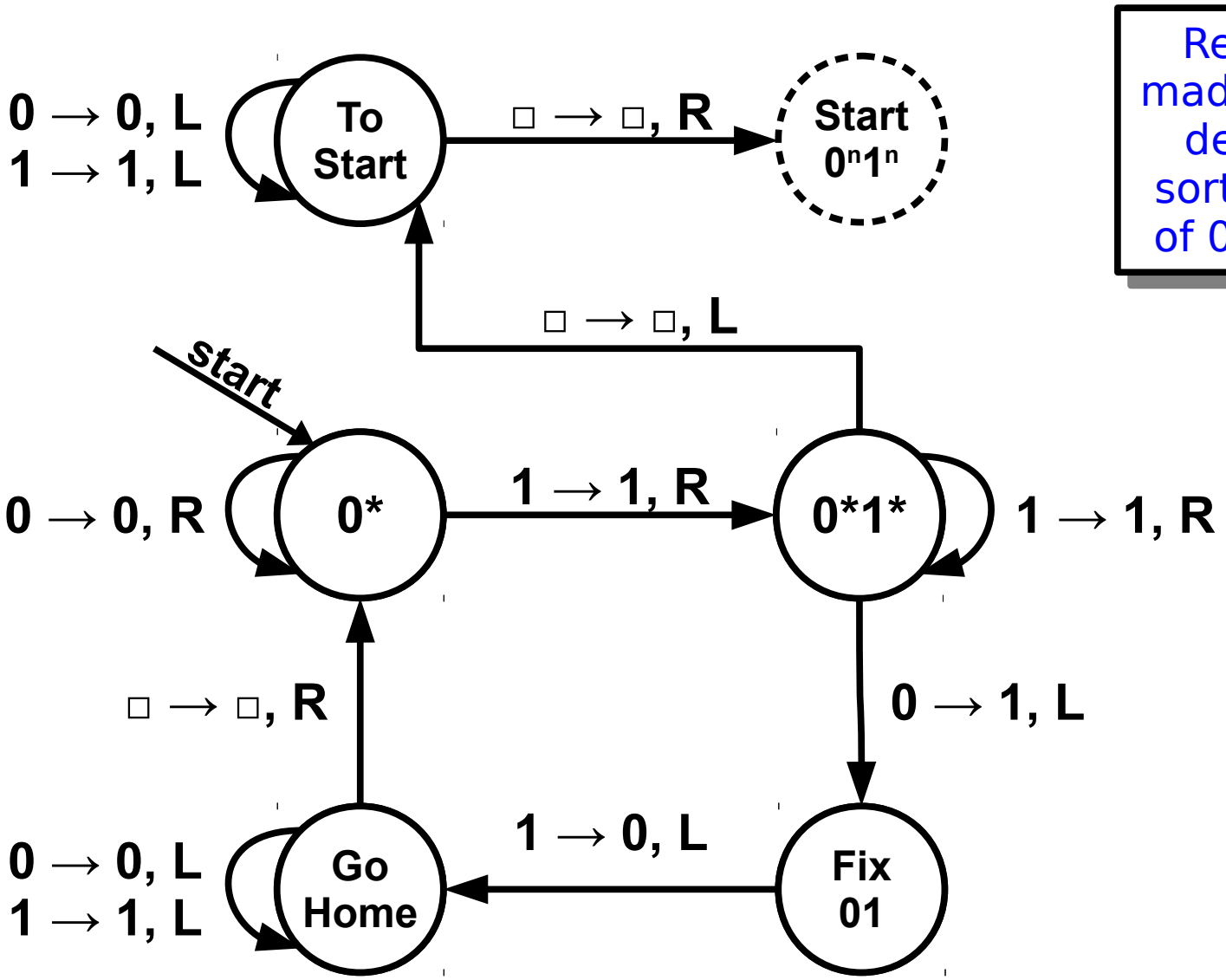
Turing Machines

Part Two

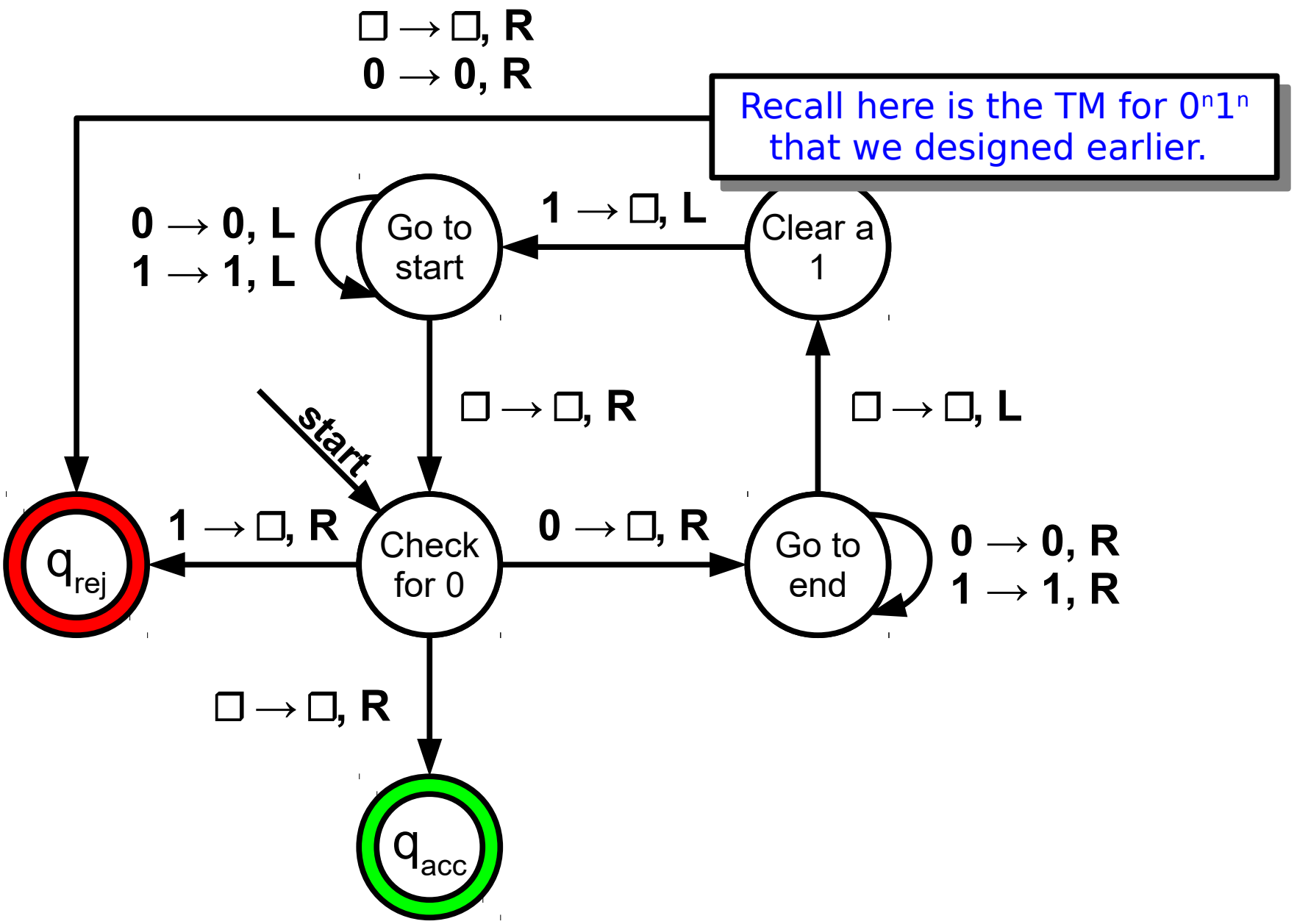
Picking up from Last Time

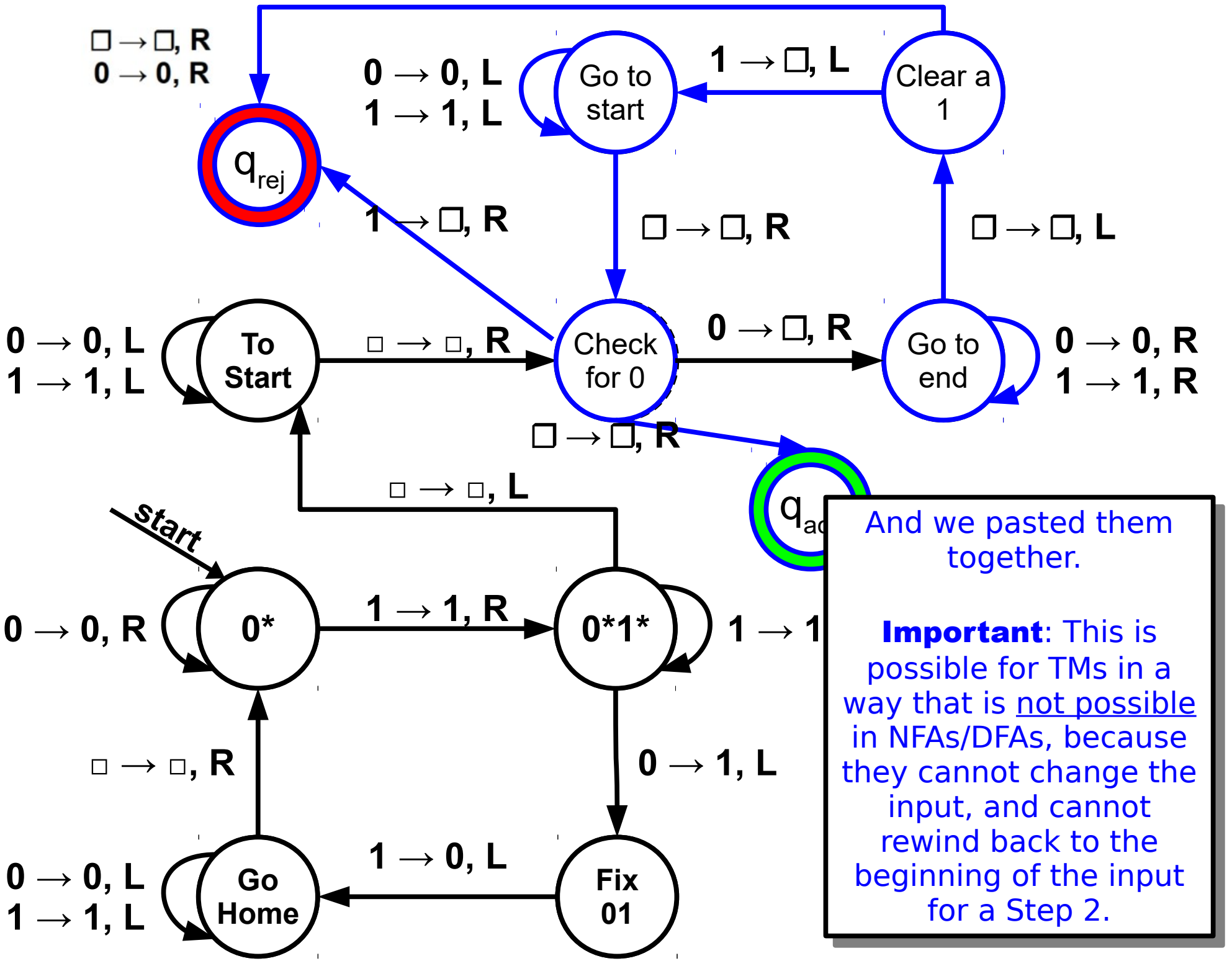
Goal: make a TM that checks if there are the same number of 0s and 1s in a string.

Step1: sort the digits of the string
Step2: use the TM we already made that can check if a string is in the form 0^n1^n



Recall we made this TM design to sort a string of 0s and 1s


□ → □, R
0 → 0, R
Recall here is the TM for 0ⁿ1ⁿ that we designed earlier.
0 → 0, L
1 → 1, L
Go to start
1 → □, L
Clear a 1
□ → □, R
□ → □, L
start
1 → □, R
Check for 0
0 → □, R
Go to end
0 → 0, R
1 → 1, R
q_rej
□ → □, R
q_acc



And we pasted them together.

Important: This is possible for TMs in a way that is <u>not possible</u> in NFAs/DFAs, because they cannot change the input, and cannot rewind back to the beginning of the input for a Step 2.

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.

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.

Little nit-picky caveat

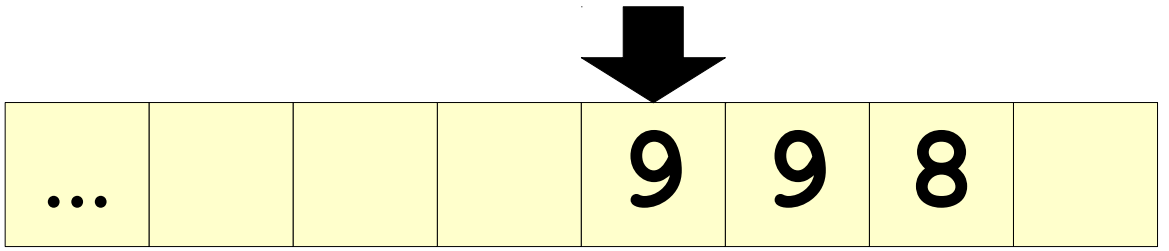
- For this problem, we will allow exactly one blank in the middle of our input string.
- Usually this is not allowed.
- To change this to adhere to our usual no-blanks rule, we could have specified a number-delimiter character other than blank.
- You could imagine doing that and then writing a pre-processor TM that finds that character and replaces it with blank, before running what we are about to design.

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.

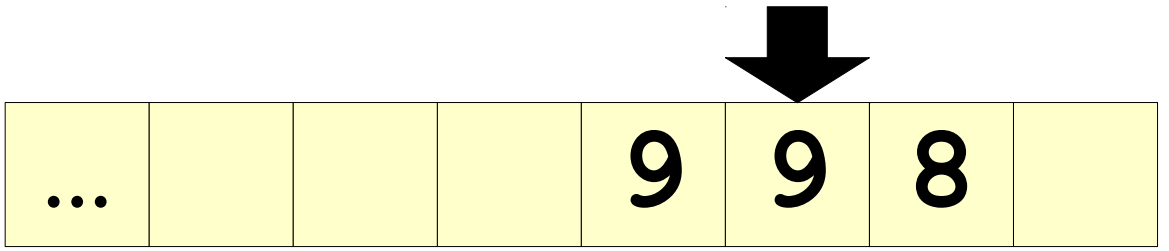
Incrementing Numbers

- 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.



