $\varepsilon \}$

```
H = "On input ⟨M⟩, where M is a Turing machine:
        · Construct the string ⟨M, ε⟩.
        · Using U_TM as a subroutine, run U_TM on ⟨M, ε⟩.
        · If U_TM accepts ⟨M, ε⟩, H accepts ⟨M⟩.
        · If U_TM rejects ⟨M, ε⟩, H rejects ⟨M⟩."
```

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts $\varepsilon \}$

H = "On input $\langle M \rangle$, where M is a Turing machine:

- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε?

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts $\varepsilon \}$

H = "On input $\langle M \rangle$, where M is a Turing machine:

- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε? **Rejects**.

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts ε $\}$

H = "On input $\langle M \rangle$, where M is a Turing machine:

- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε? **Rejects**.

What does H do on input $\langle M \rangle$ if M loops on ε?

$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$

```
H = "On input ⟨M⟩, where M is a Turing machine:
        · Construct the string ⟨M, ε⟩.
        · Using $U_{TM}$ as a subroutine, run $U_{TM}$ on ⟨M, ε⟩.
        · If $U_{TM}$ accepts ⟨M, ε⟩, H accepts ⟨M⟩.
        · If $U_{TM}$ rejects ⟨M, ε⟩, H rejects ⟨M⟩."
```

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε? **Rejects**.

What does H do on input $\langle M \rangle$ if M loops on ε? **Loops**.

$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$

H = "On input ⟨*M*⟩, where *M* is a Turing machine:

- Construct the string ⟨*M*, ε⟩.
- Using U_{TM} as a subroutine, run U_{TM} on ⟨*M*, ε⟩.
- If U_{TM} accepts ⟨*M*, ε⟩, *H* accepts ⟨*M*⟩.
- If U_{TM} rejects ⟨*M*, ε⟩, *H* rejects ⟨*M*⟩."

What does *H* do on input ⟨*M*⟩ if *M* accepts ε? **Accepts**.

What does *H* do on input ⟨*M*⟩ if *M* rejects ε? **Rejects**.

What does *H* do on input ⟨*M*⟩ if *M* loops on ε? **Loops**.

H accepts ⟨*M*⟩ iff U_{TM} accepts ⟨*M*, ε⟩

$L = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ accepts } \varepsilon \}$

H = "On input $\langle M \rangle$, where M is a Turing machine:
- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε? **Rejects**.

What does H do on input $\langle M \rangle$ if M loops on ε? **Loops**.

H accepts $\langle M \rangle$ iff U_{TM} accepts $\langle M, \varepsilon \rangle$ iff M accepts ε

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts $\varepsilon \}$

H = "On input $\langle M \rangle$, where M is a Turing machine:

- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε? **Rejects**.

What does H do on input $\langle M \rangle$ if M loops on ε? **Loops**.

H accepts $\langle M \rangle$ iff U_{TM} accepts $\langle M, \varepsilon \rangle$ iff M accepts ε iff $\langle M \rangle \in L$.

$L = \{ \langle M \rangle \mid M$ is a TM and M accepts $\varepsilon \}$

H = "On input $\langle M \rangle$, where M is a Turing machine:

- Construct the string $\langle M, \varepsilon \rangle$.
- Using U_{TM} as a subroutine, run U_{TM} on $\langle M, \varepsilon \rangle$.
- If U_{TM} accepts $\langle M, \varepsilon \rangle$, H accepts $\langle M \rangle$.
- If U_{TM} rejects $\langle M, \varepsilon \rangle$, H rejects $\langle M \rangle$."

What does H do on input $\langle M \rangle$ if M accepts ε? **Accepts**.

What does H do on input $\langle M \rangle$ if M rejects ε? **Rejects**.

What does H do on input $\langle M \rangle$ if M loops on ε? **Loops**.

H accepts $\langle M \rangle$ iff U_{TM} accepts $\langle M, \varepsilon \rangle$ iff M accepts ε iff $\langle M \rangle \in L$.

Therefore, $\boldsymbol{\mathscr{L}(H) = L}$.

TMs that Run Other TMs

- The description of the TM from the previous slide was very explicit about the operation of the machine:

```
H = “On input ⟨M⟩, where M is a Turing machine:
        · Construct the string ⟨M, ε⟩.
        · Using U_TM as a subroutine, run U_TM on ⟨M, ε⟩.
        · If U_TM accepts ⟨M, ε⟩, H accepts ⟨M⟩.
        · If U_TM rejects ⟨M, ε⟩, H rejects ⟨M⟩.”
```

- The key idea here is that *H* tries to run the TM *M* on the string ε. The details of *how* *H* does this are not particularly relevant.

