Turing Machines

Part One

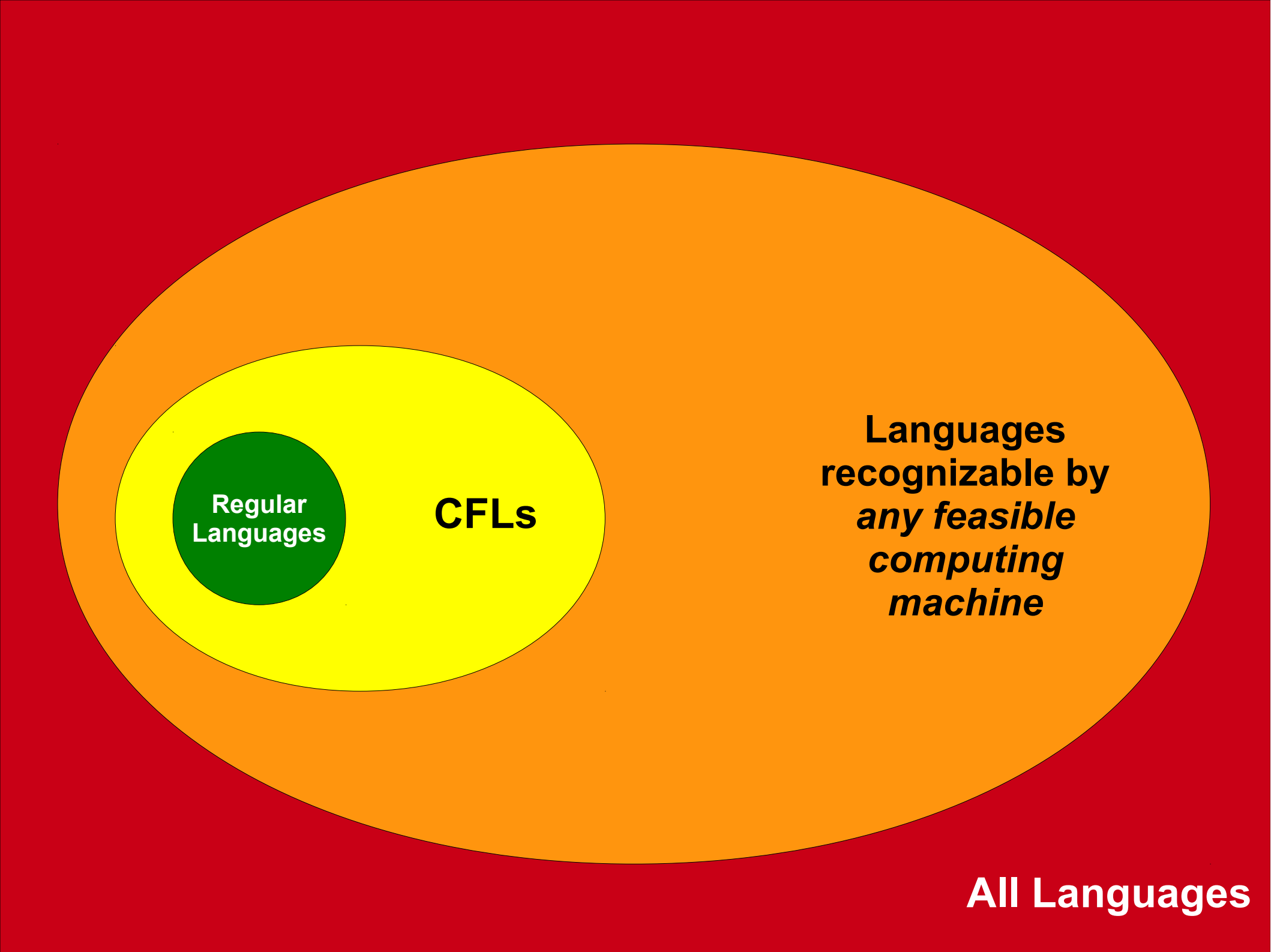
Hello Condensed Slide Readers!

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

What problems can we solve with a computer?


Regular Languages
CFLs
Languages recognizable by *any feasible computing machine*
All Languages

That same drawing, to scale.