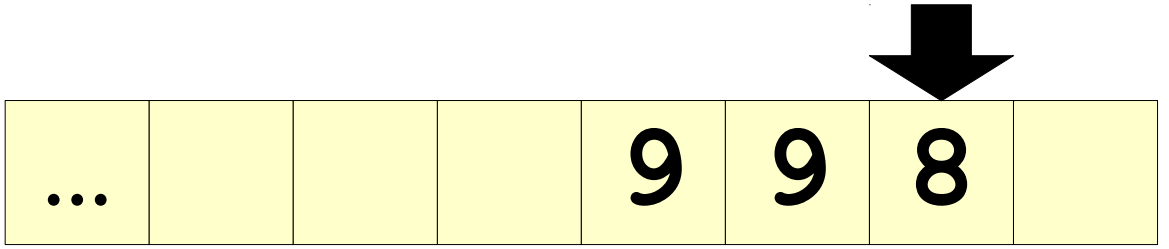
Incrementing Numbers

- 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.



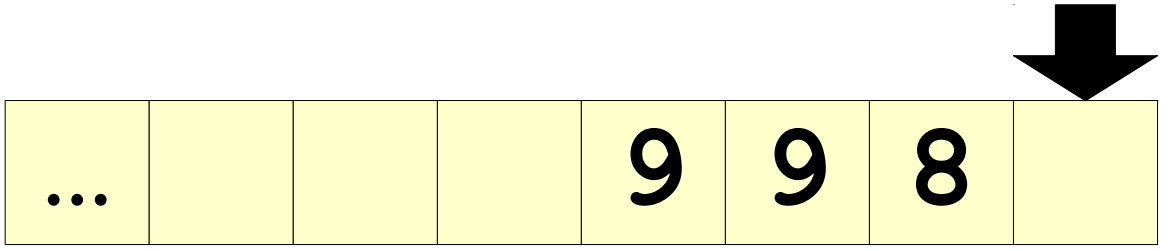
Incrementing Numbers

- 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.



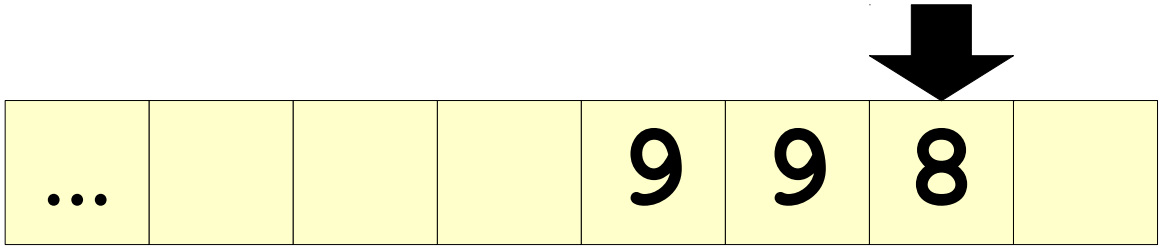
Incrementing Numbers

- 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.



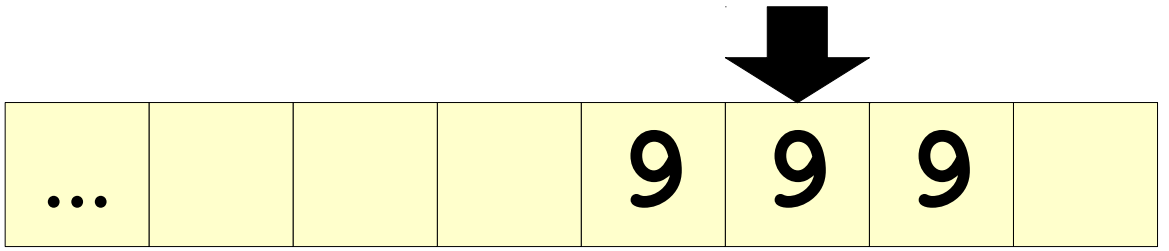
Incrementing Numbers

- 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.



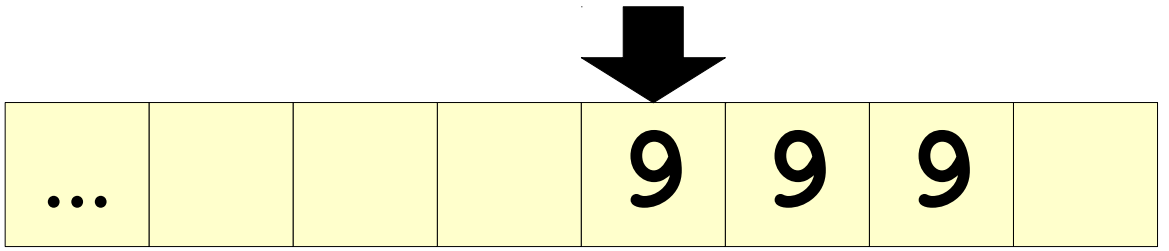
Incrementing Numbers

- 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.



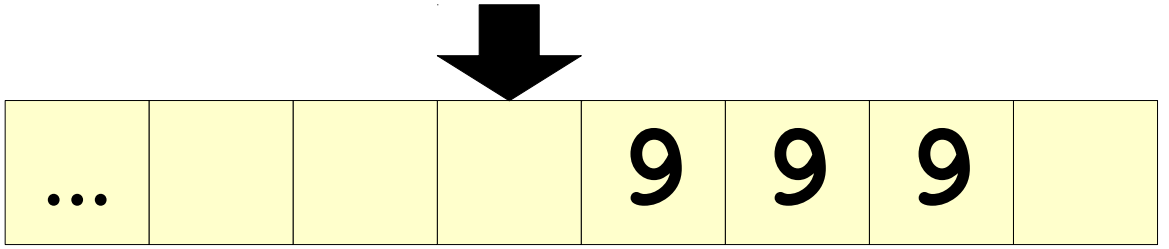
Incrementing Numbers

- 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.



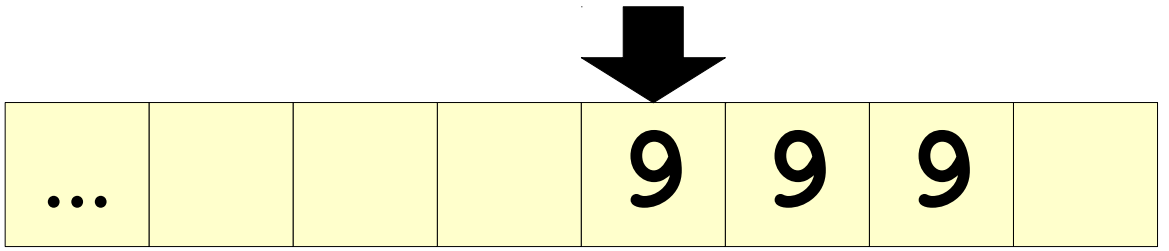
Incrementing Numbers

- 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.



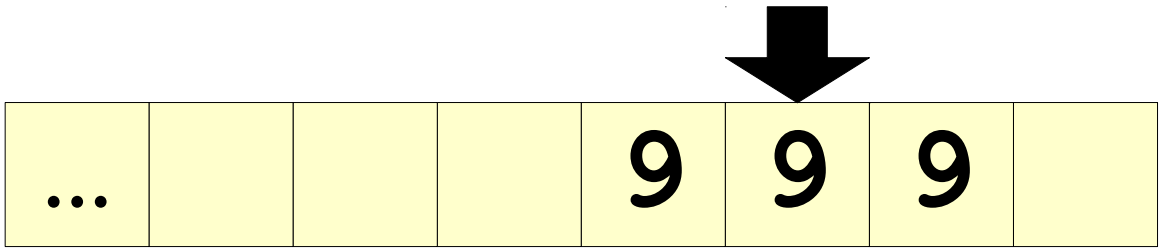
Incrementing Numbers

- 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.



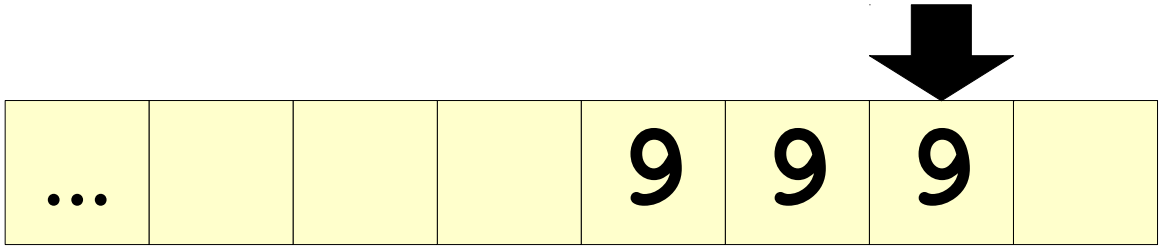
Incrementing Numbers

- 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.



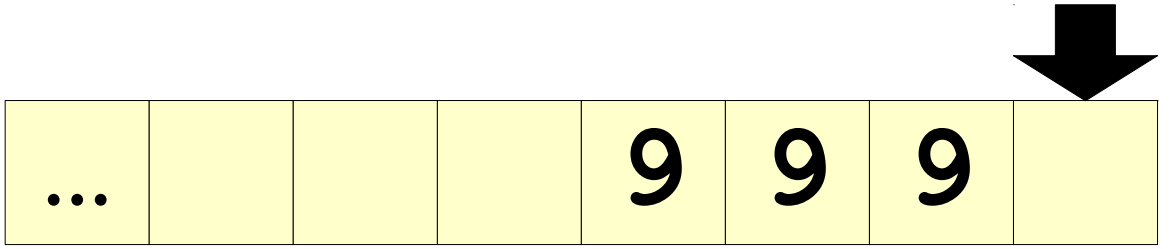
Incrementing Numbers

- 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.



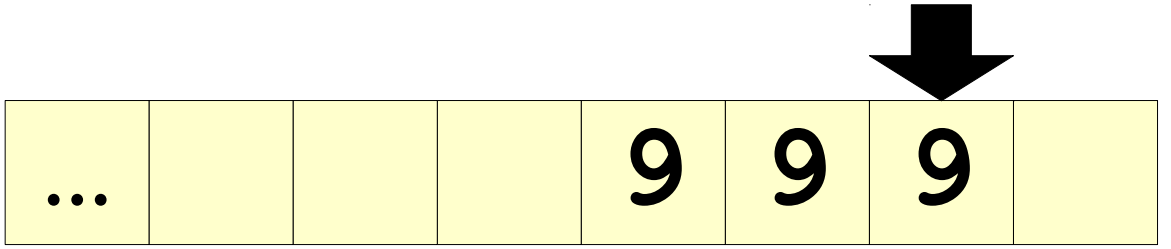
Incrementing Numbers

- 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.



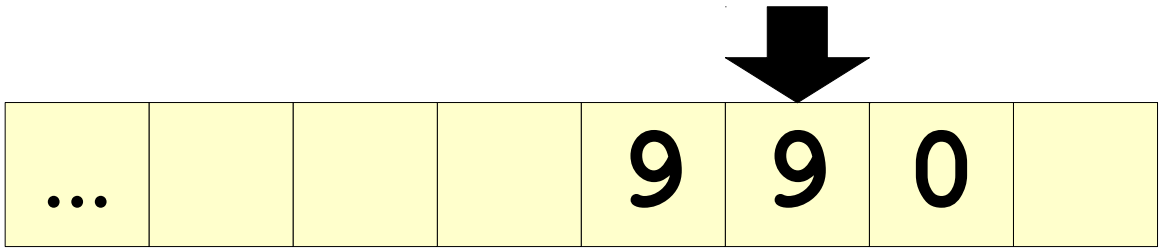
Incrementing Numbers

- 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.



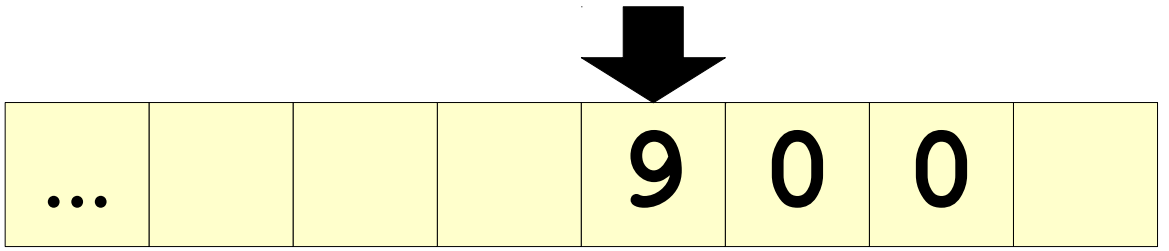
Incrementing Numbers

- 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.



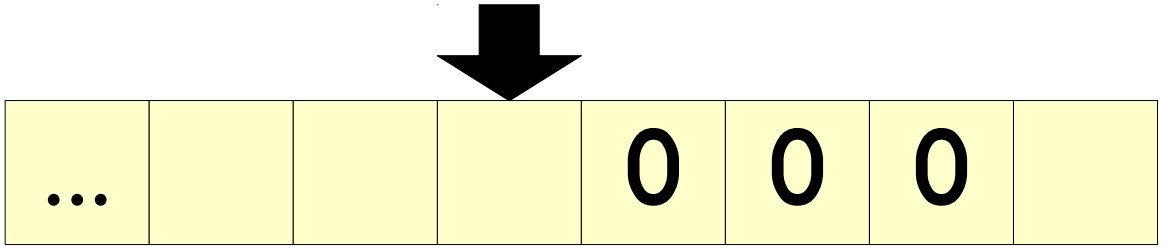
Incrementing Numbers

- 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.



