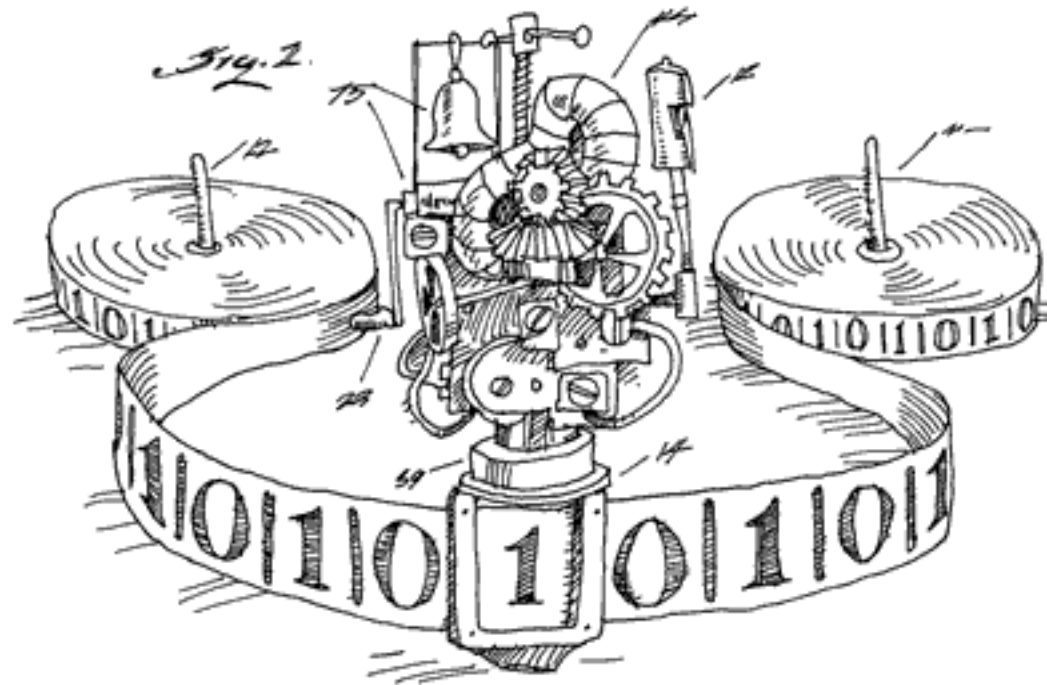


CSC4010: Introduction to Theoretical Computer Science



Lecture 3

Logistics

- Is everyone getting Blackboard emails?
- Part of Problem Set 1 is out, contains three problems. We will discuss in tutorial. You should attempt this, alone or in groups.
- Course webpage is also updated with topics to read from references. Please make sure to have a look.

Recap

1. **Computational Problems:** is a *task* which for each *valid input* produces one or more *outputs*.
2. **Two Simplifying Restrictions:**
 - Inputs are binary strings
 - Outputs are 0/1 or YES/NO.
3. **Decision Problems = Languages:**
 $L \subseteq \{0,1\}^*, x \in L \iff$ answer on x is YES.
4. **Cardinality of Languages:** The set $\{0,1\}^*$ is countable but, the set of all languages: $\mathbb{L} = \{L : L \subseteq \{0,1\}^*\}$, is uncountable.

Recap

Claim: The set of all languages: $\mathbb{L} = \{L : L \subseteq \{0,1\}^*\}$, is uncountable.

Recap

Claim: There exists a language L for which there is no python code P such that $P(x) = 1 \iff x \in L$.

Some Thoughts on the Proof

- There is nothing special about Python. You can replace it with a programming language or machine model of your choice.
- The key step: any algorithm has a finite description and we can therefore encode them as strings.
- **Objection 1:** The proof is **non-constructive**. What are the problems that cannot be solved by a python code? Are they interesting?
- **Objection 2:** The proof says: for every computational model there is a problem that cannot be solved by it. **Is there a problem that cannot be solved by all computational models simultaneously?**

Our Hero for This Lecture



Alan Turing (1912-1954)
*Founder of Computer Science
Mathematician, Philosopher,
Codebreaker,
World Class Marathoner*

- Designed the **Turing Machine**, foundation of the modern theory of computation and computability.
- Answered the **Decidability Question**.
- Contributed to breaking the German **Enigma code** during WWII.
- Designed in theory, one of the first **first computers (ACE)** and the **first programming languages (ACI)**.
- His best marathon time of 2:46:03 was only 11 minutes slower than the 1948 Olympic champion.



Computation

- Computers: modern technology?
- But **computation** is ancient.
- Can be performed by:
 - A human being with a pen and a pencil.
 - A smart-phone.
 - A supercomputer.
- We want a mathematical model that **describes all of these** and **transcends any one technology**.



Computation

- Note: Humans do the same computation that (smartphones/laptops do)
 - Less quickly and less reliably
- Consequence: we can do computation without understanding electronics.
- Computation is a familiar, everyday, **human** act.



Turing Machine

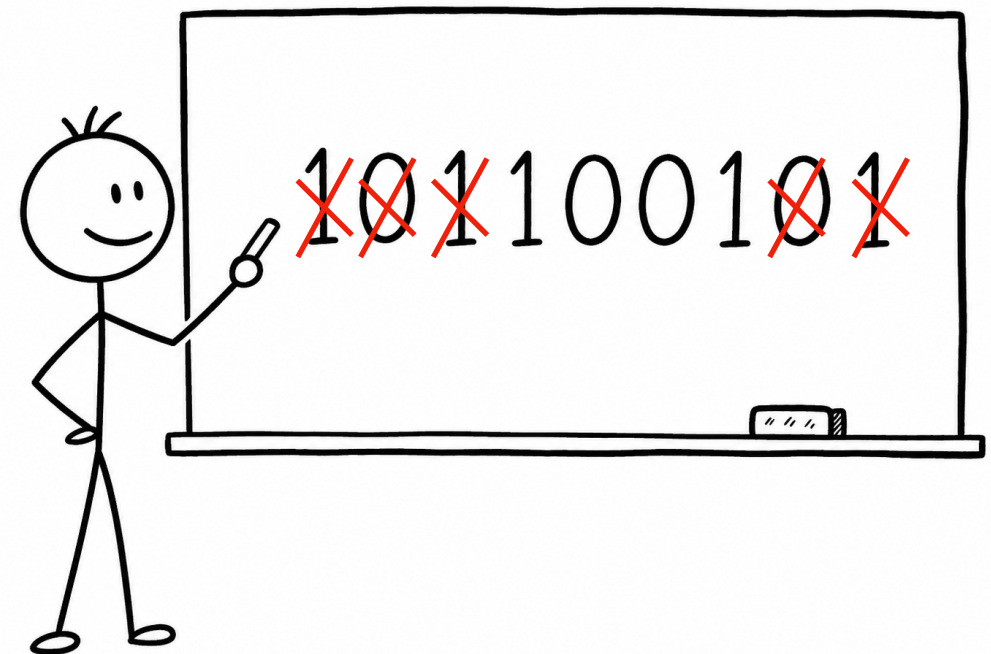
- We need a precise mathematical meaning of the term “algorithm”.
- The model should not depend on a particular program, programming language, or technology. Should capture the essence of computation.
- Turing Machines captures computation using a finite set of simple rules that allow for one small operation at a time.

Example: Palindromes

- Suppose a long string of bits is written on a blackboard.
- Our job: Figure out whether the string is a “palindrome,” i.e., whether it is the same forwards and backwards.
- What should we do?

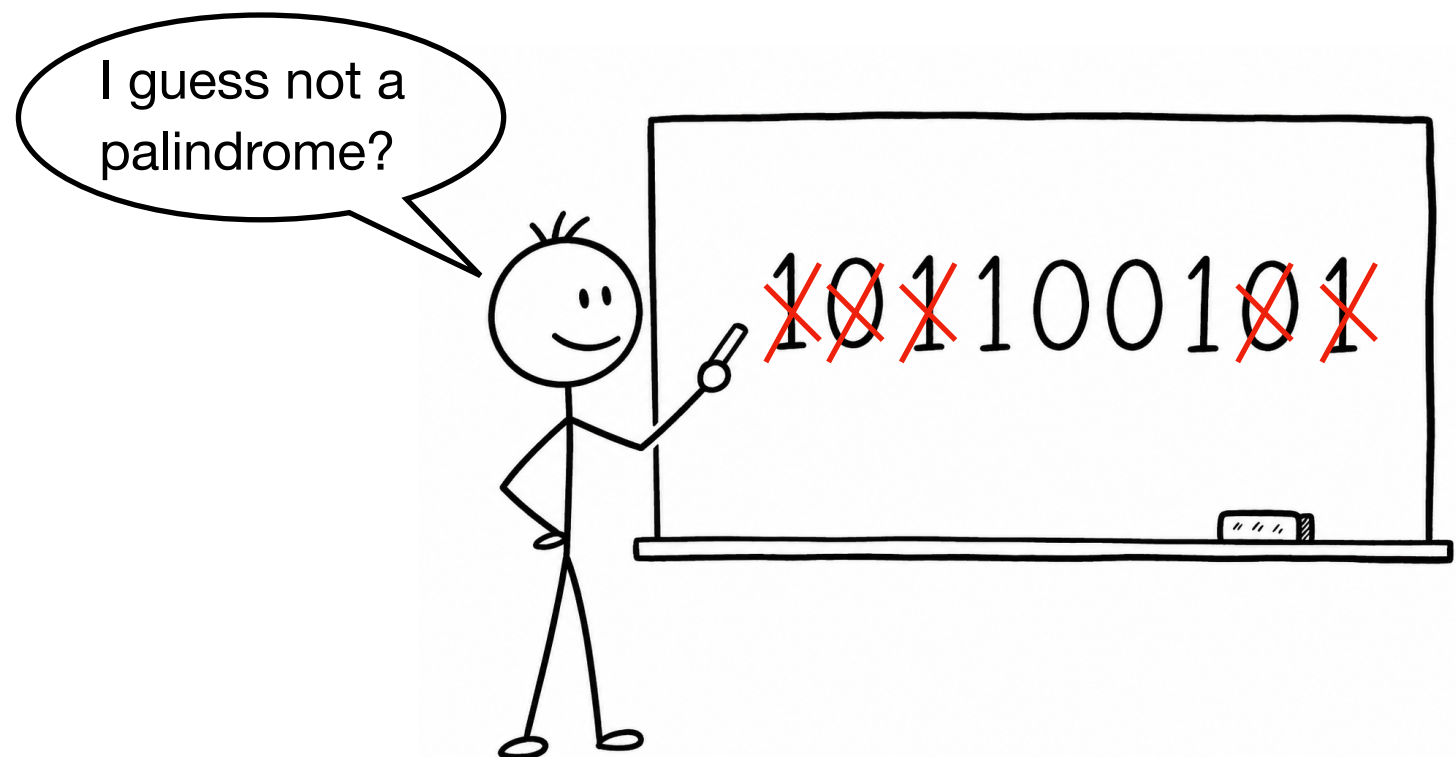
Example: Palindromes

- Idea: Compare and cross off the first and last symbols.
- Repeat until there is a mismatch or everything is crossed off.



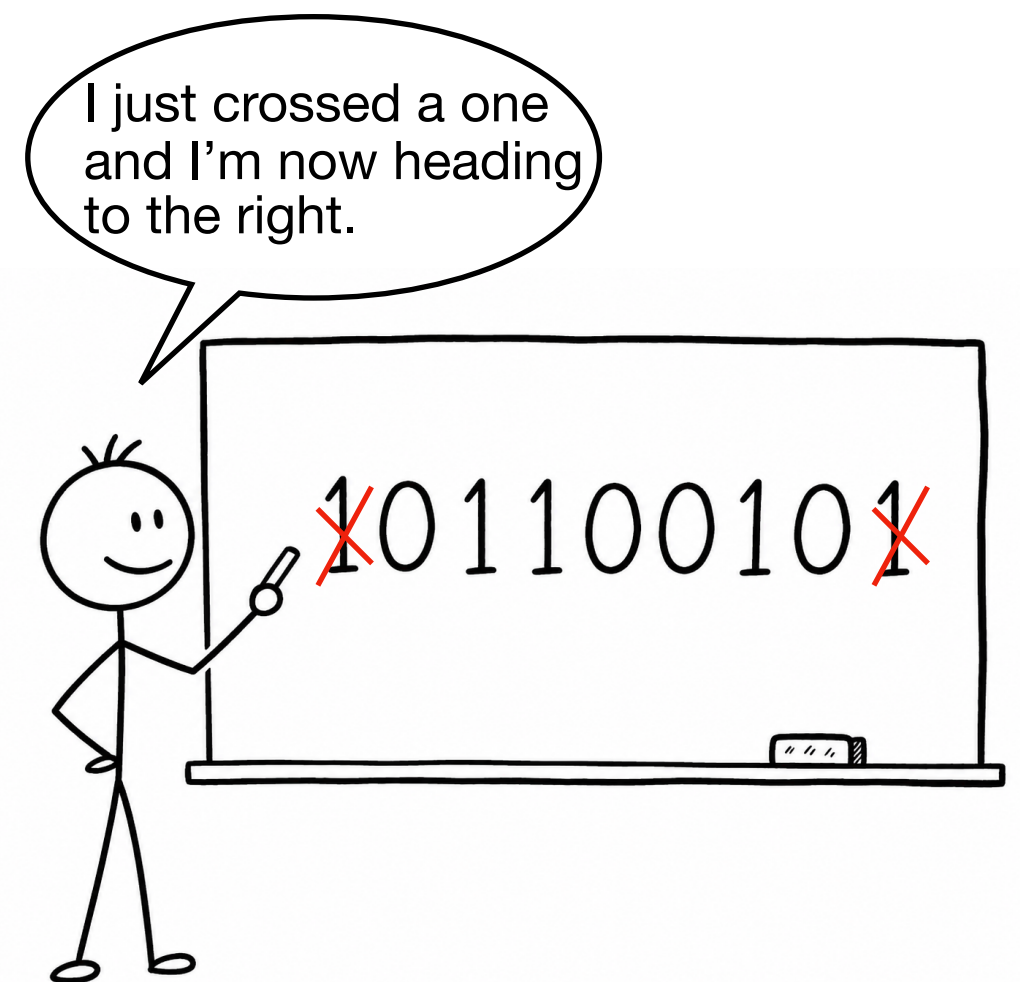
Example: Palindromes

- Idea: Compare and cross off the first and last symbols.
- Repeat until there is a mismatch or everything is crossed off.

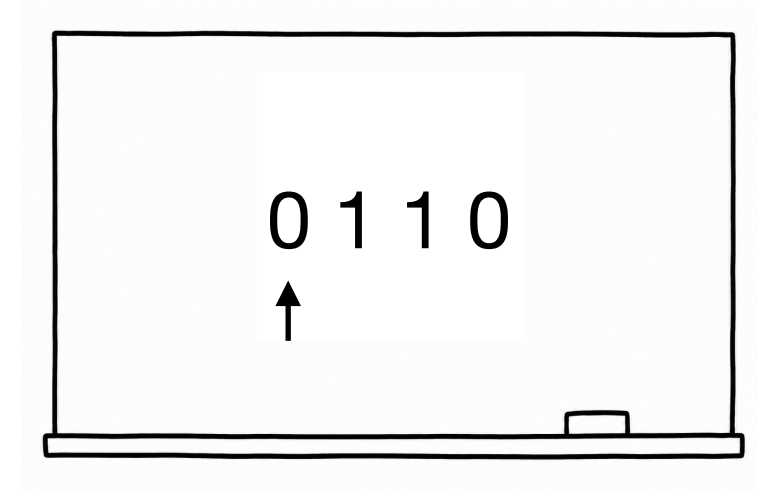


Local Decisions

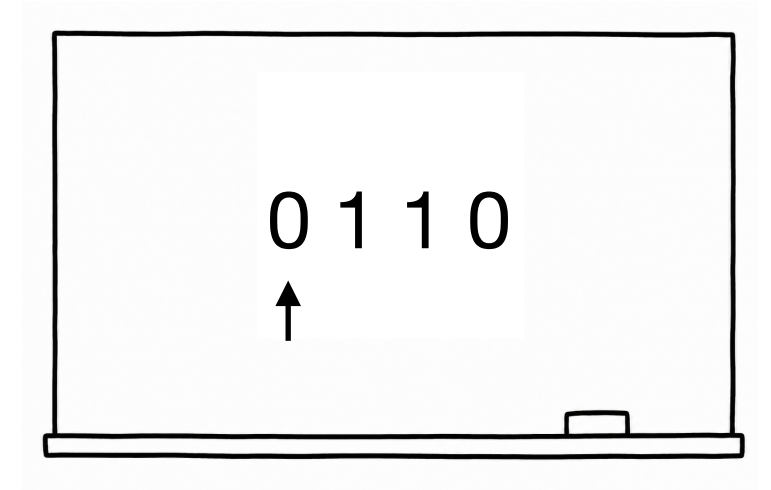
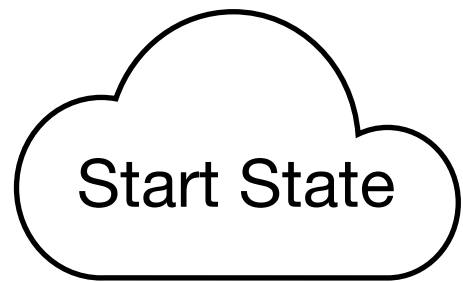
- In each step, how do we know what to do next?
 1. We keep track of some information (“**state**”) in our head.
 2. We look at the **local contents** of the blackboard. (One symbol is sufficient.)
 3. We can describe the algorithm using a “**state diagram**”



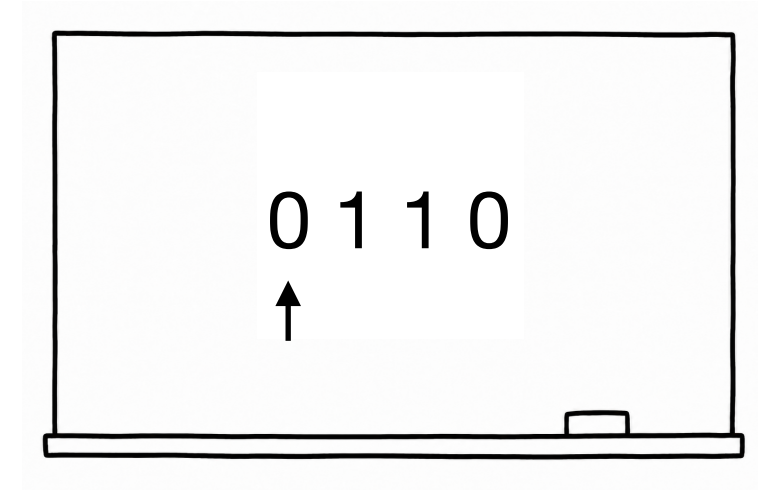
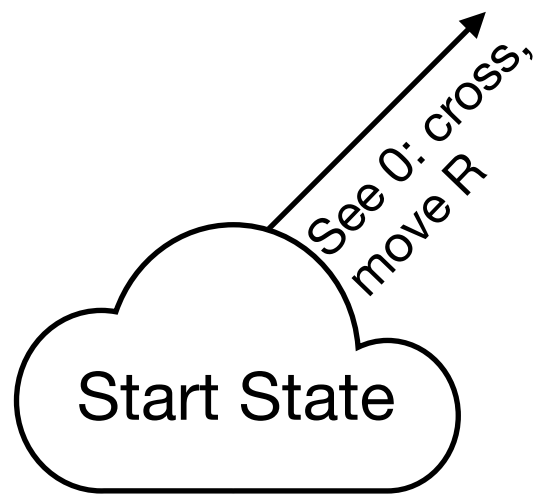
State Diagram



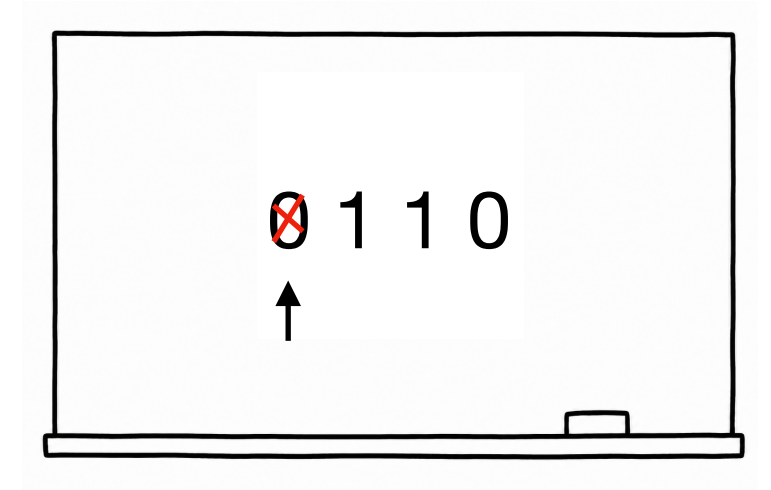
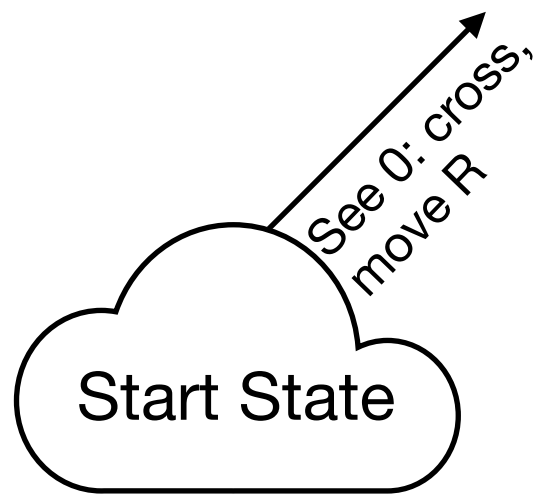
State Diagram



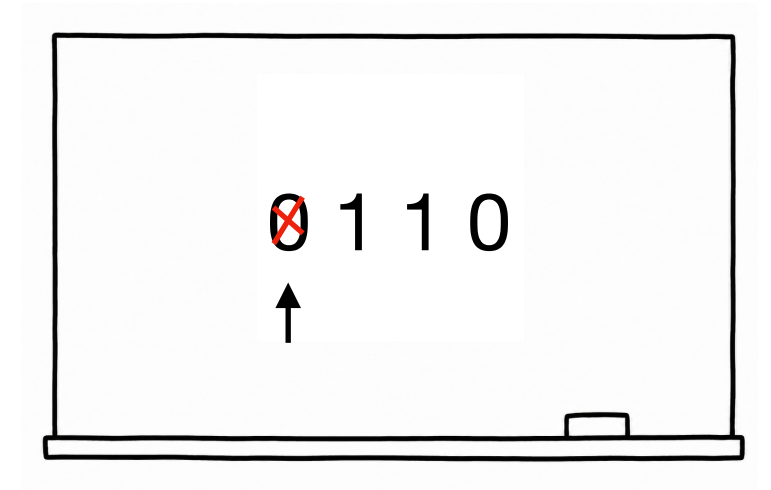
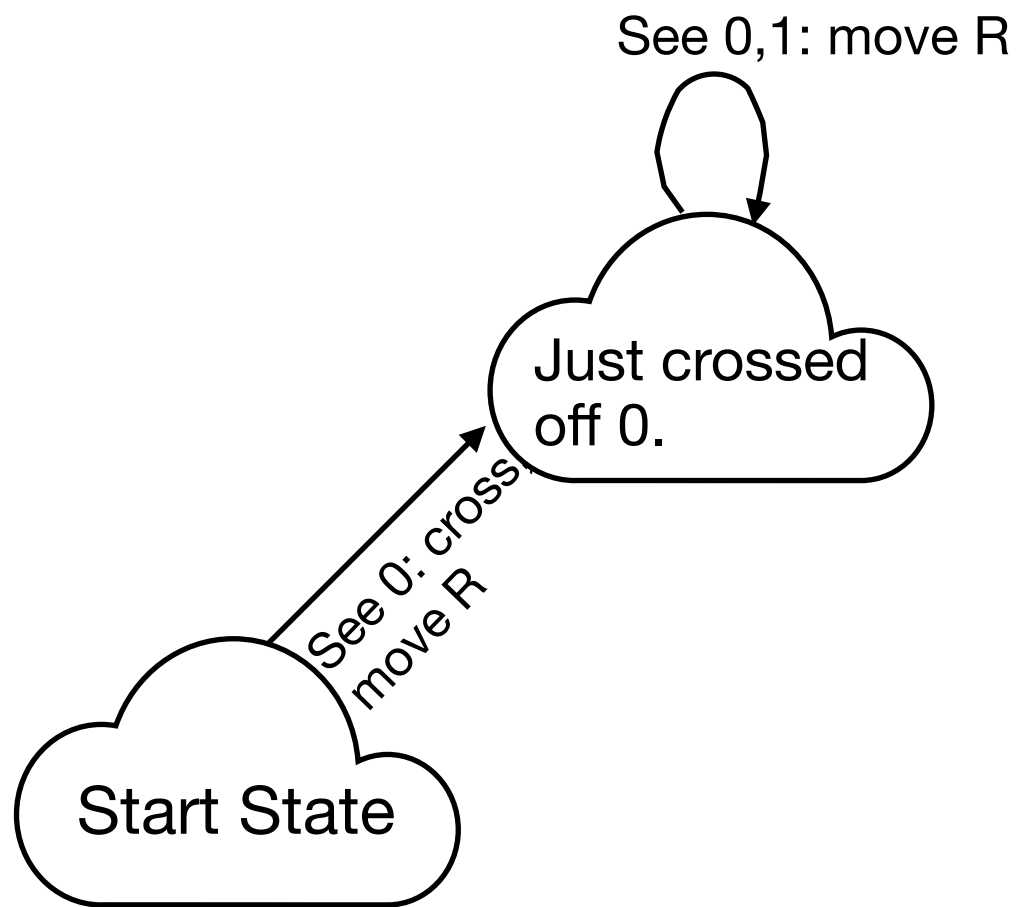
State Diagram



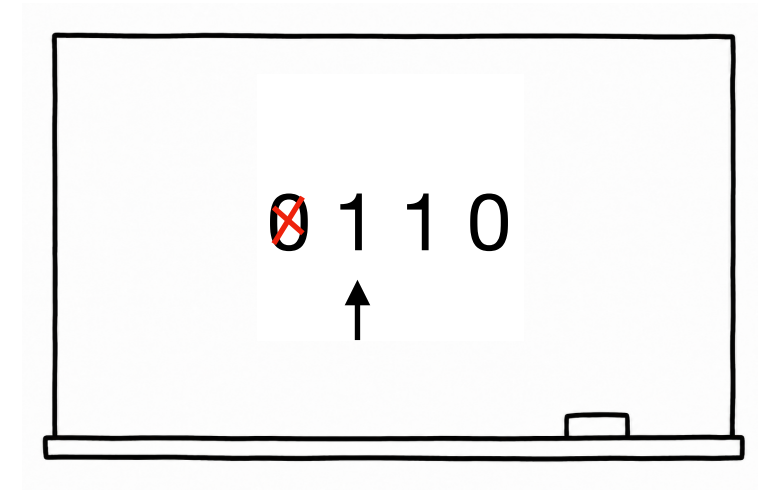
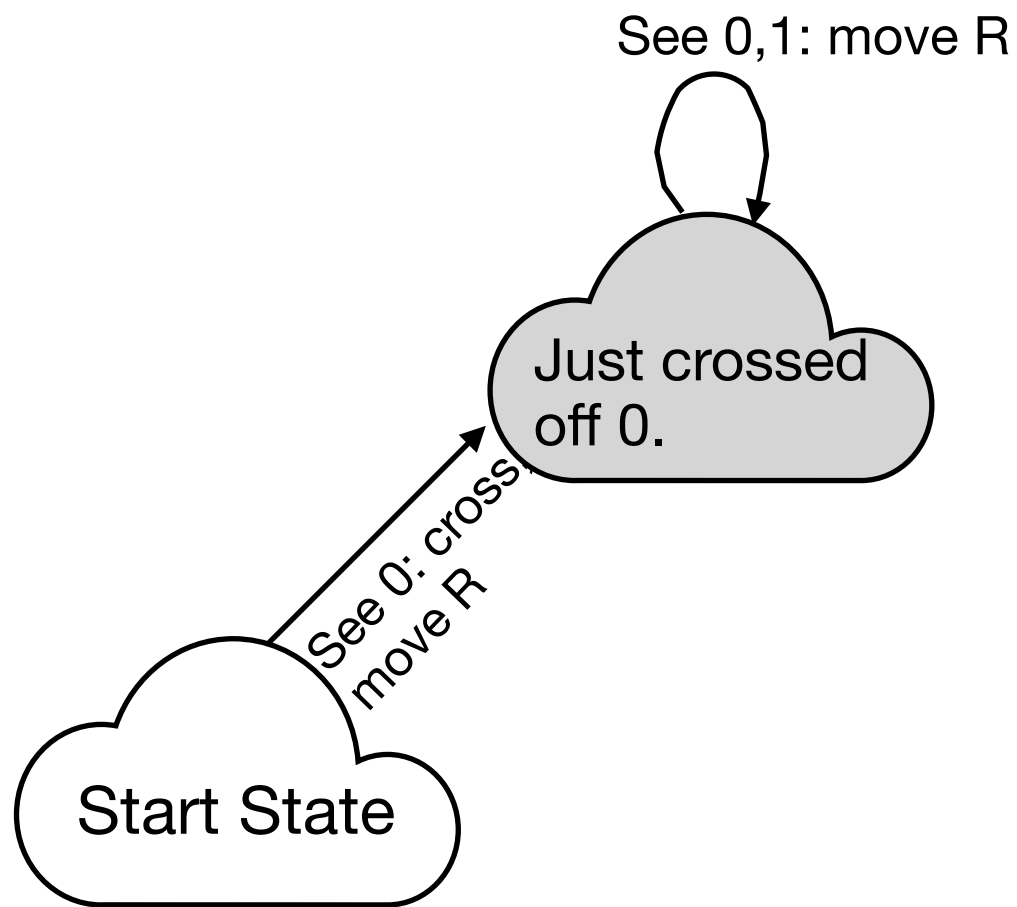
State Diagram



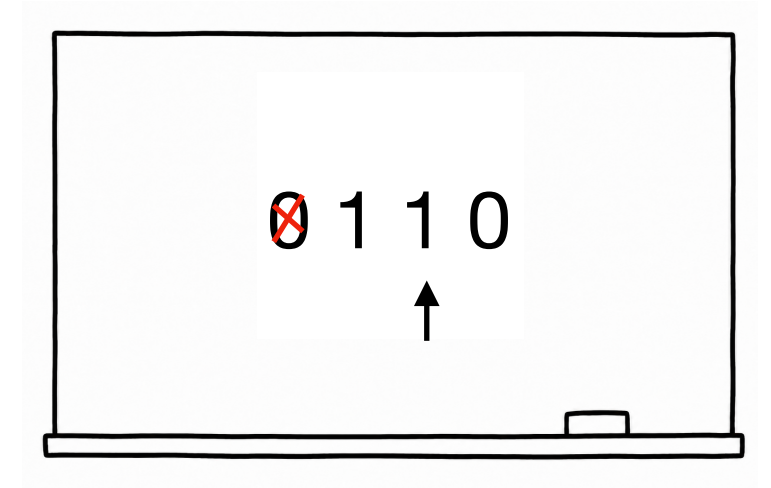
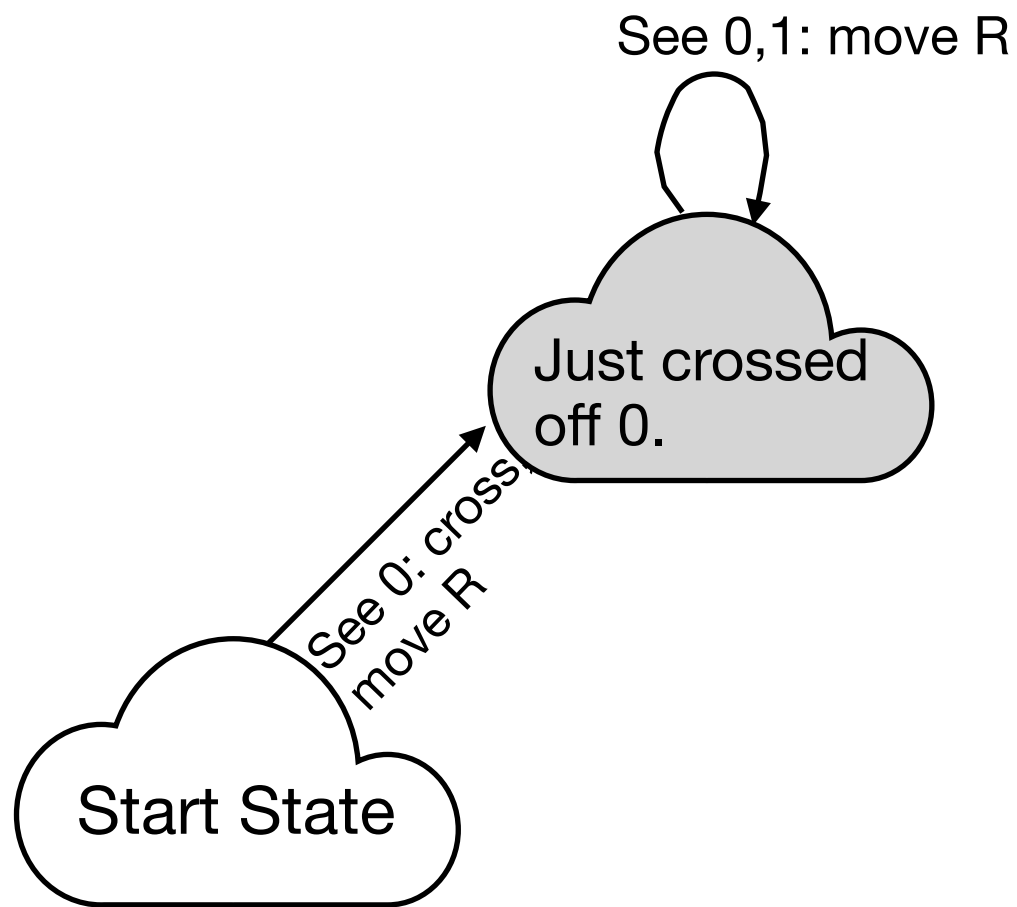
State Diagram