All Languages

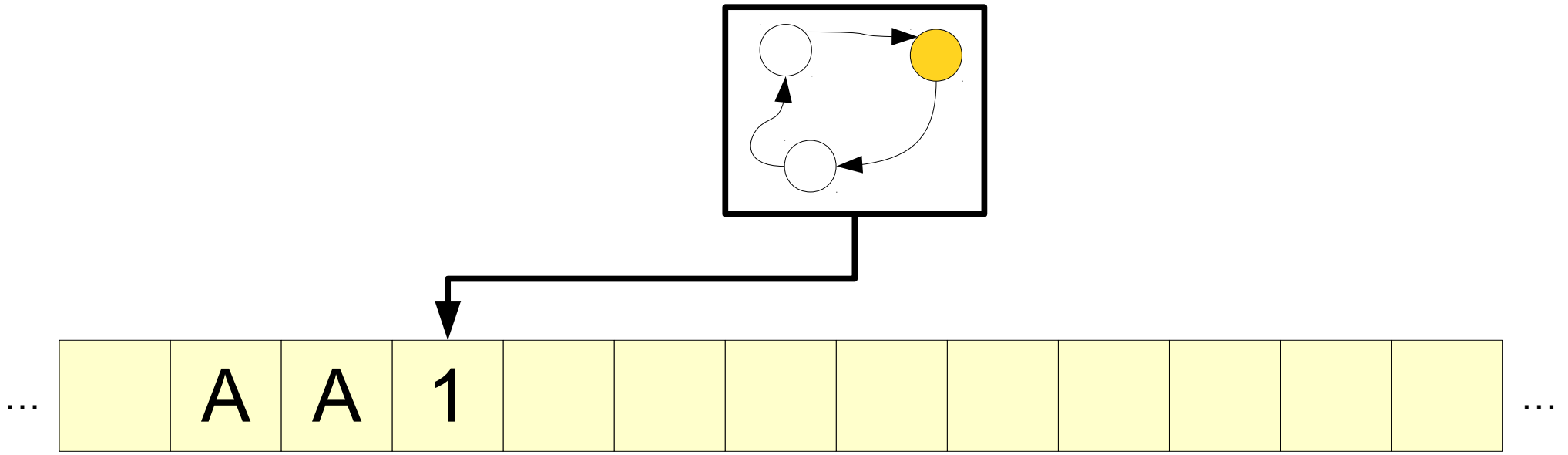
The Problem

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

A Brief History Lesson

A Better Memory Device

- A ***Turing machine*** is a deterministic 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.



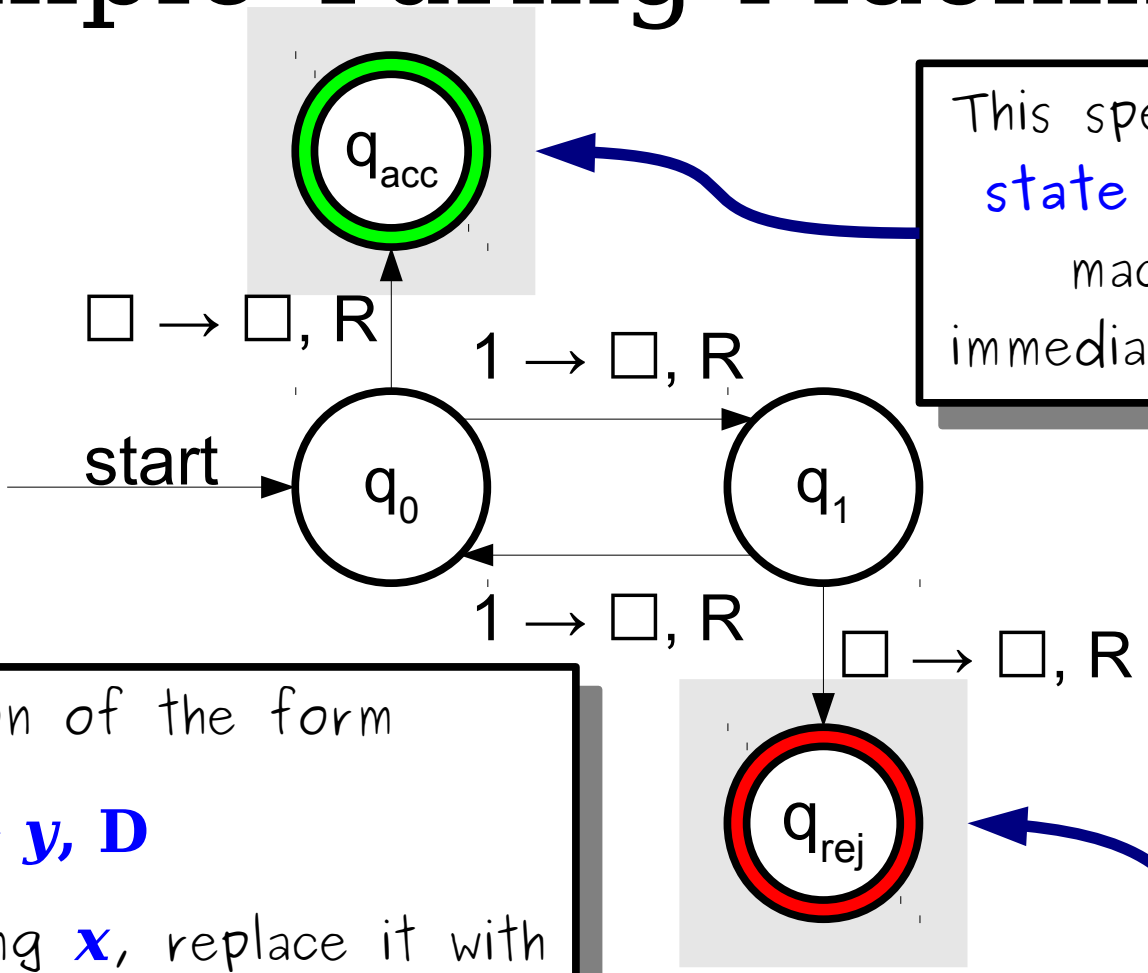
The Turing Machine

- 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 Σ ⊆ Γ. 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 □. You are guaranteed □ ∉ Σ.
- At startup, the Turing machine begins with an infinite tape of □ symbols with the input written at some location. The tape head is positioned at the start of the input.

A Simple Turing Machine



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

Each transition of the form

$$x \to y, D$$

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 □ 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...

