

Turing Machines

Part One

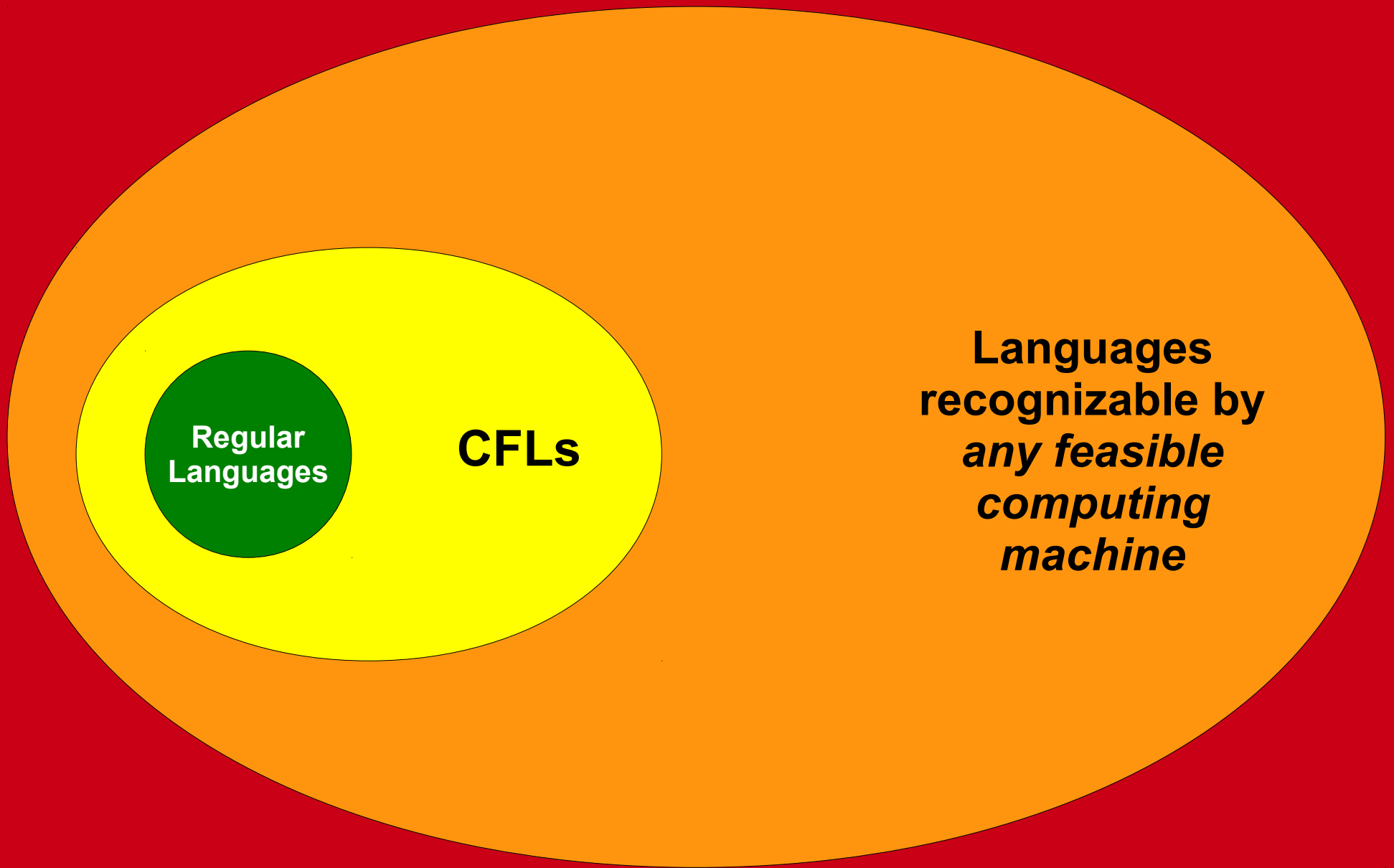
Hello Condensed Slide Readers!

The lecture slides for today's lecture consist almost entirely of TM animations showing the step-by-step process by which we create the TMs for various languages. A lot of that is missing in these animated slide decks, so I would definitely recommend checking out the fully-animated slides in addition to these.

