

Turing Machines

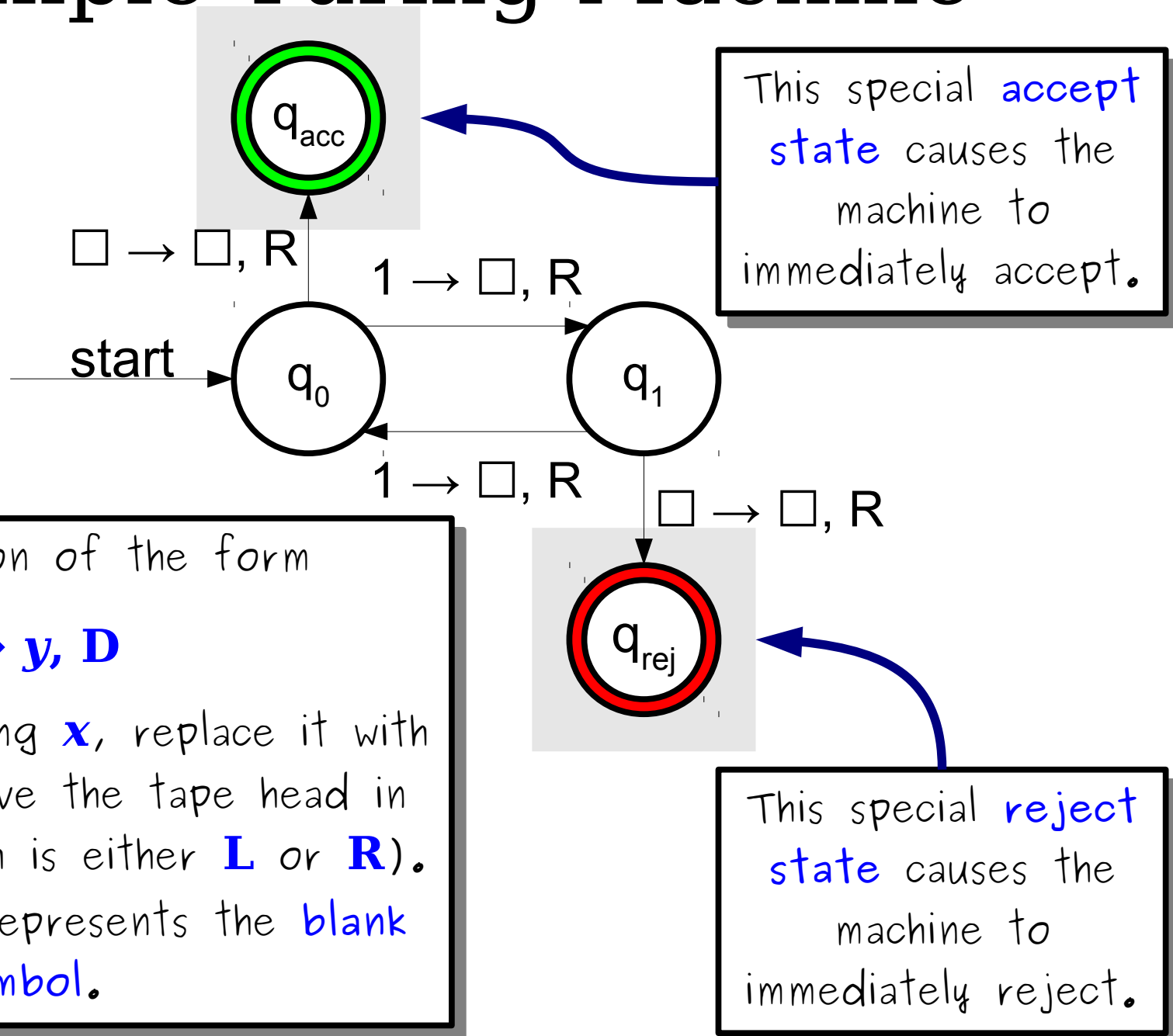
Part Two

Recap from Last Time

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.

A Simple Turing Machine



Each transition of the form

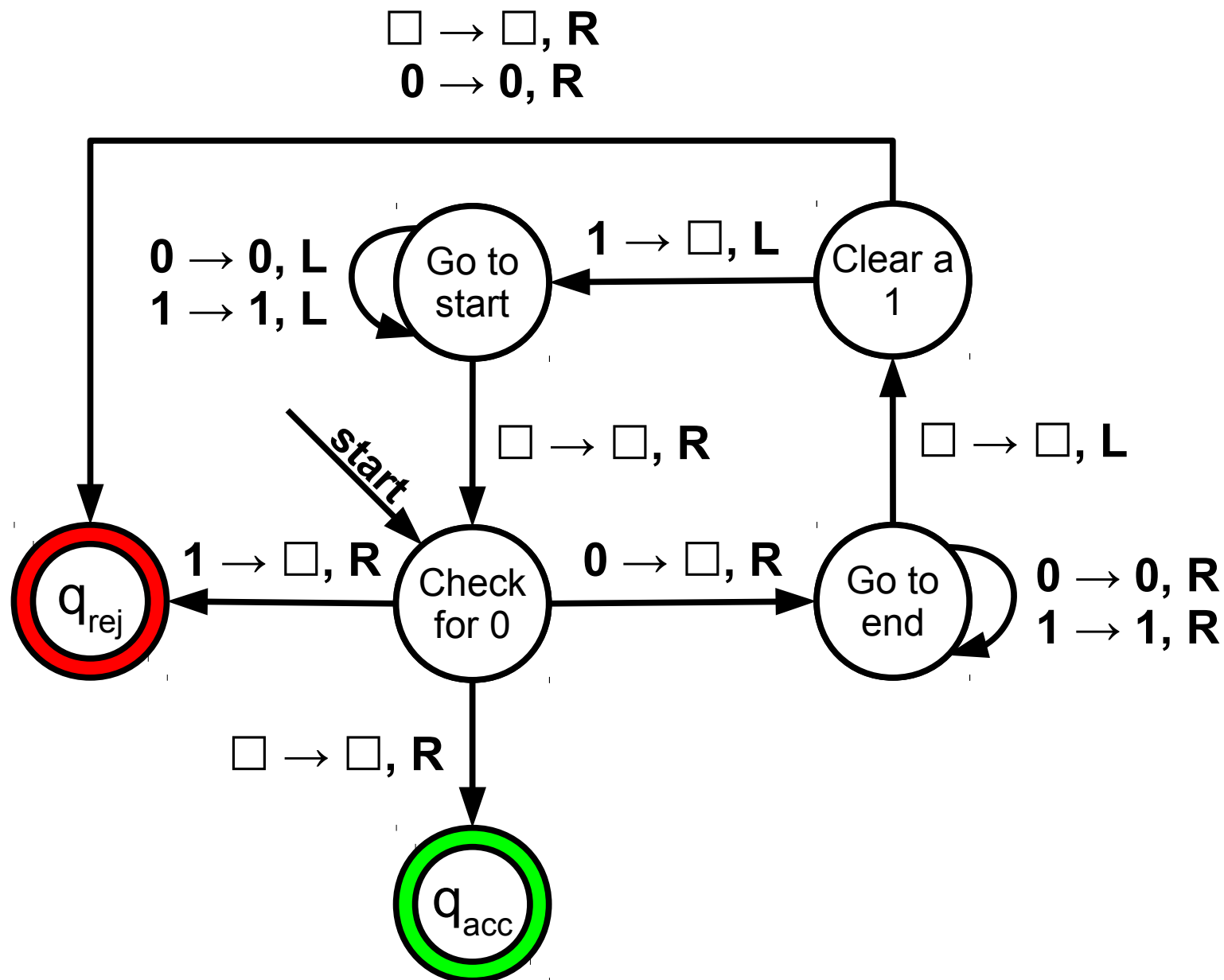
$x \rightarrow y, D$

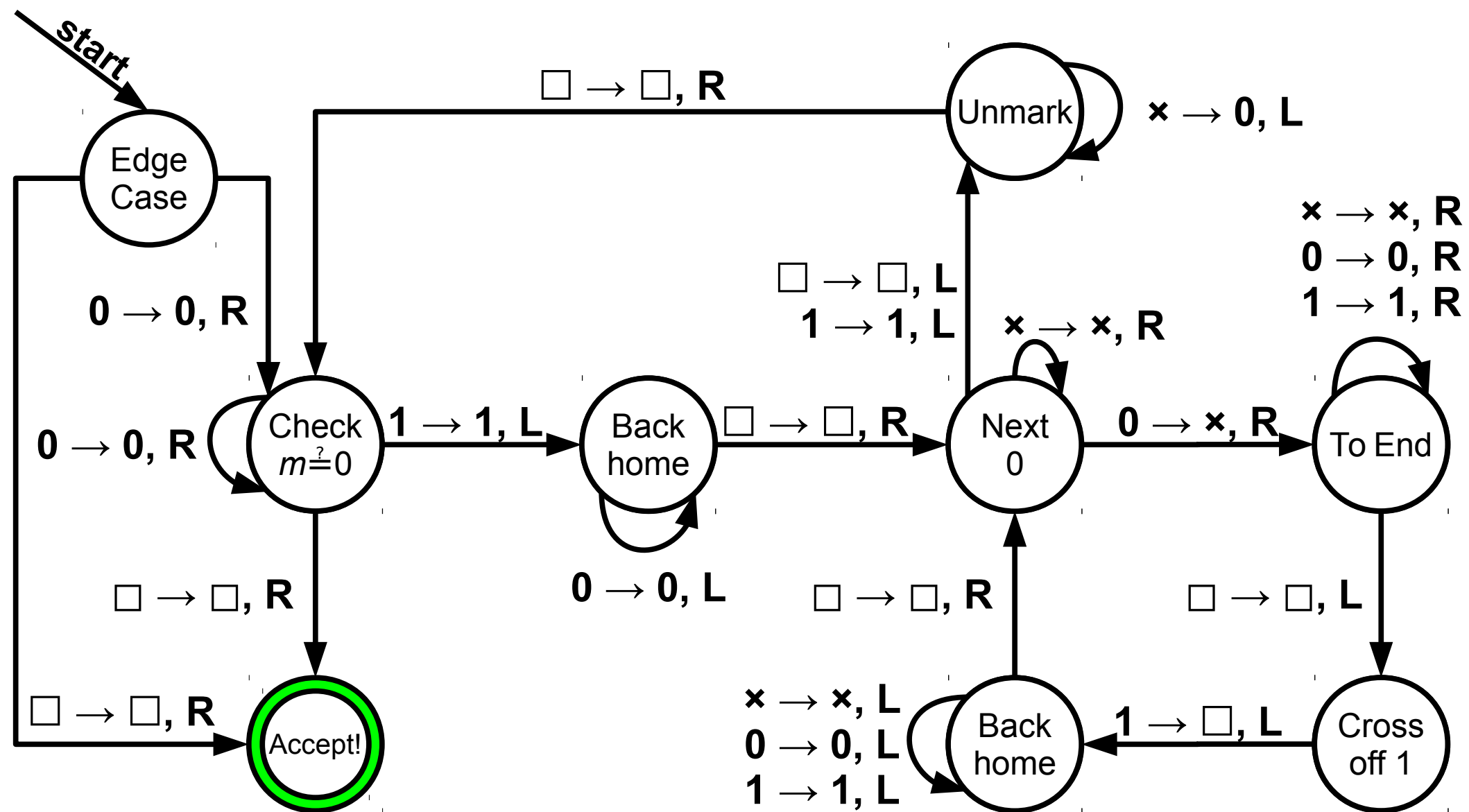
means "upon reading x , replace it with symbol y and move the tape head in direction D (which is either **L** or **R**).

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

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.





New Stuff!

Main Question for Today:

Just how powerful are Turing machines?

Another TM Design

- Last time, we designed a TM for this language over $\Sigma = \{0, 1\}$:

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

- Let's do a quick review of how it worked.

A Leap of Faith



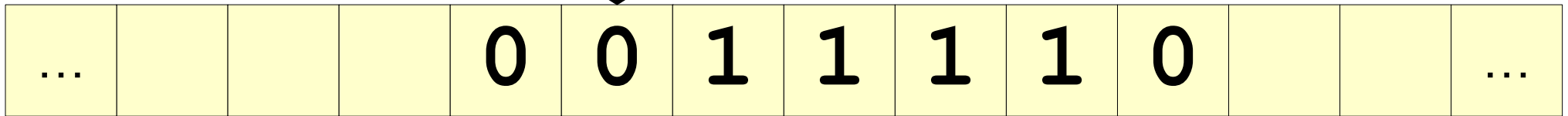
...			0	0	0	1	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

A Leap of Faith

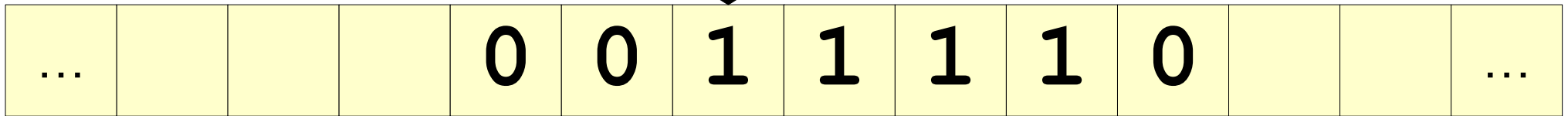


...				0	0	1	1	1	1	0			...
-----	--	--	--	---	---	---	---	---	---	---	--	--	-----

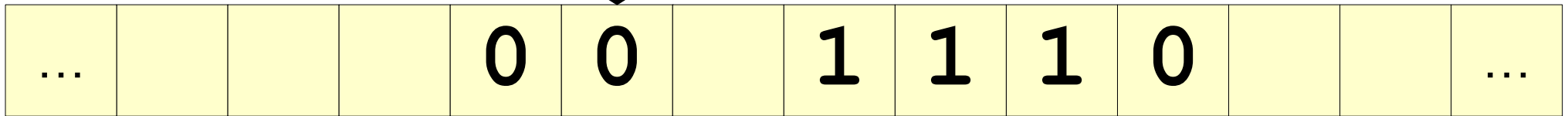
A Leap of Faith



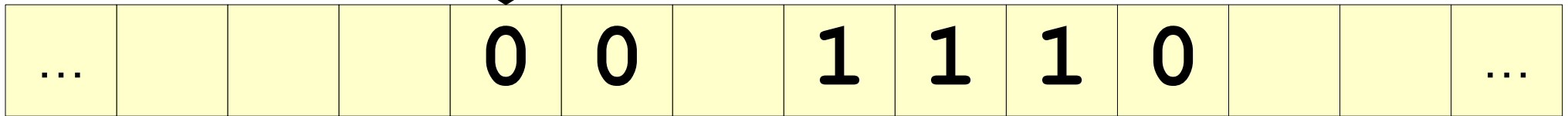
A Leap of Faith



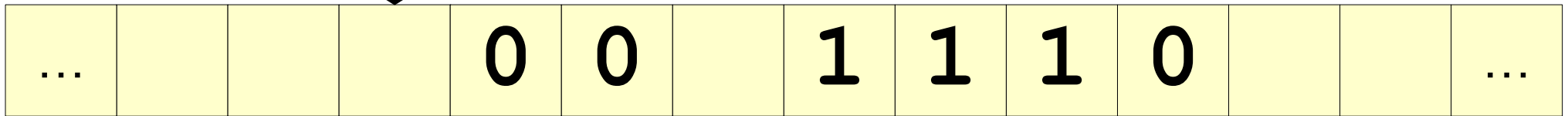
A Leap of Faith



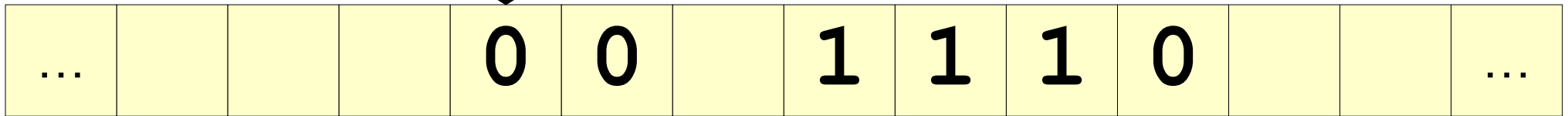
A Leap of Faith



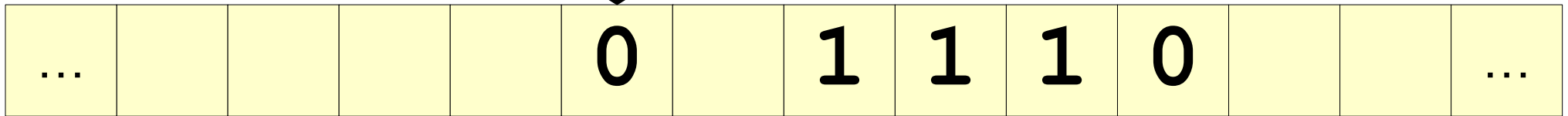
A Leap of Faith



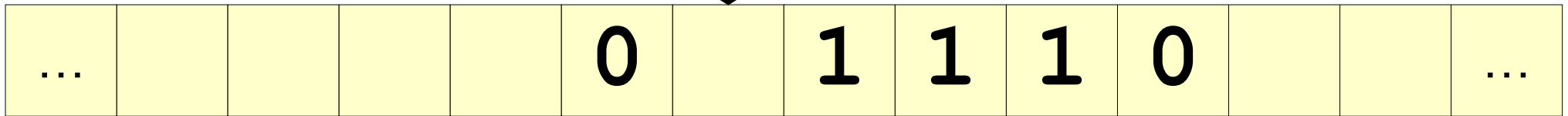
A Leap of Faith



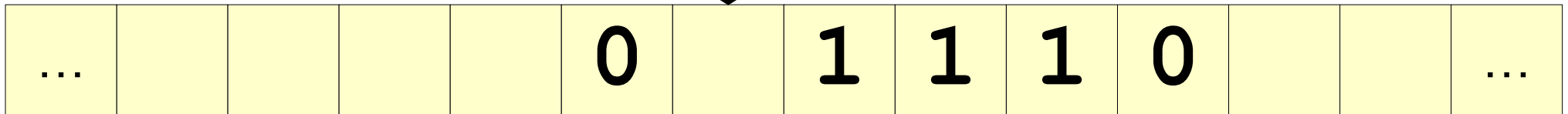
A Leap of Faith



A Leap of Faith



A Leap of Faith



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

The Solution



...			0	0	0	1	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			×	0	0	1	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			×	0	0	1	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			×	0	0	1	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			×	0	0	×	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	0	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	0	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			×	0	0	×	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	0	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	0	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	x	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	x	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	x	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	x	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	x	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

The Solution



...			x	x	0	x	x	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

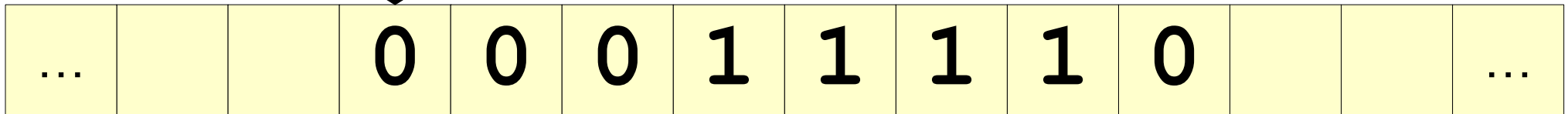
A Different Idea

A Different Strategy



...			0	0	0	1	1	1	1	0			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

A Different Strategy

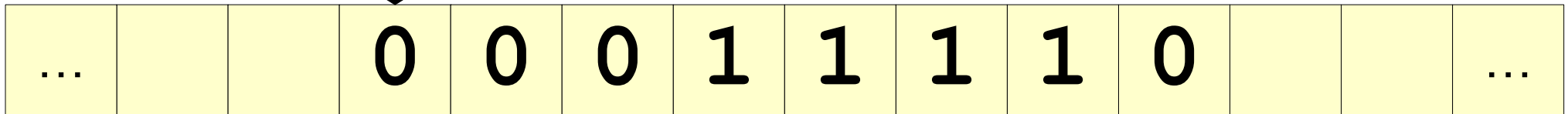


Last time, we built a machine that checks whether a string has the form 0^n1^n .

That machine almost solves this problem, but requires that the characters have to be in order.

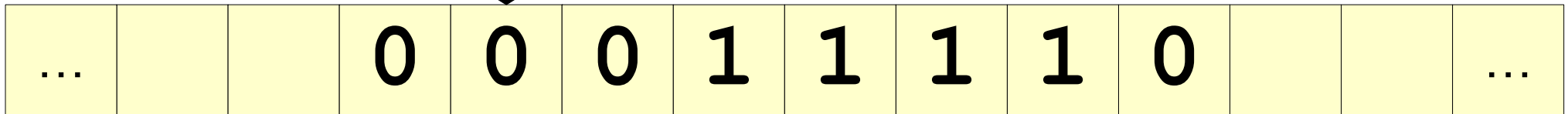
What if we sorted the input?

A Different Strategy



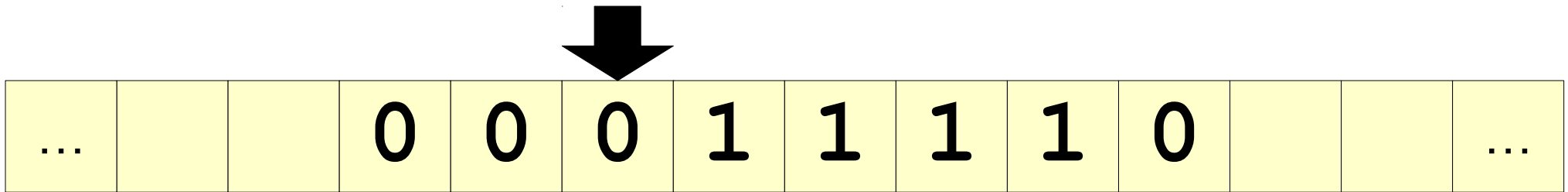
Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



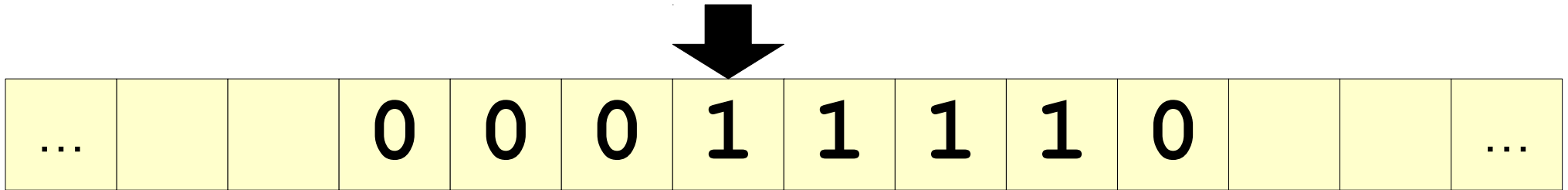
Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



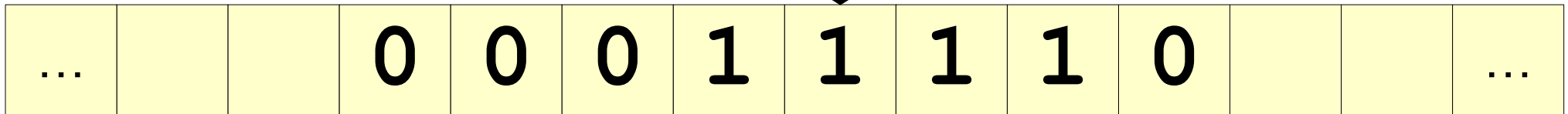
Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



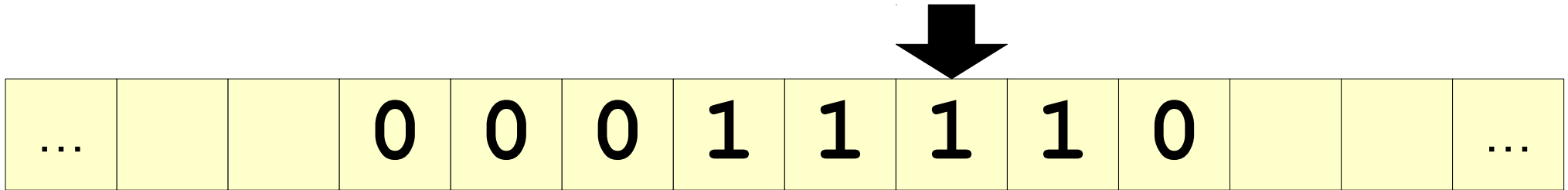
Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



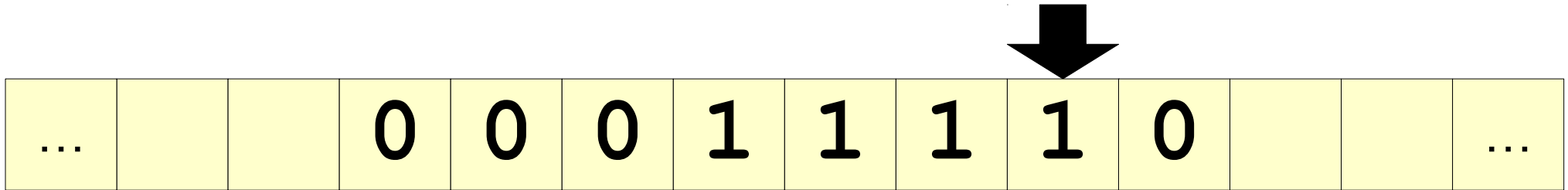
Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



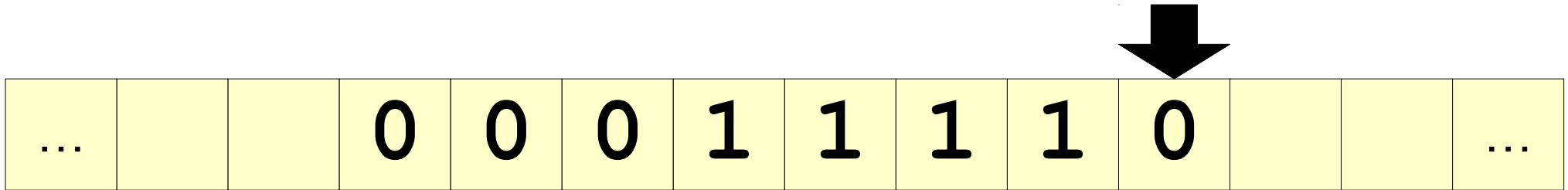
Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



Observation 1: A string
of 0s and 1s is sorted
if it matches the regex
 0^*1^* .

A Different Strategy



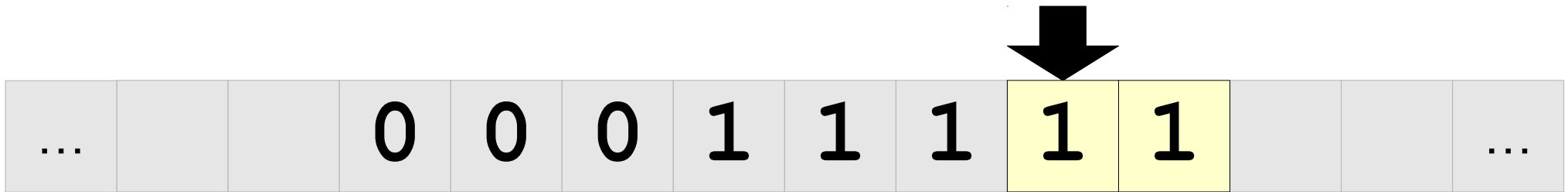
Observation 1: A string of 0s and 1s is sorted if it matches the regex 0^*1^* .

A Different Strategy



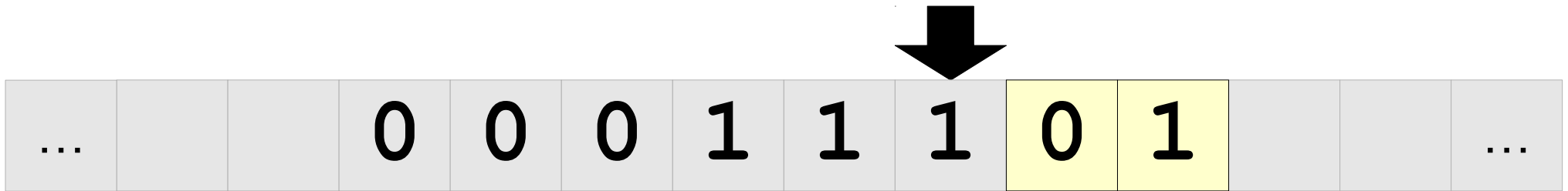
Observation 2: A string of 0s and 1s is not sorted if it contains 10 as a substring.

A Different Strategy



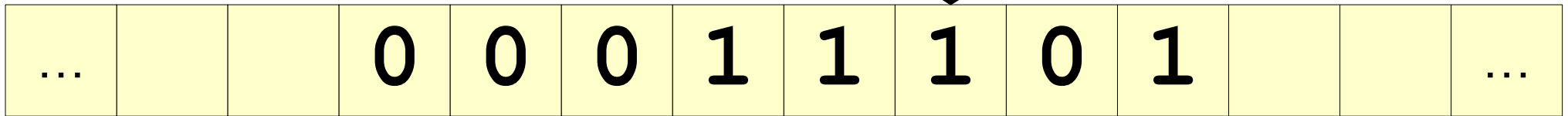
Observation 2: A string of 0s and 1s is not sorted if it contains 10 as a substring.

A Different Strategy



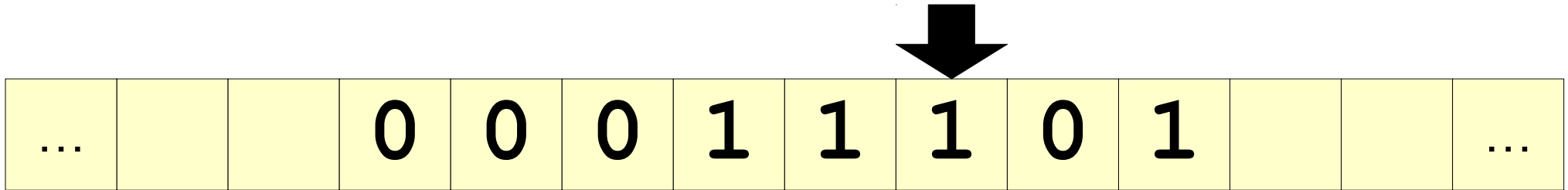
Observation 2: A string of 0s and 1s is not sorted if it contains 10 as a substring.

A Different Strategy



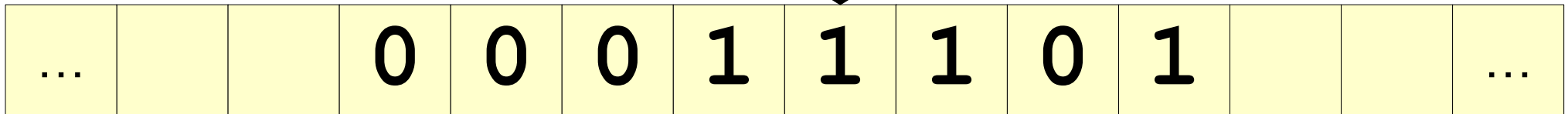
Observation 2: A string of 0s and 1s is not sorted if it contains 10 as a substring.

A Different Strategy



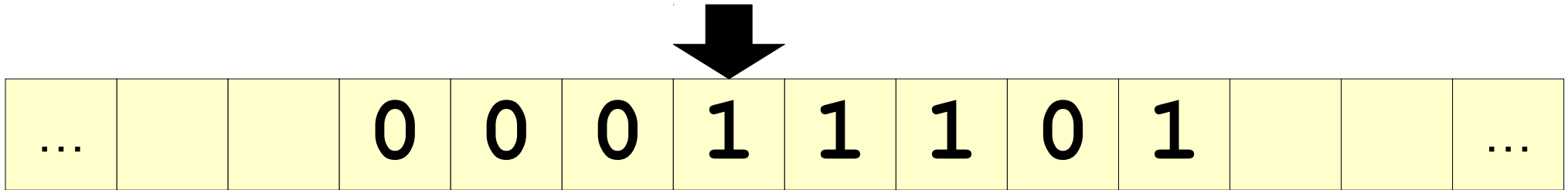
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



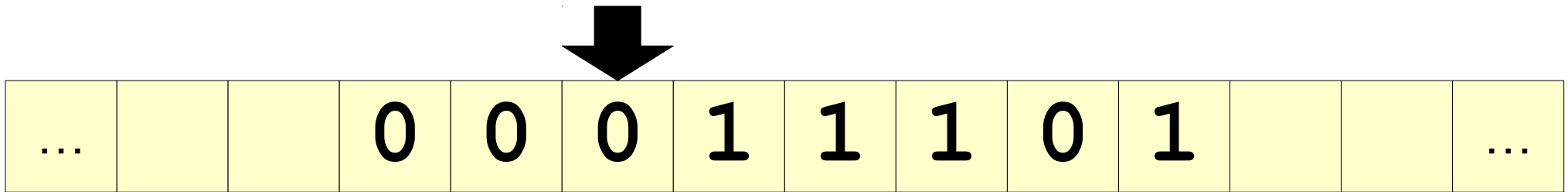
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



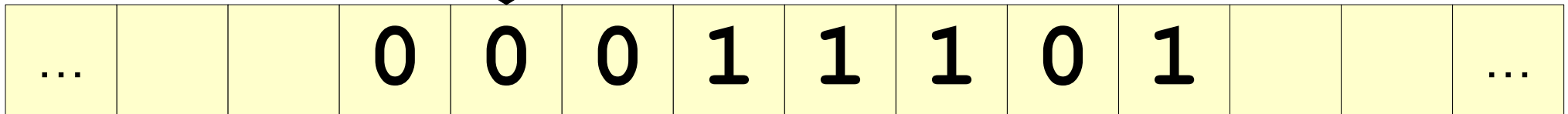
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



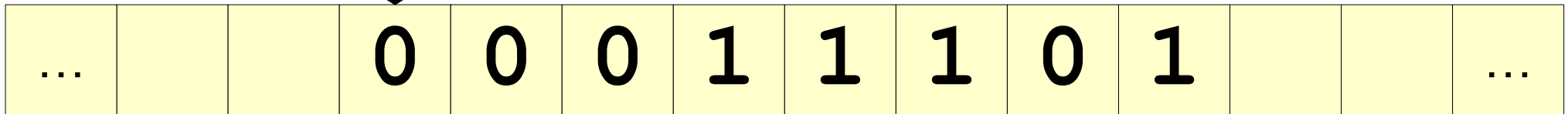
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



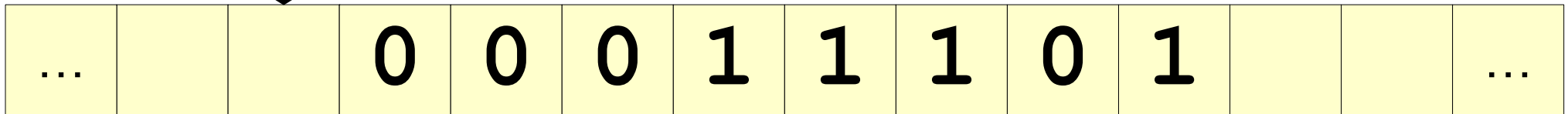
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



Idea: Repeatedly find a copy of 10 and replace it with 01.

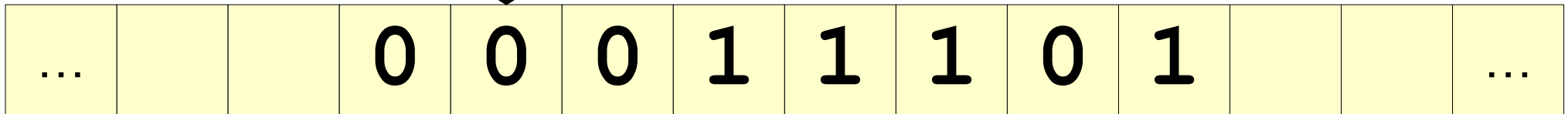
A Different Strategy



...			0	0	0	1	1	1	0	1			...
-----	--	--	---	---	---	---	---	---	---	---	--	--	-----

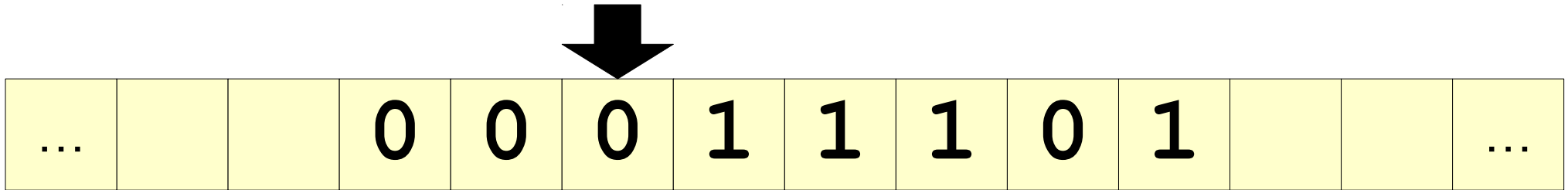
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



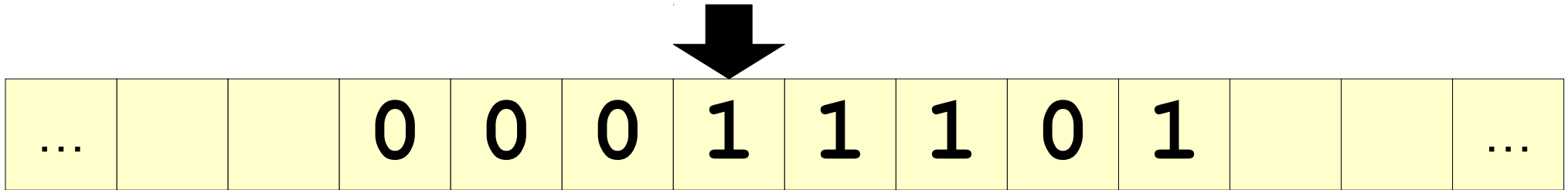
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



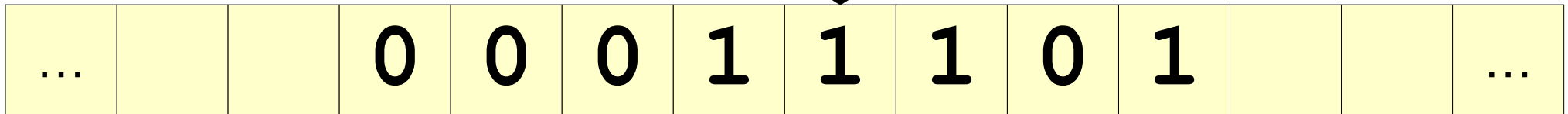
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



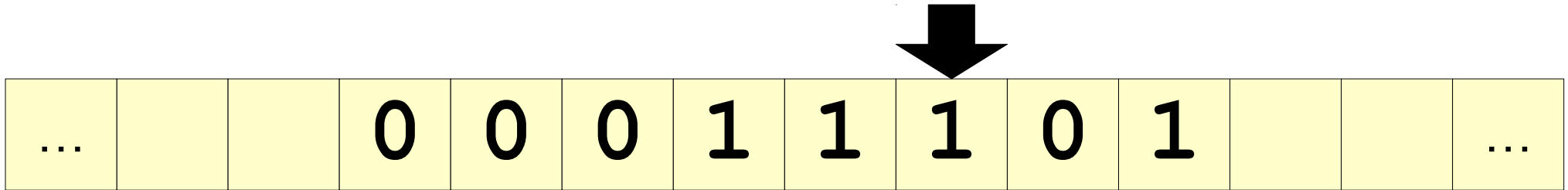
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



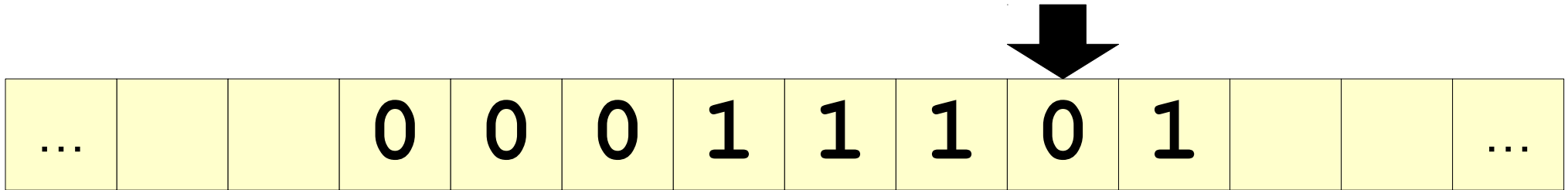
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



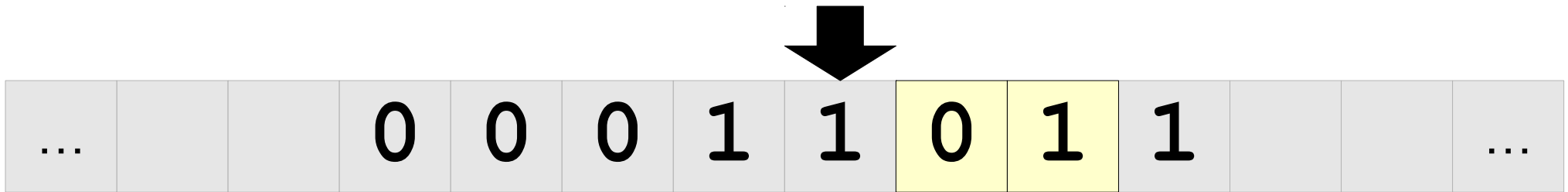
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



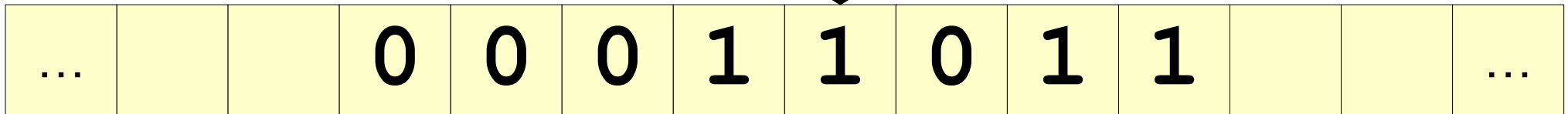
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



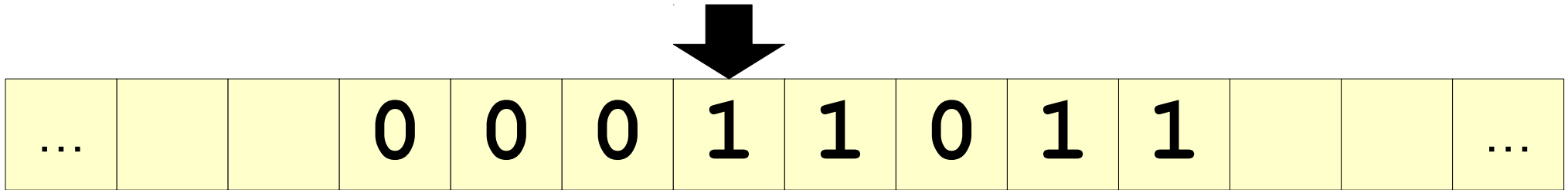
Idea: Repeatedly find a copy of 10 and replace it with 01.

A Different Strategy



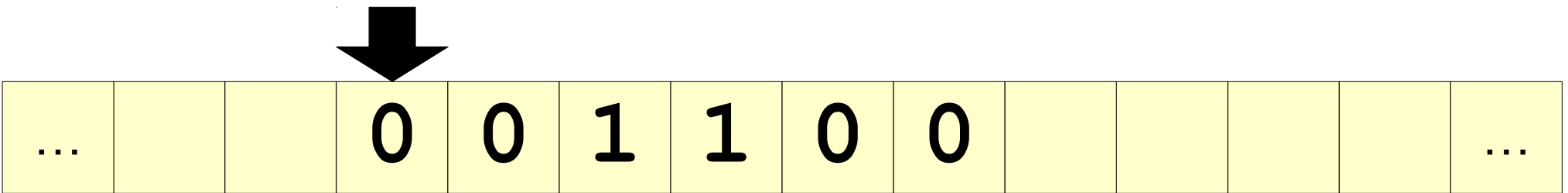
Idea: Repeatedly find a copy of 10 and replace it with 01.

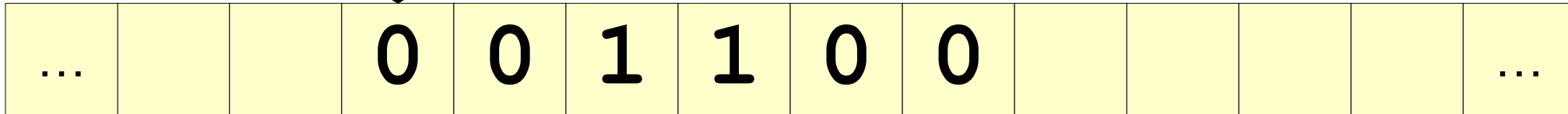
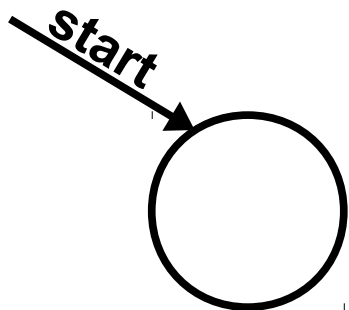
A Different Strategy

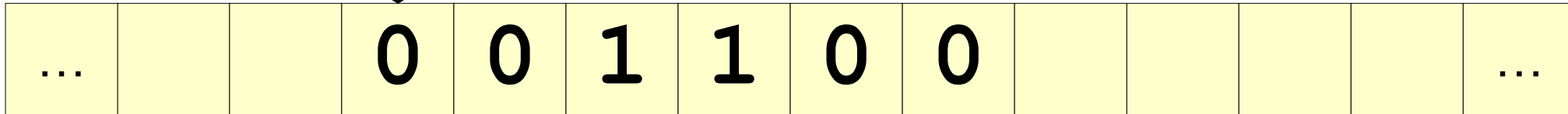
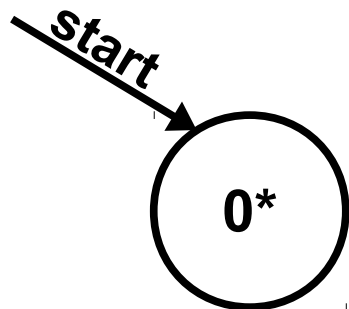


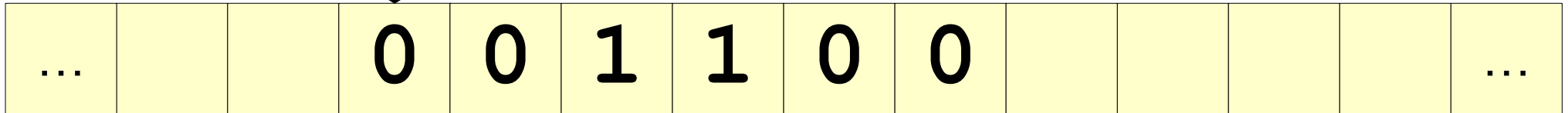
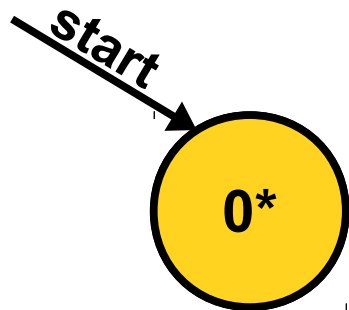
Idea: Repeatedly find a copy of 10 and replace it with 01.

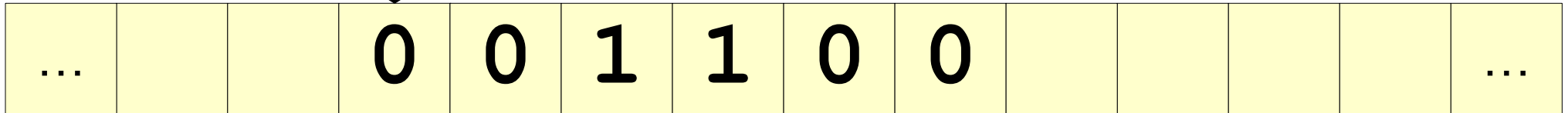
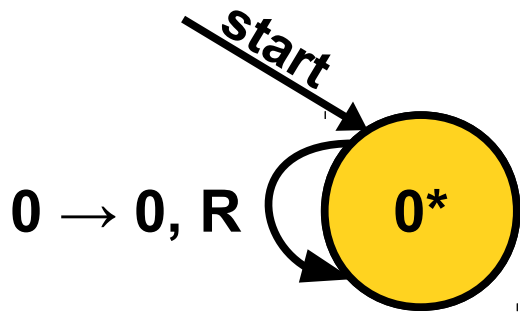
Let's Build It!

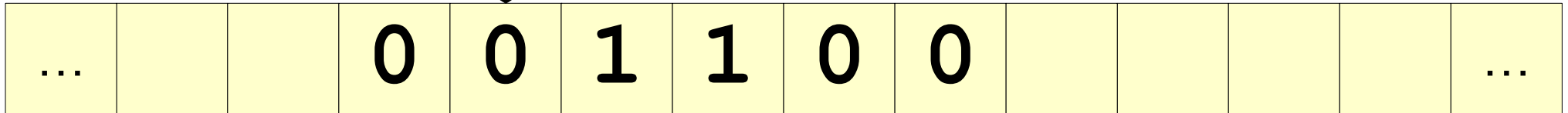
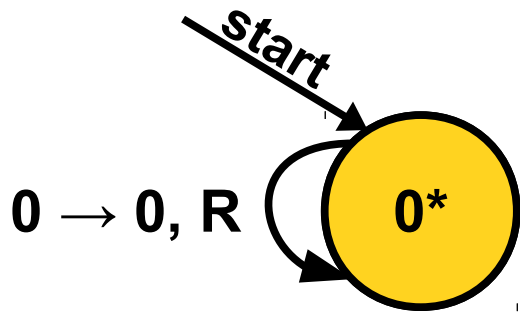


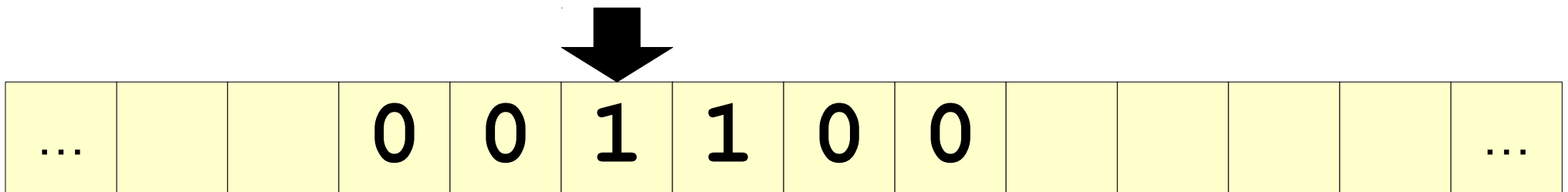
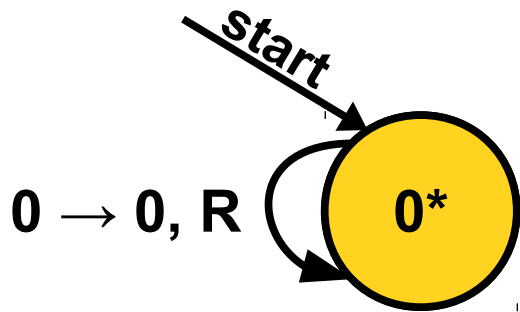


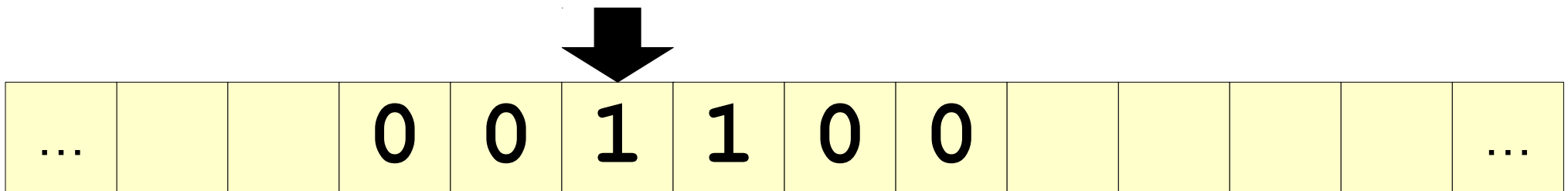
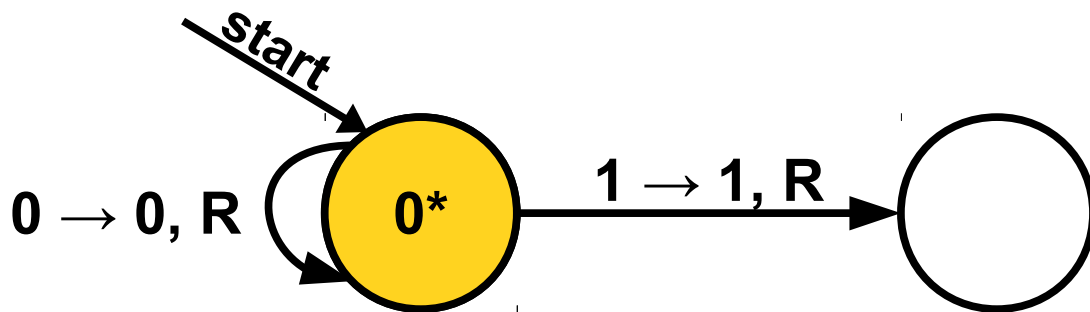


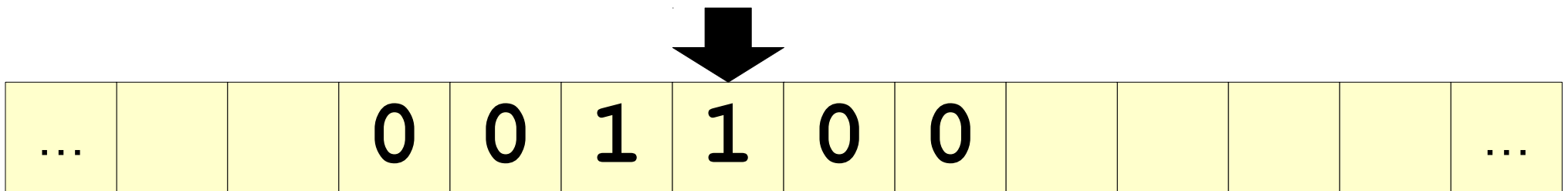
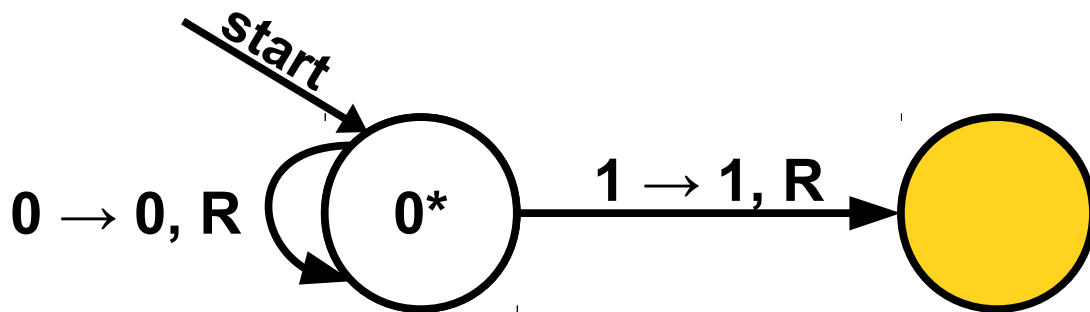


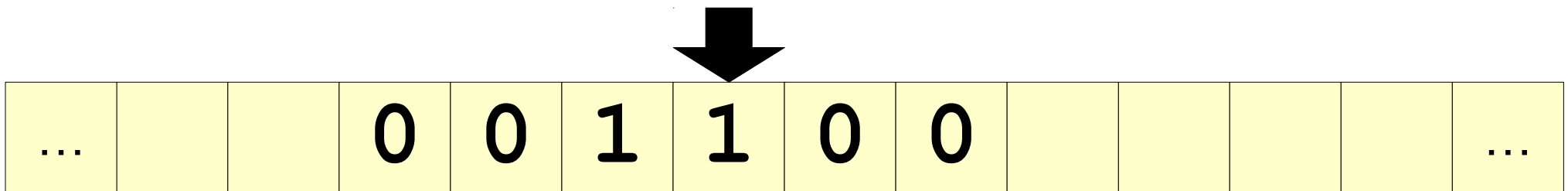
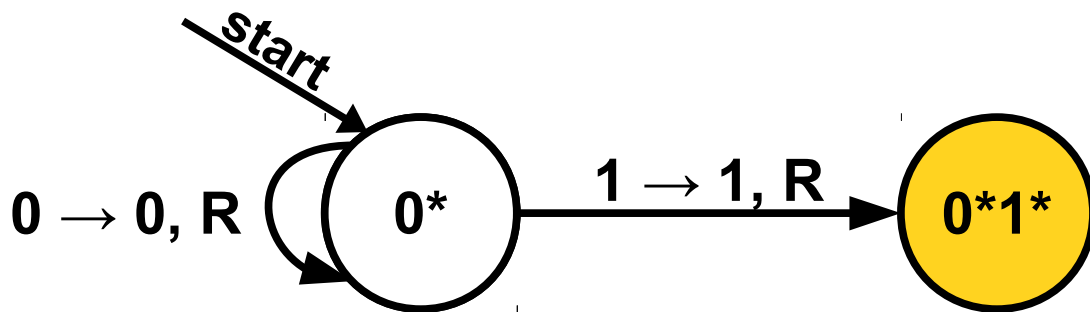


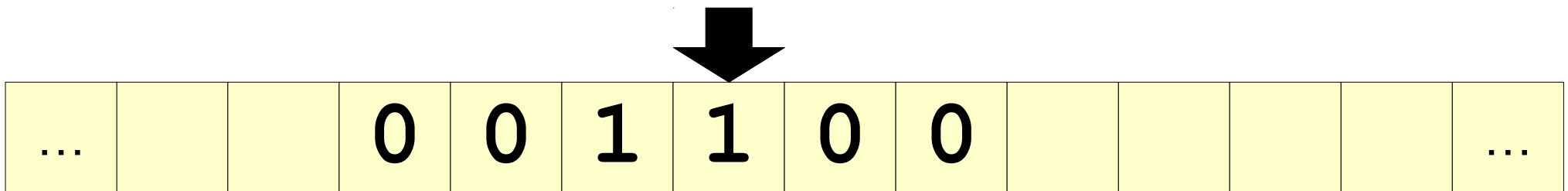
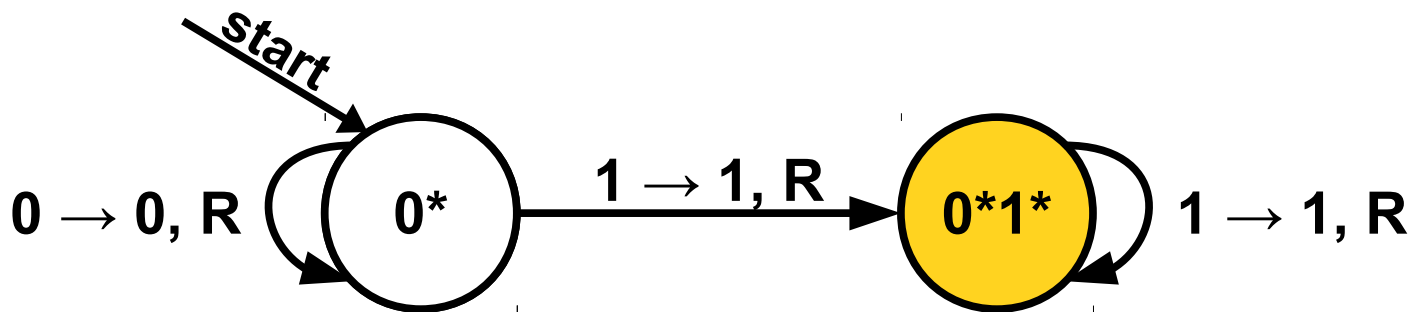


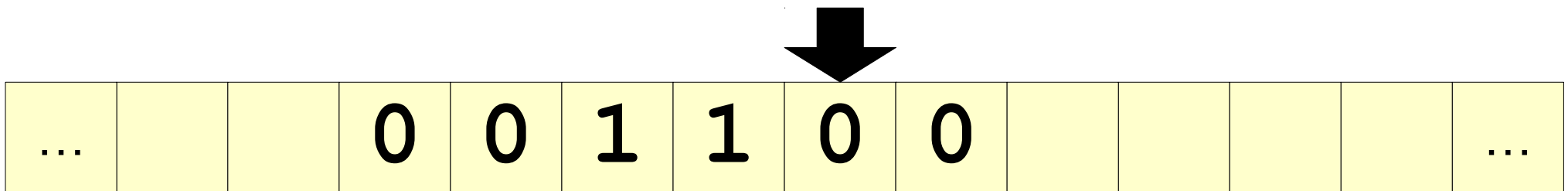
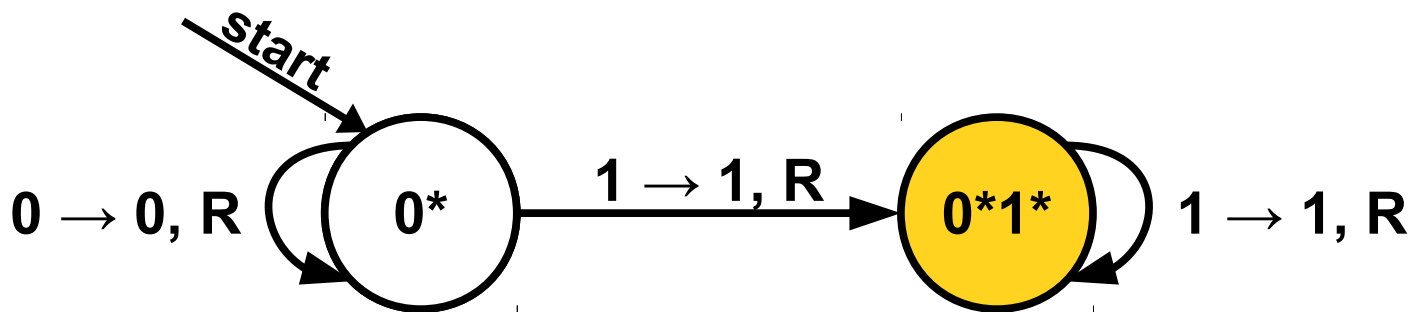


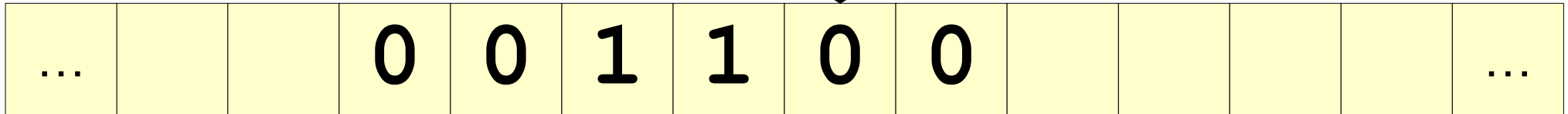
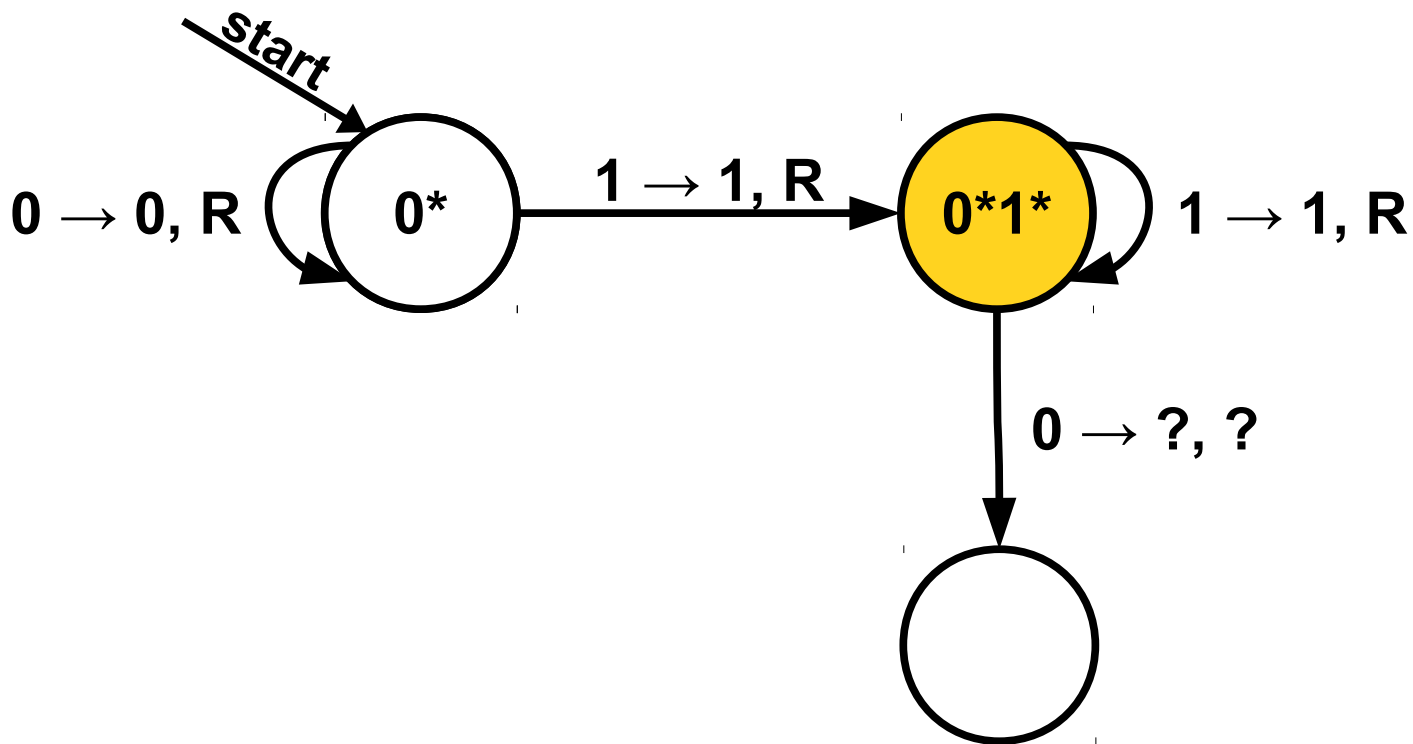


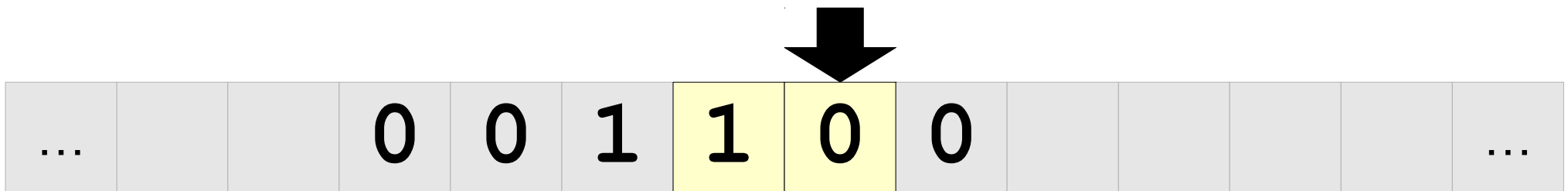
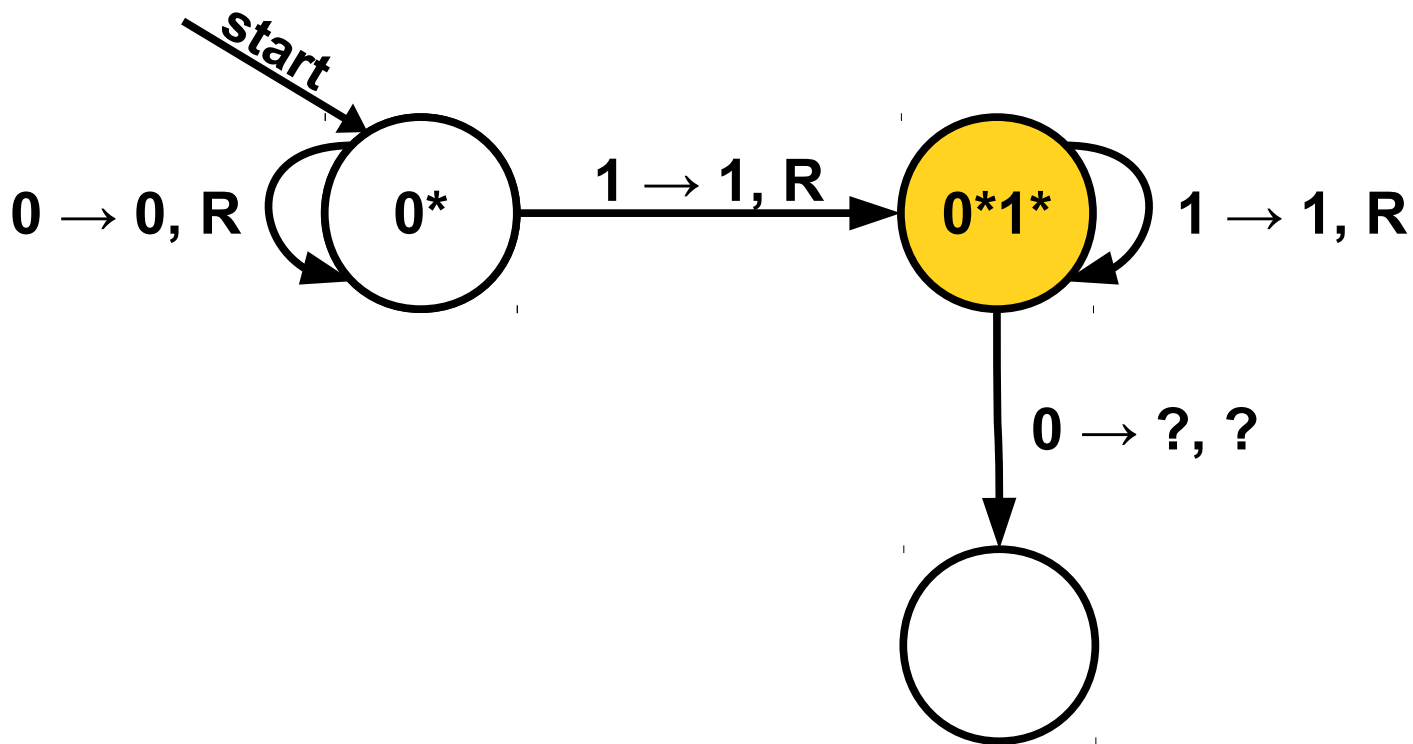


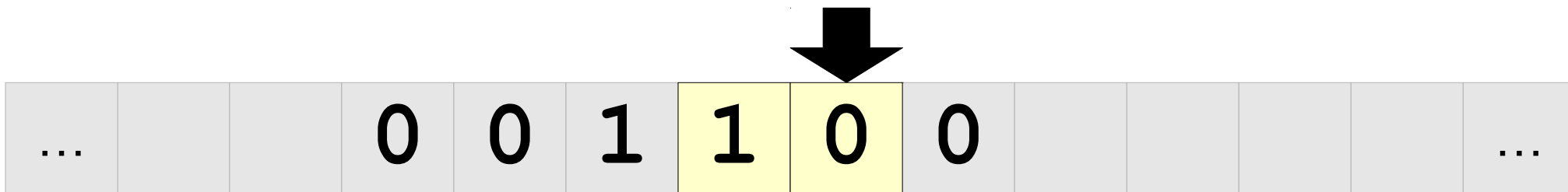
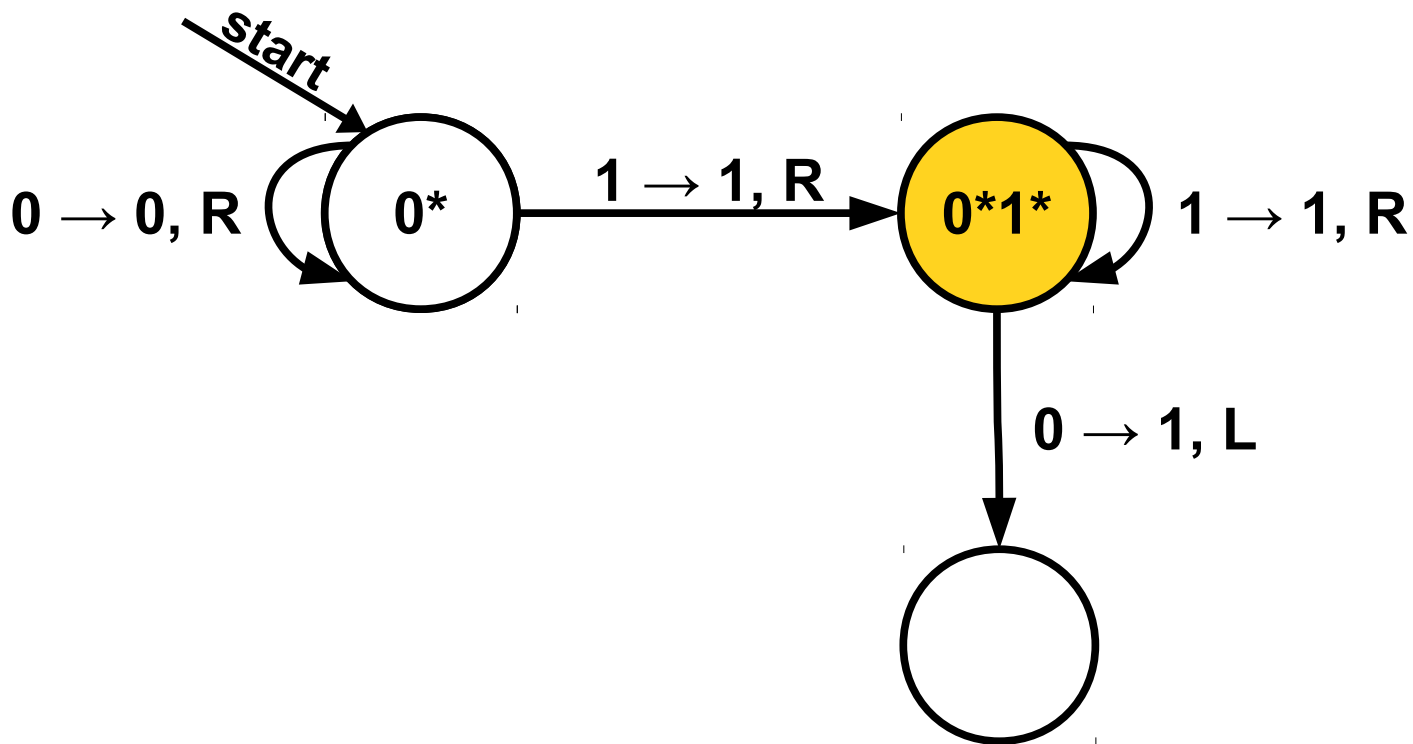


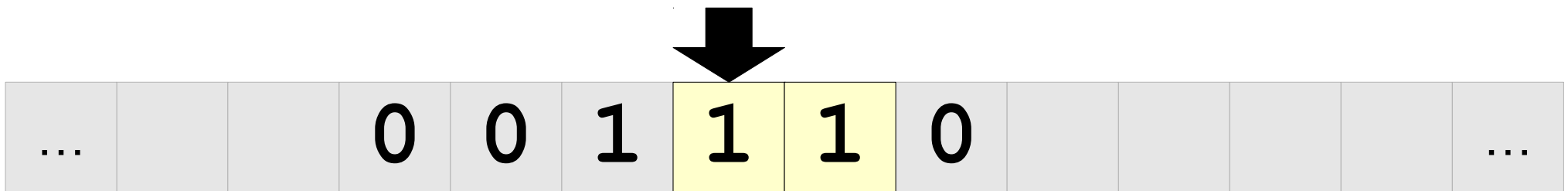
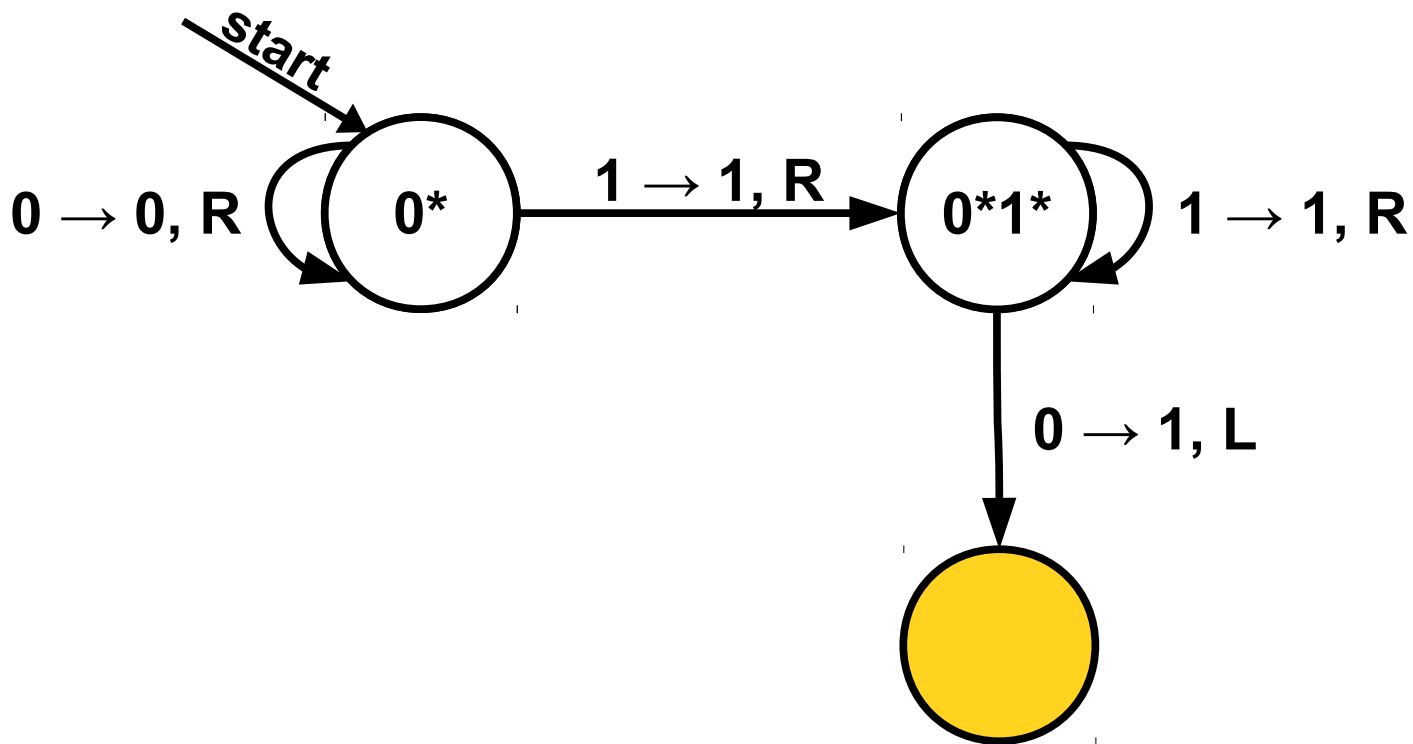


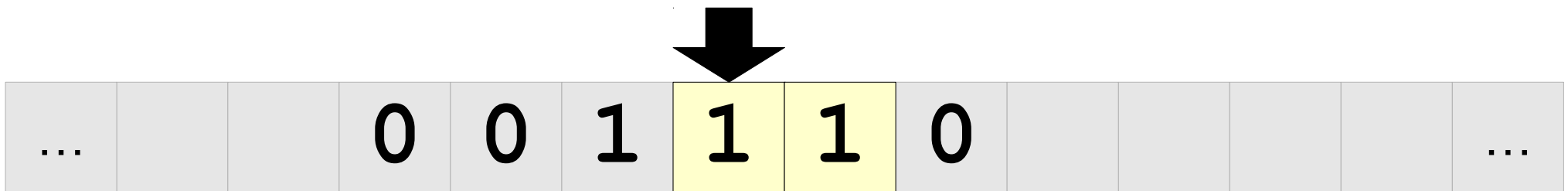
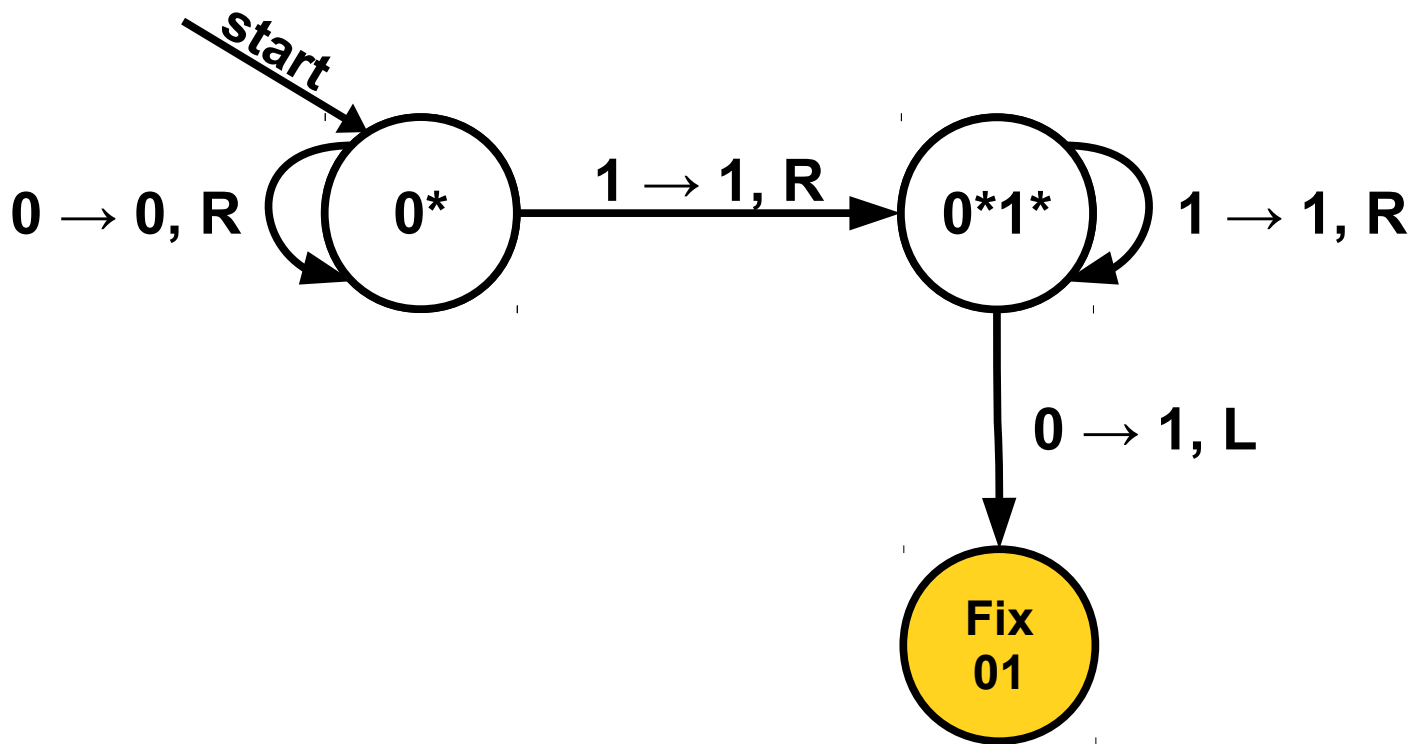


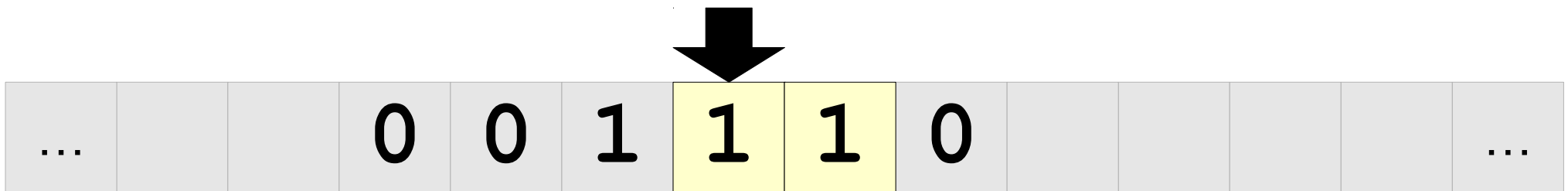
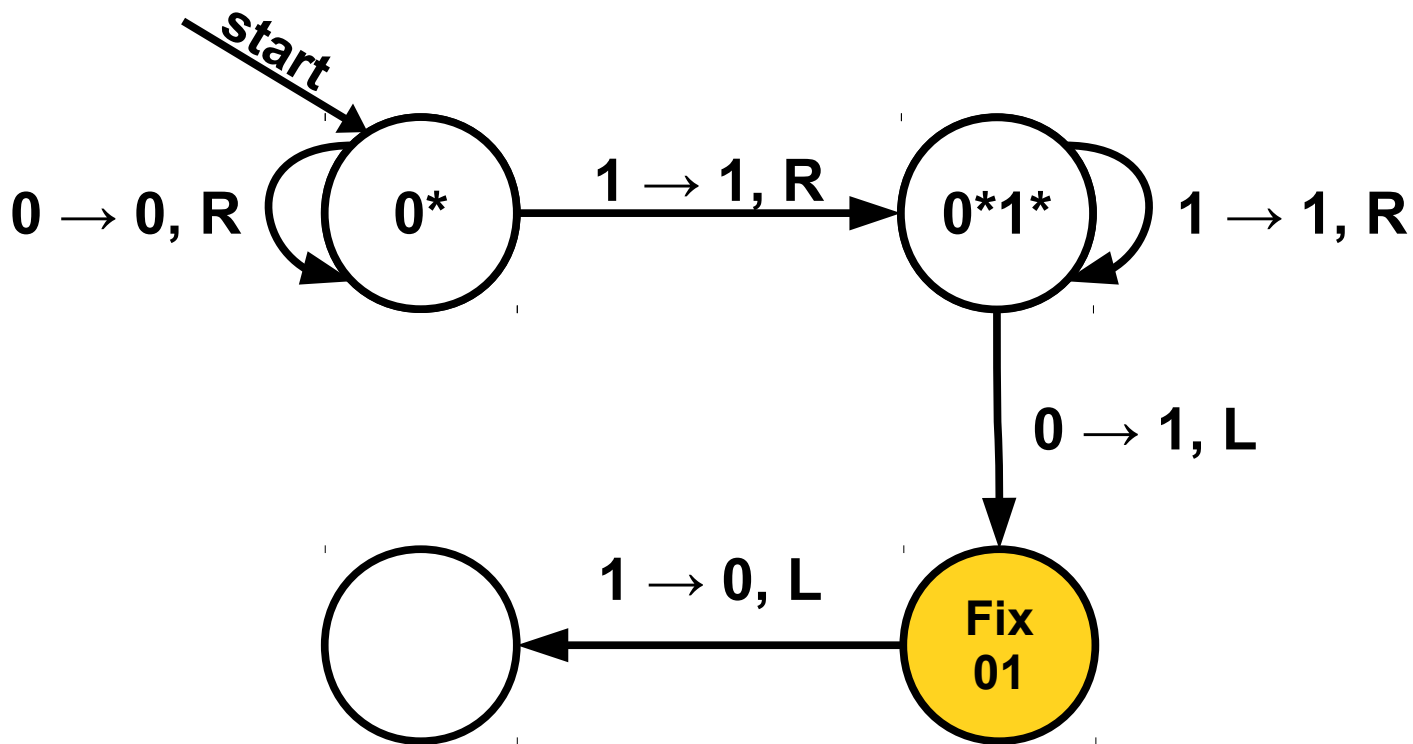


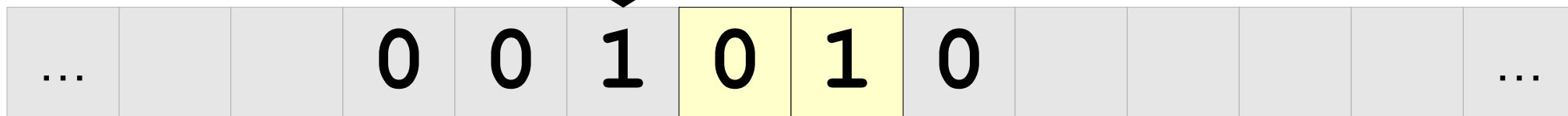
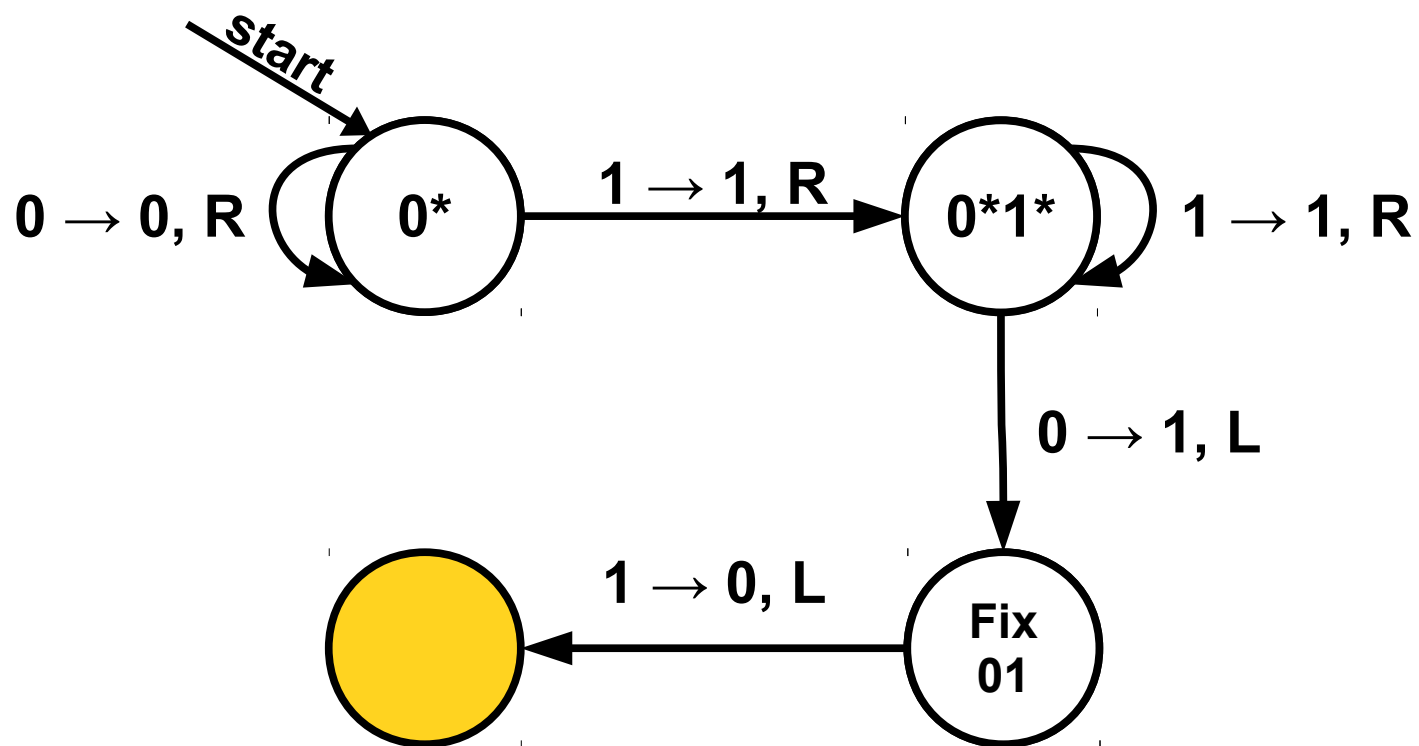


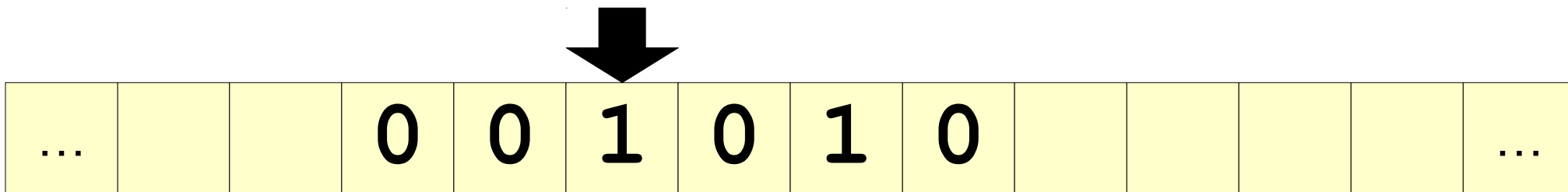
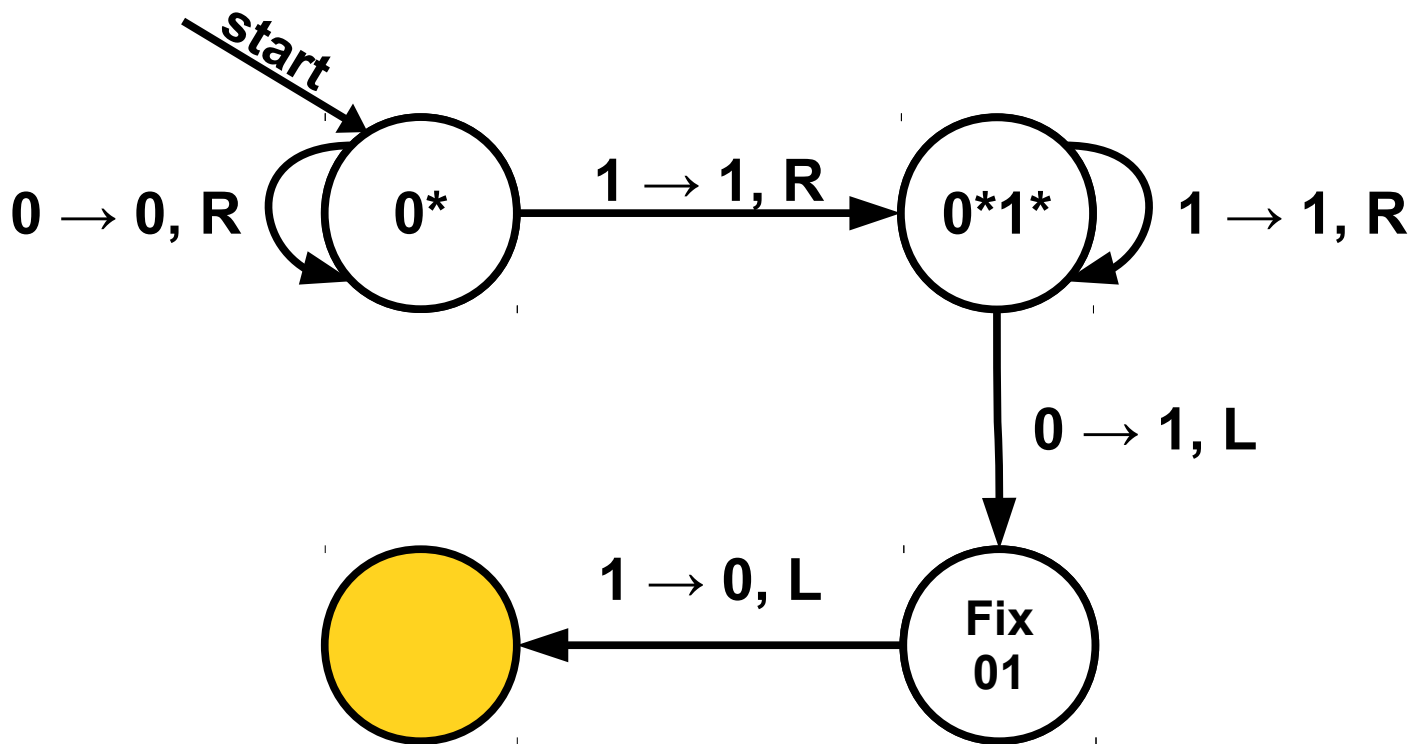


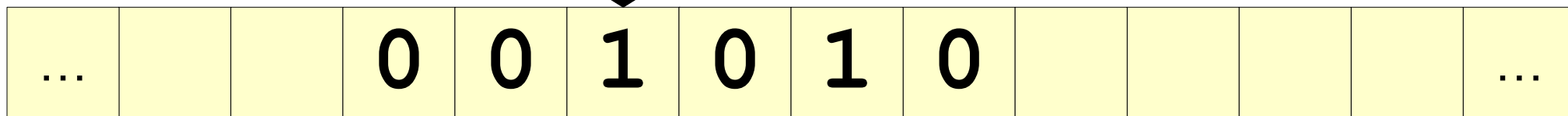
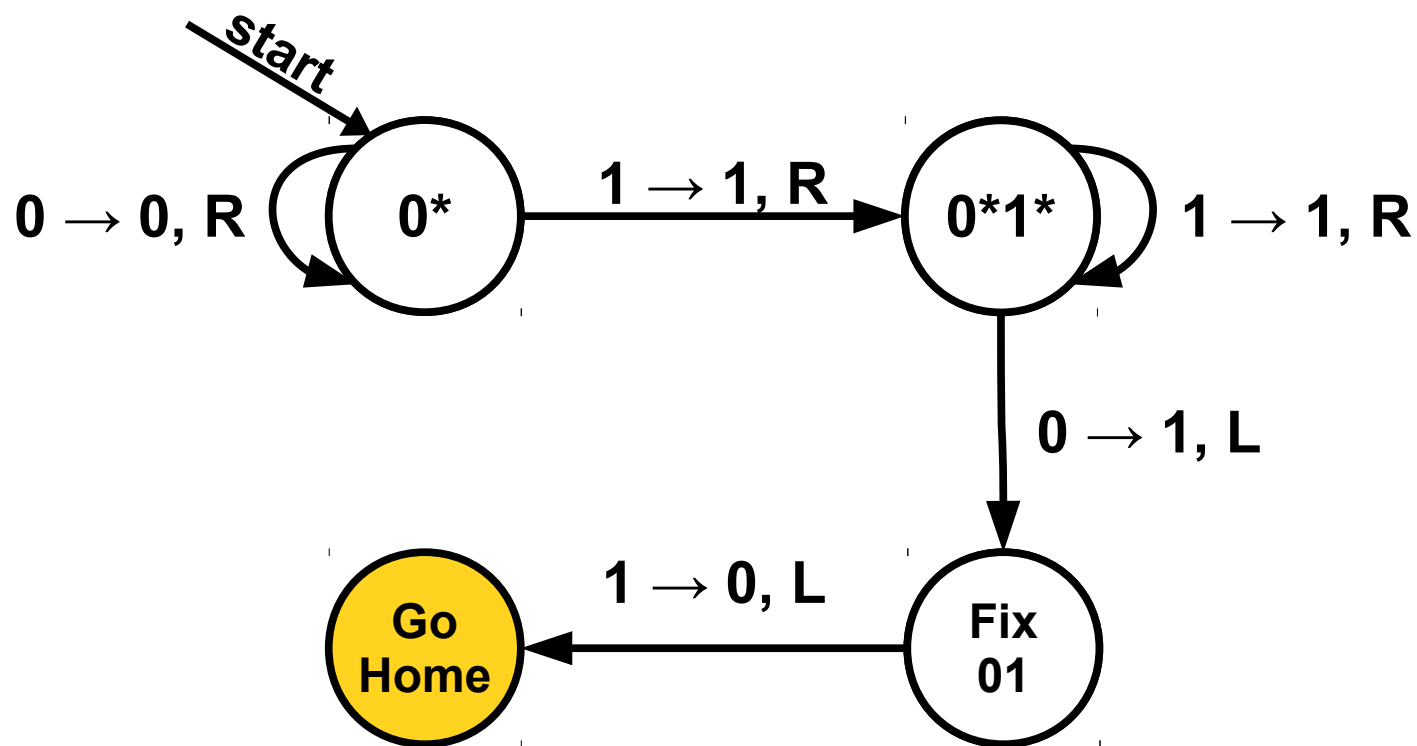


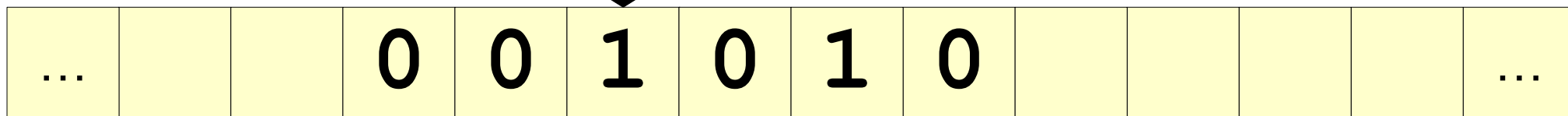
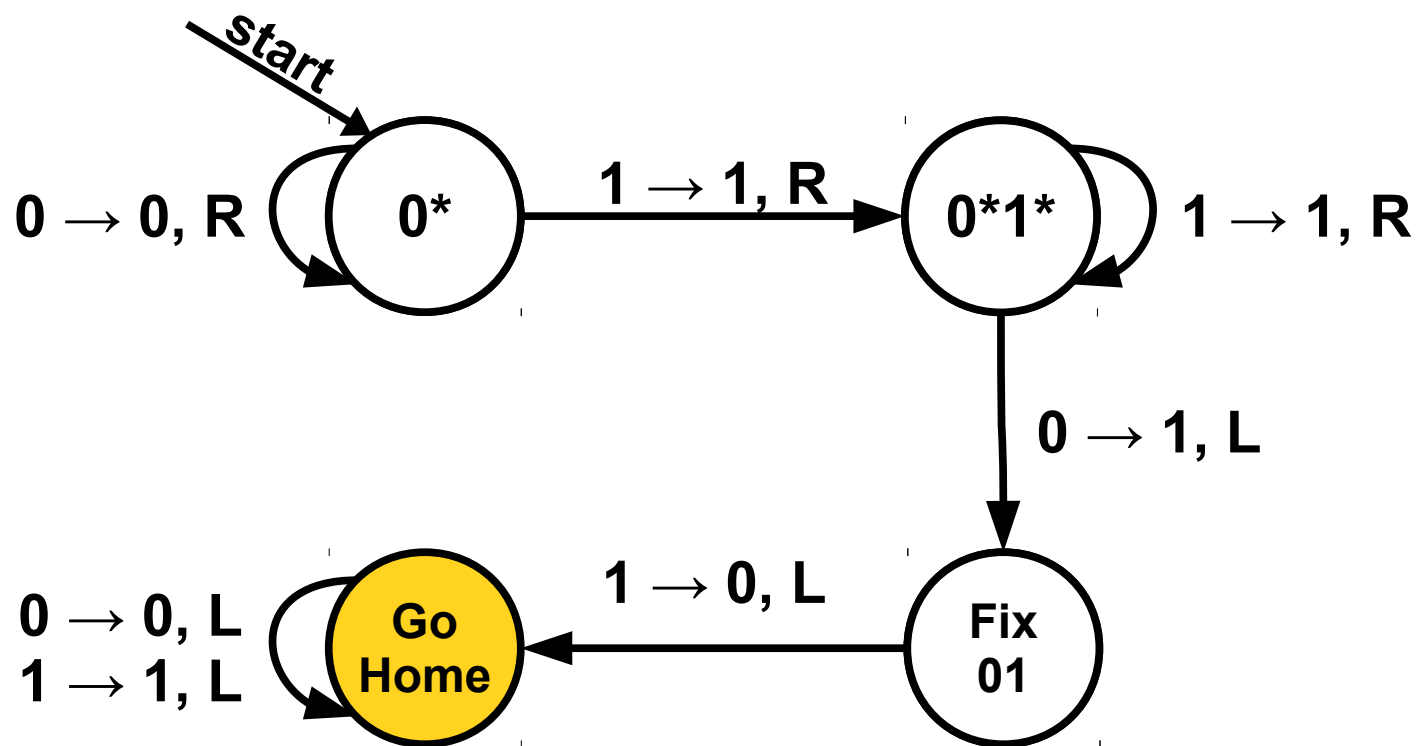


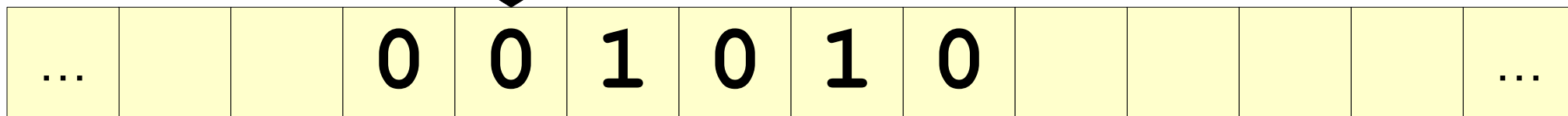
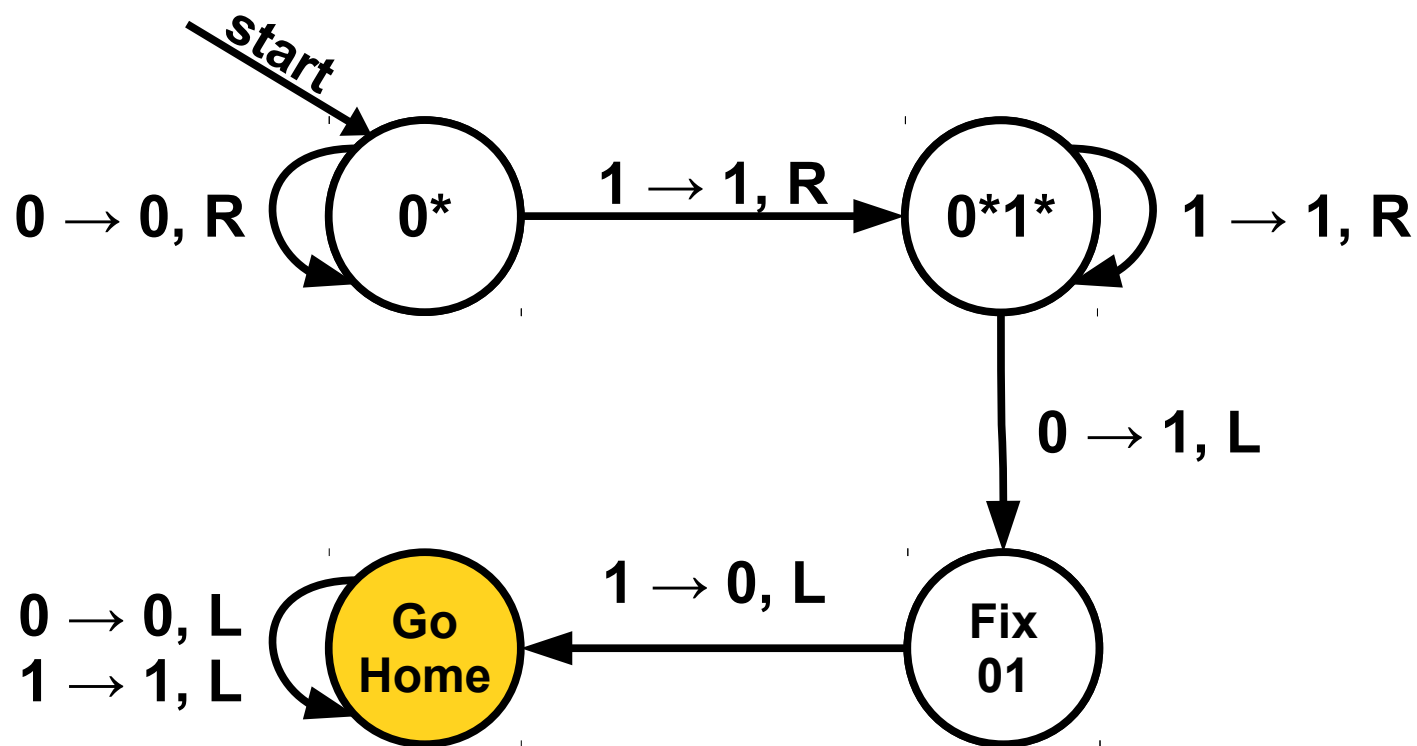


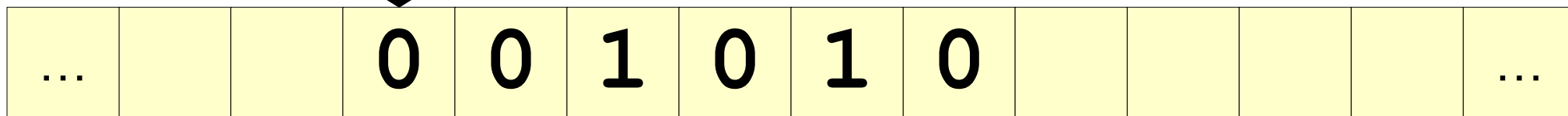
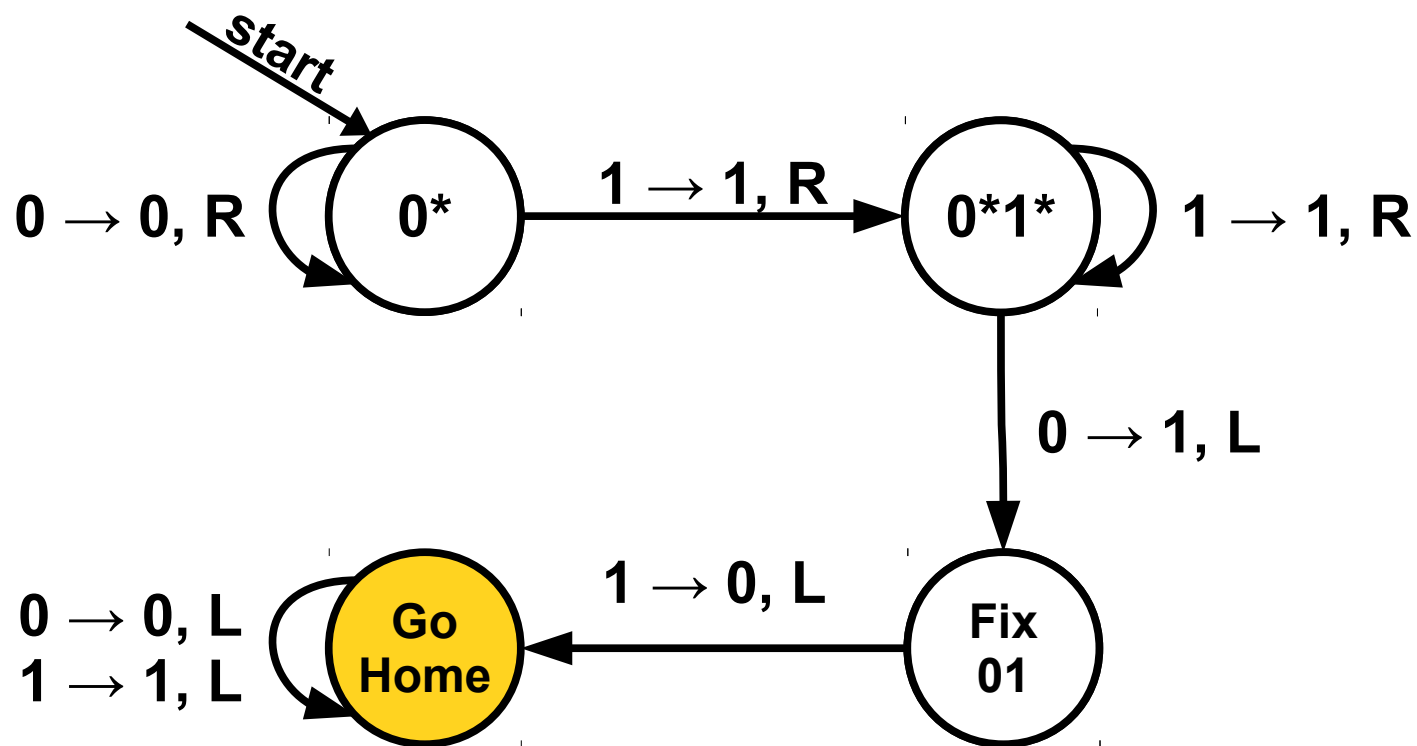


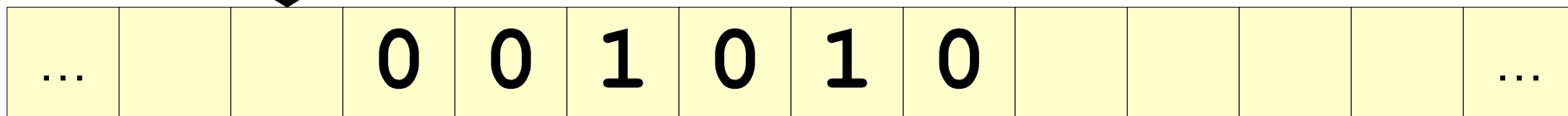
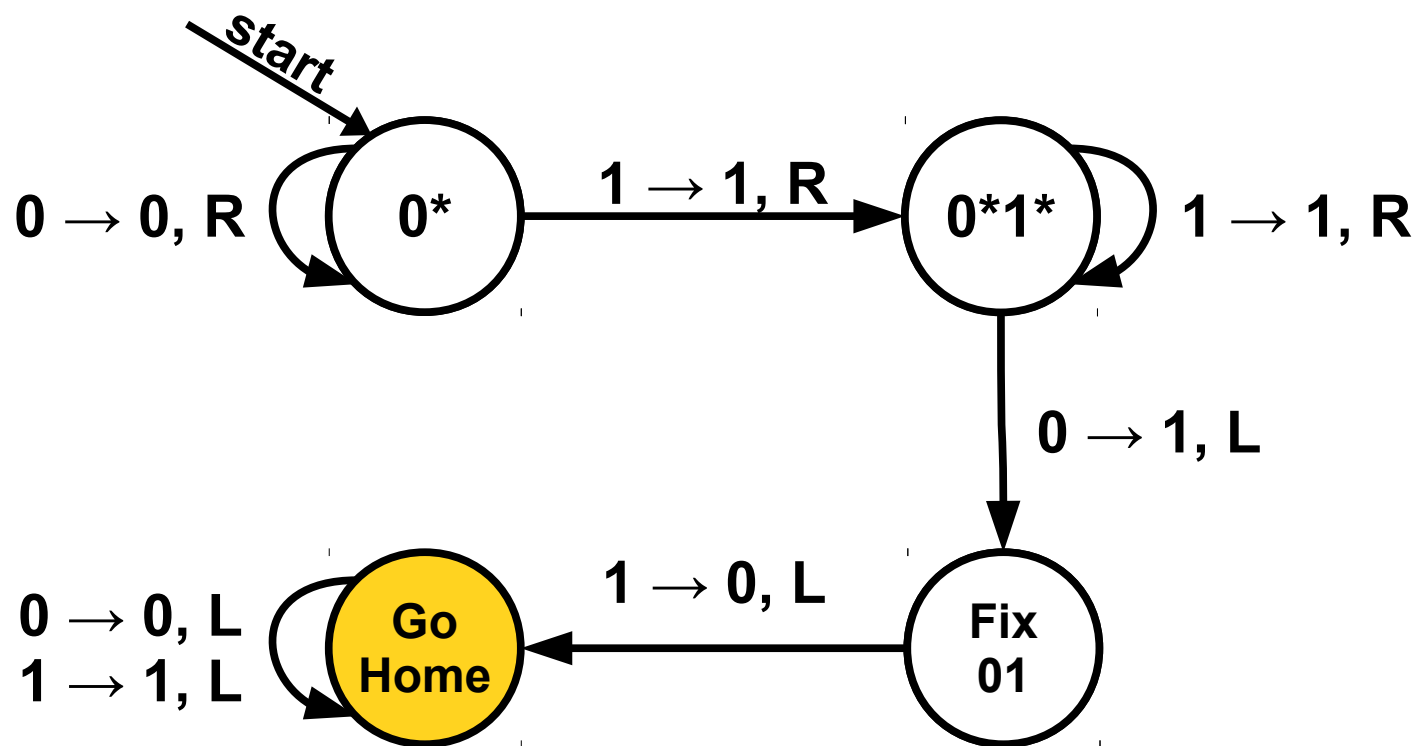


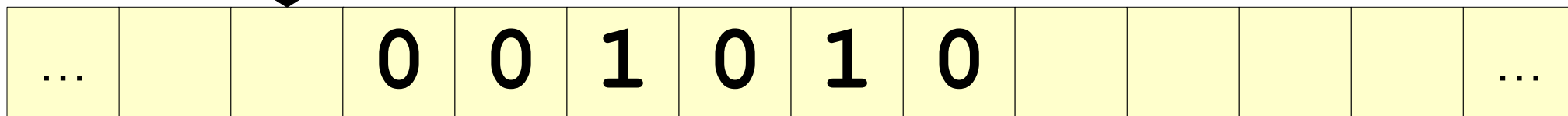
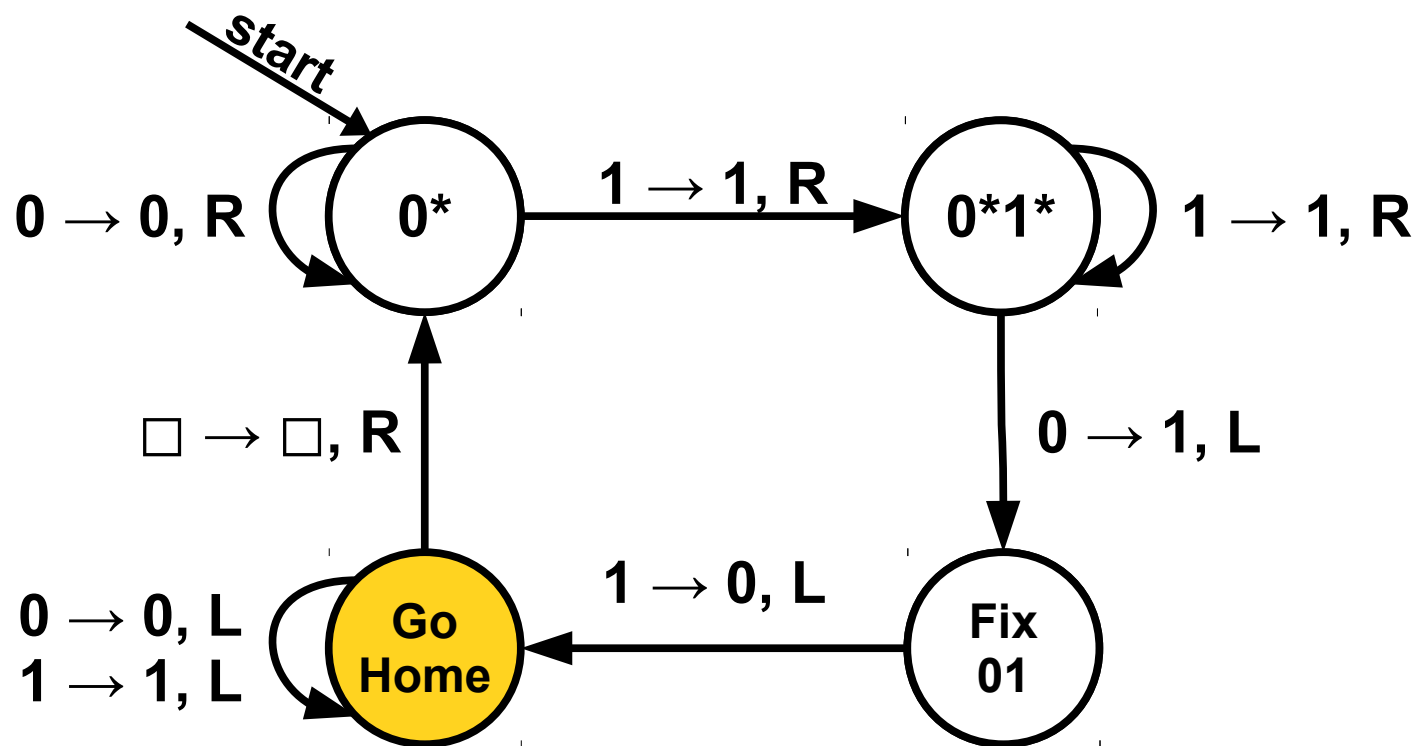


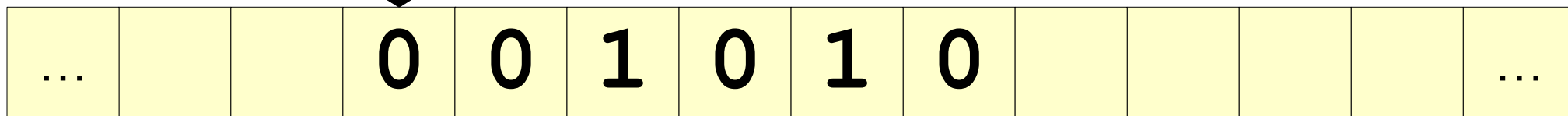
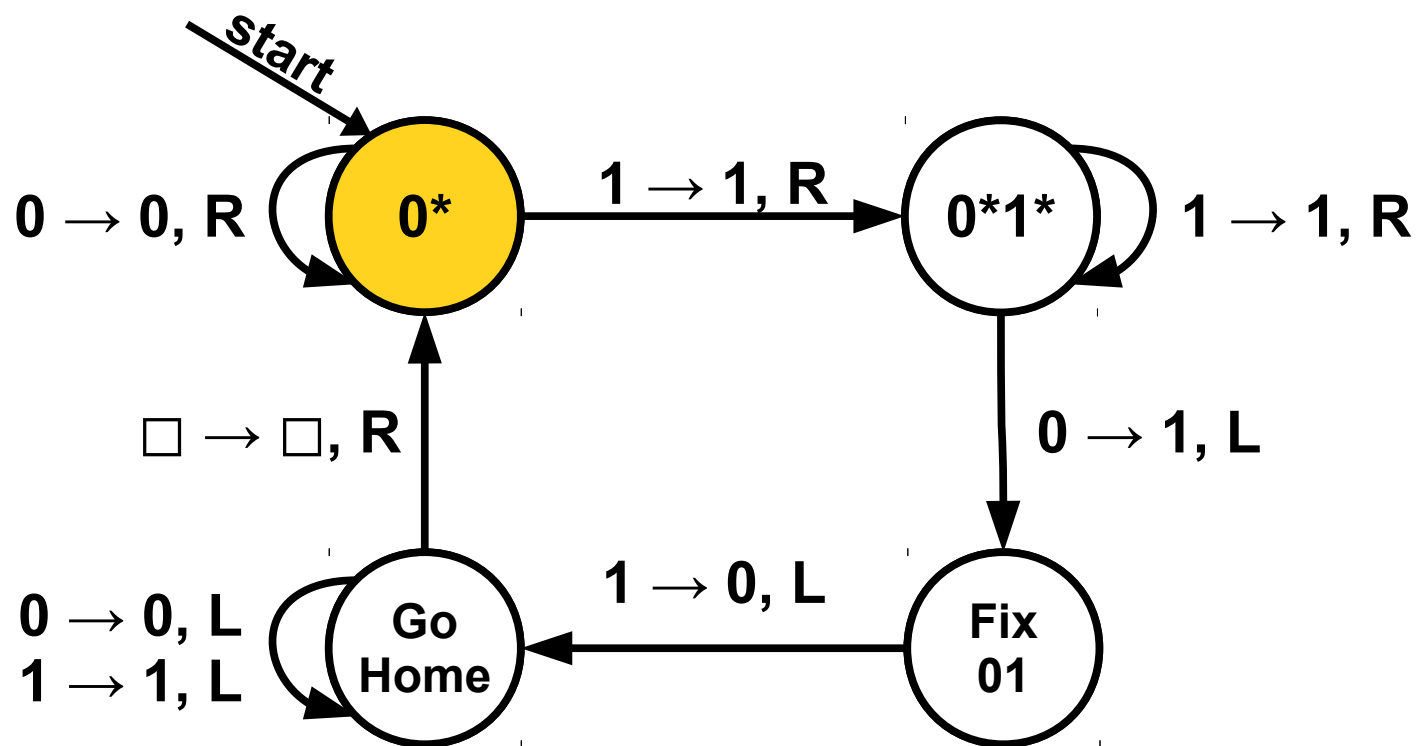


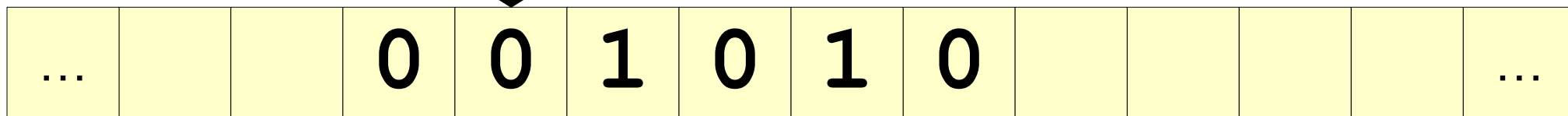
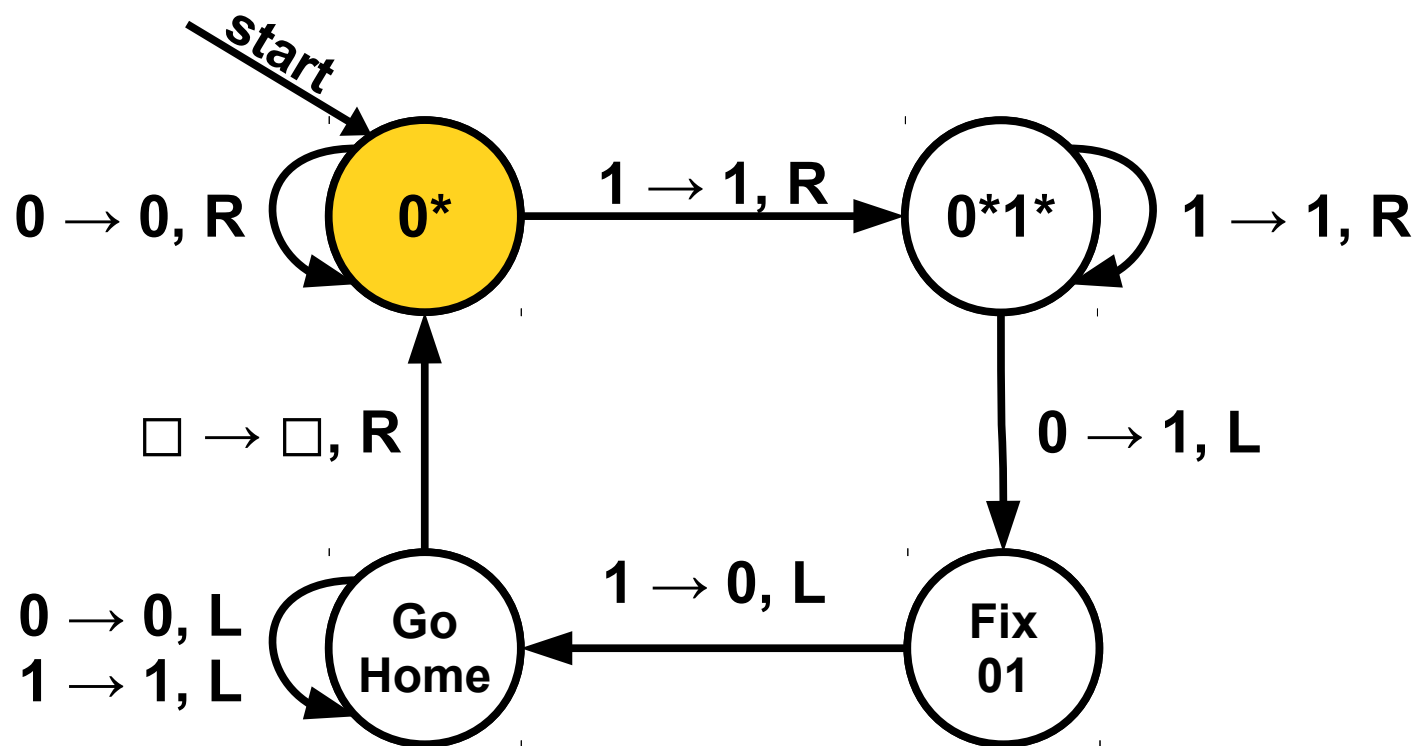


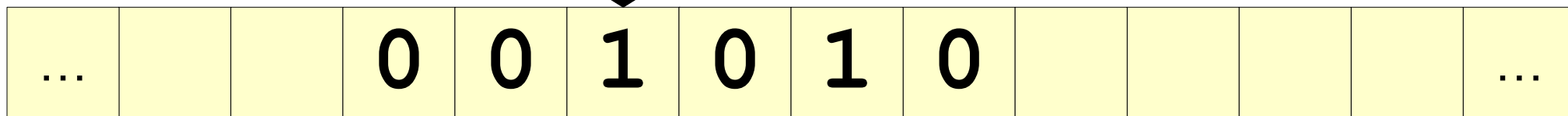
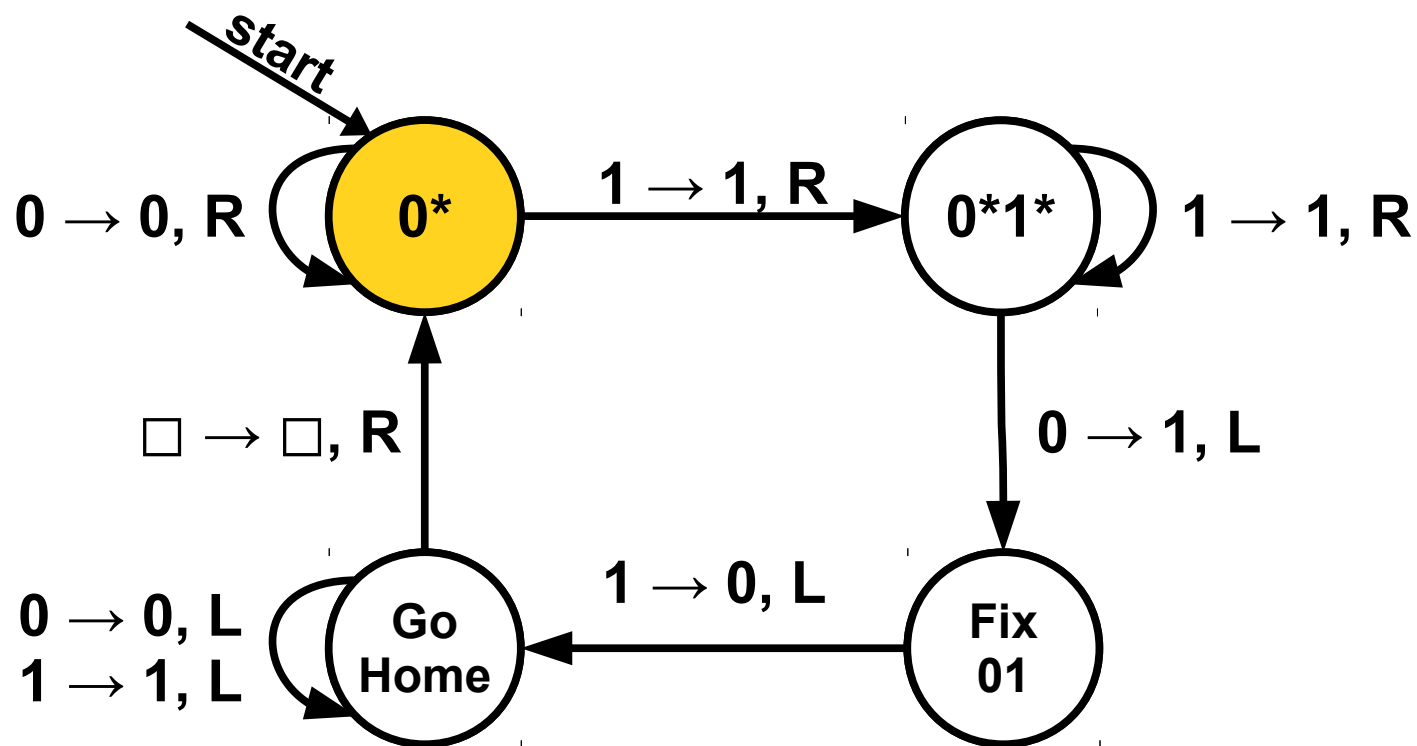


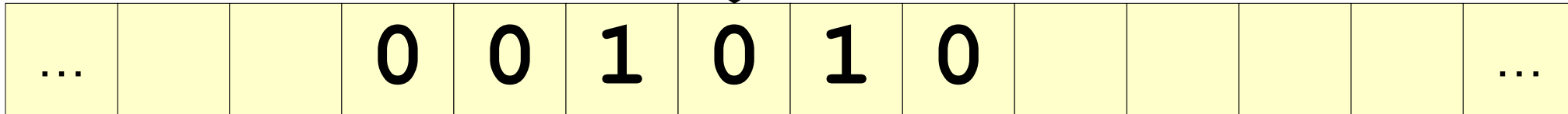
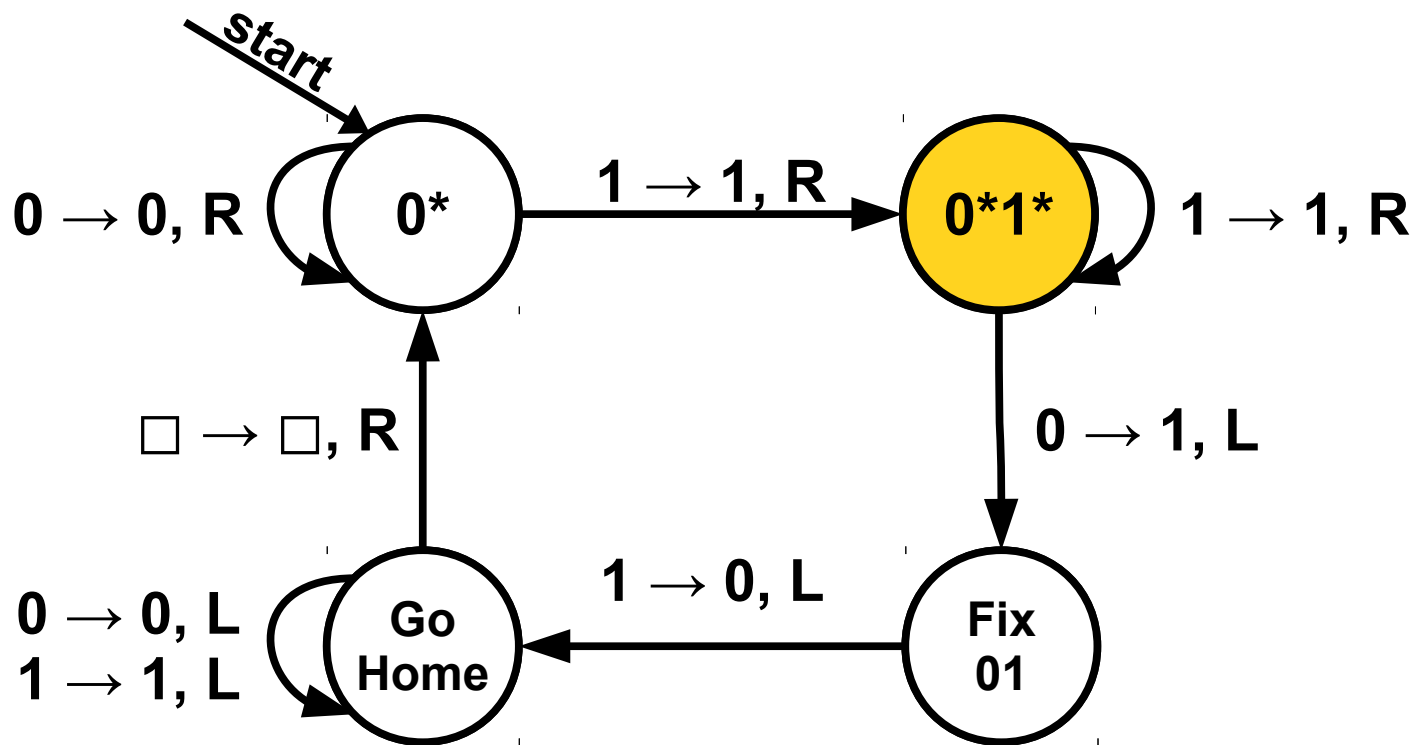


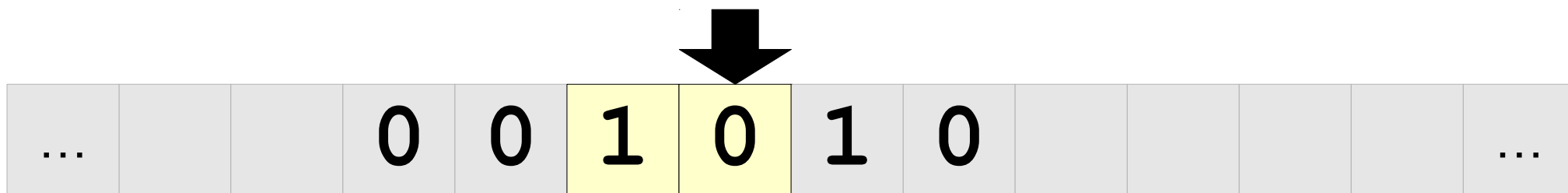
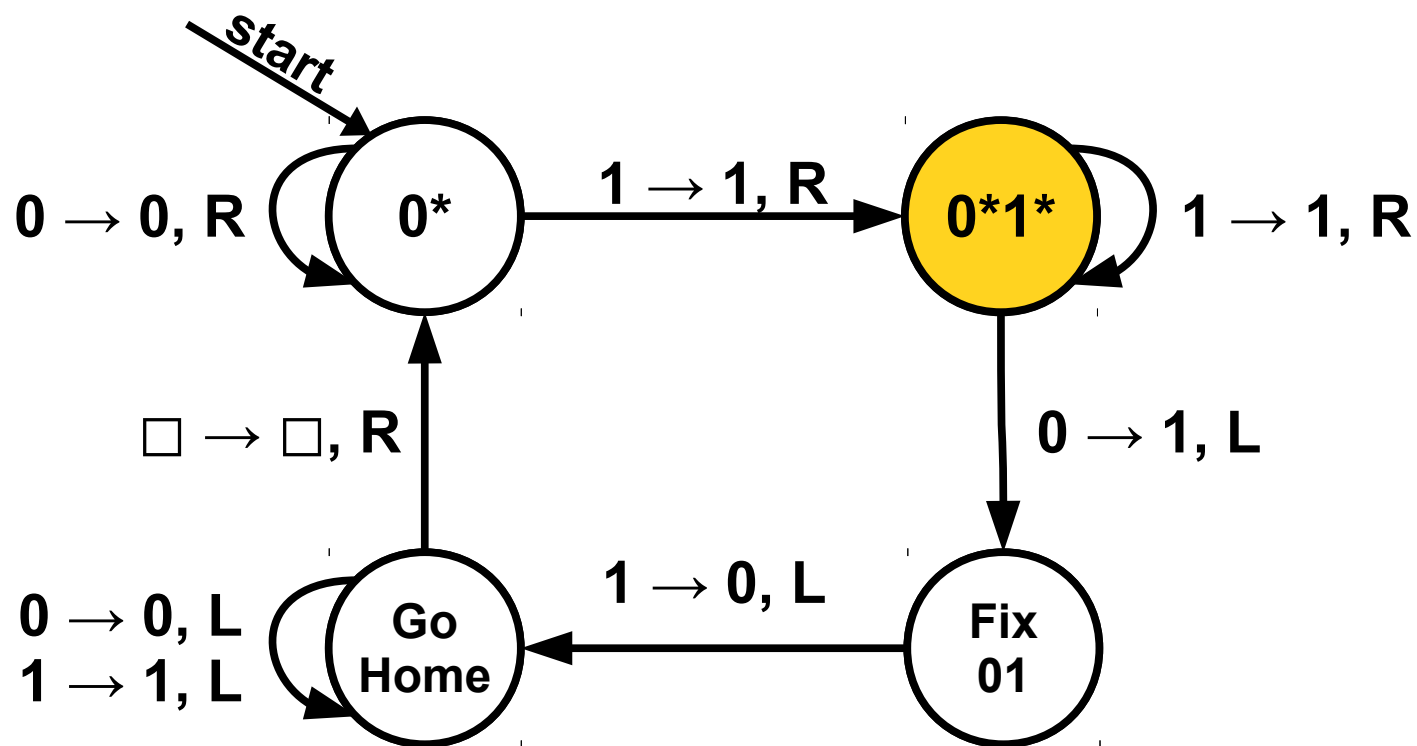


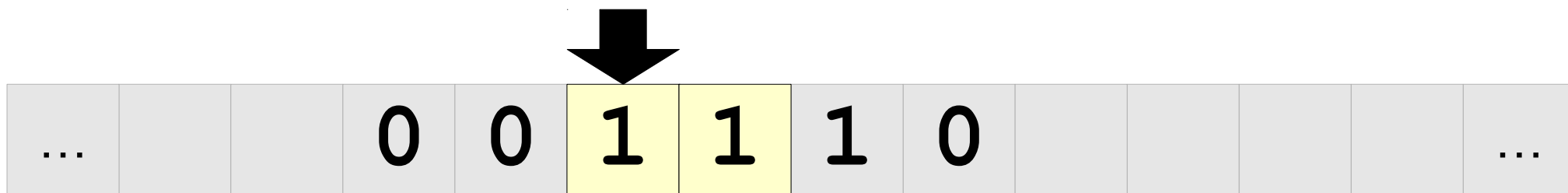
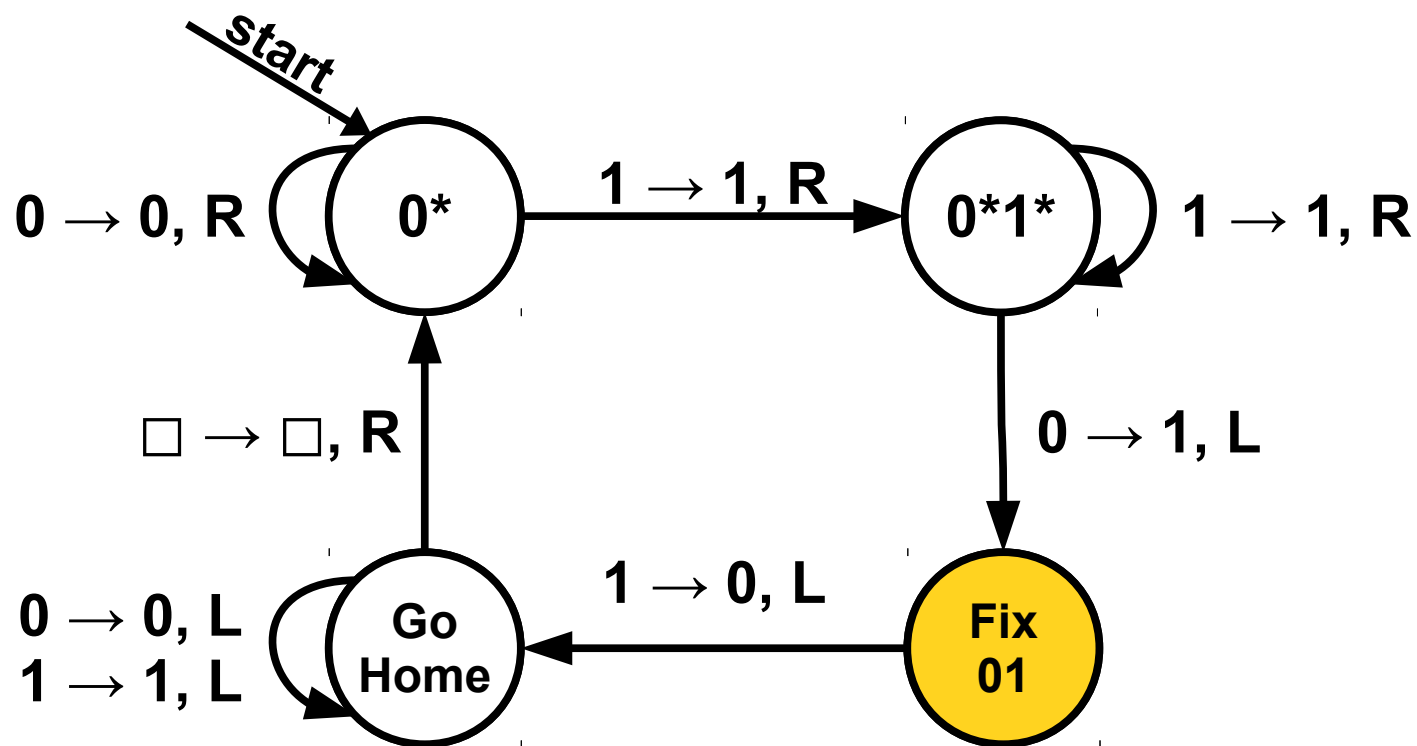


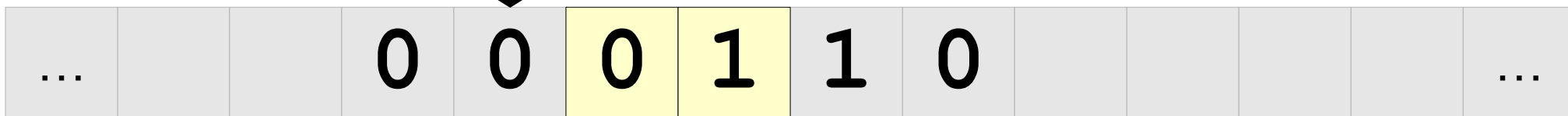
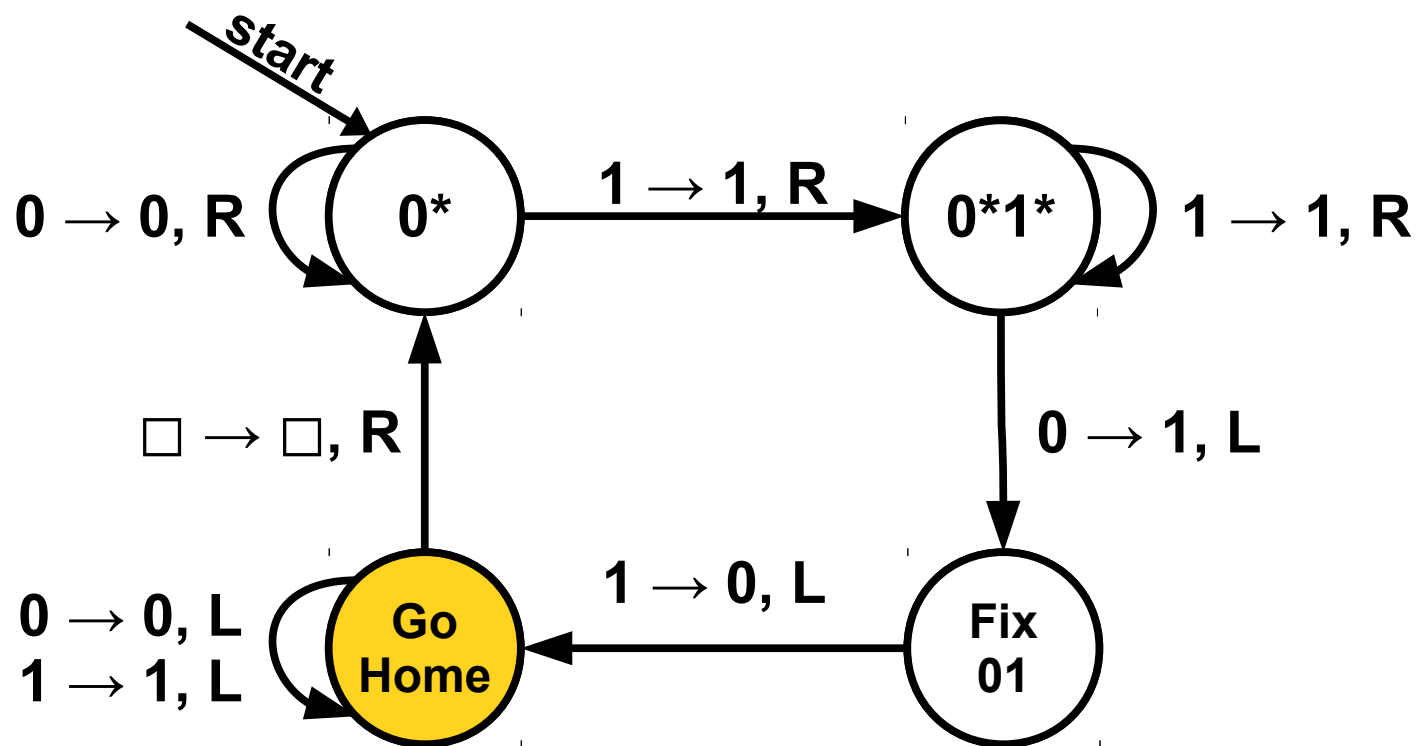