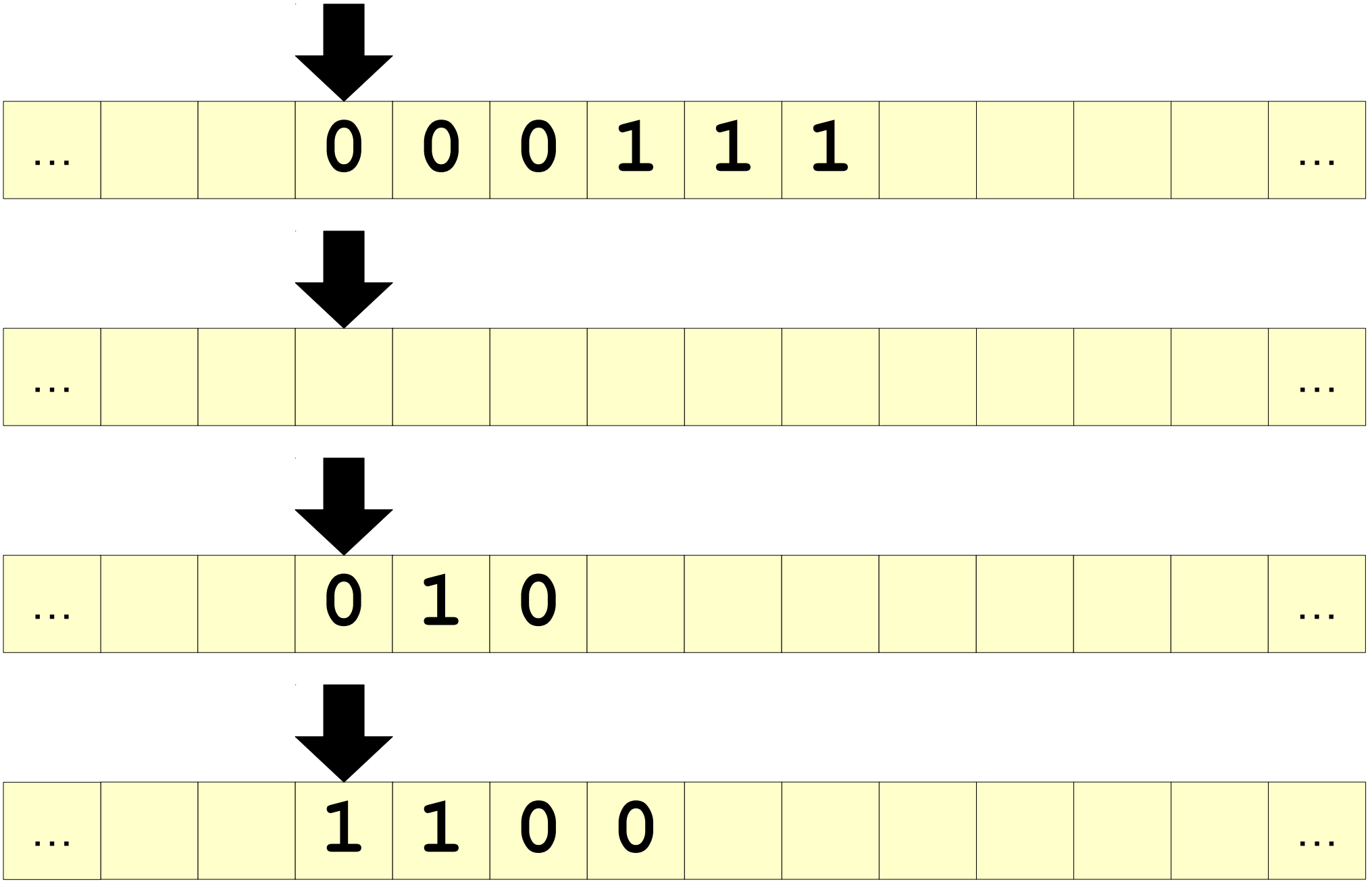
Designing Turing Machines

- 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

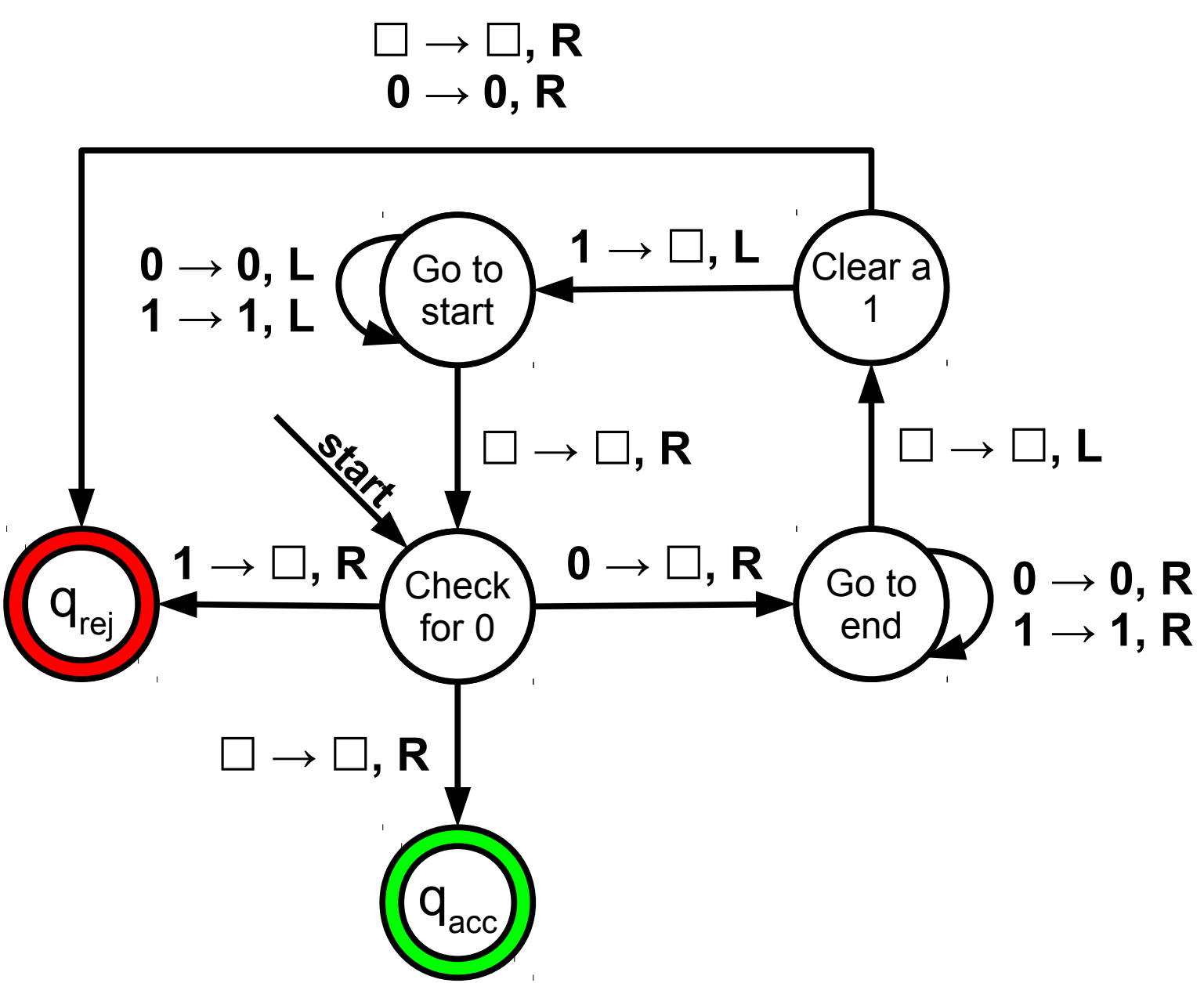
- Let $\Sigma = \{\texttt{0}, \texttt{1}\}$ and consider the language $L = \{\texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N}\}$.
- We know that L is context-free.
- How might we build a Turing machine for it?

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



A Recursive Approach

- The string ε is in L.
- The string `0`w`1` 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.