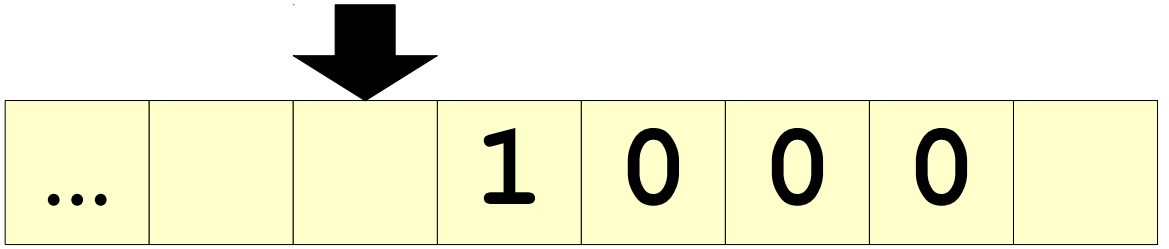
Incrementing Numbers

- 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.



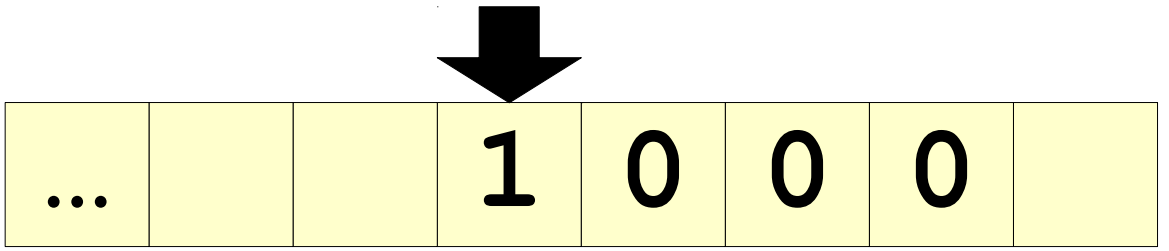
Incrementing Numbers

- 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.



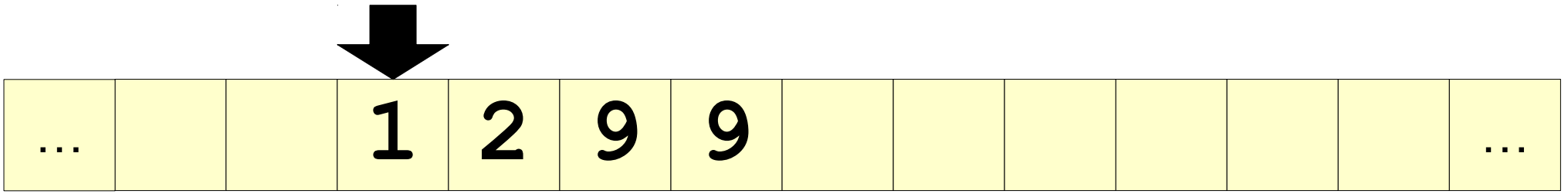
Incrementing Numbers

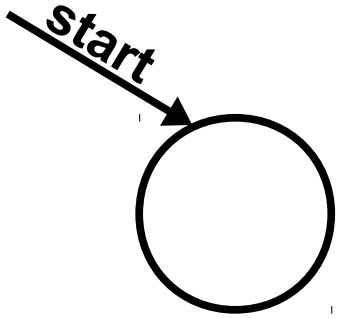
- Let's begin by building a TM that increments a number.
- We'll assume that
  - the tape head points at the start of a number,
  - there is are at least two blanks to the left of the number, and
  - that there's at least one blank at the start of the number.
- The tape head will end at the start of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

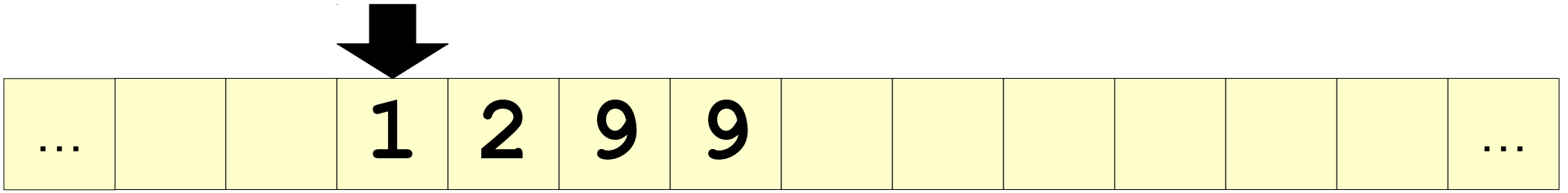


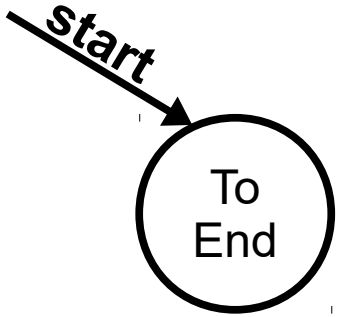
Incrementing Numbers

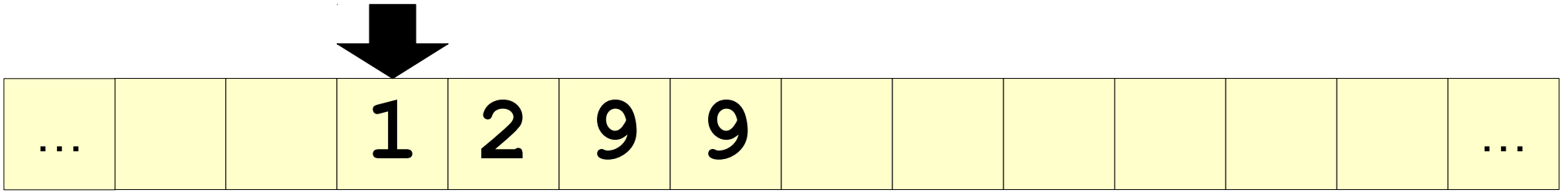
```
go to the end of the number;
while (the current digit is 9) {
    set the current digit to 0;
    back up one digit;
}
increment the current digit;
go to the start of the number;
```

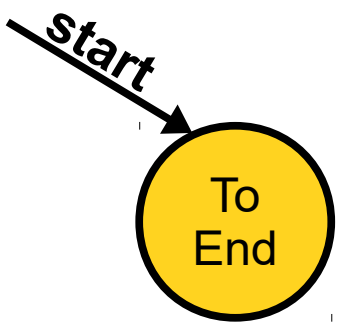

...
1 2 9 9
...

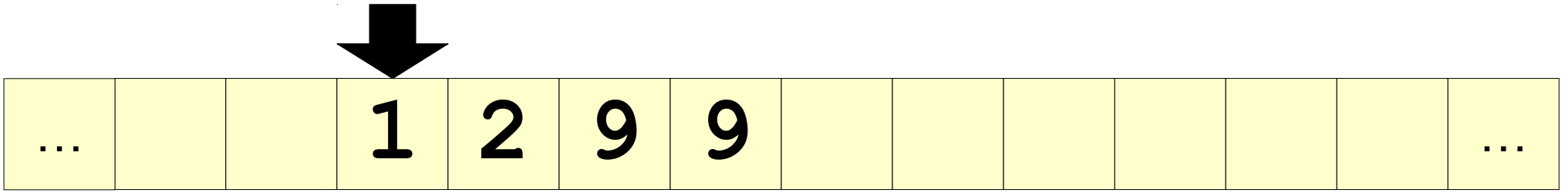

start

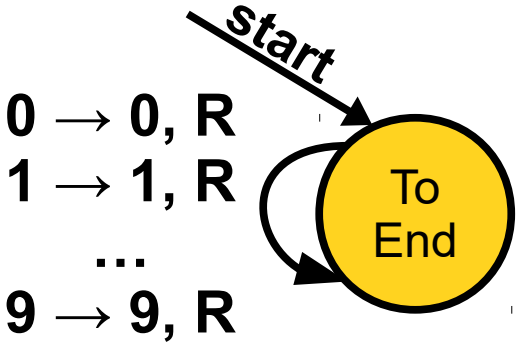

...
1
2
9
9
...

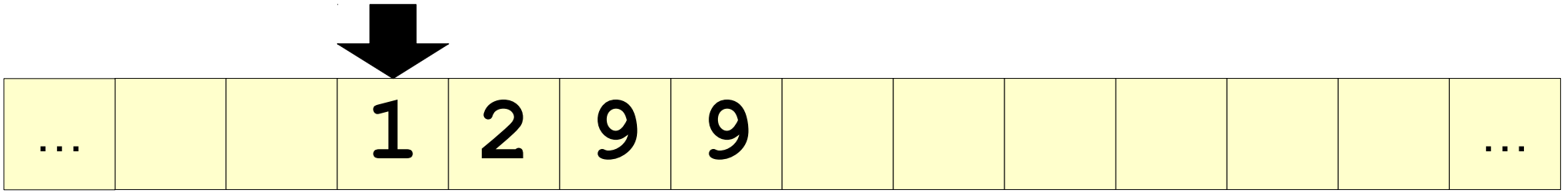

start
To
End

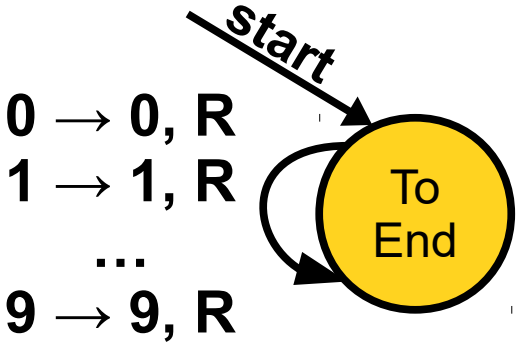

...
1
2
9
9
...

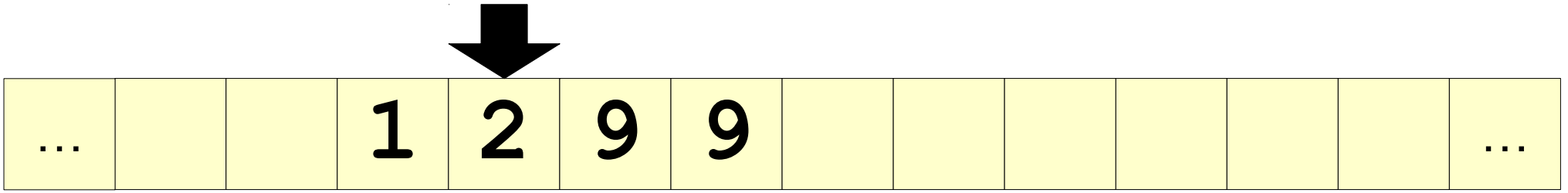

start
To
End

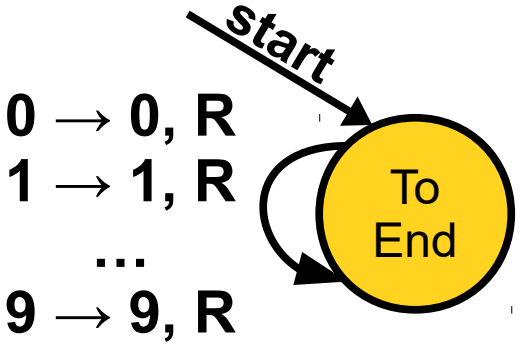

...
1
2
9
9
...