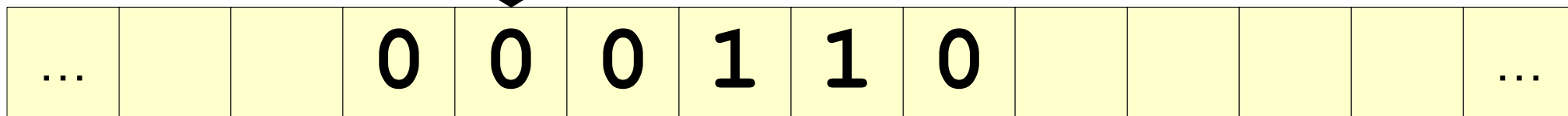
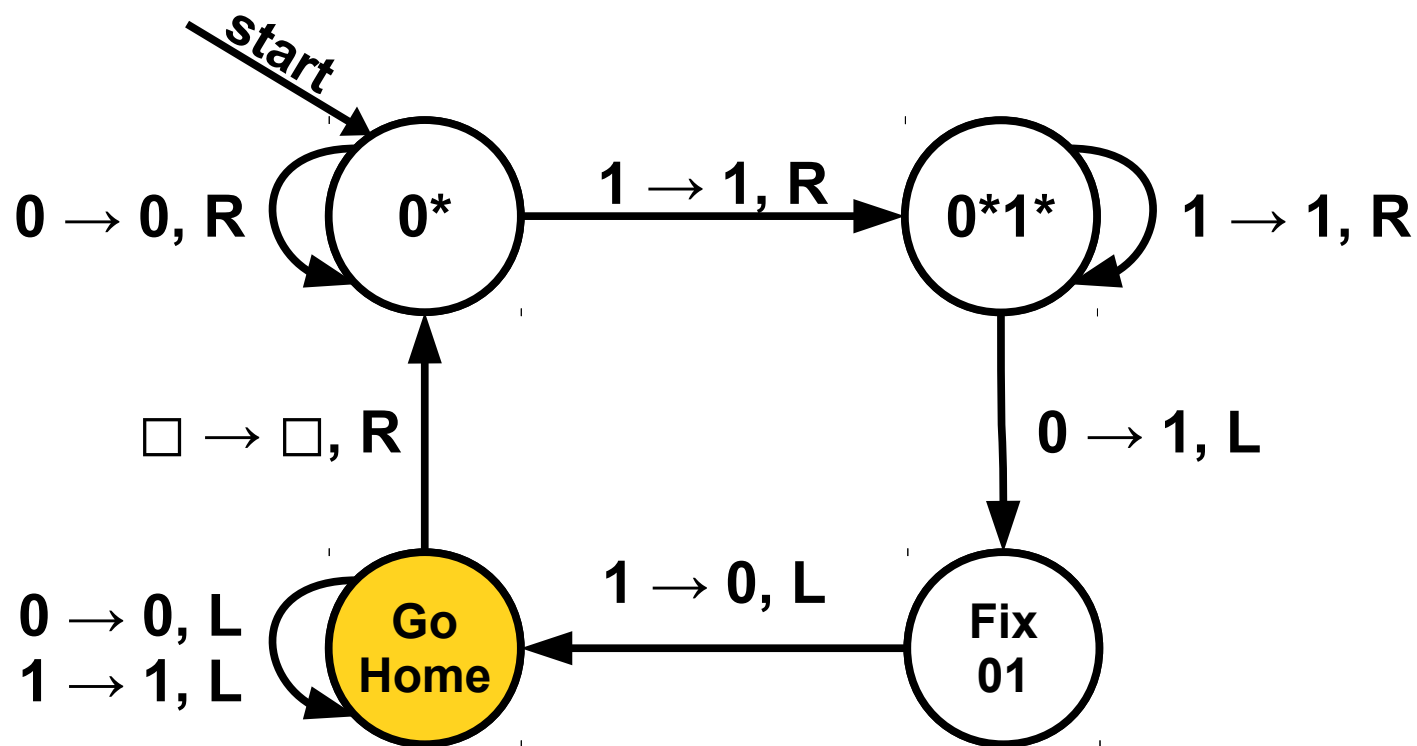


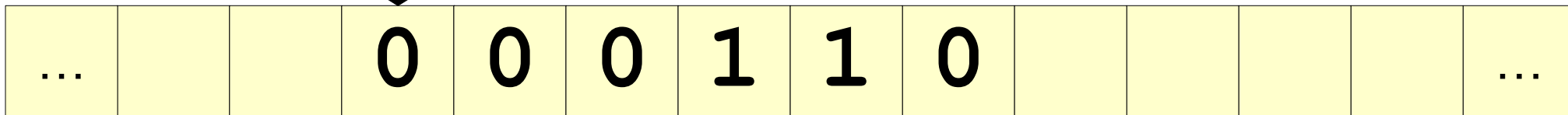
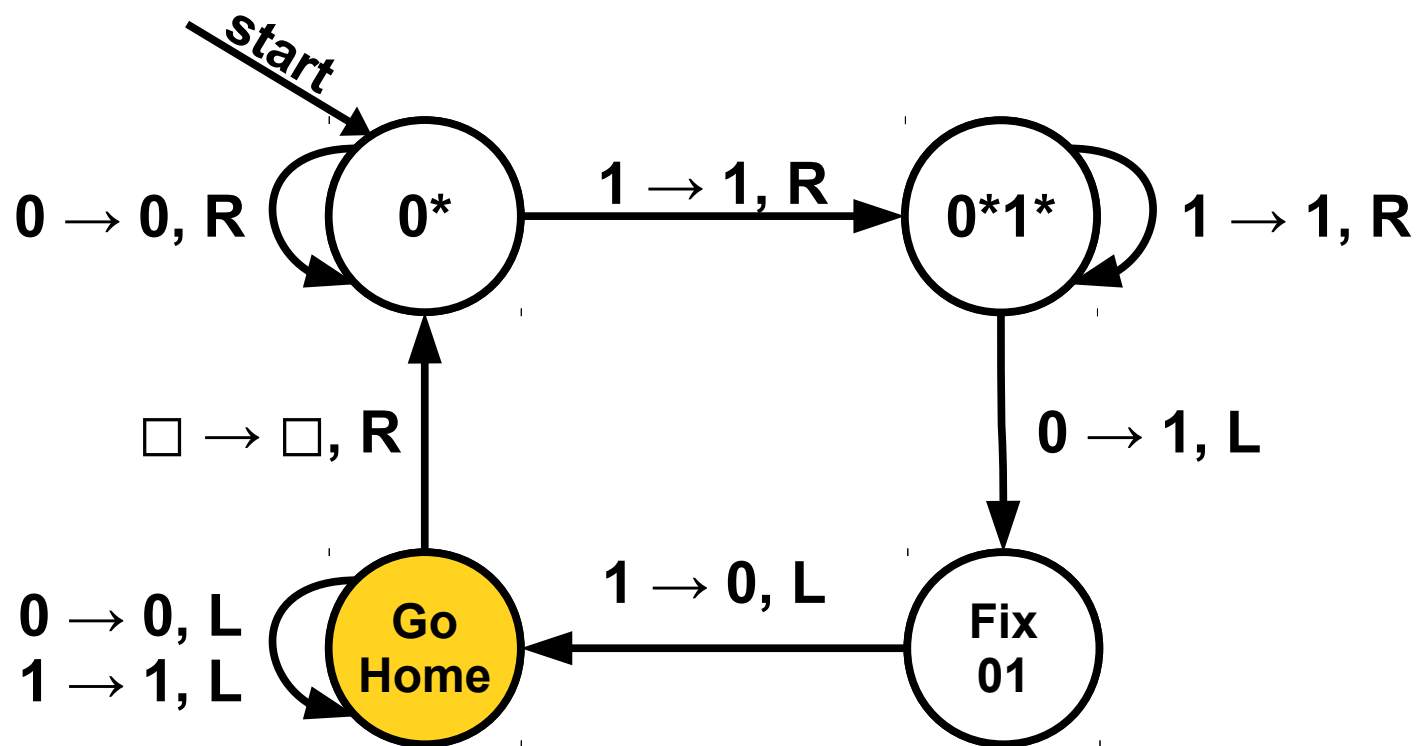


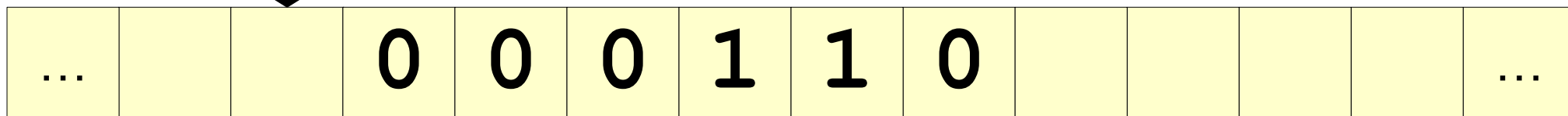
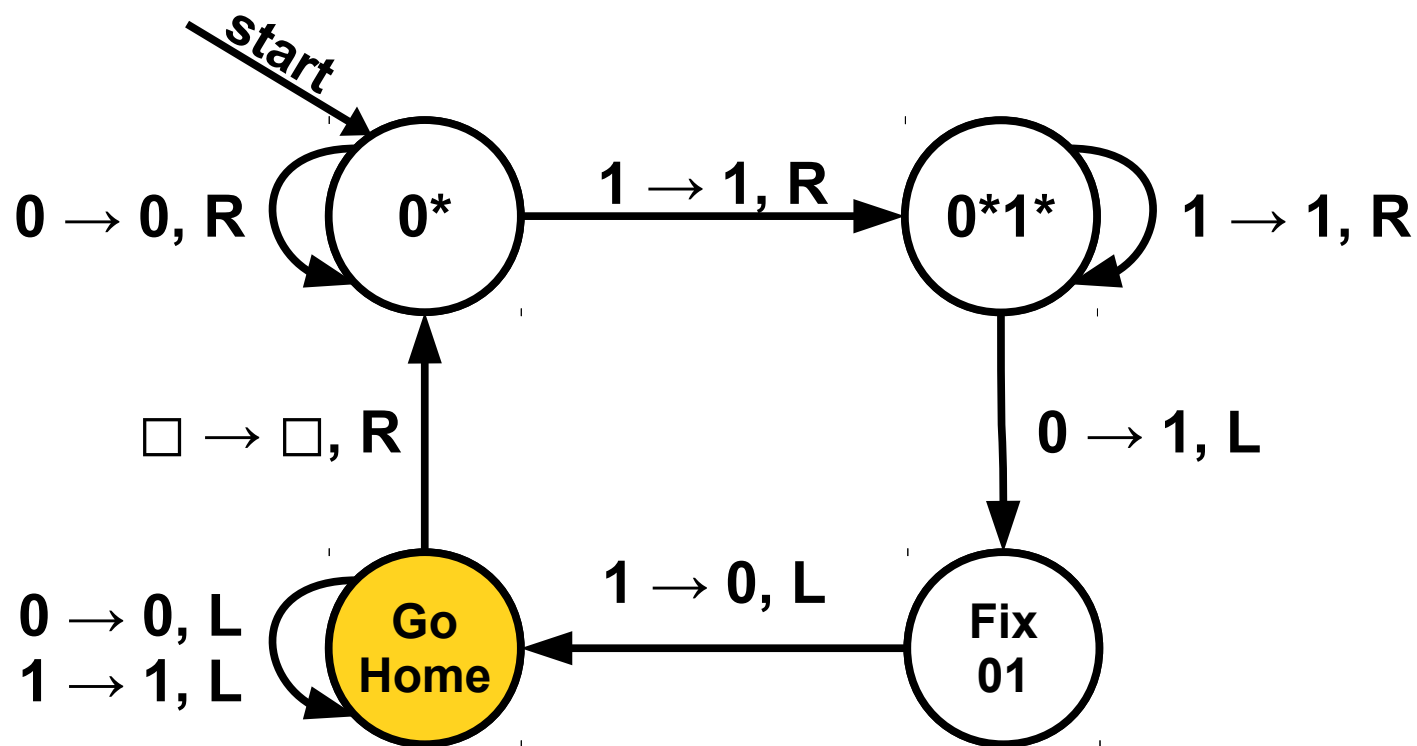


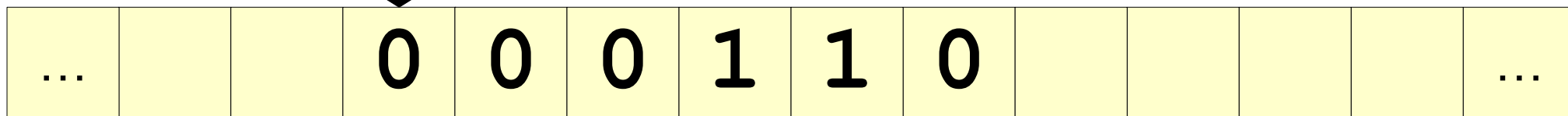
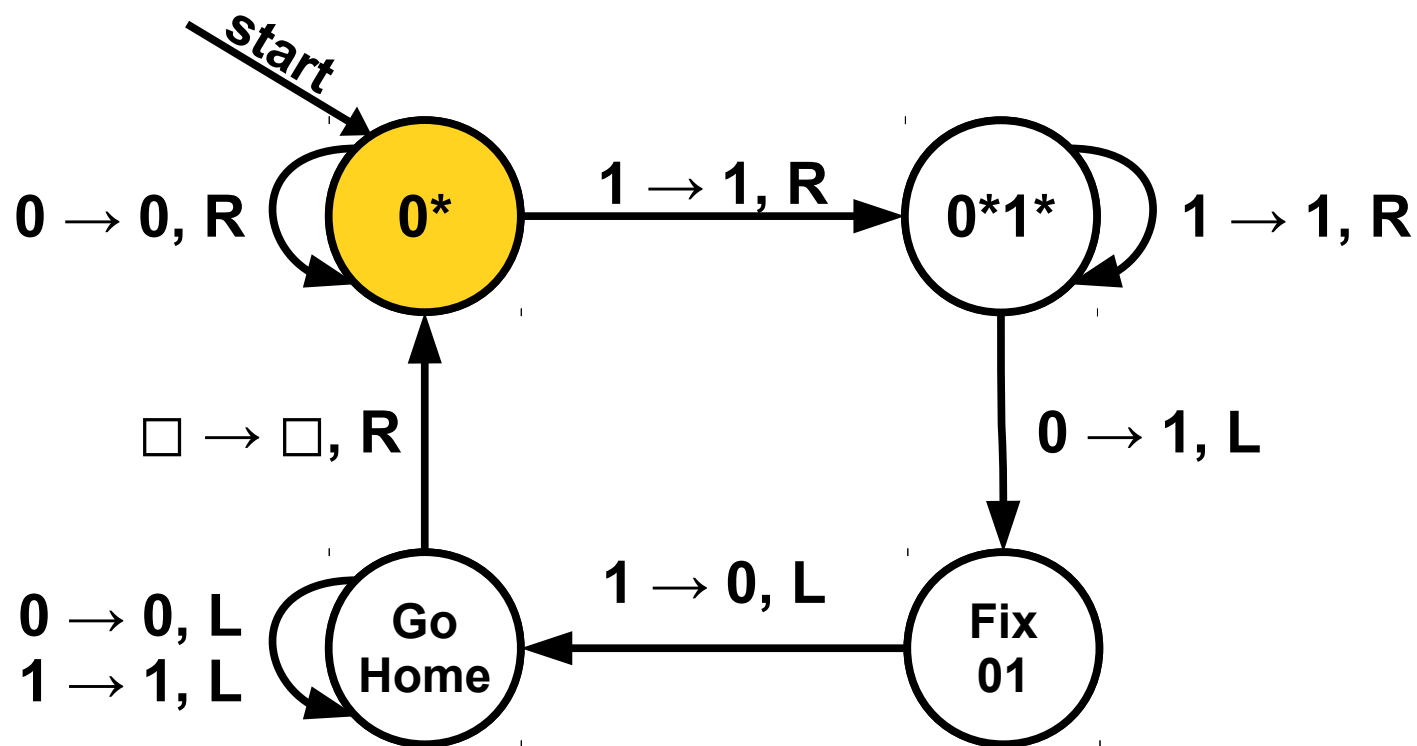


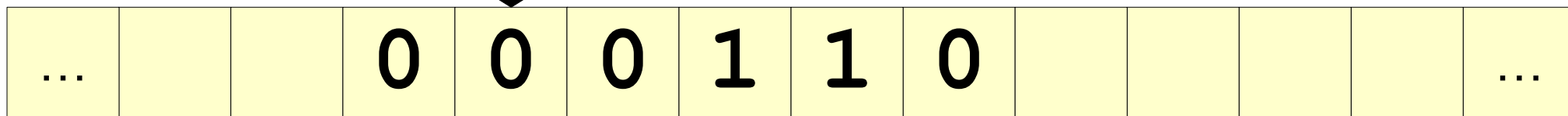
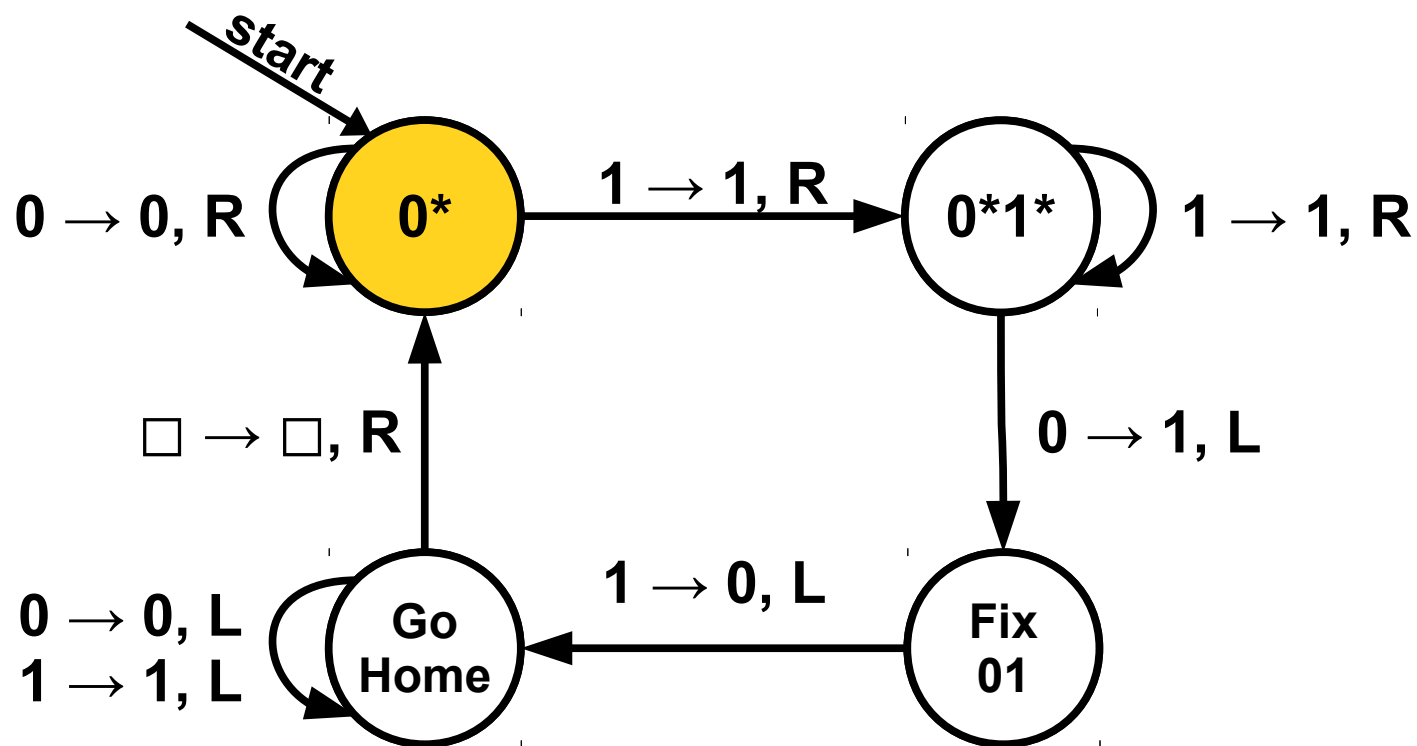


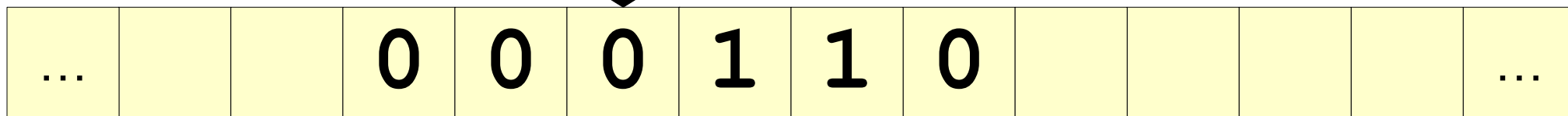
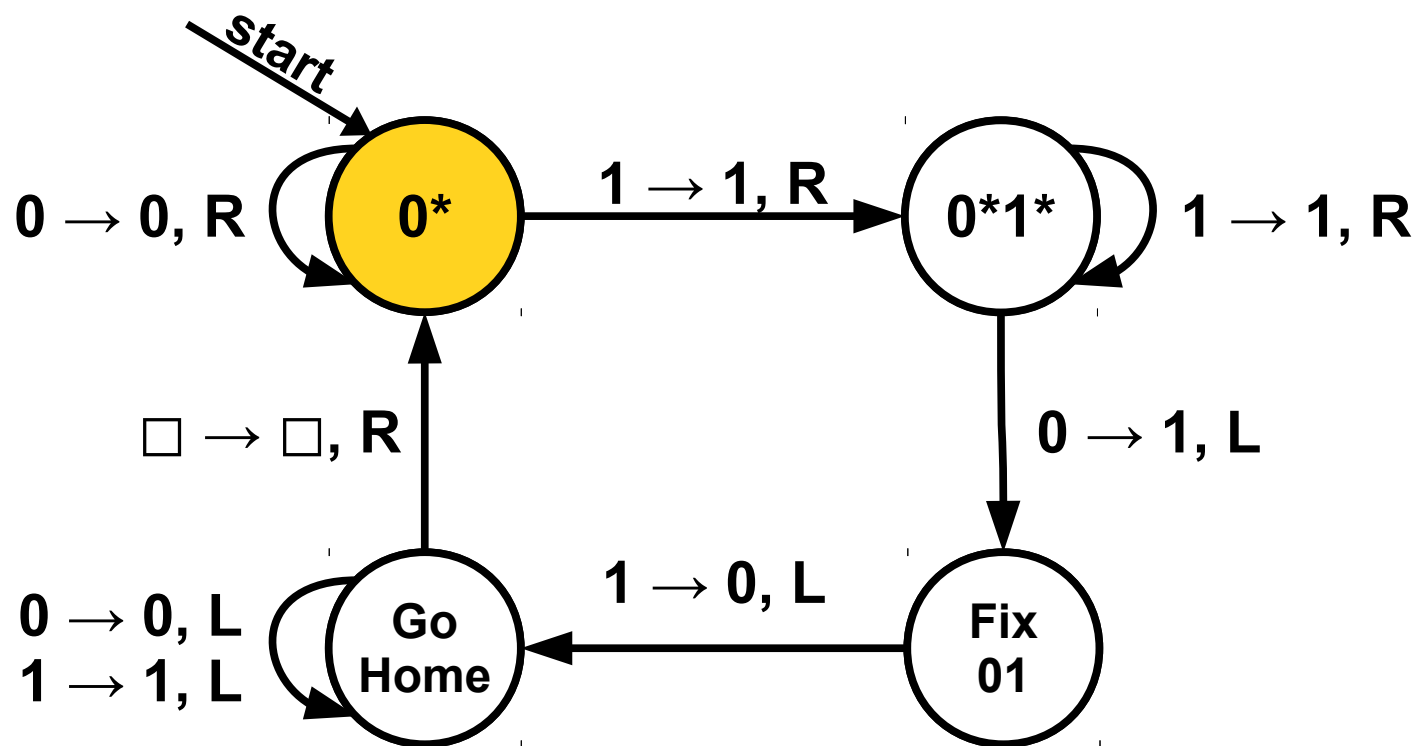


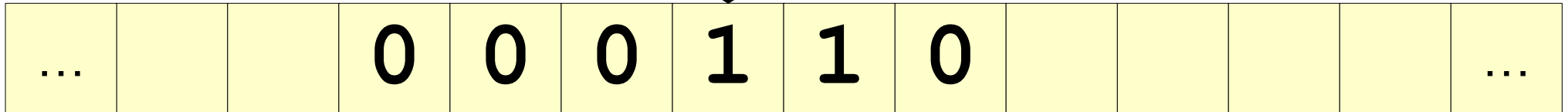
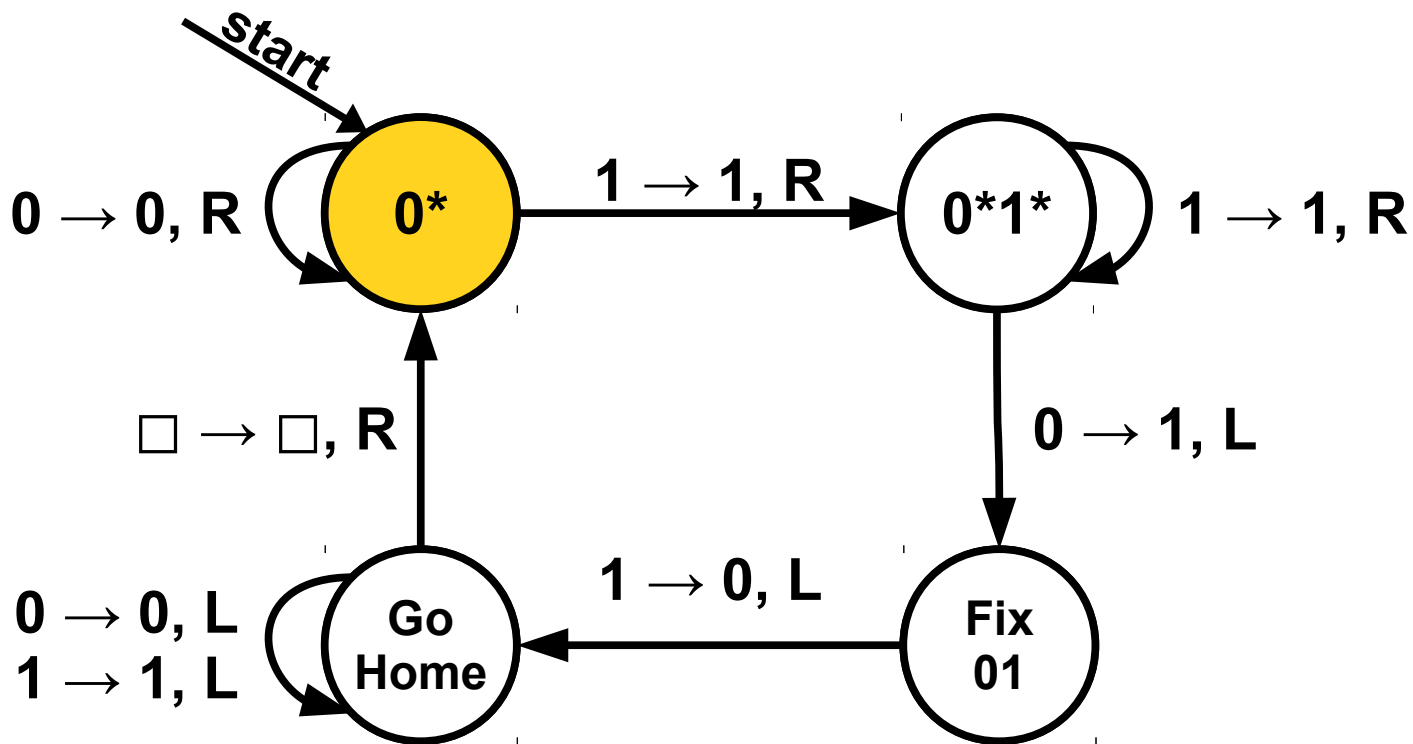


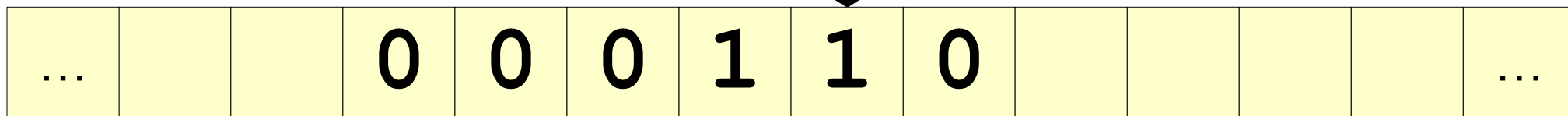
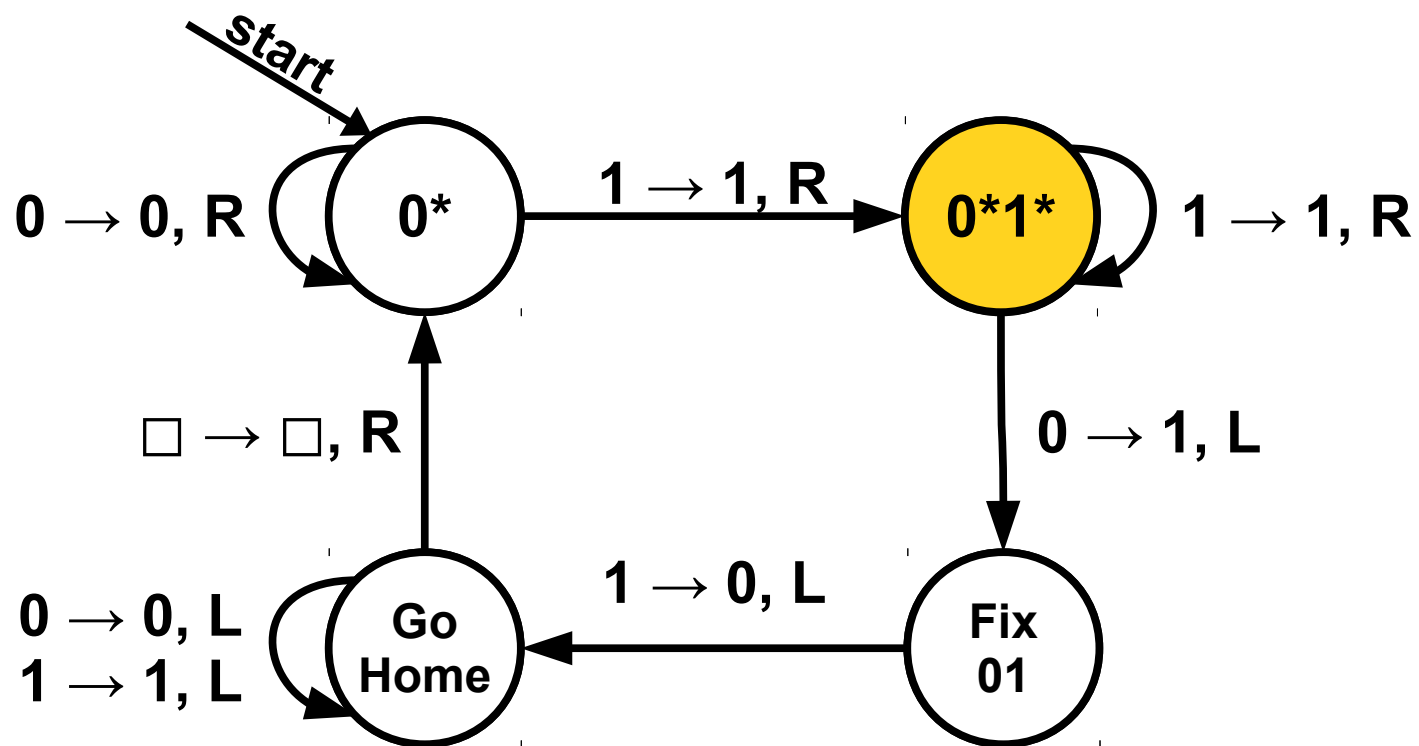


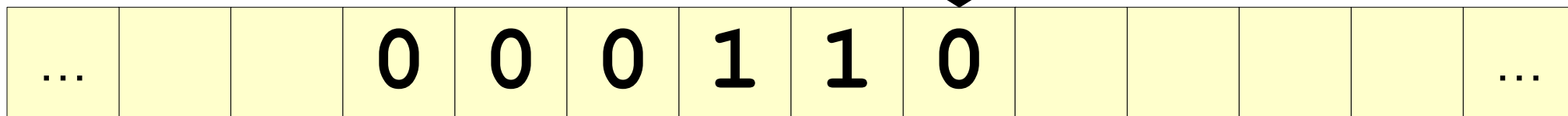
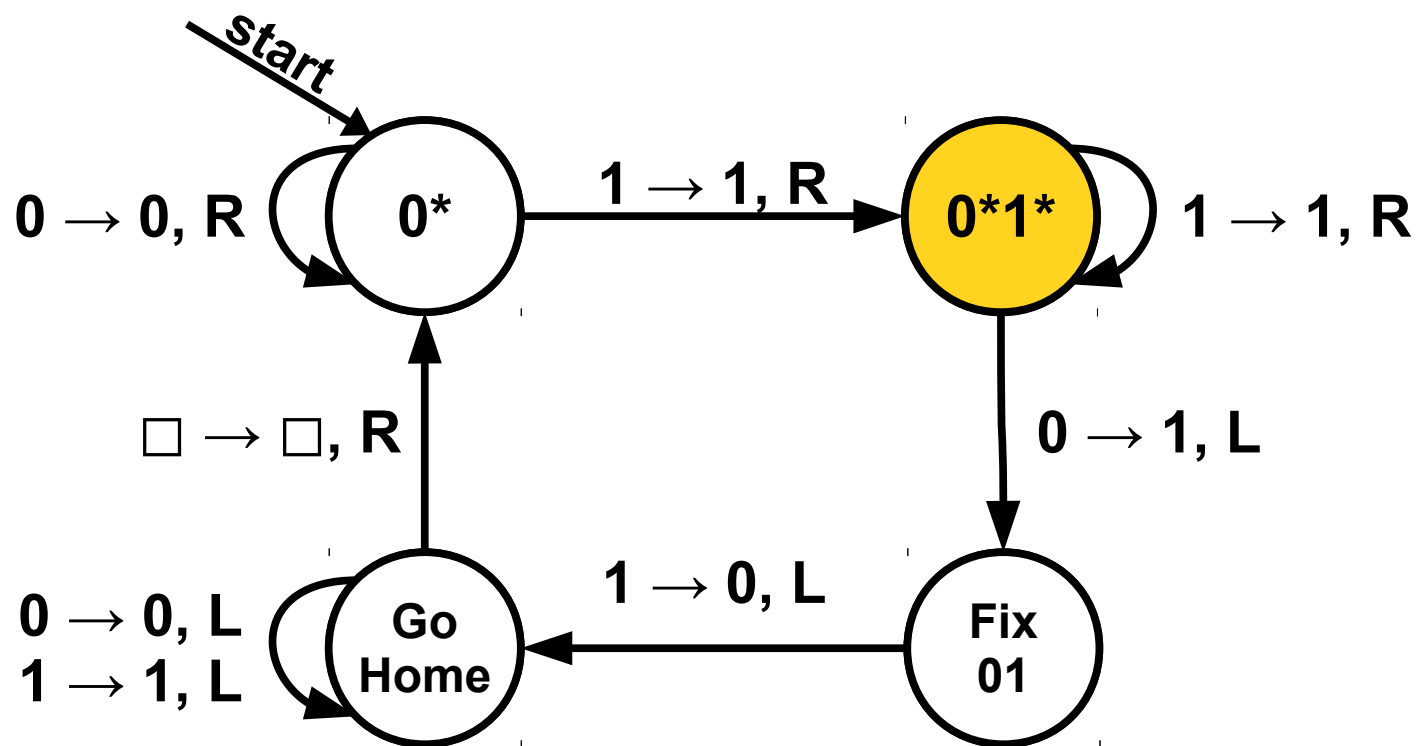


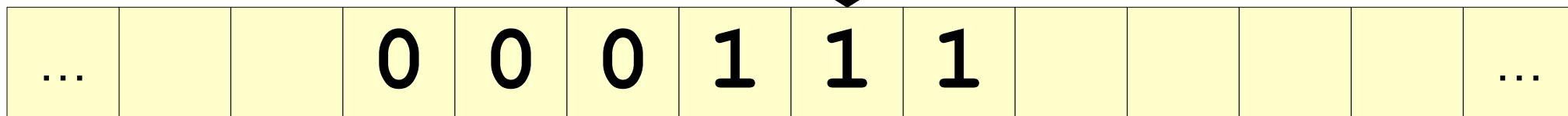
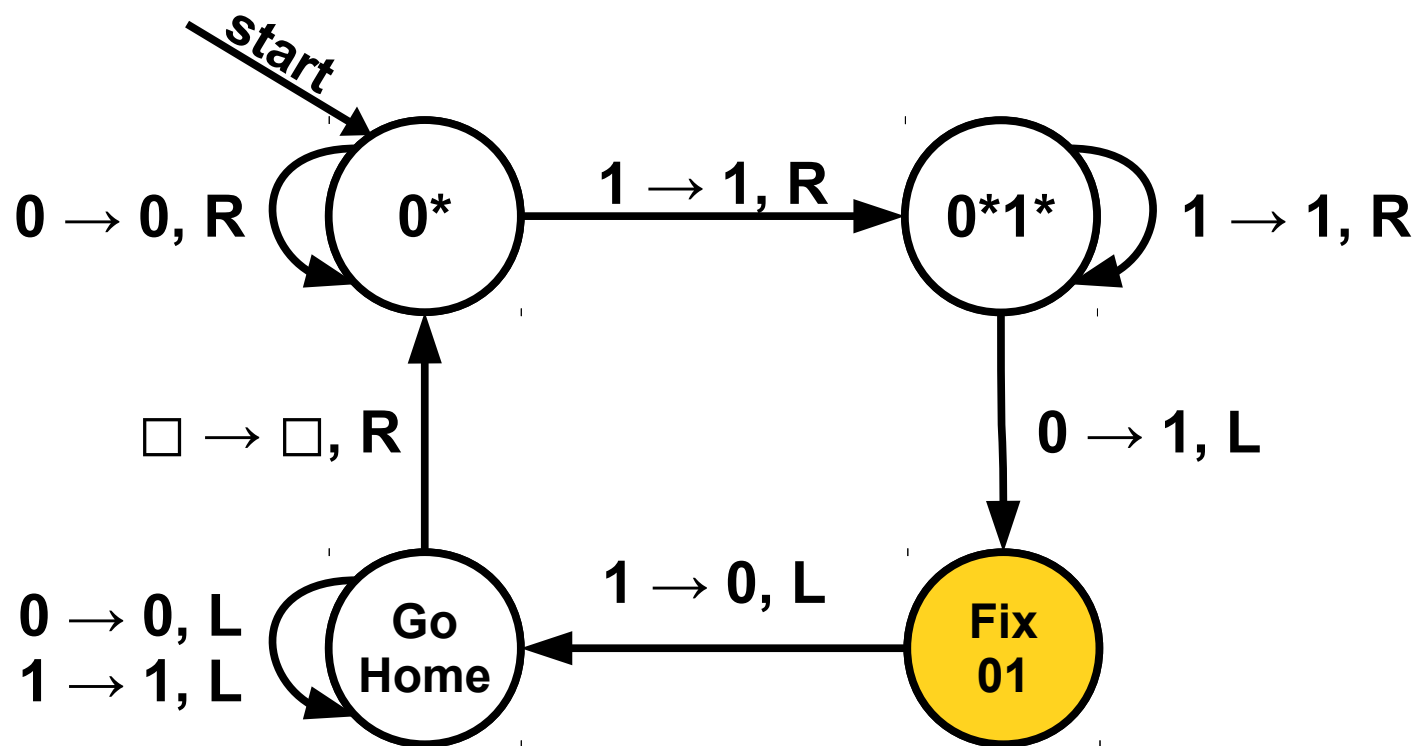


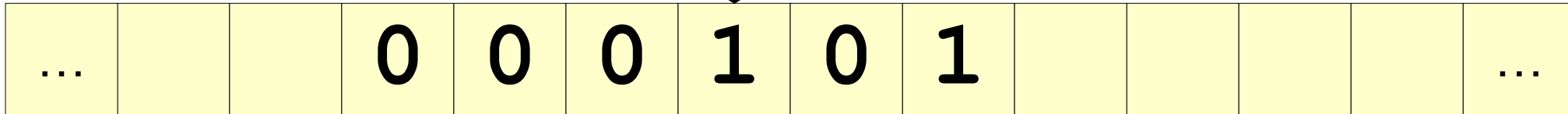
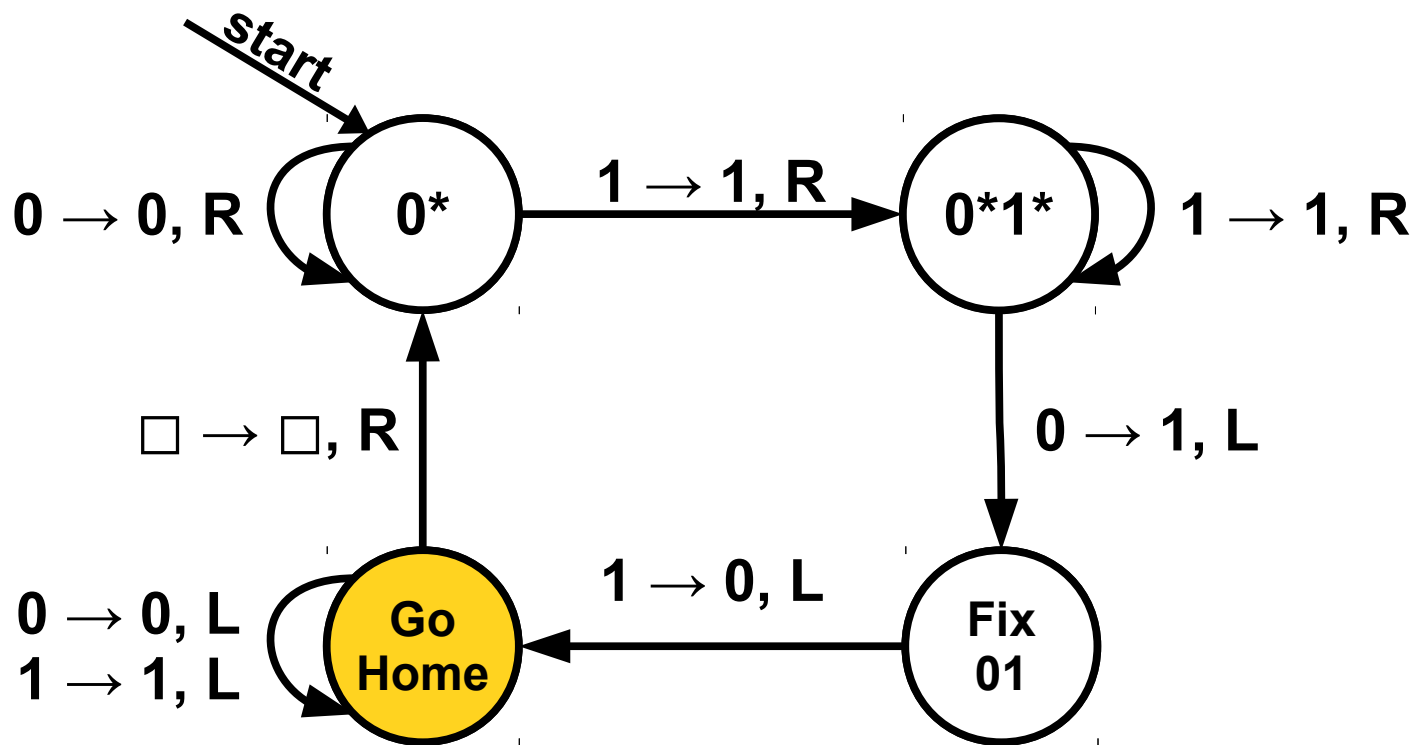


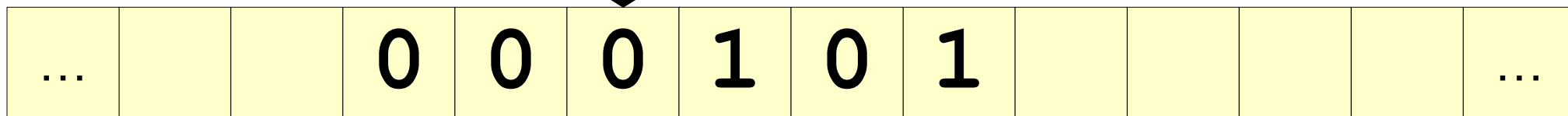
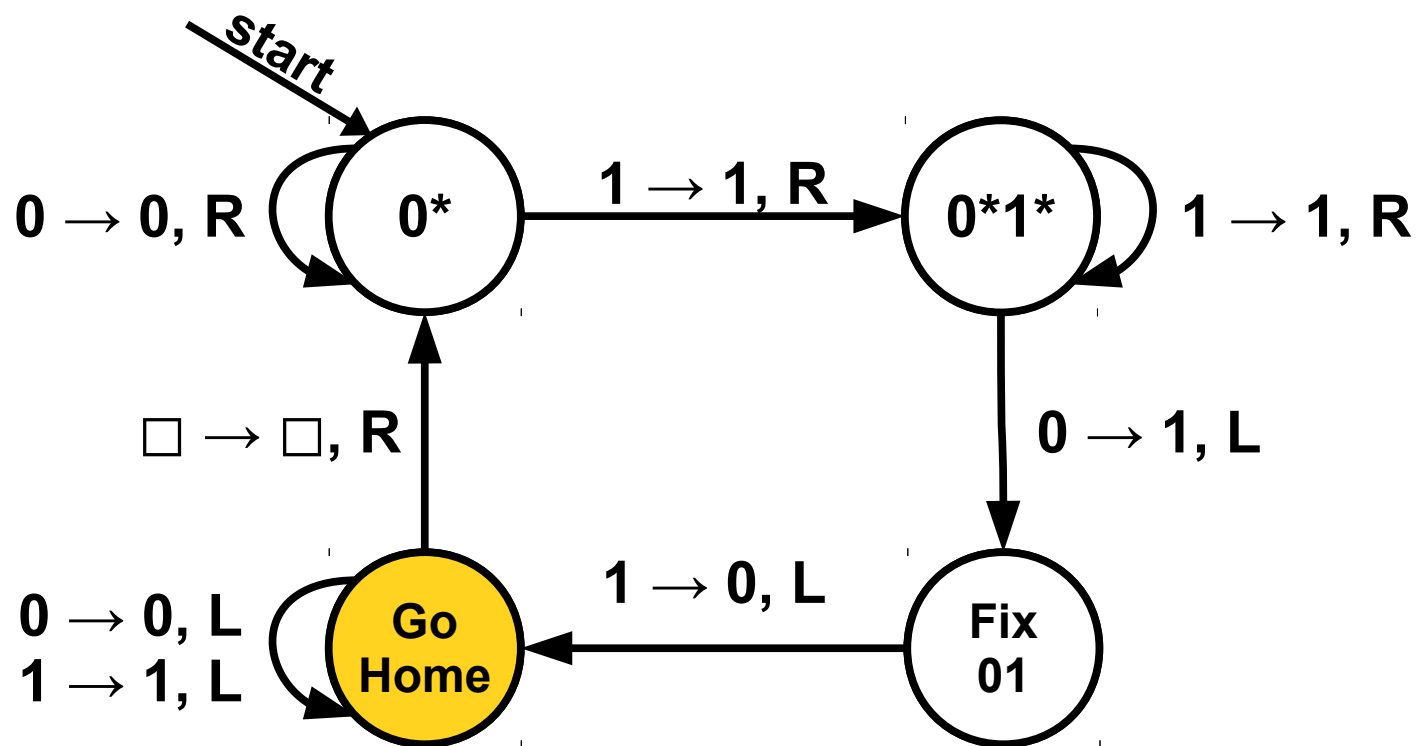


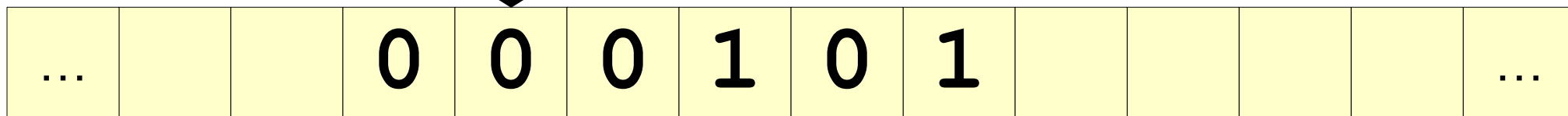
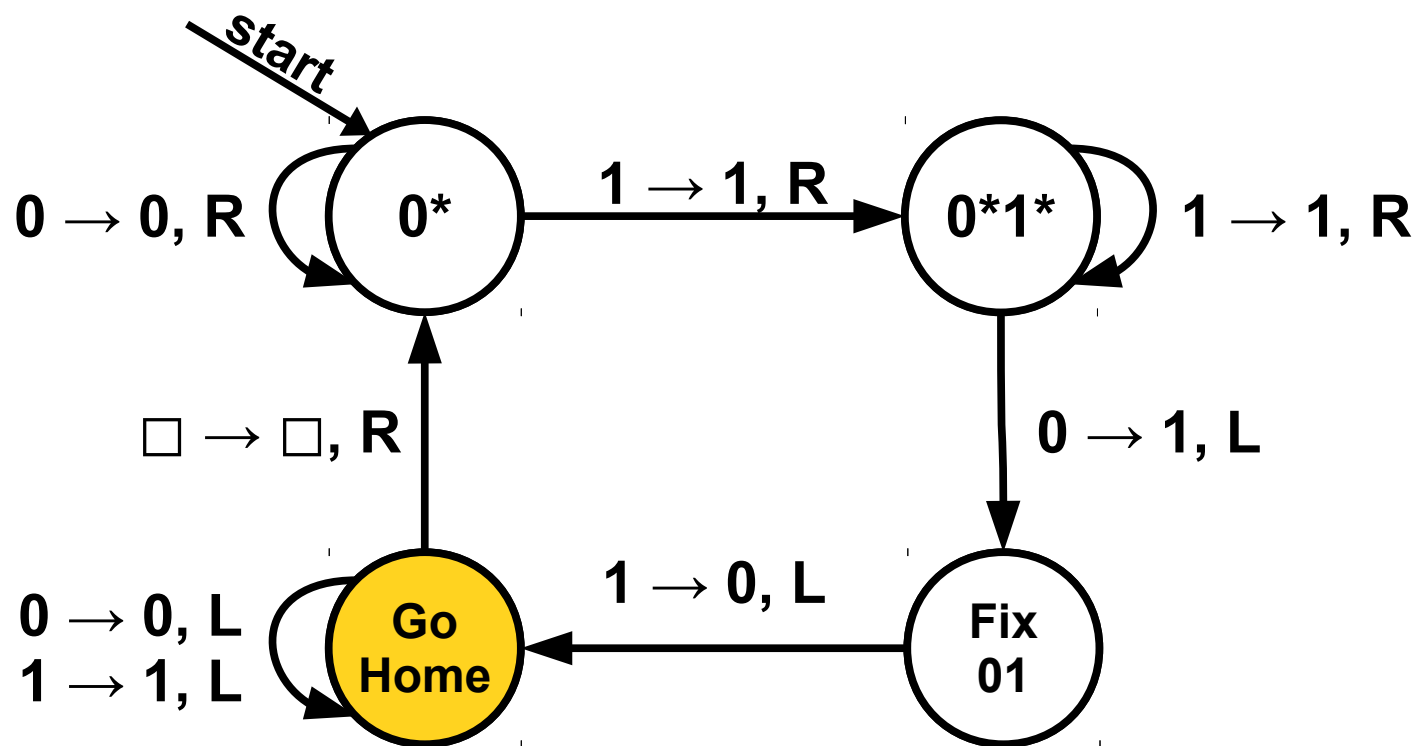


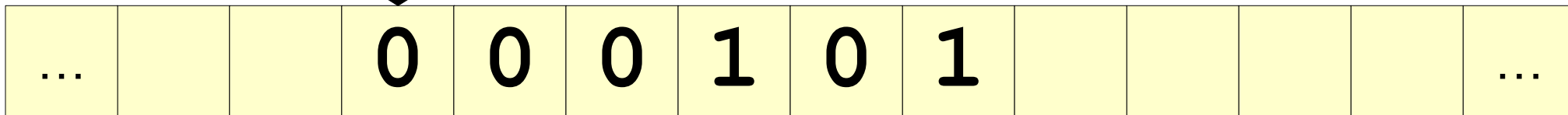
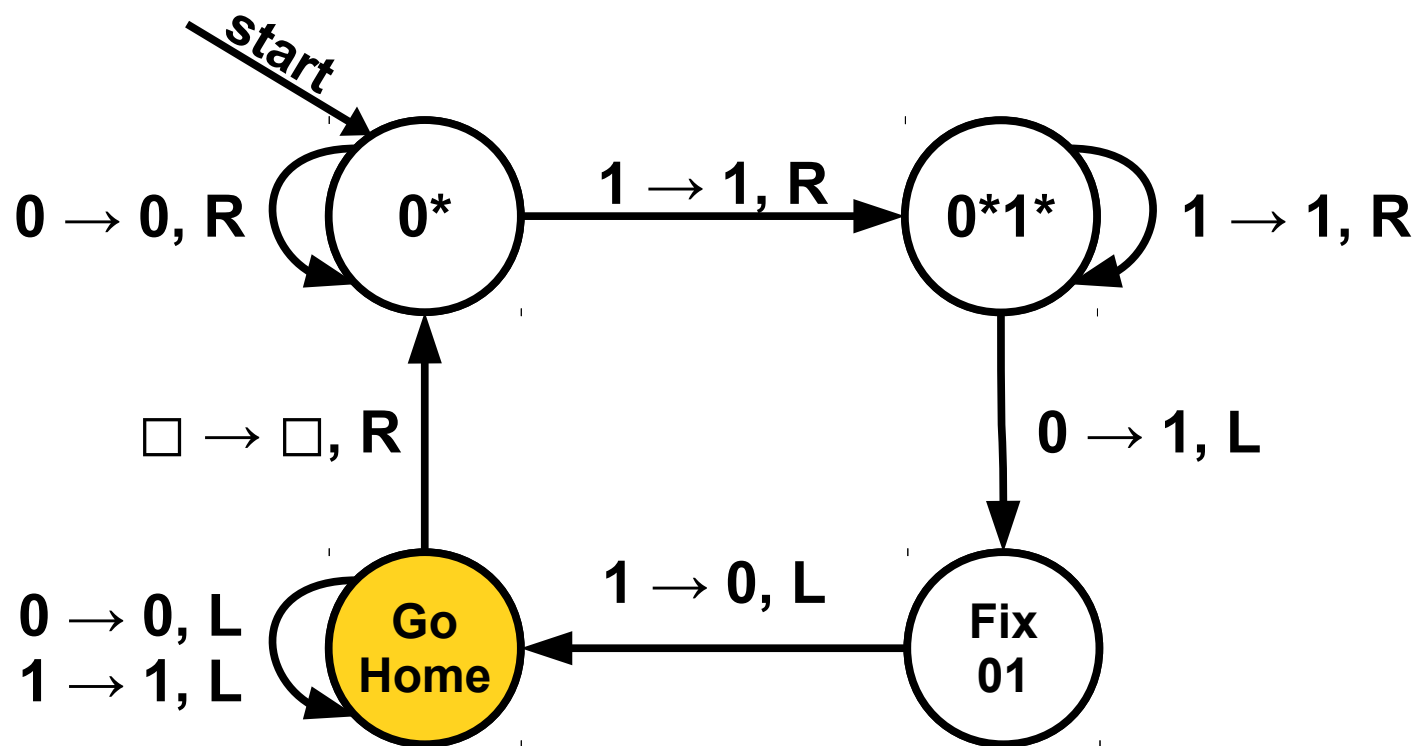


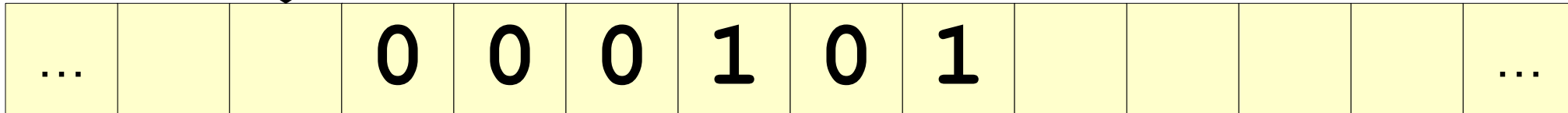
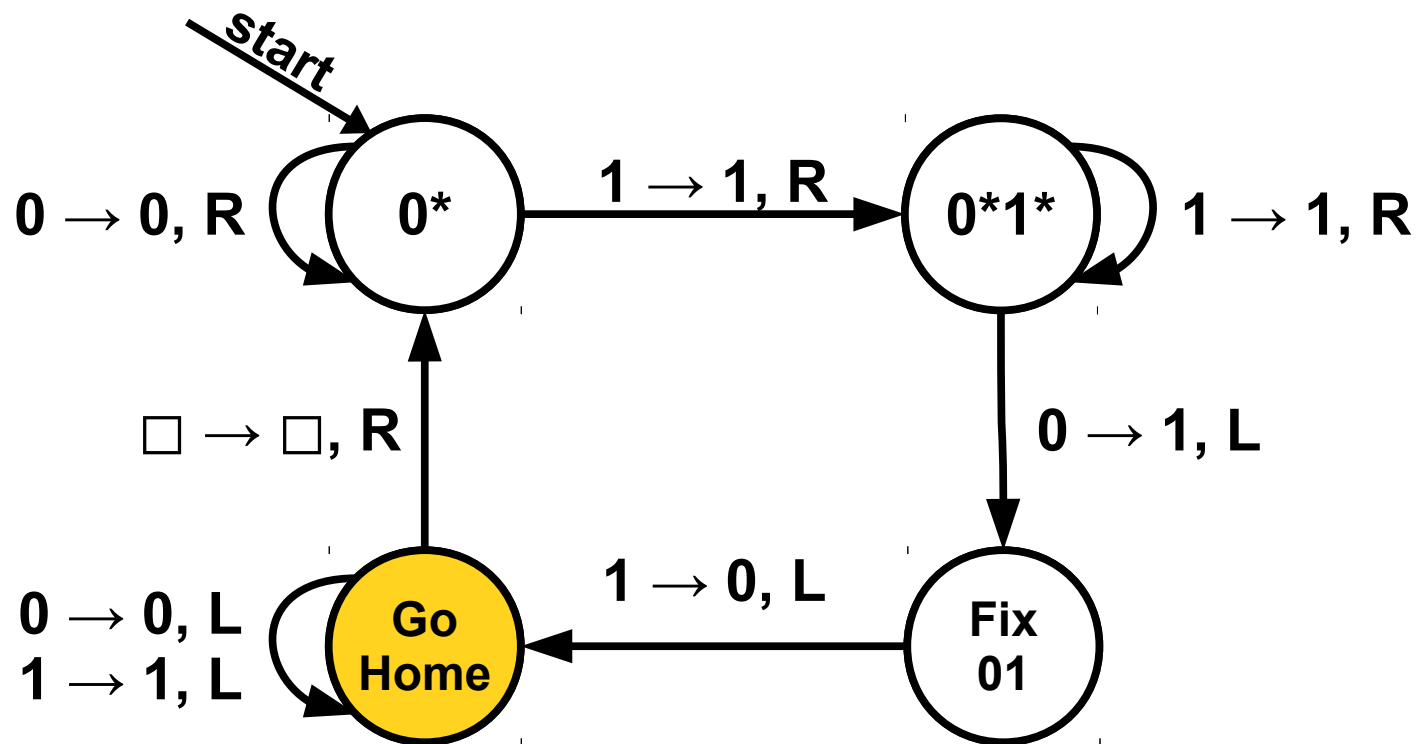


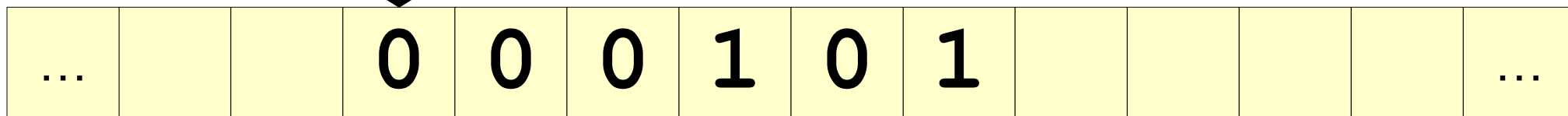
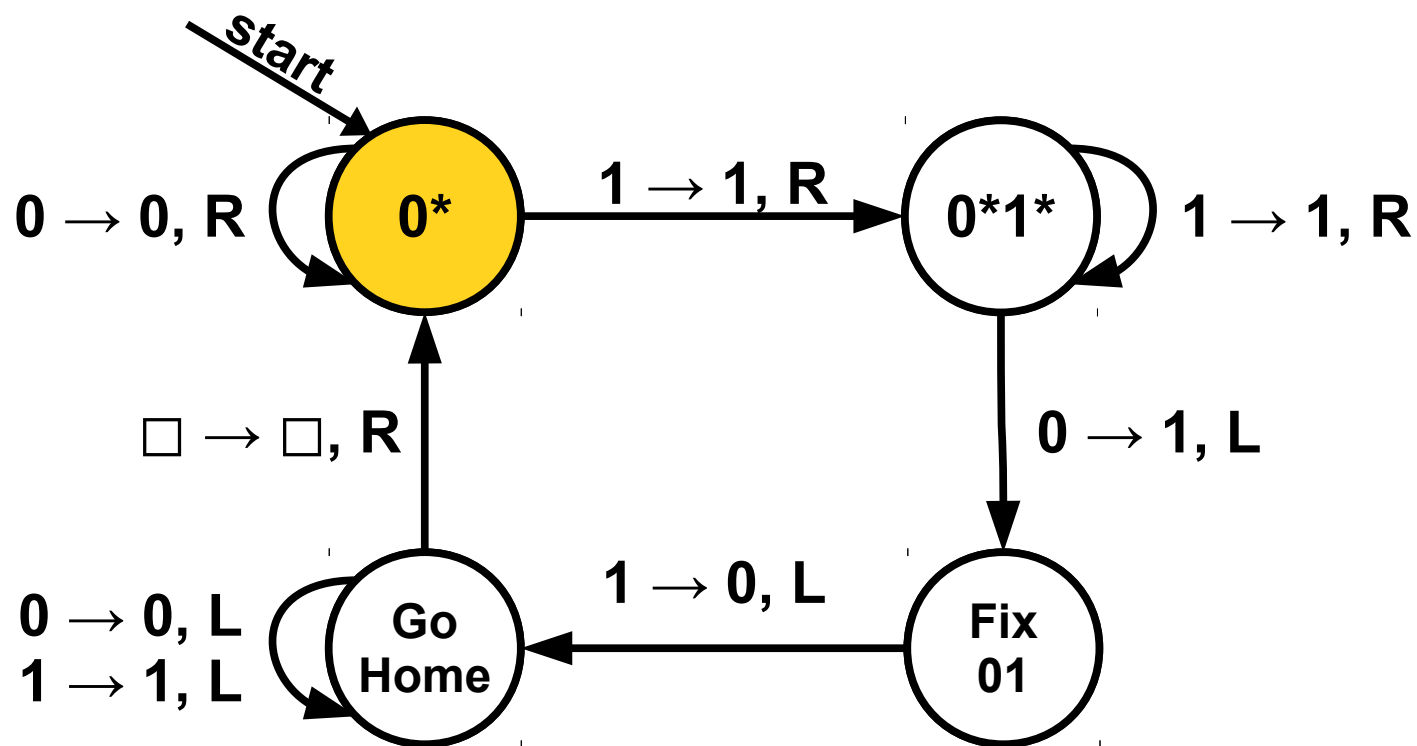


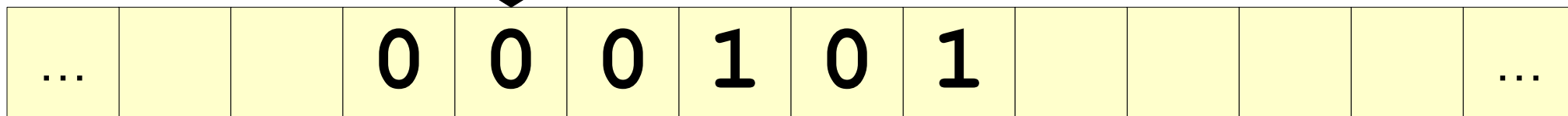
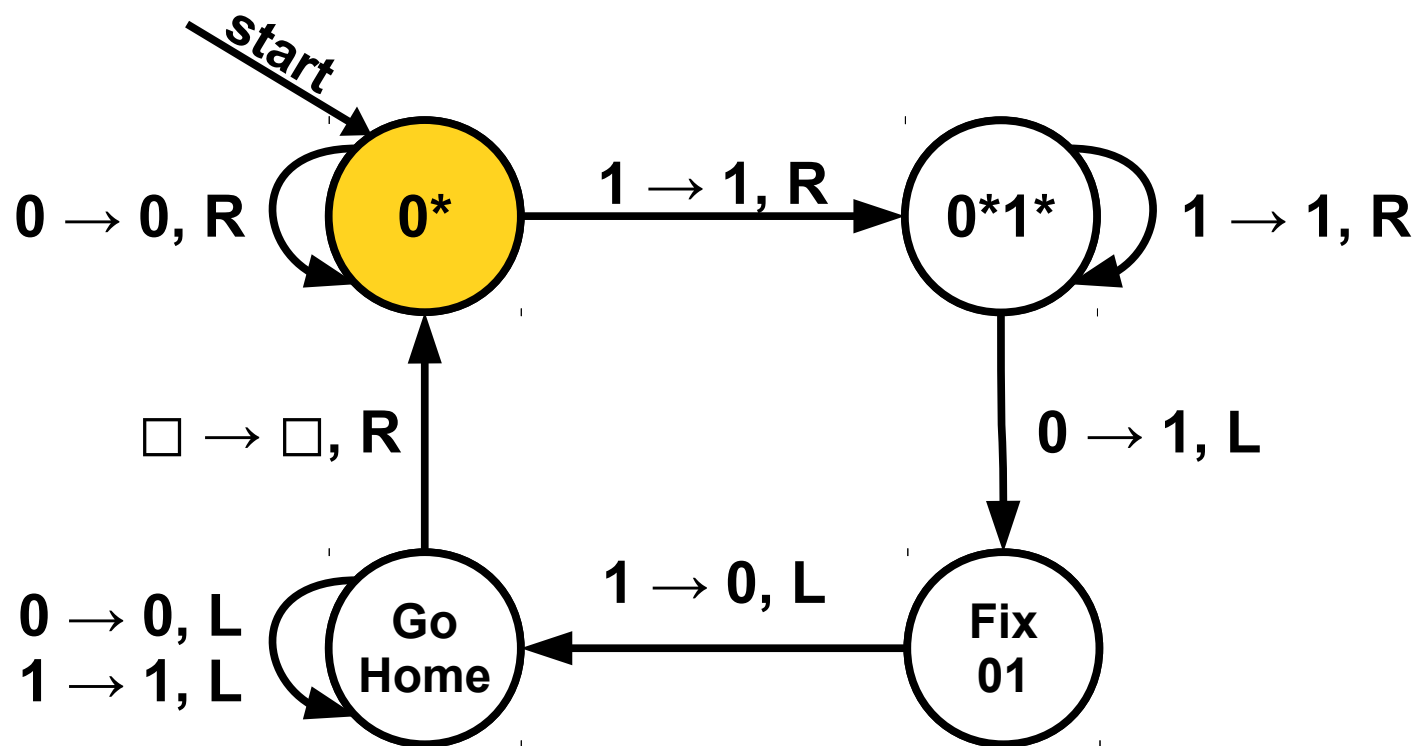


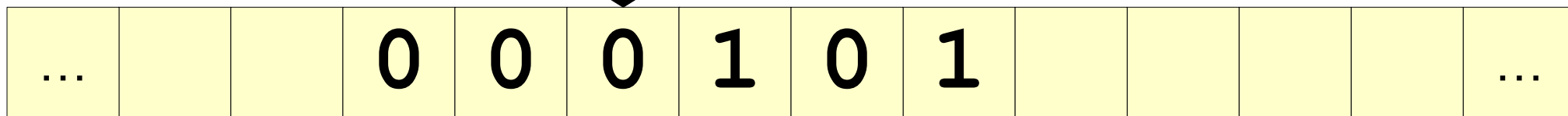
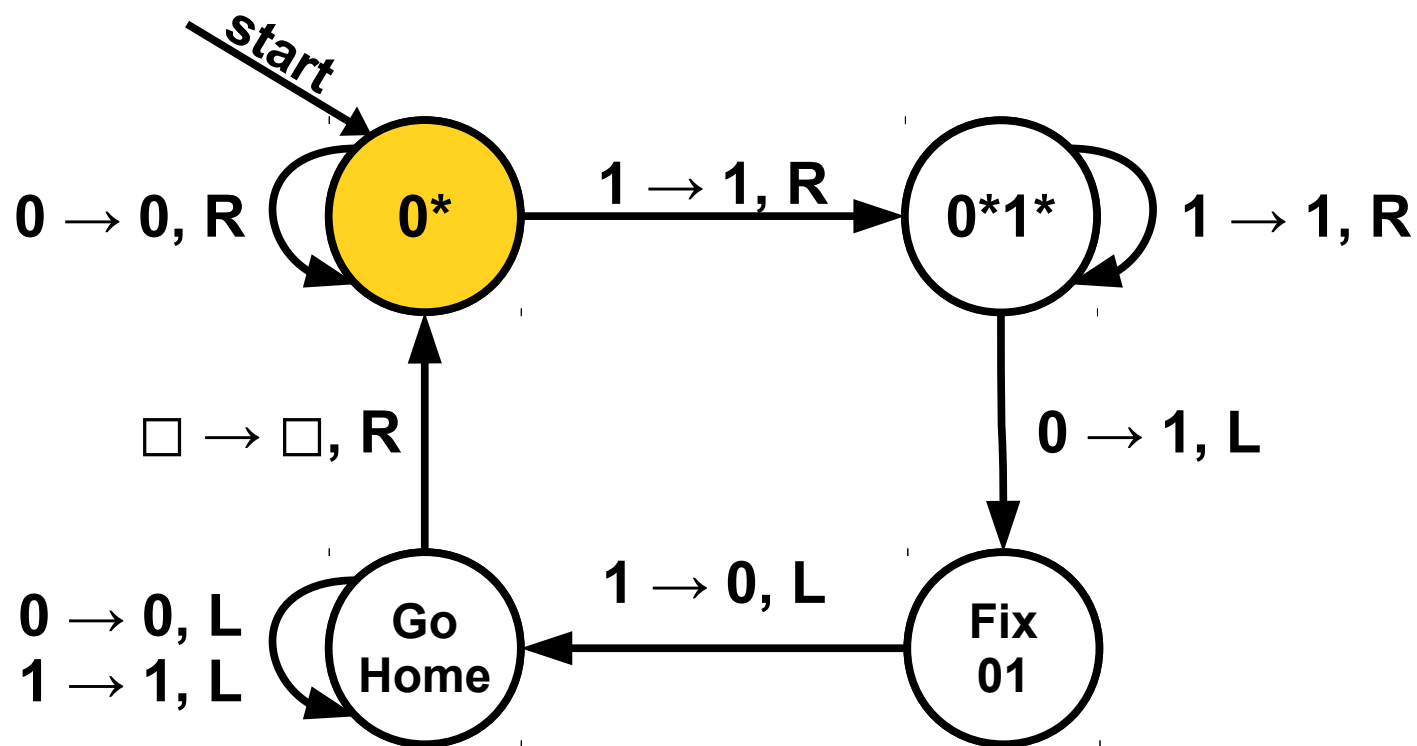


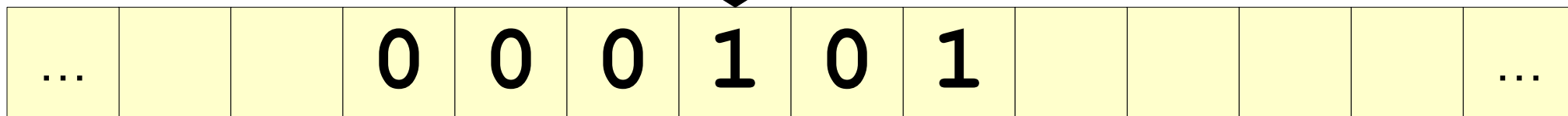
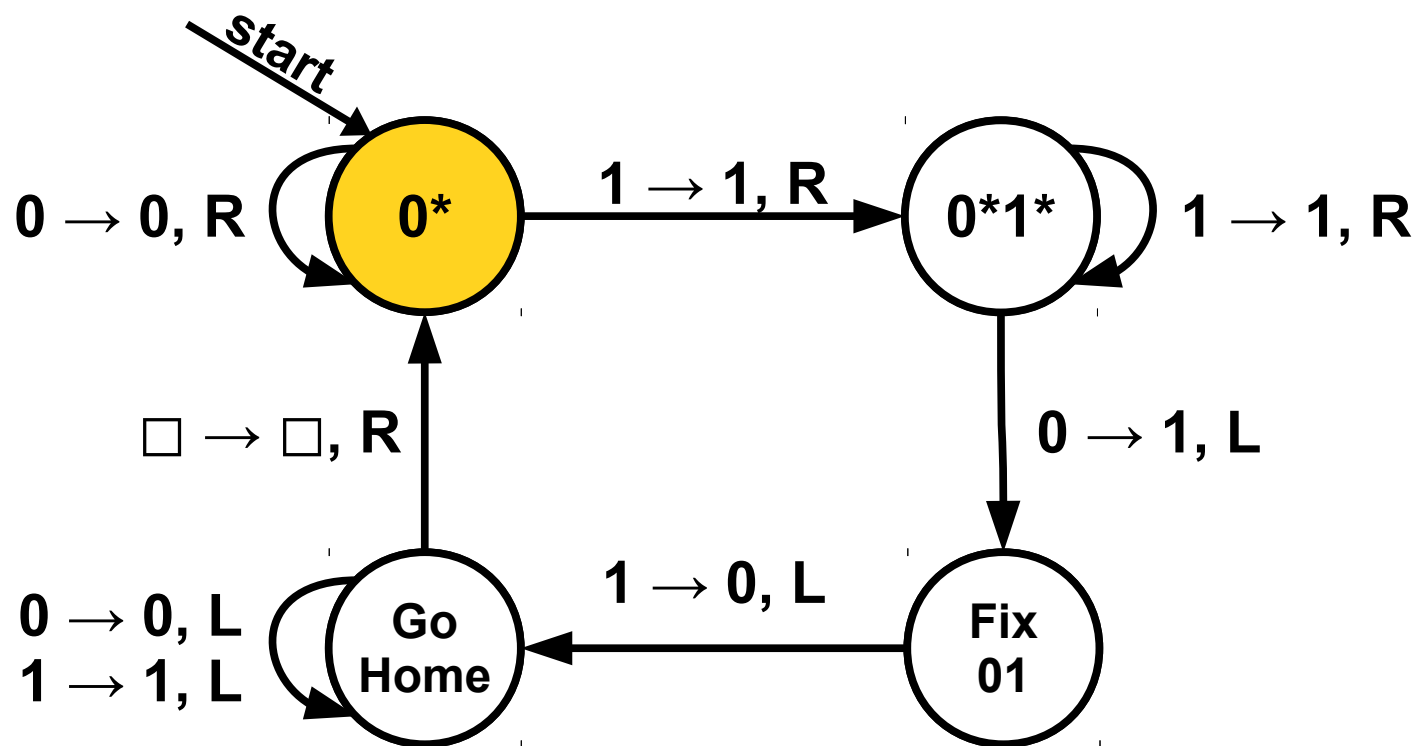


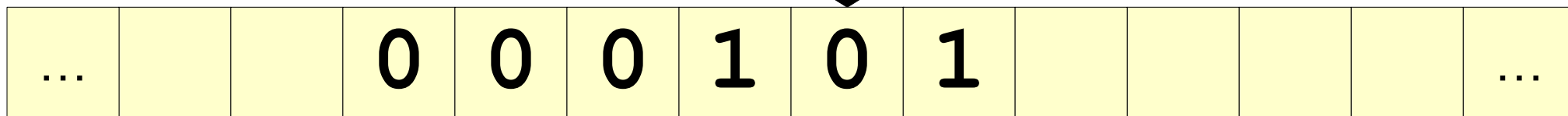
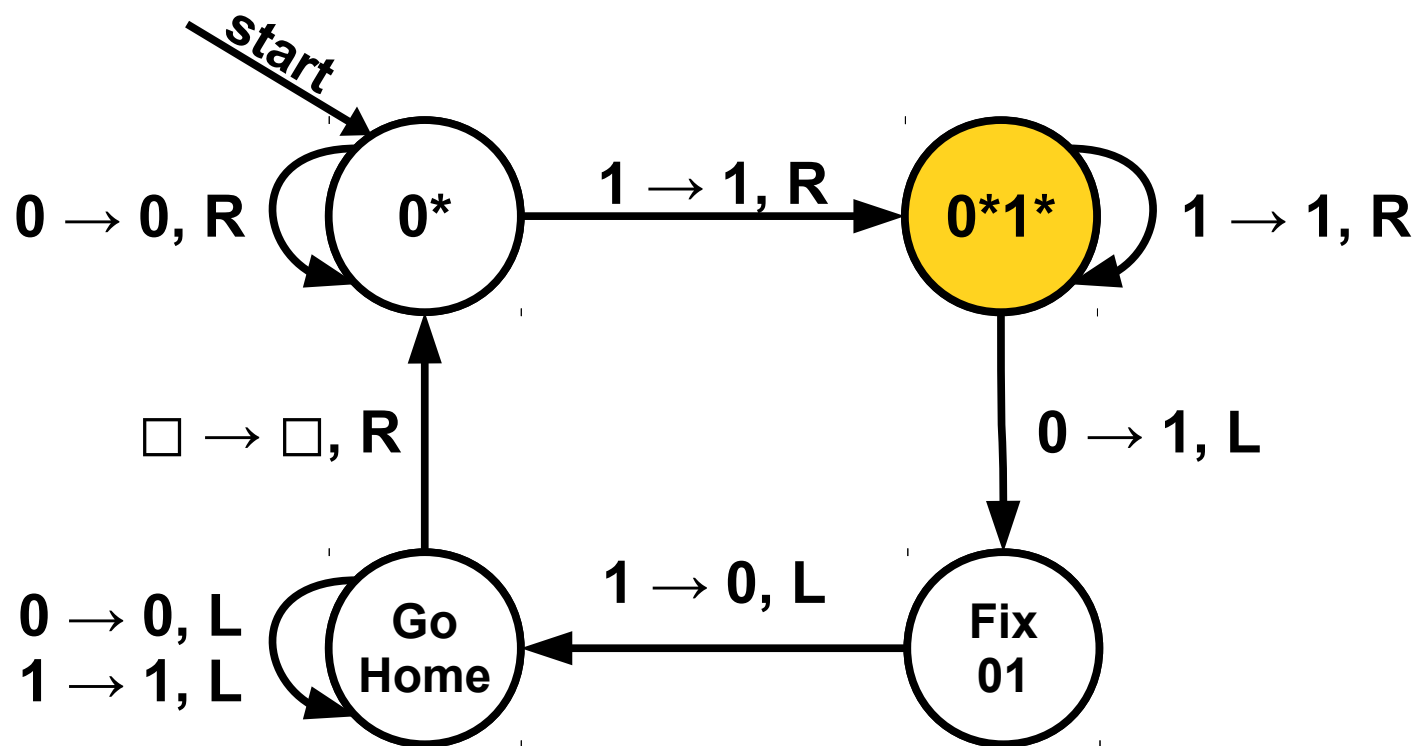


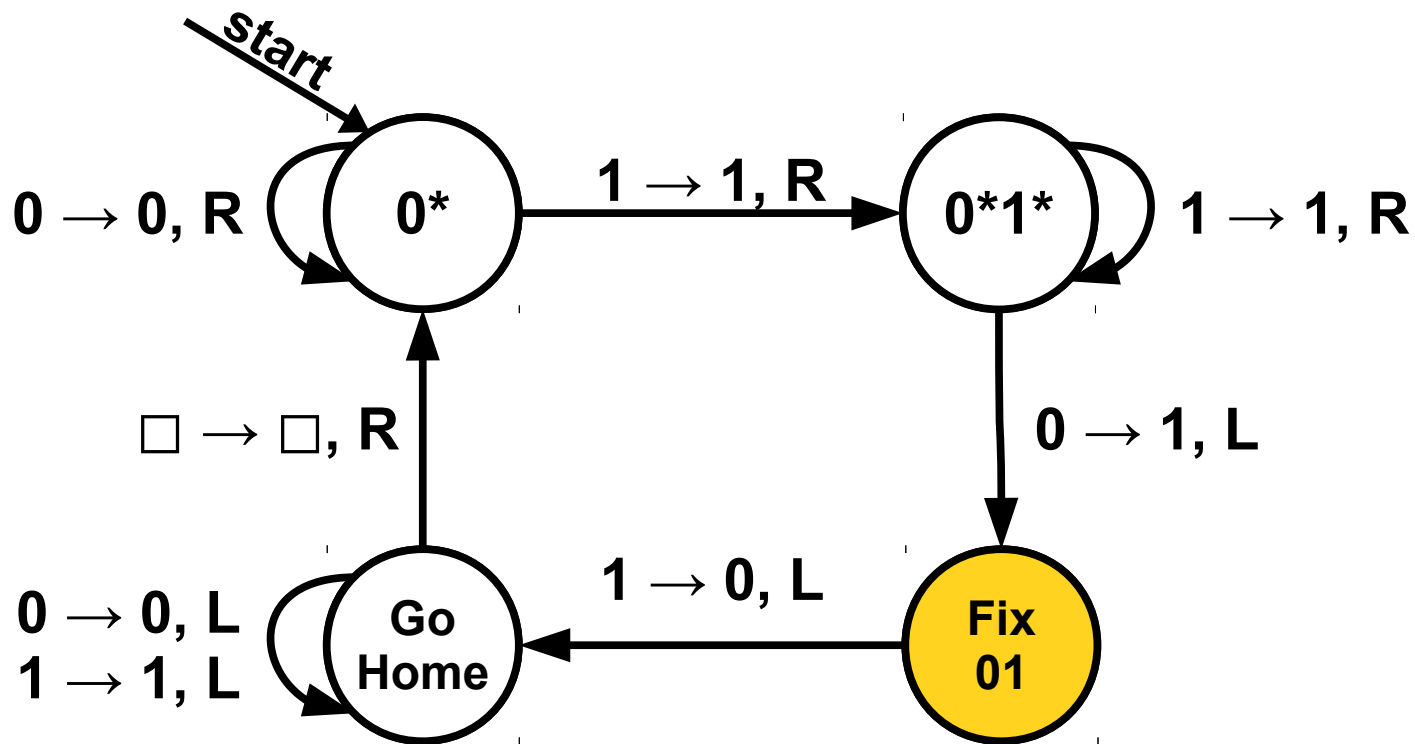


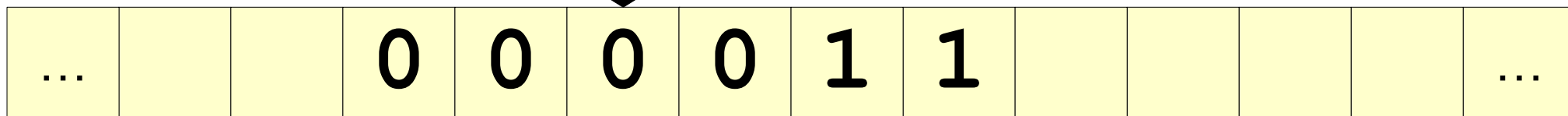
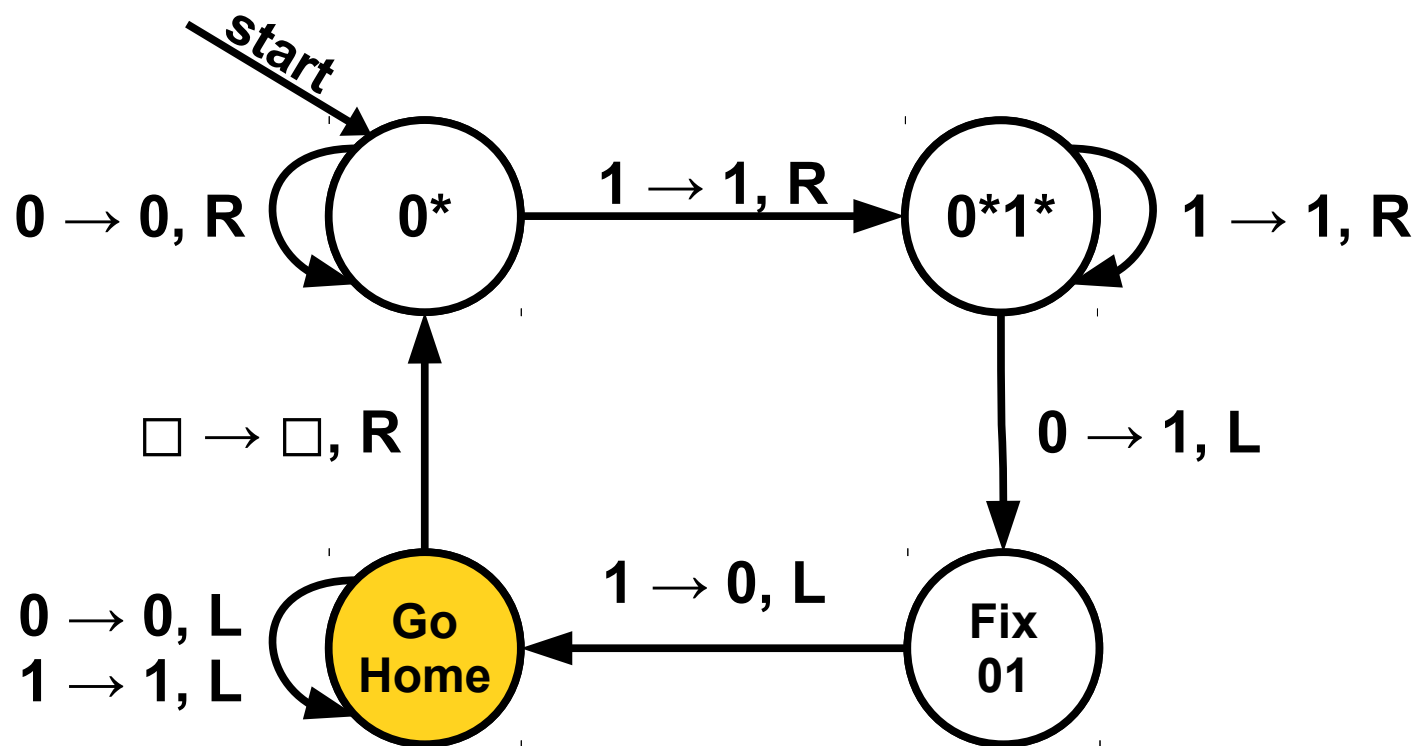


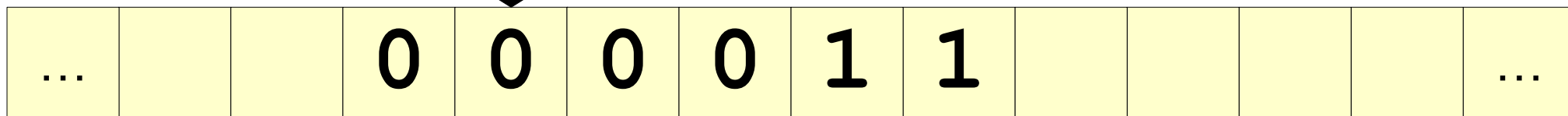
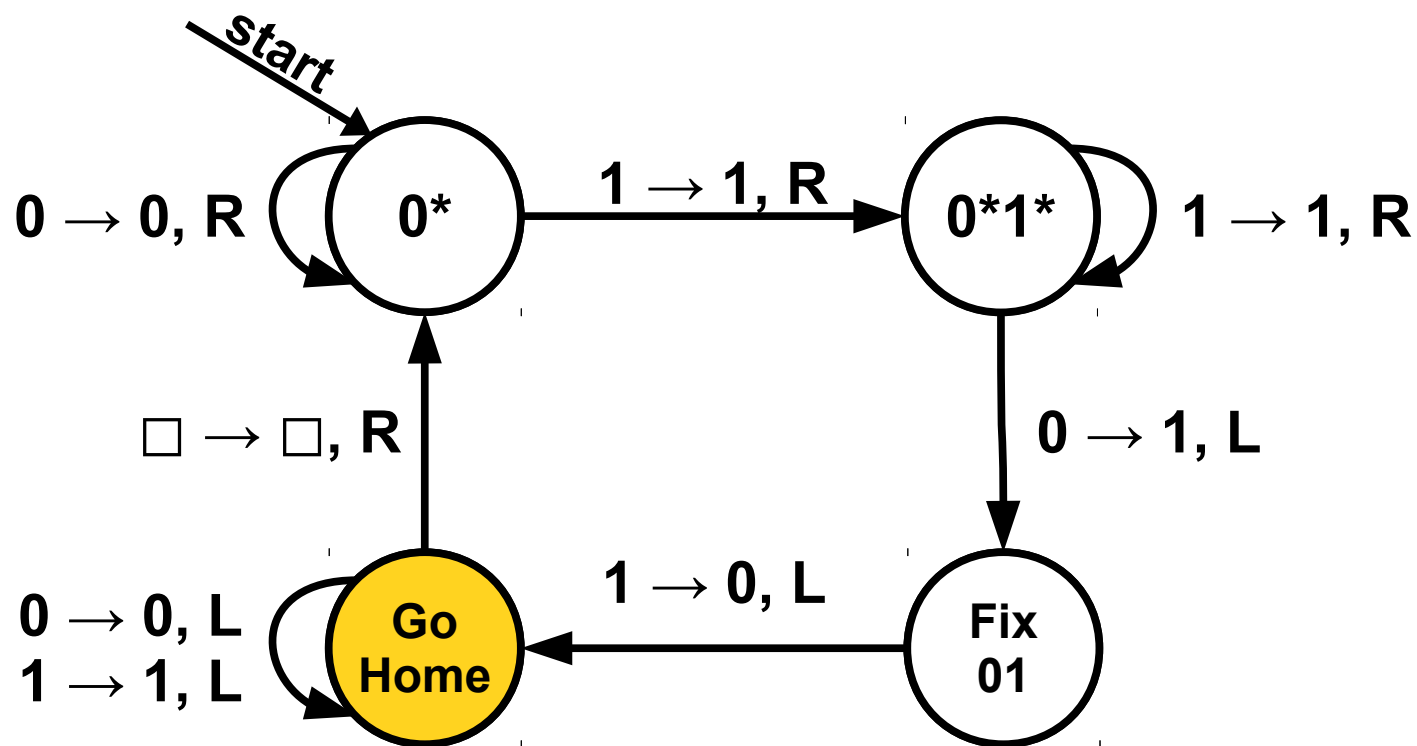


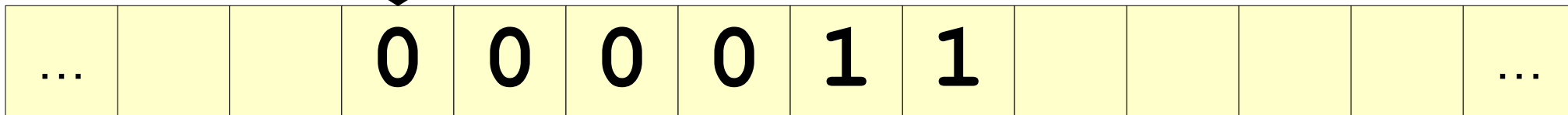
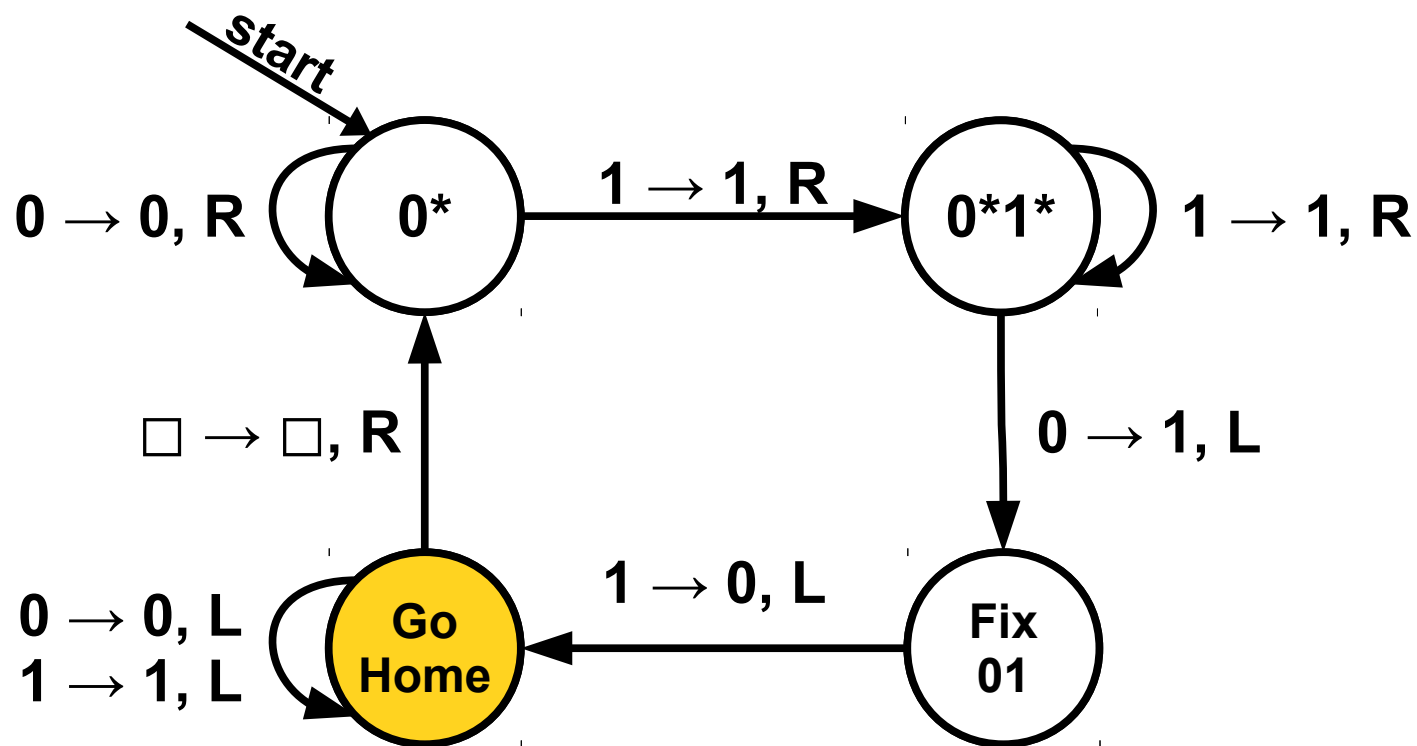


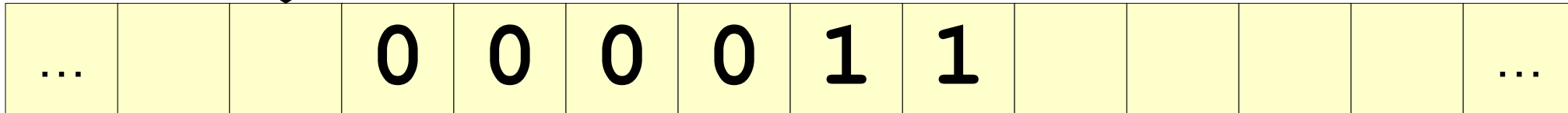
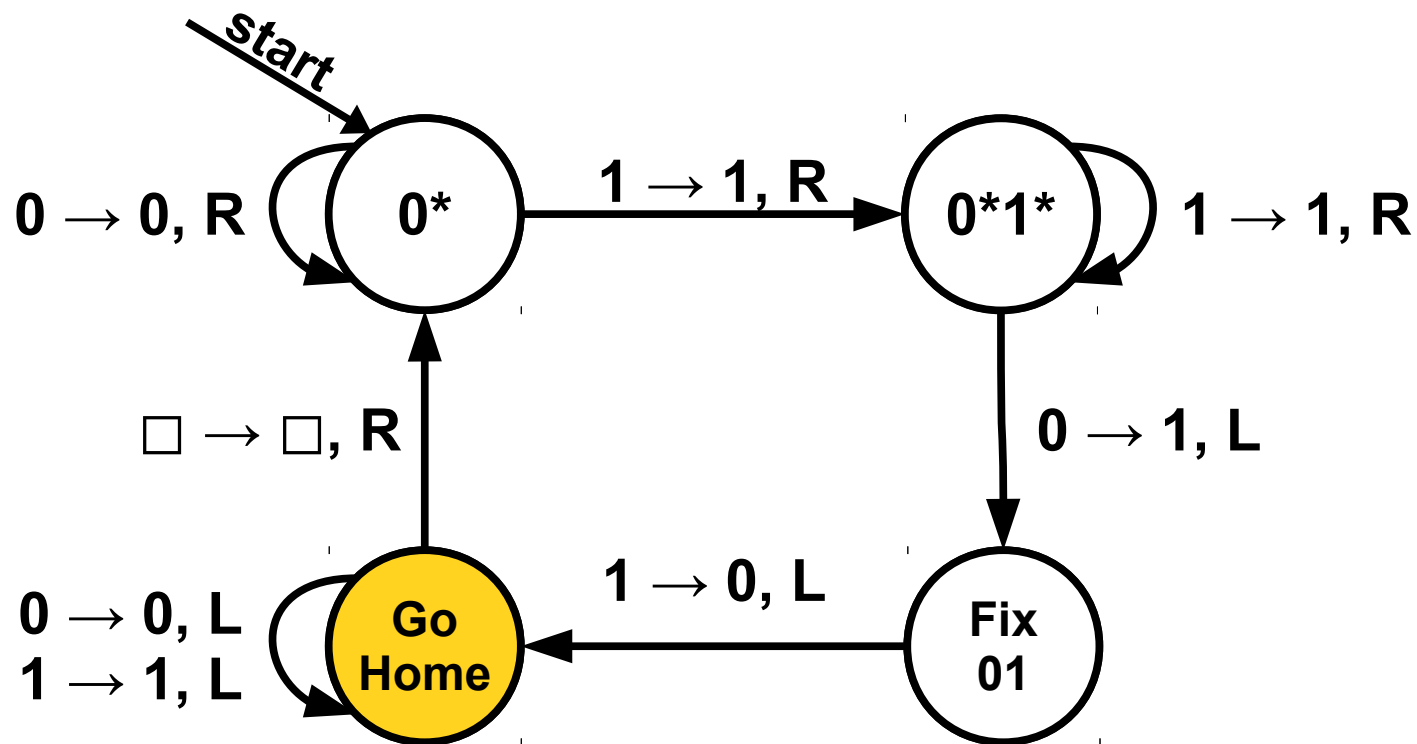


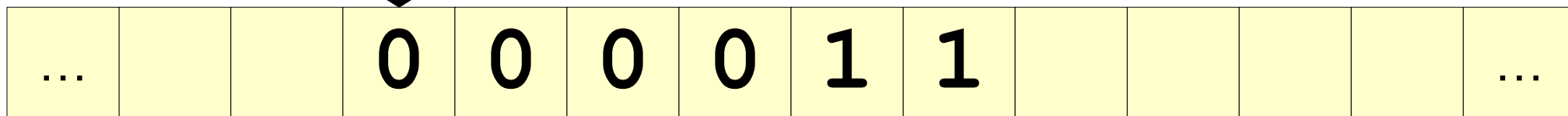
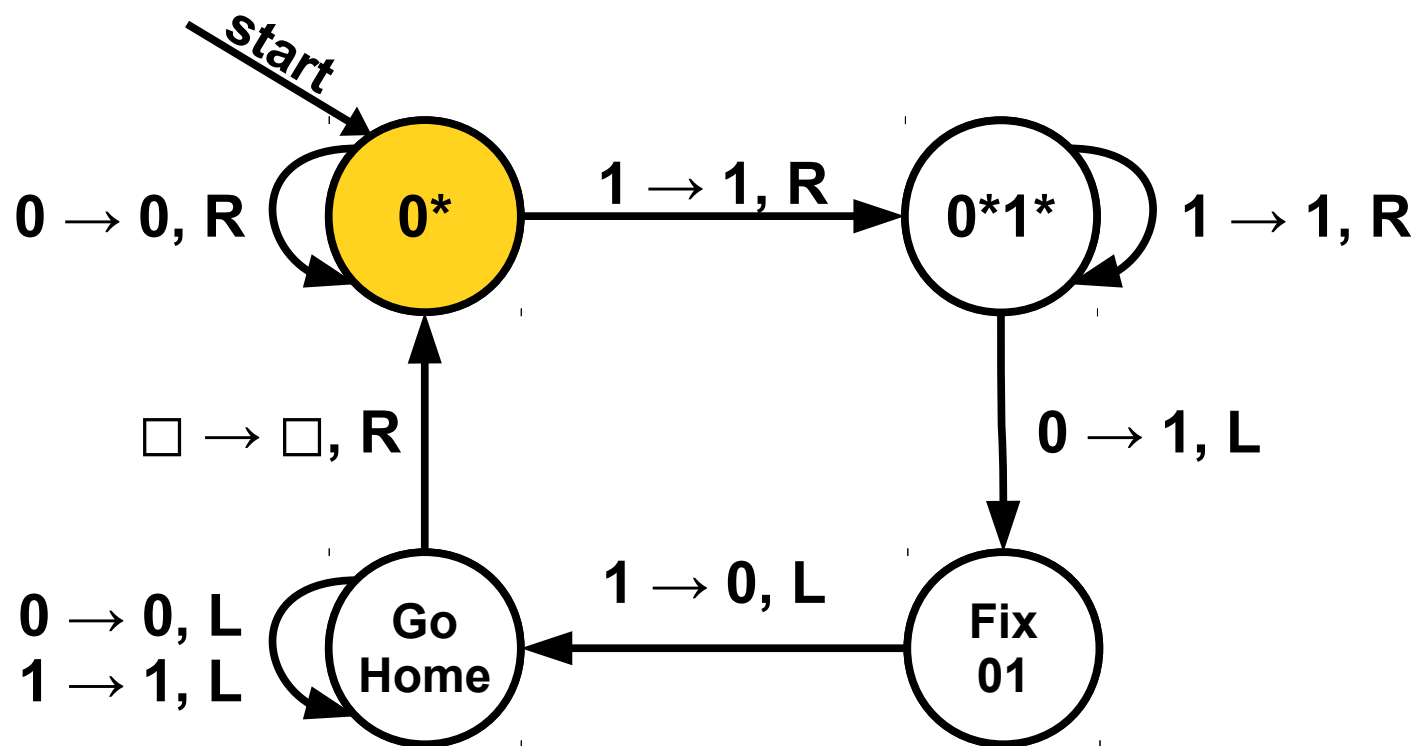


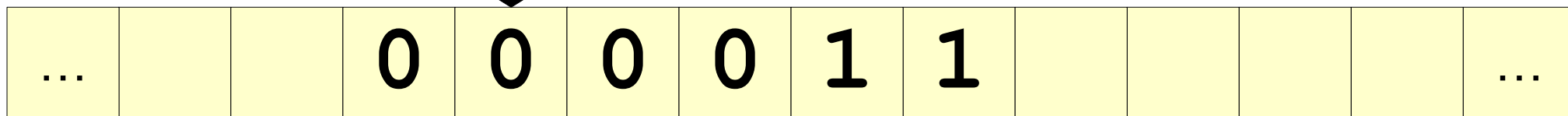
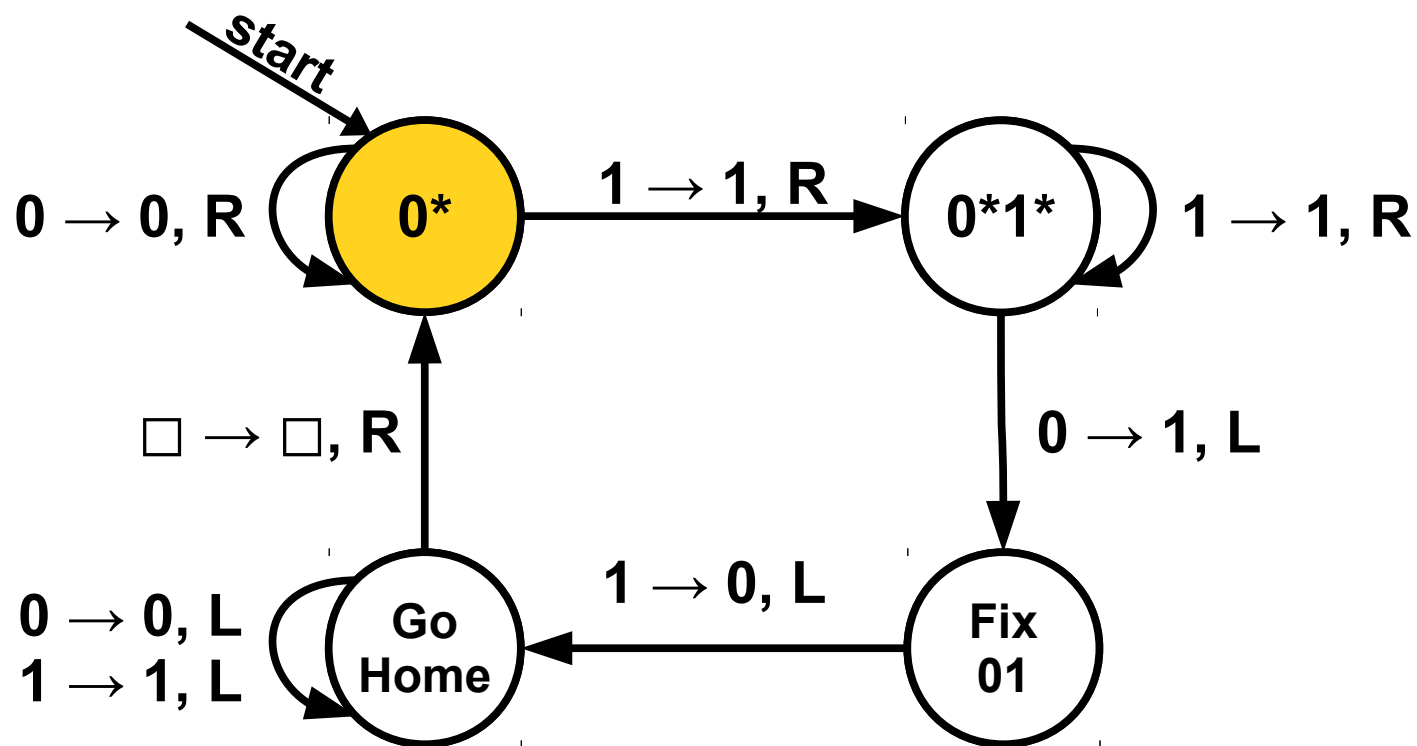


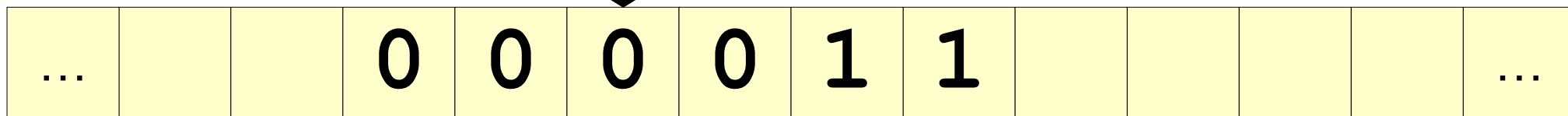
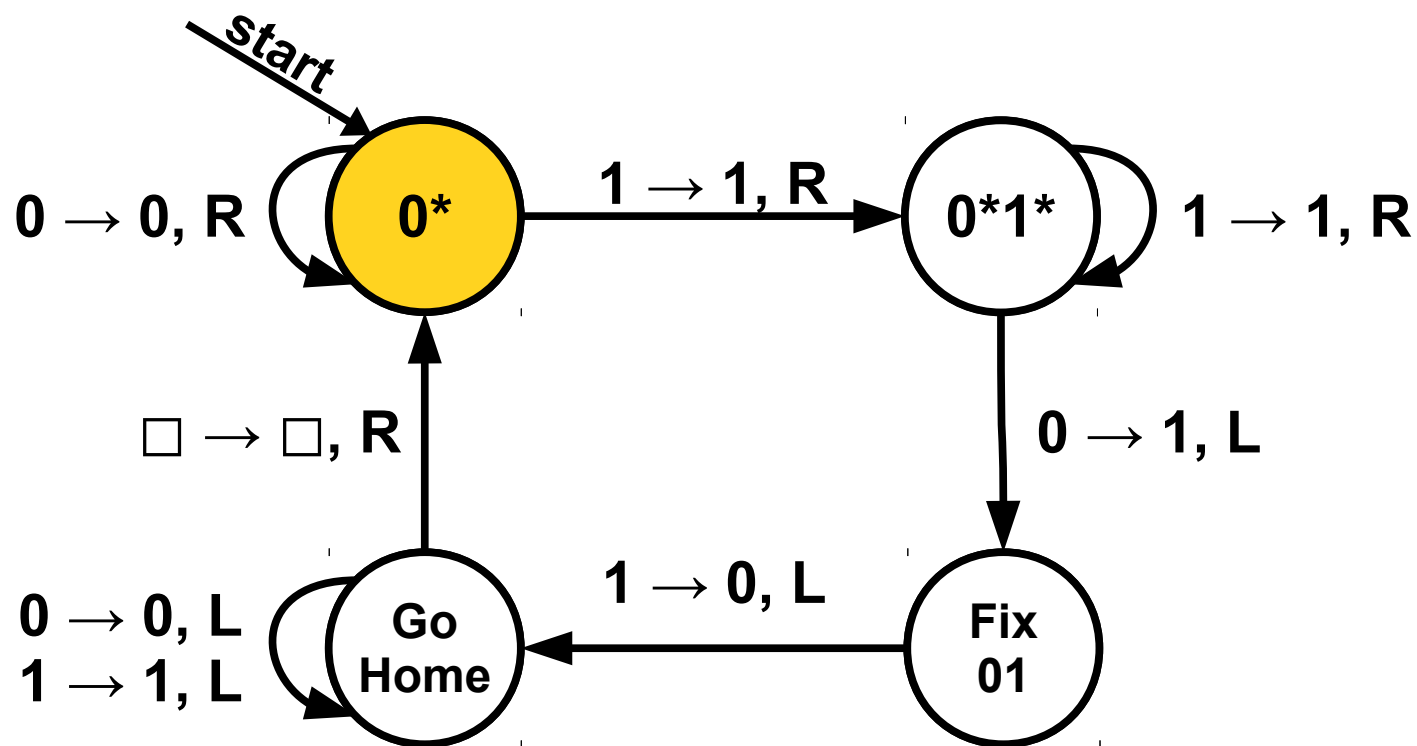


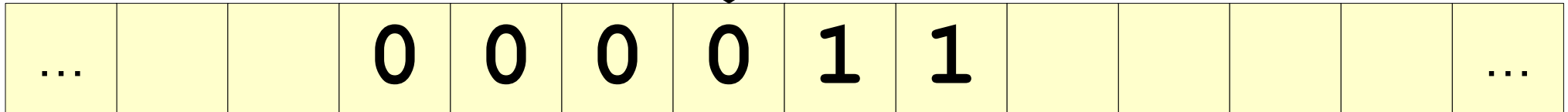
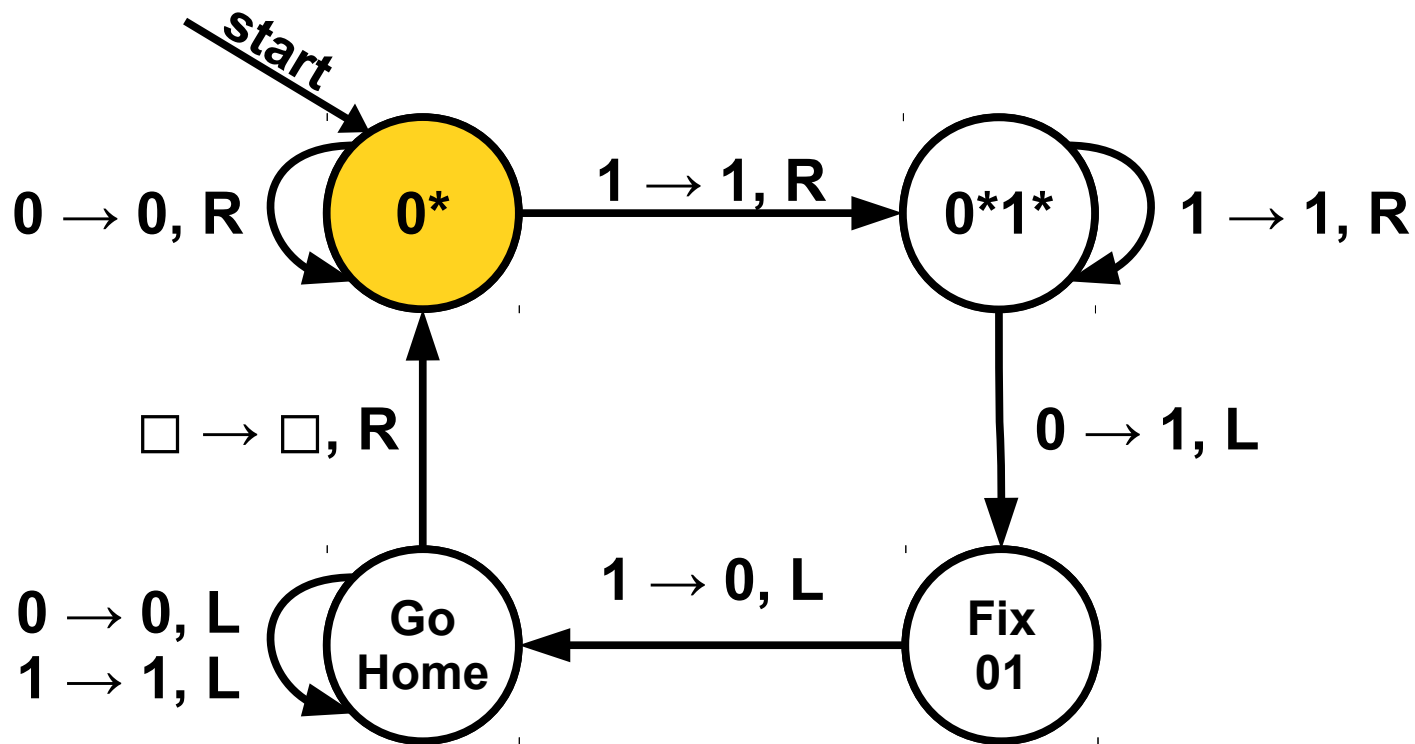


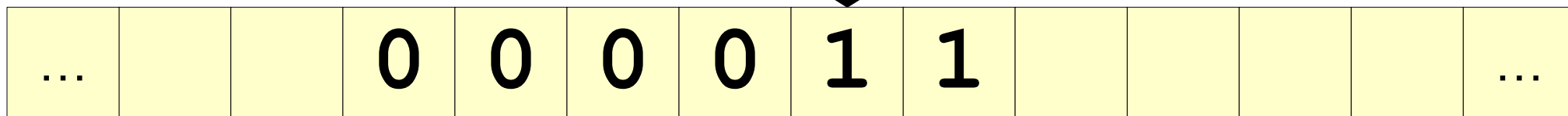
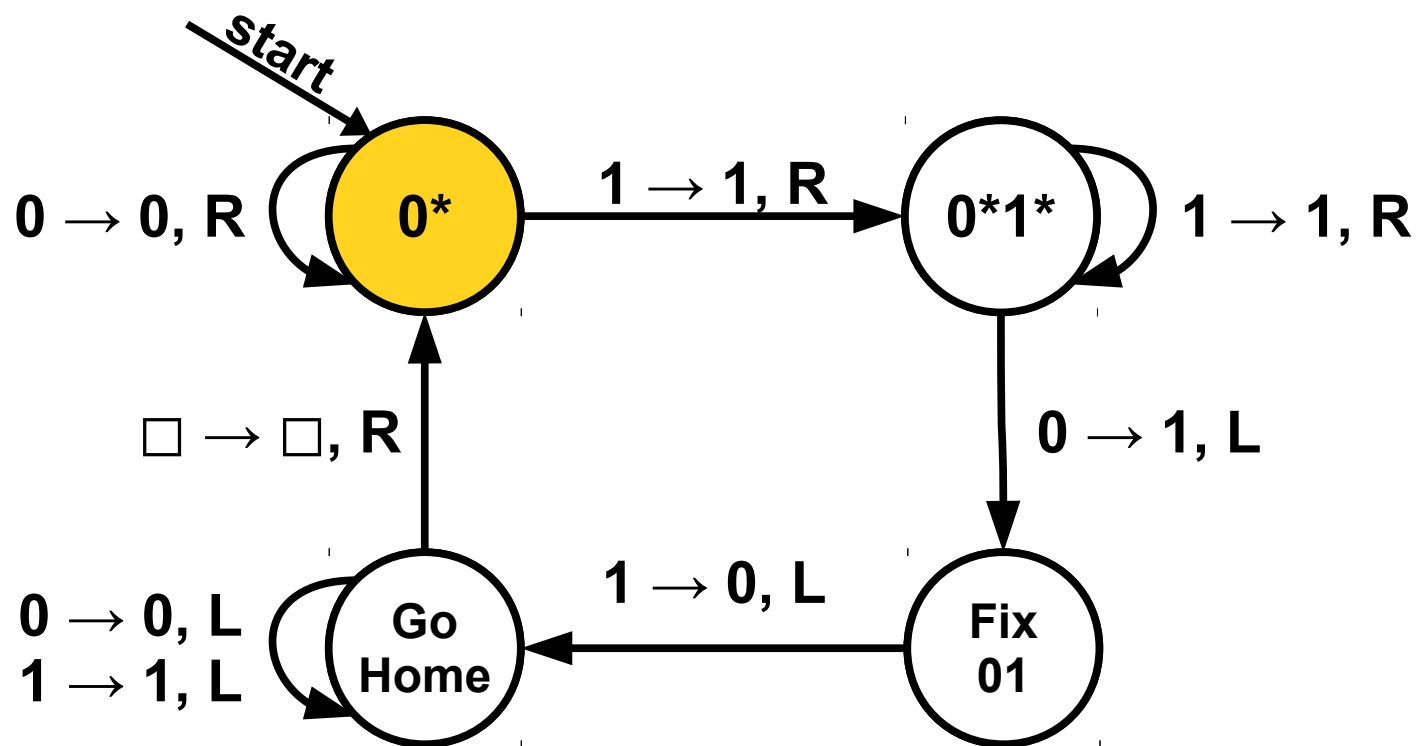


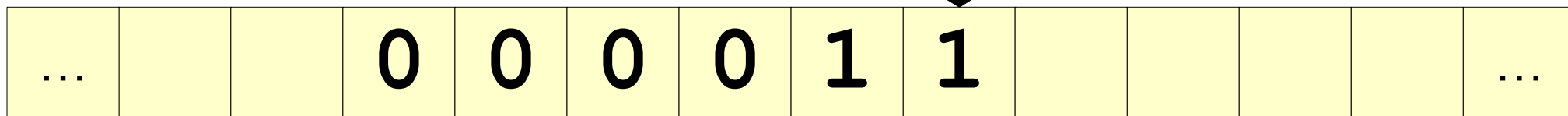
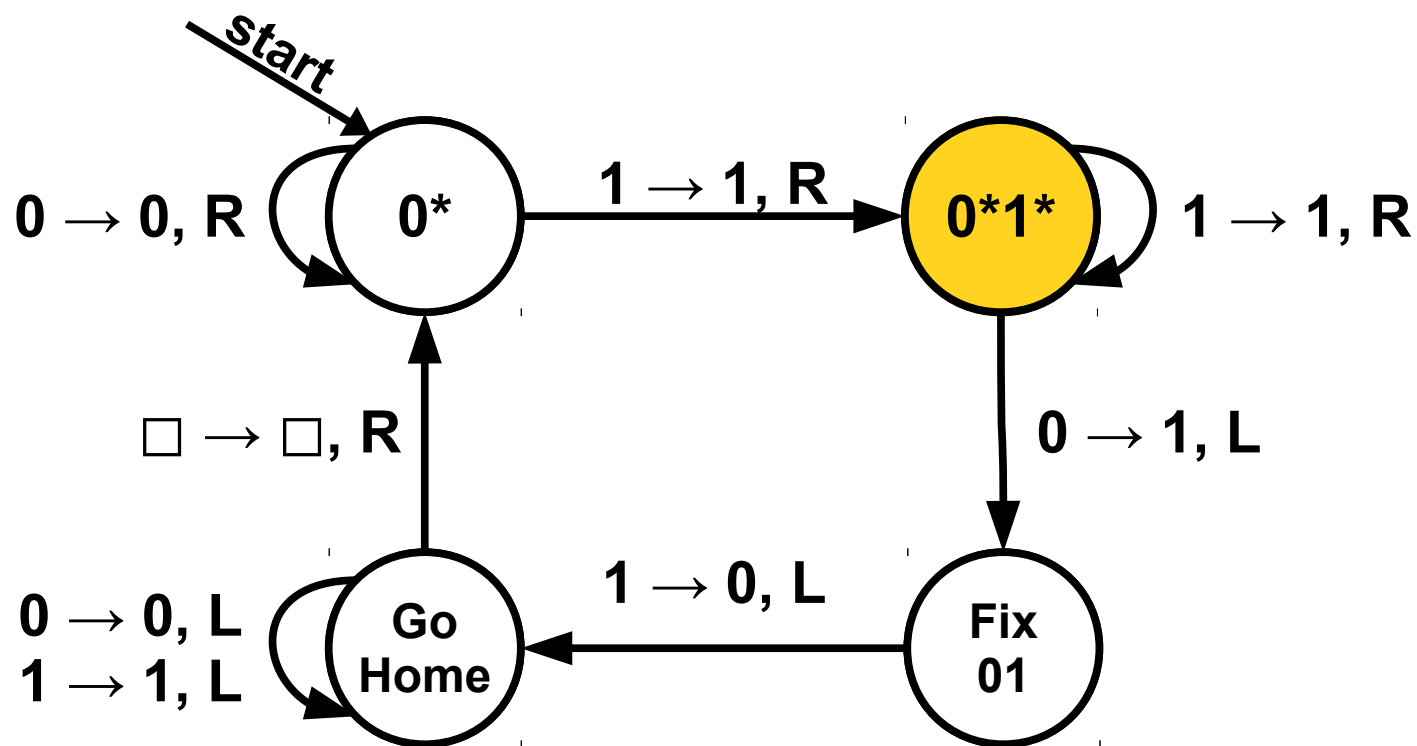


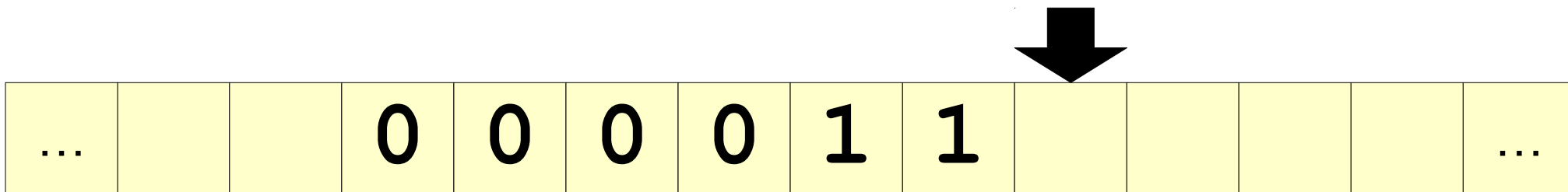
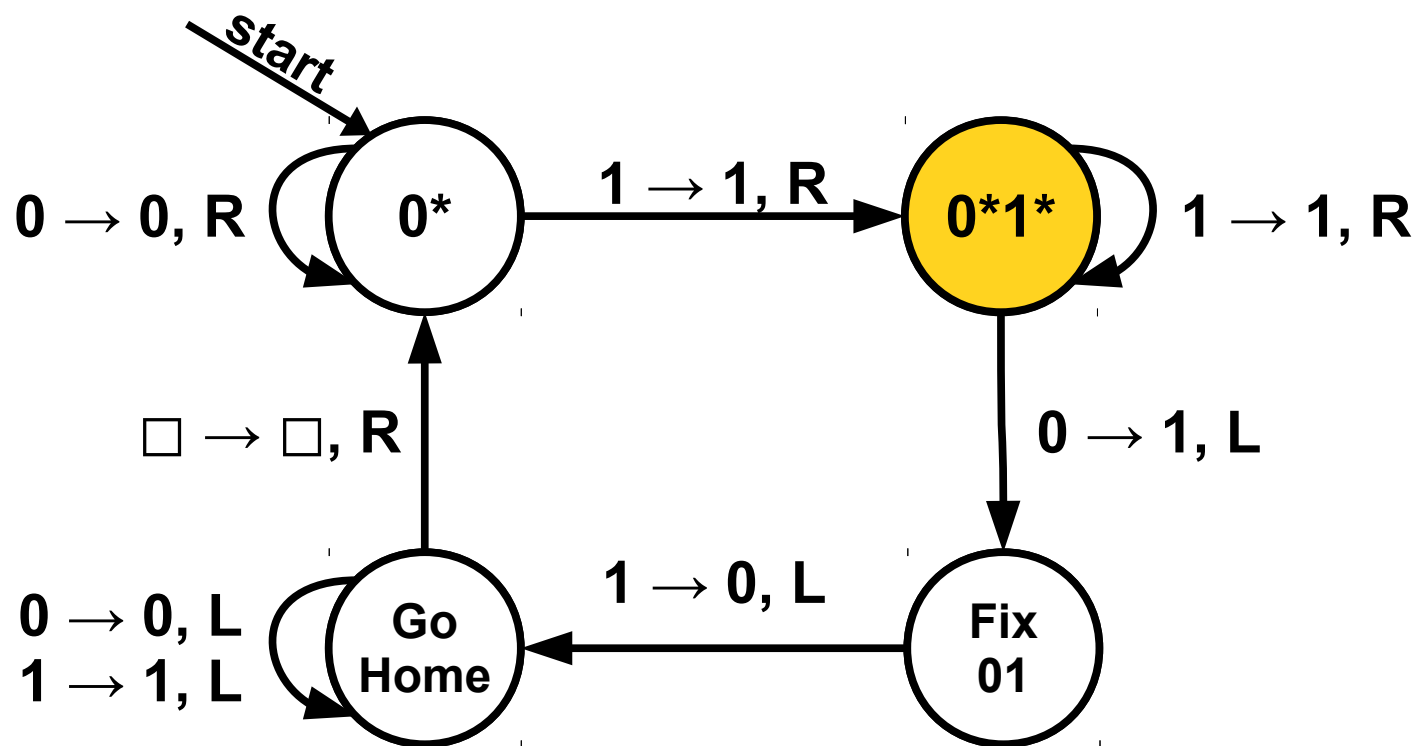




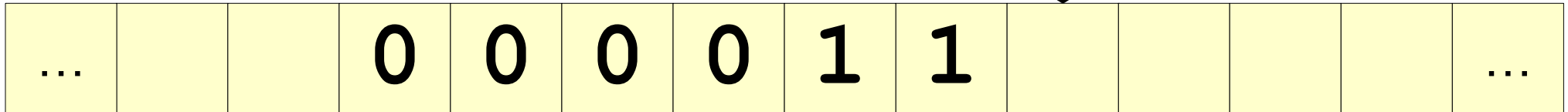
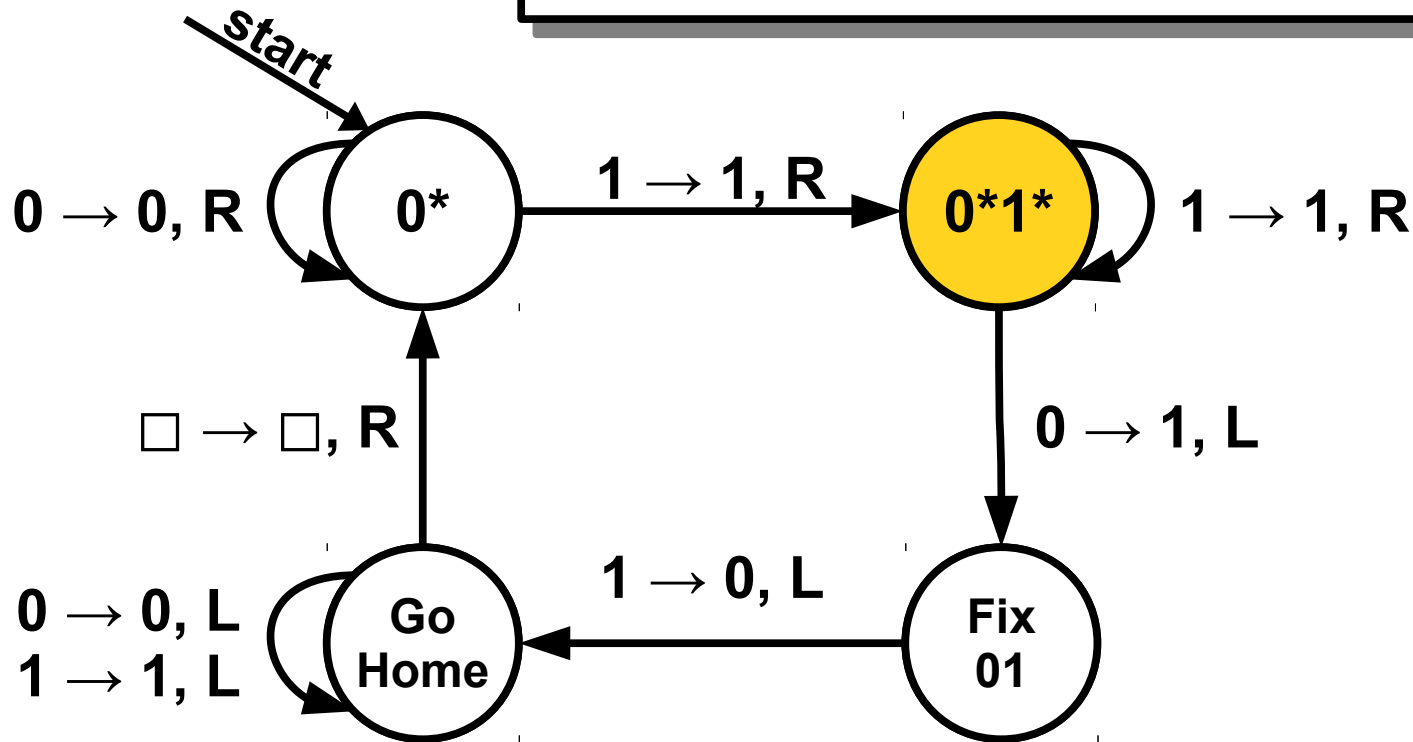


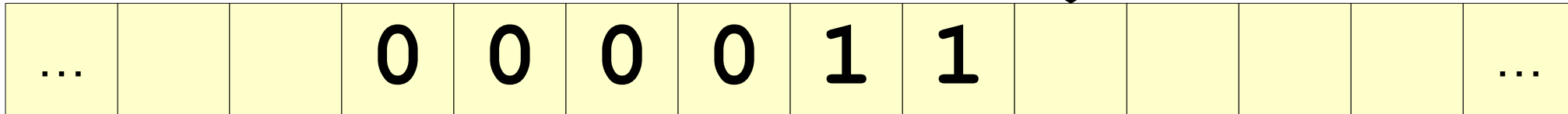
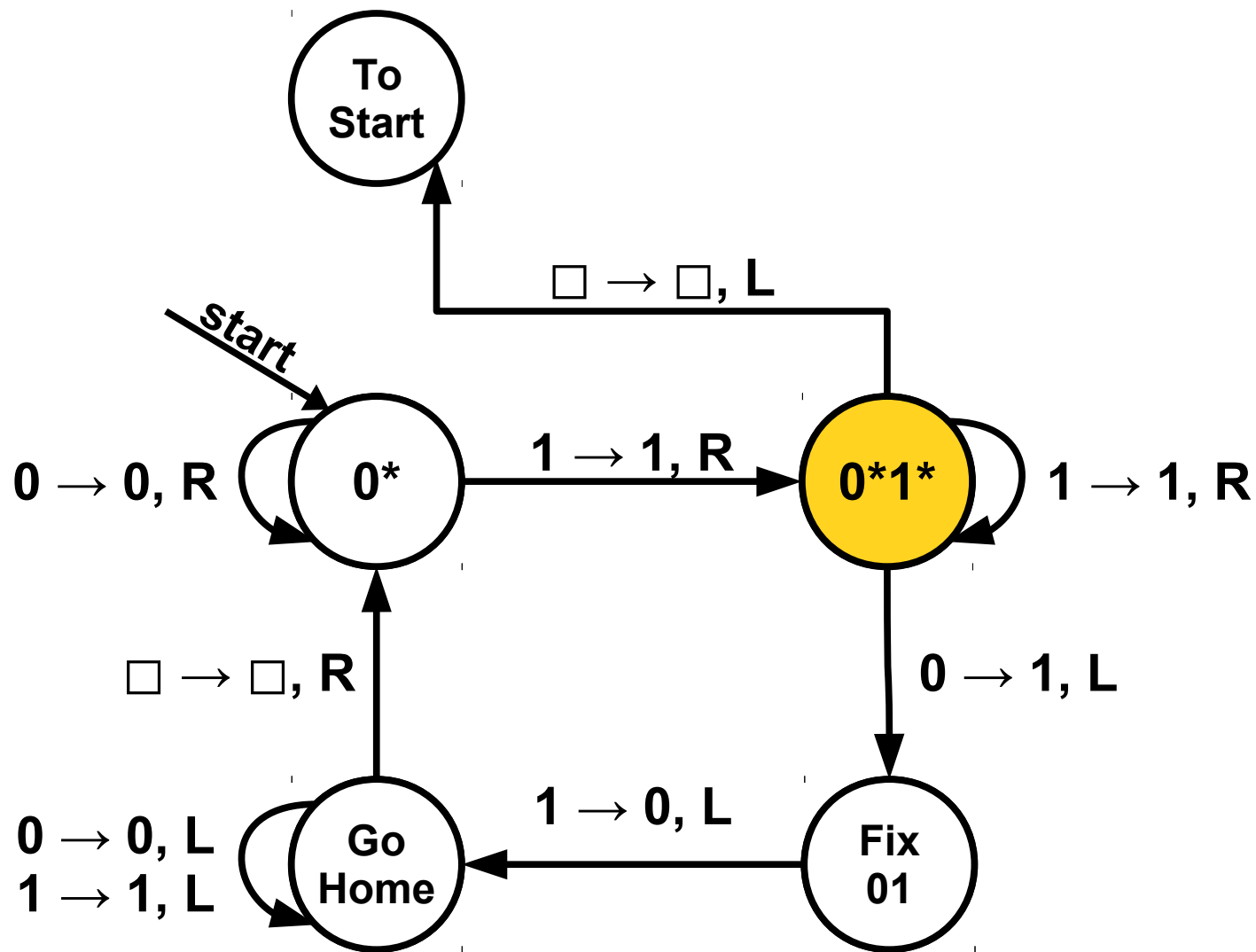


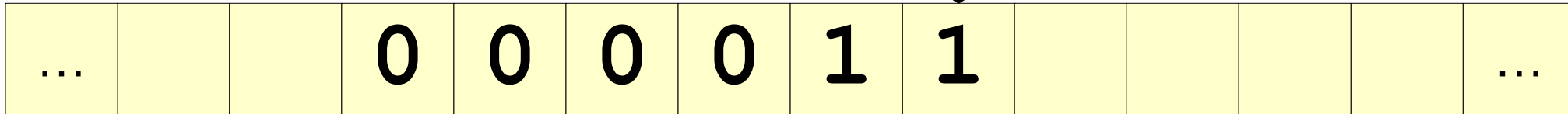
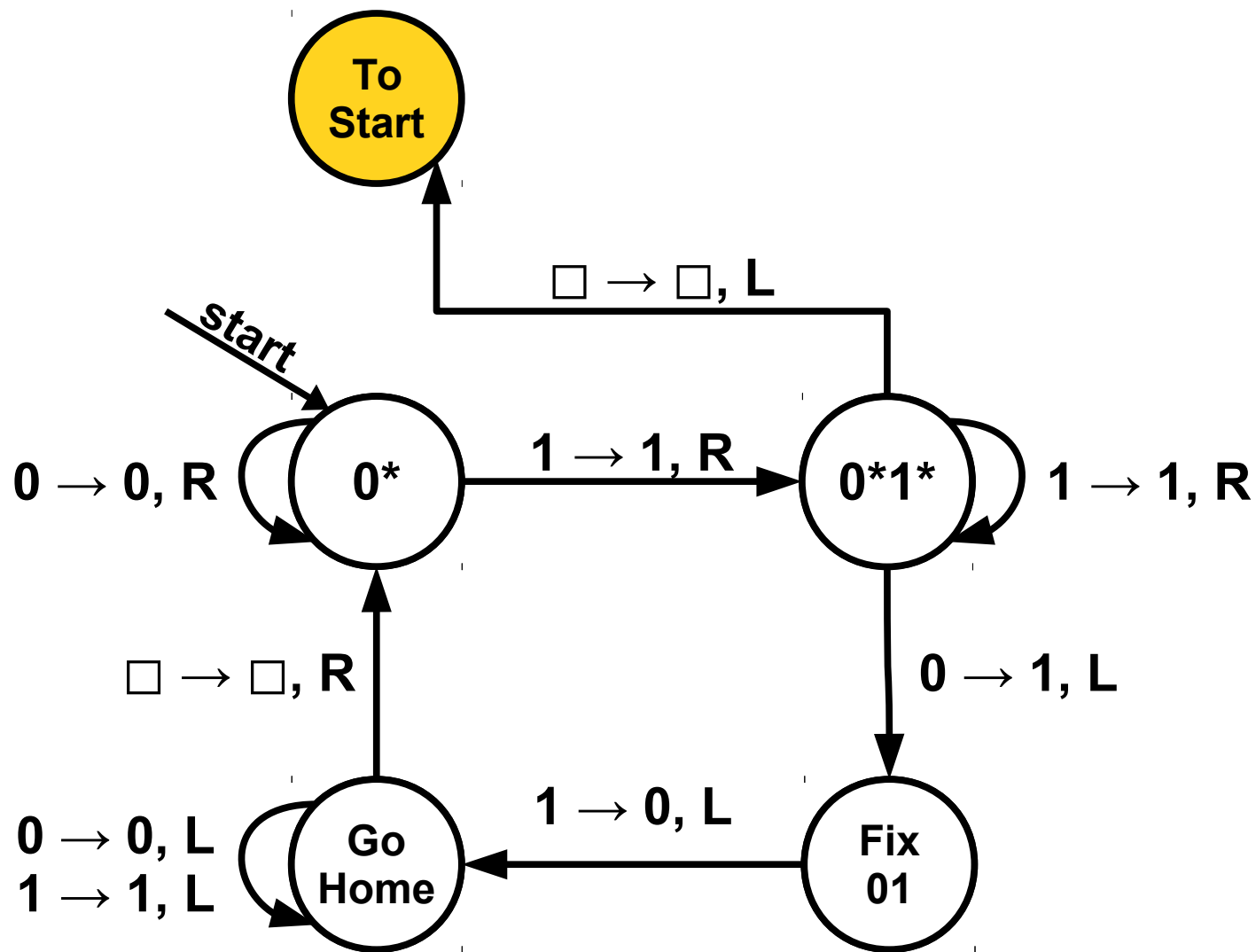


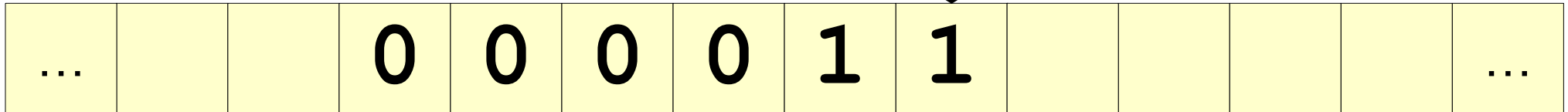
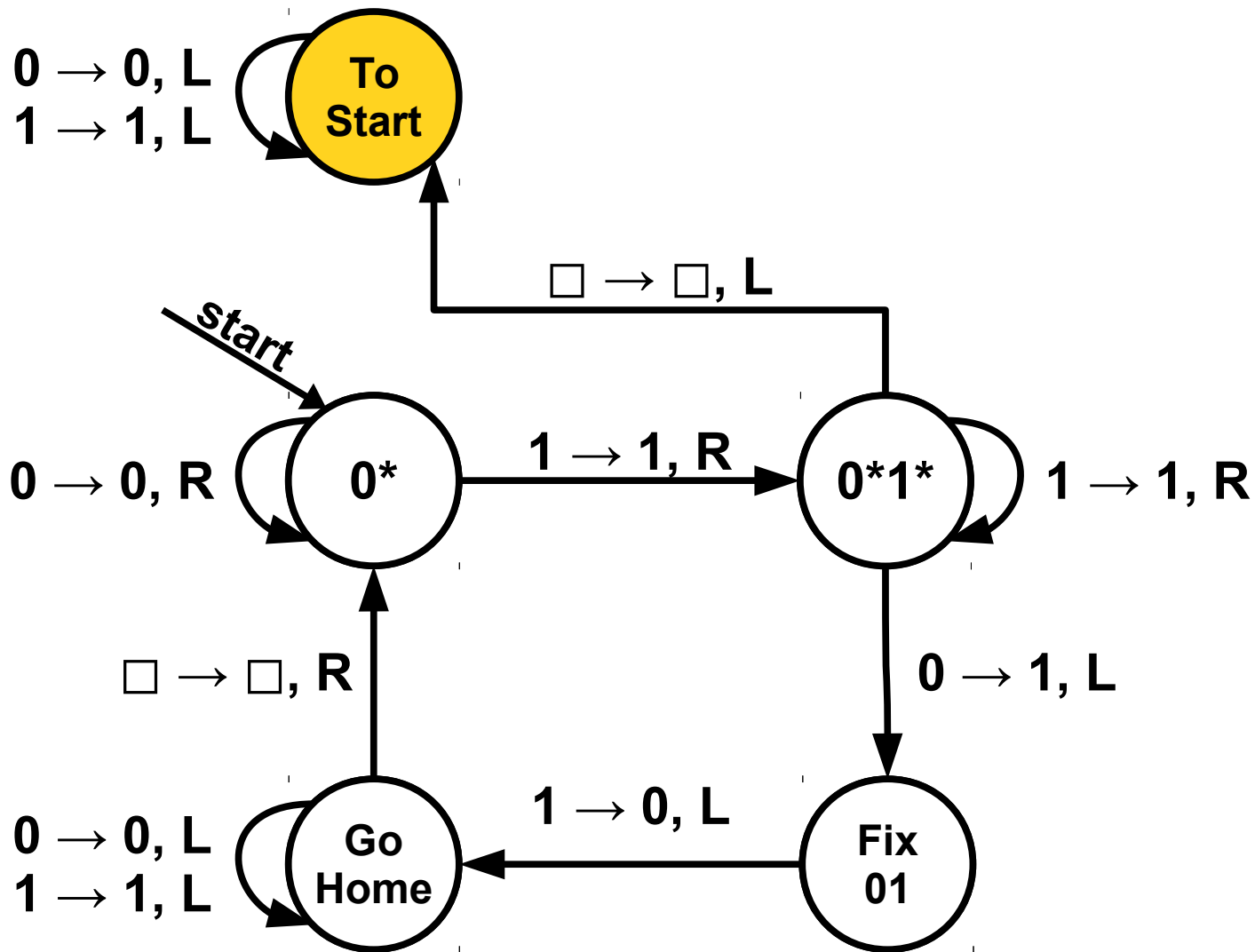


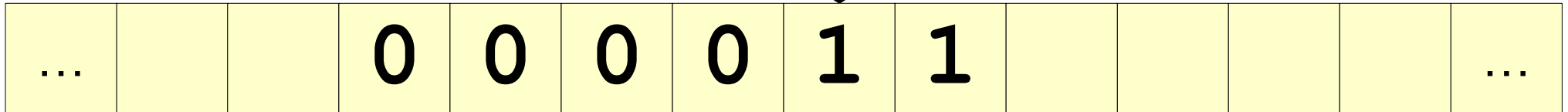
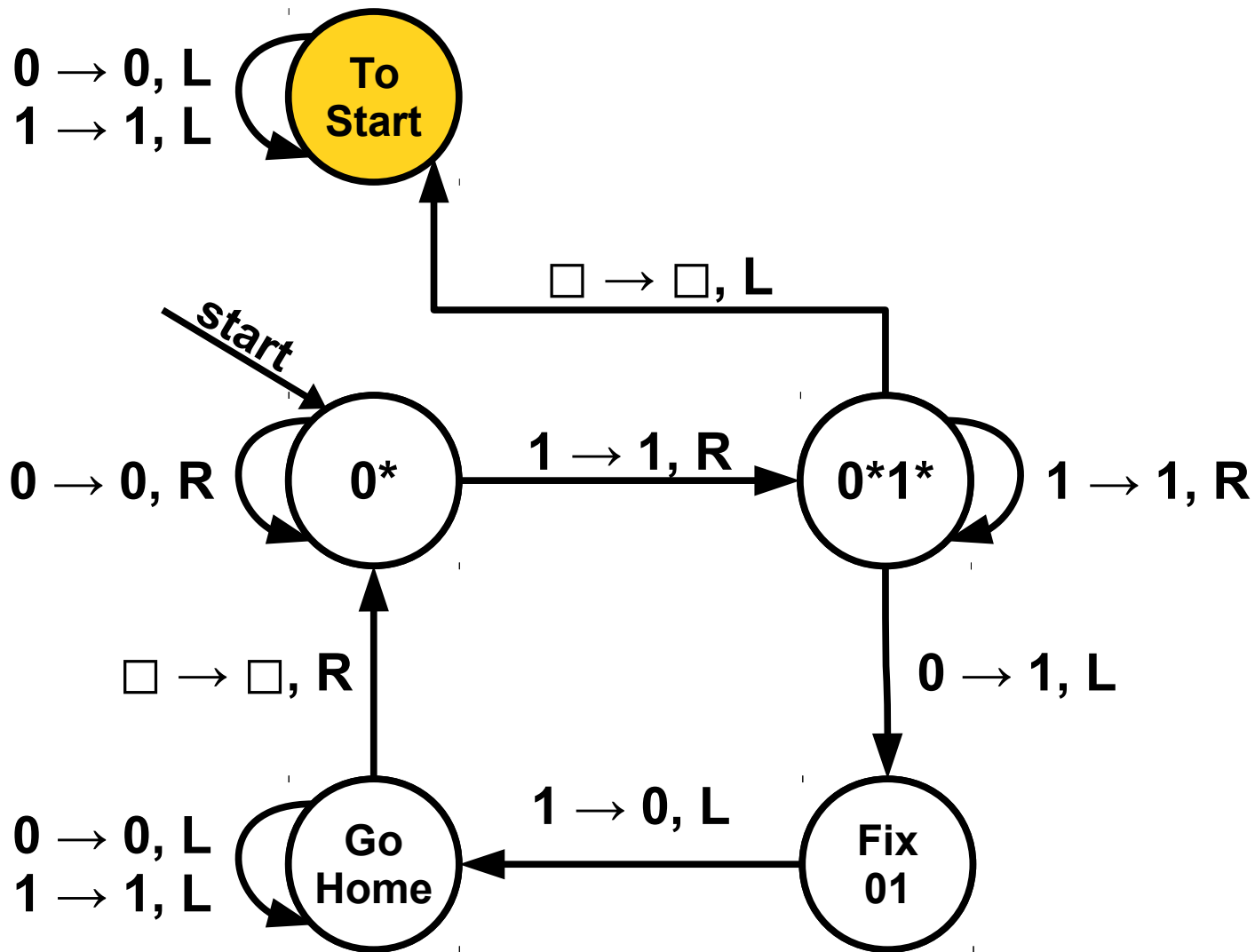
Our ultimate goal here was to sort everything so we could hand it off to the machine to check for $0^n 1^n$. Let's rewind the tape head back to the start.