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

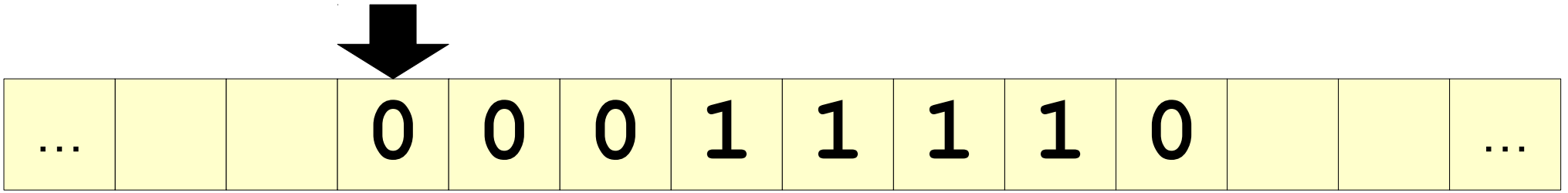
Another TM Design

- We've designed a TM for $\{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$.
- Consider this language over $\Sigma = \{ \mathtt{0}, \mathtt{1} \}$:

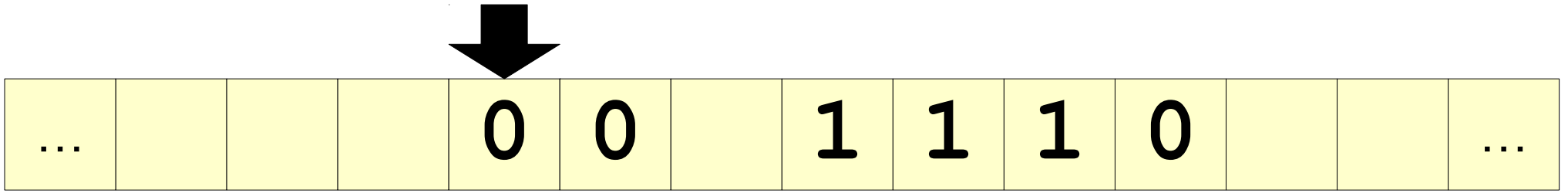
$$L = \{ w \in \Sigma^* \mid w \text{ has the same number of } \mathtt{0}\text{s and } \mathtt{1}\text{s} \}$$

- This language is also not regular, but it is context-free.
- How might we design a TM for it?