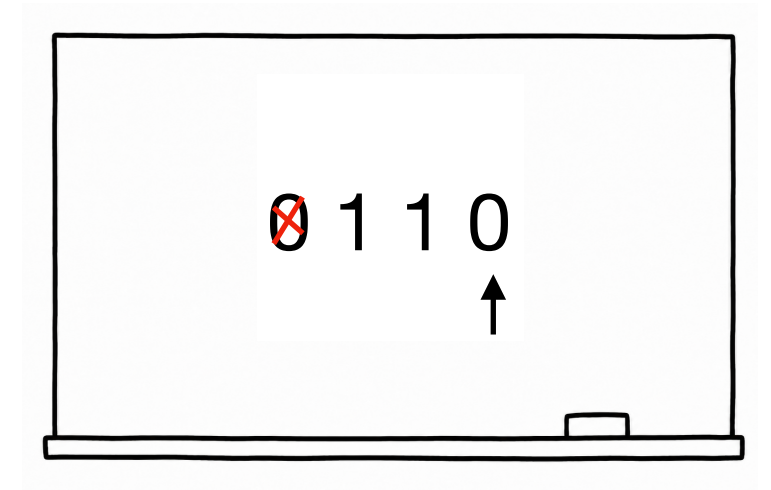
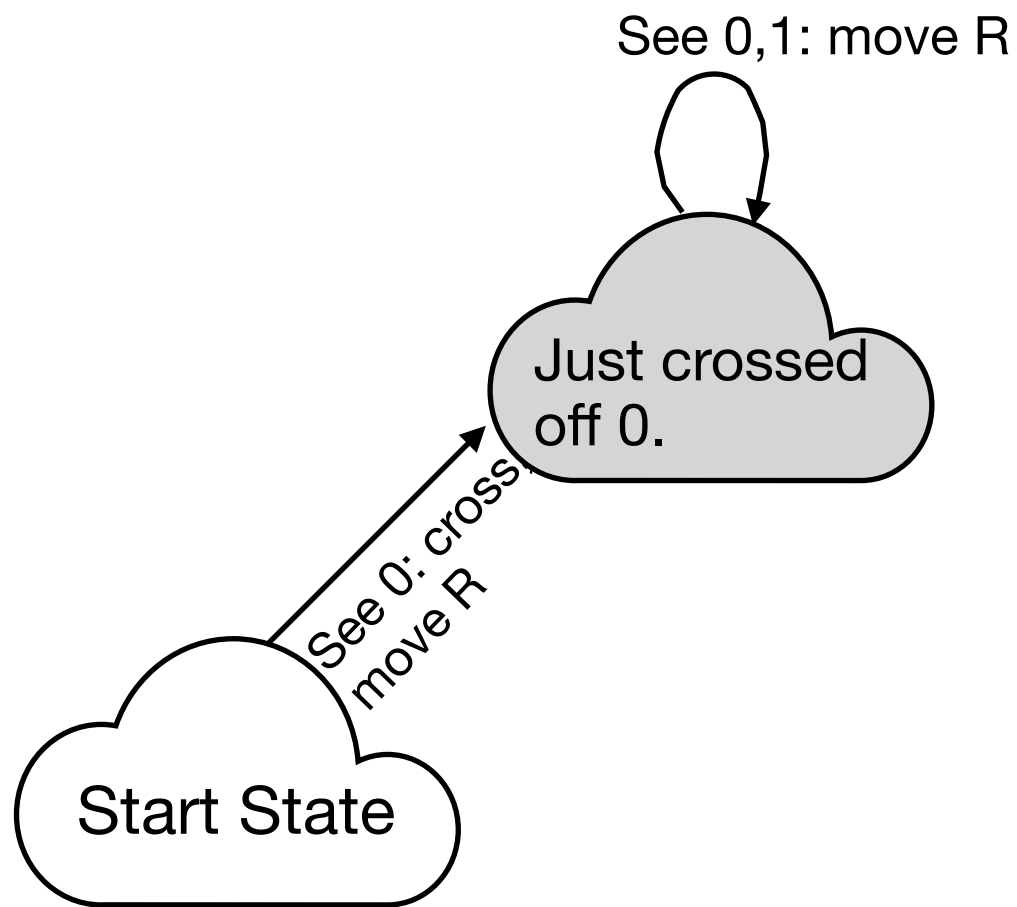
State Diagram



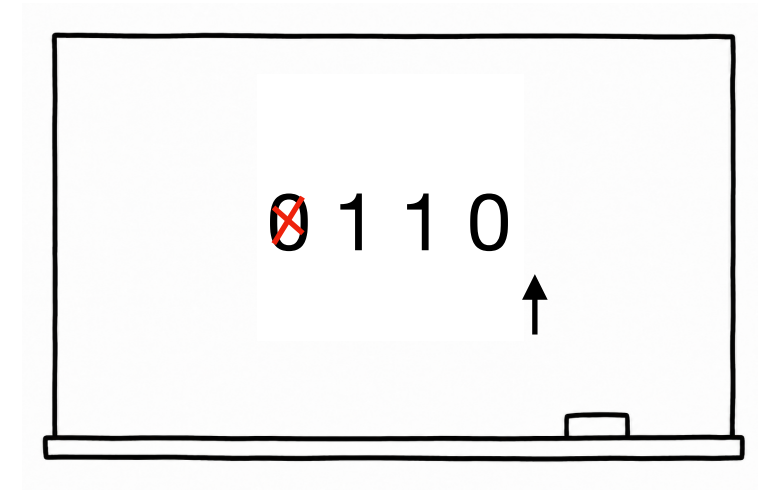
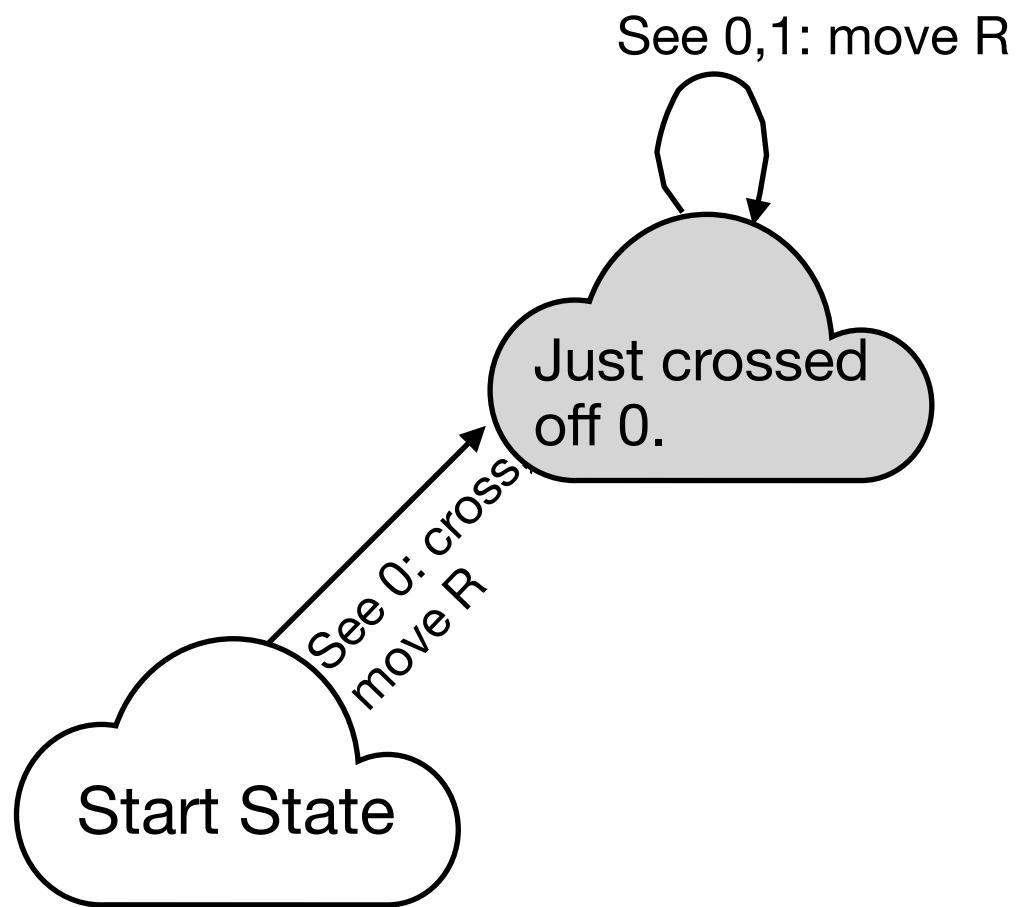
State Diagram



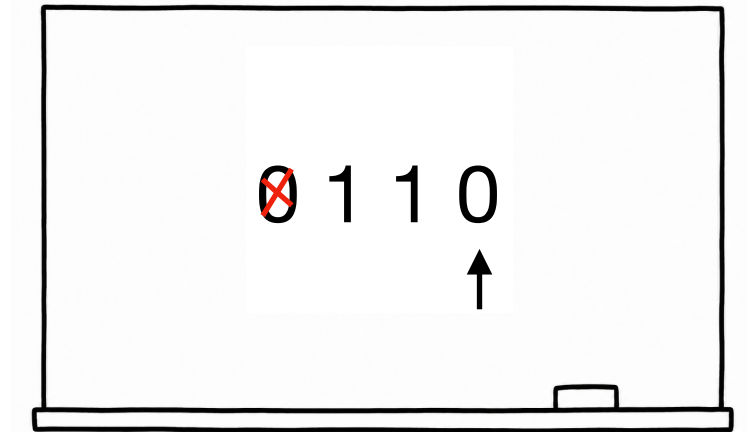
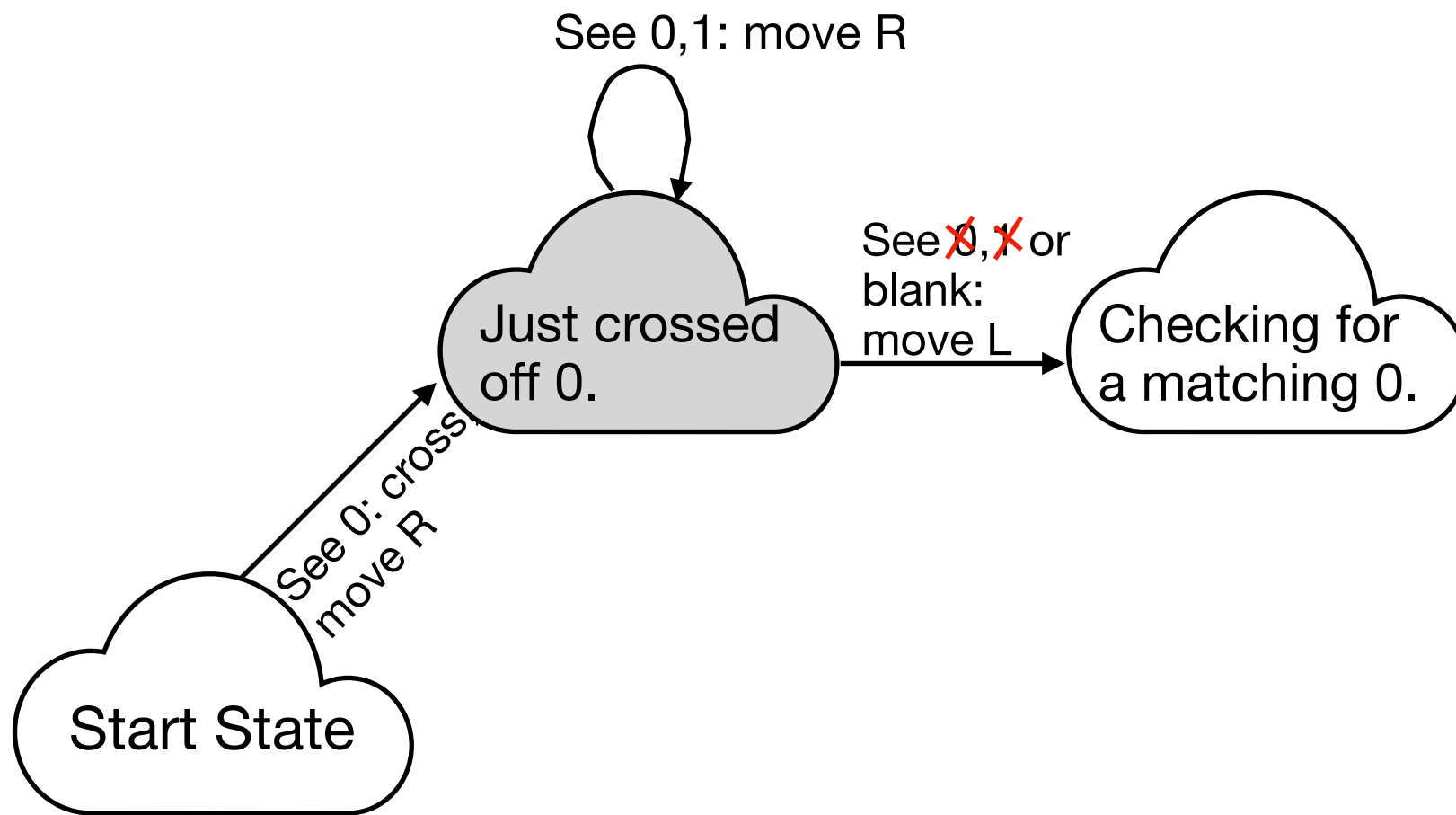
State Diagram



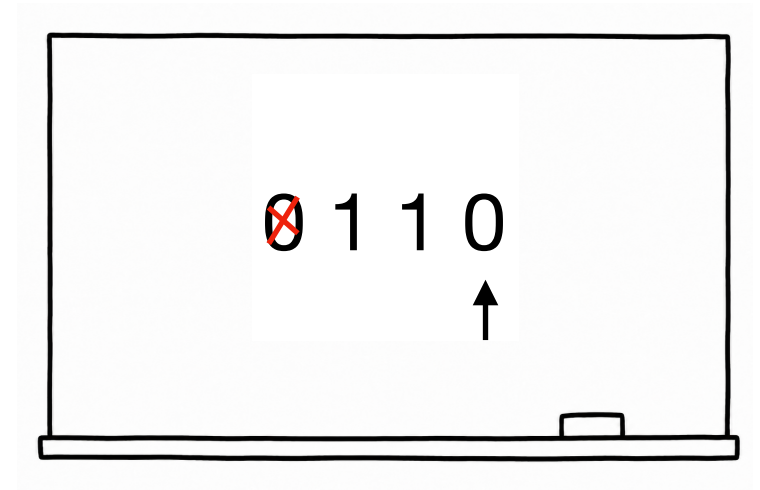
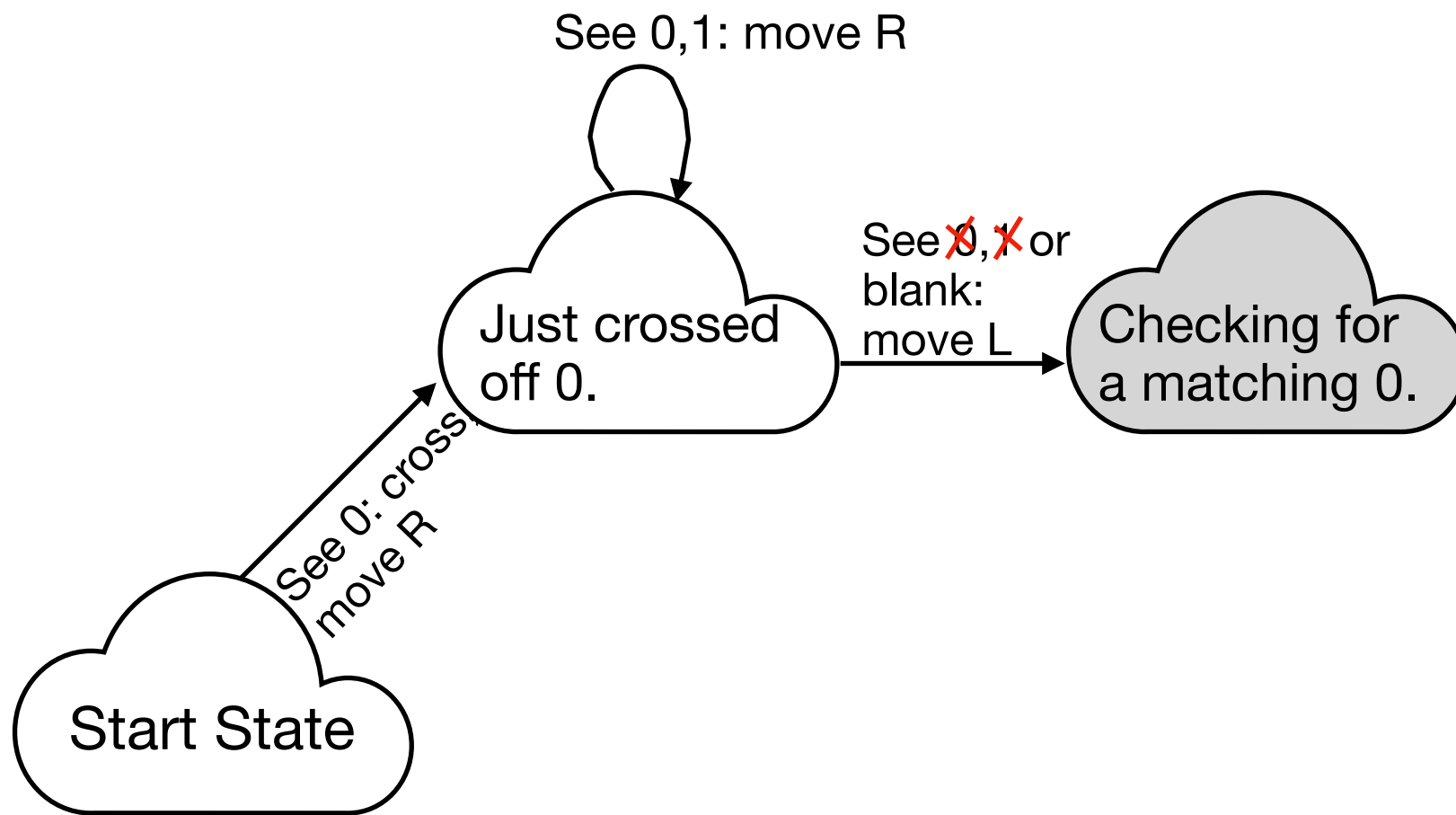
State Diagram



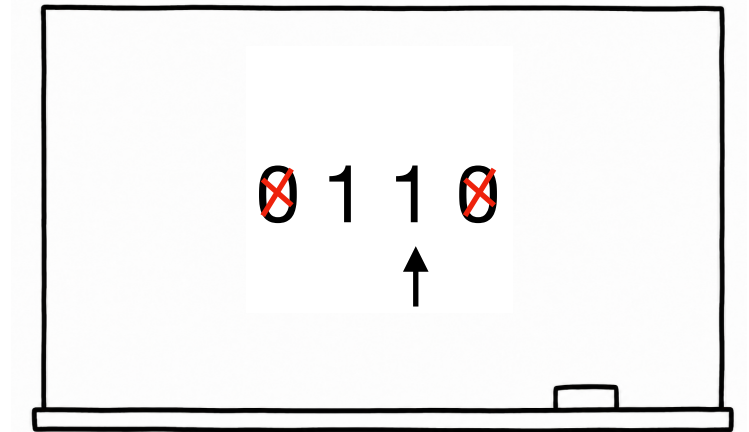
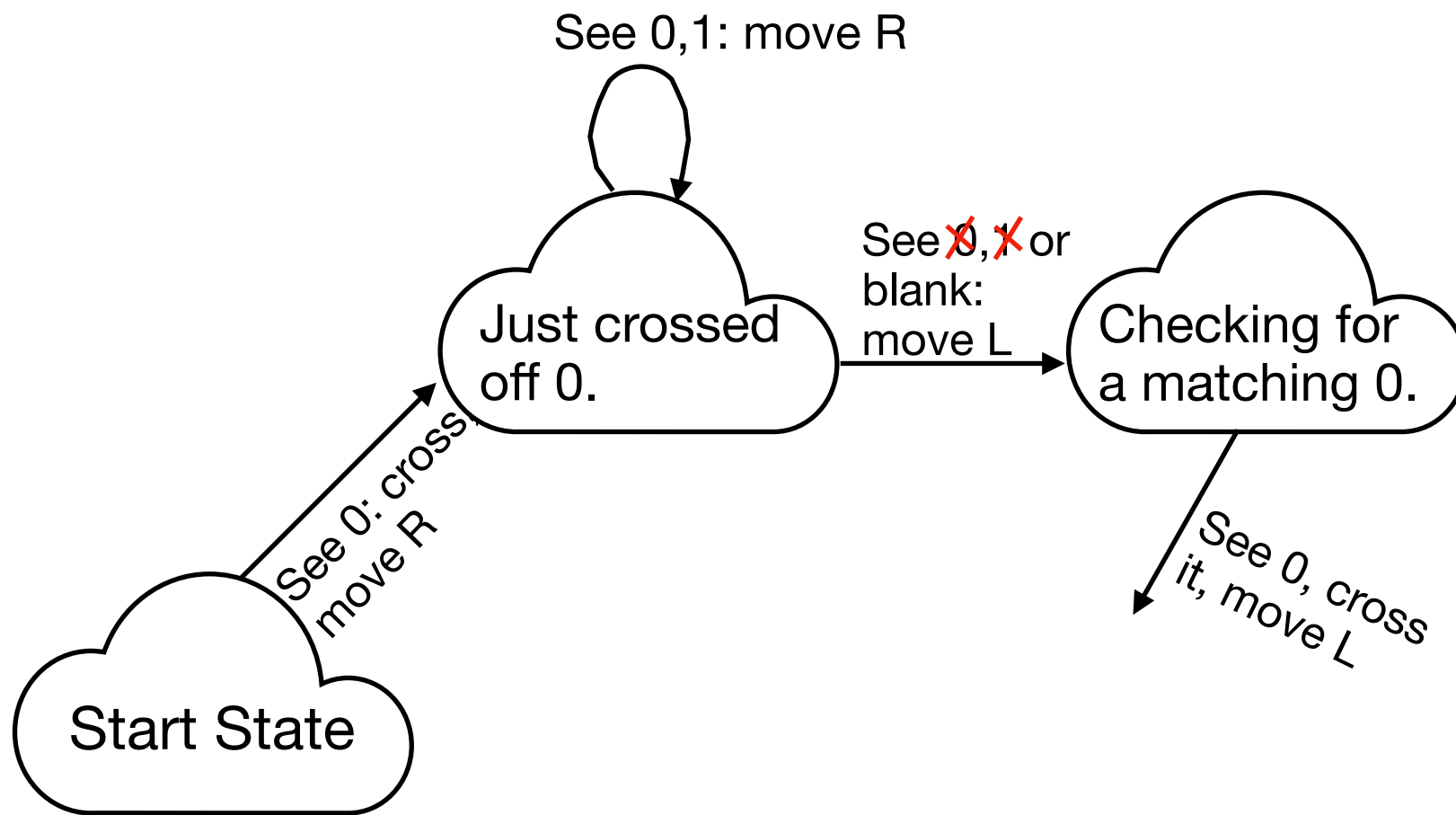
State Diagram



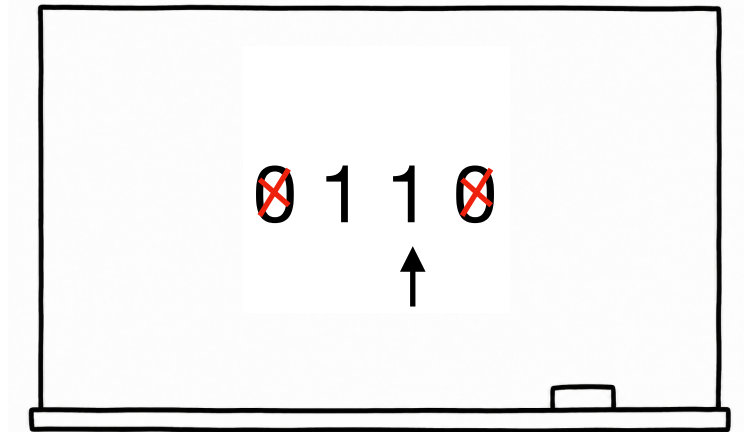
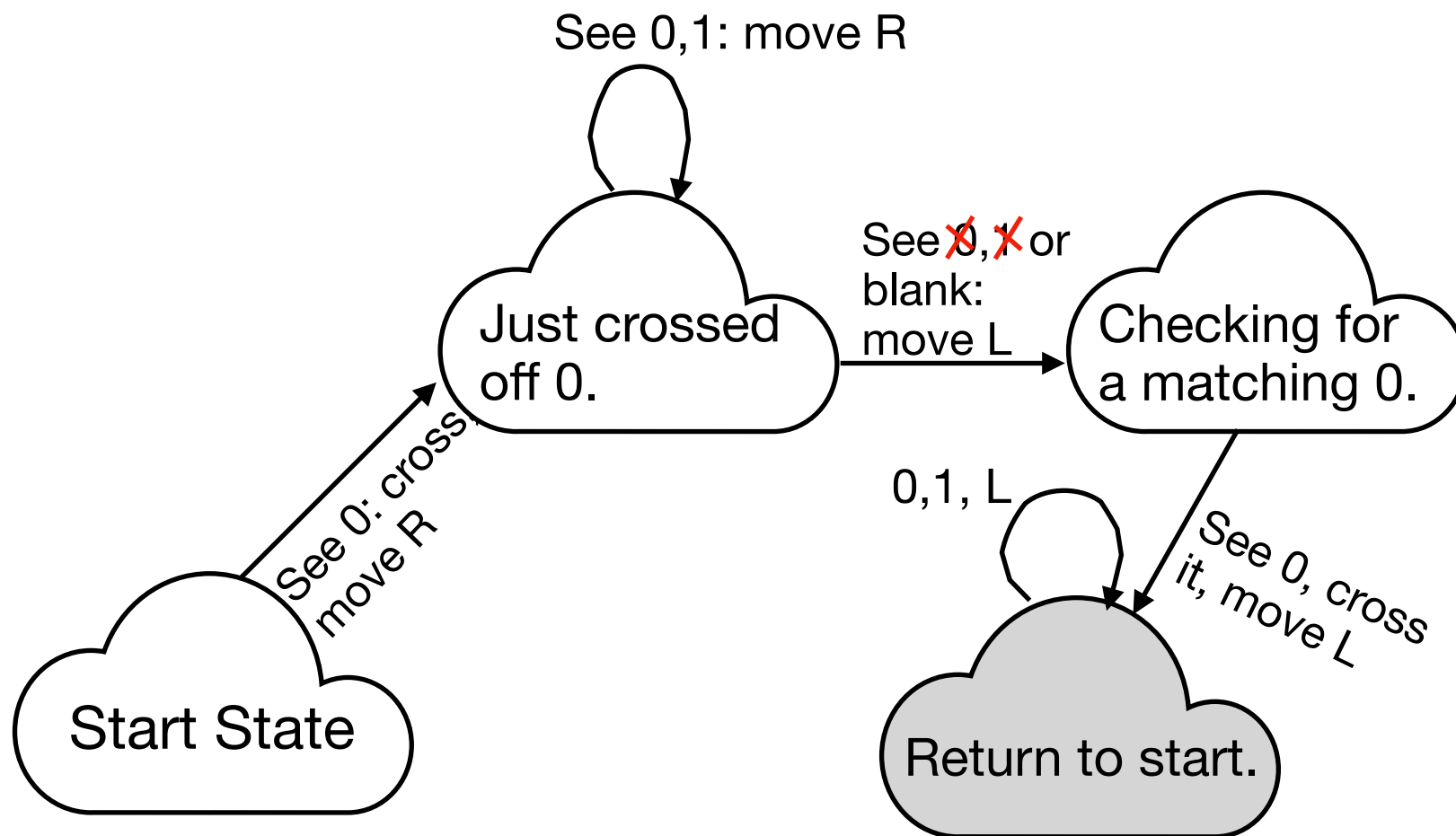
State Diagram



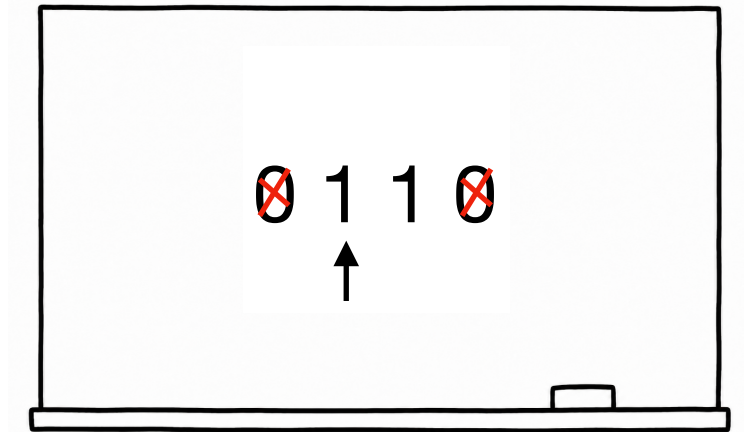
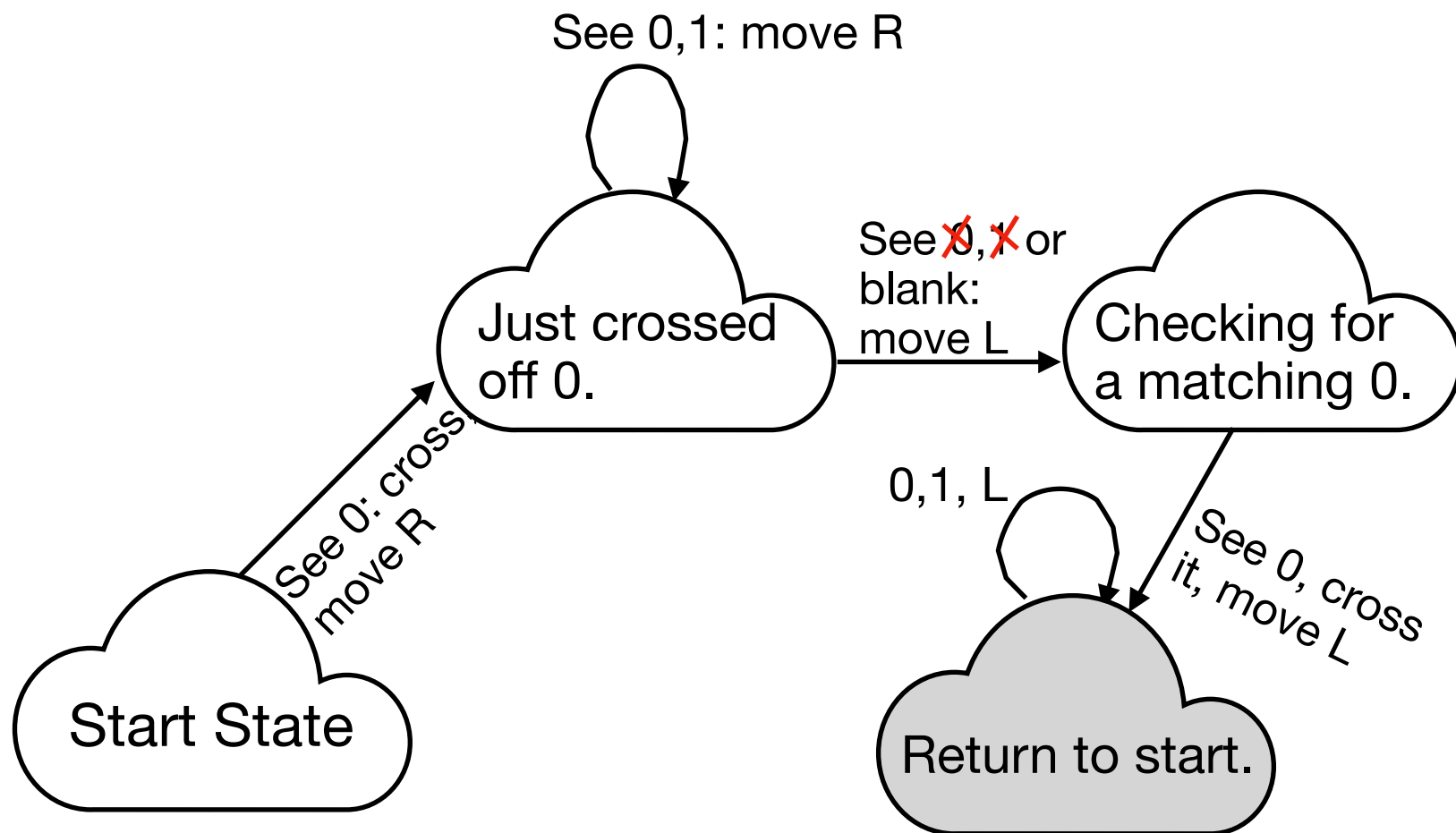
State Diagram



