The Big Picture

Announcements

- Problem Set 9 due right now.
  - Using a 72-hour extension, due Monday at 2:15PM.
- Problem Set 8 graded, should be returned at the end of lecture.
- Final Friday Four Square of the quarter!
  - Today at 4:15PM, Outside Gates
- Final exam review sessions this weekend.
  - Saturday and Sunday at 2PM in Gates 104
- **Please evaluate this course on Axess! Your feedback really does make a difference!**

The Big Picture

The Big Picture

Imagine what it must have been like to discover all of the results in this class.

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

Corollary: Unsolvable problems exist.

What problems are unsolvable?

First, we need to learn
how to prove things.

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.

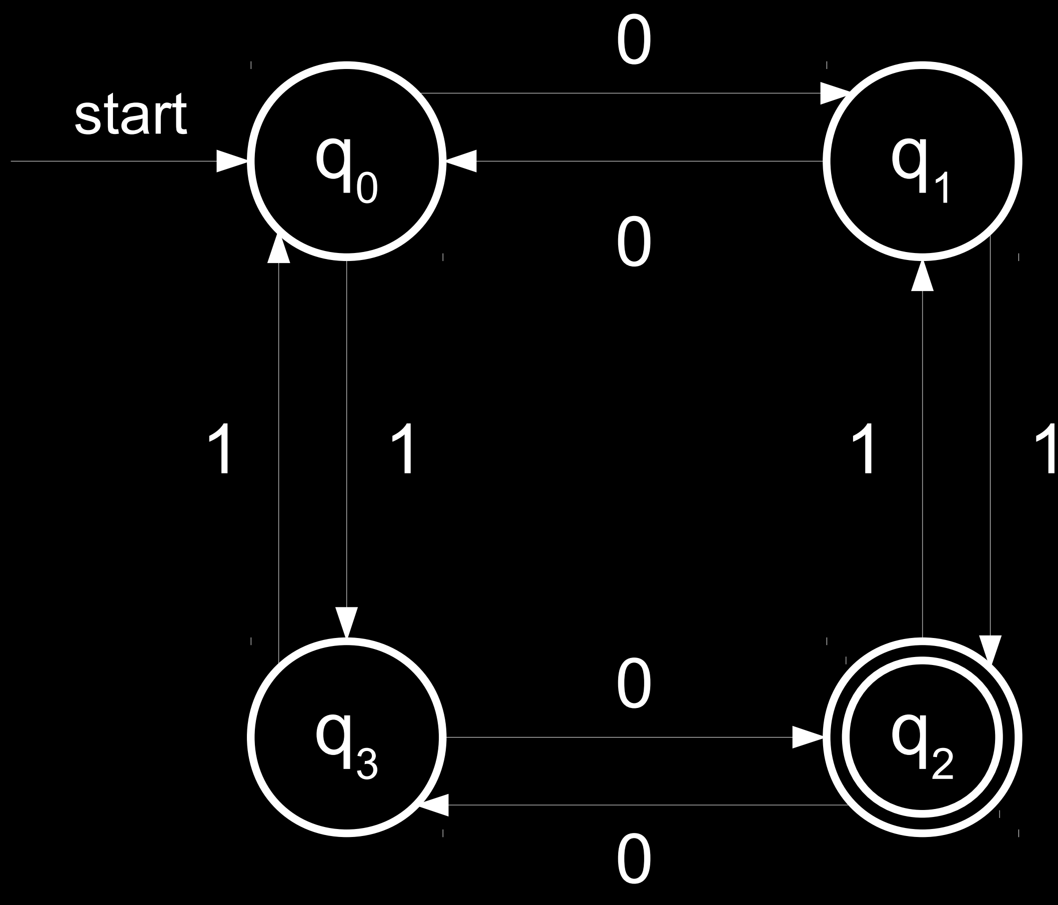
We also should be sure we have some rules about how reasoning works.

Let's add some logic into the mix.

Okay! So now we're ready to go!

What problems are unsolvable?

Well, first we need a
definition of a computer!

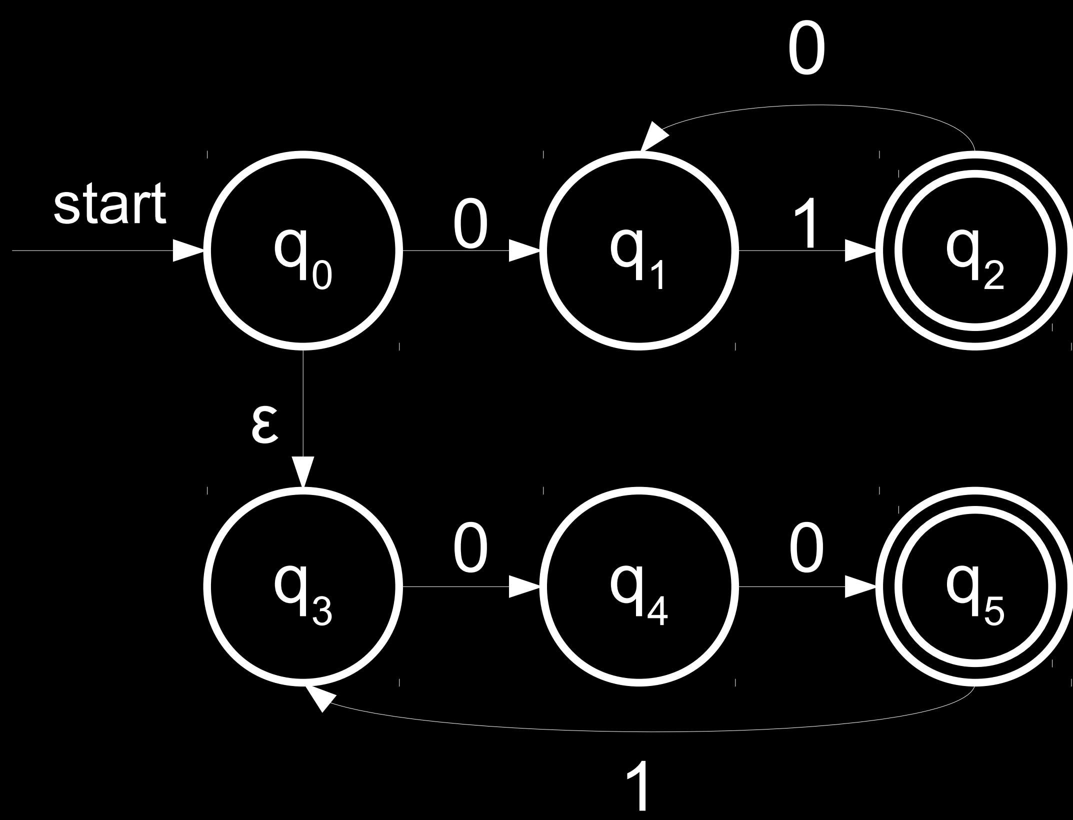

start
q_0
q_1
q_2
q_3
0
0
1
1
1
1
0
0

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**.

