Unsolvable Problems

Problem set six is due in the box up front.

Encodings

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 $\langle O \rangle$.
- This makes it much easier to specify languages.
- Examples:

  $\{ \langle R \rangle \mid R \text{ is a regular expression that matches } \varepsilon \}$

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

- 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 CFGs.

Encoding Multiple Objects

- Given several different objects O_1, …, O_n, we can represent the encoding of those n objects as $\langle O_1, O_2, \ldots, O_n \rangle$.
- 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 $\Sigma = \{\texttt{0}, \texttt{1}\}$.
  - 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 **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.

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!

Building a Universal TM

U_{TM} = "On input $\langle M, w \rangle$:

- 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."

The Language of U_{TM}

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

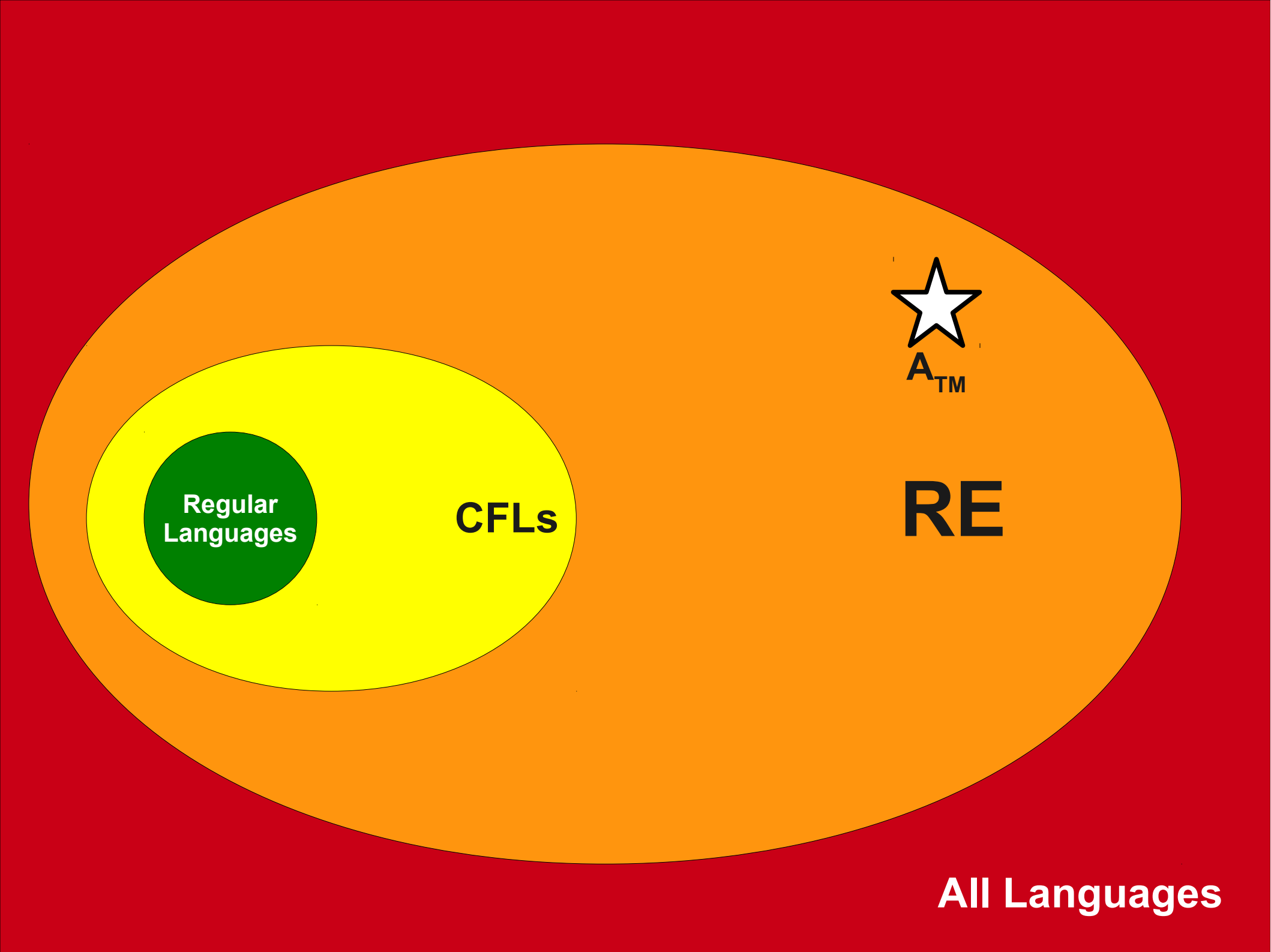
$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$

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


Regular Languages
CFLs
A_{TM}
RE
All Languages

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?
- In other words: *What questions can Turing machines answer about themselves?*

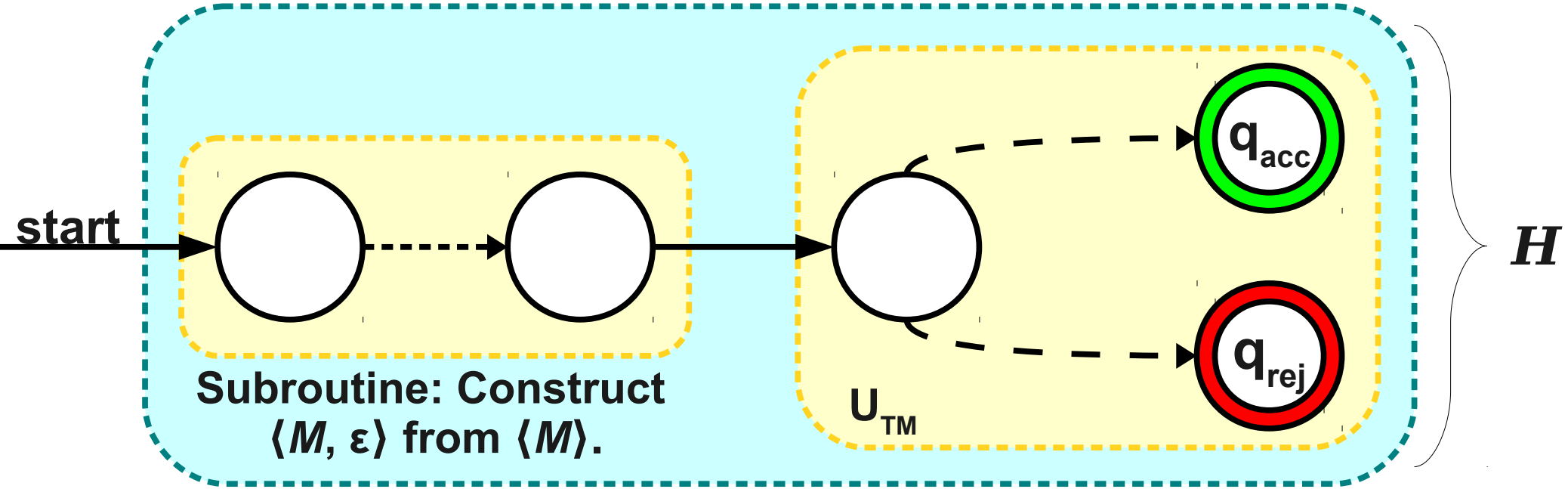
Accepting the Empty String

- 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

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

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

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



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

- Construct the string $\langle M, \varepsilon \rangle$.
- 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 $\langle M \rangle$."