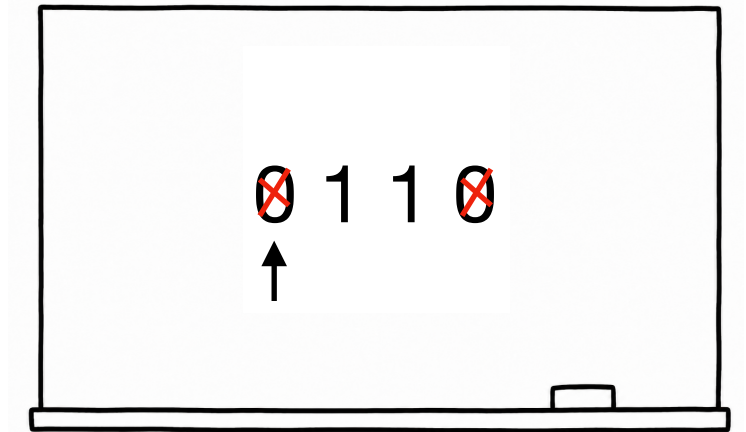
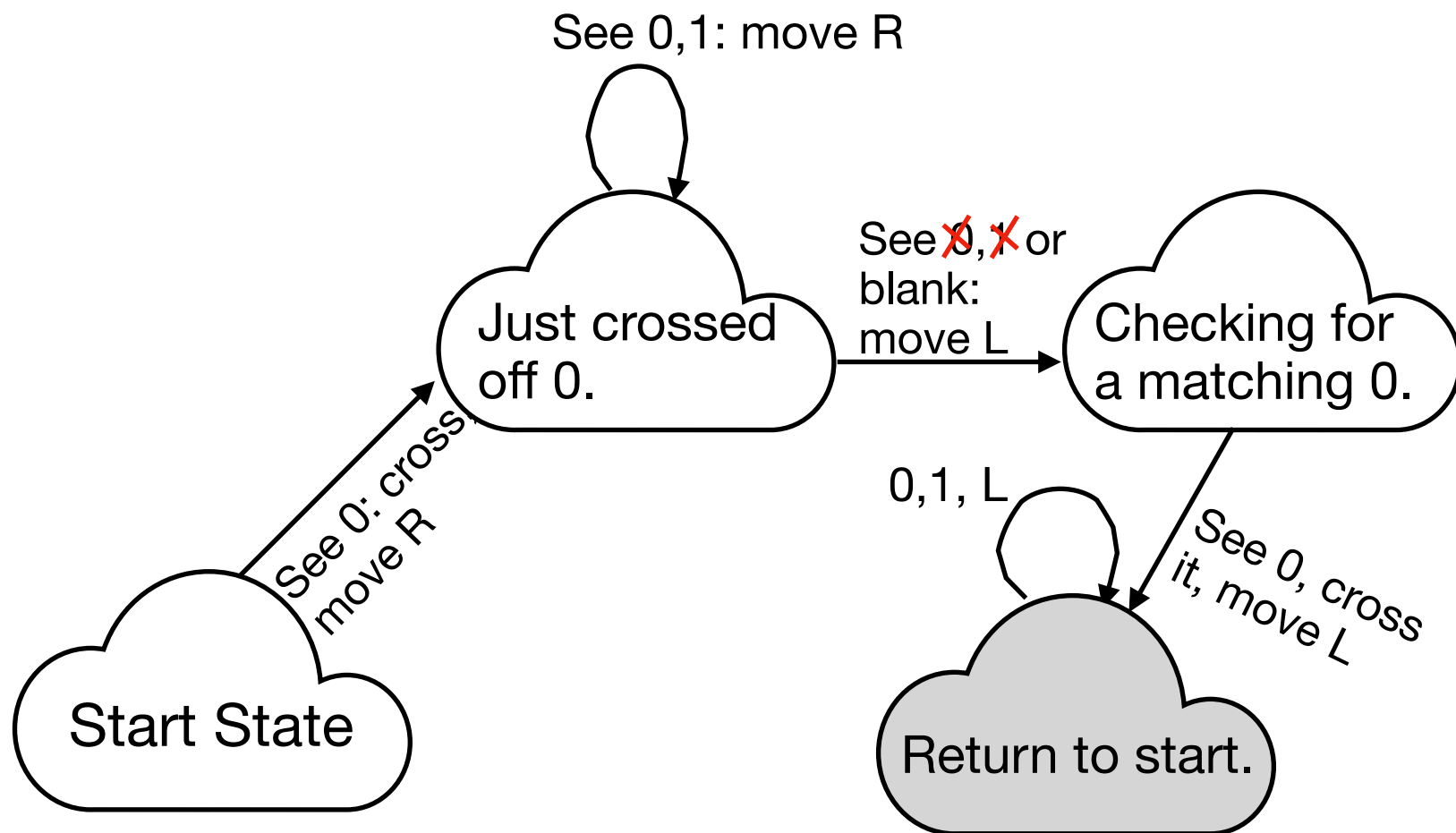
State Diagram



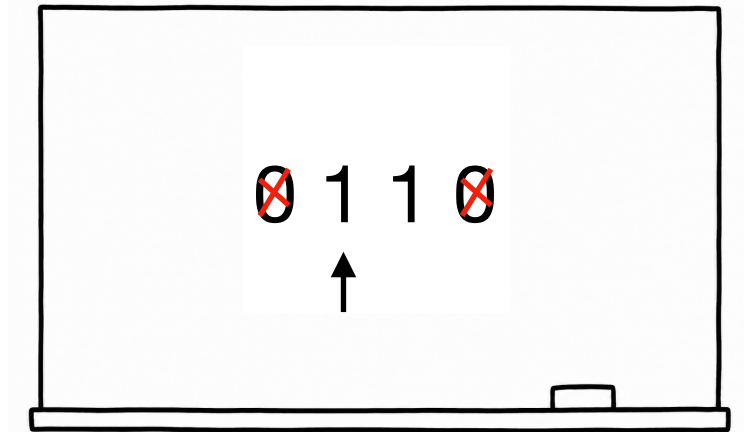
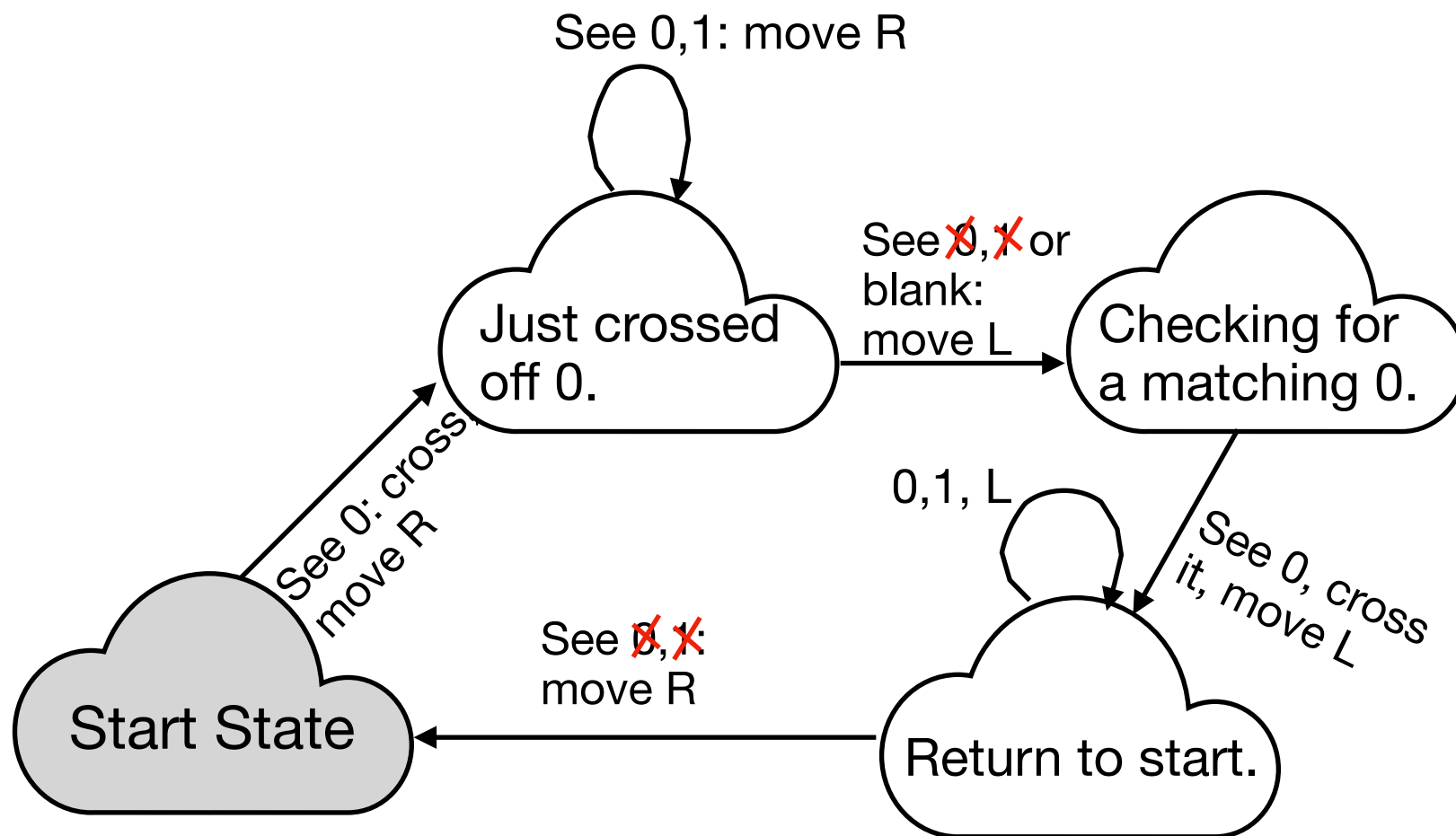
State Diagram



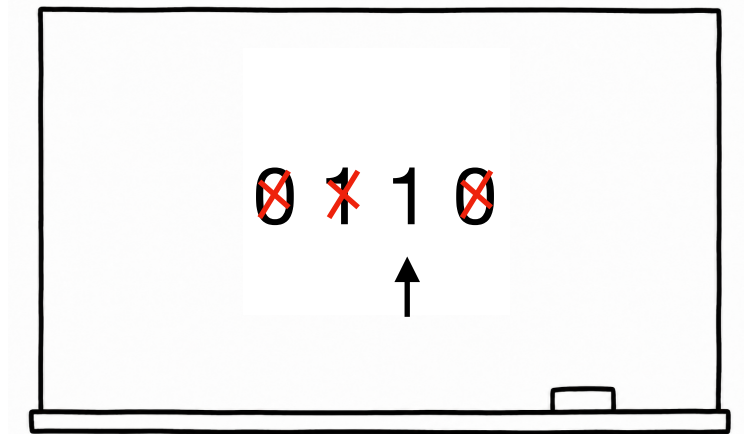
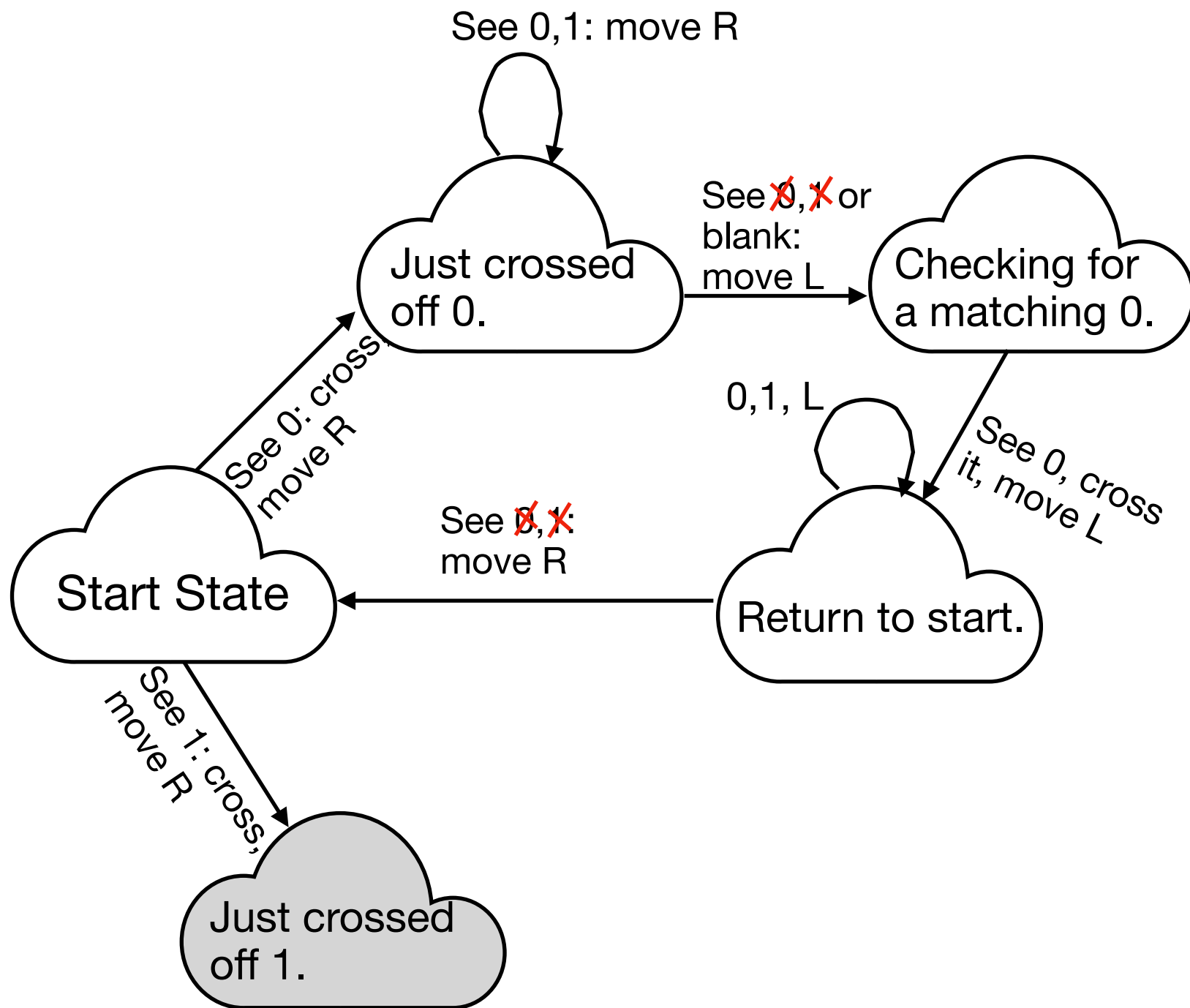
State Diagram



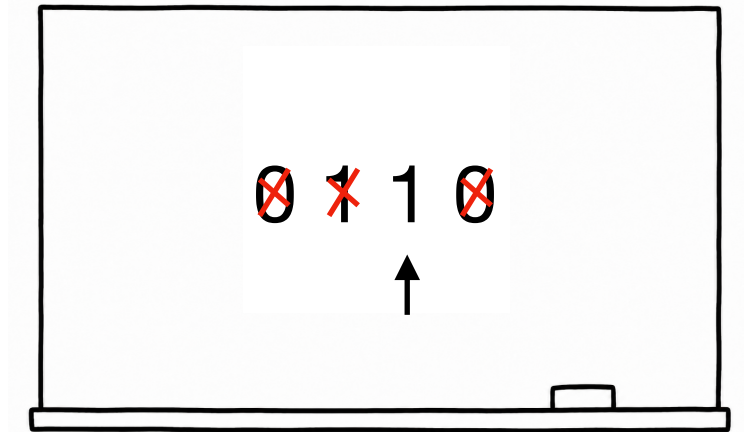
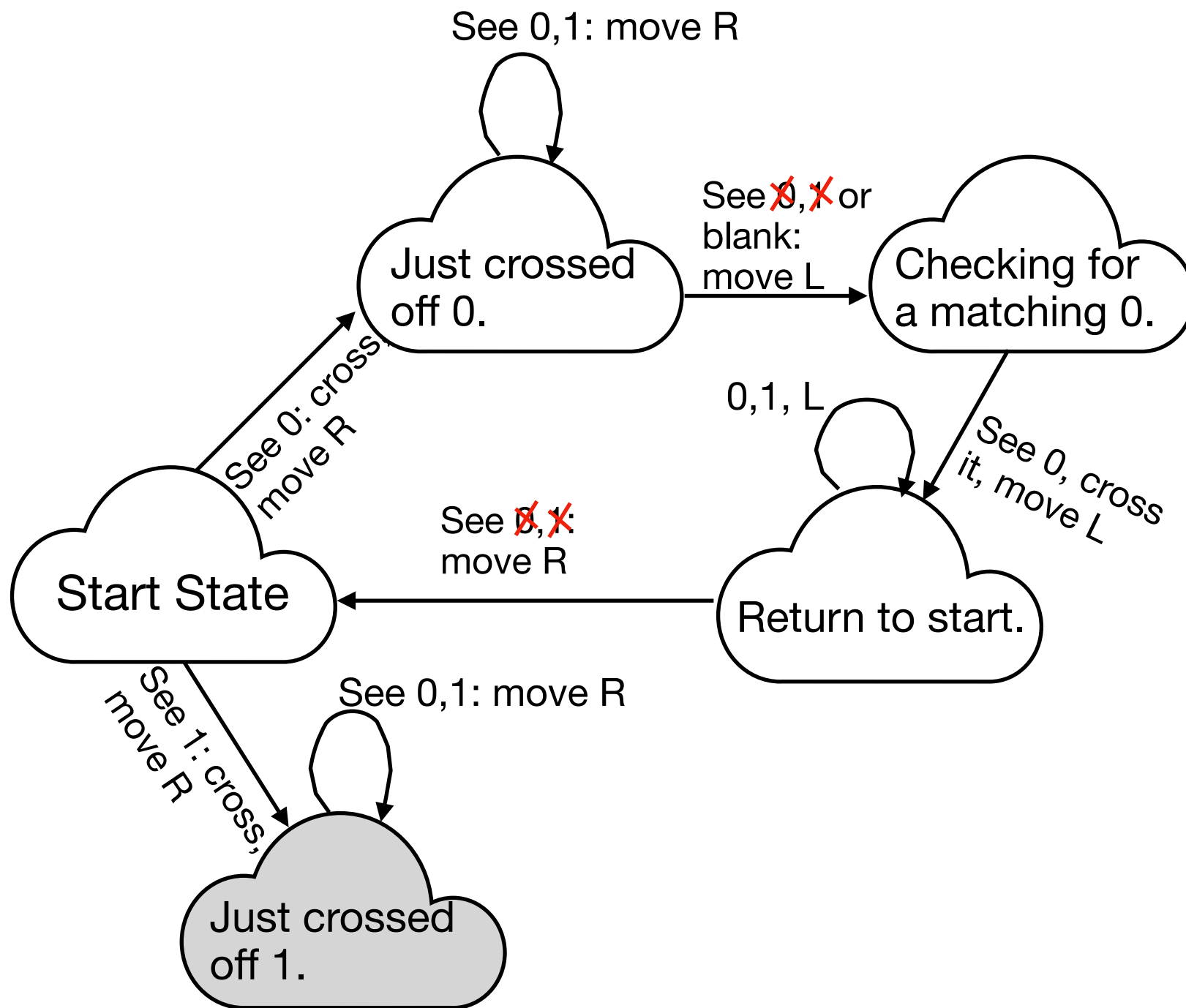
State Diagram



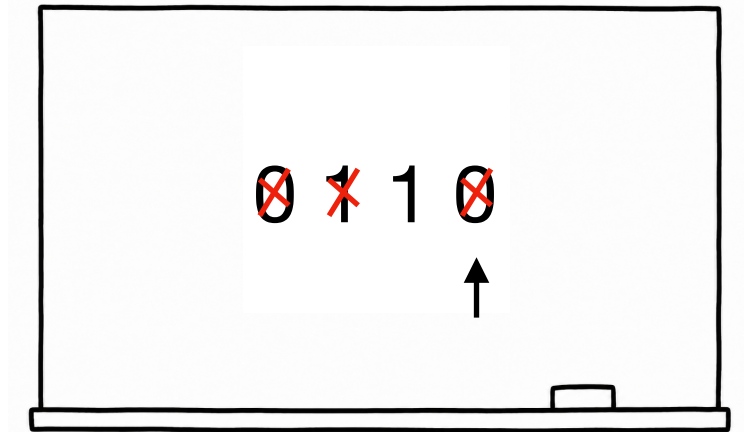
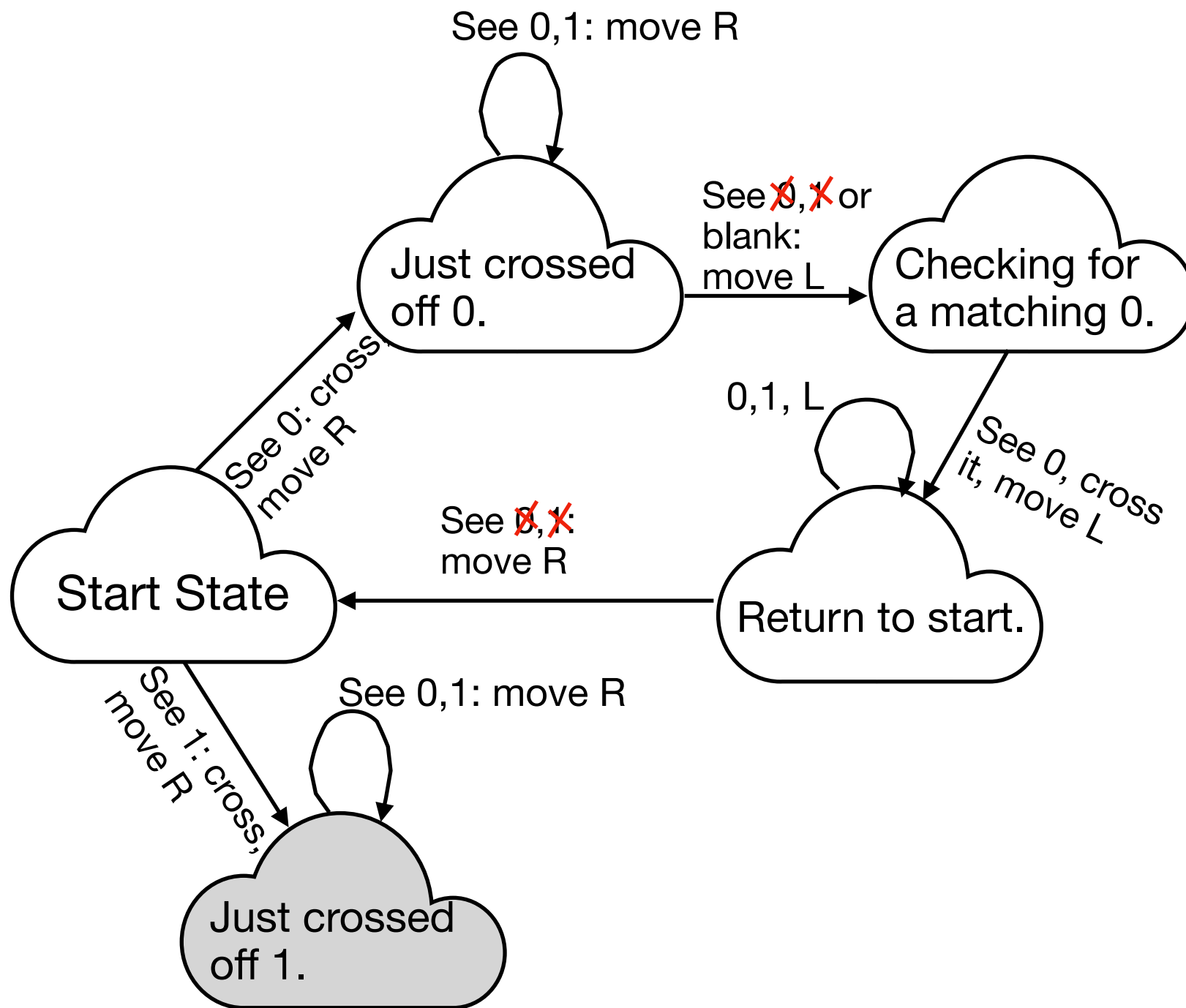
State Diagram



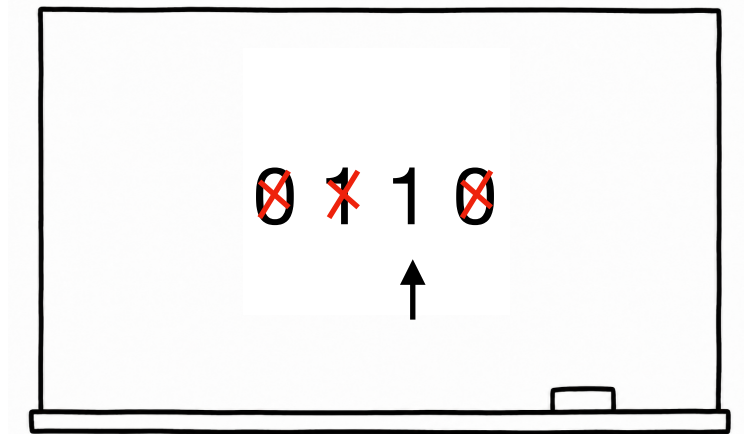
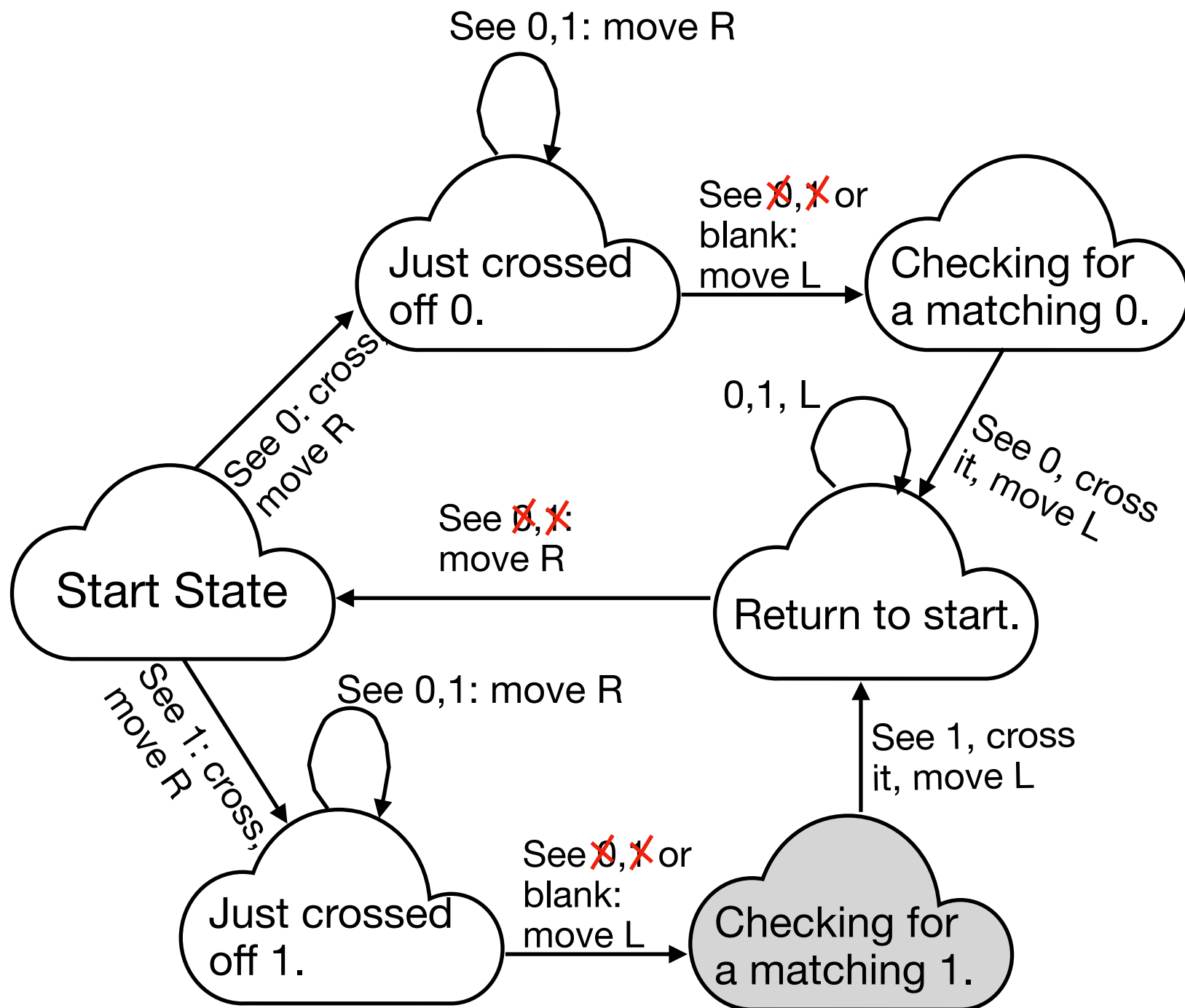
State Diagram



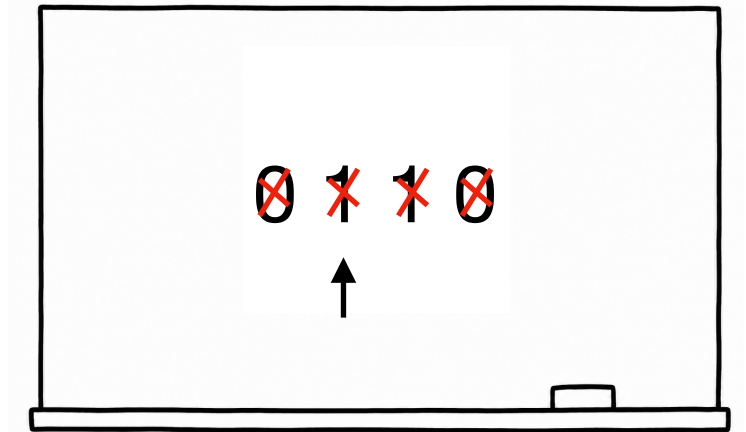
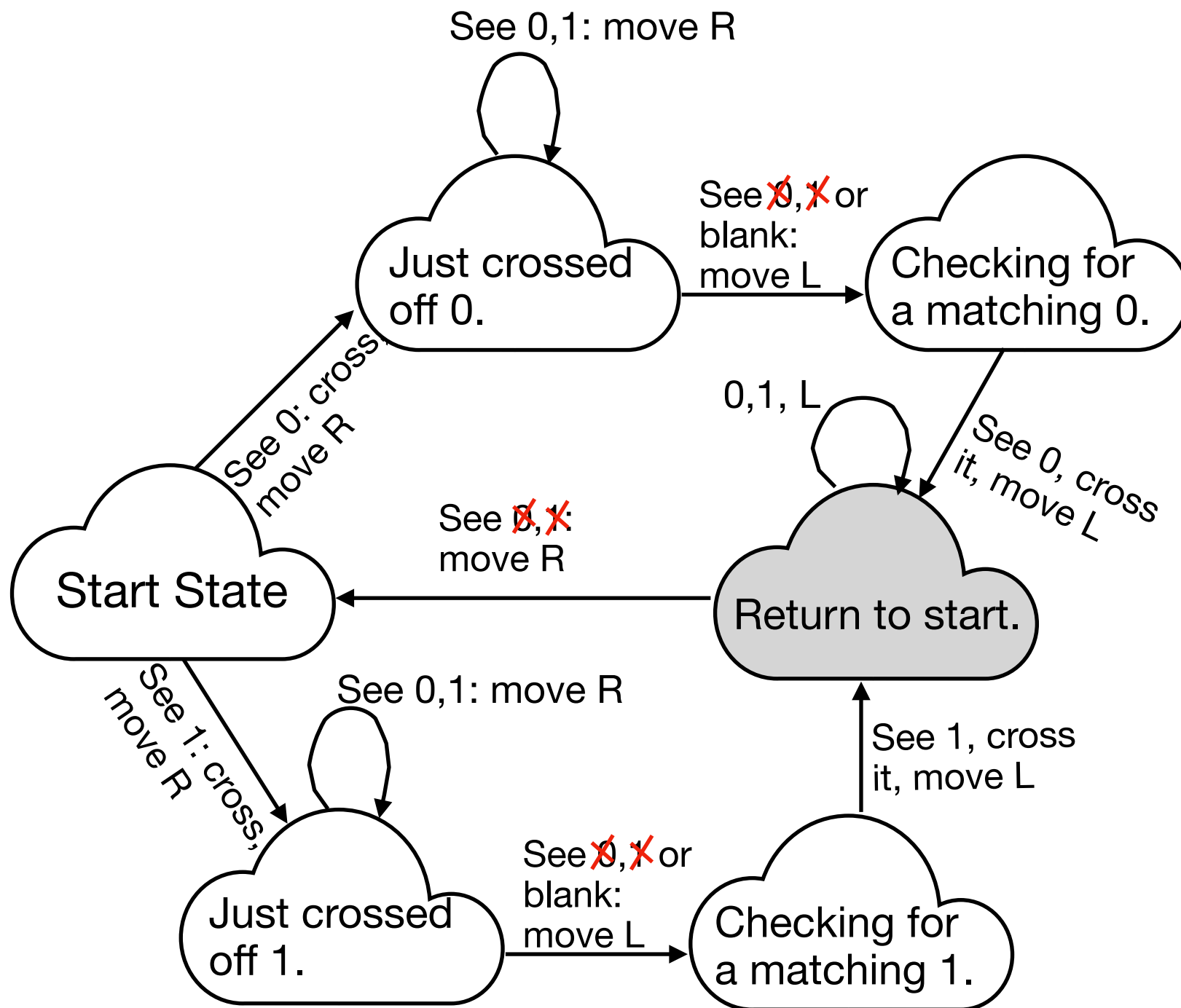
State Diagram