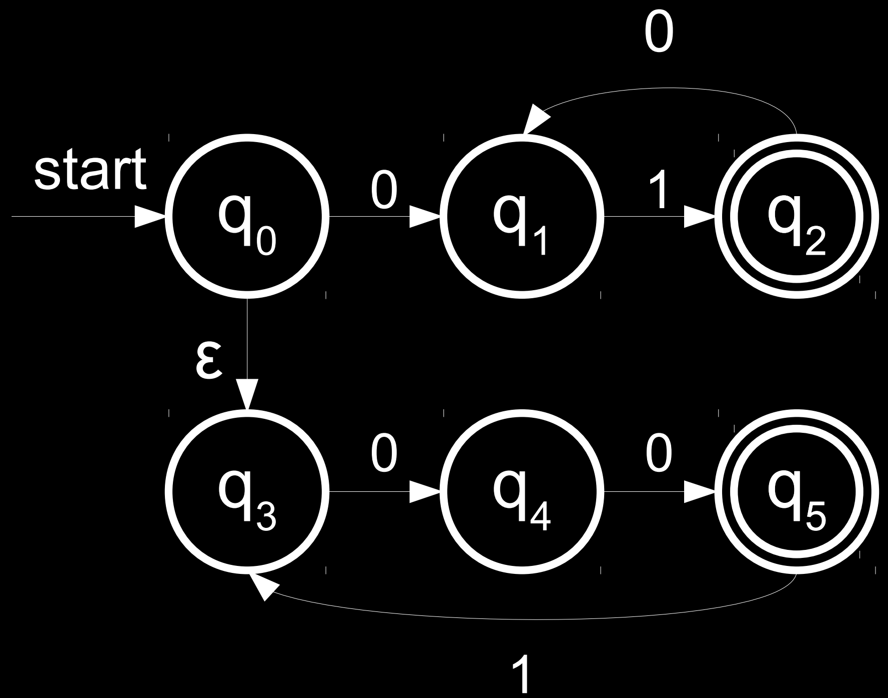
I wonder what other machines we can make?



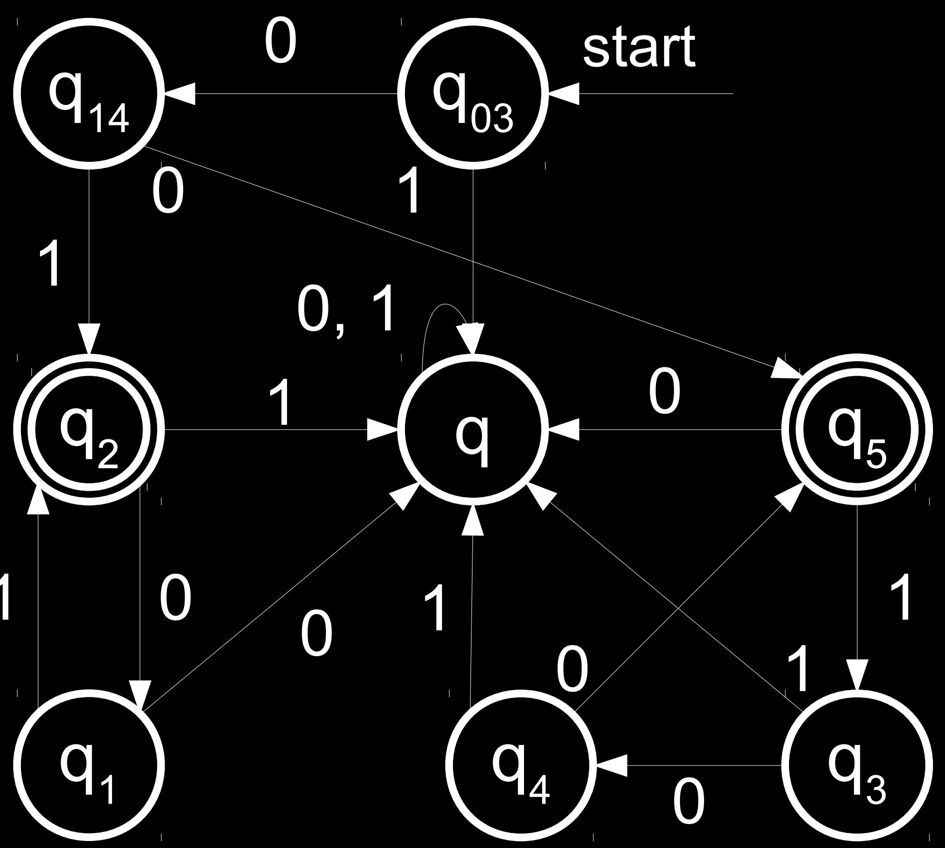
0
start
q0
0
q1
1
q2
ε
q3
0
q4
0
q5
1

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

I wonder if they're more powerful?