TMs that Run Other TMs

- Here is a different high-level description of the TM H that is totally acceptable:

```
H = "On input ⟨M⟩, where M is a Turing machine:
        · Run M on ε.
        · If M accepts ε, then H accepts ⟨M⟩.
        · If M rejects ε, then H rejects ⟨M⟩.
```

- This makes clear what the major point of the TM is. We can fill in the details if we want if we're curious how it would work, but it's not necessary.

A More Complex Application

- Given a TM *M*, that TM either accepts at least one string or it does not accept at least one string.
- We can consider the language of all TMs that do accept at least one string:

$$\boldsymbol{L_{\text{ne}} = \{ \langle M \rangle \mid M \text{ is a TM that accepts at least one string } \}}$$

- Question: Could we build a TM that can recognize TMs that accept at least one input?
- Equivalently, is $L_{\text{ne}} \in \mathbf{RE}$?

An Interesting Observation

L_{ne} **= { ⟨*M*⟩ | *M* is a TM that accepts at least one string }**

- Suppose you are given a TM *M* that accepts at least one string.
- If you knew what that string was, could you confirm that the TM indeed accepted it?
- **Yes**: Just run the TM and wait for it to accept!

Designing an NTM

L_{ne} = { ⟨*M*⟩ | *M* is a TM that accepts at least one string }

- We can build an NTM for L_{ne} that works as follows:
  - **Nondeterministically** guess a string *w*.
  - **Deterministically** run *M* on *w* and accept if *M* accepts *w*.

H = "On input ⟨*M*⟩, where *M* is a Turing machine:
- Nondeterministically guess a string *w*.
- Run *M* on *w*.
- If *M* accepts *w*, then *H* accepts ⟨*M*⟩.
- If *M* rejects *w*, then *H* rejects ⟨*M*⟩.

The Story So Far

- We can now encode arbitrary objects, *including Turing machines*, as strings.
- Turing machines are capable of running other Turing machines specified through TM encodings.
- Some properties of TMs are **RE** – there exists a TM that can confirm when other TMs have that property.

Timeline of CS103

- **Lecture 00:** Unsolvable problems exist.
- **Lecture 07:** Proof by diagonalization.
- **Lecture 18:** TMs formalize computation.
- **Lecture 19:** TMs can be encoded as strings.

We are finally ready to start answering the following question:

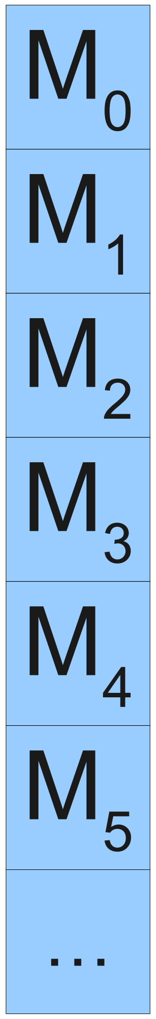
What problems cannot be solved by a computer?