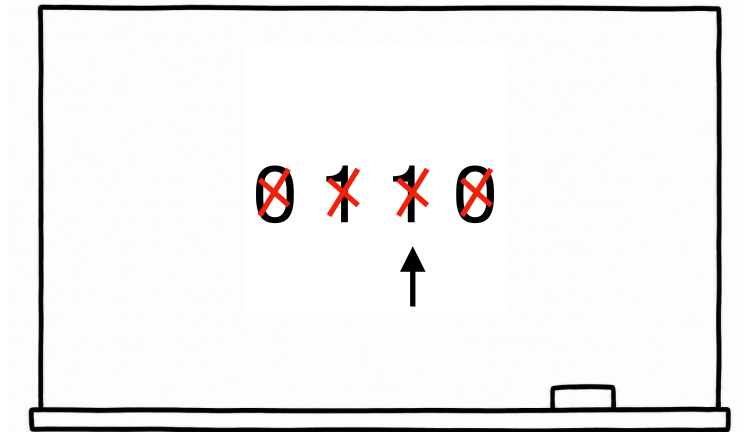
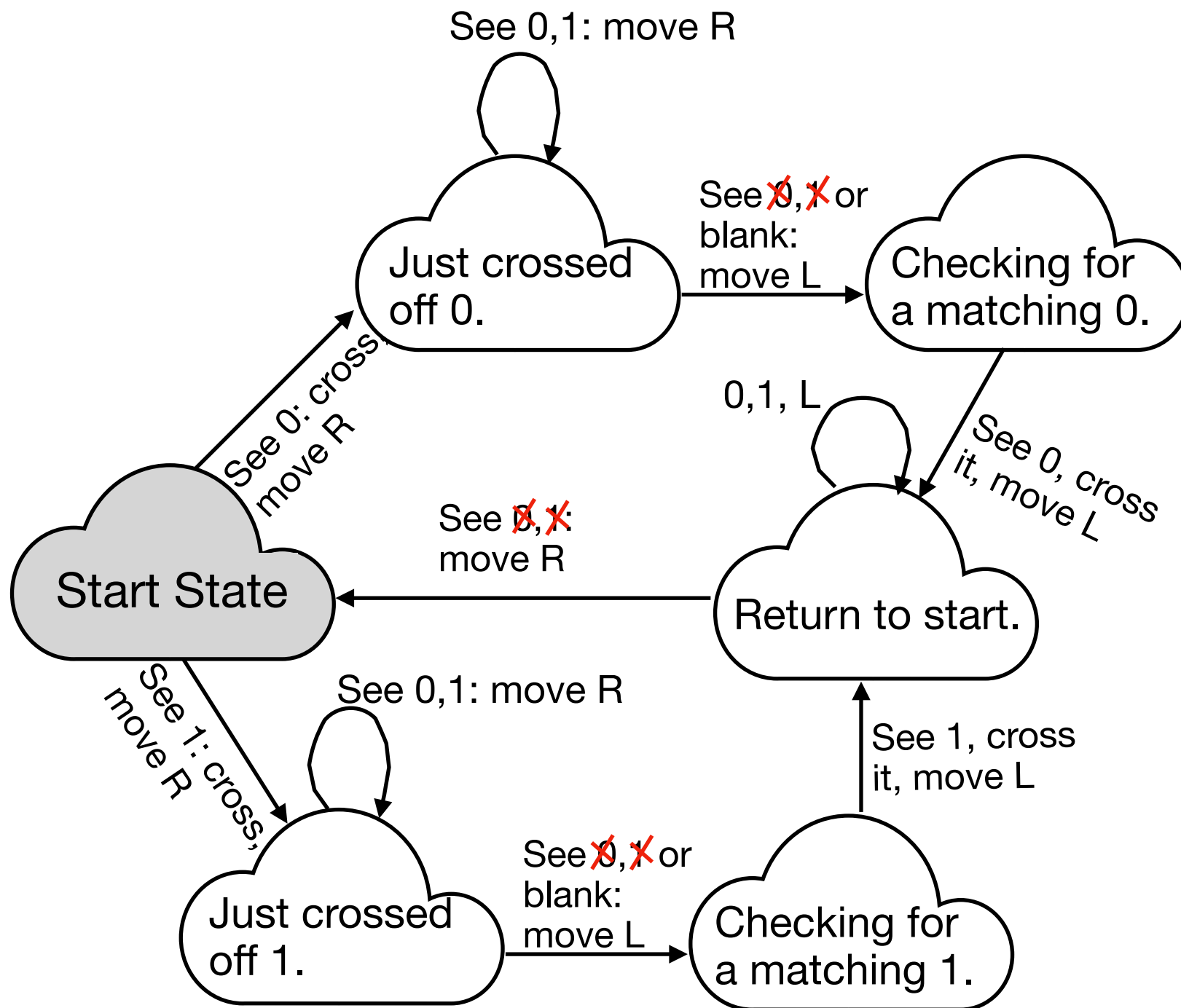
State Diagram



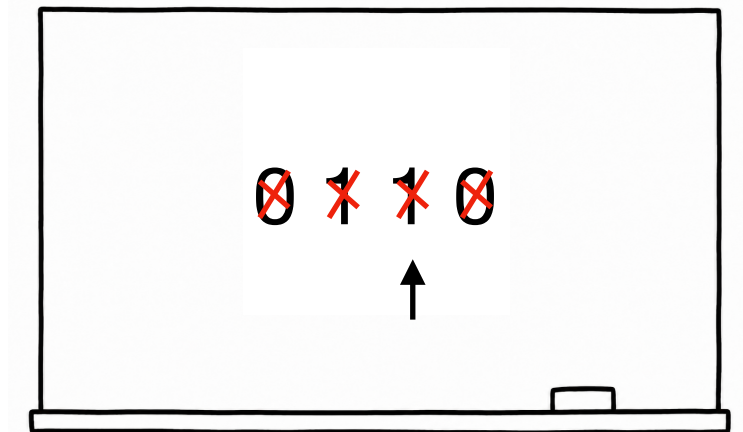
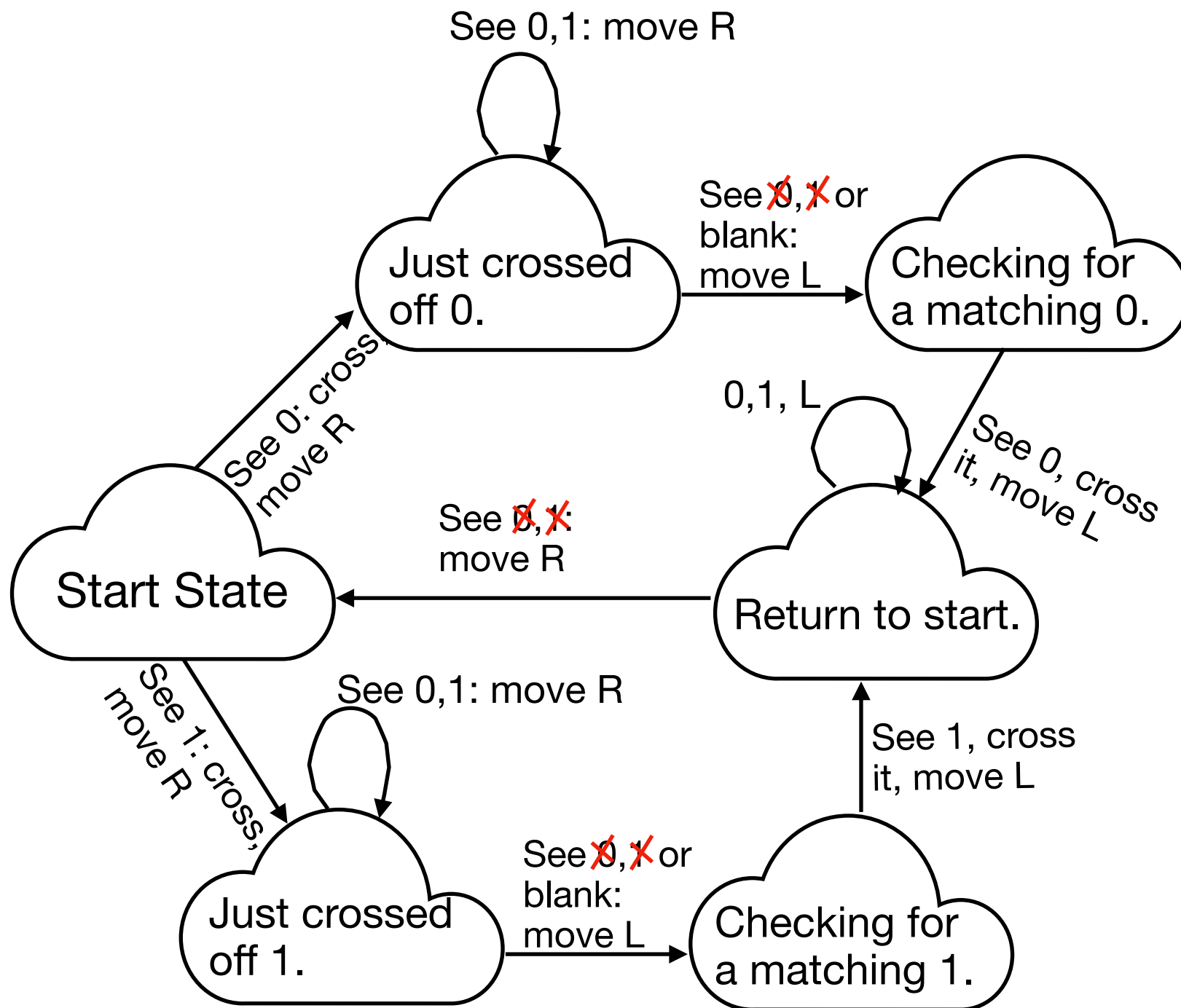
State Diagram



State Diagram

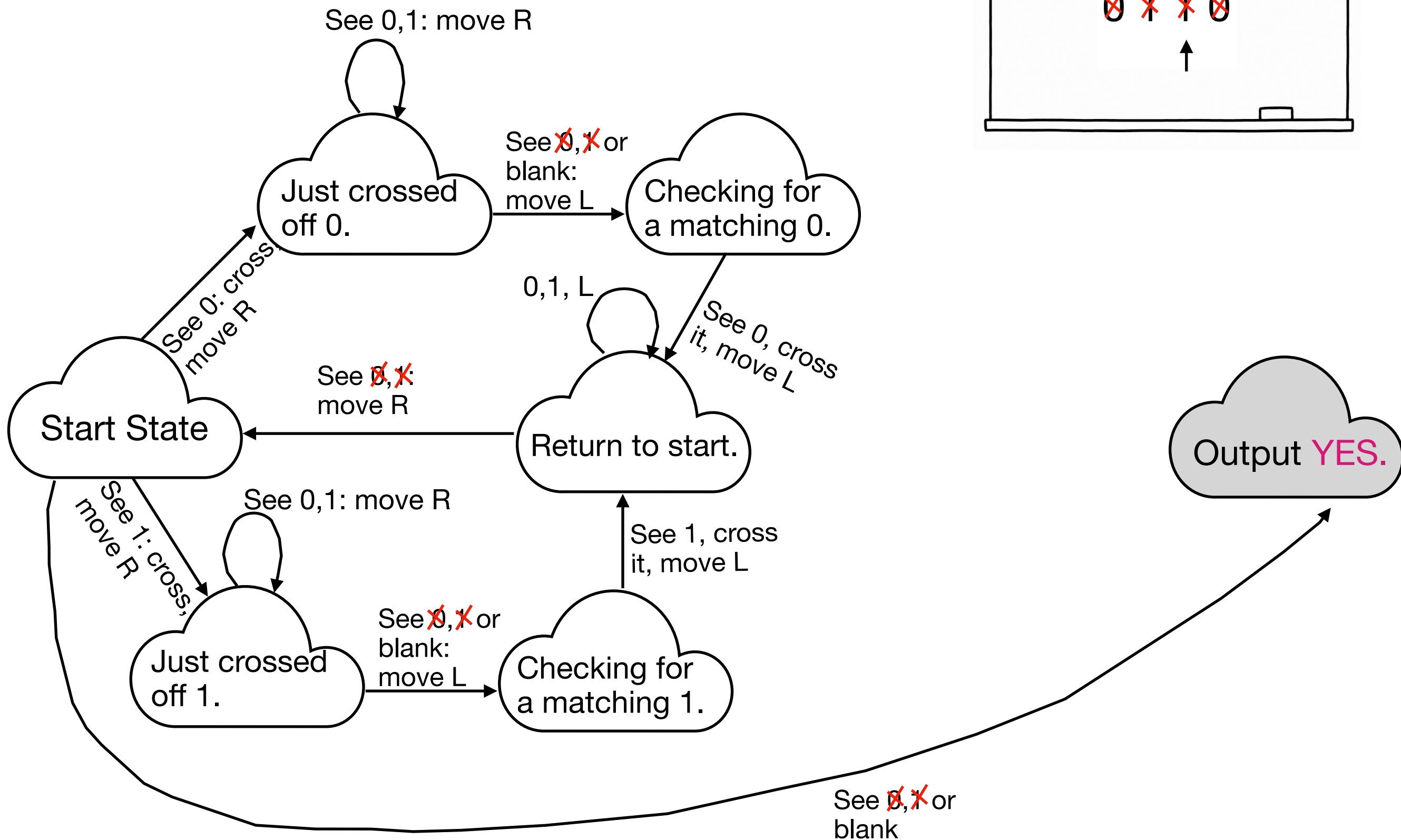


State Diagram

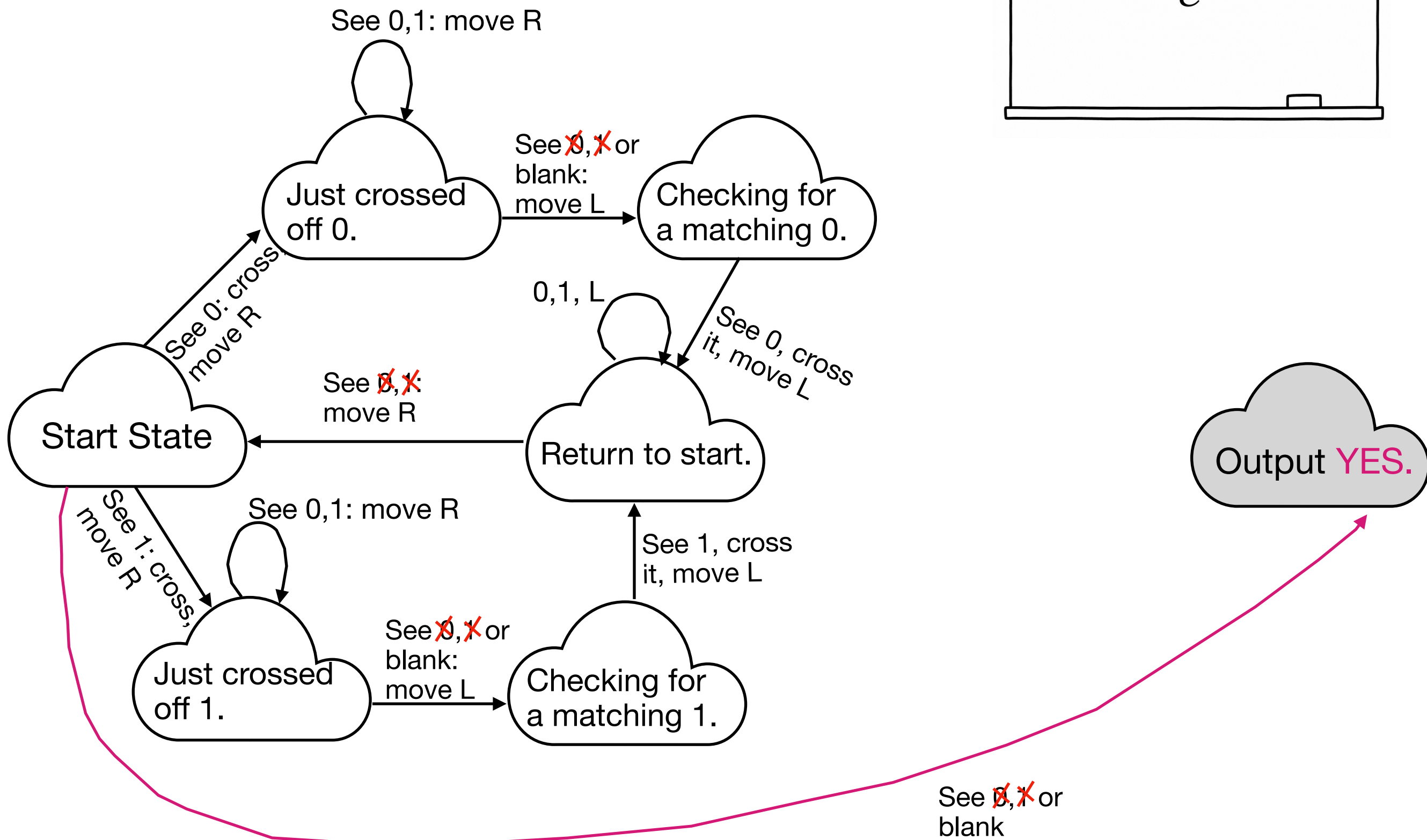


Output **YES.**

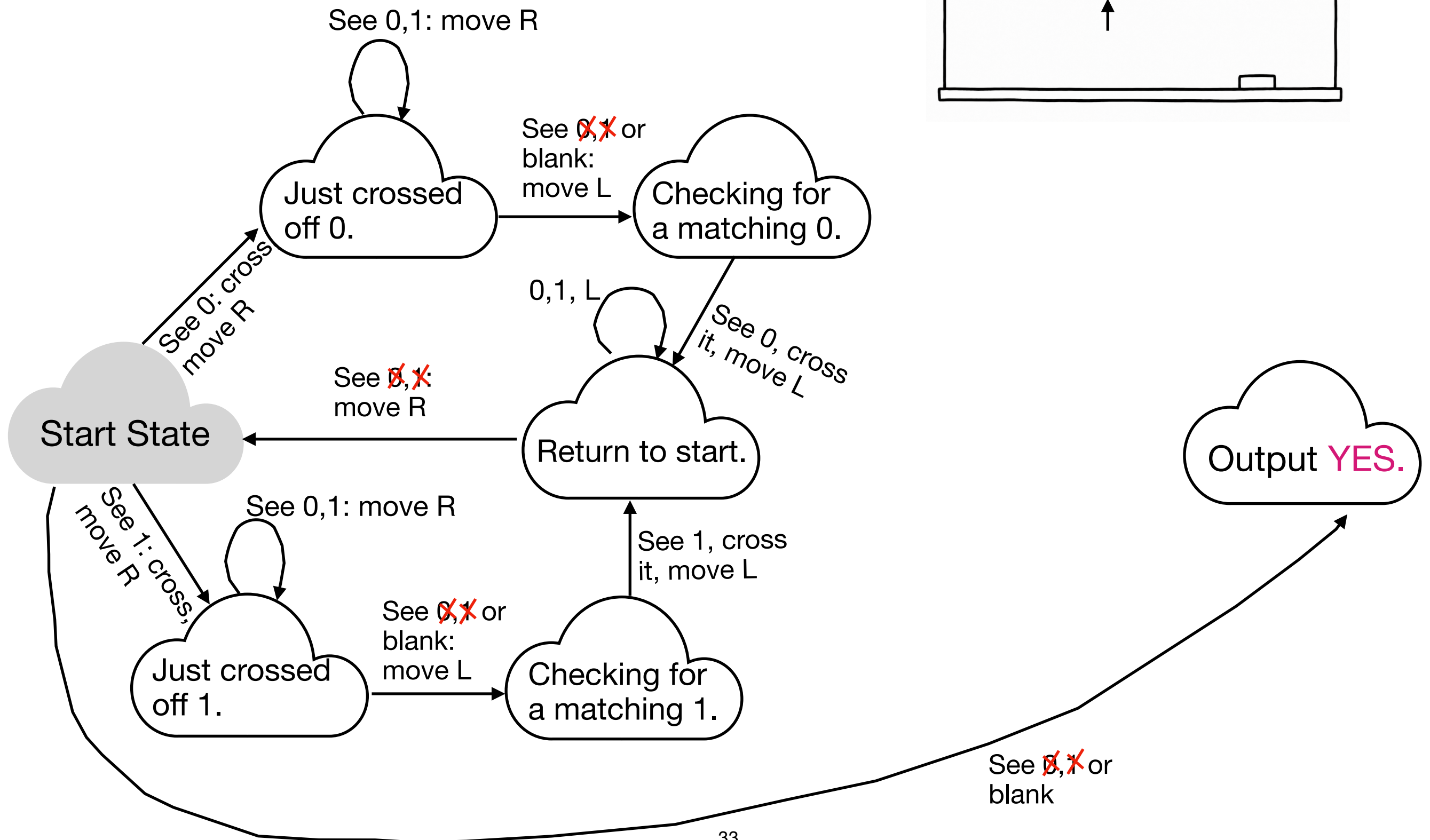
State Diagram



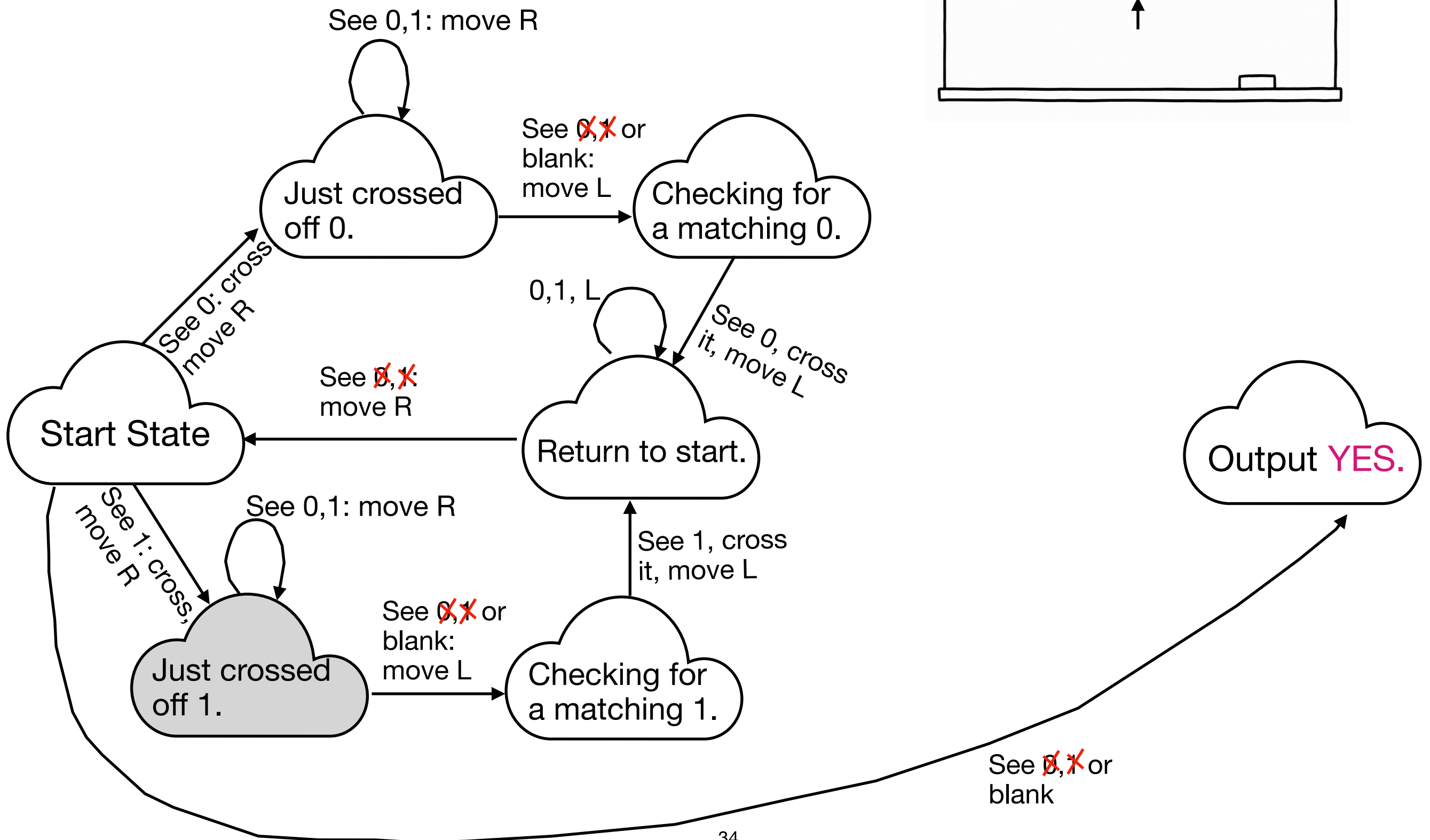
State Diagram



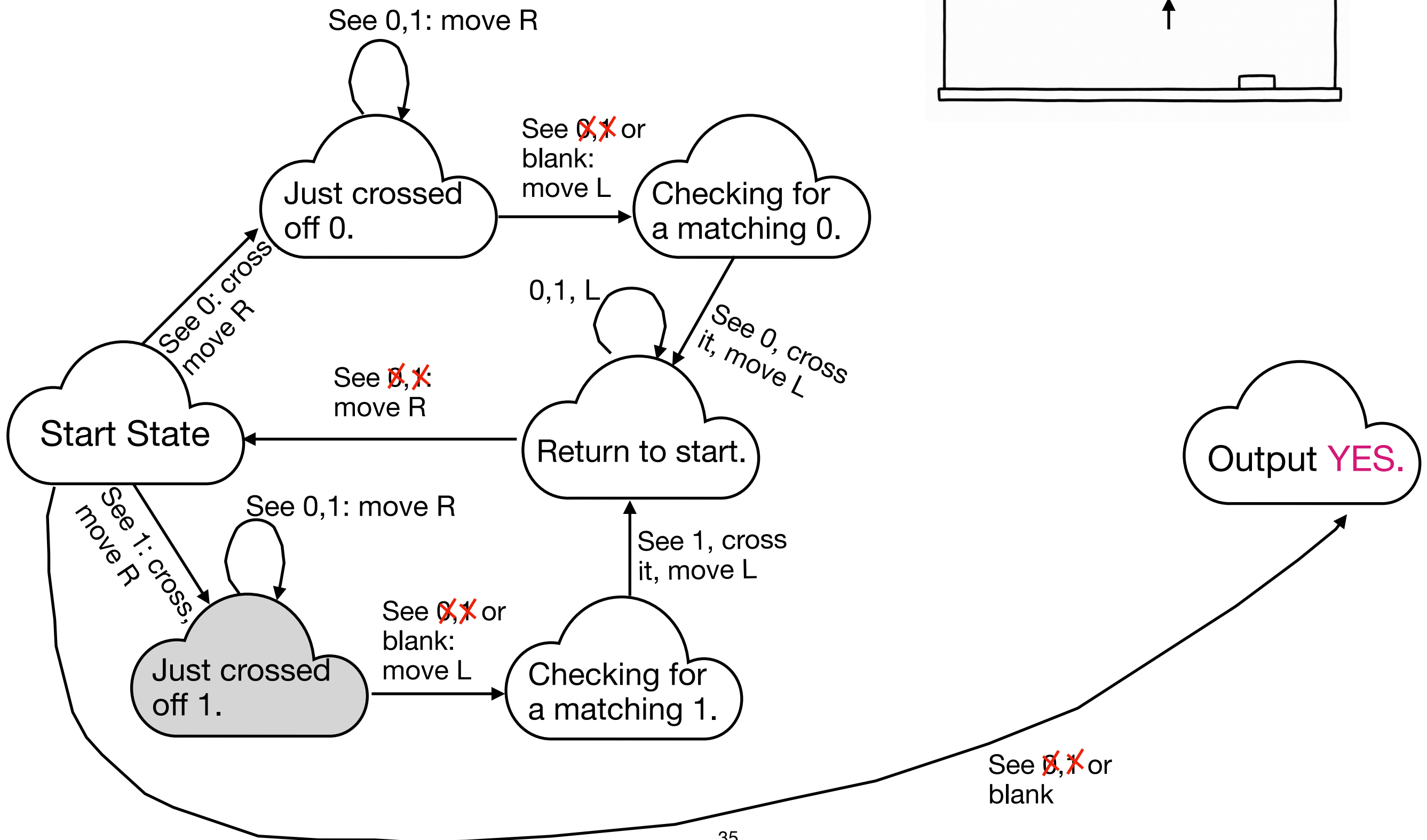
Examples



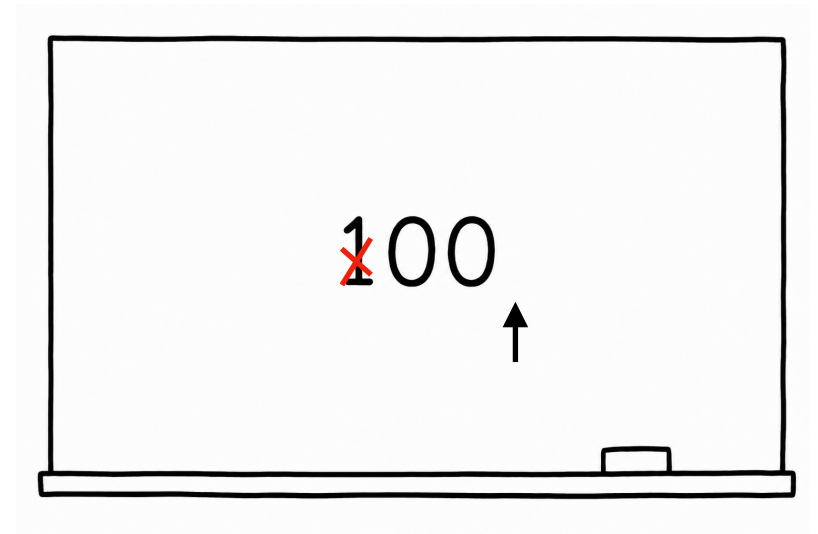
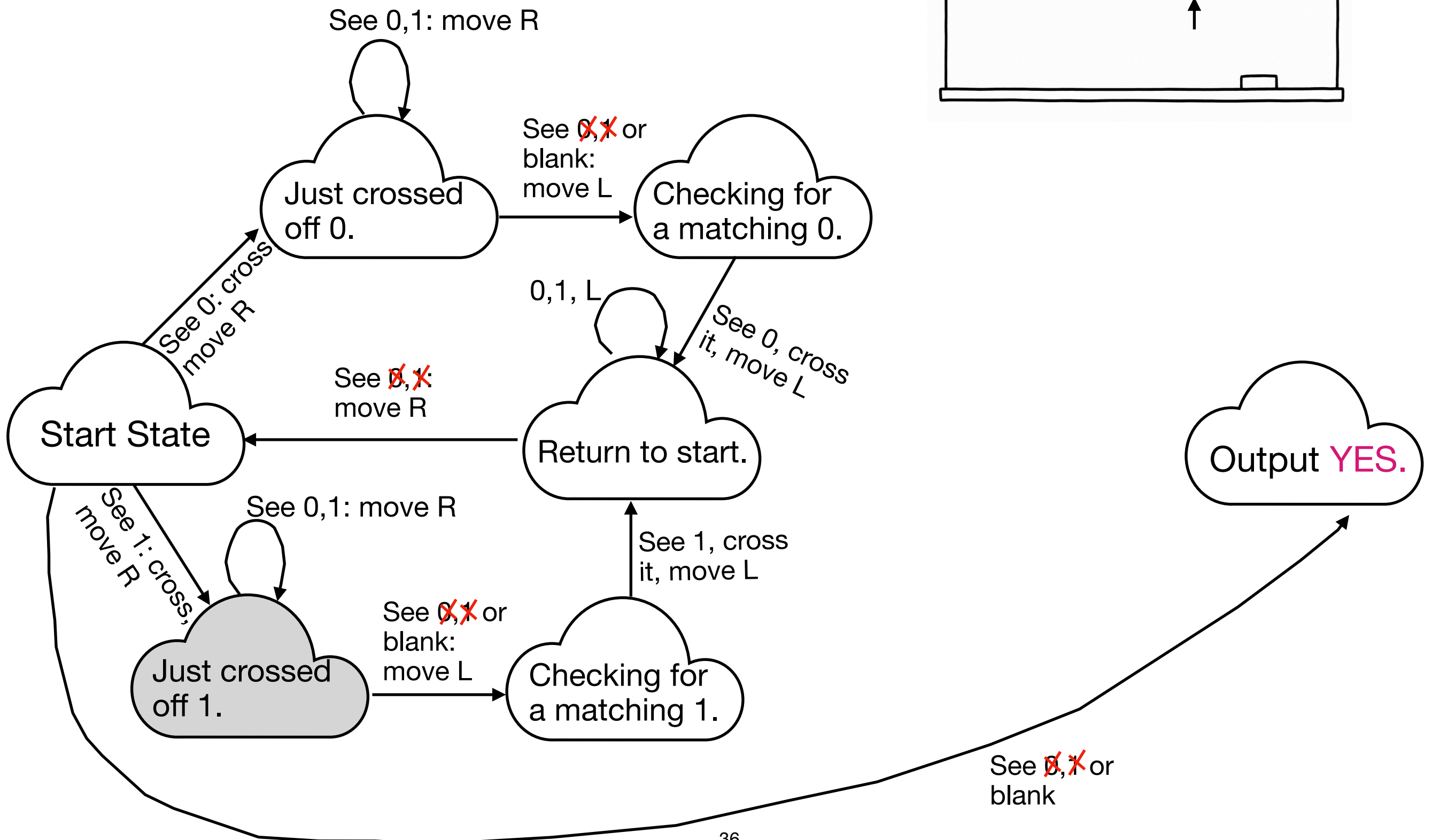
Examples