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

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

- Construct the string $\langle M, \varepsilon \rangle$.
- 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, $\mathscr{L}(H) = L$.

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 $\langle M \rangle$, where M is a Turing machine:
>
> - Construct the string $\langle M, \varepsilon \rangle$.
> - 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 $\langle M \rangle$."

- 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 $\langle M \rangle$, where M is a Turing machine:
>
> - Run M on ε.
> - If M accepts ε, then H accepts $\langle M \rangle$.
> - If M rejects ε, then H rejects $\langle M \rangle$.

- 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:

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

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

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 09:** Proof by diagonalization.
- **Lecture 19:** TMs formalize computation.
- **Lecture 20:** TMs can be encoded as strings.

We are finally ready to start answering the following question:

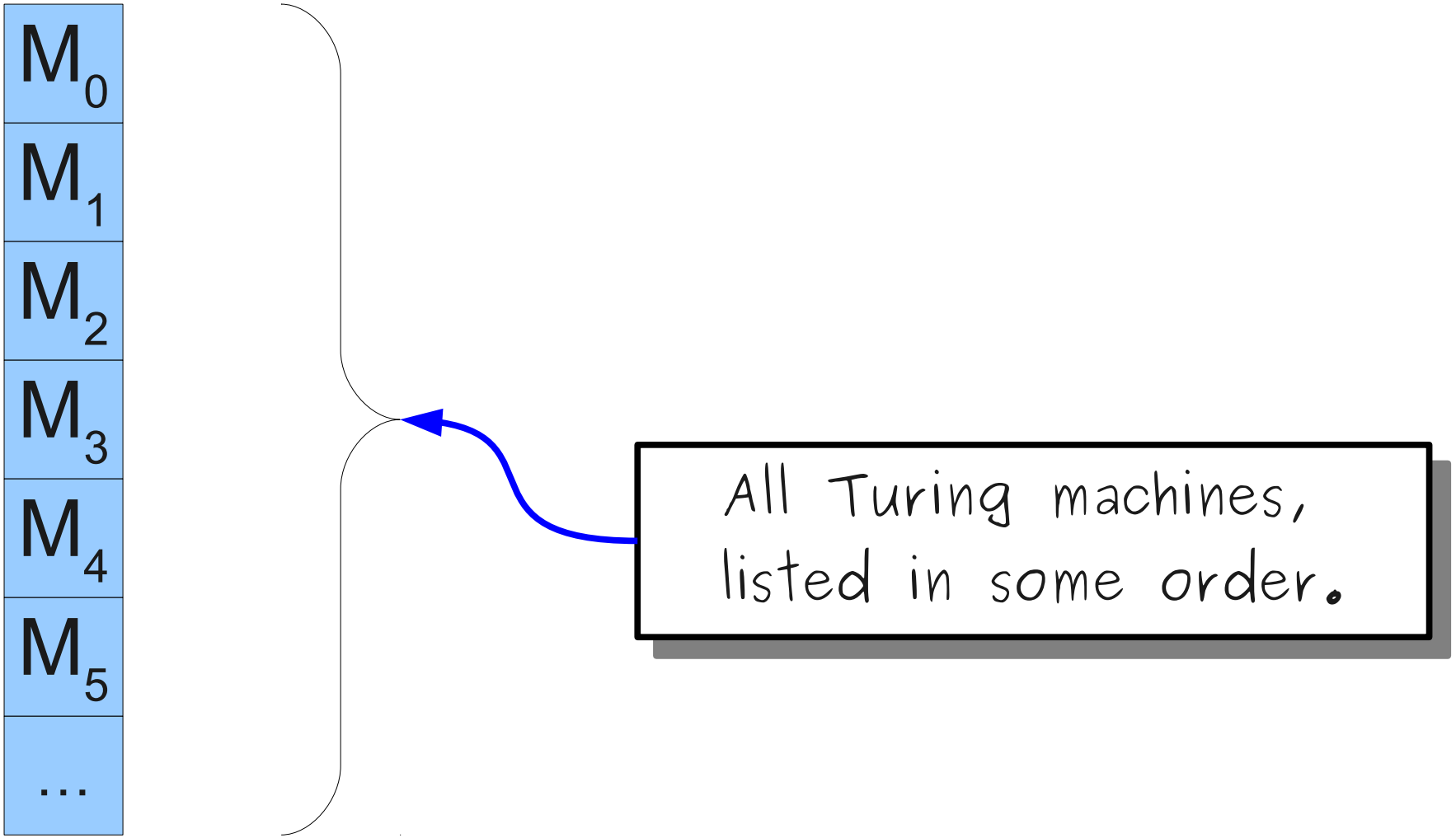
What problems cannot be solved by a computer?