start
q_0
q_1
q_2
q_3
q_4
q_5
0
1
0
ε
0
0
1

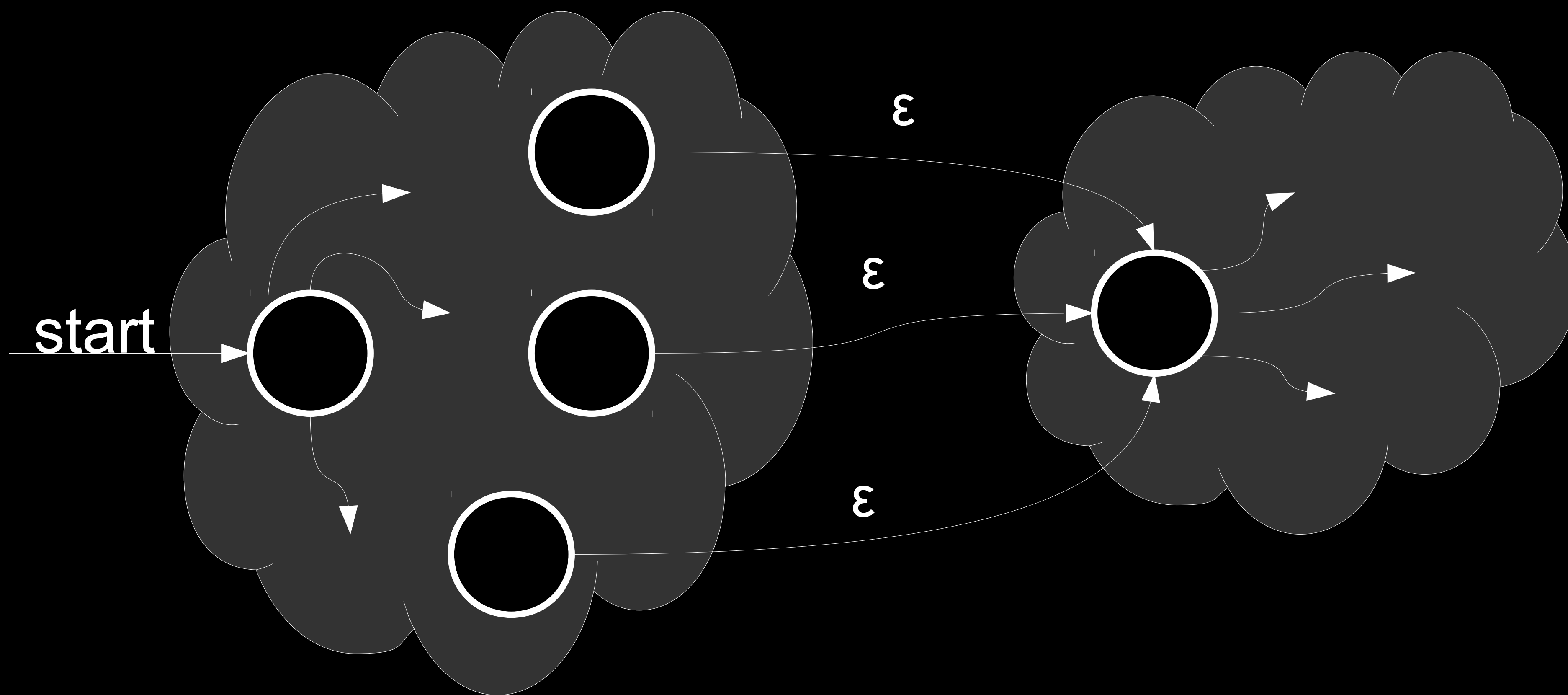


start
q_{03}
q_{14}
q_2
q
q_5
q_1
q_4
q_3
0, 1
0
1
0
1
1
0
1
0
1
0
1
0
1
0

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

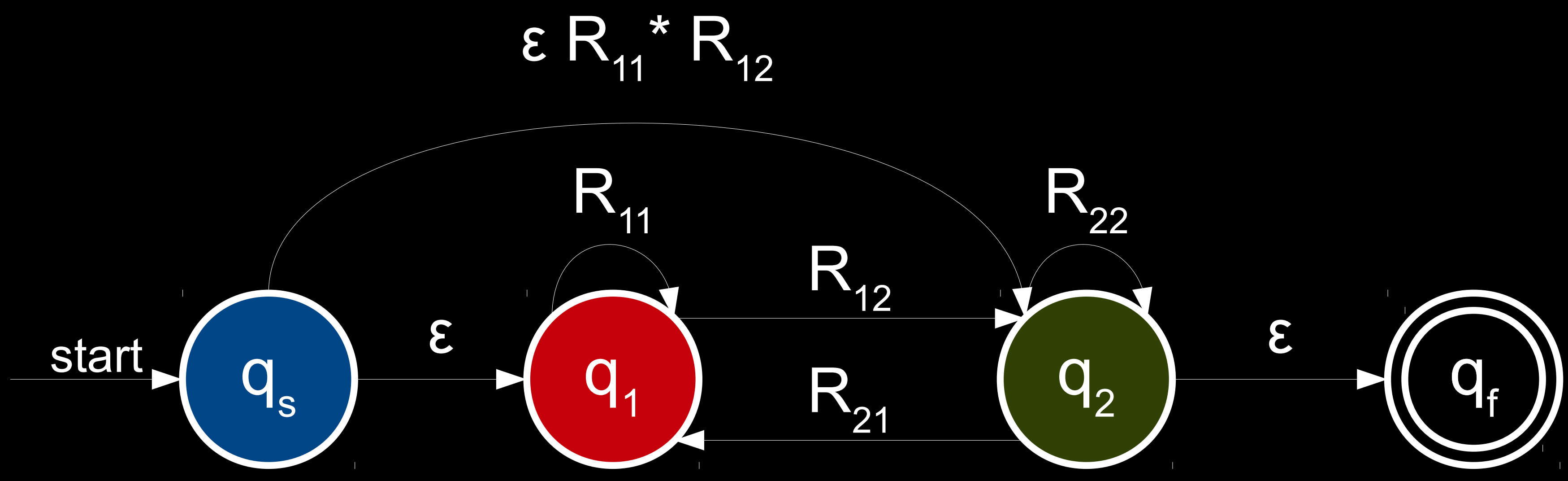
Cool! Since we can glue machines together, we can glue languages together as well.

How are we going to do that?

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

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

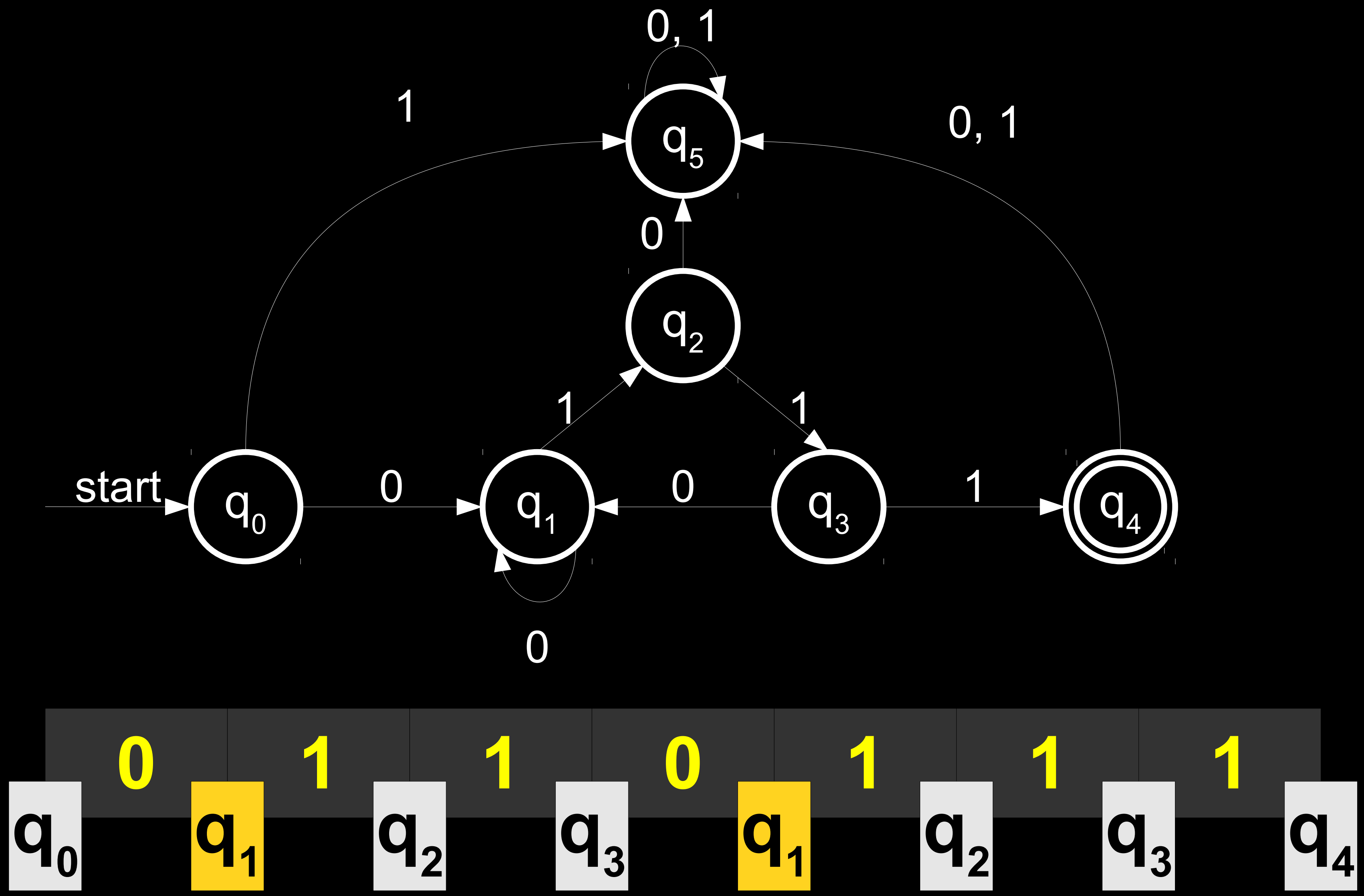
So what sorts of languages can we describe?



