The Big Picture

Announcements

- Problem Set 8 due right now. We'll release solutions right after lecture.
  - ***Congratulations – you're done with CS103 problem sets!***
- **Please evaluate this course on Axess!** Your feedback really does make a difference.
- There's a fun and completely optional handout "Timeline of CS103 Results" up on the CS103 website. It details the history of all the results from this course.

Final Exam Logistics

- Final exam is Monday, June 8 from 8:30AM – 11:30AM.
- Locations divvied up by last (family) name:
  - Aba – Leo: Go to **420-040** (here!)
  - Leu – Zoc: Go to **Hewlett 200**
- Practice final exam is Thursday (tomorrow) from 1PM – 4PM, location TBA.
- EPP7 solutions released, EPP8 goes out today.

The Big Picture

The Big Picture

Cantor's Theorem: $|S| < |\wp(S)|$

Corollary: Unsolvable problems exist.

What problems can be solved by computers?

First, we need to learn how to prove results with certainty.

Otherwise, how can we know for sure that we're right about anything?

Now, we need to learn how to prove things about processes that proceed step-by-step.

So let's learn induction.

We also should be sure we have some rules about reasoning itself.

Let's add some logic into the mix.

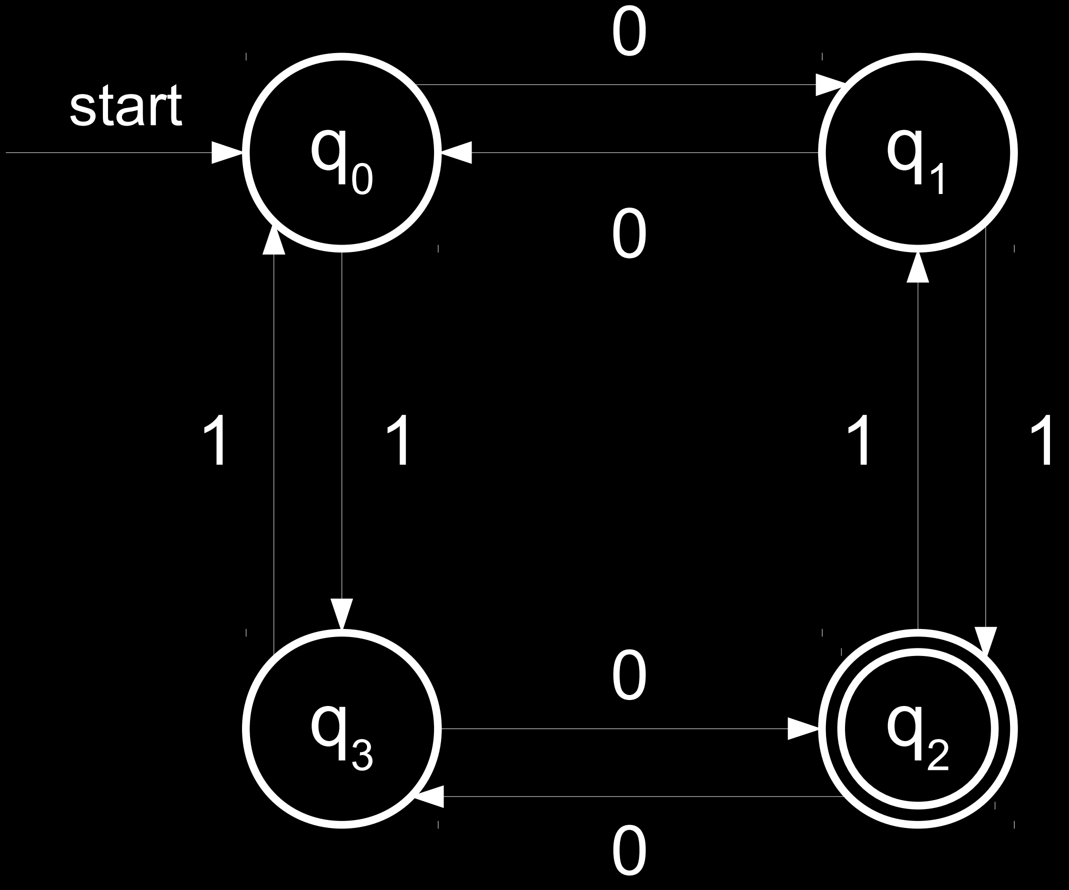
Finally, let's study a few common discrete structures.

That way, we know how to model connected structures and relationships.

Okay! So now we're ready to go!

What problems are unsolvable?

Well, first we need a
definition of a computer!