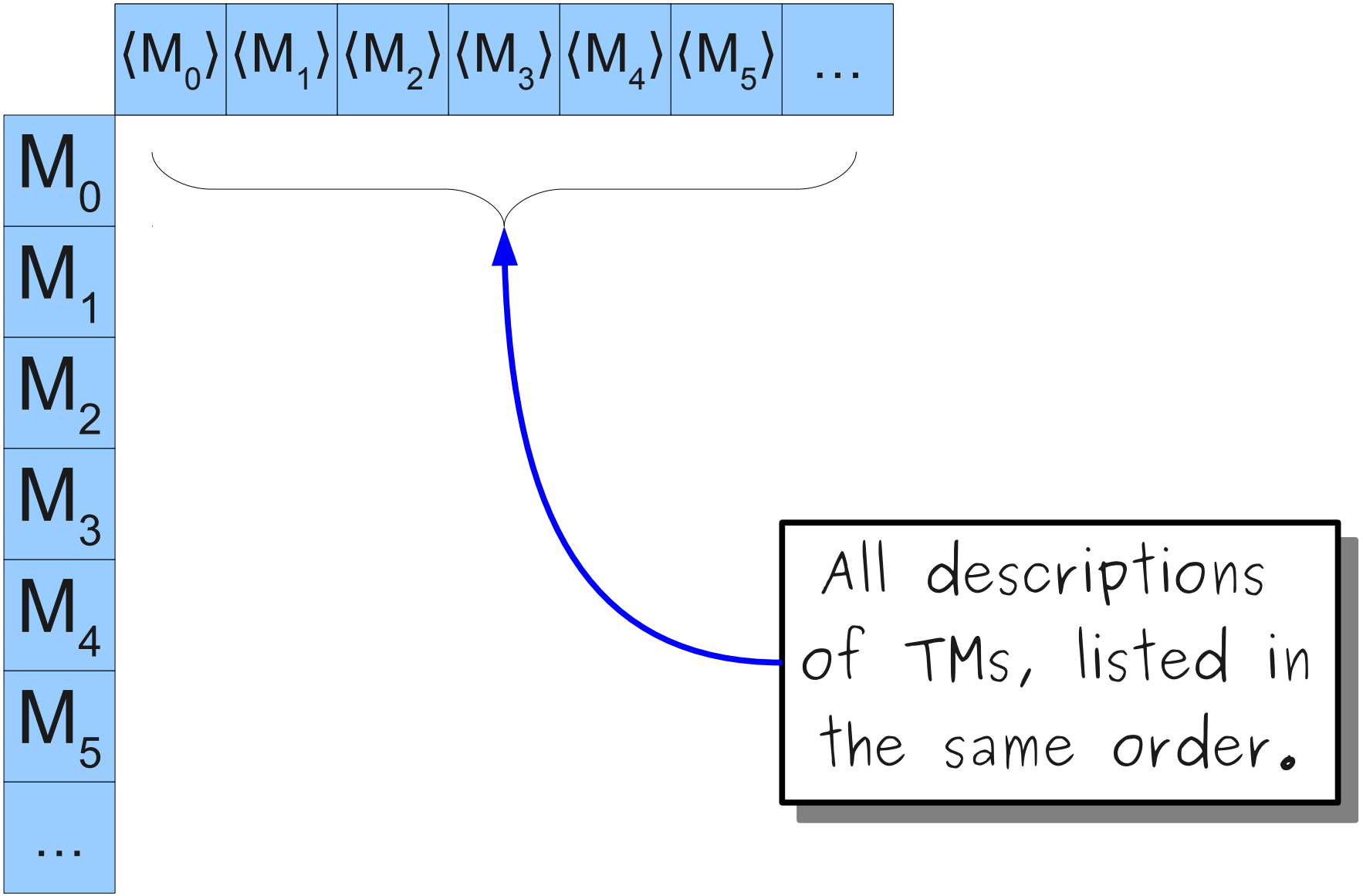
Languages, TMs, and TM Encodings

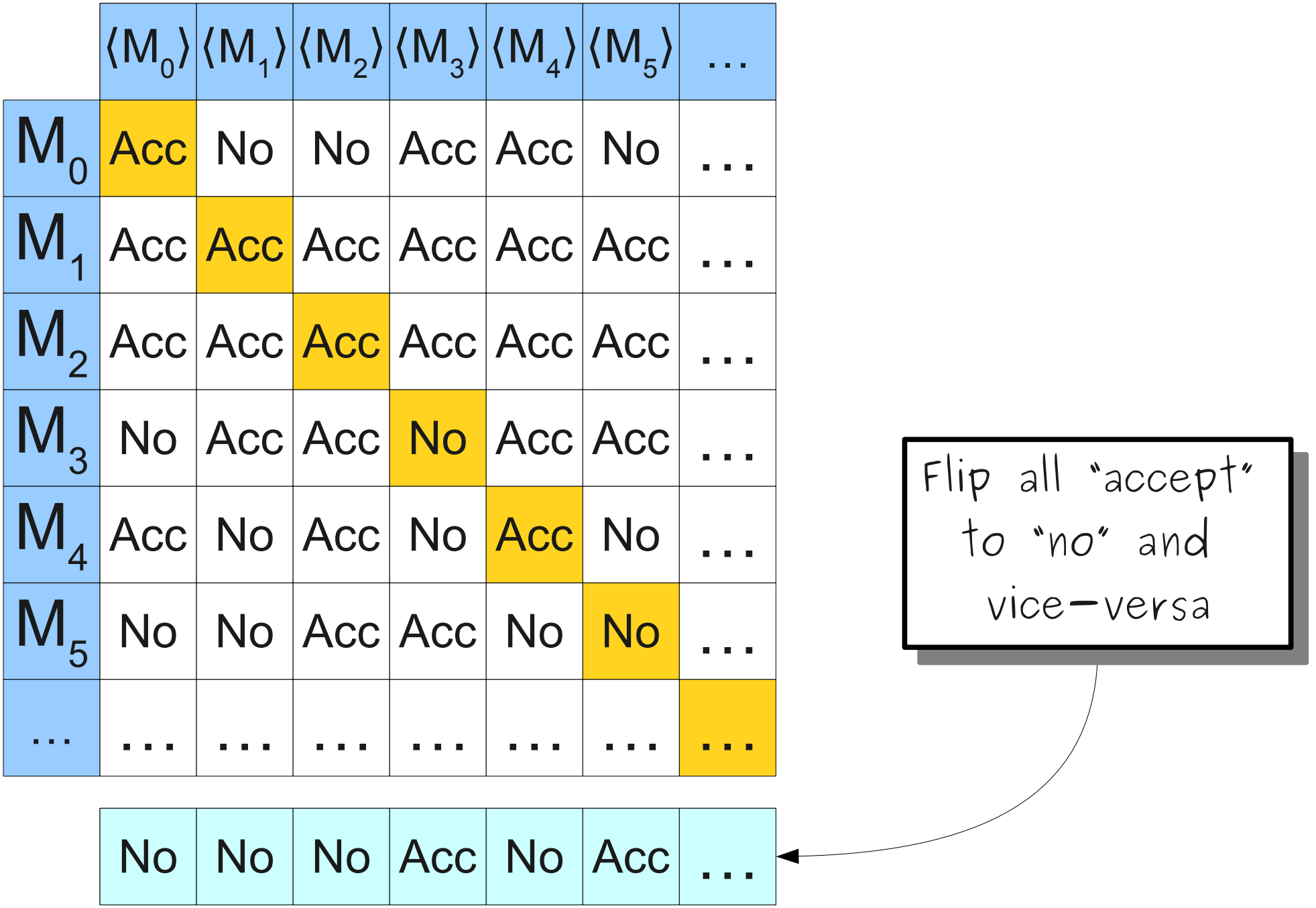
- Recall: The language of a TM M is the set