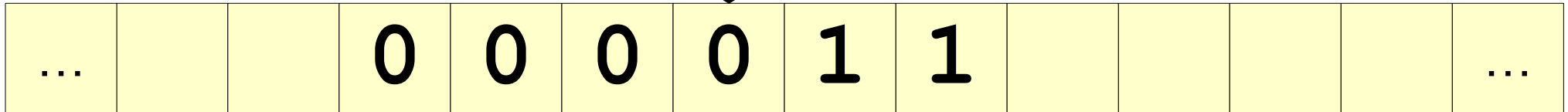
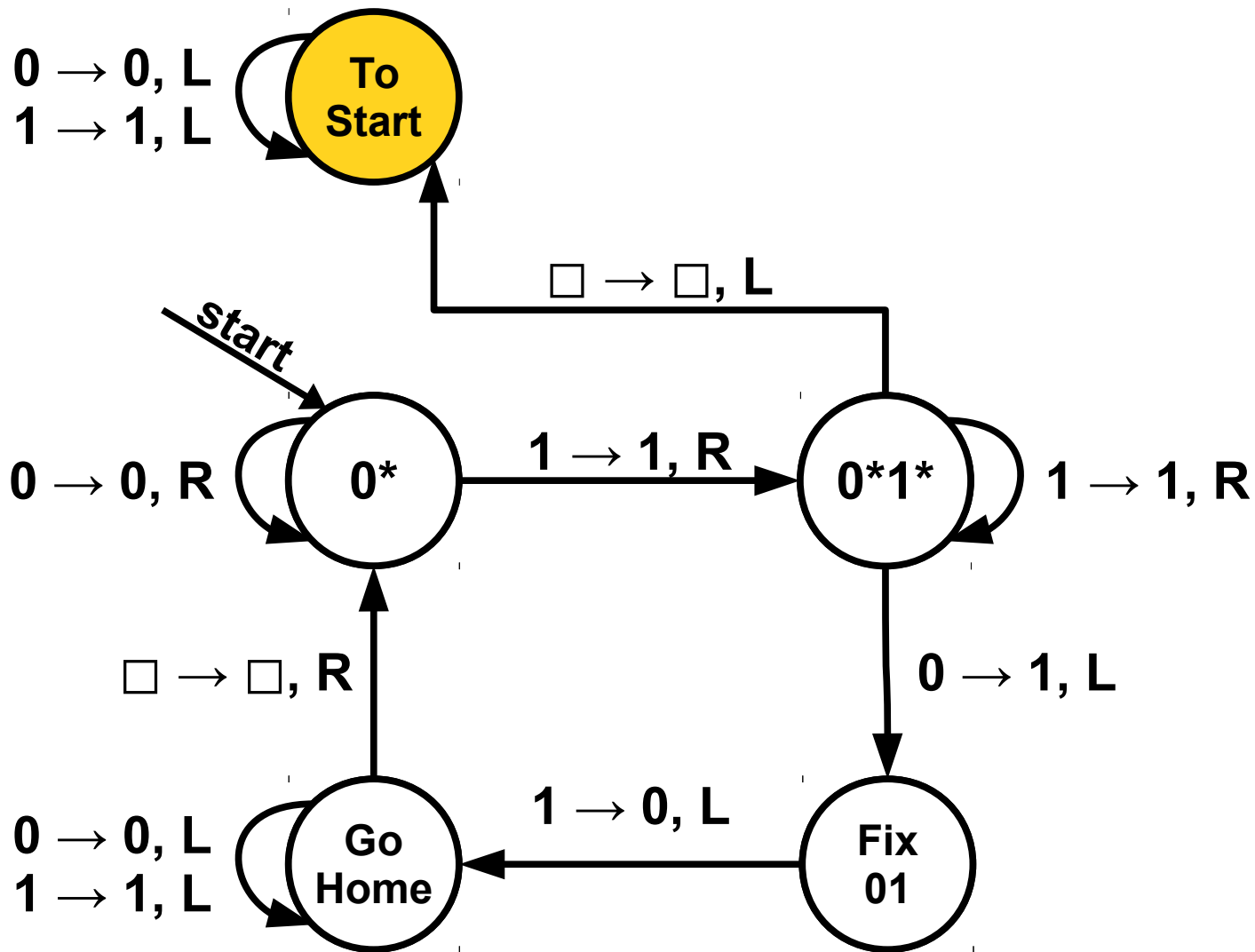


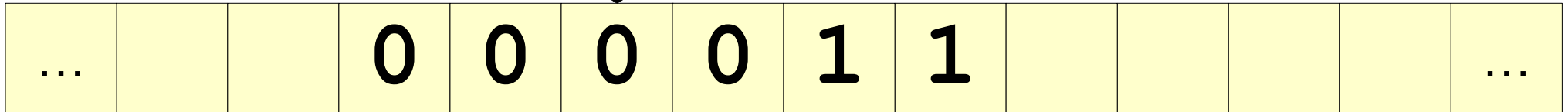
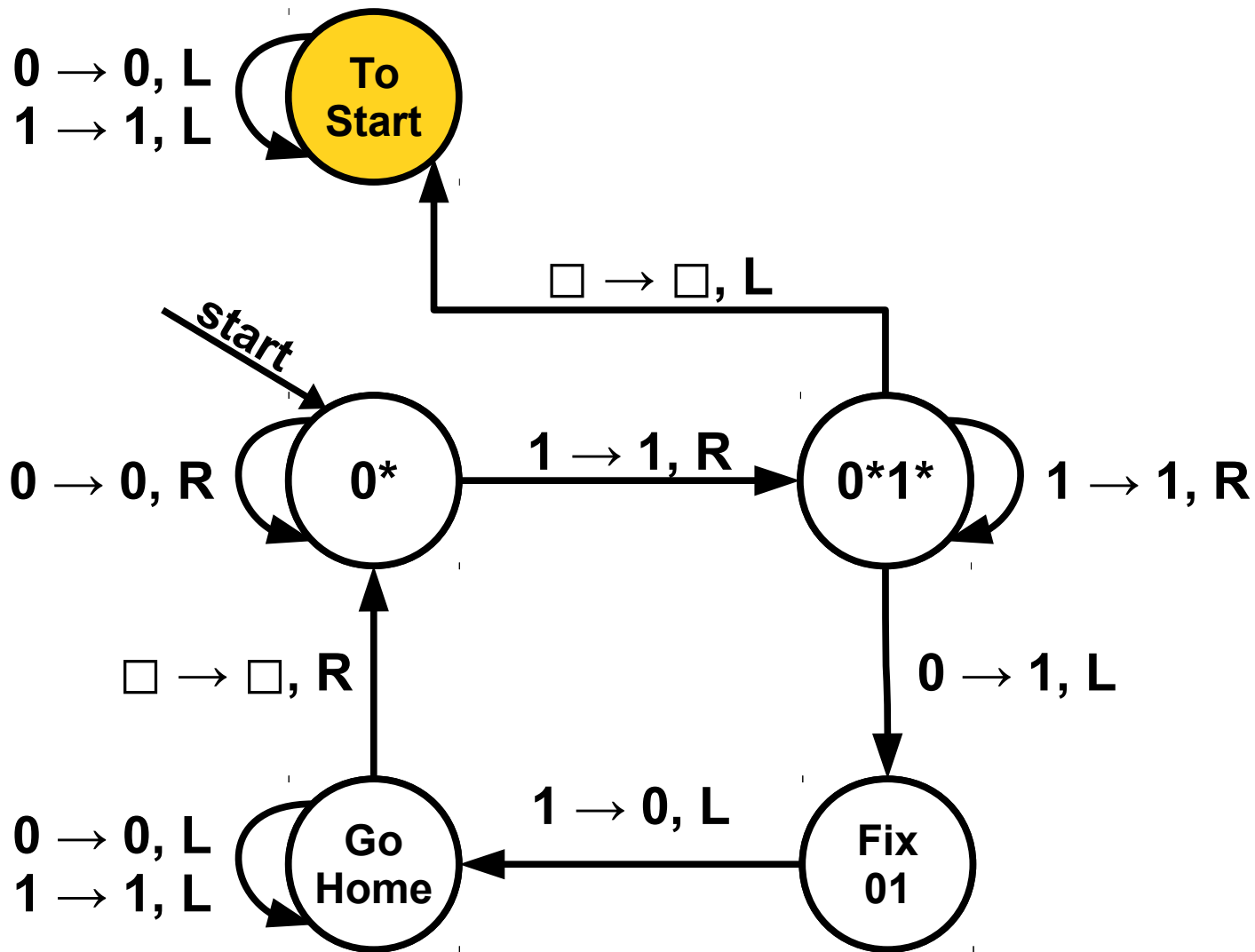


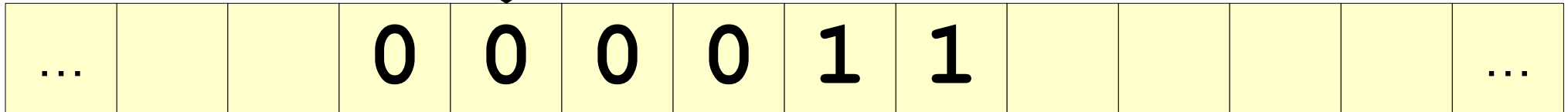
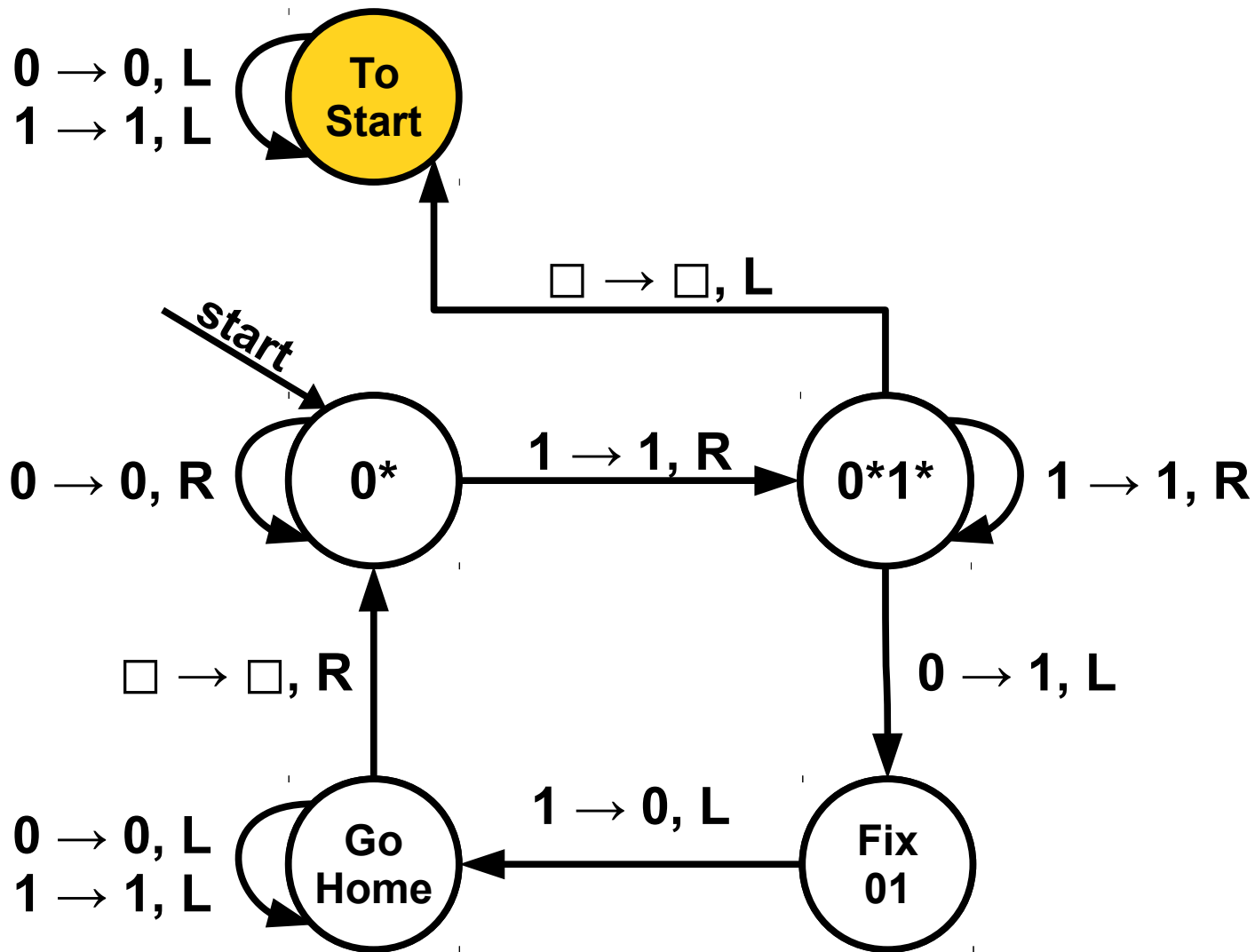


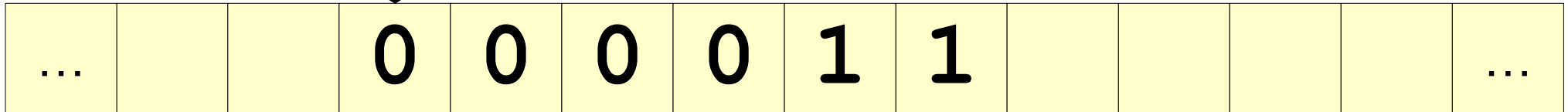
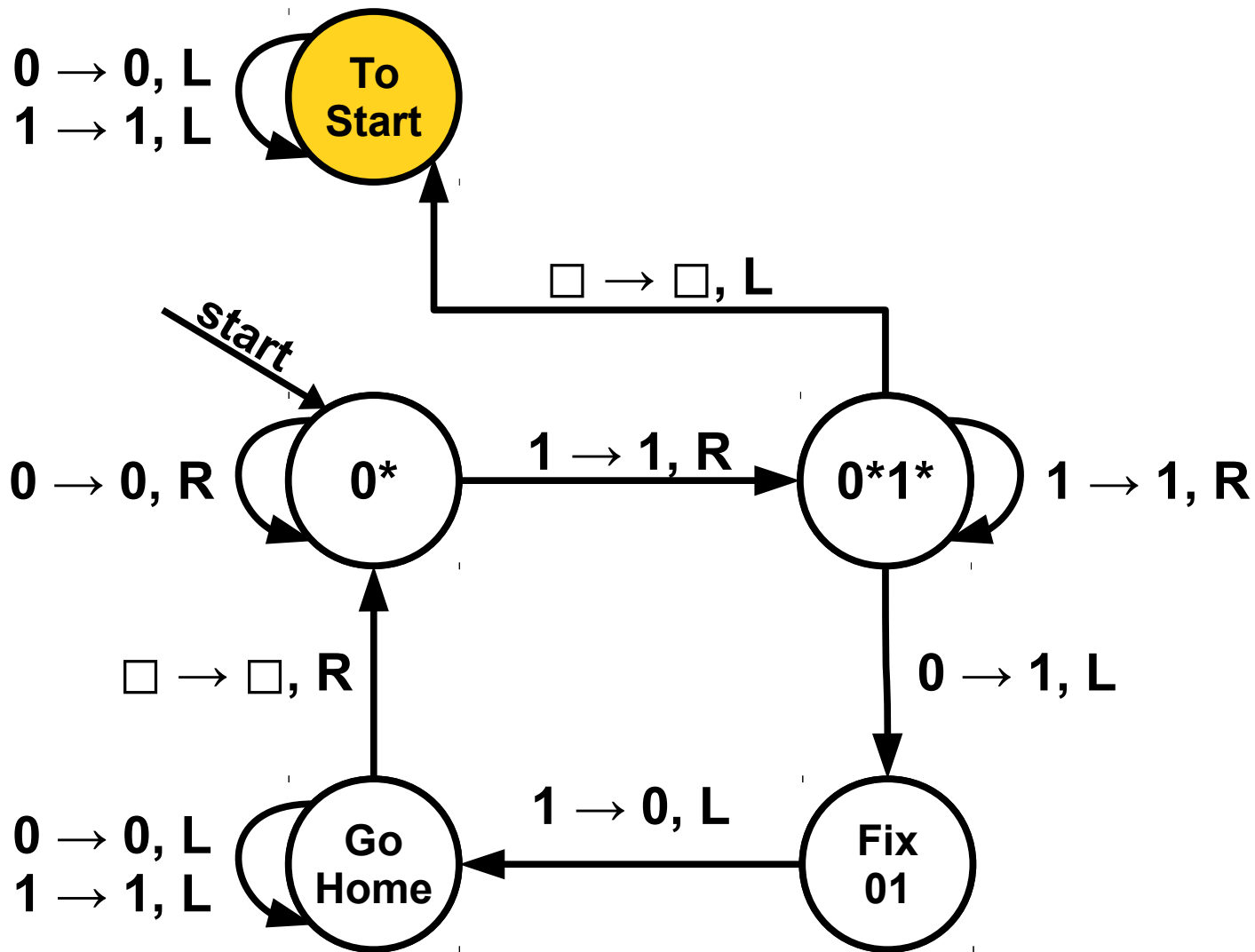


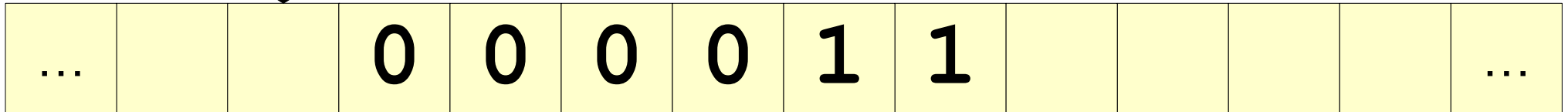
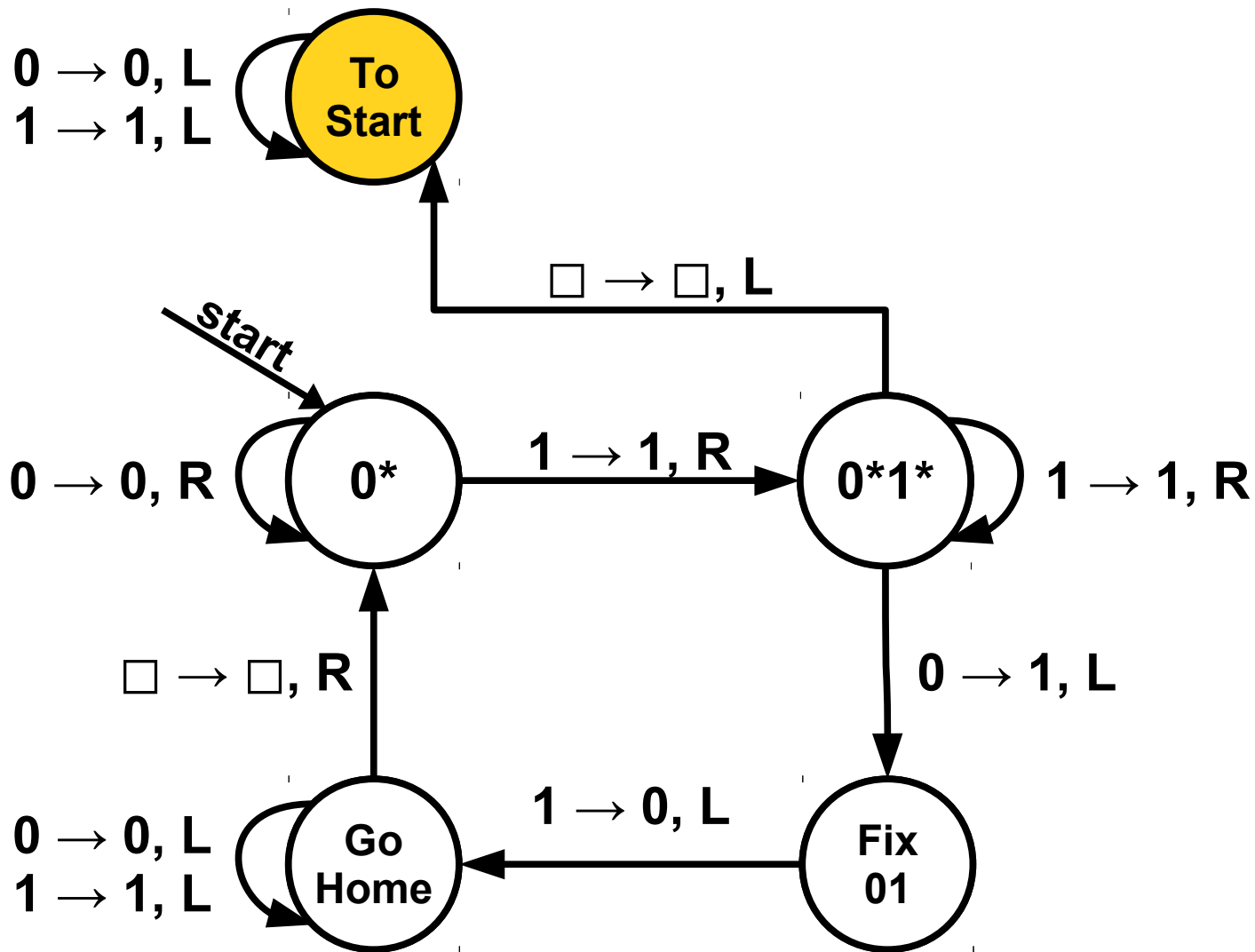


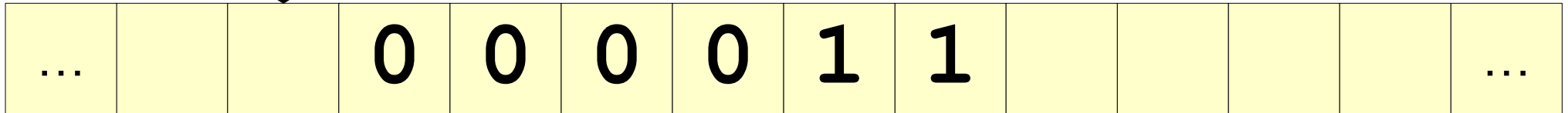
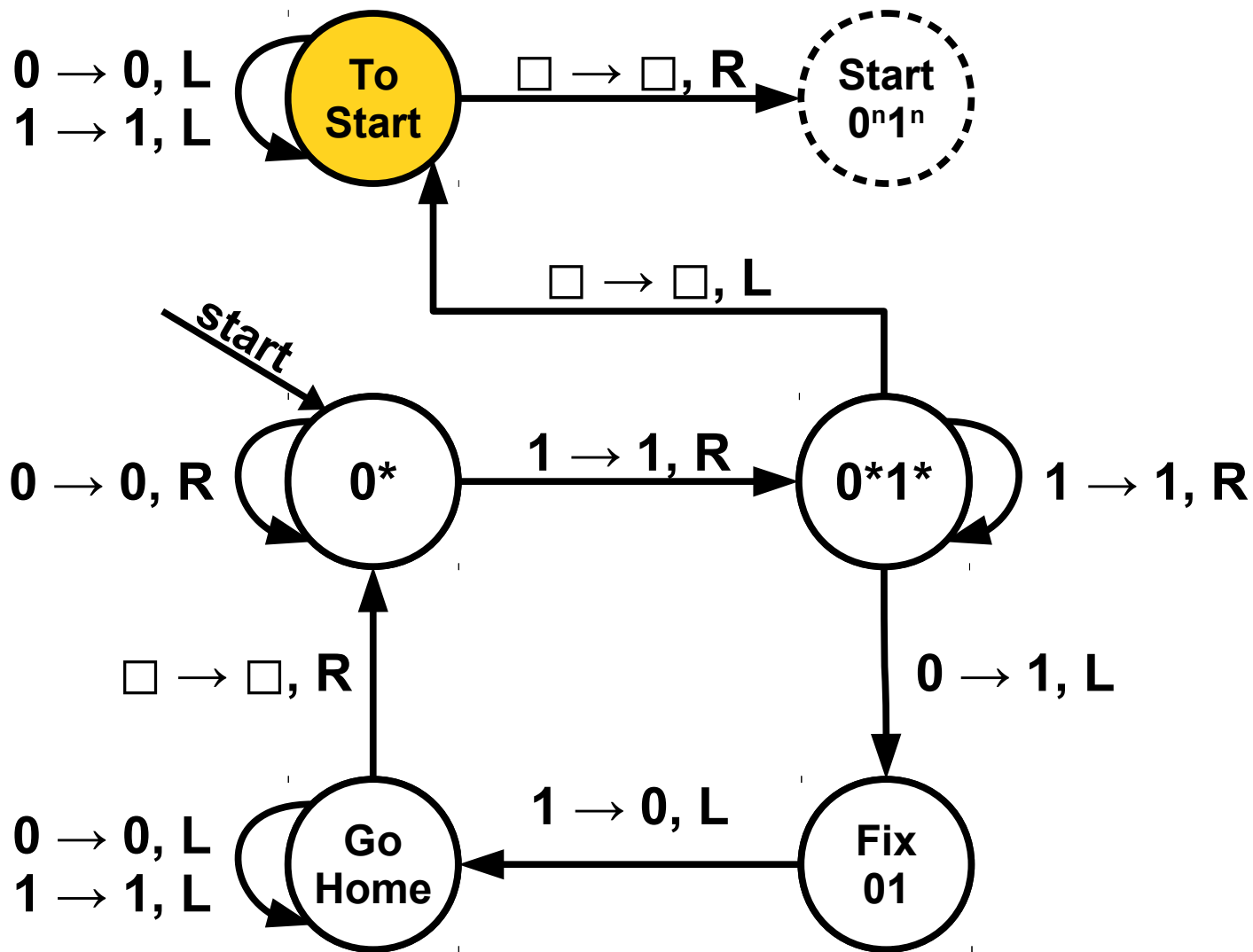


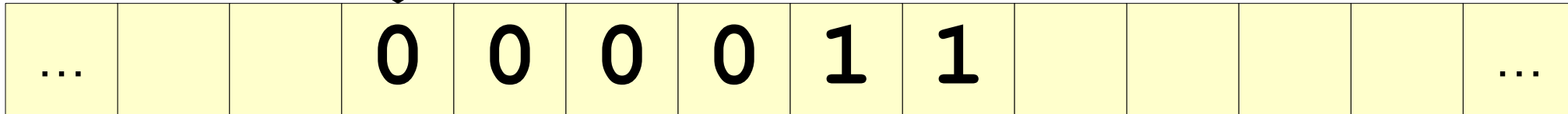
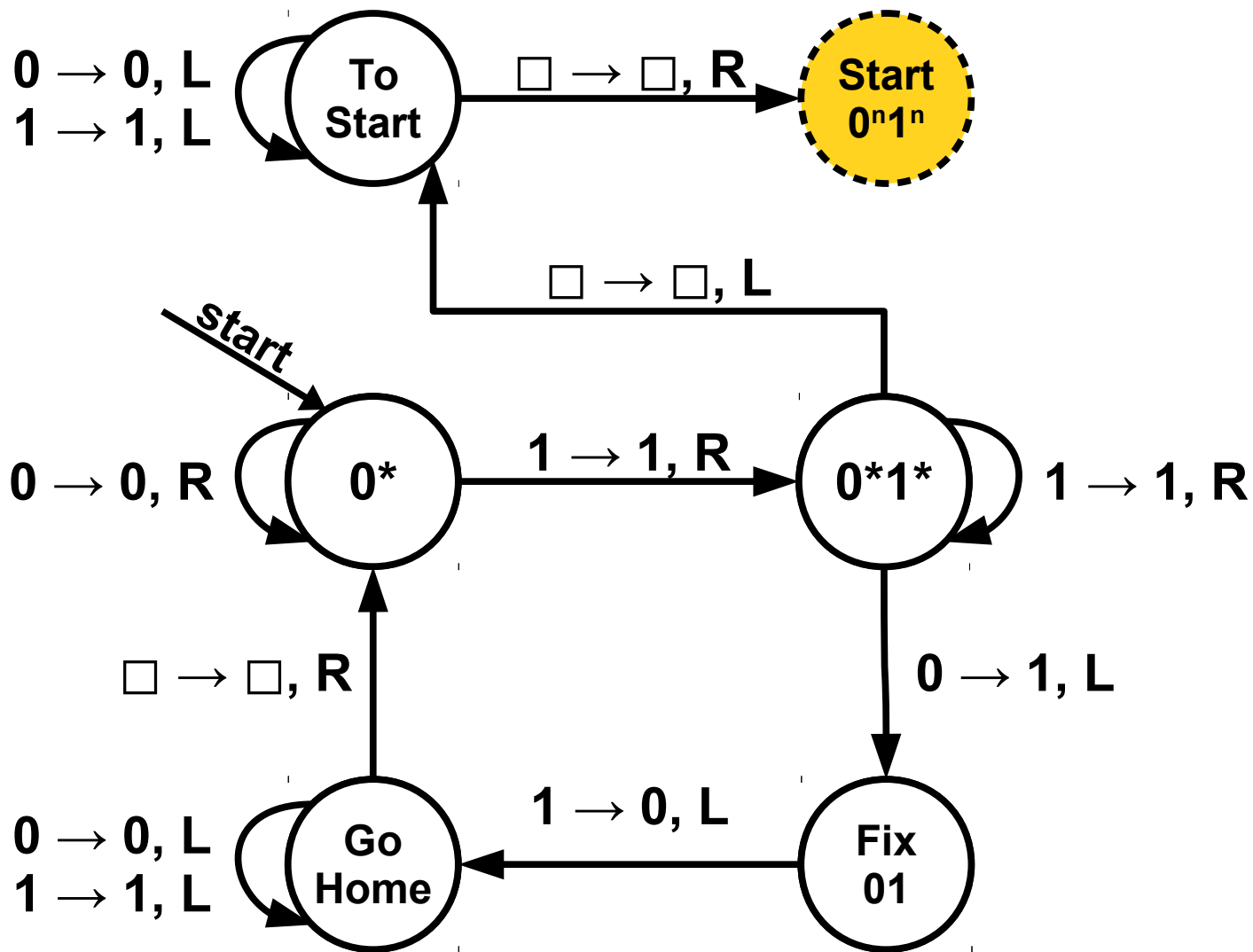


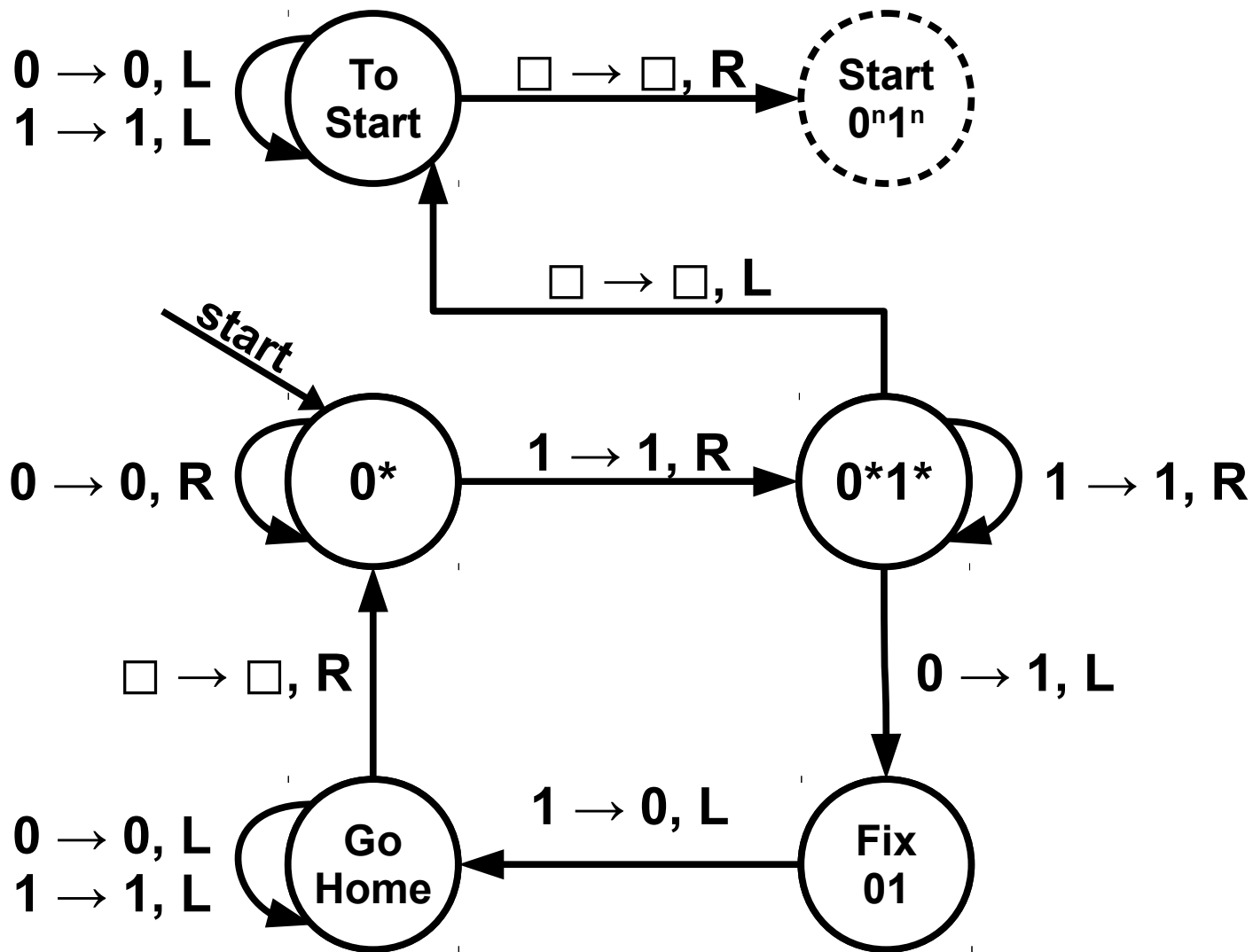


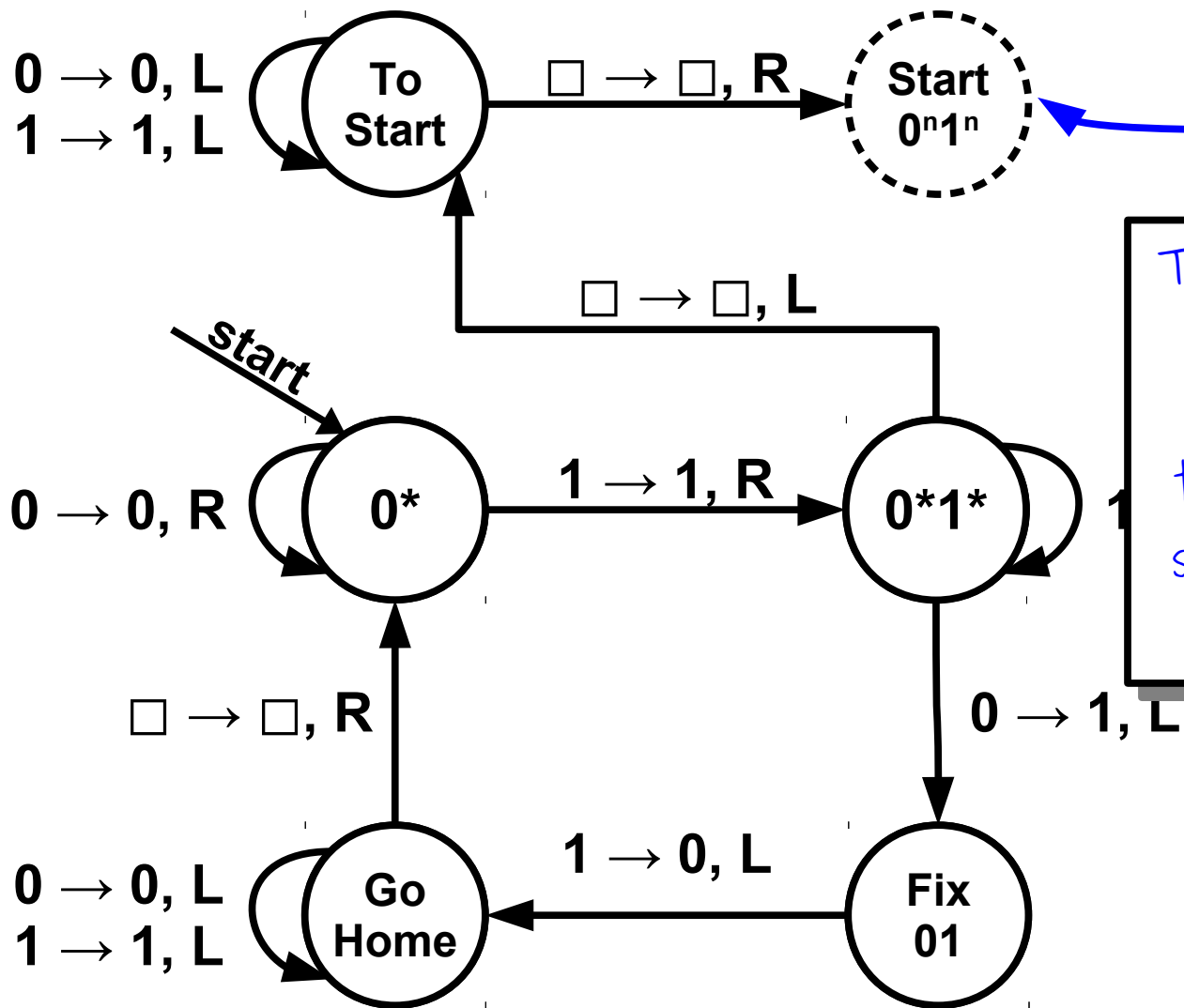




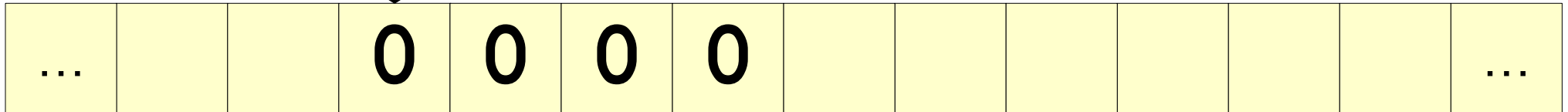
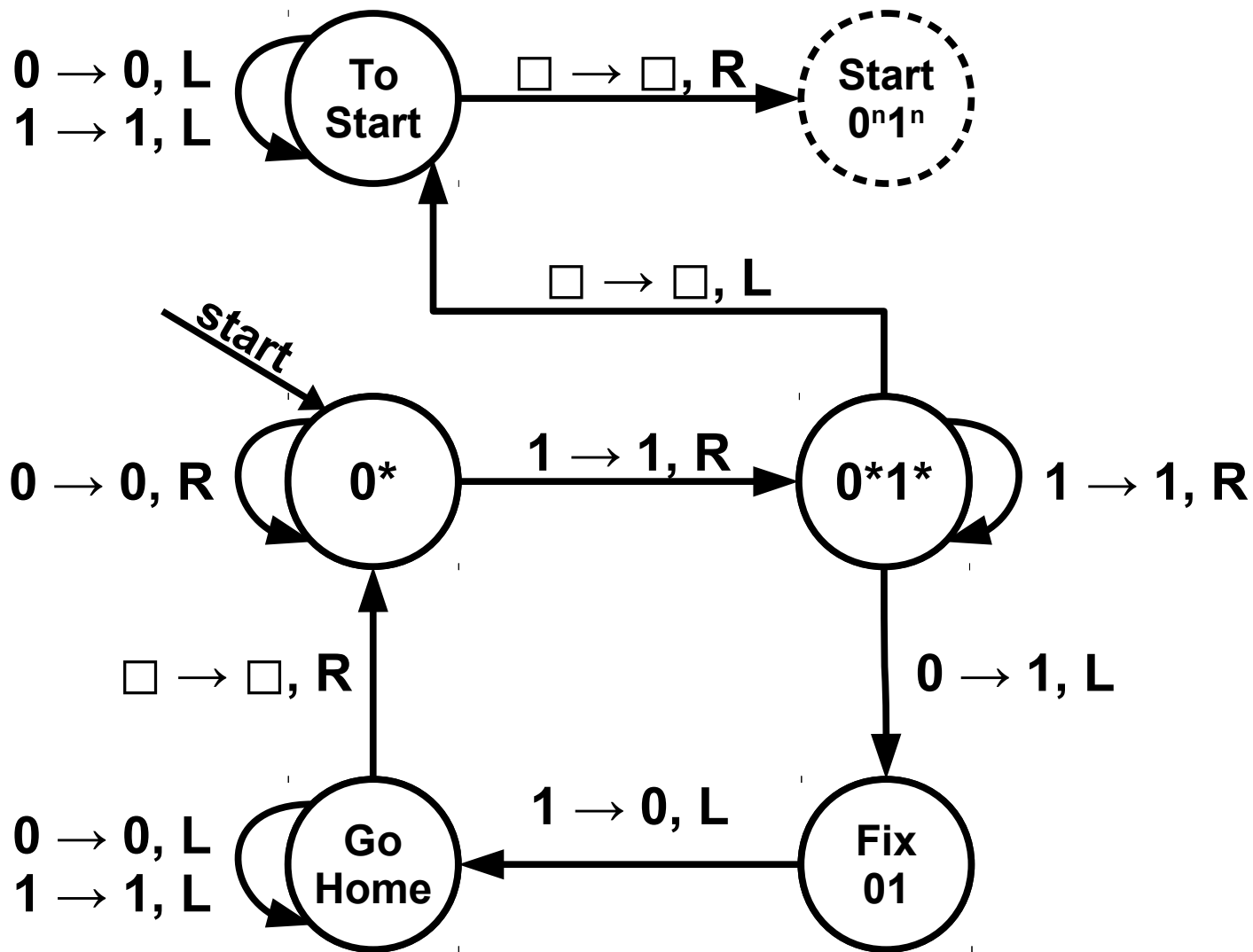


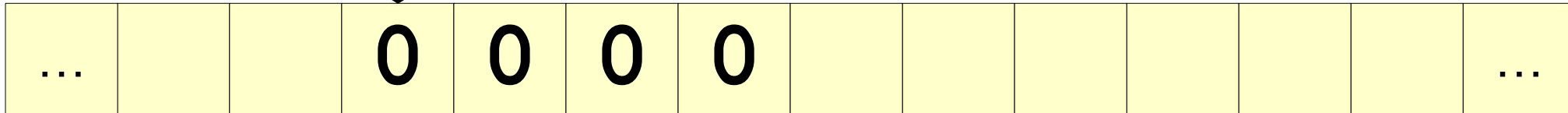
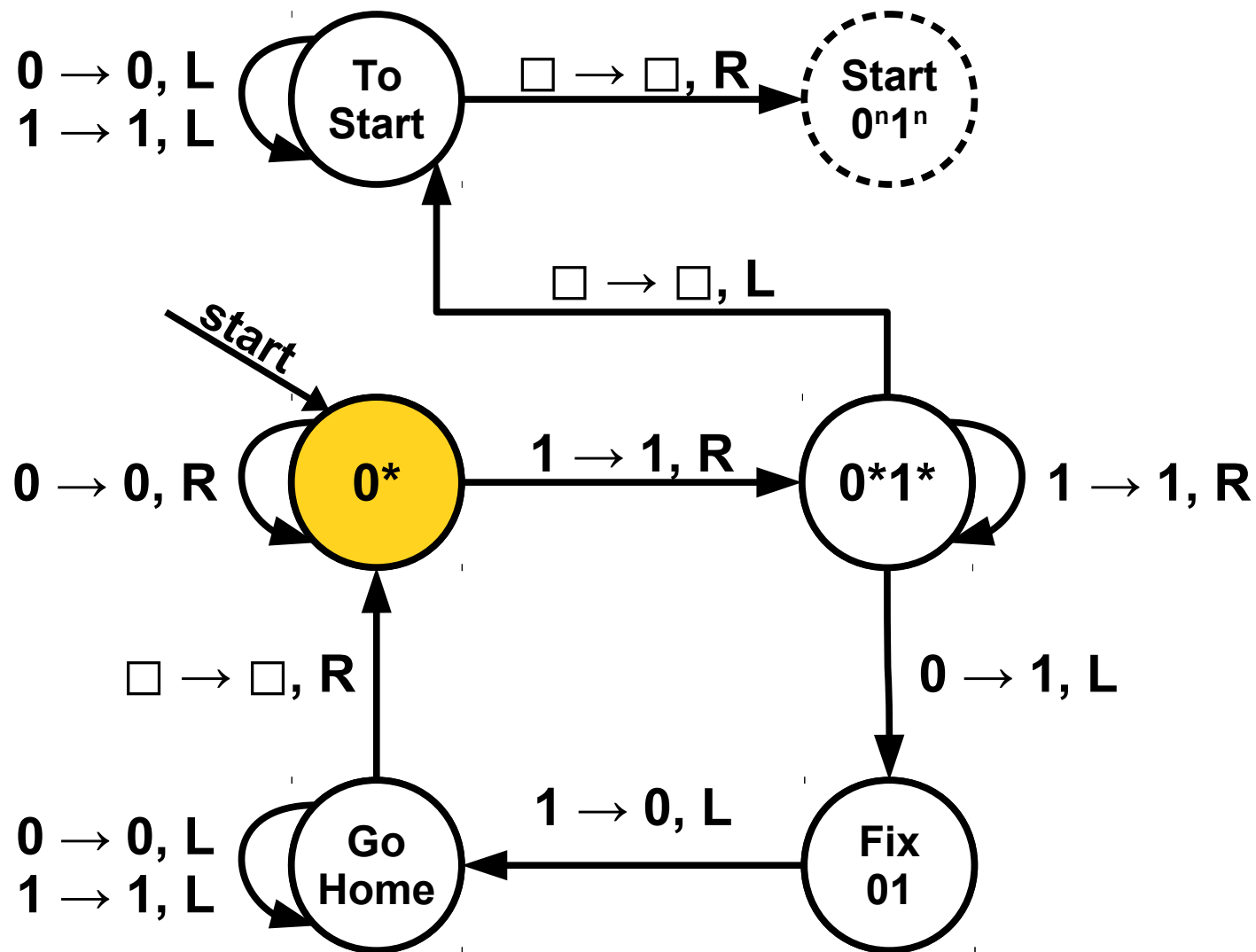


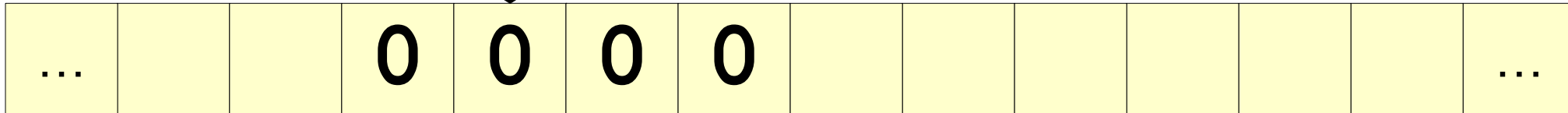
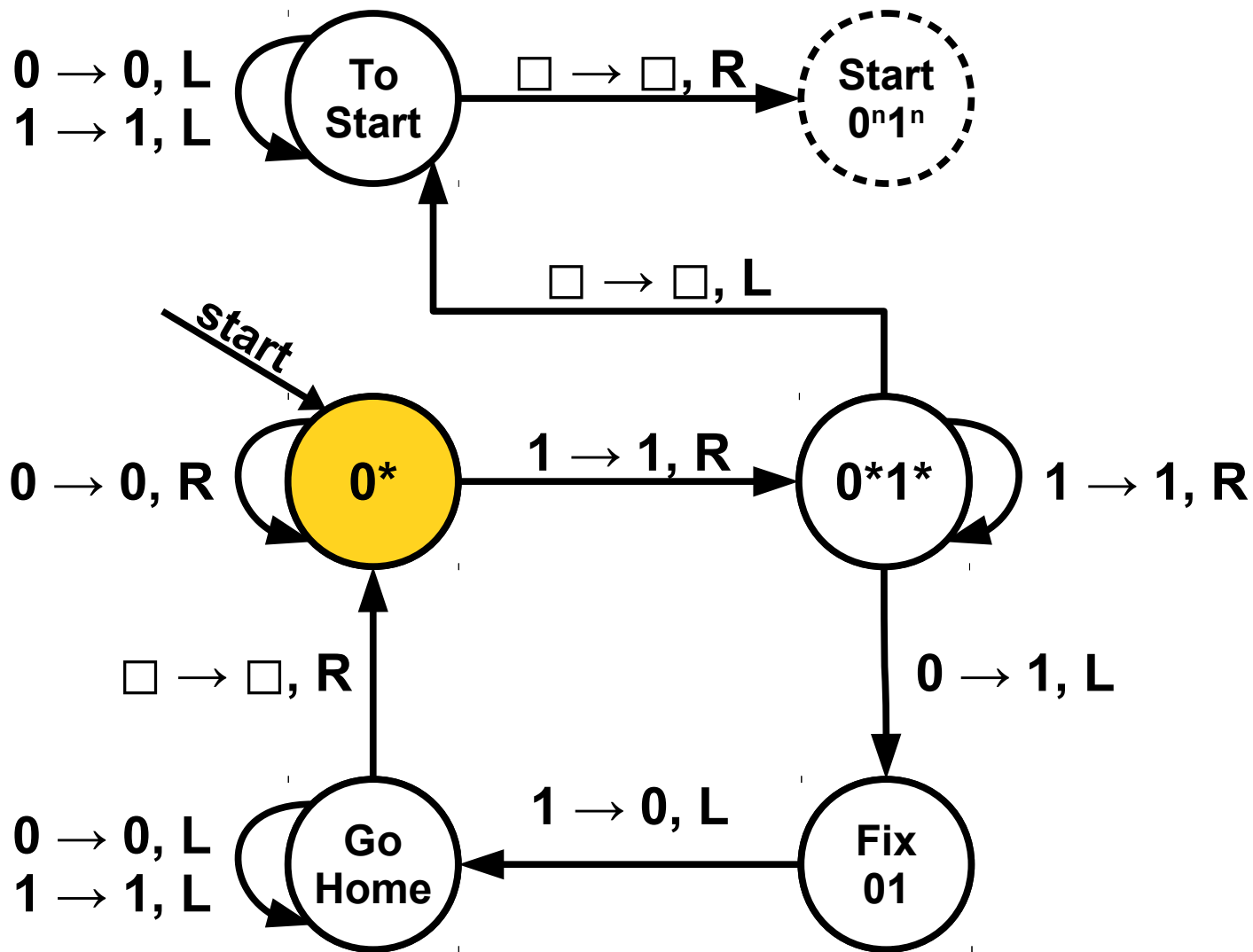


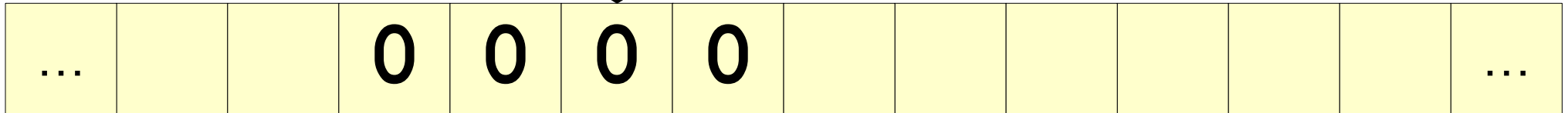
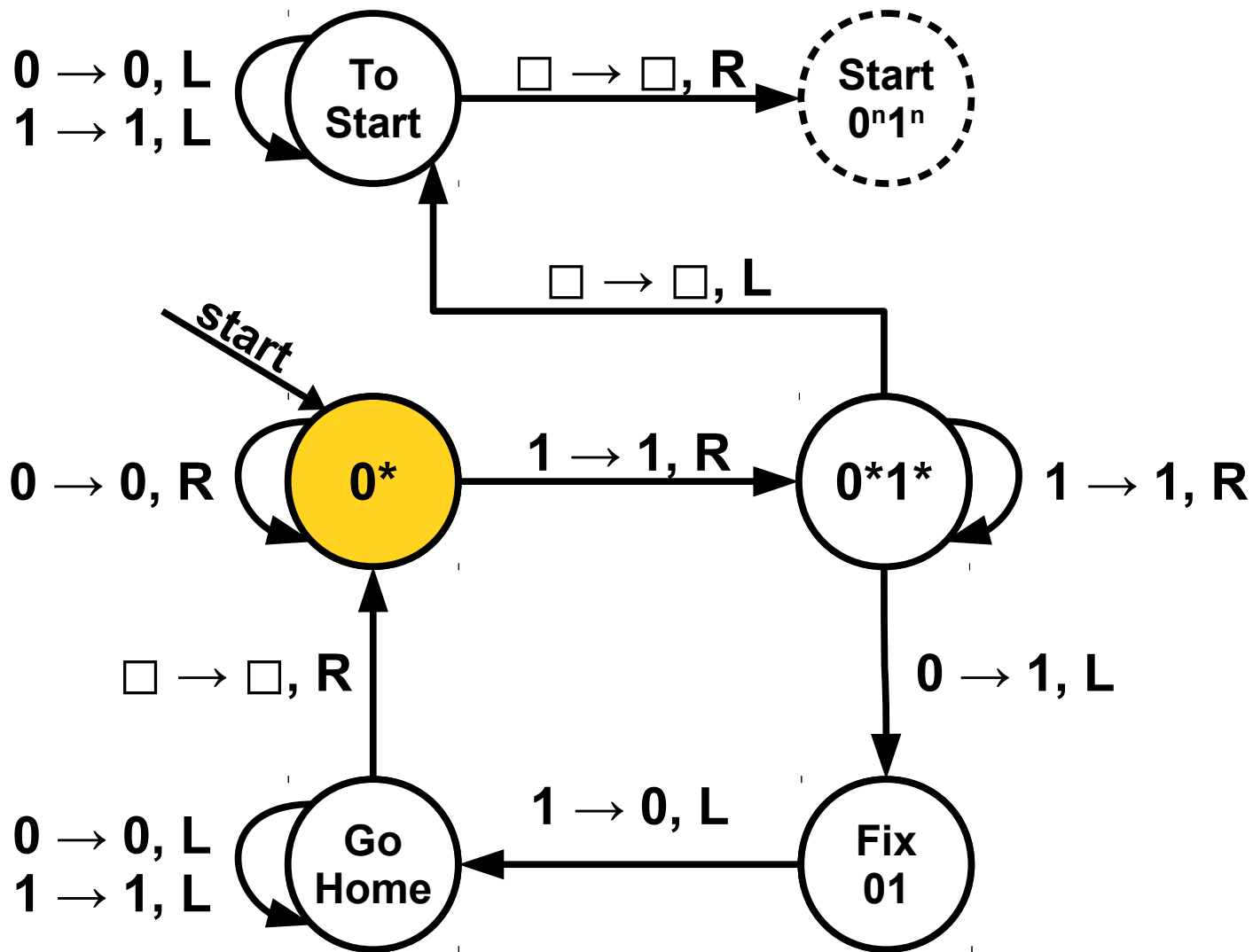


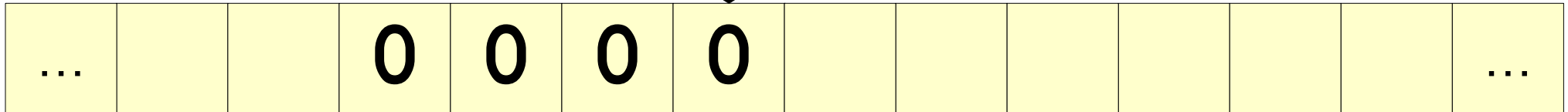
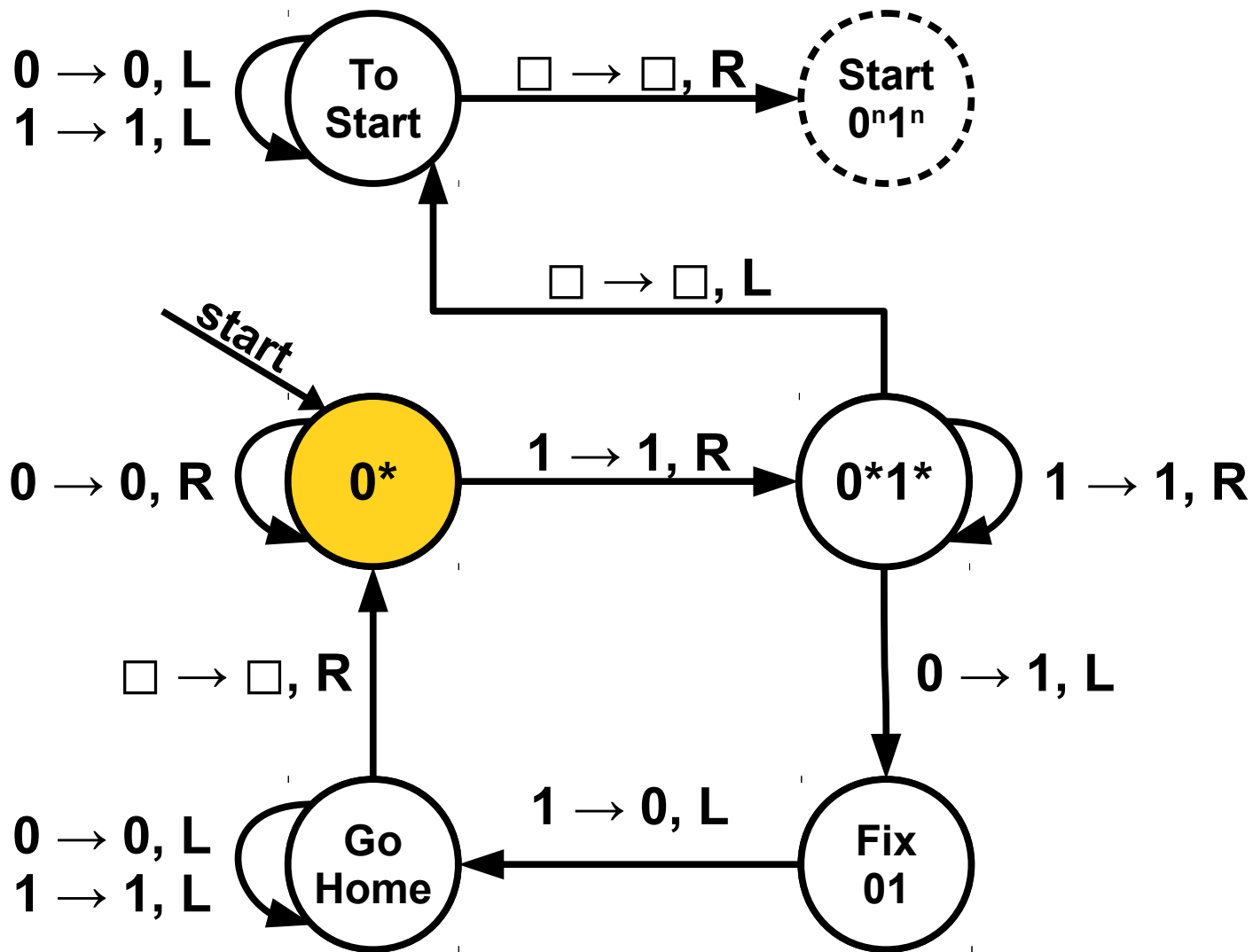
This is just a placeholder. Imagine snapping in the entire TM for $0^n 1^n$ into this diagram, putting the start state in the dashed area.

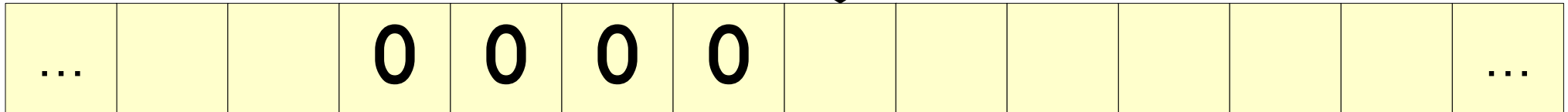
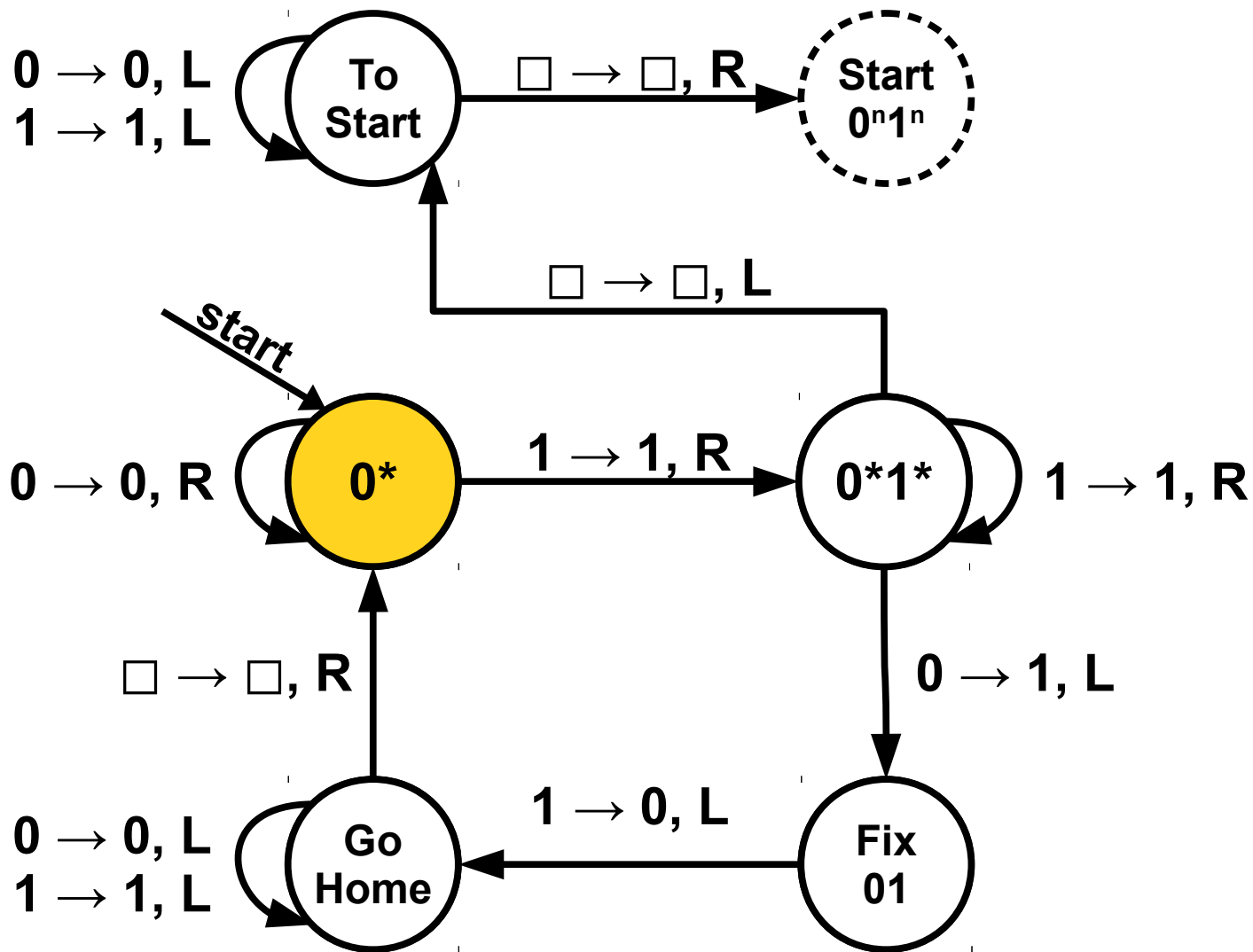


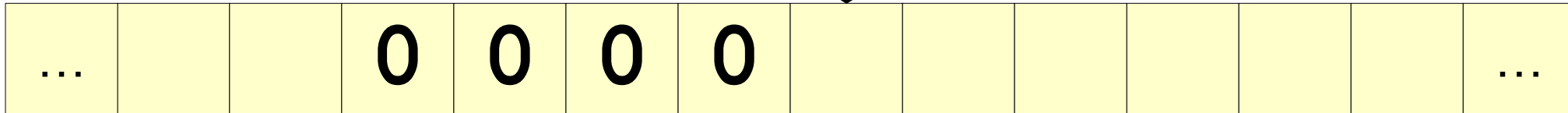
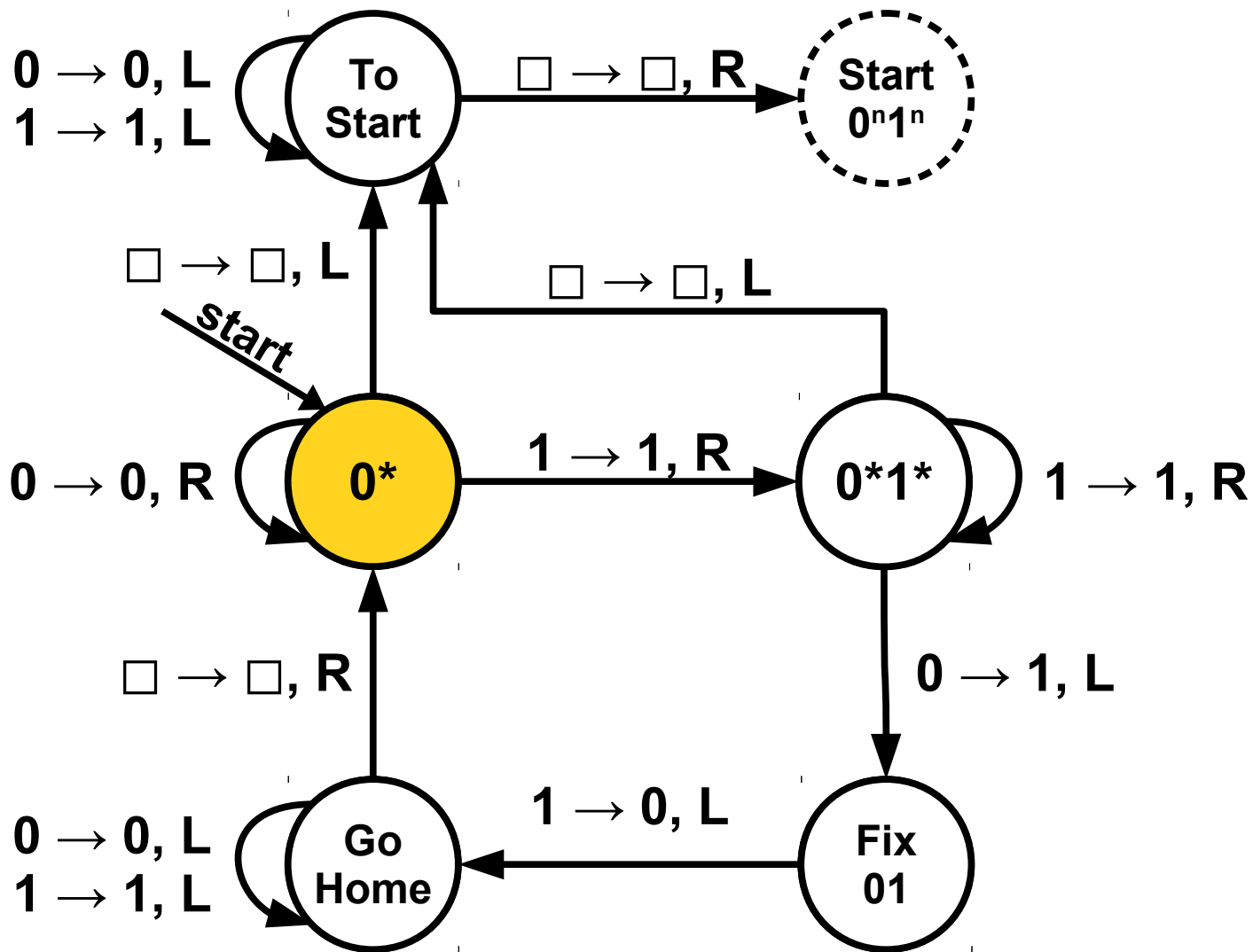


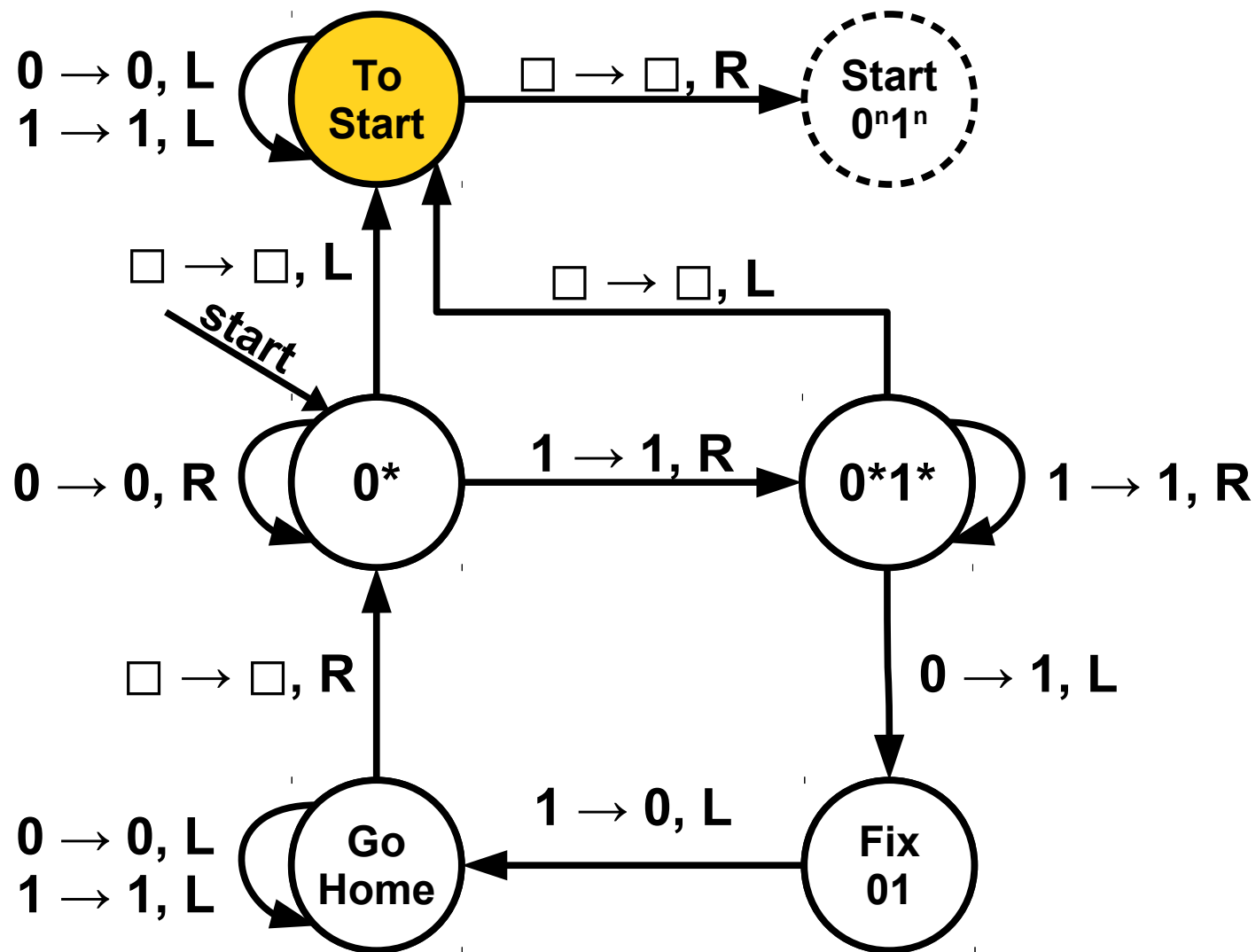


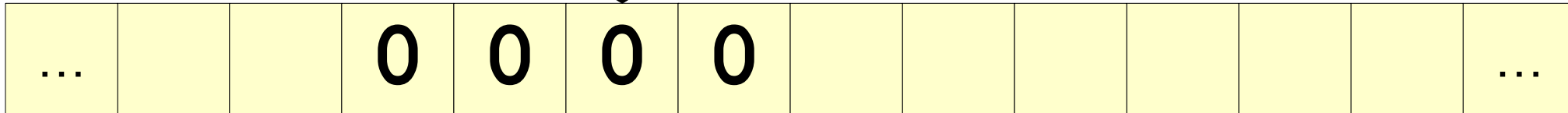
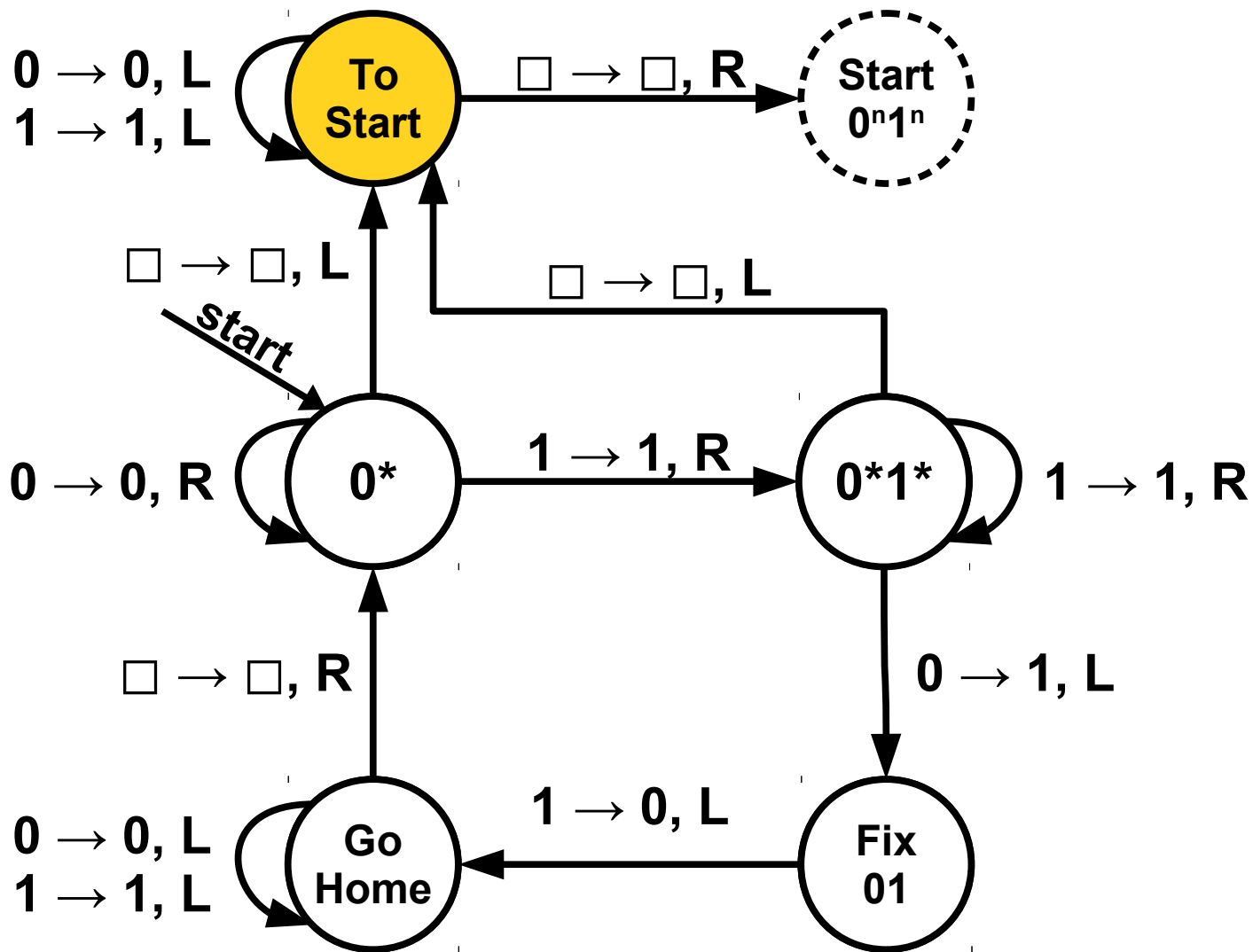


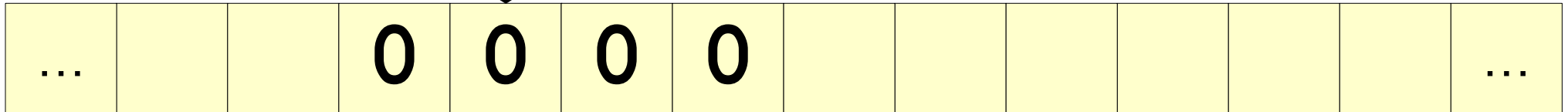
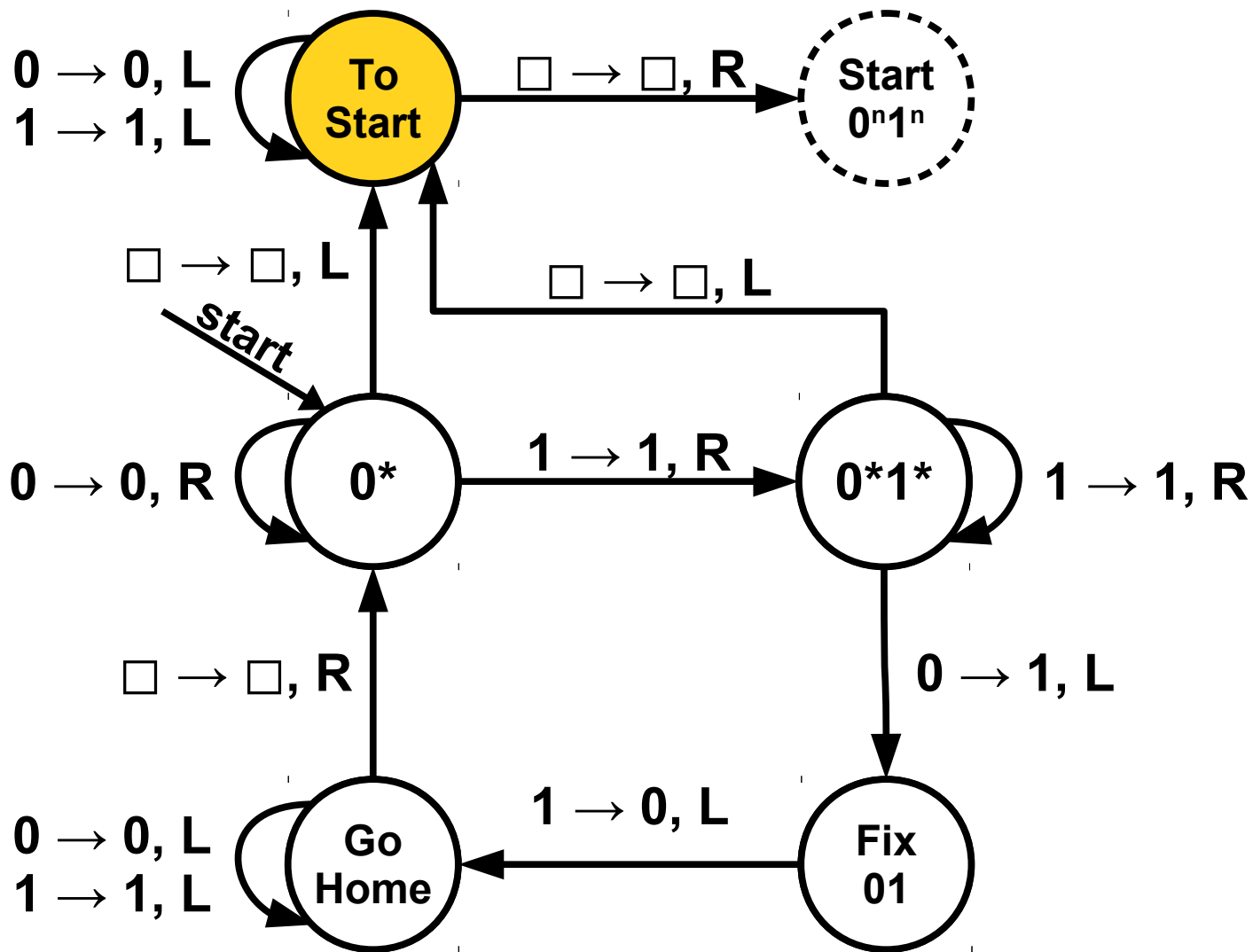


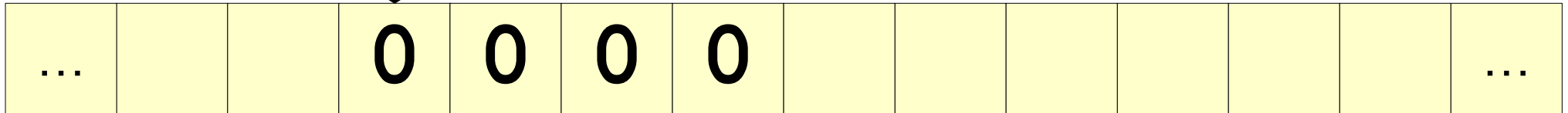
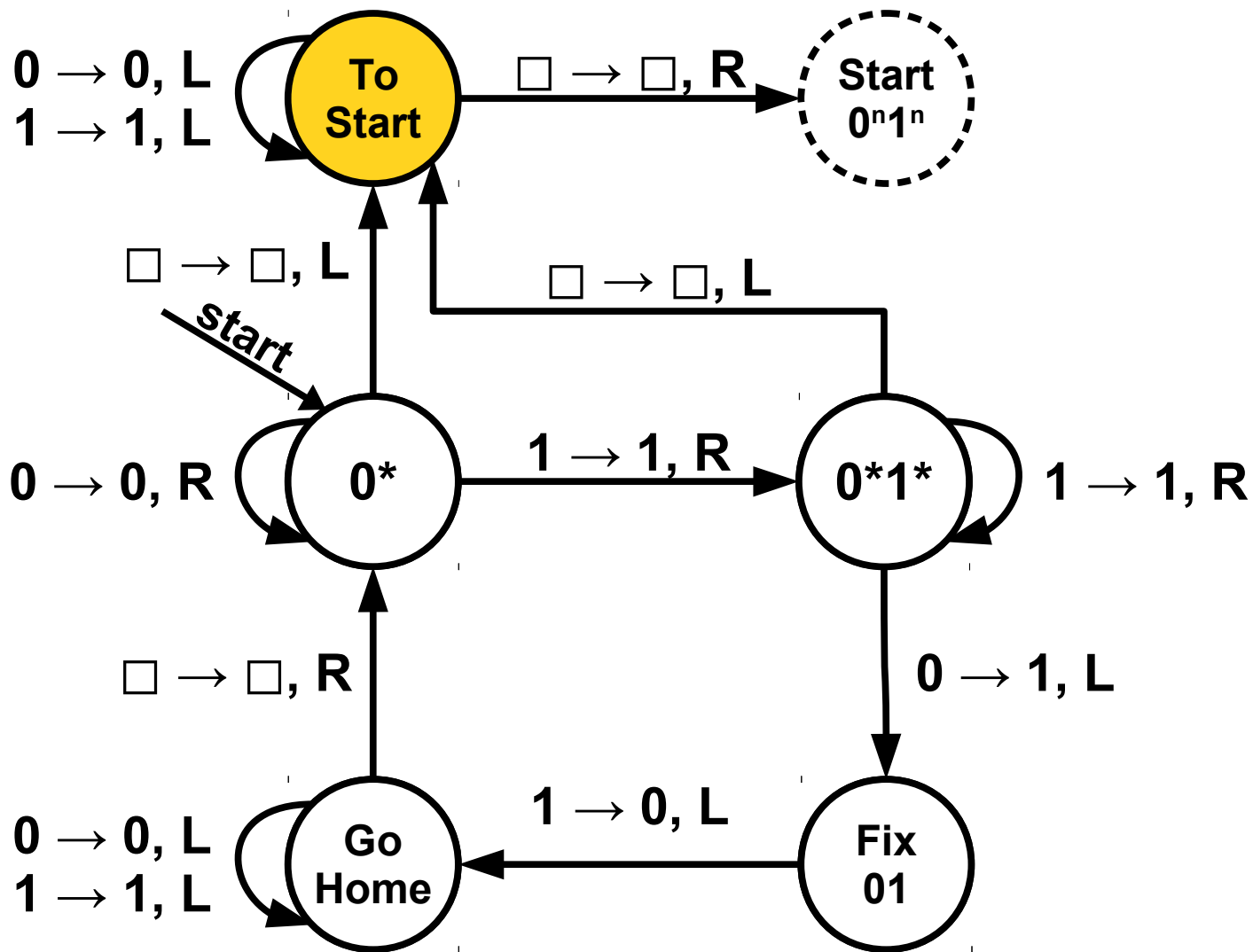


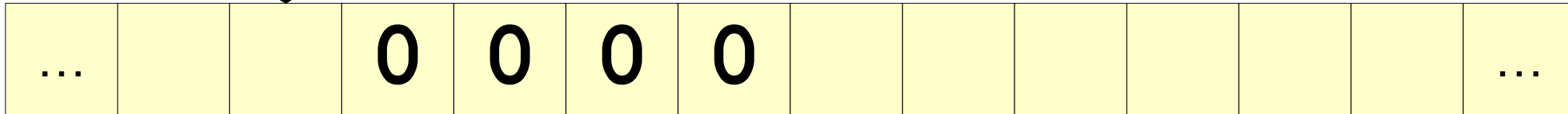
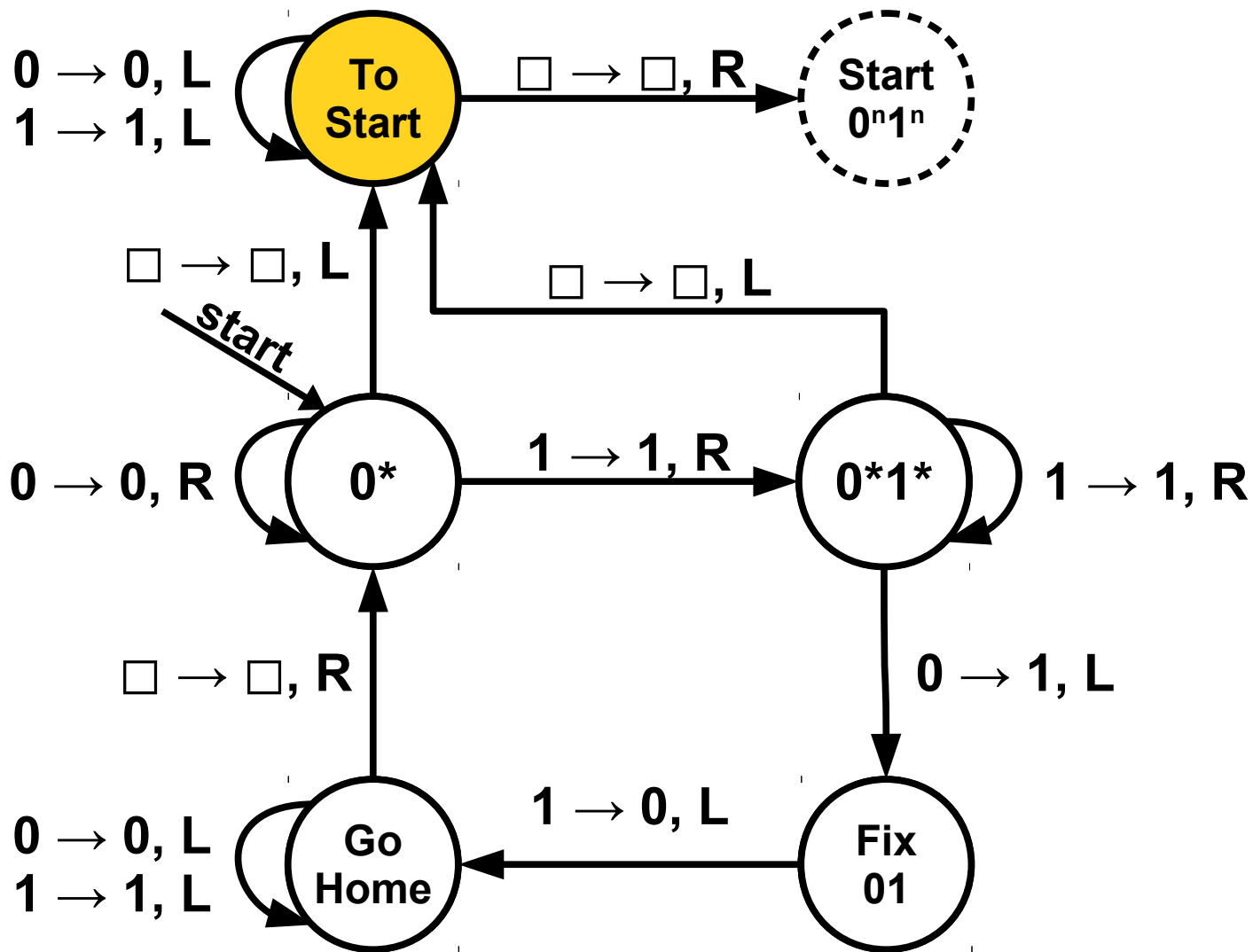


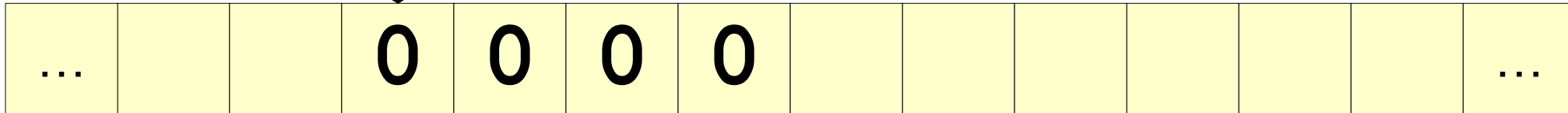
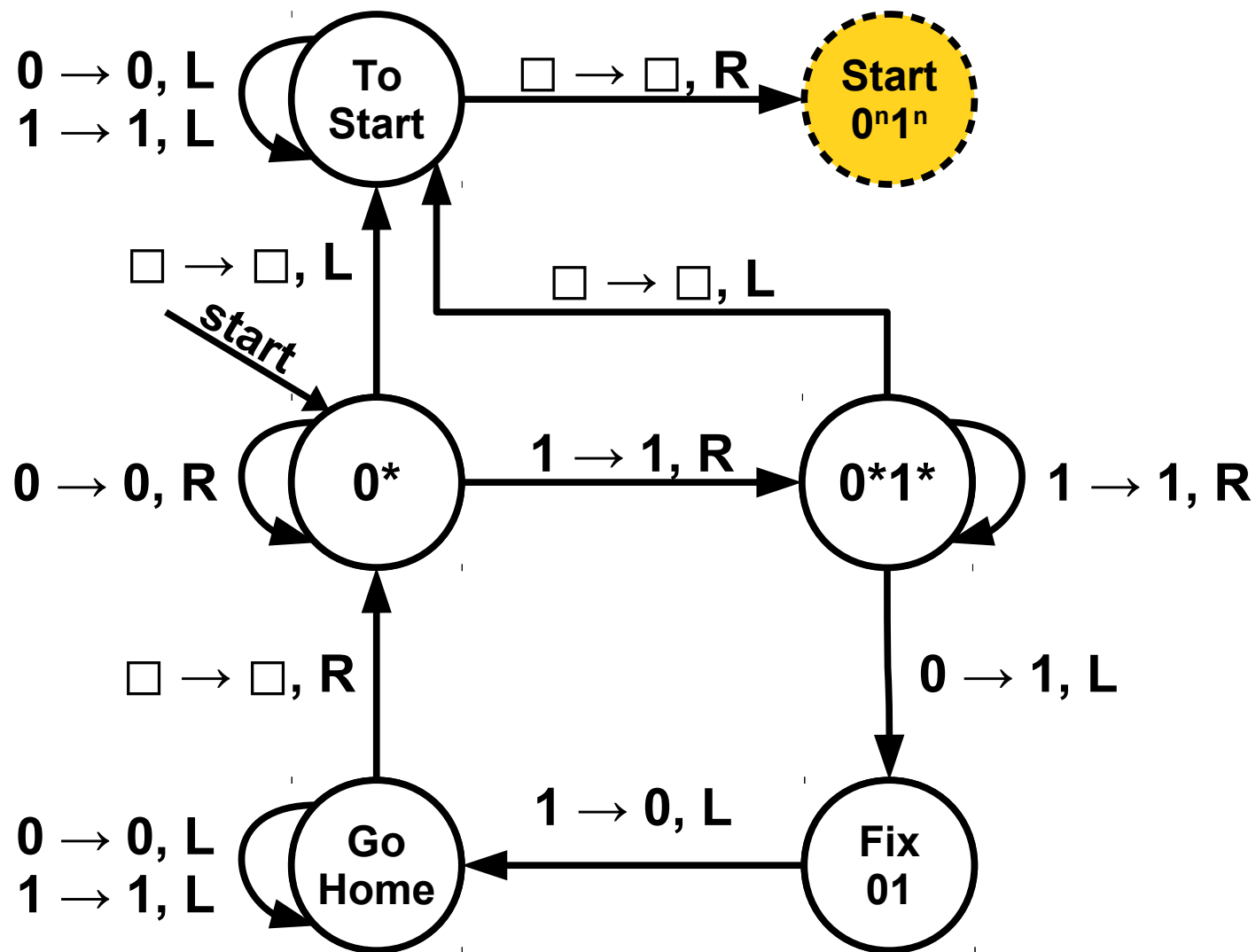


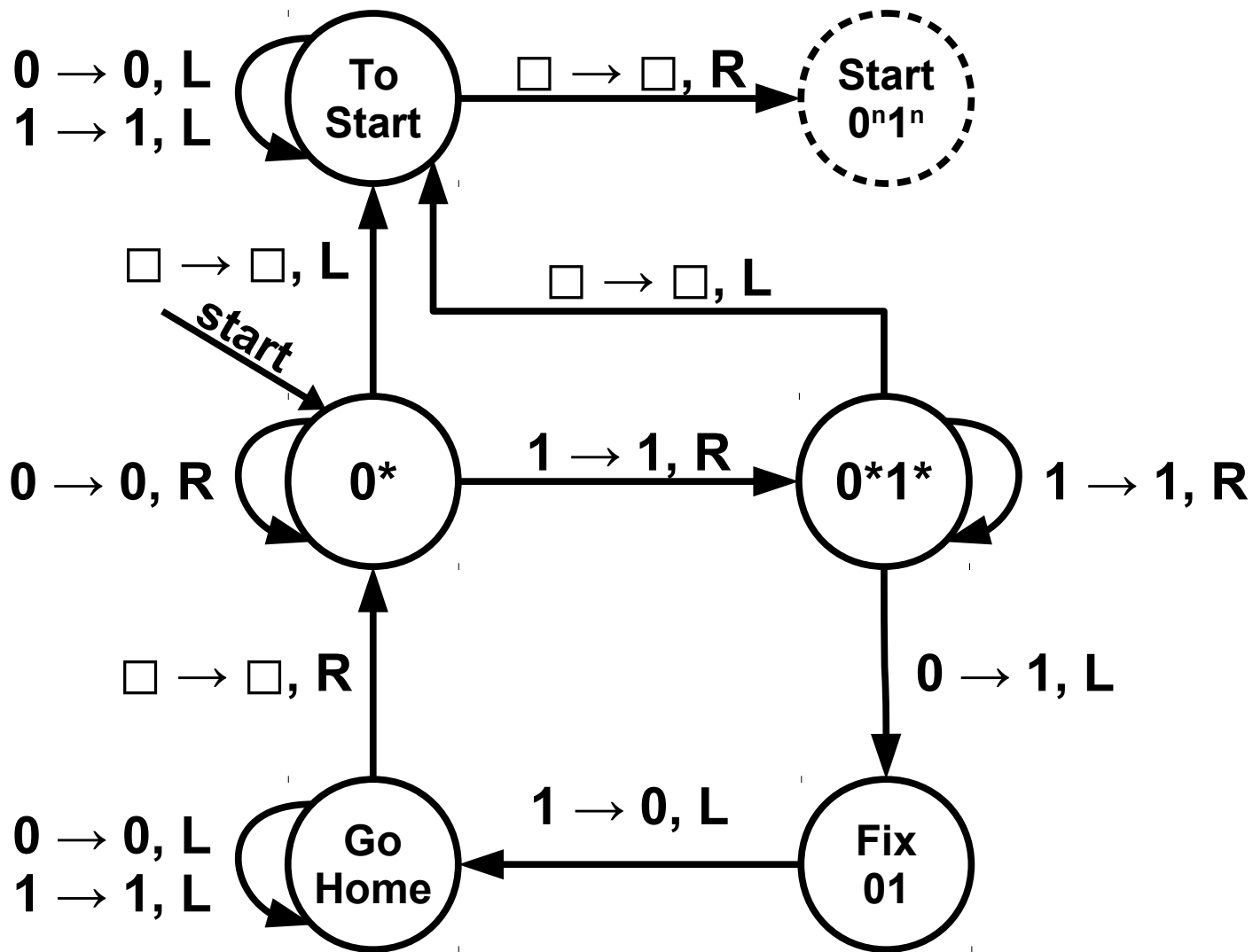


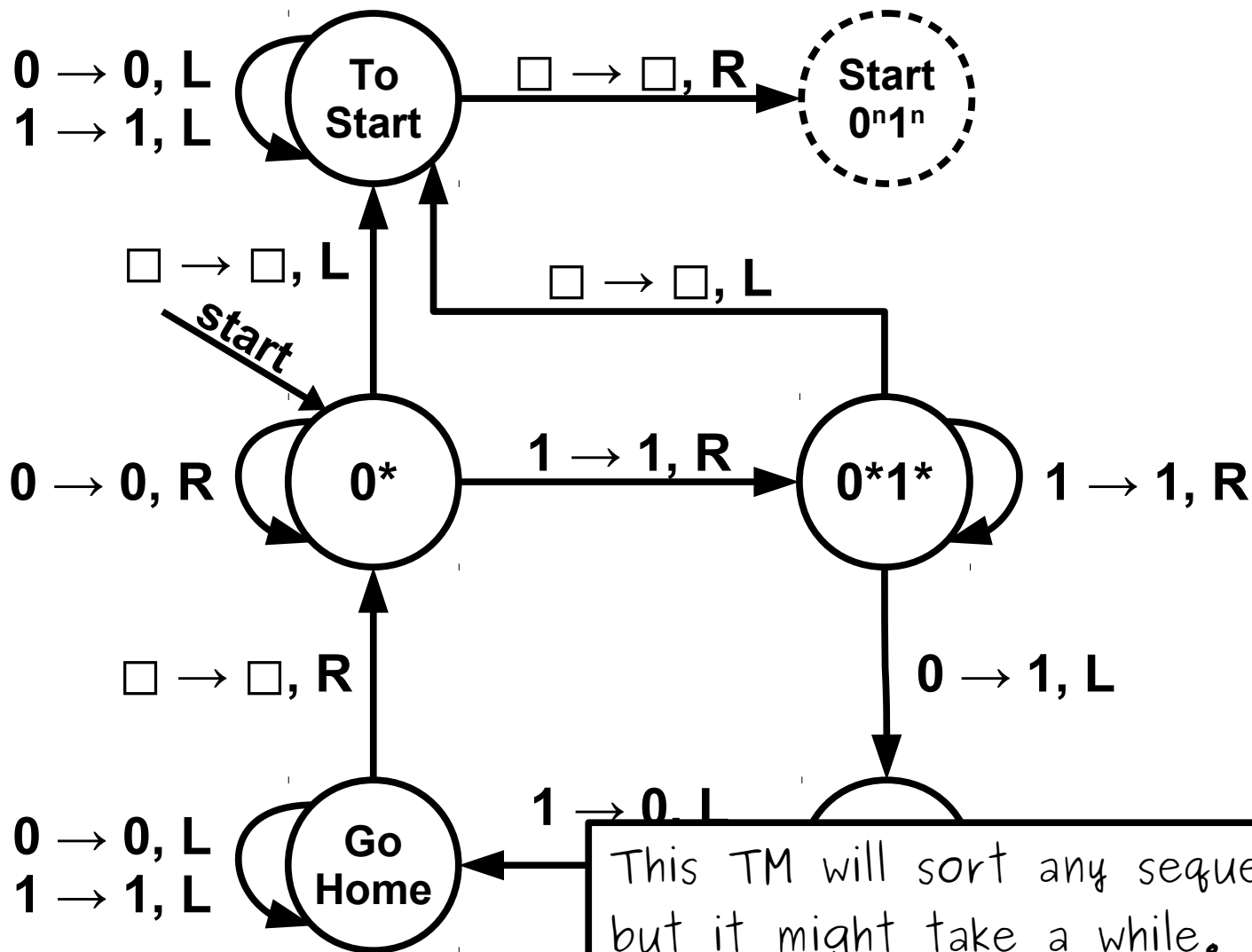












This TM will sort any sequence of 0s and 1s, but it might take a while.

Fun problem: design a TM that sorts a string of 0s and 1s, but does so while taking way fewer steps than this machine.

TM Subroutines

- A ***TM subroutine*** is a Turing machine that, instead of accepting or rejecting an input, does some sort of processing job.
- TM subroutines let us compose larger TMs out of smaller TMs, just as you'd write a larger program using lots of smaller helper functions.
- Here, we saw a TM subroutine that sorts a sequence of 0s and 1s into ascending order.

TM Subroutines

- Typically, when a subroutine is done running, you have it enter a state marked “done” with a dashed line around it.
- When we're composing multiple subroutines together – which we'll do in a bit – the idea is that we'll snap in some real state for the “done” state.

What other subroutines can we make?

TM Arithmetic

- Let's design a TM that, given a tape that looks like this:

...				1	3	7		4	2			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

ends up having the tape look like this:

...				1	7	9		0	0			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

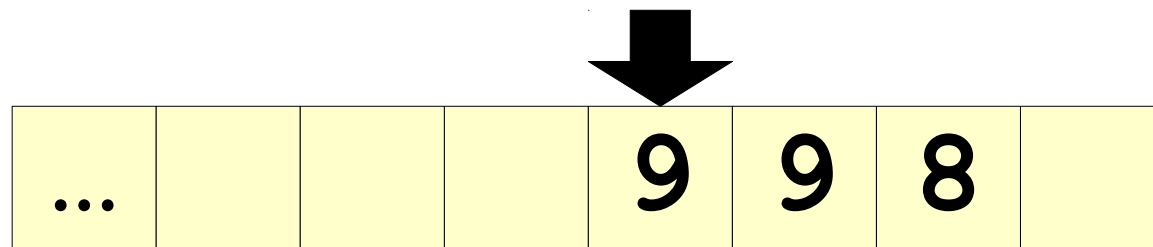
- In other words, we want to build a TM that can add two numbers.

TM Arithmetic

- There are many ways we could in principle design this TM.
- We're going to take the following approach:
 - First, we'll build a TM that increments a number.
 - Next, we'll build a TM that decrements a number.
 - Then, we'll combine them together, repeatedly decrementing the second number and adding one to the first number.

Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



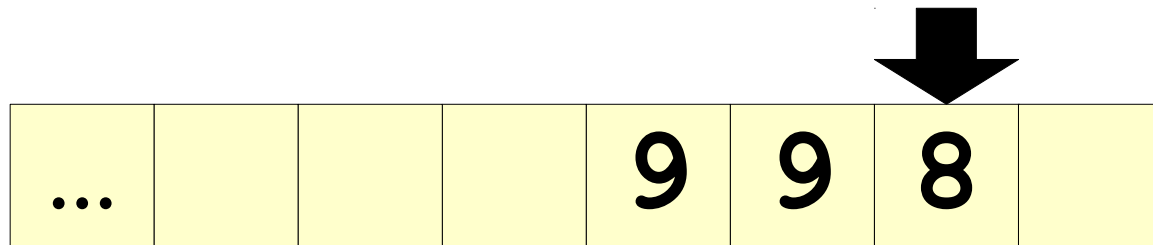
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



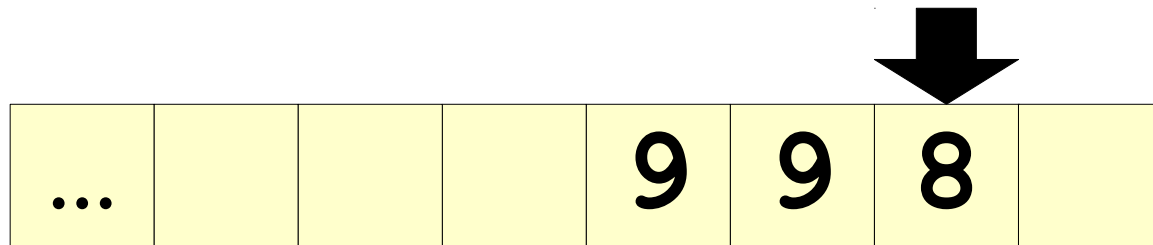
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



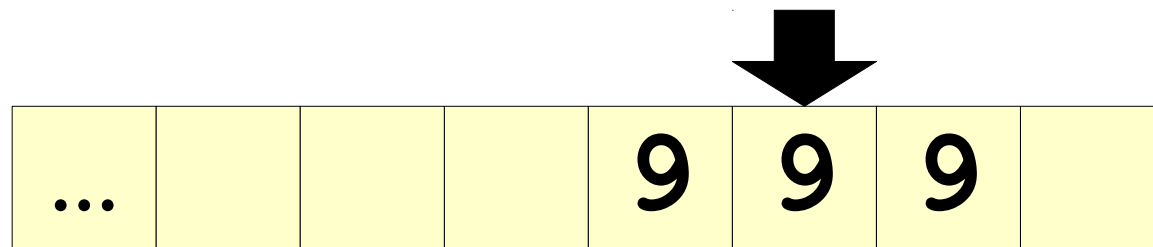
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



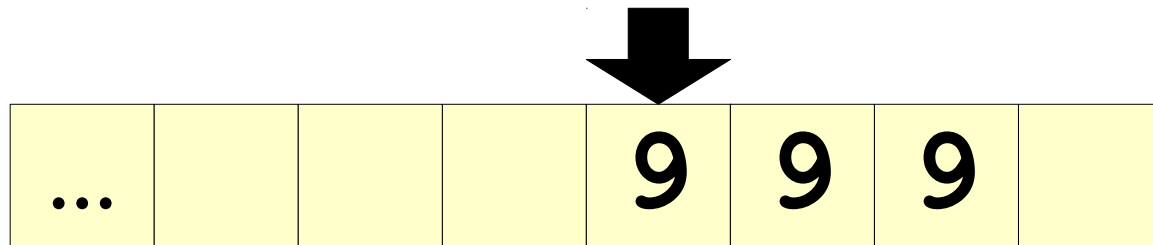
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



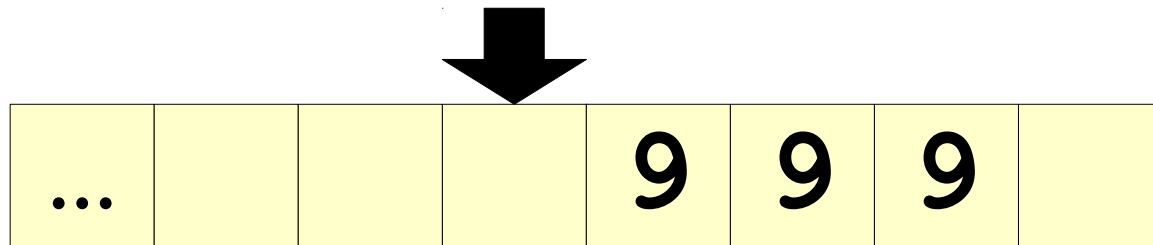
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



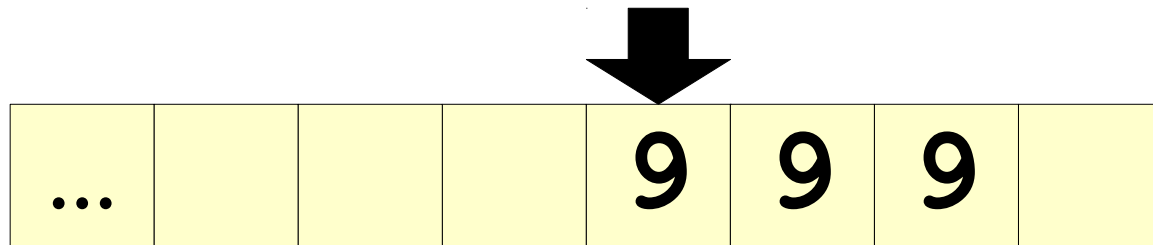
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



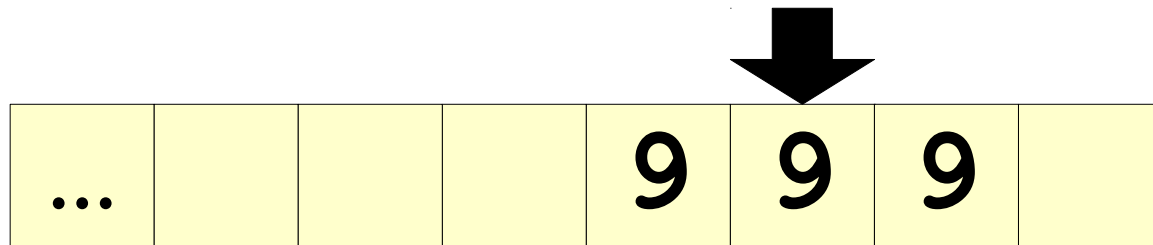
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



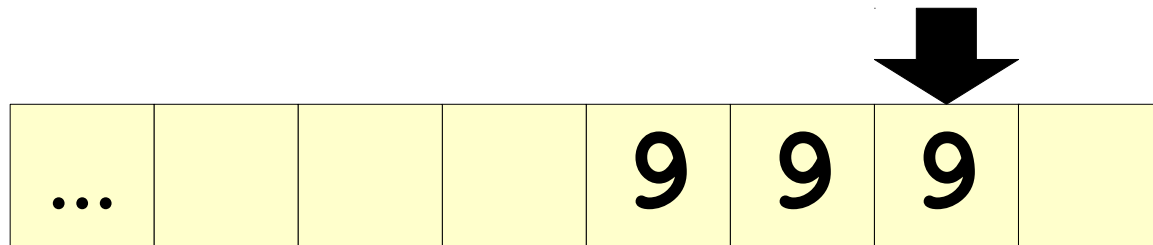
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



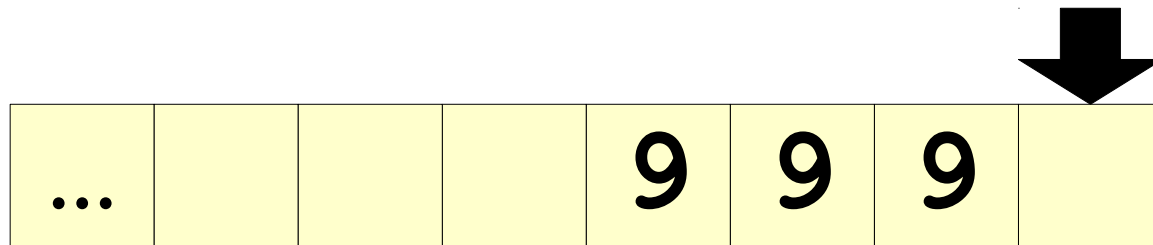
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



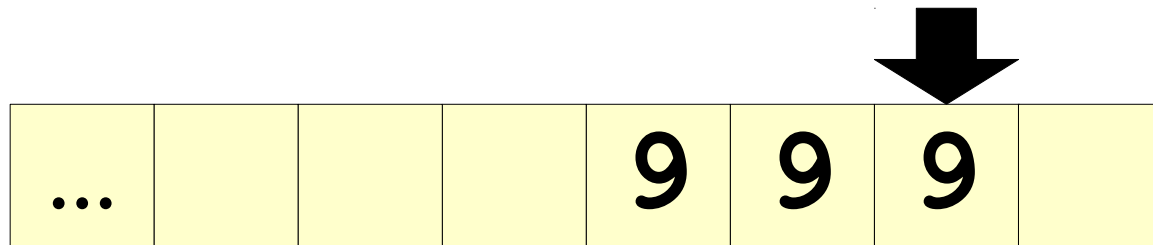
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



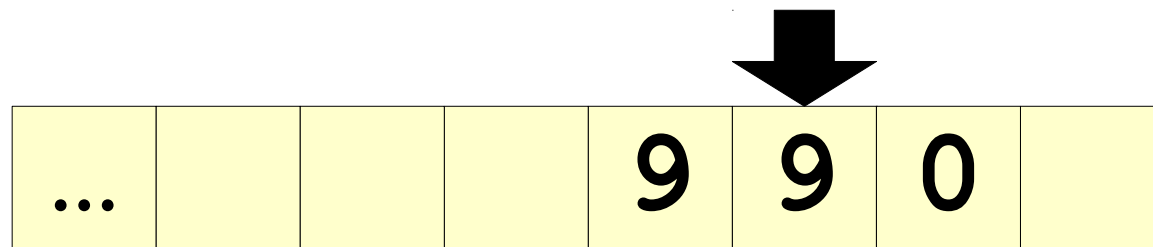
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



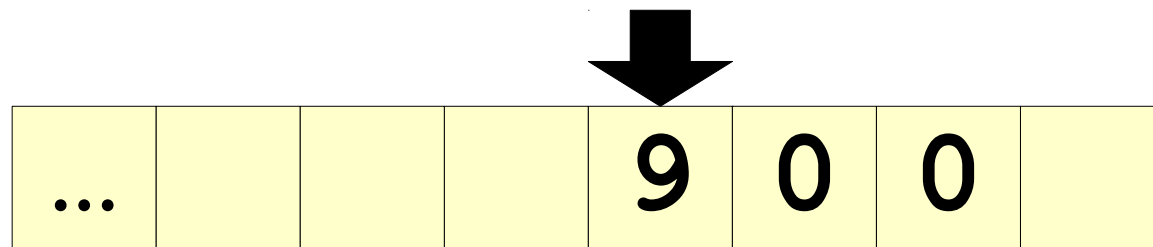
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



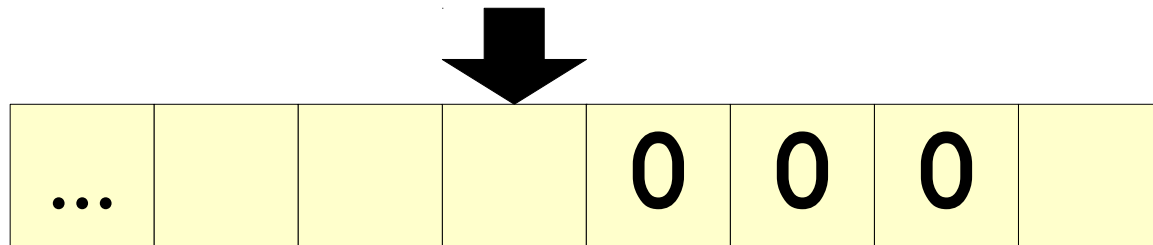
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



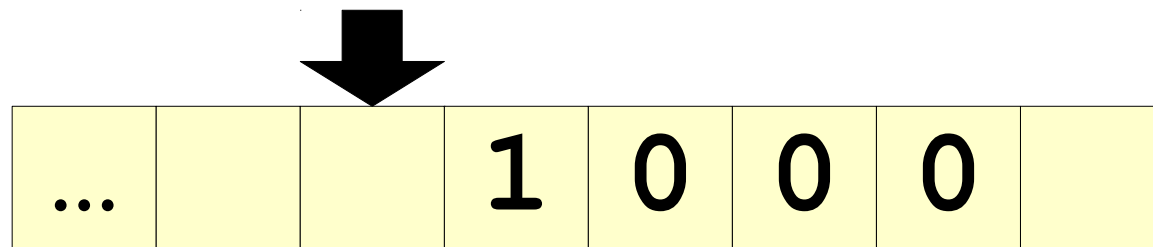
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



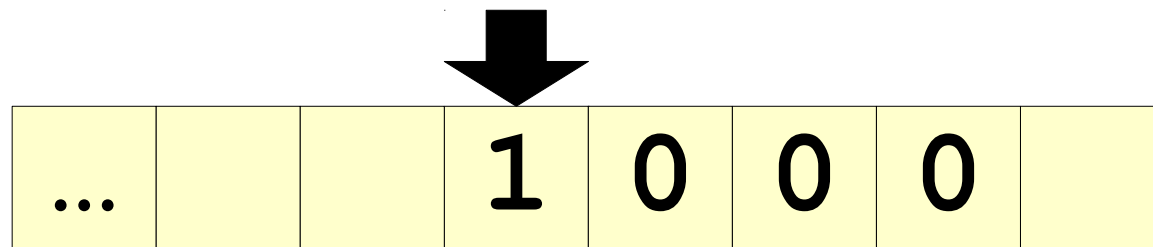
Incrementing Numbers

- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.



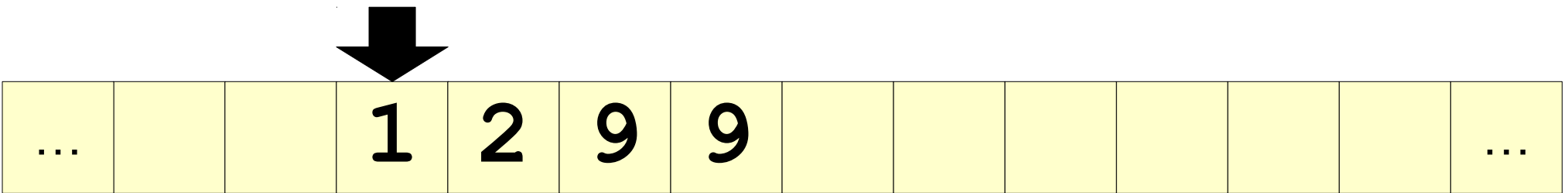
Incrementing Numbers

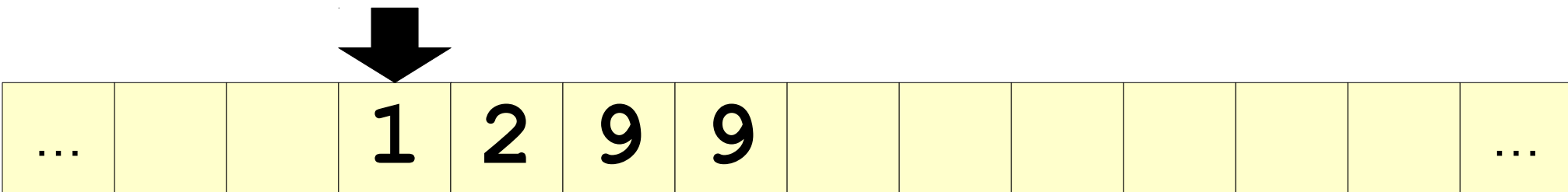
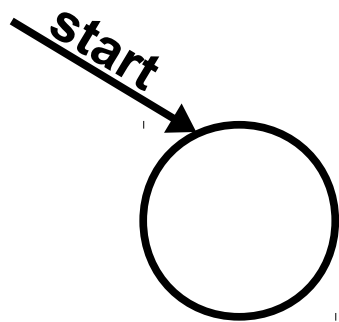
- Let's begin by building a TM that increments a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is are at least two blanks to the left of the number, and
 - that there's at least one blank at the start of the number.
- The tape head will end at the start of the number after incrementing it.

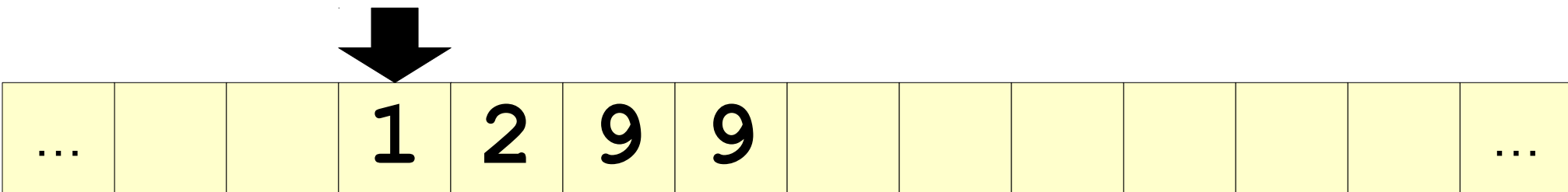
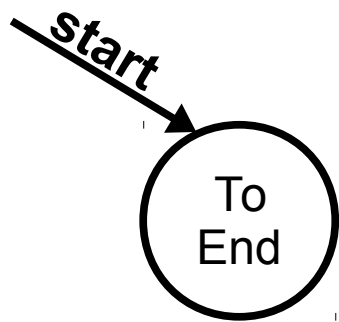


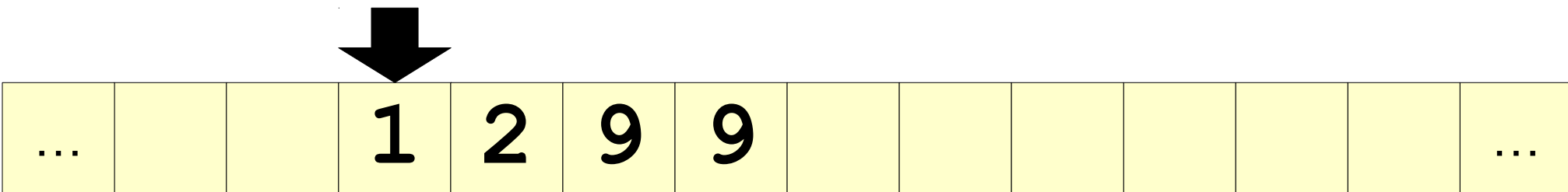
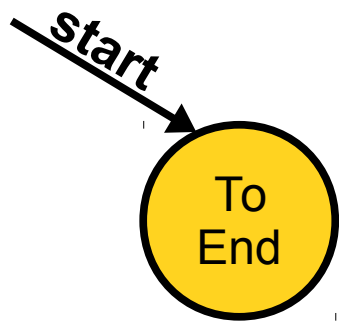
Incrementing Numbers

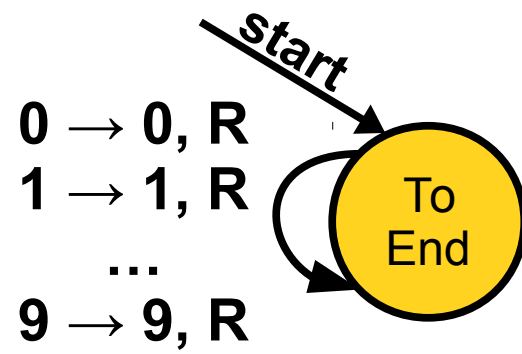
```
go to the end of the number;  
while (the current digit is 9) {  
    set the current digit to 0;  
    back up one digit;  
}  
increment the current digit;  
go to the start of the number;
```

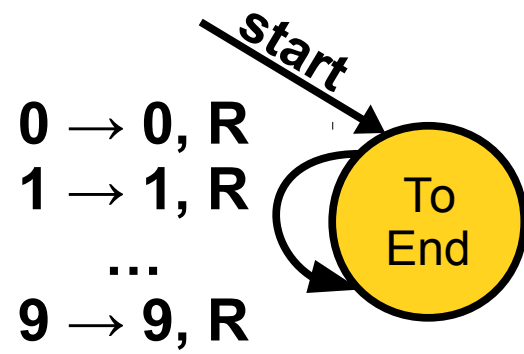


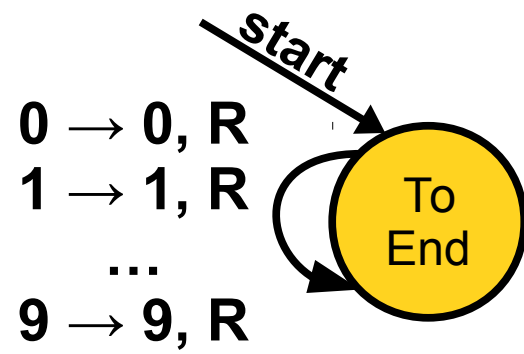


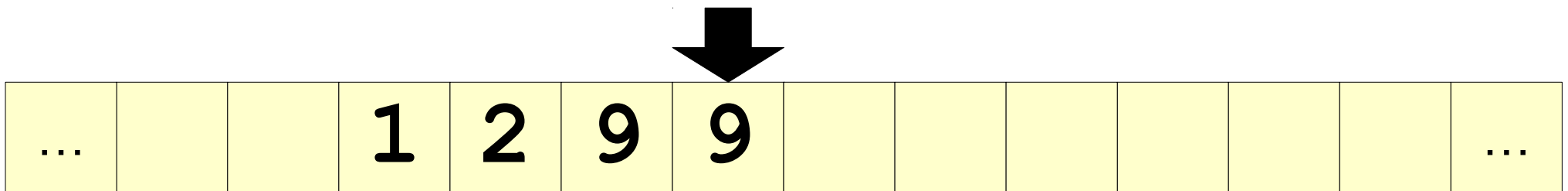
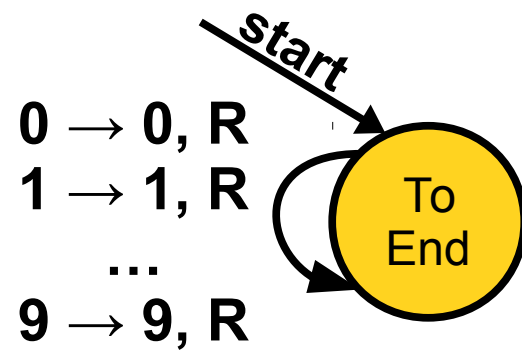


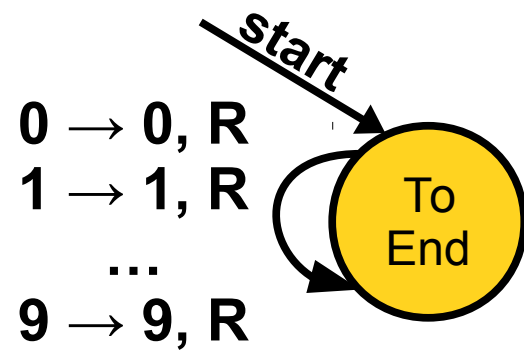


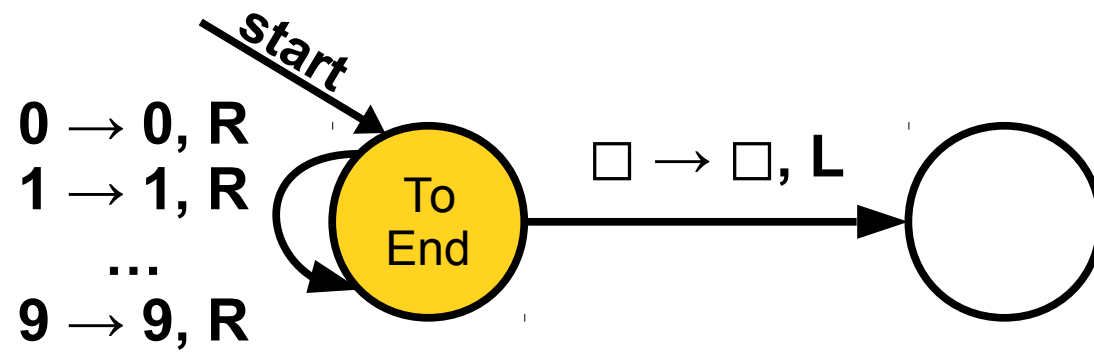


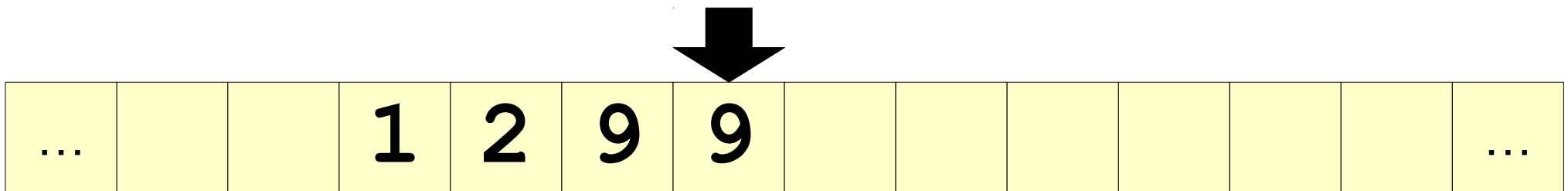
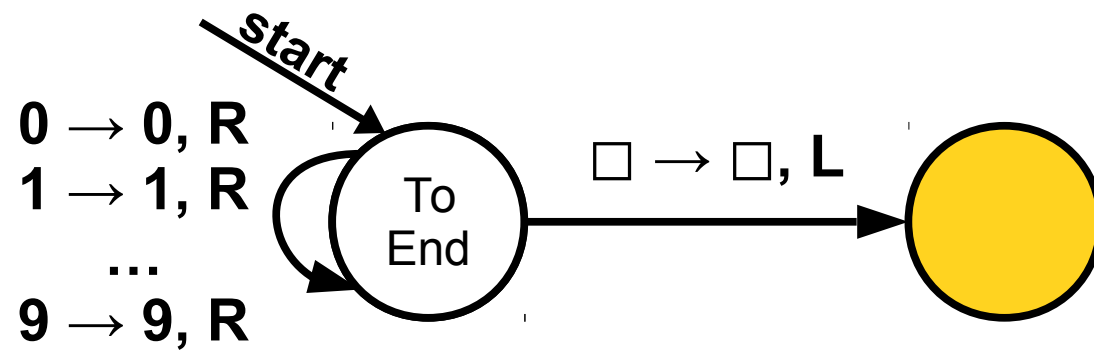


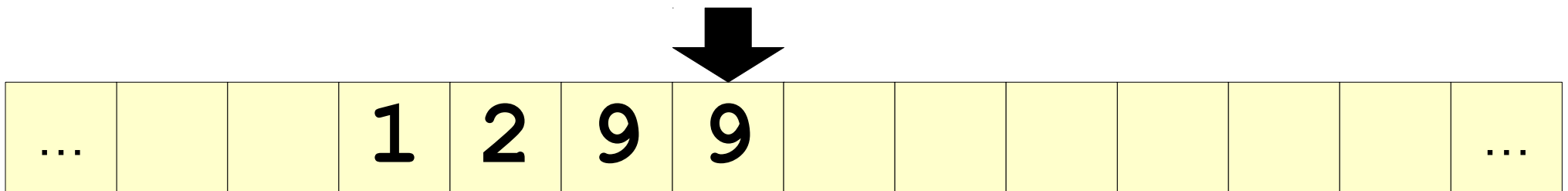
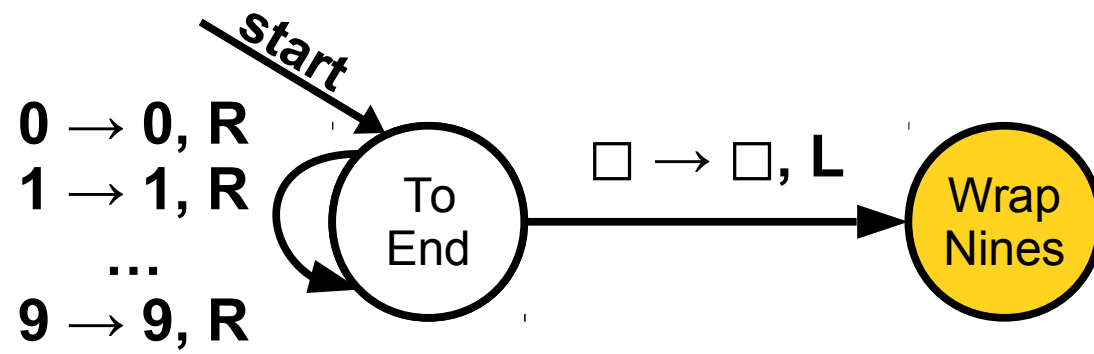


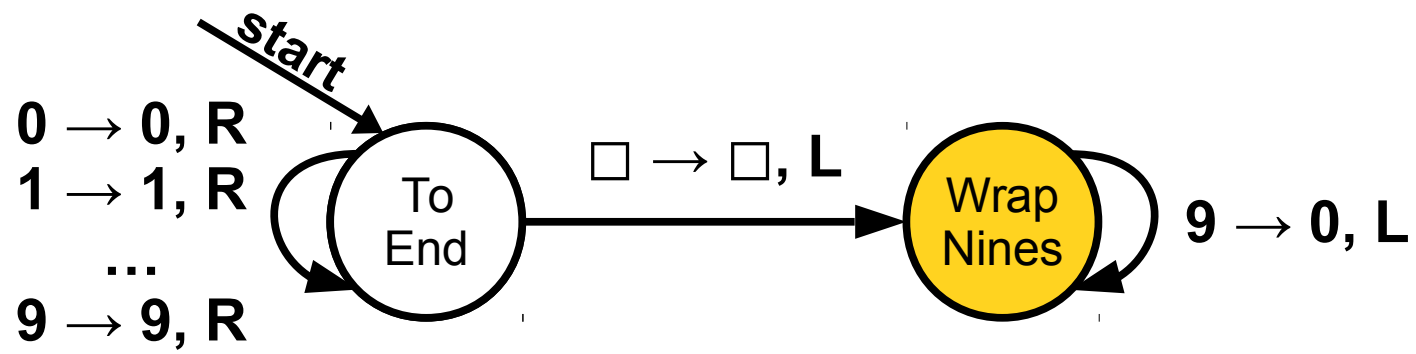


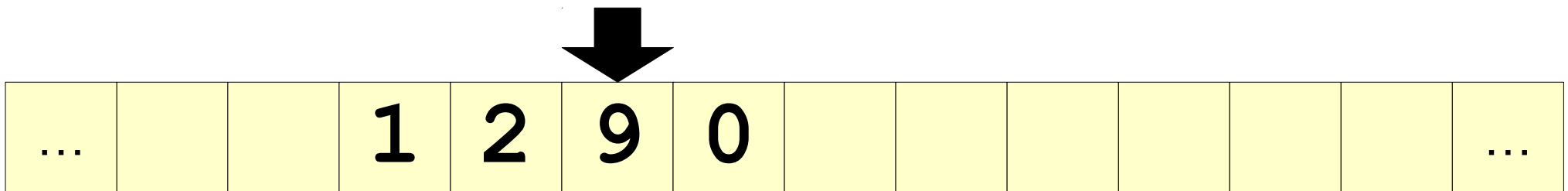
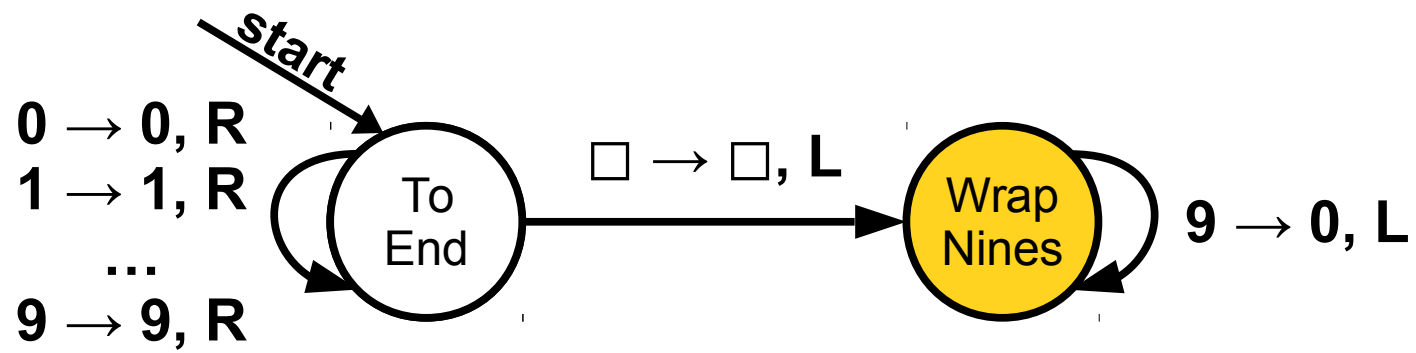


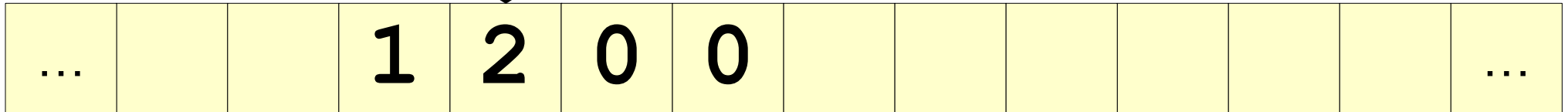
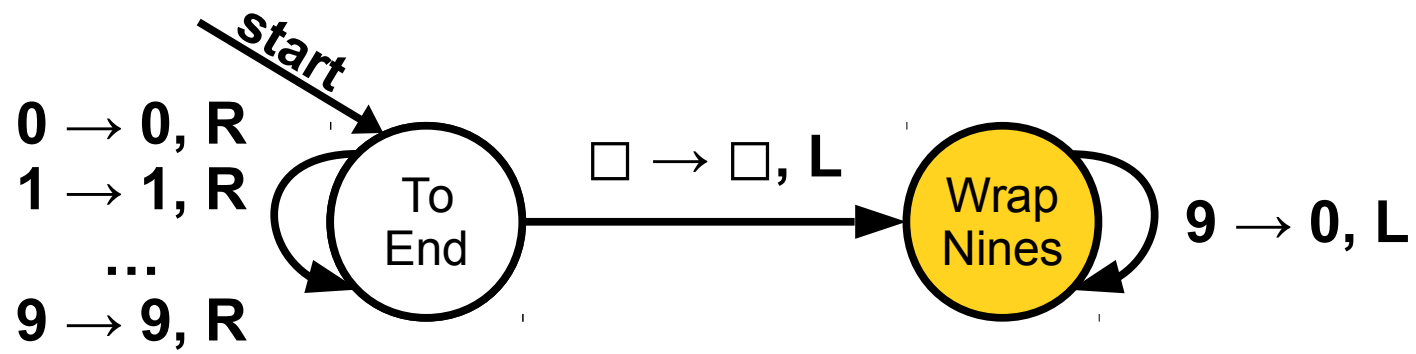


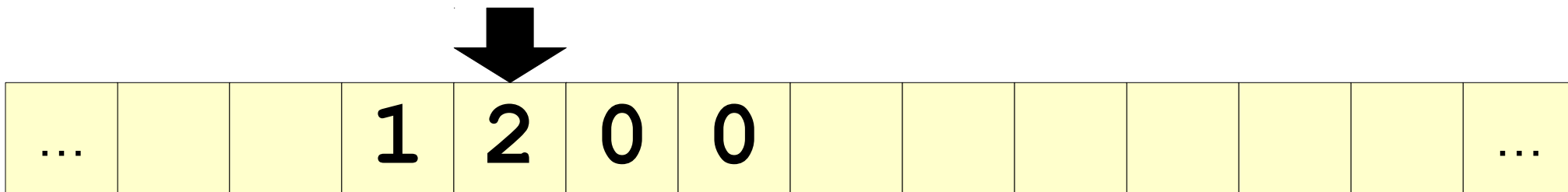
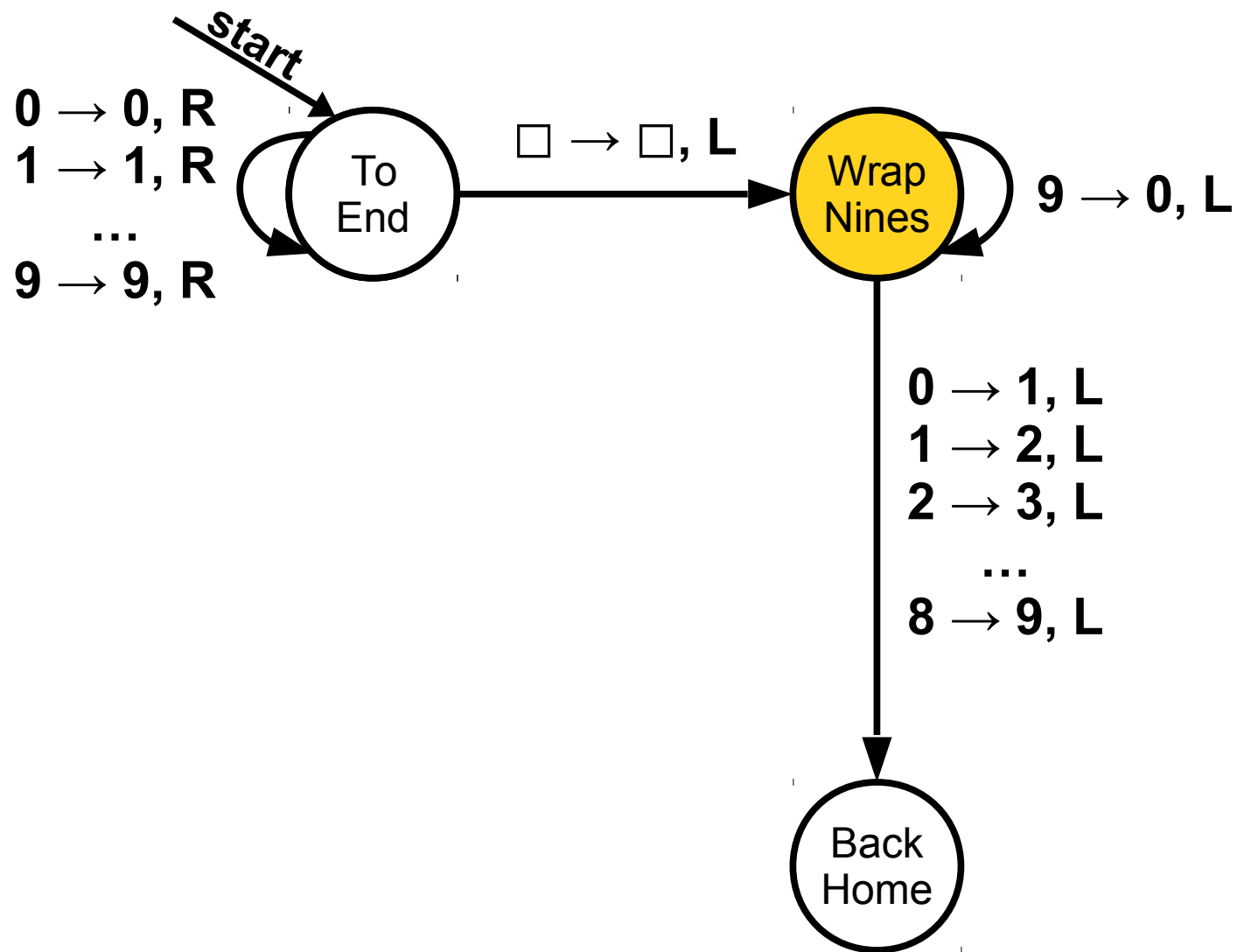


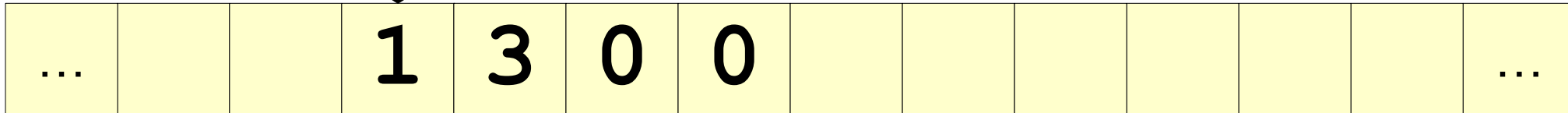
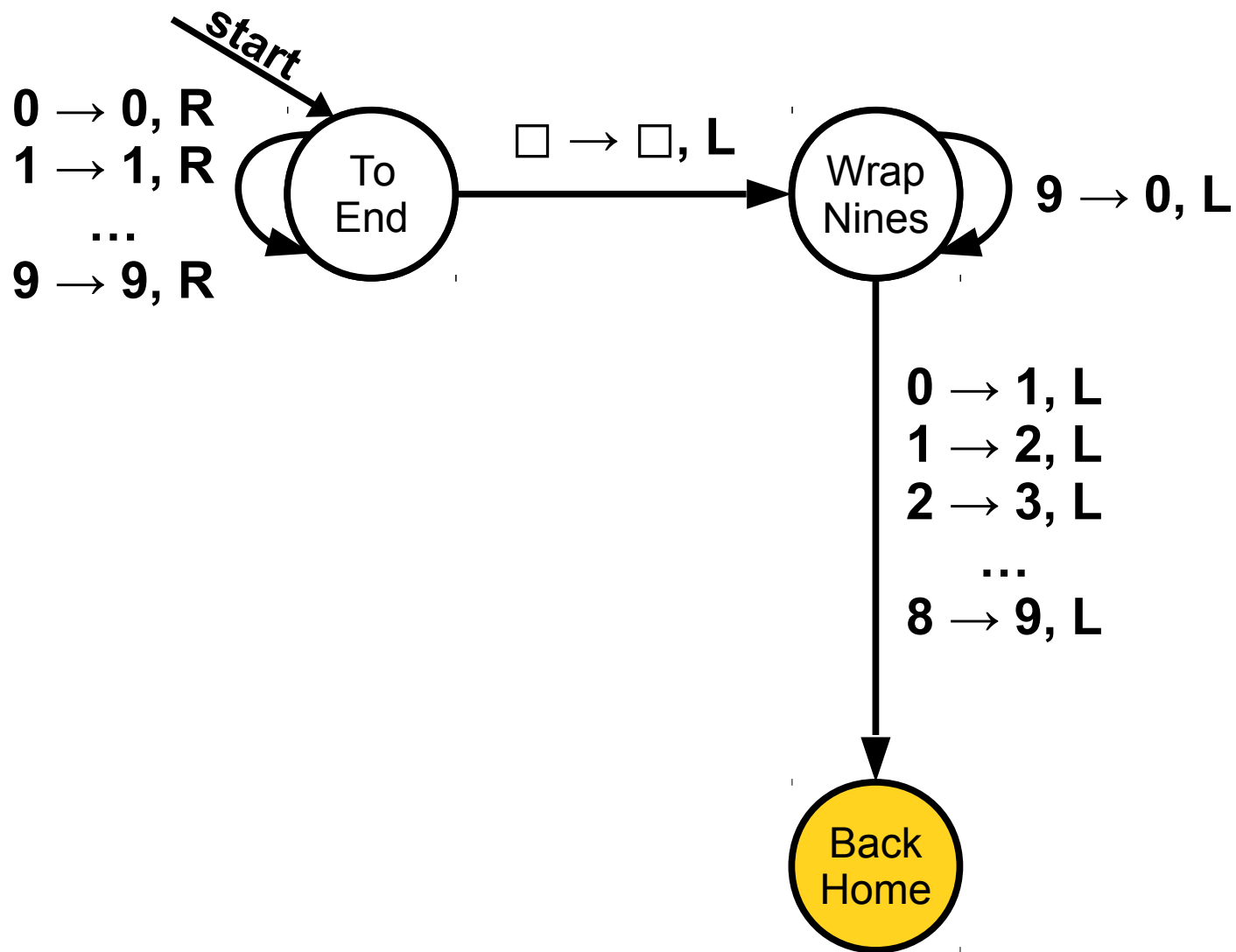


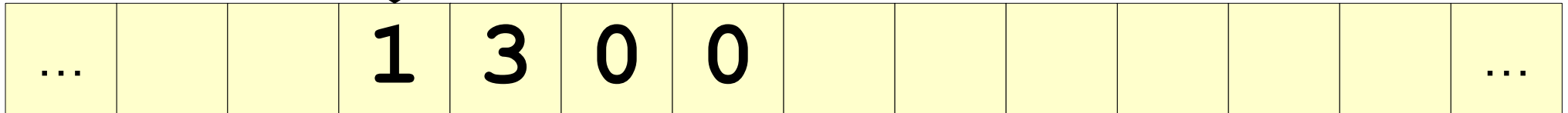
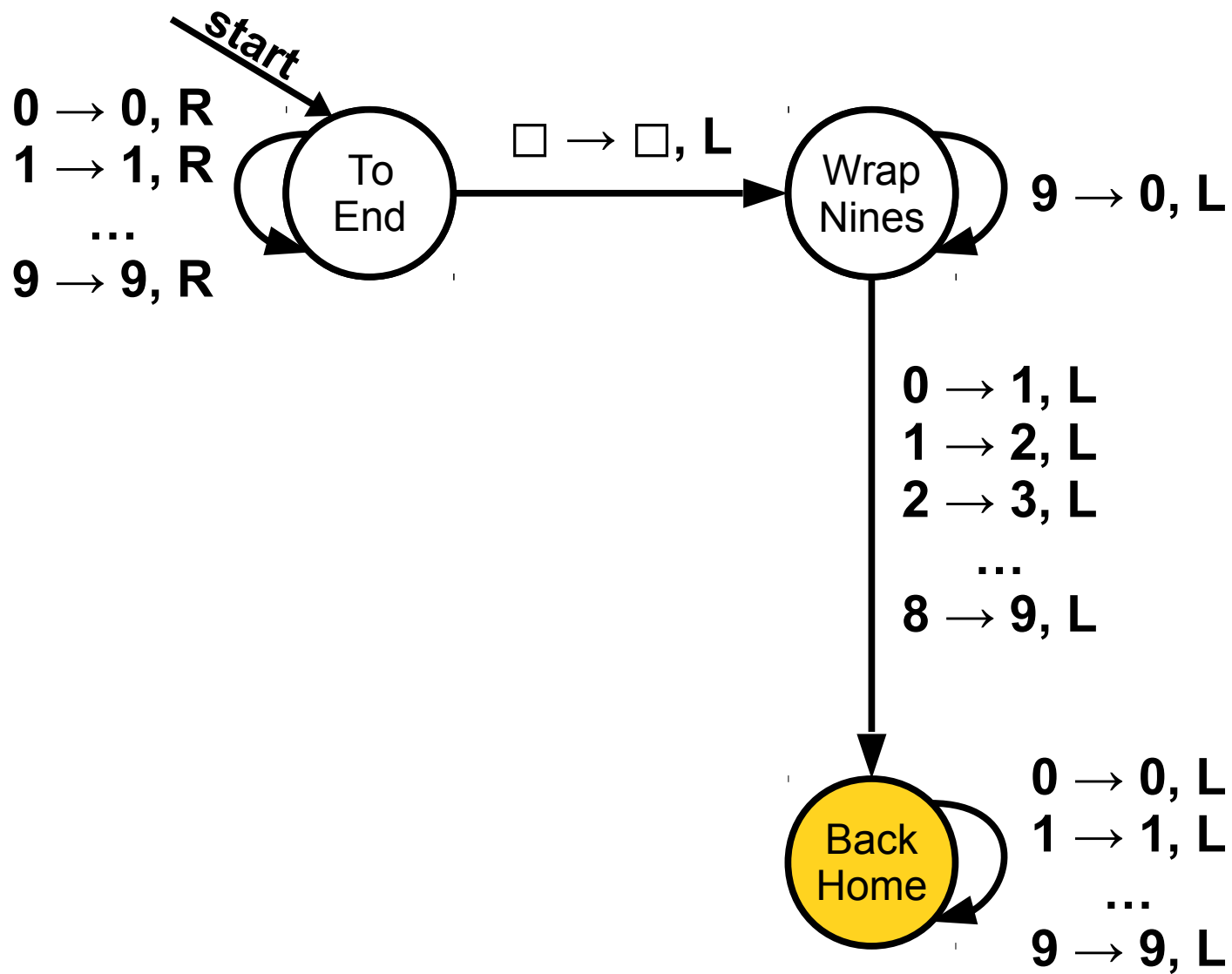