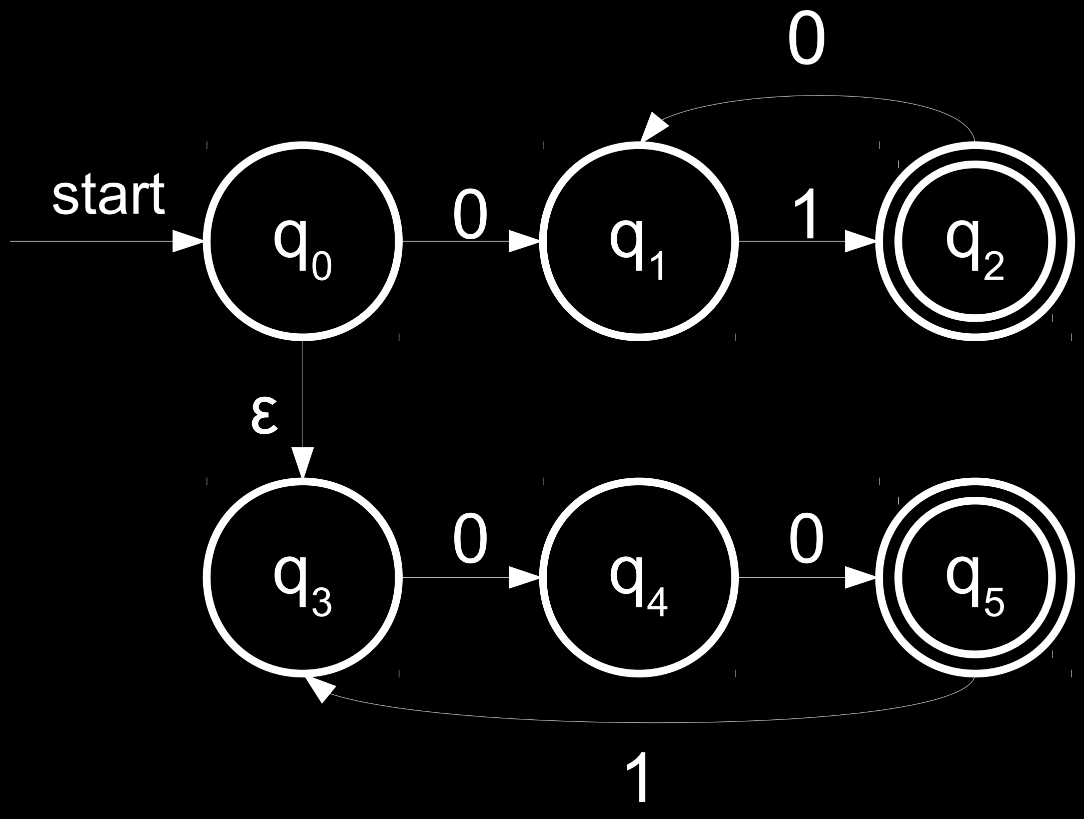

start
q0
q1
q2
q3
0
0
1
1
1
1
0
0

Cool! Now we have a model of a computer!

We're not quite sure what we can solve at this point, but that's okay for now.

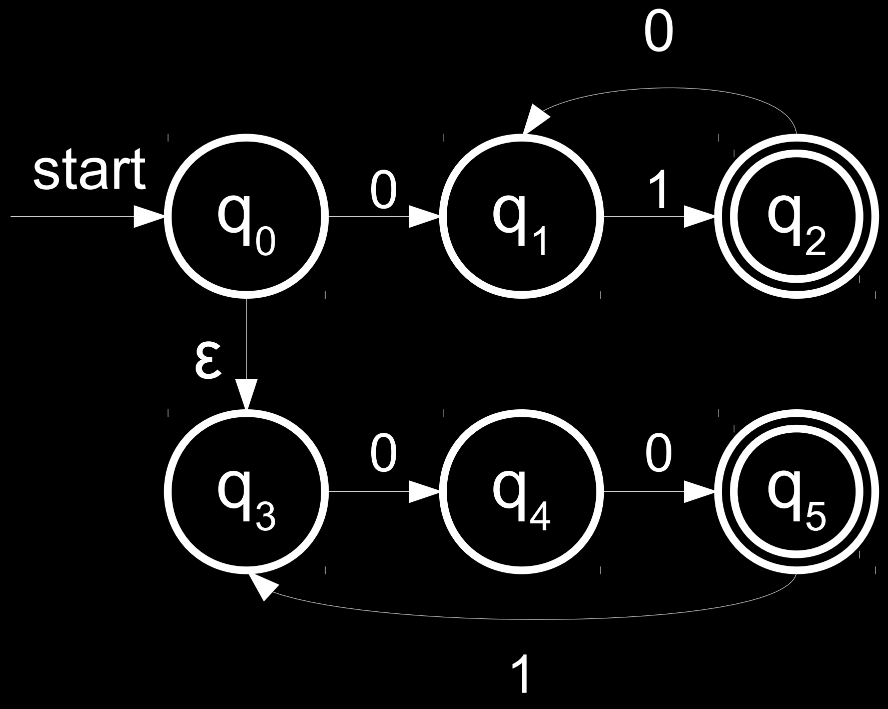
Let's call the languages we can capture this way the ***regular languages***.

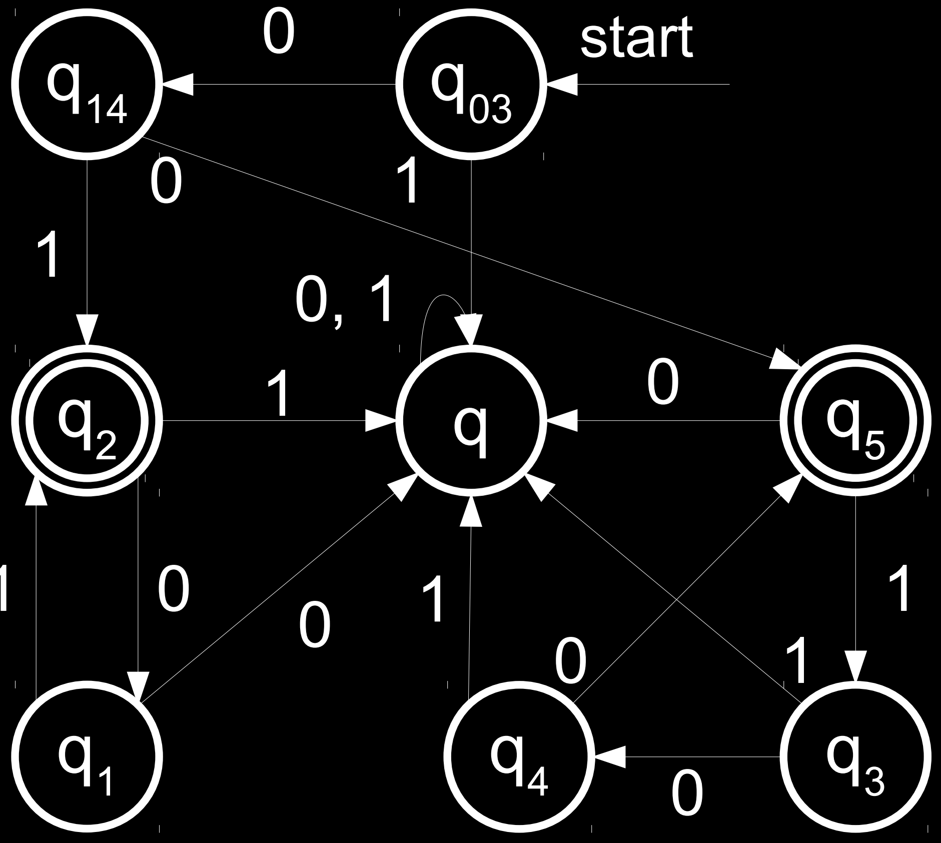
I wonder what other
machines we can make?


start
q0
q1
q2
q3
q4
q5
0
1
0
ε
0
0
1

Wow! Those new machines are
way cooler than our old ones!