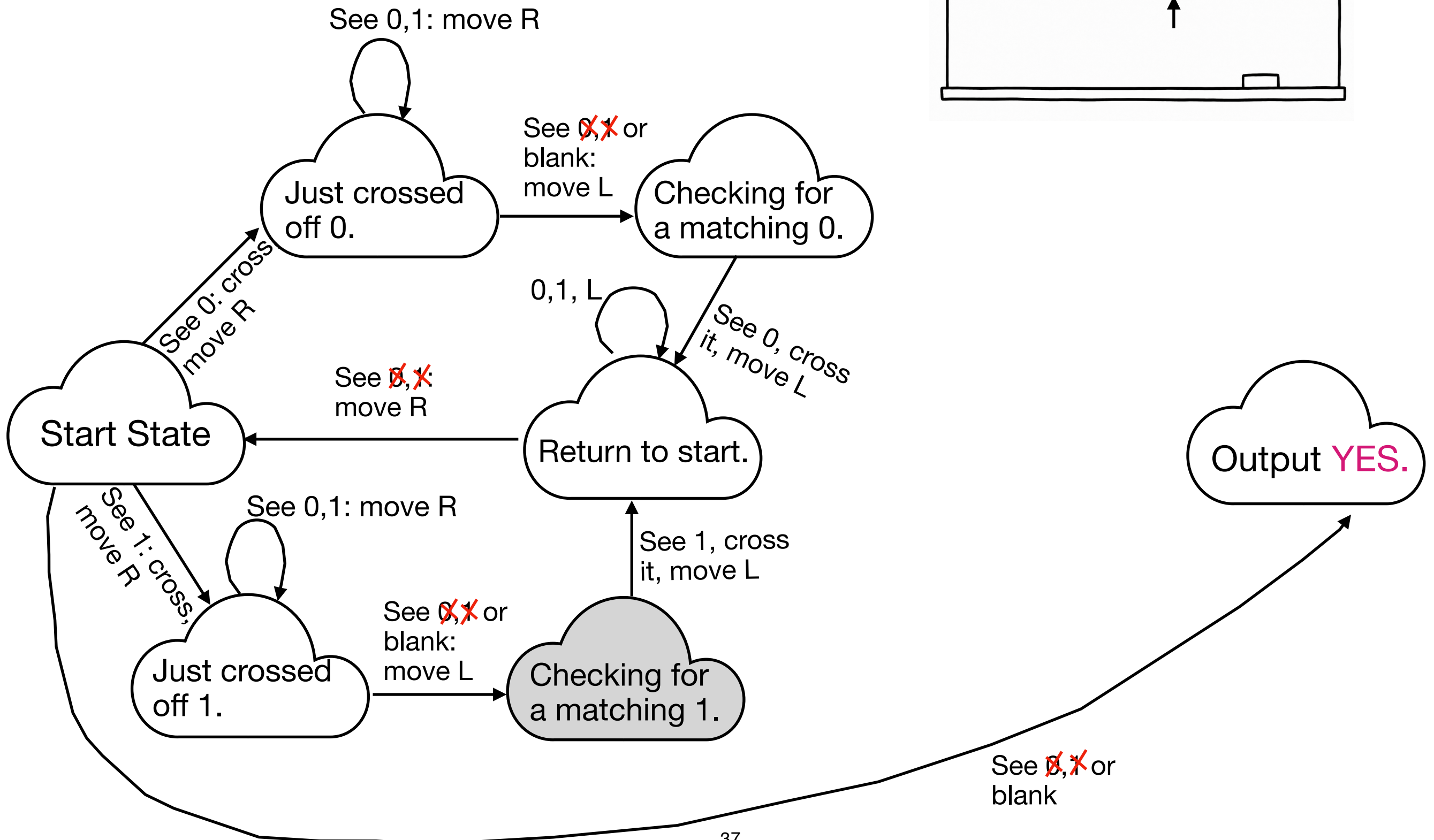
Examples



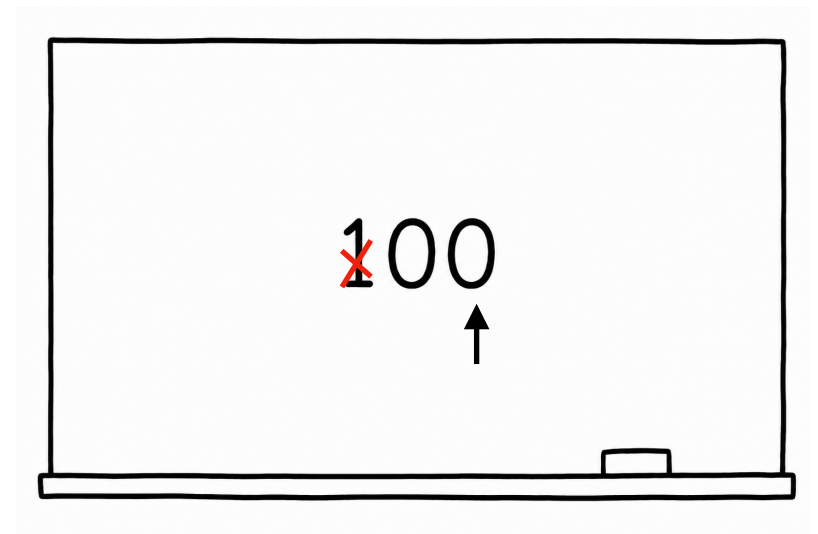
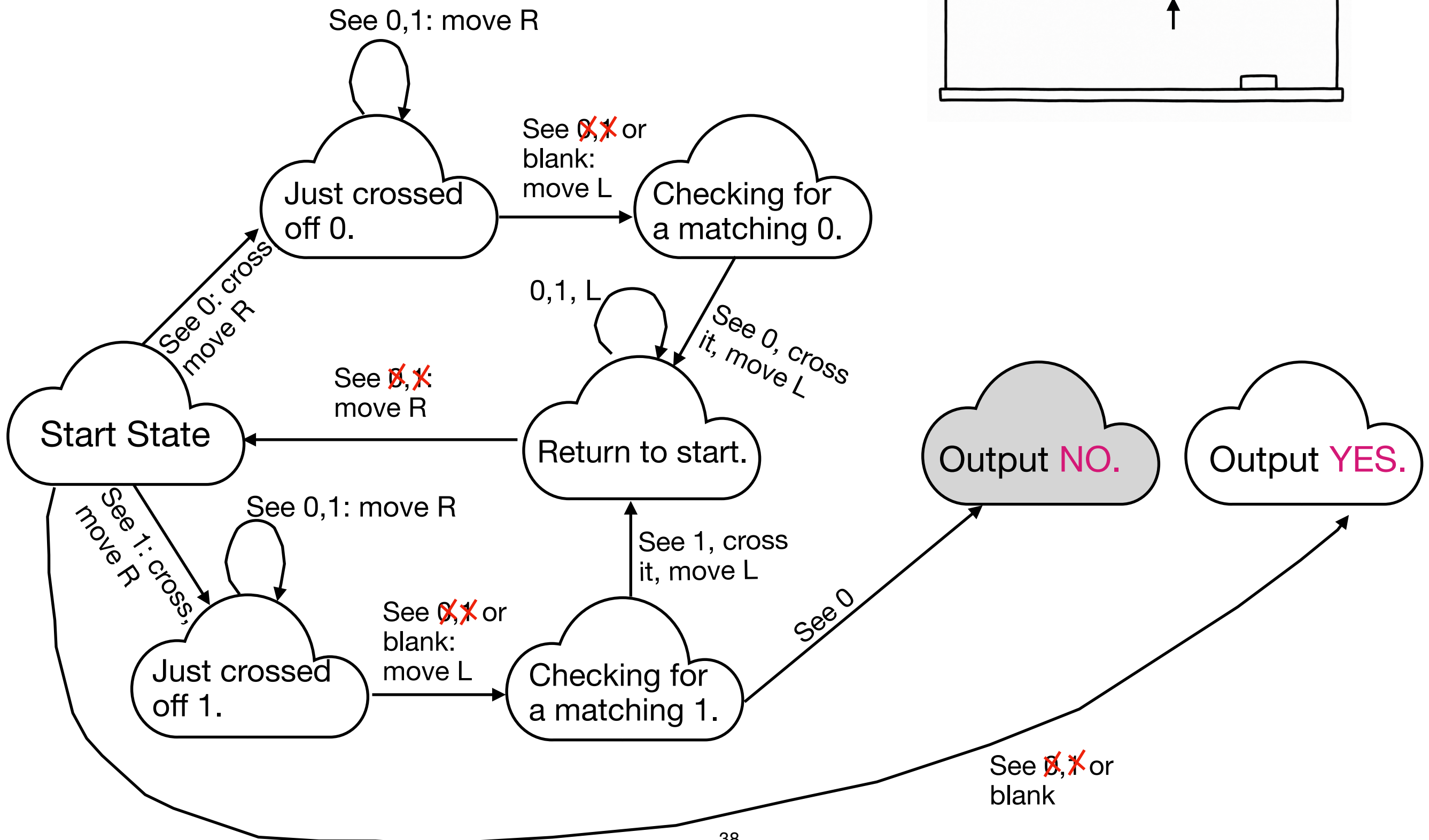
Examples



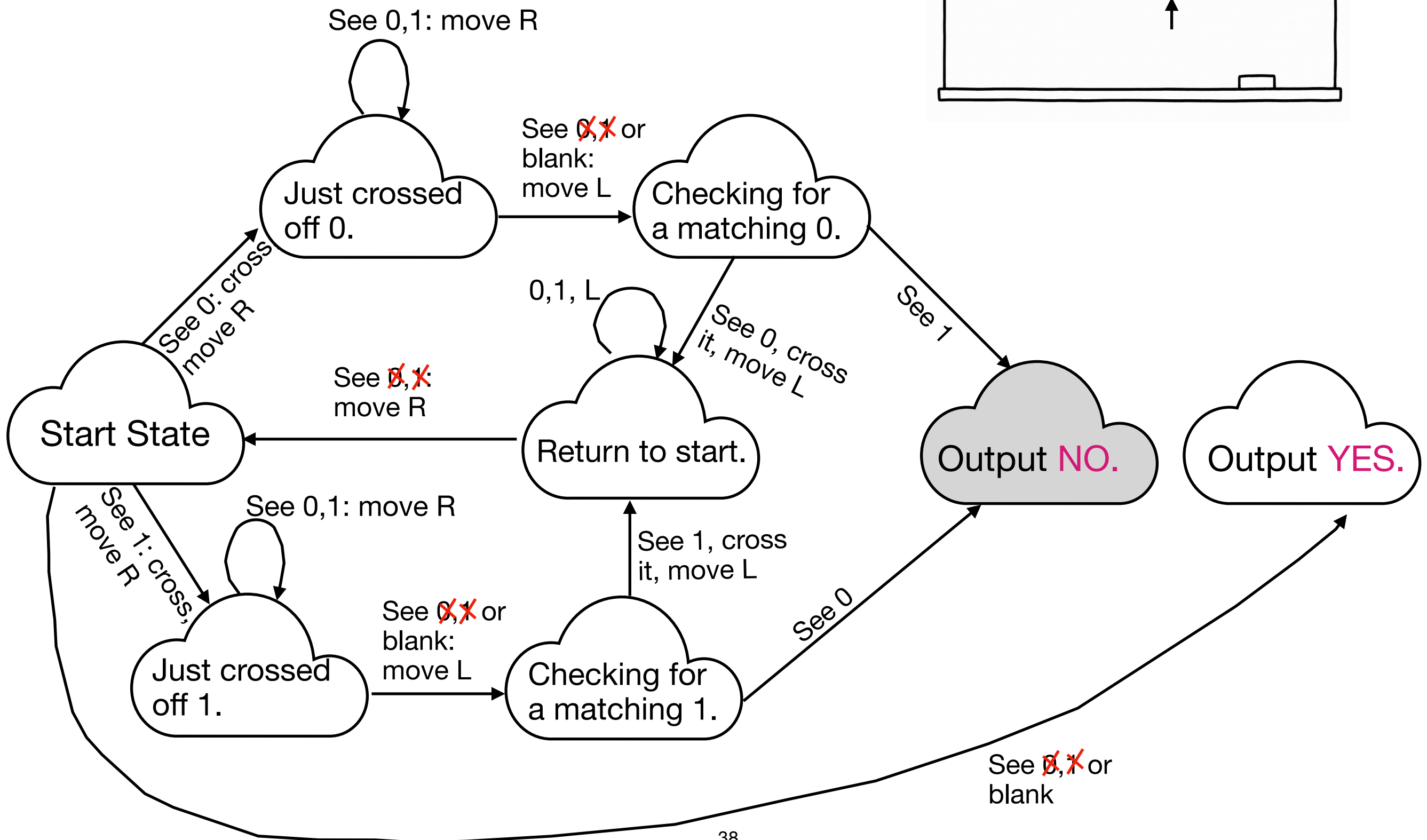
Examples



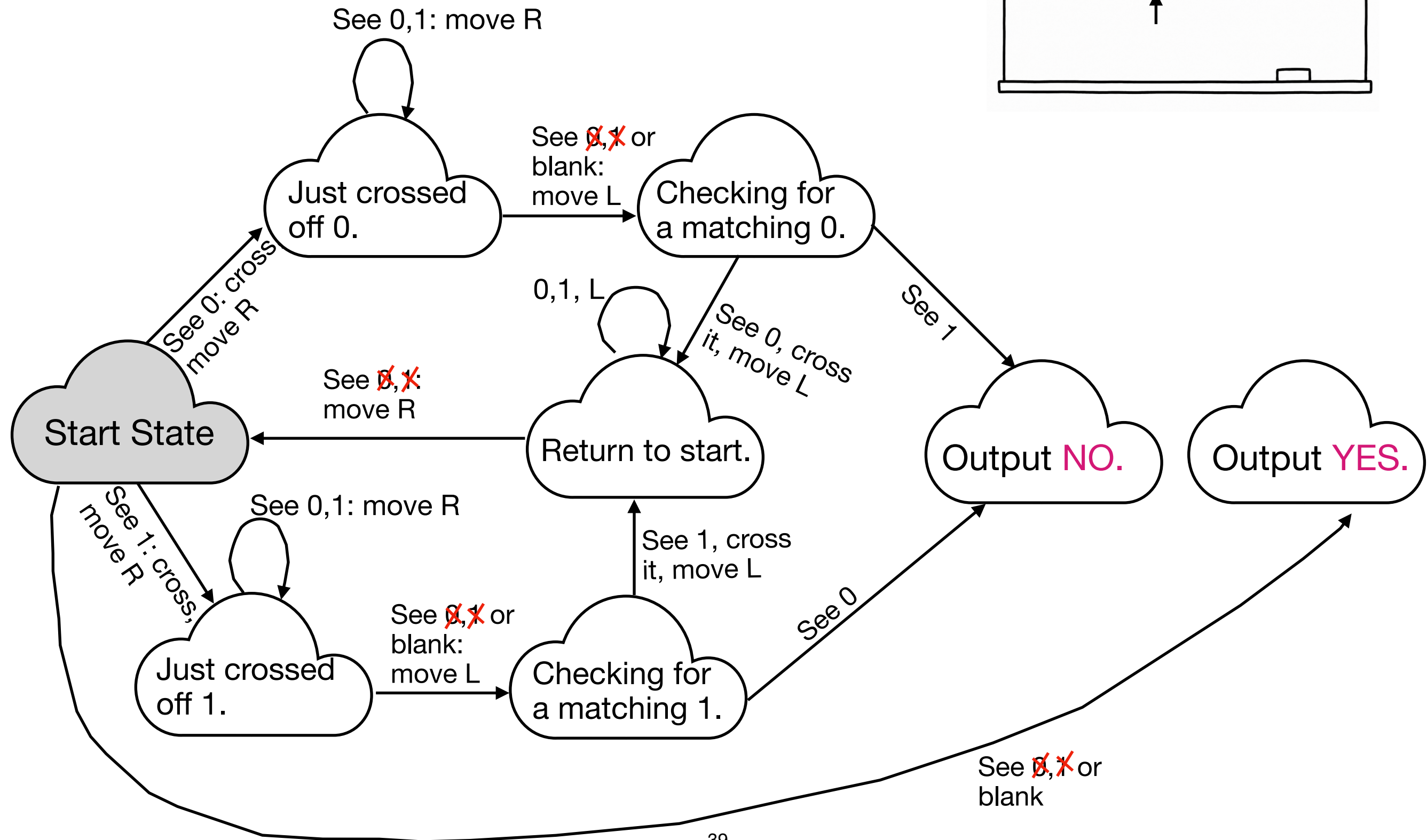
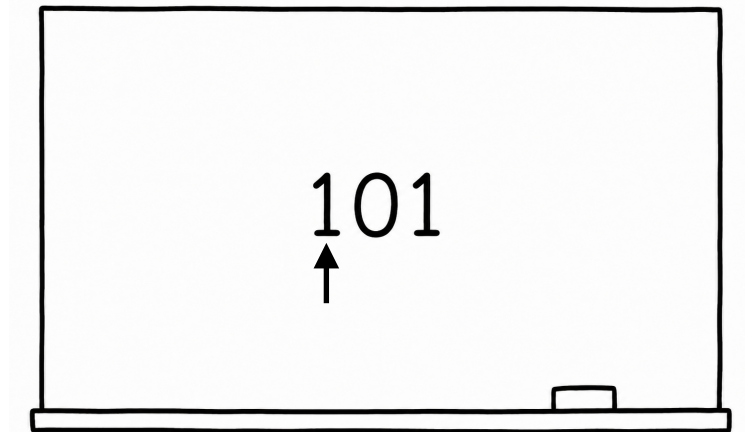
Examples



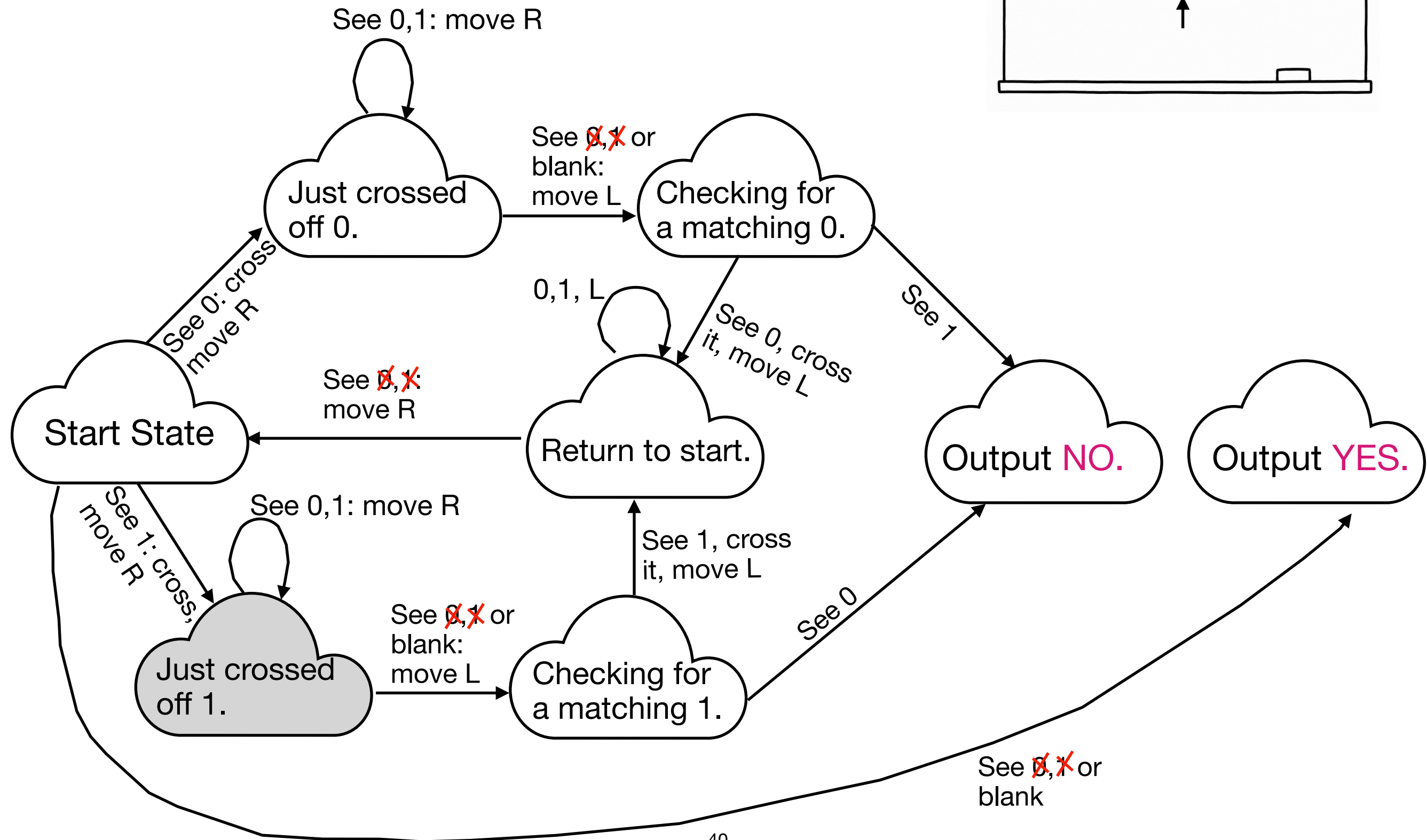
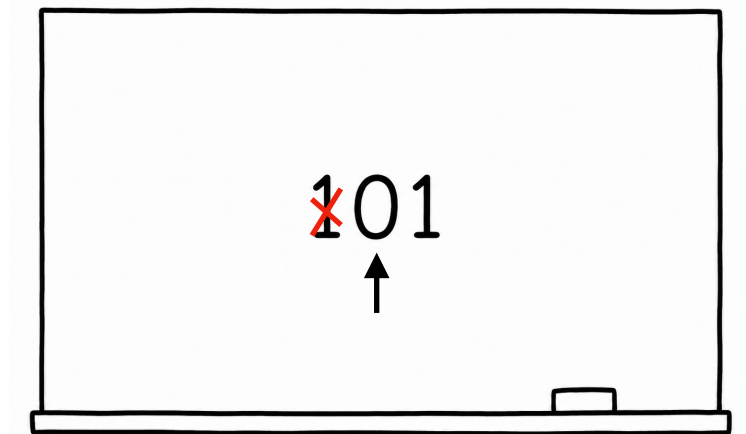
Examples



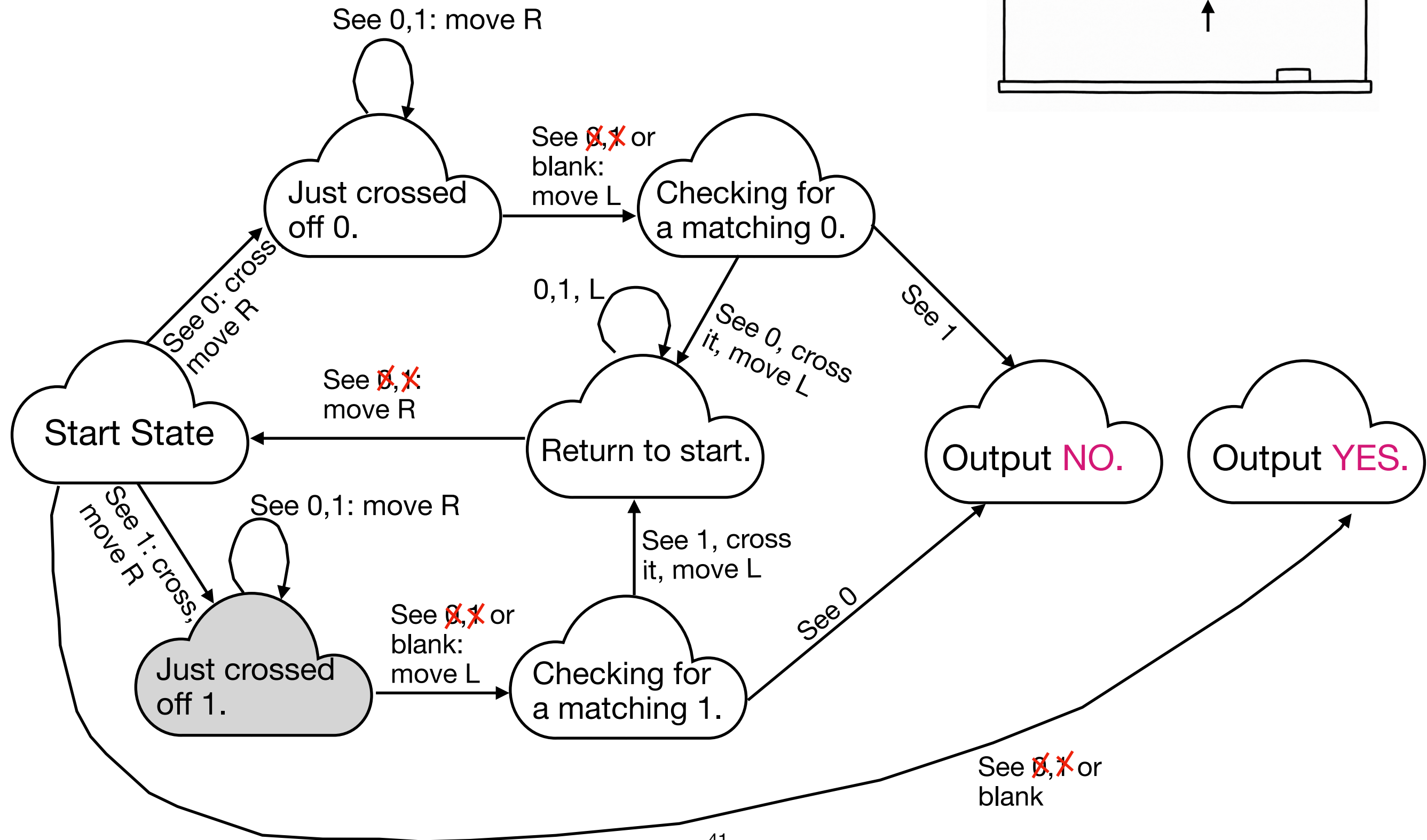
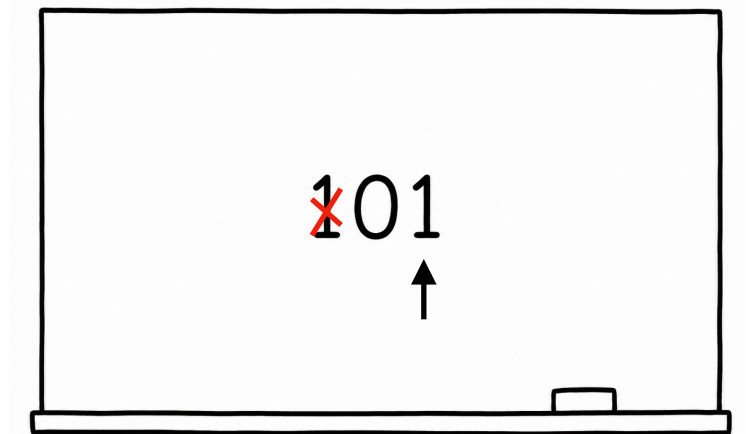
Examples



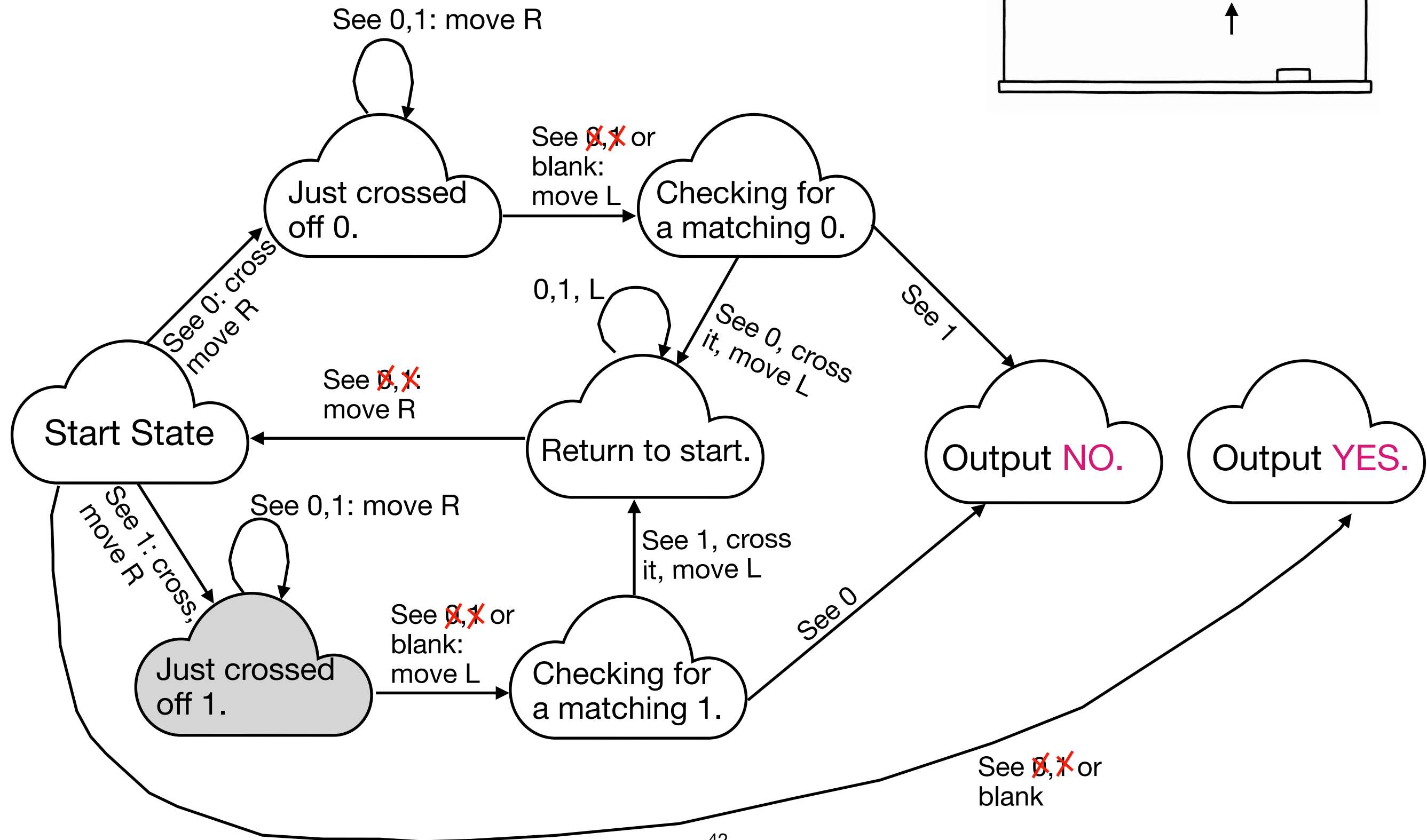
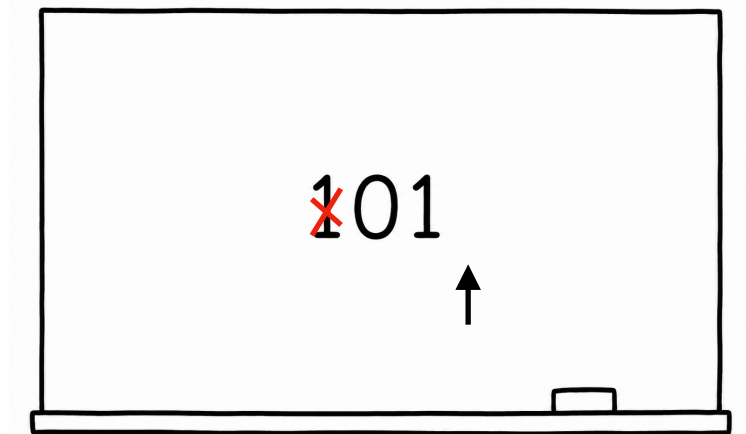
Examples