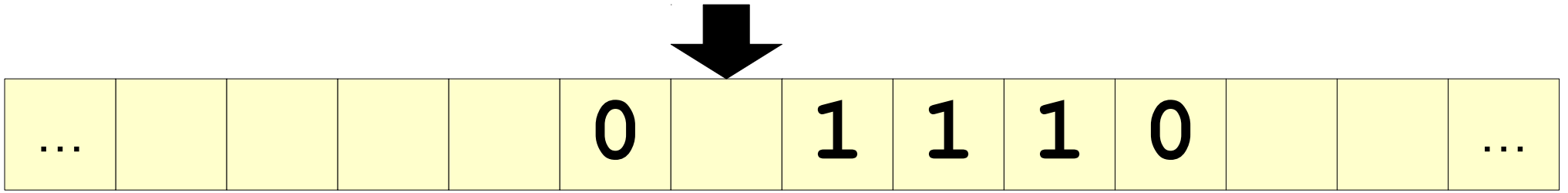
A Caveat



A Caveat

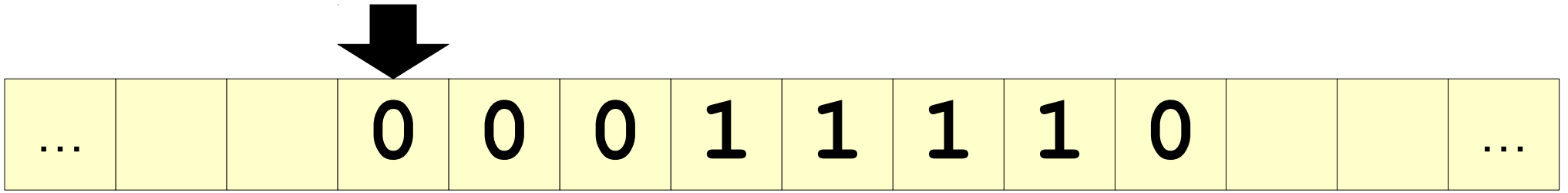


A Caveat

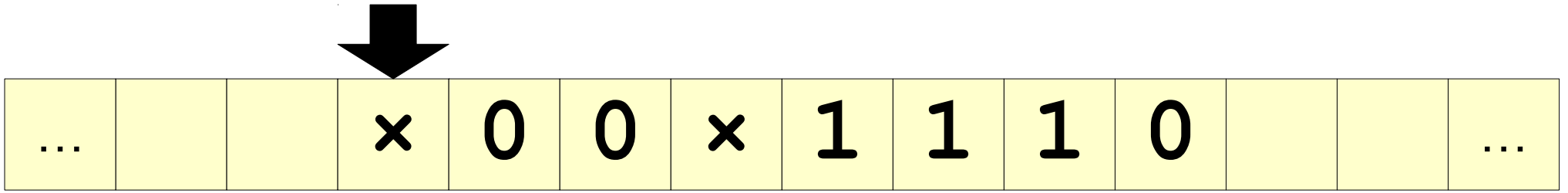


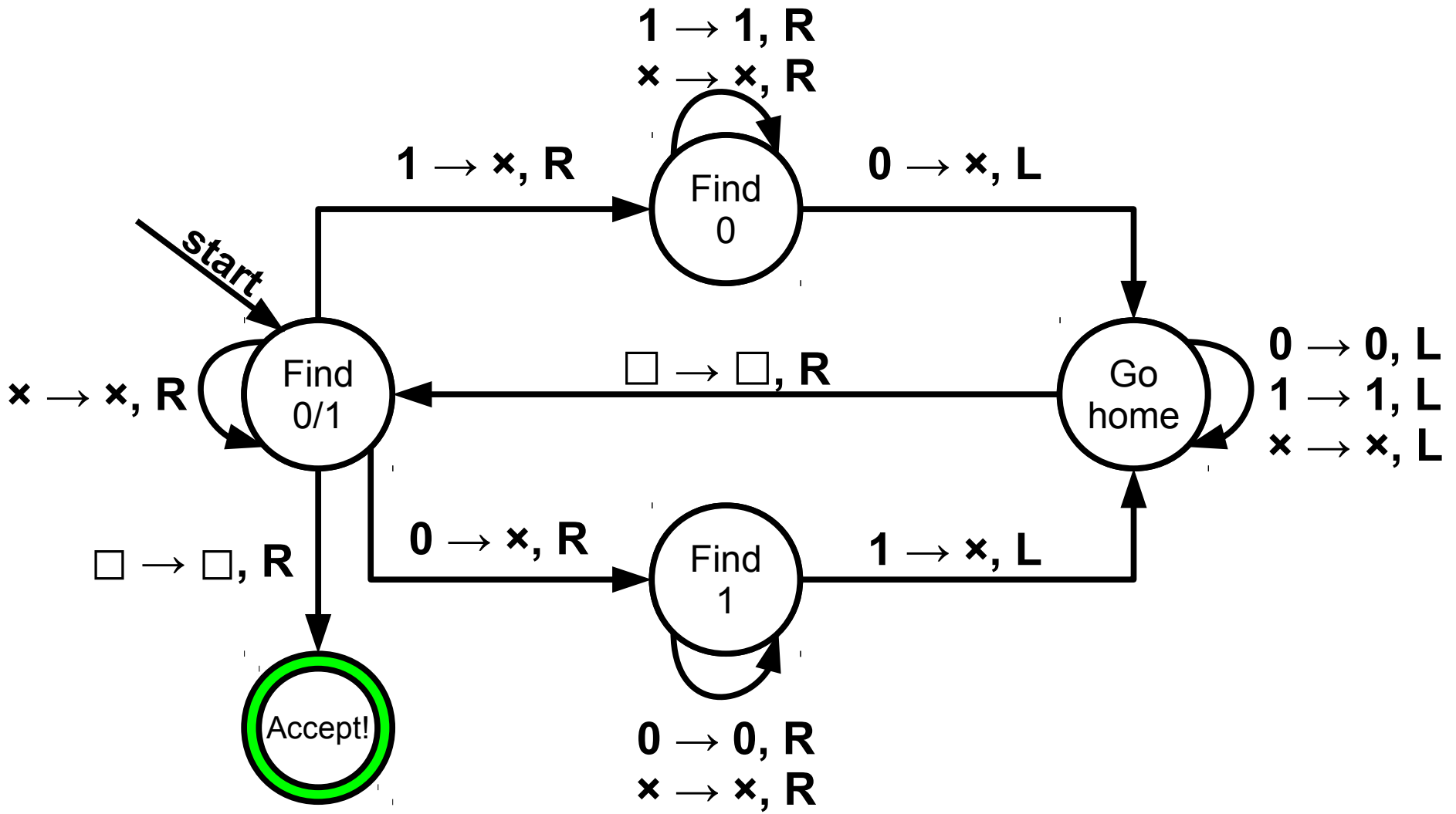
How do we know that this blank isn't one of the infinitely many blanks after our input string?