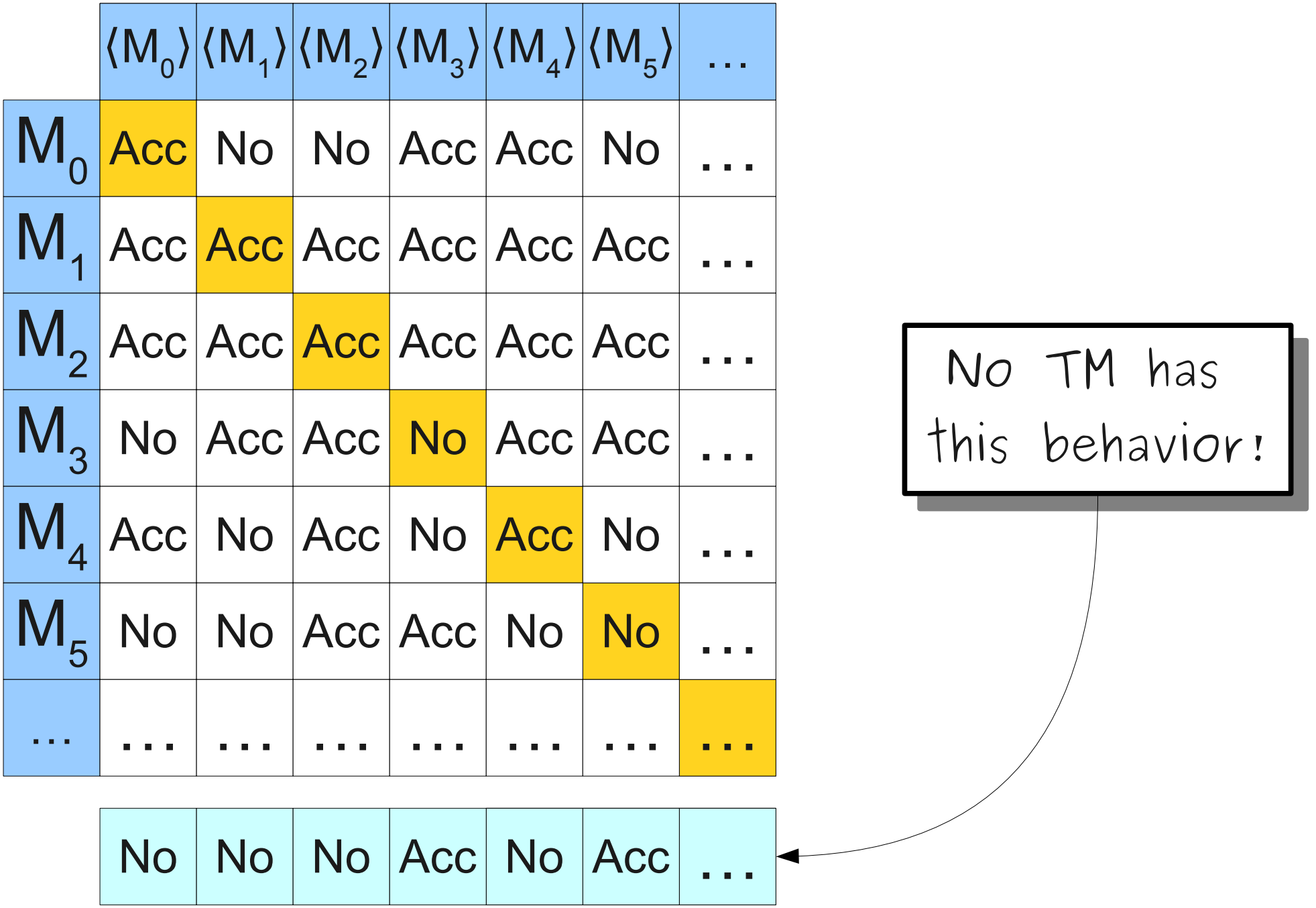
$$\mathscr{L}(M) = \{\ w \in \Sigma^* \mid M \text{ accepts } w\ \}$$