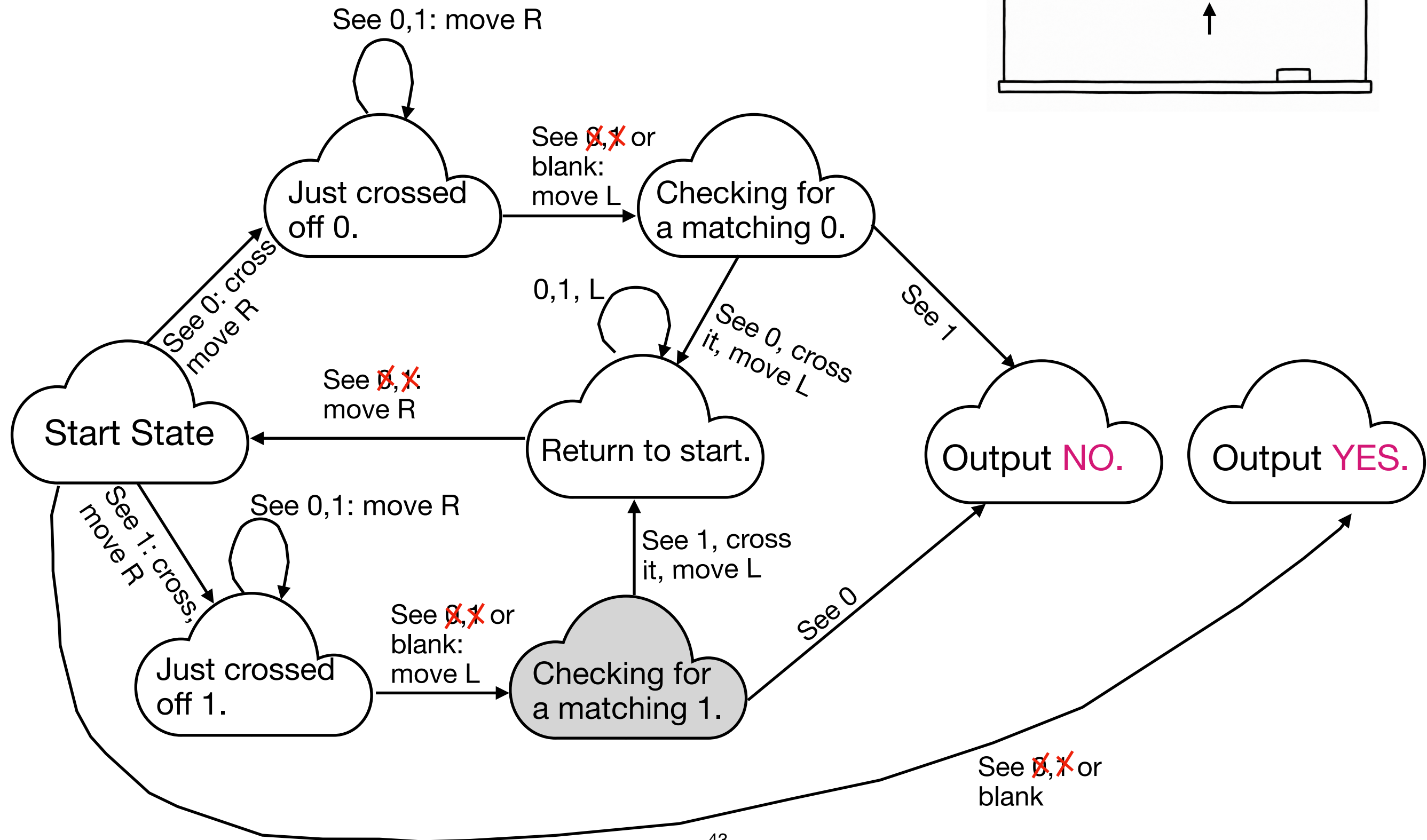
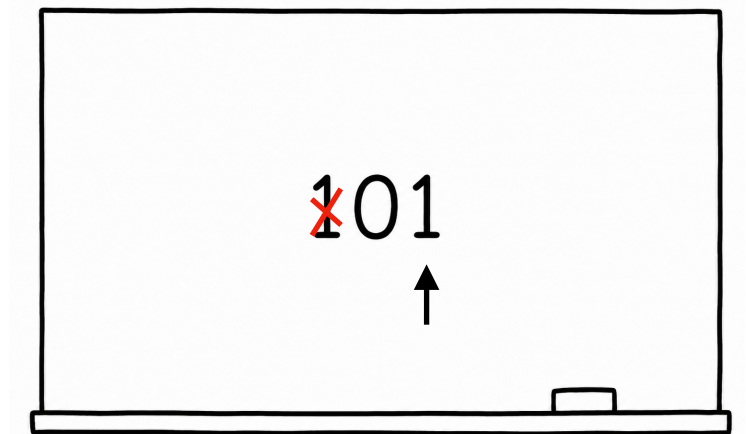
Examples



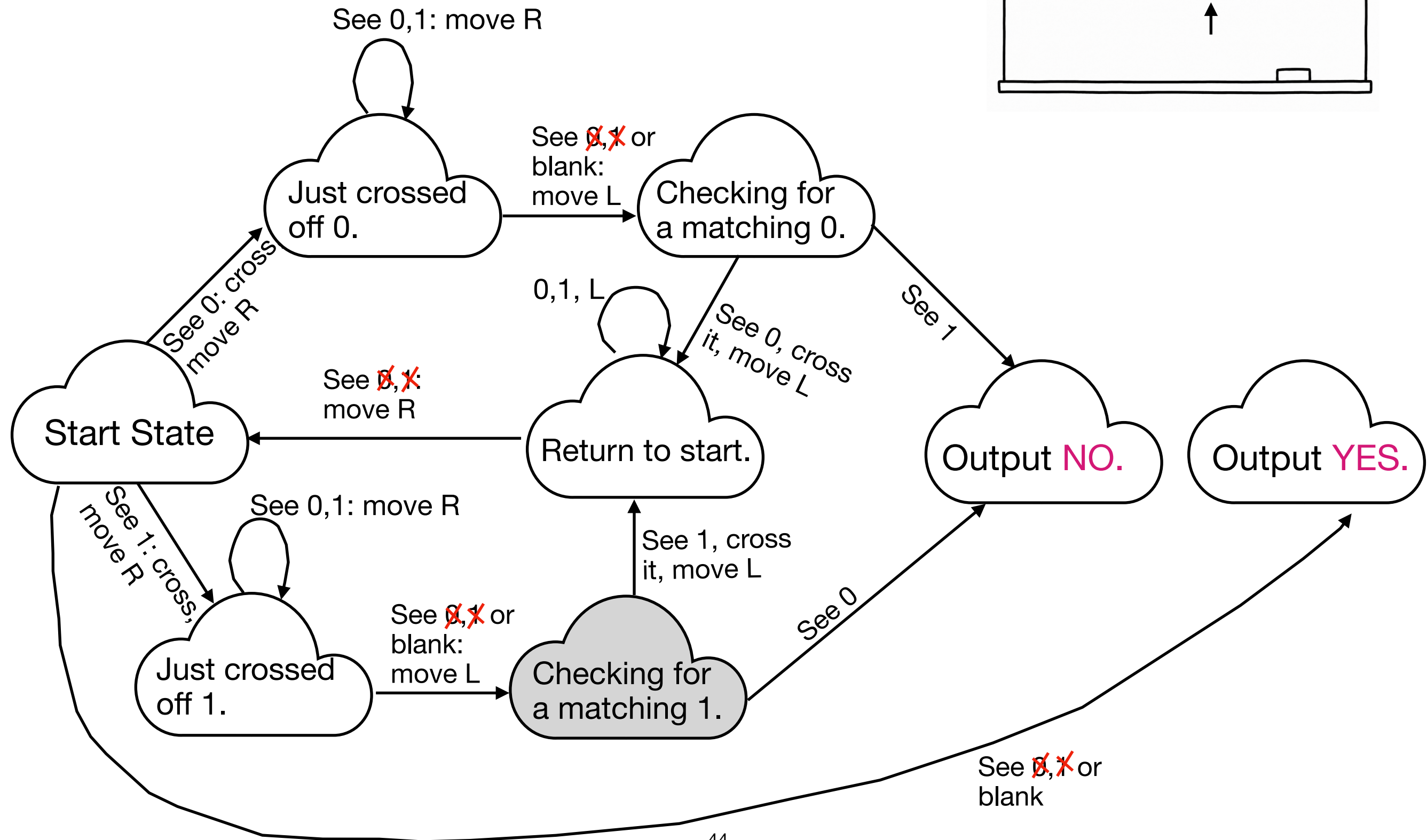
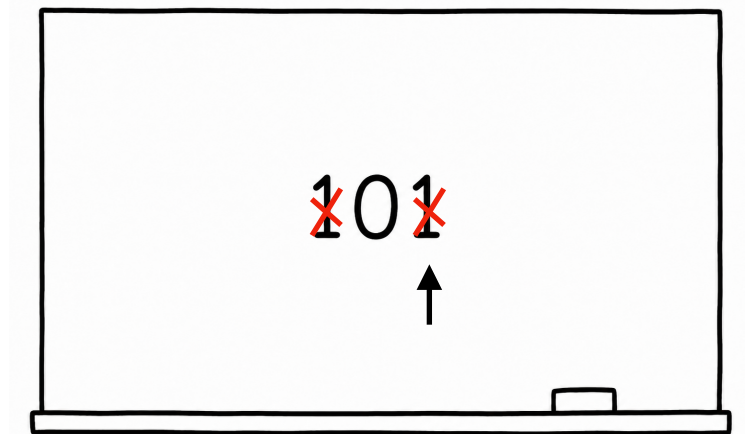
Examples



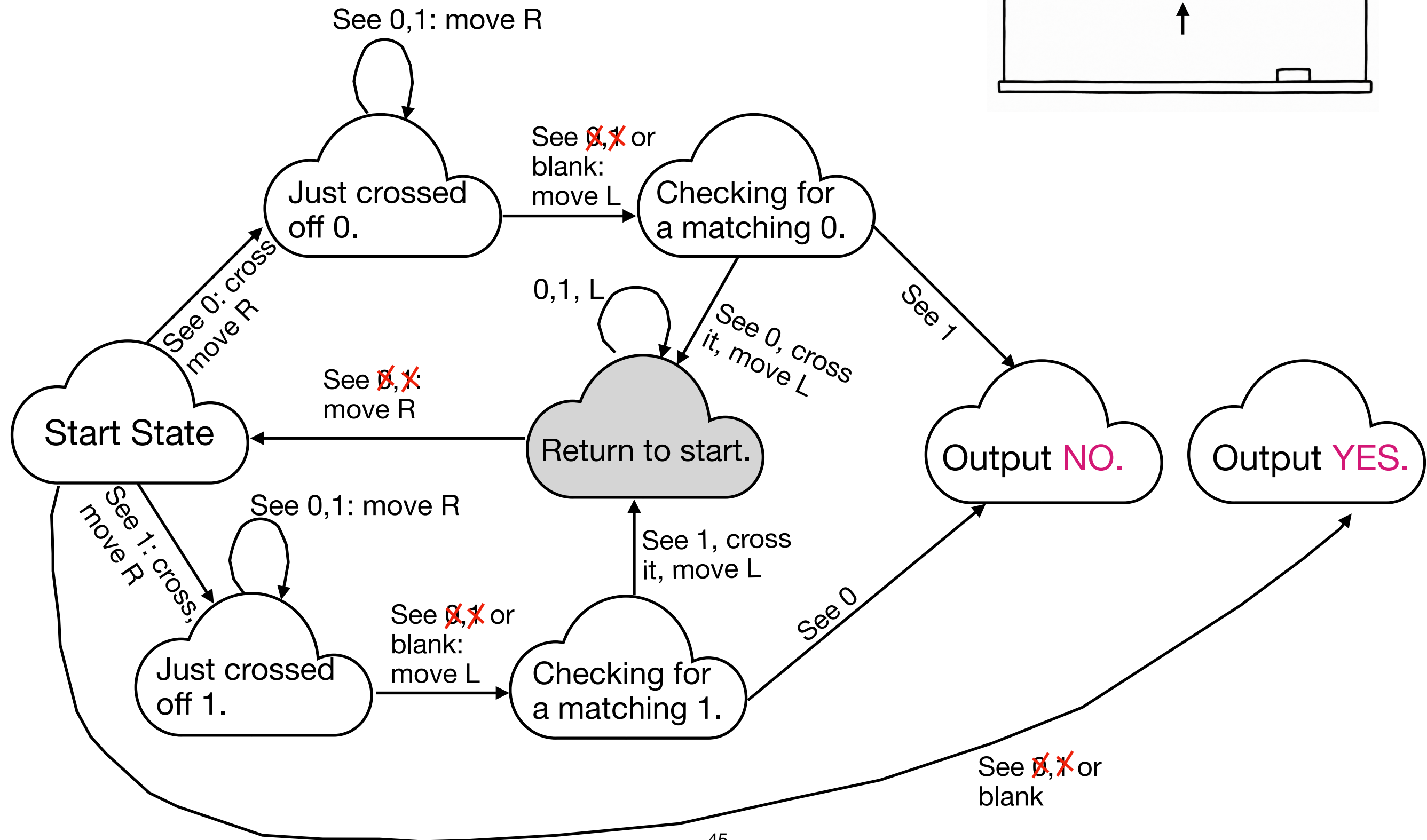
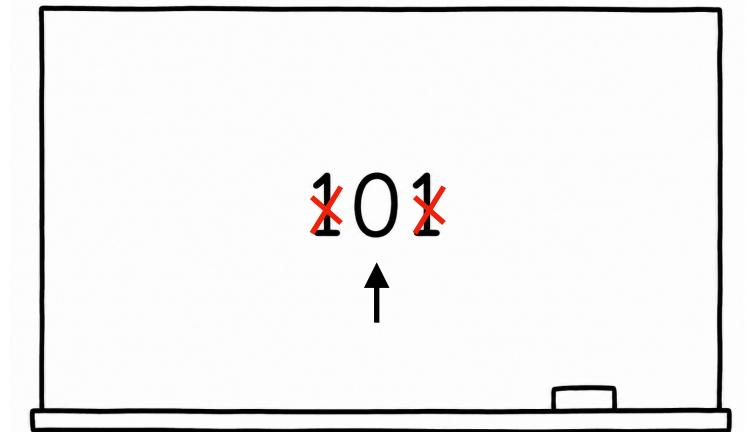
Examples



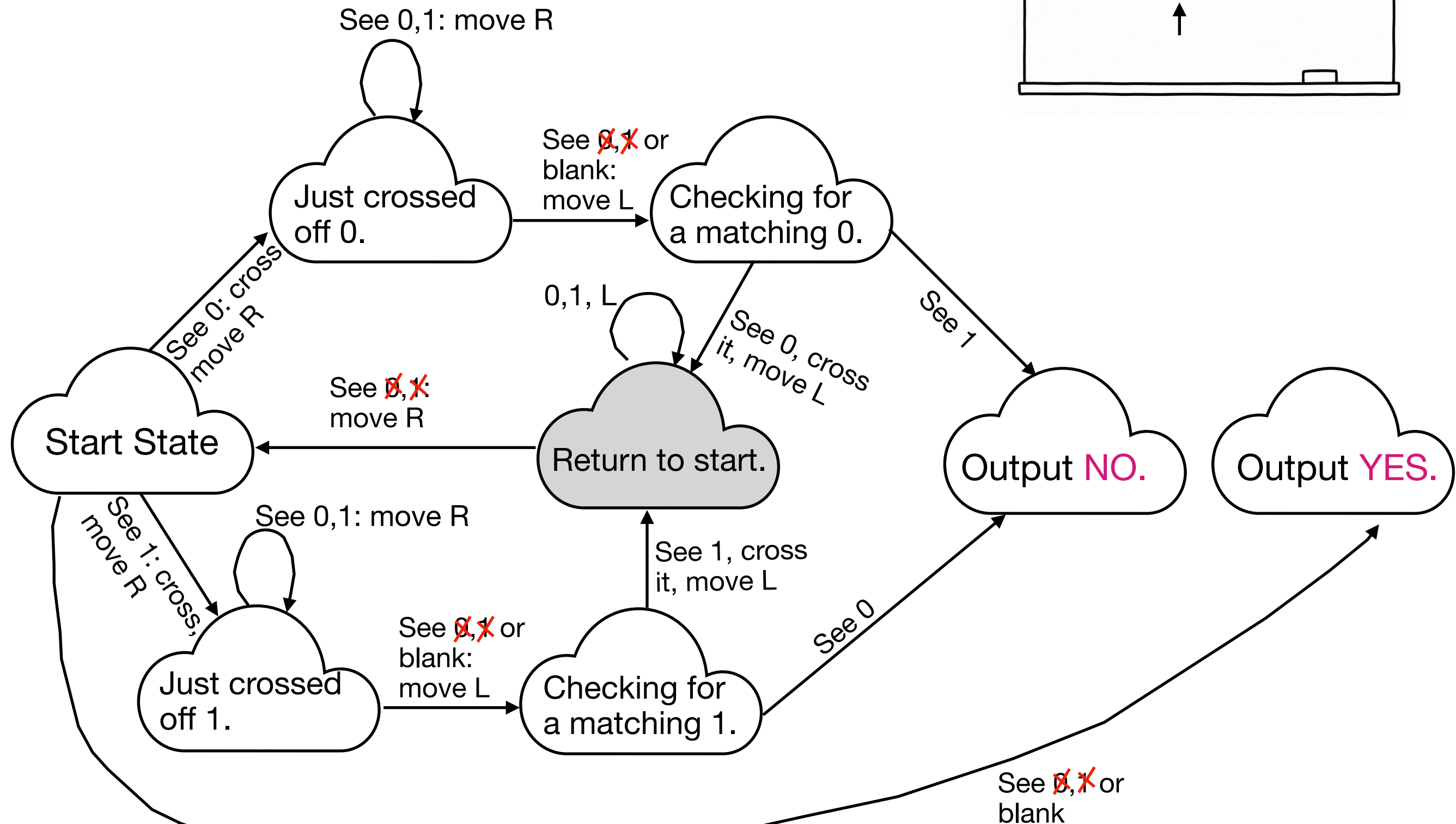
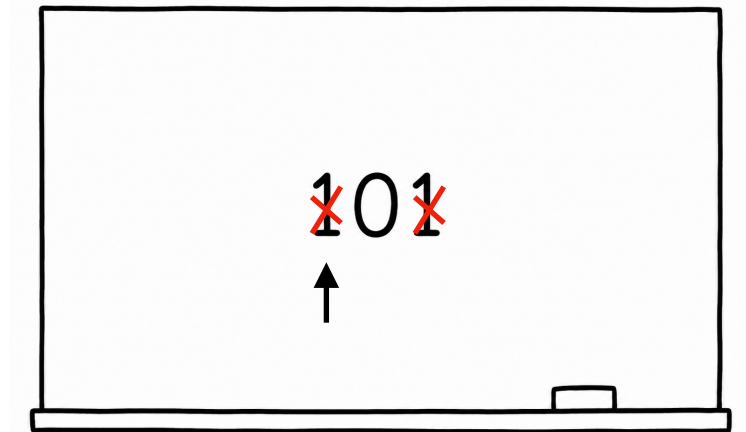
Examples



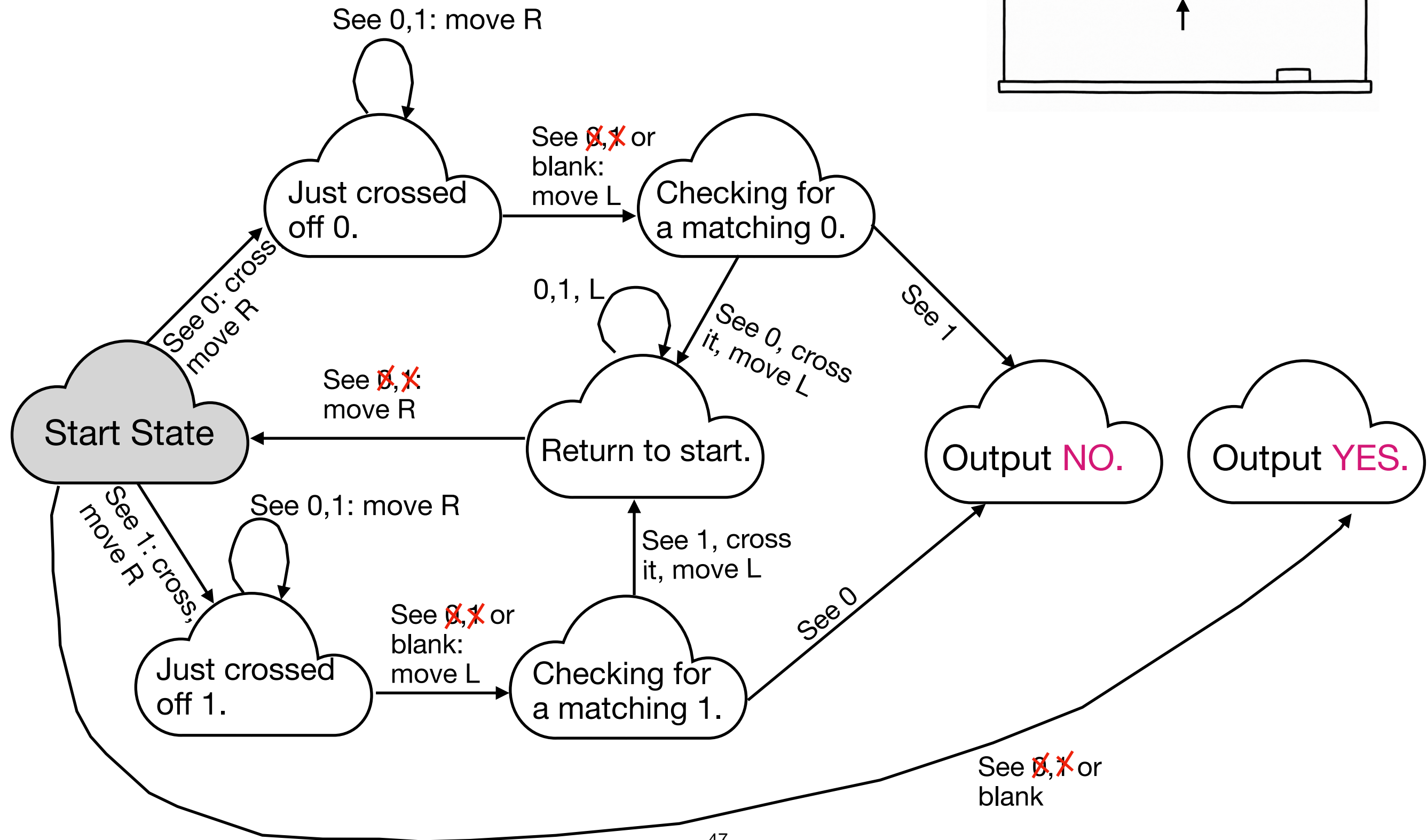
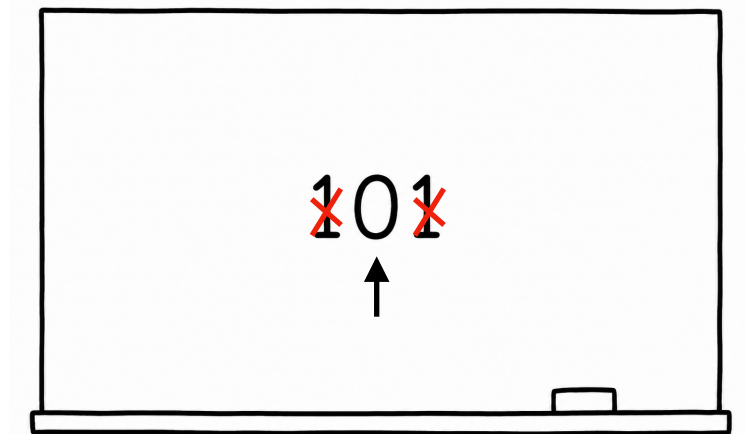
Examples



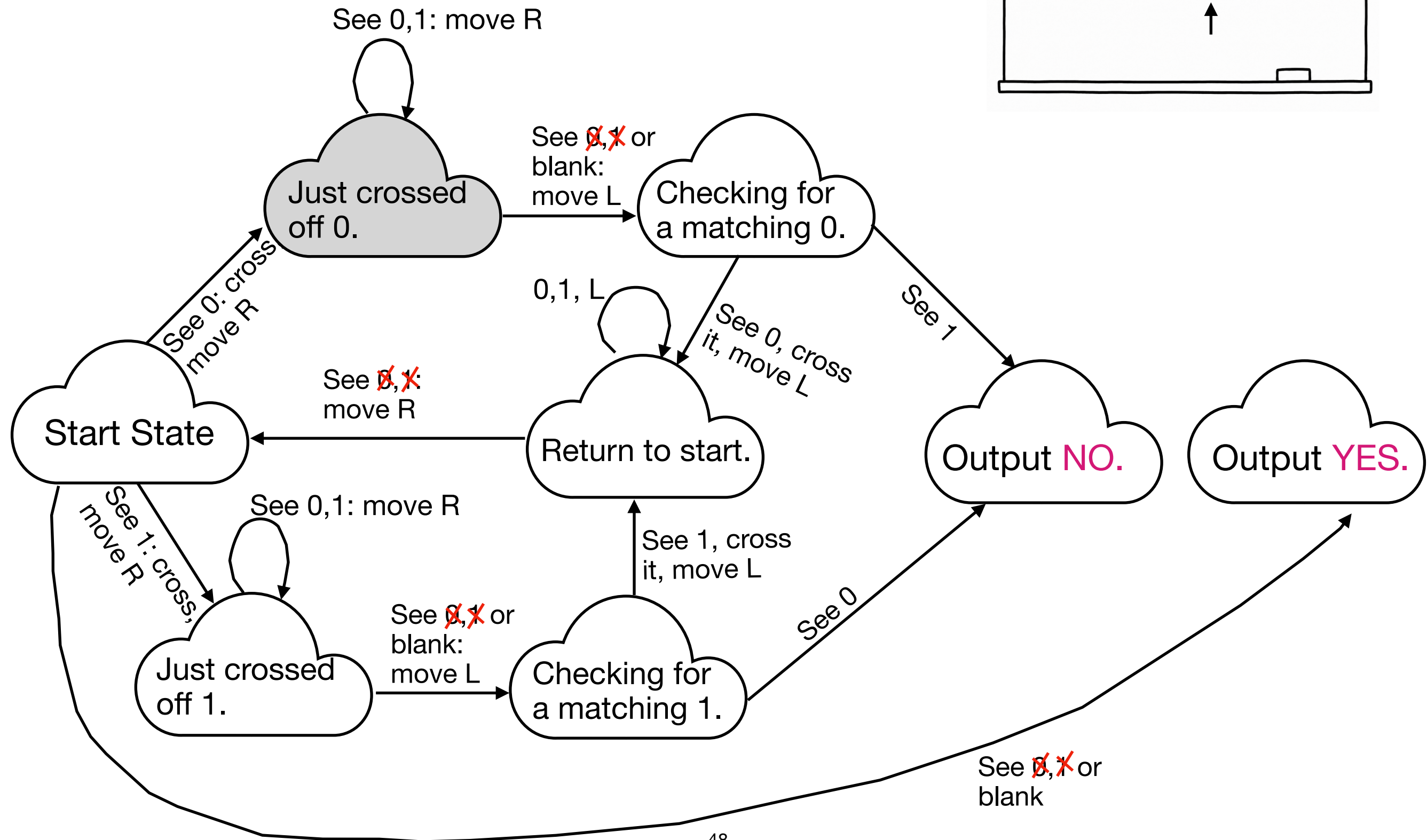
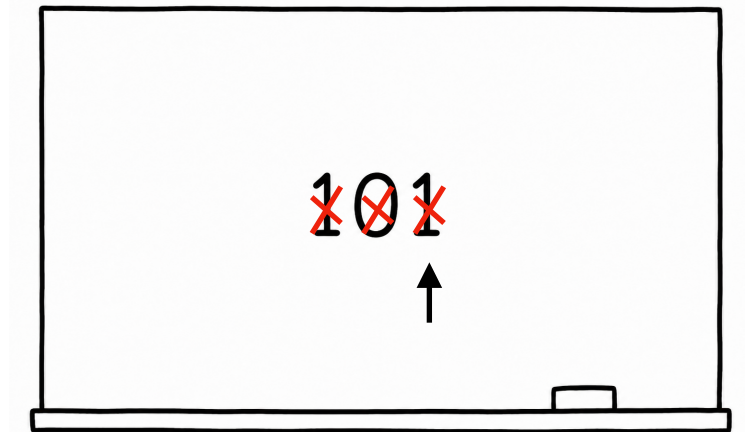
Examples



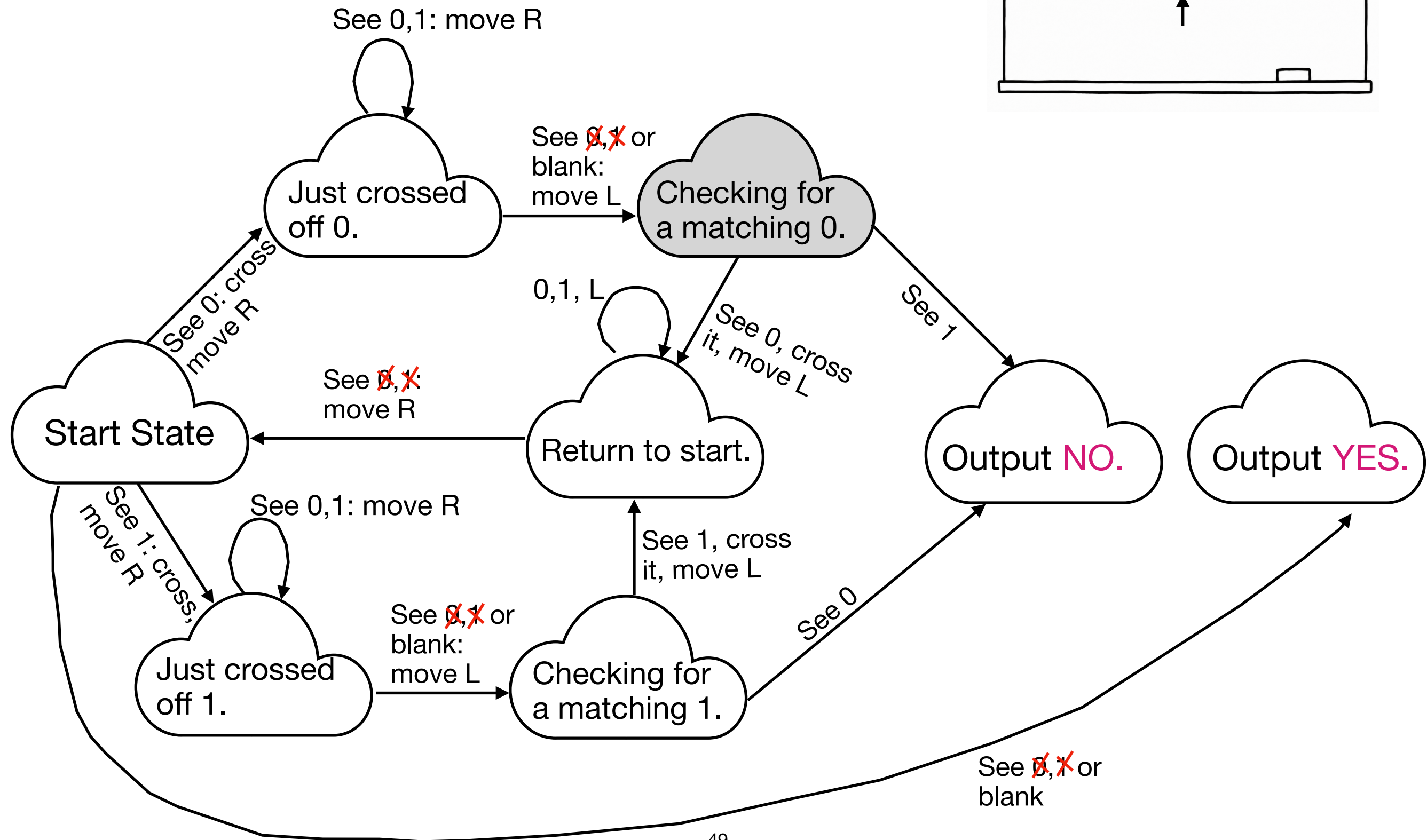
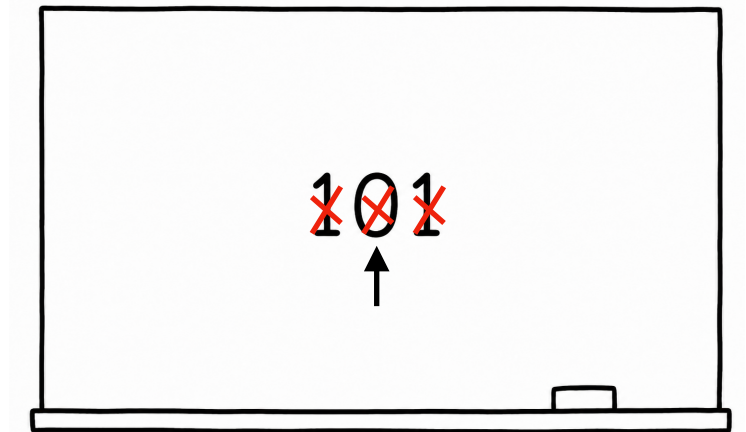
Examples