The Solution

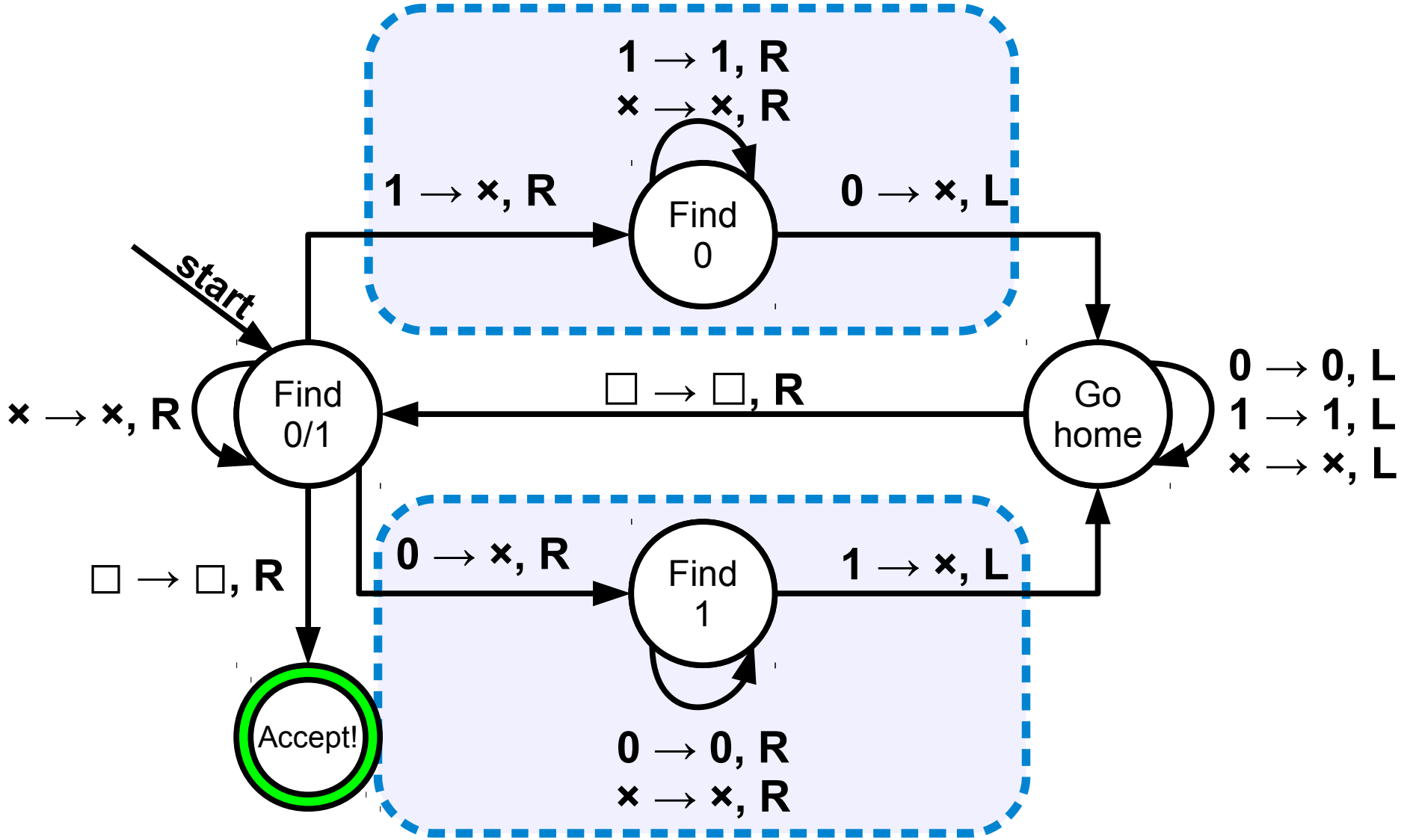


The Solution





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


1 → 1, R
× → ×, R
1 → ×, R
Find 0
0 → ×, L
start
Find 0/1
× → ×, R
□ → □, R
Go home
0 → 0, L
1 → 1, L
× → ×, L
0 → ×, R
Find 1
1 → ×, L
□ → □, R
Accept!
0 → 0, R
× → ×, R