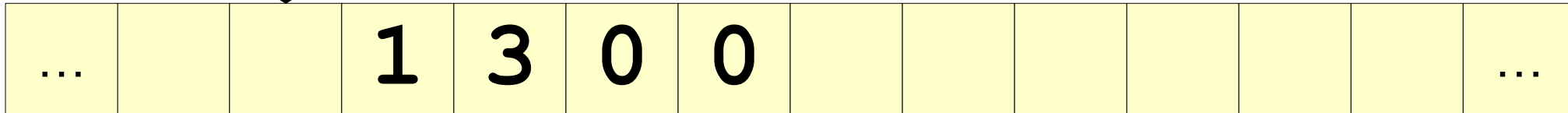
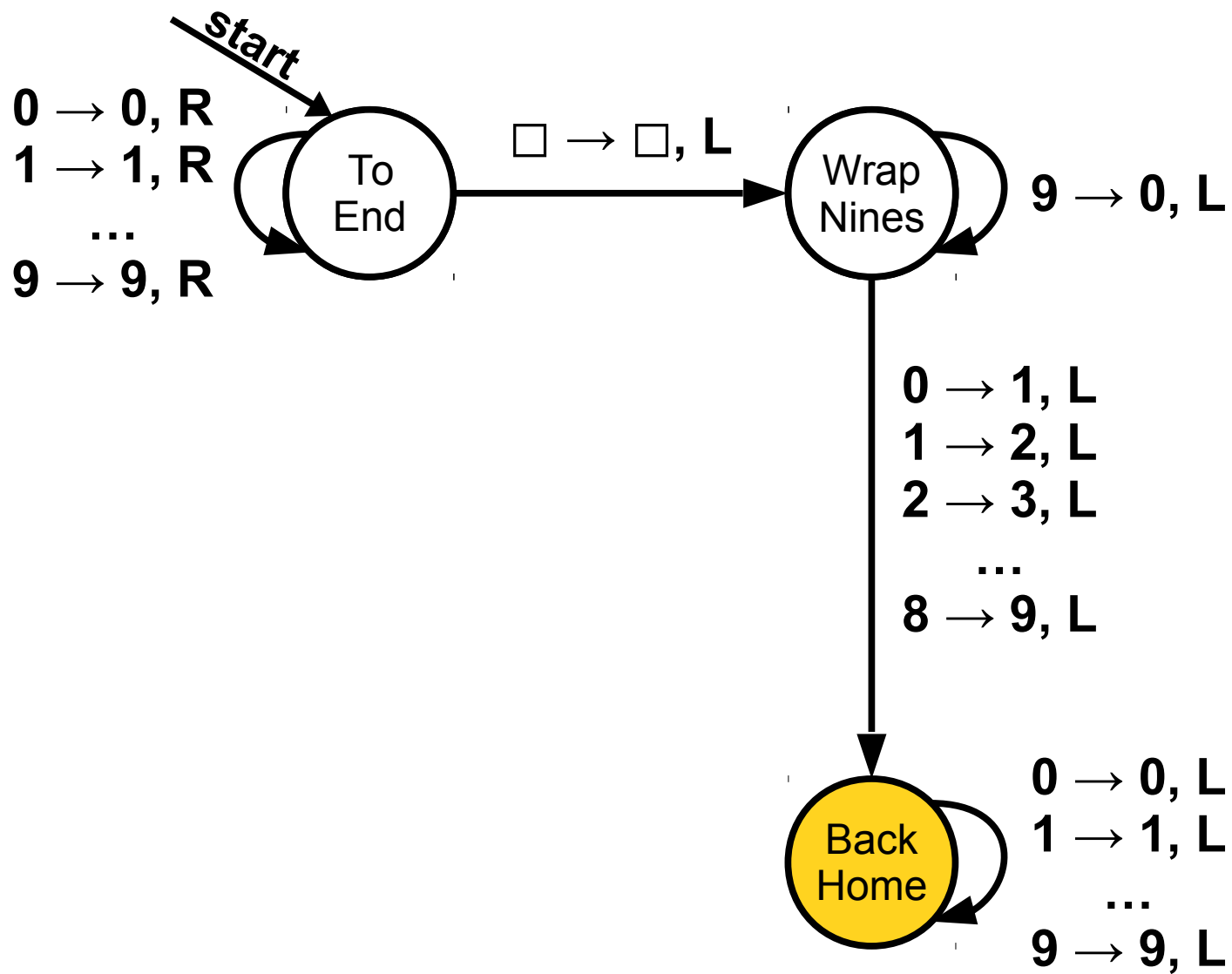


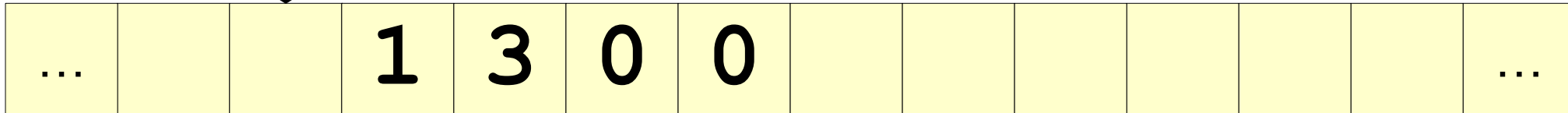
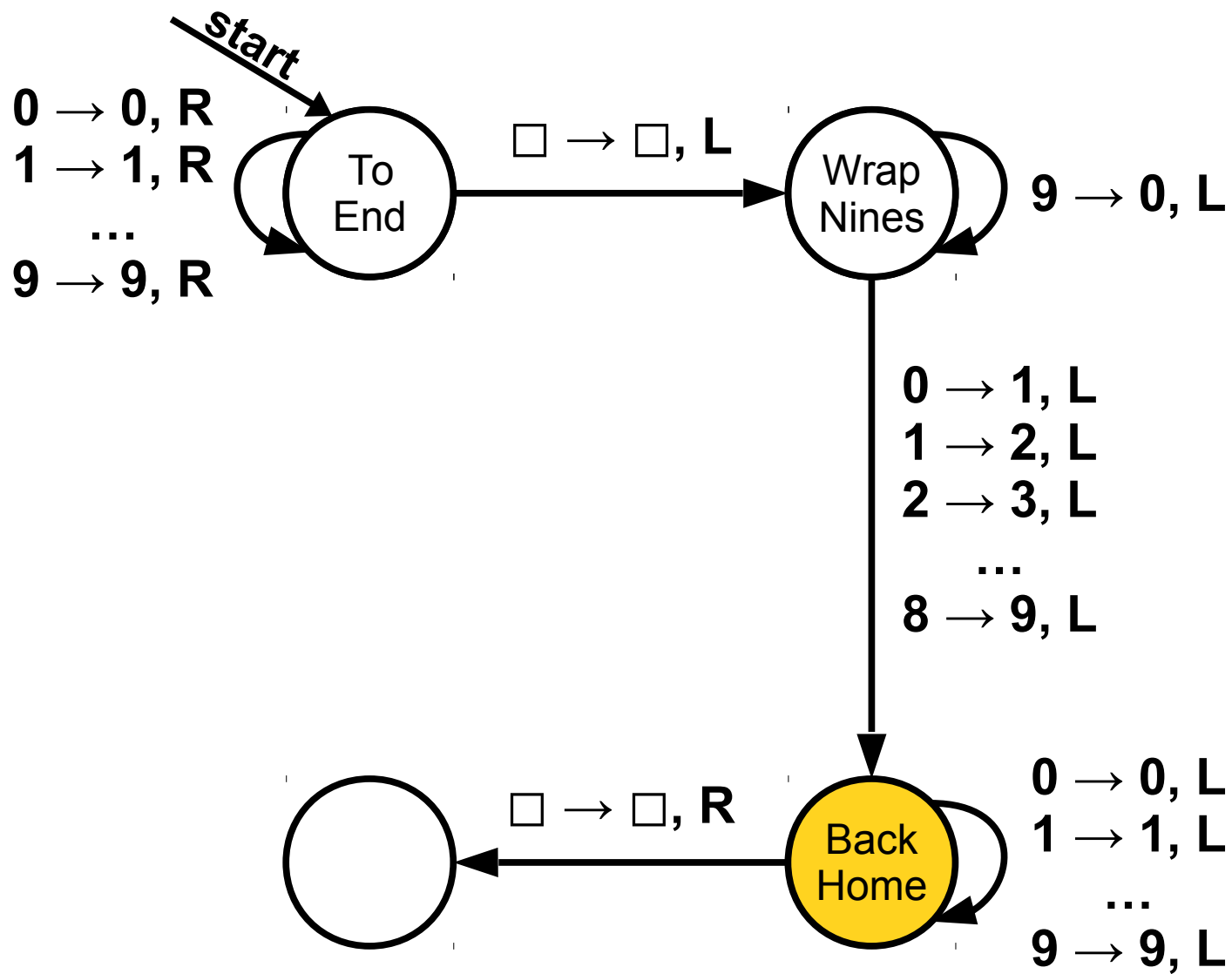


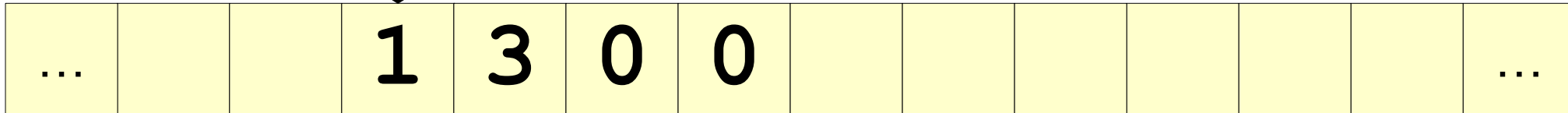
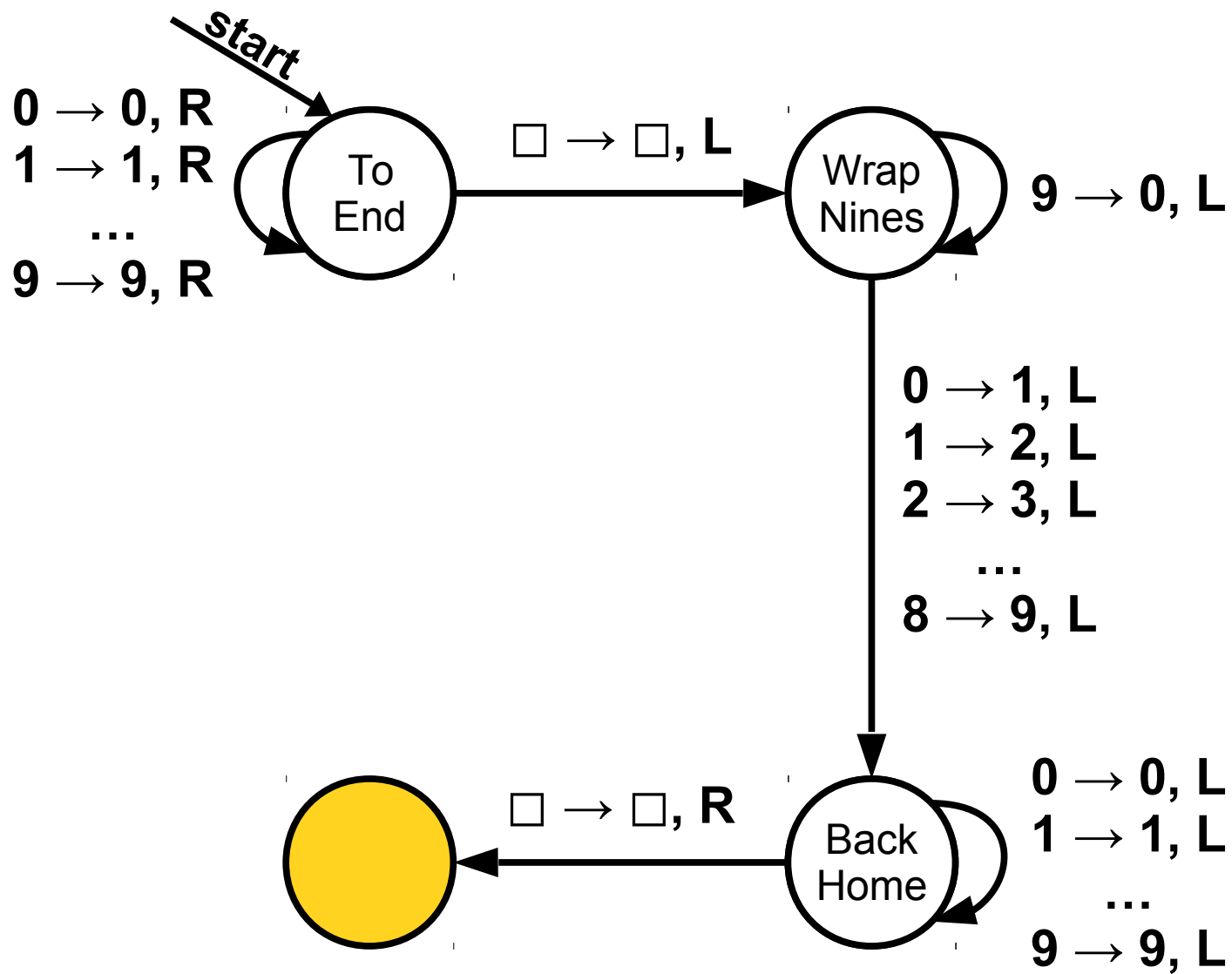


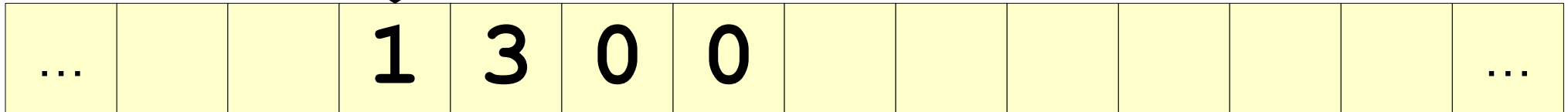
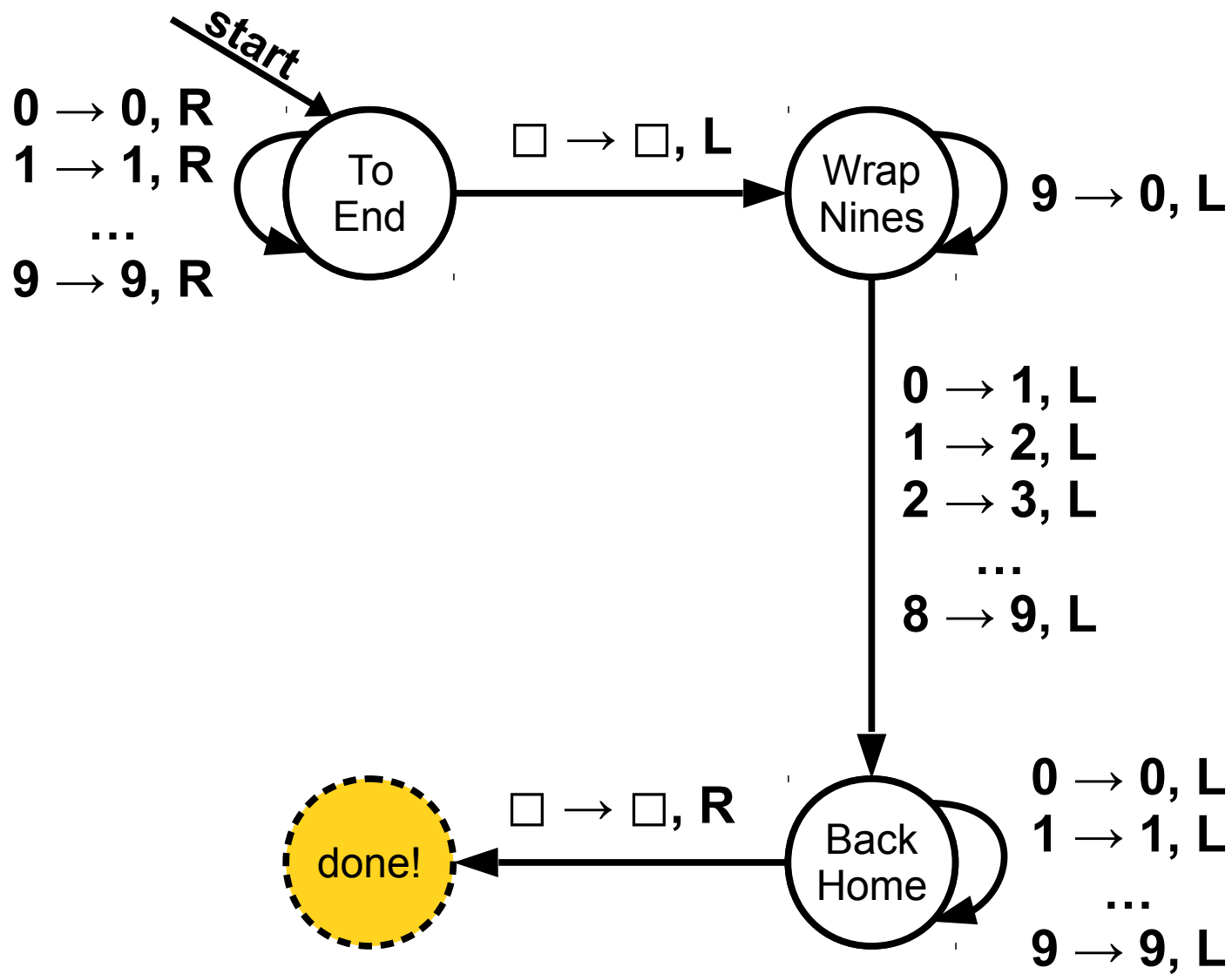


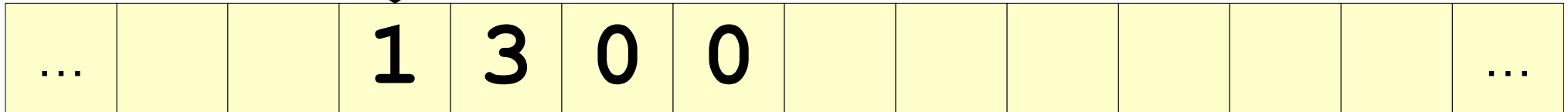
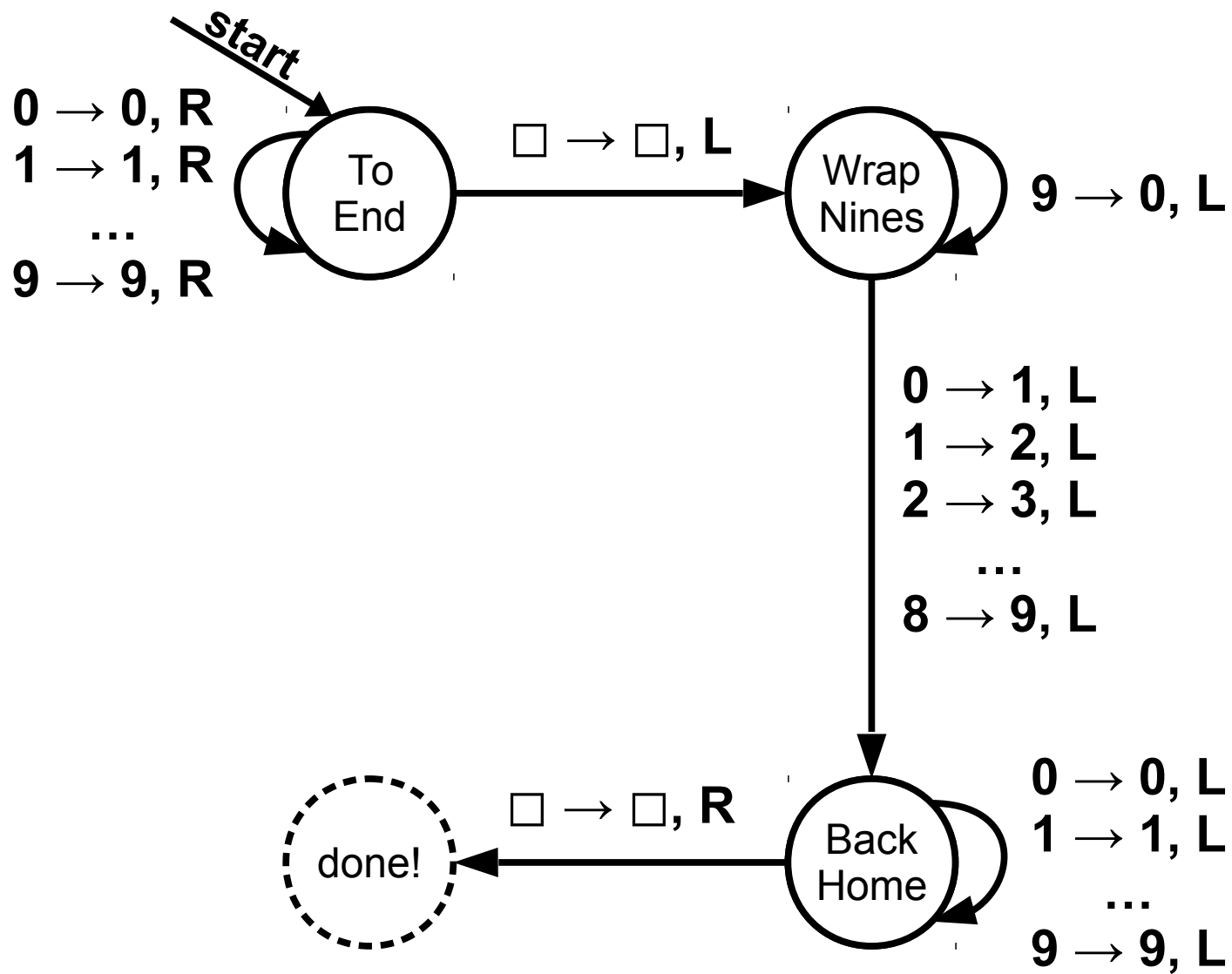


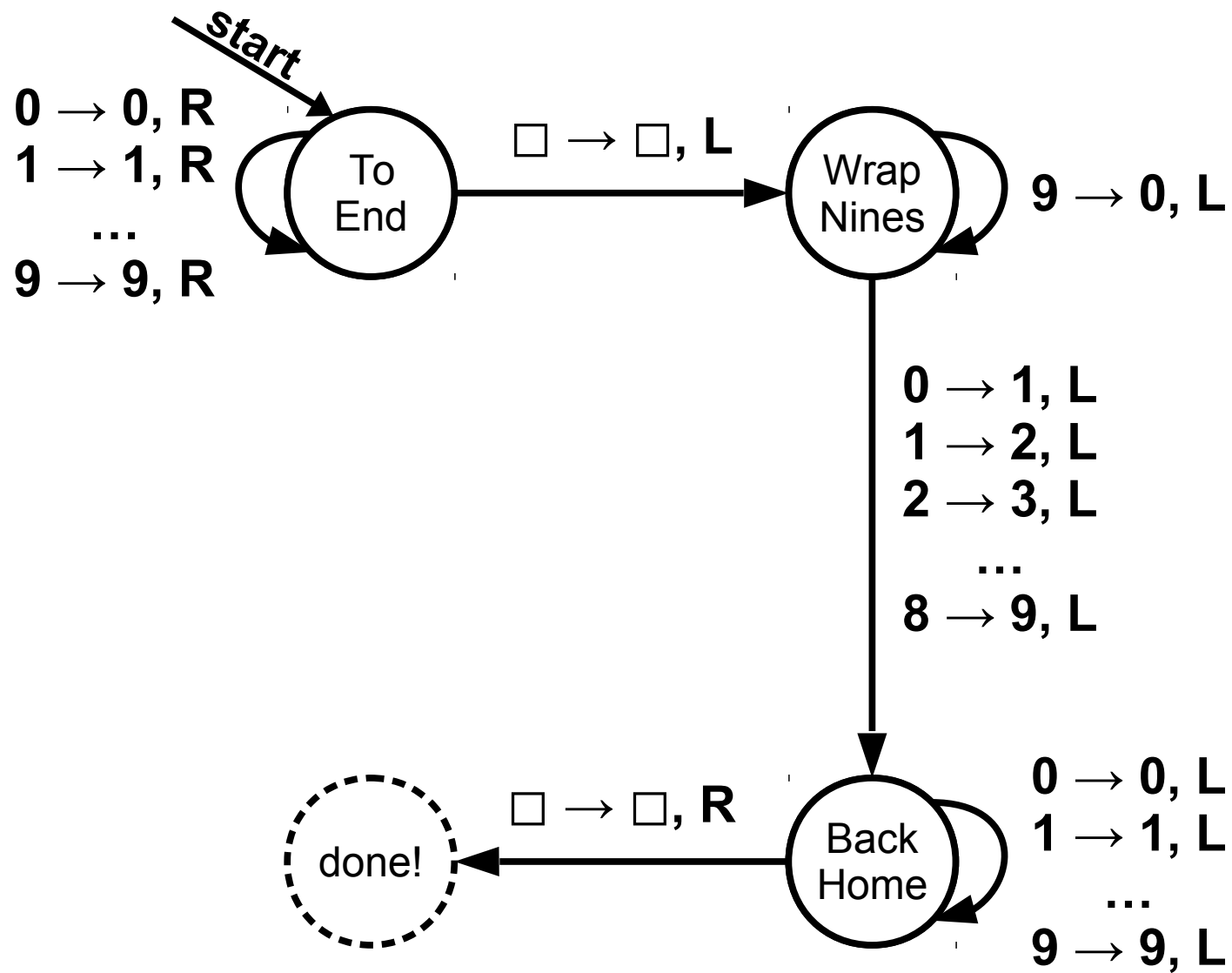


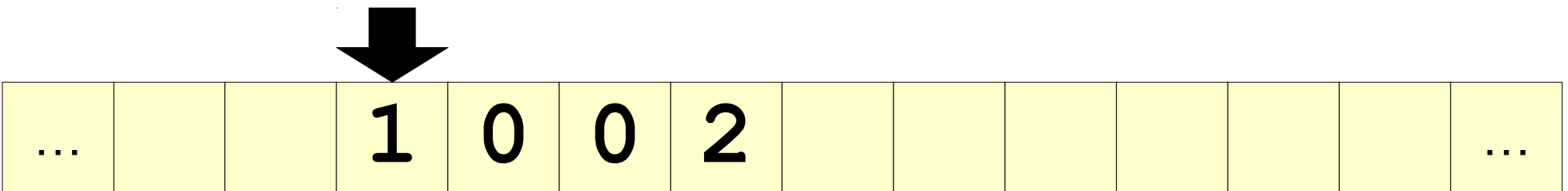
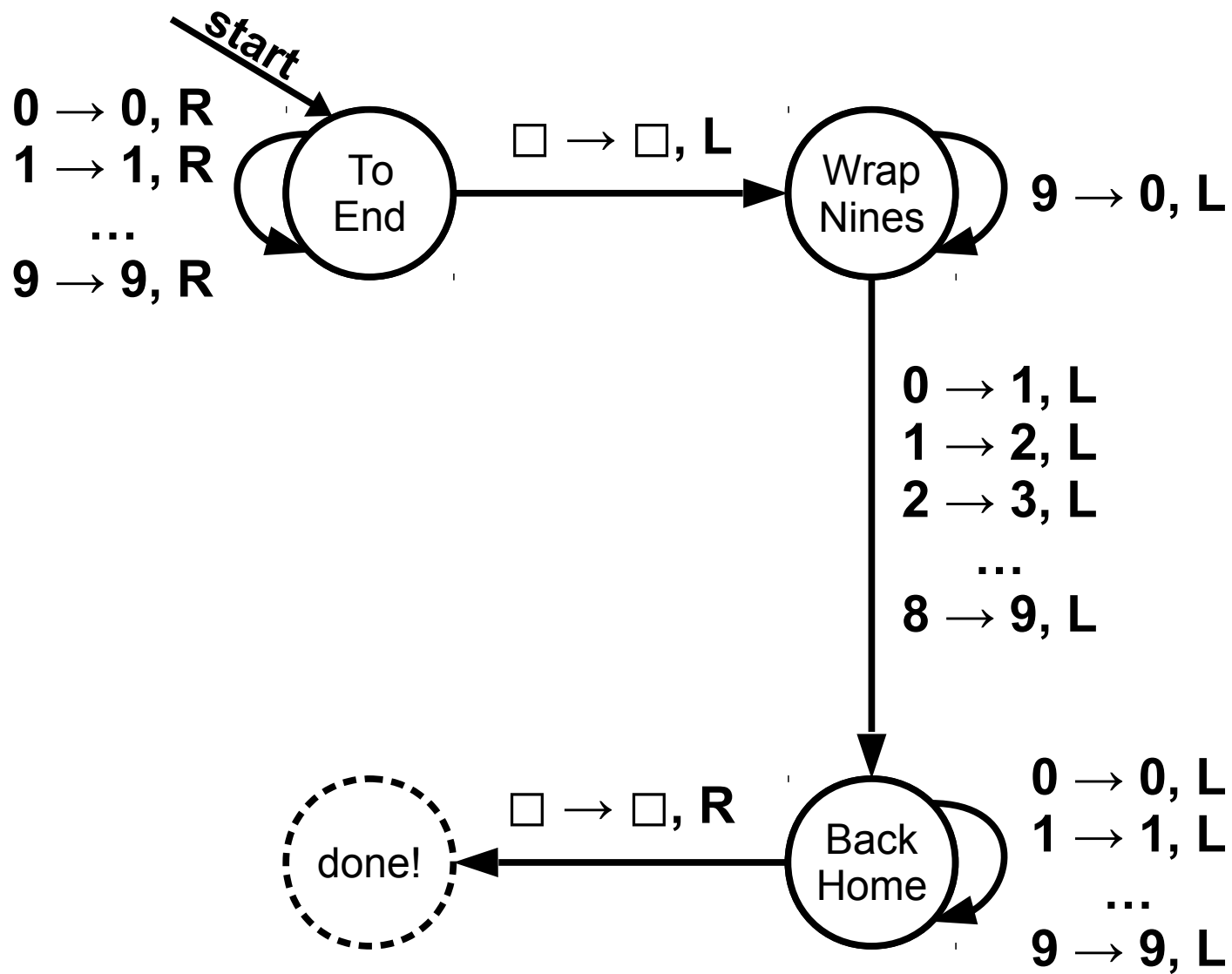


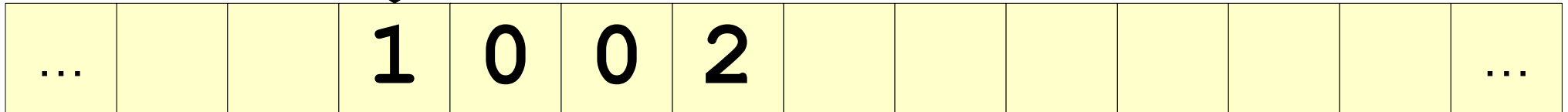
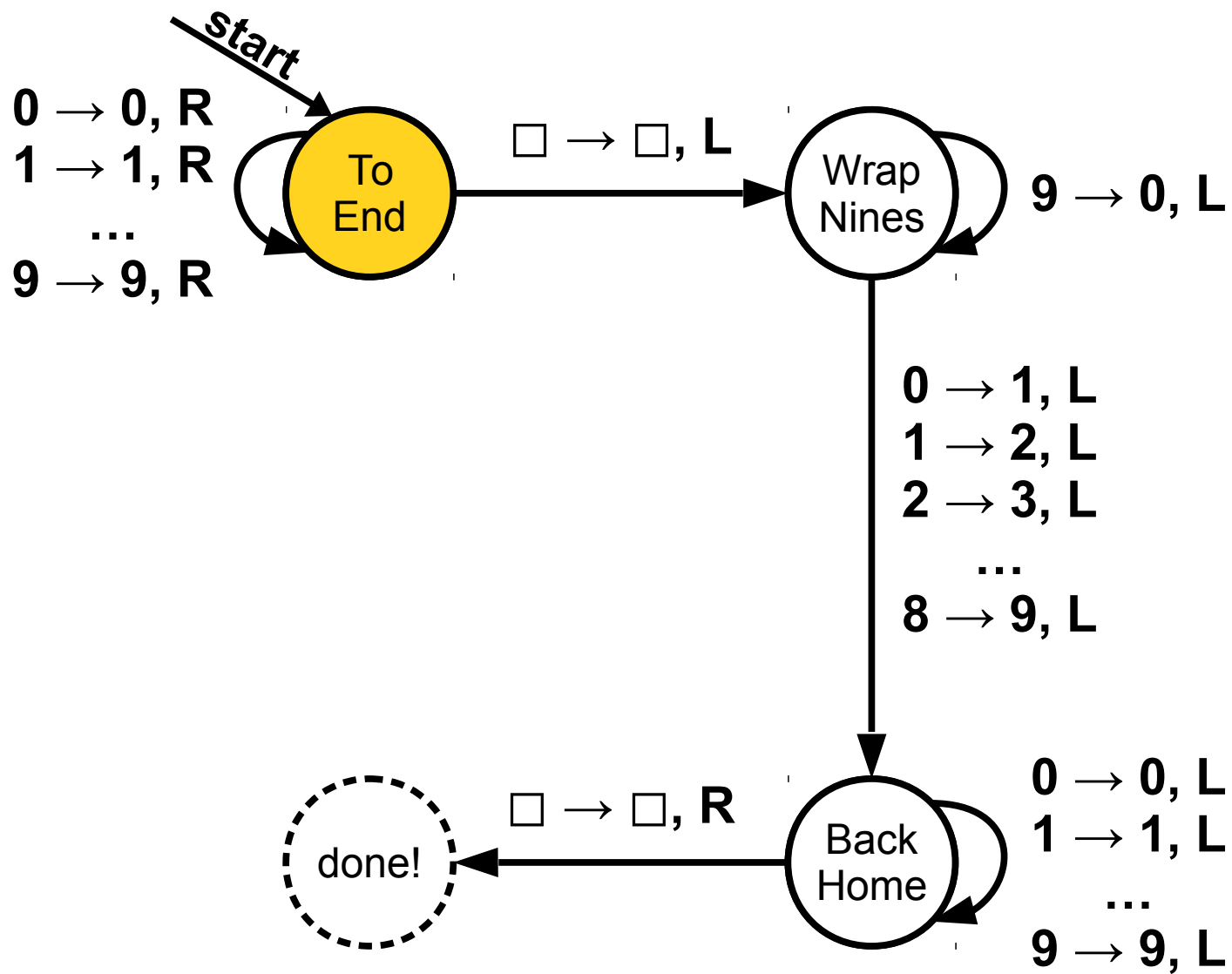


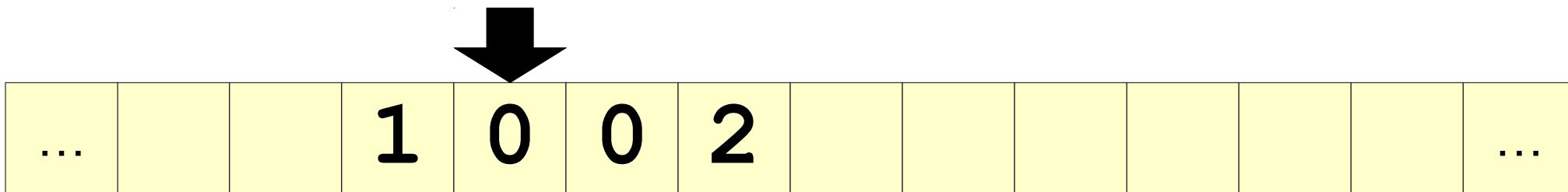
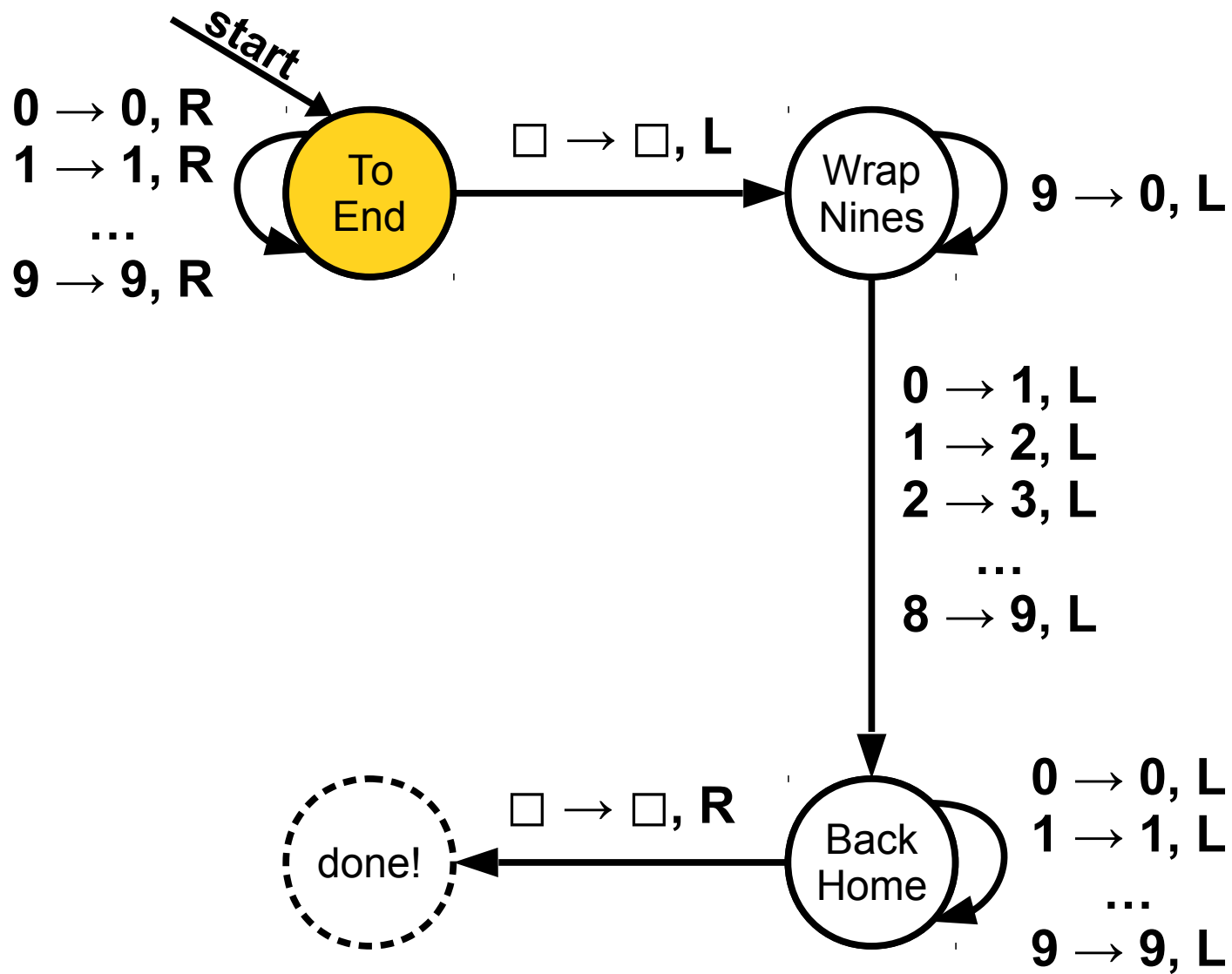


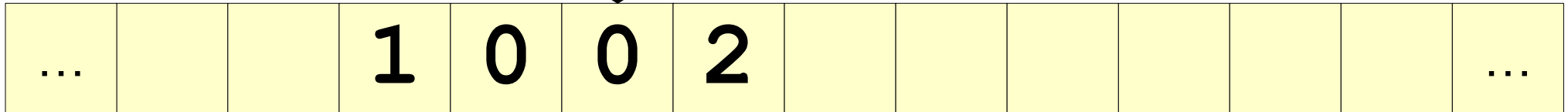
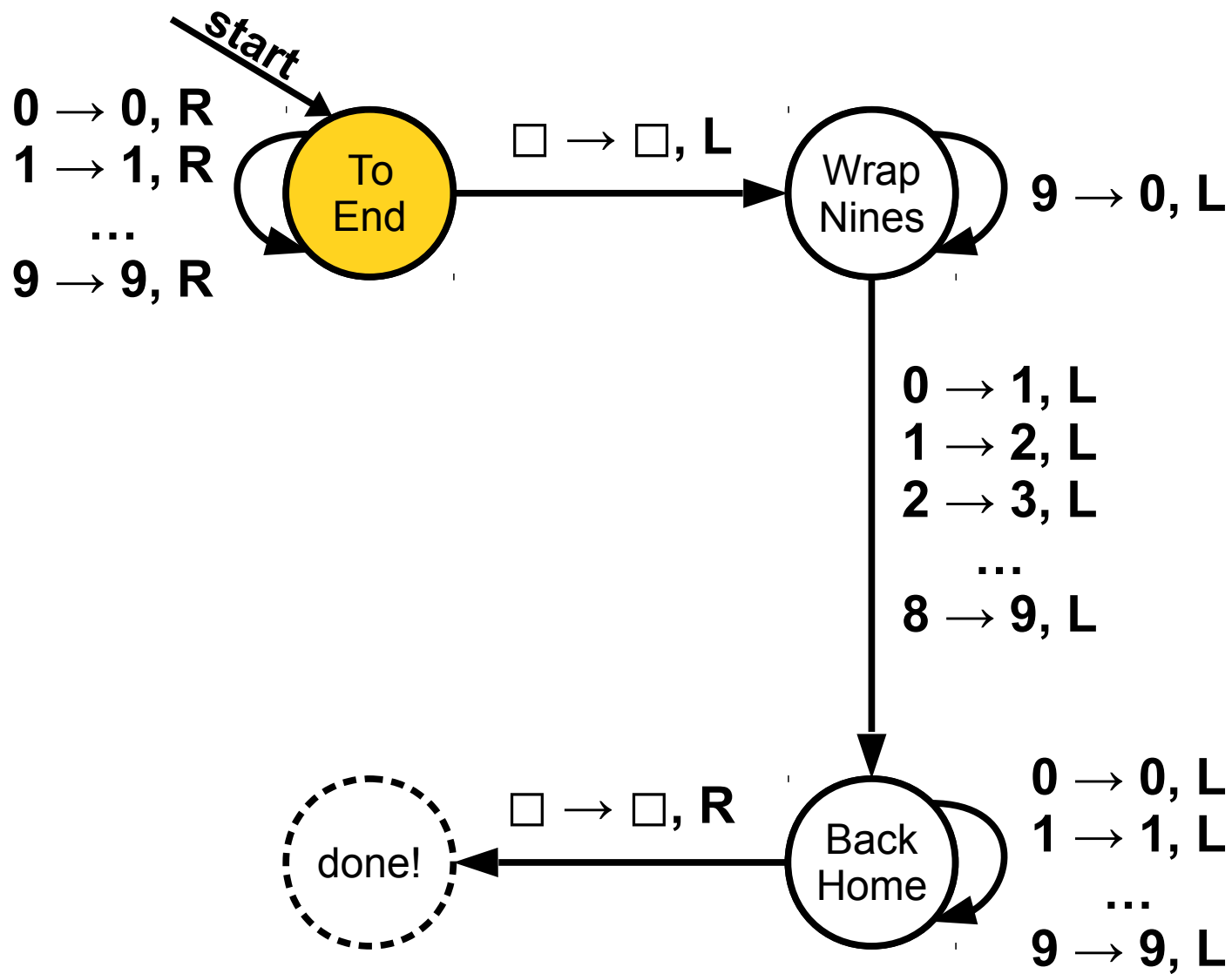


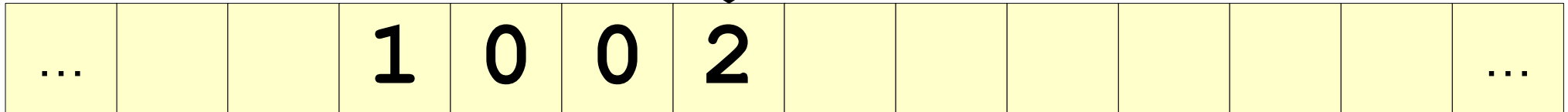
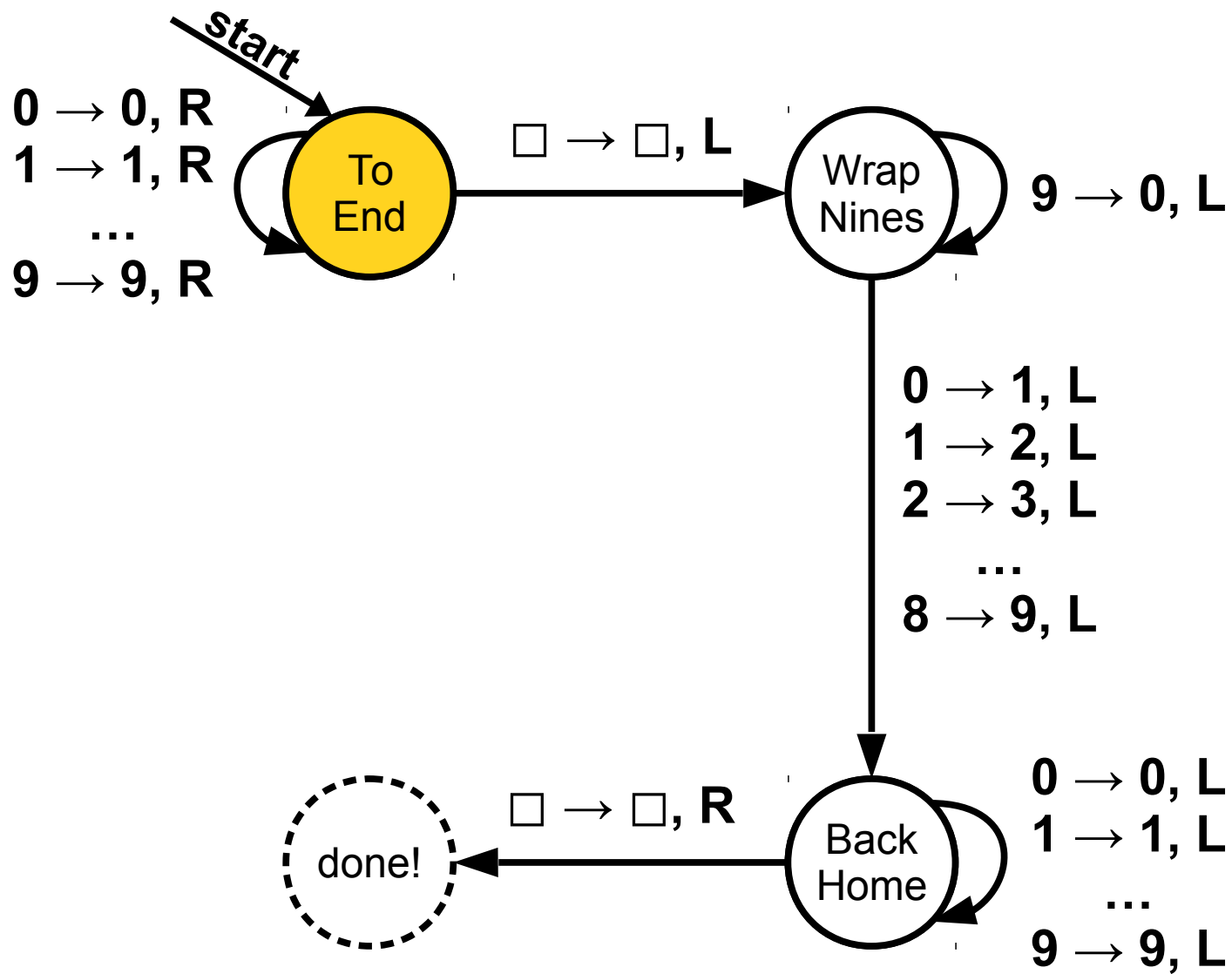


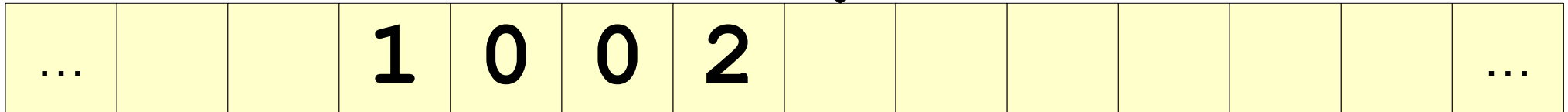
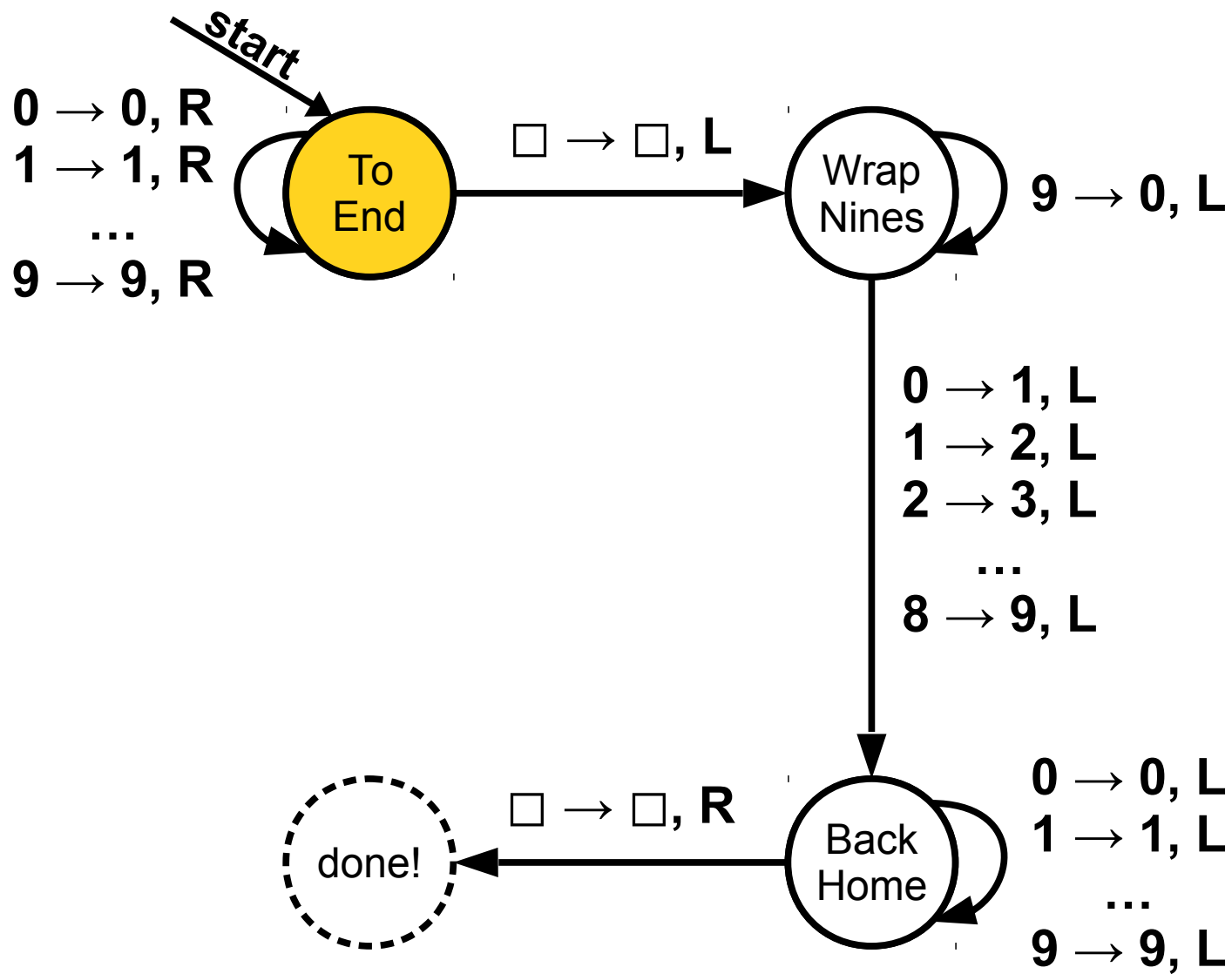


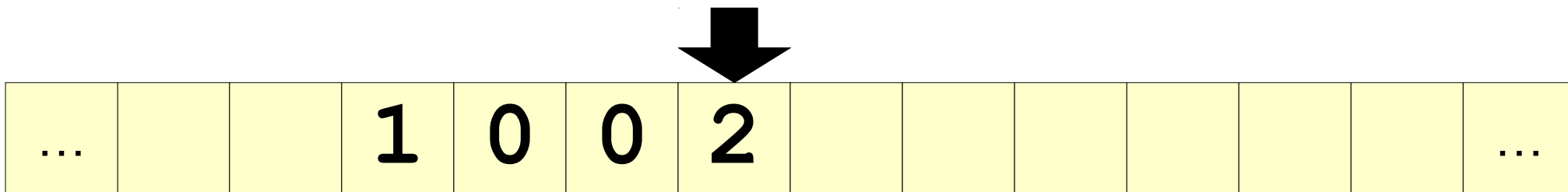
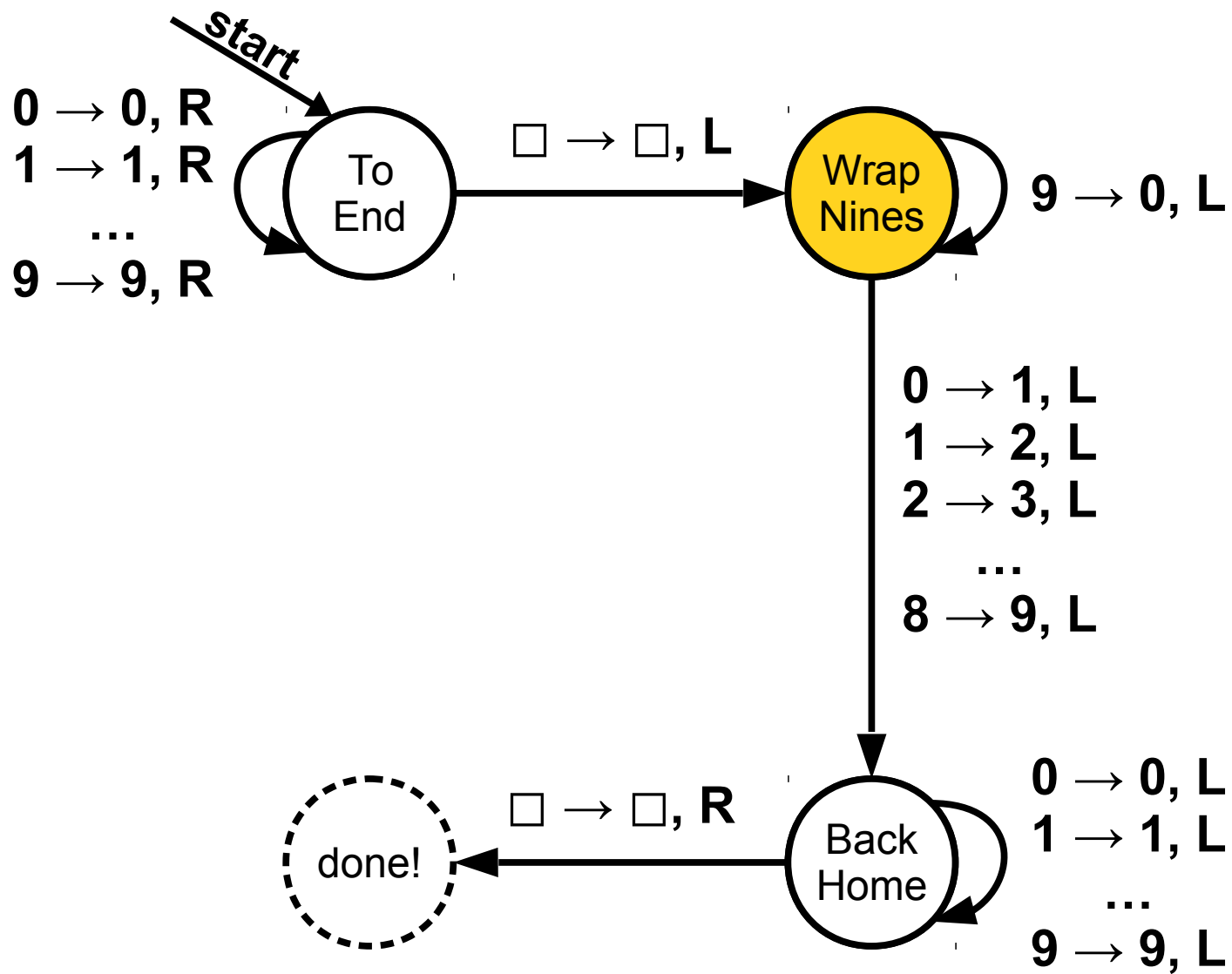


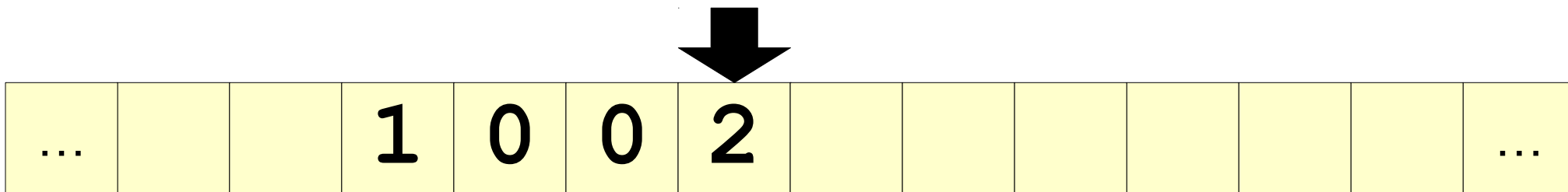
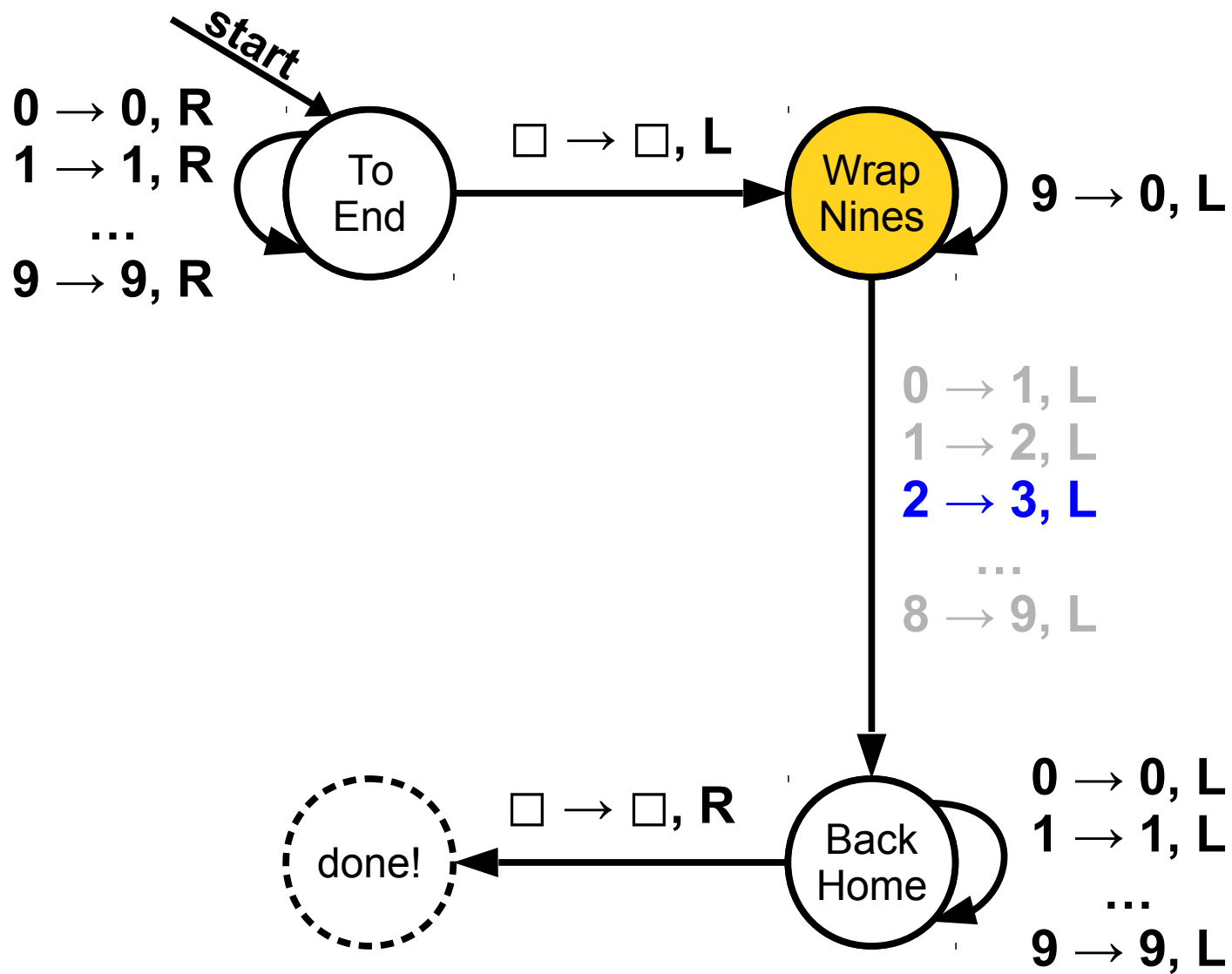


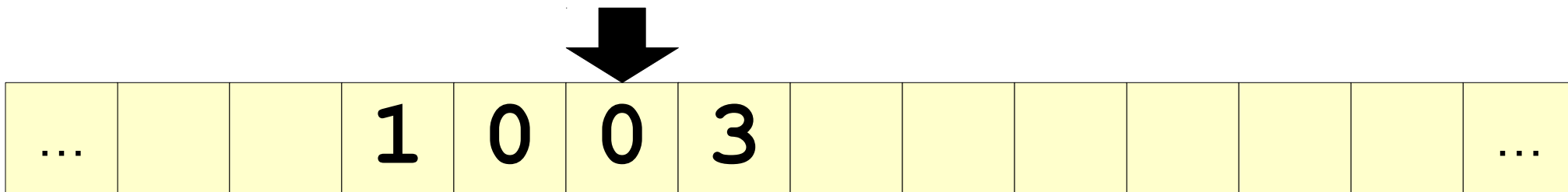
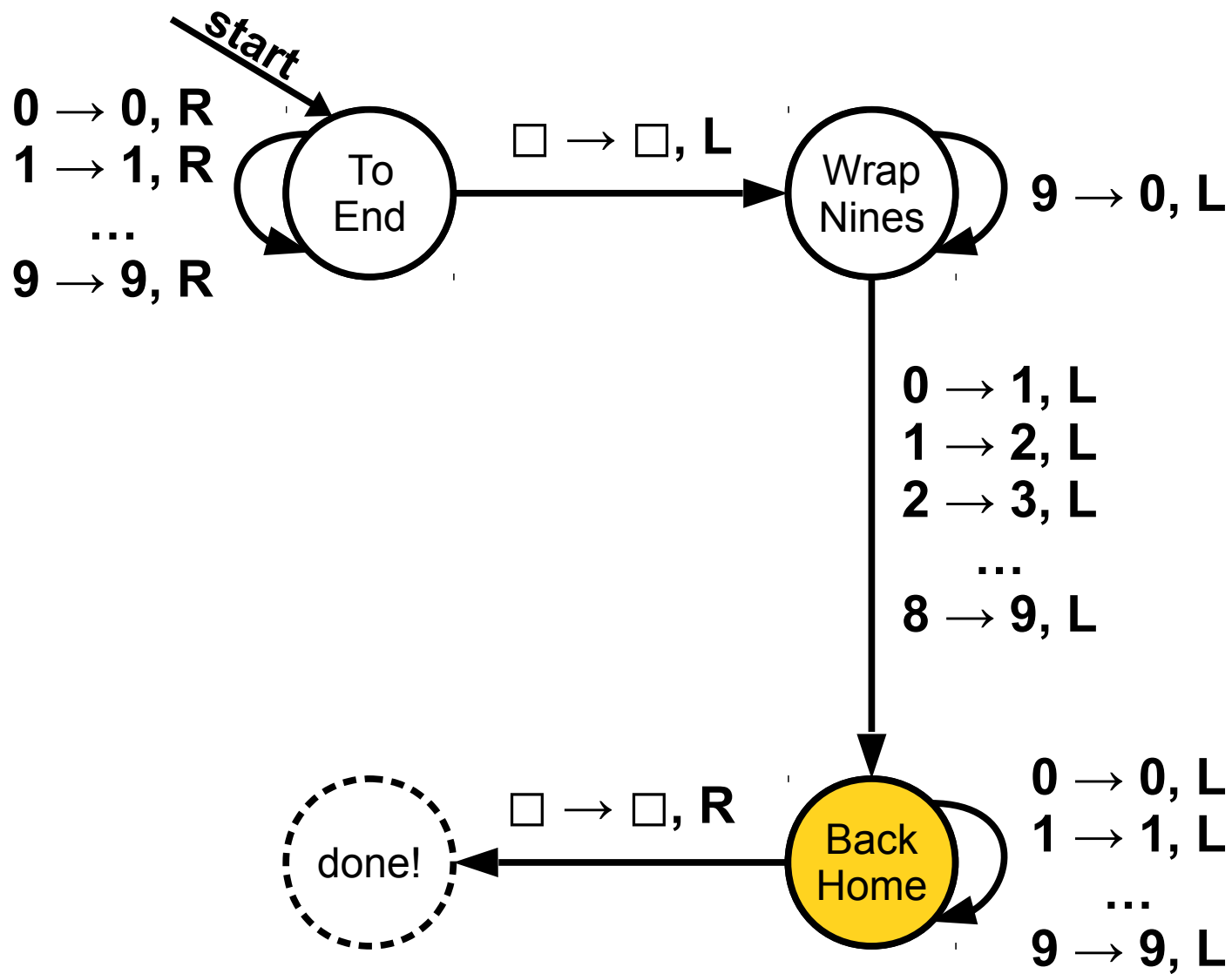


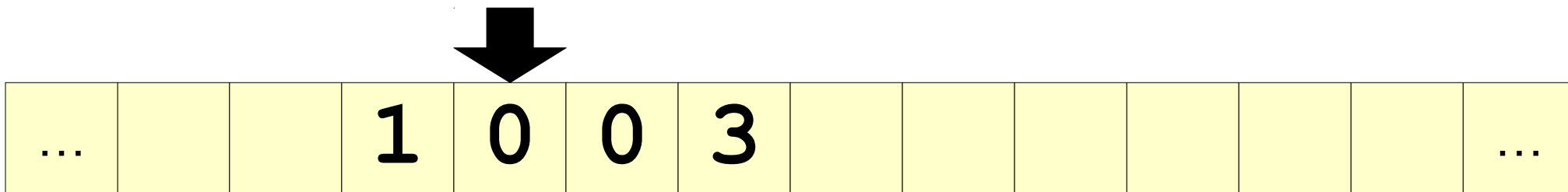
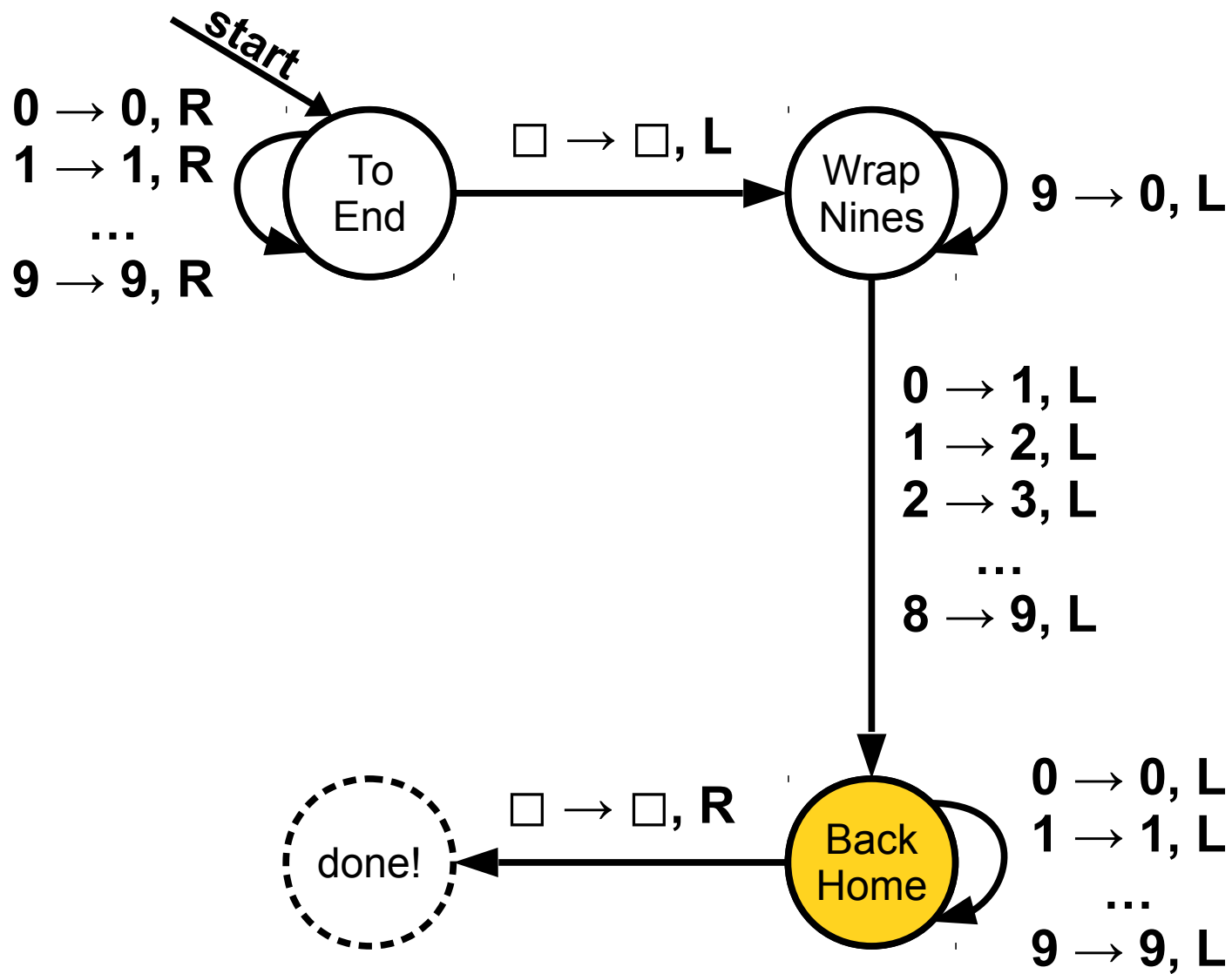


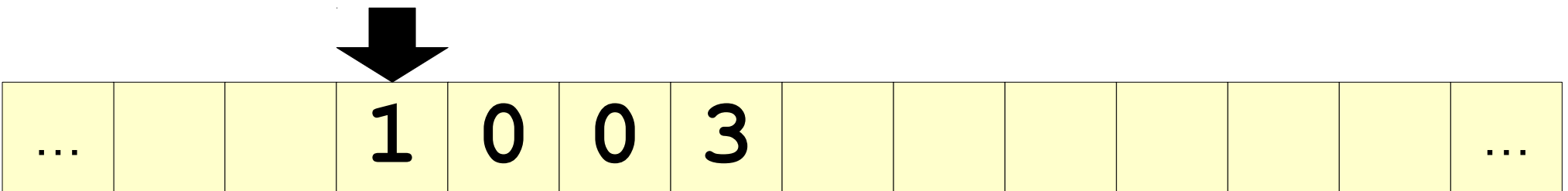
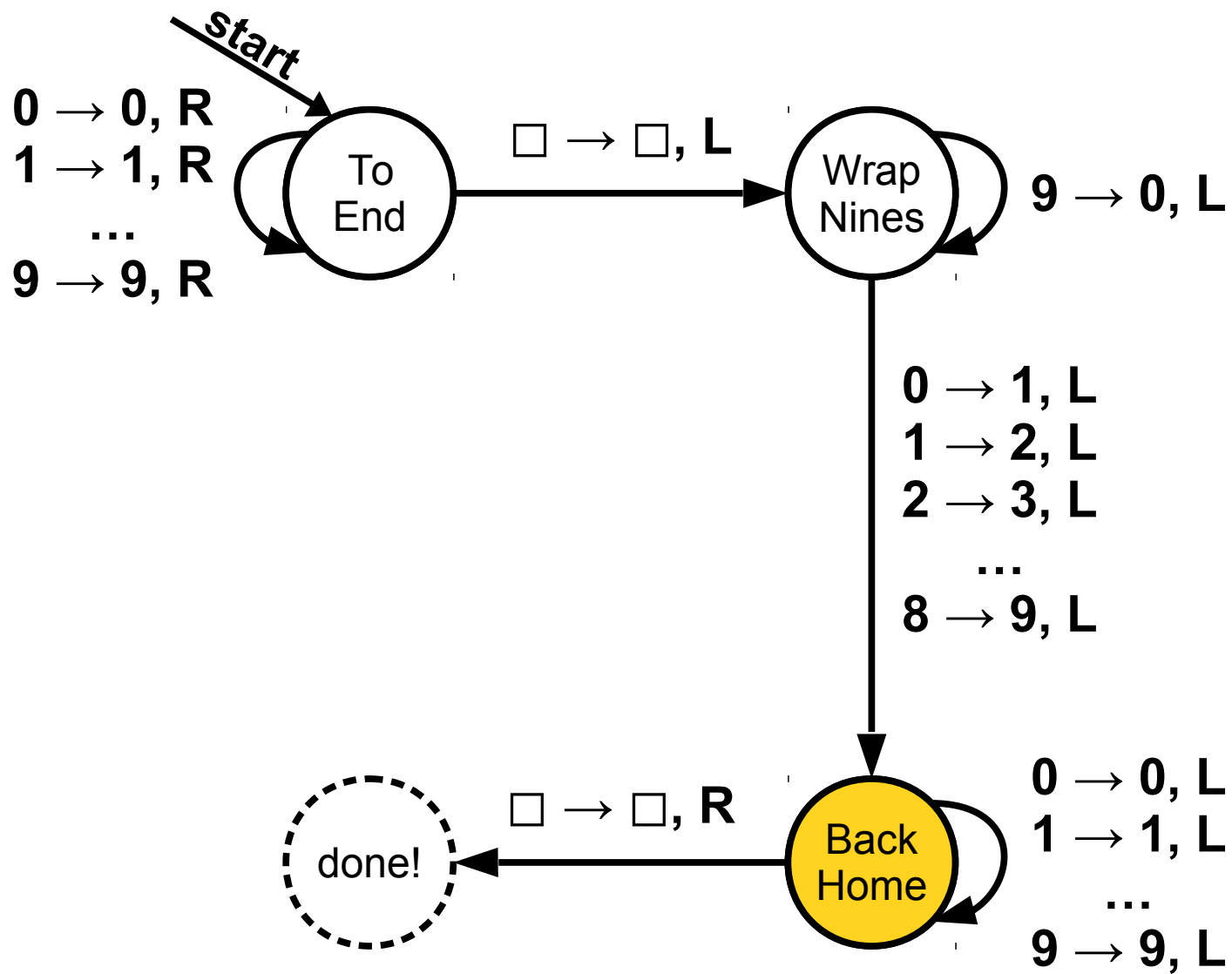


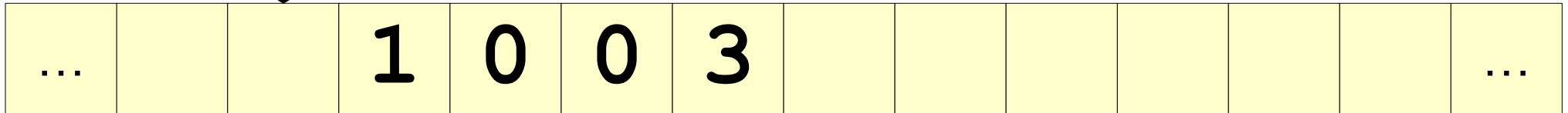
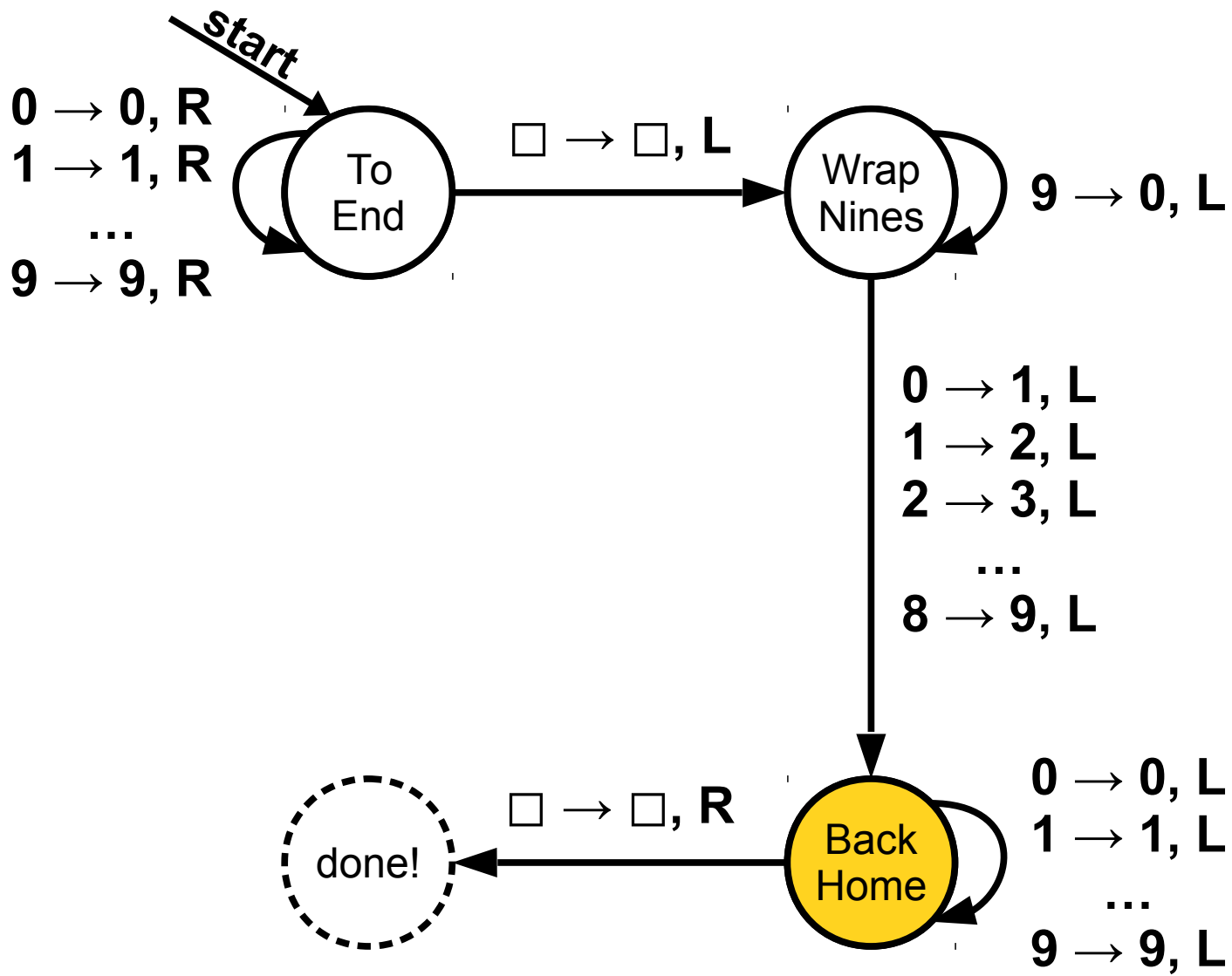


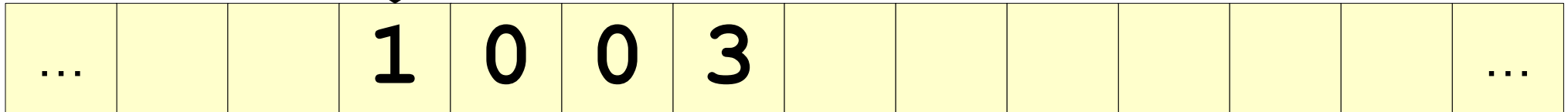
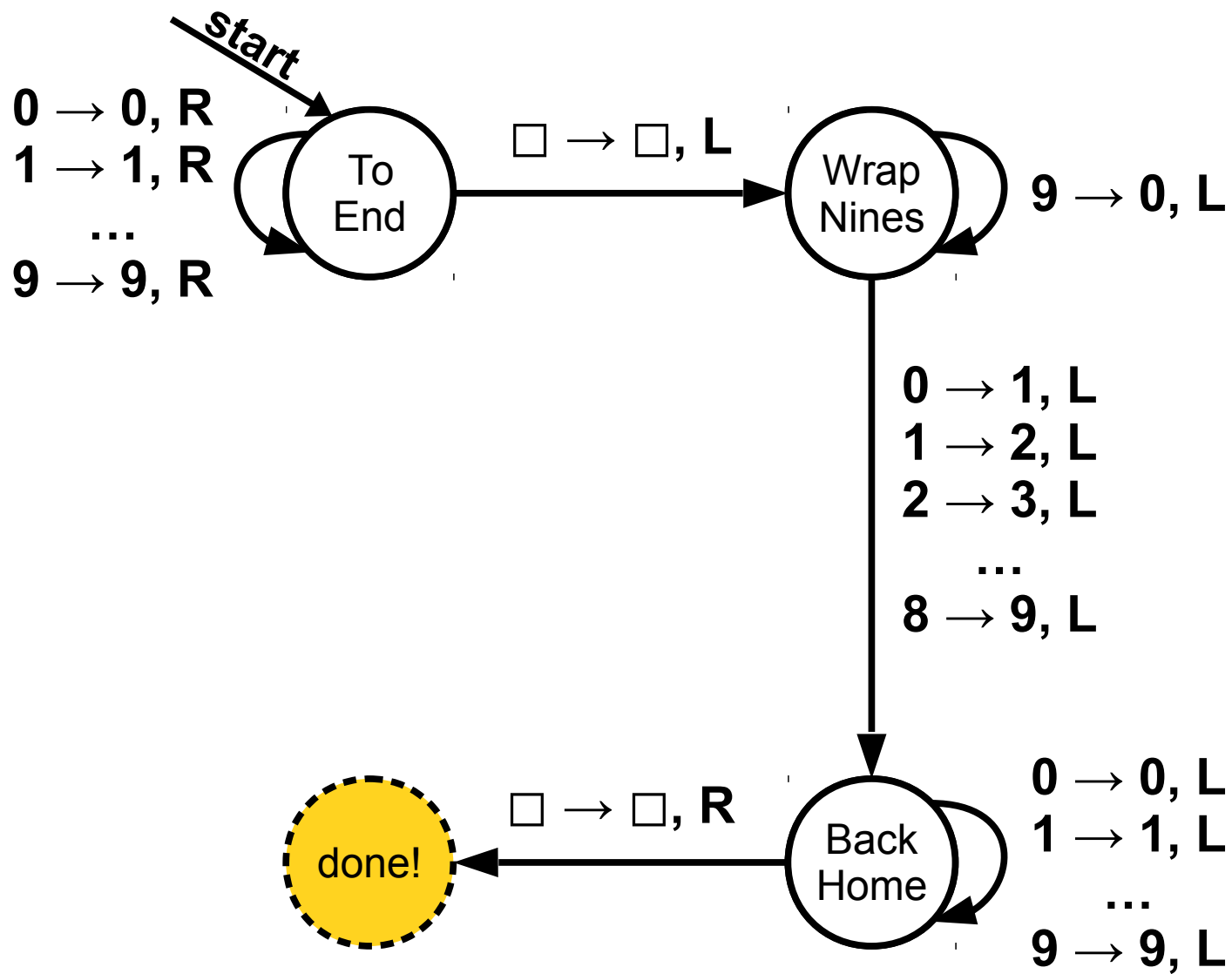


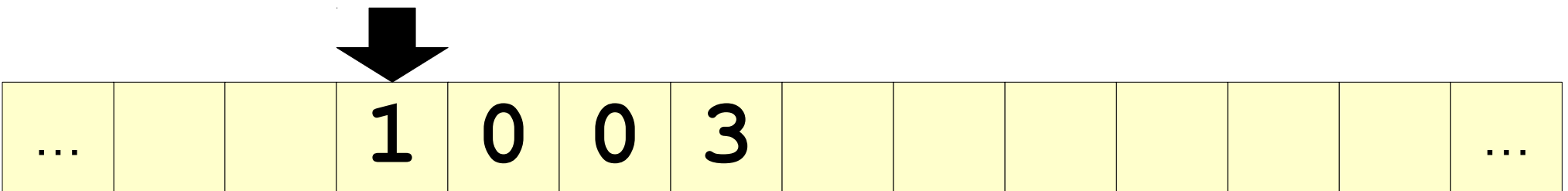
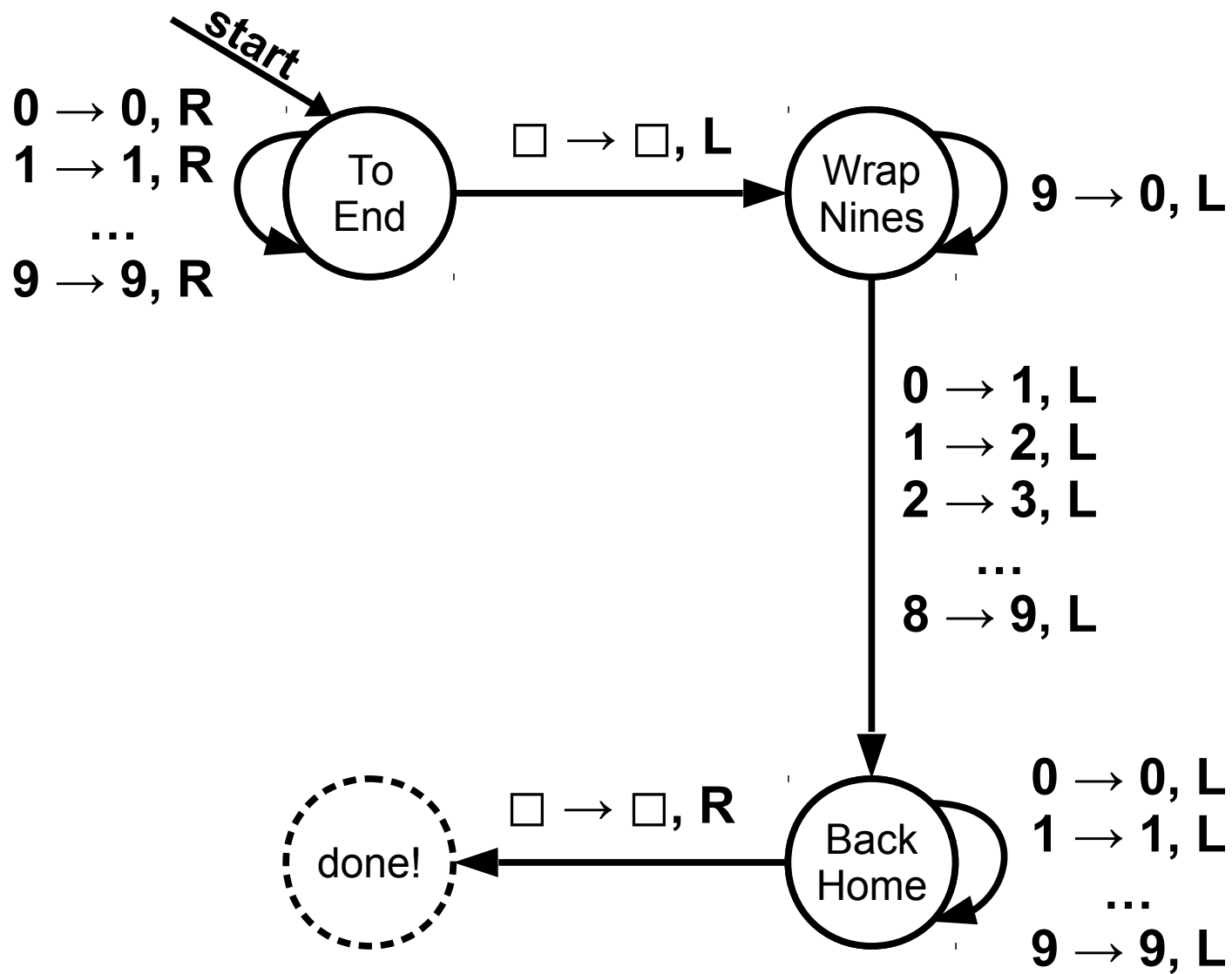


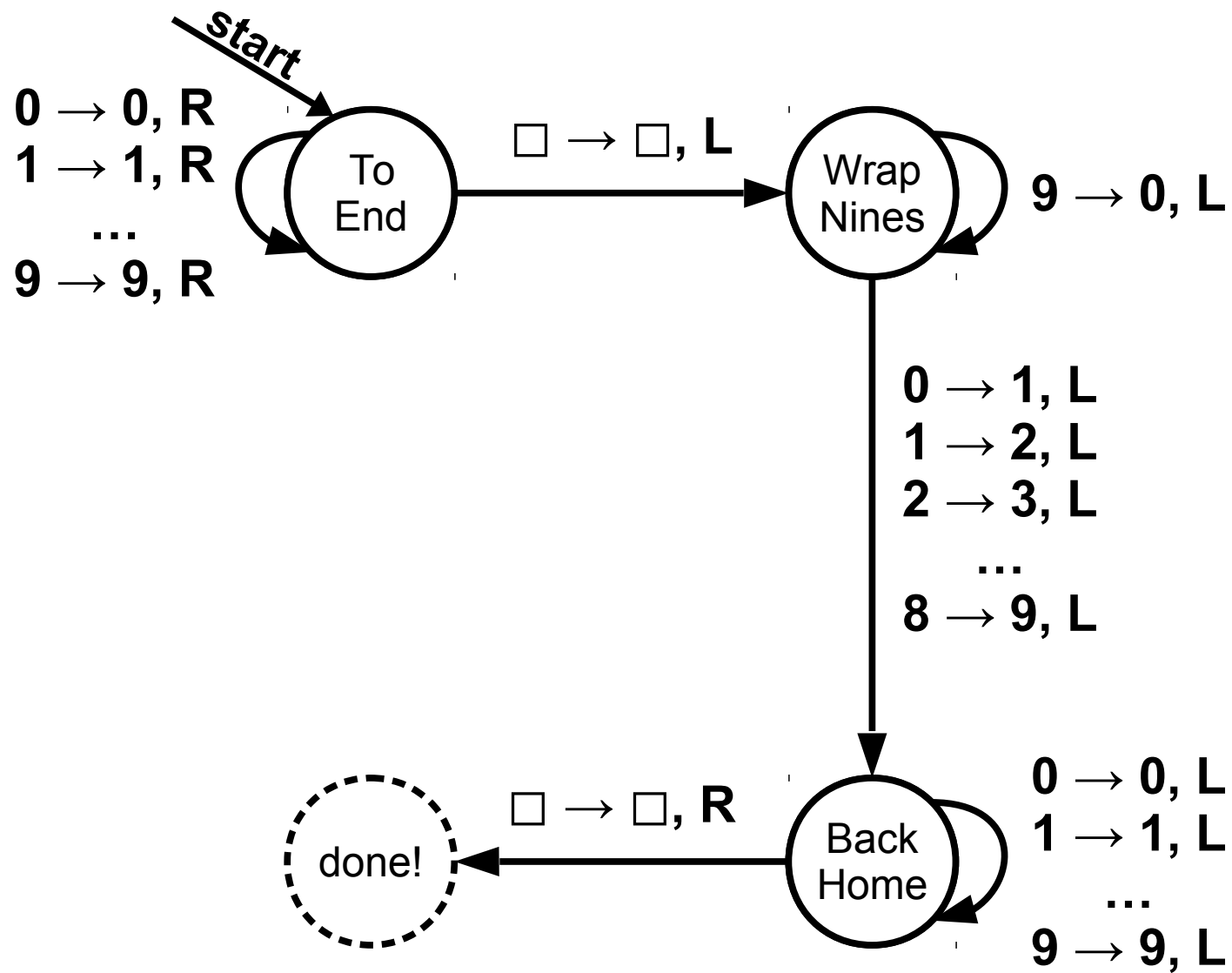


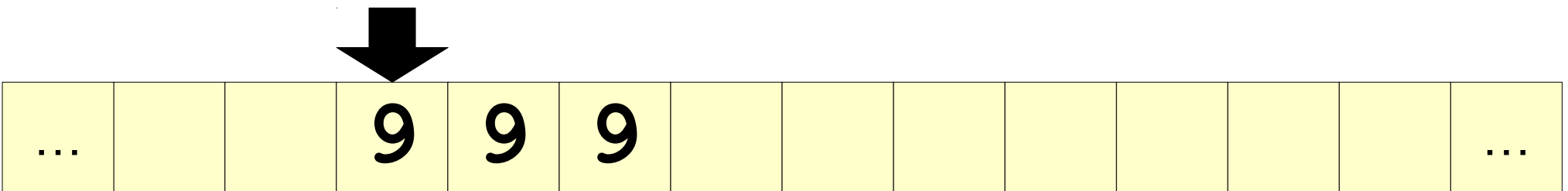
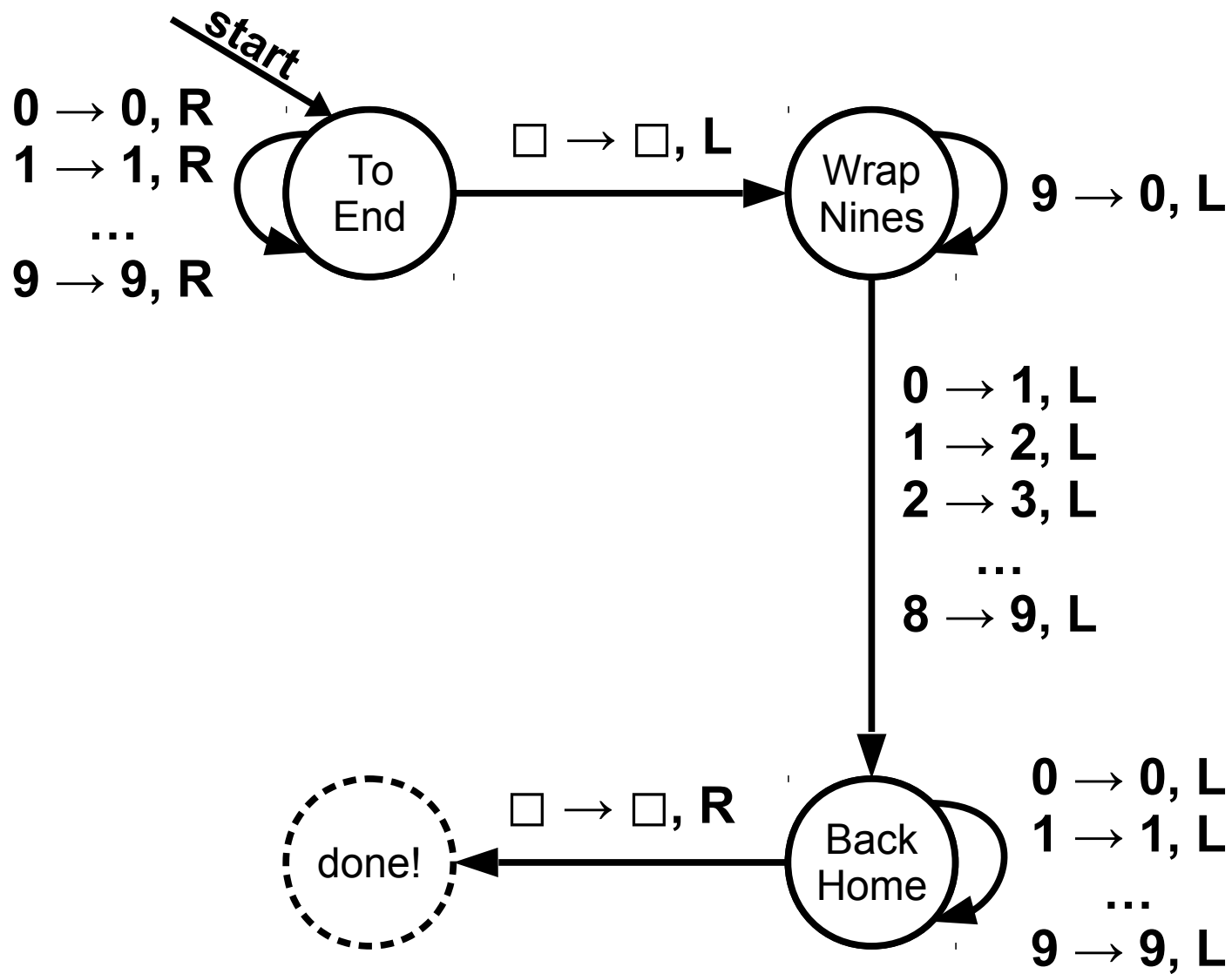


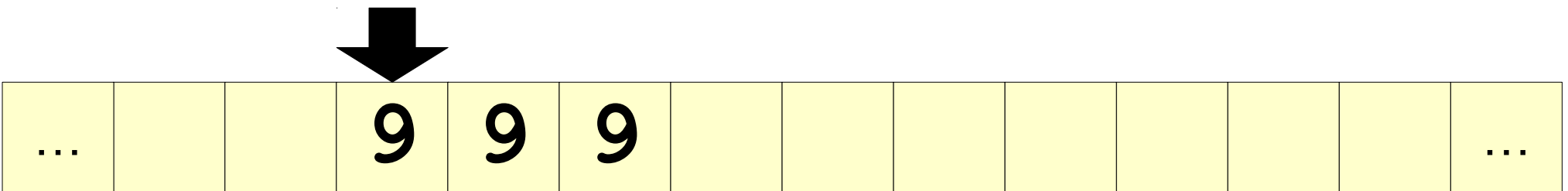
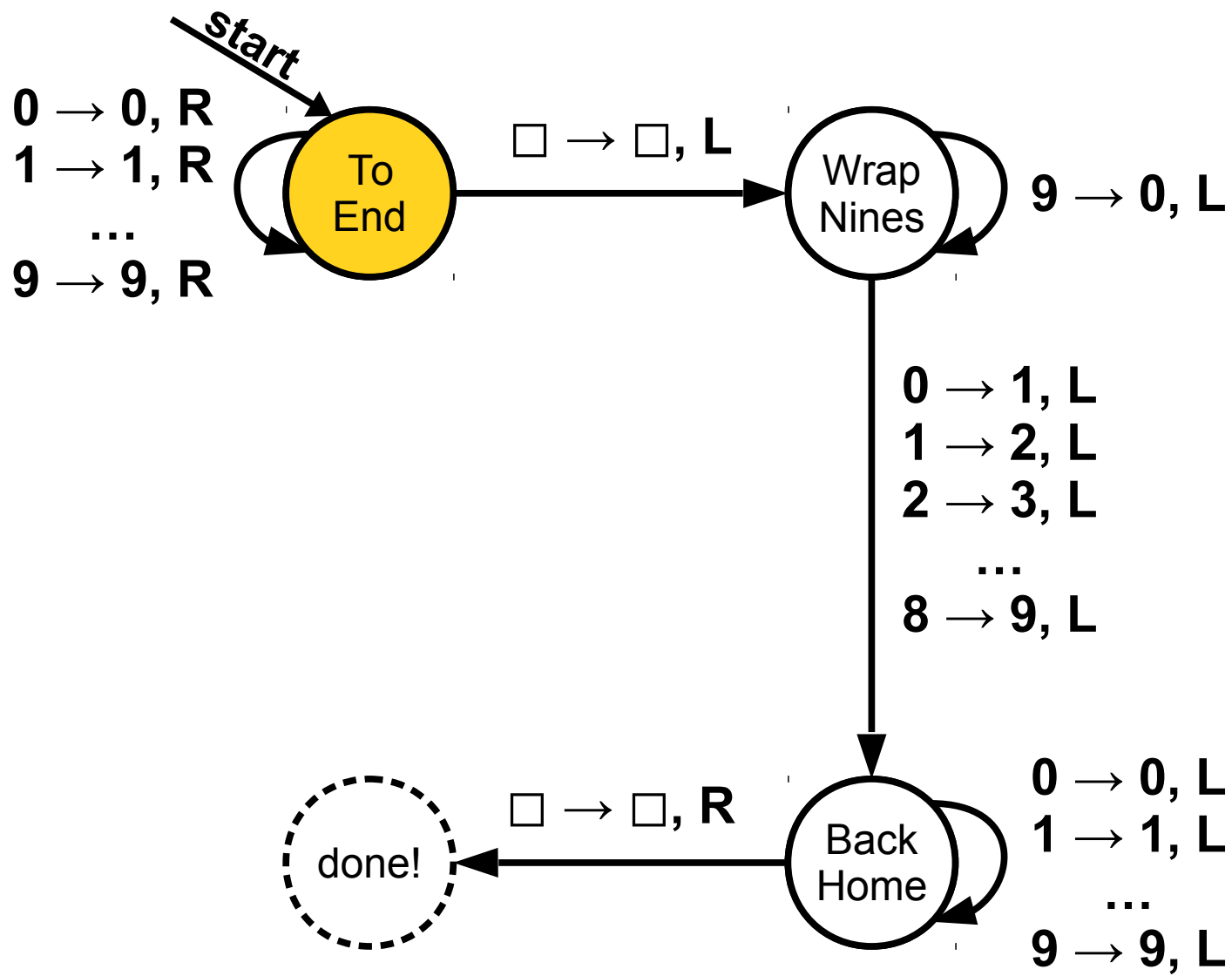


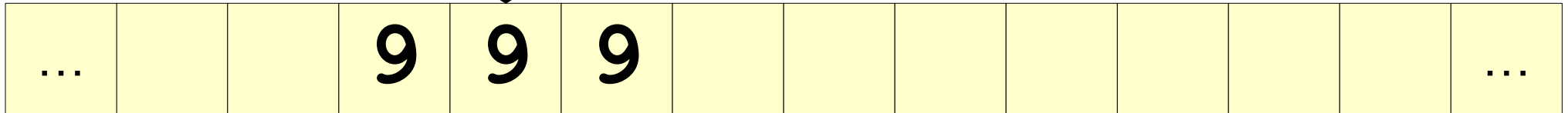
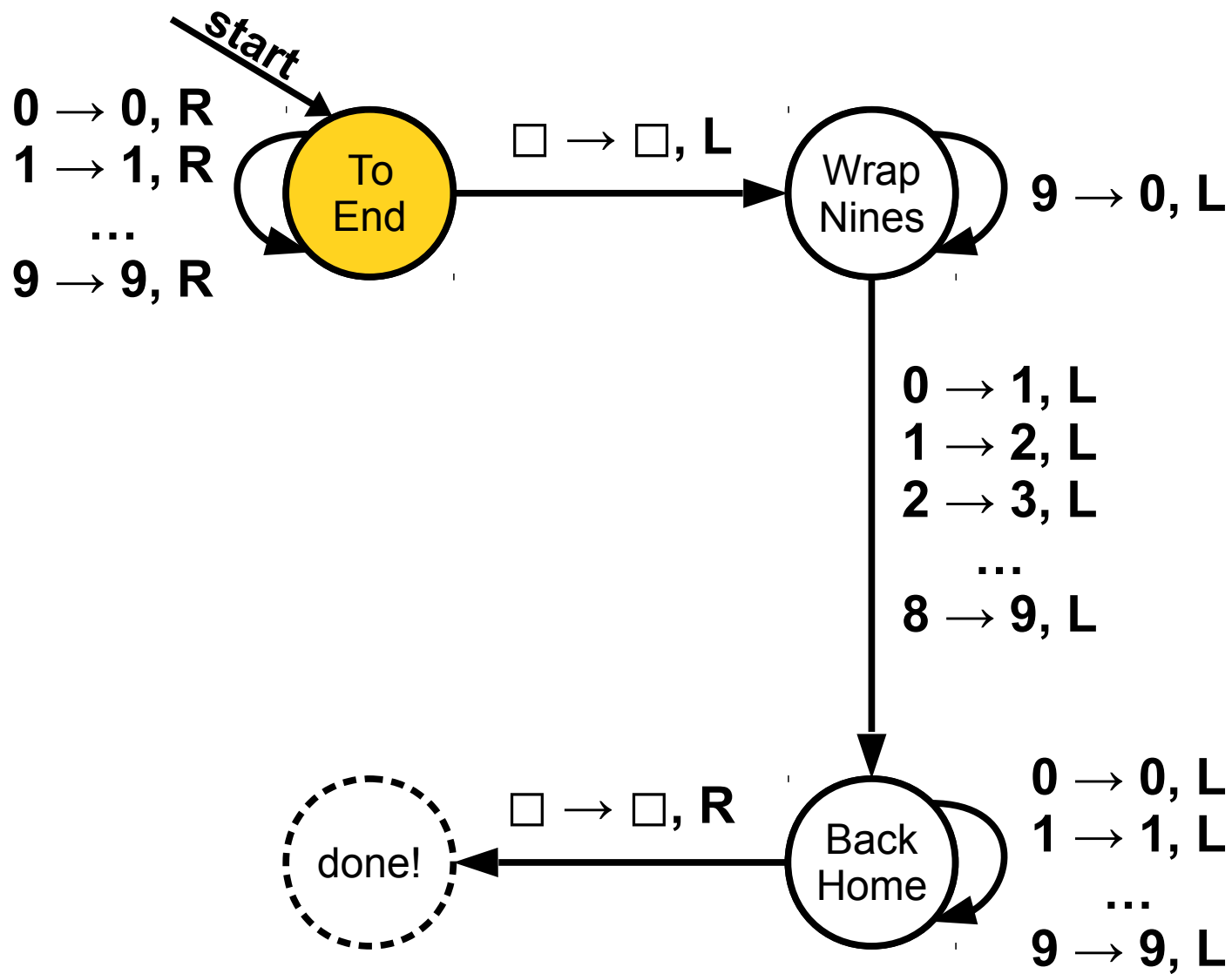


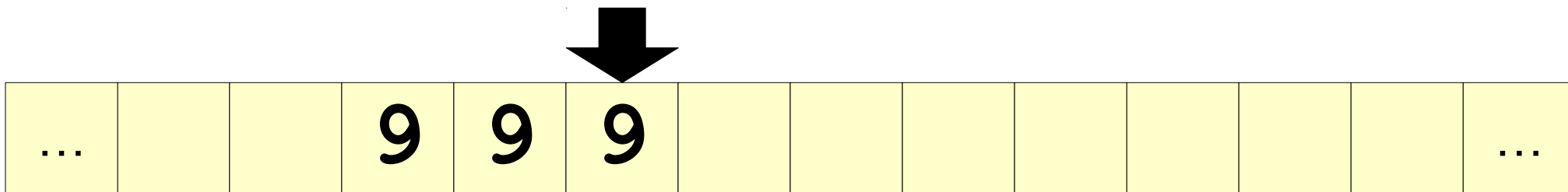
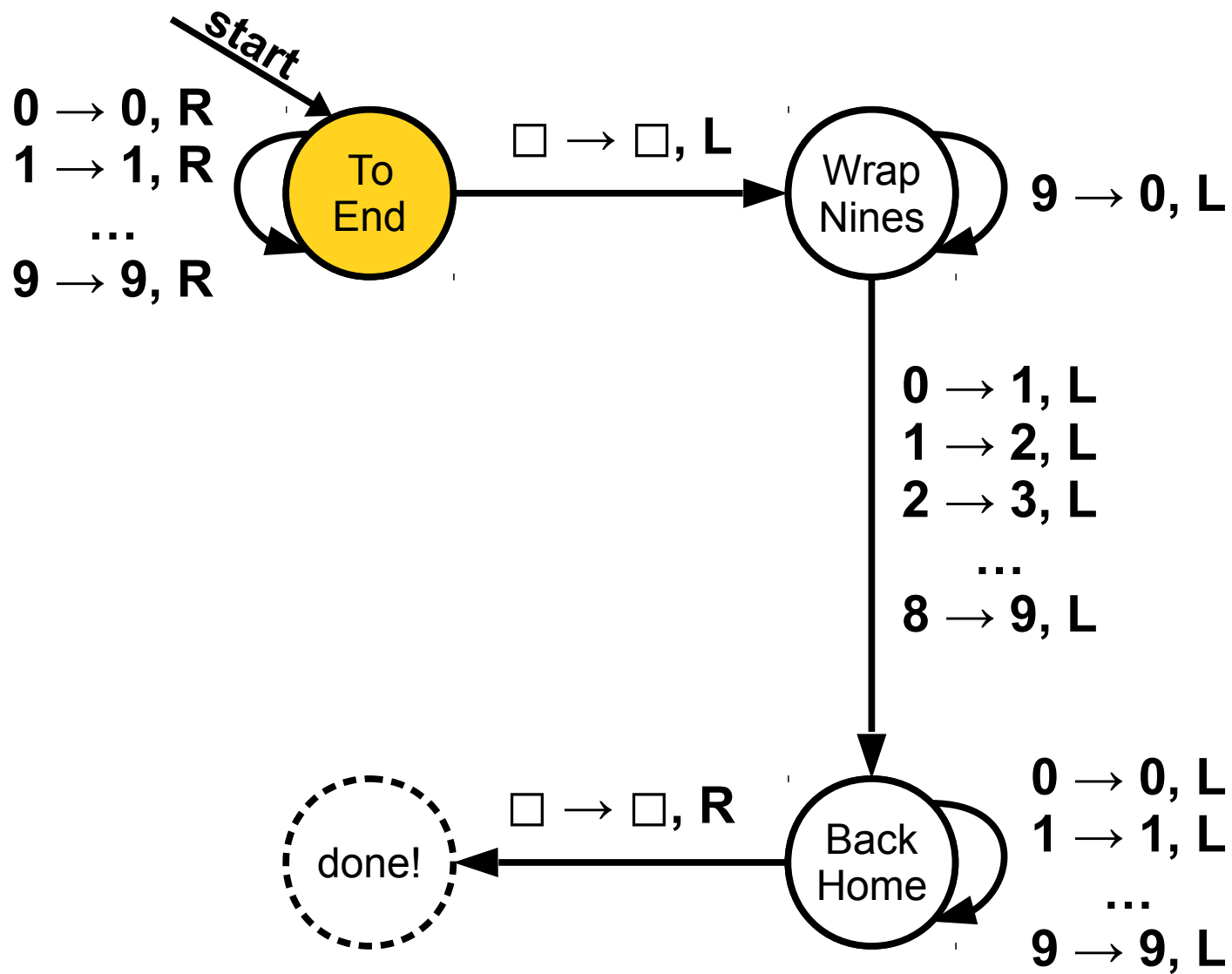


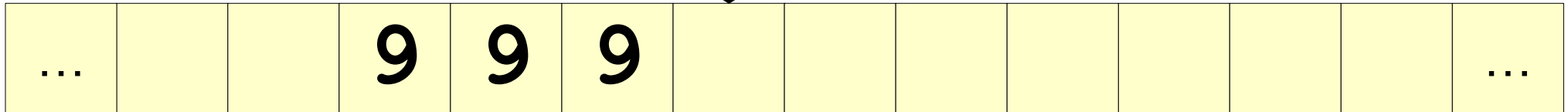
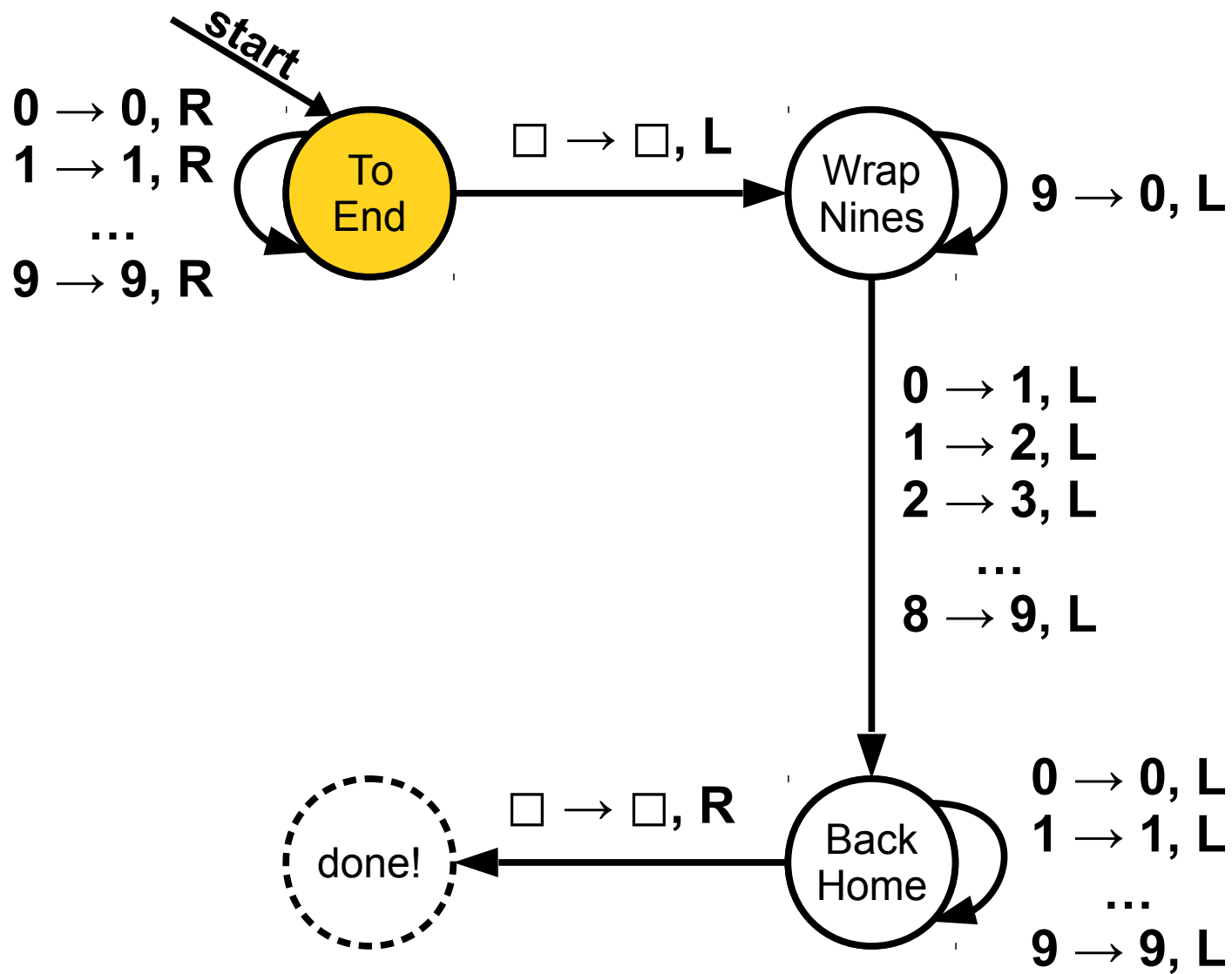


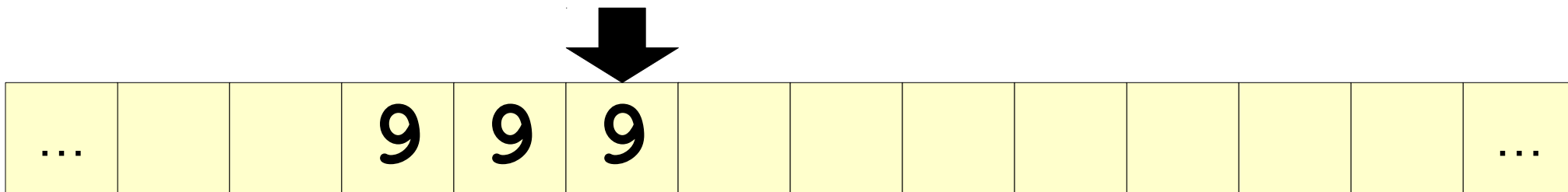
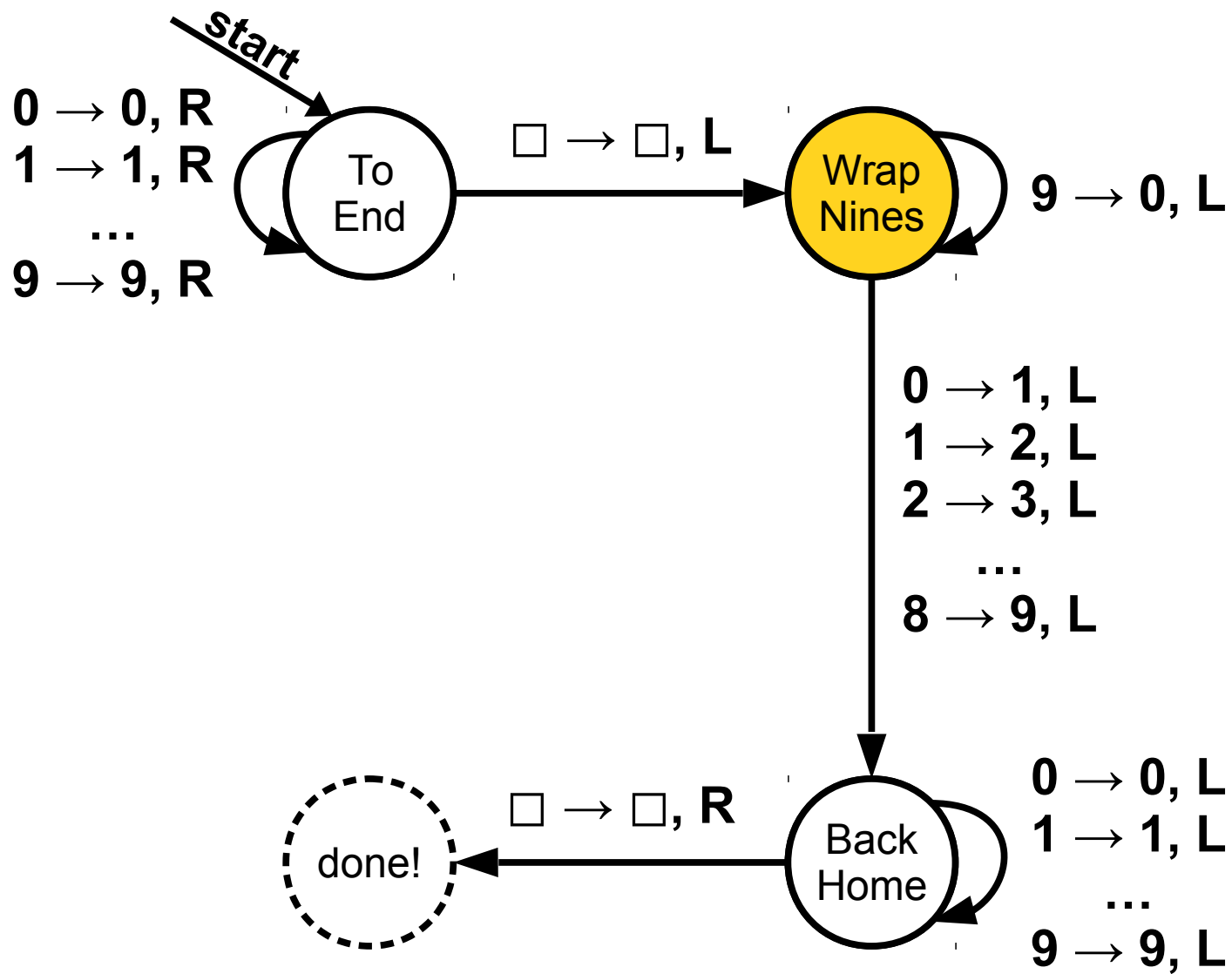


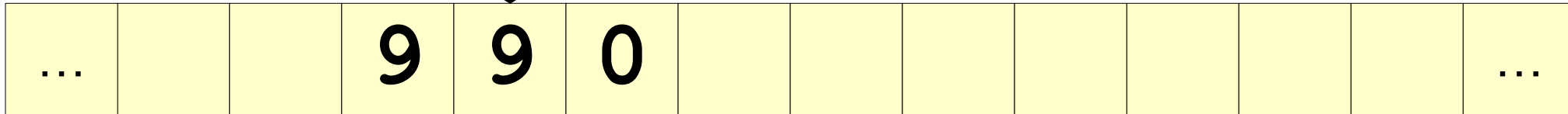
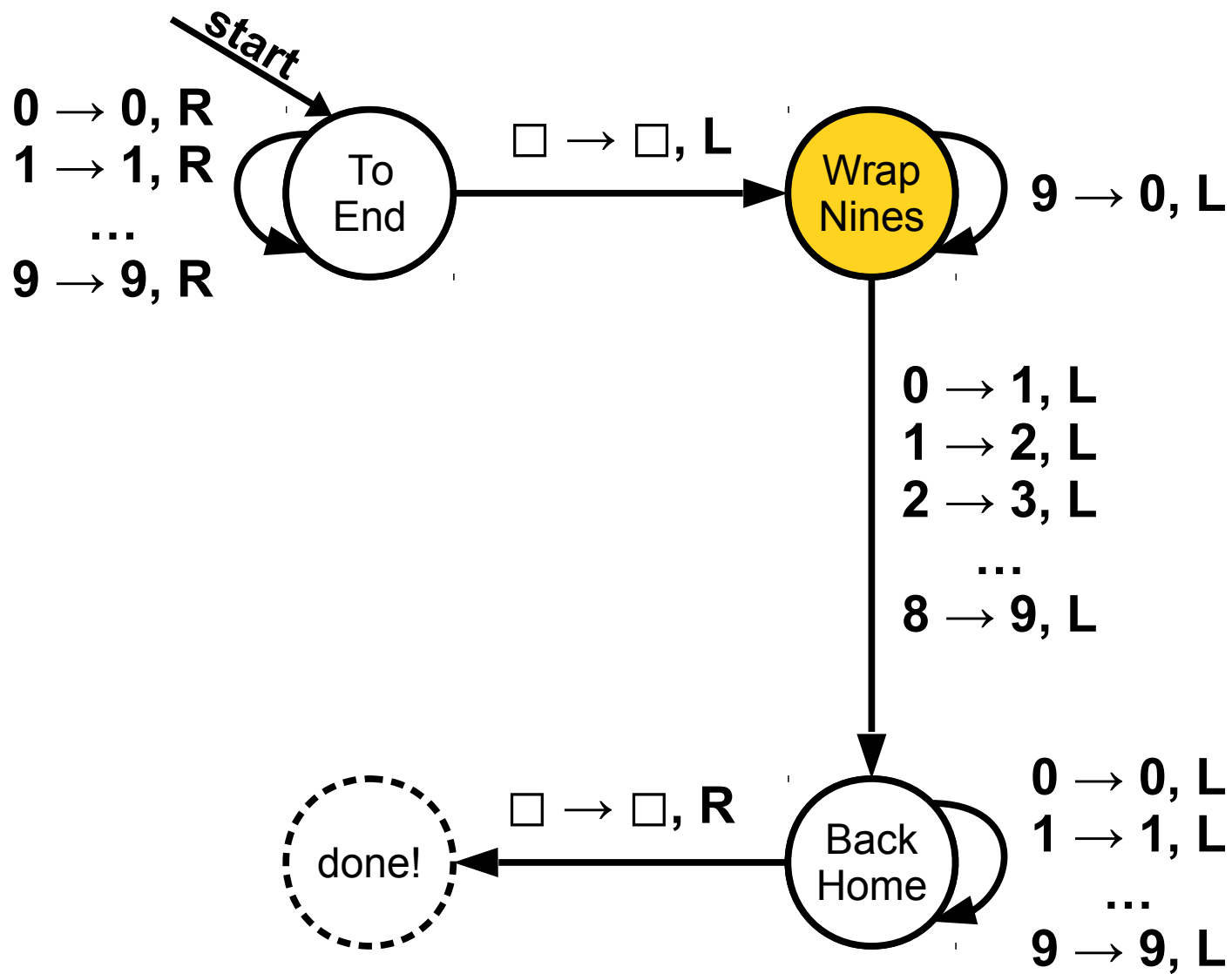


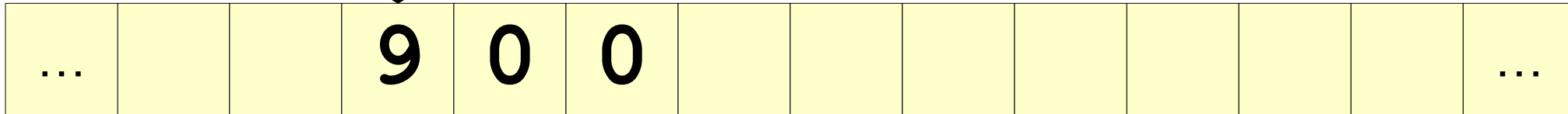
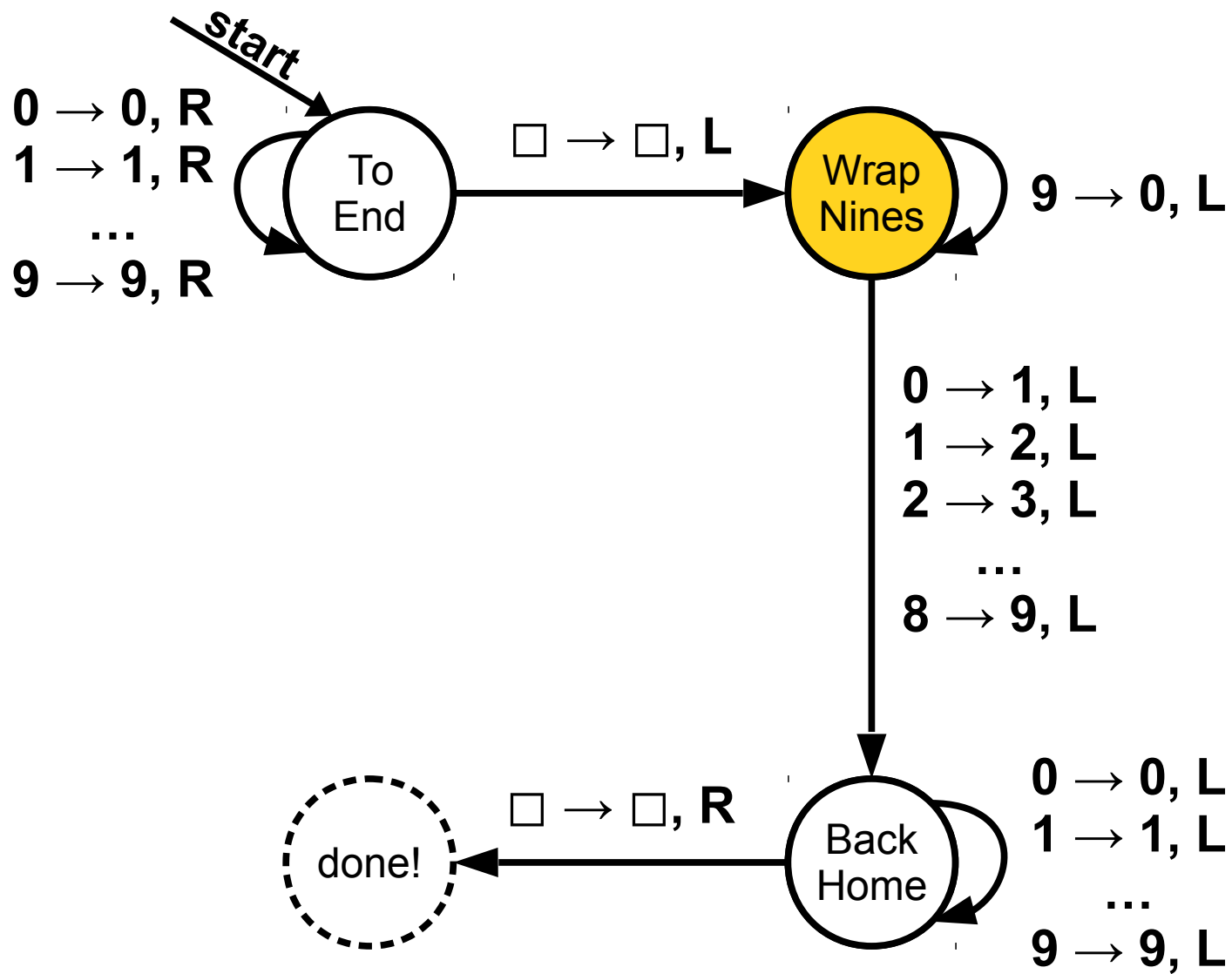


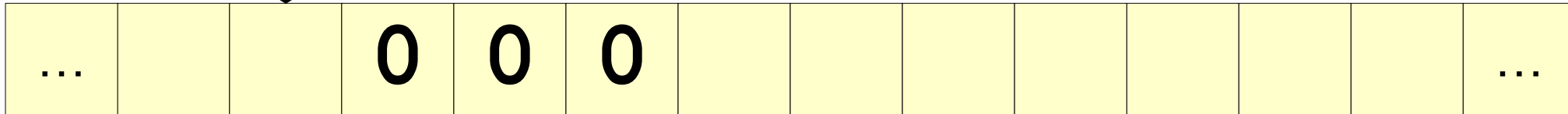
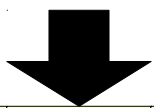
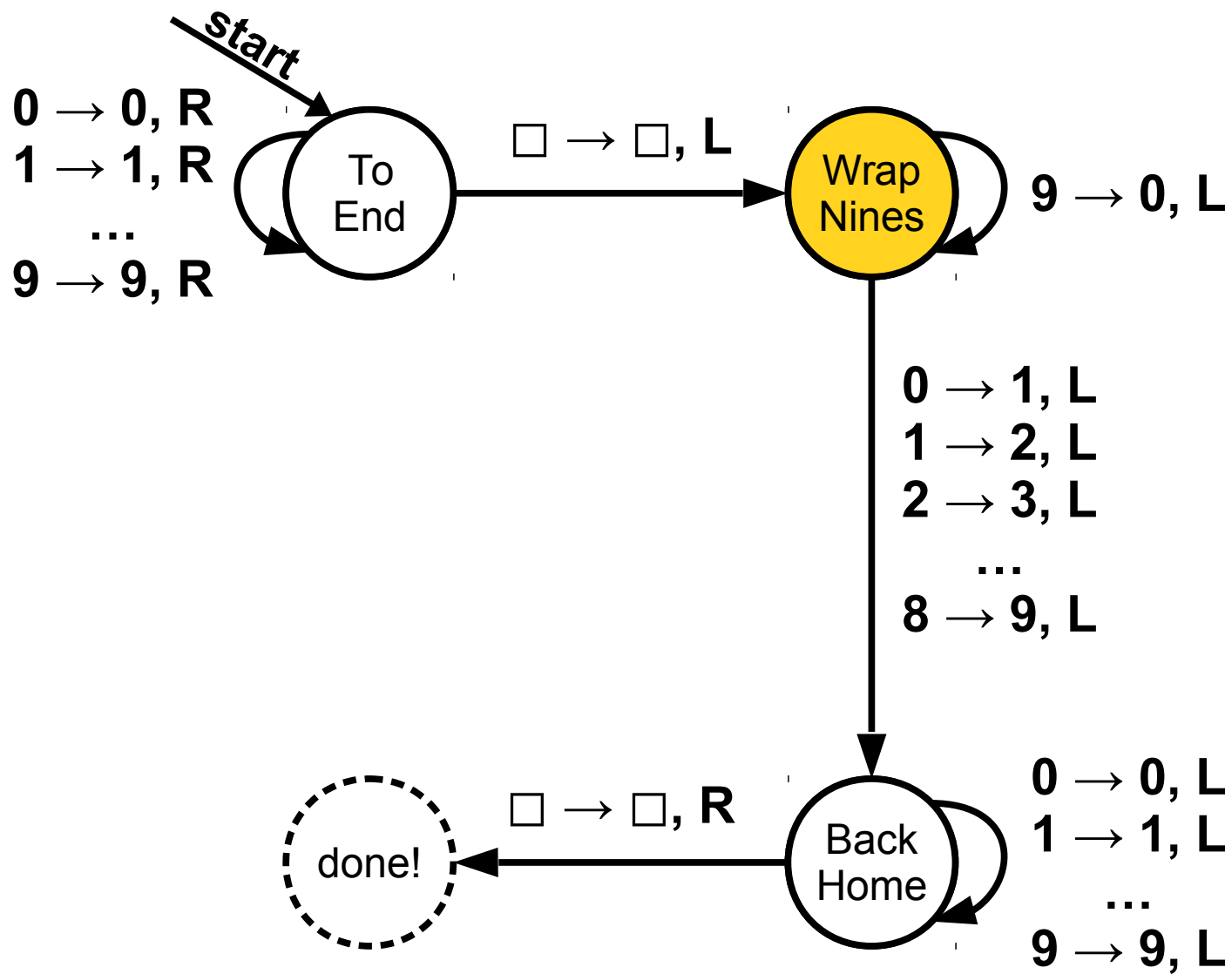


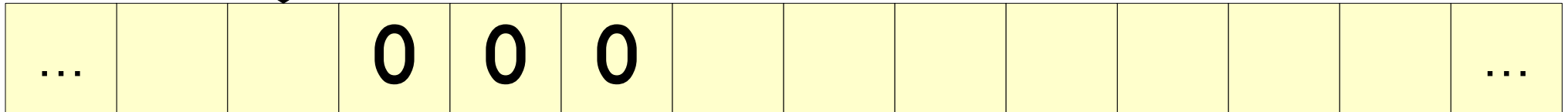
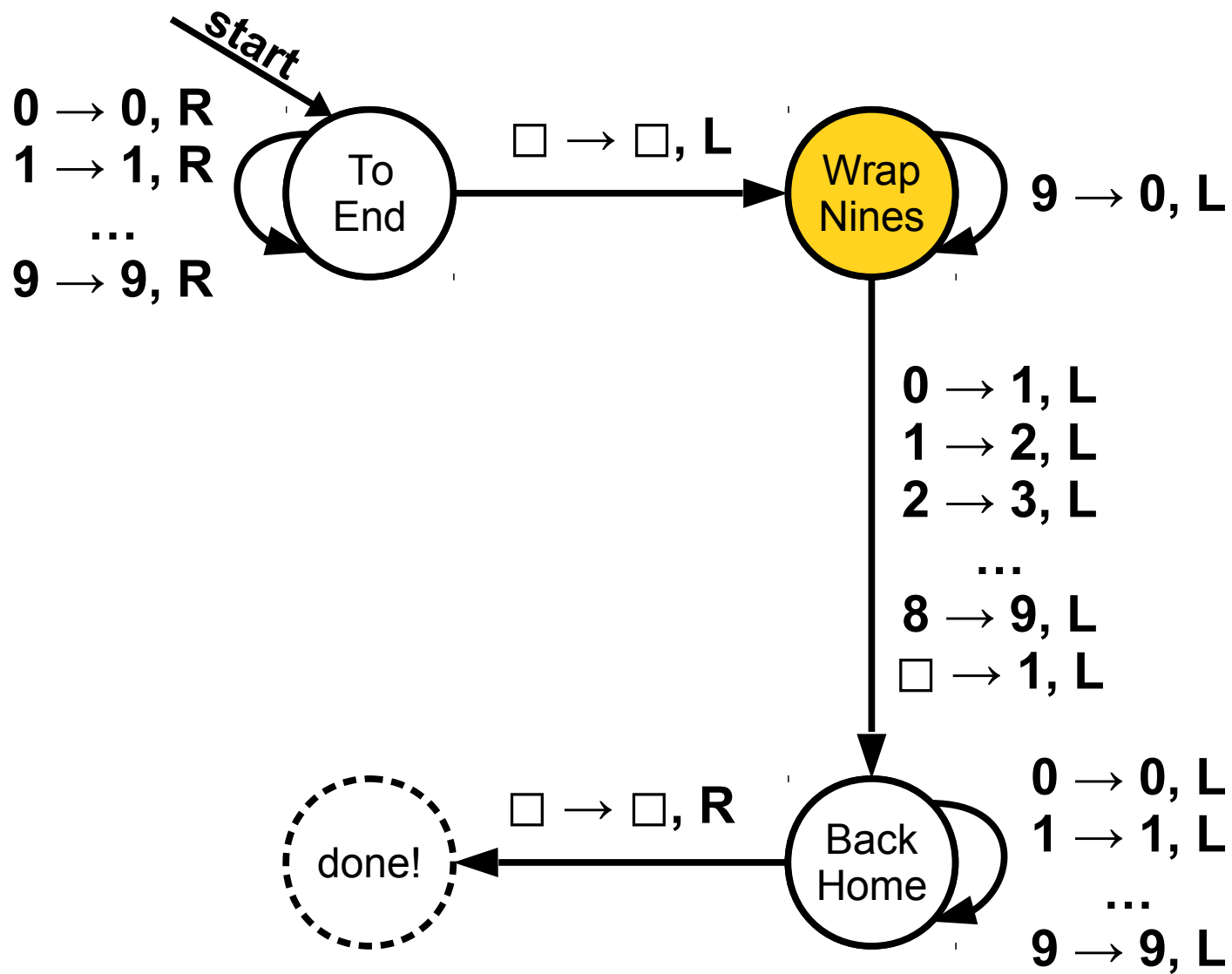


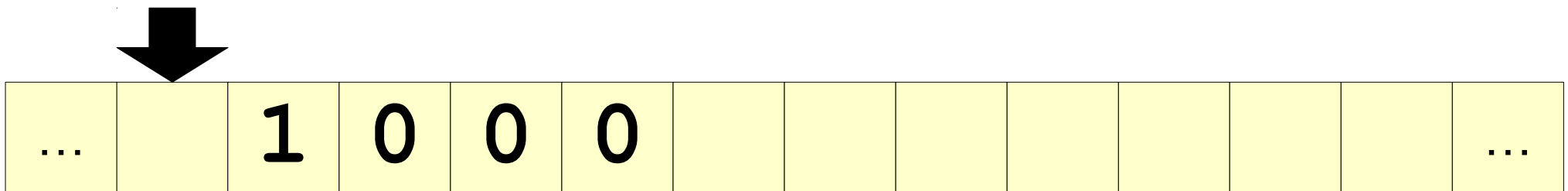
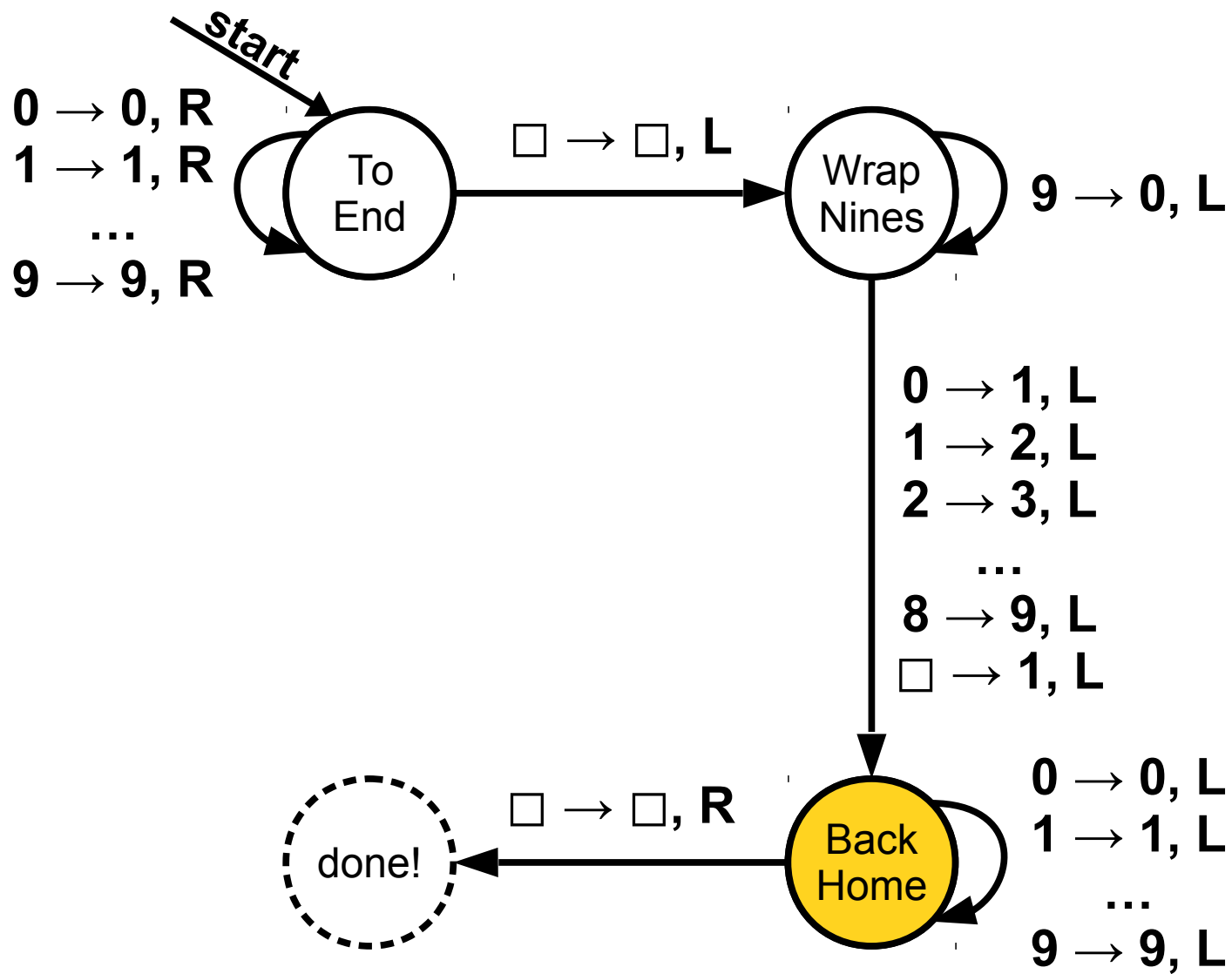


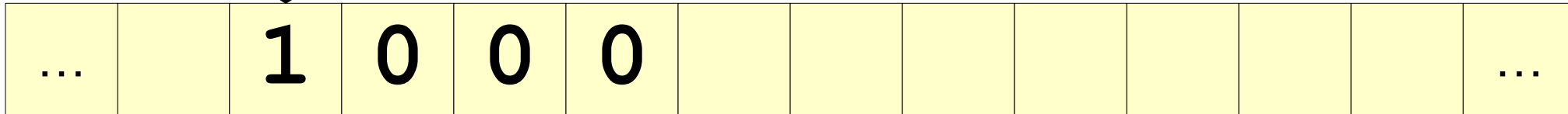
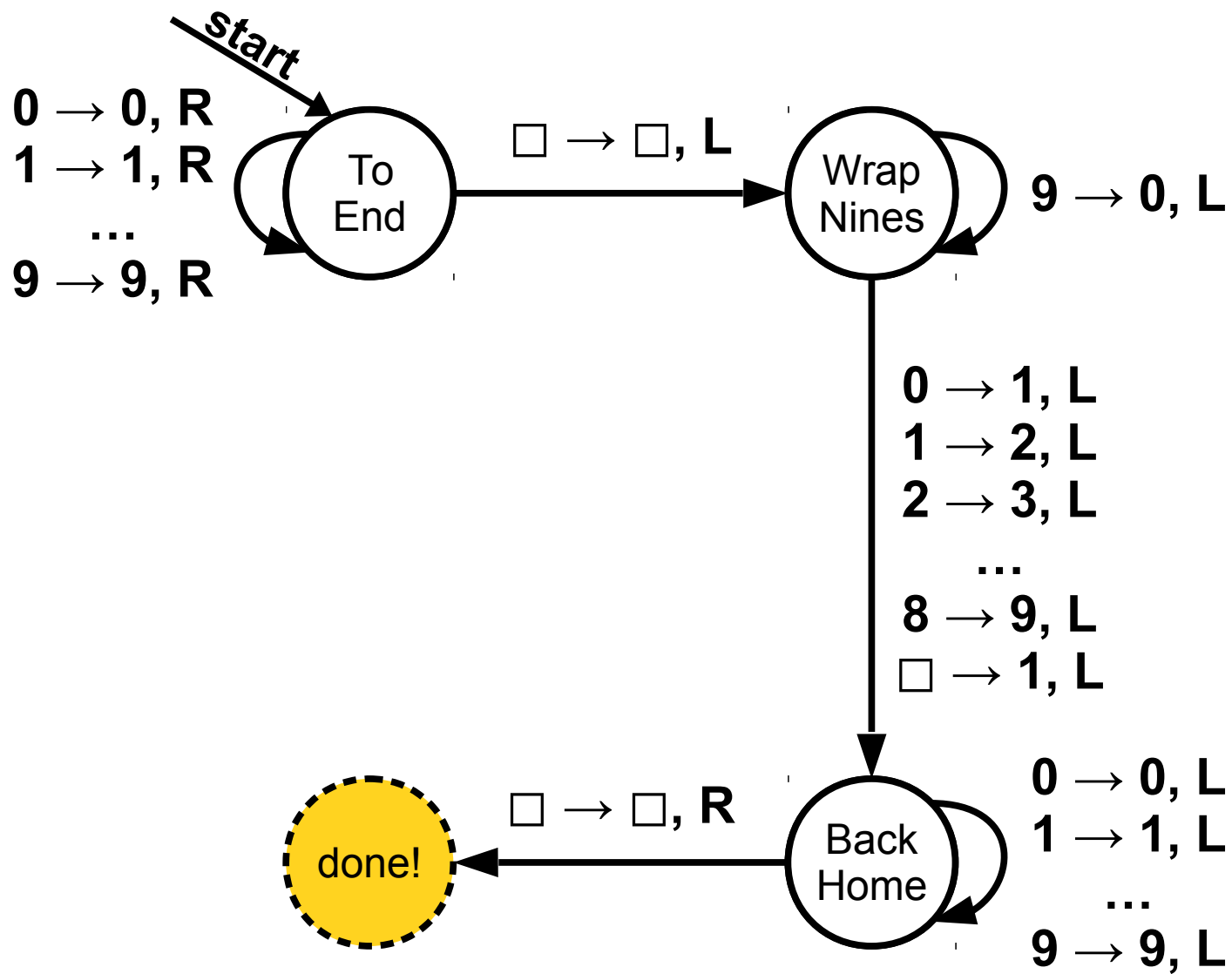


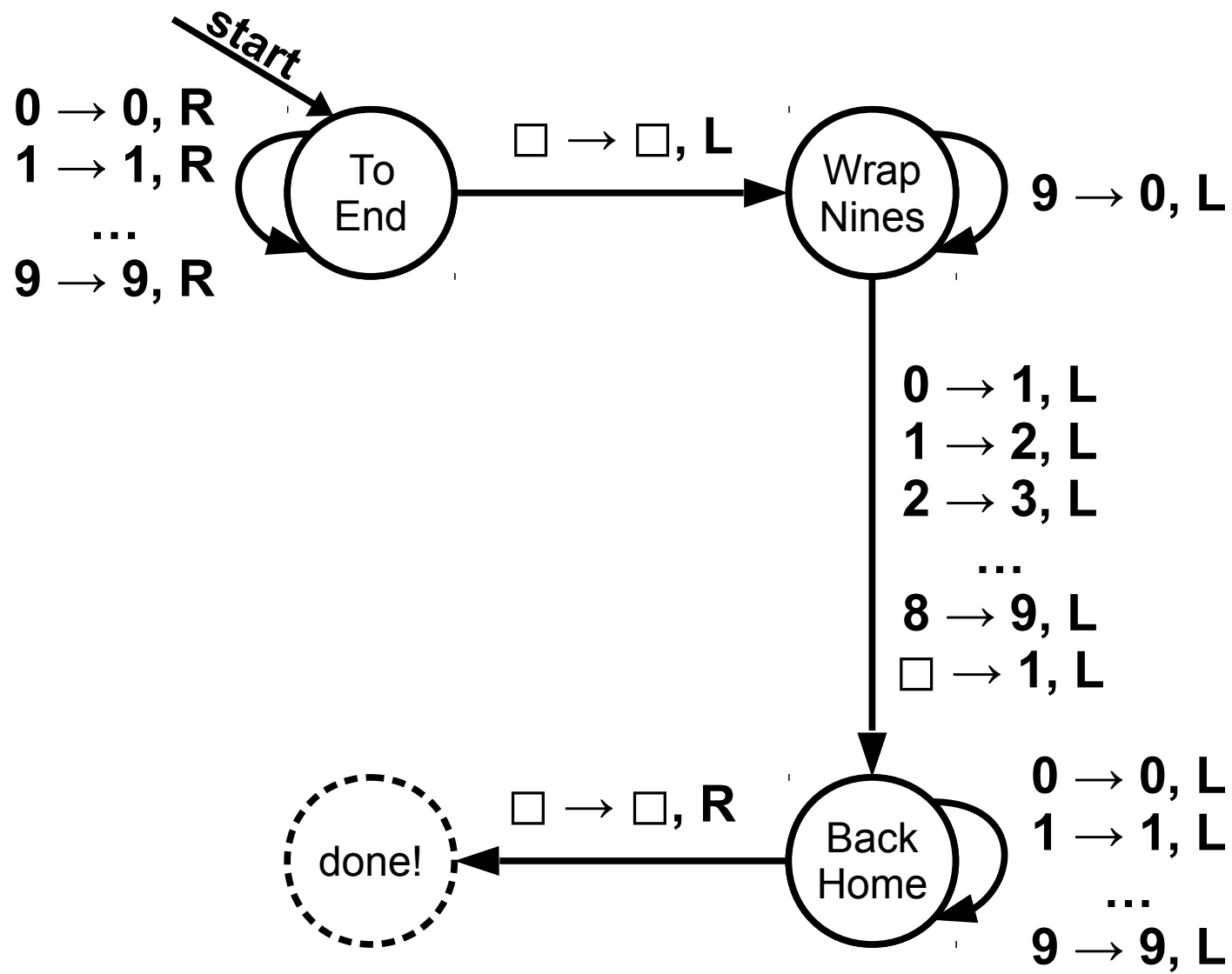






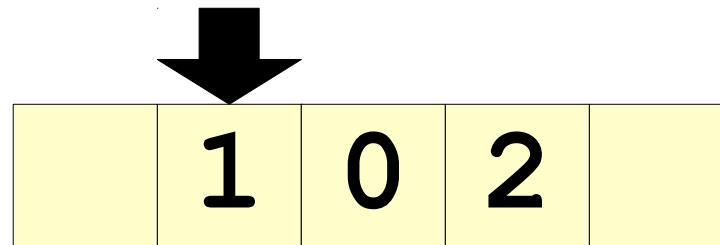






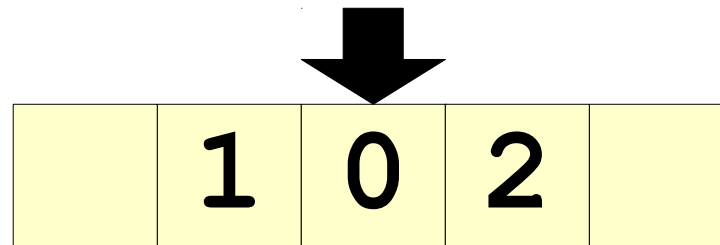
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



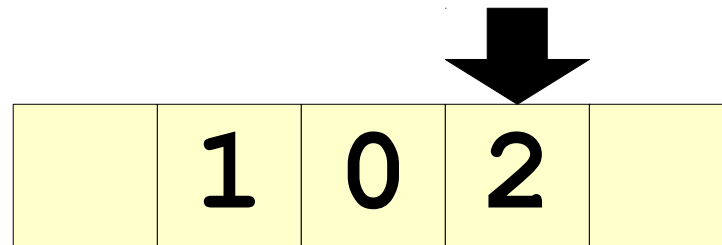
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



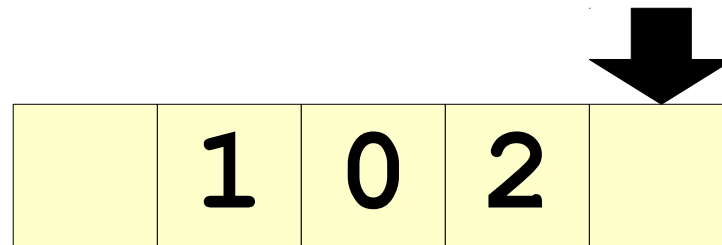
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



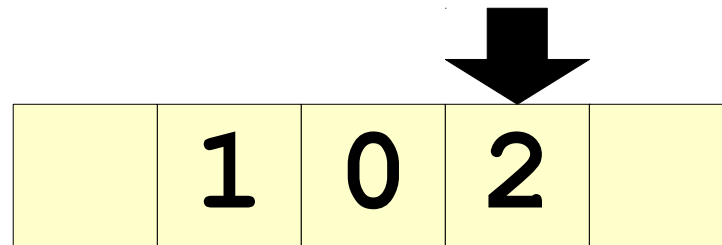
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



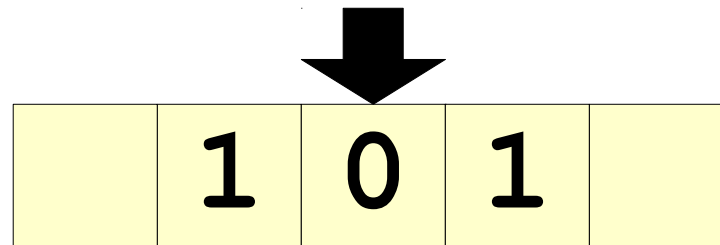
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



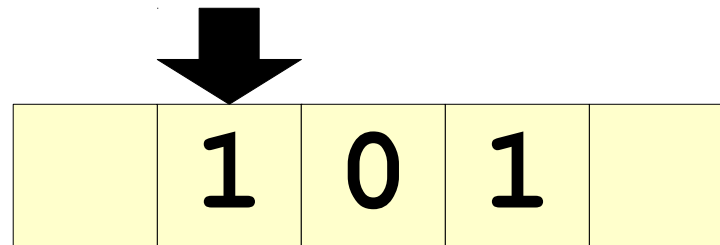
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



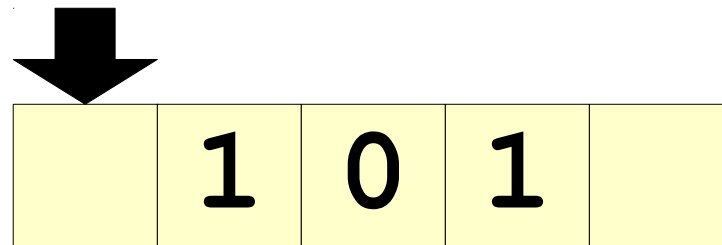
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



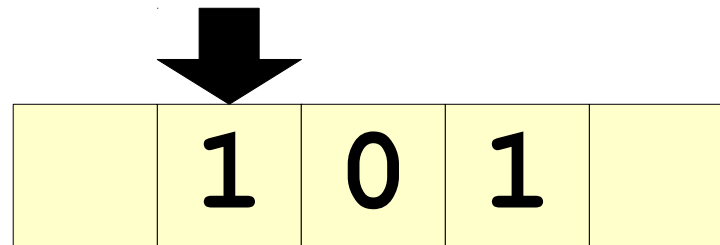
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



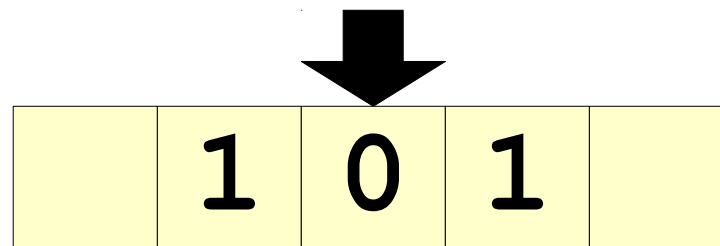
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



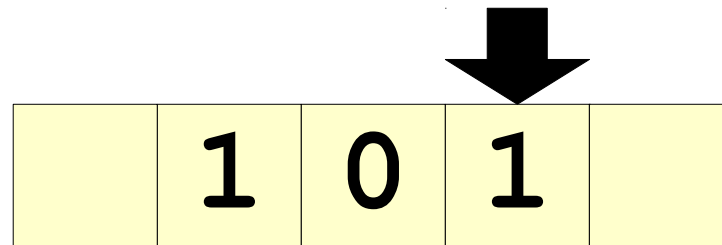
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



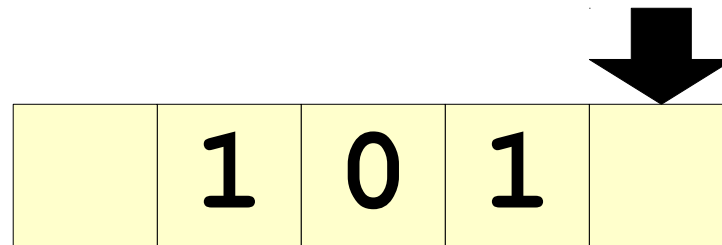
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



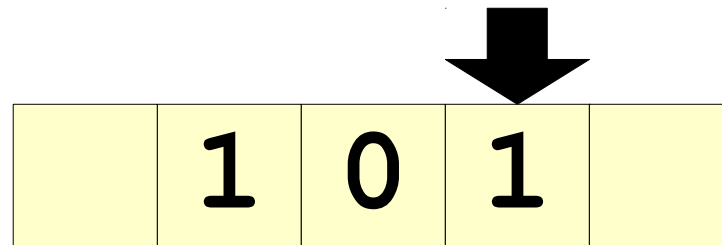
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



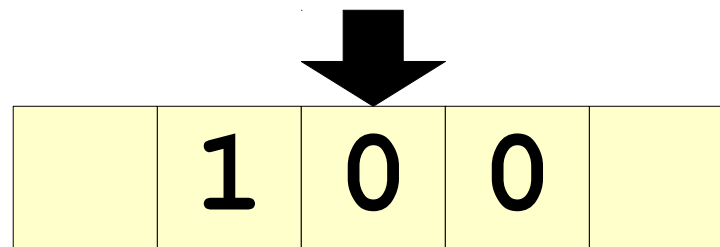
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



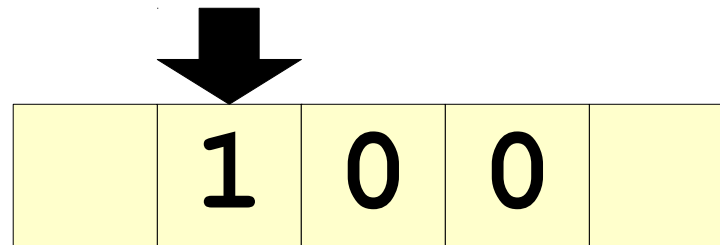
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



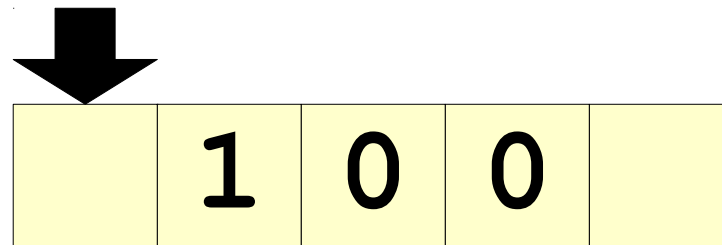
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



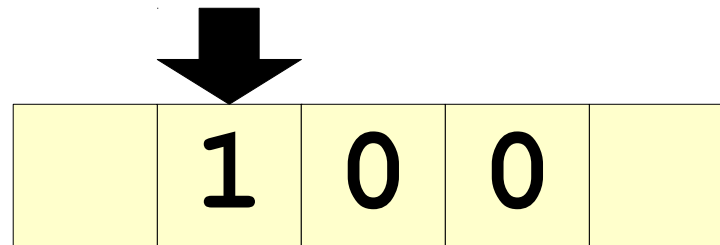
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



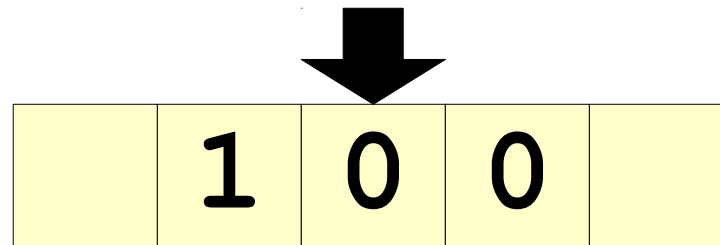
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



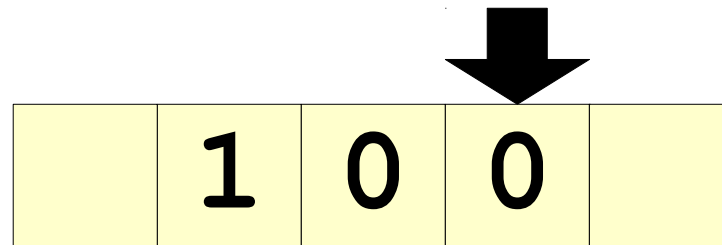
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



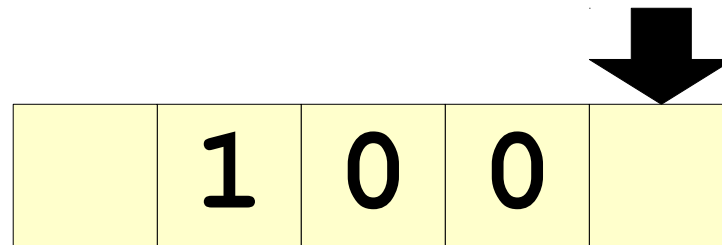
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



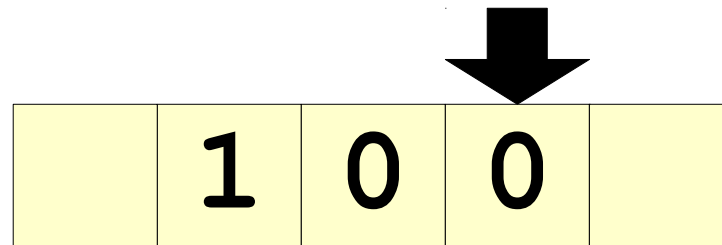
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



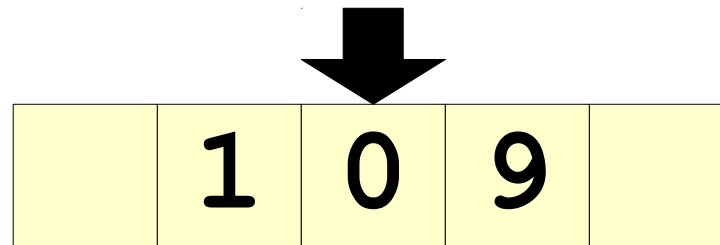
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



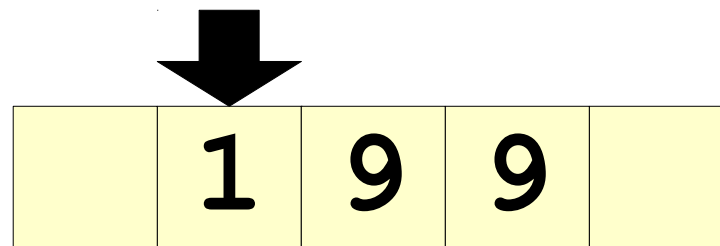
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



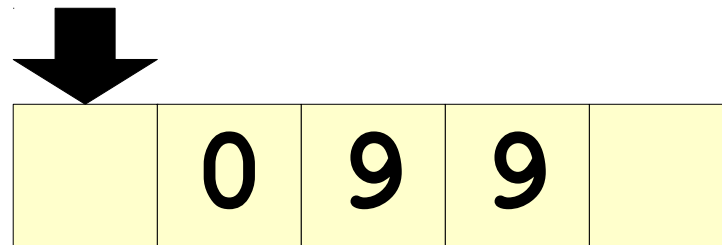
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



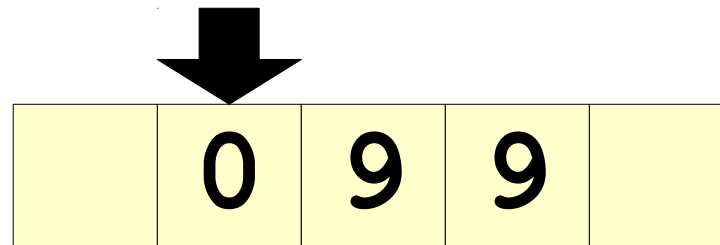
Decrementing Numbers

- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



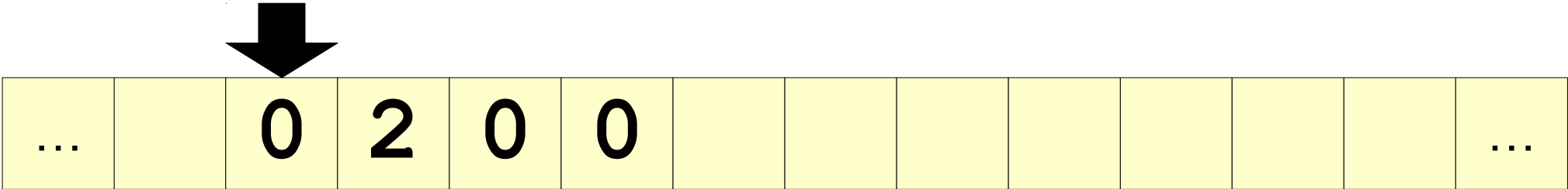
Decrementing Numbers

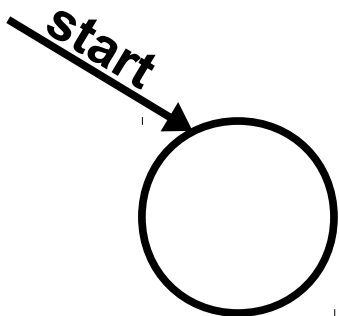
- Now, let's build a TM that decrements a number.
- We'll assume that
 - the tape head points at the start of a number,
 - there is at least one blank on each side of the number.
- The tape head will end at the start of the number after decrementing it.
- If the number is 0, then the subroutine should somehow signal this rather than making the number negative.