ε 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?



0, 1
1
0, 1
0
1
1
start
0
0
1
0
0 1 1 0 1 1 1
q_0 q_1 q_2 q_3 q_1 q_2 q_3 q_4

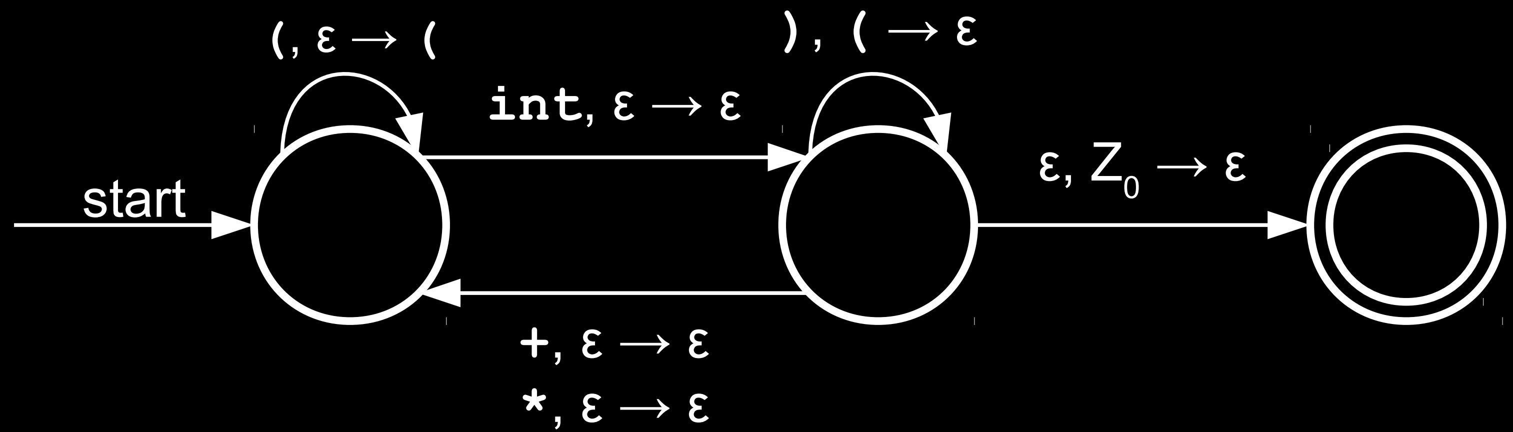
Wow, I guess not.

But we did learn something cool:

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

So what else is out there?

Well, what if we add memory
to our machines?



(, ε → (
int, ε → ε
), (→ ε
start
ε, Z_0 → ε
+, ε → ε
*, ε → ε

These machines can do more than our old machines!

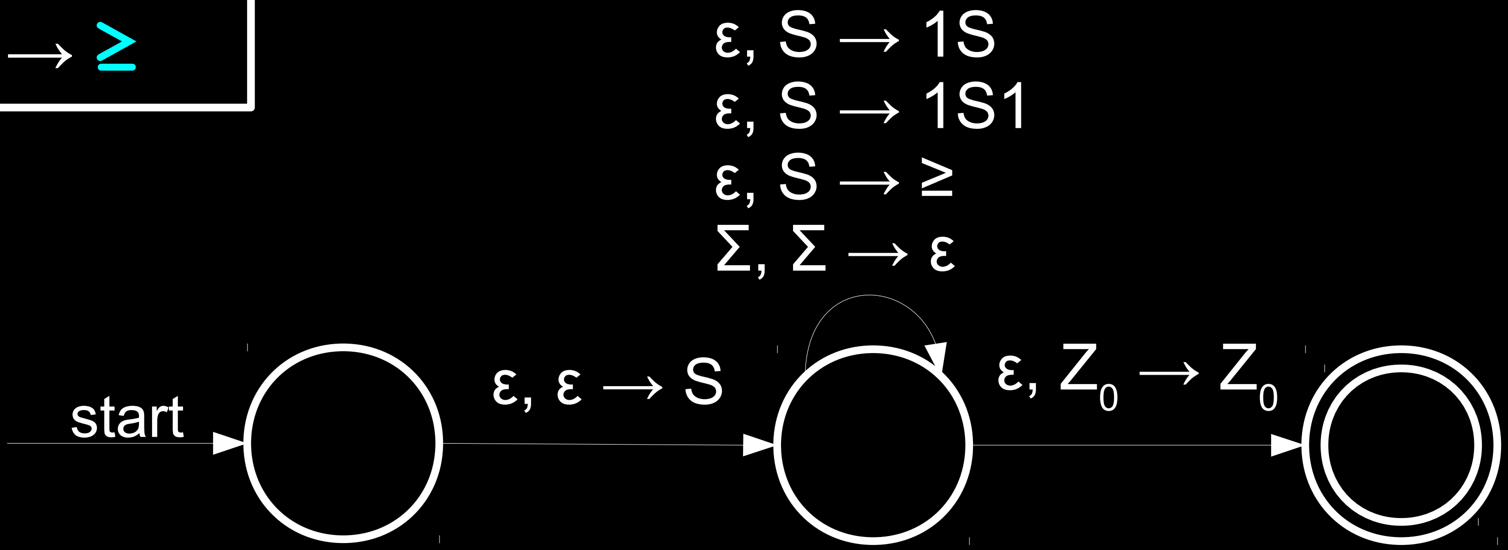
Can we describe these languages another way?

$$S \rightarrow \mathtt{a}X$$
$$X \rightarrow \mathtt{b} \mid C$$
$$C \rightarrow C\mathtt{c} \mid \varepsilon$$