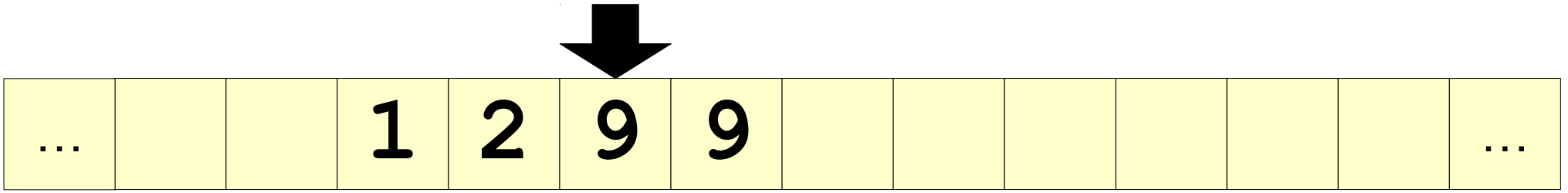

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To
End

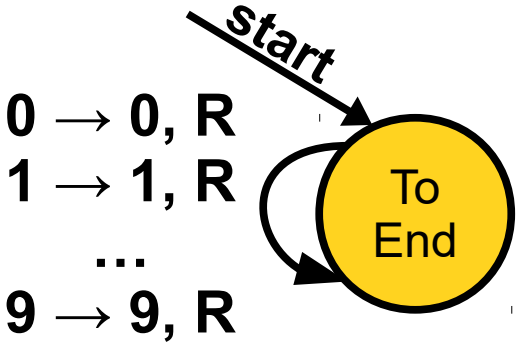

…
1
2
9
9
…

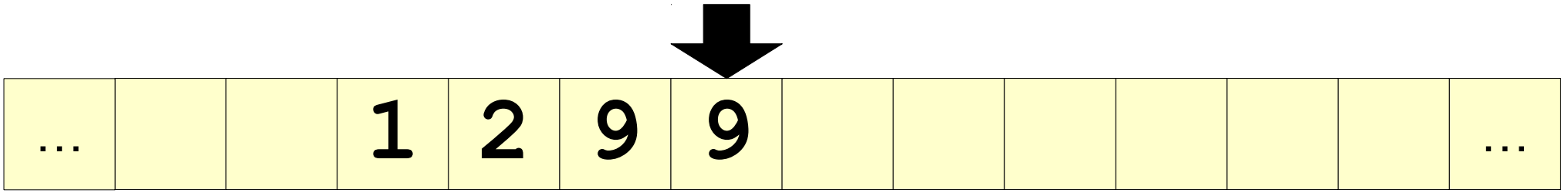

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To
End

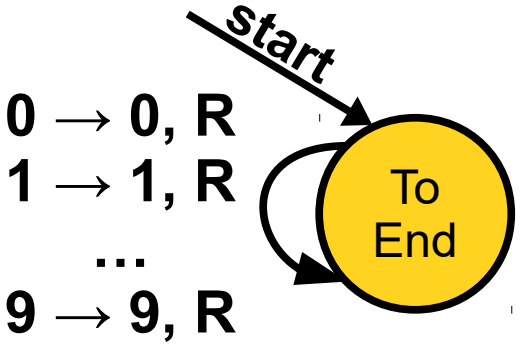

…
1
2
9
9
…

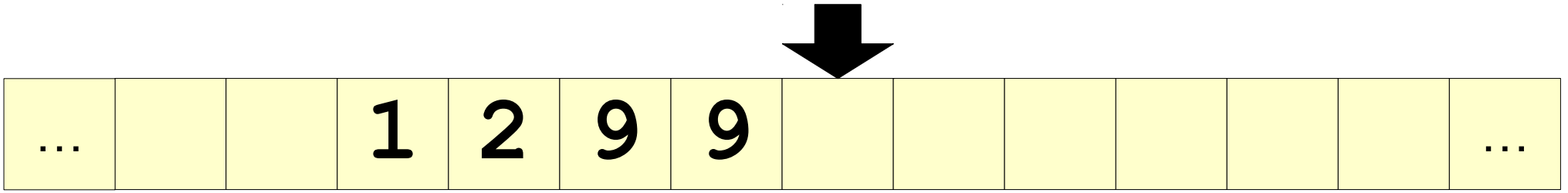

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To
End

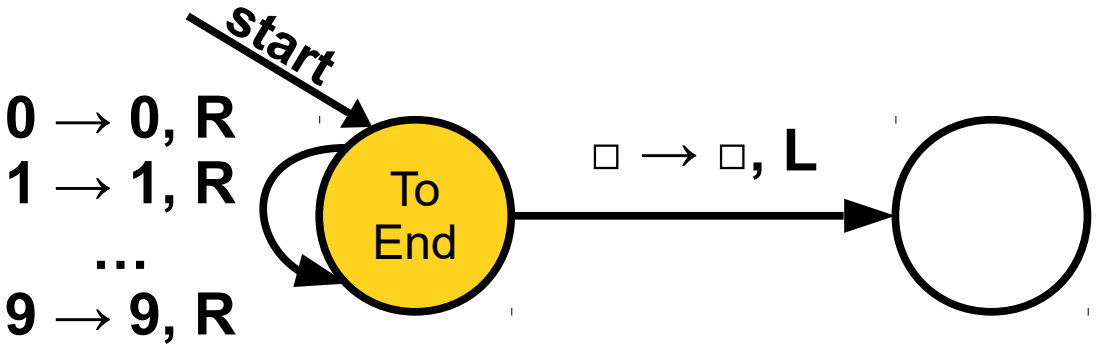

…
1
2
9
9
…

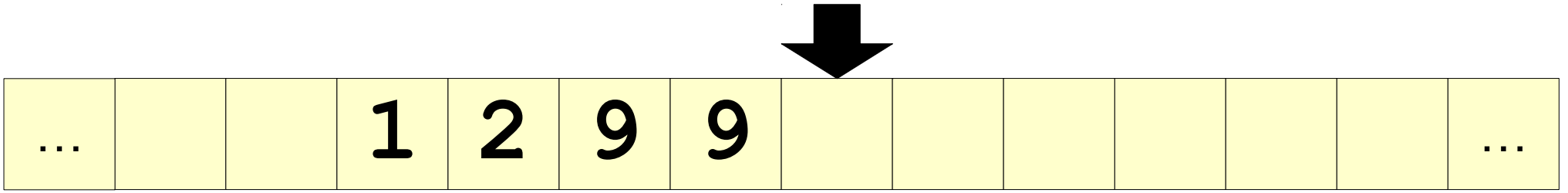

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To
End

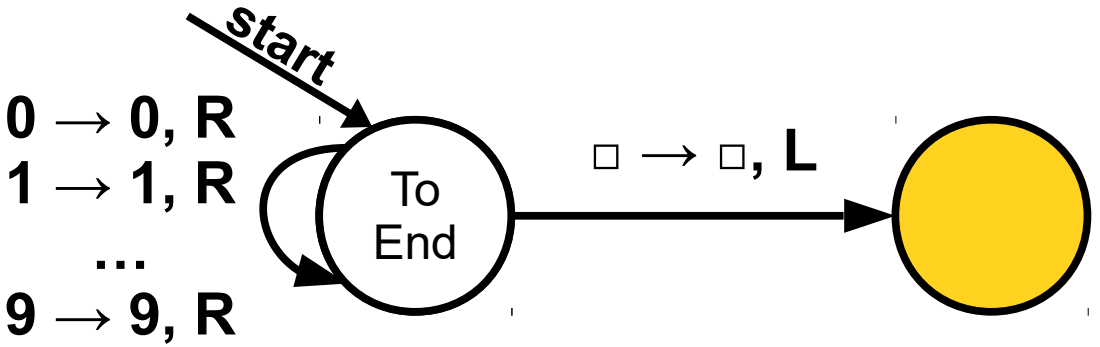

…
1
2
9
9
…

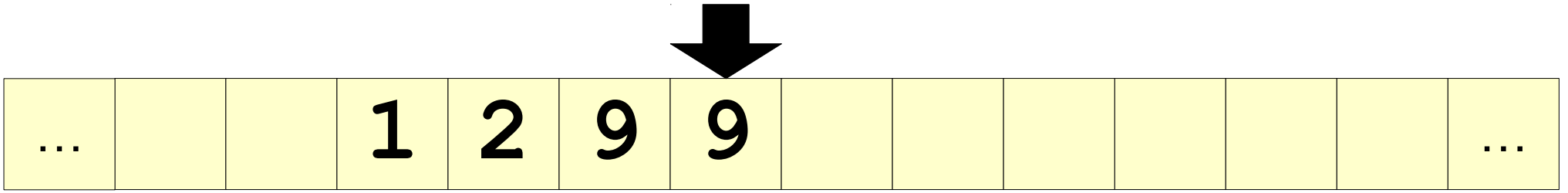

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To
End

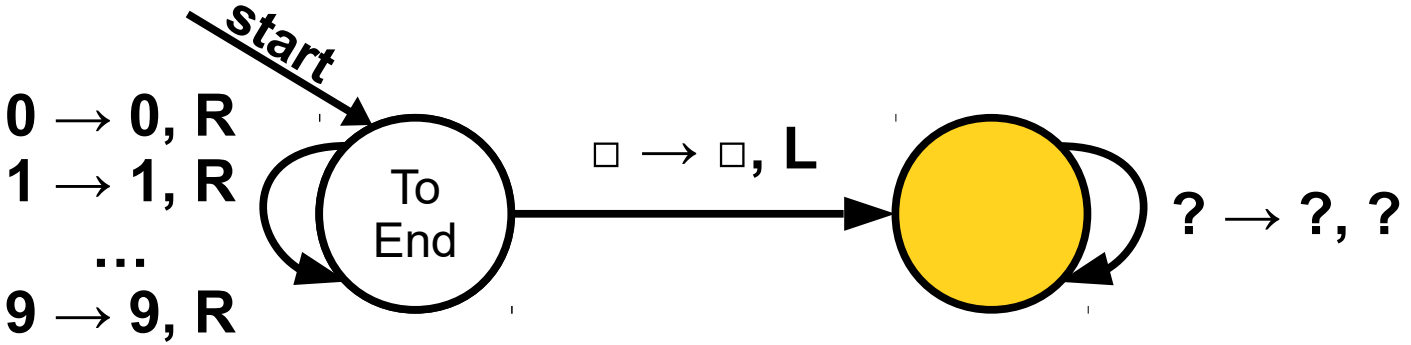

…
1
2
9
9
…


start
0 → 0, R
1 → 1, R
…
9 → 9, R
To
End
□ → □, L


…
1
2
9
9
…

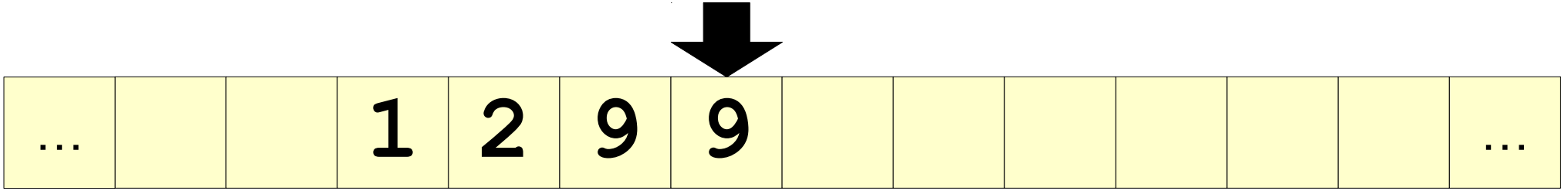

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To
End
□ → □, L


...
1
2
9
9
...

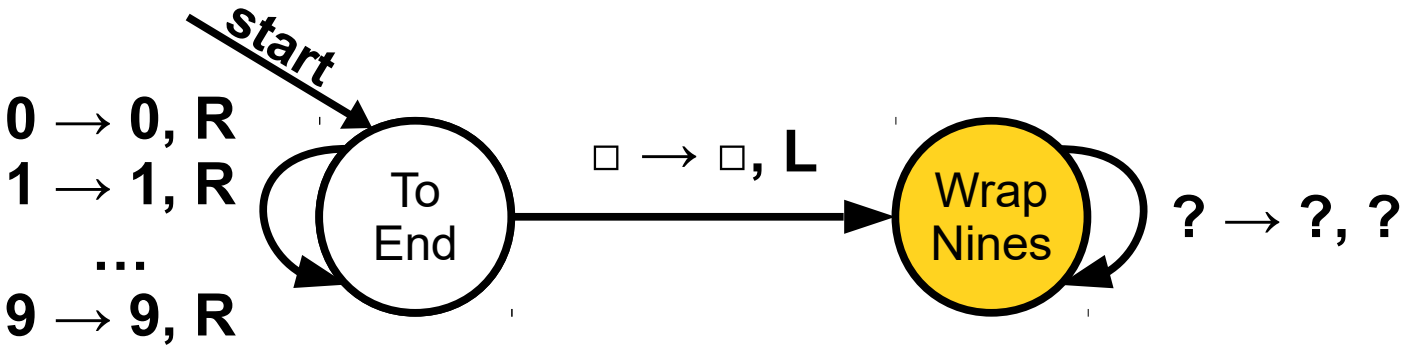


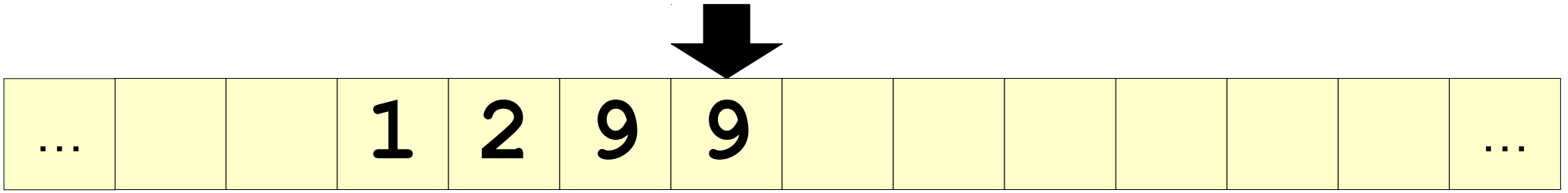
Based on we want this TM to do, what should this transition say?

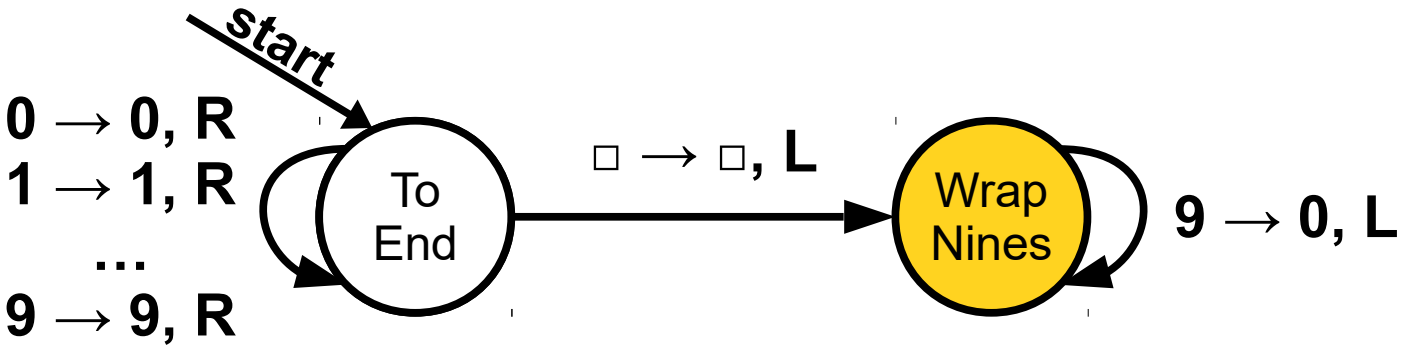
A. **0** → **9**, R
B. **0** → **9**, L
C. **9** → **0**, R
D. **9** → **0**, L
E. None of these, or two or more of these.