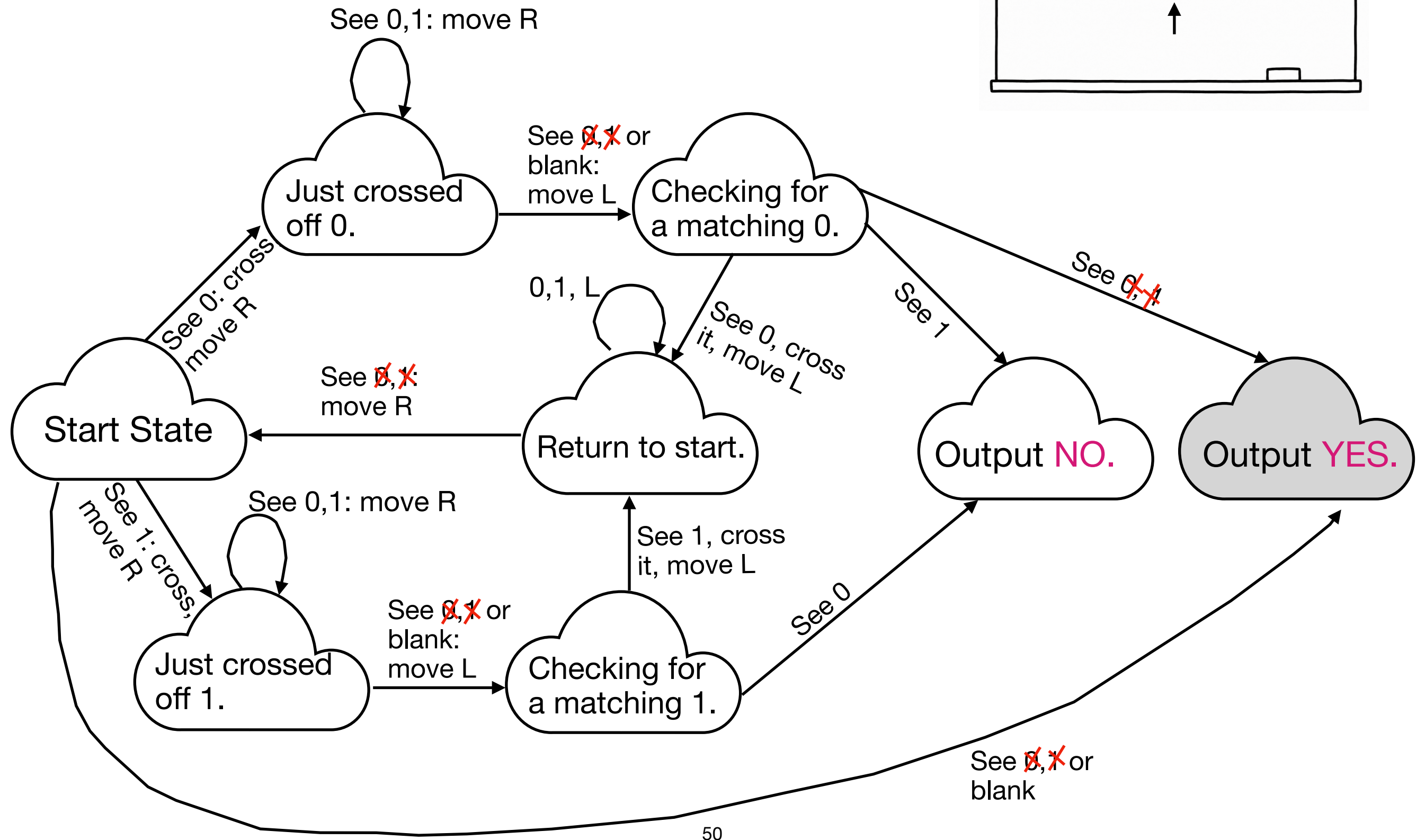
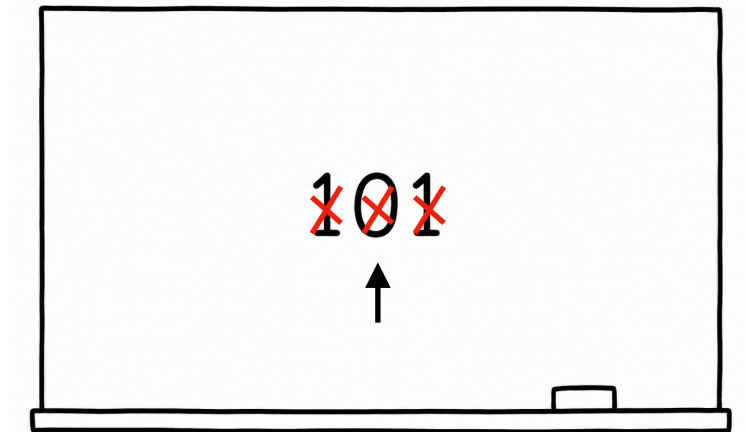
Examples



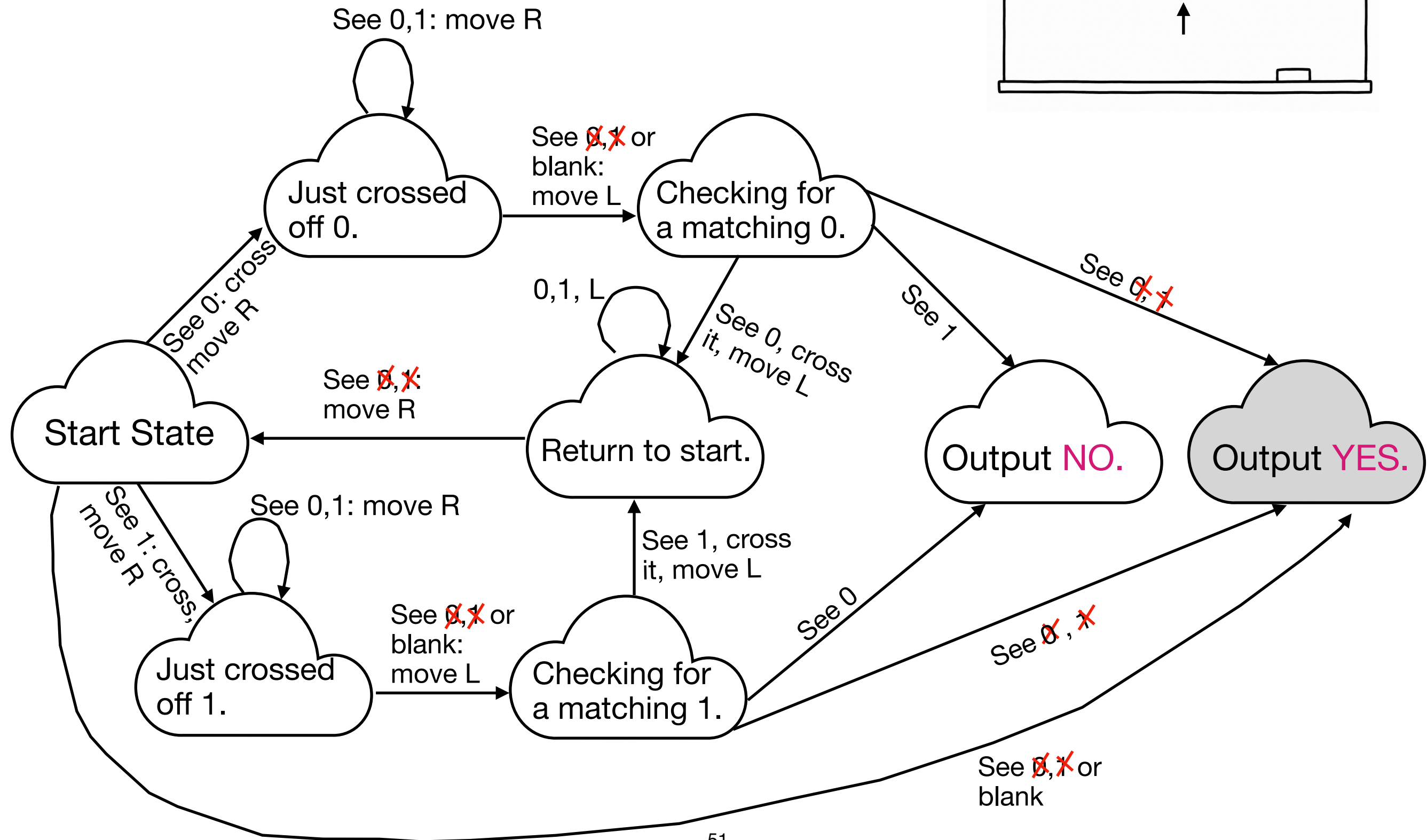
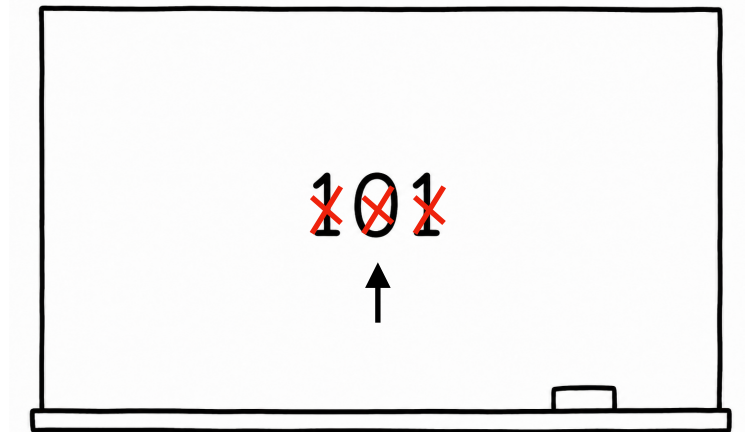
Examples



Examples



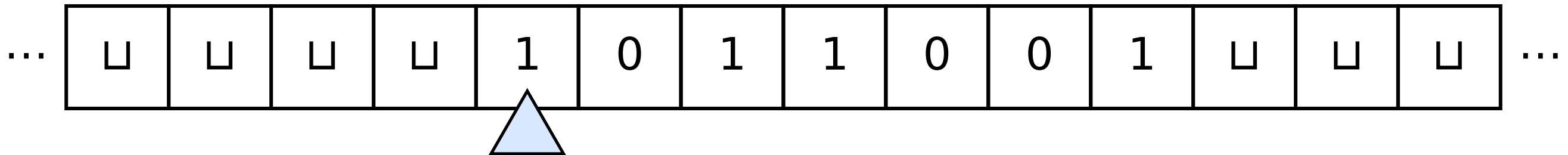
Examples



The Turing Machine Model

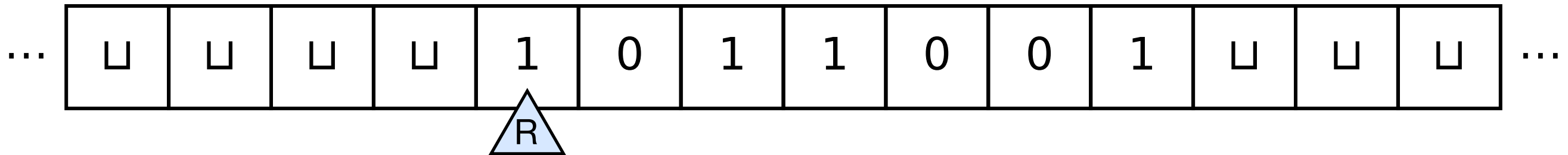
- Turing Machines: A mathematical model of **human computation**.
- The state diagram illustrates the local, step-by-step behavior that we will formalize as a Turing Machine.

The Turing Machine Model



- We imagine an infinite, one-dimensional “tape”.
- The tape divided into “cells”.
- Each cell contains a single symbol.
- There is a “**head**” pointing at one cell of the tape.
- The machine can be in one of finitely many “**internal states**”

The Turing Machine Model

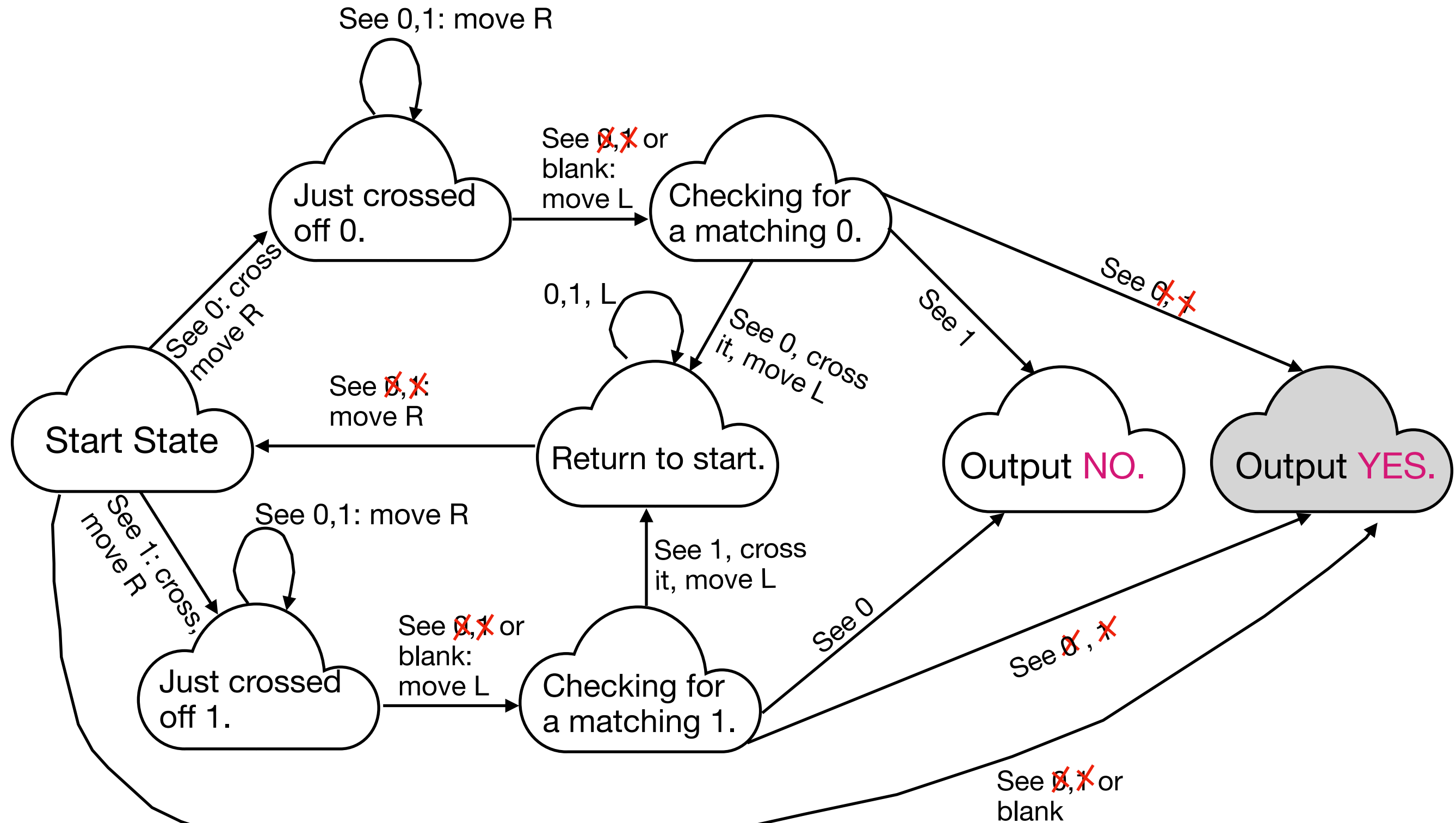


- In each step, the machine decides:
 - What to write in current cell,
 - Which cell to move in (left/right),
 - And, the next state.
- The decision is based only on the symbol being read and the current state the machine is in.

Transition Function

- Mathematically, a Turing machine is described by a **transition function**: $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$.
- Here Q is the set of states and Σ is the set of symbols.
- $\delta(q, b) = (q', b', D)$ means:
 - If we are in state q and read the symbol b .
 - Write b' (replacing b), move the head in direction D (L = left and R = right), and go to state q' .

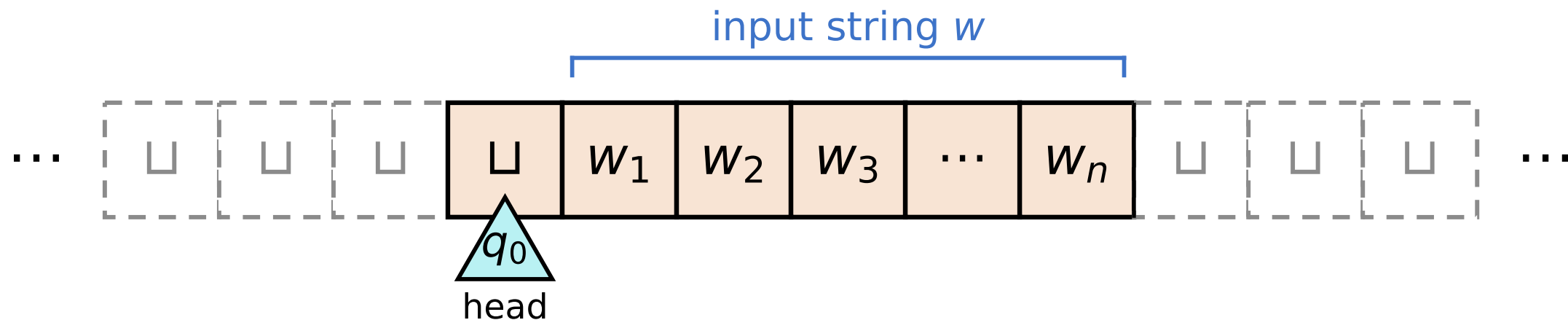
Examples



Input

- One Turing Machine represents one algorithm.
- For us, the **input** to a Turing machine will always be a **string over the binary alphabet**.

Turing Machine Initialization



- The tape initially contains an **input string** $w \in \{0,1\}^*$.
- The head is initially in a cell to the left of the input.
 - The cell contains a special “**blank symbol**” $\sqcup$.
- The machine is initially in a special “**start state**” q_0 .

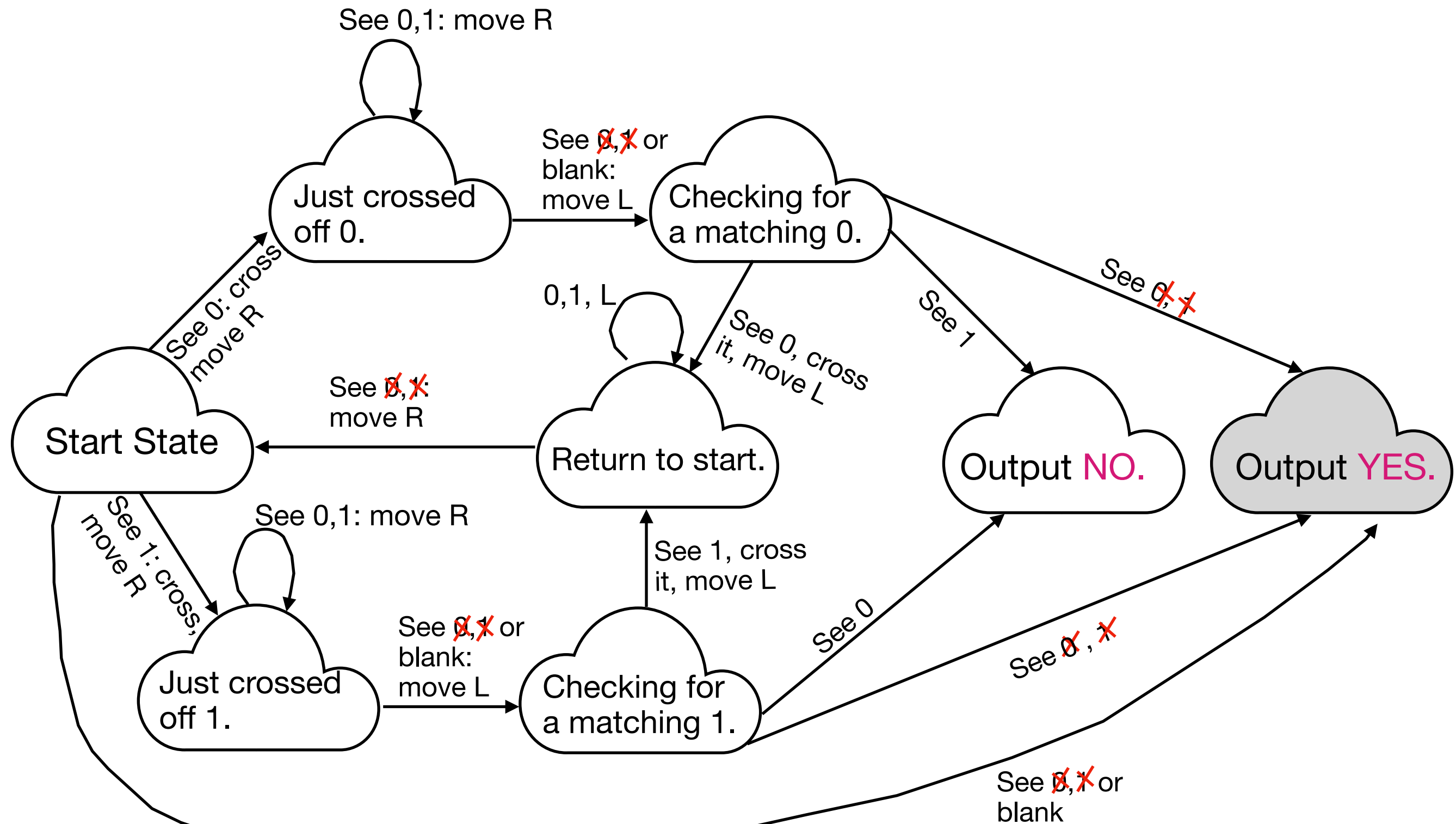
Halting

- After initialization, we repeatedly apply the transition function.
- Eventually, the computation may reach a “**halting state**”:
 - There are two halting states: q_{accept} and q_{reject} .
- In this case, the computation **halts**.

Accepting and Rejecting

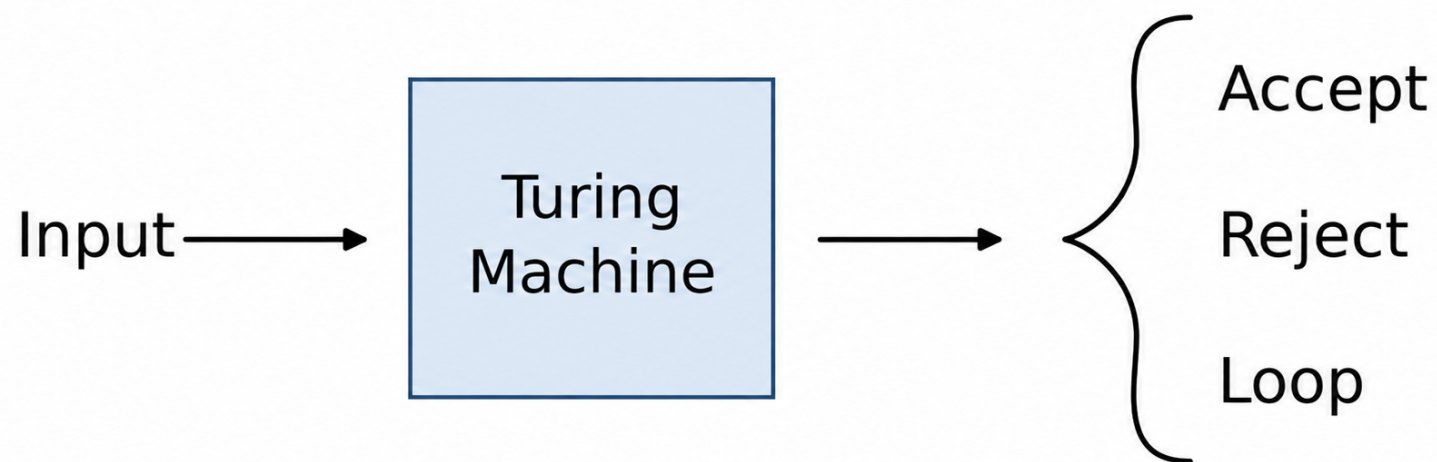
- If the machine enters q_{accept} , we say it **accepts** the input.
 - Analogy: “**return True**”.
- If the machine enters q_{reject} , we say it **rejects** the input.
 - Analogy: “**return False**”.

Examples



Looping

- It is also possible the machine runs **forever** without halting.
- In this case, we say the machine “**loops**”.



**Which problems can be solved
through computation?**

Deciding a Language

- **Recall:** In Lecture 1, decision problems = languages.
- Let M be a Turing Machine, and L be a language.
- We say M **decides** L if
 - ▶ M accepts every $x \in L$, and
 - ▶ M rejects every $x \notin L$
- This is a mathematical model of what it means to “solve a decision problem”.

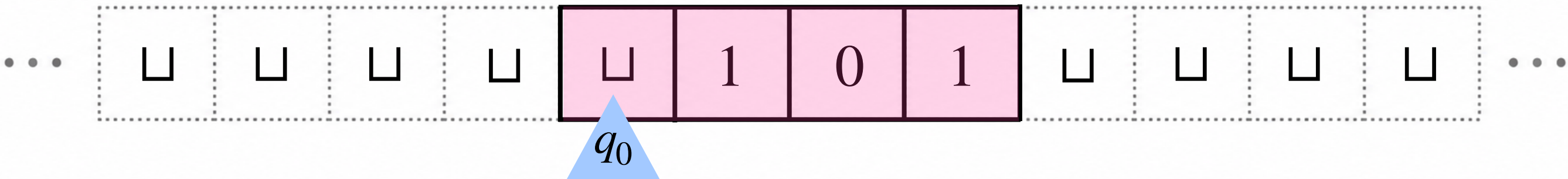
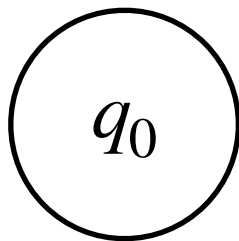
Example: Parity

- **Informal Problem Statement:** “Given $w \in \{0,1\}^*$, determine whether the number of ones in w is even or odd”.
- **Can be formulated as:**

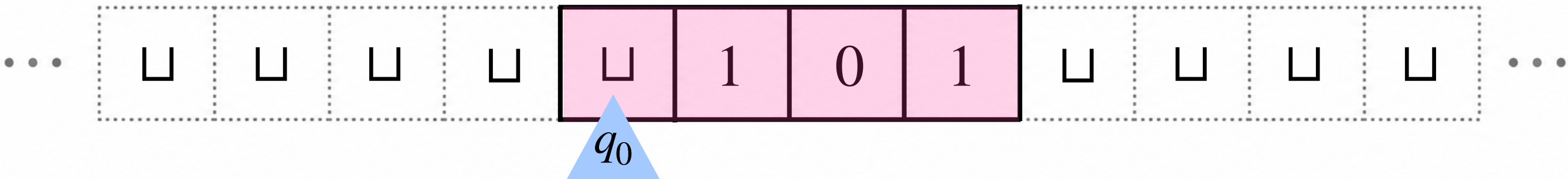
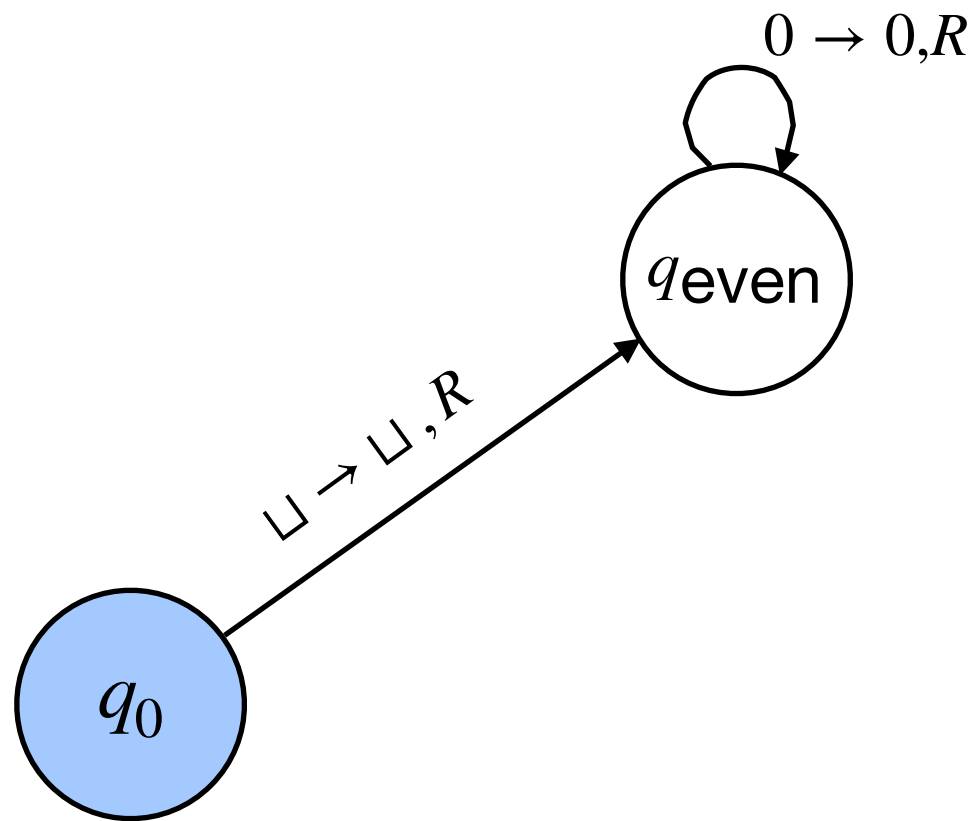
$\text{PARITY} = \{w \in \{0,1\}^* \text{ such that } w \text{ has an odd number of 1's}\}$

- Design a Turing Machine for PARITY.

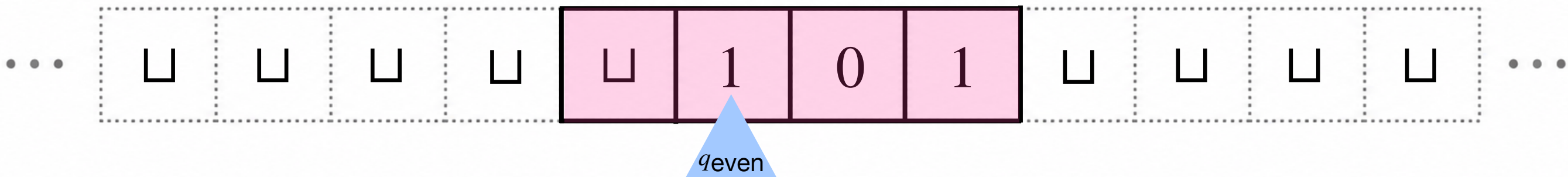
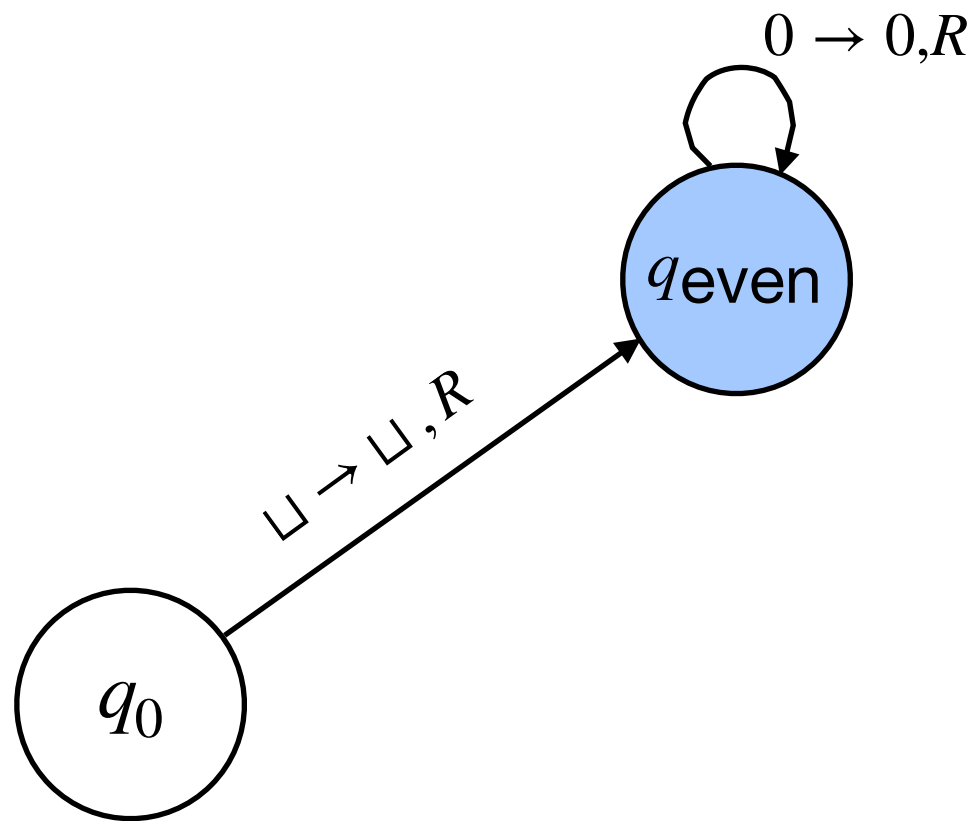
Example: Parity



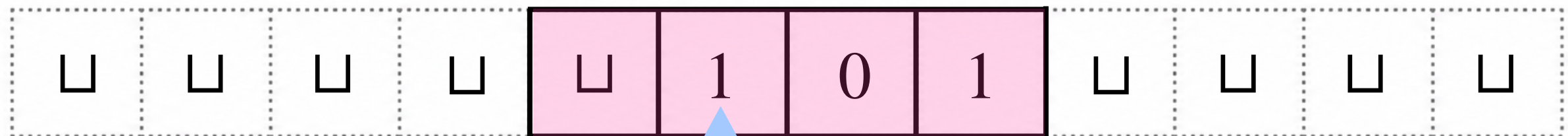
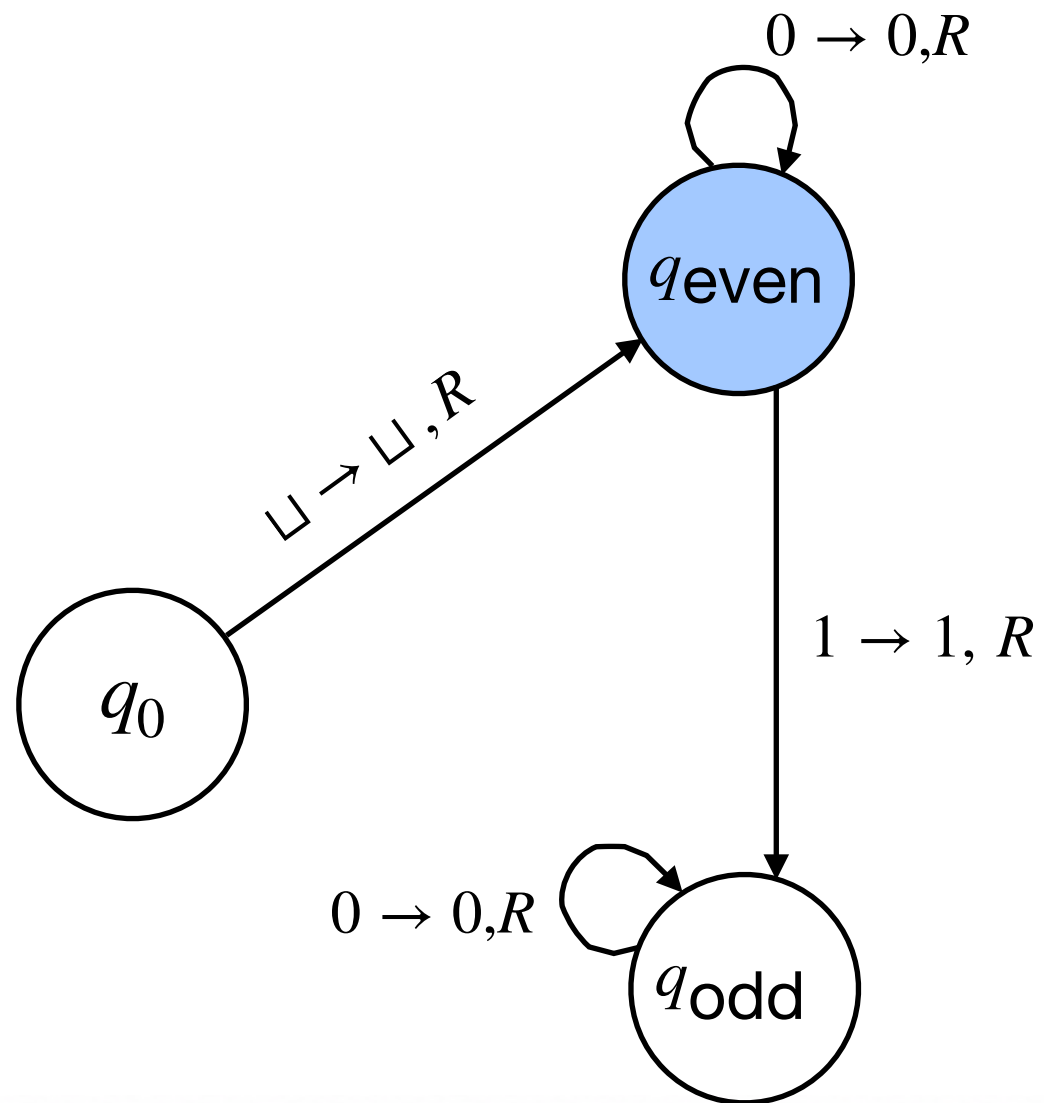
Example: Parity