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 Seven

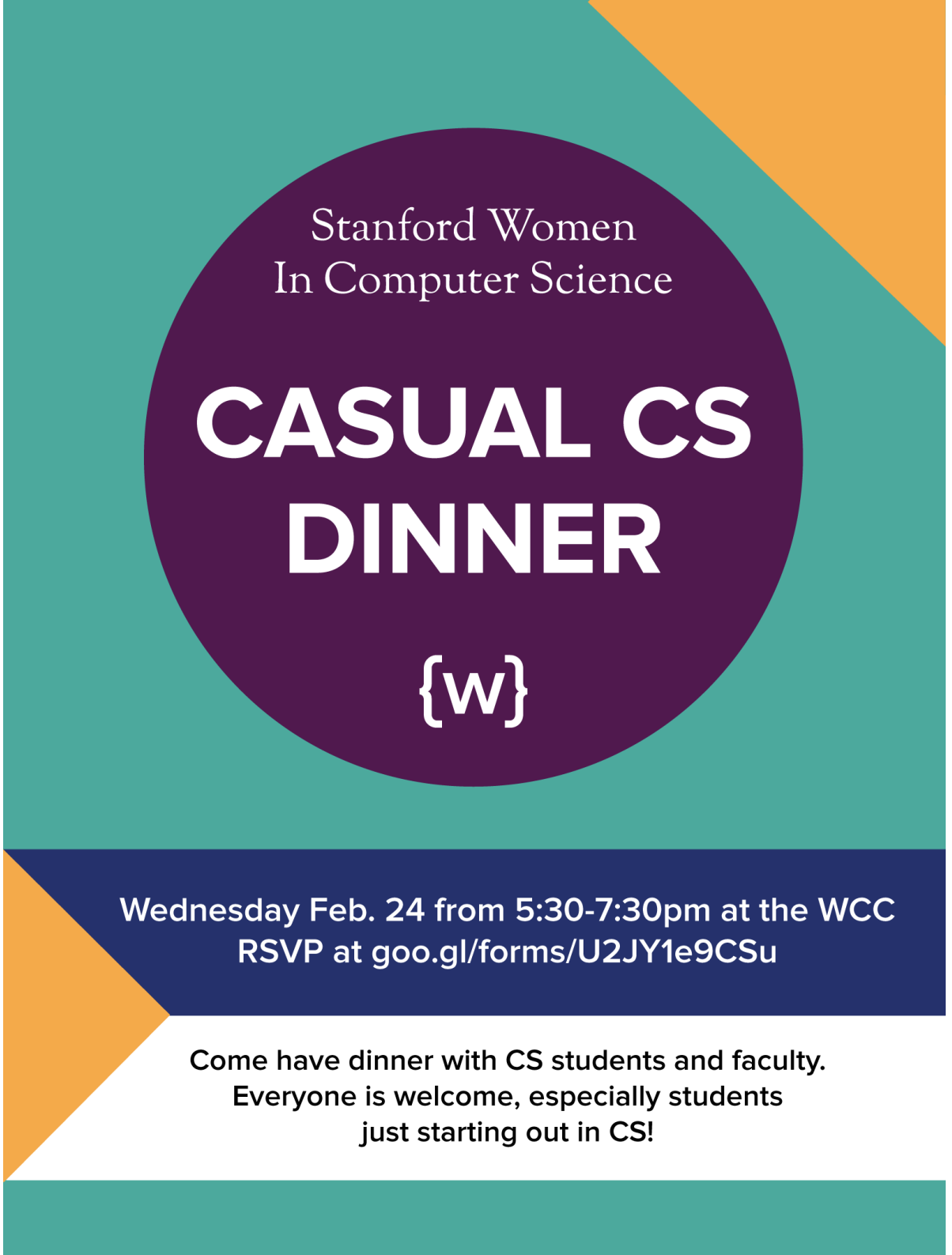
- Problem Set Seven is due on Friday.
- Have questions? Please feel free to stop by office hours or to ask questions on Piazza.

Second Midterm Exam

- The second midterm exam is next **Monday, February 29** from 7PM – 10PM, location TBA.
- Topic coverage:
  - Focus is on PS4 – PS6 and lectures 09 – 16.
  - Topics from PS7 and from lecture 17 onward will not be tested… yet. ☺
  - ***Major topics:*** strict orders, graphs, the pigeonhole principle, induction, finite automata, regular expressions, regular languages, closure properties.
- Policies and procedures same as the first midterm:
  - Three hours, four questions.
  - Closed-computer, closed-book, and limited-note. You can have a double-sided 8.5” × 11” sheet of paper with you when you take the exam.

Preparing for the Exam

- We have just released four sets of extra practice problems (EPP4 – EPP7) online. Solutions will go out on Wednesday.
- We will be holding a practice midterm on **Wednesday** from **7PM – 10PM** in **Bishop Auditorium**.
- By popular demand, we've put together *two* practice exams – one that we'll give out at the practice exam and one that will be posted online on Wednesday.
- We've also released a set of challenge problems, and this week's CS103A class will be a cumulative review.
- That's (by my reckoning) four sets of practice problems, two practice exams, one set of challenge problems, and one set of CS103A problems. If that's not enough, let me know and I'll see what I can do.


Stanford Women
In Computer Science
CASUAL CS
DINNER
{w}
Wednesday Feb. 24 from 5:30-7:30pm at the WCC
RSVP at goo.gl/forms/U2JY1e9CSu
Come have dinner with CS students and faculty.
Everyone is welcome, especially students
just starting out in CS!

Your Questions

“What's your favorite (and lesser-known) place on campus?”

There (used to be? still is?) a fire escape on one of the inner quad buildings that lets you climb up to the third floor. You get a great view and it's nice and quiet. It's a great place to study or cuddle and watch a movie.