Hope this helps!

-Keith

Are some problems inherently
harder than others?



**Regular
Languages**

CFLs

**Languages
recognizable by
*any feasible
computing
machine***

All Languages

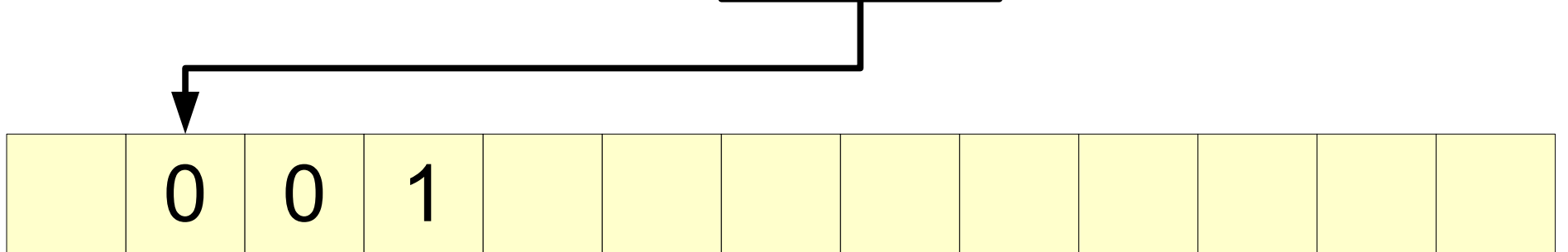
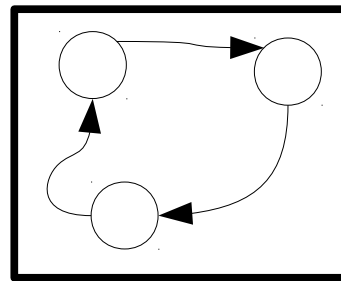
That same drawing, to scale.

The Problem

- Finite automata accept precisely the regular languages.
- We may need unbounded memory to recognize context-free languages.
 - e.g. $\{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$ requires unbounded counting.
- How do we build an automaton with finitely many states but unbounded memory?

A Better Memory Device

- A **Turing machine** is a finite automaton equipped with an **infinite tape** as its memory.
- The input is written on the tape when the computation begins, surrounded by infinitely many blank cells.
- Each transition depends on the current symbol under the tape head.