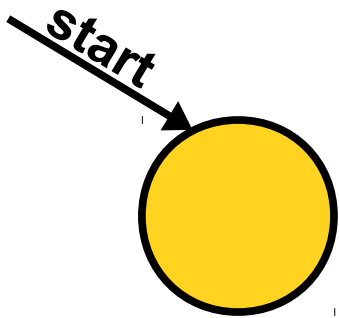
Decrementing Numbers

```
go to the end of the number;
if (every digit was 0) {
    signal that we're done;
}
while (the current digit is 0) {
    set the current digit to 9;
    back up one digit;
}
decrement the current digit;
go to the start of the number;
```

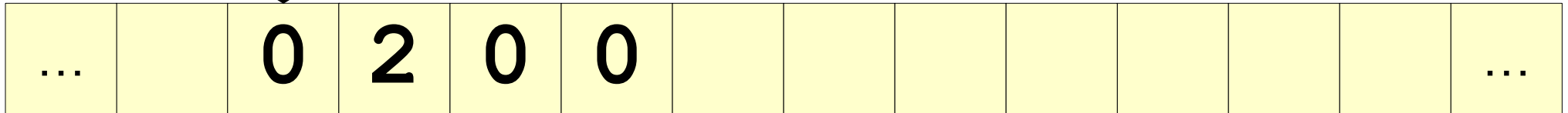
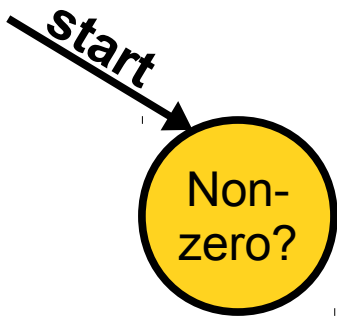



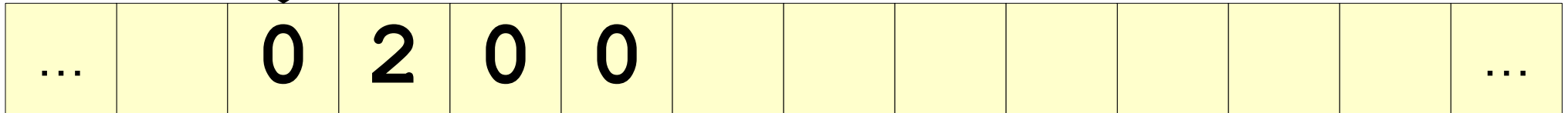
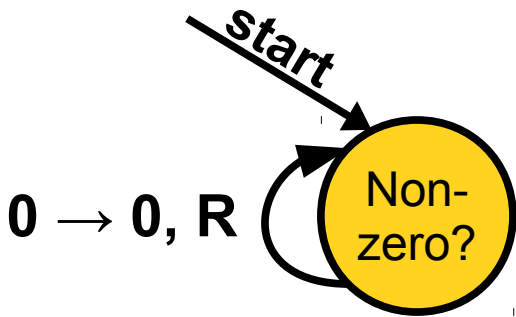


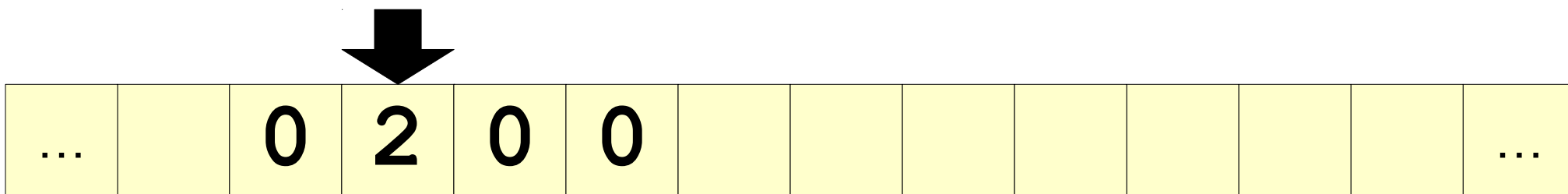
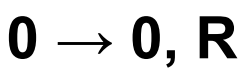
...		0	2	0	0								...
-----	--	---	---	---	---	--	--	--	--	--	--	--	-----

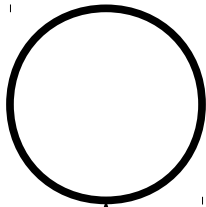


...		0	2	0	0								...
-----	--	---	---	---	---	--	--	--	--	--	--	--	-----









1 \rightarrow **1**, R

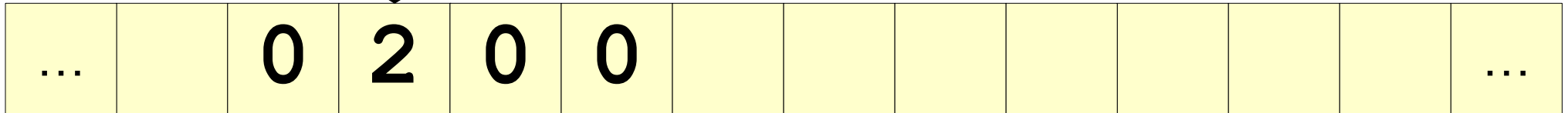
2 \rightarrow **2**, R

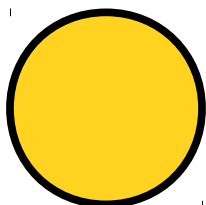
...

9 \rightarrow **9**, R



0 \rightarrow **0**, R





1 \rightarrow **1**, R

2 \rightarrow **2**, R

...

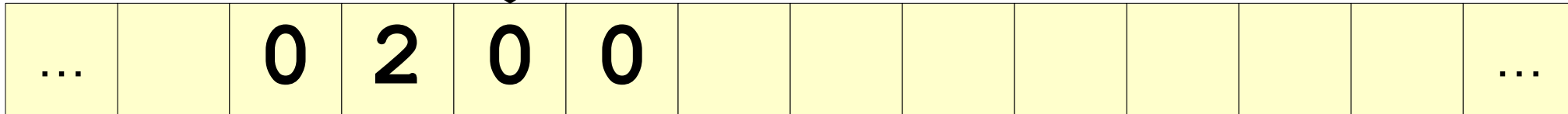
9 \rightarrow **9**, R

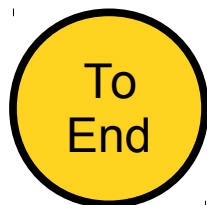
start



Non-
zero?

0 \rightarrow **0**, R





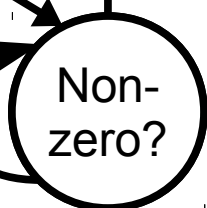
1 → 1, R

2 → 2, R

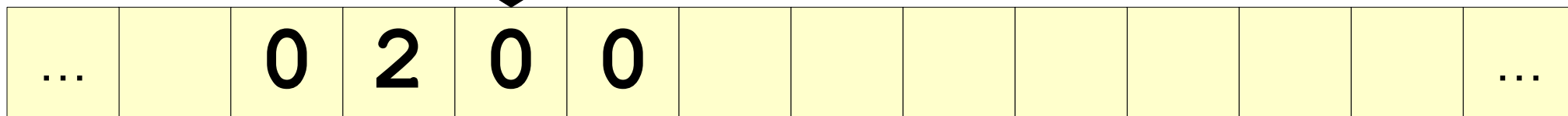
...

9 → 9, R

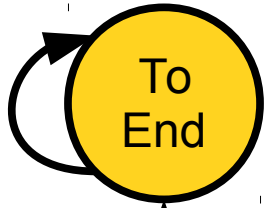
start



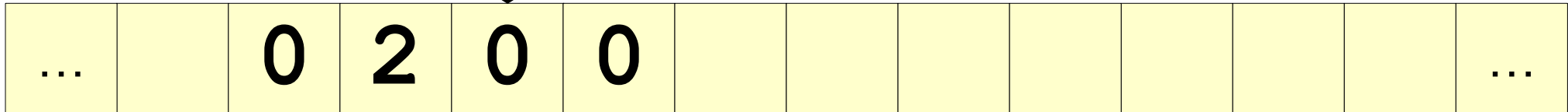
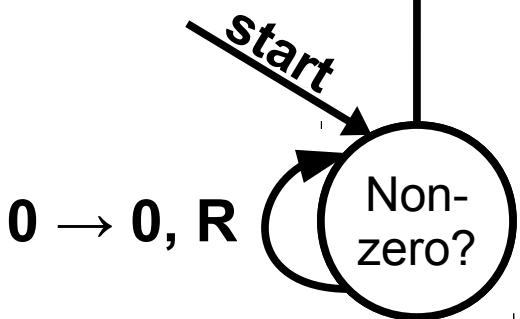
0 → 0, R

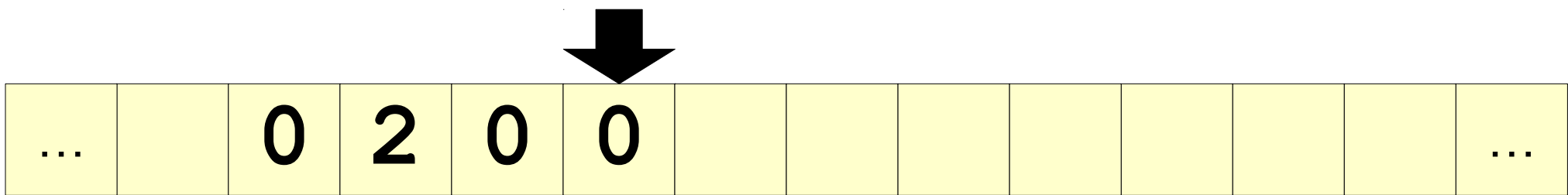
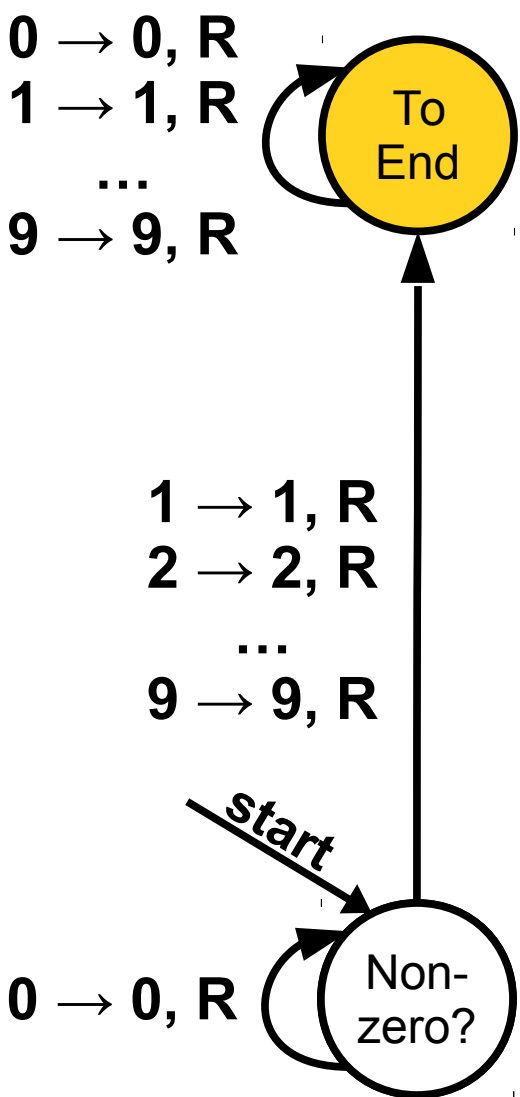


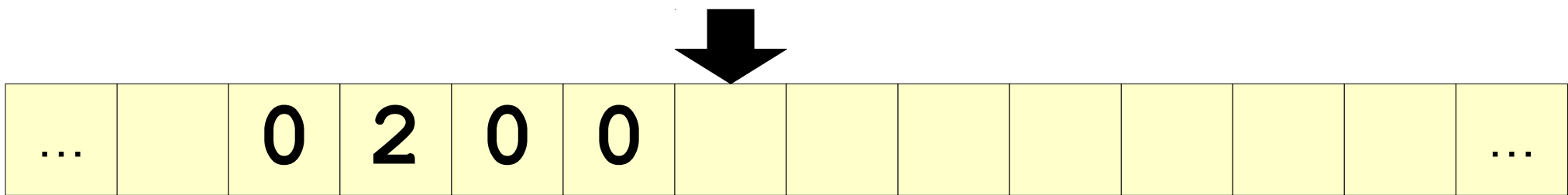
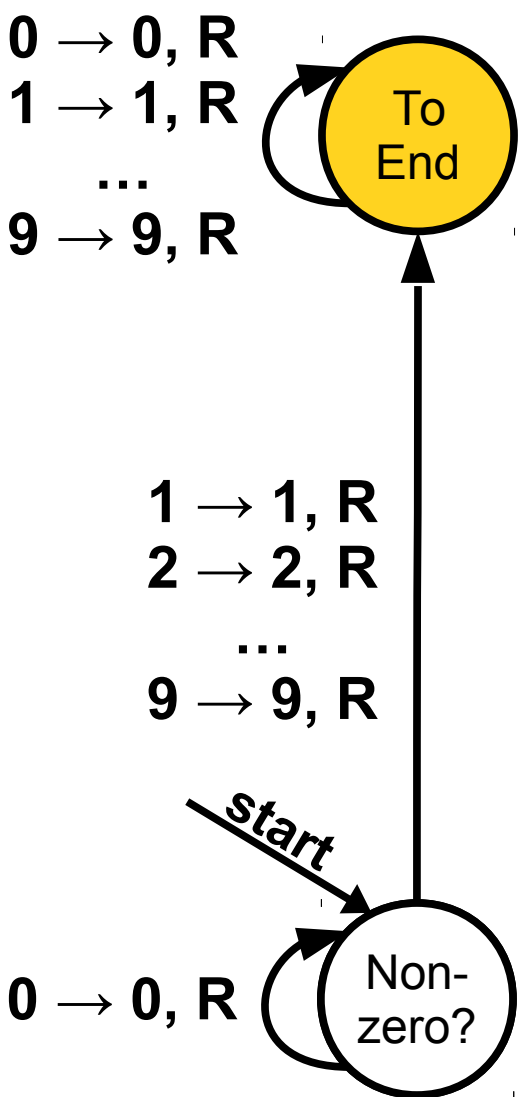
0 → 0, R
1 → 1, R
...
9 → 9, R

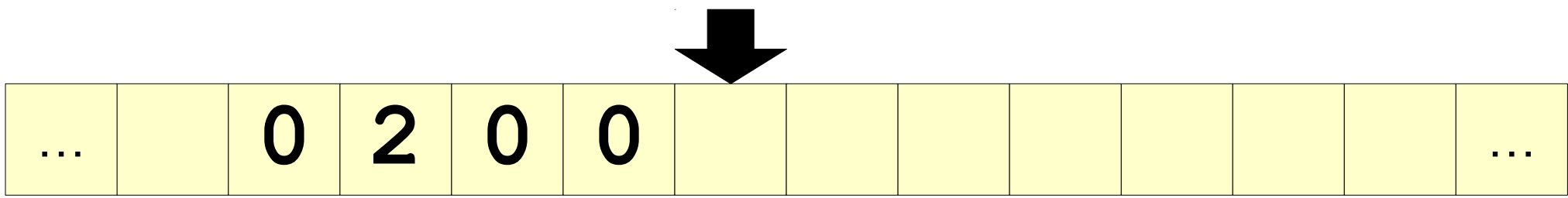
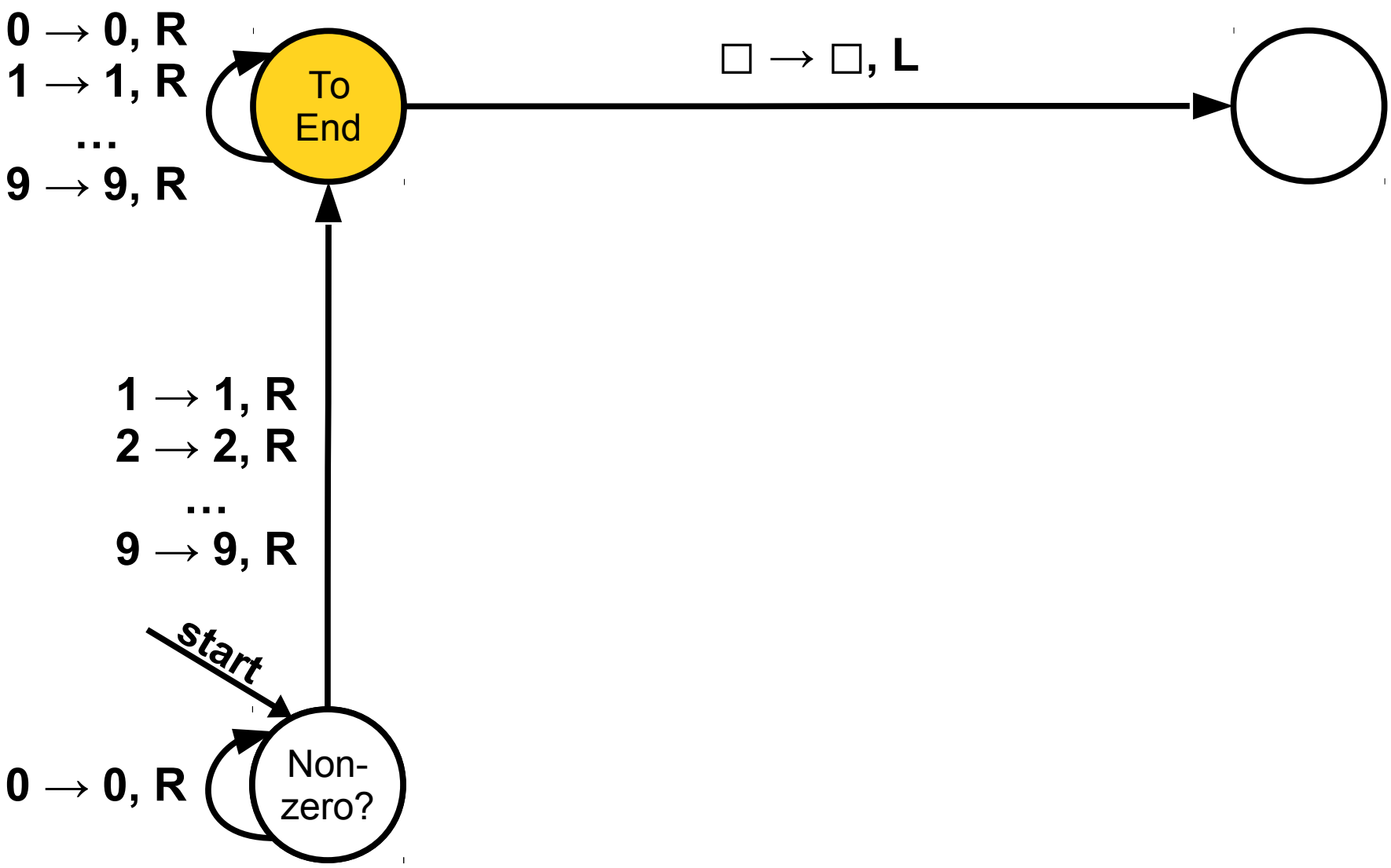


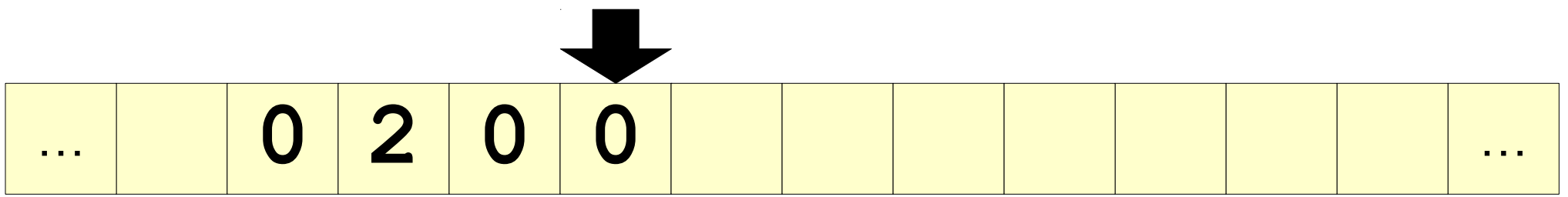
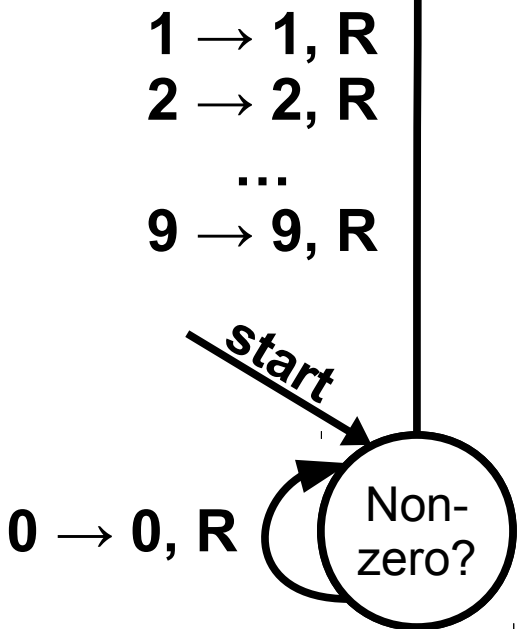
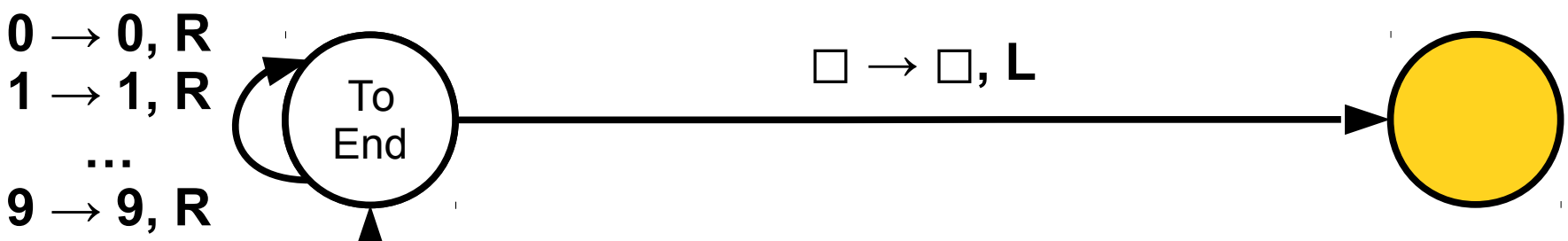
1 → 1, R
2 → 2, R
...
9 → 9, R

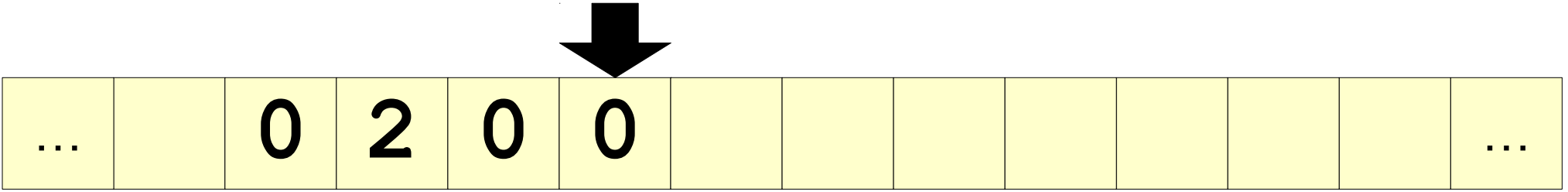
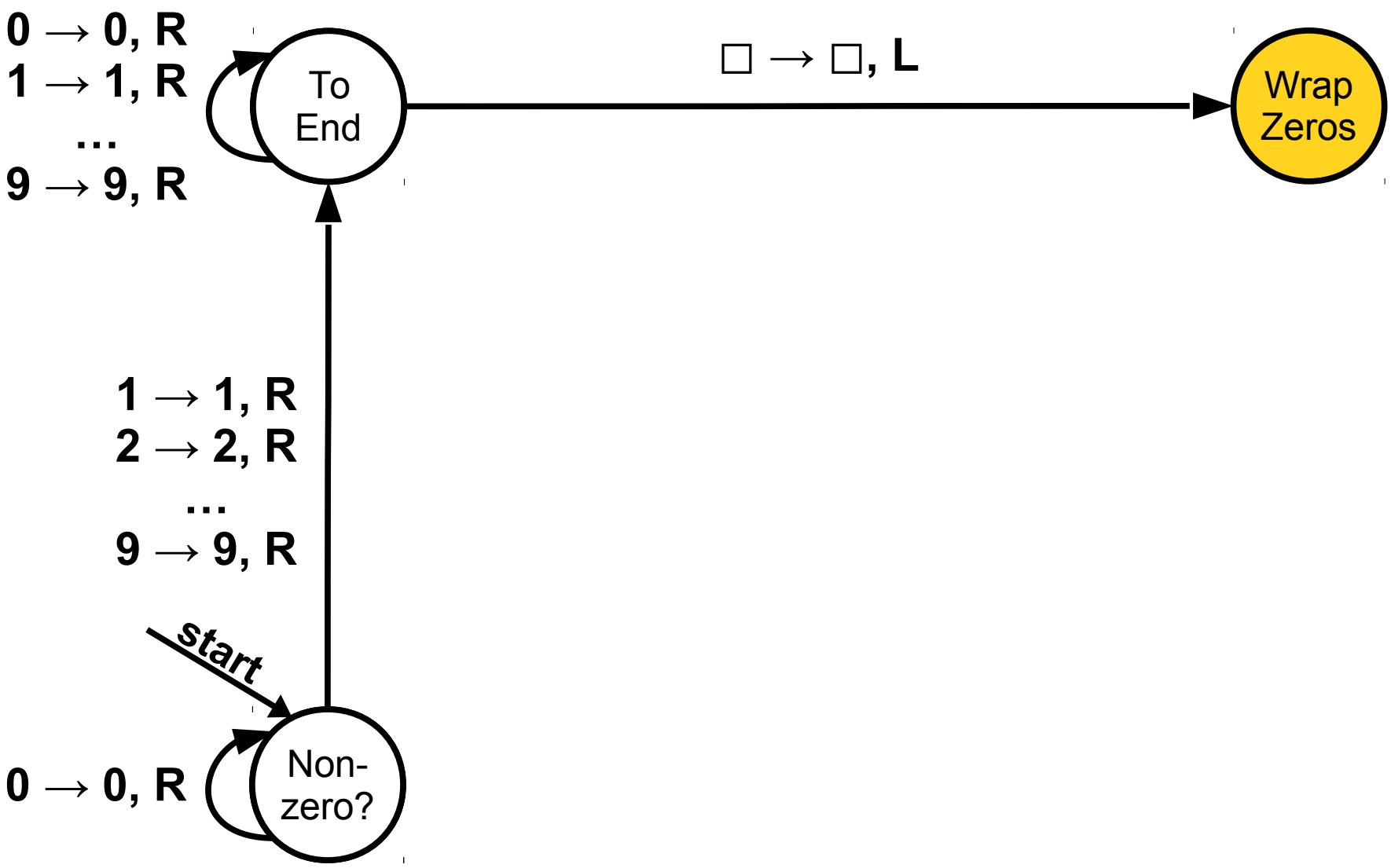


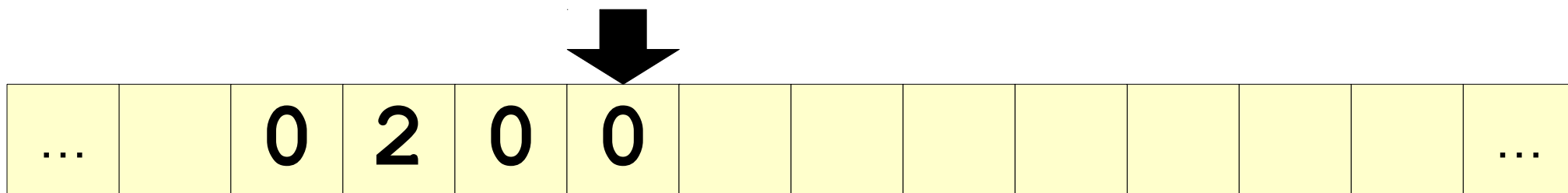
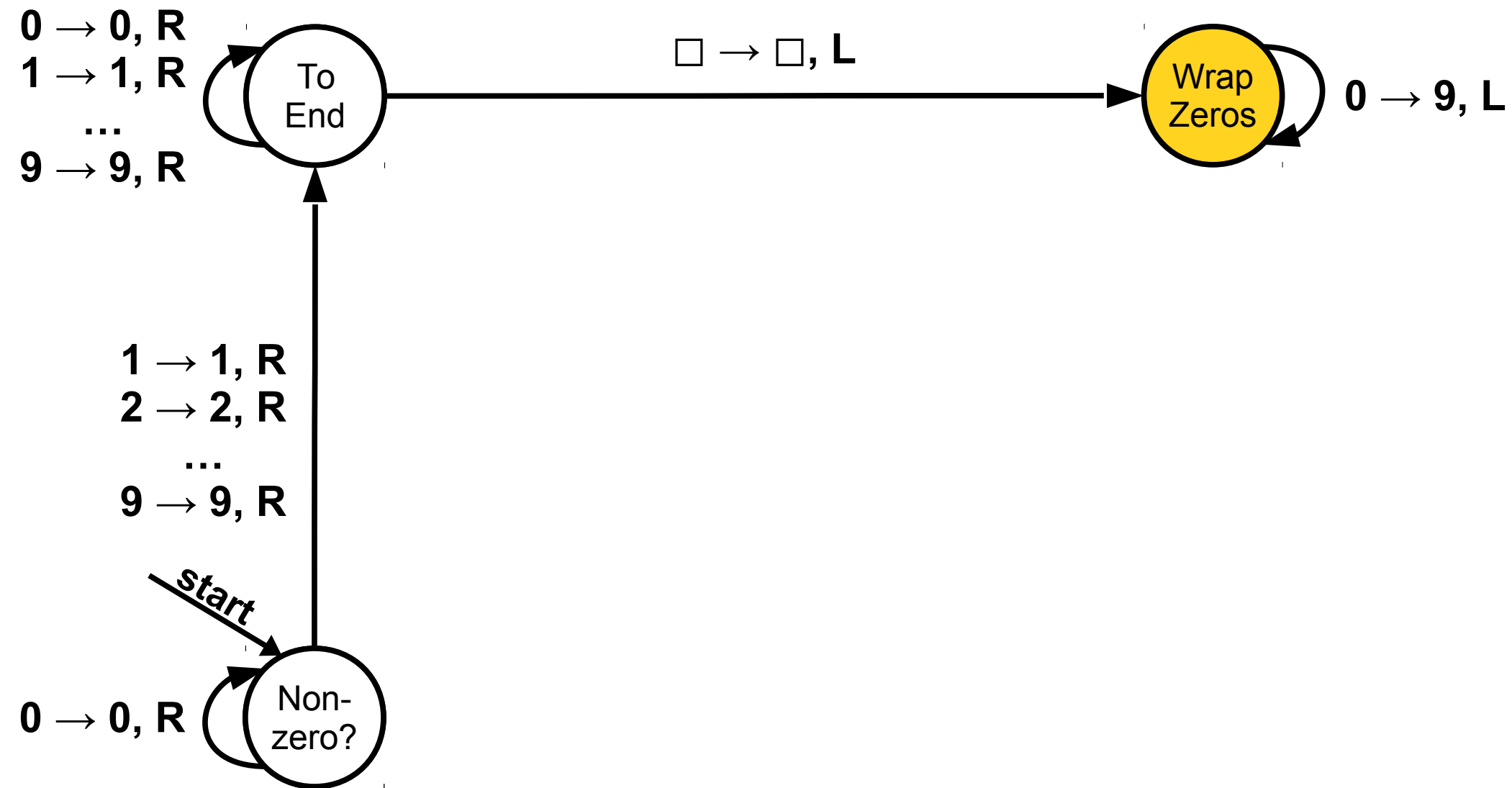


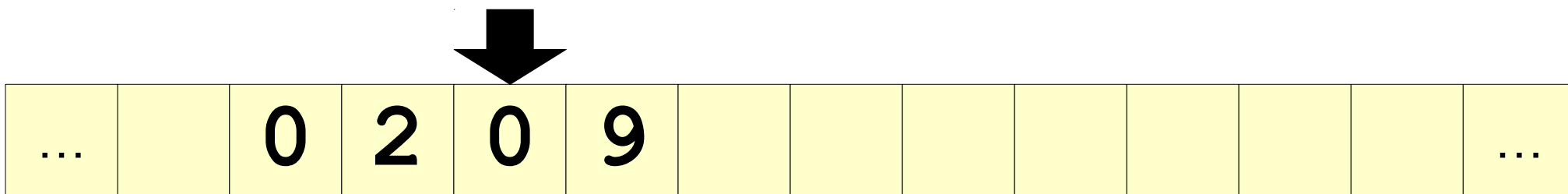
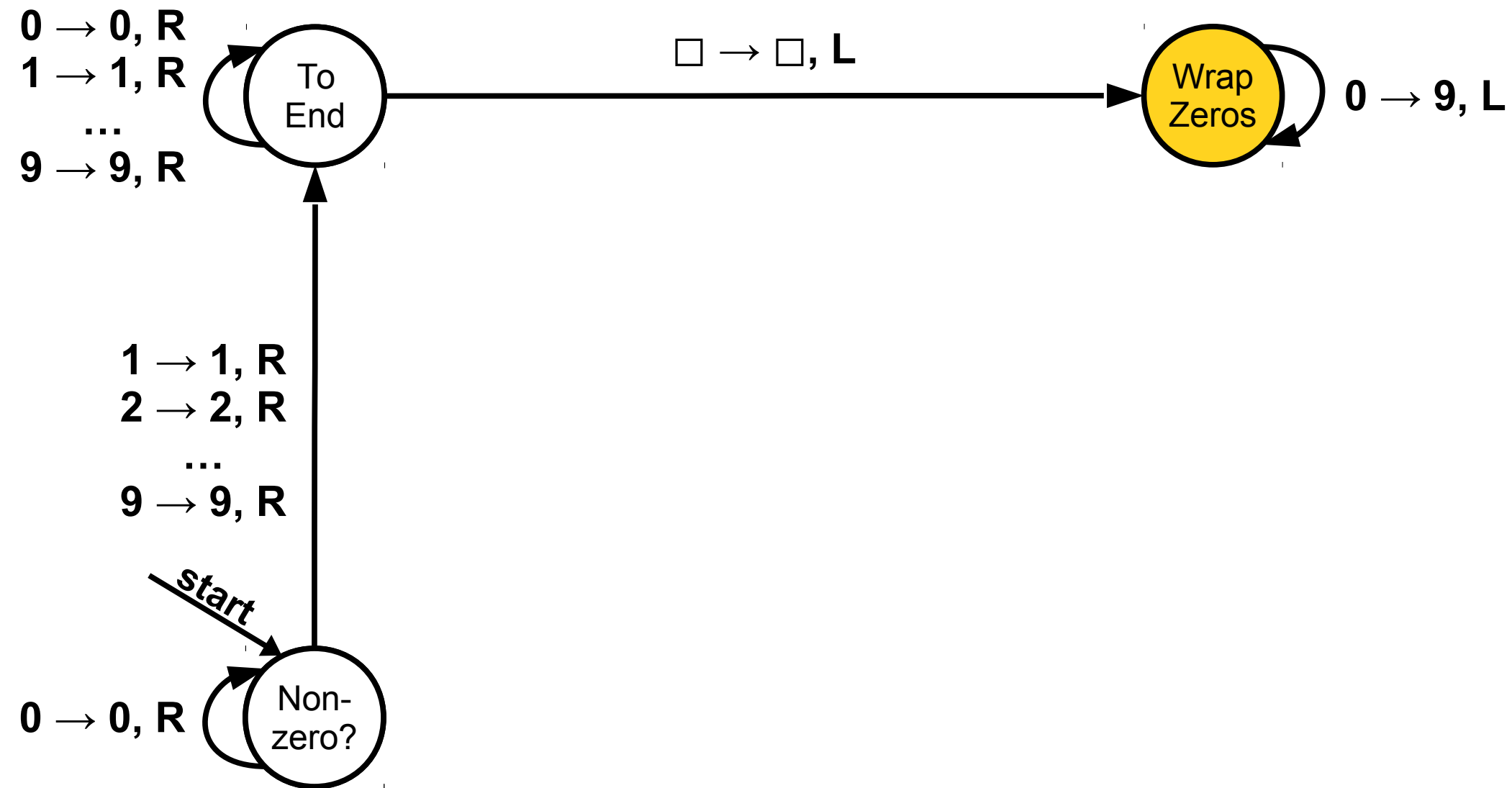


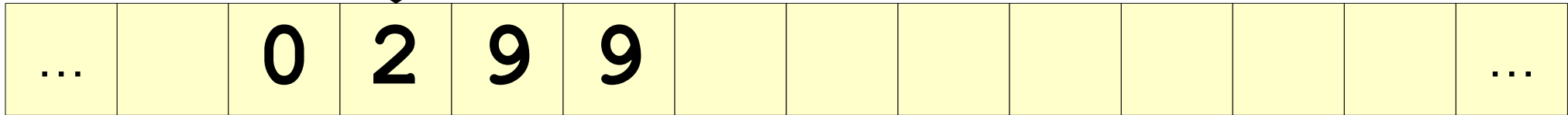
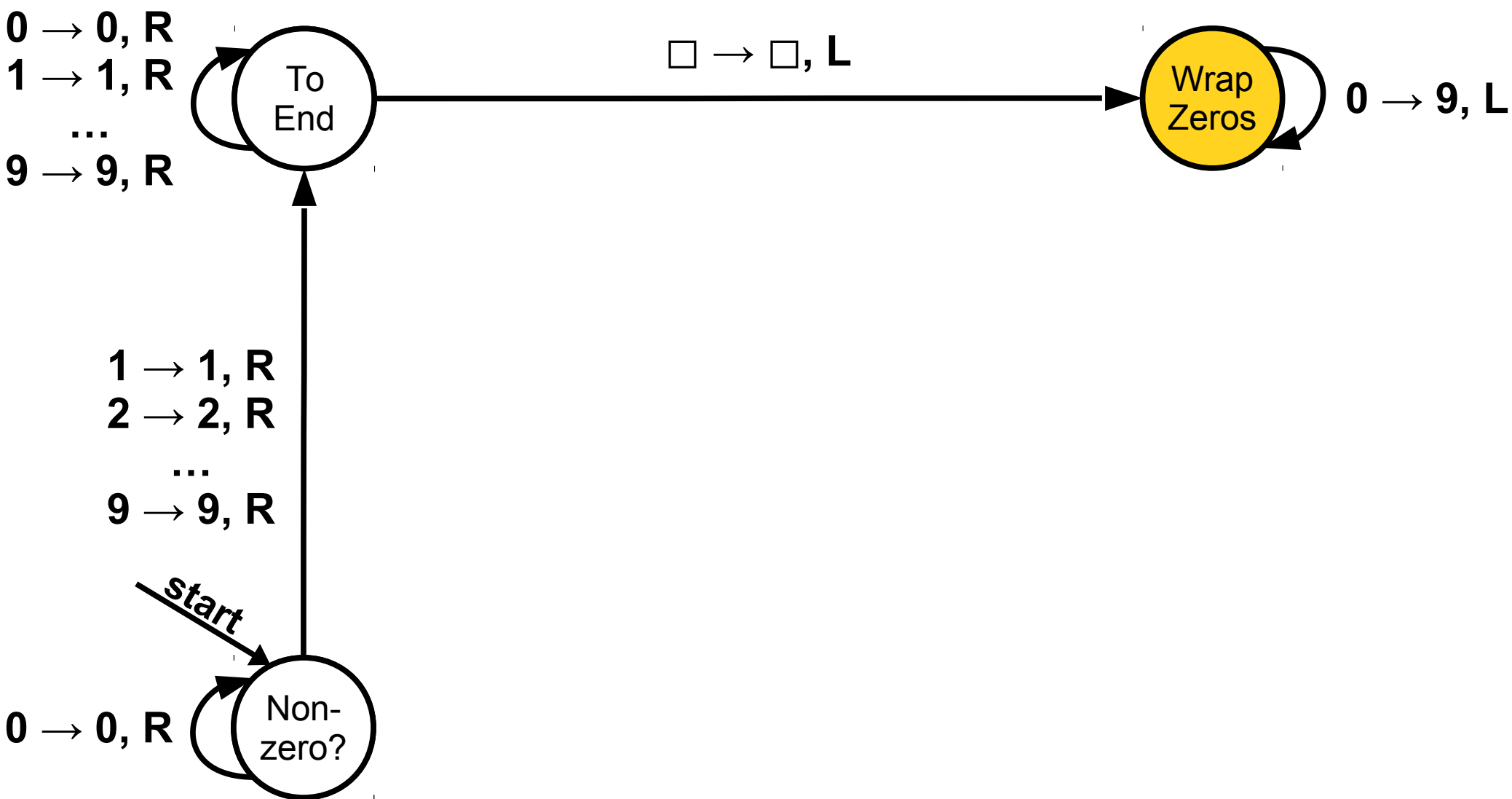


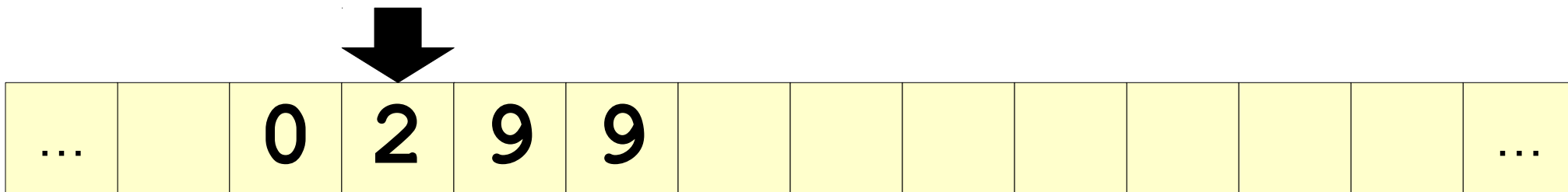
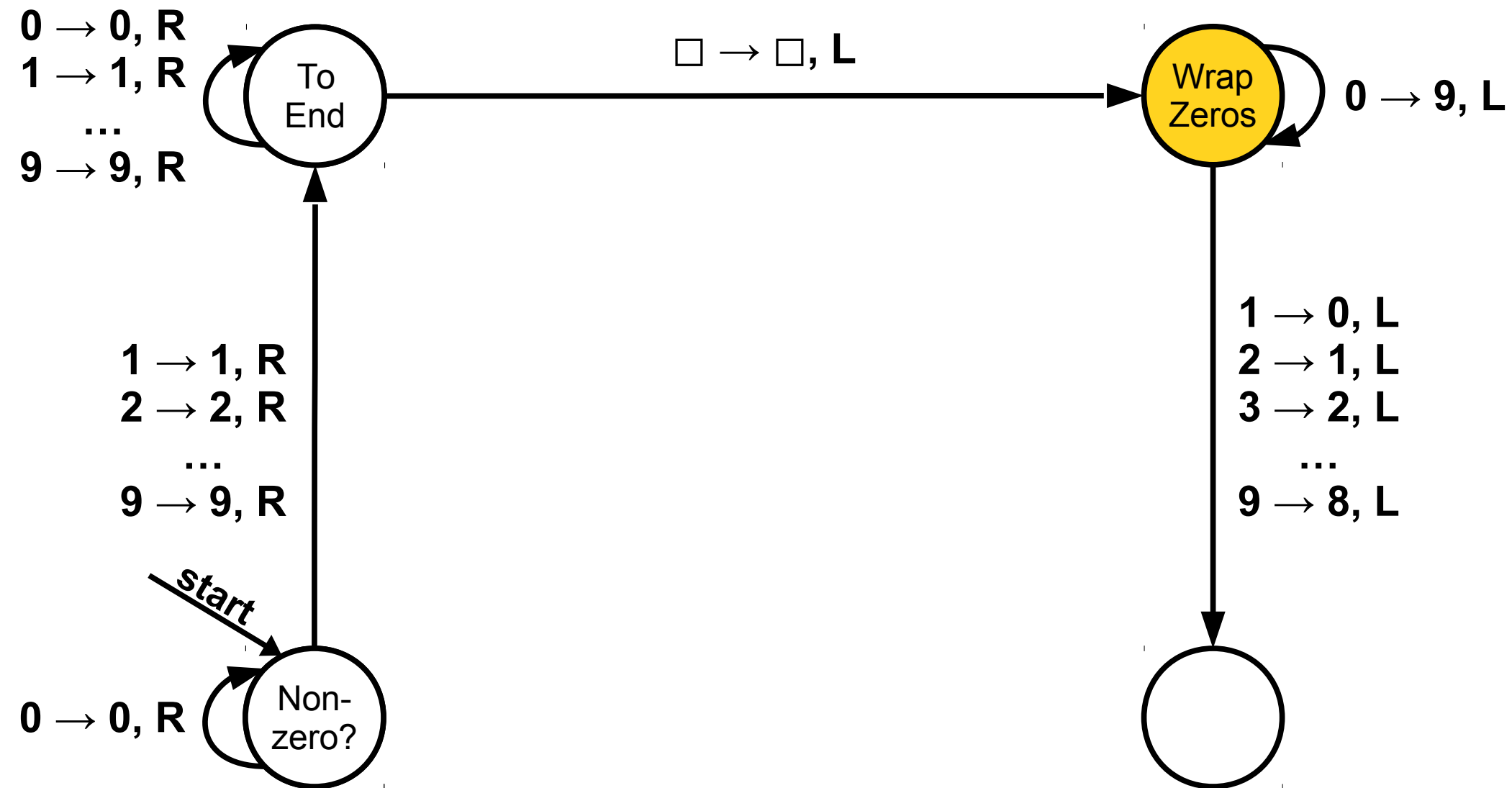


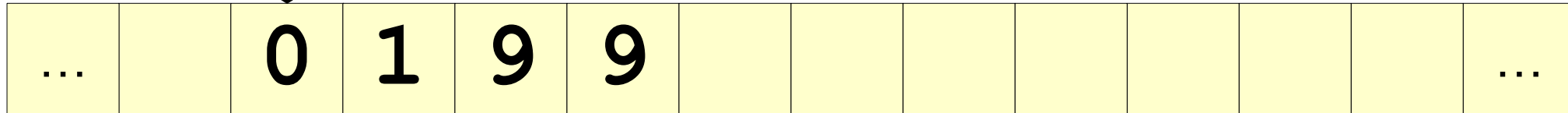
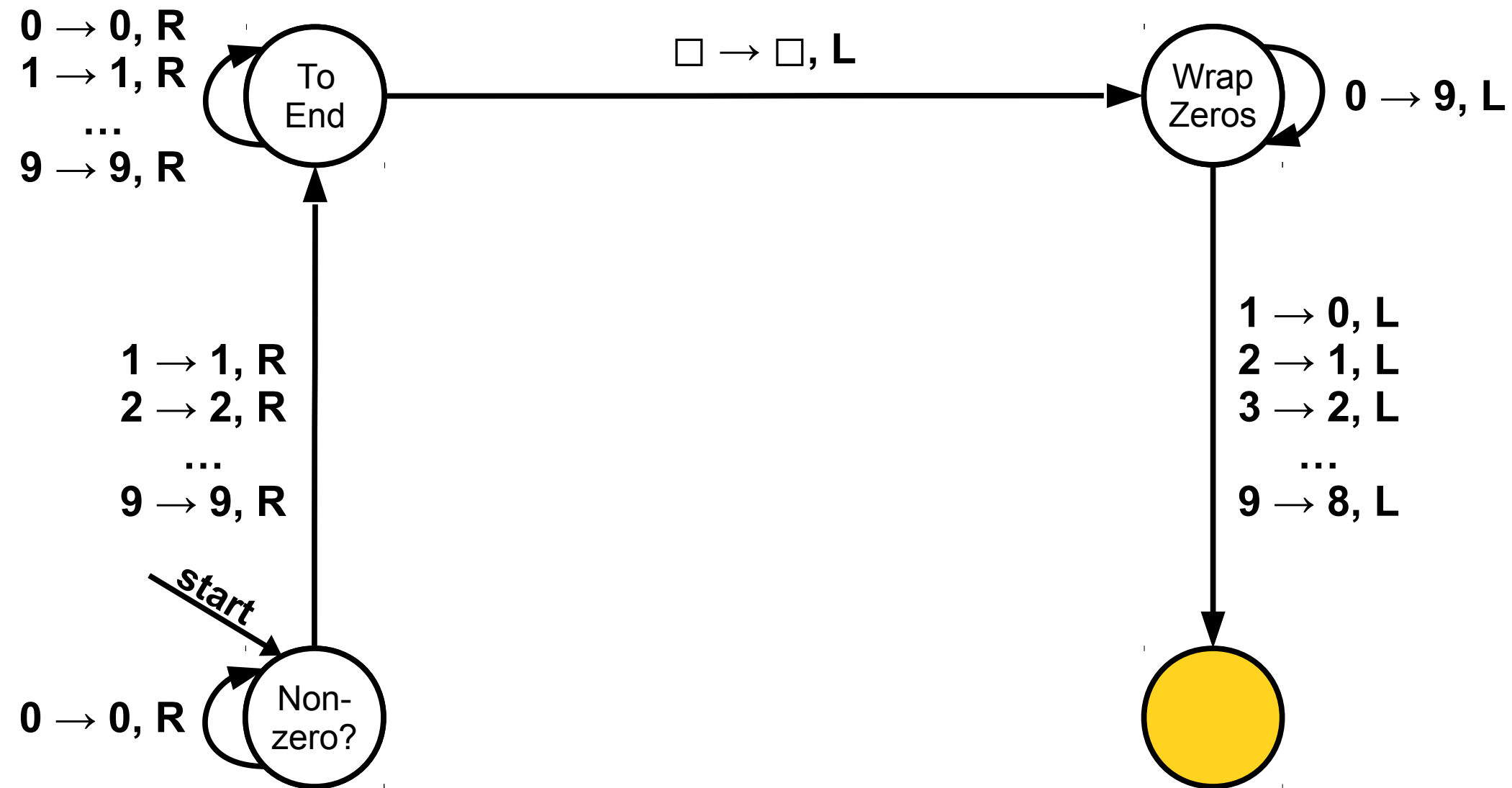


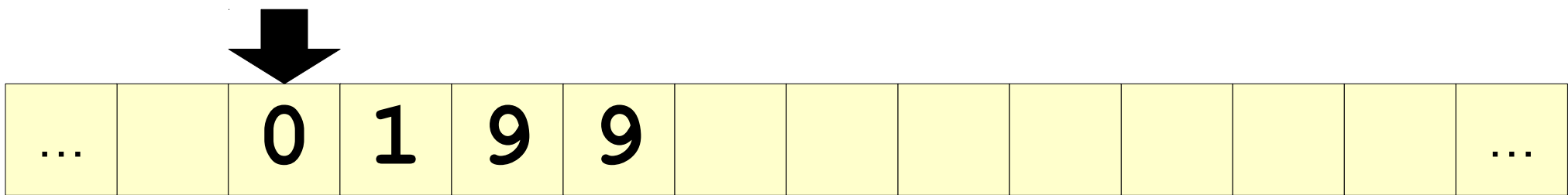
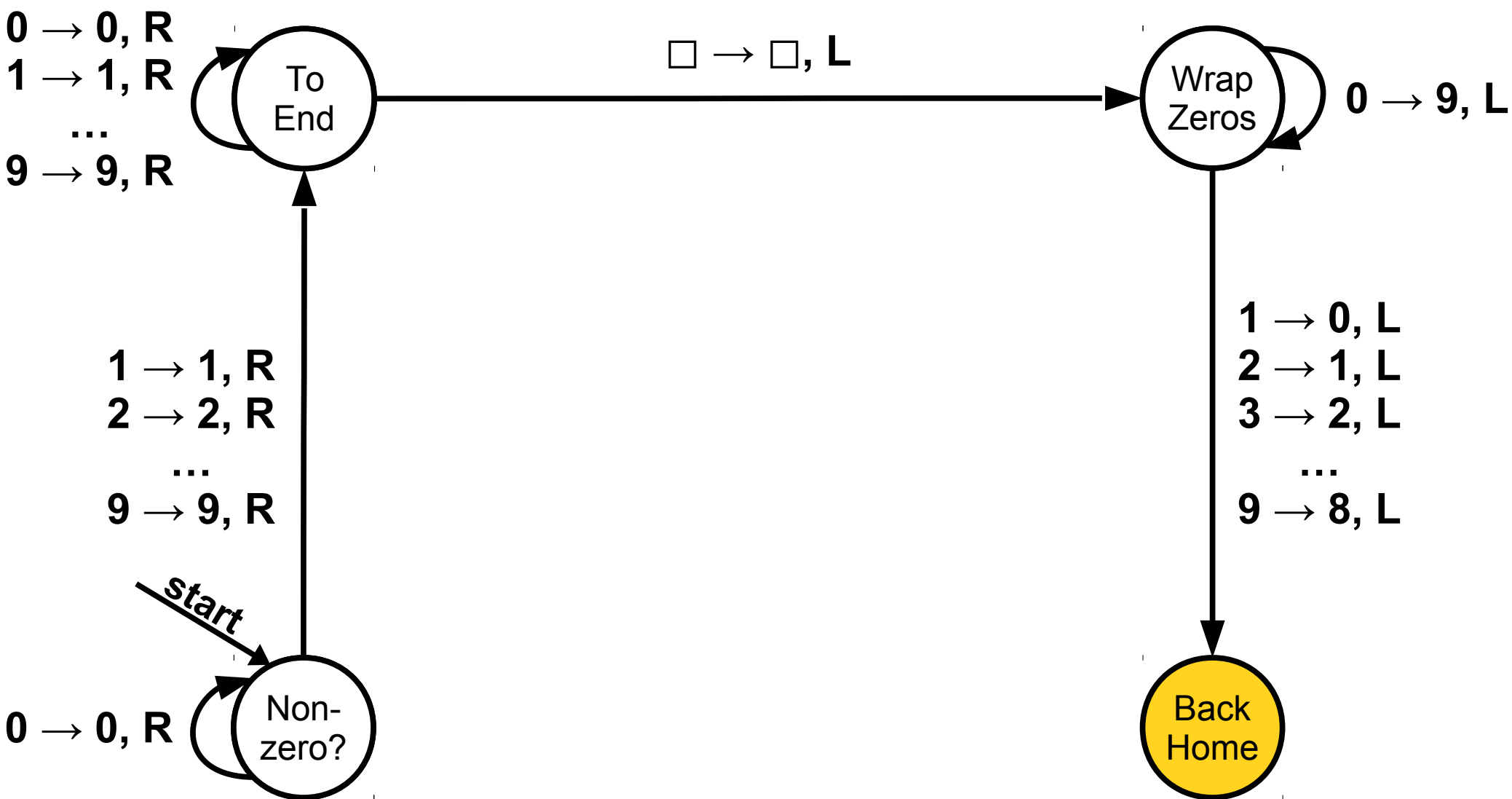


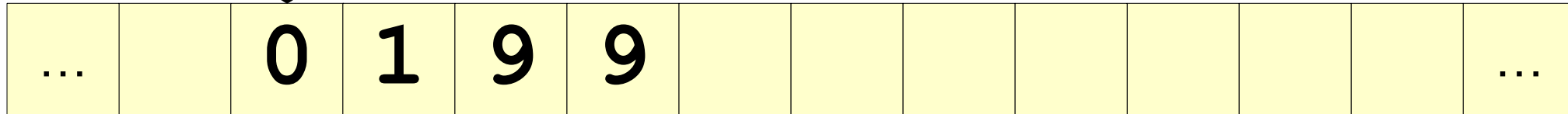
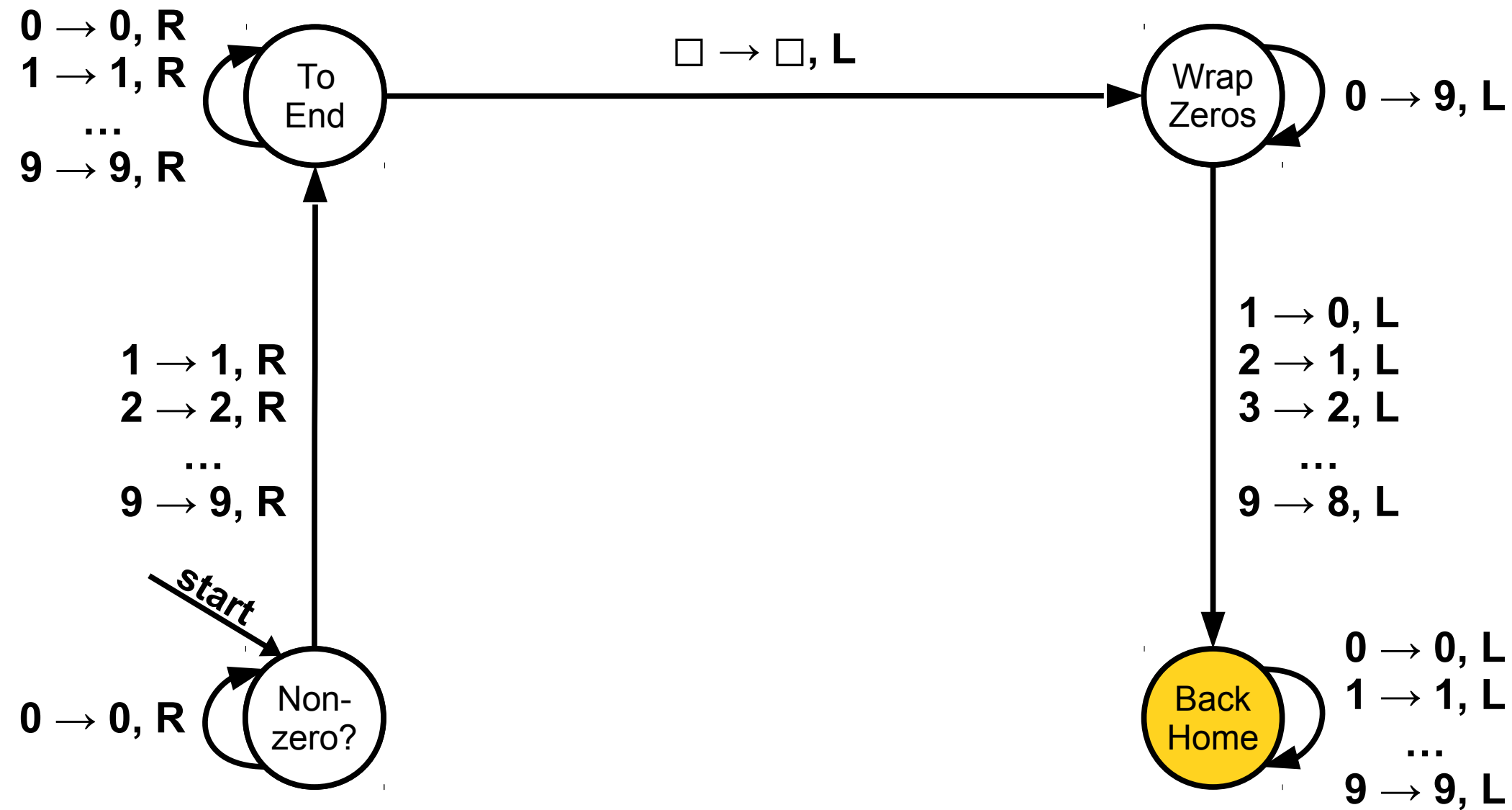


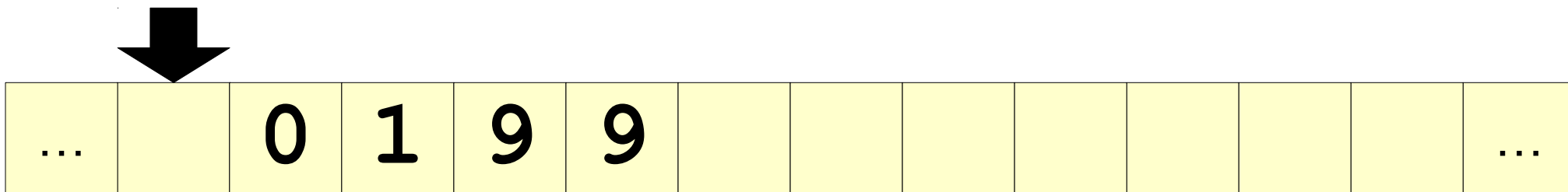
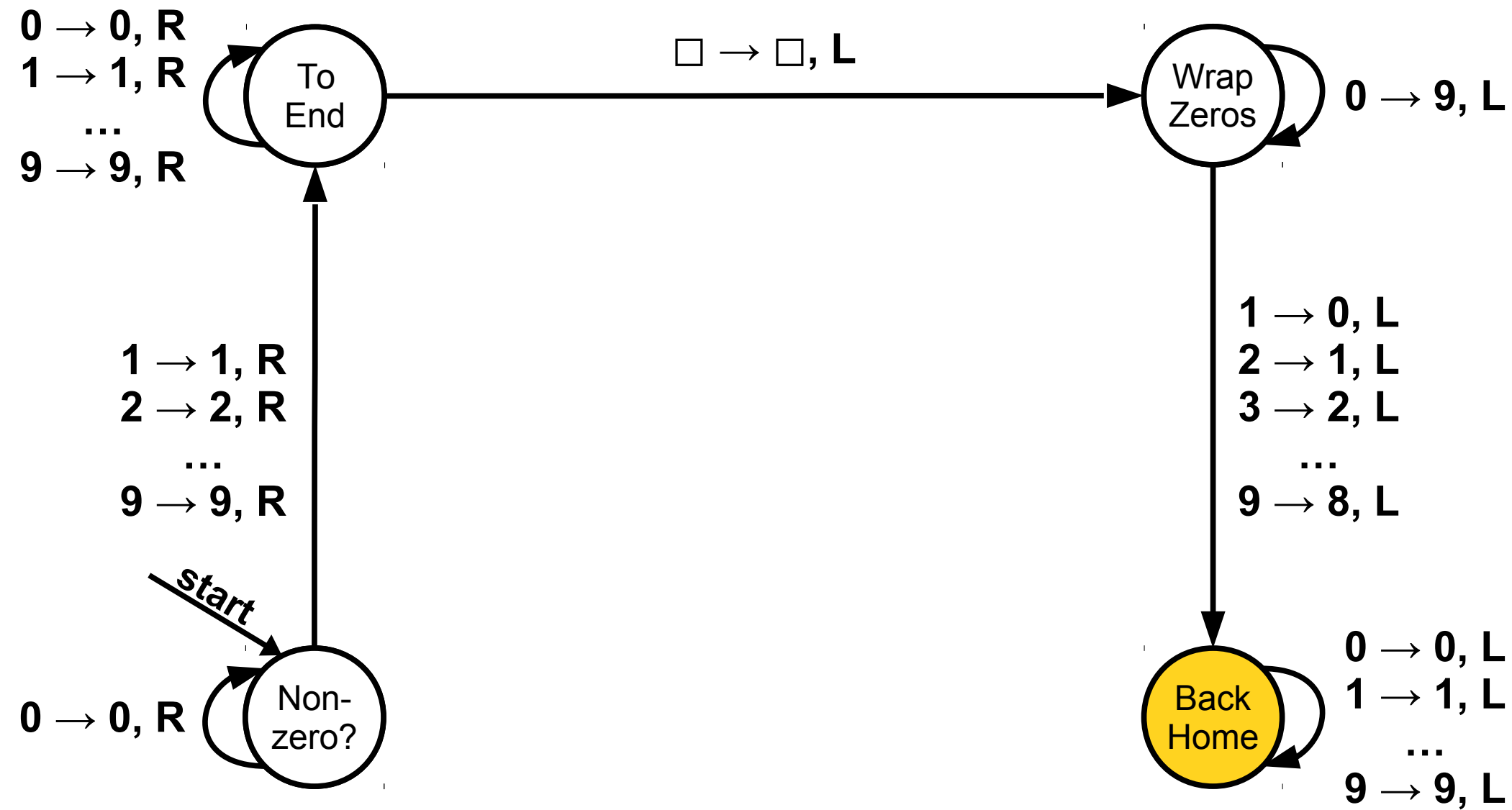


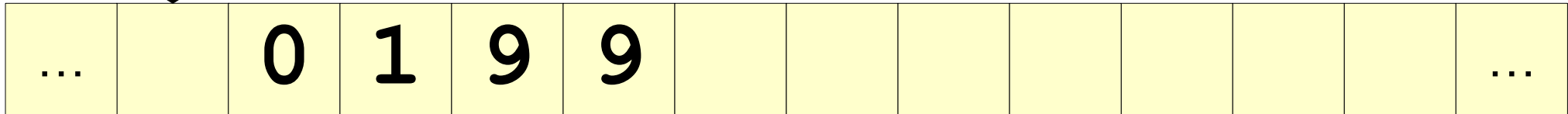
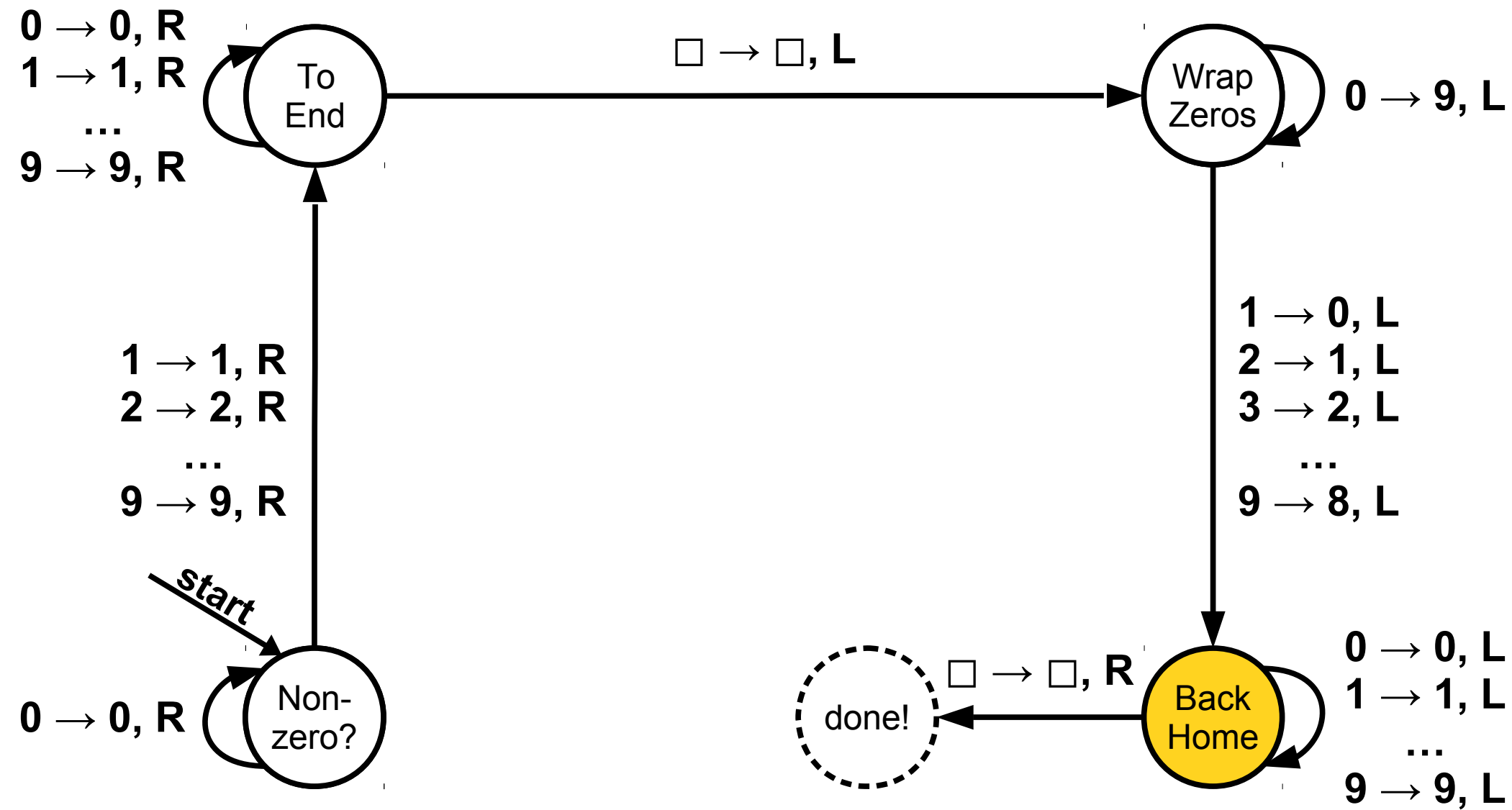


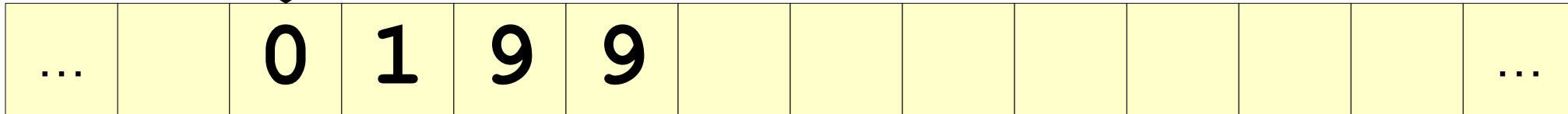
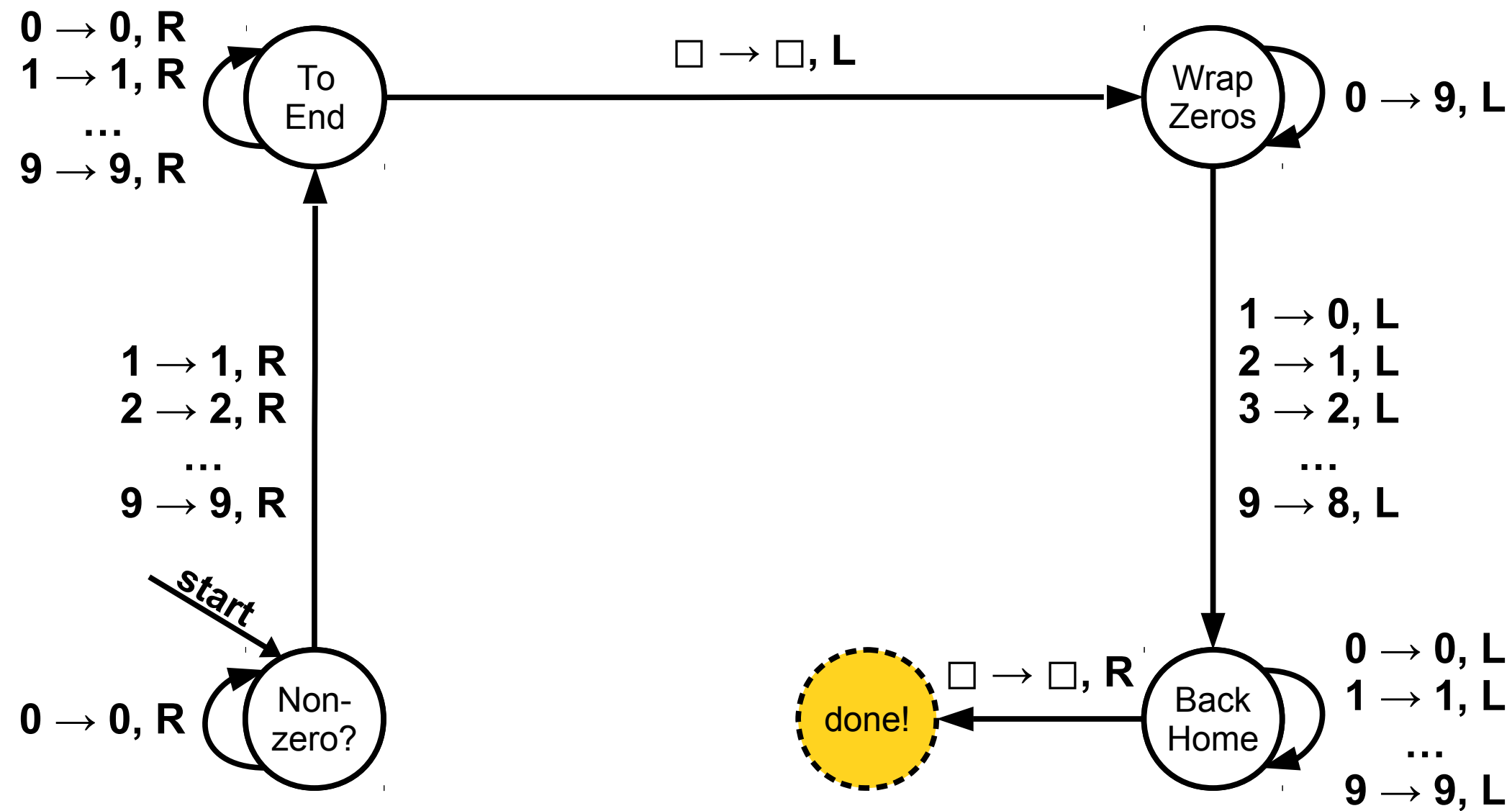


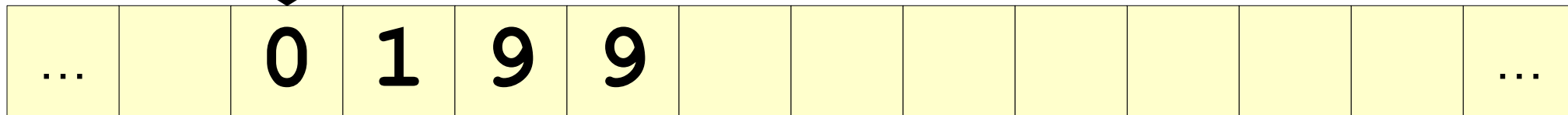
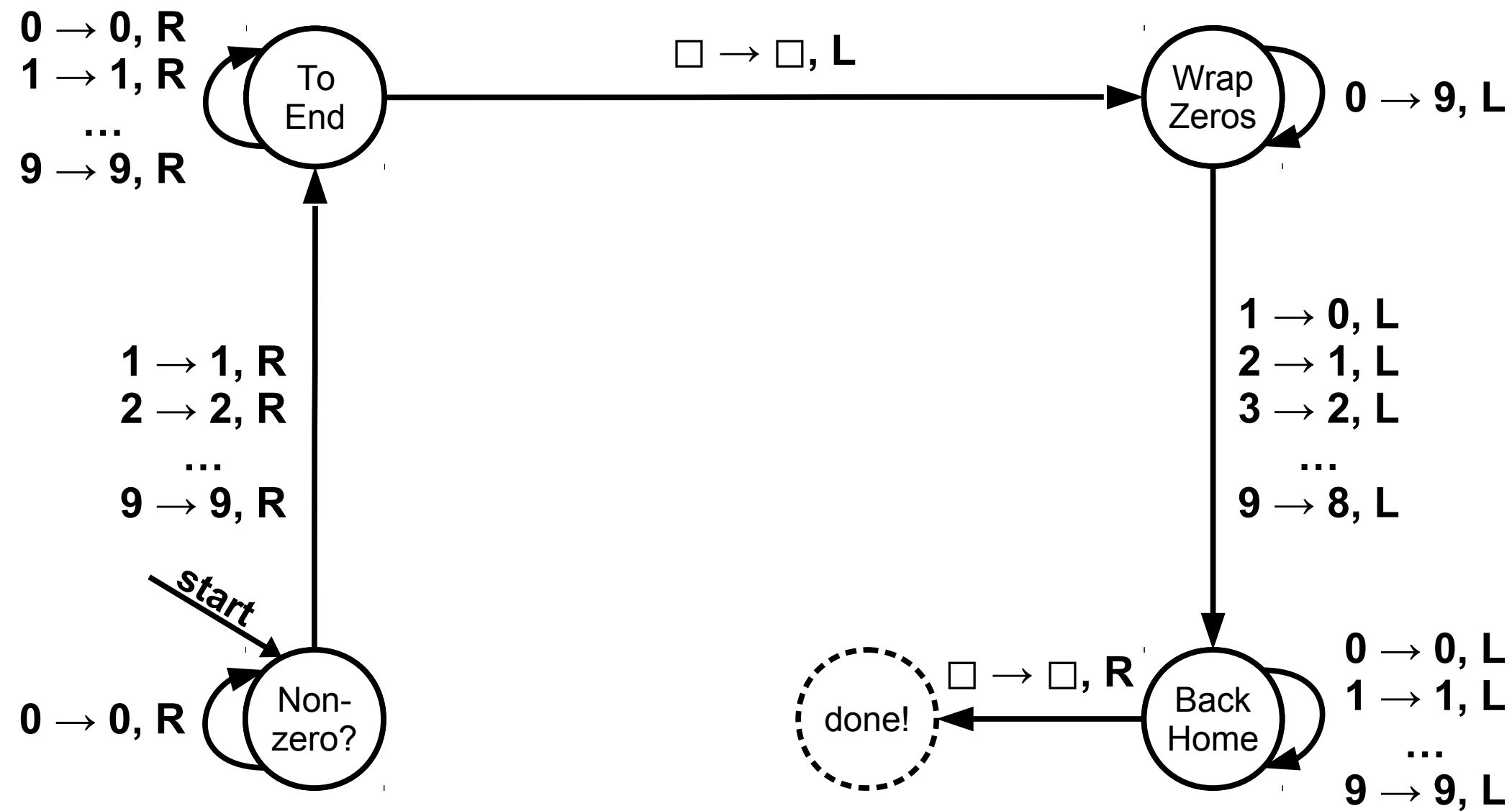


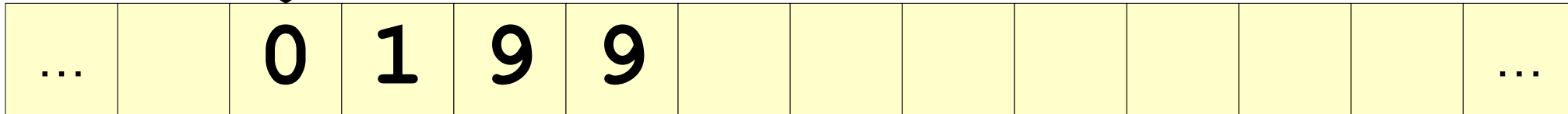
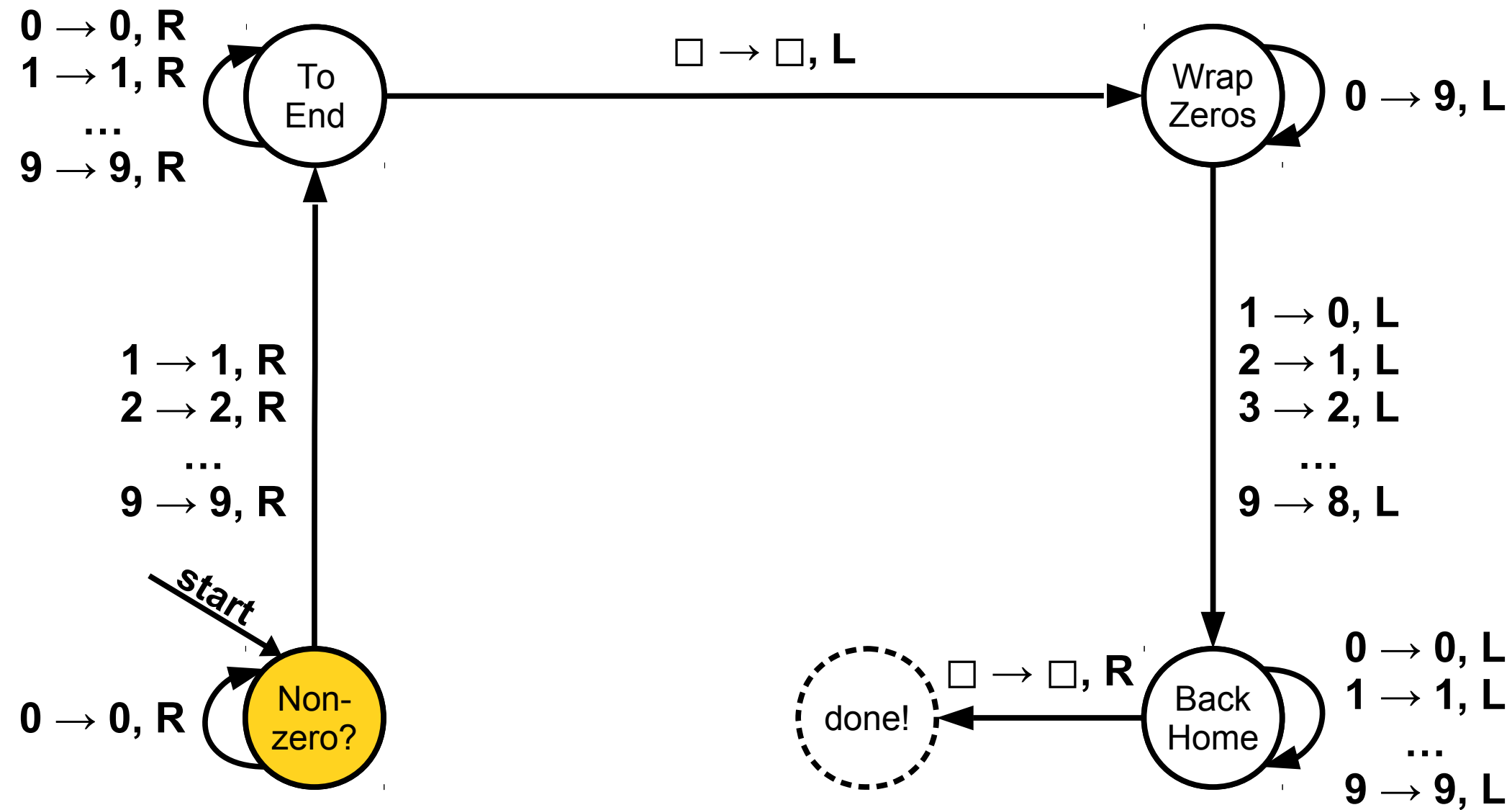


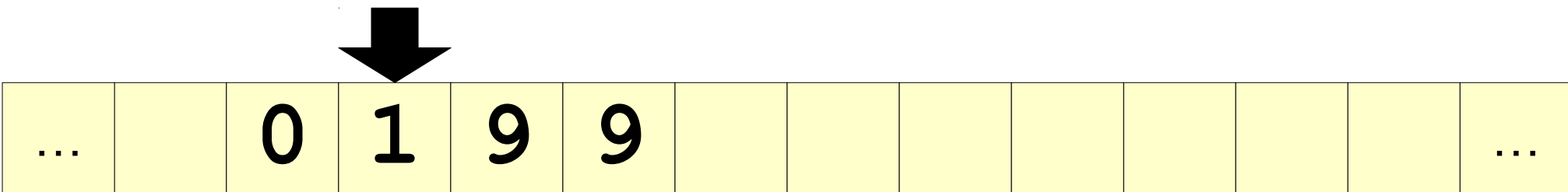
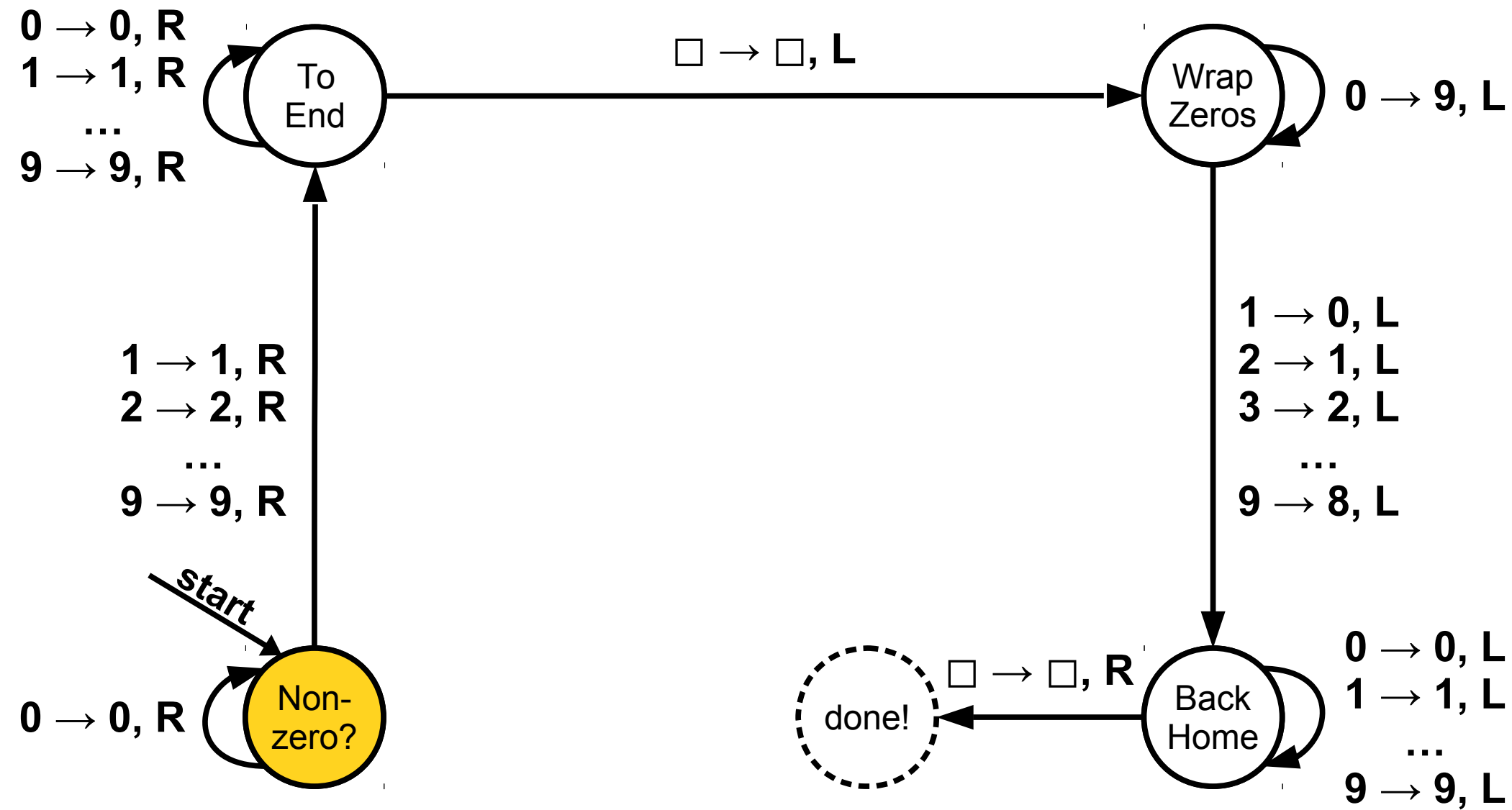


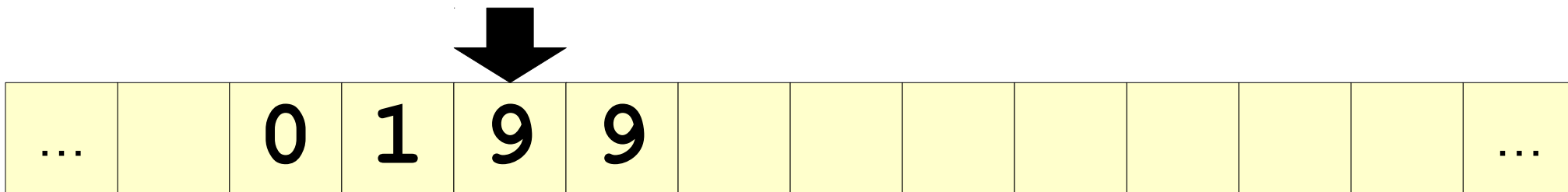
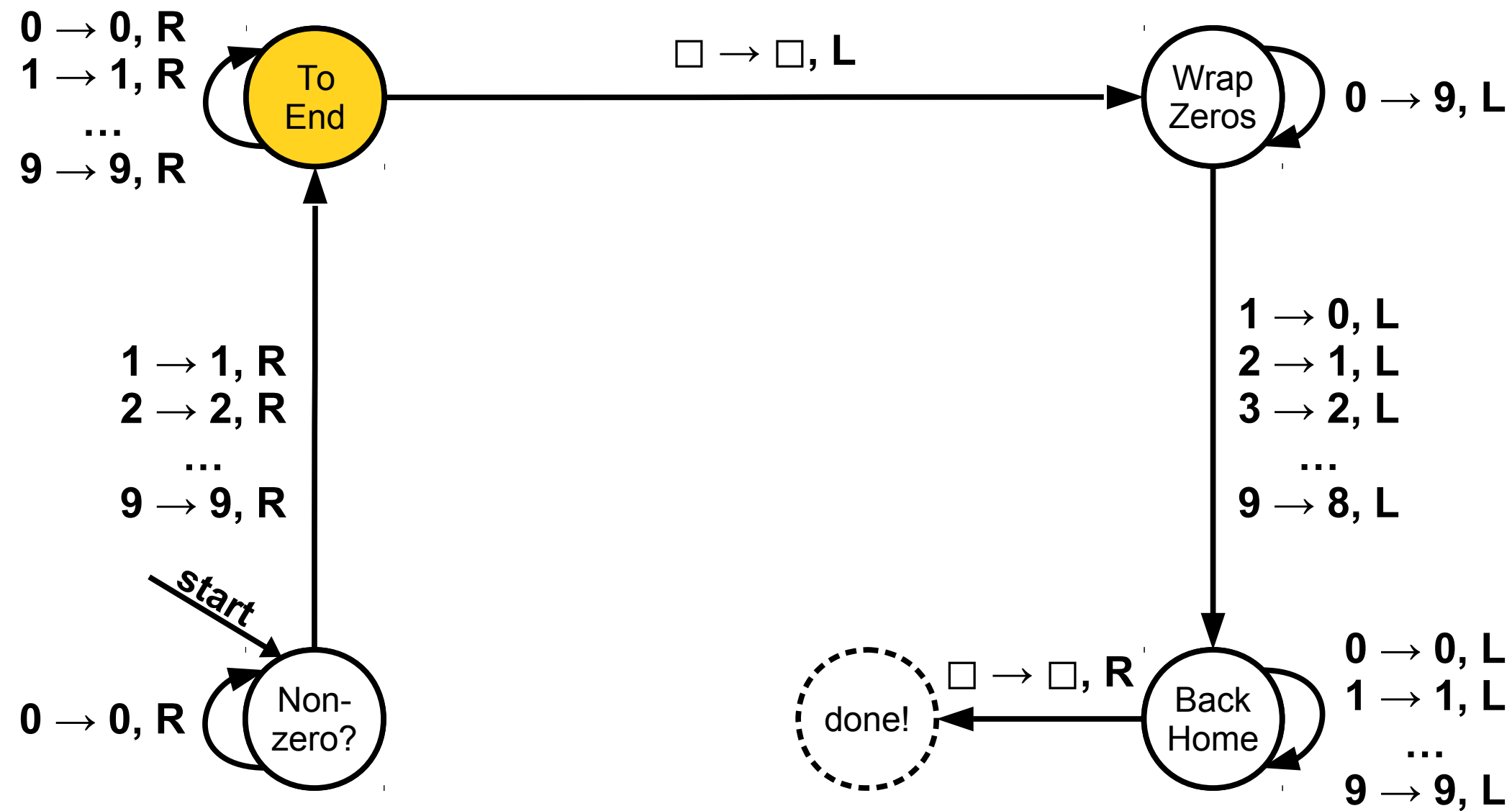


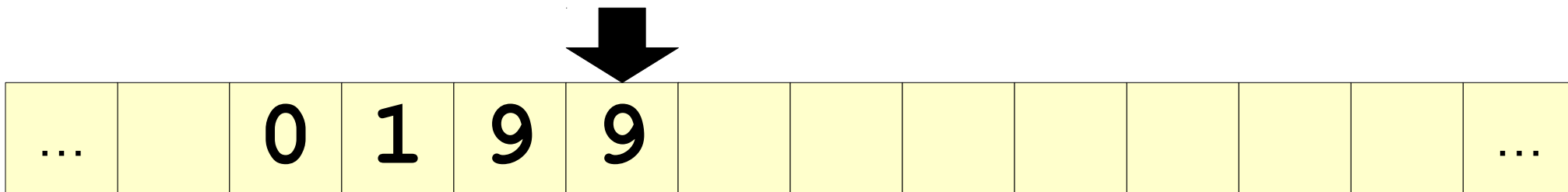
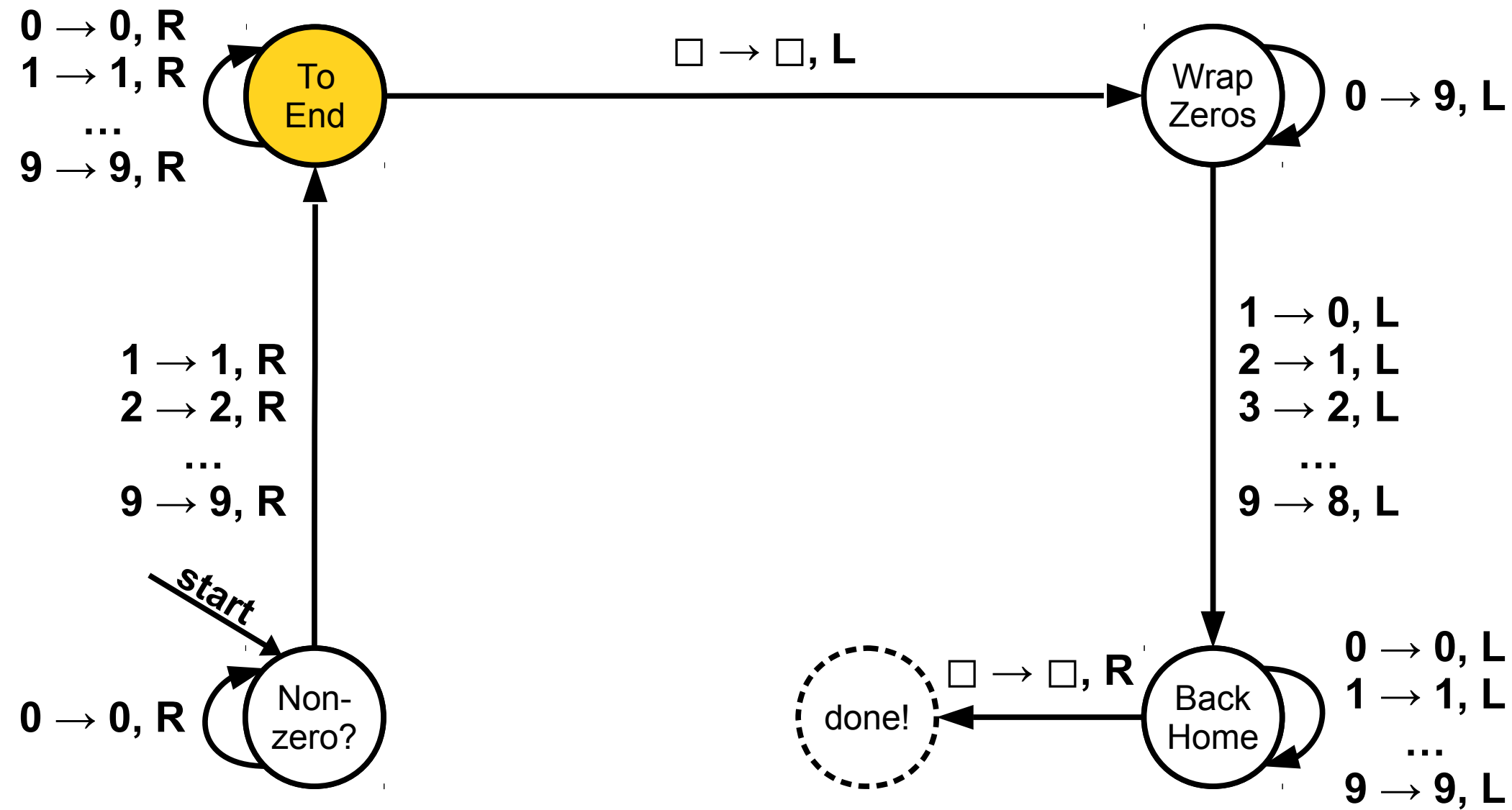


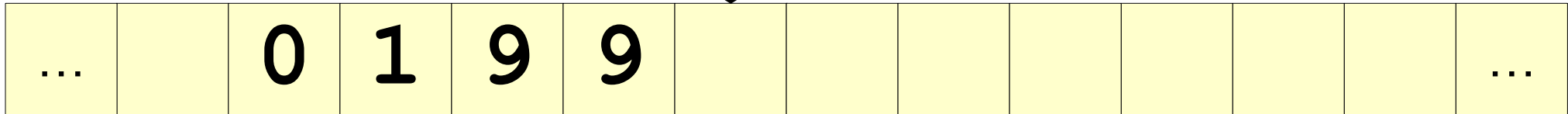
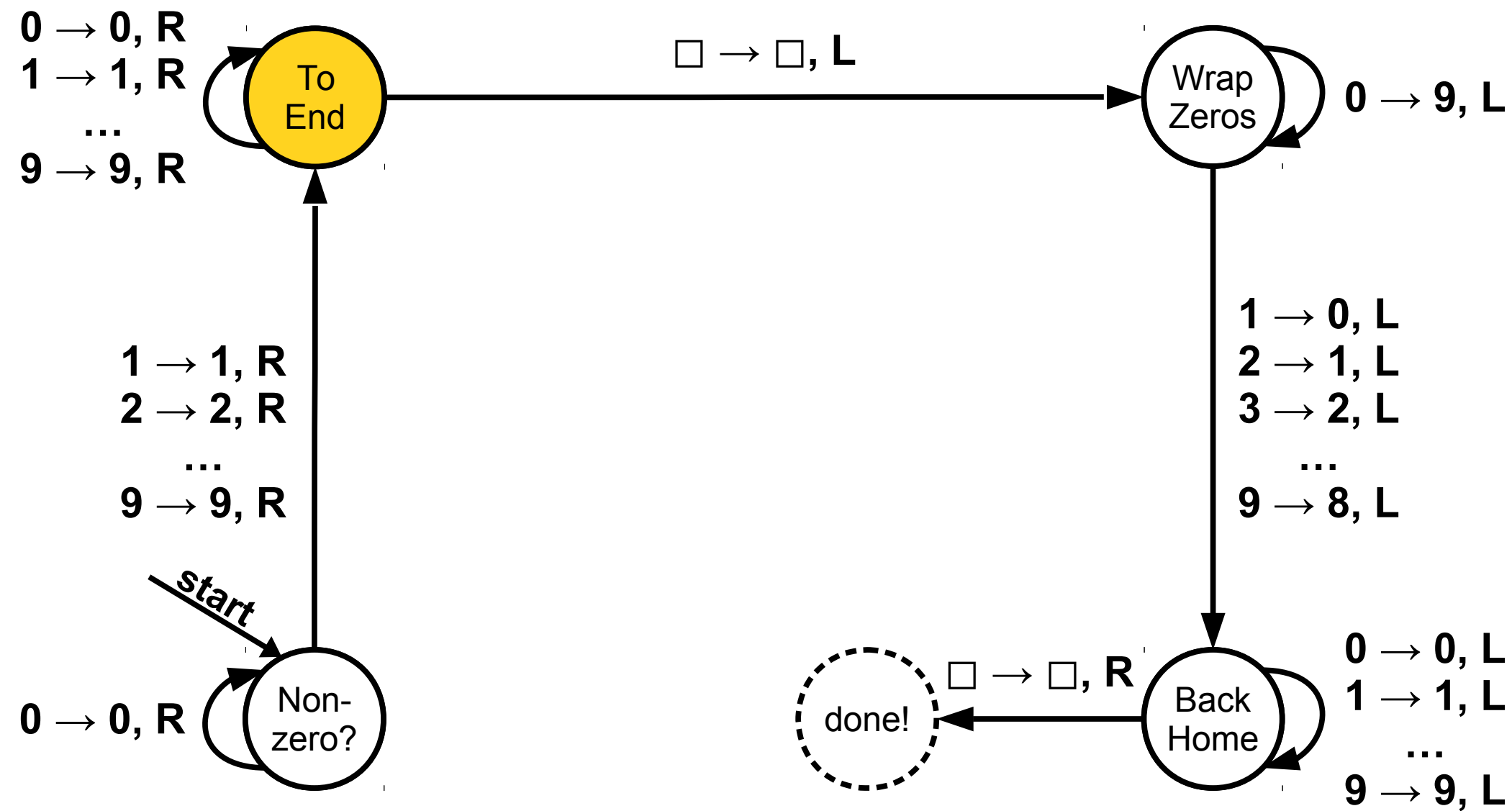


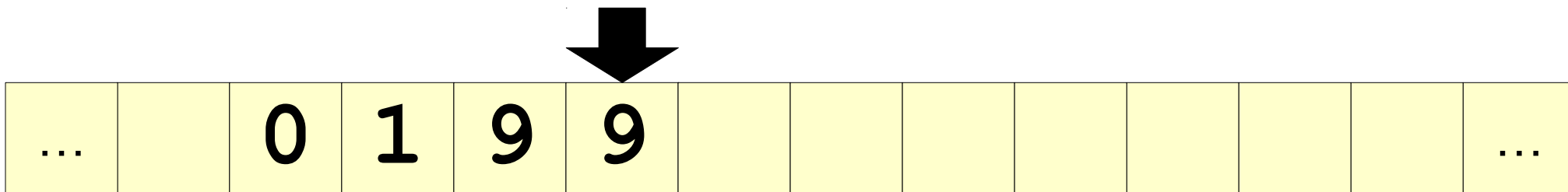
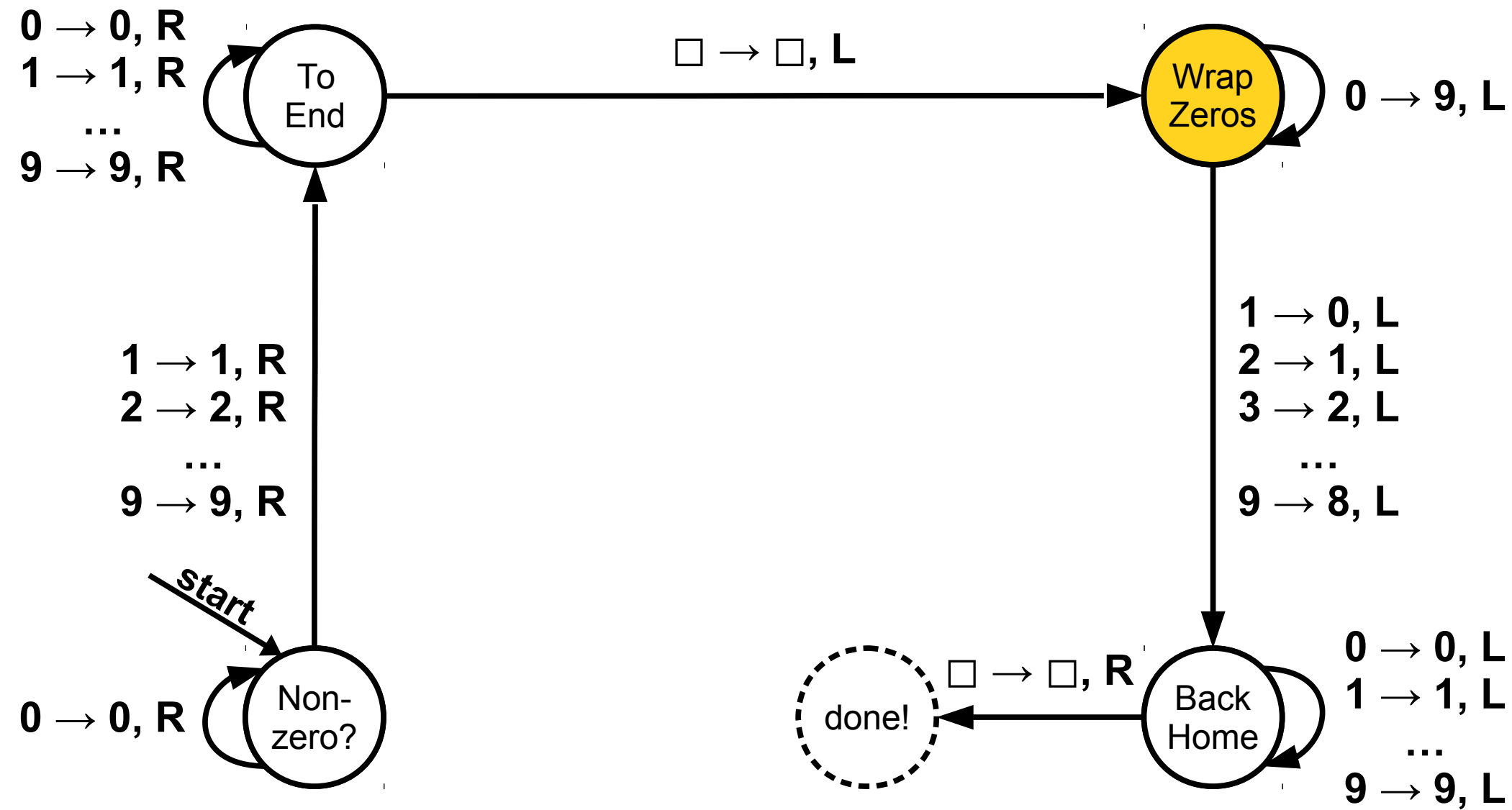


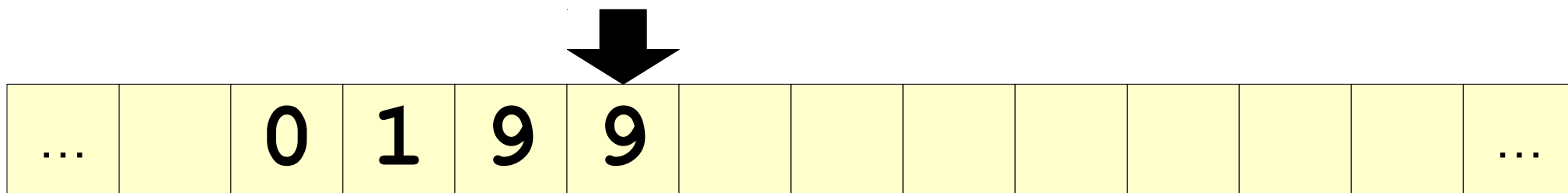
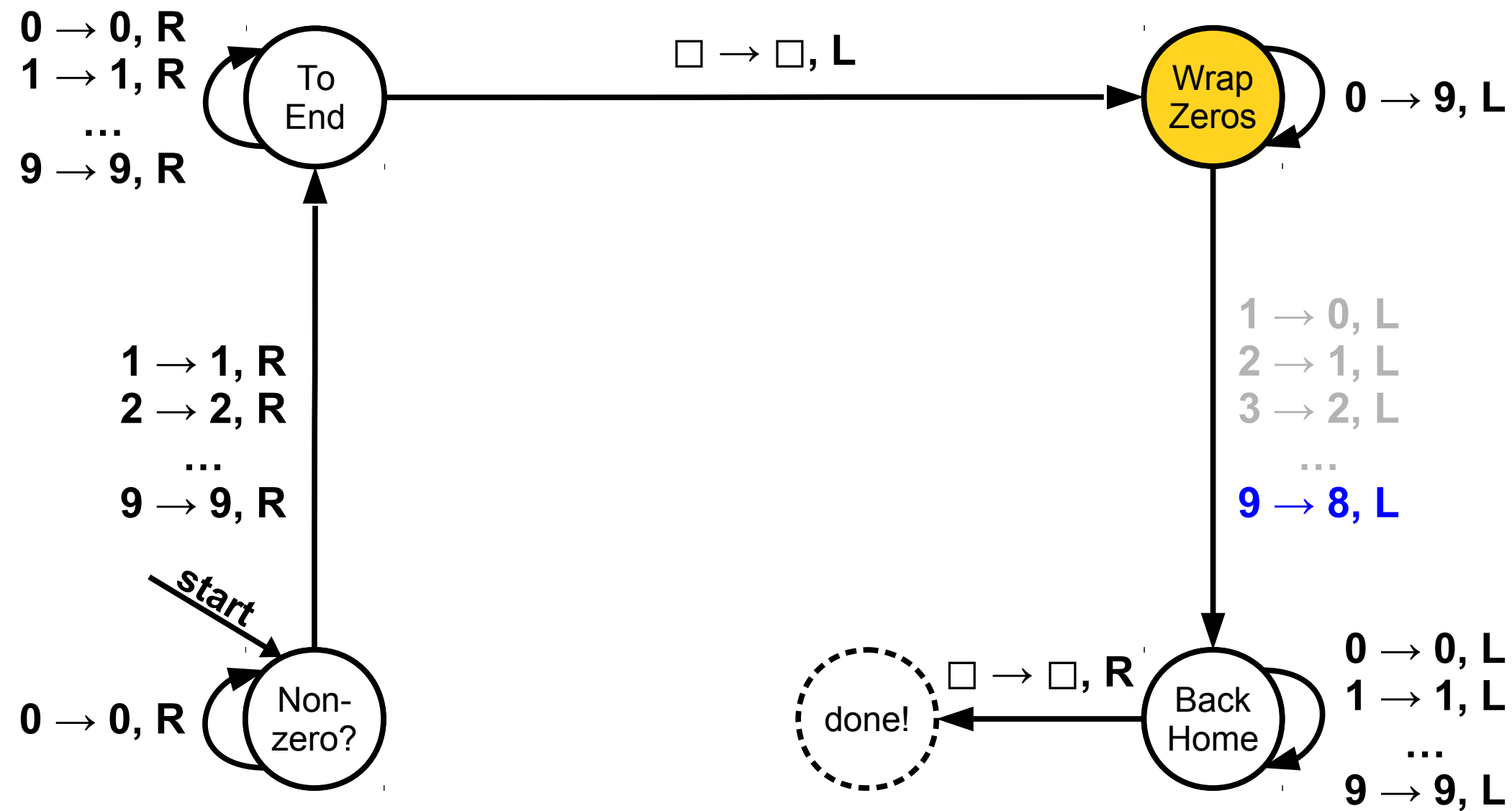


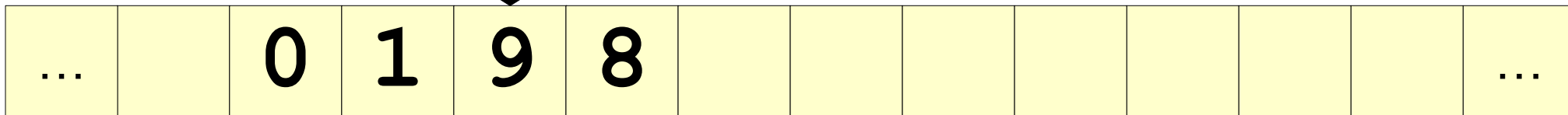
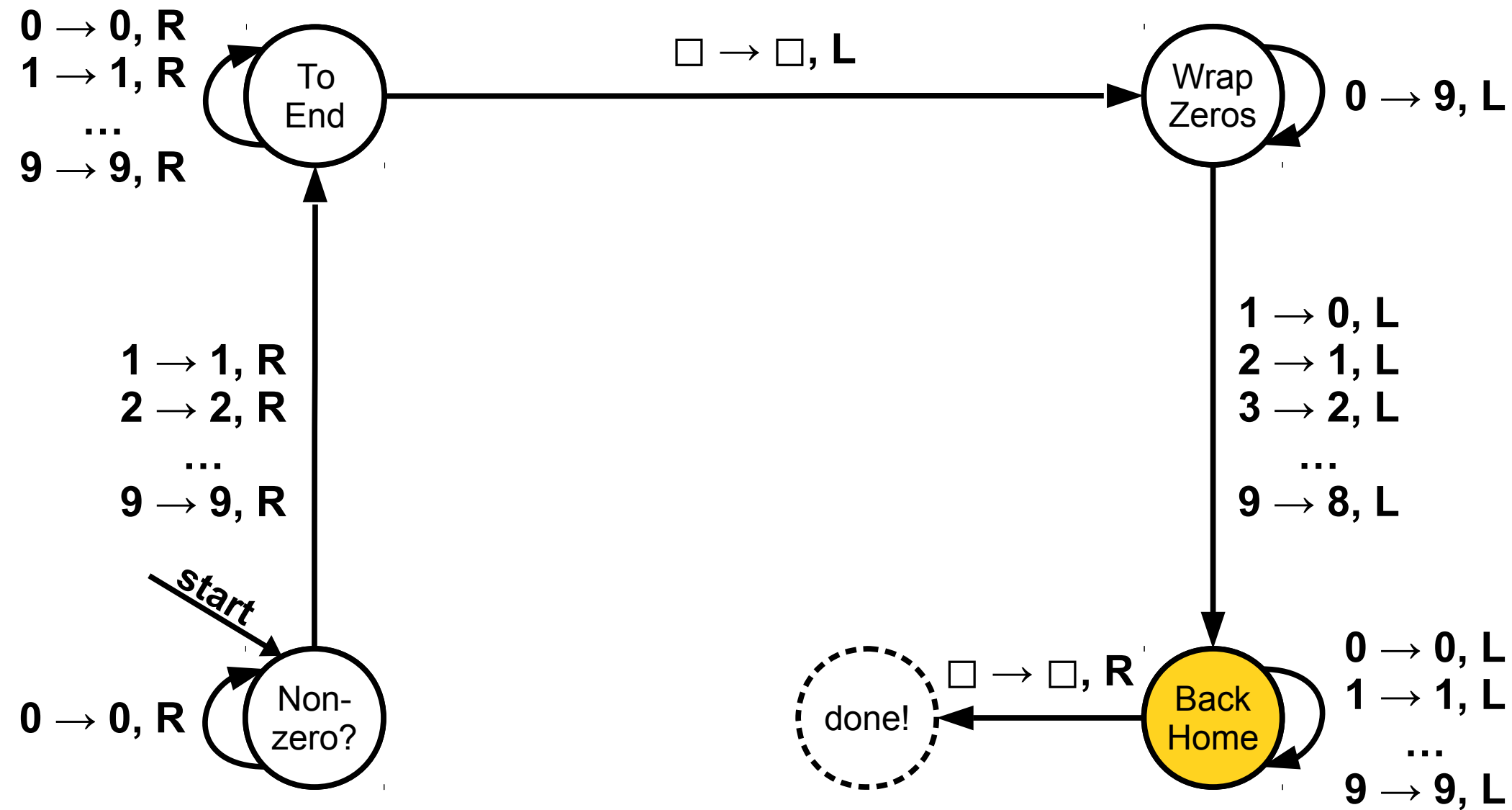


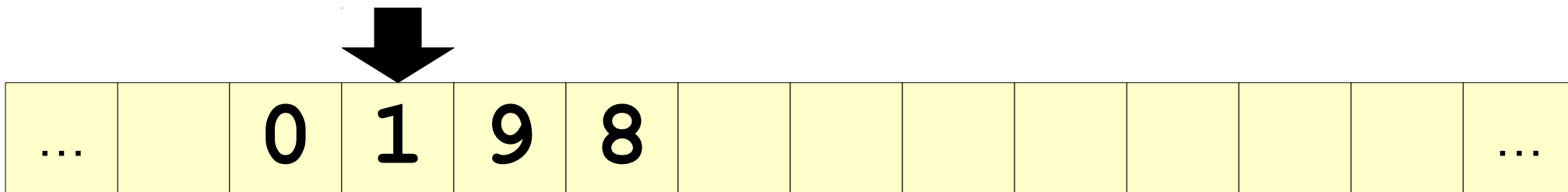
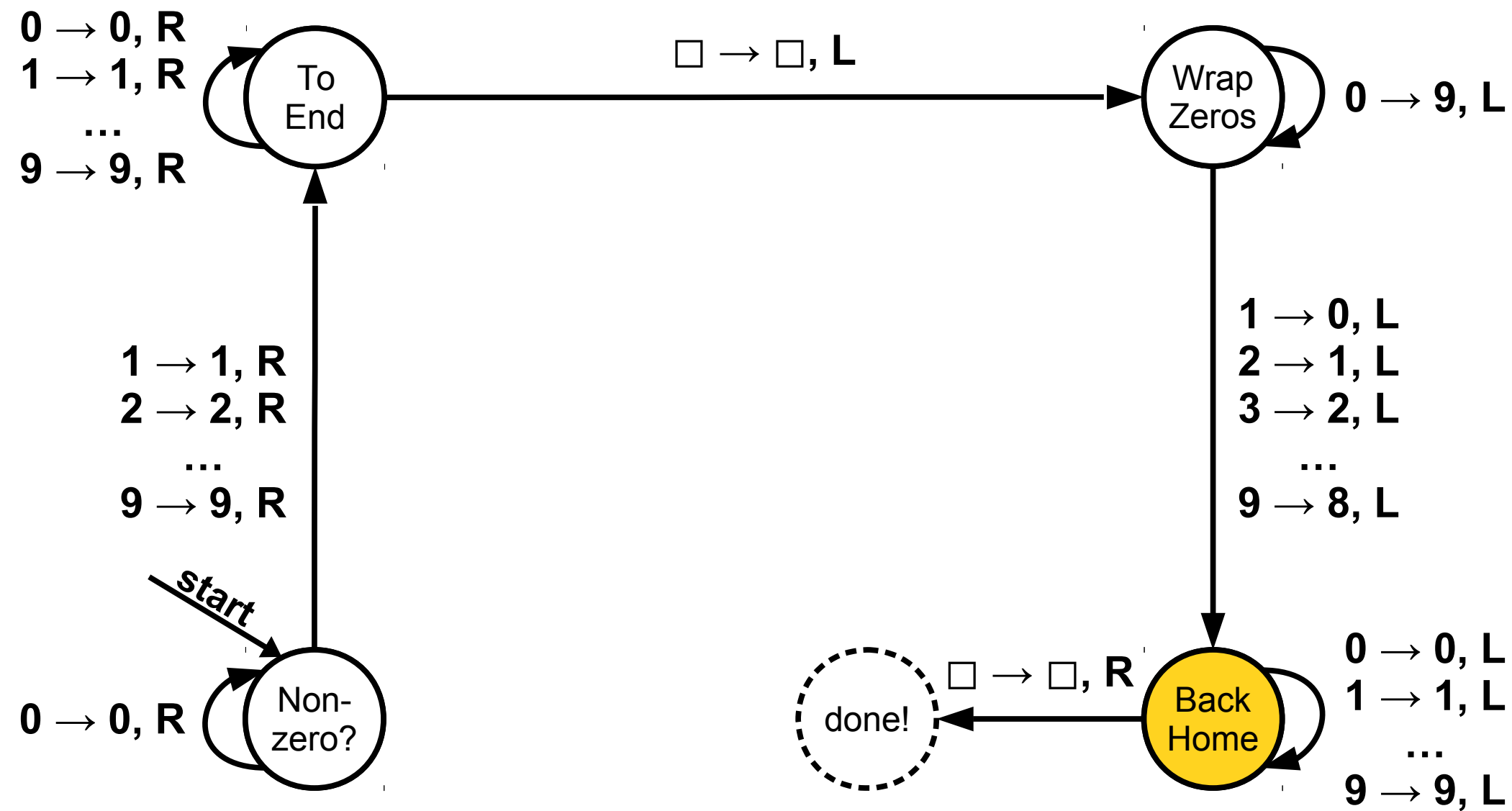


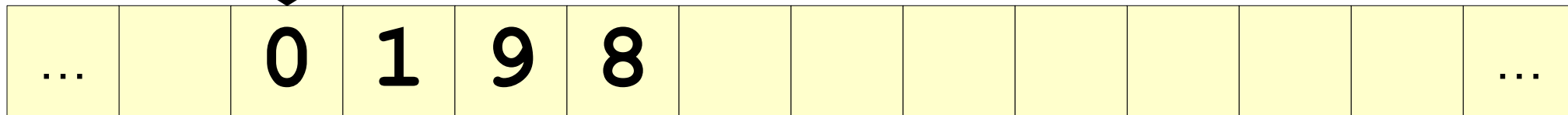
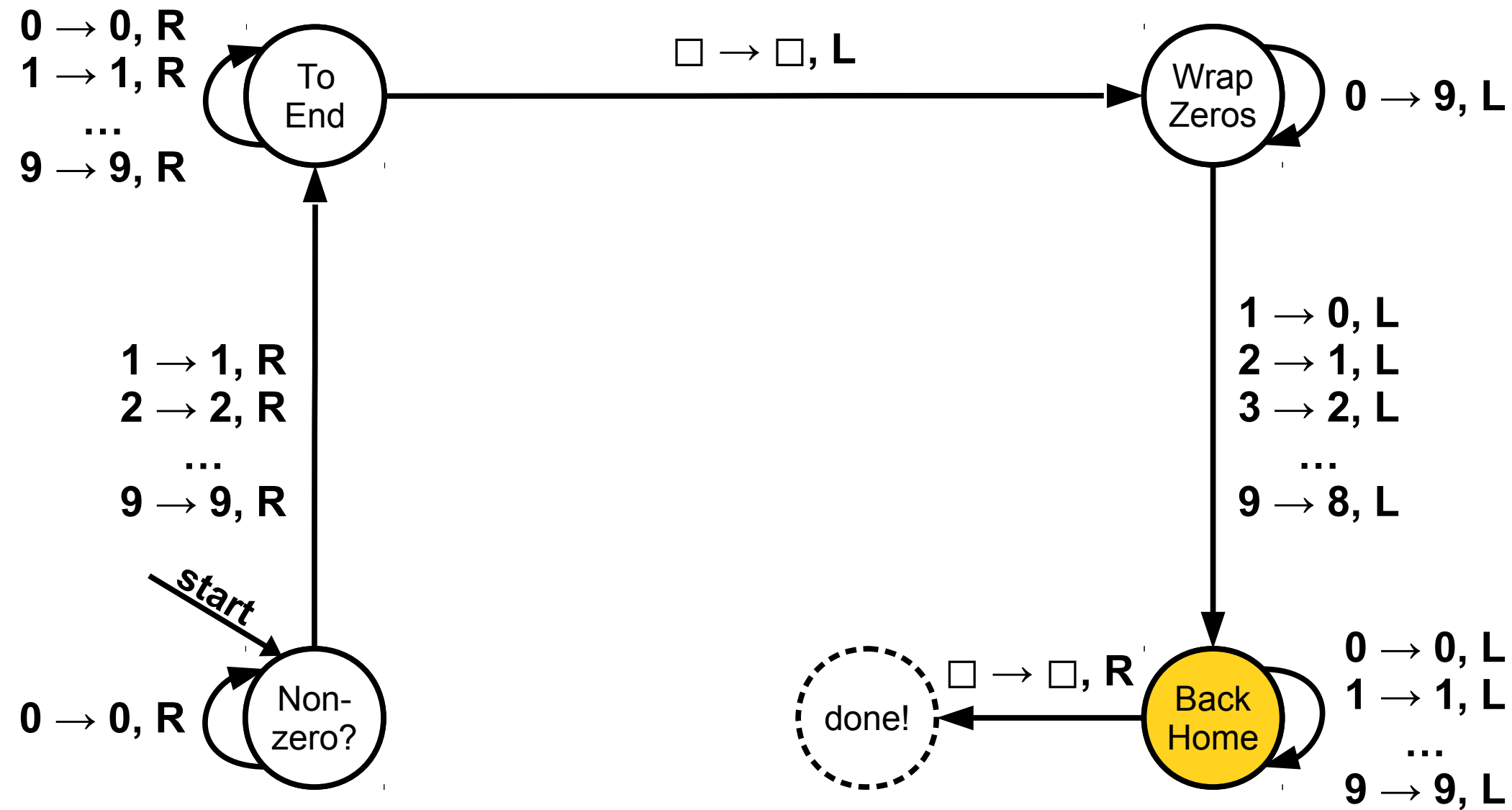


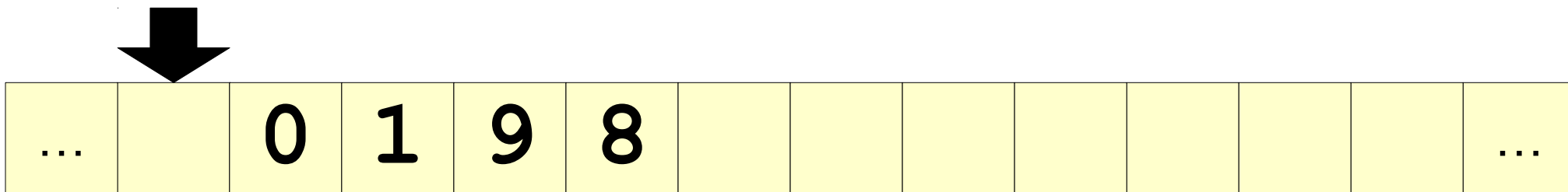
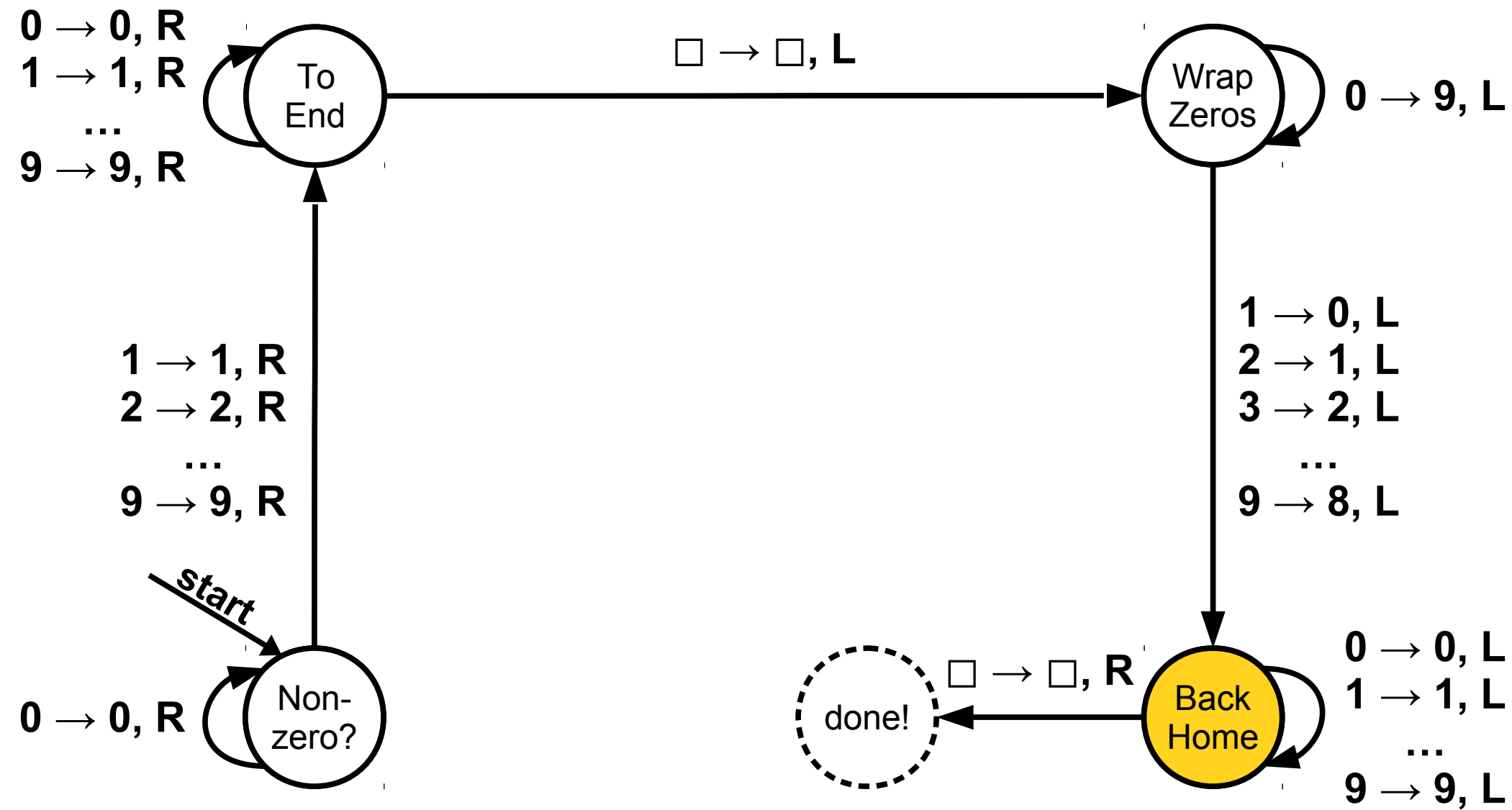