I wonder if they're more powerful?

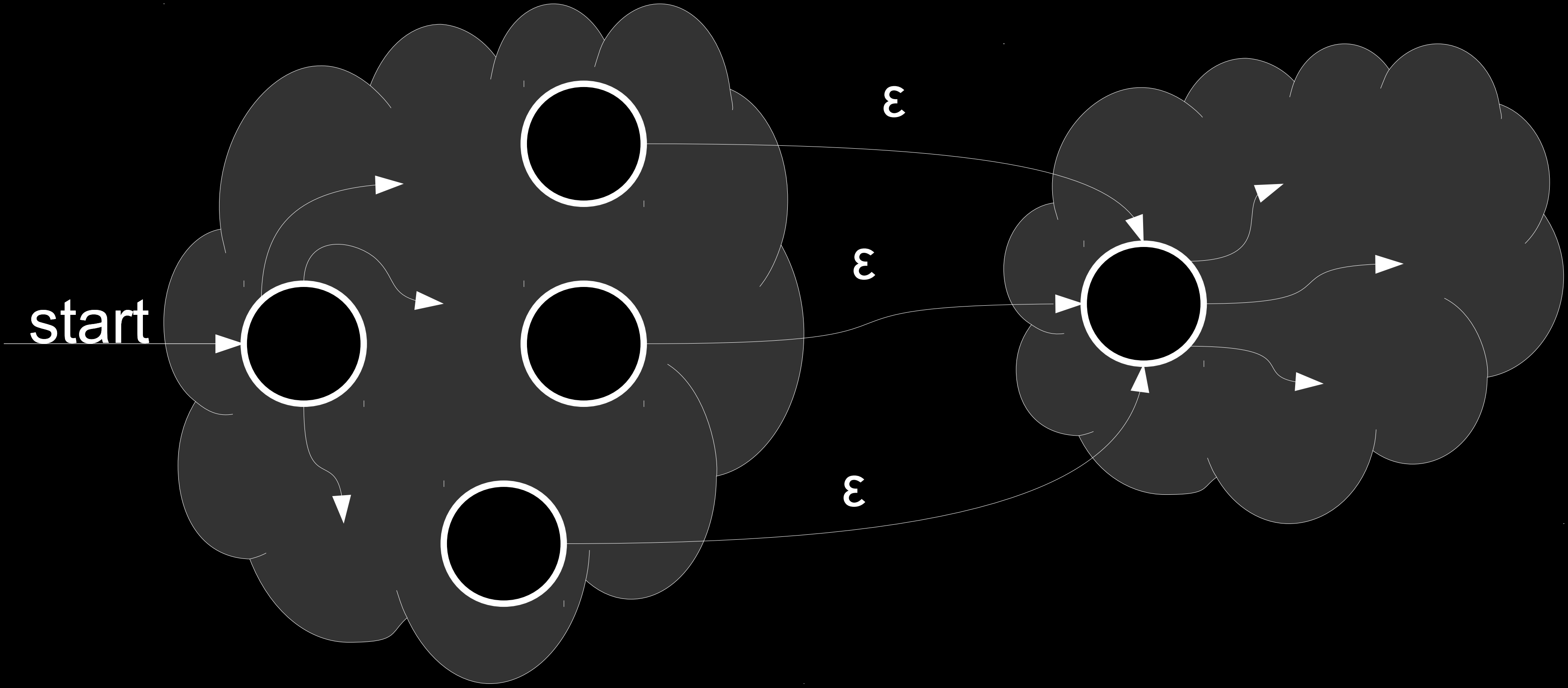

start
q0
q1
q2
q3
q4
q5
0
1
0
ε
0
0
1


start
q03
q14
q2
q
q5
q1
q4
q3
0
1
0
1
0, 1
1
0
1
0
0
1
0
1
1
0

Wow! I guess not. That's surprising!

So now we have a new way of modeling computers with finite memory!

I wonder how we can combine these machines together?


start
ε
ε
ε

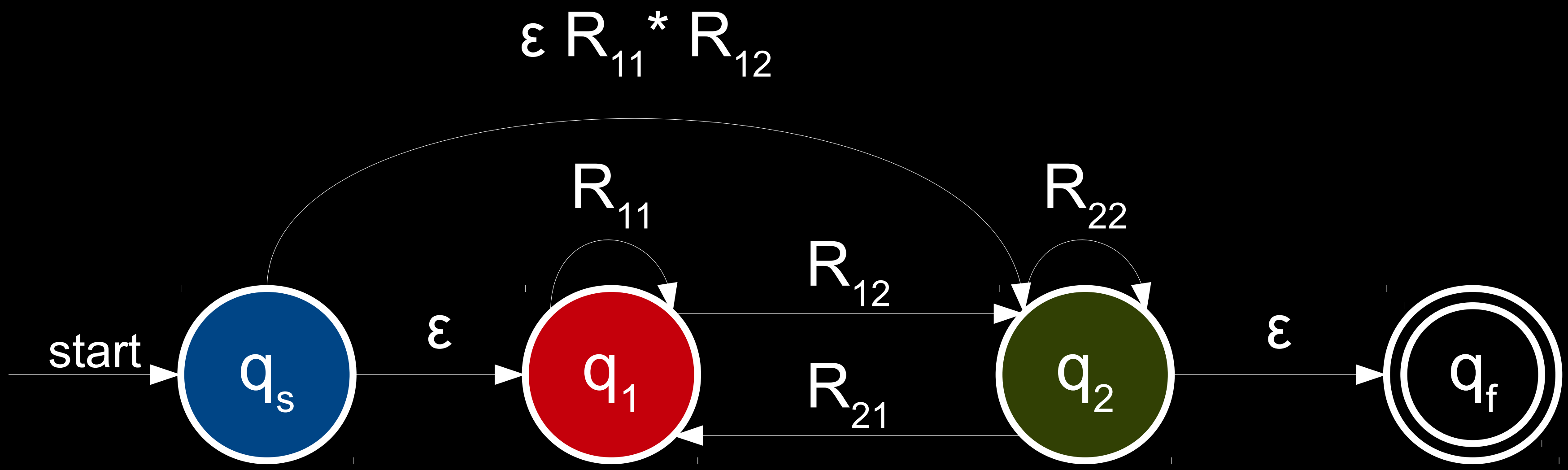
Cool! Since we can glue machines together, we can glue languages together as well.