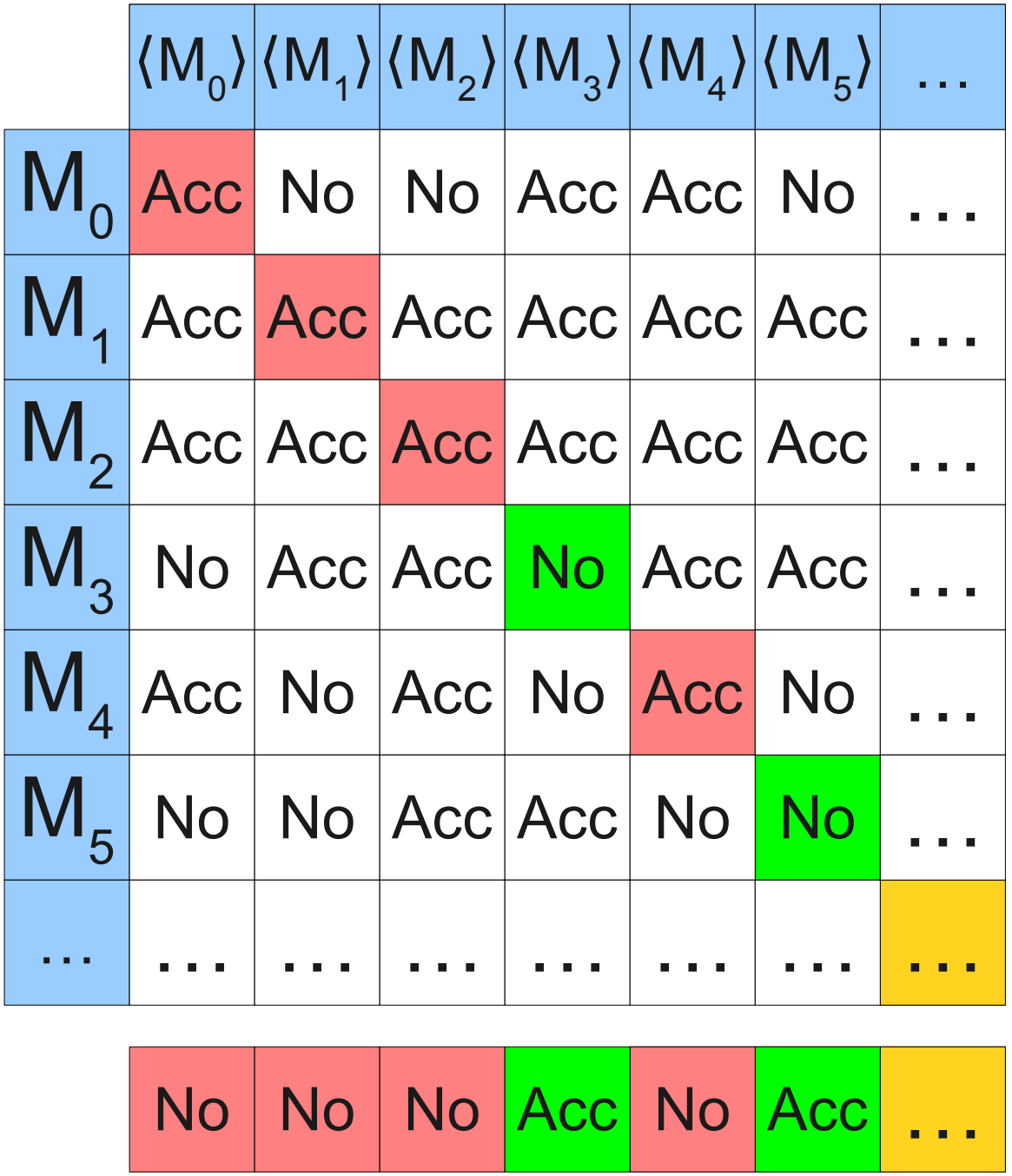
- Some of the strings in this set might be descriptions of TMs.
- What happens if we just focus on the set of strings that are legal TM descriptions?


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


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


⟨M0⟩ ⟨M1⟩ ⟨M2⟩ ⟨M3⟩ ⟨M4⟩ ⟨M5⟩ …
M0 Acc No No Acc Acc No …
M1 Acc Acc Acc Acc Acc Acc …
M2 Acc Acc Acc Acc Acc Acc …
M3 No Acc Acc No Acc Acc …
M4 Acc No Acc No Acc No …
M5 No No Acc Acc No No …
… … … … … … … …
No No No Acc No Acc …
Flip all "accept" to "no" and vice-versa


⟨M0⟩ ⟨M1⟩ ⟨M2⟩ ⟨M3⟩ ⟨M4⟩ ⟨M5⟩ …
M0 Acc No No Acc Acc No …
M1 Acc Acc Acc Acc Acc Acc …
M2 Acc Acc Acc Acc Acc Acc …
M3 No Acc Acc No Acc Acc …
M4 Acc No Acc No Acc No …
M5 No No Acc Acc No No …
… … … … … … … …
No TM has this behavior!
No No No Acc No Acc …



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

Diagonalization Revisited

- The **diagonalization language**, which we denote 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.

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

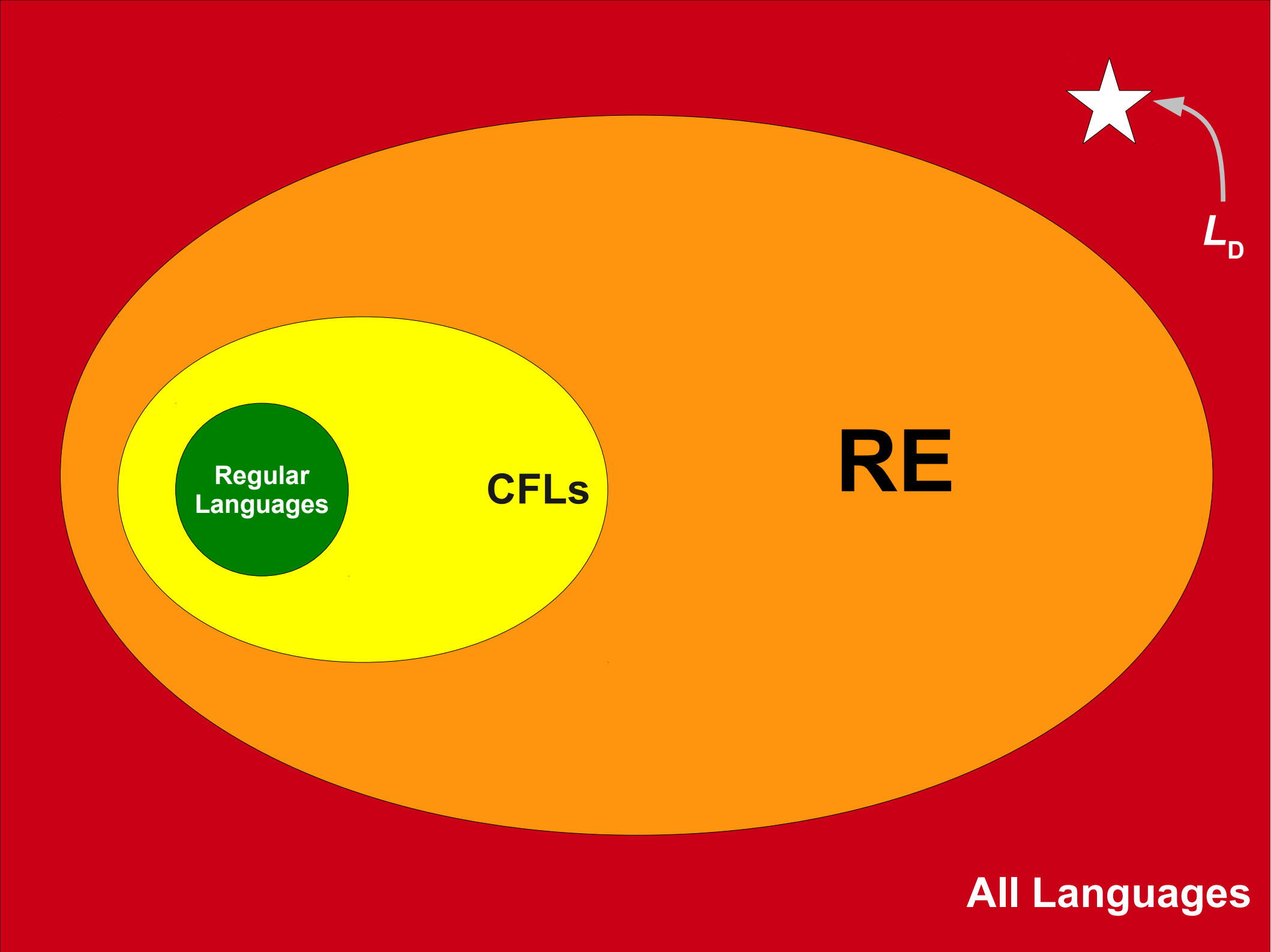
Theorem: $L_D \notin \mathbf{RE}$.