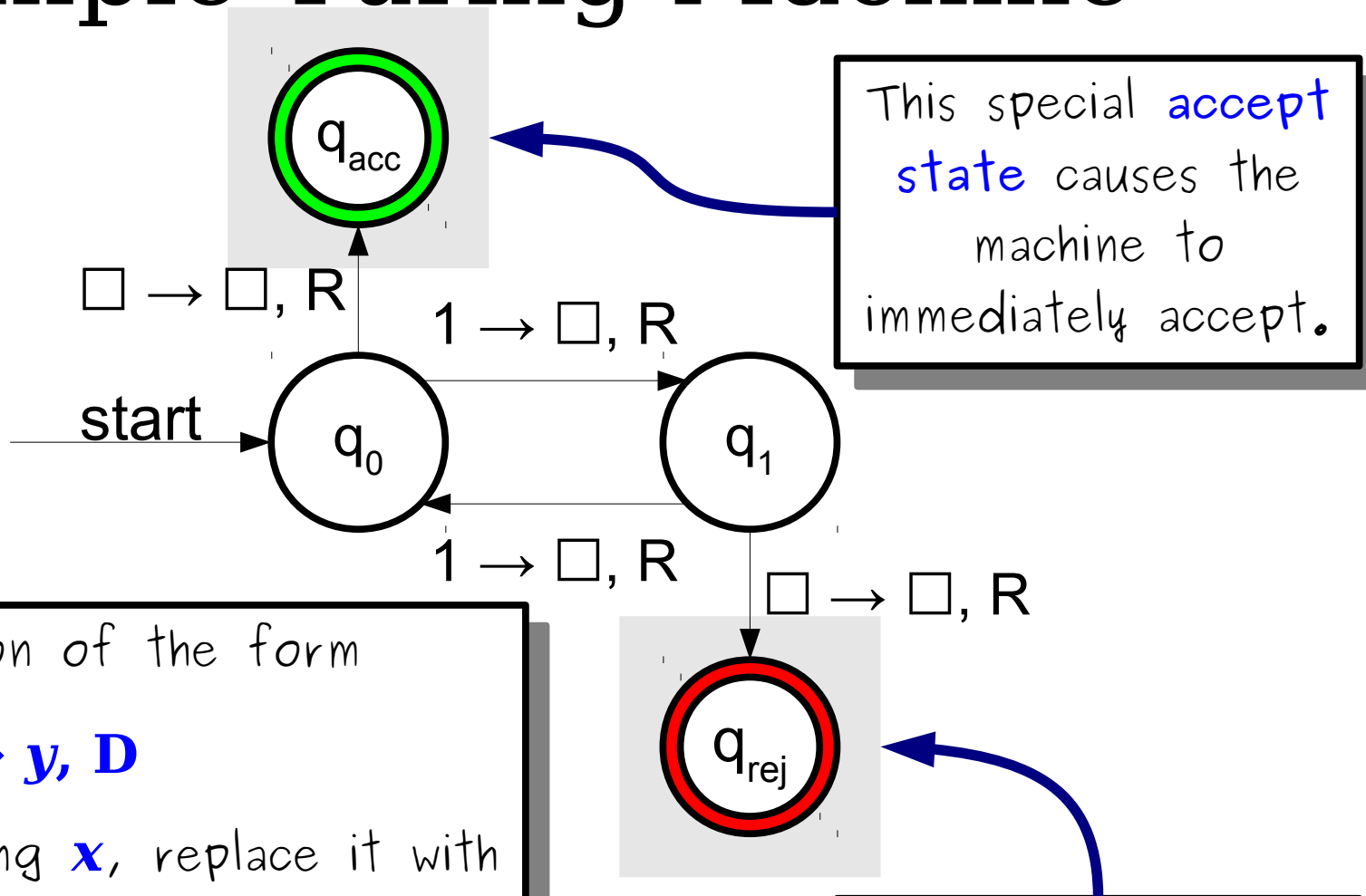
The Turing Machine

- A Turing machine consists of three parts:
 - A ***finite-state control*** that issues commands,
 - an ***infinite tape*** for input and scratch space, and
 - a ***tape head*** that can read and write a single tape cell.
- At each step, the Turing machine
 - writes a symbol to the tape cell under the tape head,
 - changes state, and
 - moves the tape head to the left or to the right.

Input and Tape Alphabets

- A Turing machine has two alphabets:
 - An **input alphabet** Σ . All input strings are written in the input alphabet.
 - A **tape alphabet** Γ , where $\Sigma \subseteq \Gamma$. The tape alphabet contains all symbols that can be written onto the tape.
- The tape alphabet Γ can contain any number of symbols, but always contains at least one **blank symbol**, denoted \square . You are guaranteed $\square \notin \Sigma$.
- At startup, the Turing machine begins with an infinite tape of \square symbols with the input written at some location. The tape head is positioned at the start of the input.

A Simple Turing Machine



Each transition of the form

$$x \rightarrow y, D$$

means "upon reading x , replace it with symbol y and move the tape head in direction D (which is either **L** or **R**).

The symbol \square represents the **blank symbol**.

This special **reject state** causes the machine to immediately reject.

Accepting and Rejecting States

- Unlike DFAs, Turing machines do not stop processing the input when they finish reading it.
- Turing machines decide when (and if!) they will accept or reject their input.
- Turing machines can enter infinite loops and never accept or reject; more on that later...