How are we going to do that?

$a^+(.a^+)^*@a^+(.a^+)^+$

Wow! We've got a new way of describing languages.

So what sorts of languages
can we describe this way?



$\varepsilon\ R_{11}{}^{*}\ R_{12}$
R_{11}
R_{22}
R_{12}
start
q_s
ε
q_1
R_{21}
q_2
ε
q_f

Awesome! We got back the exact same class of languages.

It seems like all our models give us the same power! Did we get every language?

$$xw \in L$$
$$yw \notin L$$

Wow, I guess not.

But we did learn something cool:

We have just explored what problems can be solved with finite memory.

So what else is out there?

Can we describe languages another way?

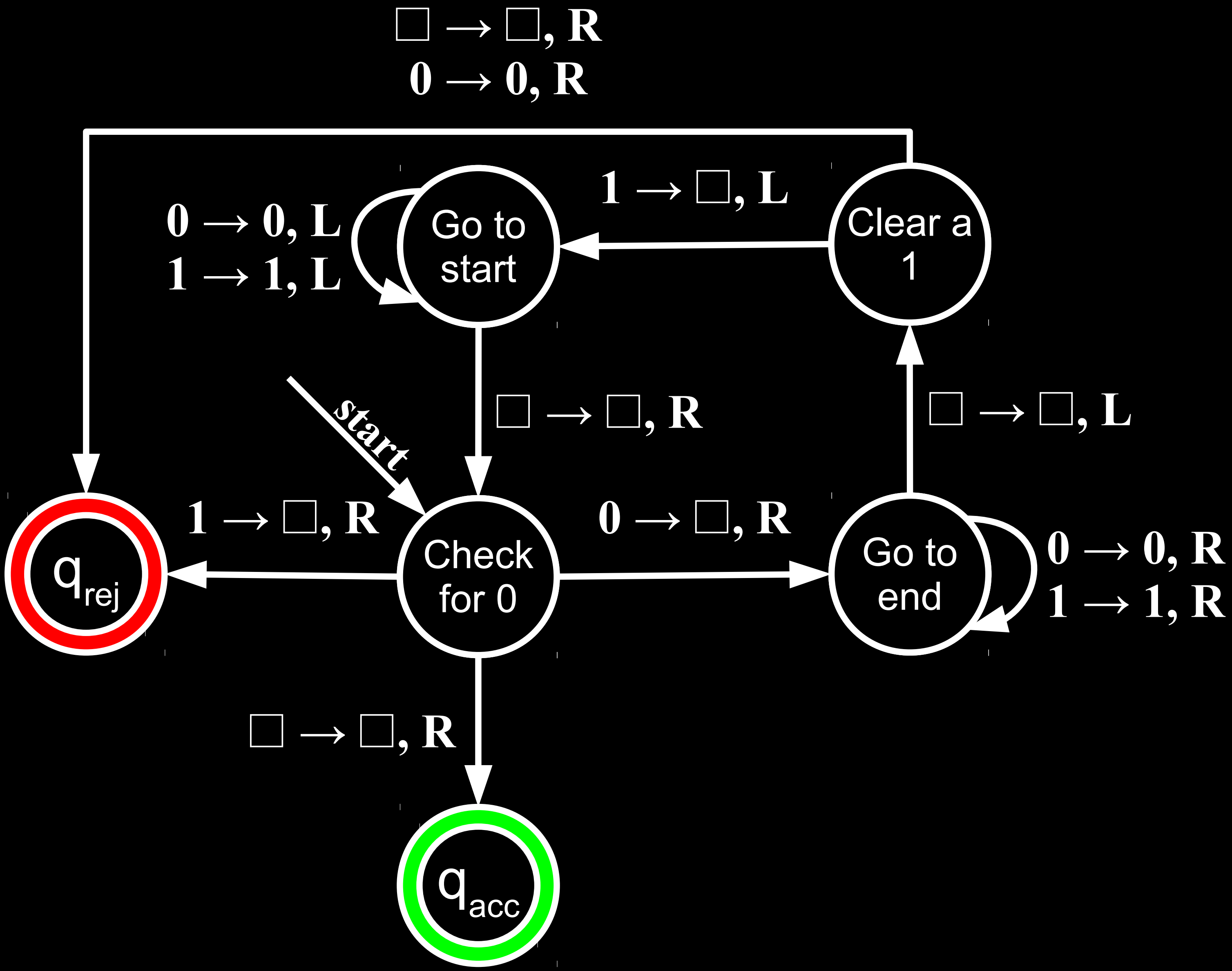
$$
\begin{aligned}
S &\rightarrow \texttt{a}X \\
X &\rightarrow \texttt{b} \mid C \\
C &\rightarrow C\texttt{c} \mid \varepsilon
\end{aligned}
$$

Awesome!

So, did we get every language yet?