Proof: By contradiction; assume that $L_D \in \mathbf{RE}$. 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 see that $\langle R \rangle \in L_D$. Since $\langle R \rangle \notin \mathscr{L}(R)$ and $\langle R \rangle \in L_D$, we learn that $\mathscr{L}(R) \neq L_D$.

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 $\langle R \rangle \notin L_D$, we learn that $\mathscr{L}(R) \neq L_D$.

In both of these cases, we see that $\mathscr{L}(R) \neq L_D$, contradicting the fact that $\mathscr{L}(R) = L_D$. We have reached a contradiction, so our assumption must have been wrong. Thus $L_D \notin \mathbf{RE}$. ■


L_D
RE
CFLs
Regular Languages
All Languages

Time Out For Announcements!

Problem Set Seven

- Problem Set Seven goes out right now and is due next Monday.
  - Design some Turing machines!
  - Explore the properties of the **RE** languages!
  - Explore the borders of what's possible with computers!
- Two questions rely on material from Wednesday; they're marked as such. Sorry about that!

Problem Set Return

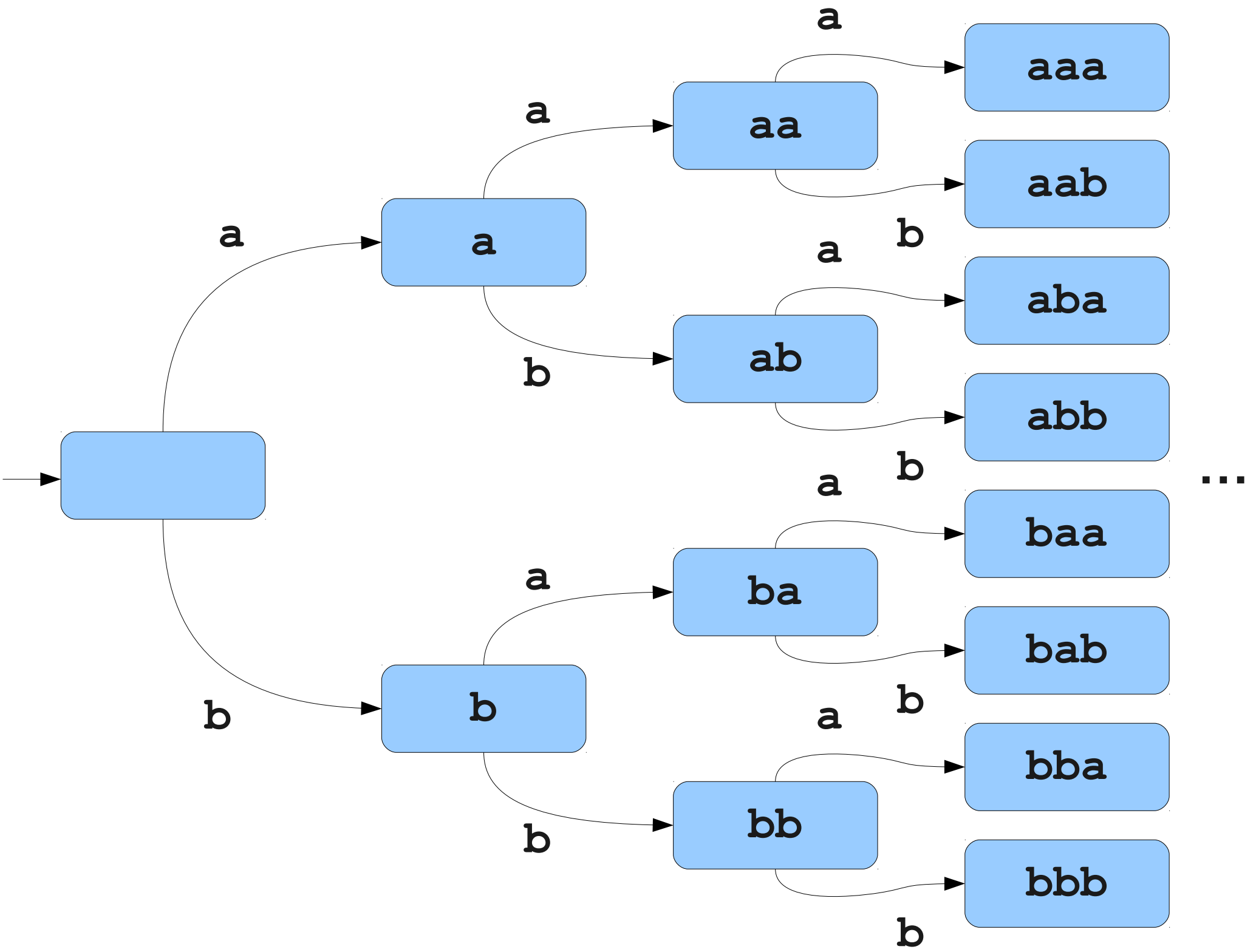
- Problem Set Four graded; will be returned at end of lecture today.
- Aiming to get Problem Set Five graded and returned by Wednesday.
- Sorry for the delays!
- **One unnamed problem set**: Contact us if you can't find your problem set.

Your Questions

“What were your favorite classes as an undergrad here at Stanford?”

“Are there any rules to determine if a language is context-free? Does it have to do with whether you can solve it using recursion/induction?”

“Turing machines are an expansion of FAs that use infinite memory. Couldn't we expand on FAs by giving them infinite states as well? Would that somehow expand the power of such a method of computation?”


a
b
a
b
aa
ab
a
b
aaa
aab
aba
abb
...
baa
bab
bba
bbb

Back to CS103!