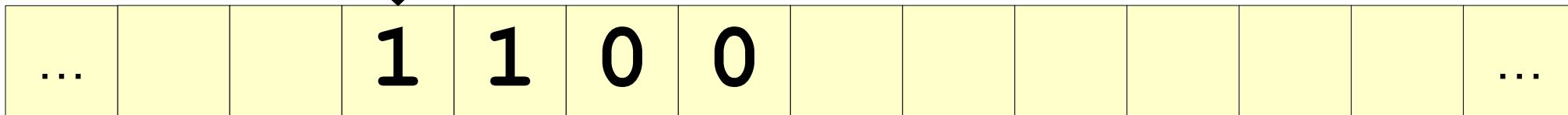
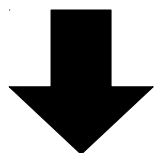
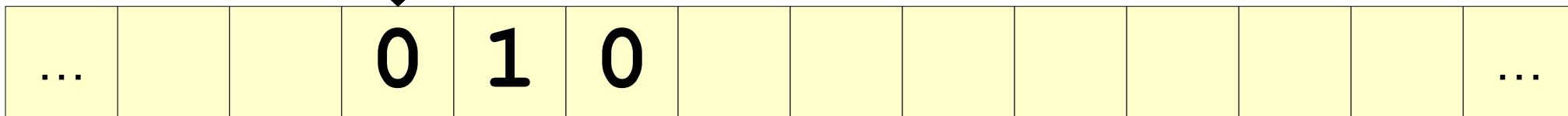
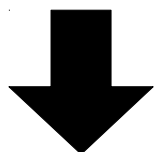
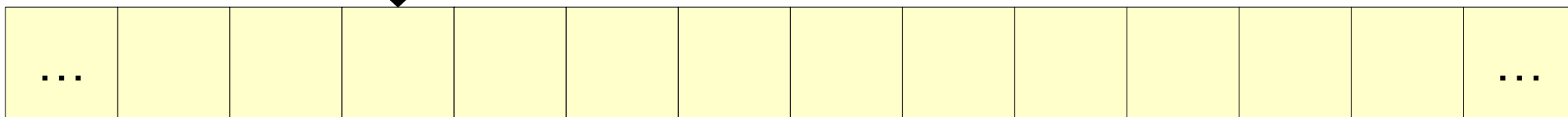
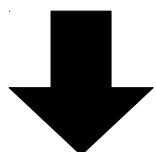
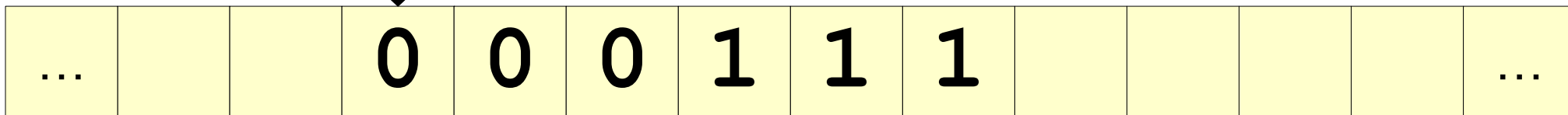
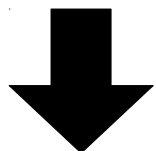
Designing Turing Machines

- Despite their simplicity, Turing machines are very powerful computing devices.
- Today's lecture explores how to design Turing machines for various languages.

Designing Turing Machines

- Let $\Sigma = \{0, 1\}$ and consider the language $L = \{0^n 1^n \mid n \in \mathbb{N}\}$.
- We know that L is context-free.
- How might we build a Turing machine for it?

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$



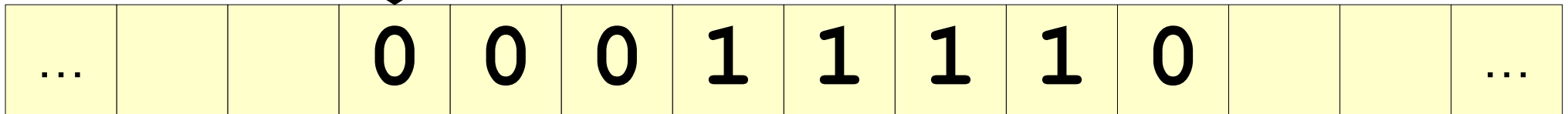
A Recursive Approach

- The string ε is in L .
- The string $0w1$ is in L iff w is in L .
- Any string starting with 1 is not in L .
- Any string ending with 0 is not in L .