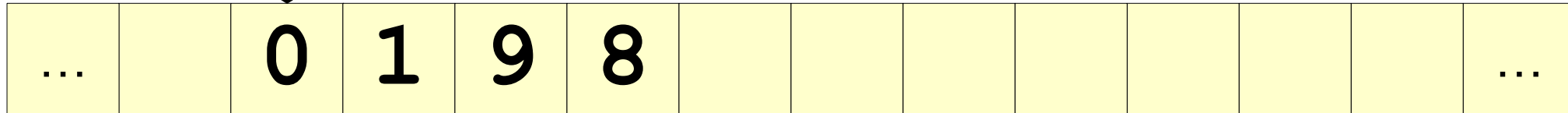
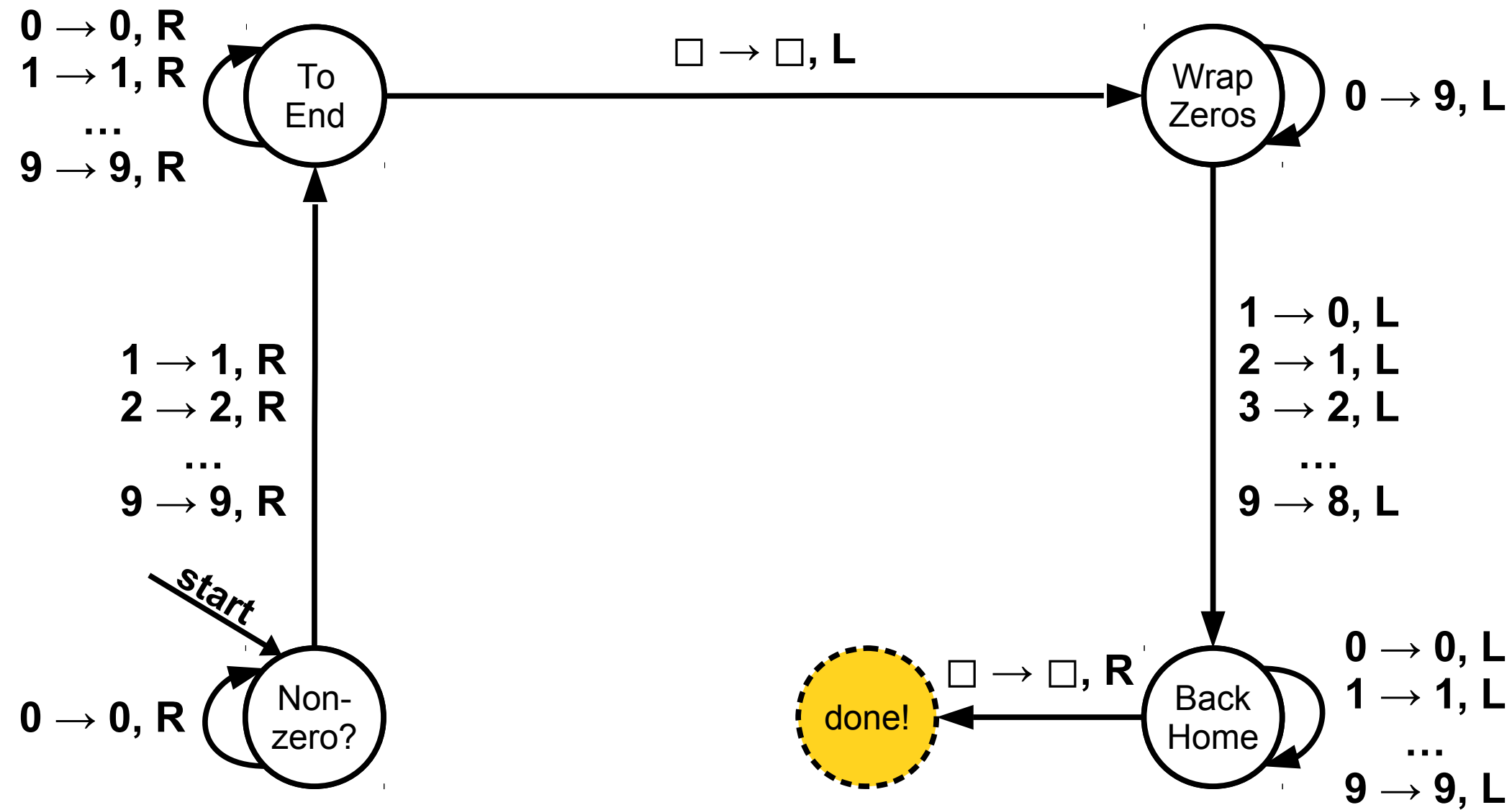


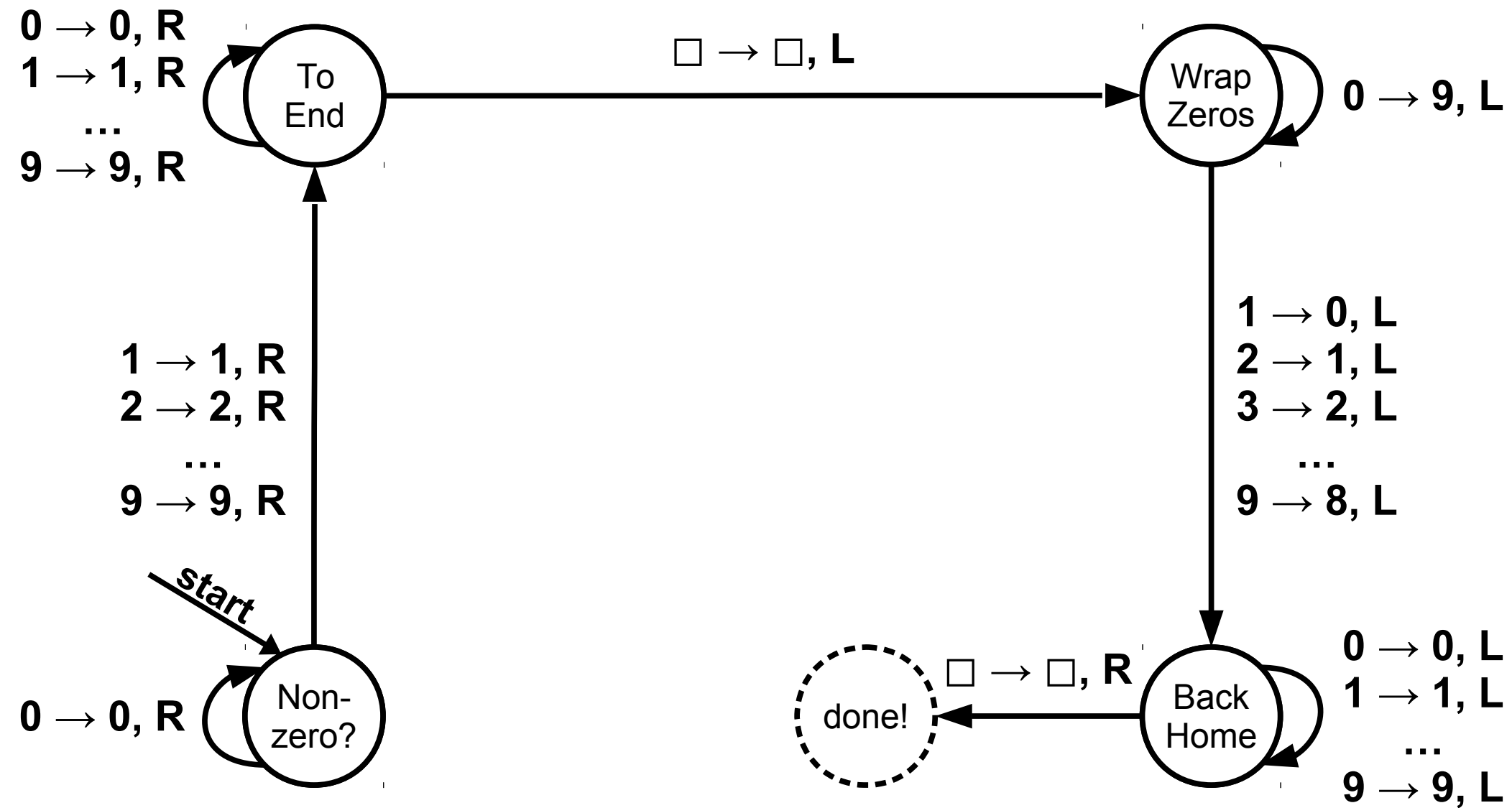


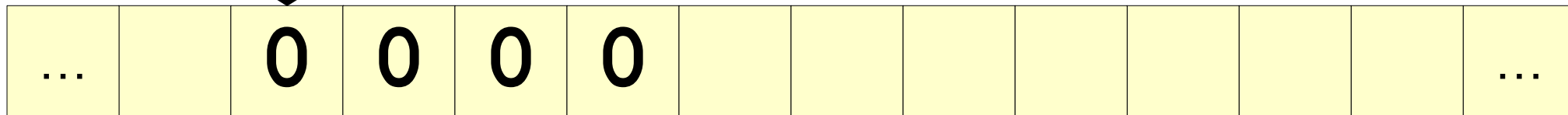
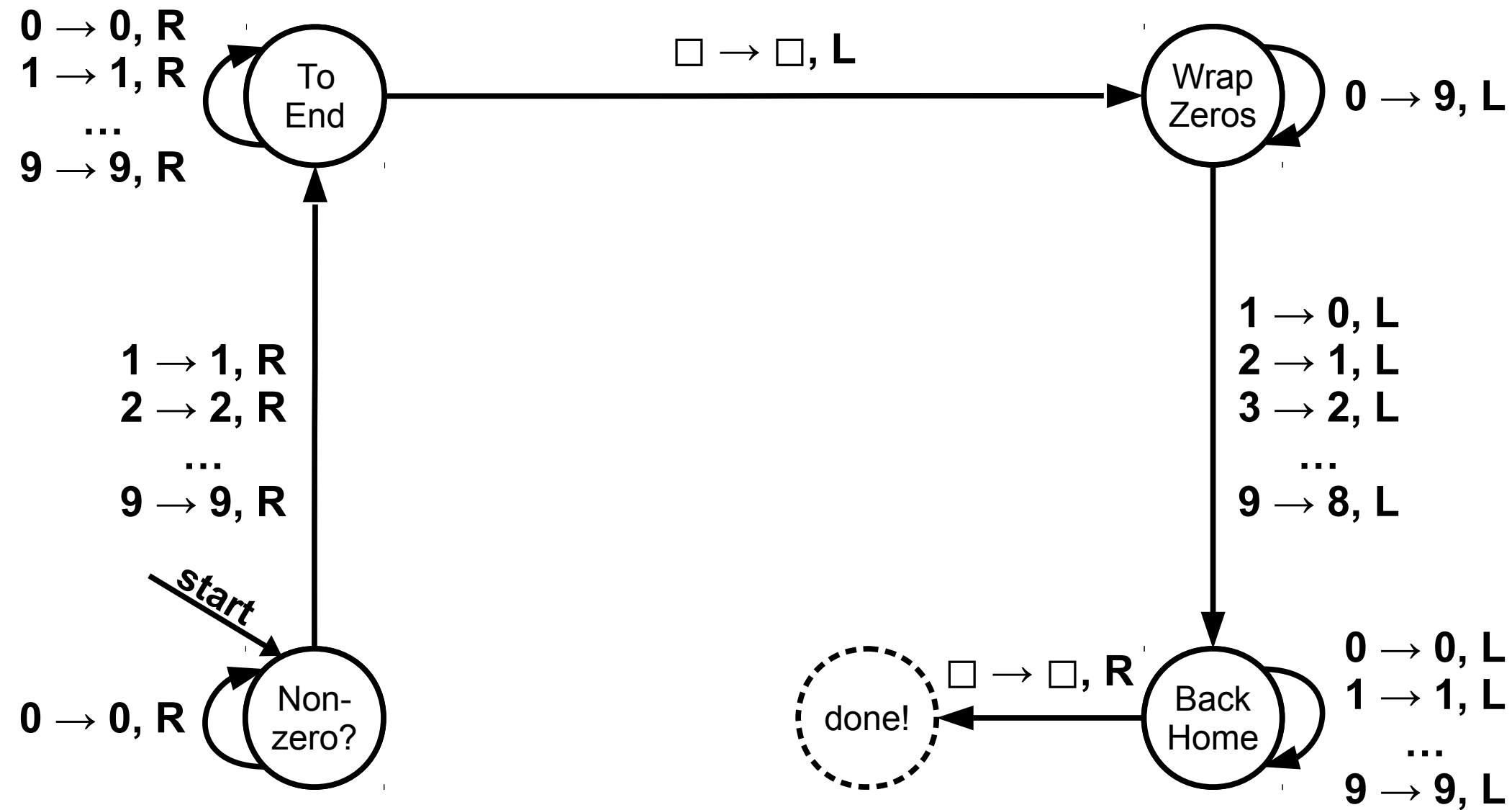


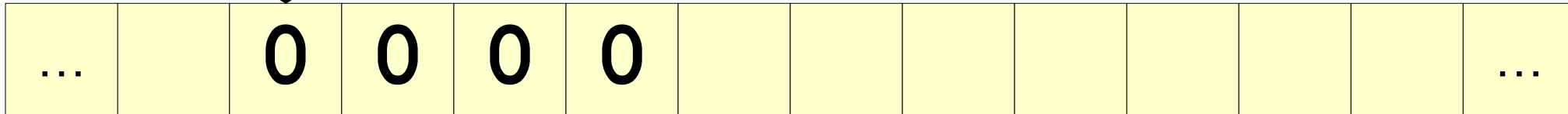
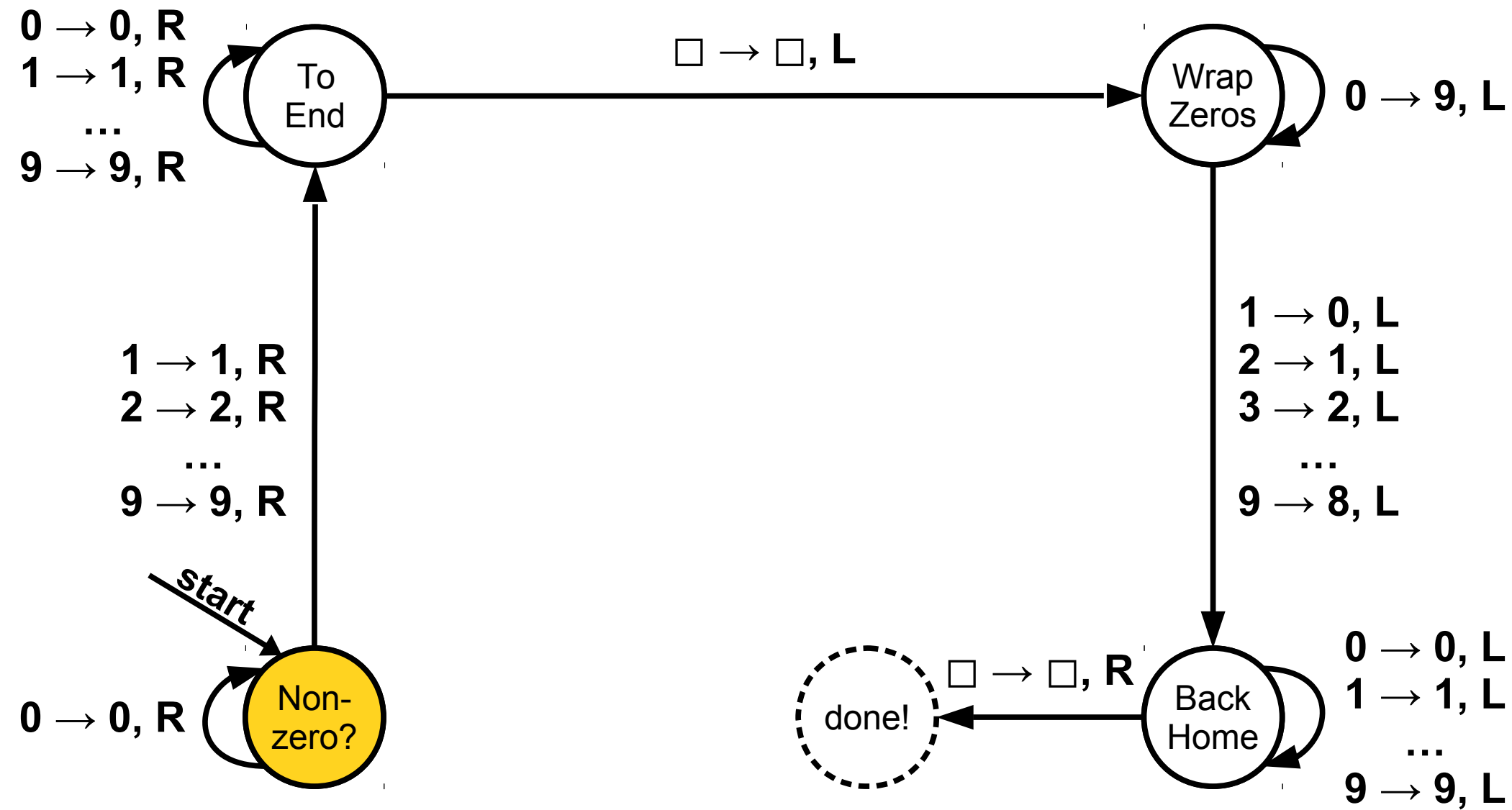


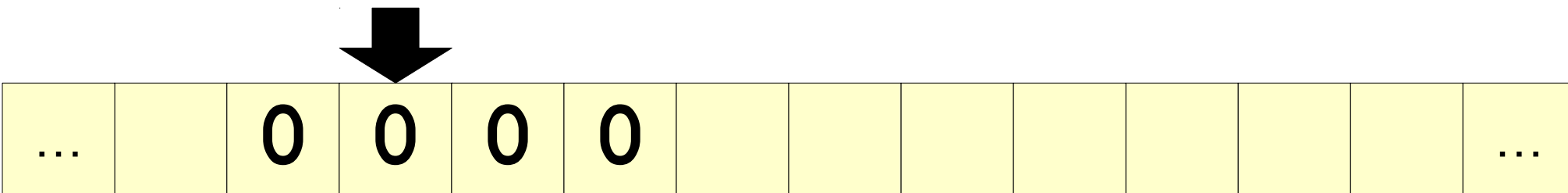
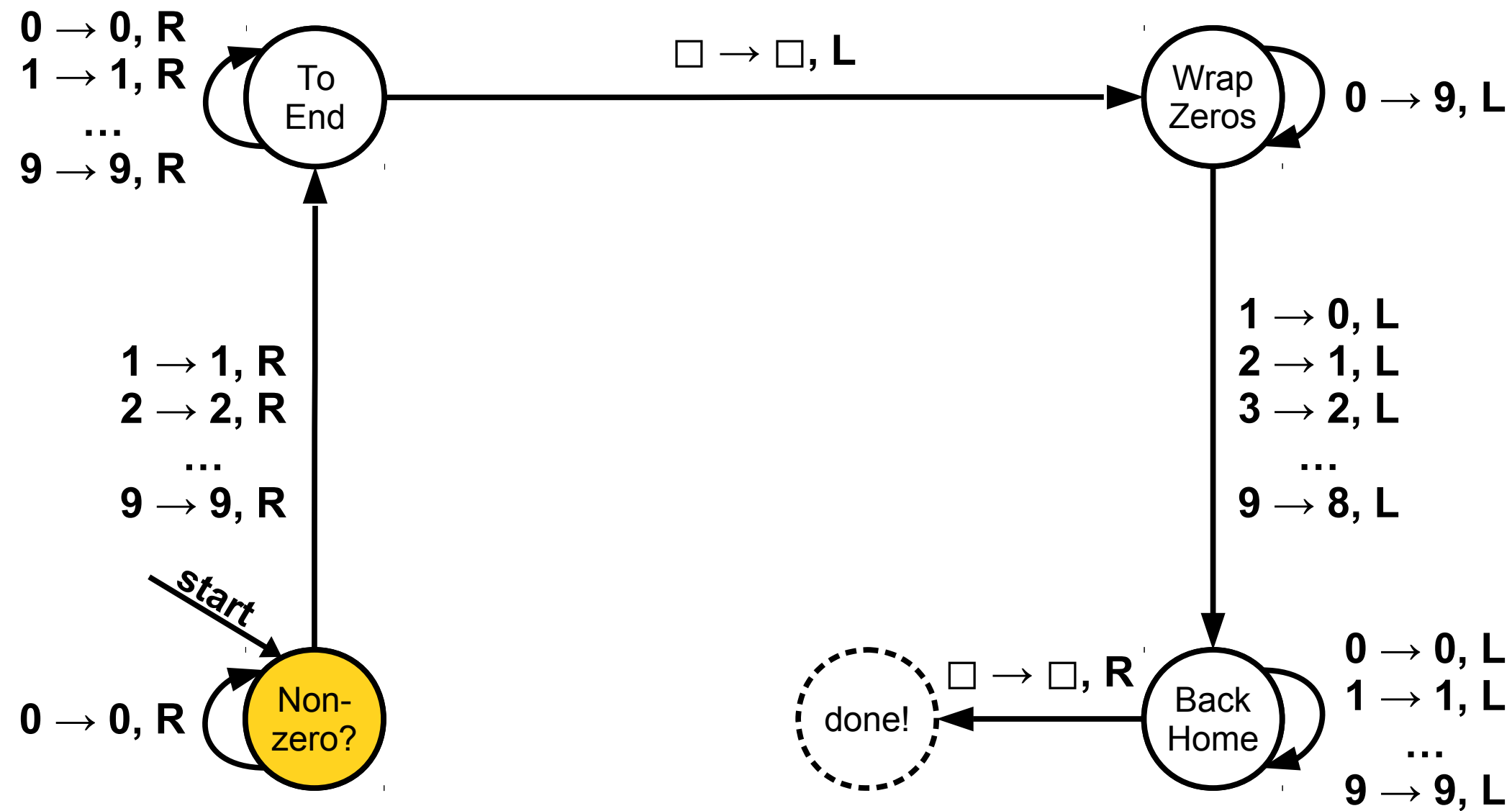


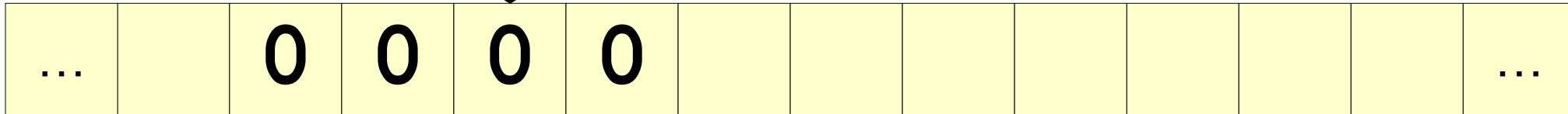
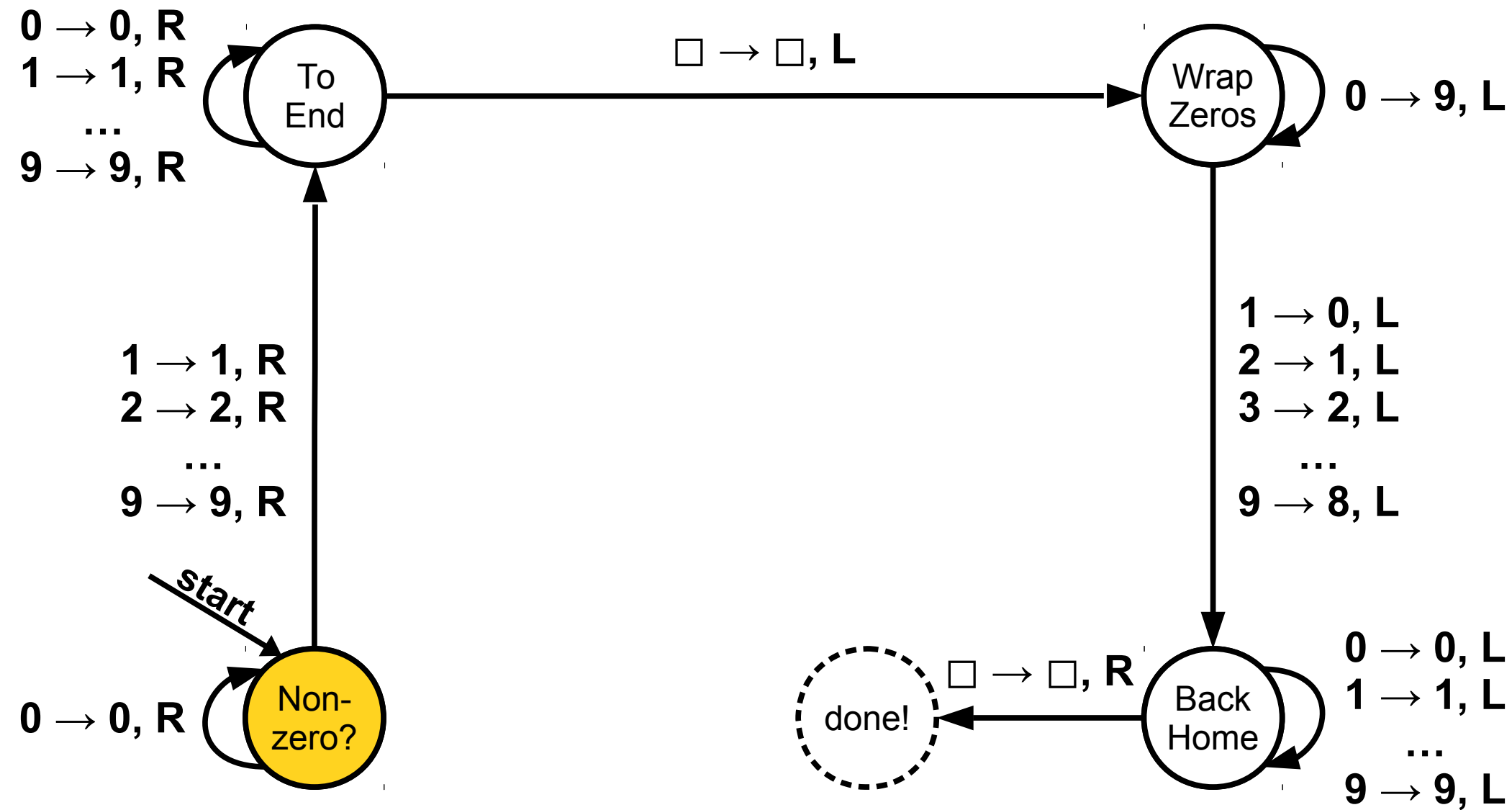


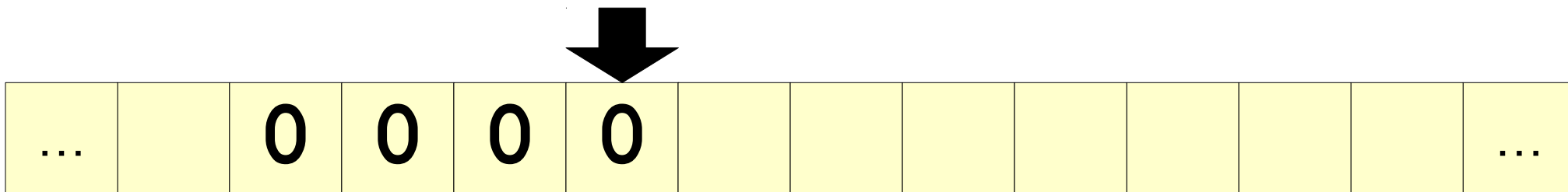
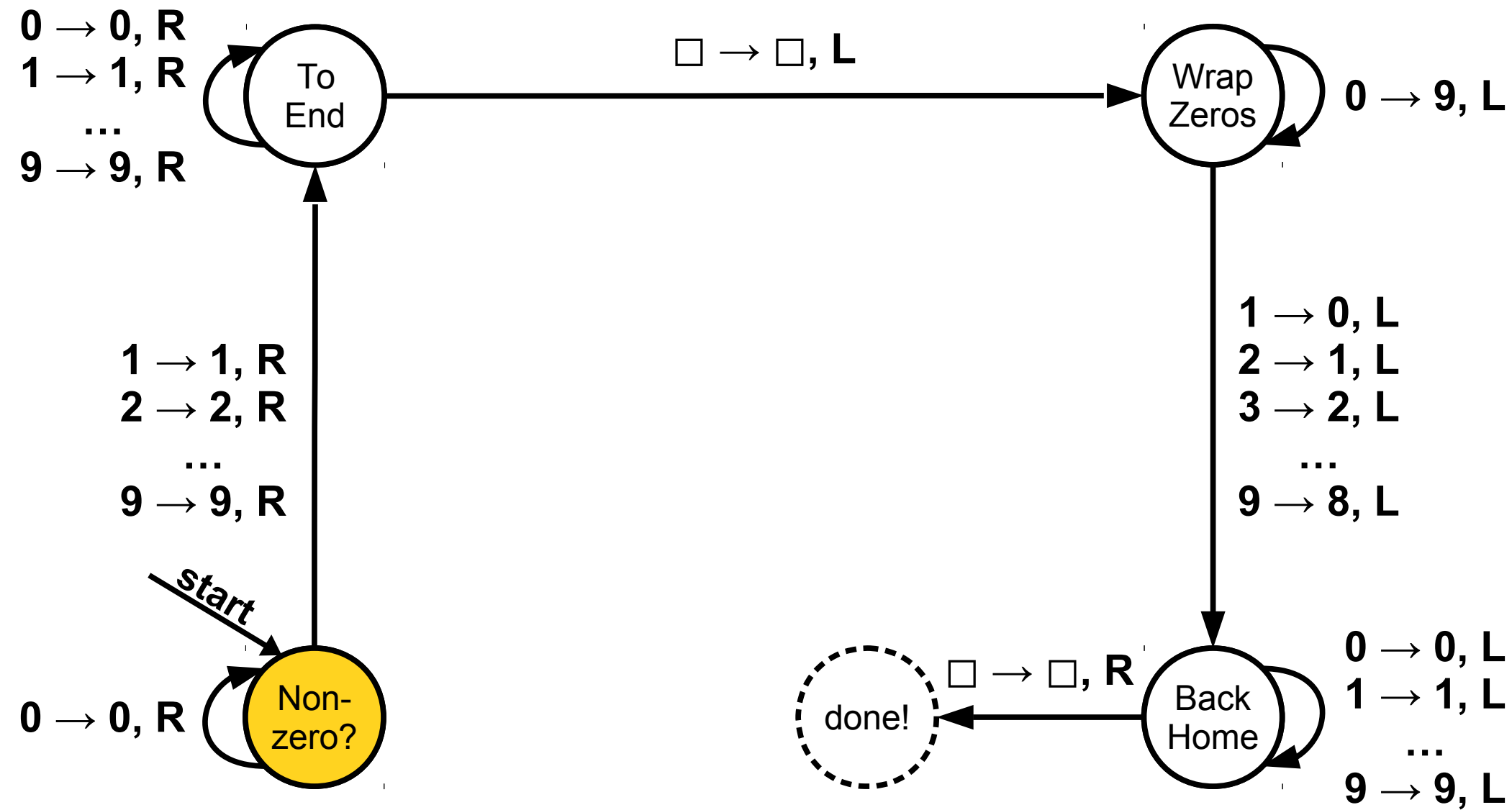


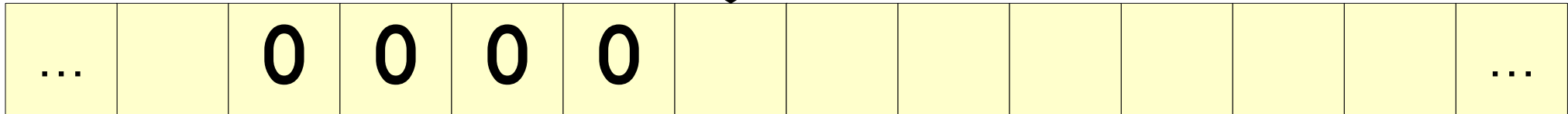
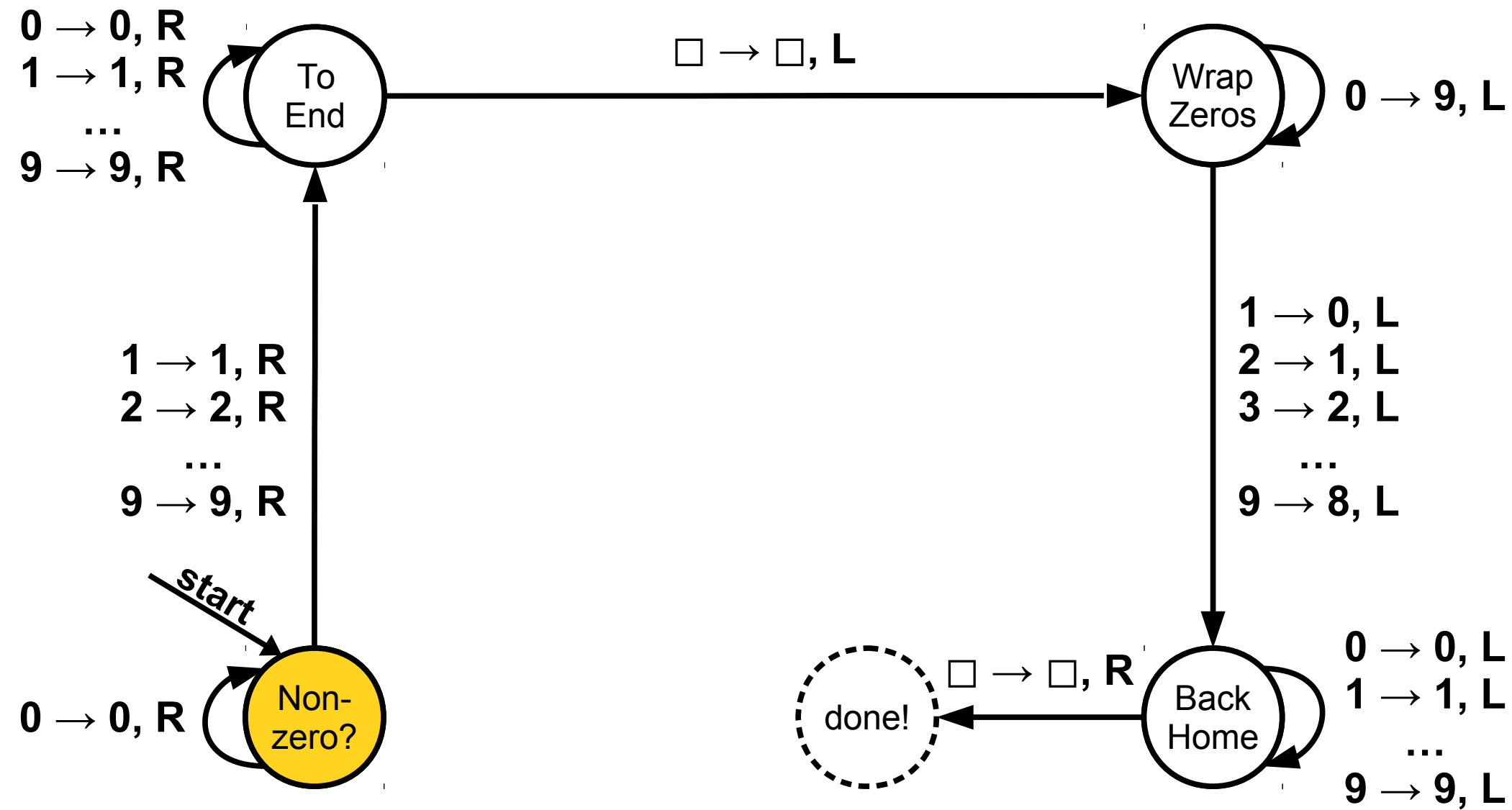


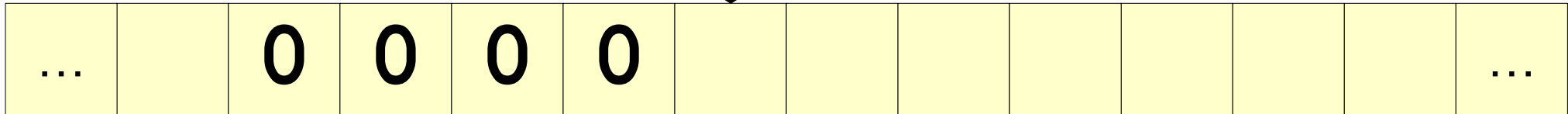
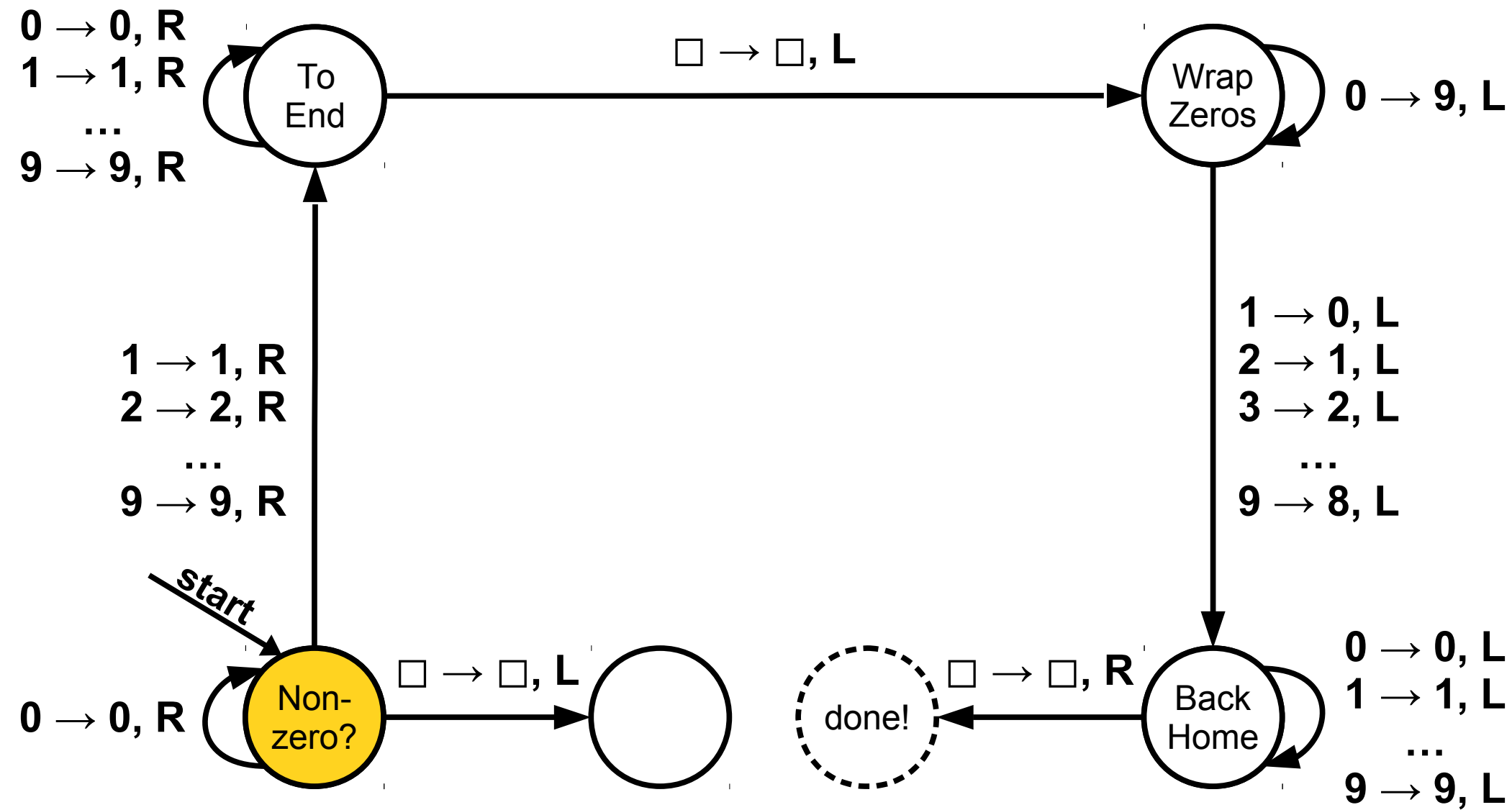


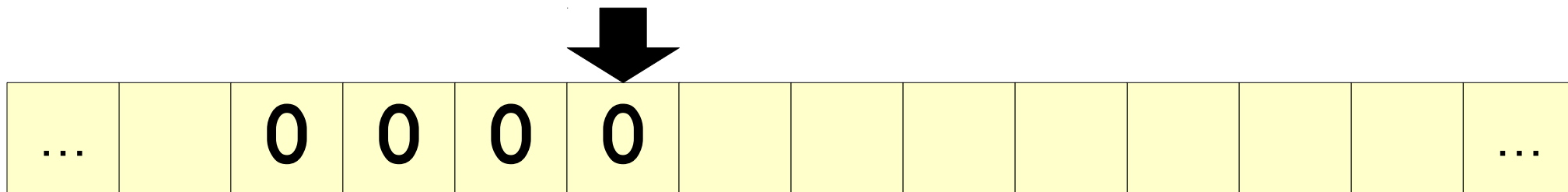
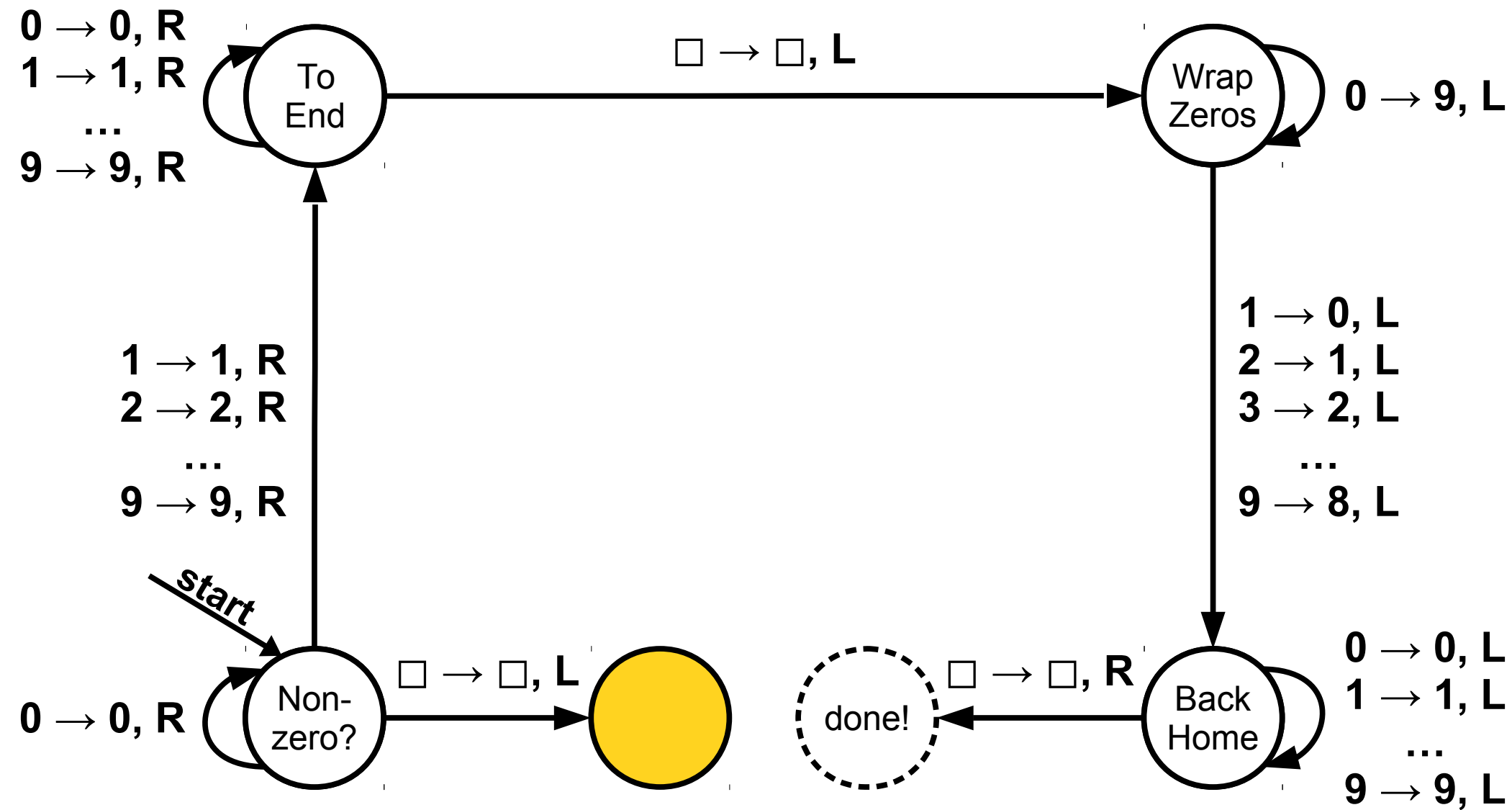


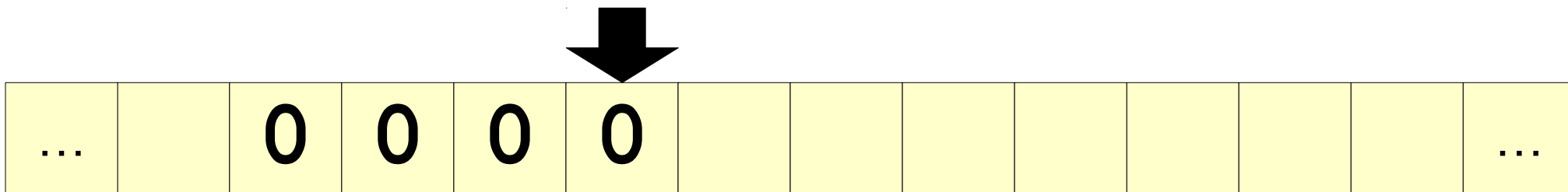
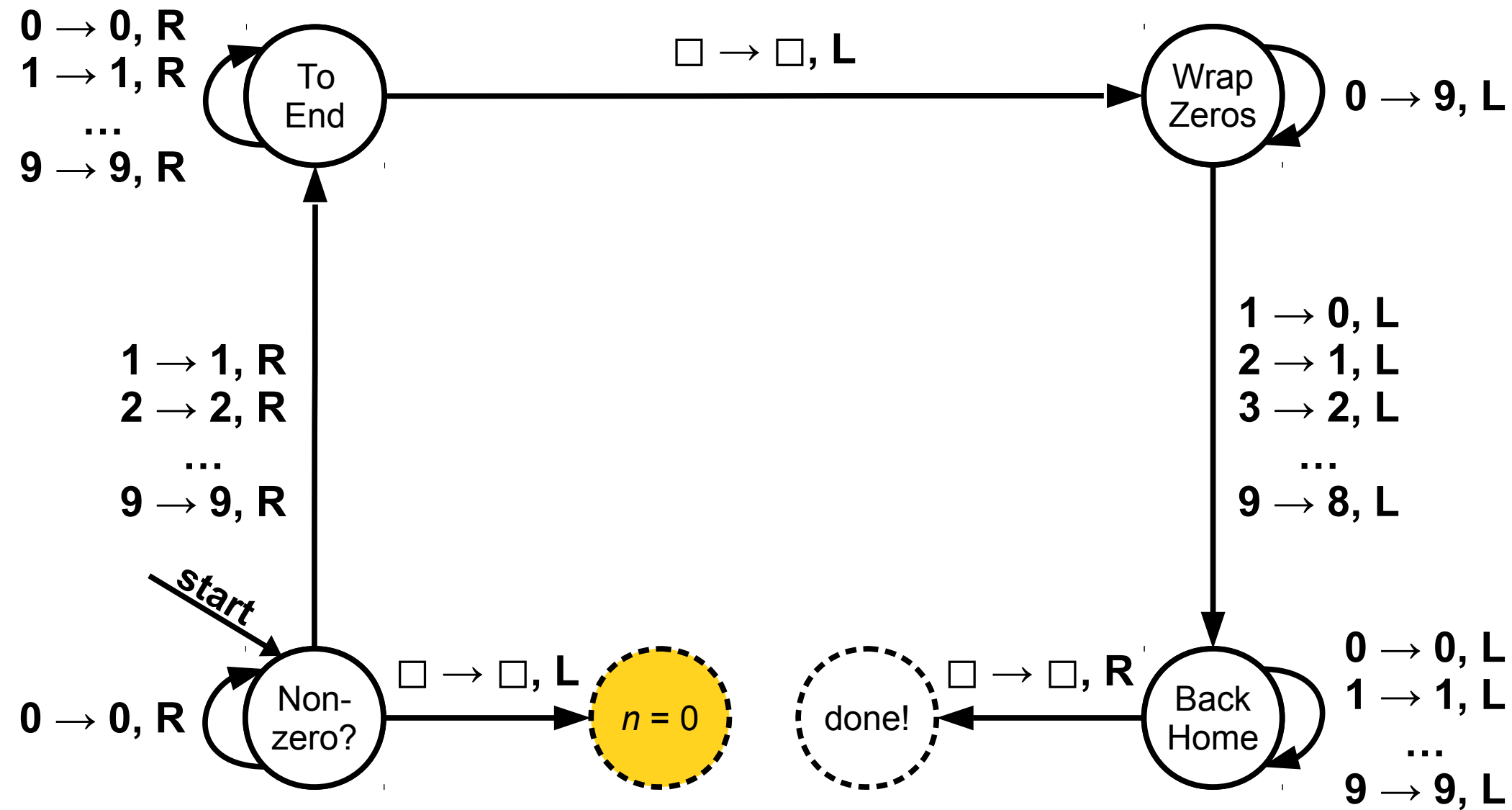


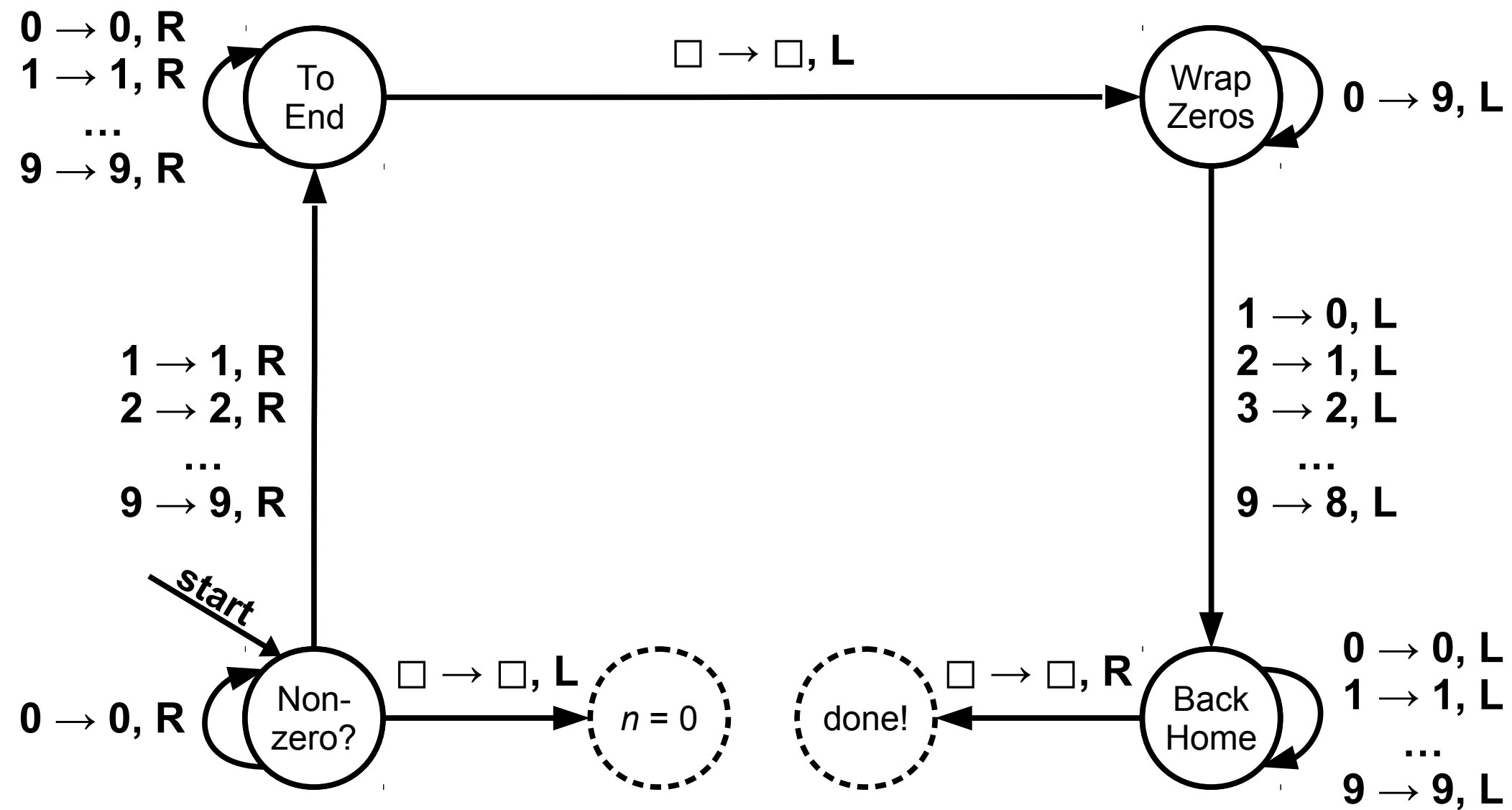










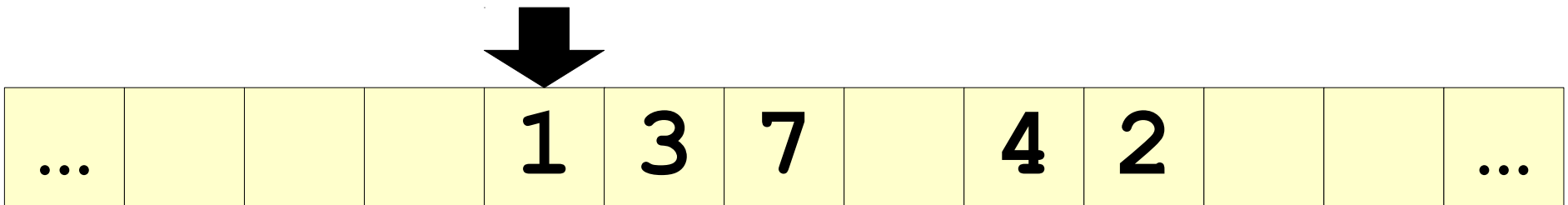


TM Subroutines

- Sometimes, a subroutine needs to report back some information about what happened.
- Just as a function can return multiple different values, we'll allow subroutines to have different “done” states.
- Each state can then be wired to a different state, so a TM using the subroutine can control what happens next.

Putting it All Together

- Our goal is to build a TM that, given two numbers, adds those numbers together.
- Before:

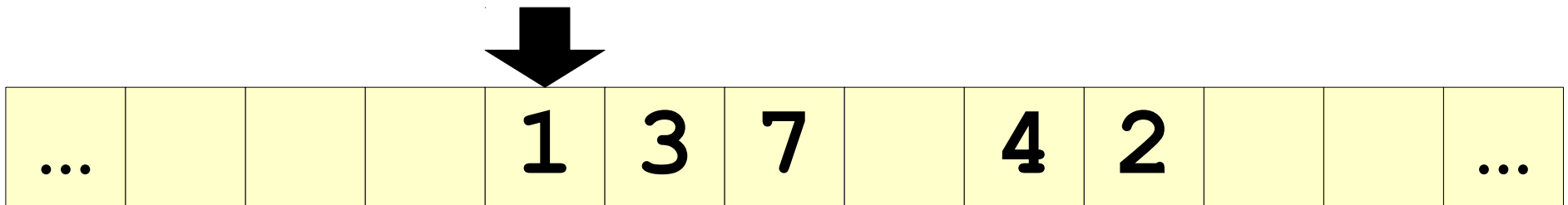


- After:



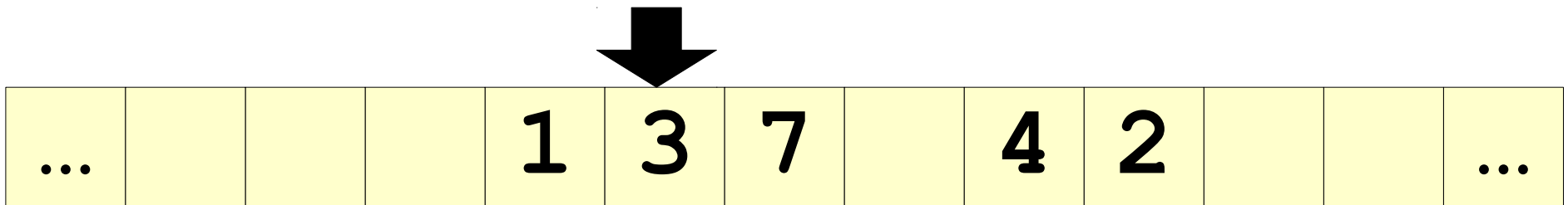
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

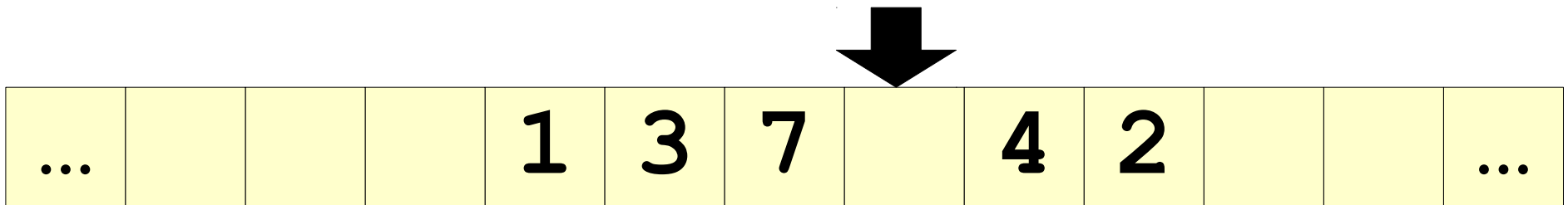
- We'll build our new machine using our existing increment and decrement subroutines:



...				1	3	7		4	2			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

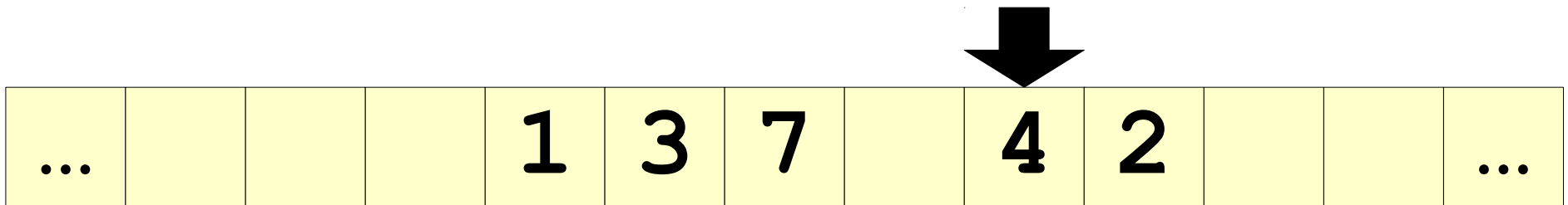
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



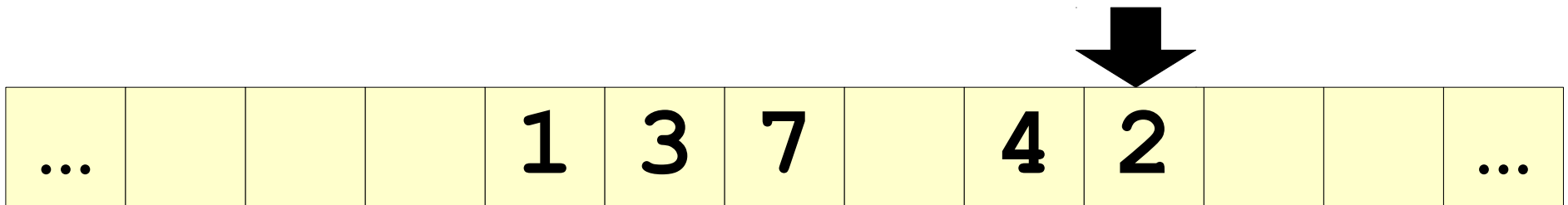
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



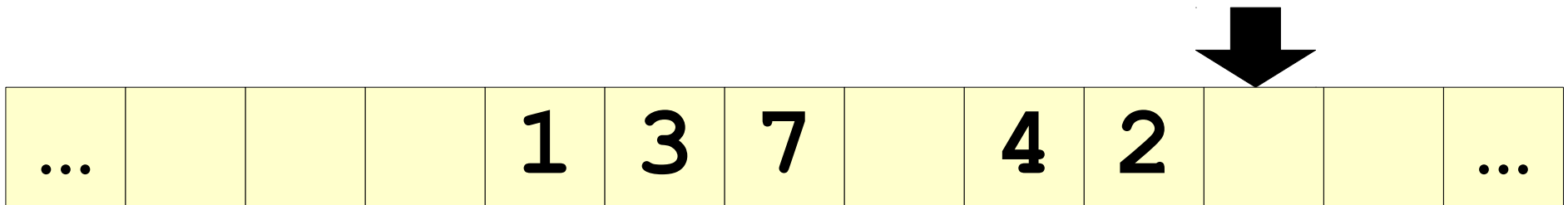
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



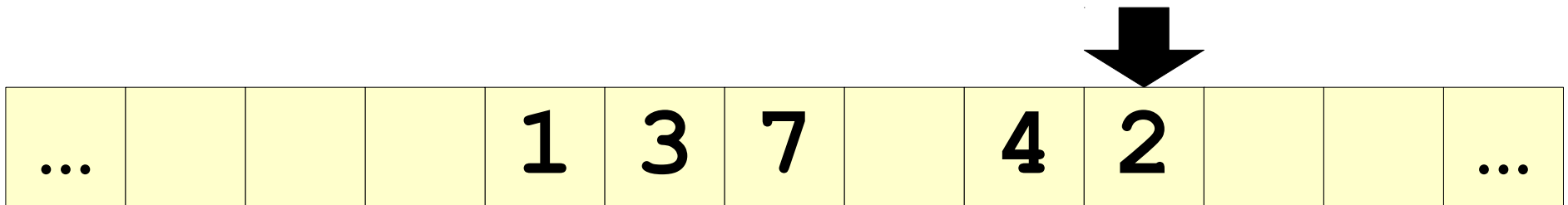
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



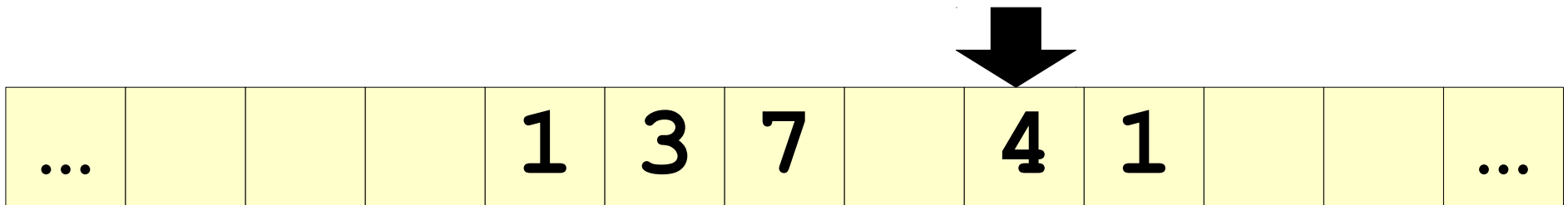
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



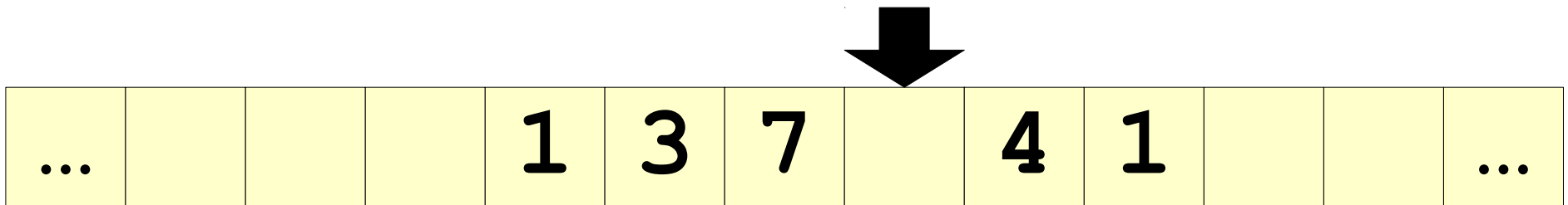
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

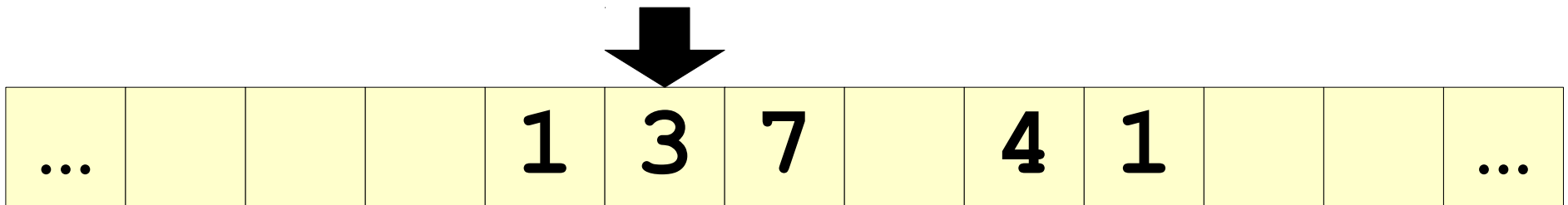
- We'll build our new machine using our existing increment and decrement subroutines:



...				1	3	7		4	1			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

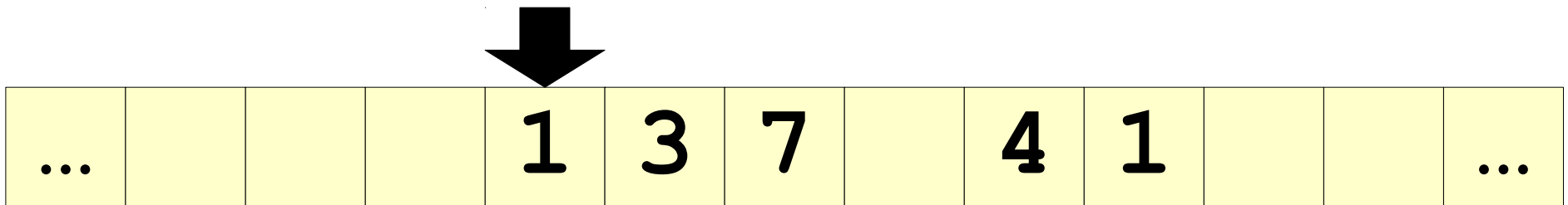
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

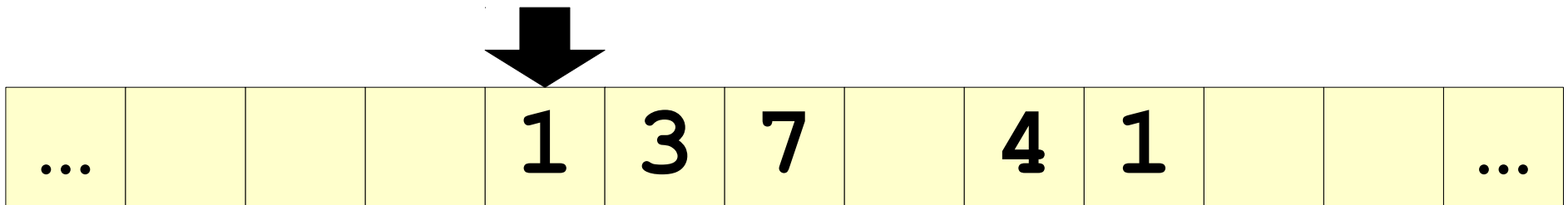
- We'll build our new machine using our existing increment and decrement subroutines:



...				1	3	7		4	1			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

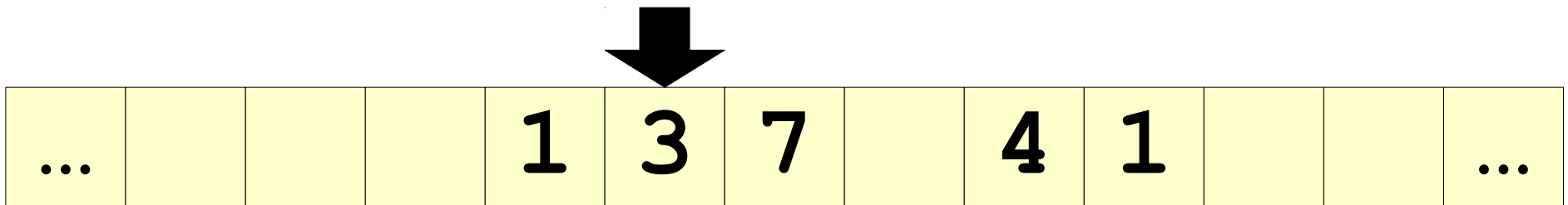
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

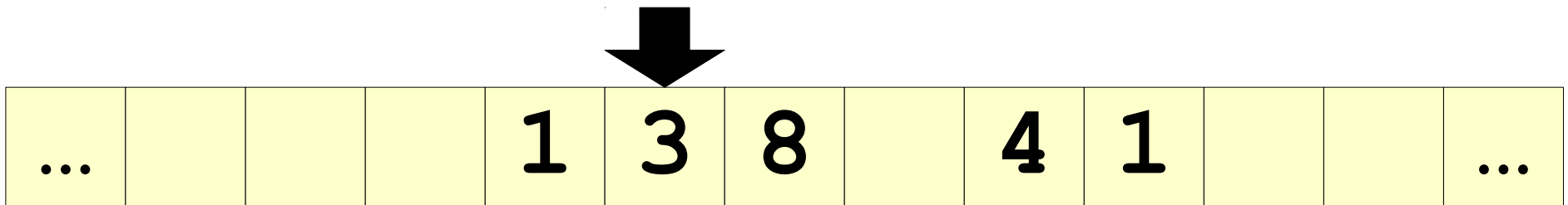
- We'll build our new machine using our existing increment and decrement subroutines:



...				1	3	7		4	1			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

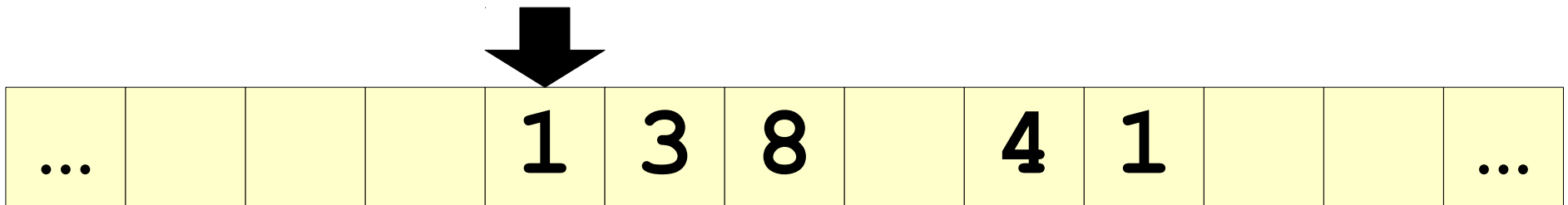
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:



Using Our Subroutines

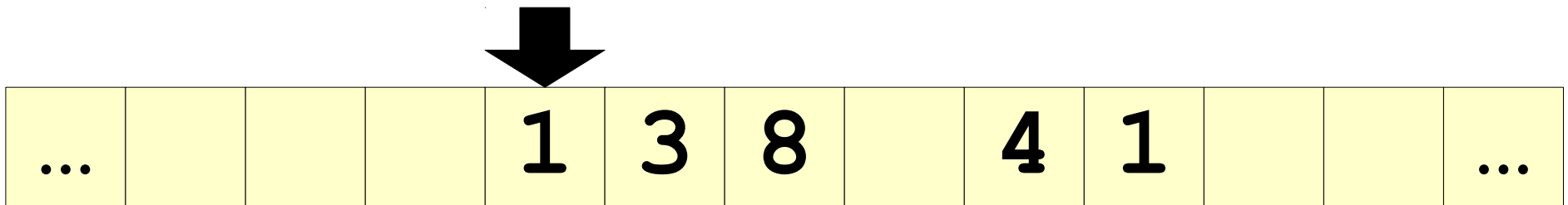
- We'll build our new machine using our existing increment and decrement subroutines:



...				1	3	8		4	1			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

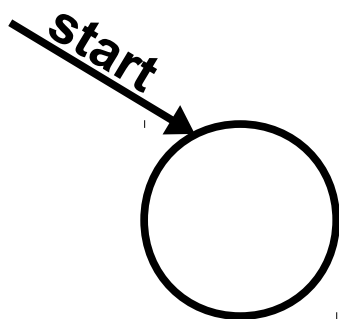
Using Our Subroutines

- We'll build our new machine using our existing increment and decrement subroutines:

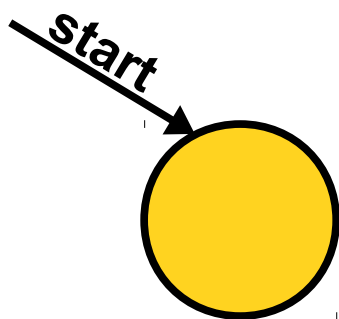


...		1	3	7		4	2							...
-----	--	---	---	---	--	---	---	--	--	--	--	--	--	-----

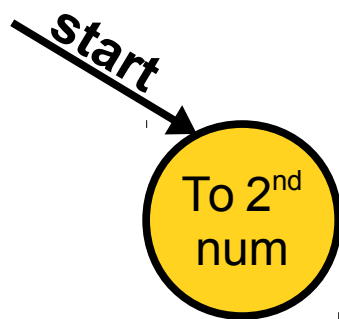




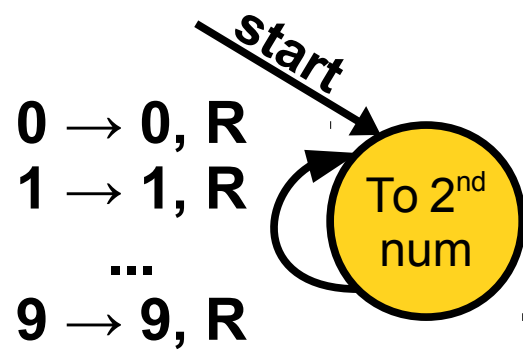
...		1	3	7		4	2						...
-----	--	---	---	---	--	---	---	--	--	--	--	--	-----



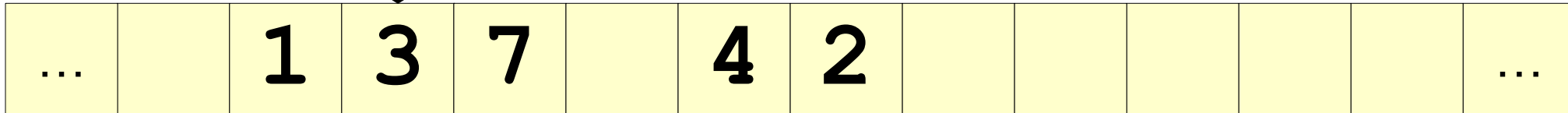
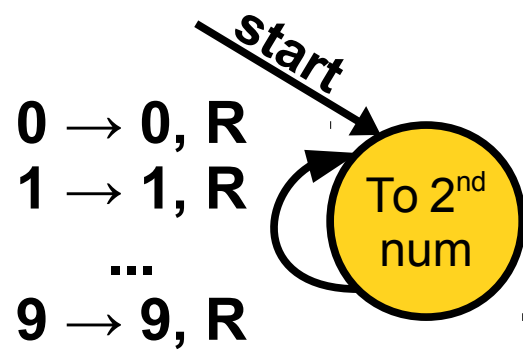
...		1	3	7		4	2						...
-----	--	---	---	---	--	---	---	--	--	--	--	--	-----

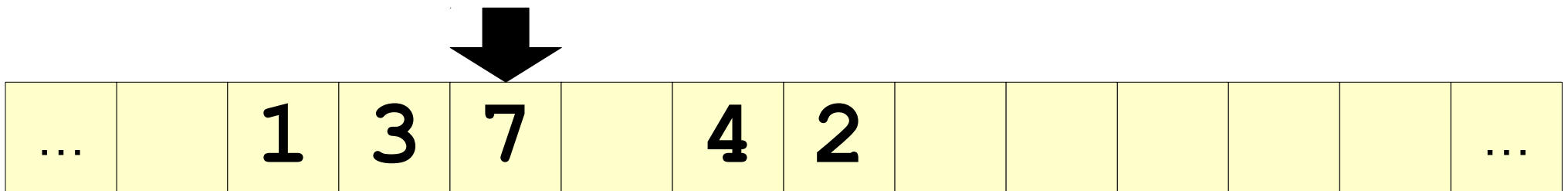
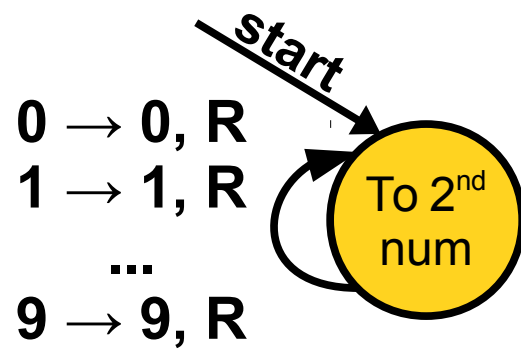


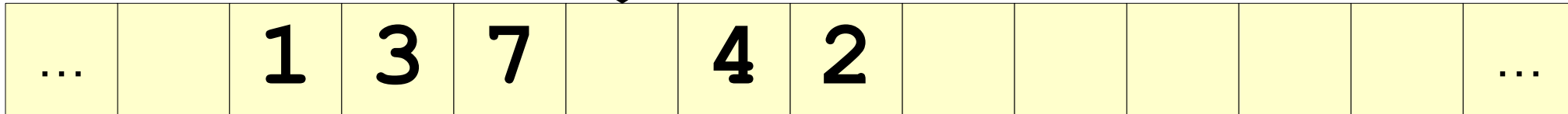
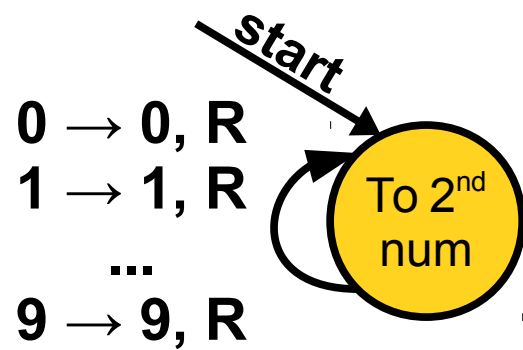
...		1	3	7		4	2							...
-----	--	---	---	---	--	---	---	--	--	--	--	--	--	-----

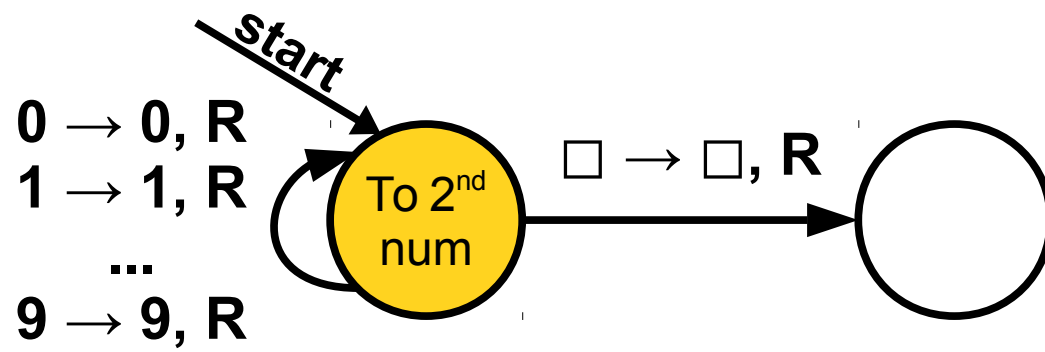


...		1	3	7		4	2						...
-----	--	---	---	---	--	---	---	--	--	--	--	--	-----

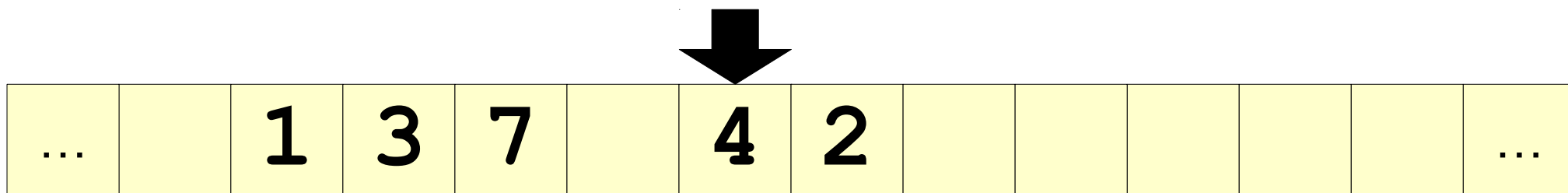
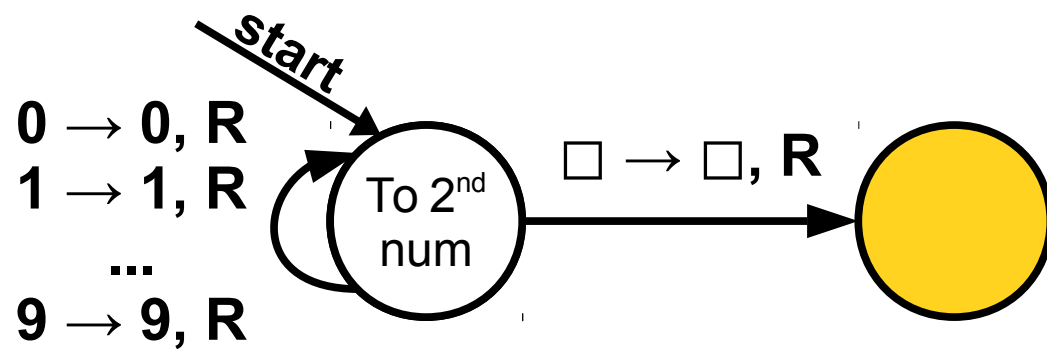


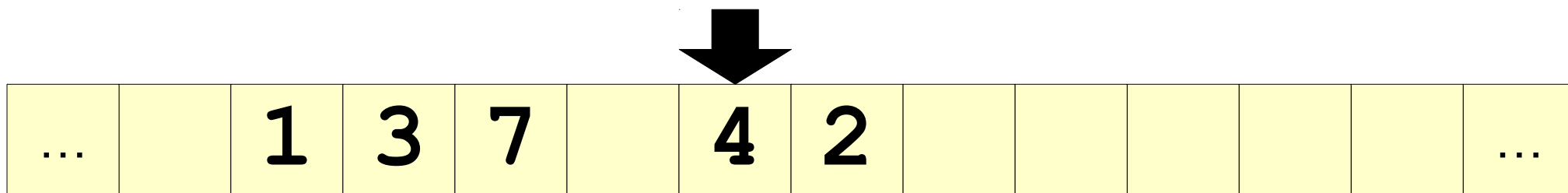
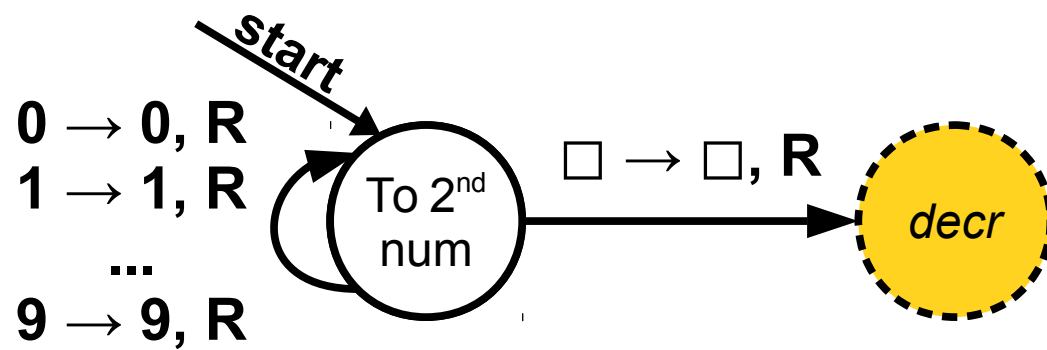


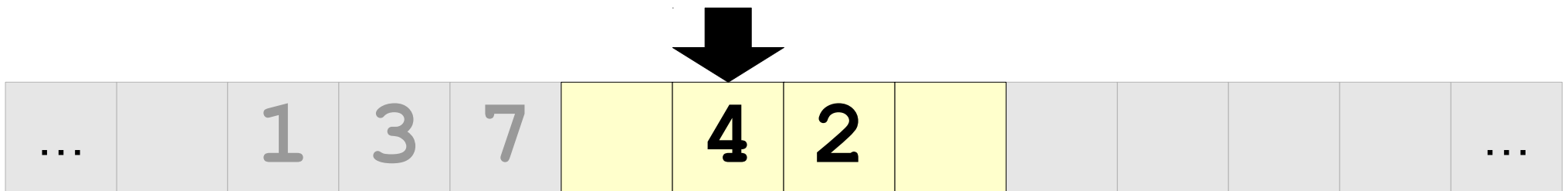
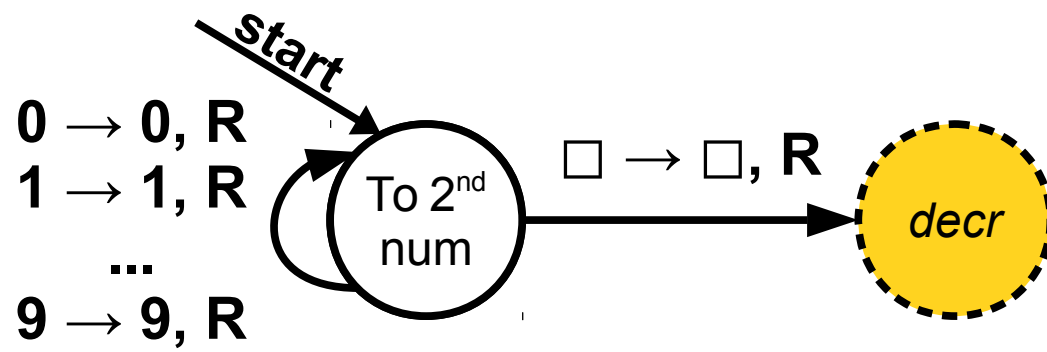


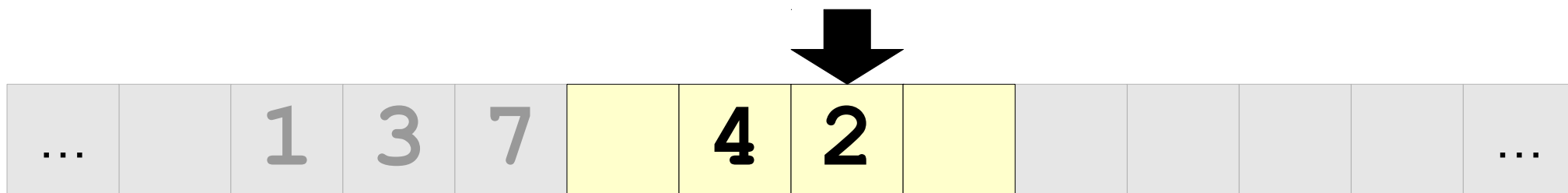
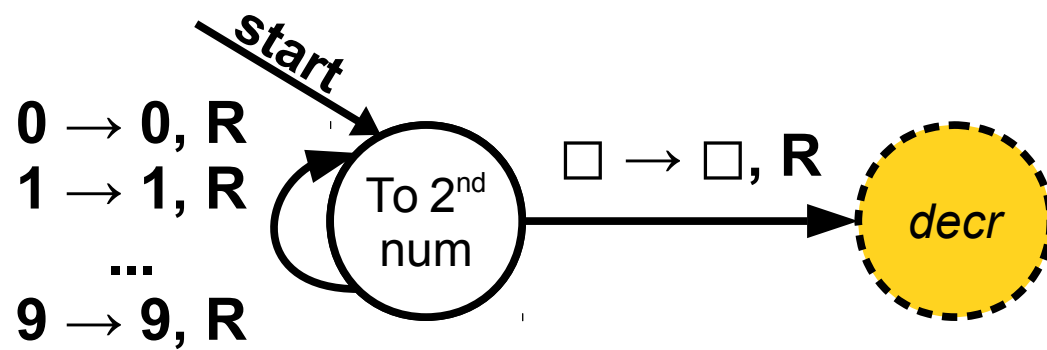


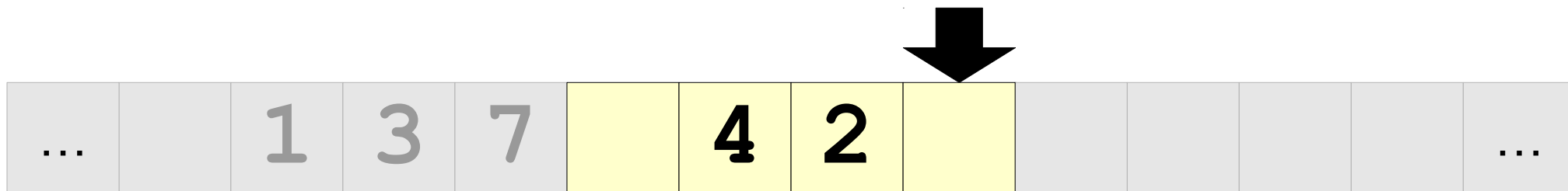
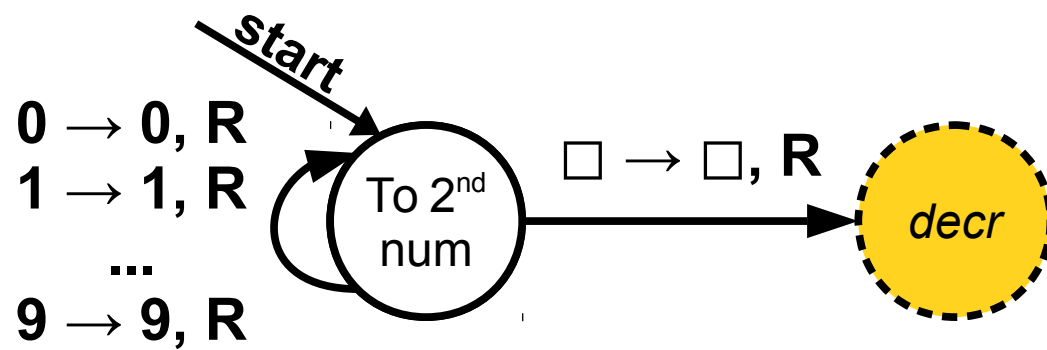
...		1	3	7		4	2						...
-----	--	---	---	---	--	---	---	--	--	--	--	--	-----

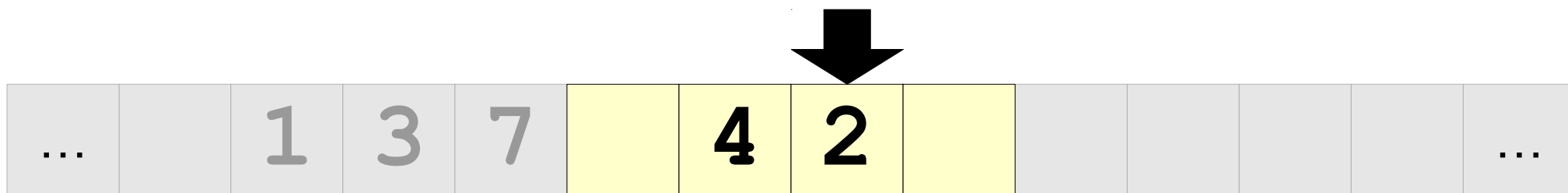
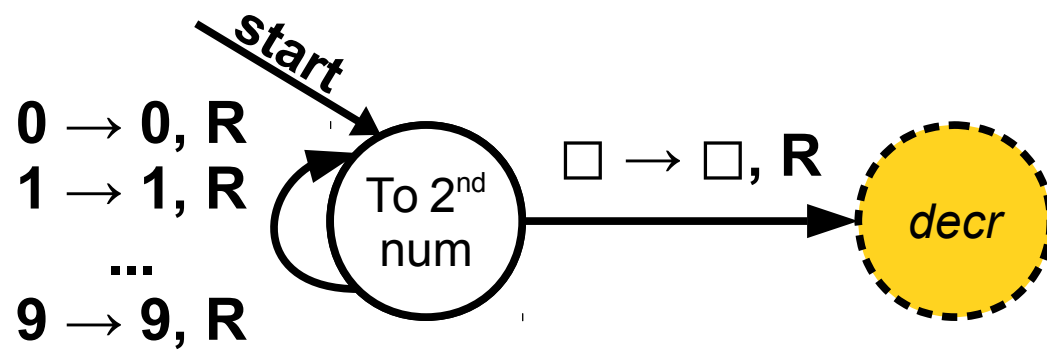


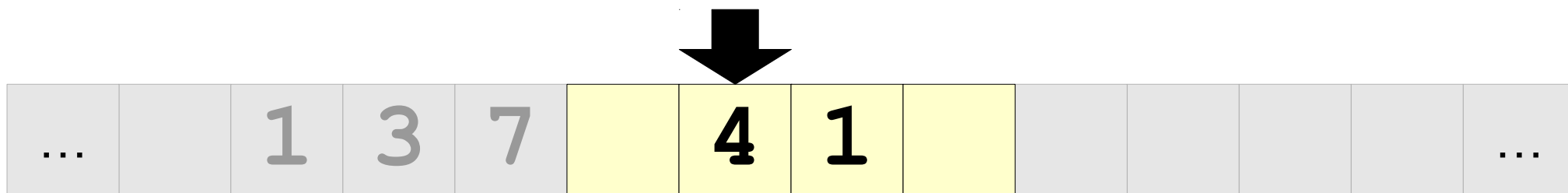
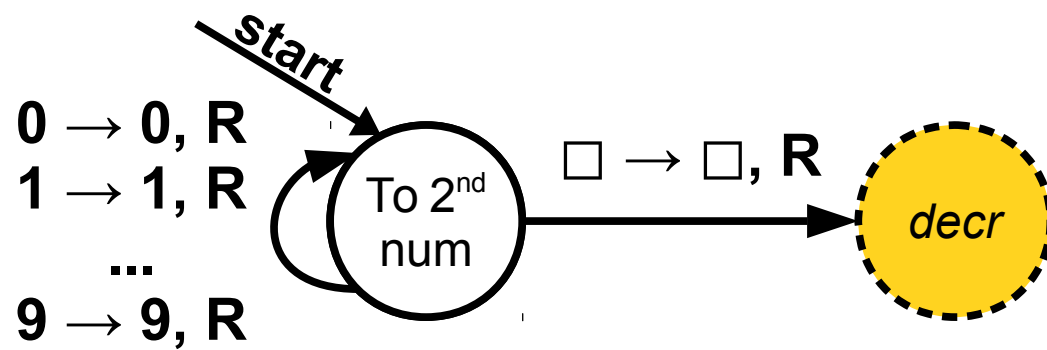


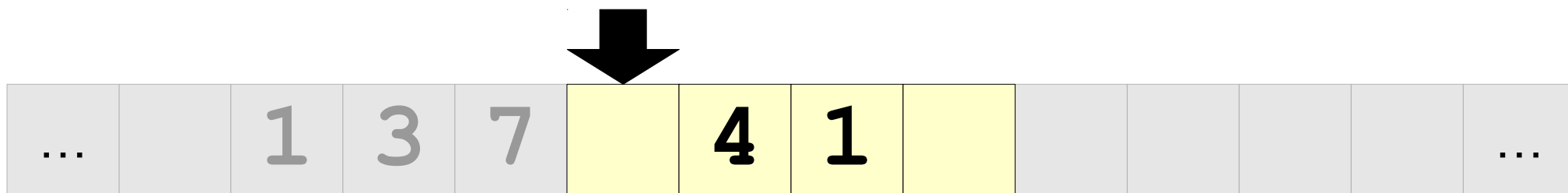
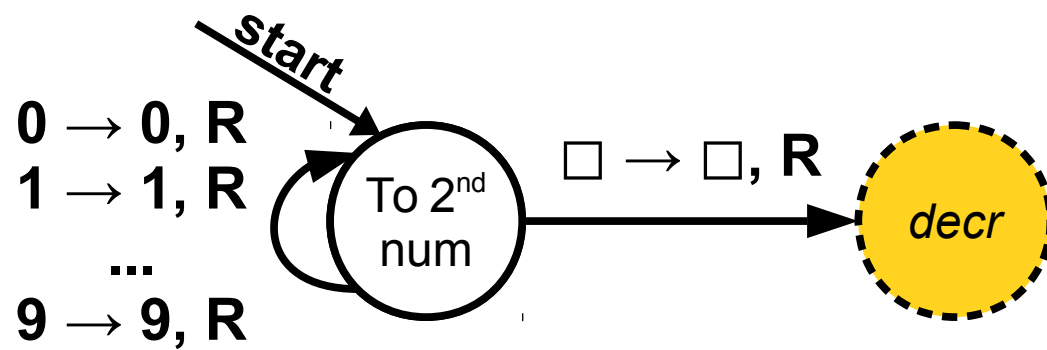


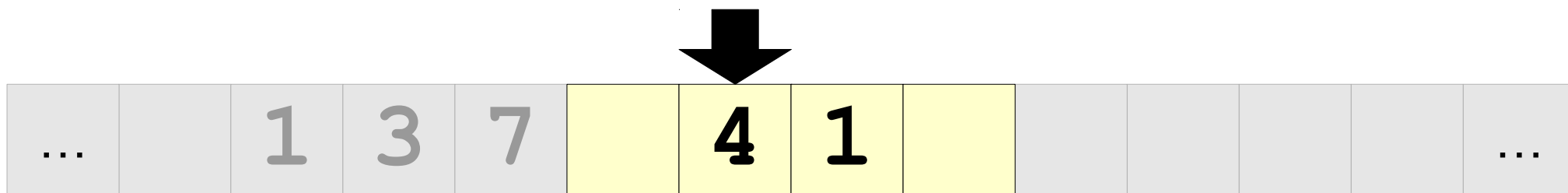
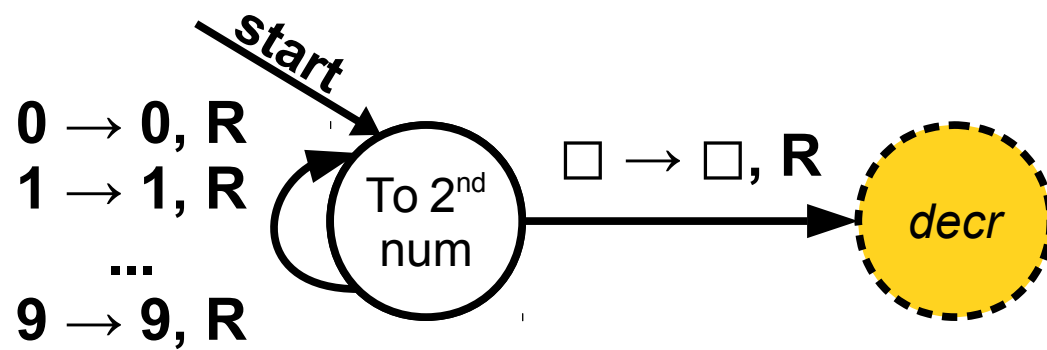


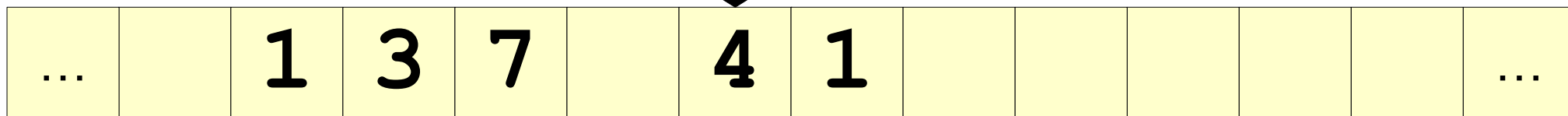
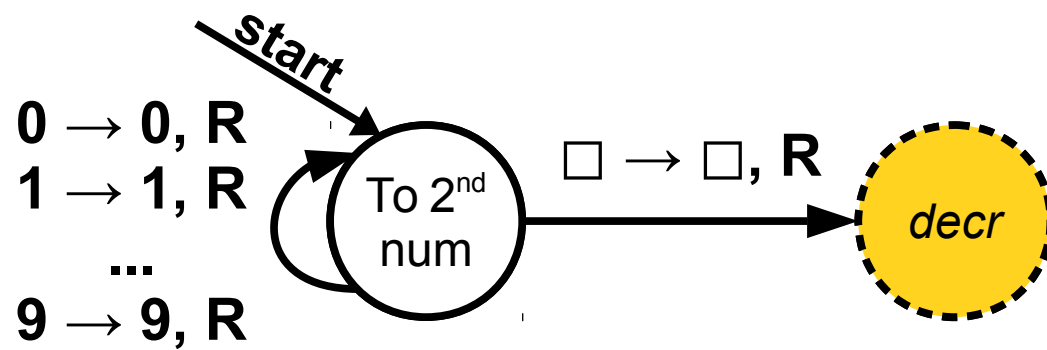


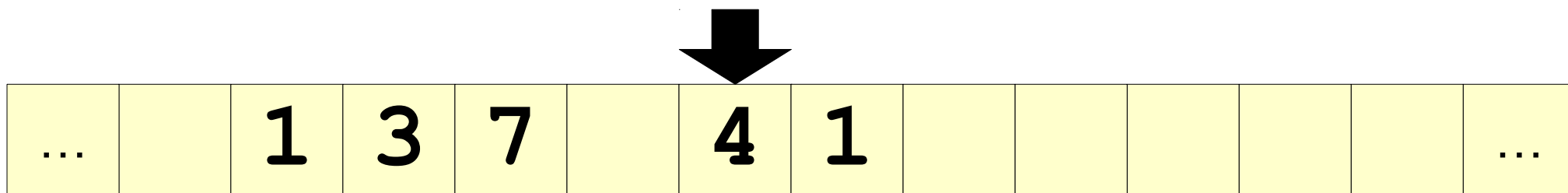
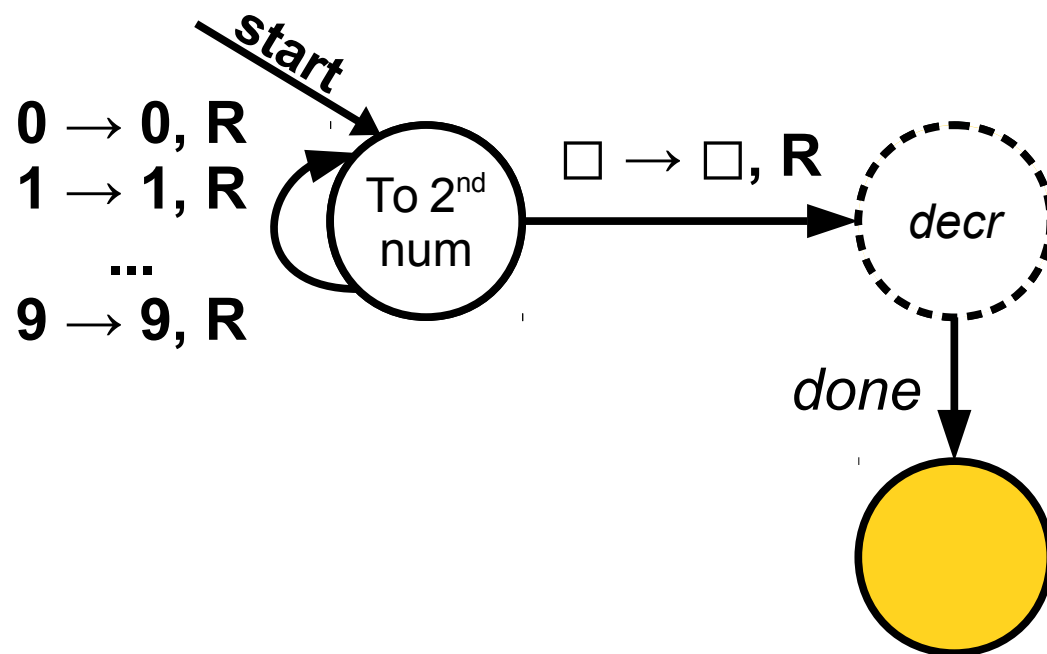


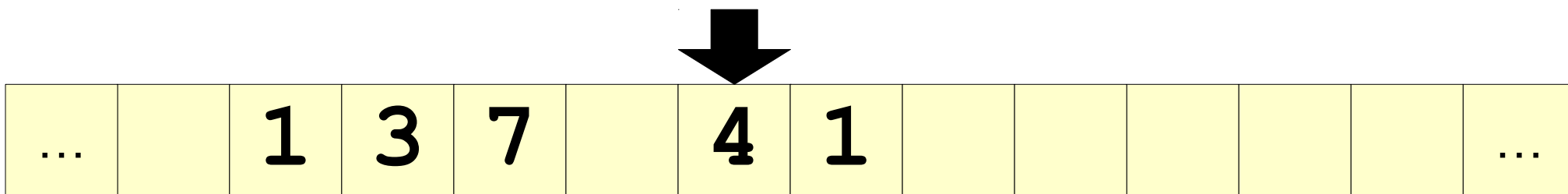
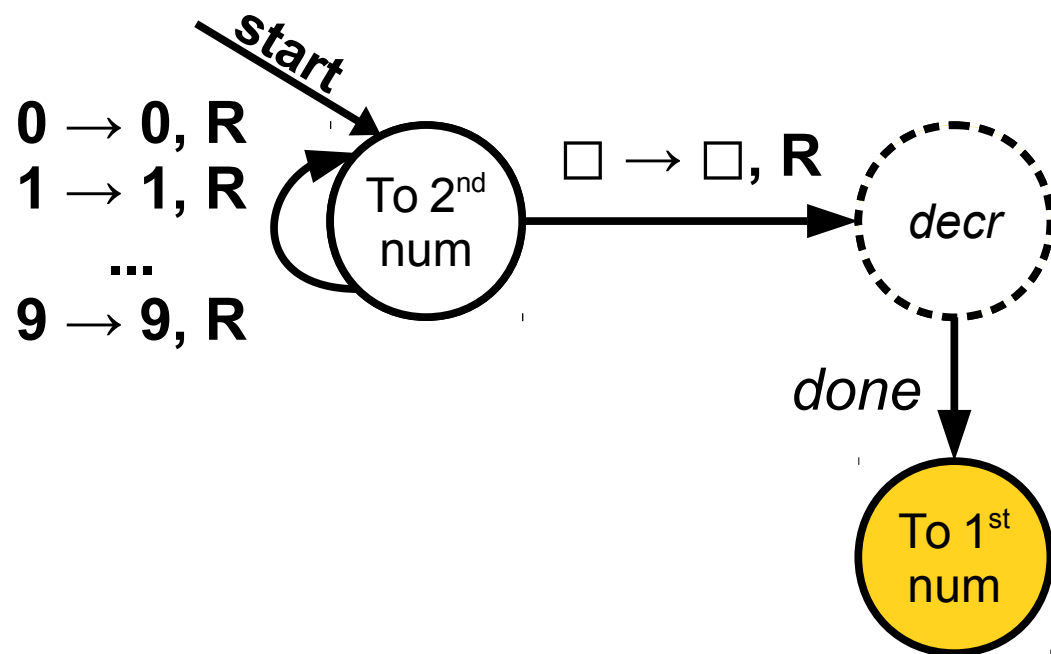


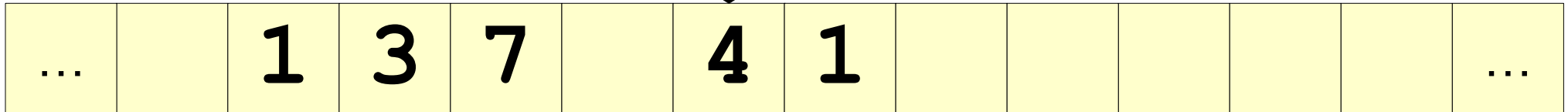
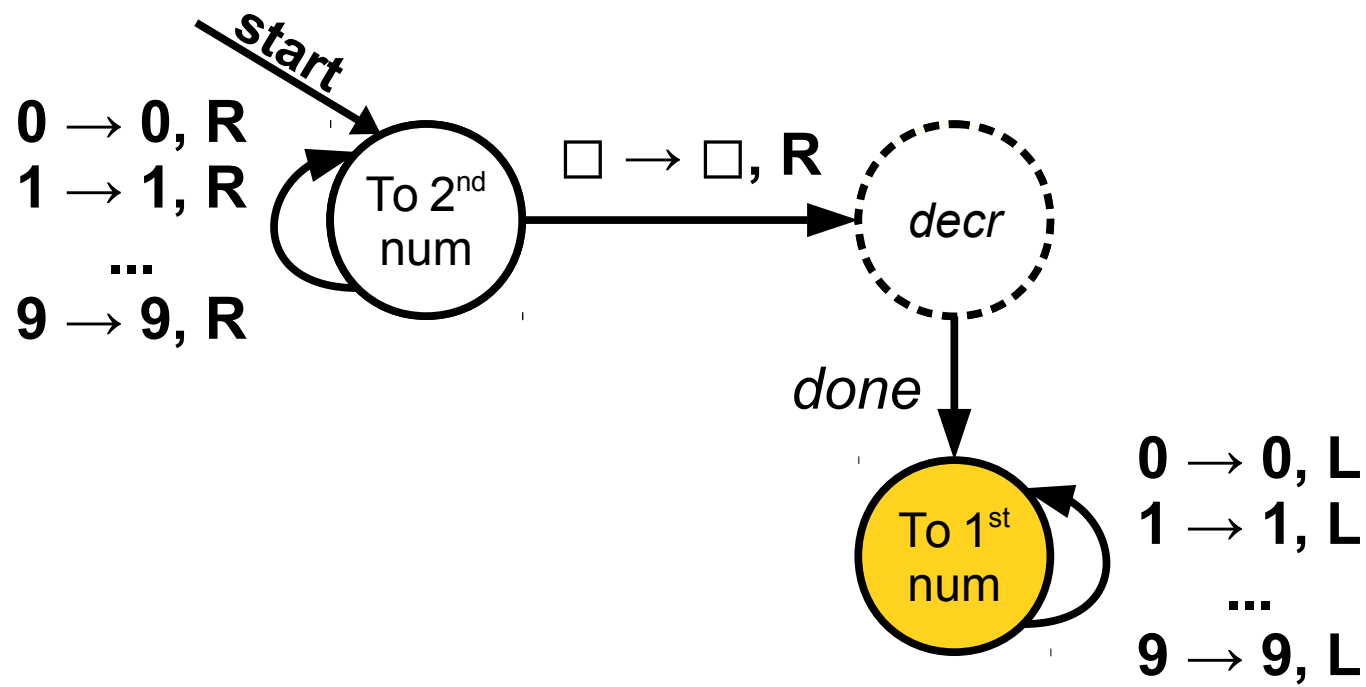


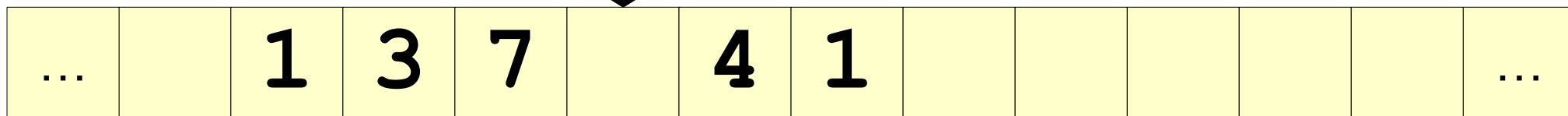
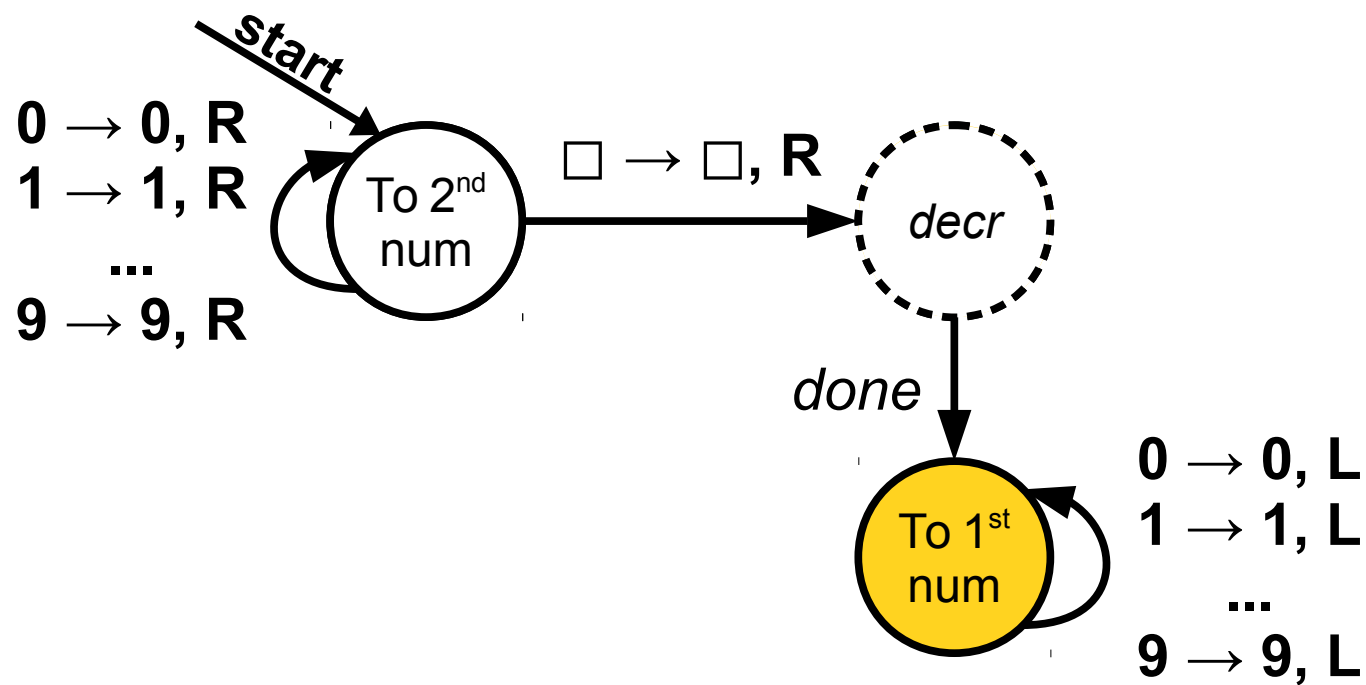


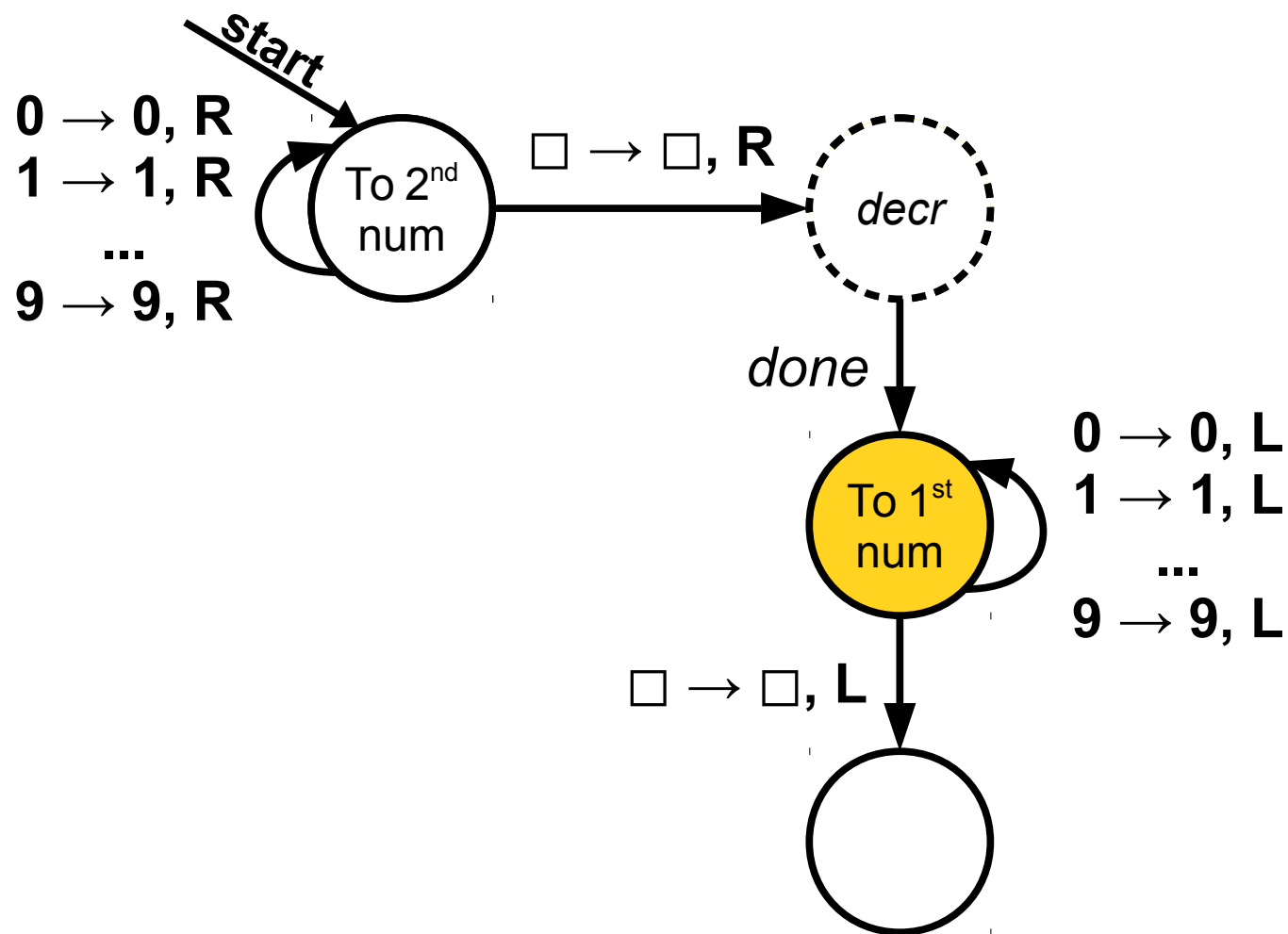




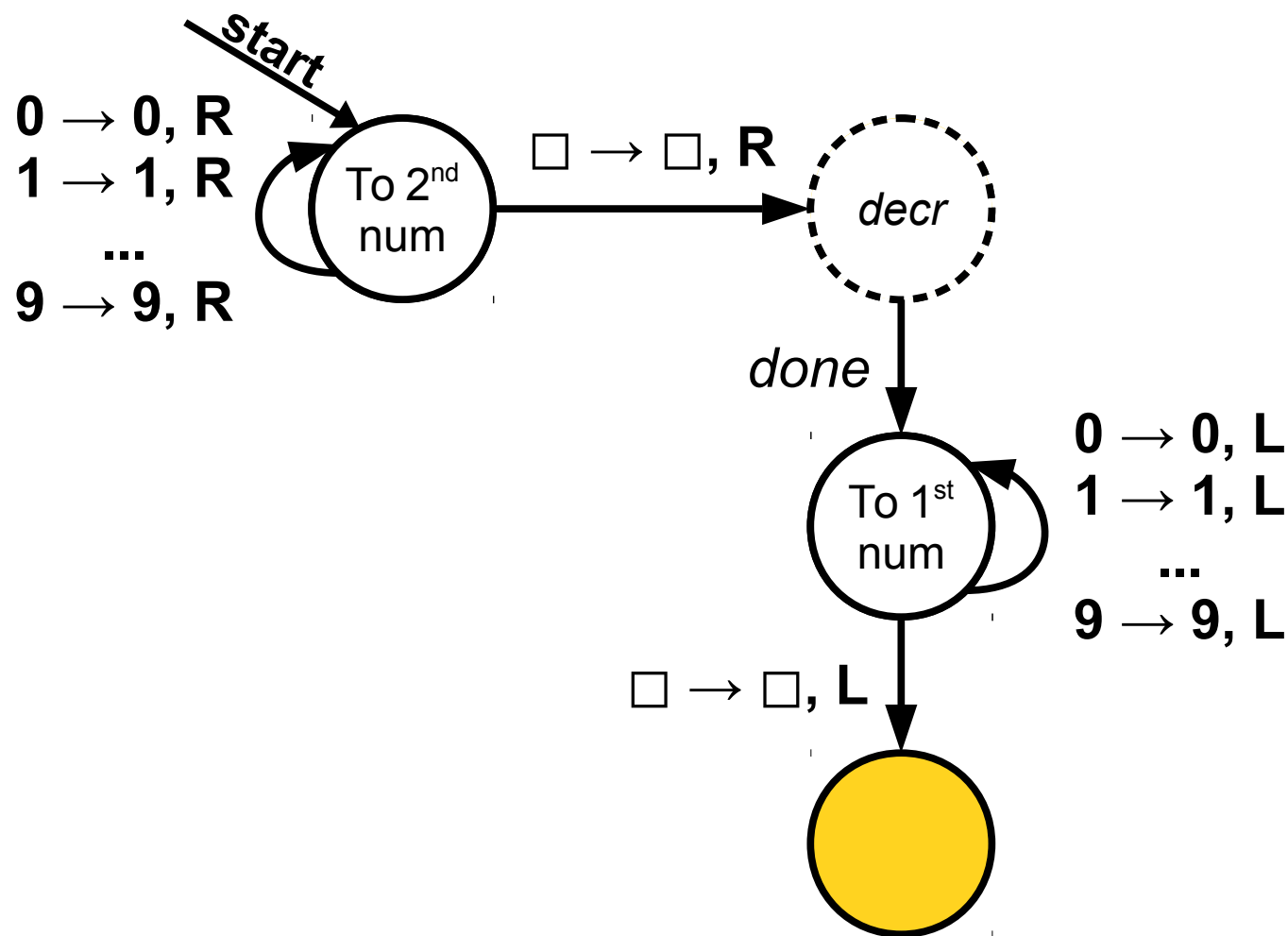




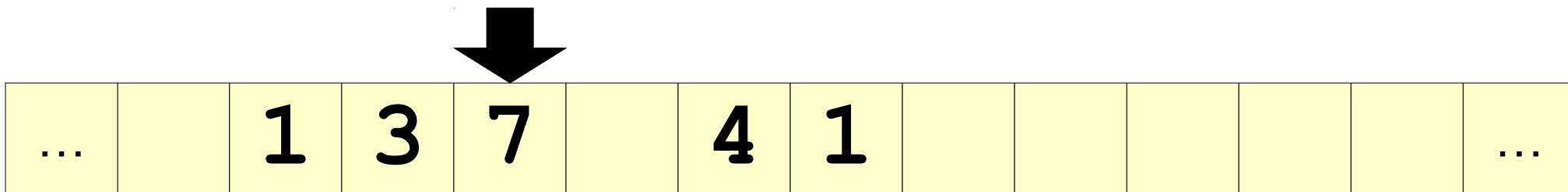
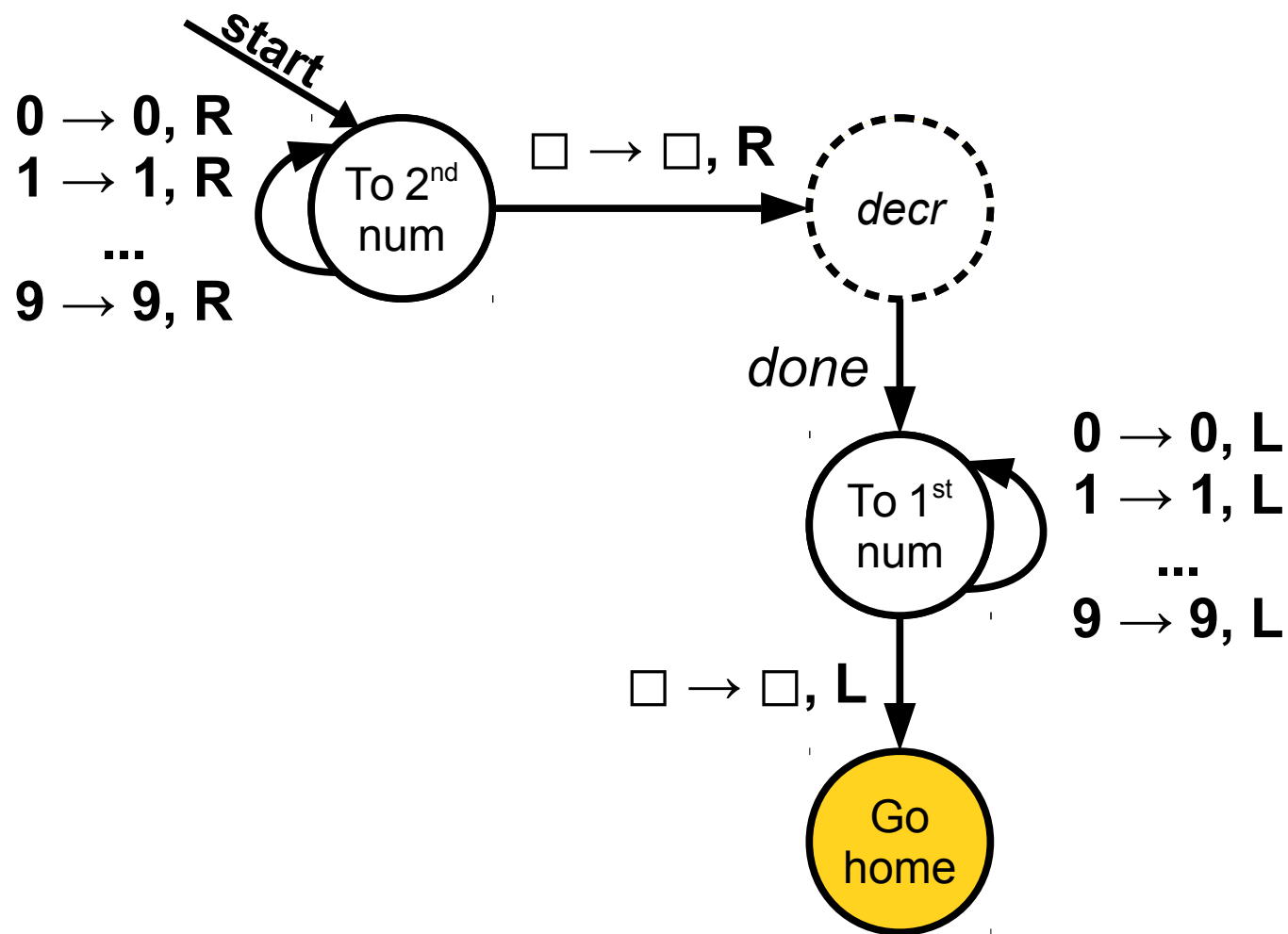


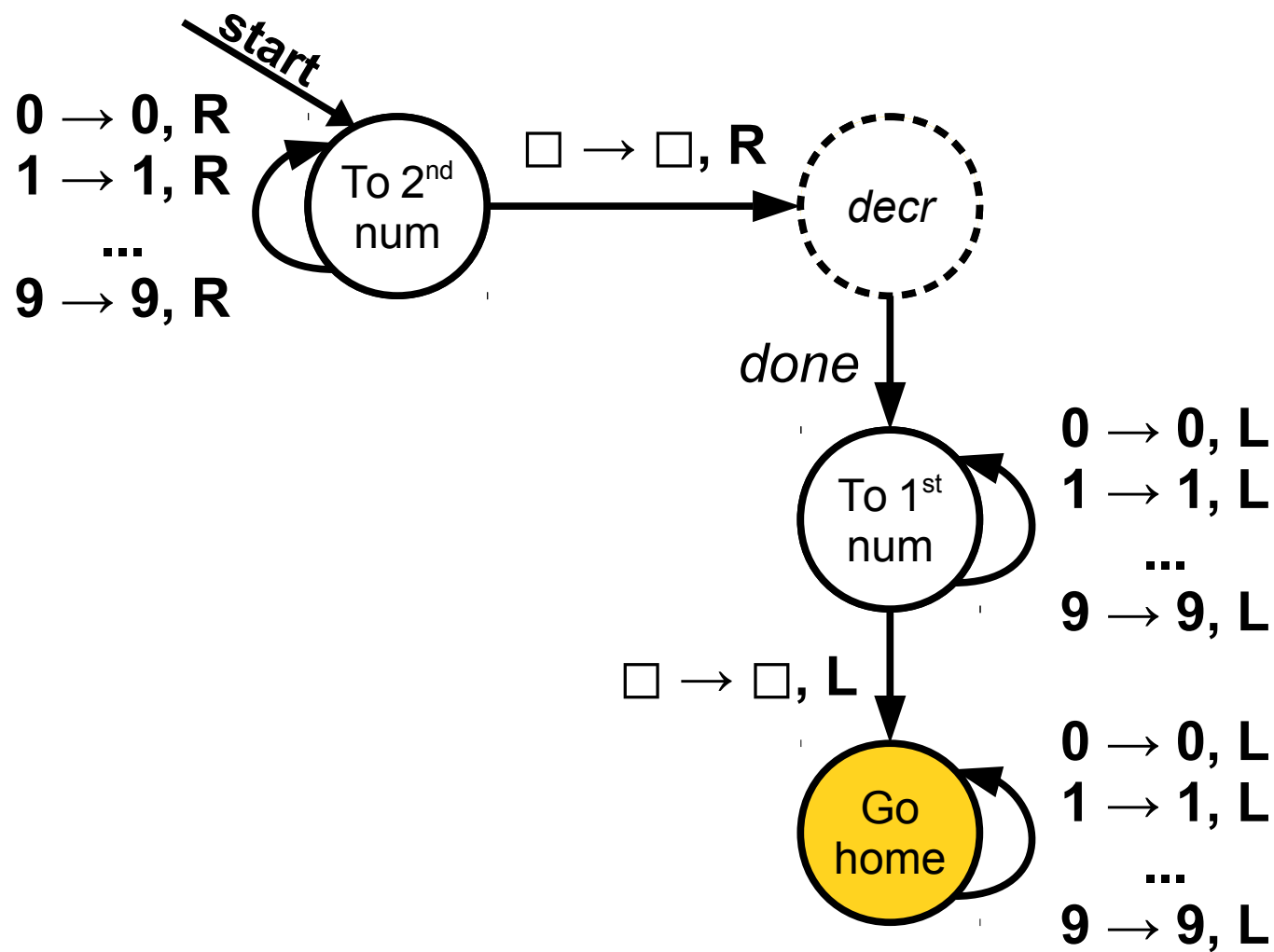


...		1	3	7		4	1						...
-----	--	---	---	---	--	---	---	--	--	--	--	--	-----

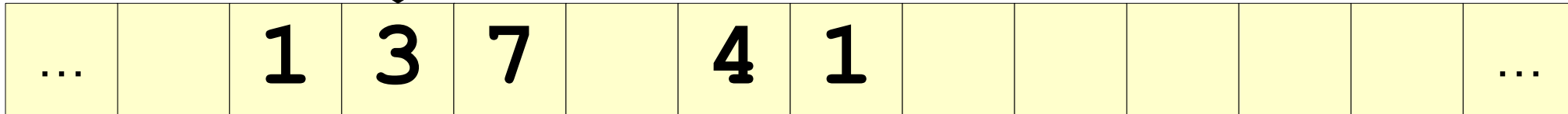
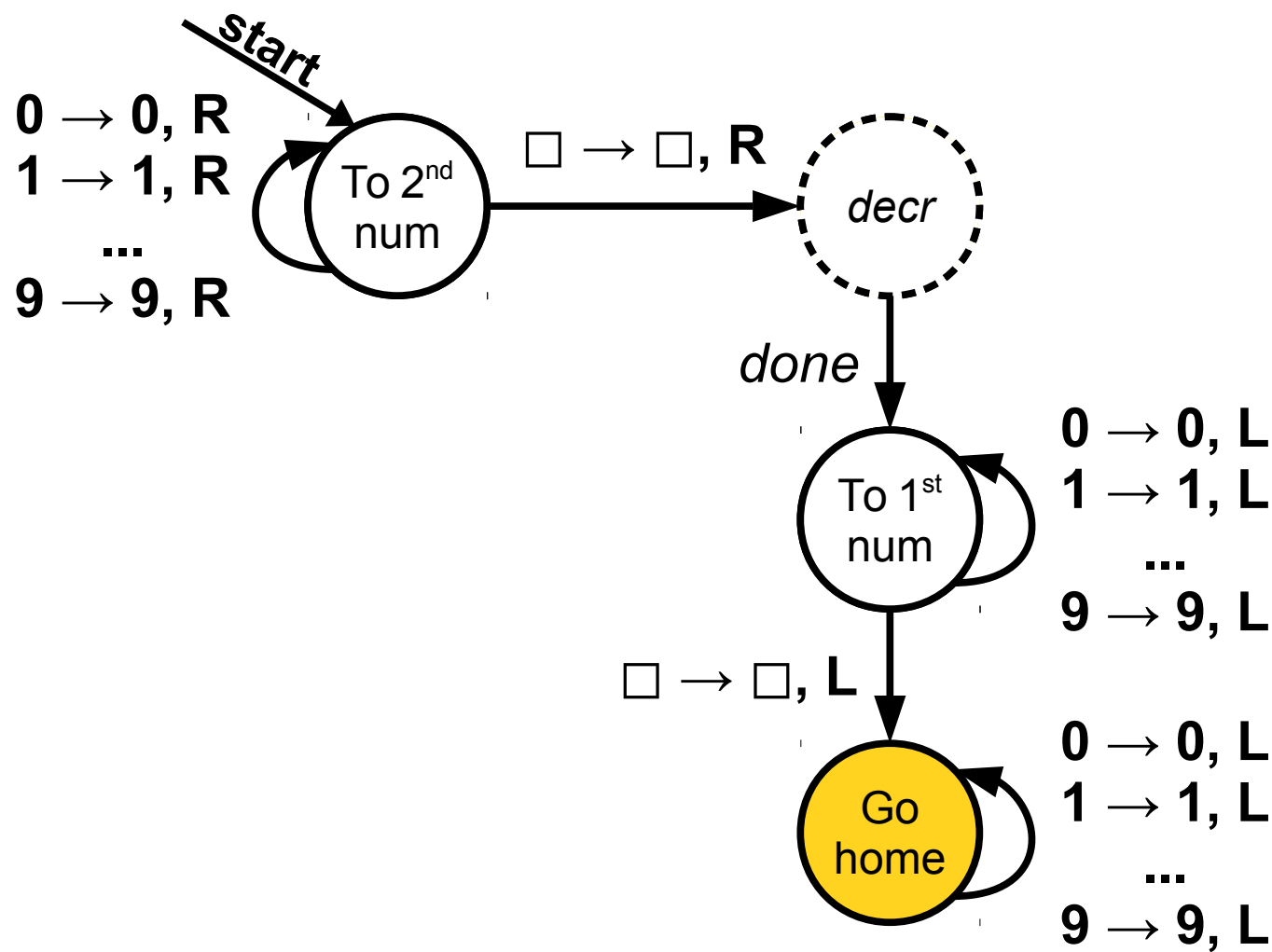


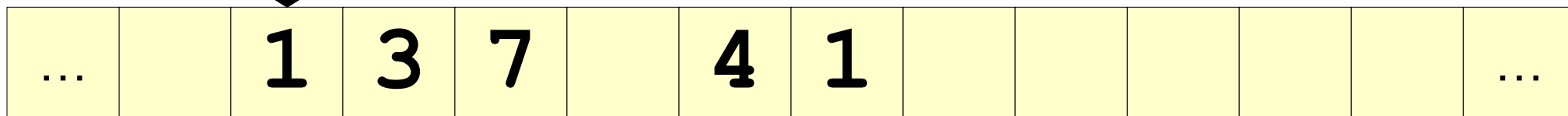
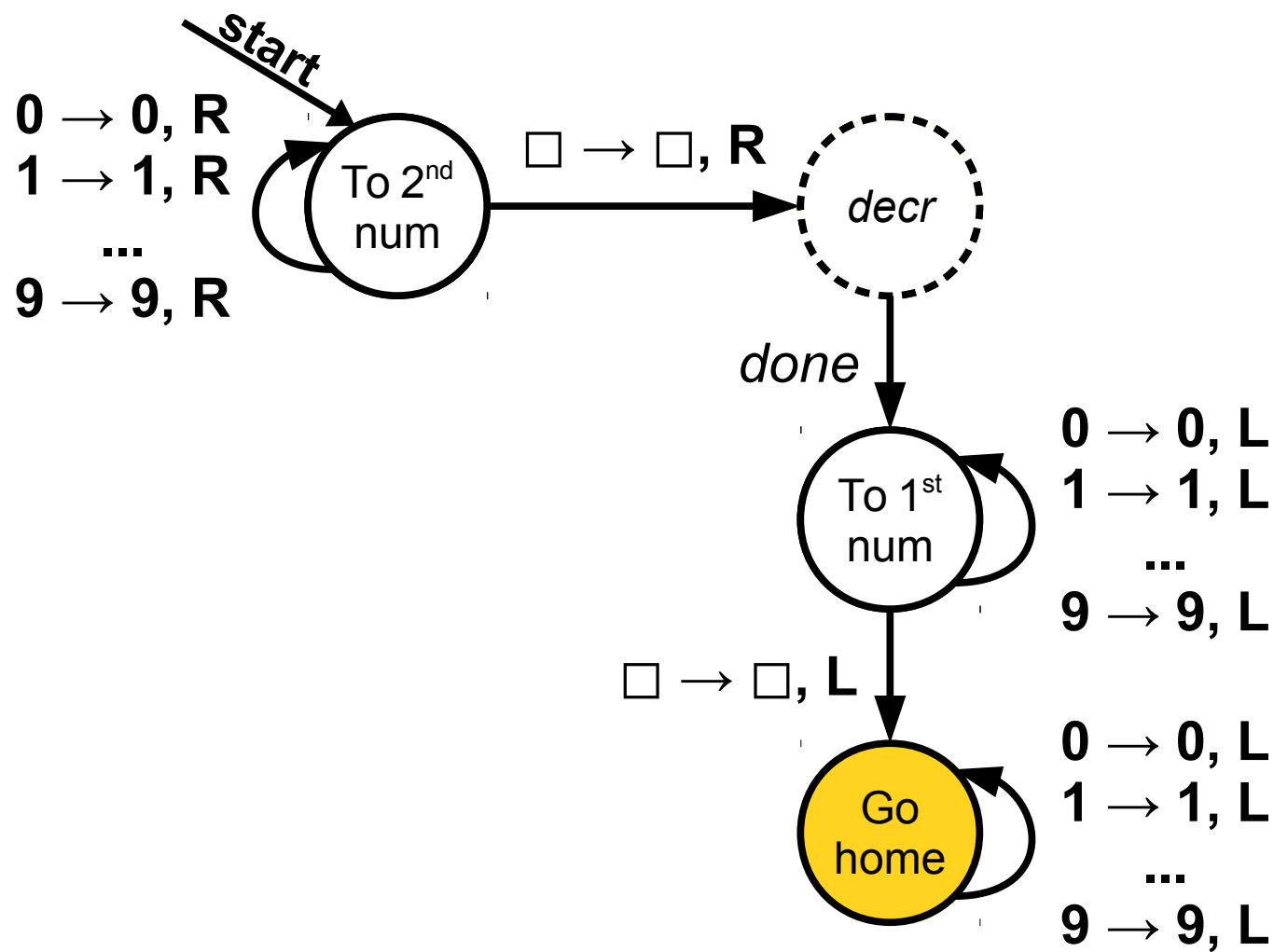
...	1	3	7	4	1							...
-----	---	---	---	---	---	--	--	--	--	--	--	-----

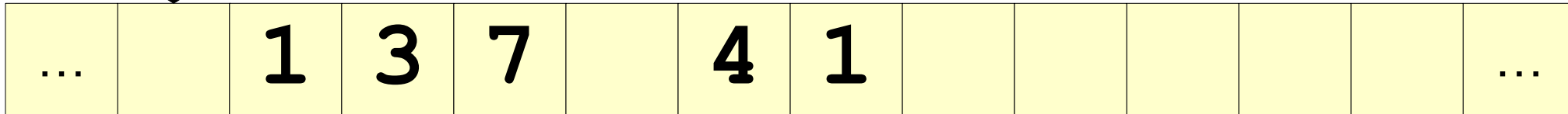
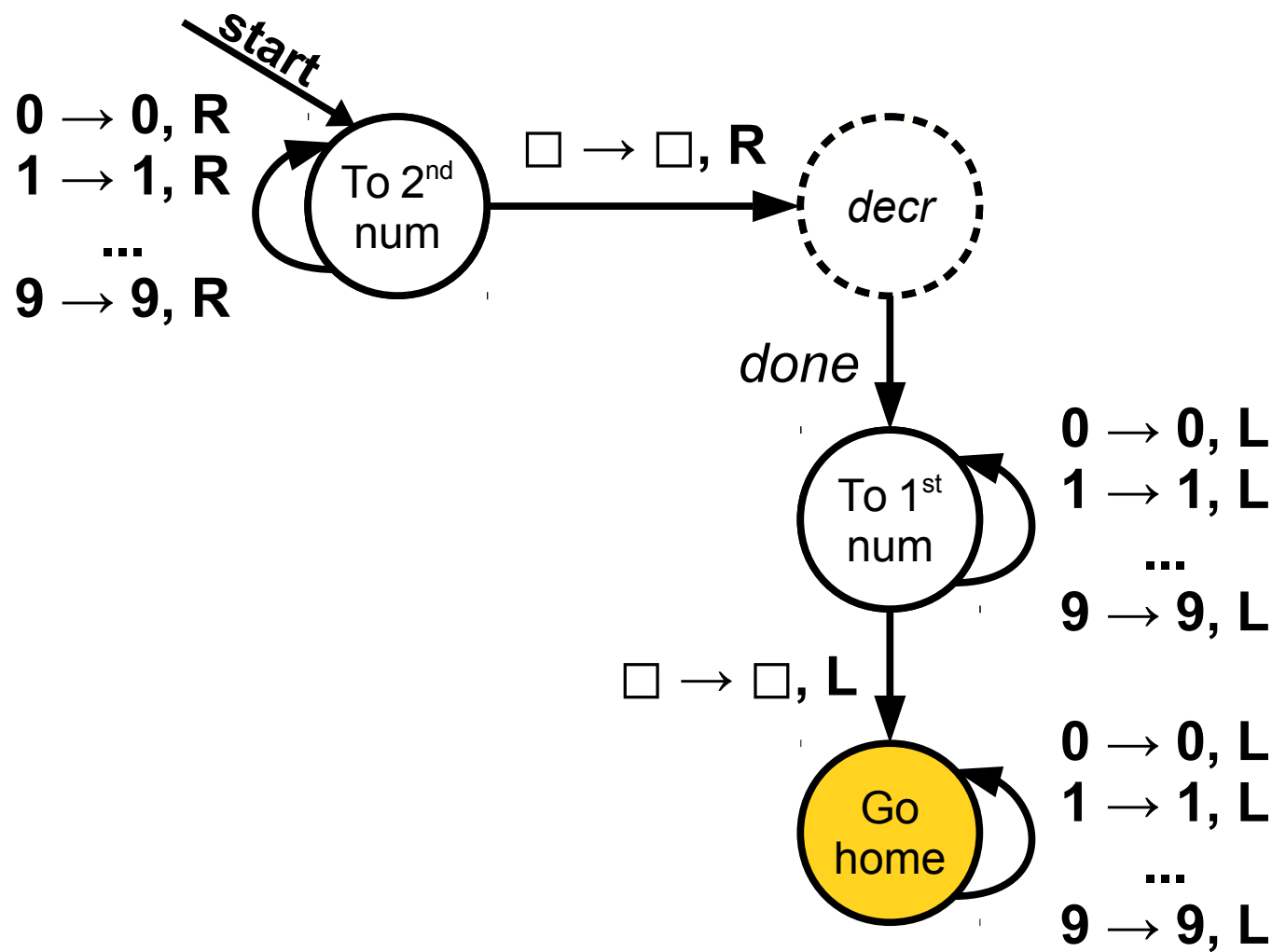


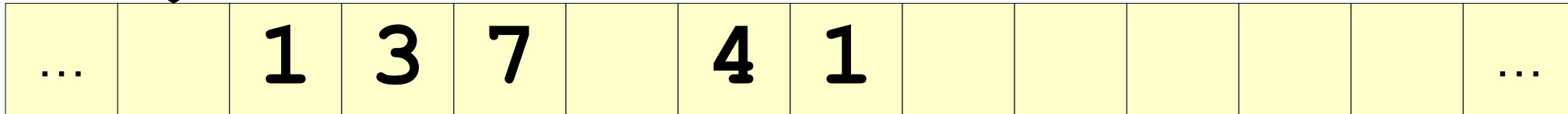
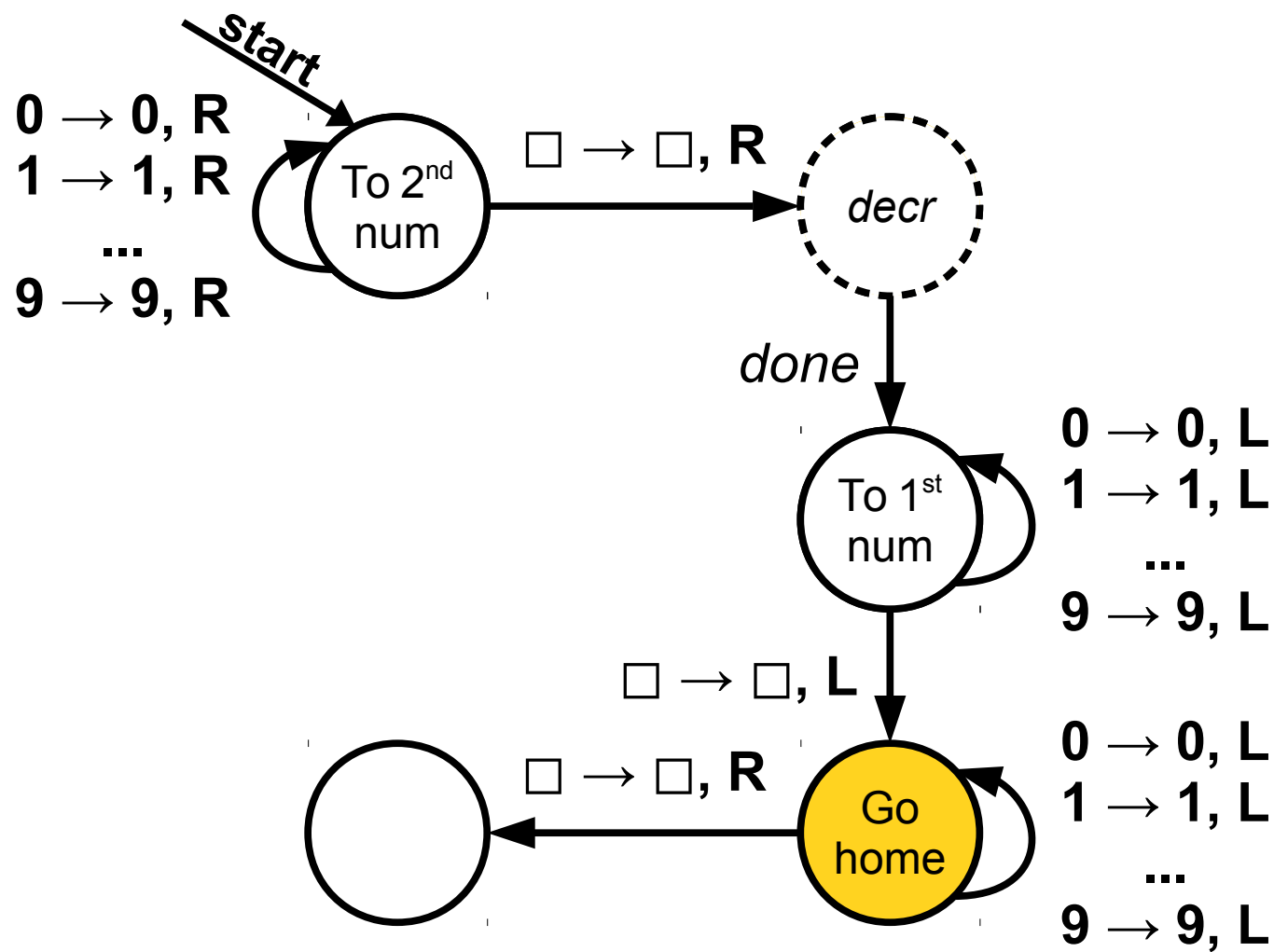