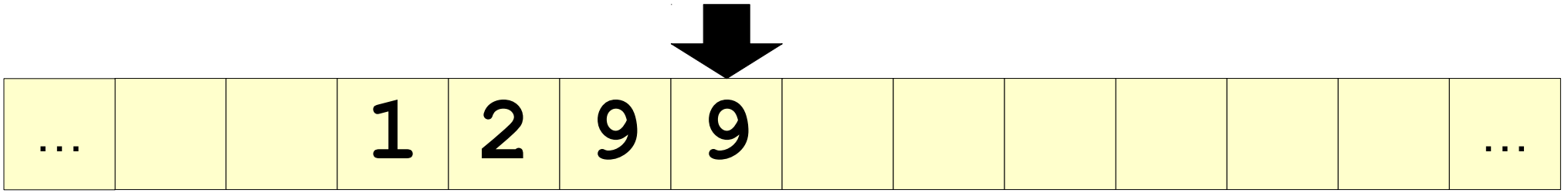


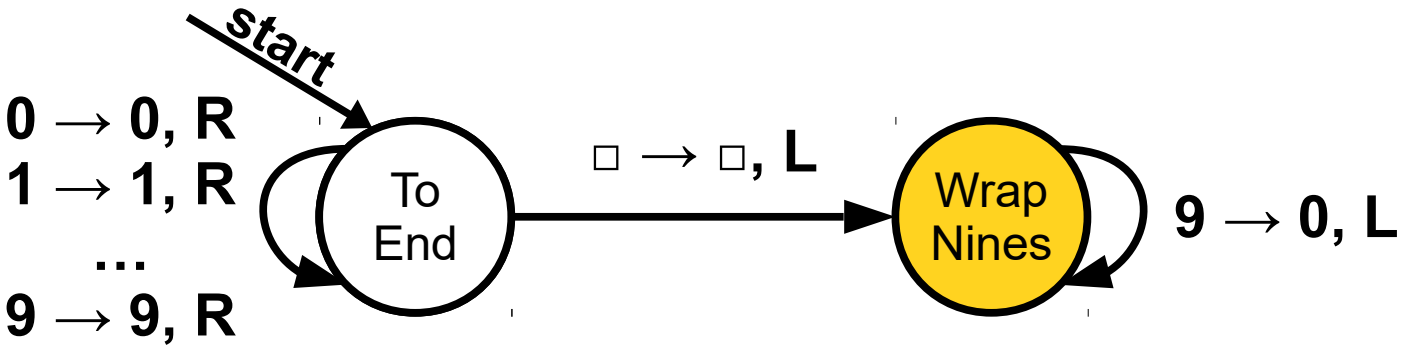
Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then ***A***, ***B***, ***C***, ***D***, or ***E***.

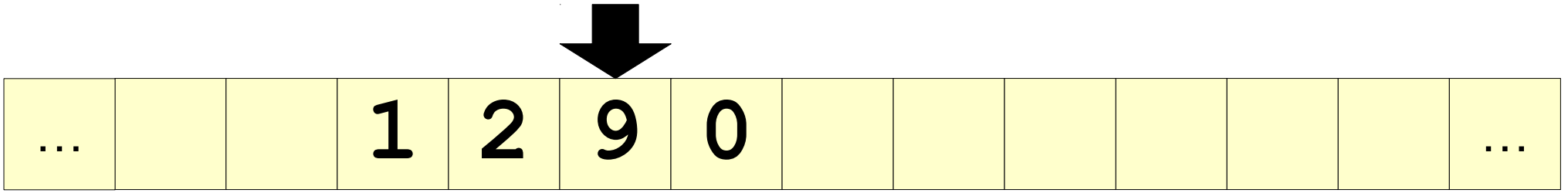

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To End
□ → □, L
Wrap Nines
? → ?, ?

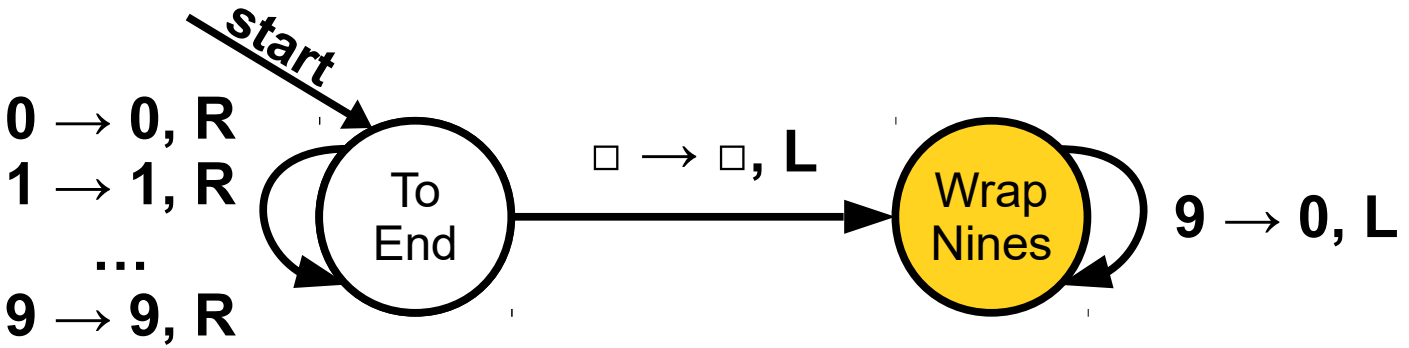

...
1
2
9
9
...