“What keeps you motivated?”

I never cease to be amazed by the world. There's always so much to learn and discover.

Which reminds me – I haven't yet asked you the Three Key Questions yet!

Three Questions

- What is something that you now know that, at the start of the quarter, you knew you didn't know?
- What is something that you now know that, at the start of the quarter, you *didn't* know you didn't know?
- What is something that you *don't* know that, at the start of the quarter, you didn't know you didn't know?

Back to CS103!

Another TM Design

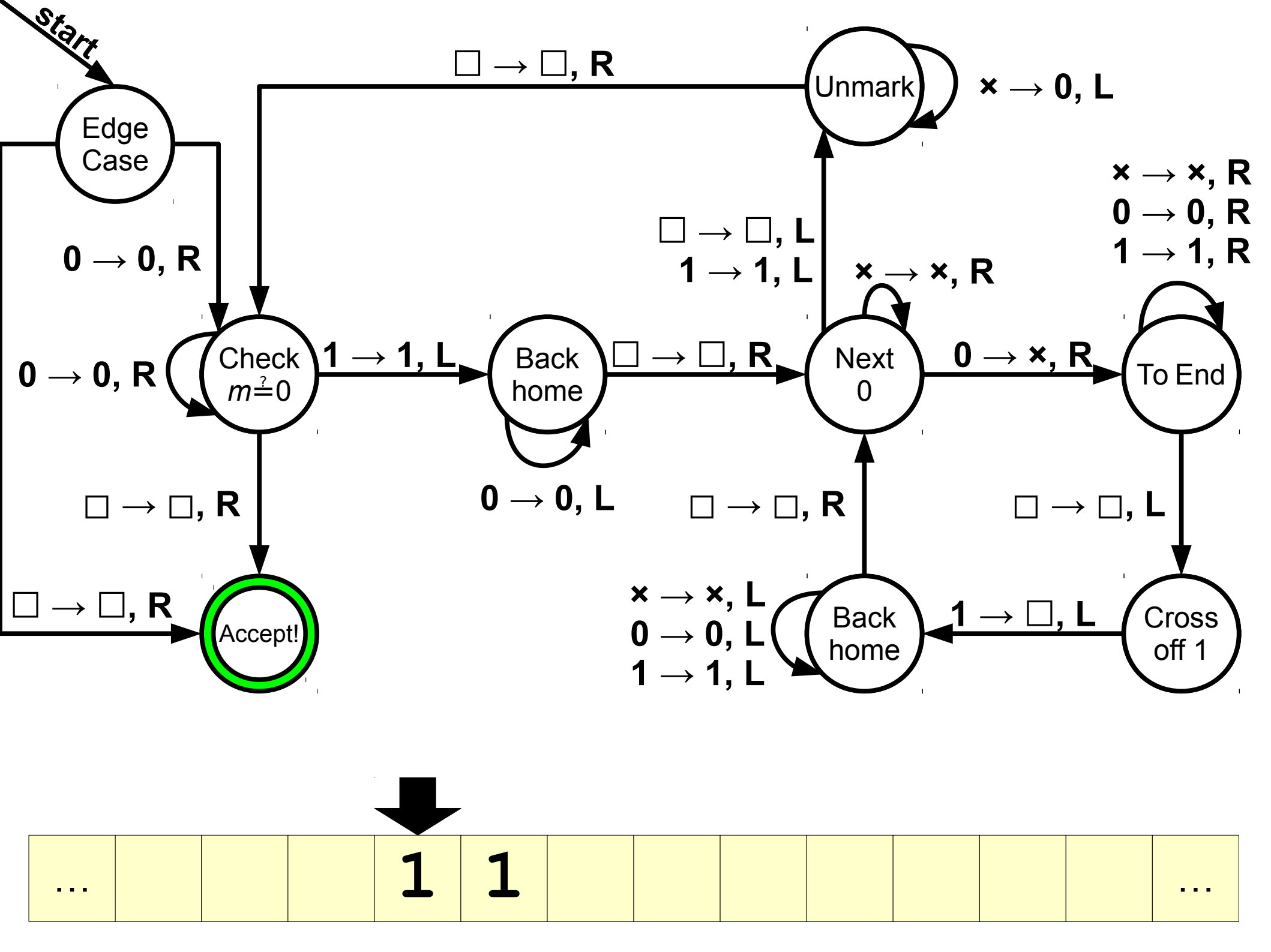
- Consider the following language over $\Sigma = \{\mathtt{0}, \mathtt{1}\}$:

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

- 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!)


start
Edge Case
Check $m \stackrel{?}{=} 0$
Back home
Unmark
Next 0
To End
Back home
Cross off 1
Accept!
□ → □, R
0 → 0, R
1 → 1, L
0 → 0, L
□ → □, L
1 → 1, L
× → ×, R
× → 0, L
0 → ×, R
× → ×, L
1 → □, L
1 1
…

Concepts from Today

- Turing machines are a generalization of finite automata equipped with an infinite tape.
- It's often helpful to think recursively when designing Turing machines.
- It's often helpful to introduce new symbols into the tape alphabet.
- Watch for edge cases that might lead to infinite loops – though we'll say more about that later on.

Next Time

- **TM Subroutines**
  - Combining multiple TMs together!
- **The Church-Turing Thesis**
  - Just how powerful *are* Turing machines?
- **Objects and Encodings**
  - Computing on graphs, images, etc.