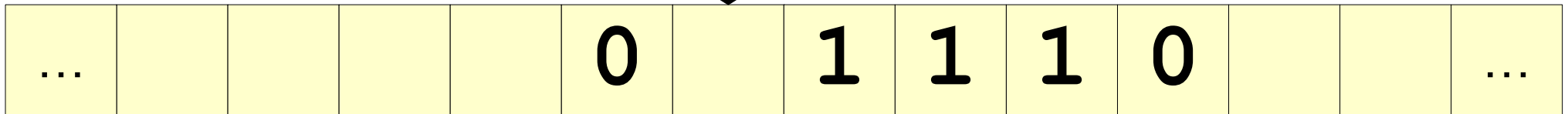
Another TM Design

- We've designed a TM for $\{0^n 1^n \mid n \in \mathbb{N}\}$.
- Consider this language over $\Sigma = \{0, 1\}$:
$$L = \{ w \in \Sigma^* \mid w \text{ has the same number of } 0\text{s and } 1\text{s} \}$$
- This language is also not regular, but it is context-free.
- How might we design a TM for it?

A Caveat

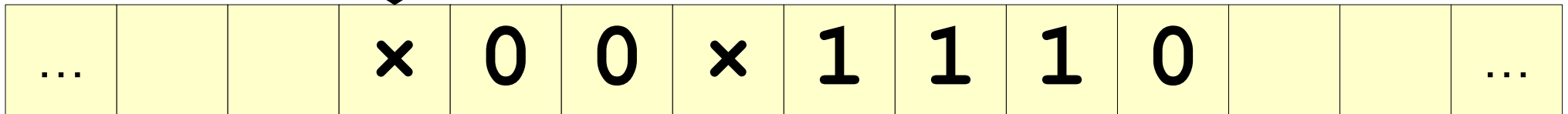


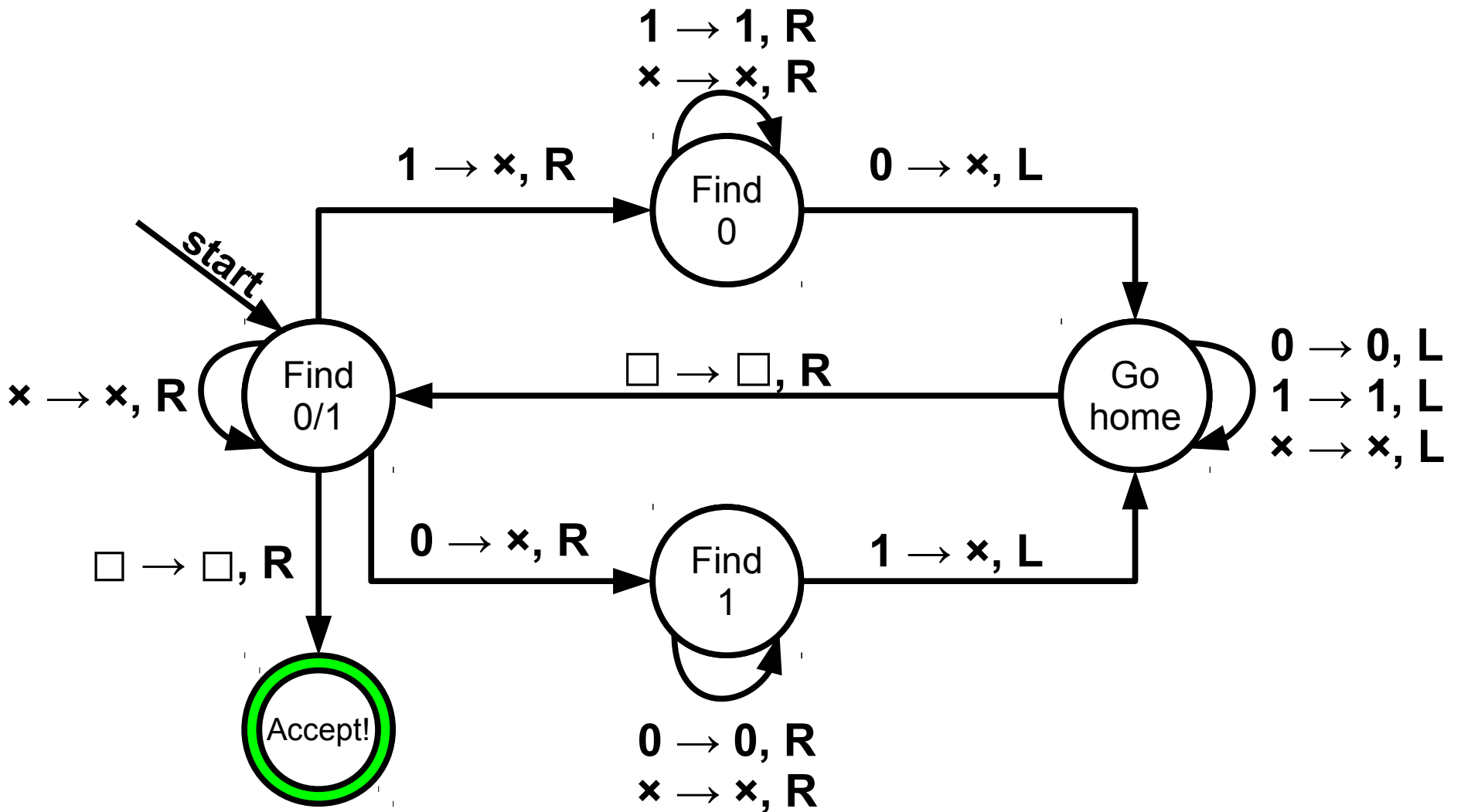
A Caveat



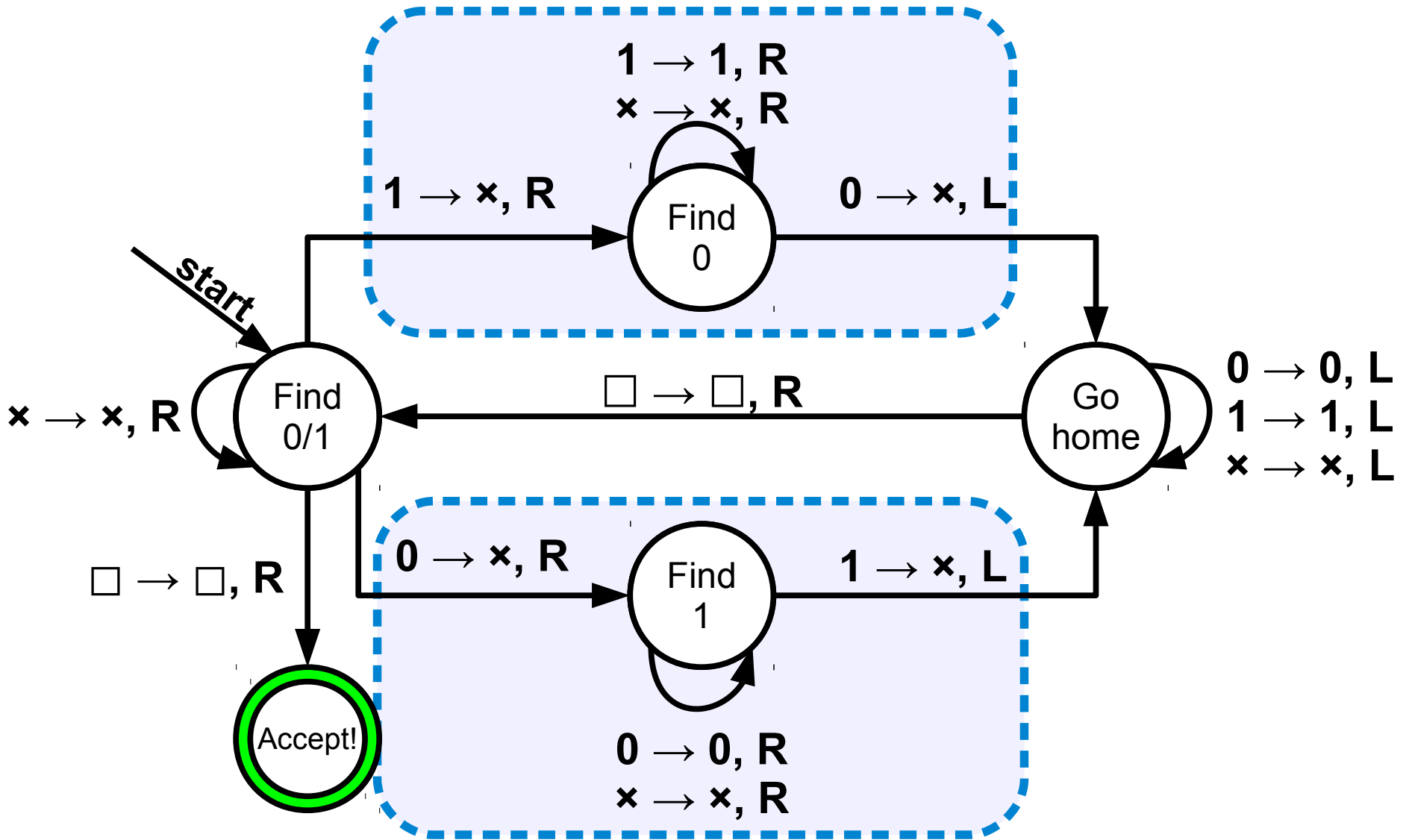
How do we know that this blank isn't one of the infinitely many blanks after our input string?