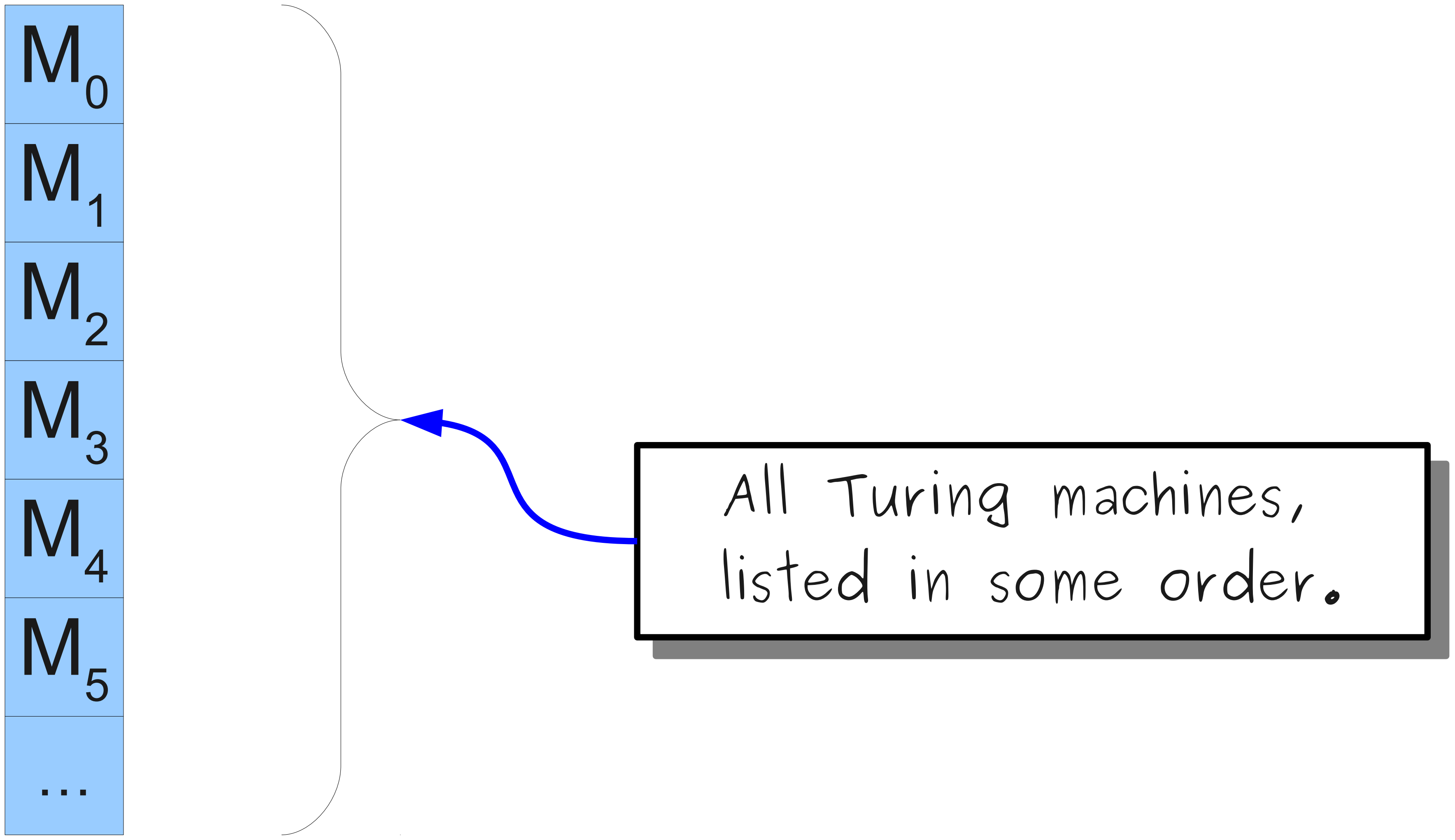
Languages, TMs, and TM Encodings

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

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

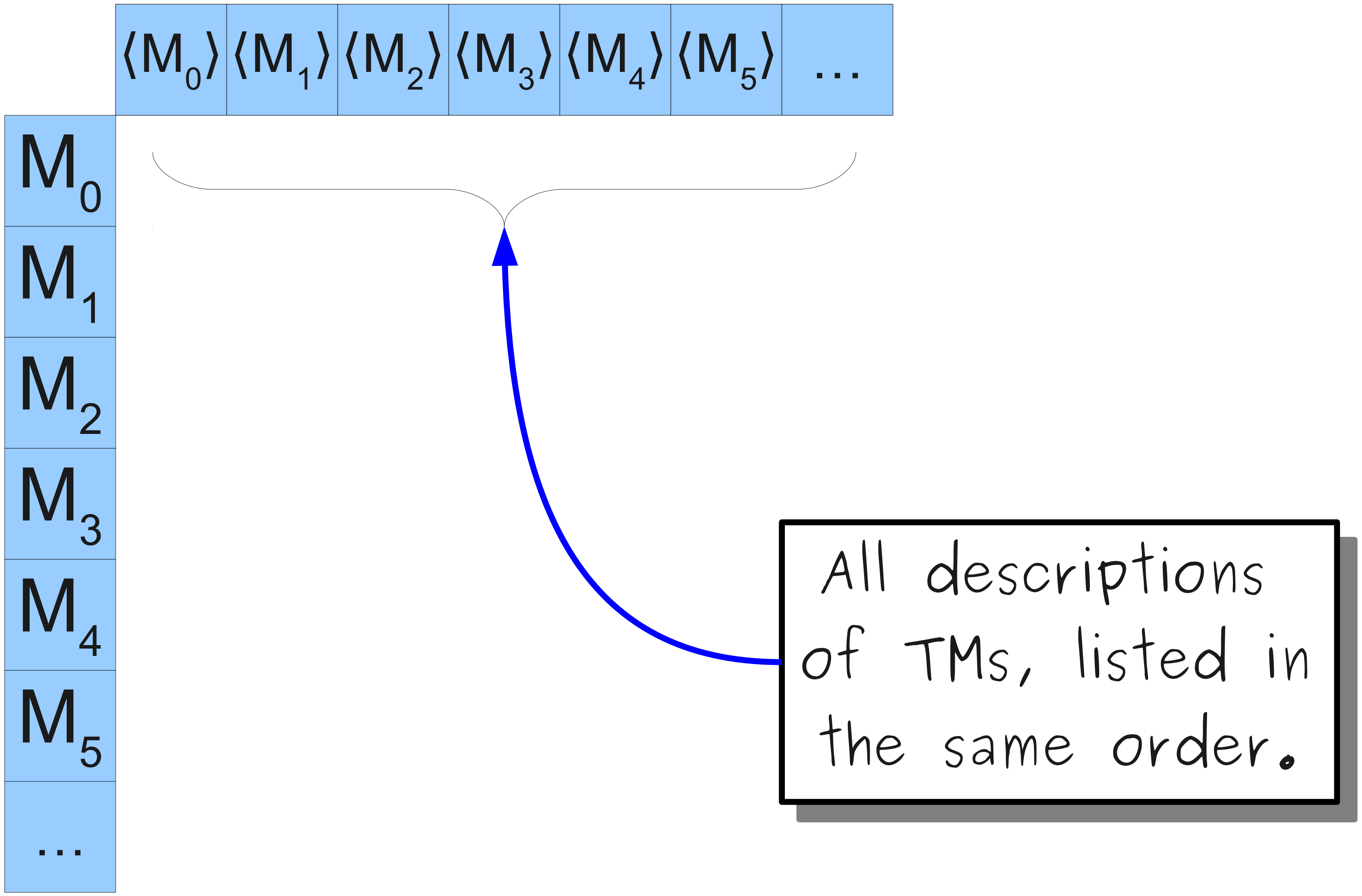
- Some of the strings in this set might be descriptions of TMs.
- What happens if we just focus on the set of strings that are legal TM descriptions?


M_0
M_1
M_2
M_3
M_4
M_5
...



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

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



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

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

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

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

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

	⟨M_0⟩	⟨M_1⟩	⟨M_2⟩	⟨M_3⟩	⟨M_4⟩	⟨M_5⟩	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5							
…							

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

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

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

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

Acc	Acc	Acc	No	Acc	No	…

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

No	No	No	Acc	No	Acc	…

Flip all "accept" to "no" and vice-versa

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

What TM has this behavior?

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

No TM has this behavior!

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

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

No	No	No	Acc	No	Acc	…

"The language of all TMs that do not accept their own description."

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

No	No	No	Acc	No	Acc	…

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

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

No	No	No	Acc	No	Acc	…

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

Diagonalization Revisited

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$. Since $\langle R \rangle \in \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \in L_D$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$. Since $\langle R \rangle \in \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \in L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \notin L_D$.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$. Since $\langle R \rangle \in \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \in L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \notin L_D$.

In either case we reach a contradiction, so our assumption must have been wrong.

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

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

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$. Since $\langle R \rangle \in \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \in L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \notin L_D$.

In either case we reach a contradiction, so our assumption must have been wrong. Thus $L_D \notin$ **RE**.