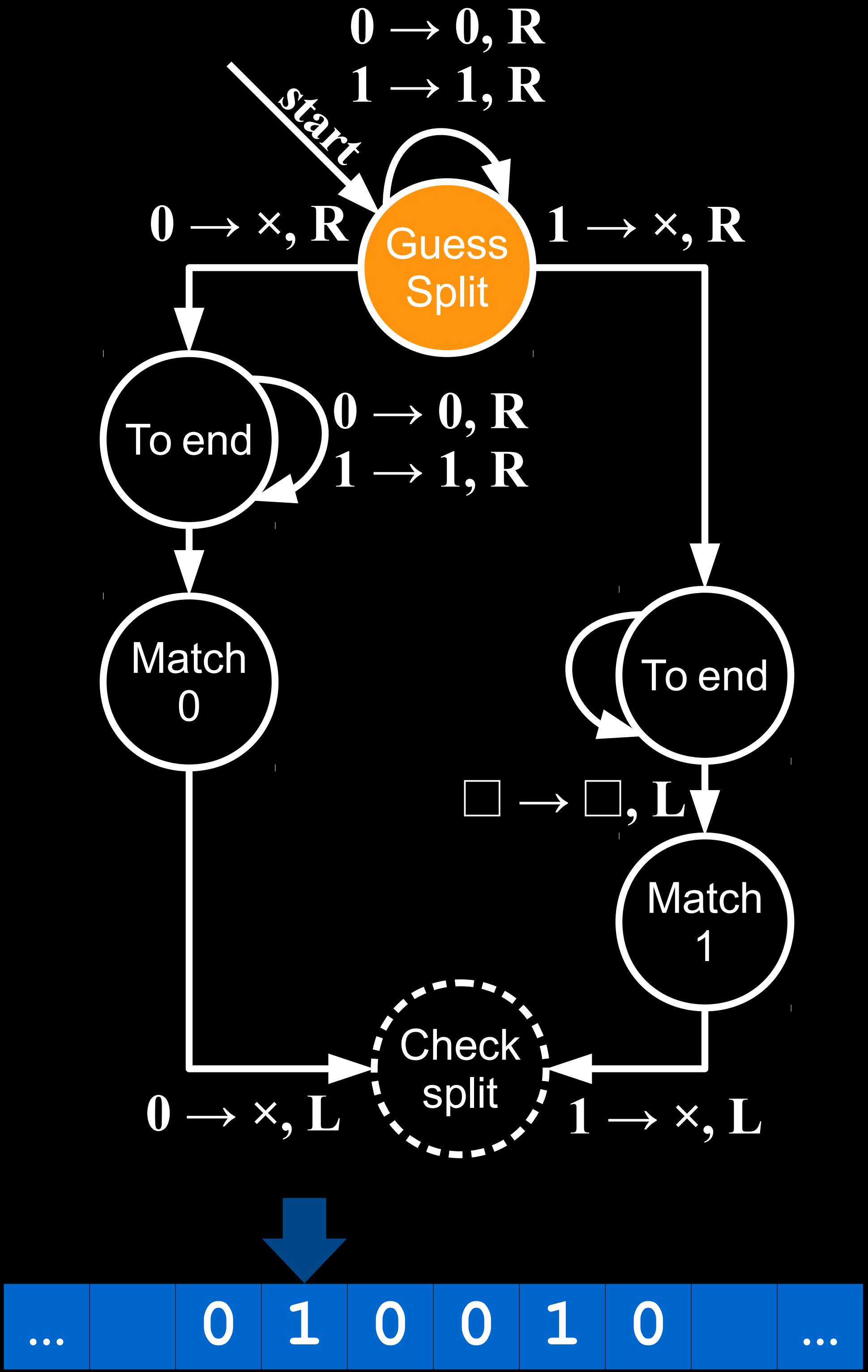
Hmmm… guess not.

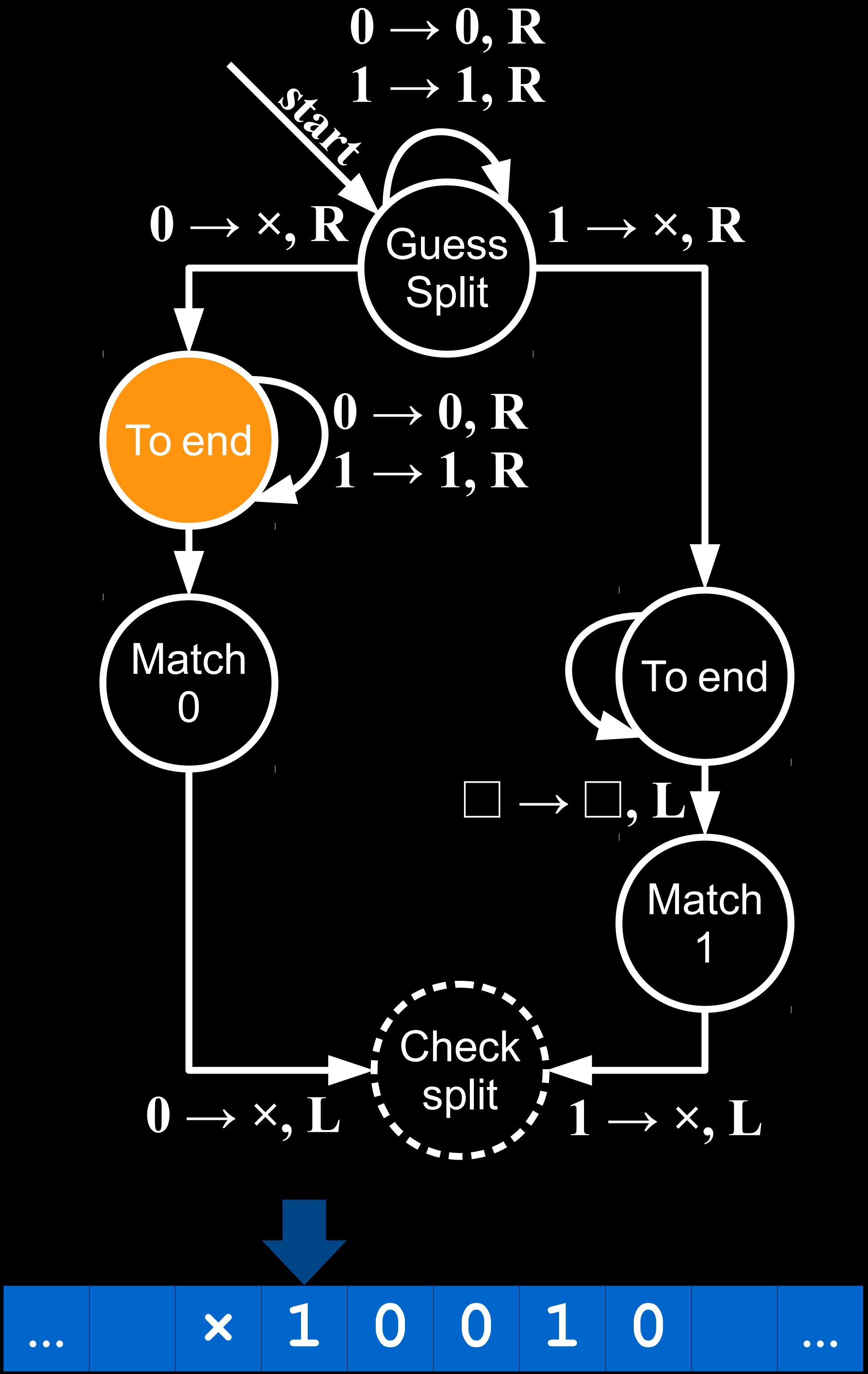
So what if we make our memory a little better?