$S \rightarrow 1S1$

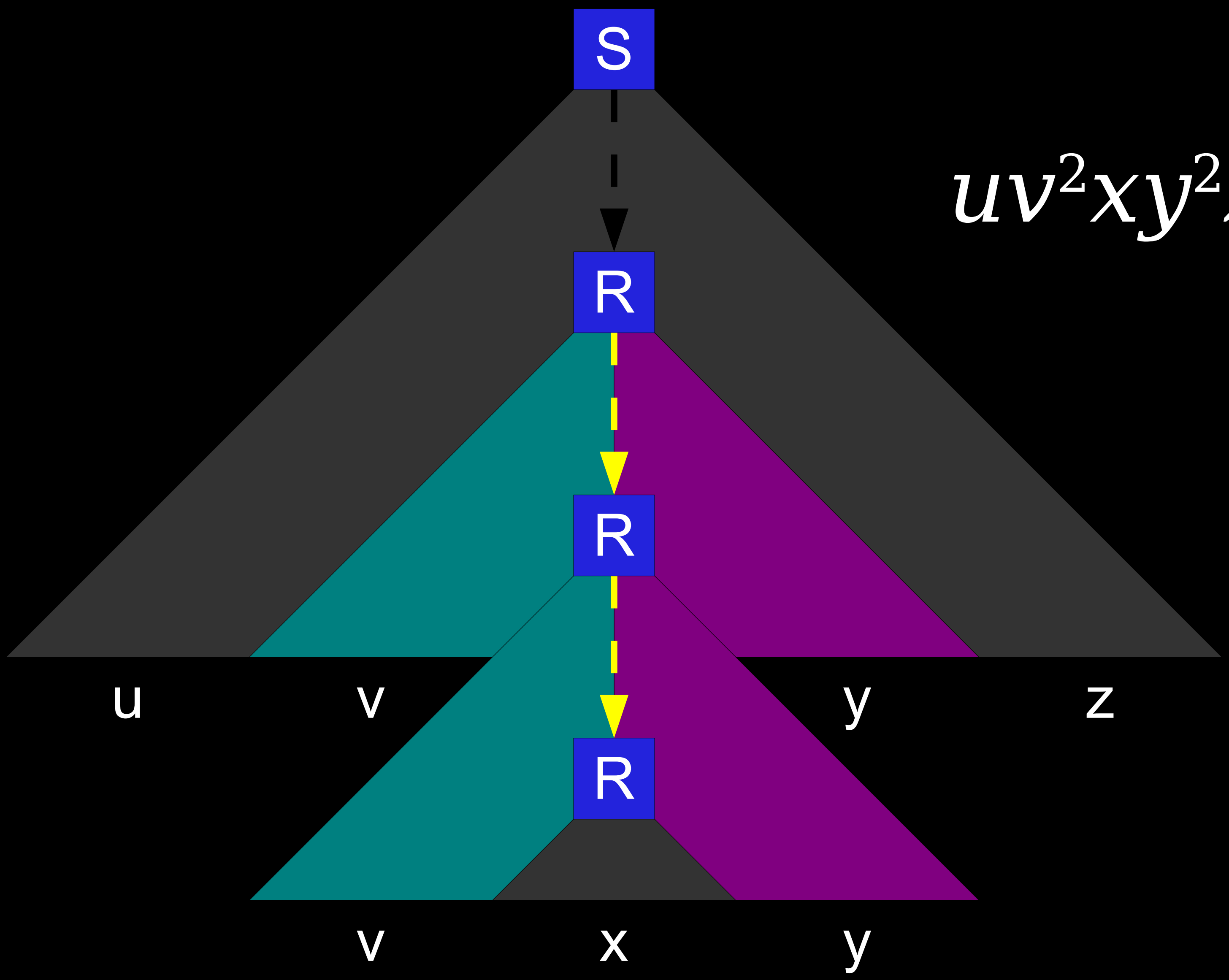
$S \rightarrow 1S$

$S \rightarrow$ ≥



Awesome!

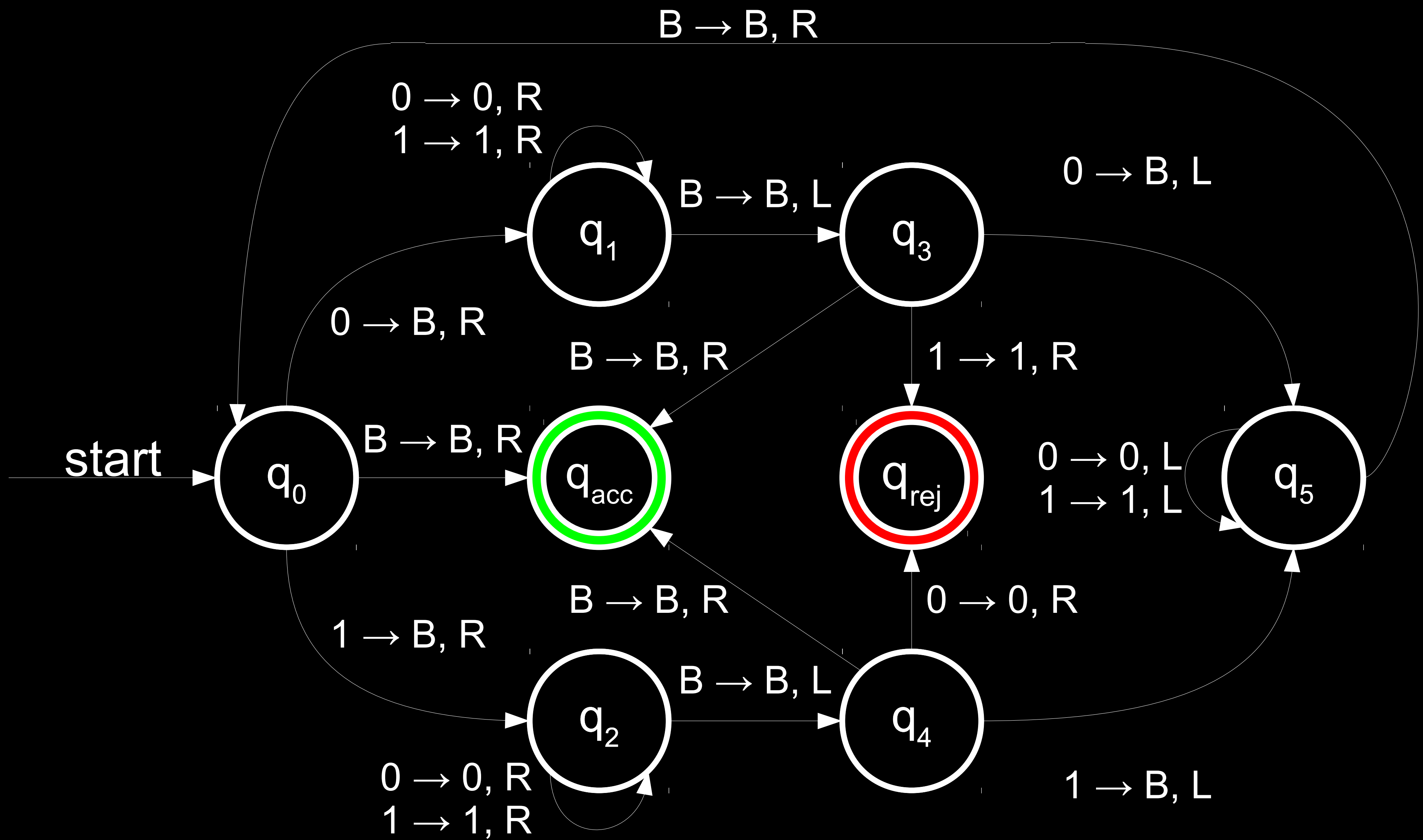
So did we get every language yet?



$$uv^2xy^2z \in L$$

Hmmm... guess not.