Additional Unsolvable Problems

Finding Unsolvable Problems

- We can use the fact that $L_D \notin \mathbf{RE}$ to show that other languages are also not **RE**.
- General proof approach: to show that some language L is not **RE**, we will do the following:
  - Assume for the sake of contradiction that $L \in \mathbf{RE}$, meaning that there is some TM M for it.
  - Show that we can build a TM that uses M as a subroutine in order to recognize L_D.
  - Reach a contradiction, since no TM recognizes L_D.
  - Conclude, therefore, that $L \notin \mathbf{RE}$.

The Complement of A_{TM}

- Recall: the language A_{TM} is the language of the universal Turing machine U_{TM}:

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

- The complement of A_{TM} (denoted $\overline{A}_{TM}$) is the language of all strings not contained in A_{TM}.
- Questions:
  - What language is this?
  - Is this language **RE**?

A_{TM} and $\overline{A}_{TM}$

- The language A_{TM} is defined as

  $\{\langle M, w\rangle \mid M$ is a TM that accepts $w\}$

- Equivalently:

  $\{x \mid x = \langle M, w\rangle$ for some TM M and string w, and M accepts $w\}$

- Thus $\overline{A}_{TM}$ is

  $\{x \mid x \neq \langle M, w\rangle$ for any TM M and string w, or M is a TM that does not accept $w\}$

Cheating With Math

- As a mathematical simplification, we will assume the following:

Every string can be decoded into any collection of objects.

- Every string is an encoding of some TM M.
- Every string is an encoding of some TM M and string w.
- Can do this as follows:
  - If the string is a legal encoding, go with that encoding.
  - Otherwise, pretend the string decodes to some predetermined group of objects.

Cheating With Math

- Example: Every string will be a valid C++ program.
- If it's already a C++ program, just compile it.
- Otherwise, pretend it's this program:

```
int main() {
    return 0;
}
```

A_{TM} and $\overline{A}_{TM}$

- The language A_{TM} is defined as

  $\{\langle M, w\rangle \mid M$ is a TM that accepts $w\}$

- Thus $\overline{A}_{TM}$ is the language

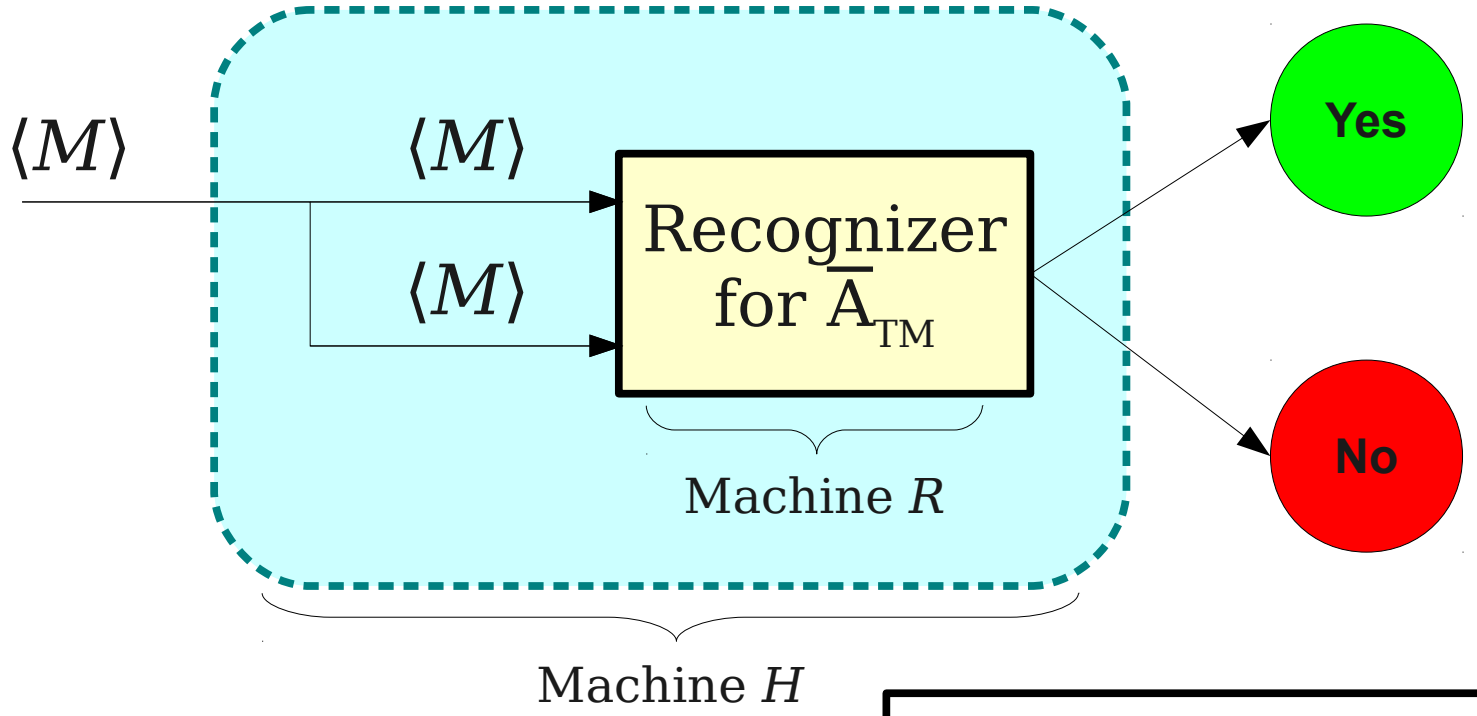
  $\{\langle M, w\rangle \mid M$ is a TM that doesn't accept $w\}$

$\overline{A}_{TM} \notin \mathbf{RE}$

- Although the language A_{TM} is in **RE** (since it's the language of U_{TM}), its complement $\overline{A}_{TM}$ is not in **RE**.
- We will prove this as follows:
  - Assume, for contradiction, that $\overline{A}_{TM} \in \mathbf{RE}$.
  - This means there is a TM R for $\overline{A}_{TM}$.
  - Using R as a subroutine, we will build a TM H that will recognize L_D.
  - This is impossible, since $L_D \notin \mathbf{RE}$.
  - Conclude, therefore, that $\overline{A}_{TM} \notin \mathbf{RE}$.

Comparing L_D and $\overline{A}_{TM}$

- The languages L_D and $\overline{A}_{TM}$ are closely related:
  - L_D: Does M not accept $\langle M \rangle$?
  - $\overline{A}_{TM}$: Does M not accept string w?
- Given this connection, we will show how to turn a hypothetical recognizer for $\overline{A}_{TM}$ into a hypothetical recognizer for L_D.