□ → □, R
0 → 0, R
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
start
□ → □, R
□ → □, L
1 → □, R
q_rej
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
□ → □, R
q_acc

Cool! Can we make these more powerful?


start
0 → 0, R
1 → 1, R
Guess Split
0 → ×, R
1 → ×, R
To end
0 → 0, R
1 → 1, R
Match 0
To end
□ → □, L
Match 1
Check split
0 → ×, L
1 → ×, L
… 0 1 0 0 1 0 …


start
0 → 0, R
1 → 1, R
Guess Split
0 → ×, R
1 → ×, R
To end
0 → 0, R
1 → 1, R
Match 0
To end
□ → □, L
Match 1
Check split
0 → ×, L
1 → ×, L
… × 1 0 0 1 0 …

V = "On input $\langle w, T\rangle$, where T is a sequence of transitions:

- Run N on w, following transitions in the order specified in T.
- If any of the transitions in T are invalid or can't be followed, reject.
- If after following the transitions N accepts w, accept; otherwise reject.

Wow! Looks like we can't get any more powerful.

(The ***Church-Turing thesis*** says that this is not a coincidence!)

So why is that?

```
U_TM = "On input ⟨M, w⟩, where M is a TM and w ∈ Σ*:
          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.
          }"
```

Wow! Our machines can simulate one another!

This is a theoretical justification for why all these models are equivalent to one another.

So… can we solve everything yet?

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

const vector<string> kToPrint = {
  /* … */
};

void printProgramInQuotes() {
  for (string line: kToPrint) {
    cout << "  \"";
    for (char ch: line) {
      if (ch == '\"') cout << "\\\"";
      else if (ch == '\\') cout << "\\\\";
      else cout << ch;
    }
    cout << "\"," << endl;
  }
}

int main() {
  for (string line: kToPrint) {
    if (line == "@") printProgramInQuotes();
    else cout << line << endl;
  }
}
```

Weird! Programs can gain access to their own source code!

Why does that matter?

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

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

Crazy! The power of self-reference immediately limits what TMs can do!

What if we think about solving problems in a different way?

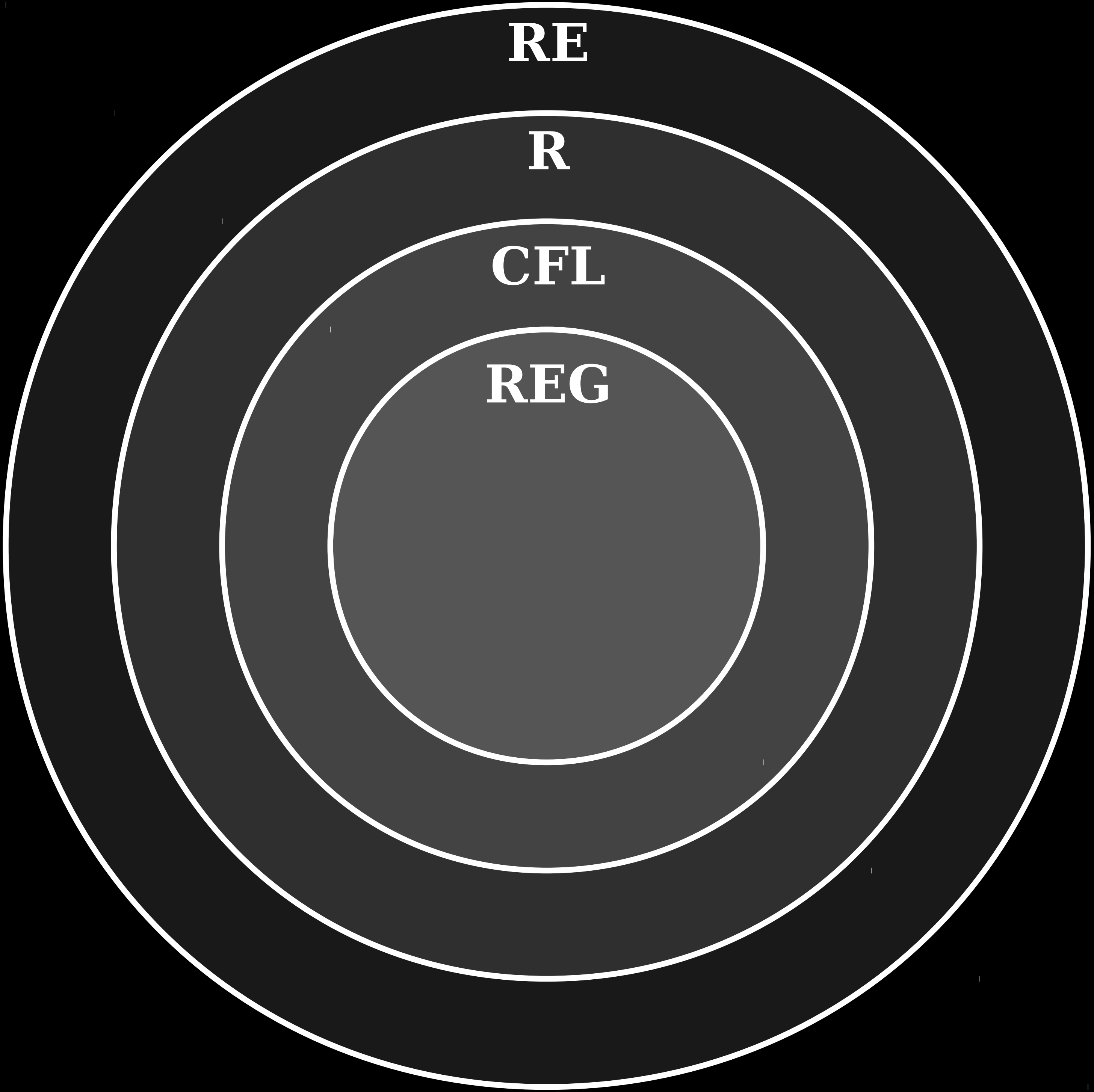
```
int main() {
   string me = mySource();
   string input = getInput();