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

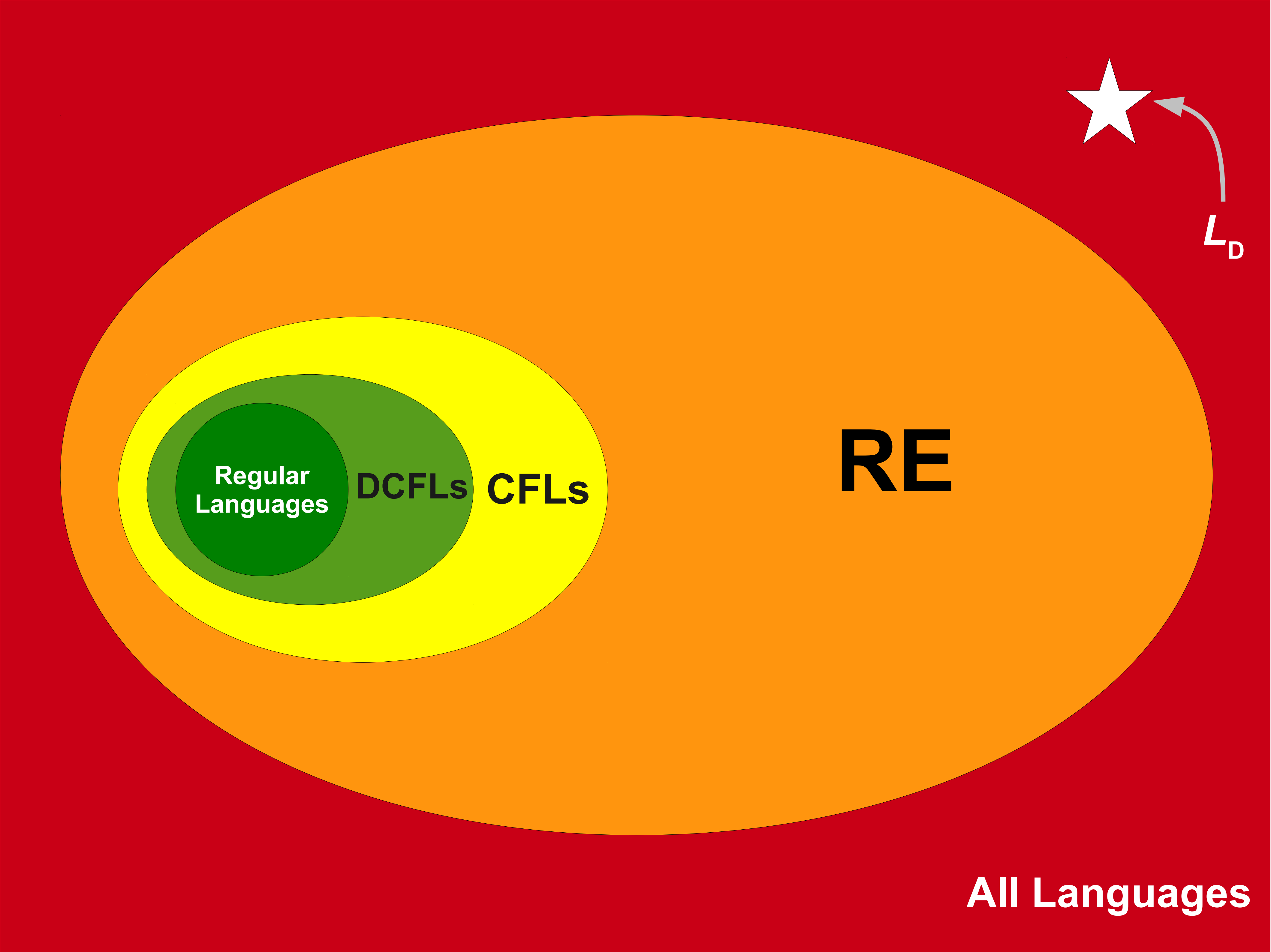
Theorem: $L_D \notin$ **RE**.

Proof: By contradiction; assume that $L_D \in$ **RE**. Then there must be some TM R such that $\mathscr{L}(R) = L_D$. We know that either $\langle R \rangle \notin \mathscr{L}(R)$ or $\langle R \rangle \in \mathscr{L}(R)$. We consider each case separately:

Case 1: $\langle R \rangle \notin \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \notin \mathscr{L}(R)$, we know that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \notin L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \in L_D$.

Case 2: $\langle R \rangle \in \mathscr{L}(R)$. By definition of L_D, since $\langle R \rangle \in \mathscr{L}(R)$, we know that $\langle R \rangle \notin L_D$. Since $\langle R \rangle \in \mathscr{L}(R)$ and $\mathscr{L}(R) = L_D$, we know that $\langle R \rangle \in L_D$. But this is impossible, since it contradicts the fact that $\langle R \rangle \notin L_D$.

In either case we reach a contradiction, so our assumption must have been wrong. Thus $L_D \notin$ **RE**. ■


Regular Languages
DCFLs
CFLs
RE
L_D
All Languages

Next Time

- **Decidability**
  - How do we formalize the definition of an algorithm?
  - What problems can we learn the answer to?
- **Undecidable Problems**
  - What problems cannot be solved by algorithms?