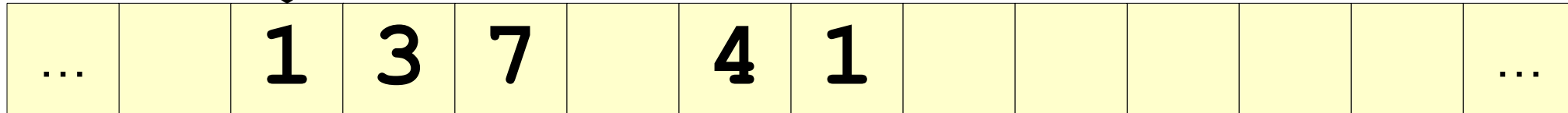
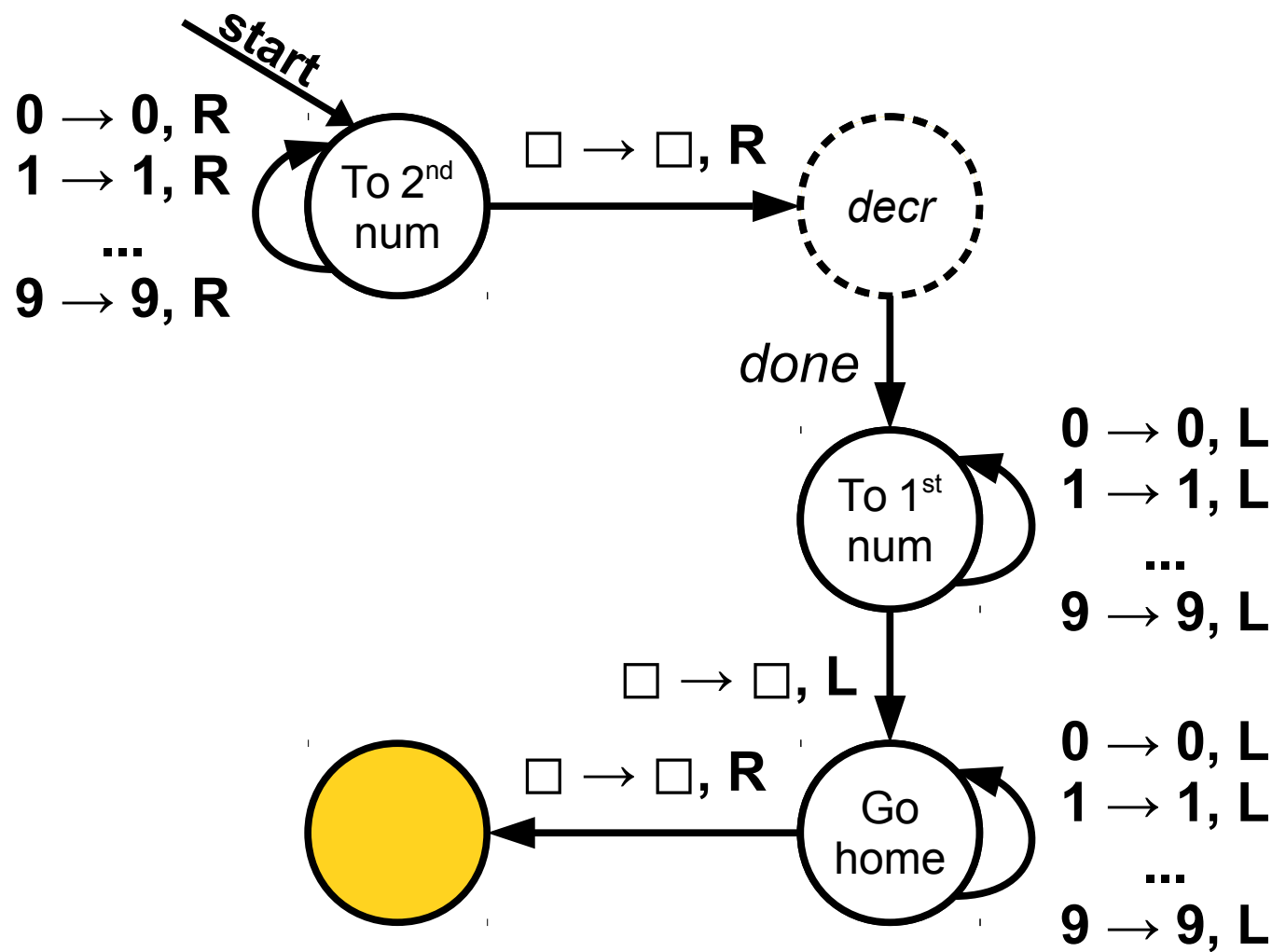
...	1	3	7	4	1							...
-----	---	---	---	---	---	--	--	--	--	--	--	-----

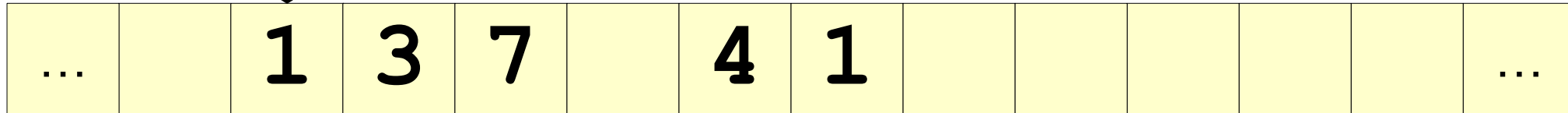
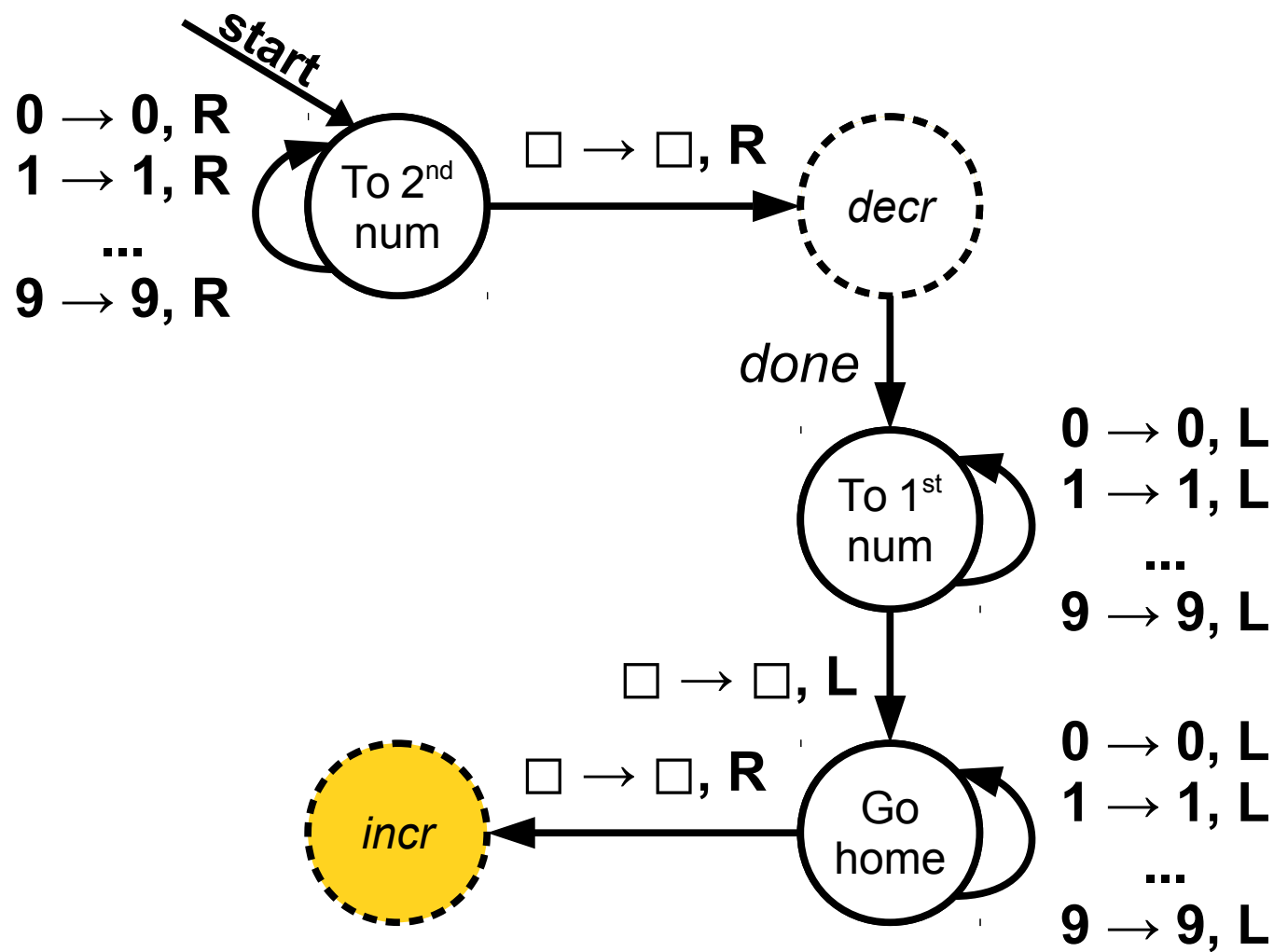


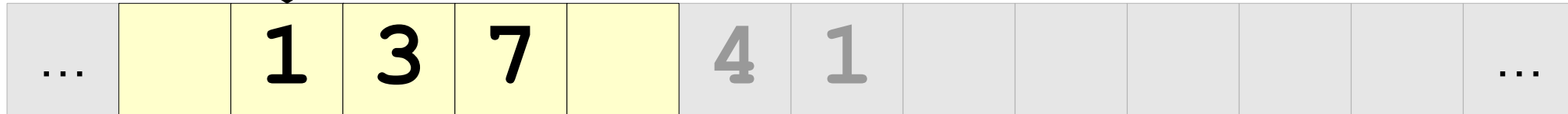
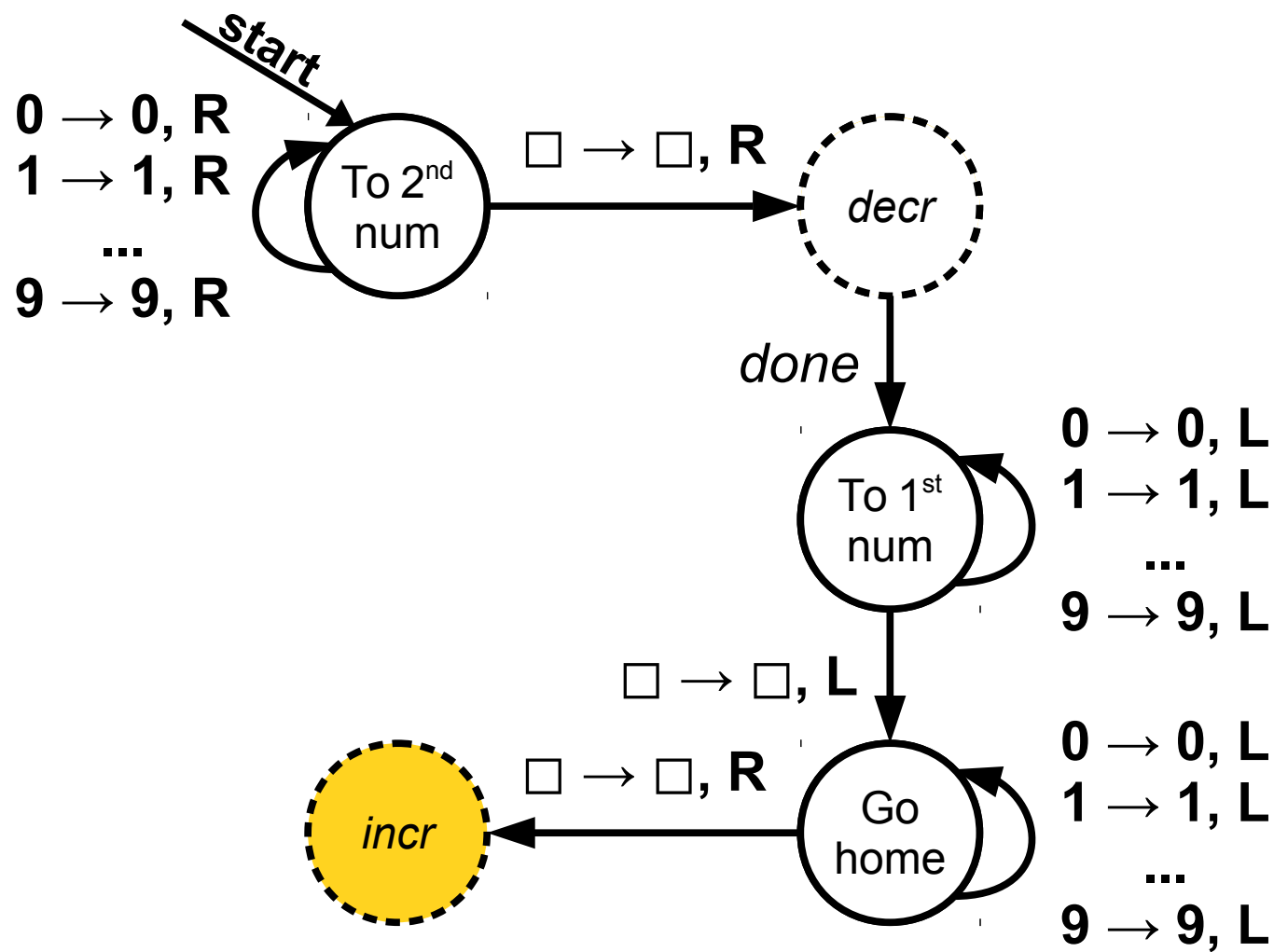


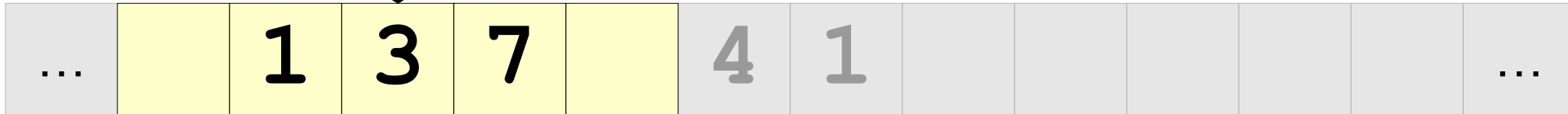
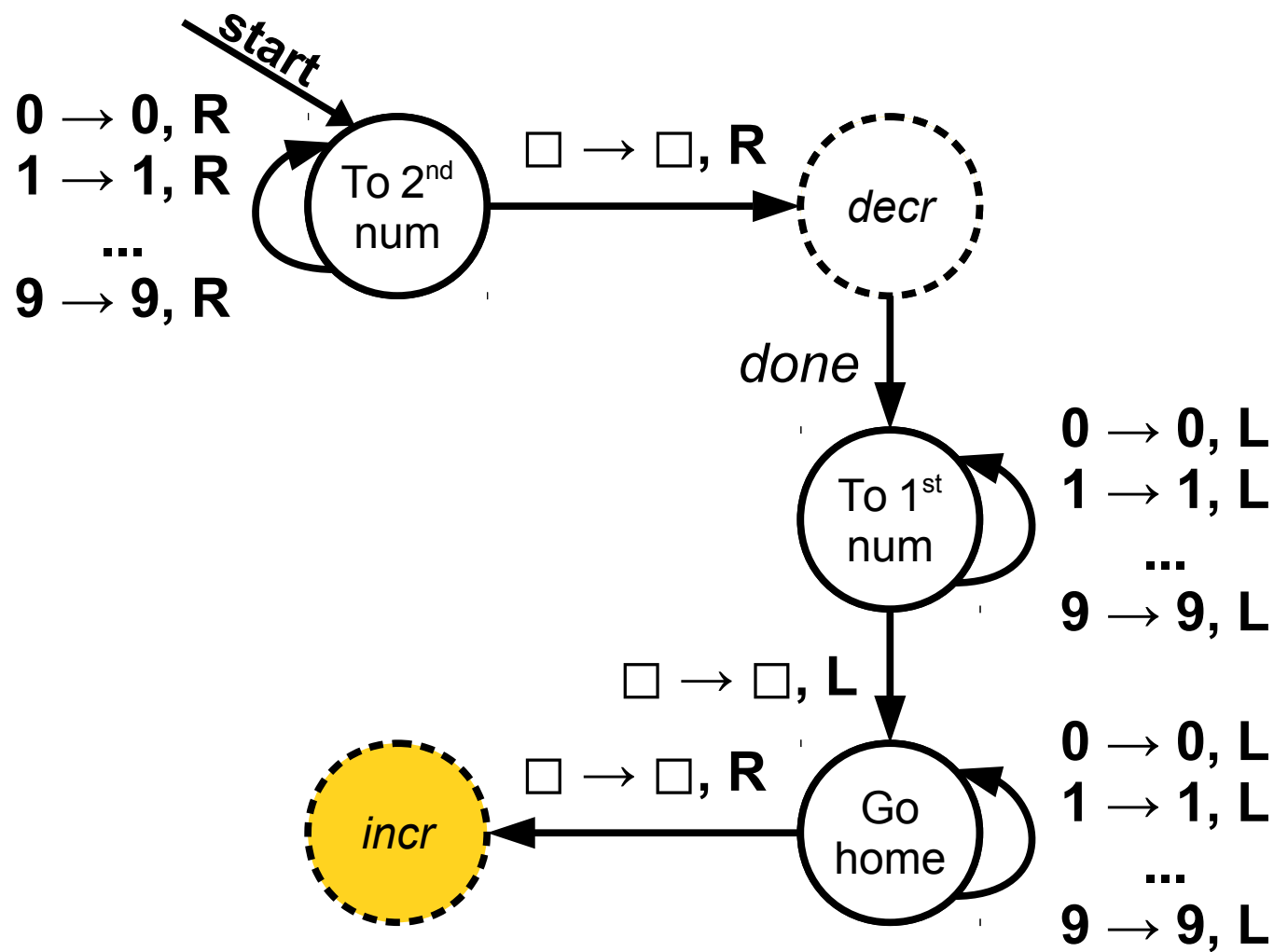


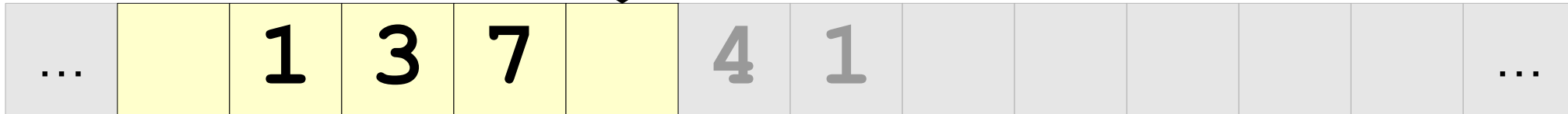
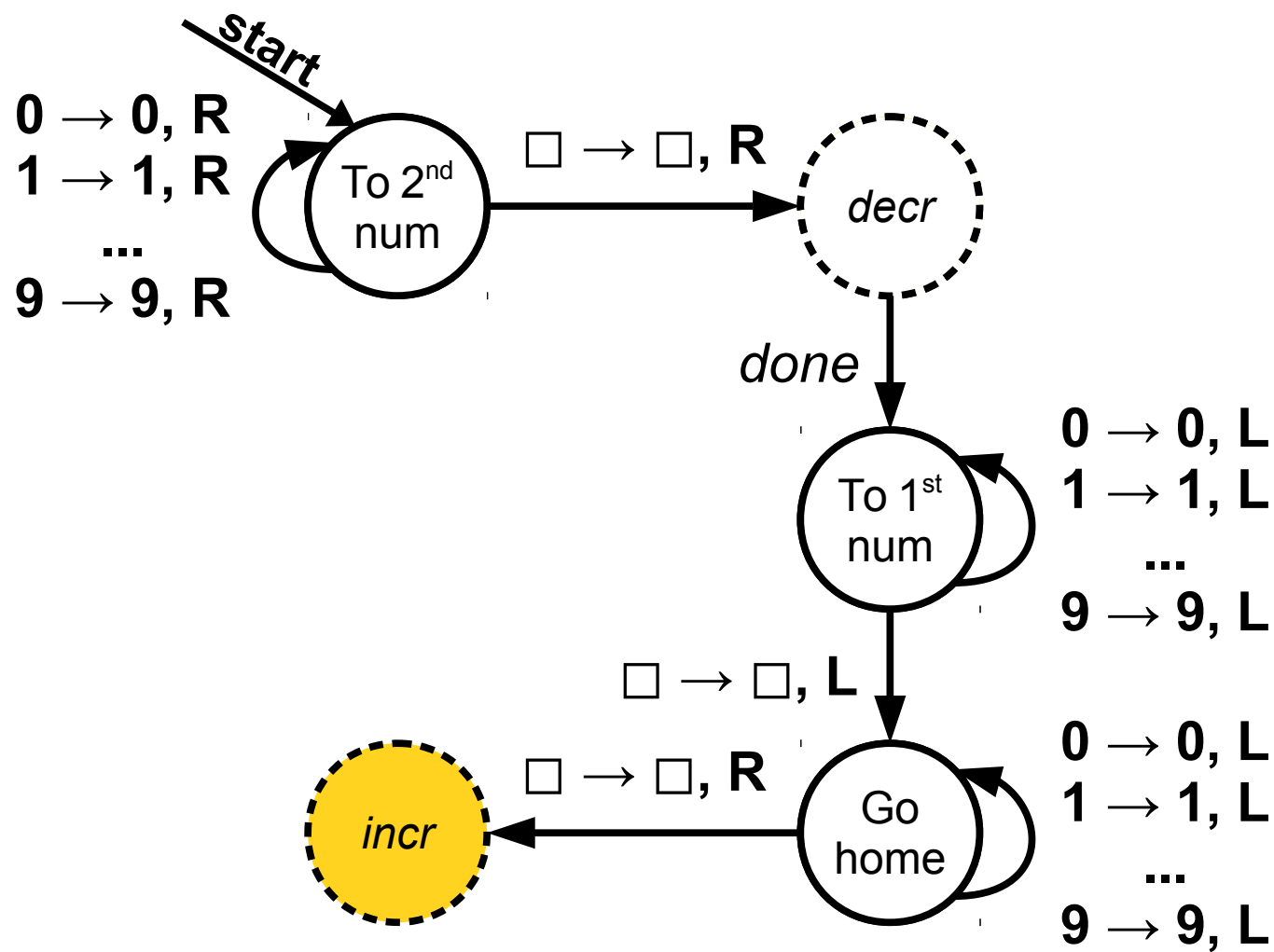


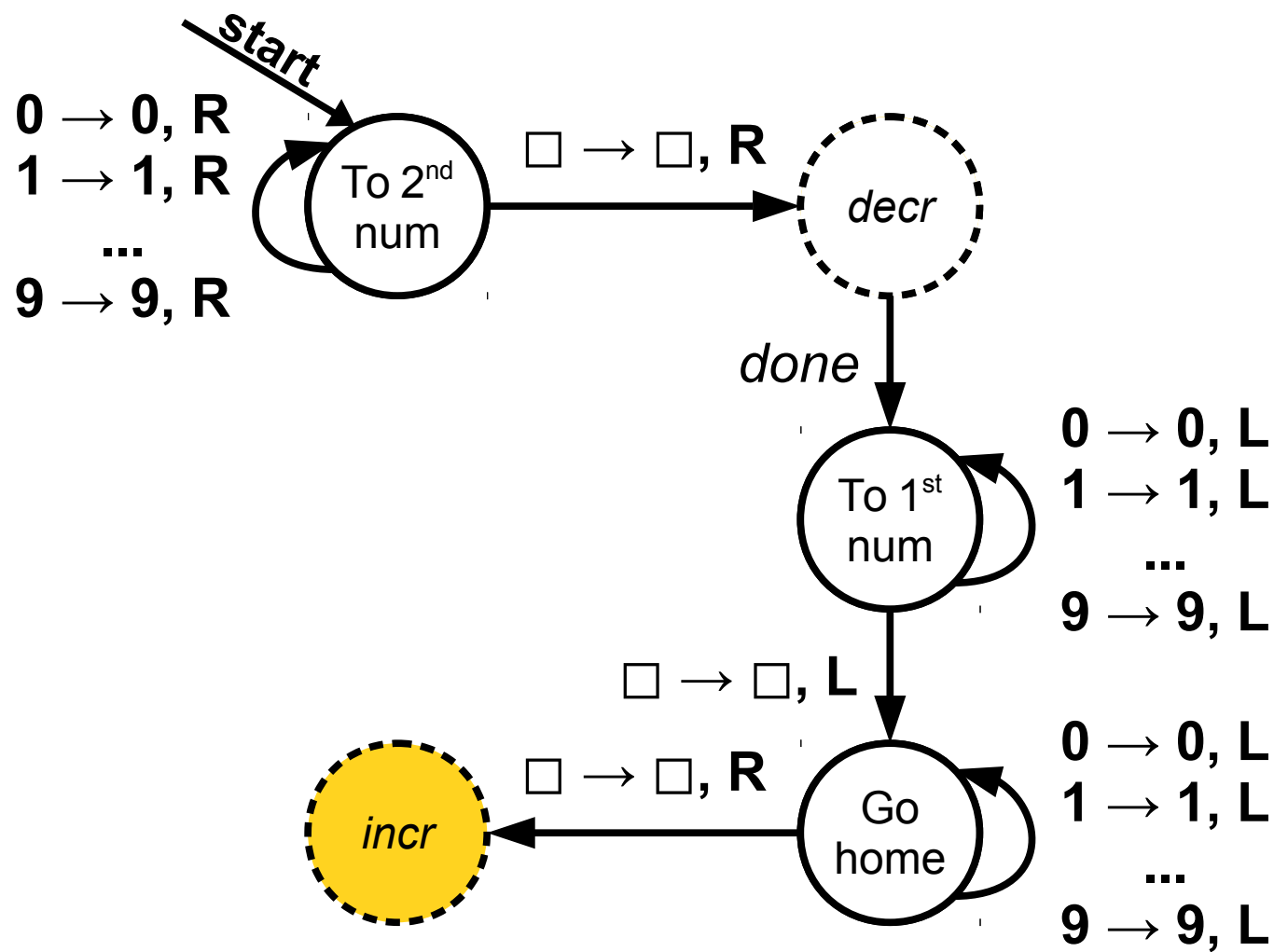


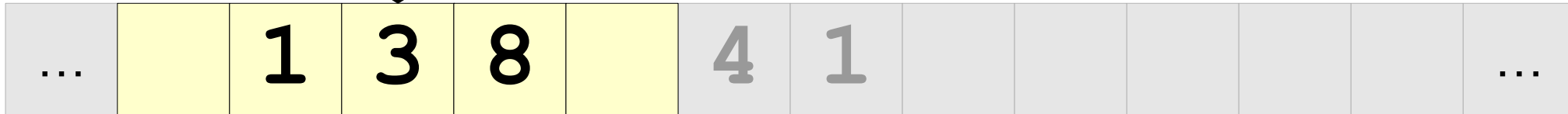
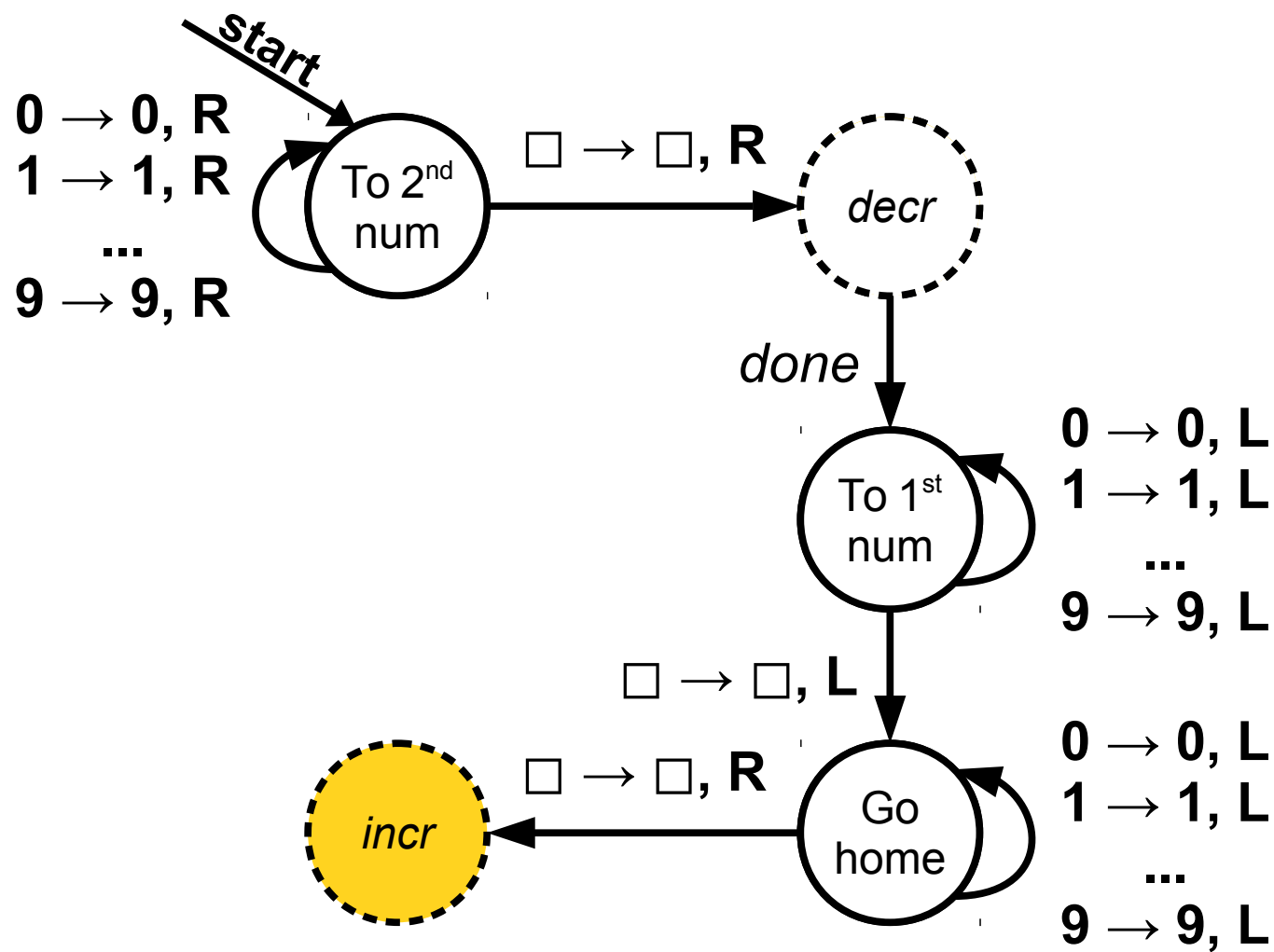


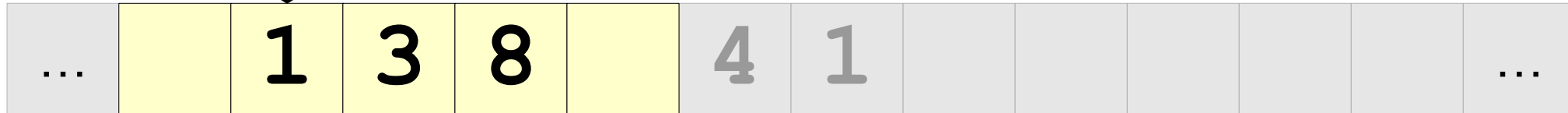
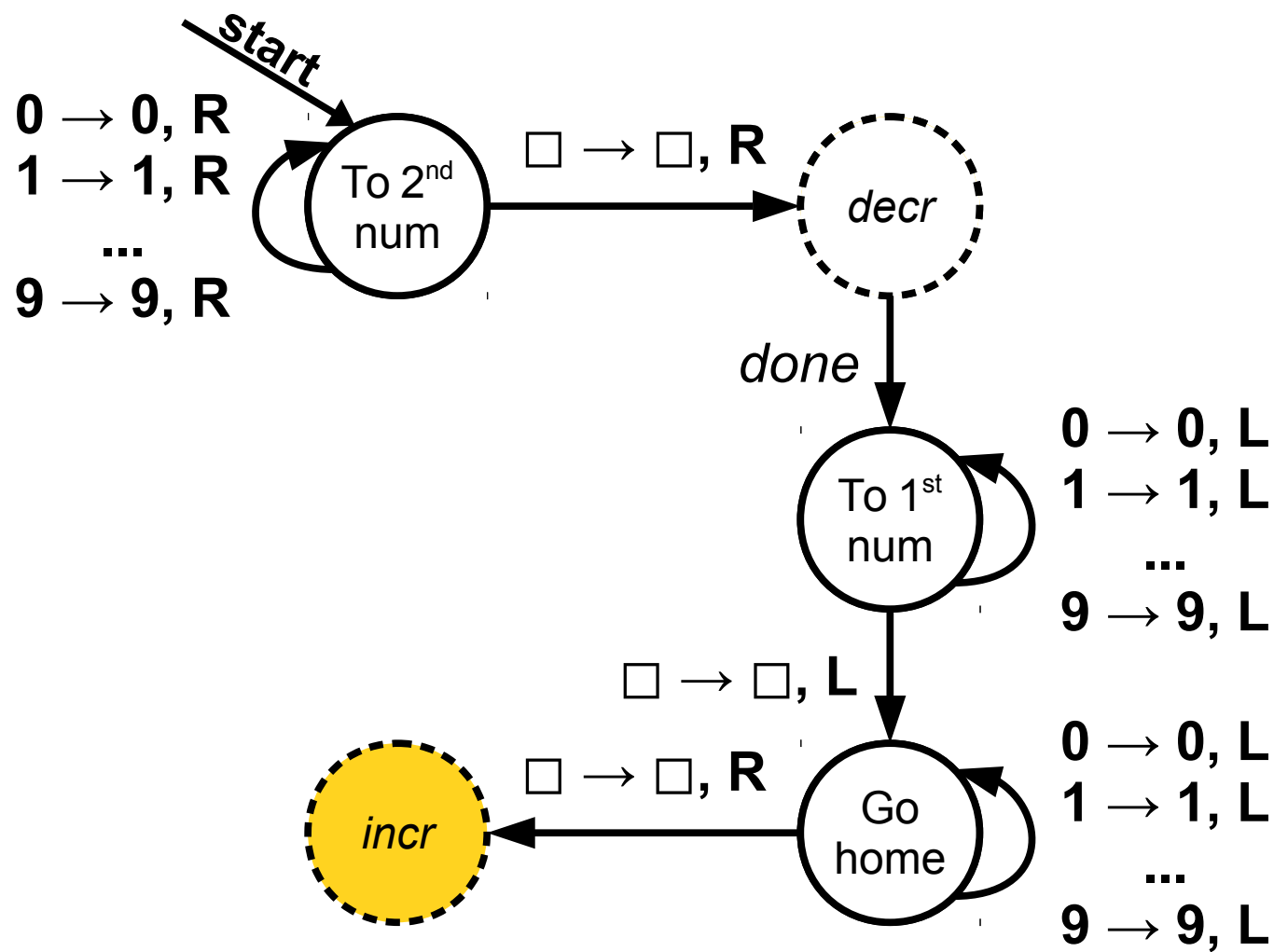


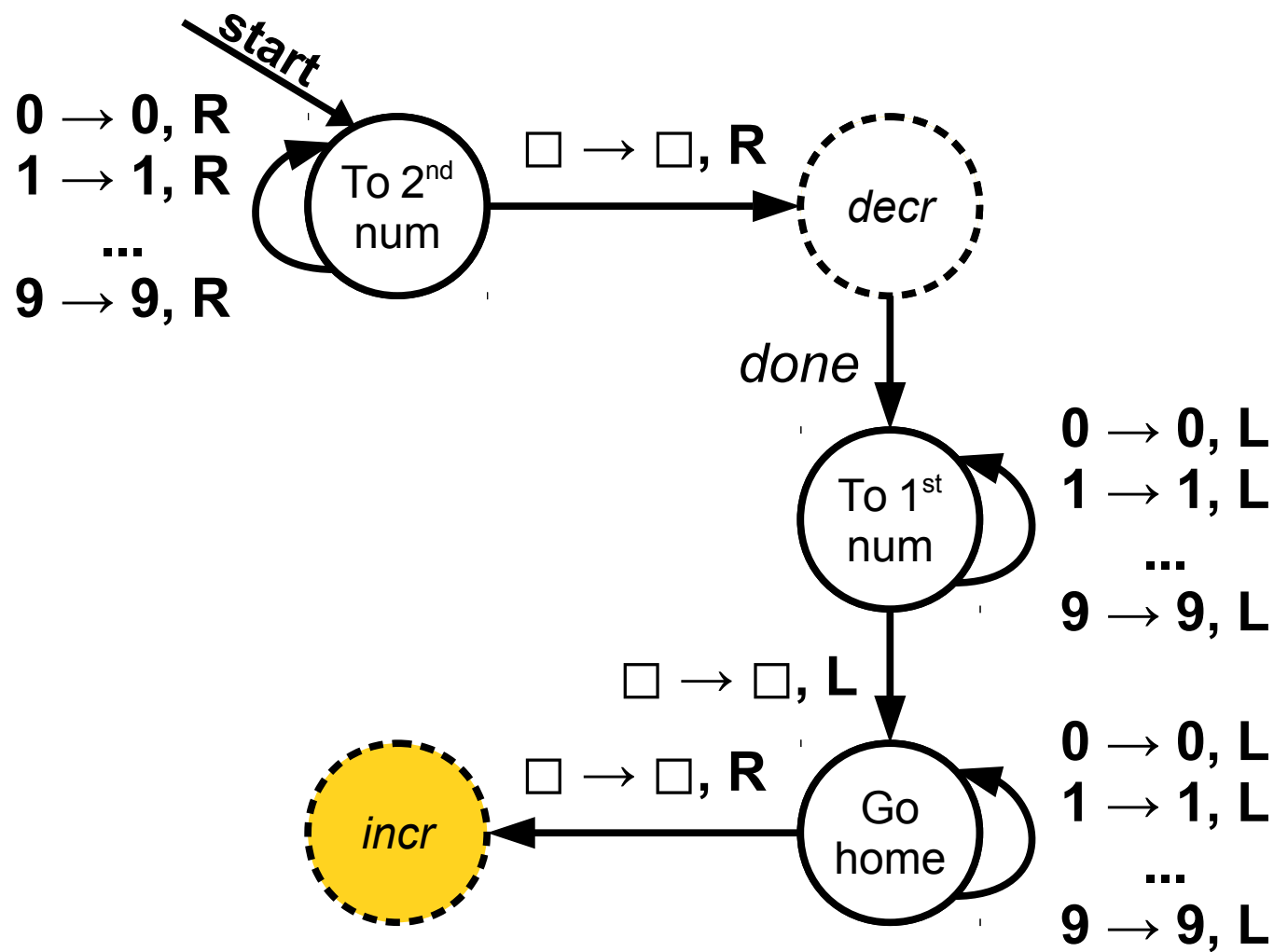


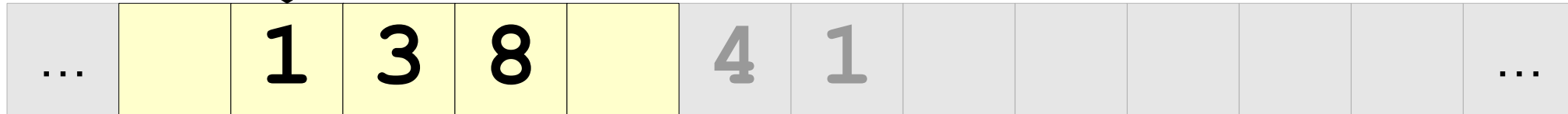
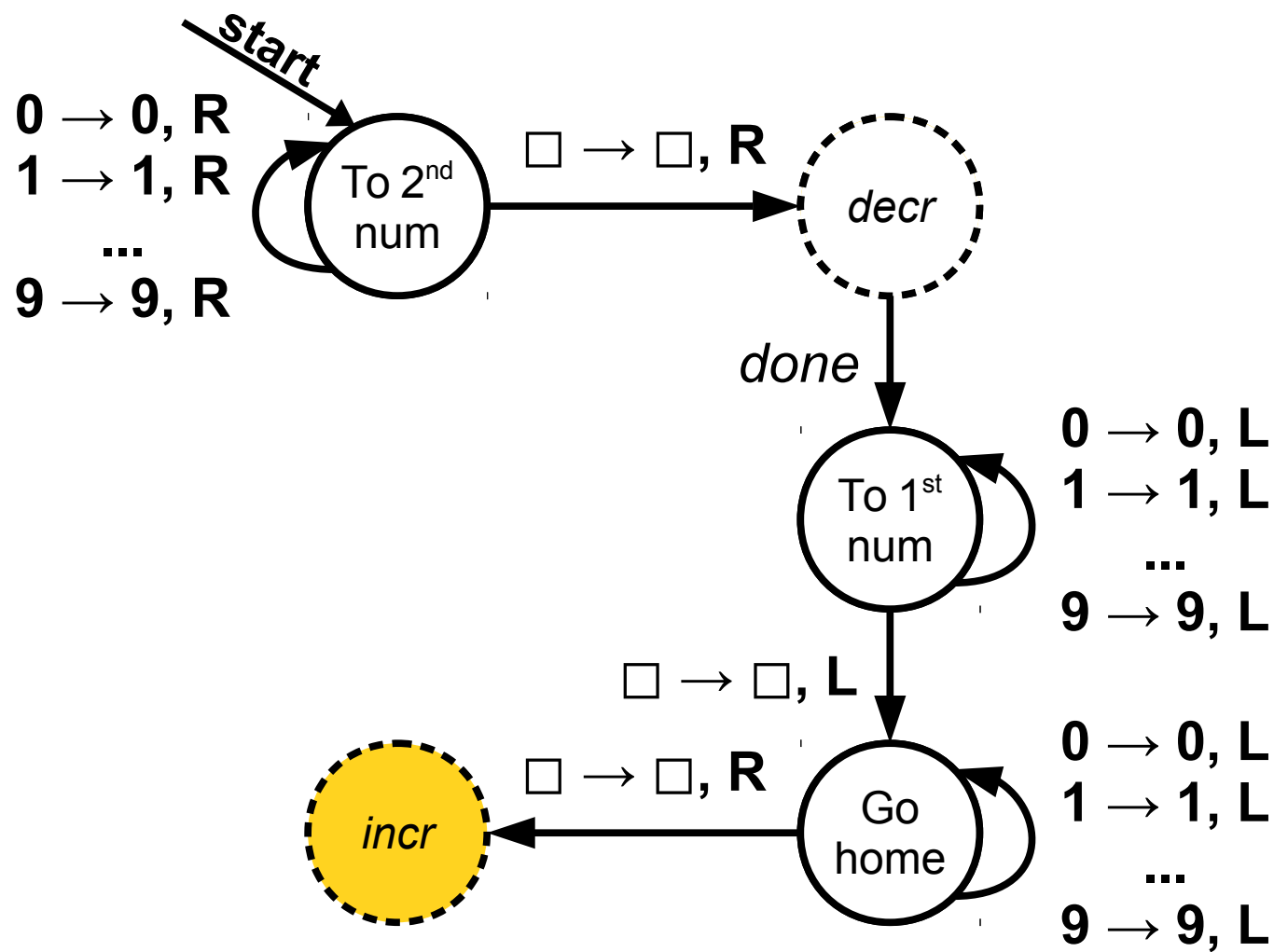


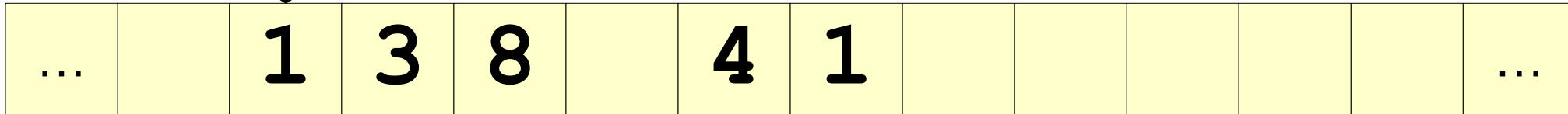
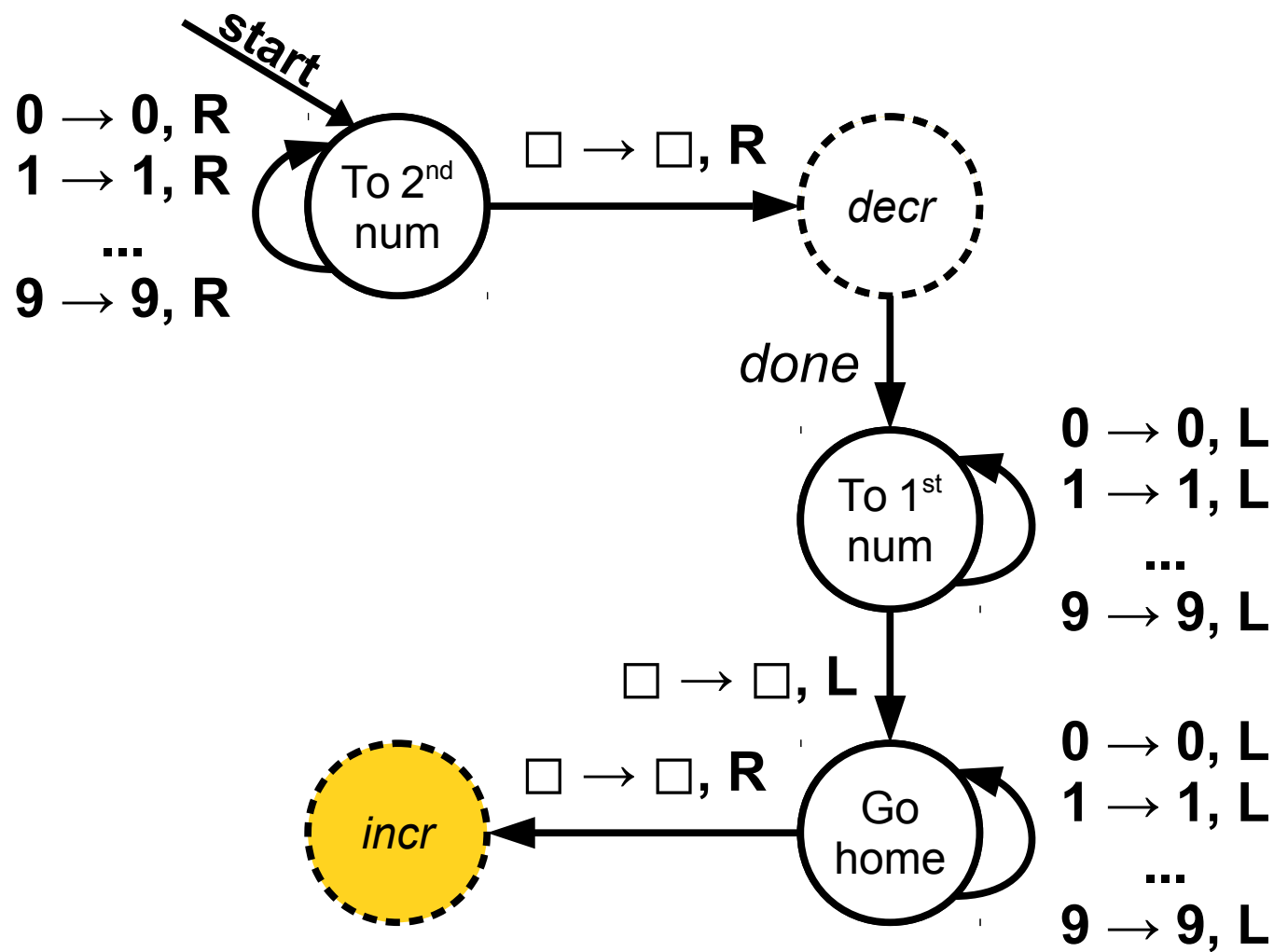


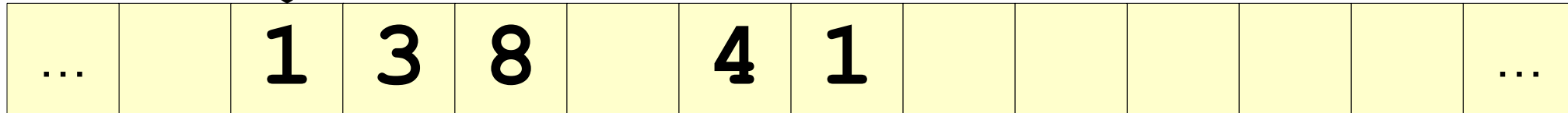
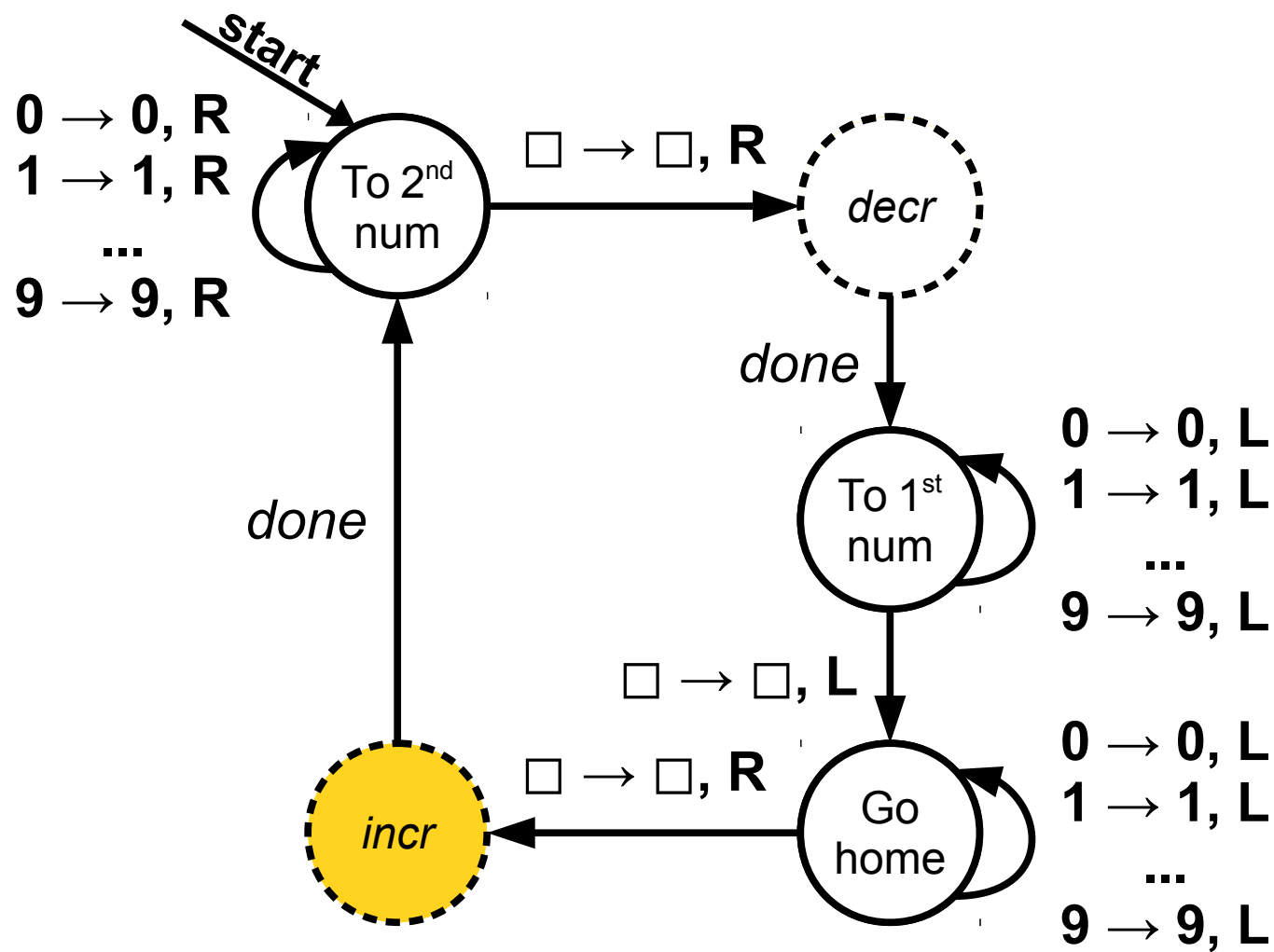


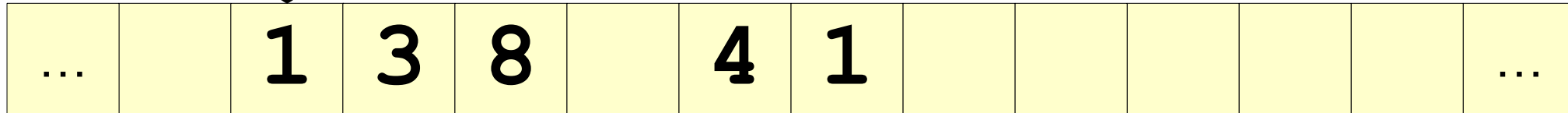
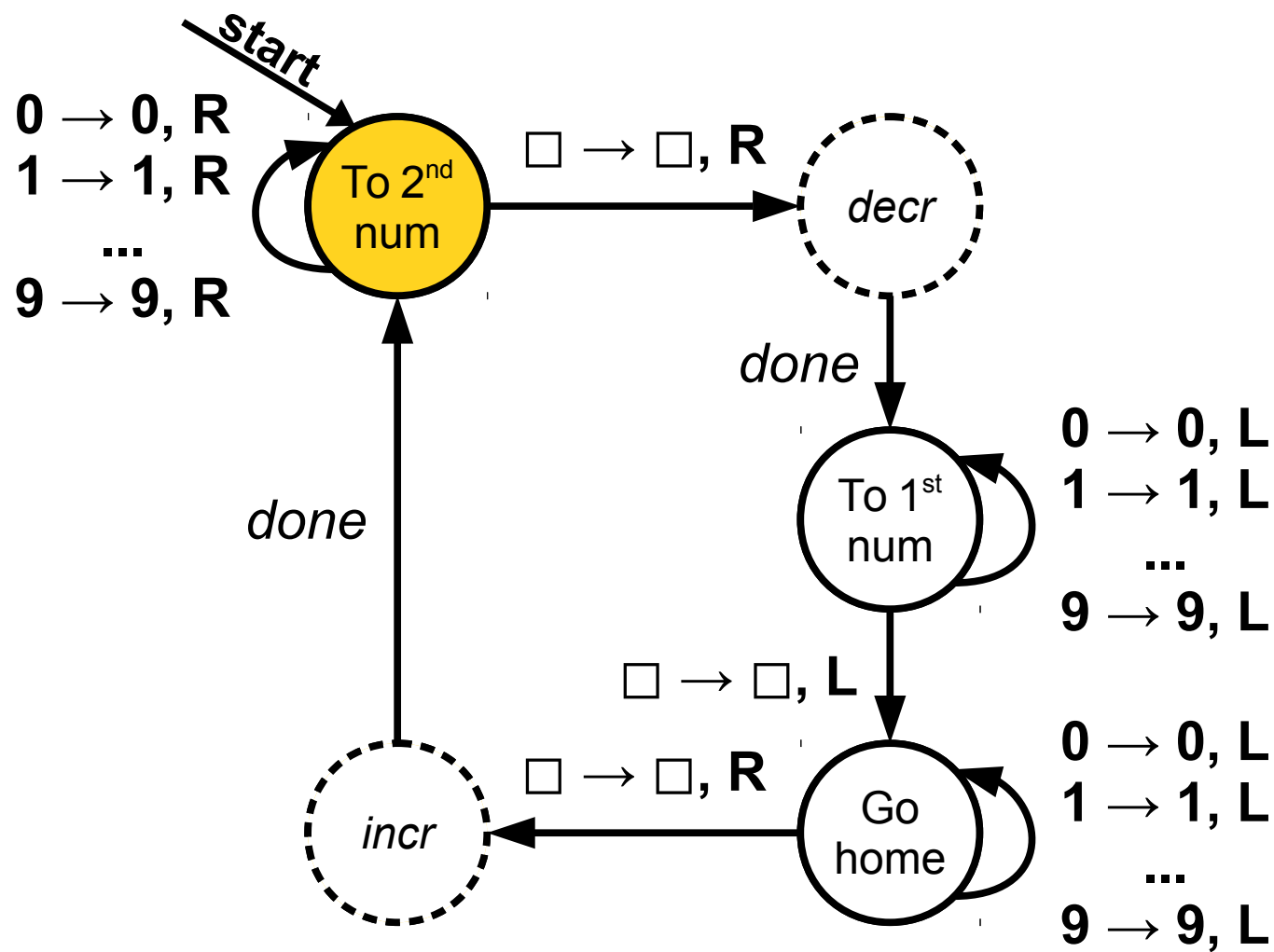


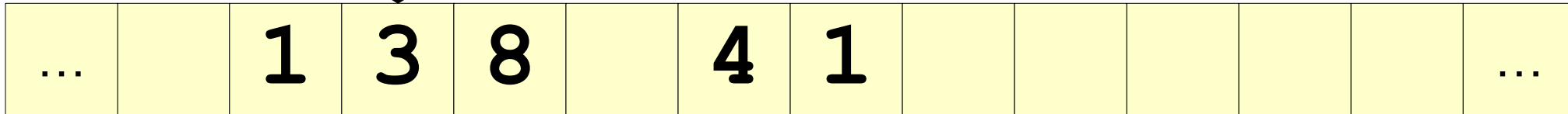
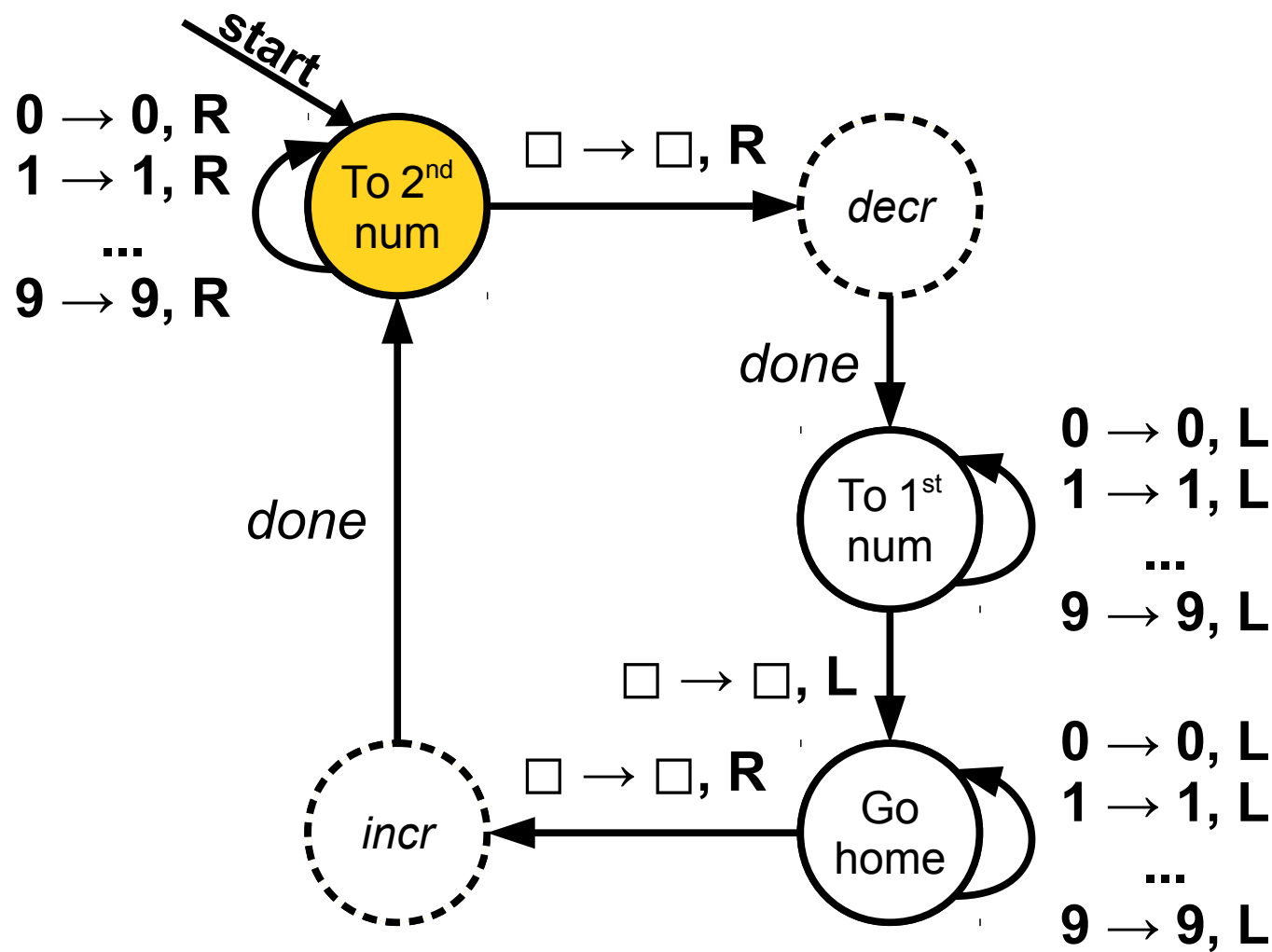


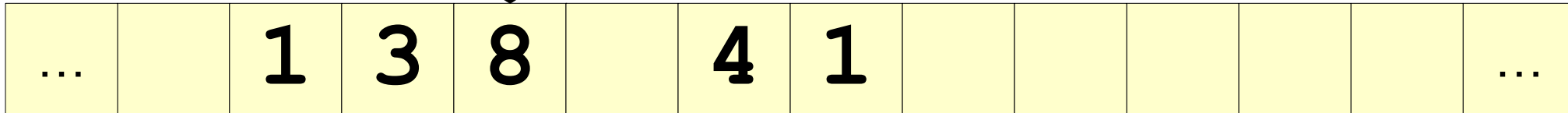
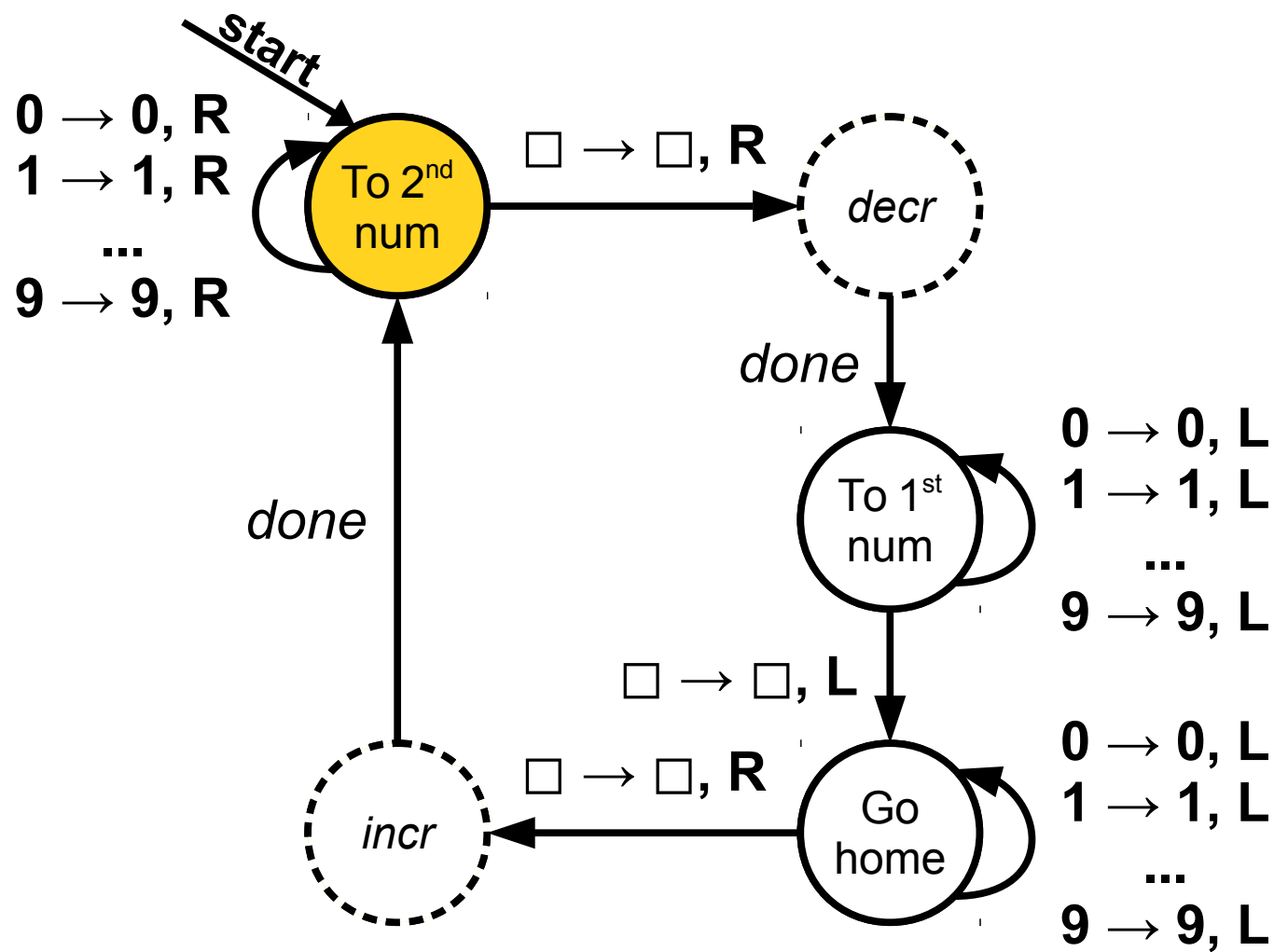


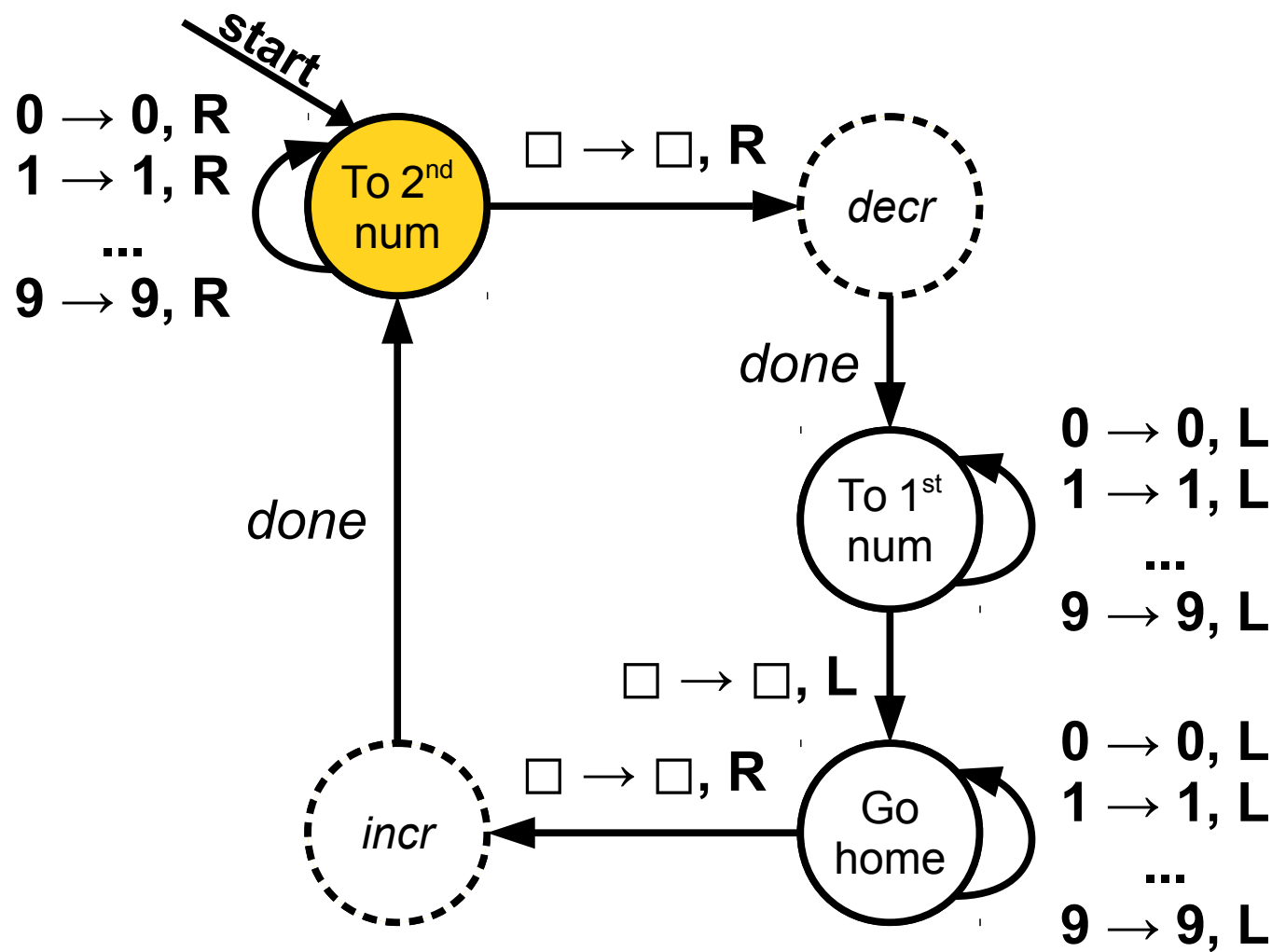




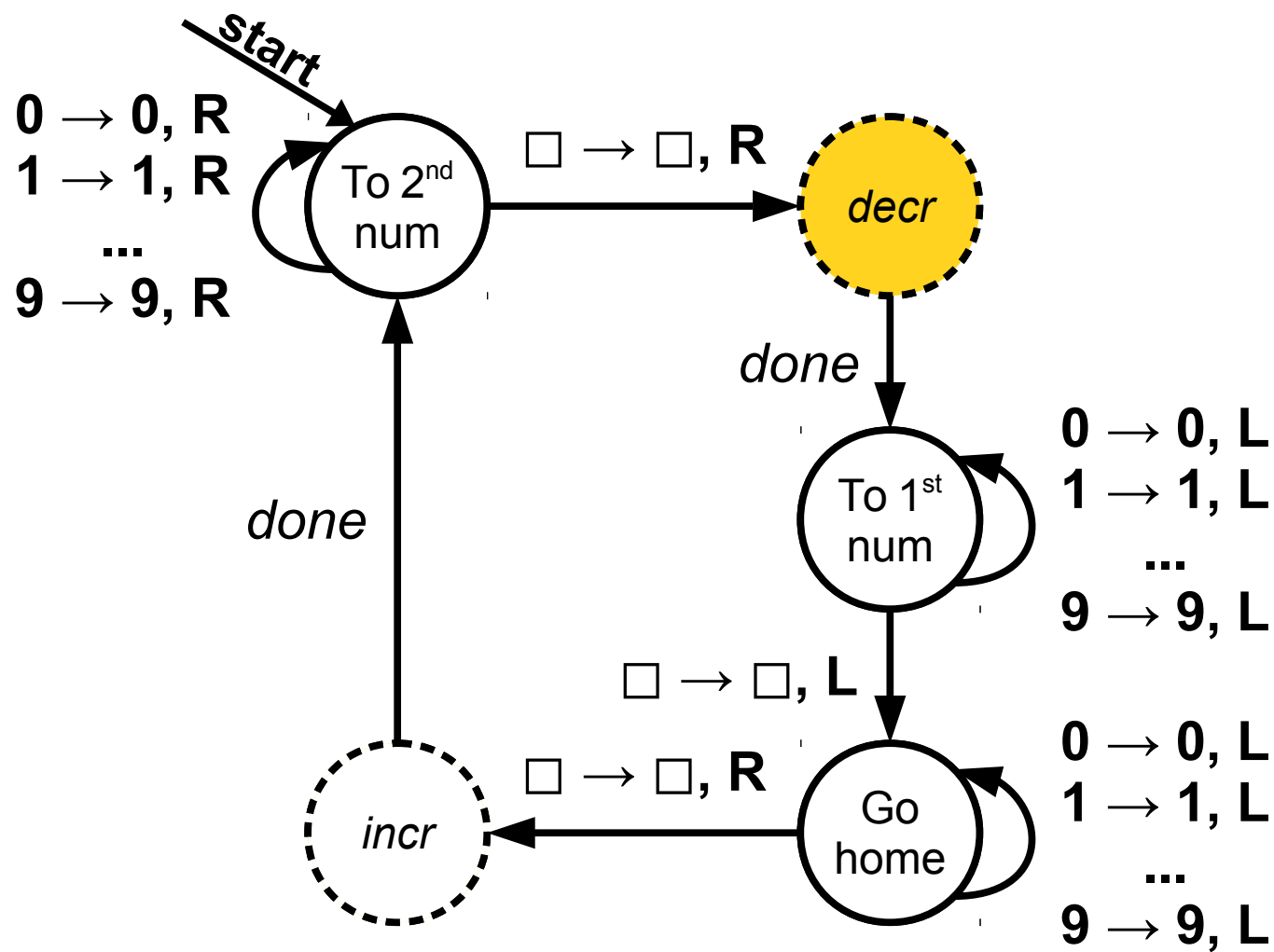




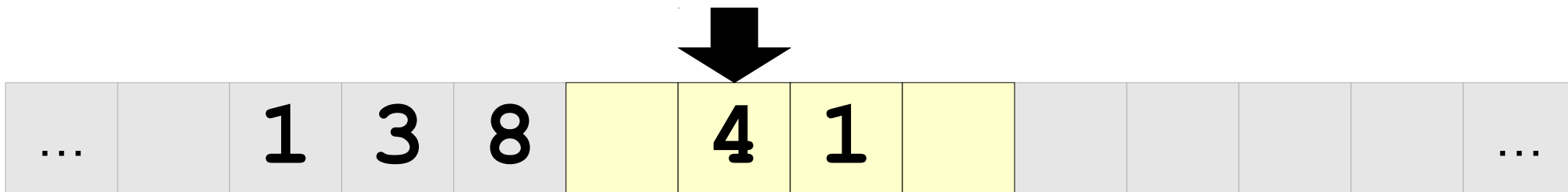
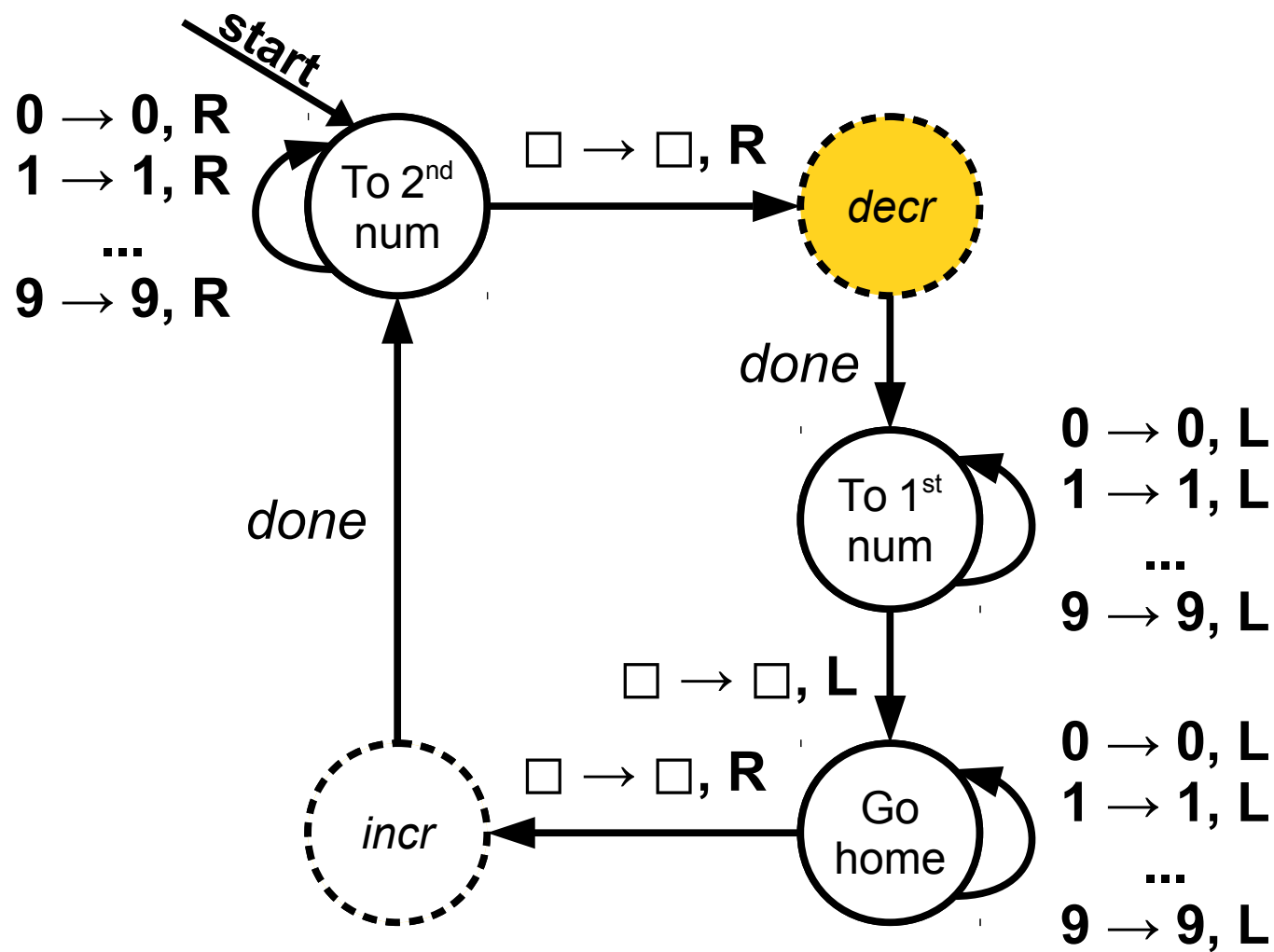


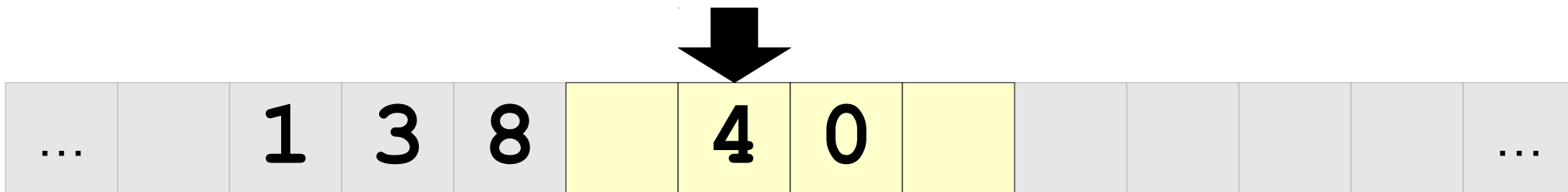
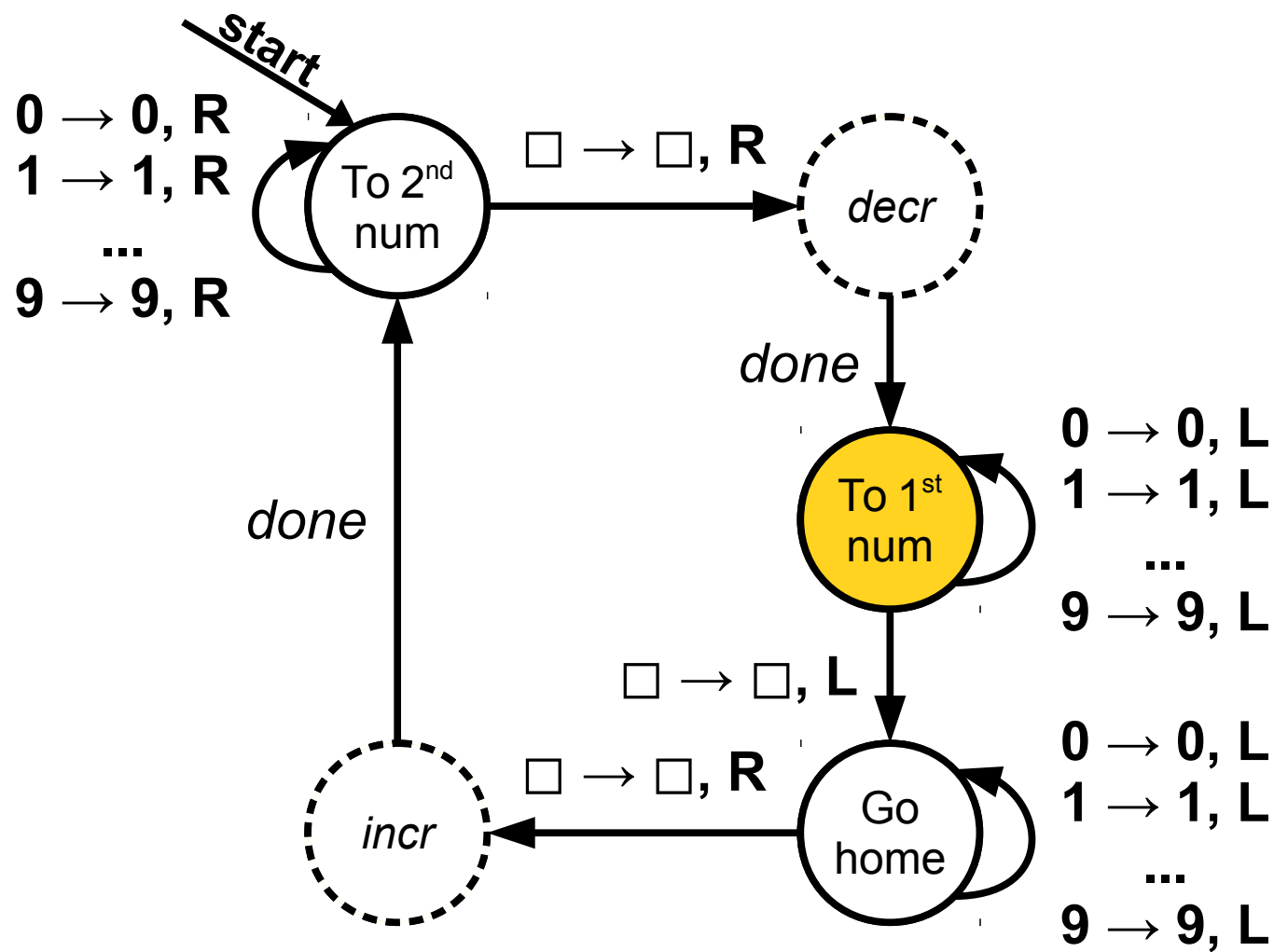


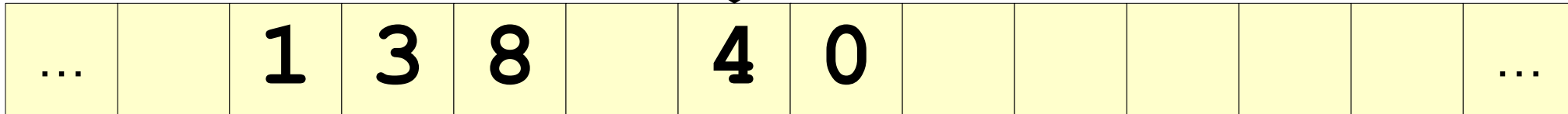
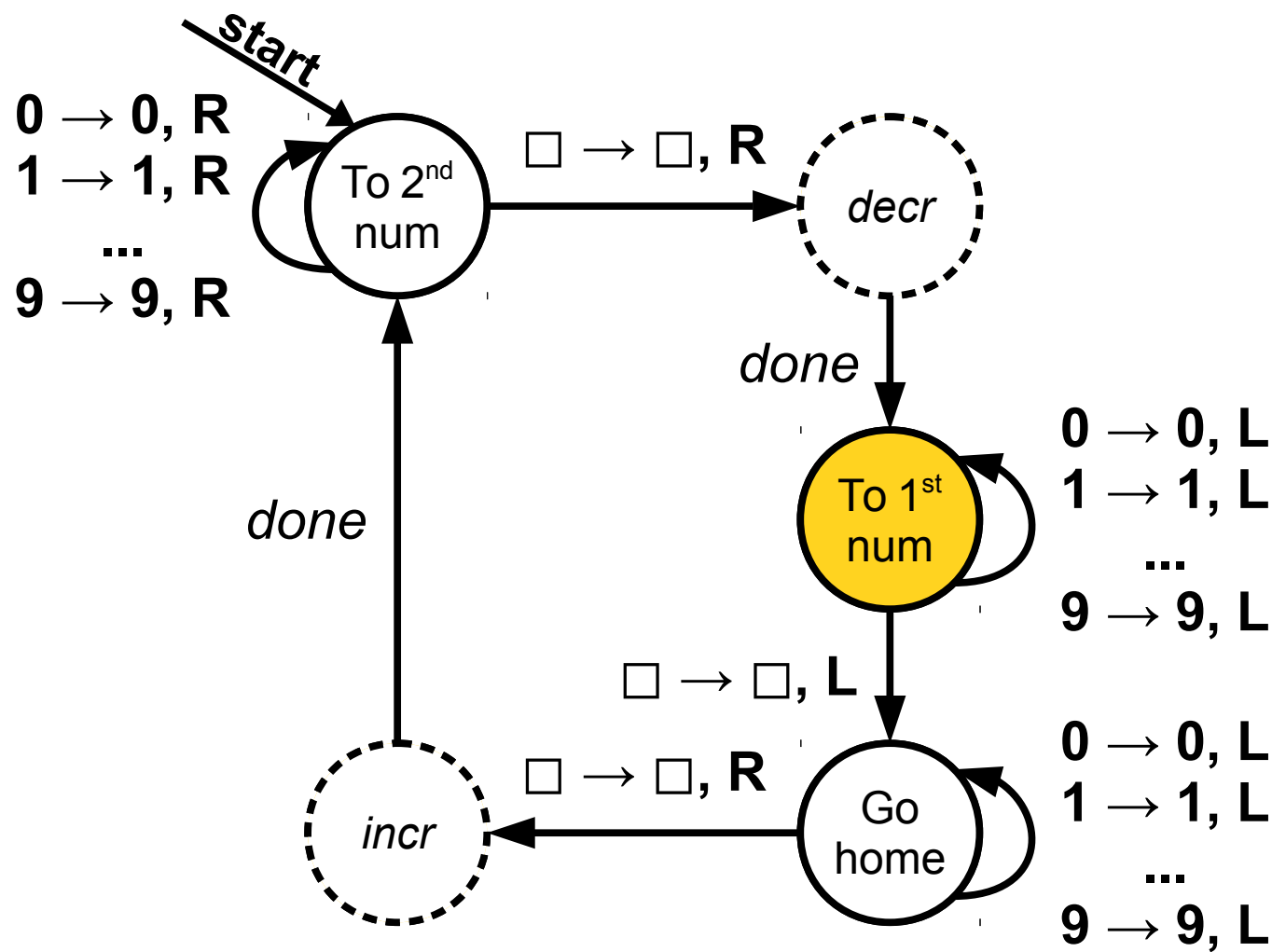
...		1	3	8		4	1						...
-----	--	---	---	---	--	---	---	--	--	--	--	--	-----

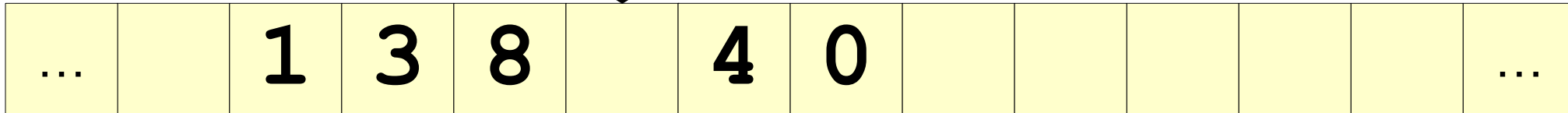
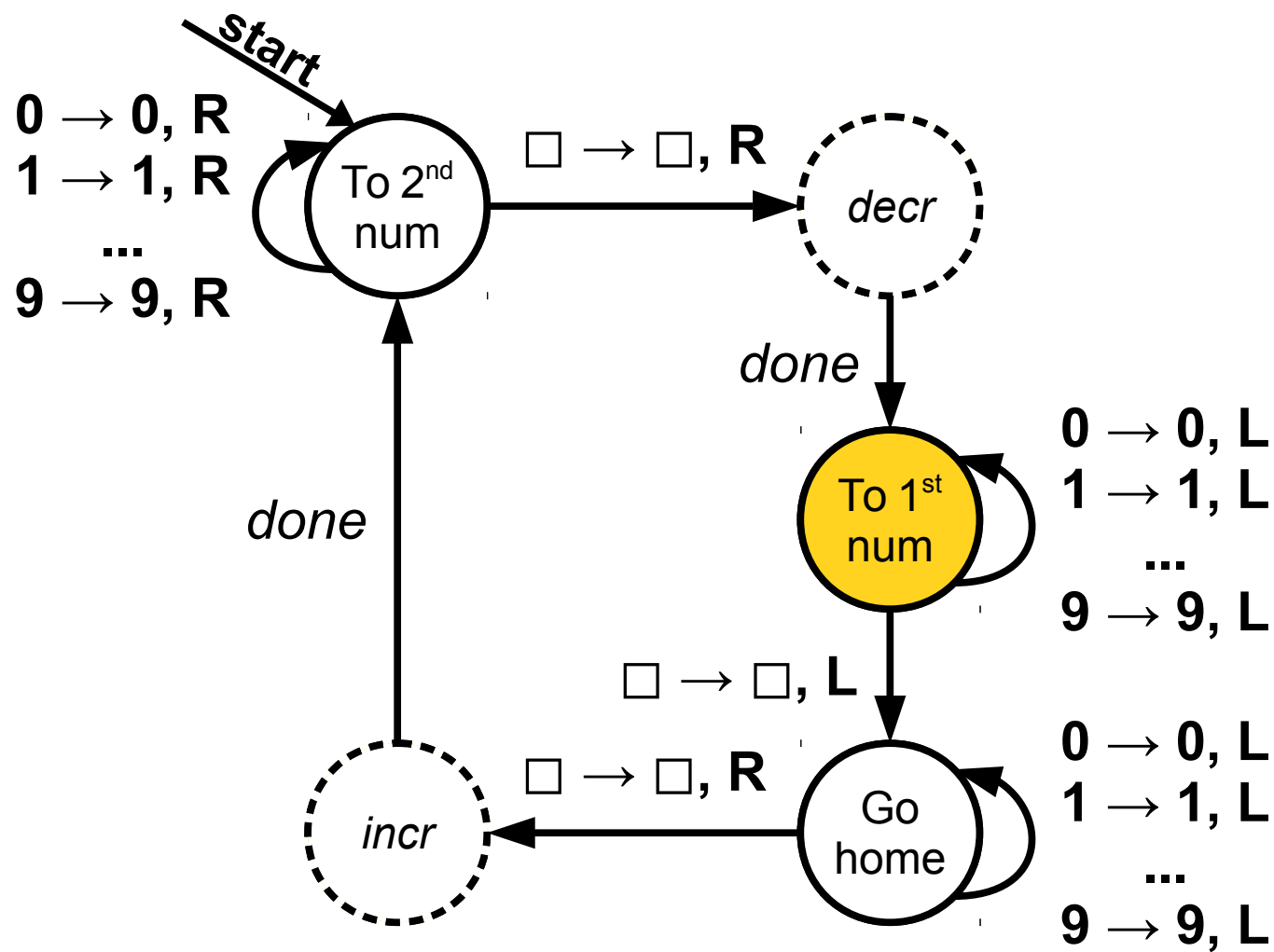


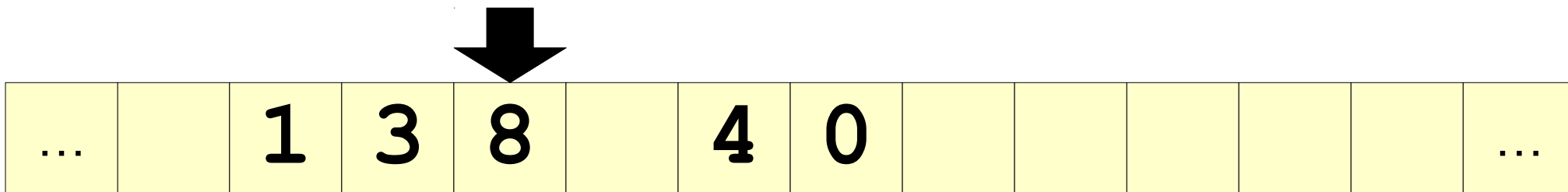
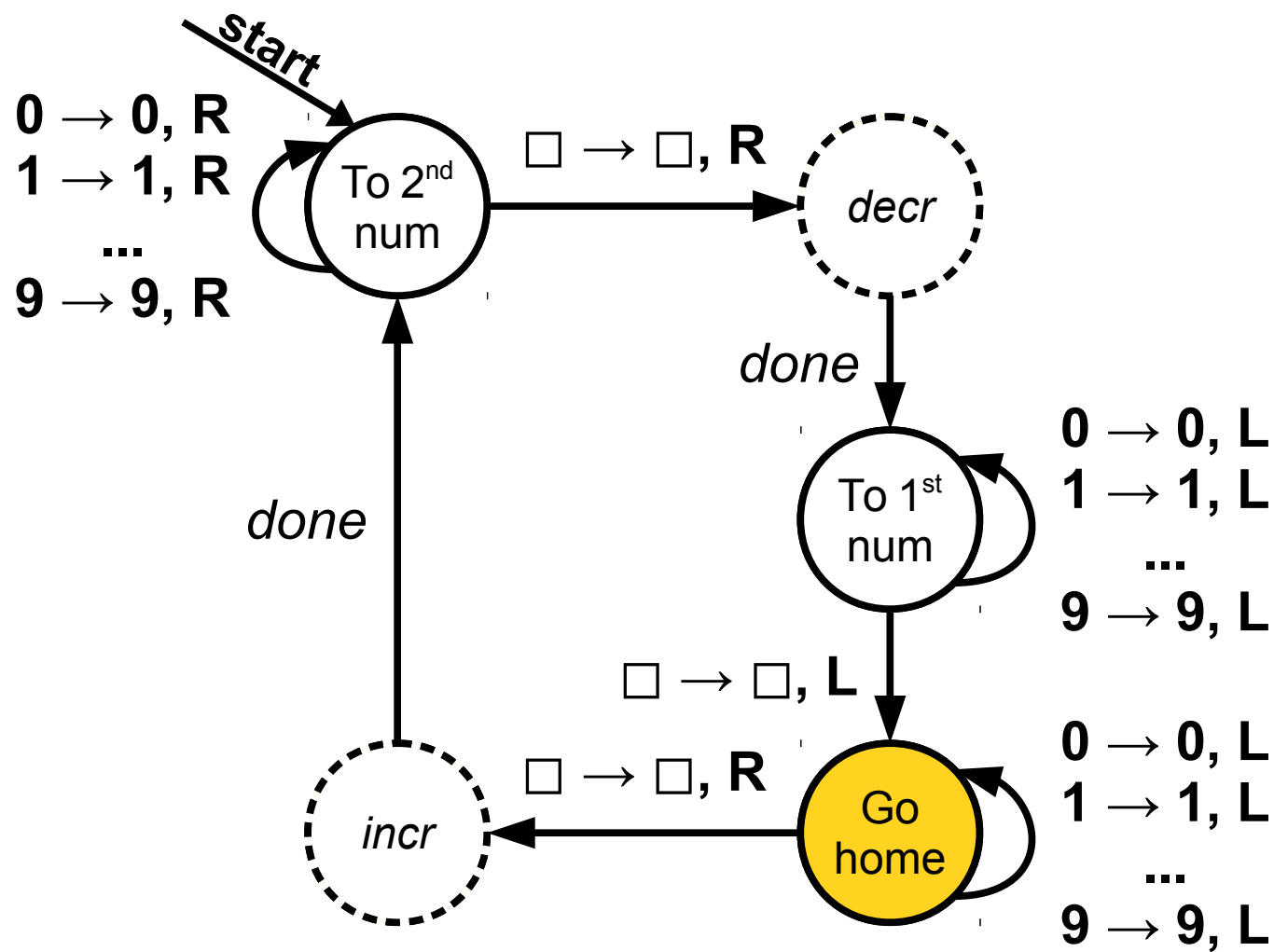
...	1	3	8	4	1							...
-----	---	---	---	---	---	--	--	--	--	--	--	-----

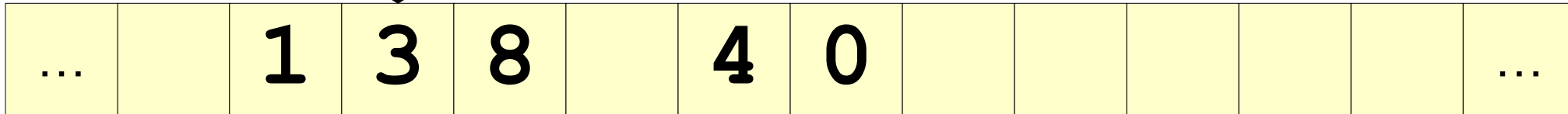
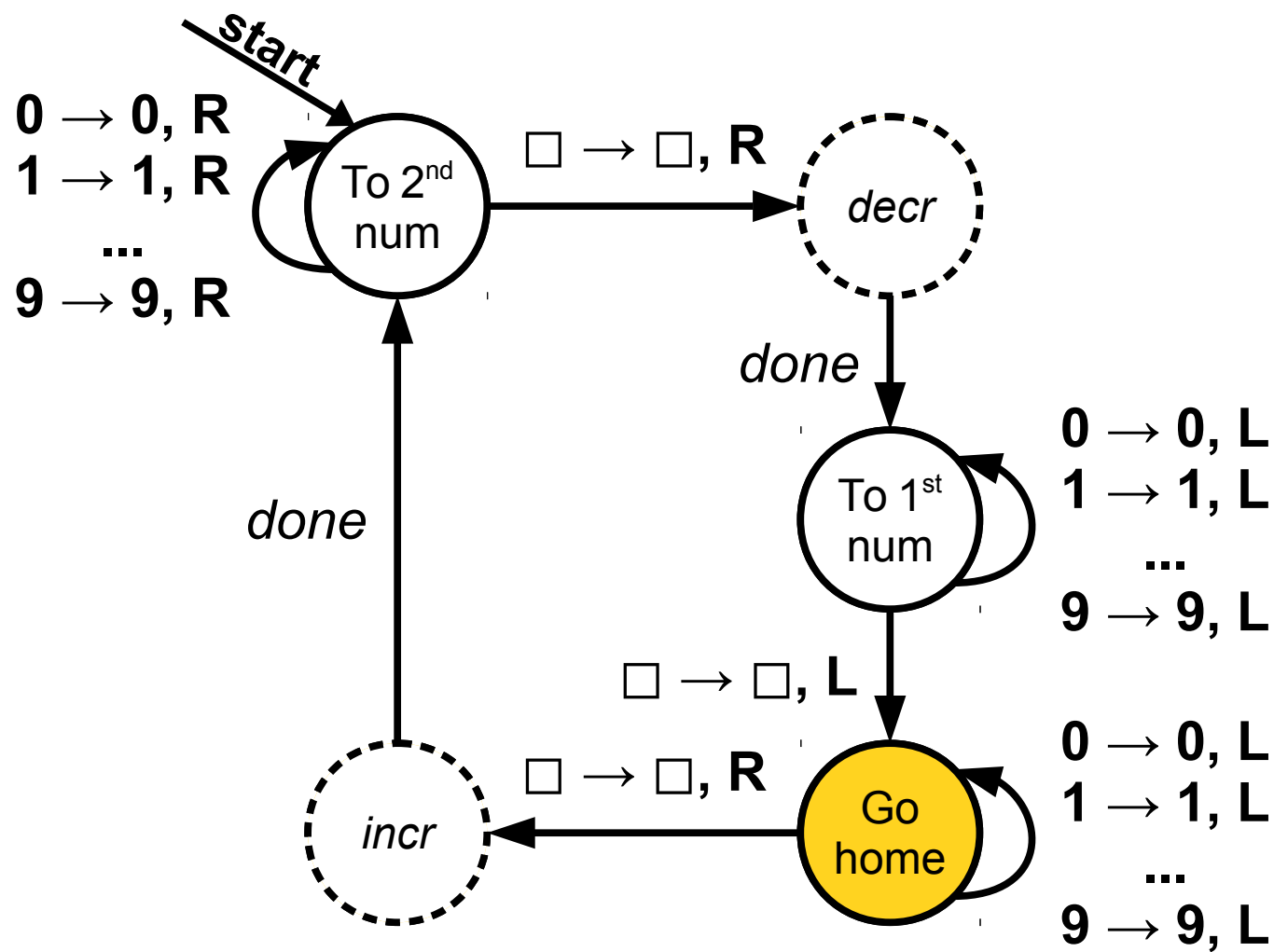


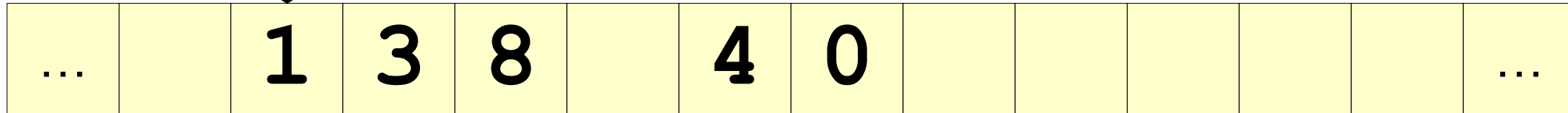
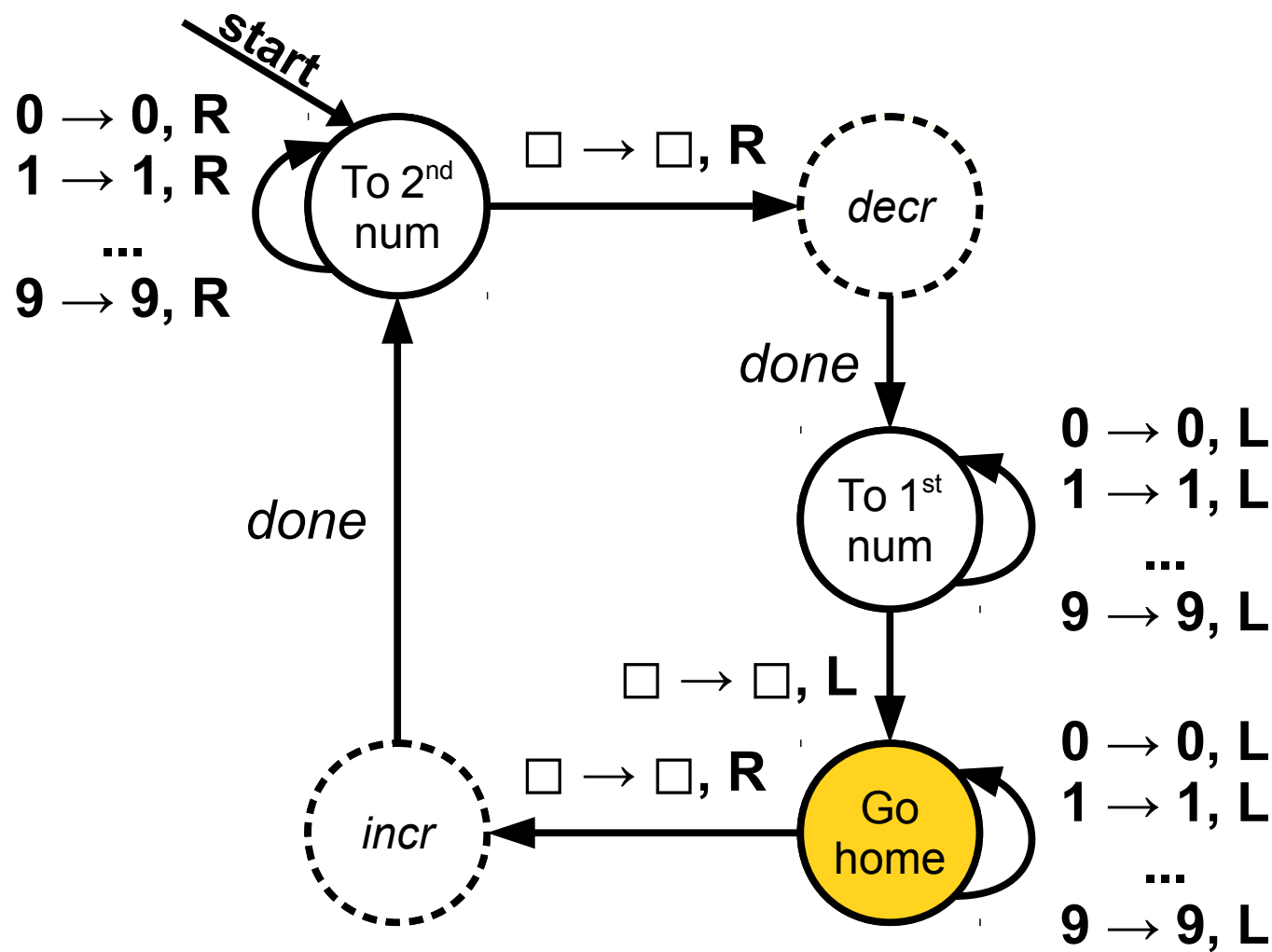


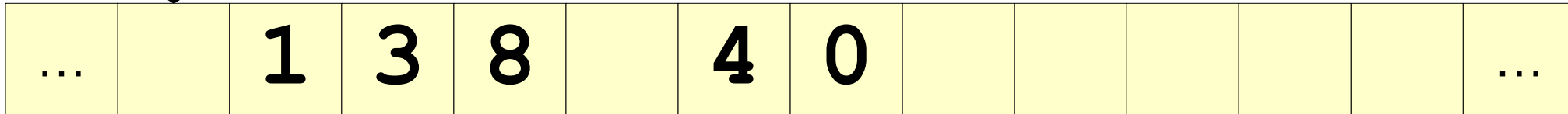
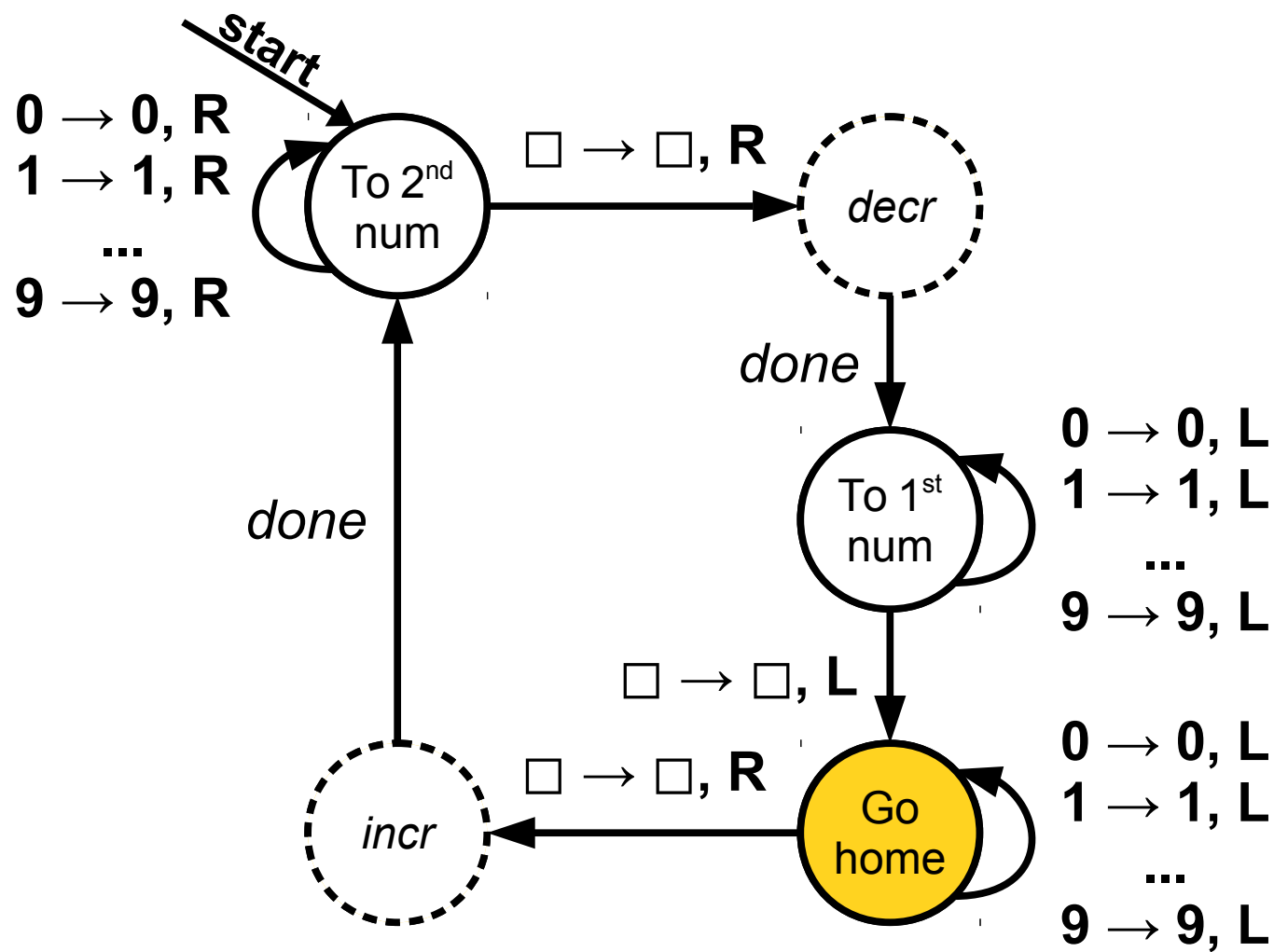


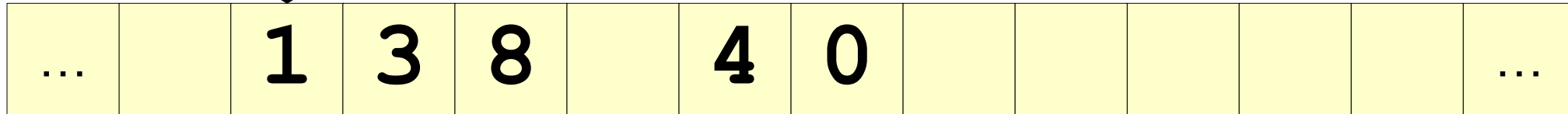
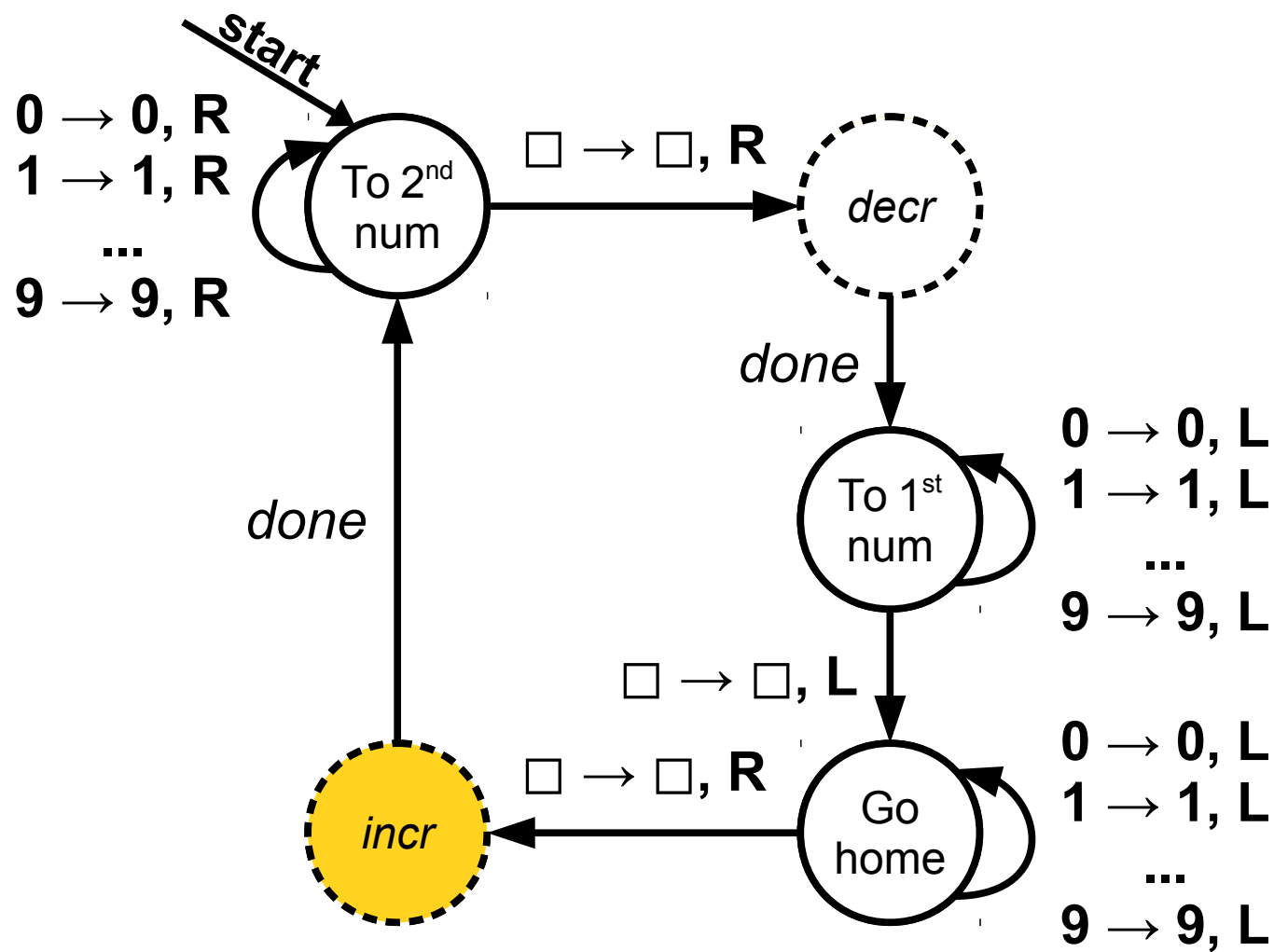


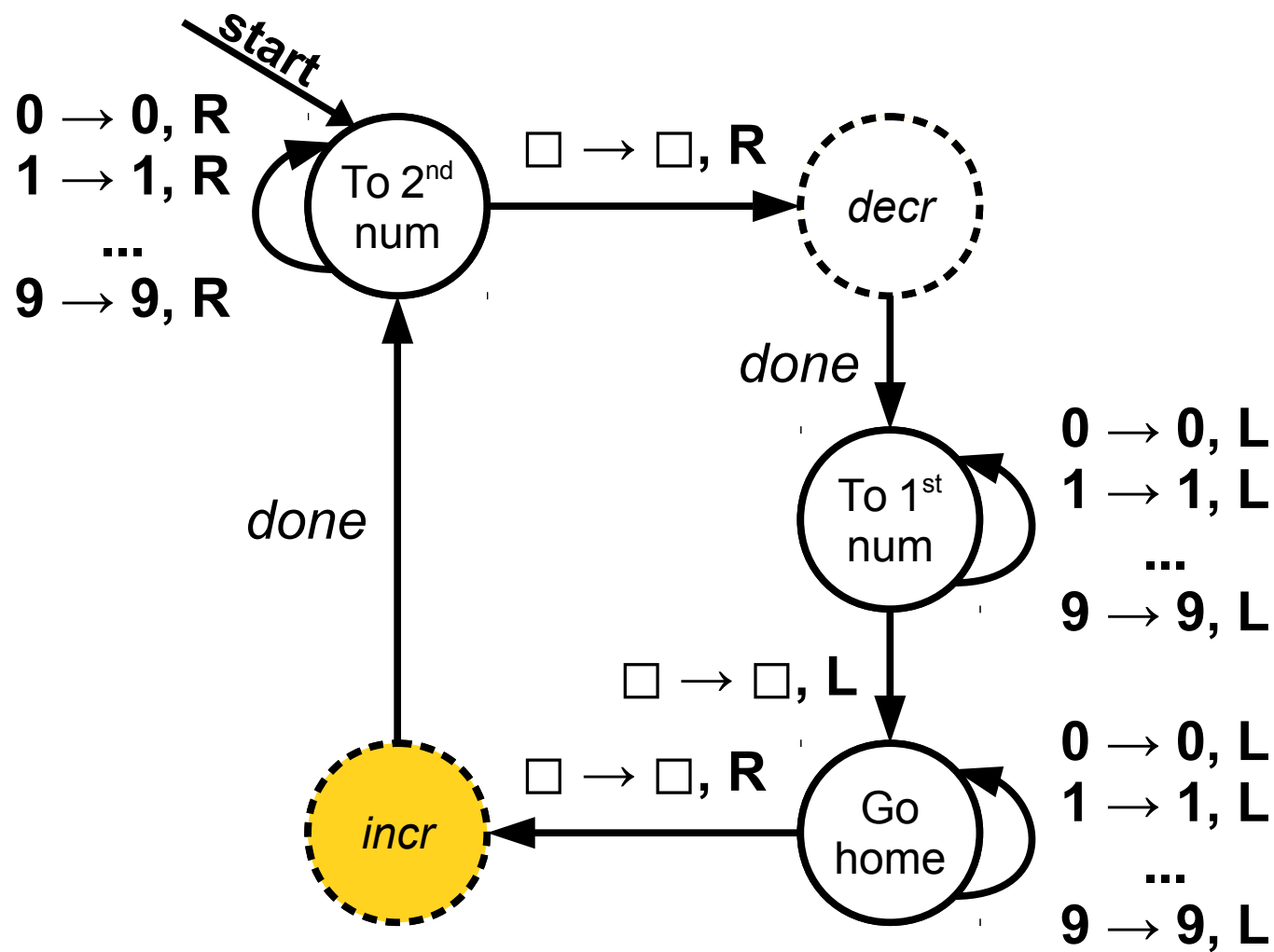


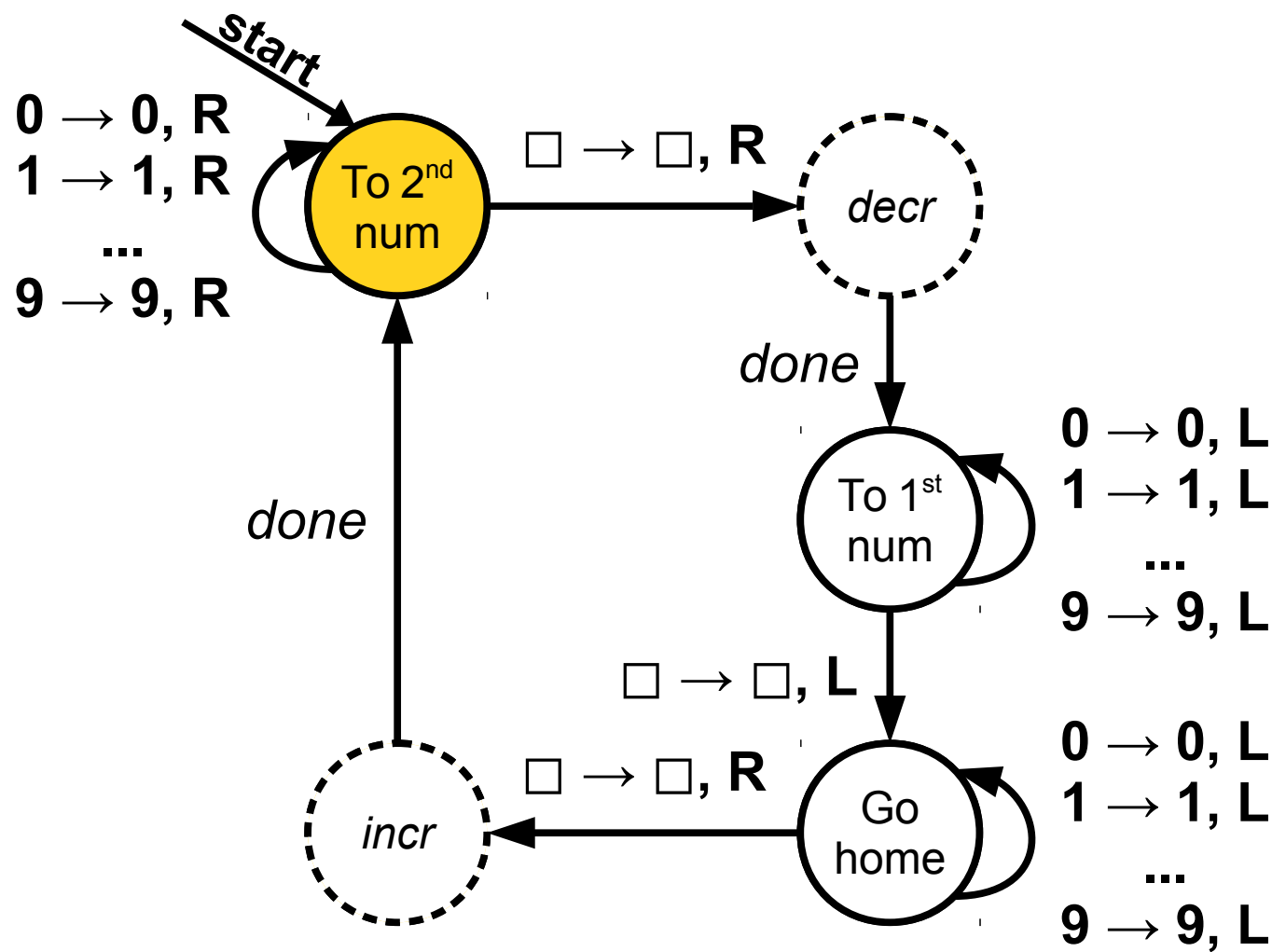


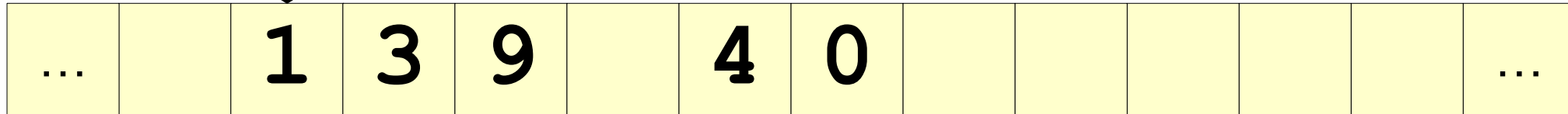
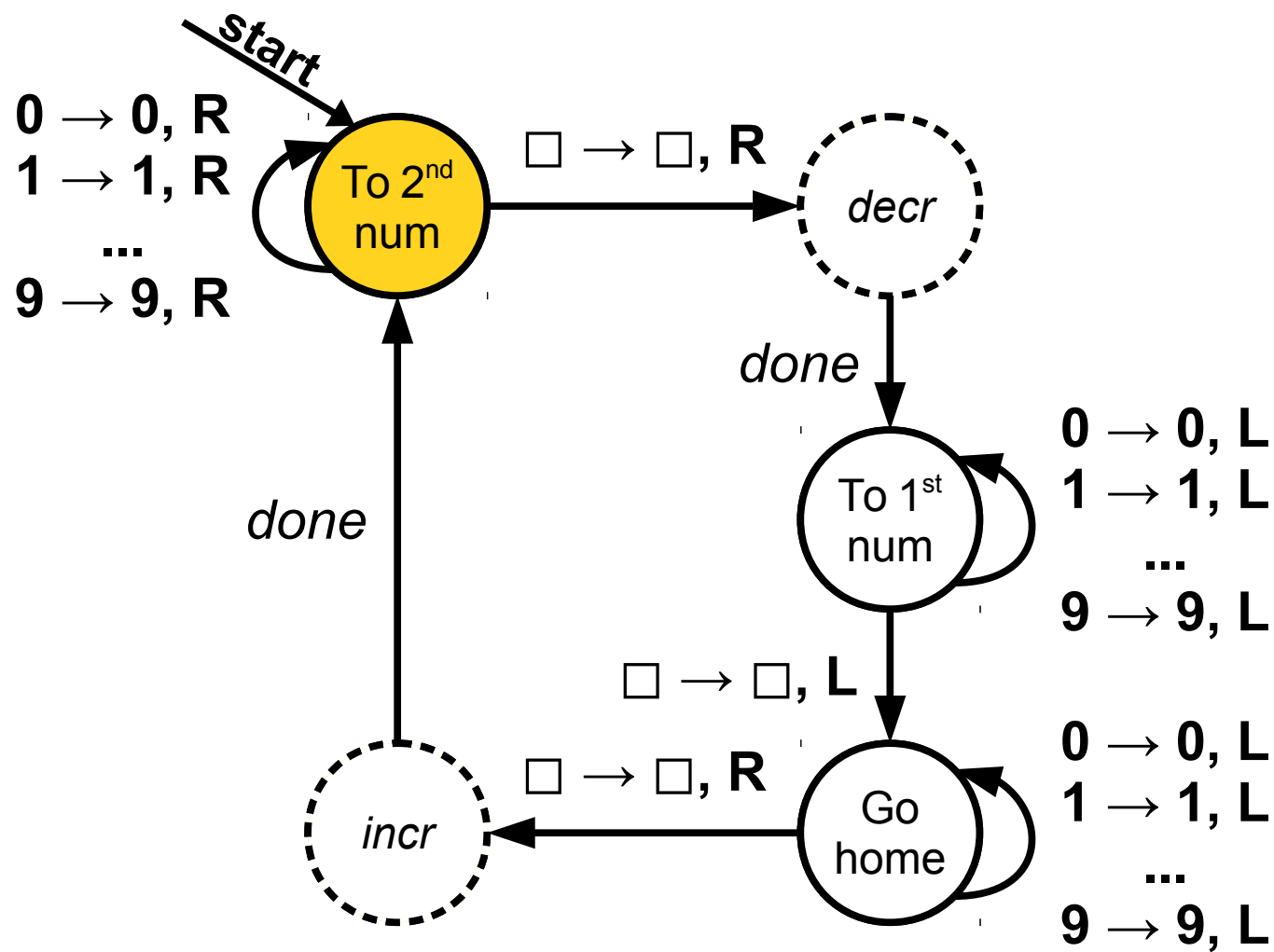


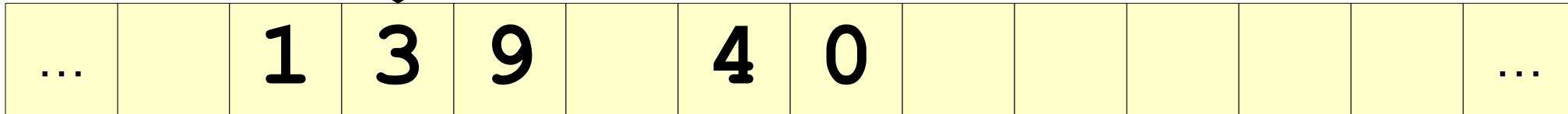
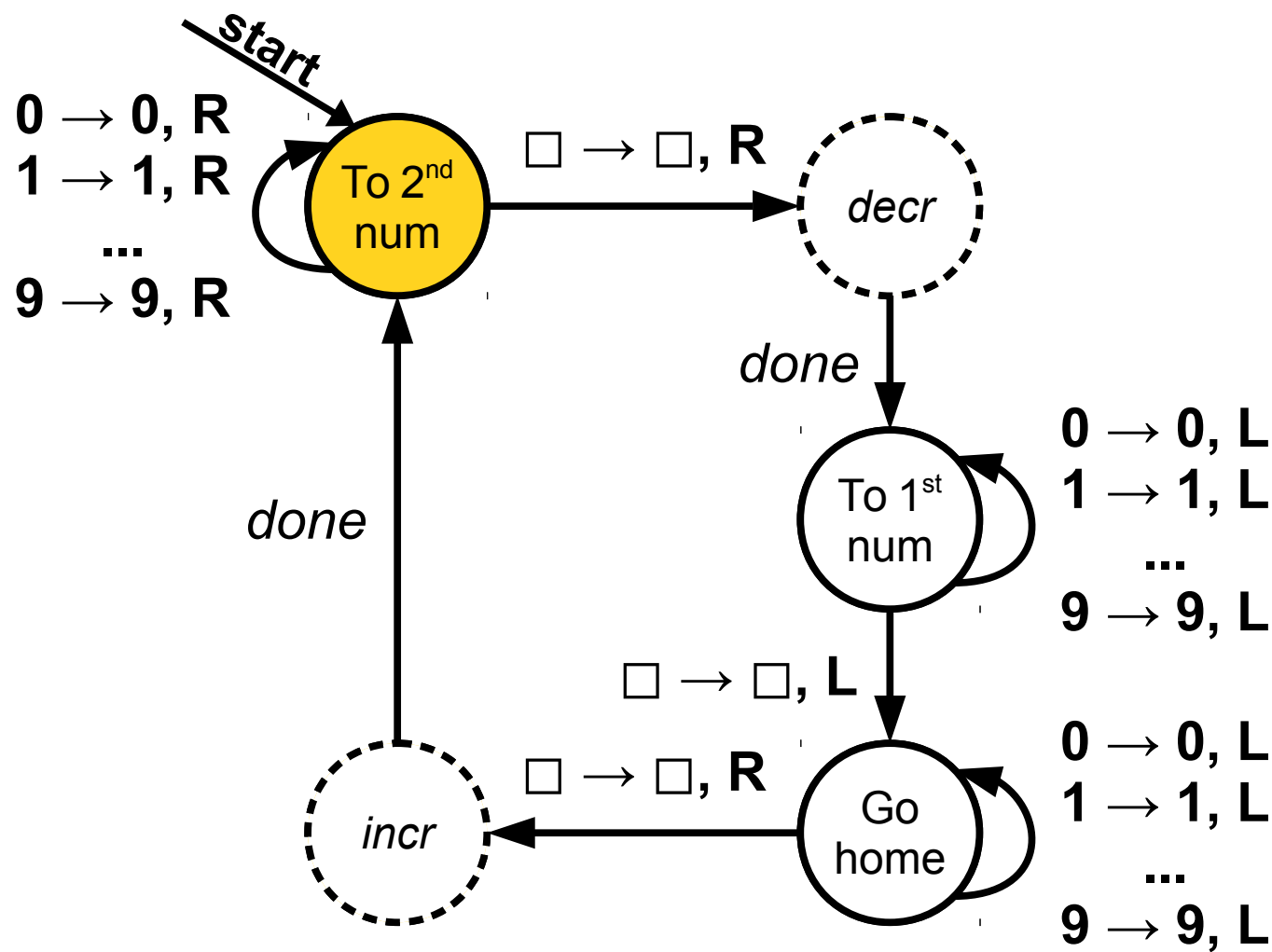


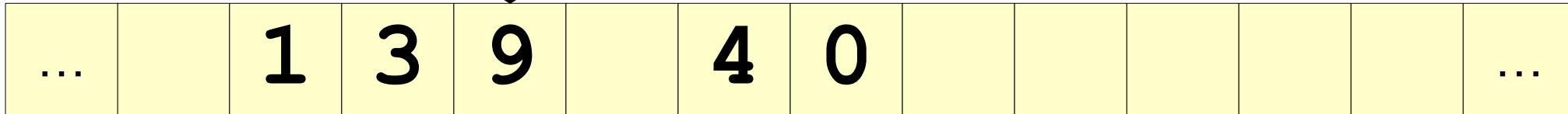
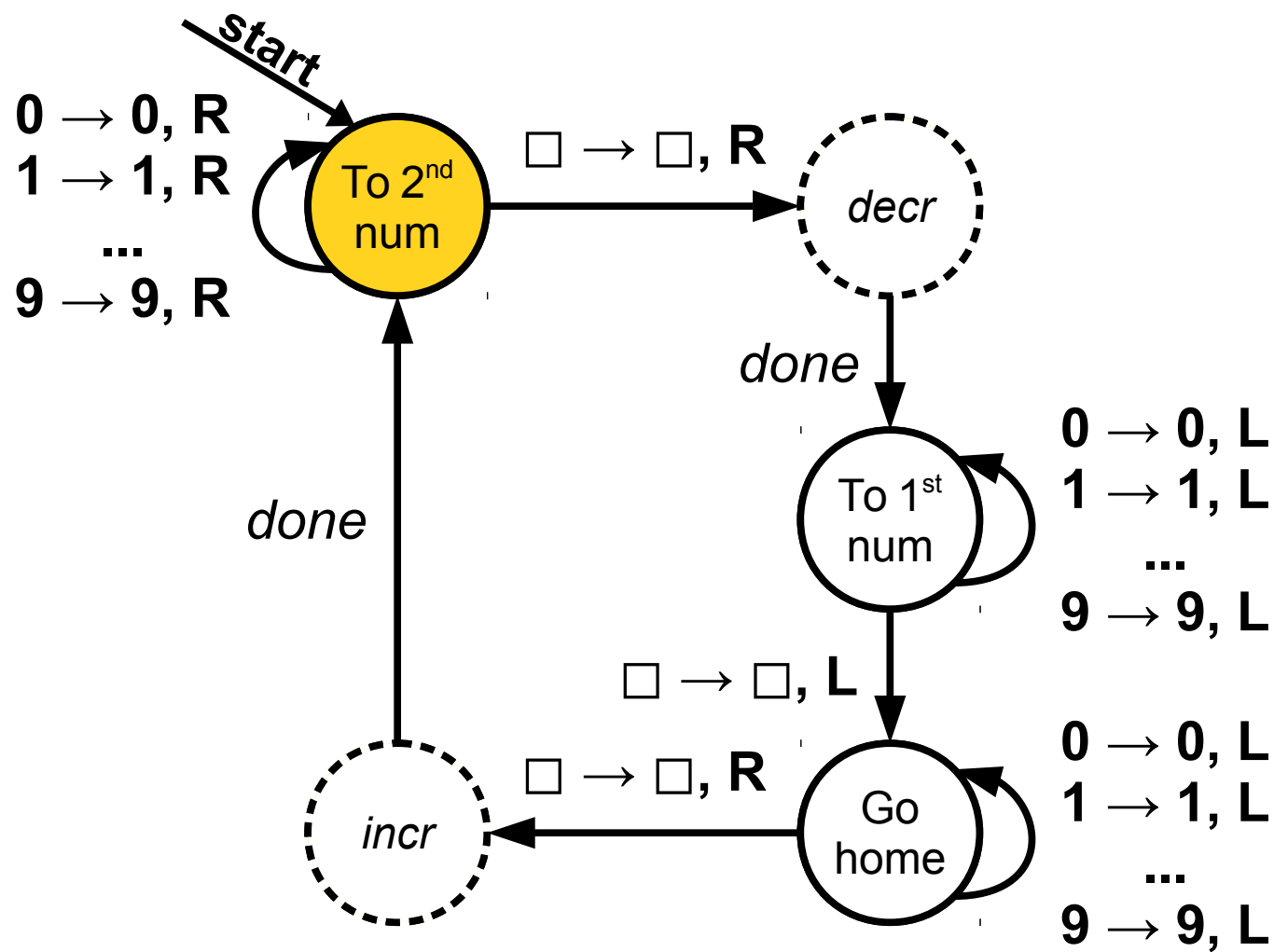


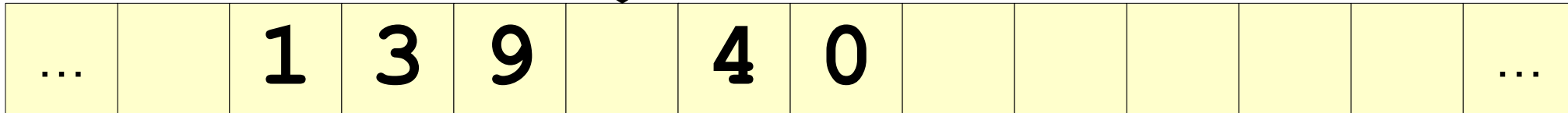
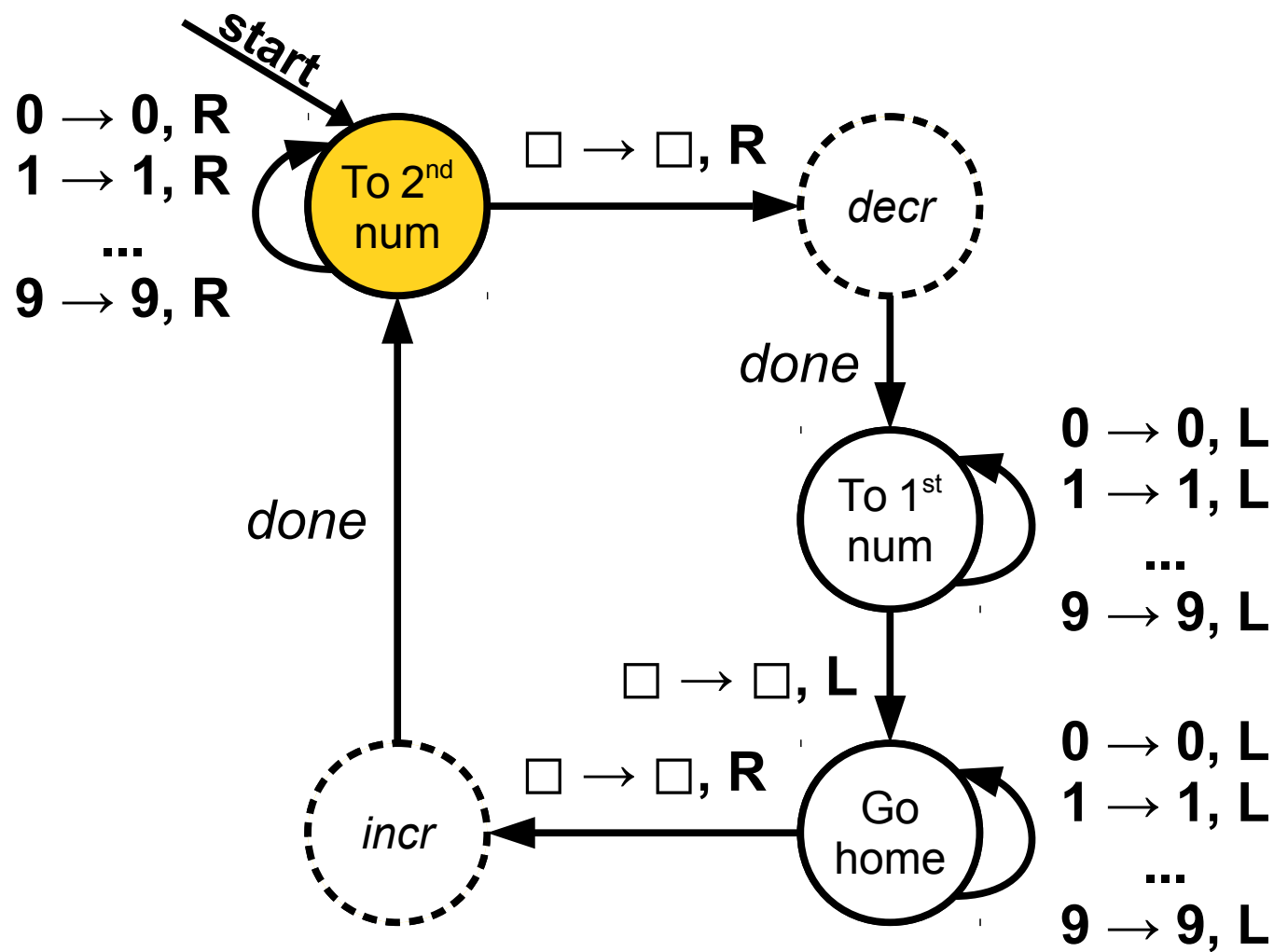


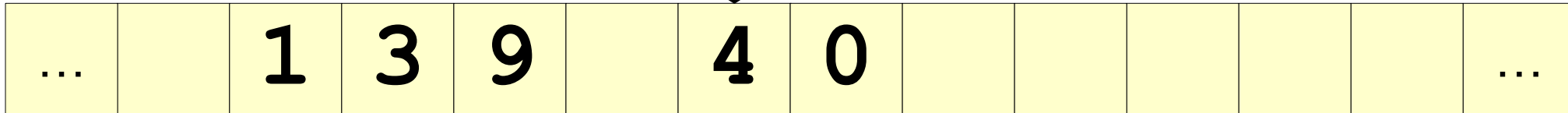
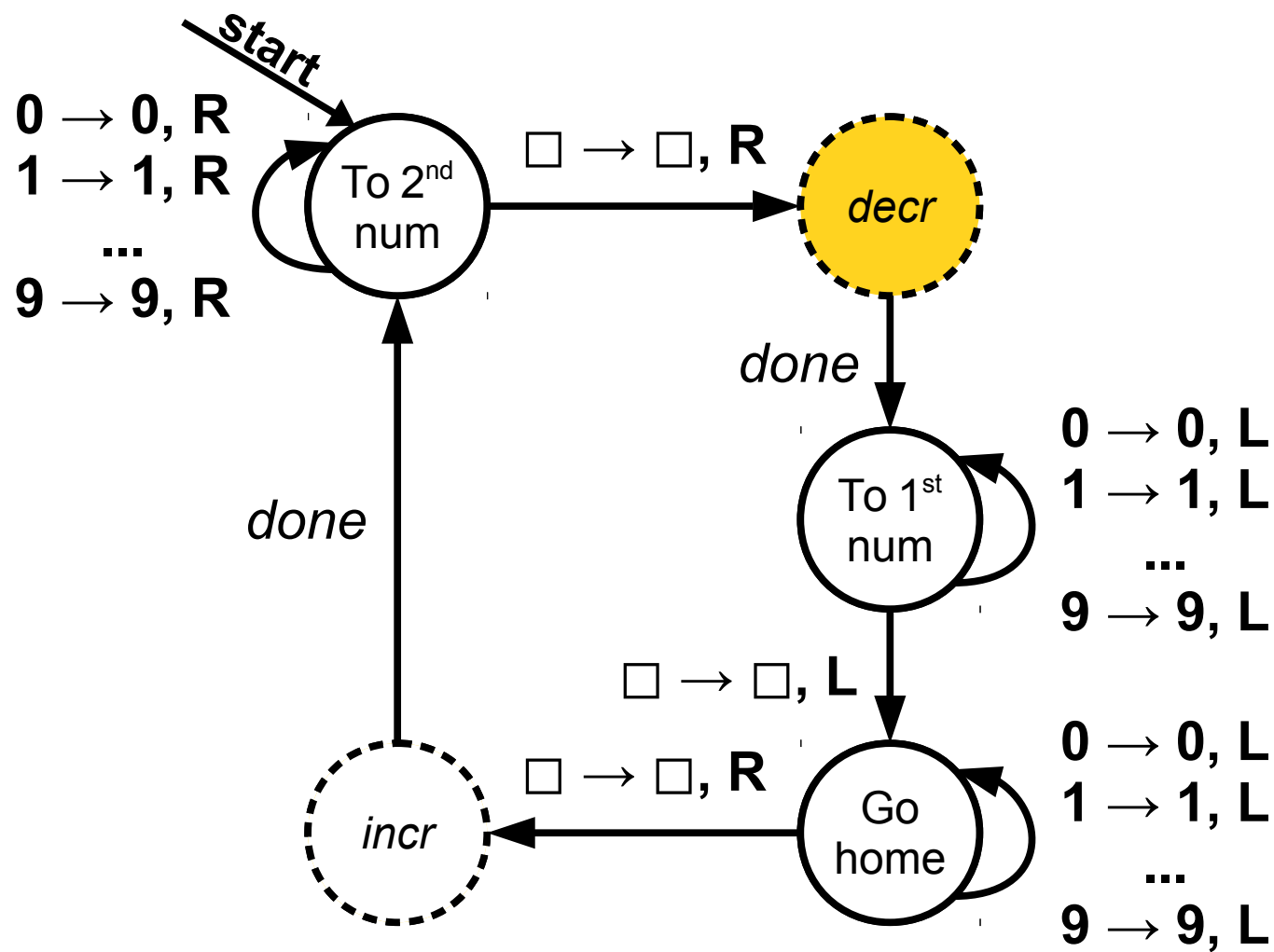


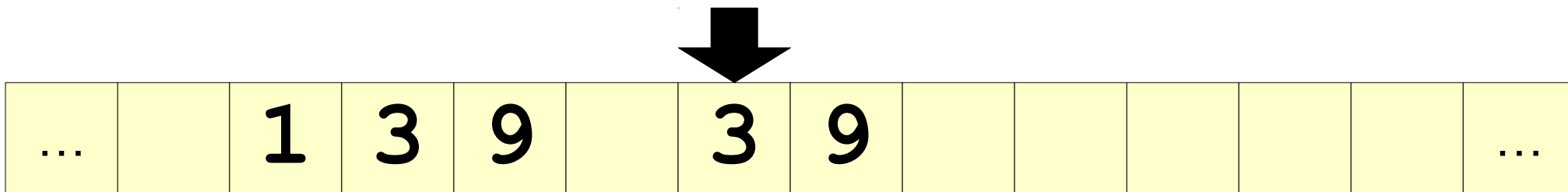
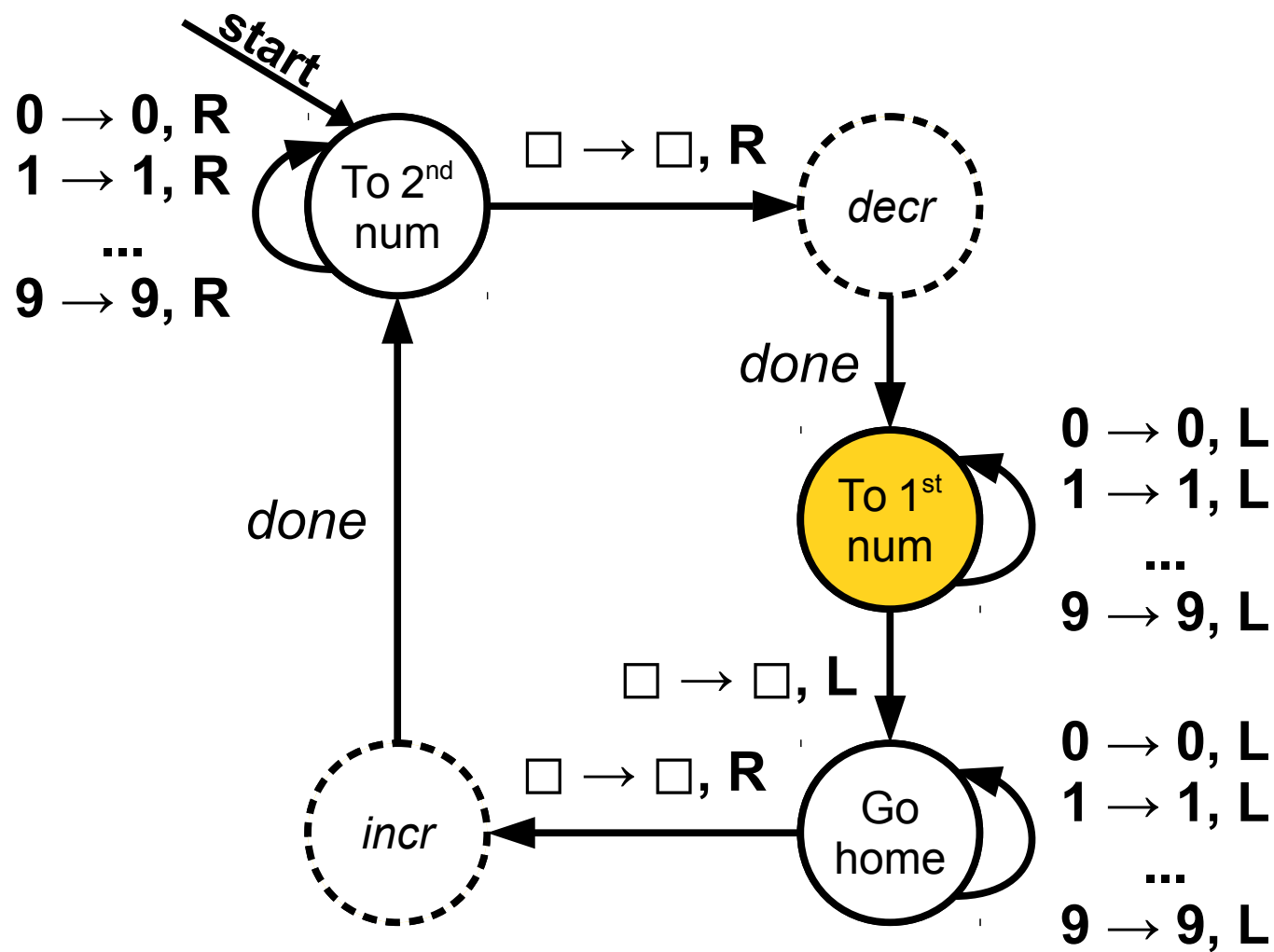


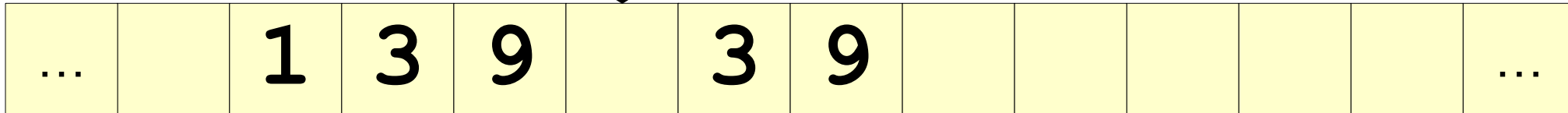
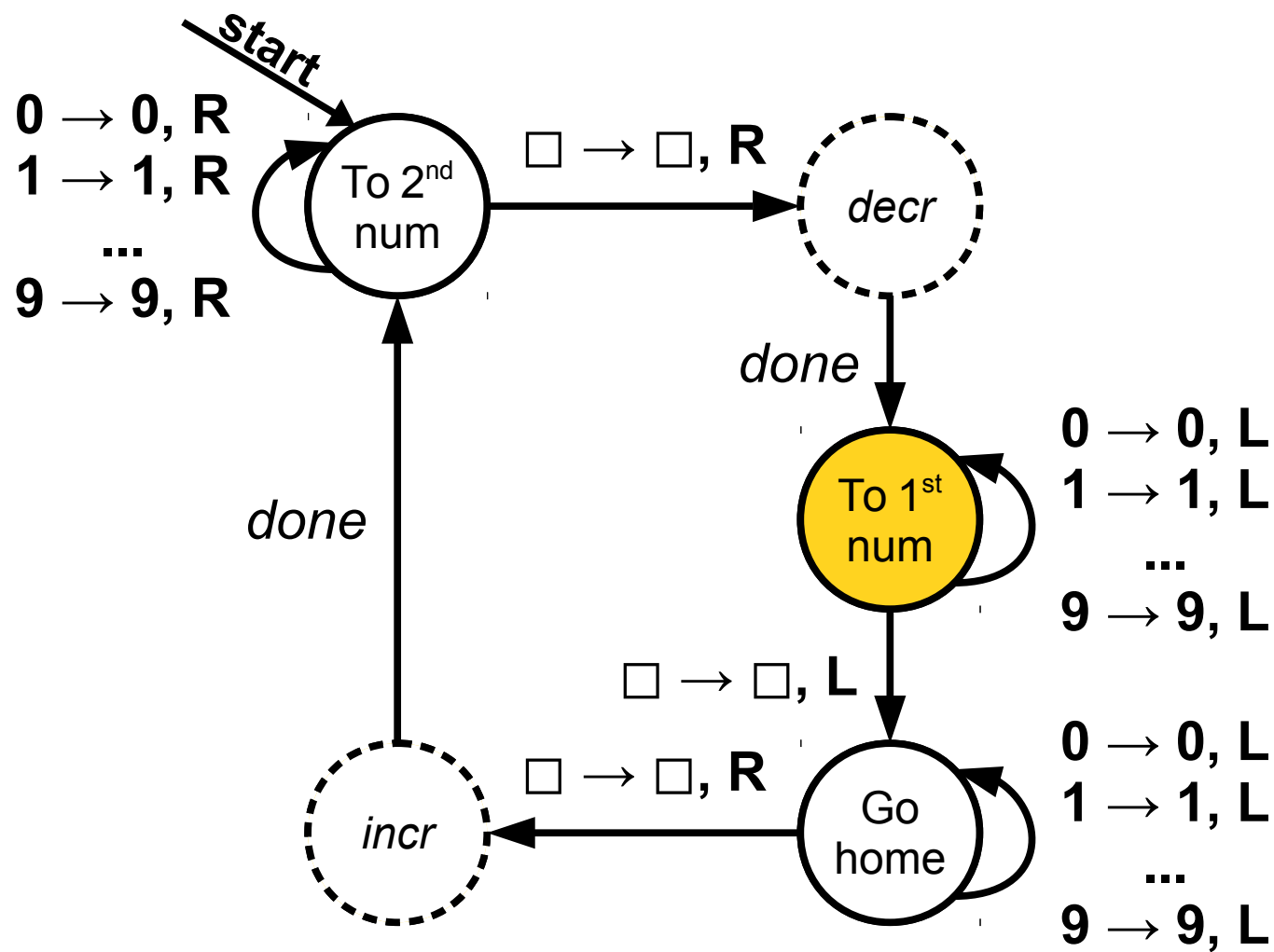