   for (each string c) {
      if (imConvincedWillLoop(me, input, c) {
         accept();
      }
   }
}
```

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

Oh great. Some problems are impossible to solve.

But look what we learned along the way!


RE
R
CFL
REG

Wow. That's pretty deep.

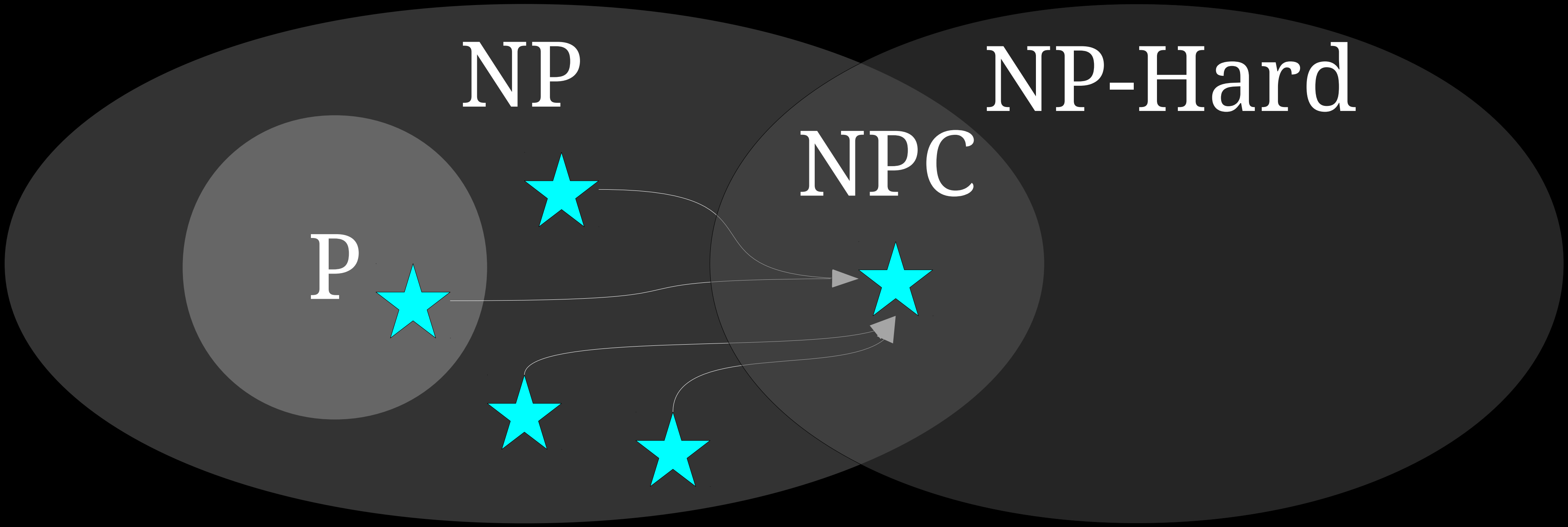
So… what can we do efficiently?

NP

So... how are you two related again?

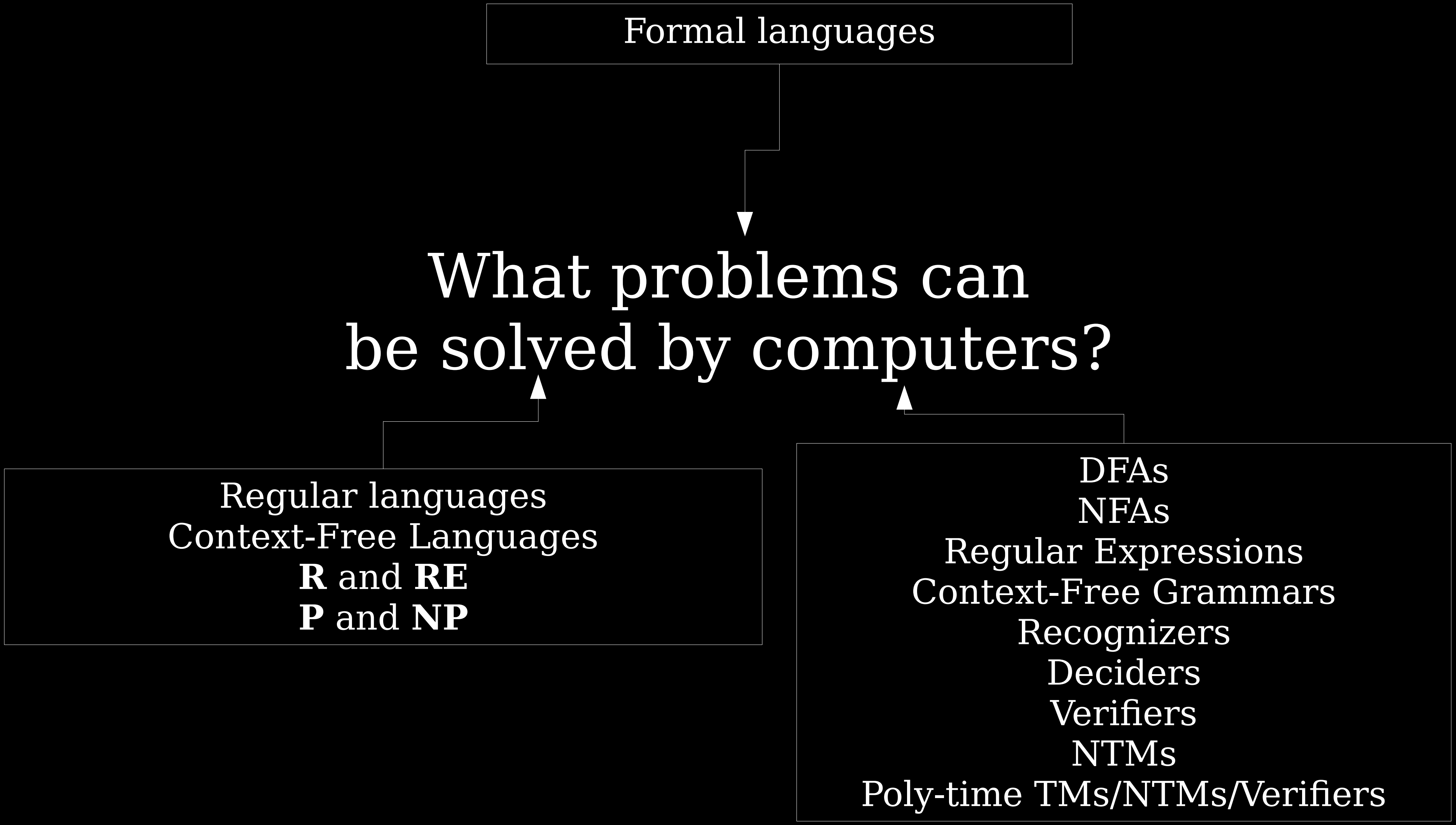
No clue.

But what do we know about them?


NP
NP-Hard
NPC
P

Congratulations on making it this far!

What's next in CS theory?


Formal languages
What problems can be solved by computers?
Regular languages
Context-Free Languages
R and **RE**
P and **NP**
DFAs
NFAs
Regular Expressions
Context-Free Grammars
Recognizers
Deciders
Verifiers
NTMs
Poly-time TMs/NTMs/Verifiers

Function problems (CS254)
Counting problems (CS254)

What problems can be solved by computers?

Interactive proof systems (CS254)
Approximation algorithms (CS261/361)
Average-case efficiency (CS264)
Randomized algorithms (CS265/254)
Parameterized complexity (CS266)
Communication complexity (CS369E)

Oracle machines (CS154)
Space-Bounded TMs (CS154/254)
Machines with Advice (CS254/354)
Streaming algorithms (CS263)
μ-Recursive functions (CS258)
Quantum computers (CS259Q)
Circuit complexity (CS354)

How do we actually get the computer to effectively solve problems?

DFA design intuitions
Guess-and-check
Massive parallelism
Myhill-Nerode lower bounds
Verification
Polynomial-time reductions

How do we actually get the computer to effectively solve problems?

Algorithm design (CS161)
Efficient data structures (CS166)
Modern algorithmic techniques (CS168)
Approximation algorithms (CS261)
Average-case efficient algorithms (CS264)
Randomized algorithms (CS265)
Parameterized algorithms (CS266)
Geometric algorithms (CS268)
Game-theoretic algorithms (CS364A/B)

Where does CS theory meet CS practice?

Finite state machines
Regular expressions
CFGs and programming languages
Password-checking
Autograding
“This program is not responding”
Polynomial-time reducibility
NP-hardness and **NP**-completeness

Where does CS theory meet CS practice?

Compilers (CS143)
Computational logic (CS157)
Program optimization (CS243)
Data mining (CS246)
Cryptography (CS255)
Programming languages (CS258)
Network protocol analysis (CS259)
Techniques in big data (CS263)
Graph algorithms (CS267)
Computational geometry (CS268)
Algorithmic game theory (CS364)

A Whole World of Theory Awaits!

What's being done here at Stanford?

Hardness results for easy problems
(Virginia Williams)

Algorithms ∩ Game theory
(Tim Roughgarden)