Example: Parity

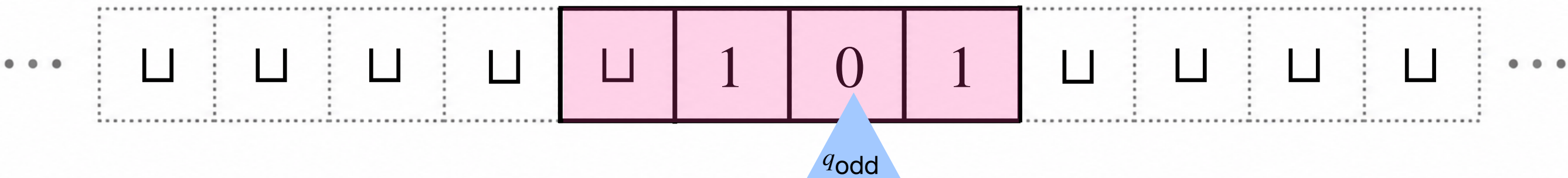
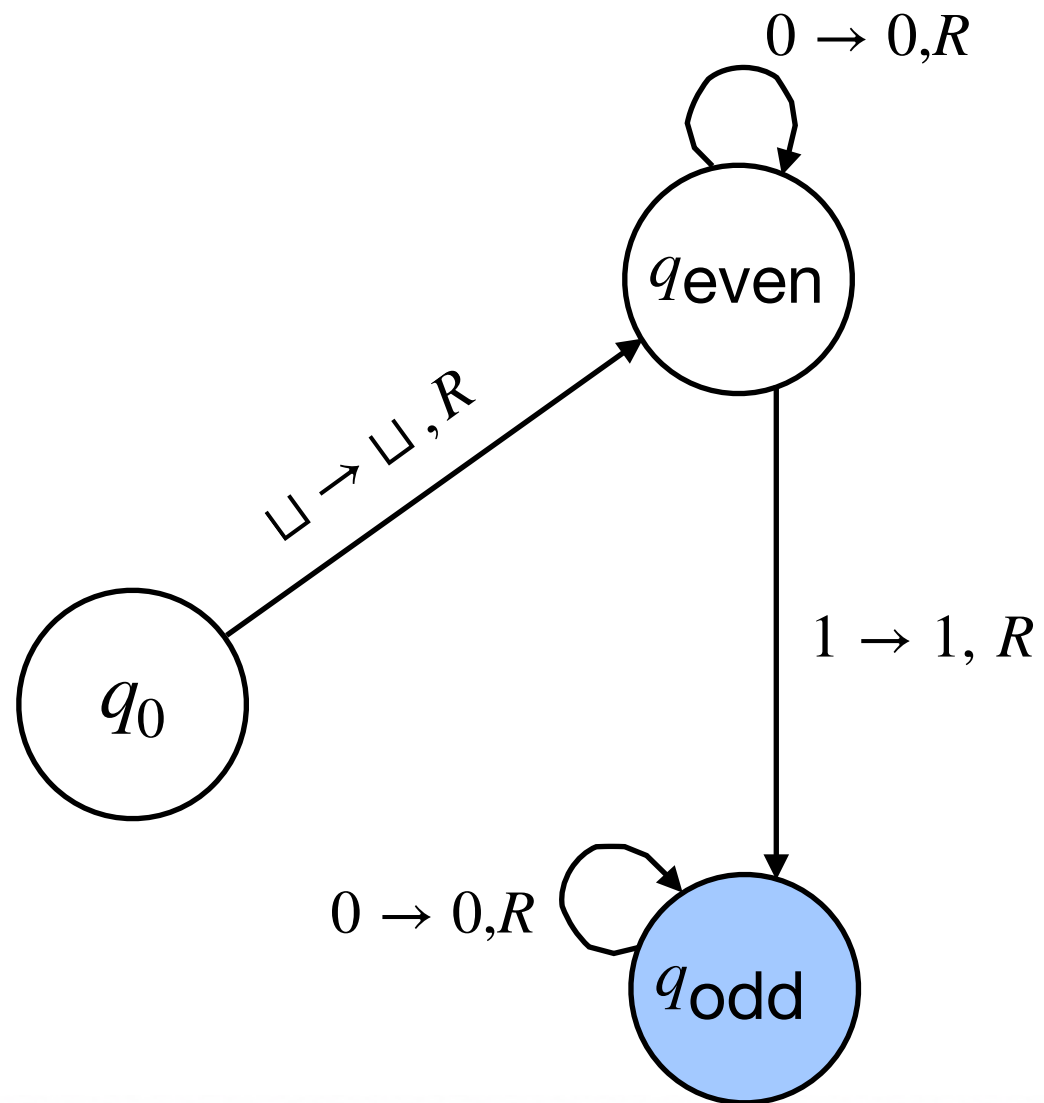


Example: Parity

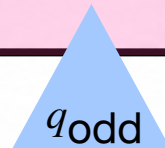
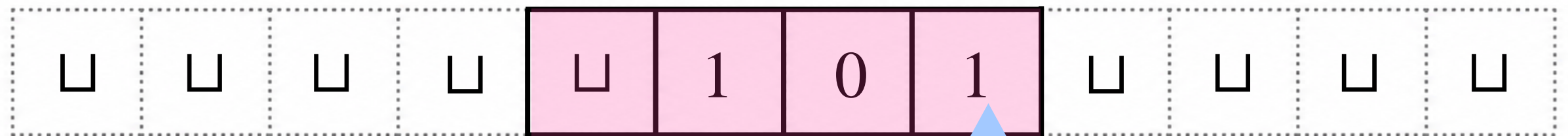
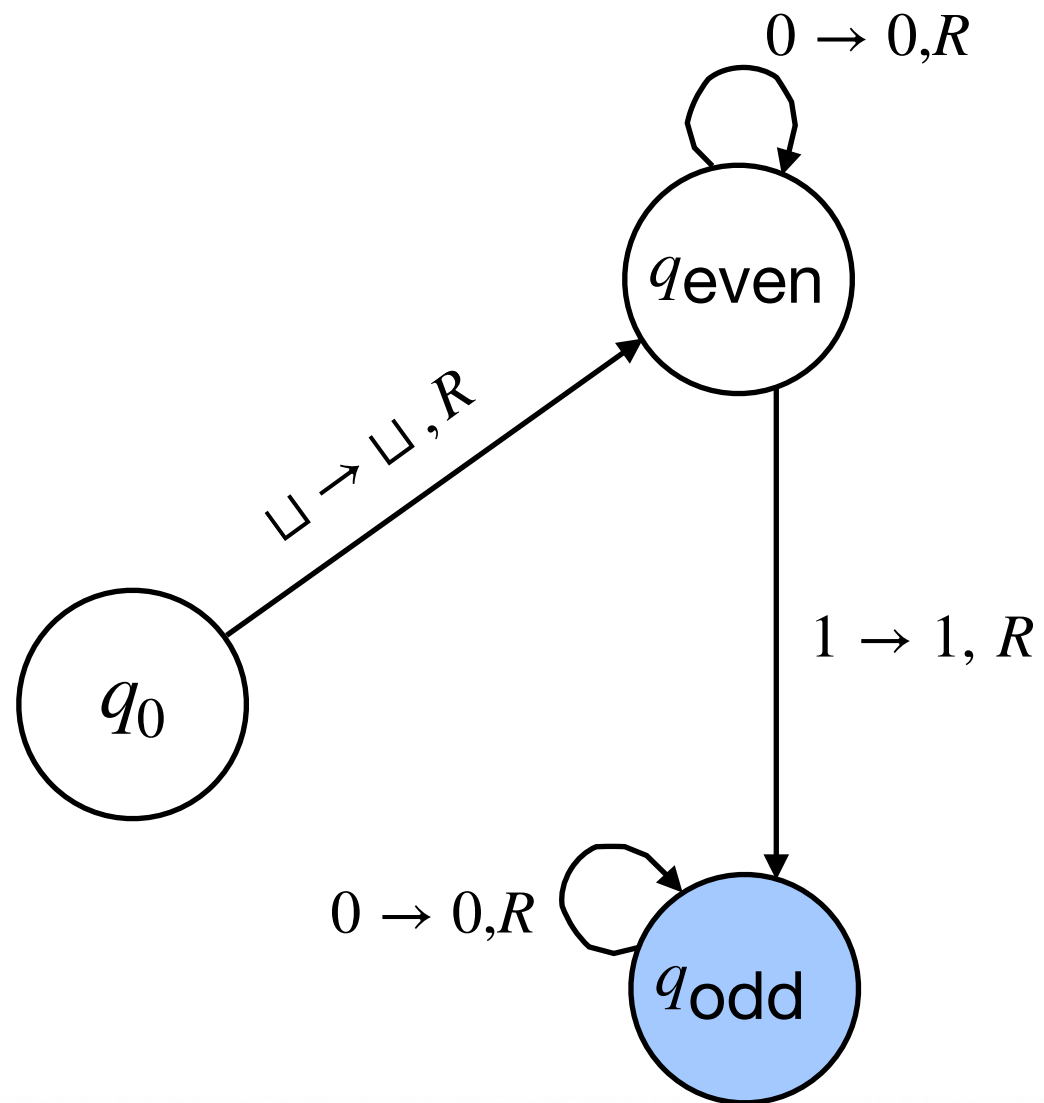


q_{even}

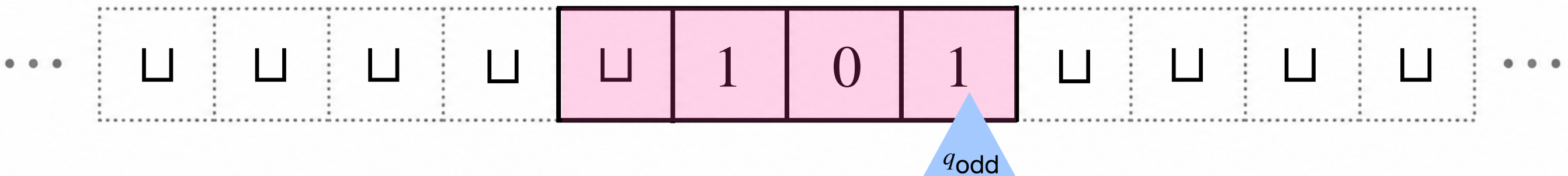
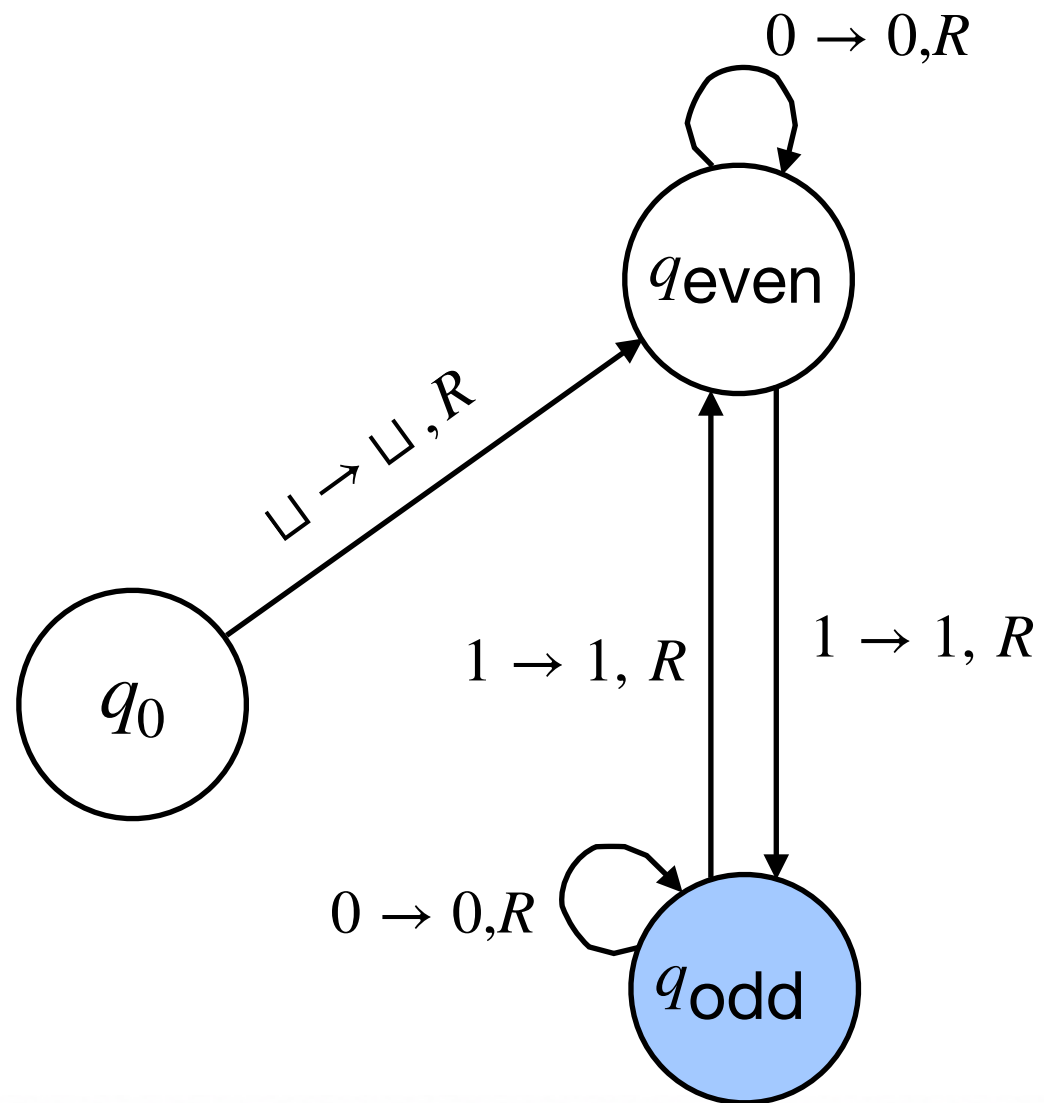
Example: Parity



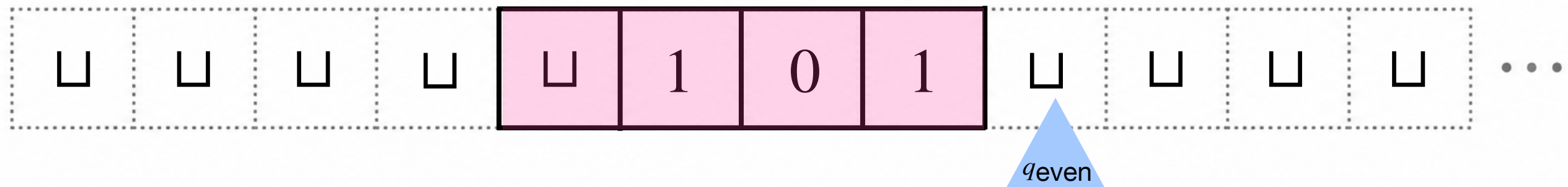
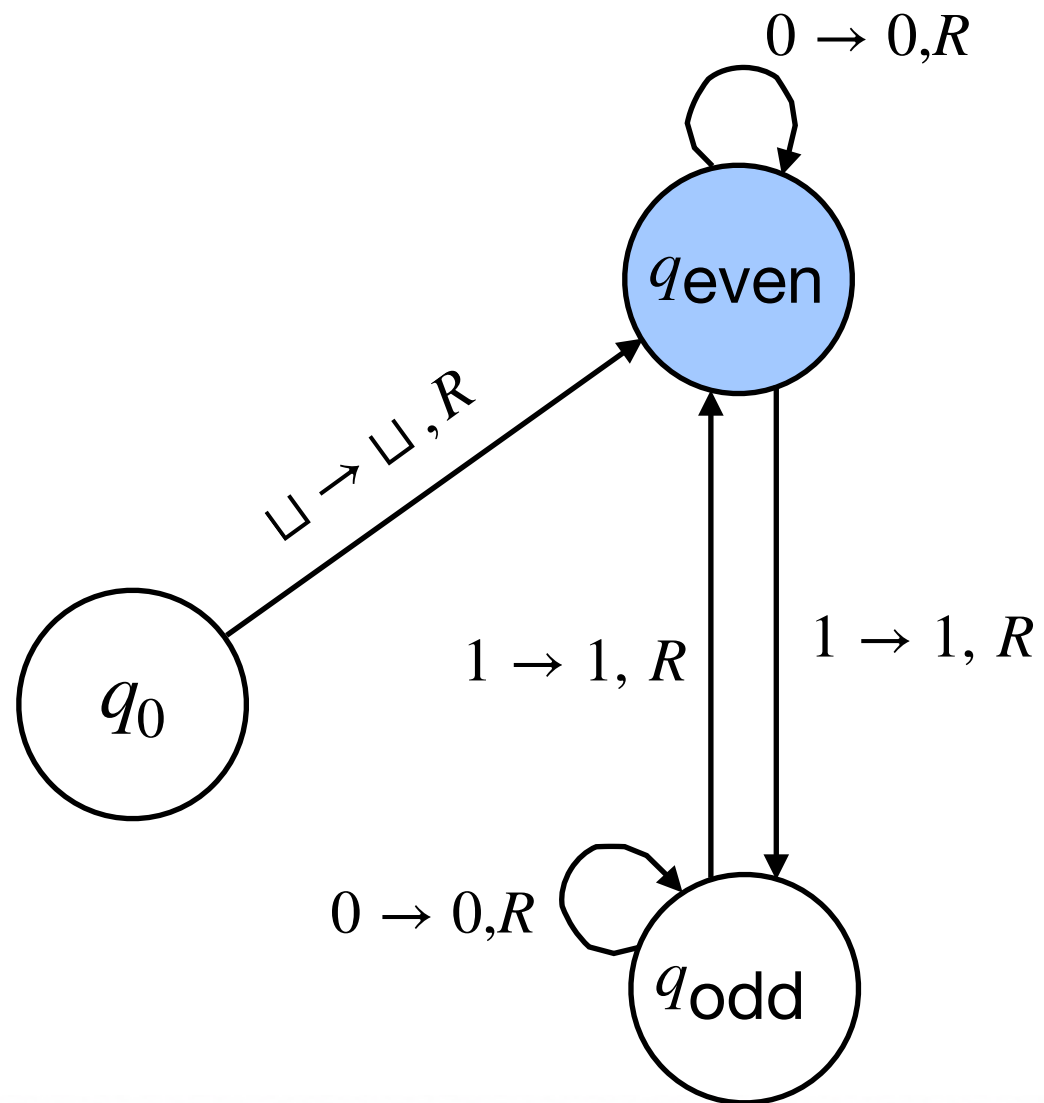
Example: Parity



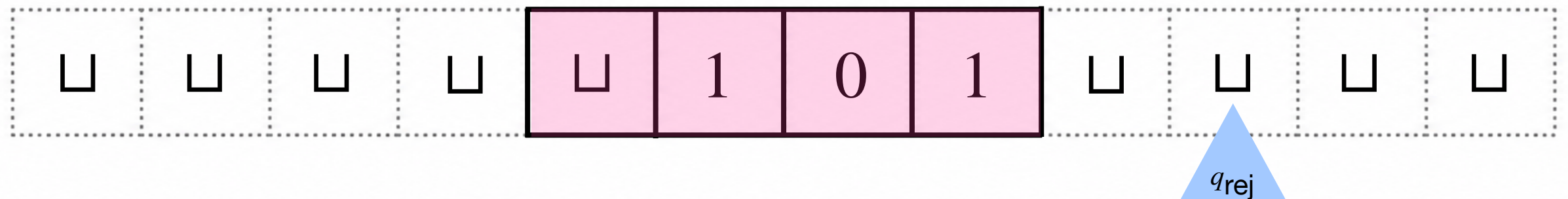
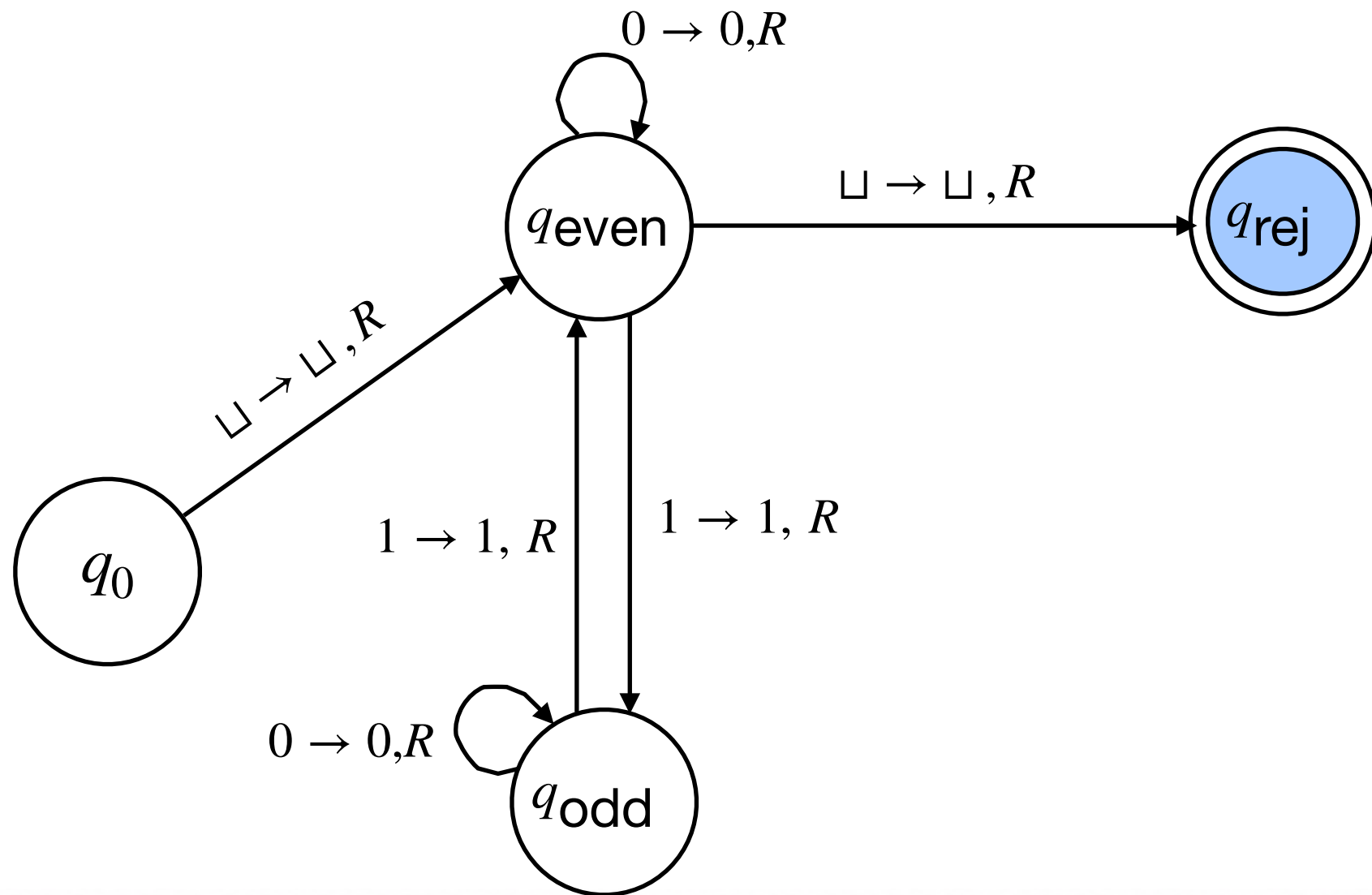
Example: Parity



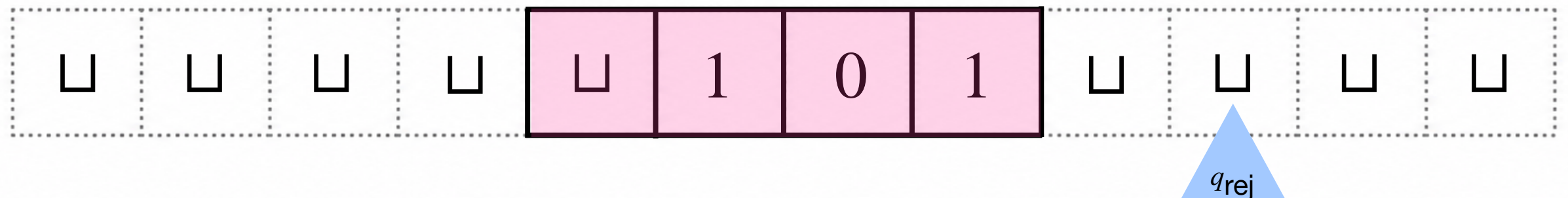
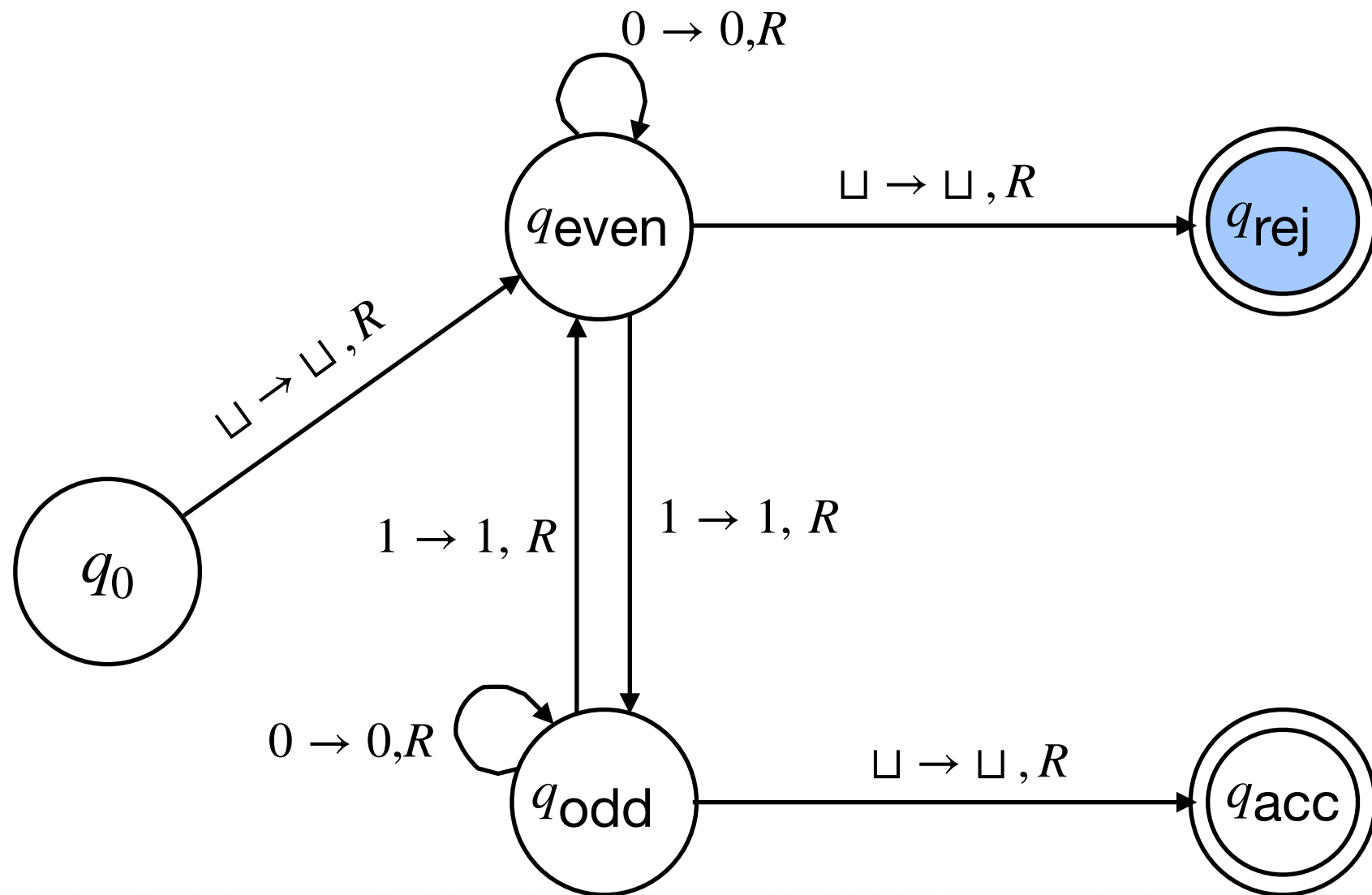
Example: Parity



Example: Parity



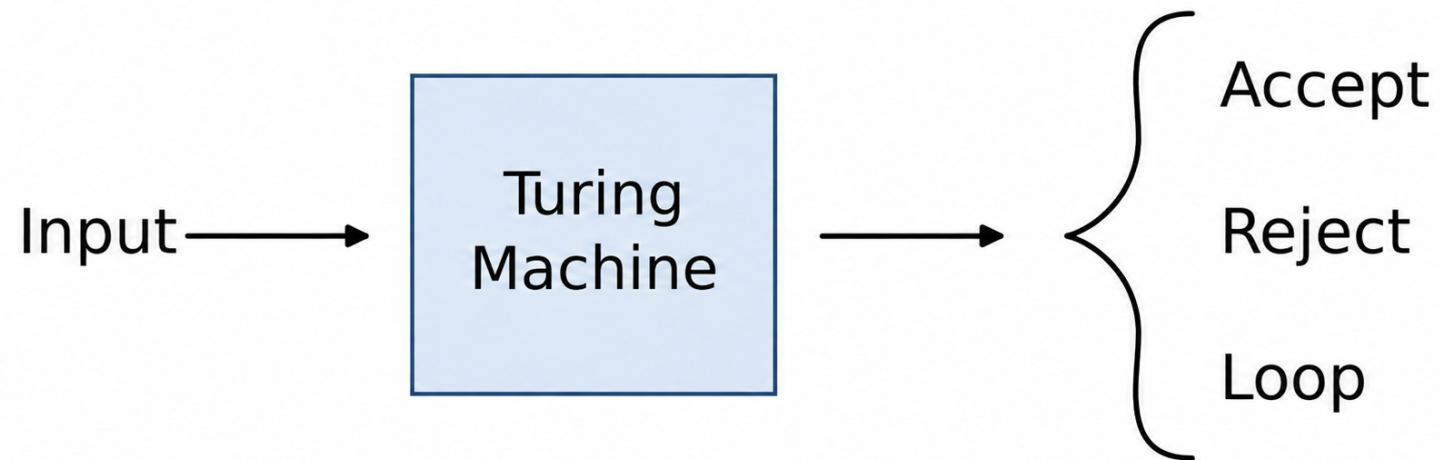
Example: Parity



Decidable Languages

- A language L is **decidable** if and only if there exists a Turing Machine M that decides it.
- Otherwise, it is **undecidable**.

Recognizing a Language



- The Turing Machine M **recognizes** the language $L \subseteq \{0,1\}^*$ if for every $x \in \{0,1\}^*$, M accepts x if and only if $x \in L$.
- Note that by this definition, the machine M can reject or loop forever on inputs $x \notin L$.

Turing Machine as a Recognizer

- Every Turing Machine M recognizes some language $L(M) = \{x \in \{0,1\}^* \text{ such that } M \text{ accepts } x\}$.
- Not every Turing machine decides a language.

Decidable and Recognizable Languages

- Decidable Languages $\subseteq$ Recognizable Languages
- Is this a proper subset?
- What is an example of an undecidable language?

Summary

- Turing Machines: a mathematical model of computation:
 - an unbounded **tape**,
 - a head that moves **left/right**, and **reads/writes** one cell,
 - has finitely many **states**
 - each decision is based on the state and the current symbol being read. Modeled as a **transition function** $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$.
 - On input a Turing Machine may **accept/reject/loop**.

Summary

- M **decides** a language L if it accepts every $x \in L$ and rejects every $x \notin L$.
- M **recognizes** a language L if it accepts $x \iff x \in L$.
- A language L is **decidable** if and only if there exists a Turing machine that decides it. Otherwise it is **undecidable**.
- A language L is **recognizable** if and only if there exists a Turing machine that recognizes it.

Next Week: Formal Definition of TMs, and More Decidable Languages.