H = "On input $\langle M \rangle$:

- Construct the string $\langle M, \langle M \rangle \rangle$.
- Run R on $\langle M, \langle M \rangle \rangle$.
- If R accepts $\langle M, \langle M \rangle \rangle$, then H accepts $\langle M \rangle$.
- If R rejects $\langle M, \langle M \rangle \rangle$, then H rejects $\langle M \rangle$."

What happens if...

M does not accept $\langle M \rangle$?
Accept

M accepts $\langle M \rangle$?
Reject or **Loop**

H is a TM for L_D!

Theorem: $\overline{A}_{TM} \notin \mathbf{RE}$.

Proof: By contradiction; assume that $\overline{A}_{TM} \in \mathbf{RE}$. Then there must be a recognizer for $\overline{A}_{TM}$; call it R.

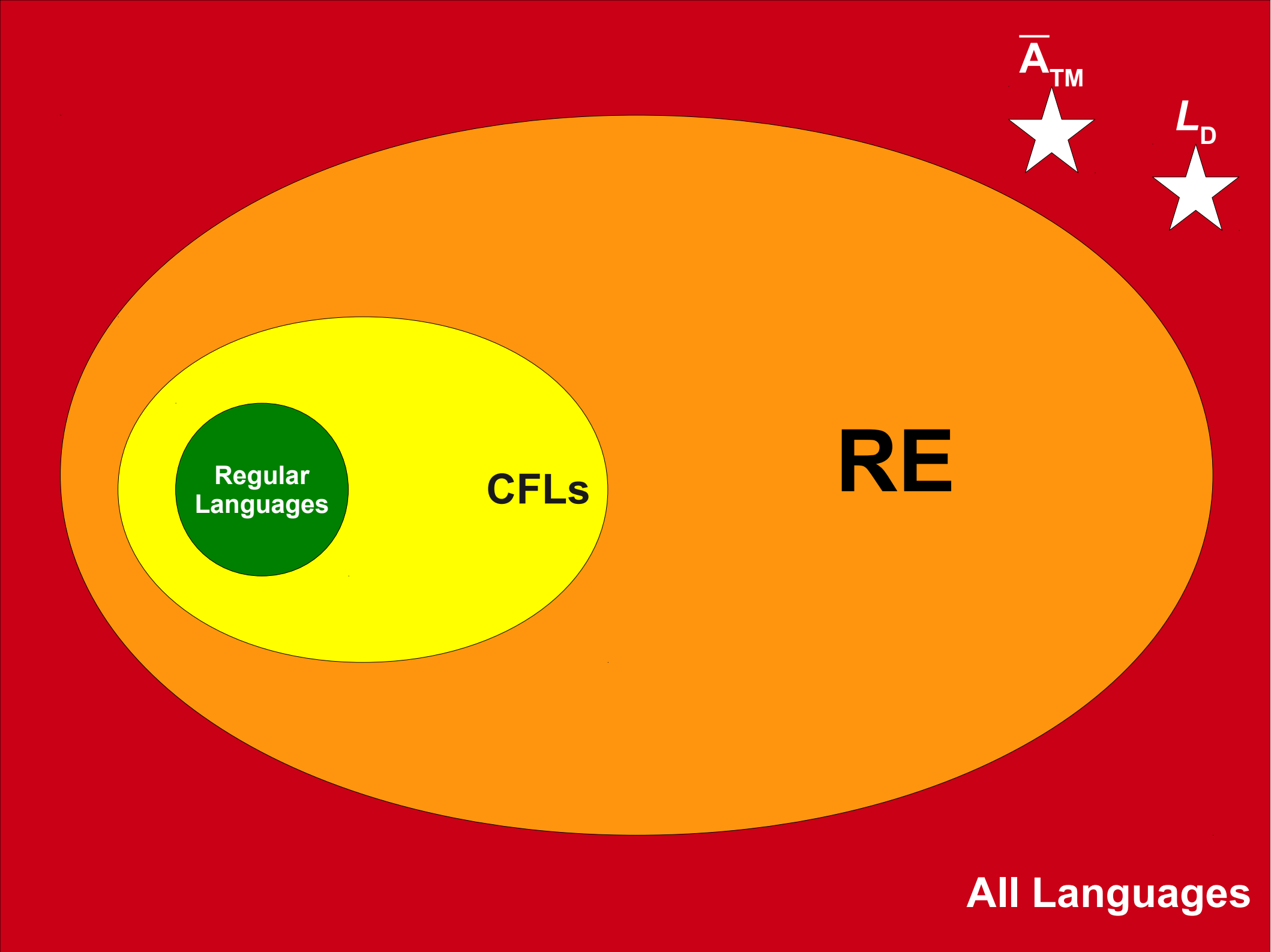
Consider the TM H defined below:

> H = "On input $\langle M \rangle$, where M is a TM:
> Construct the string $\langle M, \langle M \rangle \rangle$.
> Run R on $\langle M, \langle M \rangle \rangle$.
> If R accepts $\langle M, \langle M \rangle \rangle$, H accepts $\langle M \rangle$.
> If R rejects $\langle M, \langle M \rangle \rangle$, H rejects $\langle M \rangle$."

We claim that $\mathscr{L}(H) = L_D$. We will prove this by showing that $\langle M \rangle \in L_D$ iff H accepts $\langle M \rangle$.

By construction we have that H accepts $\langle M \rangle$ iff R accepts $\langle M, \langle M \rangle \rangle$. Since R is a recognizer for $\overline{A}_{TM}$, R accepts $\langle M, \langle M \rangle \rangle$ iff M does not accept $\langle M \rangle$. Finally, note that M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_D$. Therefore, we have H accepts $\langle M \rangle$ iff $\langle M \rangle \in L_D$, so $\mathscr{L}(H) = L_D$. But this is impossible, since $L_D \notin \mathbf{RE}$.

We have reached a contradiction, so our assumption must have been incorrect. Thus $\overline{A}_{TM} \notin \mathbf{RE}$, as required. ■


$\overline{A}_{TM}$
L_D
Regular Languages
CFLs
RE
All Languages

Why All This Matters

- We *finally* have found concrete examples of unsolvable problems!
- We are starting to see a line of reasoning we can use to find unsolvable problems:
  - Start with a known unsolvable problem.
  - Try to show that the unsolvability of that problem entails the unsolvability of other problems.
- We will see this used extensively in the upcoming weeks.

Next Time

- **Decidability**
  - What is an algorithm?
- **Undecidability**
  - What problems can't be solved by algorithms?
- **co-Recognizability**
  - How do we resolve the asymmetry of **RE**?