Learning patterns in randomness
(Greg Valiant)

Optimizing programs… randomly

(Alex Aiken)

Computing on encrypted data
(Dan Boneh)

Interpreting structure from shape
(Leonidas Guibas)

Lower bounds from upper bounds

(Ryan Williams)

So many options – what to do next?

Interested in trying out CS?

Continue on to CS109!

Really enjoyed this class?

Give CS154 a try!

Want to see this material come to life?

Check out CS143!

Want to just go write code?

Take CS107!

Keep on exploring! There's so much more to learn!

A Final “Your Questions”

"All our proofs that prove undecidability/unrecognizability so far have relied on self-referential Turing machines. Could it be the case that, say, the halting problem restricted to non-self-referential machines is decidable? Is that even provable?"

Self-reference is sneaky and inherent to computation itself. It's impossible to eliminate it without destroying the ability for the computer to solve interesting problems.

“About equivalent TMs, you said something like 'This problem is unrecognizable, which is why we need section leaders to grade your code!' What makes section leaders inherently different than computers?”

We ultimately want to solve problems for a reason. Putting humans in the loop lets us make judgment calls and factor in concerns that we just can't program into the machine.

“Please tell us a CS joke.”

Why didn't the brakes on the programmer's Segway work?

“What's one piece of advice you would give someone at Stanford?”

Stay happy.
Stay healthy.

“Thanks for answering all our questions Keith! Now, what is one question you would ask us?”

It's a multipart question. I'm going to save it for the end of today.

Anything else?

CS theory is all about asking what's possible in computer science.

There are more problems to solve than there are programs capable of solving them.

There is so much more to explore and so many big questions to ask – ***many of which haven't been asked yet!***

What We've Covered

- Sets
- Proof Techniques
- Induction
- Graphs
- Logic
- Pigeonhole Principle
- Functions
- Relations
- DFAs
- NFAs
- Regular Expressions
- Closure Properties
- Nonregular Languages
- CFGs
- Turing Machines
- **R** and **RE**
- The Recursion Theorem
- NTMs and Verifiers
- Unsolvable Problems
- Reductions
- Time Complexity
- **P**
- **NP**
- **NP**-Completeness

Final Thoughts

You now know what problems we can solve, what problems we can't solve, and what problems we believe we can't solve efficiently.

My questions to you:

What problems will you ***choose*** to solve?
Why do those problems matter to you?
And how are you going to solve them?