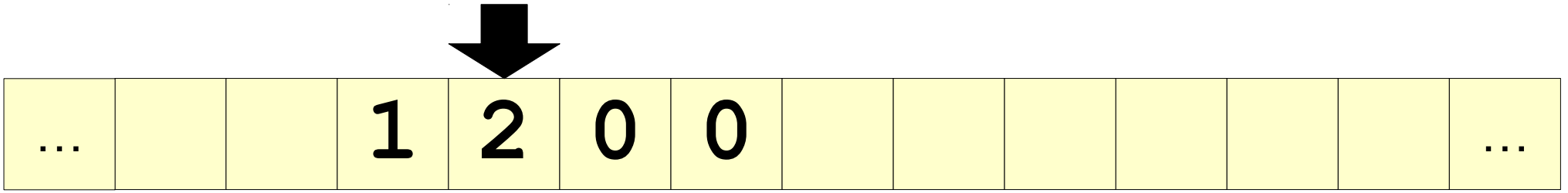

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L

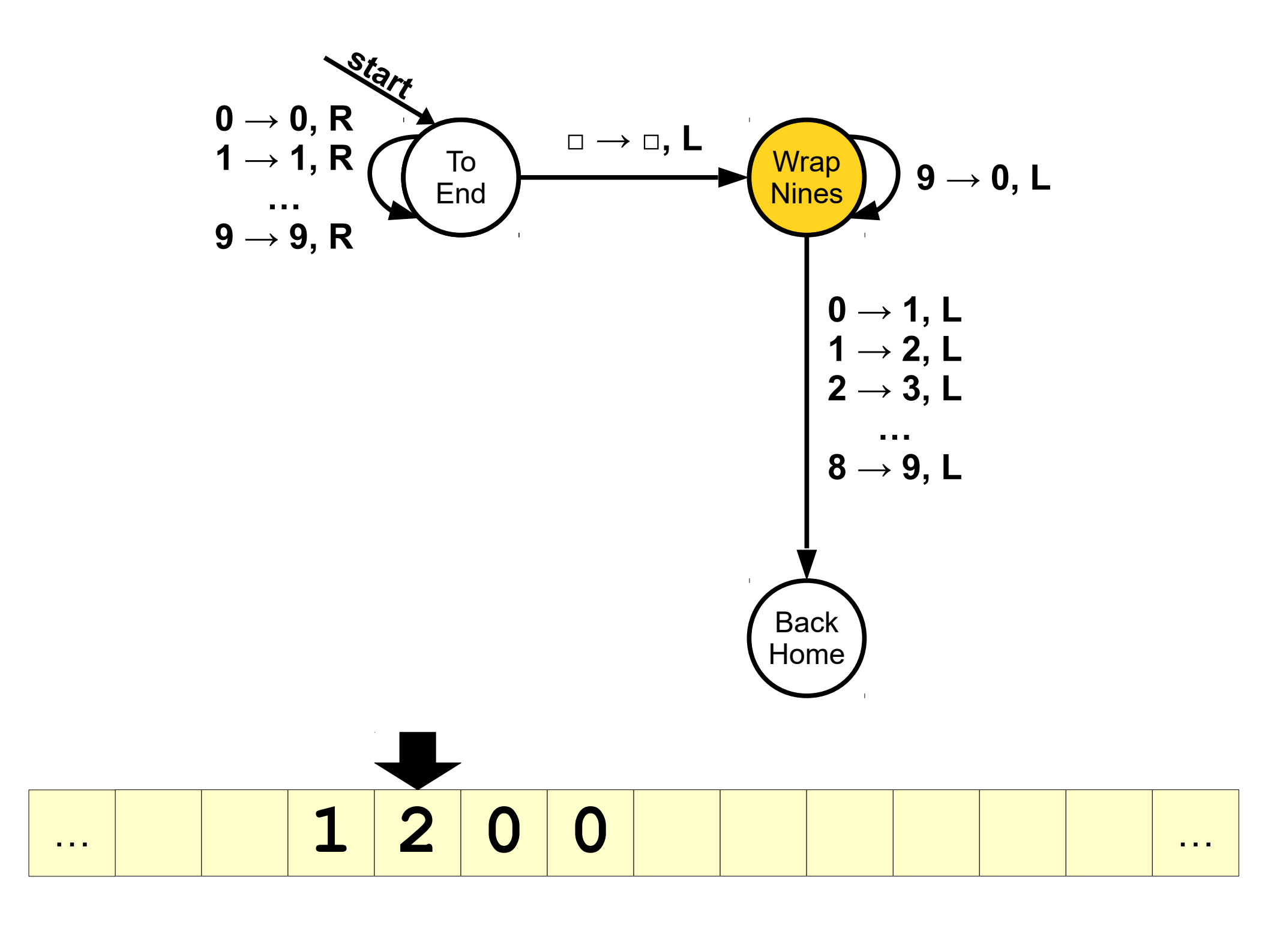

…
1
2
9
9
…

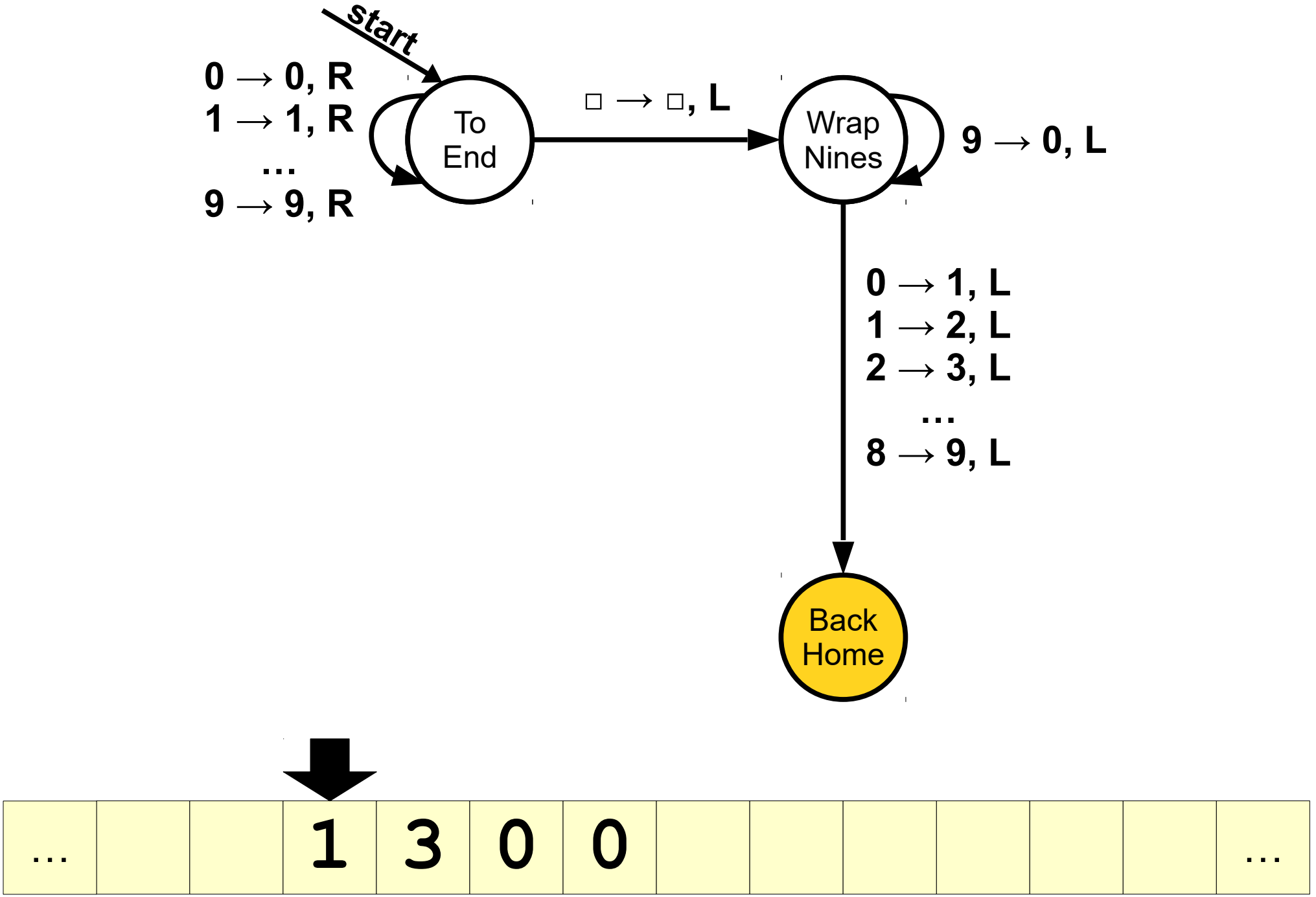

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L

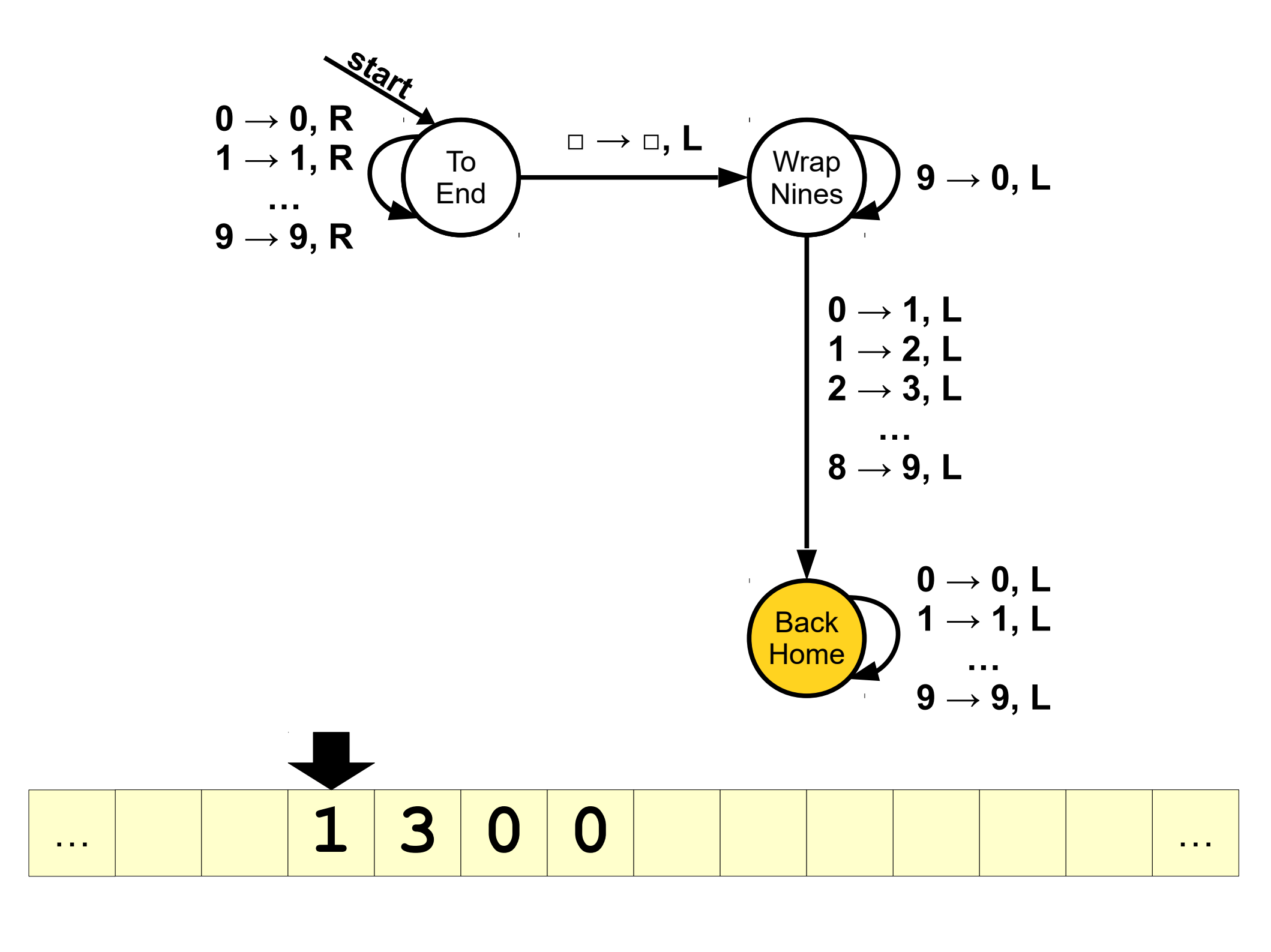

…
1
2
9
0
…

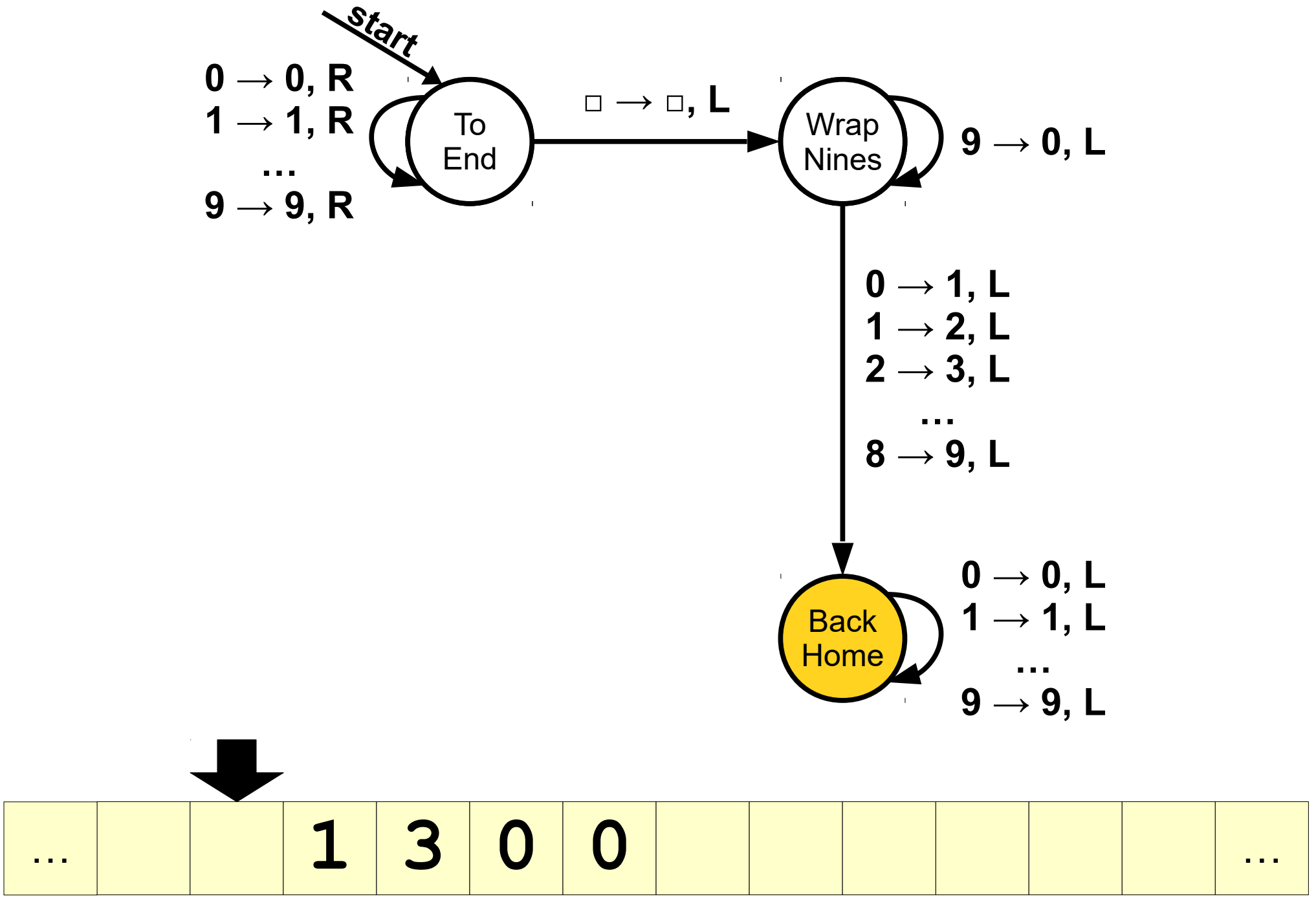

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L

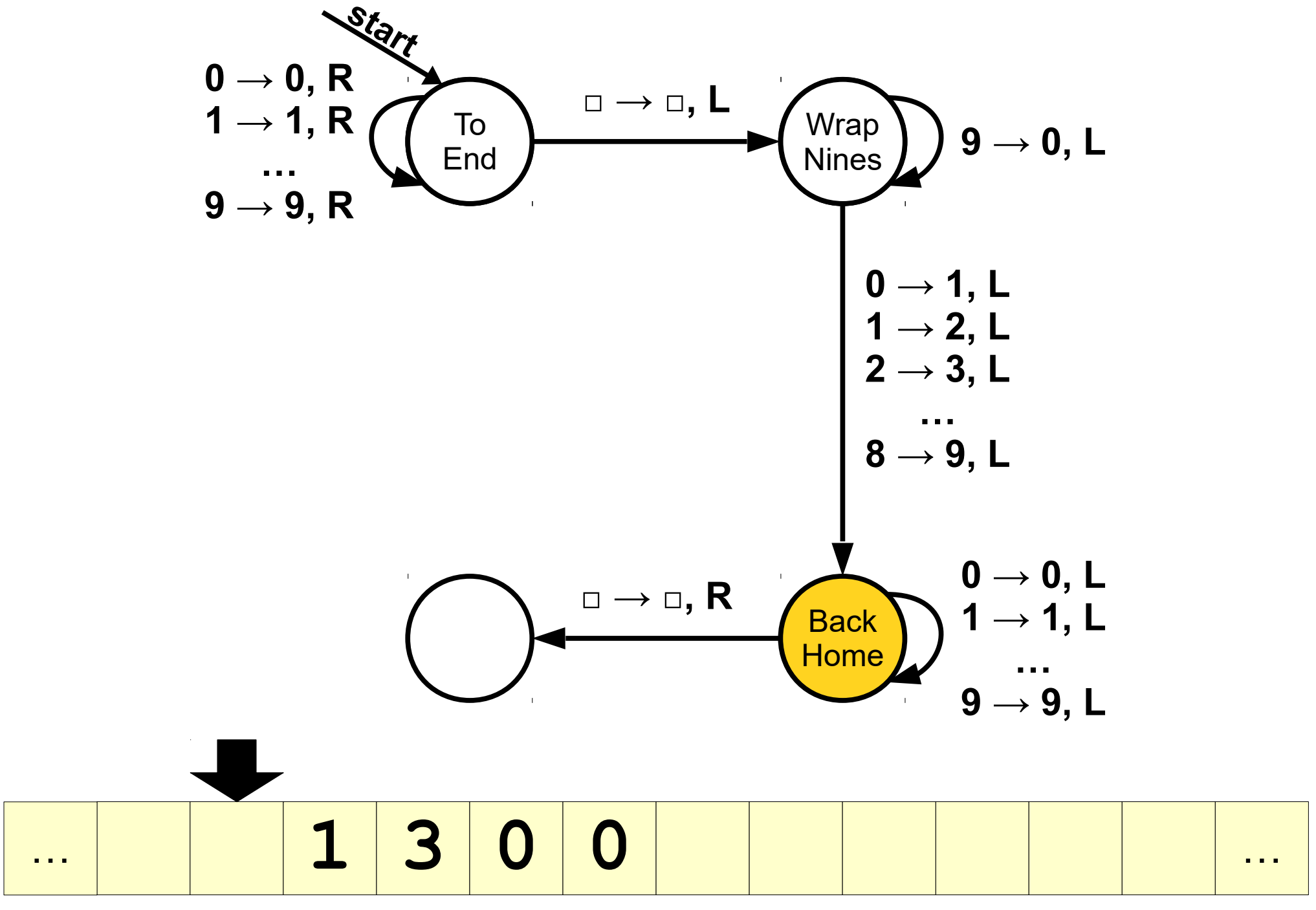

…
1
2
0
0
…

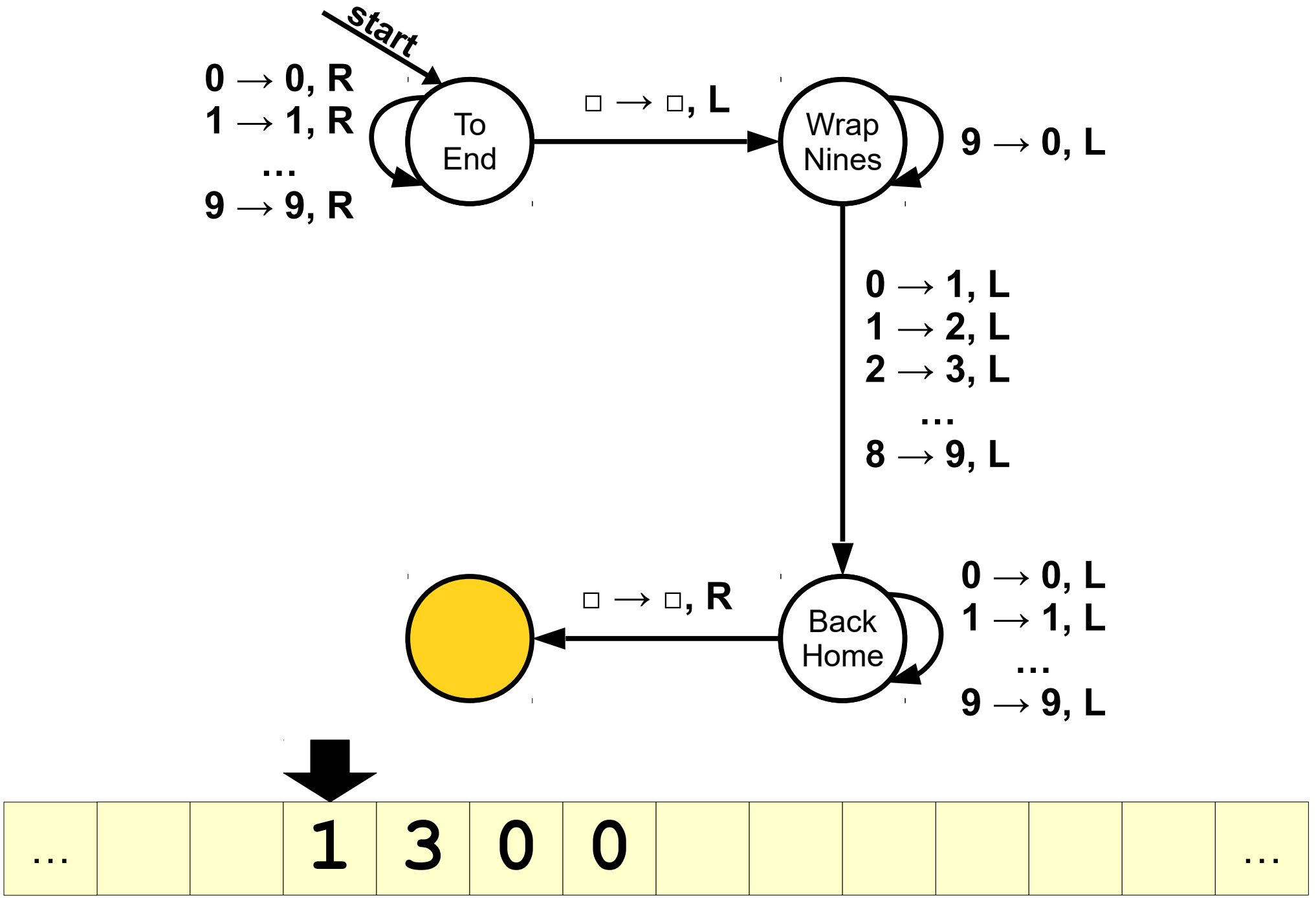

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
…
8 → 9, L
Back Home
…
1
2
0
0
…