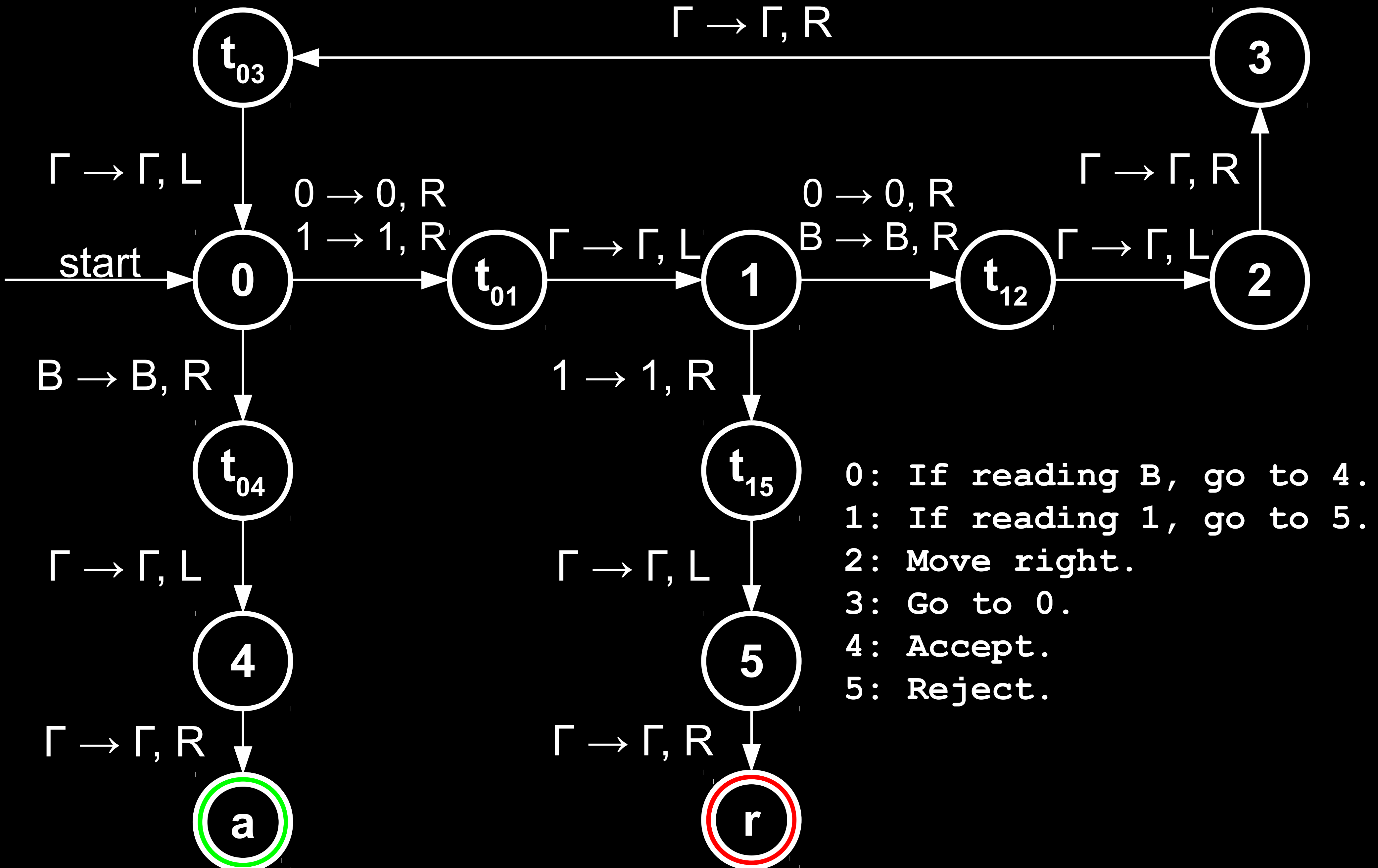
So what if we make our memory a little better?


B → B, R
0 → 0, R
1 → 1, R
q_1
B → B, L
q_3
0 → B, L
0 → B, R
B → B, R
1 → 1, R
start
q_0
B → B, R
q_{acc}
q_{rej}
0 → 0, L
1 → 1, L
q_5
B → B, R
0 → 0, R
1 → B, R
q_2
B → B, L
q_4
0 → 0, R
1 → 1, R
1 → B, L

Wow, these are hard to design.

Is there an easier way?

```
// Start
 0: If reading 0, go to M0.
 1: If reading 1, go to M1.
 2: Accept

// M0
 3: Write B.
 4: Move right.
 5: If reading 0, go to 4.
 6: If reading 1, go to 4.
 7: Move left.
 8: If reading 0, go to Next.
 9: Reject.

// M1
10: Write B.
11: Move right.
12: If reading 0, go to 11.
13: If reading 1, go to 11.
14: Move left.
15: If reading 1, go to Next.
16: Reject.

// Next
17: Write B.
18: Move left.
19: If reading 0, go to 18
20: If reading 1, go to 18
21: Move right
22: Go to Start.
```


Γ → Γ, R
Γ → Γ, L
0 → 0, R
1 → 1, R
Γ → Γ, L
0 → 0, R
B → B, R
Γ → Γ, L
Γ → Γ, R
start
B → B, R
1 → 1, R
Γ → Γ, L
Γ → Γ, L
Γ → Γ, R
Γ → Γ, R
t_{03}
3
0
t_{01}
1
t_{12}
2
t_{04}
t_{15}
4
5
a
r
0: If reading B, go to 4.
1: If reading 1, go to 5.
2: Move right.
3: Go to 0.
4: Accept.
5: Reject.

Much better! So let's add
some new features.