The Solution





Going forward, we'll ignore the missing transitions and pretend they implicitly reject.



Constant Storage

- Sometimes, a TM needs to remember some additional information that can't be put on the tape.
- In this case, you can use similar techniques from DFAs and introduce extra states into the TM's finite-state control.
- The finite-state control can only remember one of finitely many things, but that might be all that you need!

Time-Out for Announcements!

Problem Set Six

- Problem Set Five was due at the start of lecture today.
 - Due Tuesday with one late day and Wednesday with two late days.
- Problem Set Six goes out now, is due at the start of next Monday's lecture.
 - Play around with nonregular languages, the Myhill-Nerode theorem, and context-free grammars!
- The second midterm is a week from Thursday.
We do not recommend using late days on Problem Set Six.

Your Questions

“What do you think about entrepreneurship?
Have you ever considered becoming an
entrepreneur? Why or why not?”

I had a brilliant idea for a startup in my freshman year, but then Google did it. 😊

It's a mixed bag! I think the entrepreneurial spirit is great in that it challenges people to just go fix the problems they see. I think it's a bit unhealthy in that people feel pressured to make startups when they honestly should just keep studying and learning more about the world.

“How are you working towards making CS a factor in making the lives of the less fortunate better when all it seems that CS, outside of academia, can do is solve problems for the rich?”

I guarantee you I'm not doing enough. I can talk about some of the things that I'm currently doing / hoping to do.

Computing gives people a chance to climb up the economic ladder and can empower the weak and vulnerable if used correctly. I hope that I'm giving people the tools to help make this happen.

“If I did poorly on the midterm (failed) and well on the problem sets, what is my standing in the class? Am i at risk of failing?”

Here's full disclosure on how I compute grades!

1. I compute raw scores weighted by the amounts I said I was going to weight everything by (each problem set has a weight printed on the front, the midterms are 15% each, and the final is 30%).

2. I compute a grading curve. I never curve down: a 90% is always an A-, an 80% is always a B-, etc. I usually put the median as the cutoff between B/B+ and usually put the 25th percentile as the C+/B- cutoff. The B-/A- cutoff fluctuates a bit, but it's usually around the 60th percentile mark. I always do a follow-up check to make sure that I can explain all the grades I'm giving. I also leave out extra credit when designing the curve, but leave it in when assigning letter grades.

You can definitely pass the class if you failed the first exam - you can pass even if you didn't take it! Just try to avoid having two bad exams - that will really hurt your grade.

Back to CS103!

Another TM Design

- Consider the following language over $\Sigma = \{0, 1\}$:

$$L = \{0^n 1^m \mid n, m \in \mathbb{N} \text{ and } m \text{ is a multiple of } n \}$$

- Is this language regular?
- How might we design a TM for this language?

An Observation

- We can recursively describe when one number m is a multiple of n :
 - If $m = 0$, then m is a multiple of n .
 - Otherwise, m is a multiple of n iff $m - n$ is a multiple of n .
- **Idea:** Repeatedly subtract n from m until m becomes zero (good!) or drops below zero (bad!)