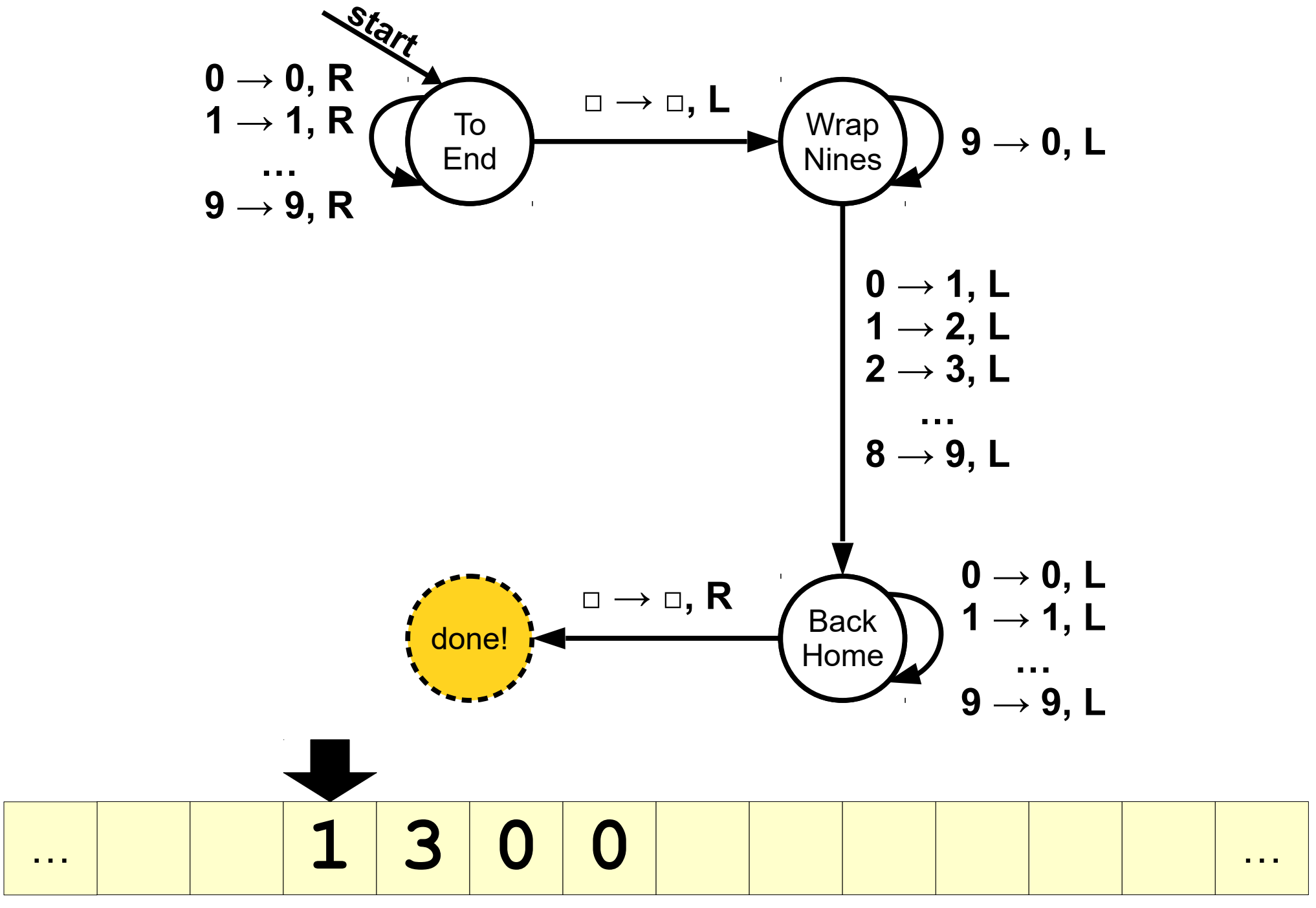

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
...
8 → 9, L
Back Home
...
1 3 0 0
...

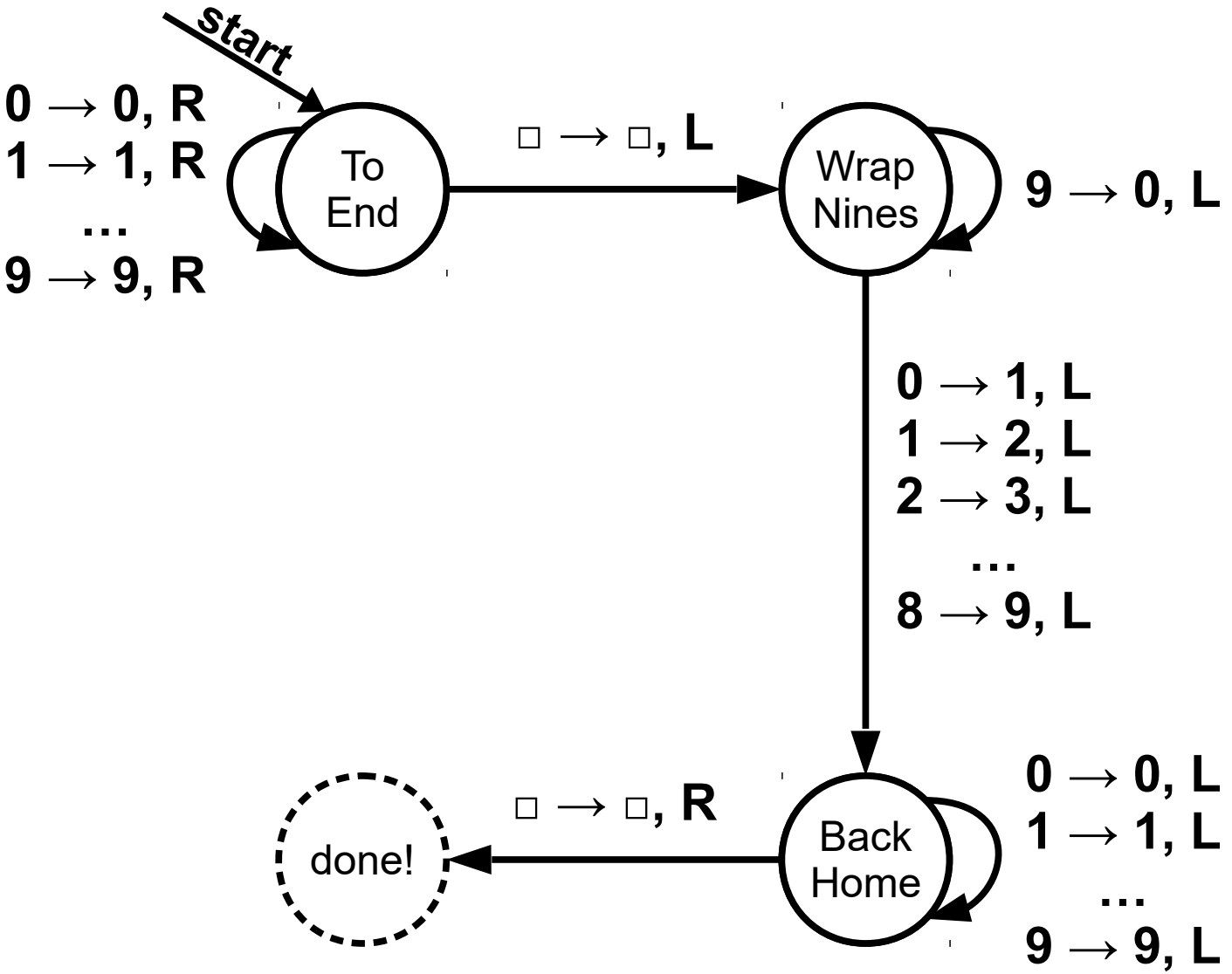

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
...
8 → 9, L
Back Home
0 → 0, L
1 → 1, L
...
9 → 9, L
...
1 3 0 0
...

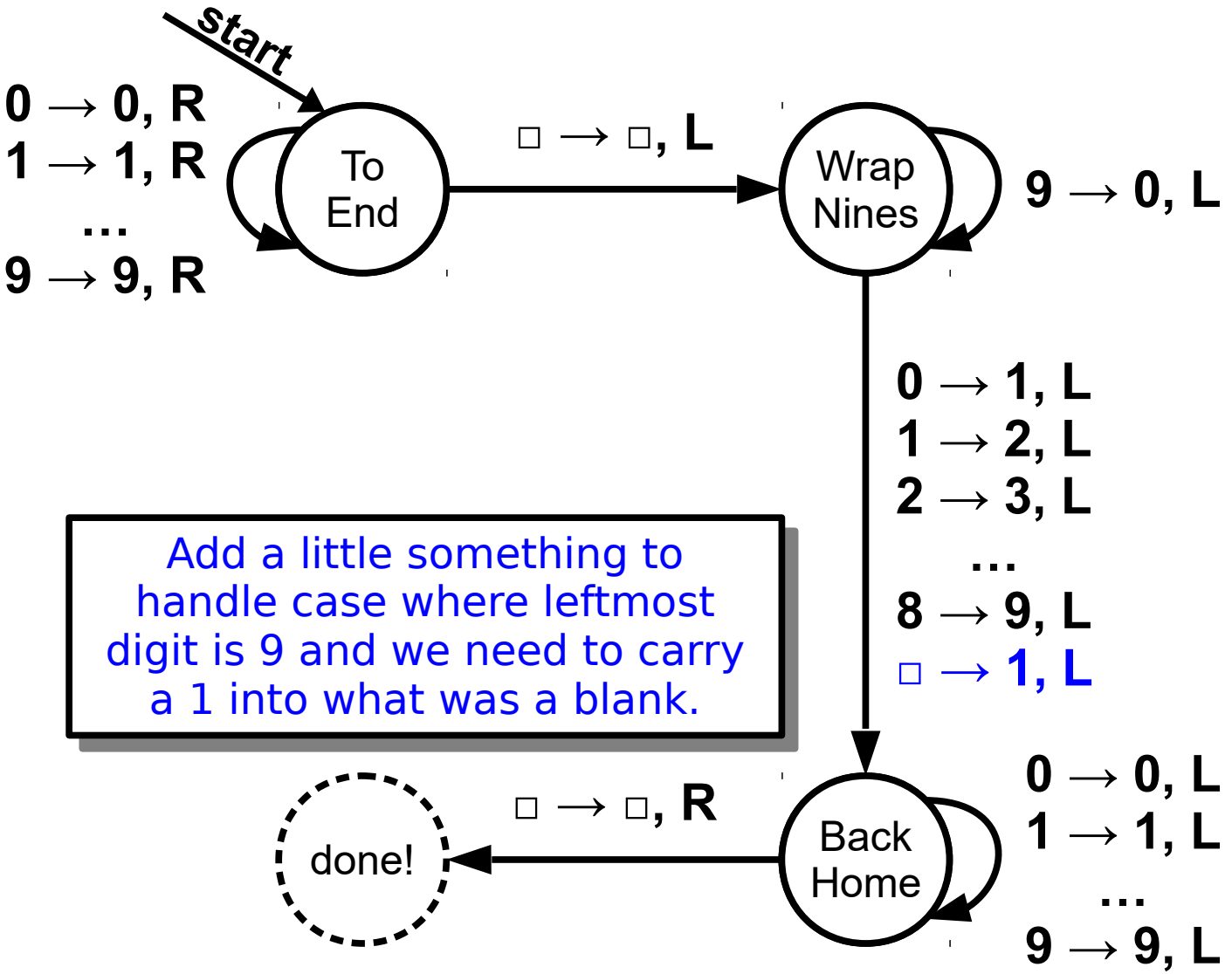

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
...
8 → 9, L
Back Home
0 → 0, L
1 → 1, L
...
9 → 9, L
...
1 3 0 0
...


start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
…
8 → 9, L
Back Home
□ → □, R
0 → 0, L
1 → 1, L
…
9 → 9, L
…
1 3 0 0
…


start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
…
8 → 9, L
Back Home
□ → □, R
0 → 0, L
1 → 1, L
…
9 → 9, L
… 1 3 0 0 …

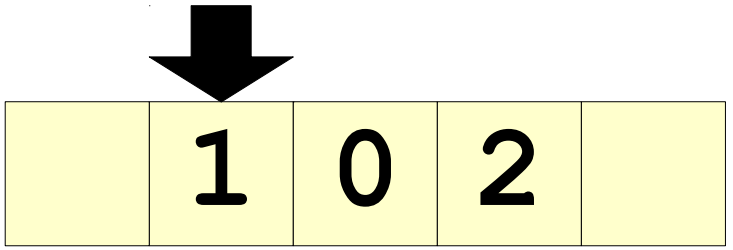

start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
…
8 → 9, L
Back Home
0 → 0, L
1 → 1, L
…
9 → 9, L
□ → □, R
done!
… 1 3 0 0 …


start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
…
8 → 9, L
Back Home
0 → 0, L
1 → 1, L
…
9 → 9, L
□ → □, R
done!


start
0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Nines
9 → 0, L
0 → 1, L
1 → 2, L
2 → 3, L
…
8 → 9, L
□ → 1, L
Add a little something to handle case where leftmost digit is 9 and we need to carry a 1 into what was a blank.
Back Home
0 → 0, L
1 → 1, L
…
9 → 9, L
□ → □, R
done!

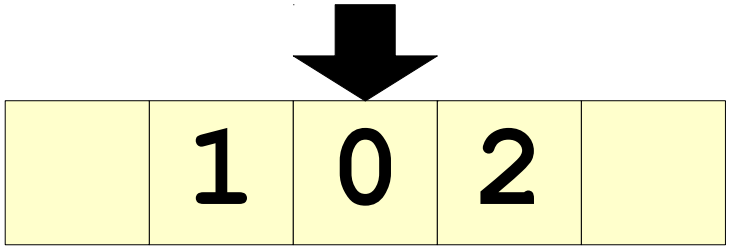
Decrementing Numbers

- 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.



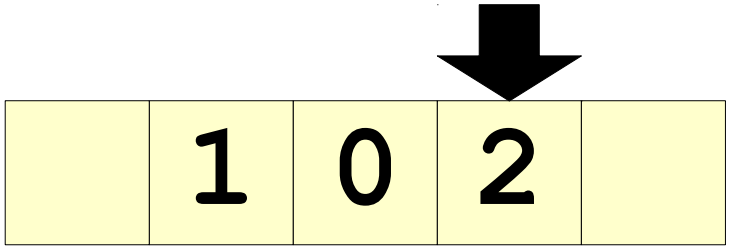
Decrementing Numbers

- 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.