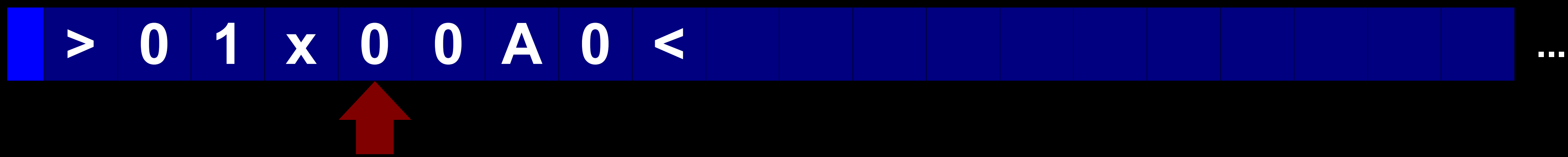
```
// Match
 7: Move tape 2 left until {B}
 8: Move tape 2 right.
 9: Move tape 1 right.
10: Write $ to tape 1, track 2.
11: If B on tape 2, go to Acc.
12: If B on tape 1, go to Rej.
13: Load tape 1, track 1 into X.
14: Load tape 2 into Y.
15: If X = Y, go to 17.
16: Go to Mismatch.
17: Move tape 1 right.
18: Move tape 2 right.
19: Go to 11.

// Mismatch
20: Move tape 1.2 left until {$}
21: Go to Match.

// Acc
22: Accept.

// Rej
23: Reject.
```

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

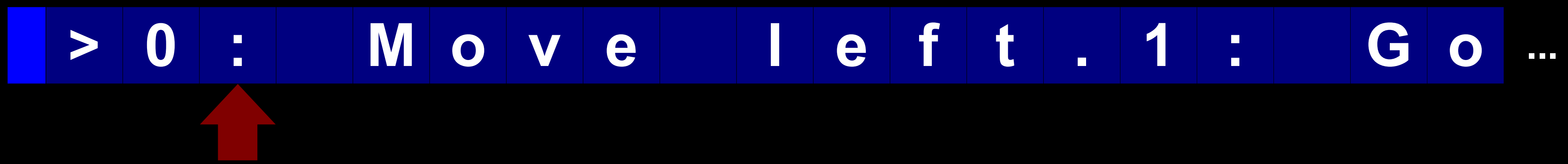
(The **Church-Turing thesis** says that this is not a coincidence!)

So why is that?

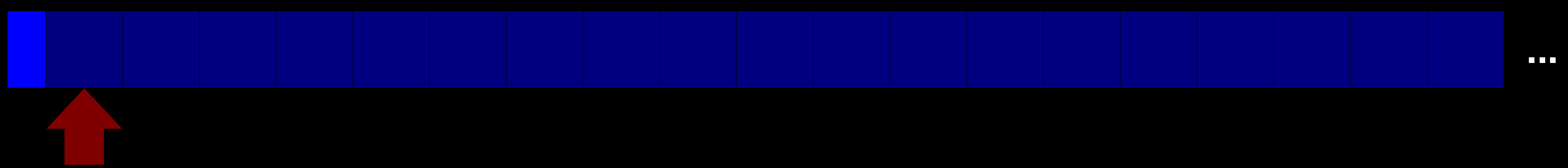
Simulated tape of the program being executed.



Program tape holding the program being executed.



Scratch tape for intermediate computation.



Variables for intermediate storage.

Instr **Letter**

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?

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

So is there just one
problem we can't solve?

$$L_D \leq_M \overline{\mathrm{A}_{\mathrm{TM}}}$$

$$\mathrm{A}_{\mathrm{TM}} \in \mathbf{RE}$$

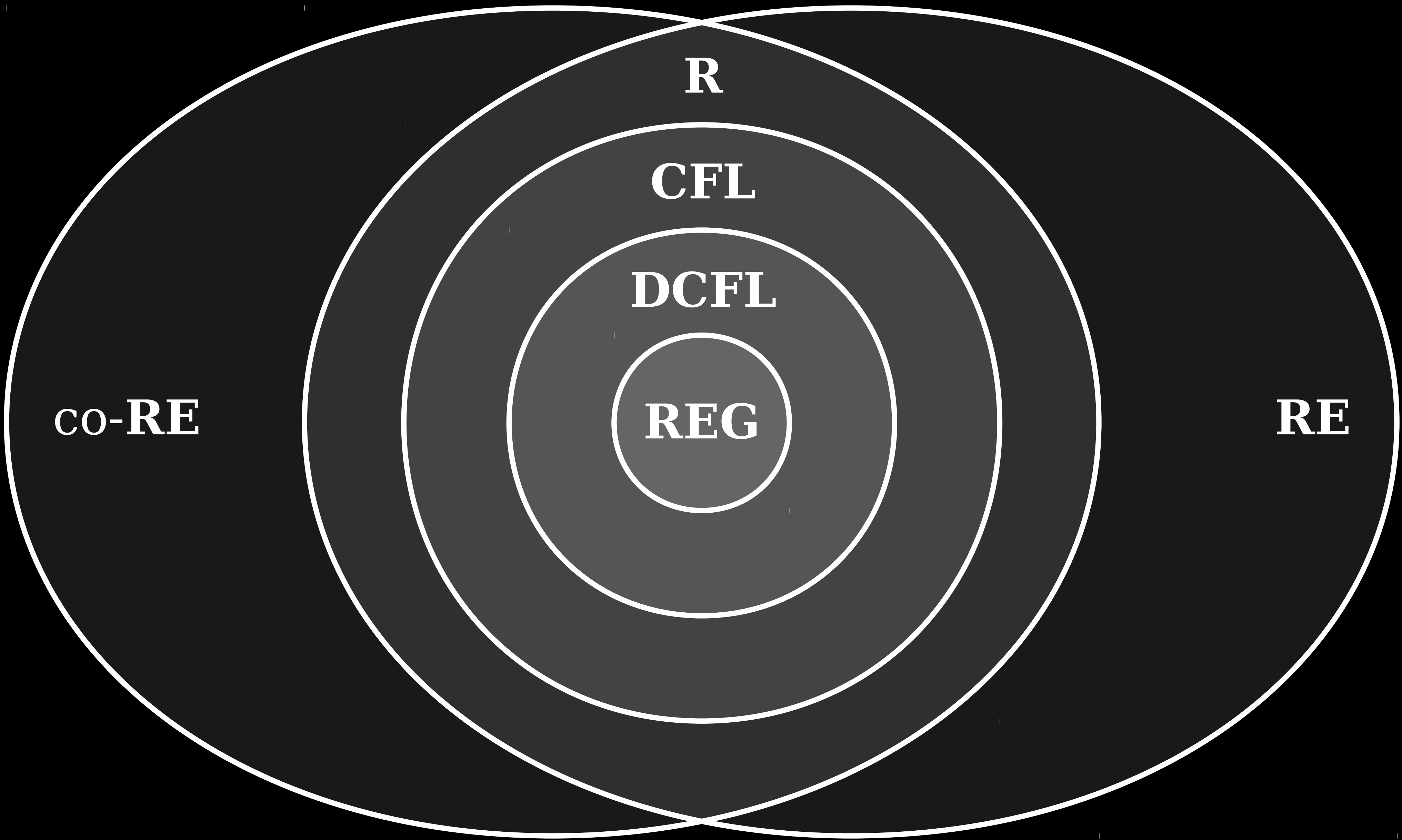
$$\mathrm{A}_{\mathrm{TM}} \notin \mathbf{R}$$

Okay... maybe we can't decide
or recognize everything.