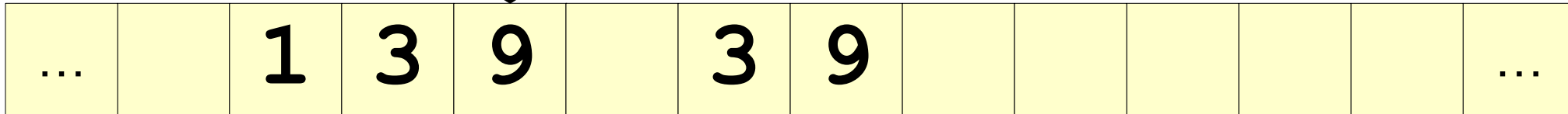
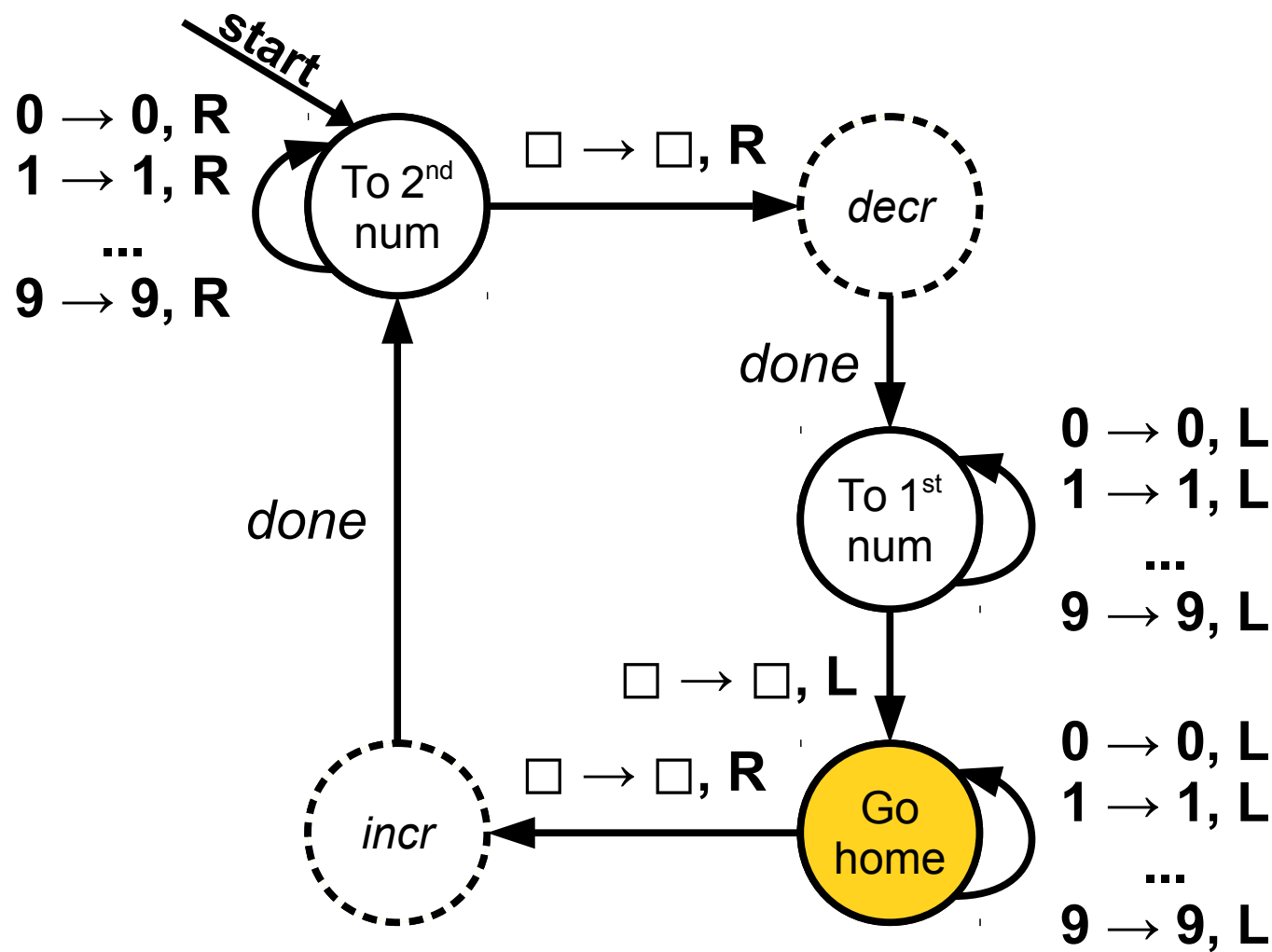


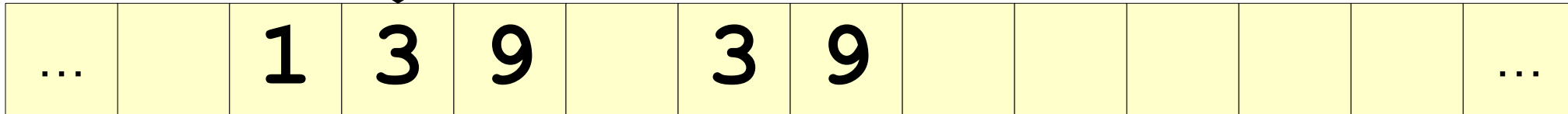
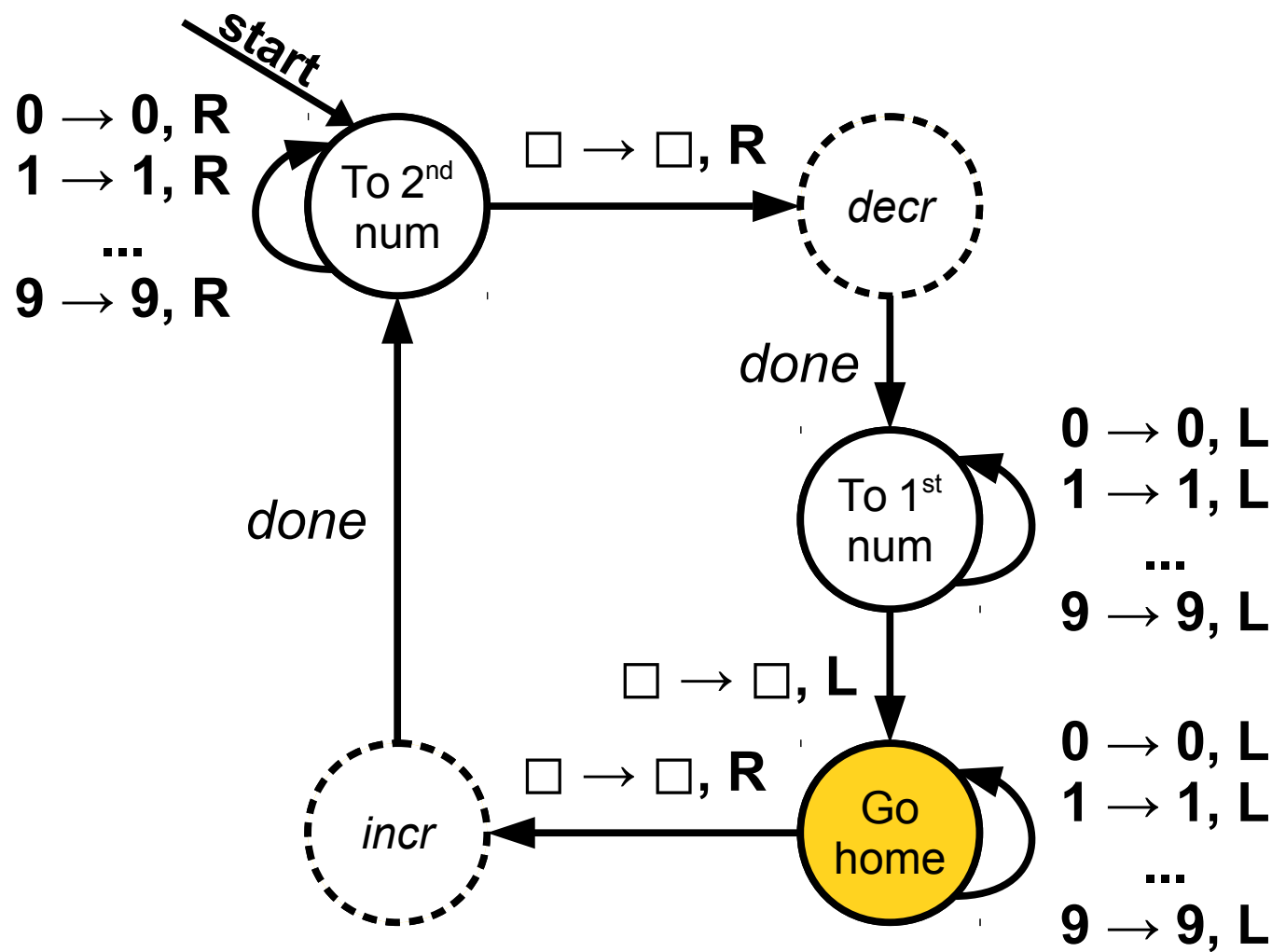


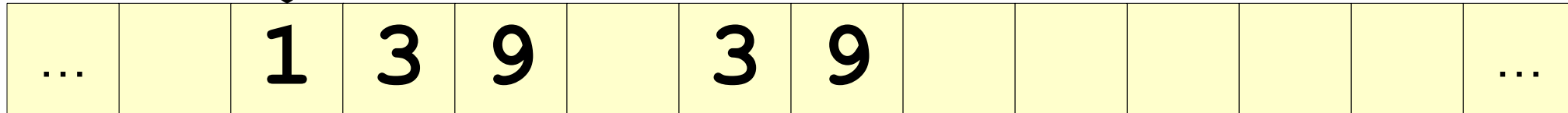
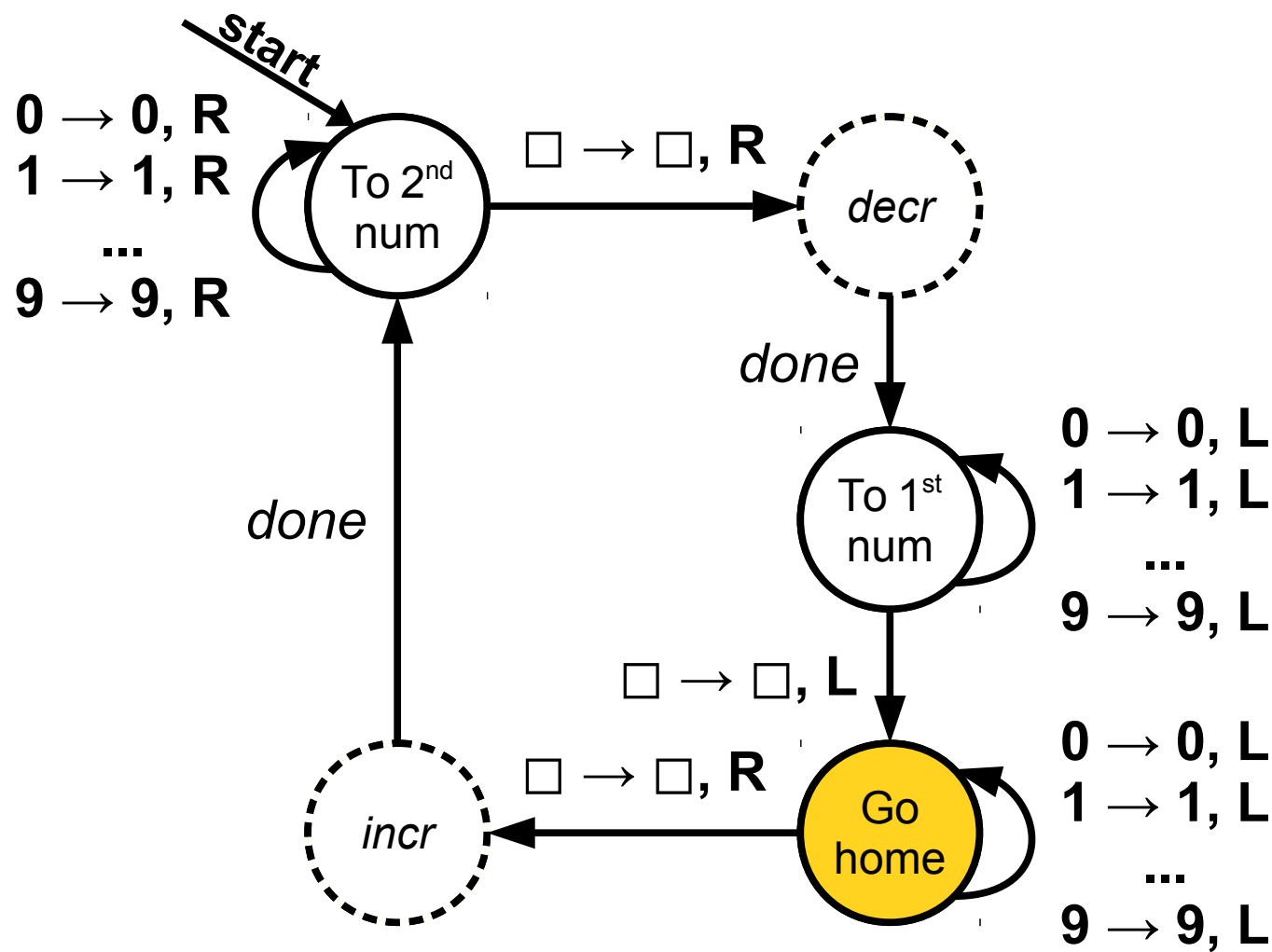


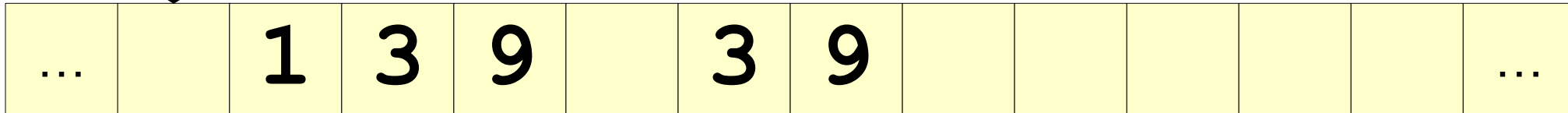
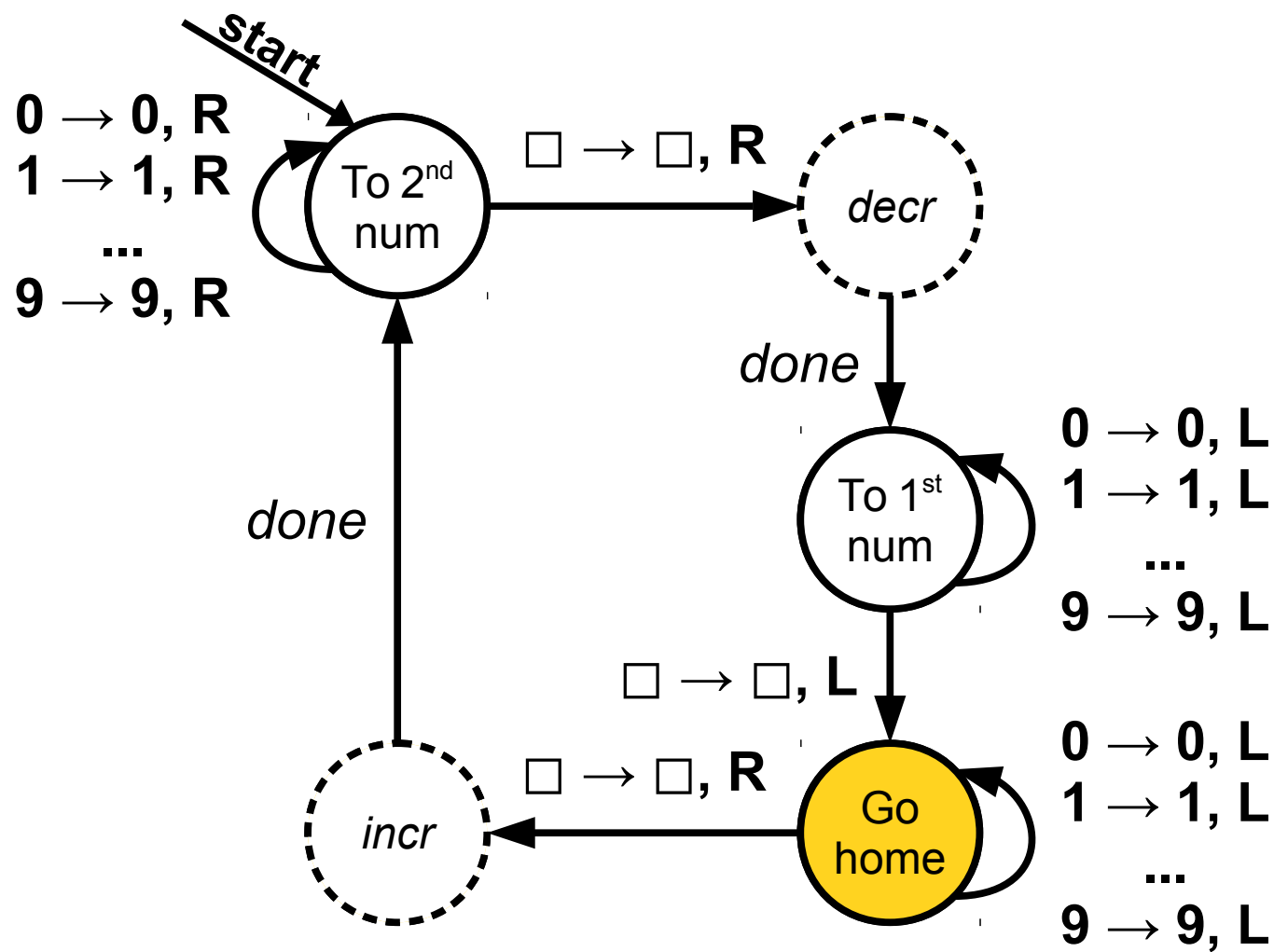


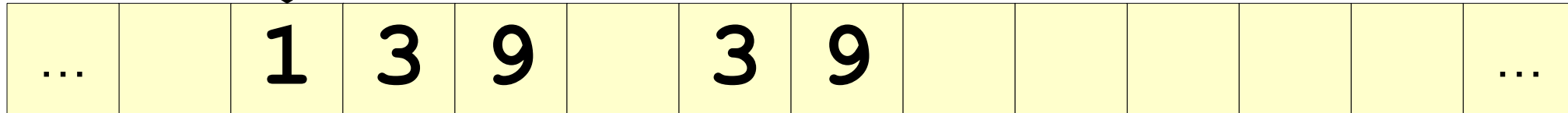
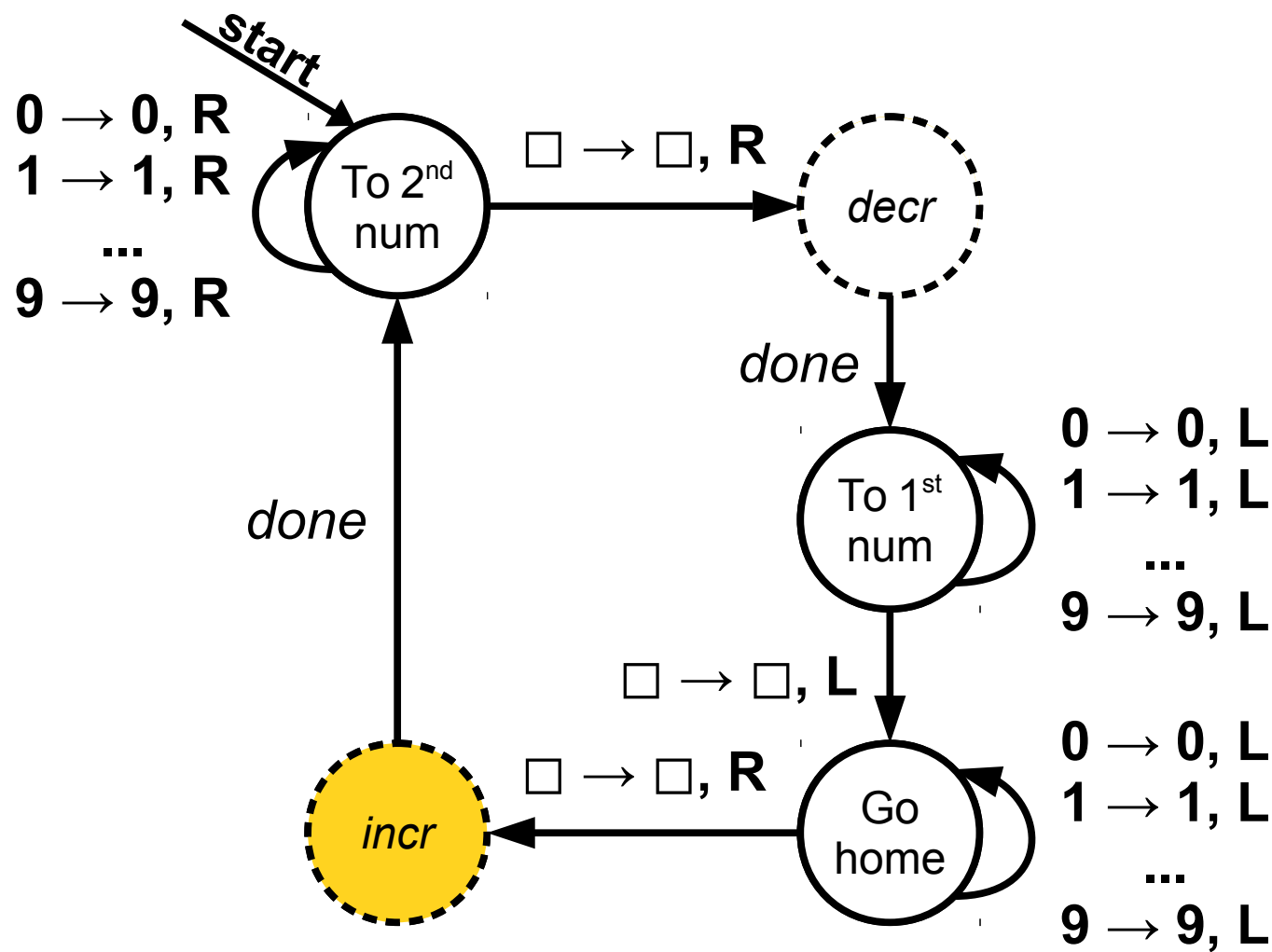


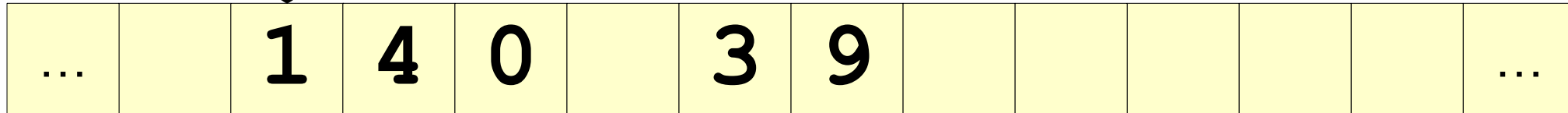
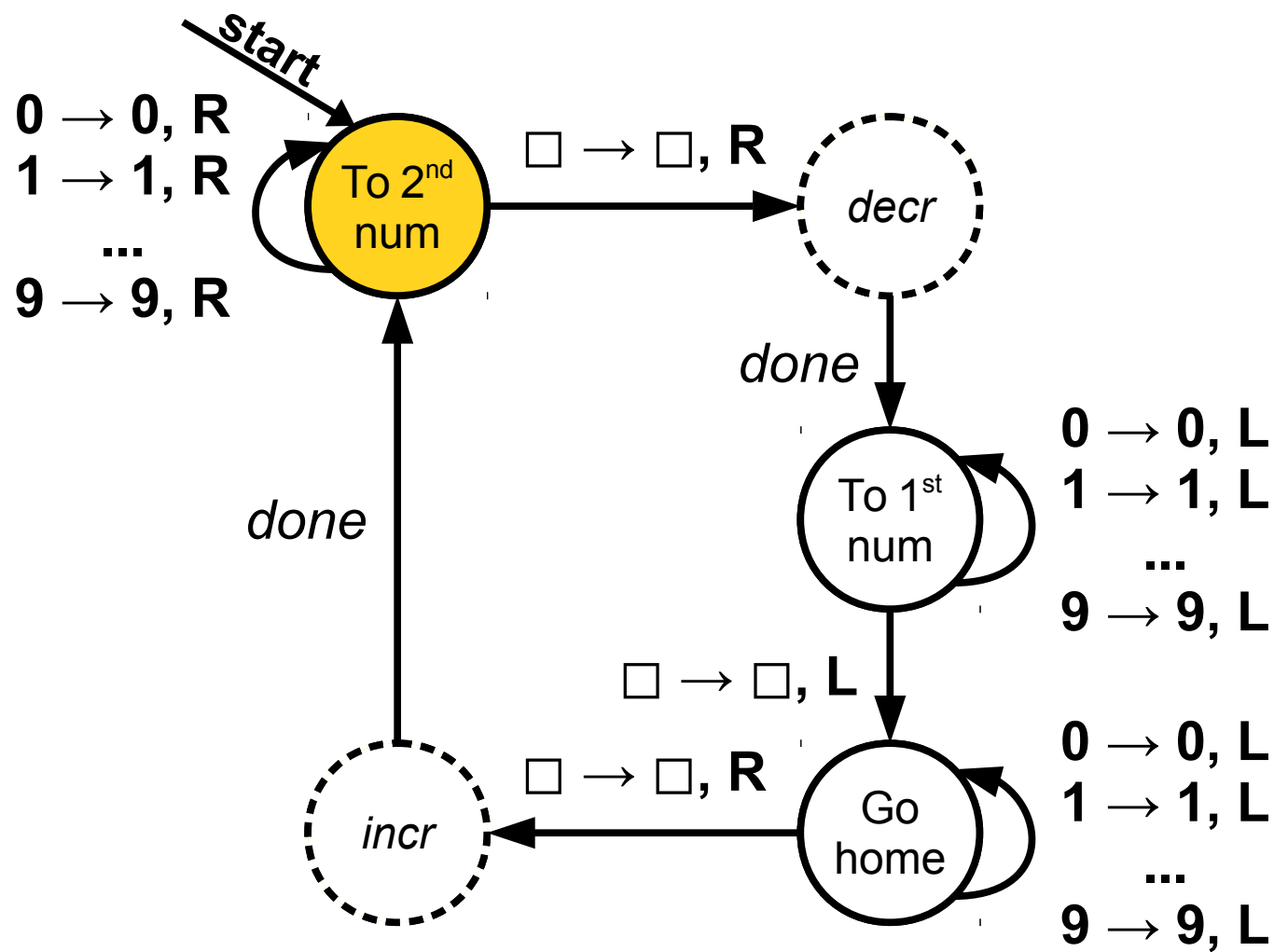


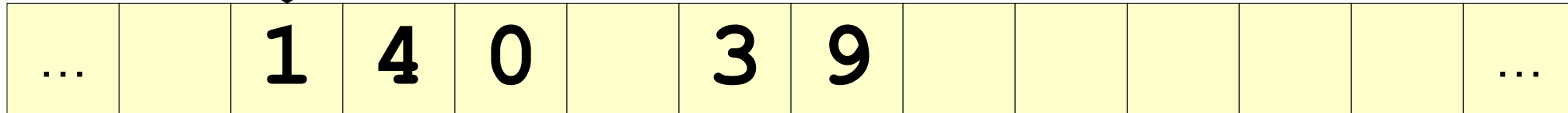
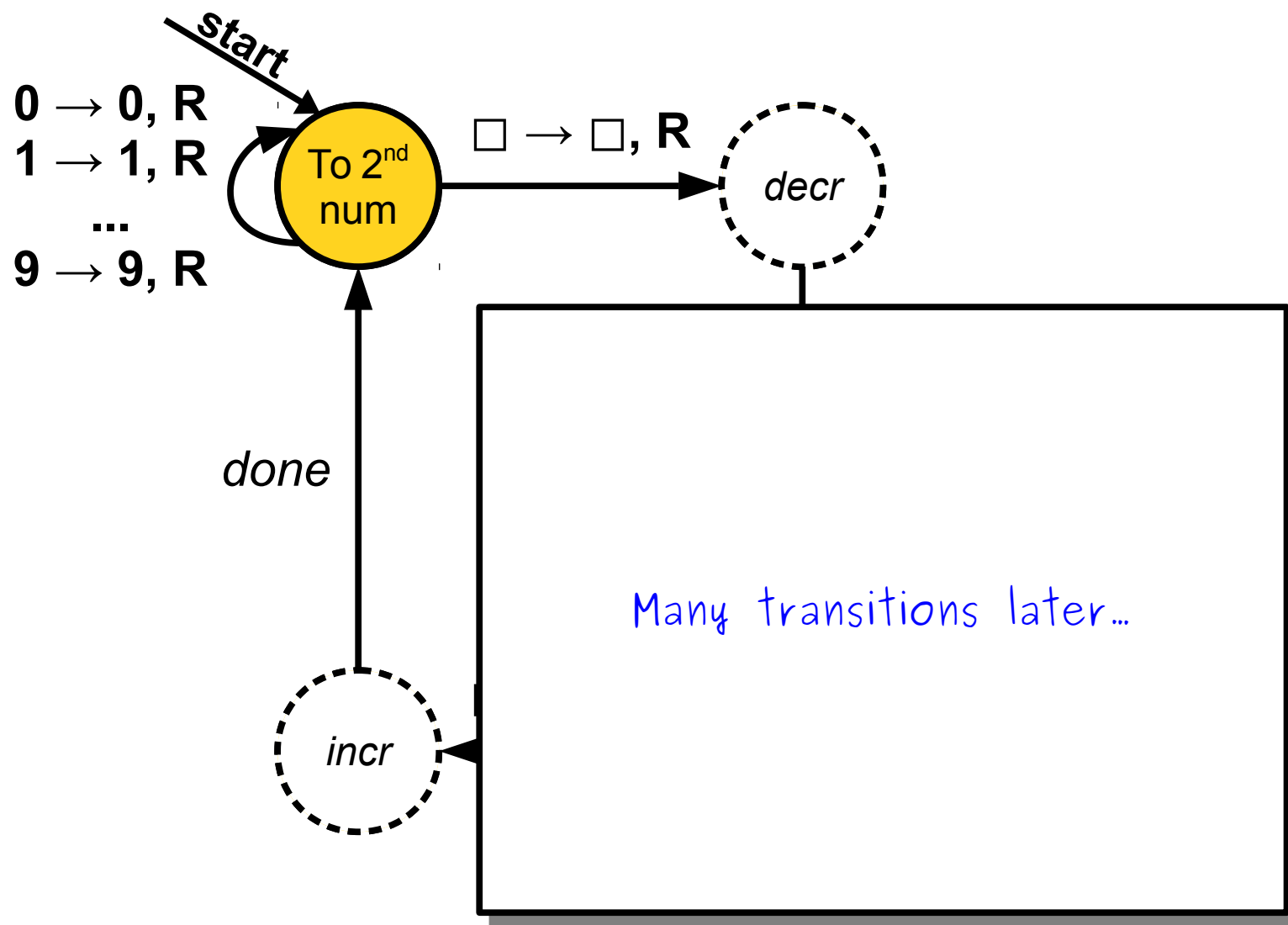


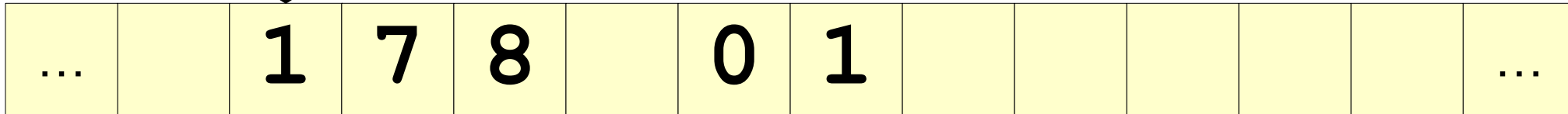
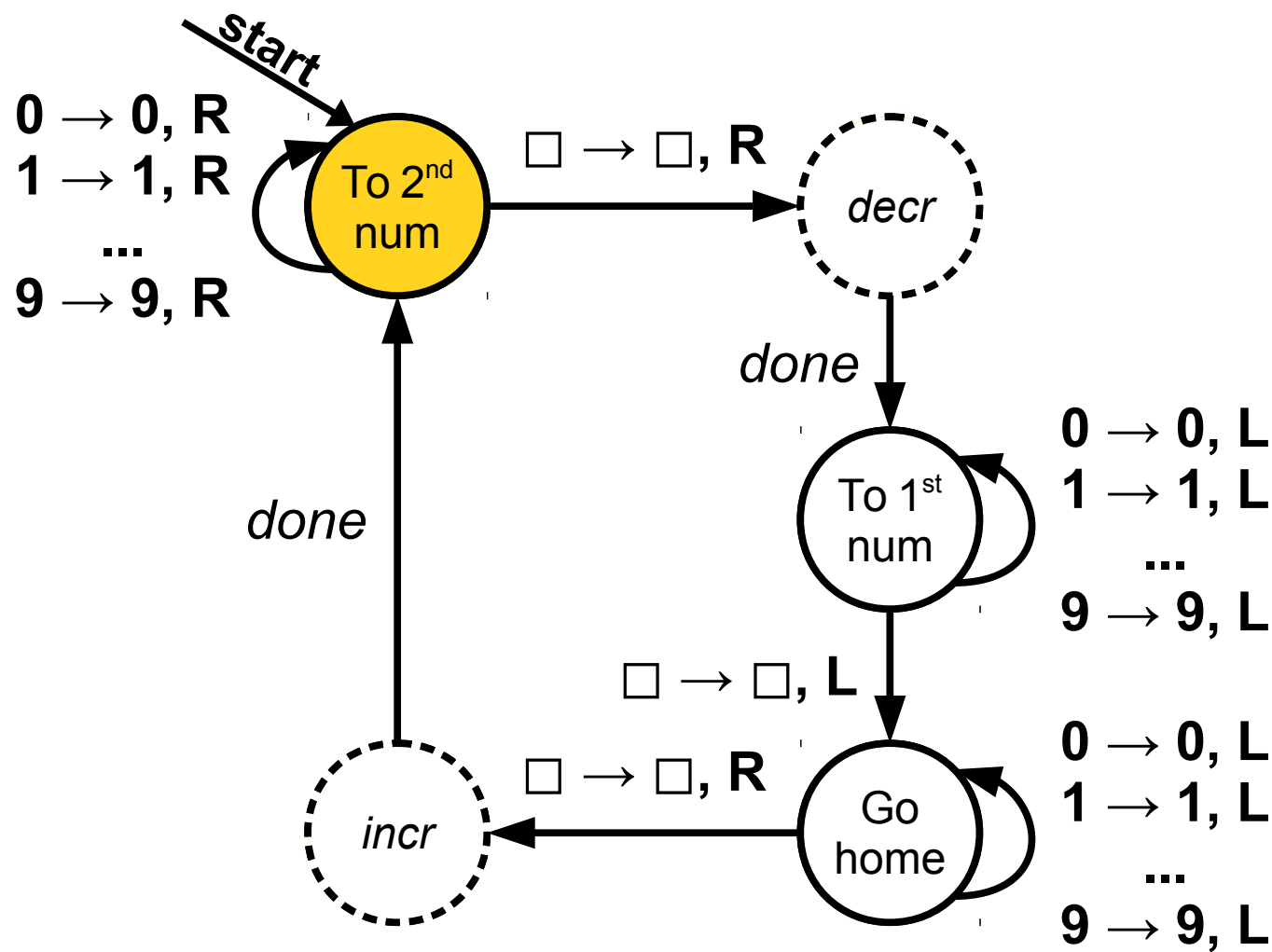


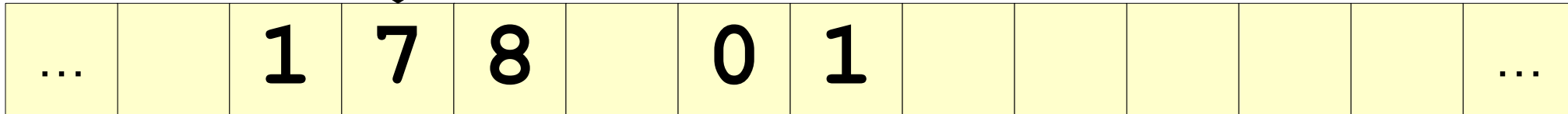
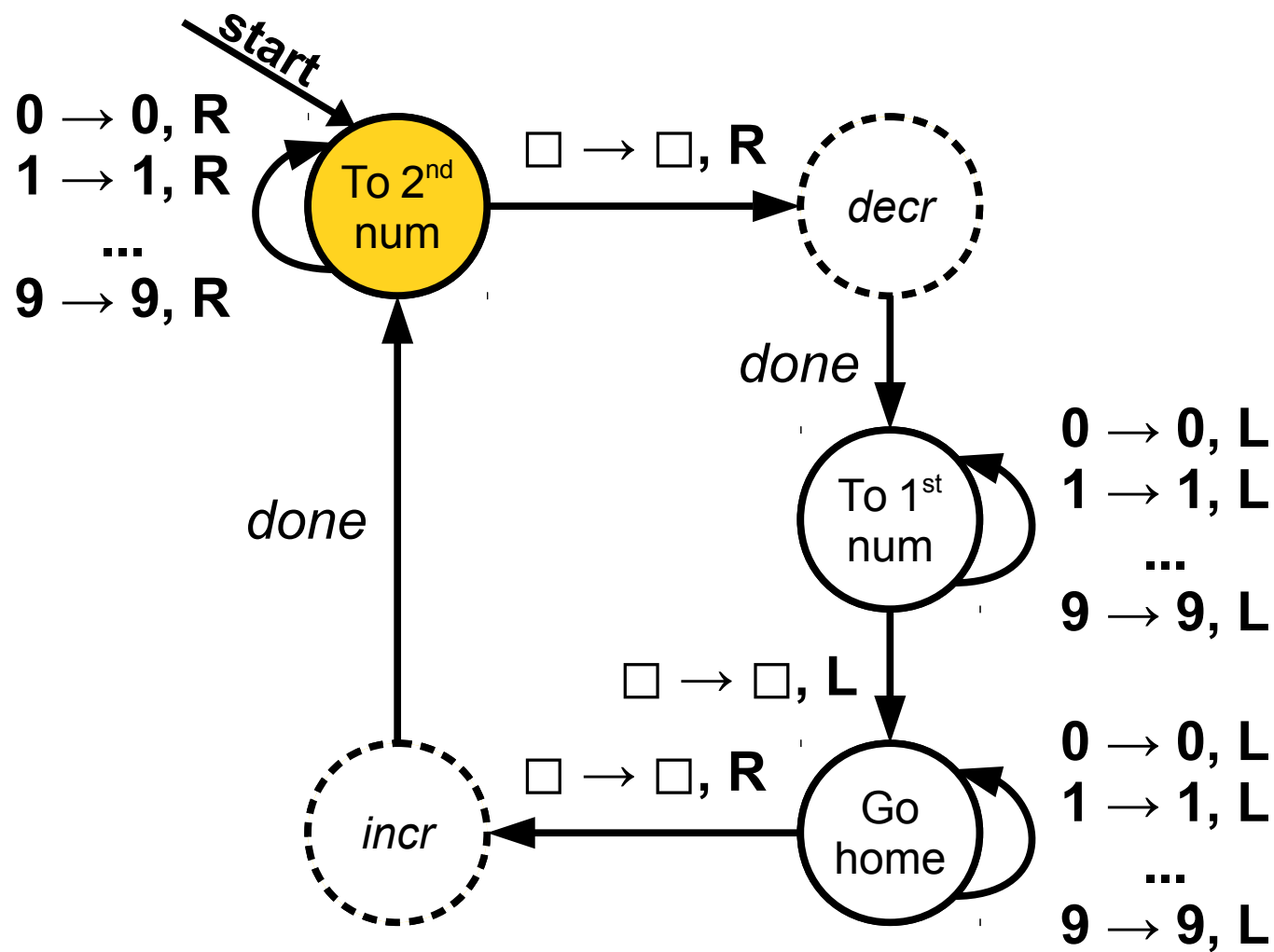


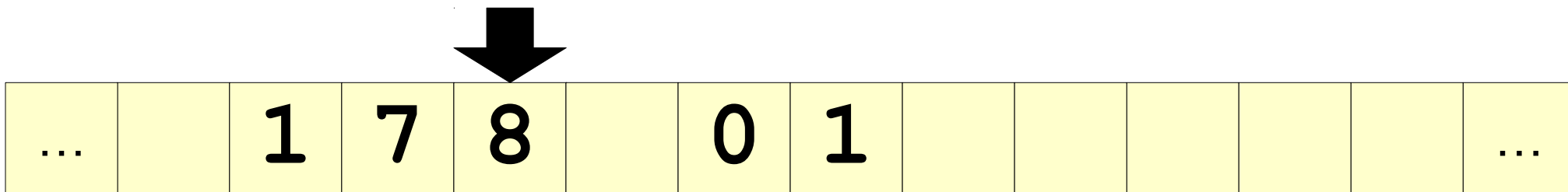
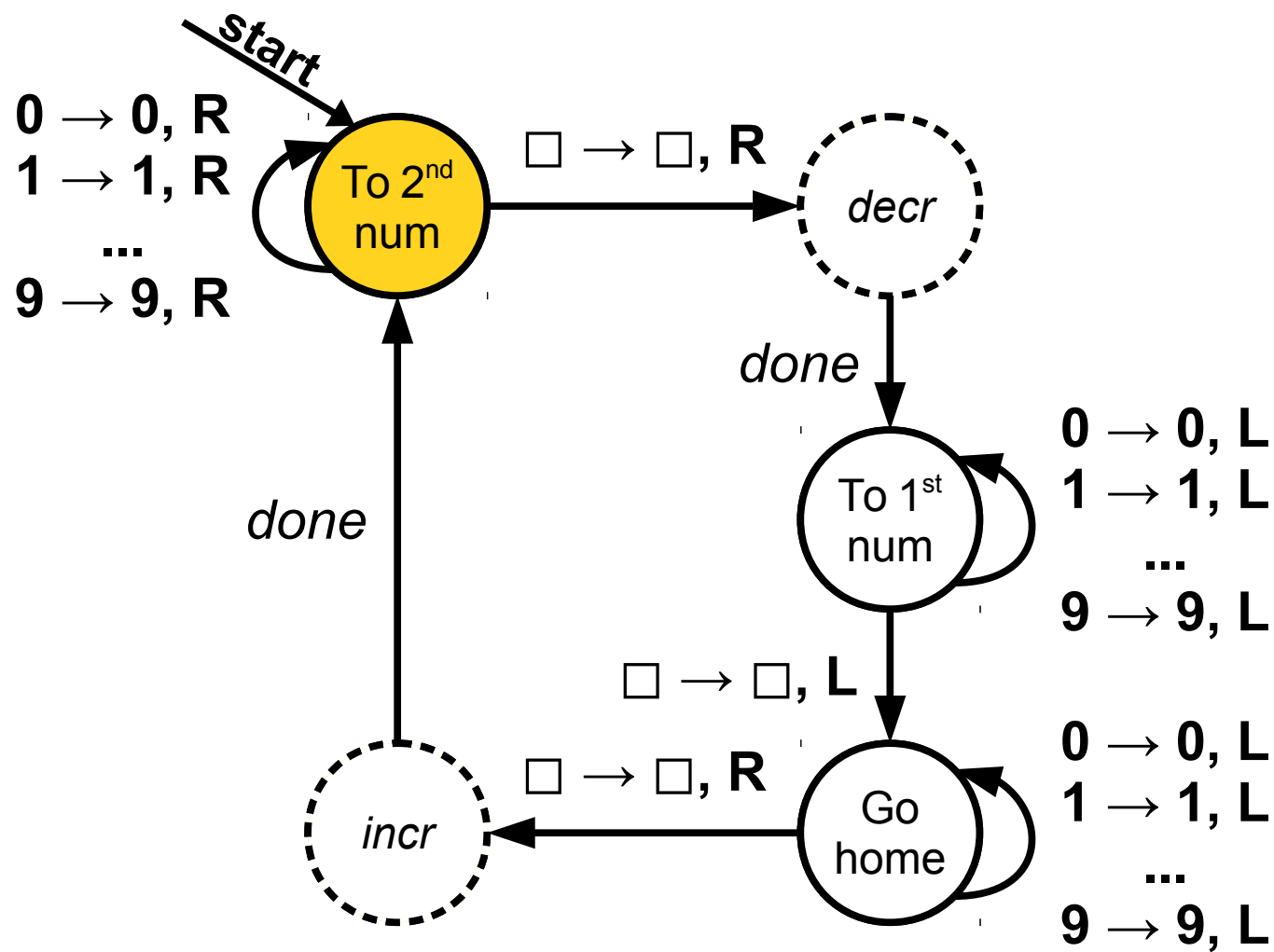


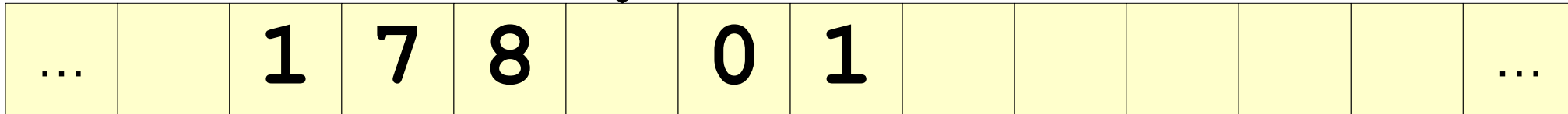
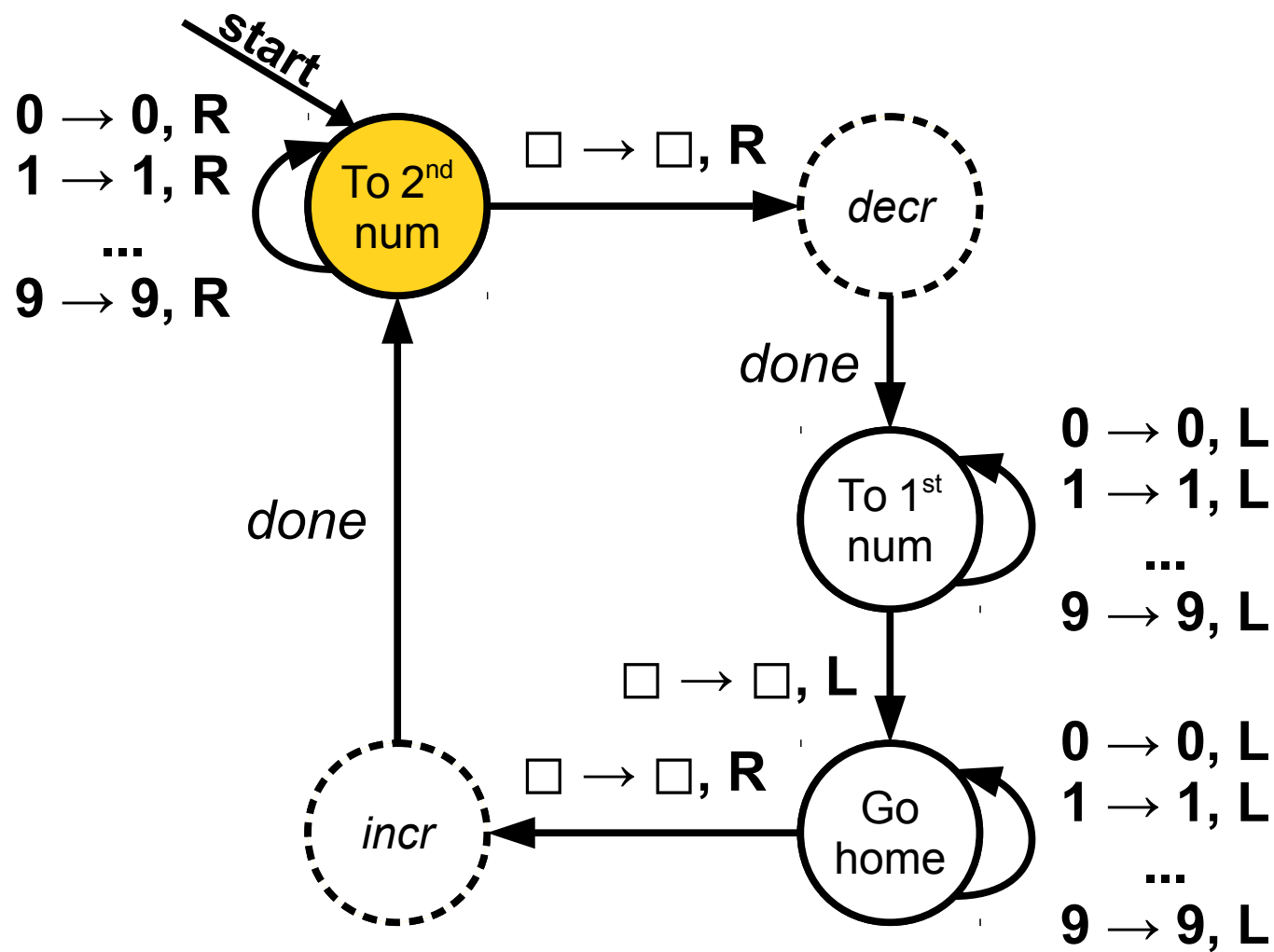


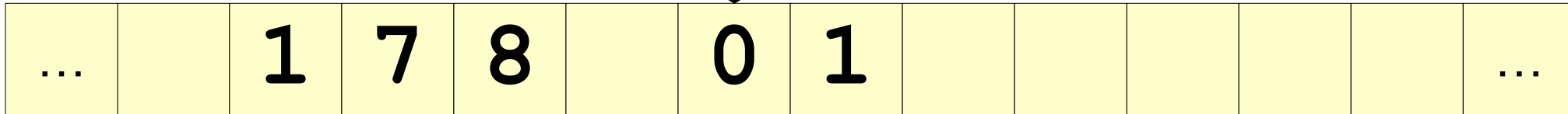
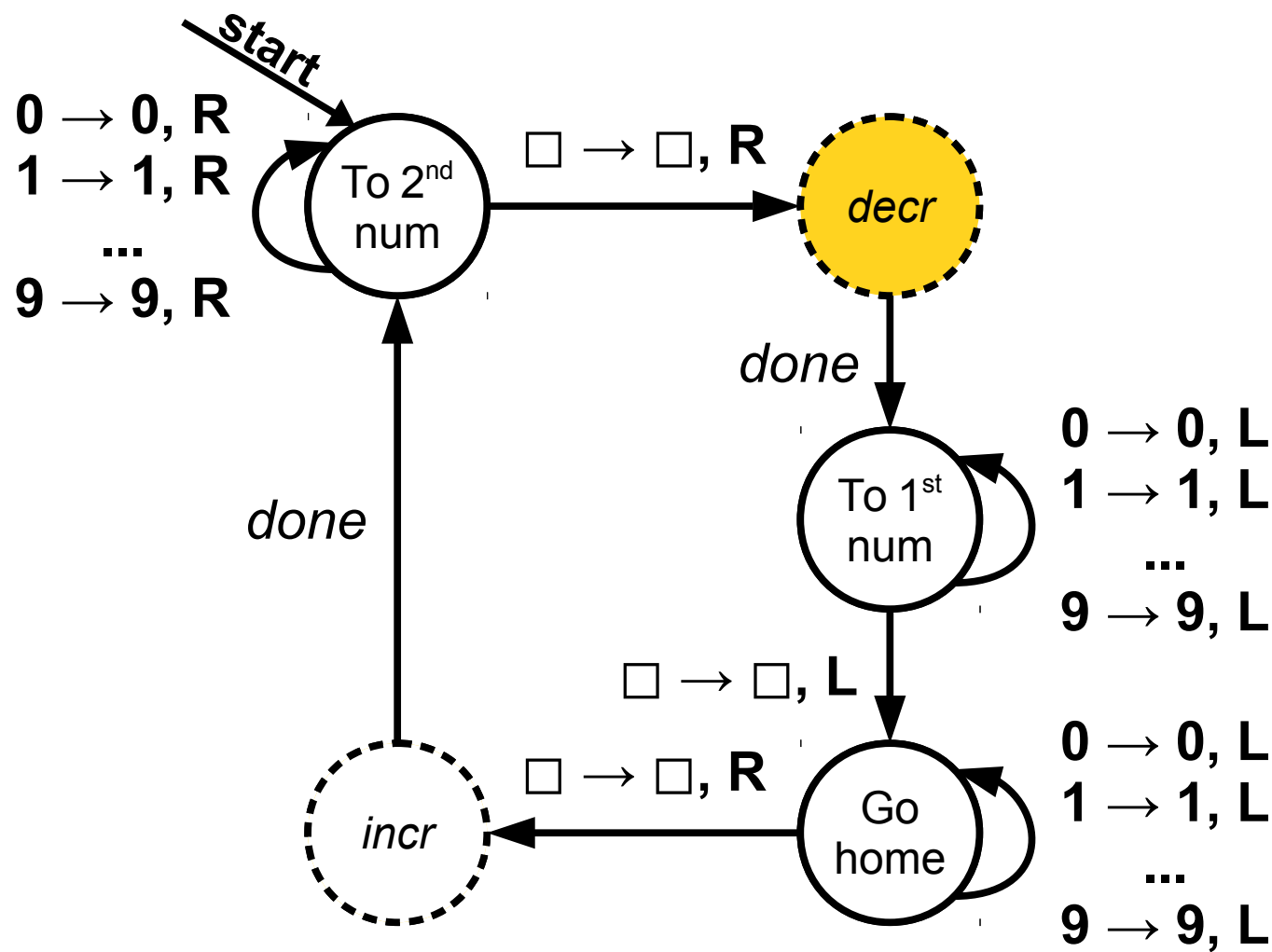


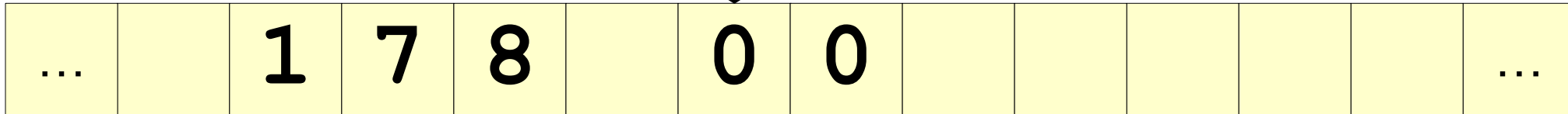
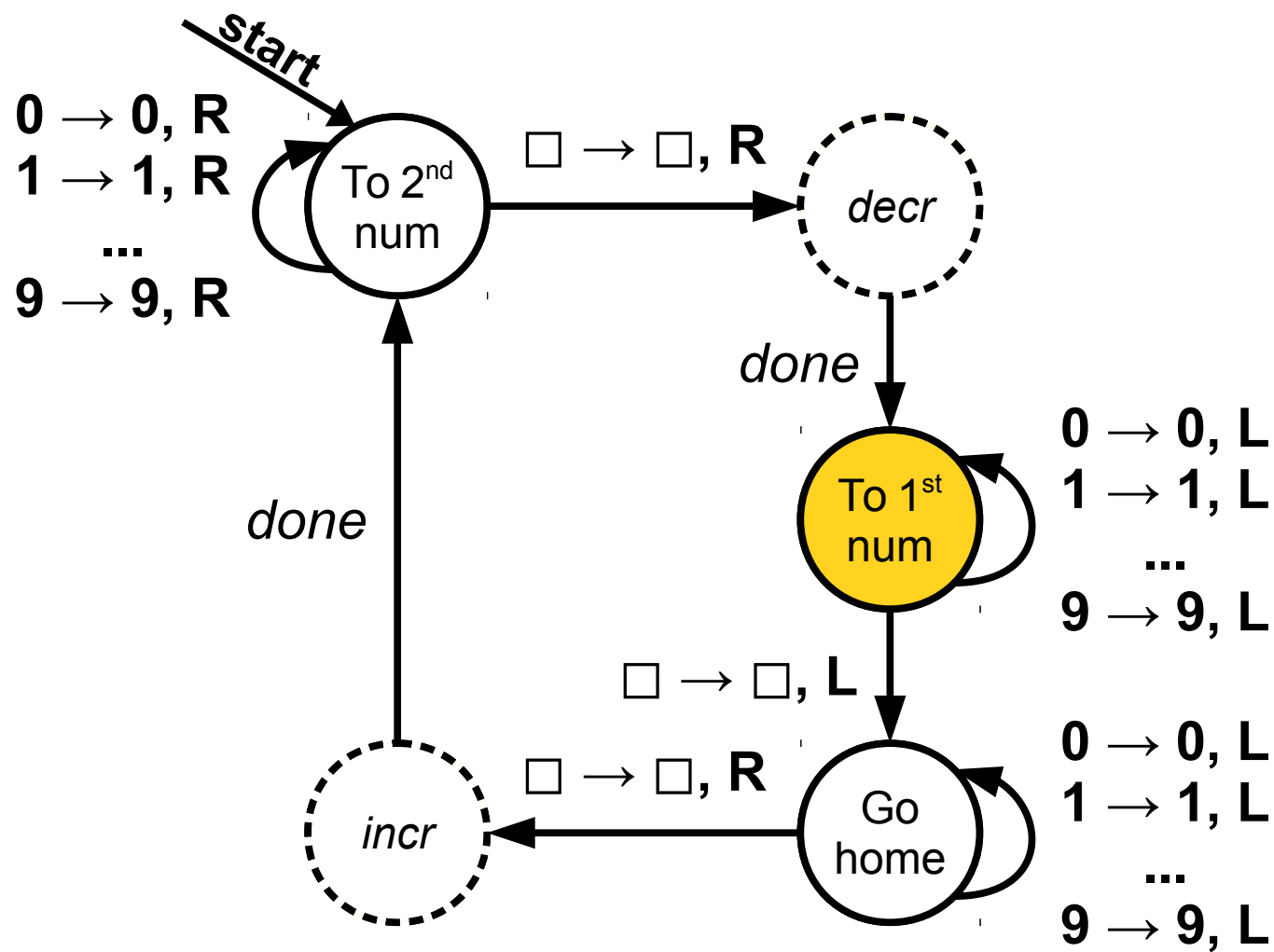


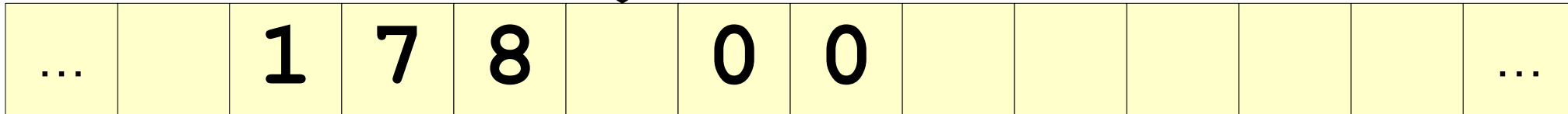
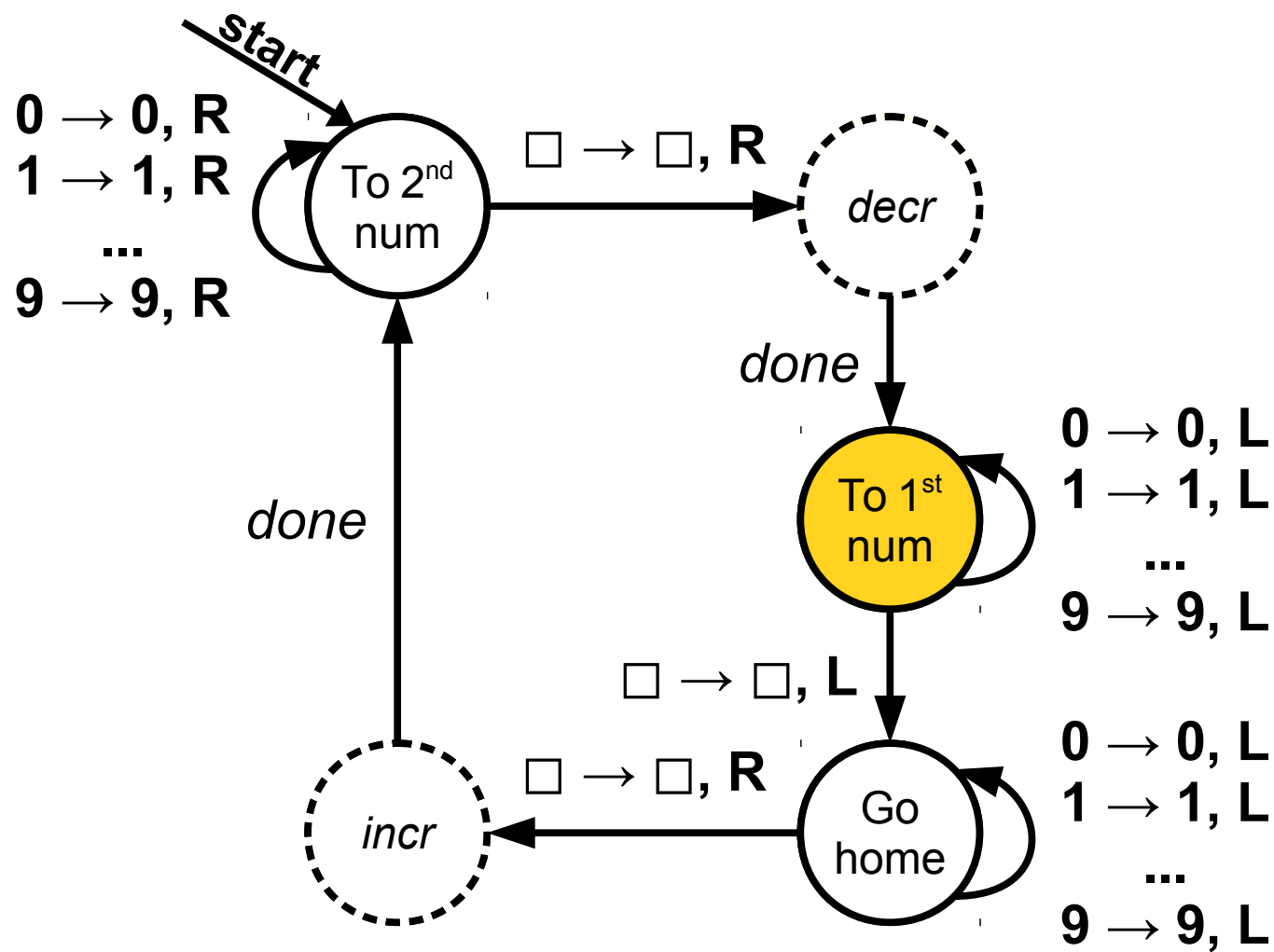


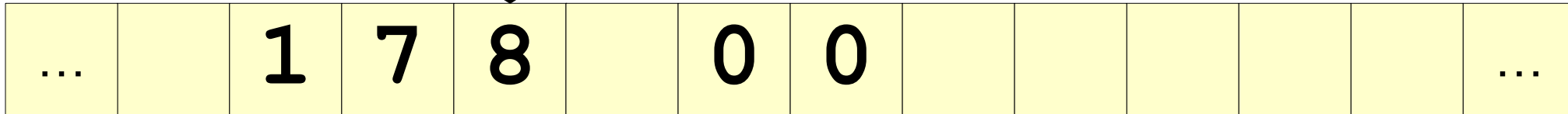
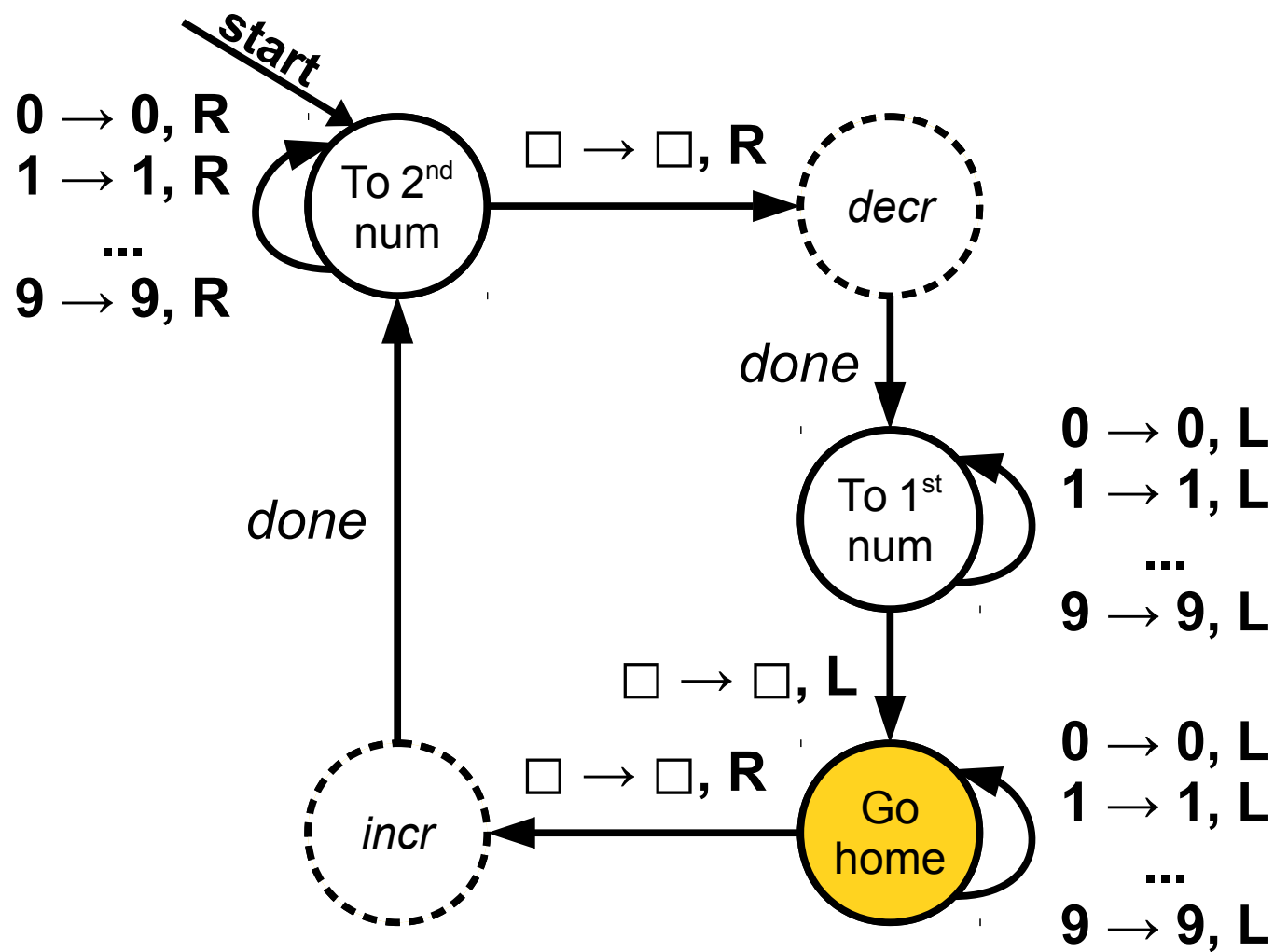


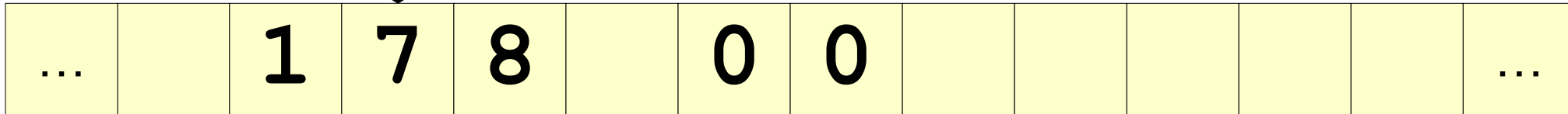
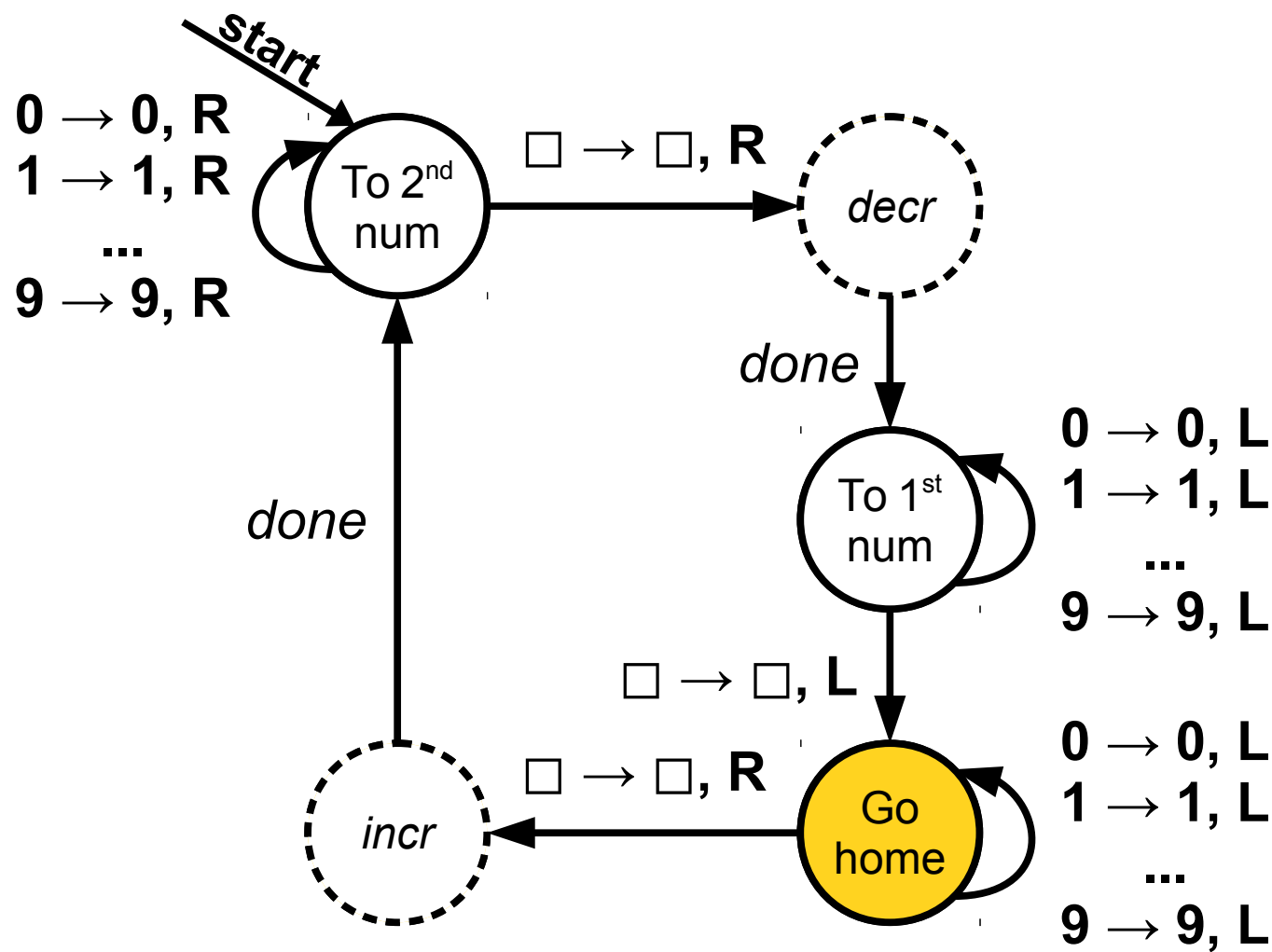


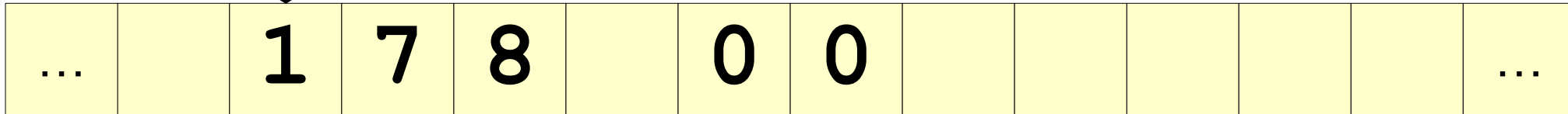
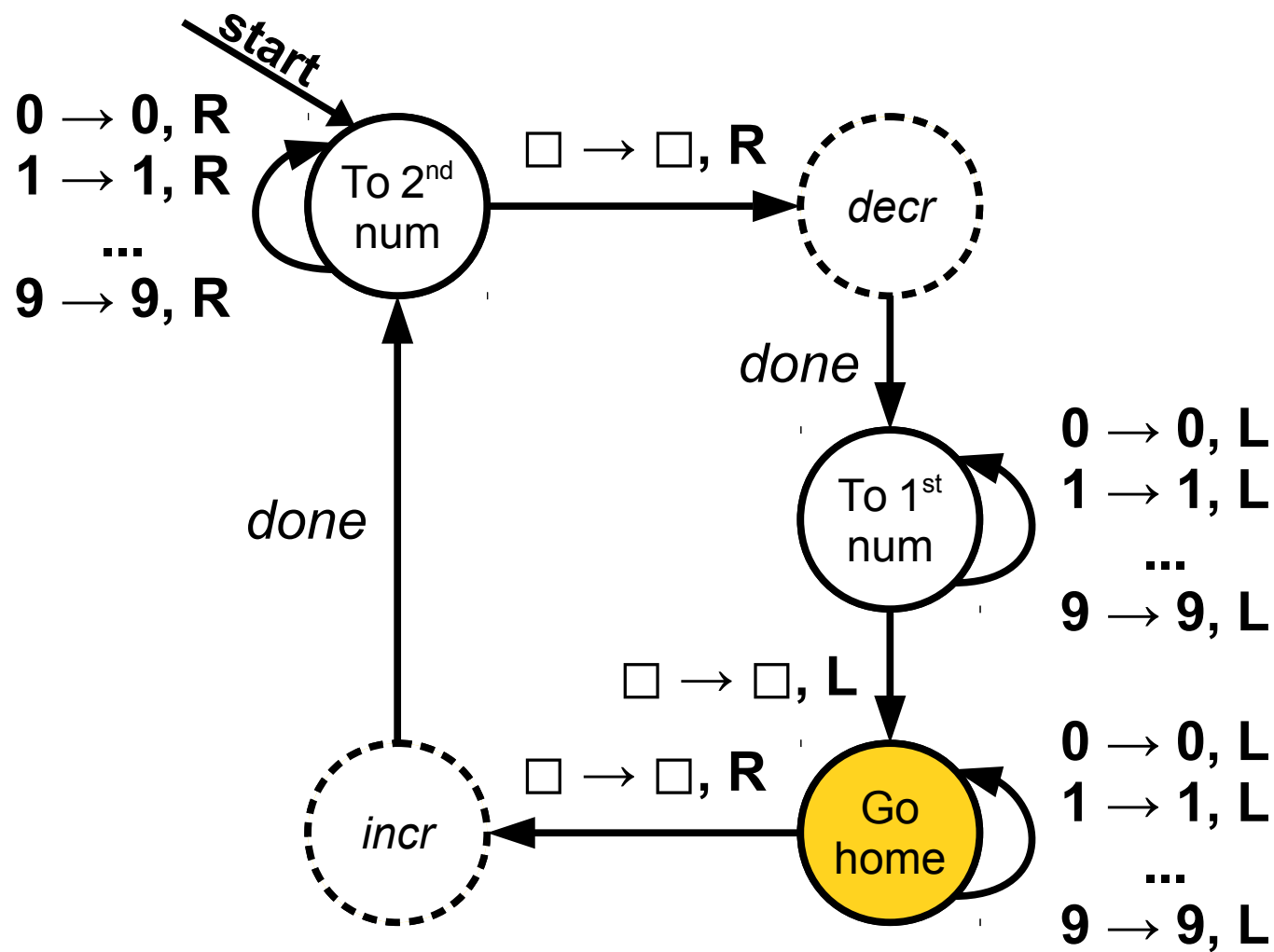


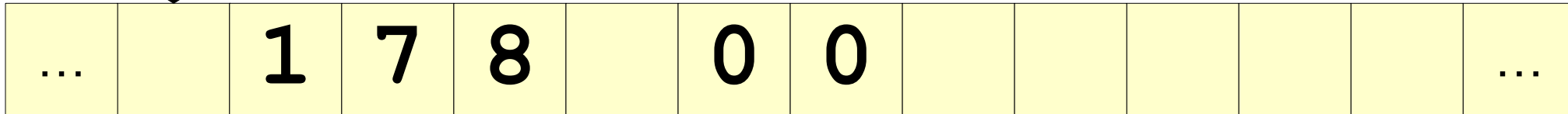
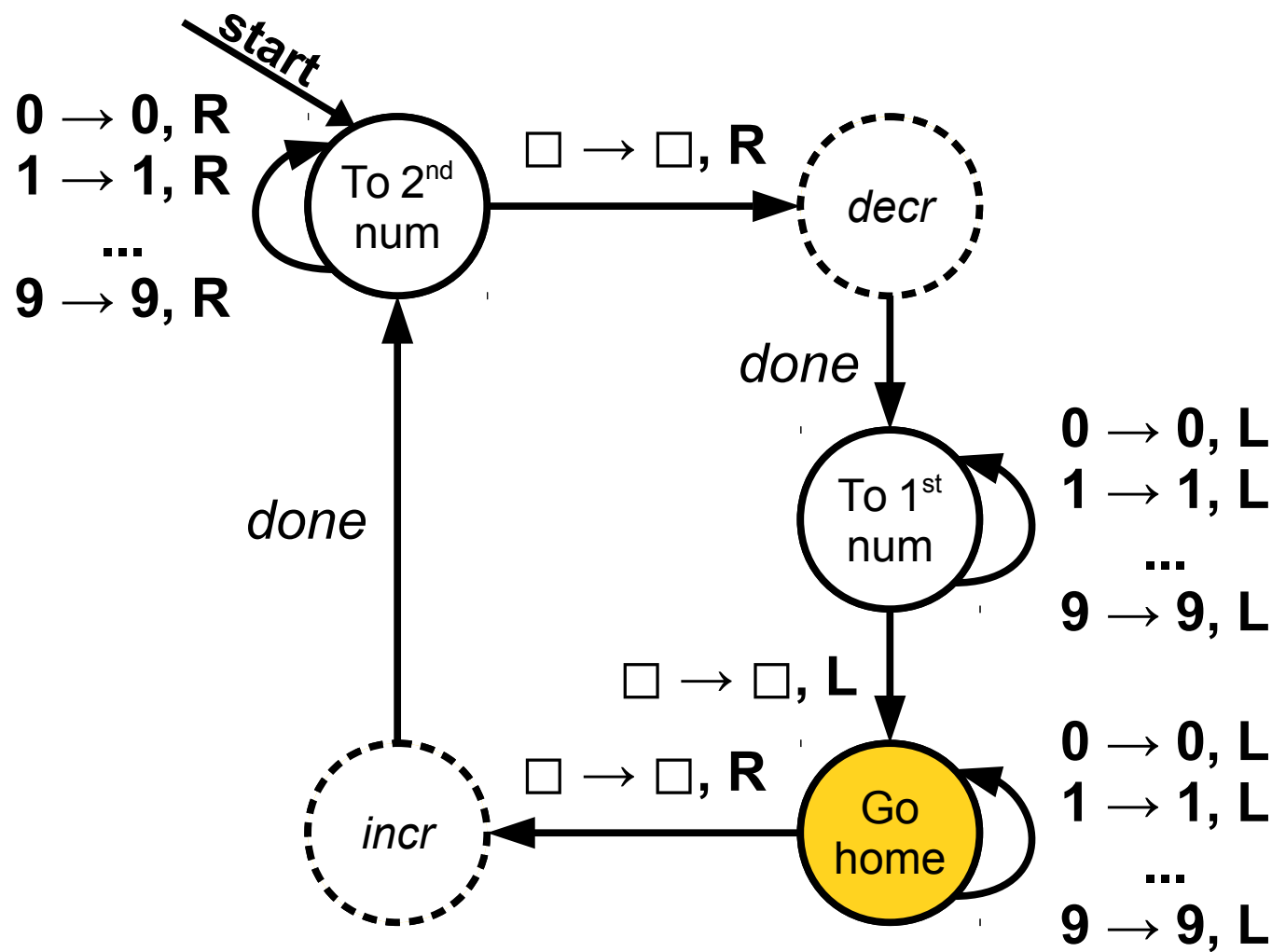


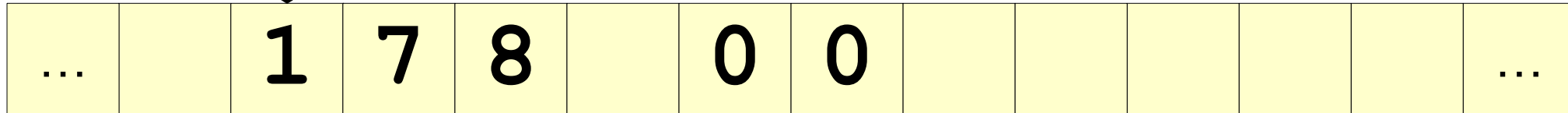
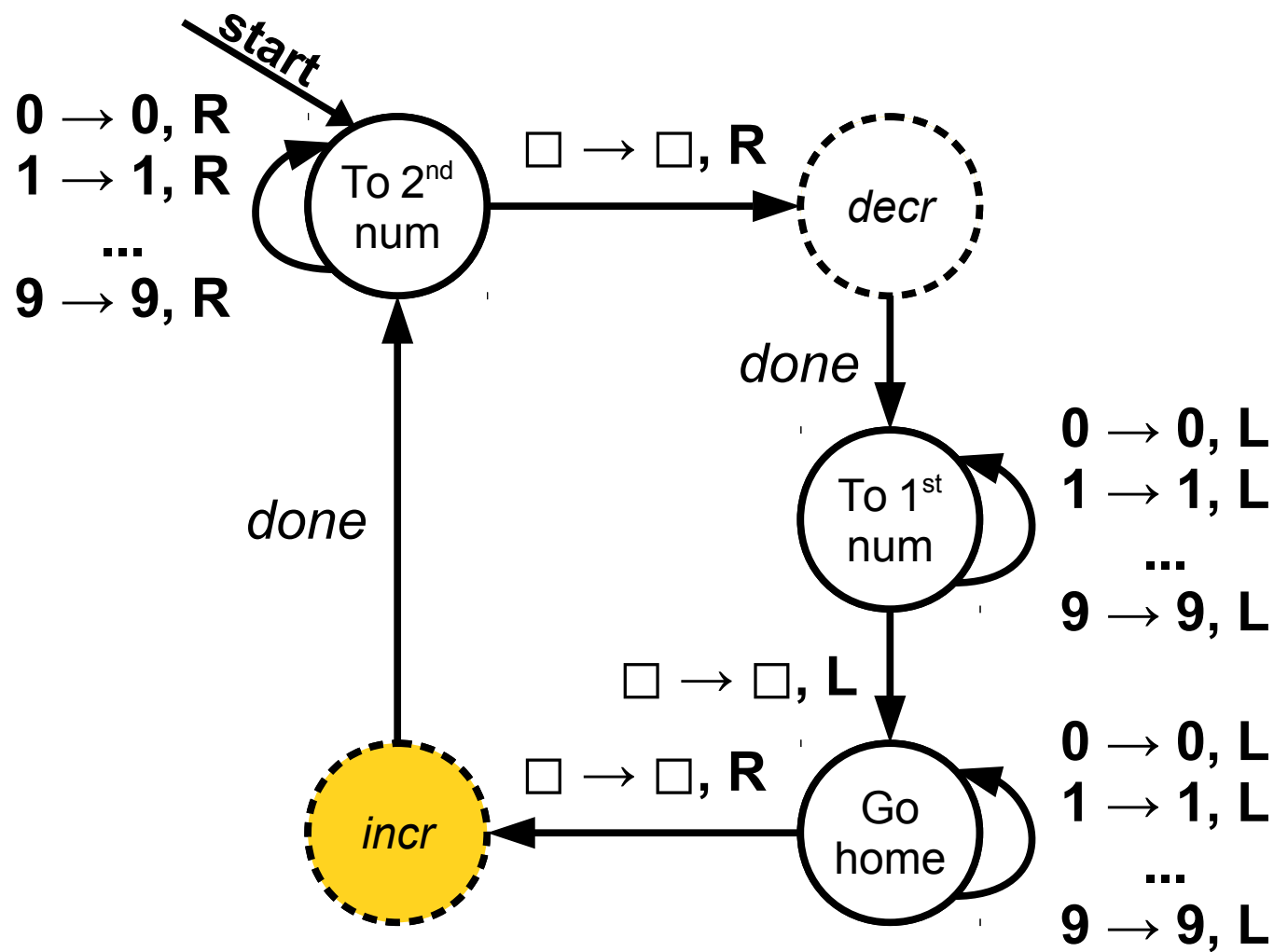


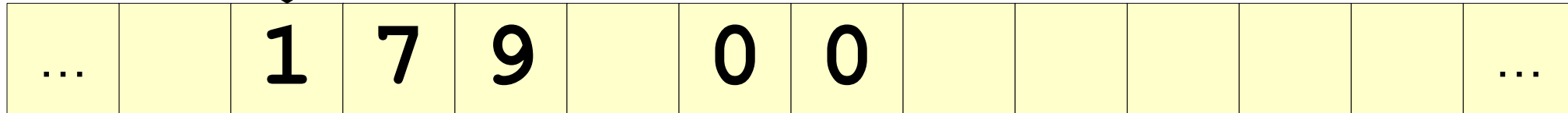
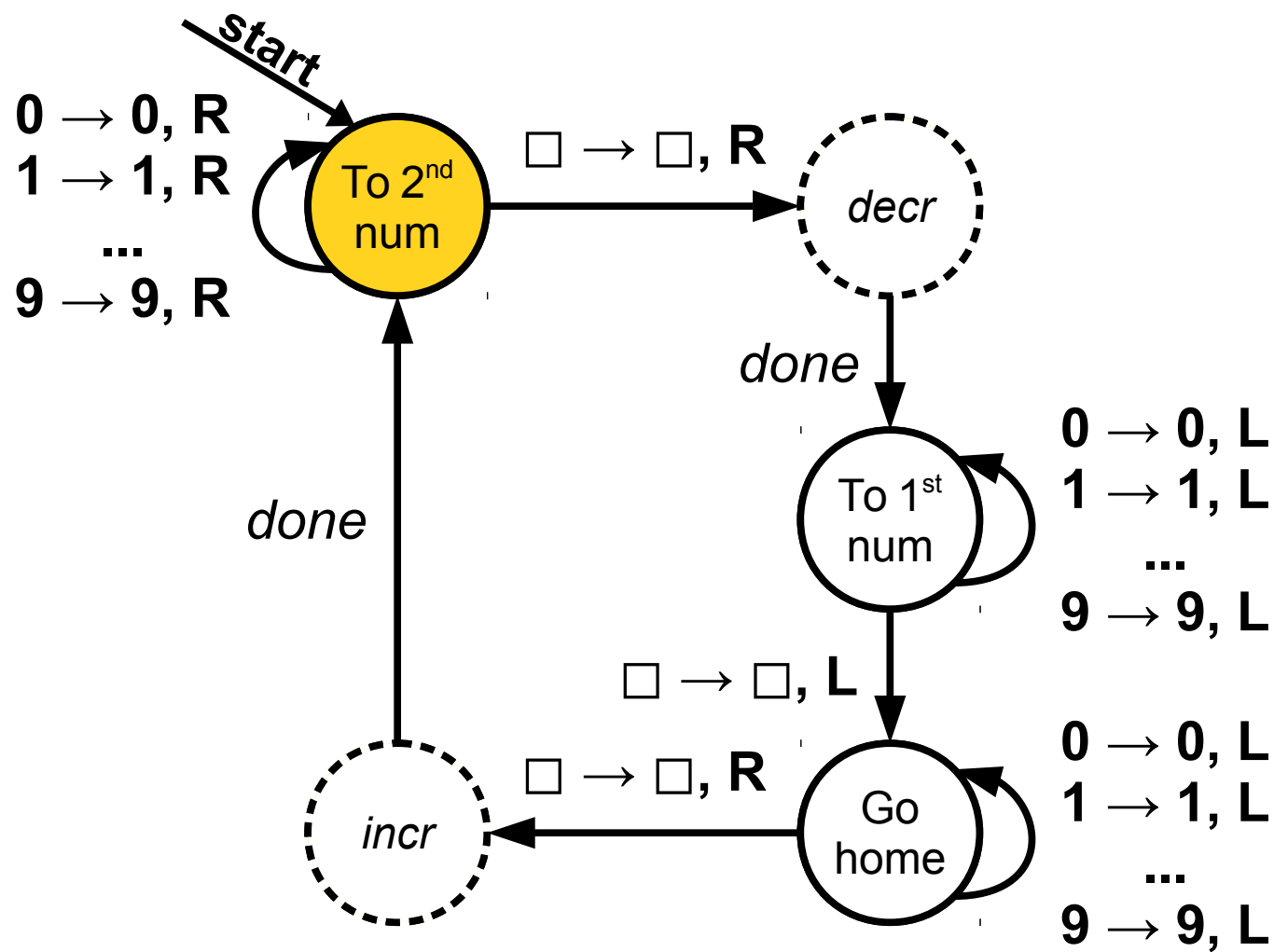


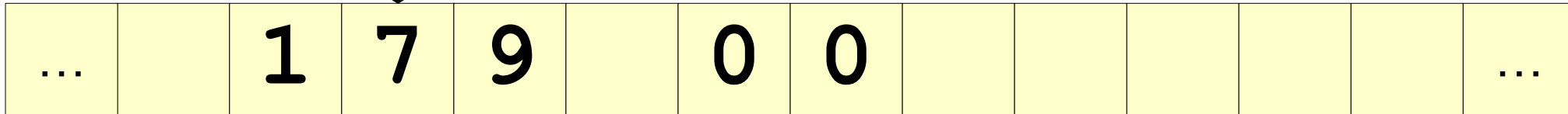
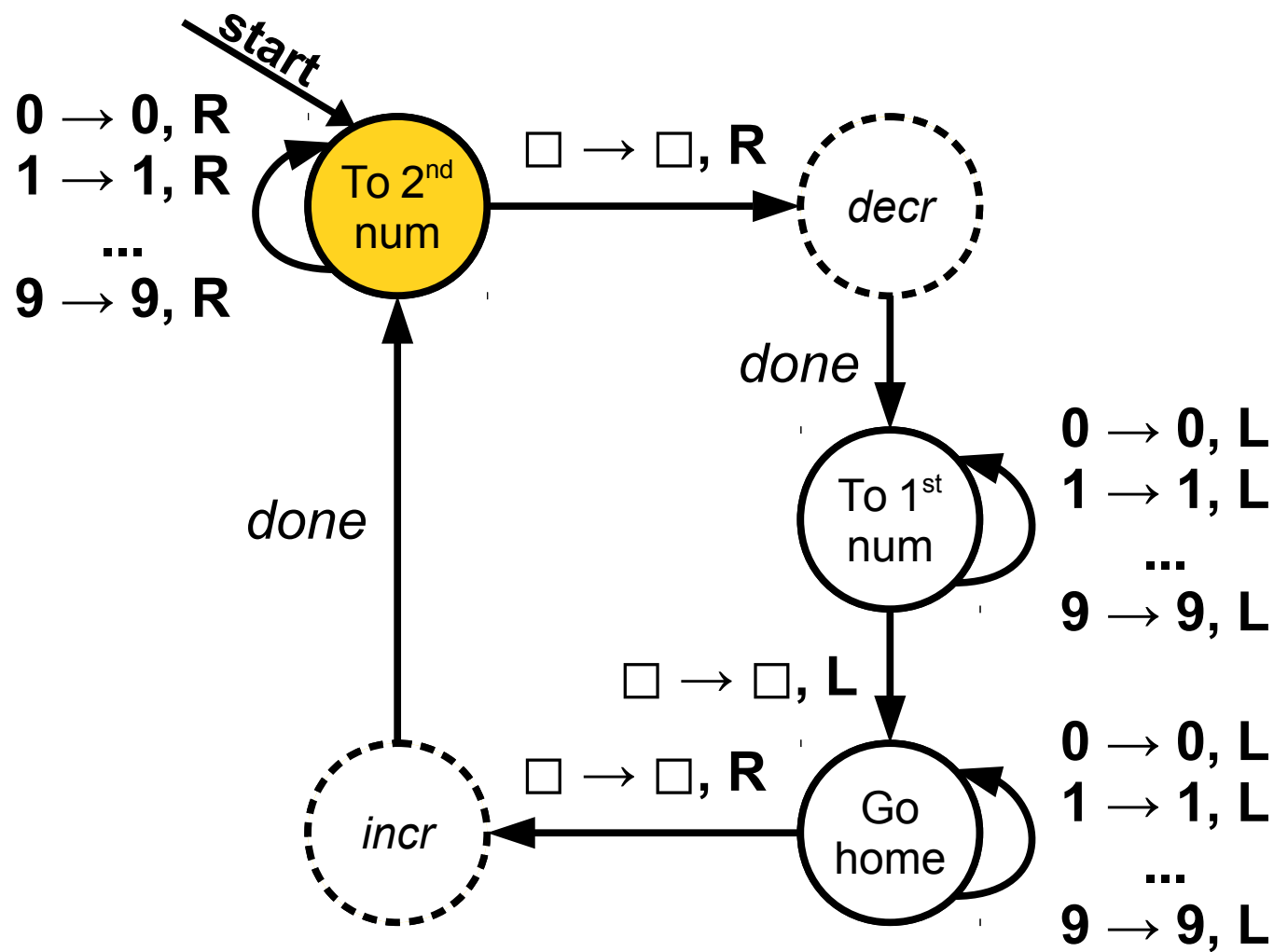


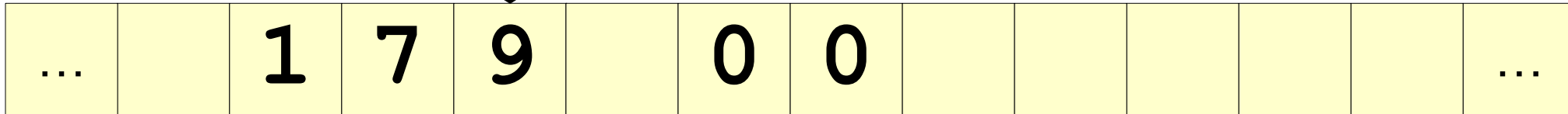
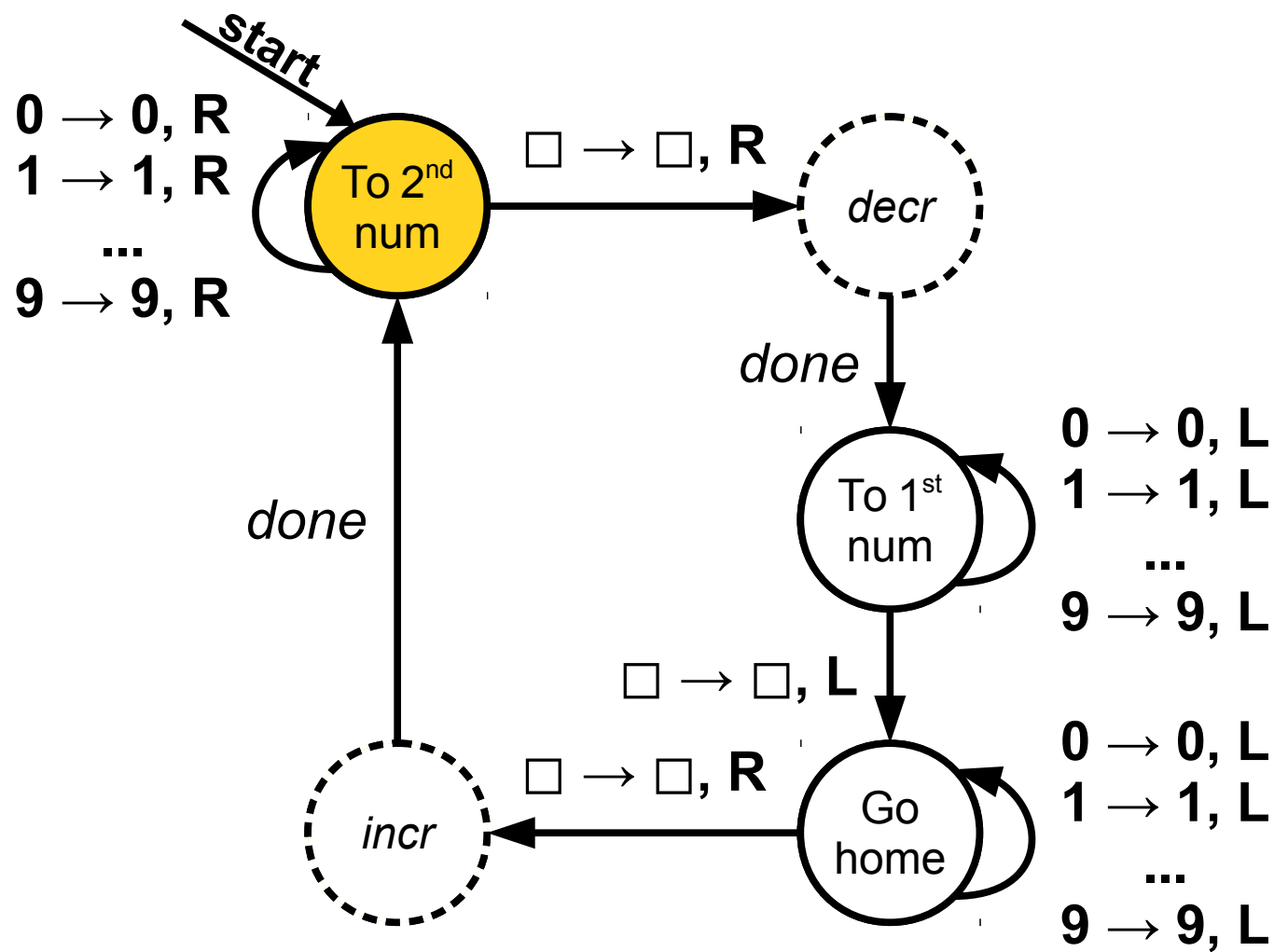


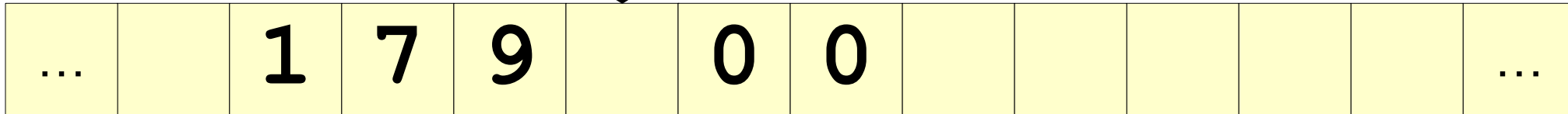
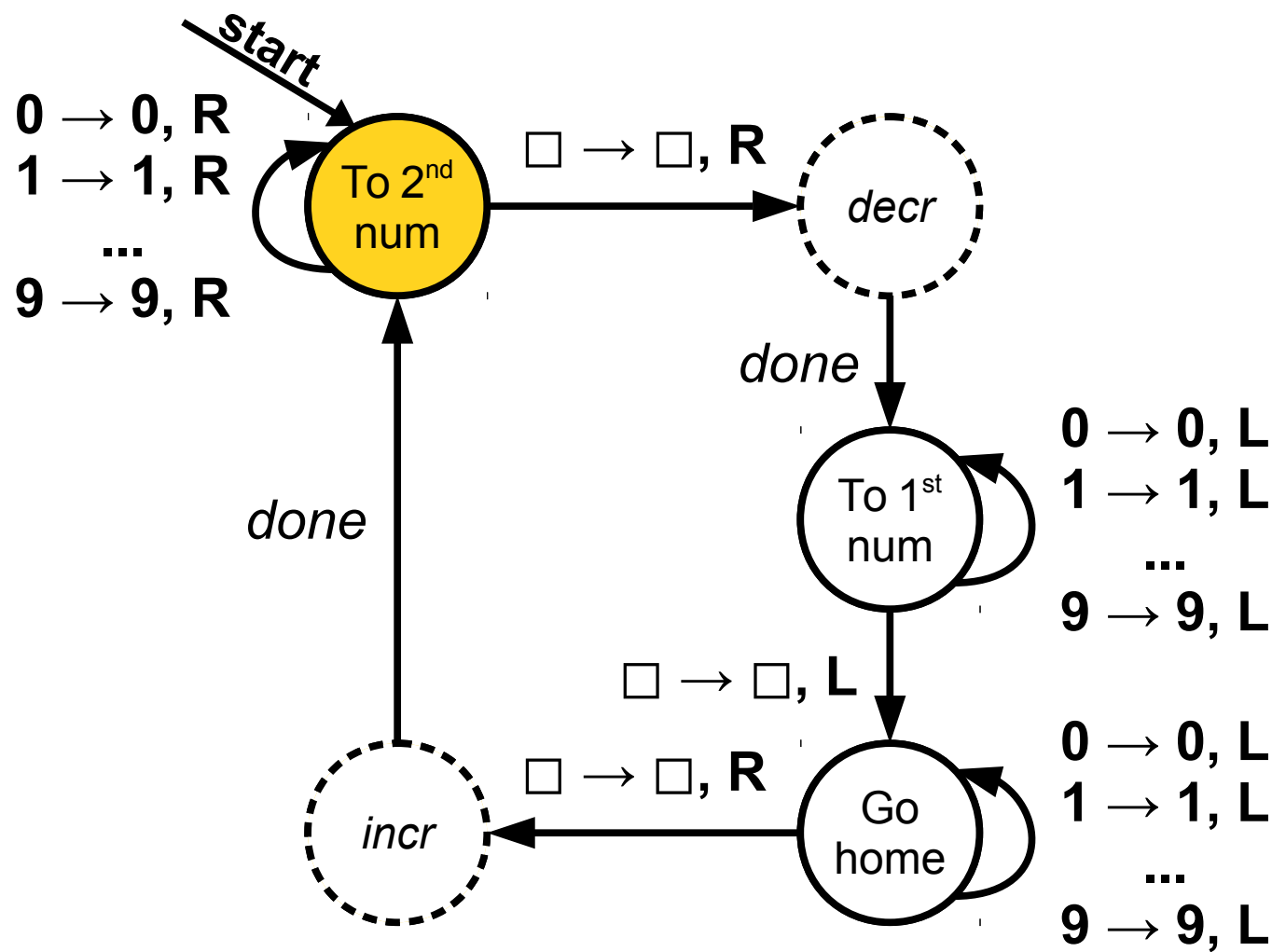


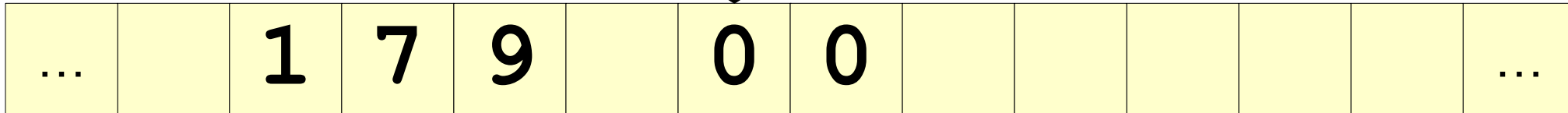
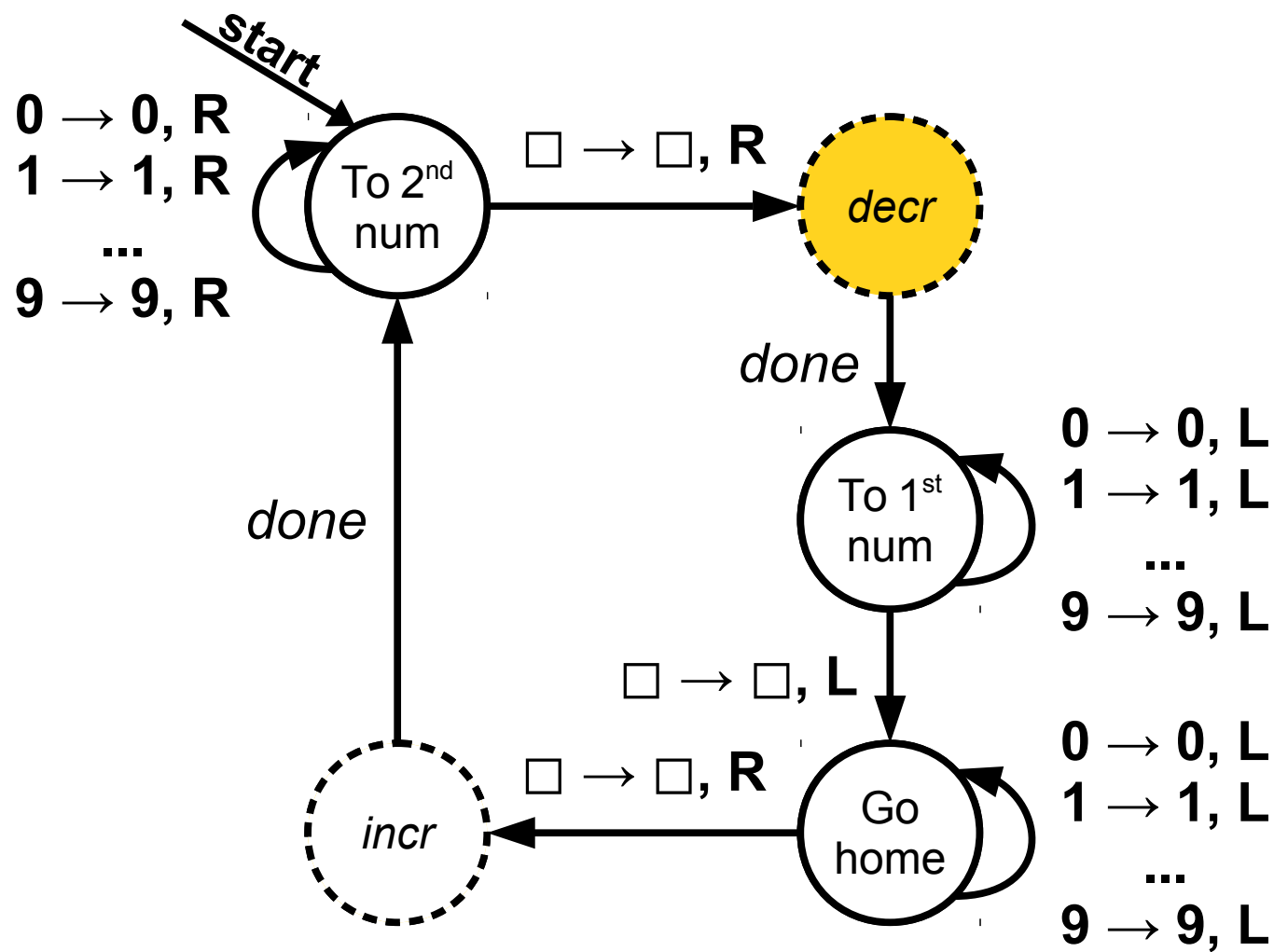


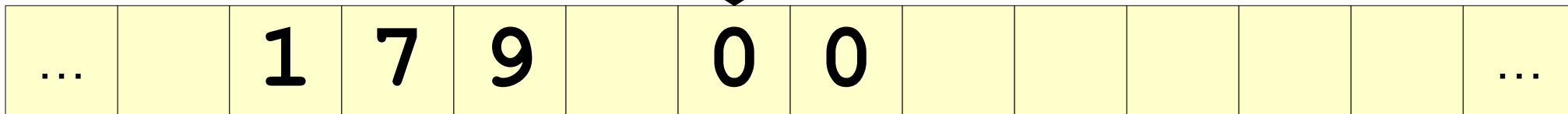
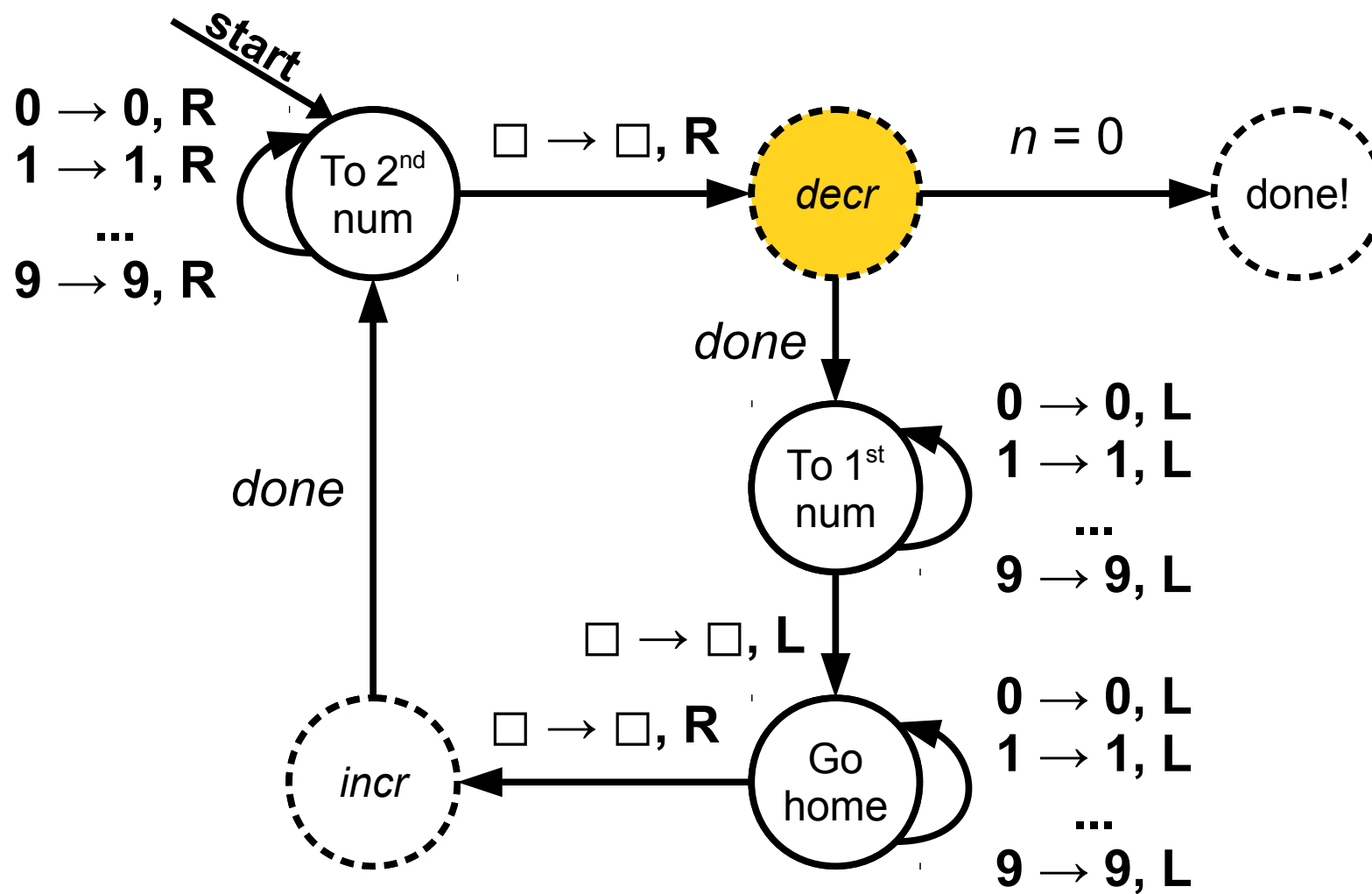


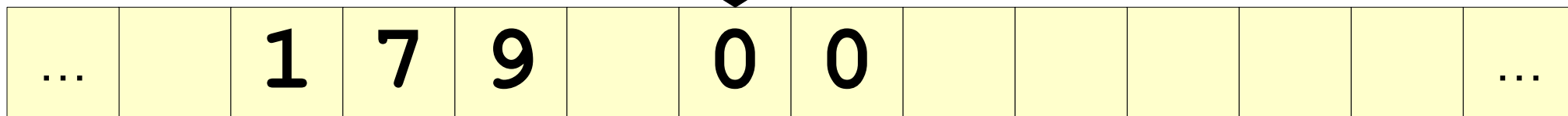
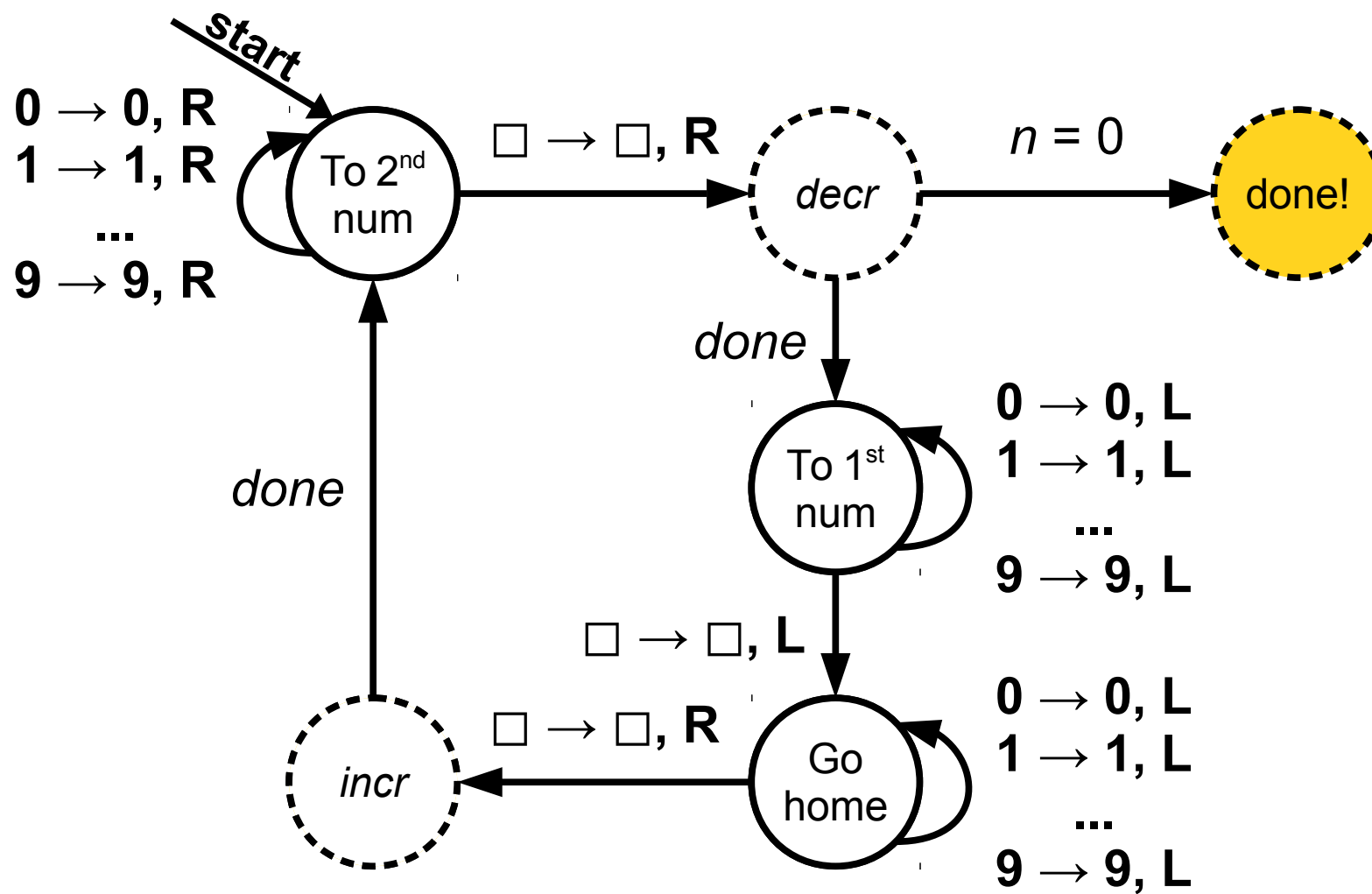


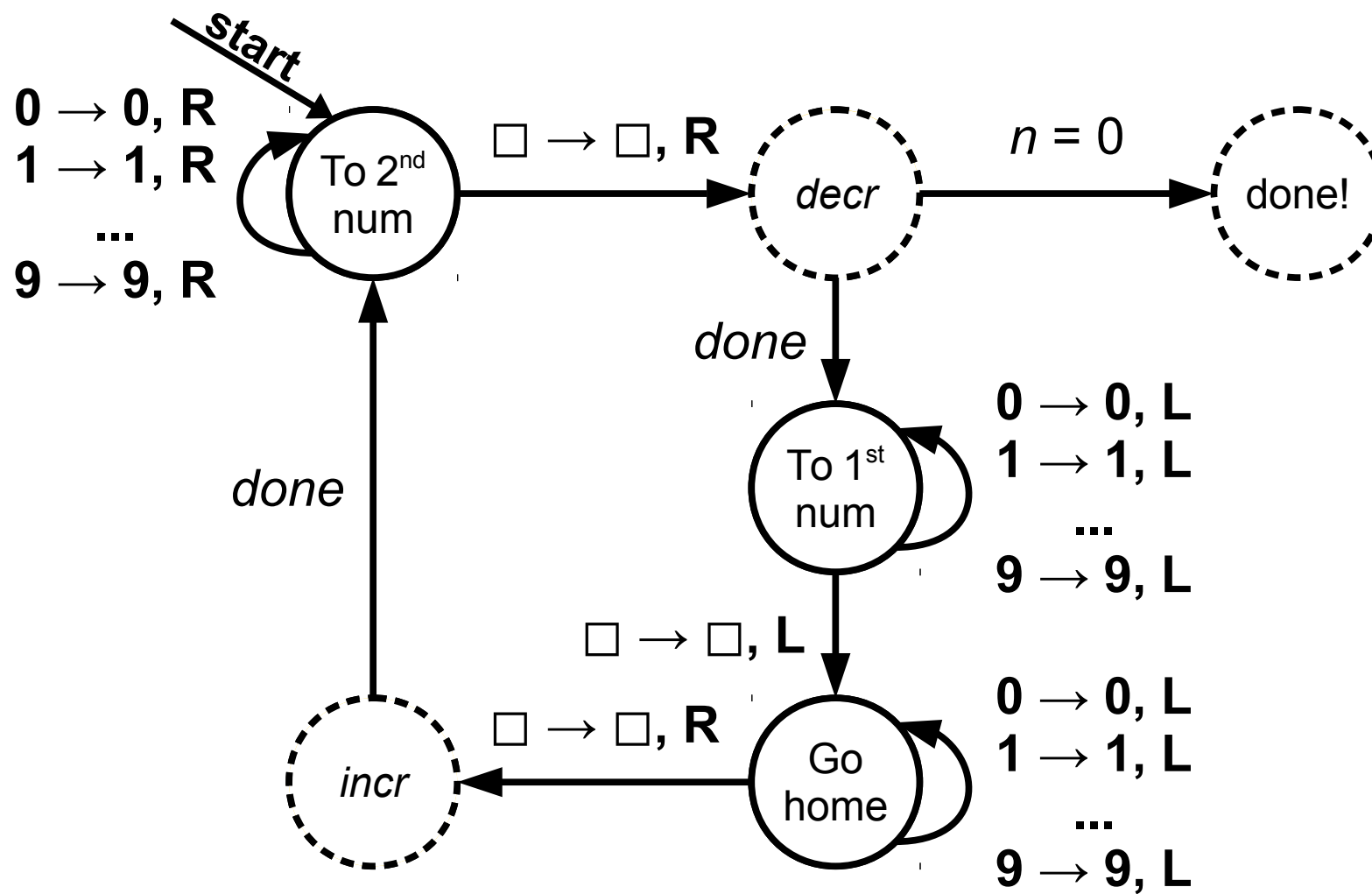






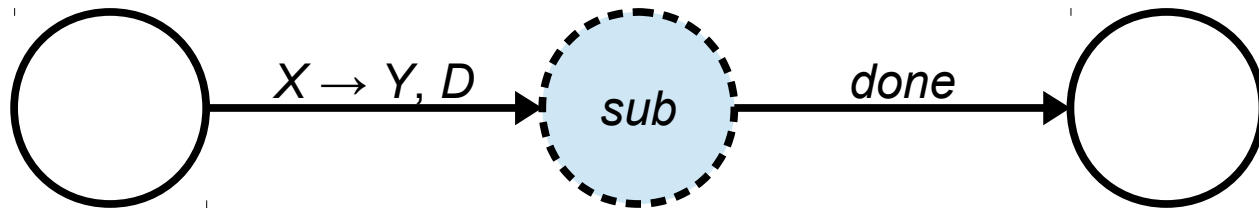






Using Subroutines

- Once you've built a subroutine, you can wire it into another TM with something that, schematically, looks like this:



- Intuitively, this corresponds to transitioning to the start state of the subroutine, then replacing the “done” state of the subroutine with the state at the end of the transition.

Time-Out for Announcements!

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 not tested.
 - **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.

Midterm Locations

- The midterm is in Hewlett.
- Specifically, locations are divvied up by last (family) name:
 - **Abd** – **Pre**: Go to Hewlett 200.
 - **Pri** – **Vil**: Go to Hewlett 201.
 - **Vo** – **Xie**: Go to Hewlett 101.
 - **Yan** – **Zhu**: Go to Hewlett 103.
- Need to take the exam at an alternate time for non-OAE reasons? Let us know *immediately*!

Preparing for the Exam

- We will be holding a practice midterm ***tonight*** 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 tonight.
 - Can't make the whole exam? Just show up to part of it! Can't make it at all? It will be online.
- Solutions to EPP4 – EPP7 are now available in hardcopy now and, after lecture, at Gates.

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

“Do grades matter? I'm a cs student going to industry and have no cares to pursue anything beyond my B.S. I put in time to understand the material but should I be worrying out my GPA? Whats a good gpa for a cs field? Is Stanford grade inflation real?”

Respectively: kinda, kinda, not a terrible one, and kinda. I'm just going to take this question live, since I can't fit all my answers onto this slide. 😊

“I scored below the 25th percentile on the first midterm and feel like I have no hope now. I really enjoy the material in class but I feel like my grade is doomed now. Is it possible for me to still get within the A range? What should my next steps be...”

Sorry to hear that things didn't go well on the exam.

A few things to consider. First, the median course grade – which is usually $(79 \pm 2)\%$ – is the dividing line between a B and a B+ grade. A single bad exam score won't irreversibly destroy your grade in this course. You're not at risk of failing, and you're not “doomed.” My advice is to figure out what went wrong with the exam. Talk to the course staff. Get a sense of what specific skills you need to work on. Do a lot of practice problems and check them over with the course staff. Rewrite old problem set questions without looking at the solutions and make sure you understand the solutions. It is not at all uncommon for students to really turn things around from the first exam to the second, so hang in there!

“Hey Keith! Can you tell us a funny story to brighten up our Week 8 blues? :)”



Back to CS103!

Main Question for Today:

Just how powerful are Turing machines?

How Powerful are TMs?

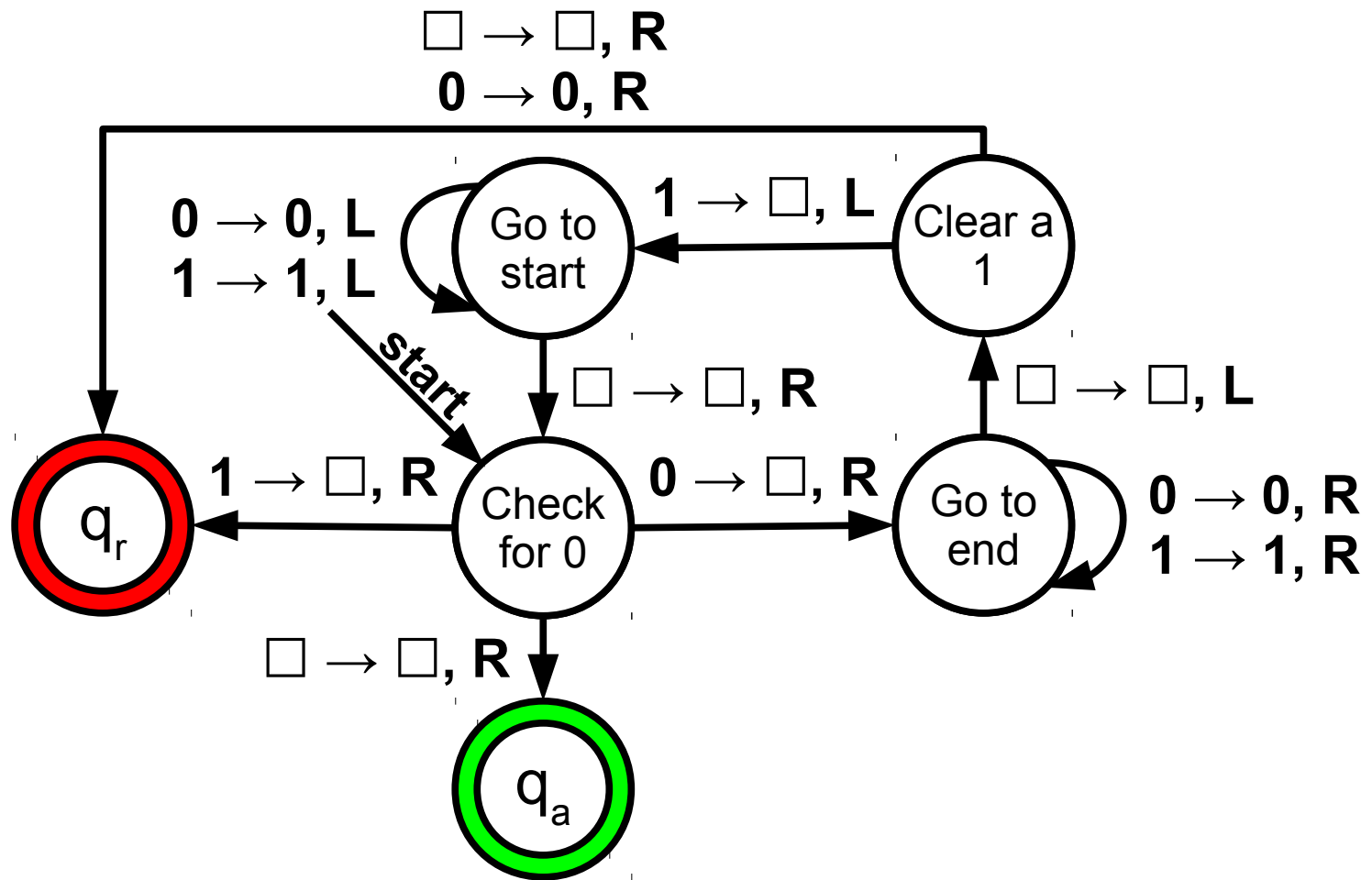
- Regular languages, intuitively, are as powerful as computers with finite memory.
- TMs by themselves seem like they can do a fair number of tasks, but it's unclear specifically what they can do.
- Let's explore their expressive power.

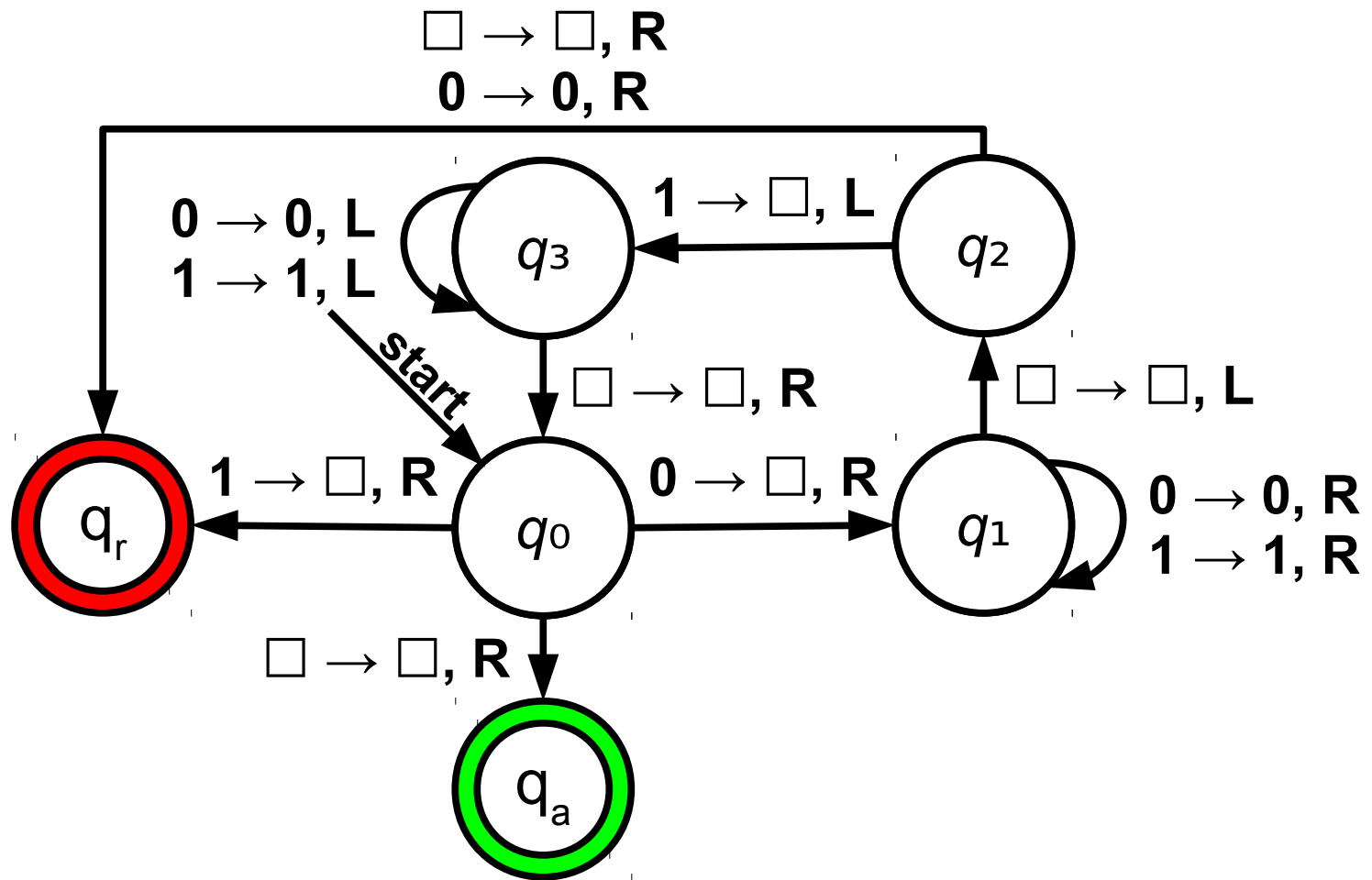
Real and “Ideal” Computers

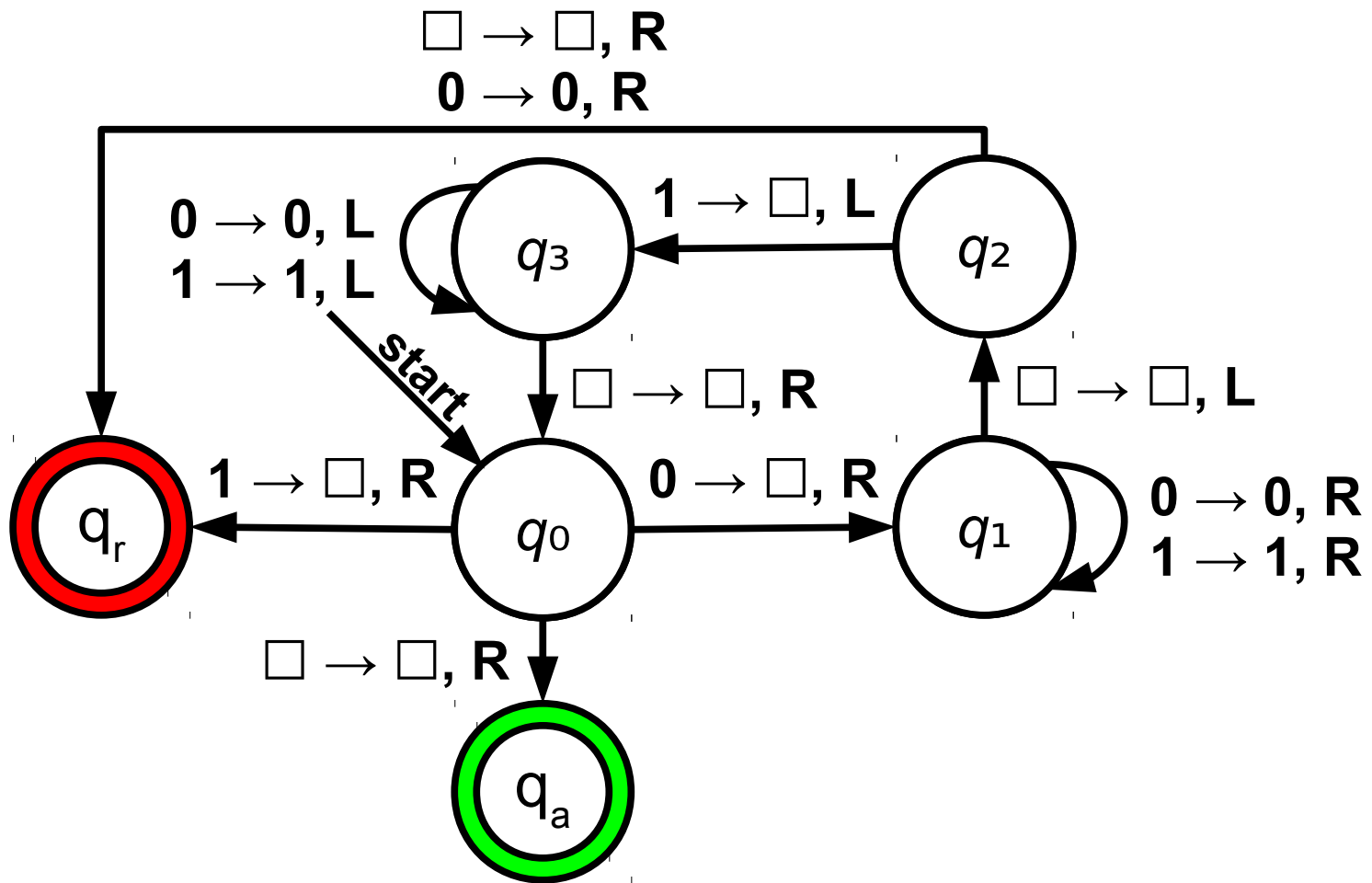
- A real computer has memory limitations: you have a finite amount of RAM, a finite amount of disk space, etc.
- However, as computers get more and more powerful, the amount of memory available keeps increasing.
- An ***idealized computer*** is like a regular computer, but with unlimited RAM and disk space. It functions just like a regular computer, but never runs out of memory.

Claim 1: Idealized computers can simulate Turing machines.

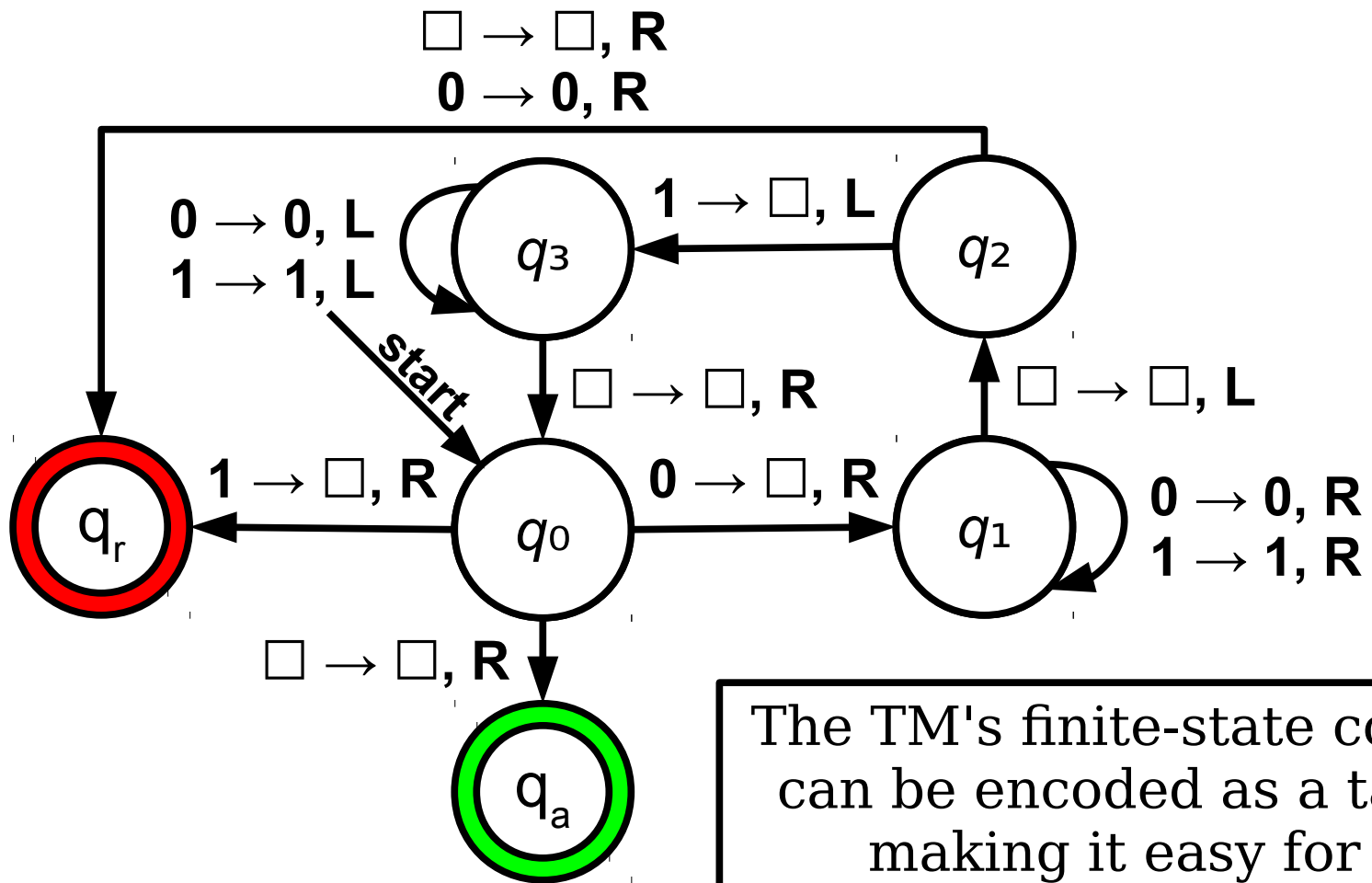
“Anything that can be done with a TM can also be done with an unbounded-memory computer.”







	0			1			□		
q_0	q_1	□	R	q_r	□	R	q_a	□	R
q_1	q_1	0	R	q_1	1	R	q_2	□	L
q_2	q_r	0	R	q_3	□	L	q_r	□	R
q_3	q_3	0	L	q_3	1	L	q_0	□	R



The TM's finite-state control can be encoded as a table, making it easy for a computer to look up transitions information.

		0		1		
q_0	q_1	\square	R	q_r	\square	R
q_1	q_1	0	R	q_1	1	R
q_2	q_r	0	R	q_3	\square	L
q_3	q_3	0	L	q_3	1	L

Simulating a TM

- To simulate a TM, the computer would need to be able to keep track of
 - the finite-state control,
 - the current state,
 - the position of the tape head, and
 - the tape contents.
- The tape contents are infinite, but that's because there are infinitely many blanks on both sides.
- We only need to store the “interesting” part of the tape (the parts that have been read from or written to so far.)

...				1	7	9		0	0			...
-----	--	--	--	---	---	---	--	---	---	--	--	-----

Simulating a TM

- To simulate a TM, the computer would need to be able to keep track of
 - the finite-state control,
 - the current state,
 - the position of the tape head, and
 - the tape contents.
- The tape contents are infinite, but that's because there are infinitely many blanks on both sides.
- We only need to store the “interesting” part of the tape (the parts that have been read from or written to so far.)



Simulating a TM

- To simulate a TM, the computer would need to be able to keep track of
 - the finite-state control,
 - the current state,
 - the position of the tape head, and
 - the tape contents.
- The tape contents are infinite, but that's because there are infinitely many blanks on both sides.
- We only need to store the “interesting” part of the tape (the parts that have been read from or written to so far.)

1	7	9		0	0
---	---	---	--	---	---

Claim 2: Turing machines can simulate idealized computers.

“Anything that can be done with an unbounded-memory computer can be done with a TM.”

What We've Seen

- TMs can
 - implement loops (basically, every TM we've seen).
 - make function calls (subroutines).
 - keep track of natural numbers (written in unary or in decimal on the tape).
 - perform elementary arithmetic (equality testing, multiplication, addition, increment, decrement, etc.).
 - perform if/else tests (different transitions based on different cases).

What Else Can TMs Do?

- Maintain variables.
 - Have a dedicated part of the tape where the variables are stored.
 - We've seen this before: take a look at our machine for composite numbers, or for increment/decrement.
- Maintain arrays and linked structures.
 - Divide the tape into different regions corresponding to memory locations.
 - Represent arrays and linked structures by keeping track of the ID of one of those regions.

A CS107 Perspective

- Internally, computers execute by using basic operations like
 - simple arithmetic,
 - memory reads and writes,
 - branches and jumps,
 - register operations,
 - etc.
- Each of these are simple enough that they could be simulated by a Turing machine.

A Leap of Faith

- It may require a leap of faith, but anything you can do a computer (excluding randomness and user input) can be performed by a Turing machine.
- The resulting TM might be colossal, or really slow, or both, but it would still faithfully simulate the computer.
- We're going to take this as an article of faith in CS103. If you curious for more details, come talk to me after class.

Just how powerful *are* Turing machines?

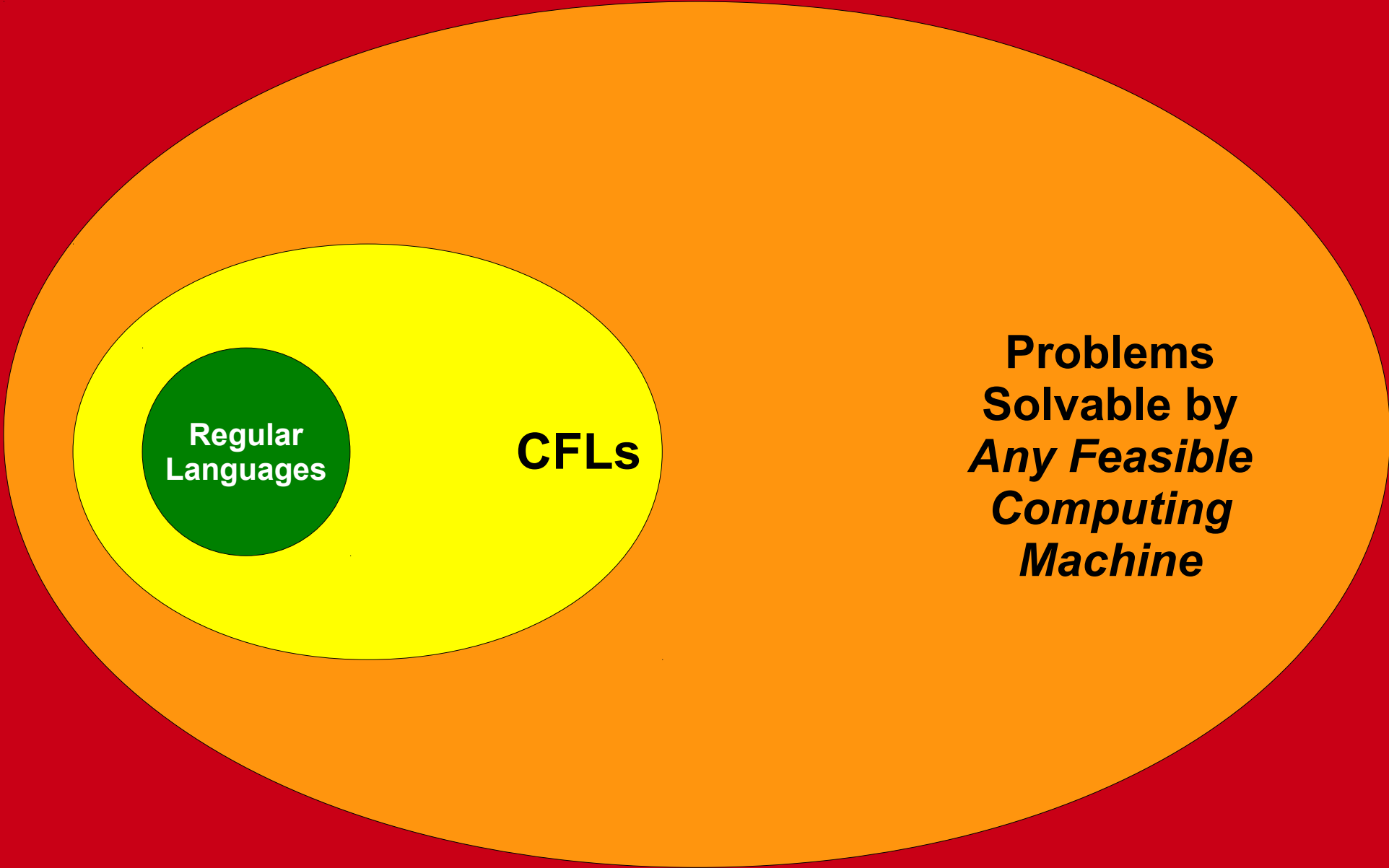
Effective Computation

- An ***effective method of computation*** is a form of computation with the following properties:
 - The computation consists of a set of steps.
 - There are fixed rules governing how one step leads to the next.
 - Any computation that yields an answer does so in finitely many steps.
 - Any computation that yields an answer always yields the correct answer.
- This is not a formal definition. Rather, it's a set of properties we expect out of a computational system.

The ***Church-Turing Thesis*** claims that
every effective method of computation is either
equivalent to or weaker than a Turing machine.

“This is not a theorem – it is a
falsifiable scientific hypothesis.
And it has been thoroughly
tested!”

- Ryan Williams



Regular
Languages

CFLs

**Problems
Solvable by
*Any Feasible
Computing
Machine***

All Languages

**Regular
Languages**

CFLs

**Problems
solvable by
Turing
Machines**

All Languages

TMs \approx Computers

- Because Turing machines have the same computational powers as regular computers, we can (essentially) reason about Turing machines by reasoning about actual computer programs.
- Going forward, we're going to switch back and forth between TMs and computer programs based on whatever is most appropriate.
- In fact, our eventual proofs about the existence of impossible problems will involve a good amount of pseudocode. Stay tuned for details!