Can we at least verify or refute everything?

$$L_{\mathrm{D}} \leq \mathrm{REGULAR}_{\mathrm{TM}}$$

$$\overline{L_{\mathrm{D}}} \leq \mathrm{REGULAR}_{\mathrm{TM}}$$


R
CFL
DCFL
REG
co-RE
RE

Wow. That's pretty deep.

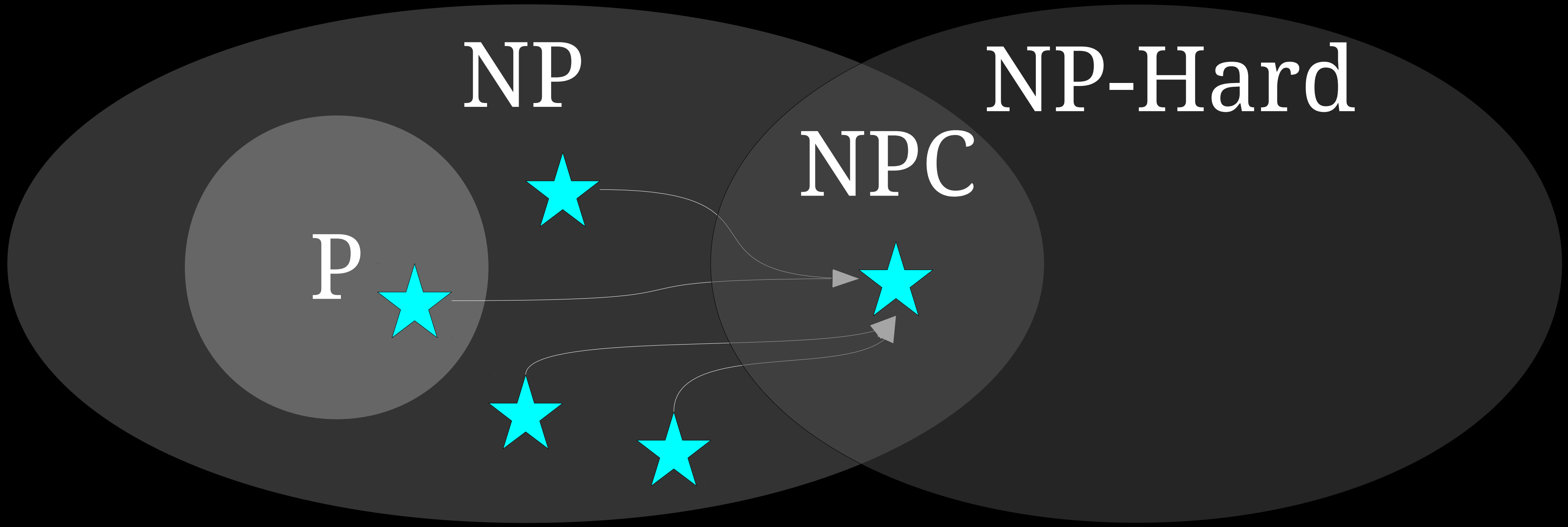
So... what can we do efficiently?

NP

So... how are you two related again?

No clue.

But what do we know about them?


NP
NP-Hard
NPC
P

What other mysteries remain in theoretical computer science?

A Whole World of Theory Awaits!

Theory is all about exploring, experimenting, and discovering.

We've barely scratched the surface of theoretical computer science.

Theory is all about exploring, experimenting, and discovering.

We've barely scratched the surface of theoretical computer science.

Where to Go From Here


Theoretical
Applied

CS154

- **Intro to Automata and Complexity Theory**
- An in-depth treatment of automata, computability, and complexity.
- Emphasis on theoretical results in automata theory and complexity.
- Launching point for more advanced courses (CS254, CS354)

CS258

- **Intro to Programming Language Theory**
- Explore questions of computability in terms of recursion and recursive functions.
- Excellent complement to the material in this course; highly recommended.
- Offered every other year; consider checking it out!

CS109

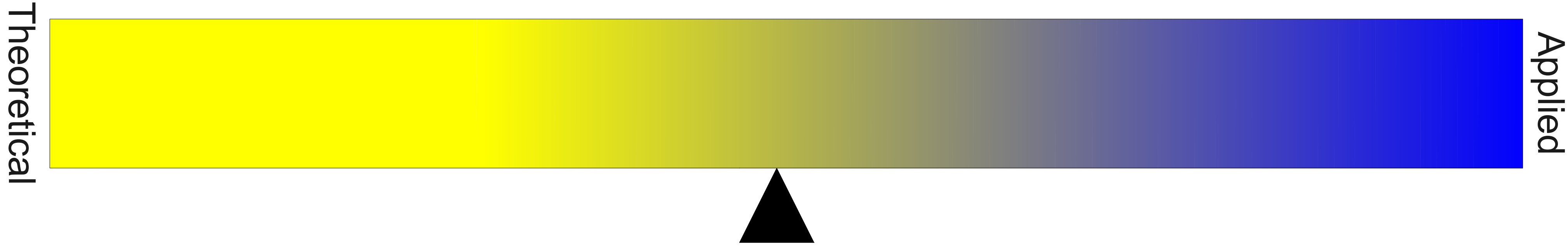
- **Intro to Probability for Computer Scientists**
- Learn to embrace randomness.
- Use your newly acquired proof skills in an entirely different domain.
- See how computers can use statistics to learn patterns.

CS255

- **Intro to Cryptography**
- Use hard problems to your advantage!
- Explore **NP**-hardness and its relation to cryptography.
- See how to design secure systems out of hard problems.

CS161

- **Design and Analysis of Algorithms**
- Learn how to approach new problems and solve them efficiently.
- Learn how to deal with **NP**-hardness in the real world.
- Learn how to ace job interviews



CS143

- **Compilers**
- Watch automata, grammars, undecidability, and **NP**-completeness come to life by building a complete working compiler from scratch.
- See just how much firepower you can get from all this material.

CS107

- **Computer Organization and Systems**
- You don't need to be a theoretician to love computer science!
- If you want to learn how the machine works under the hood, look no further.

There are more problems to solve than there are programs to solve them.

Where We've Been

- **Given this hard theoretical limit, what *can* we compute?**
  - What are the hardest problems we *can* solve?
  - How powerful of a computer do we need to solve these problems?
  - Of what we can compute, what can we compute *efficiently*?
- **What tools do we need to reason about this?**
  - How do we build mathematical models of computation?
  - How can we reason about these models?

What We've Covered

- Sets
- Graphs
- Proof Techniques
- Relations
- Functions
- Cardinality
- Induction
- Logic
- Pigeonhole Principle
- Trees
- DFAs
- NFAs
- Regular Expressions
- CFGs
- PDAs
- Pumping Lemmas
- Turing Machines
- **R**, **RE**, and co-**RE**
- Unsolvable Problems
- Reductions
- Time Complexity
- **P**
- **NP**
- **NP**-Completeness

My Email Address:

htiek@cs.stanford.edu

Final Thoughts