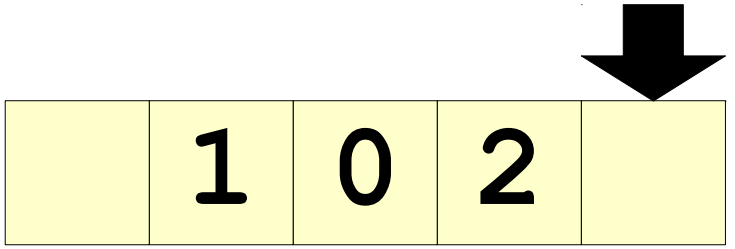
Decrementing Numbers

- 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.



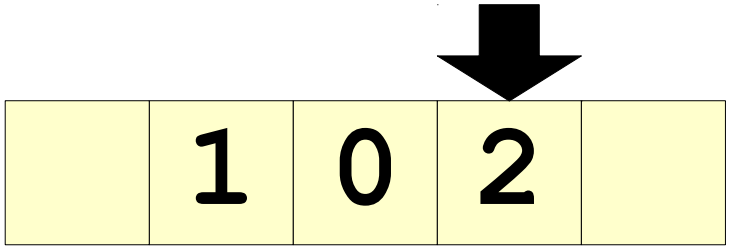
Decrementing Numbers

- 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.



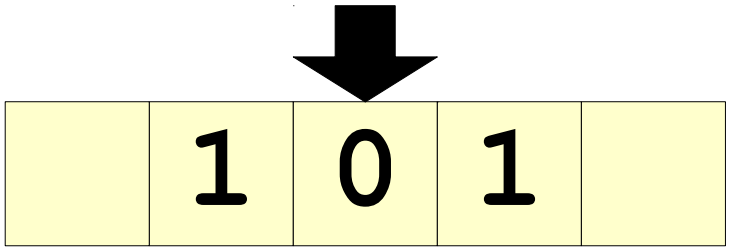
Decrementing Numbers

- 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.



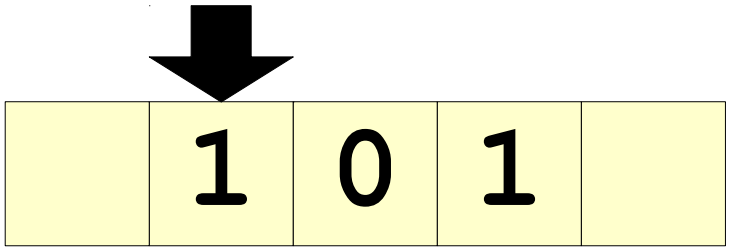
Decrementing Numbers

- 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.



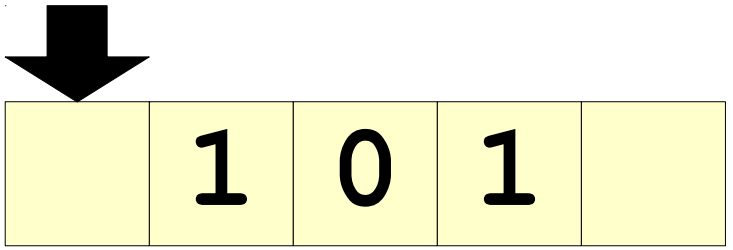
Decrementing Numbers

- 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.



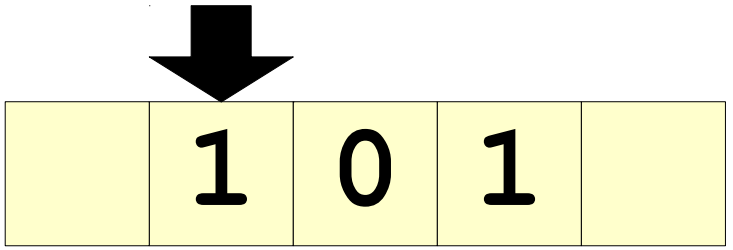
Decrementing Numbers

- 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.



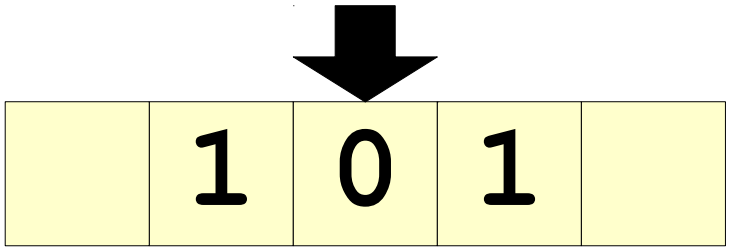
Decrementing Numbers

- 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.



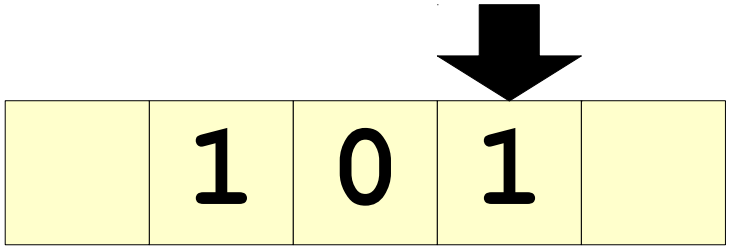
Decrementing Numbers

- 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.



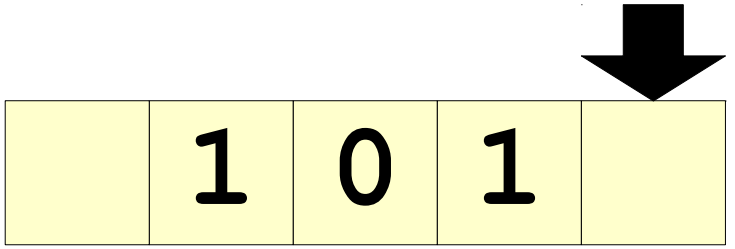
Decrementing Numbers

- 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.



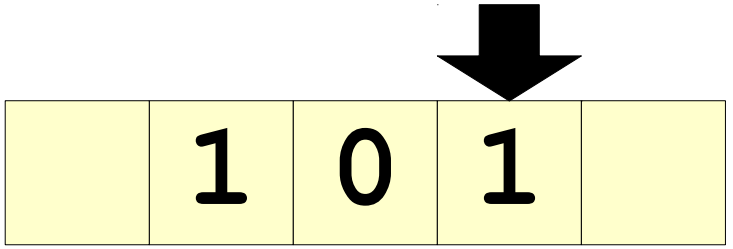
Decrementing Numbers

- 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.



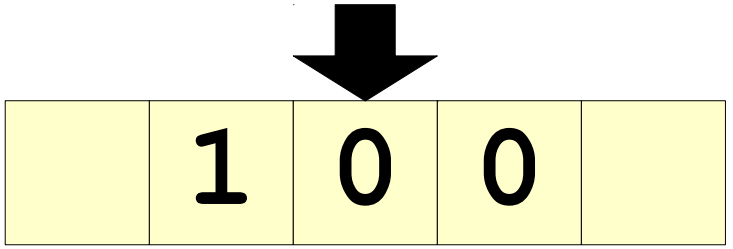
Decrementing Numbers

- 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.



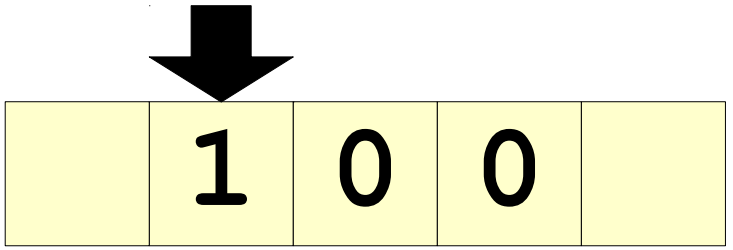
Decrementing Numbers

- 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.



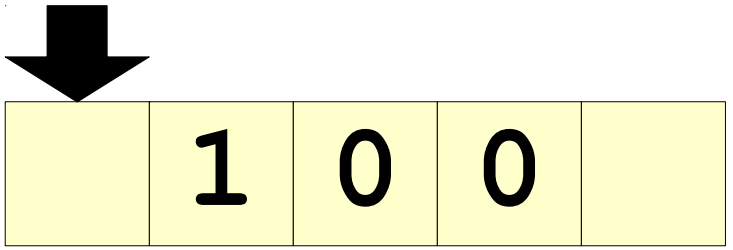
Decrementing Numbers

- 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.



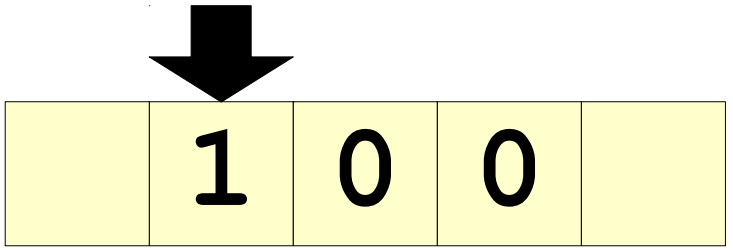
Decrementing Numbers

- 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.



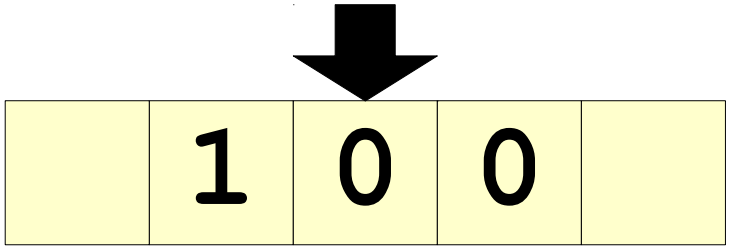
Decrementing Numbers

- 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.



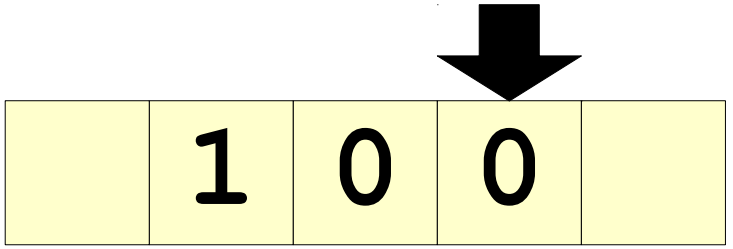
Decrementing Numbers

- 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.



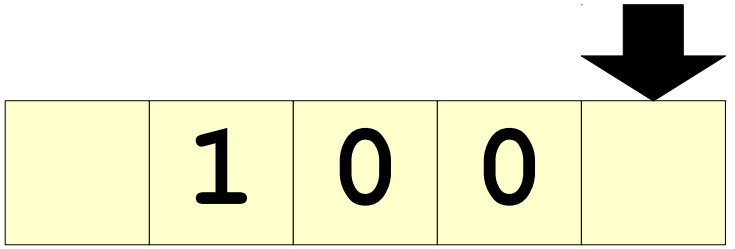
Decrementing Numbers

- 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.



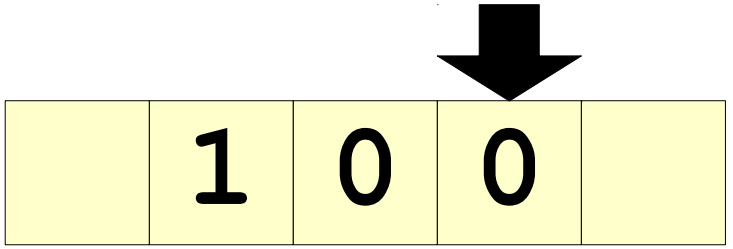
Decrementing Numbers

- 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.



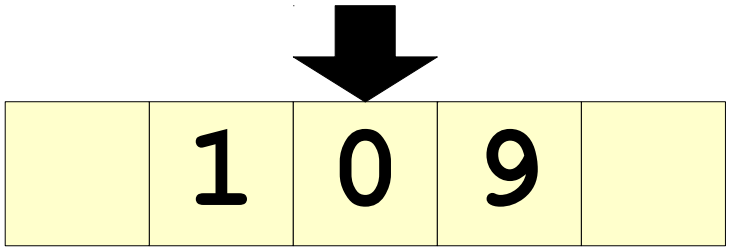
Decrementing Numbers

- 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.



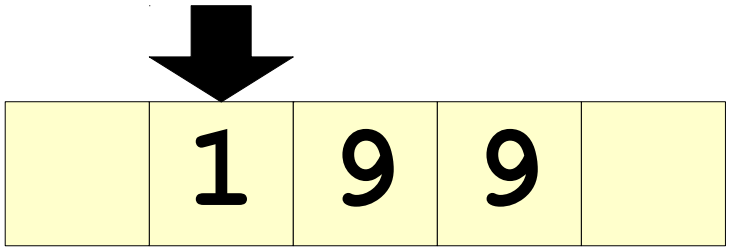
Decrementing Numbers

- 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.



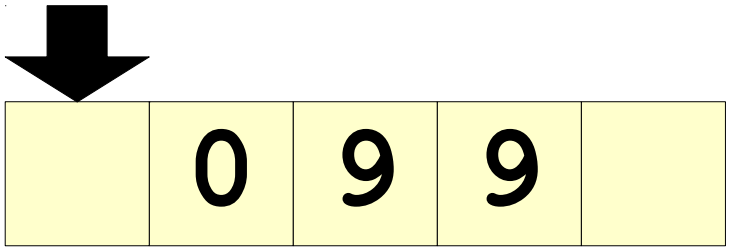
Decrementing Numbers

- 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.



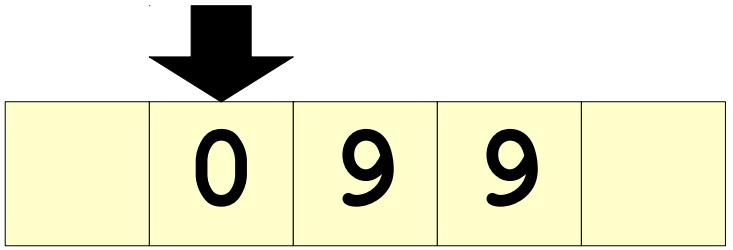
Decrementing Numbers

- 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.



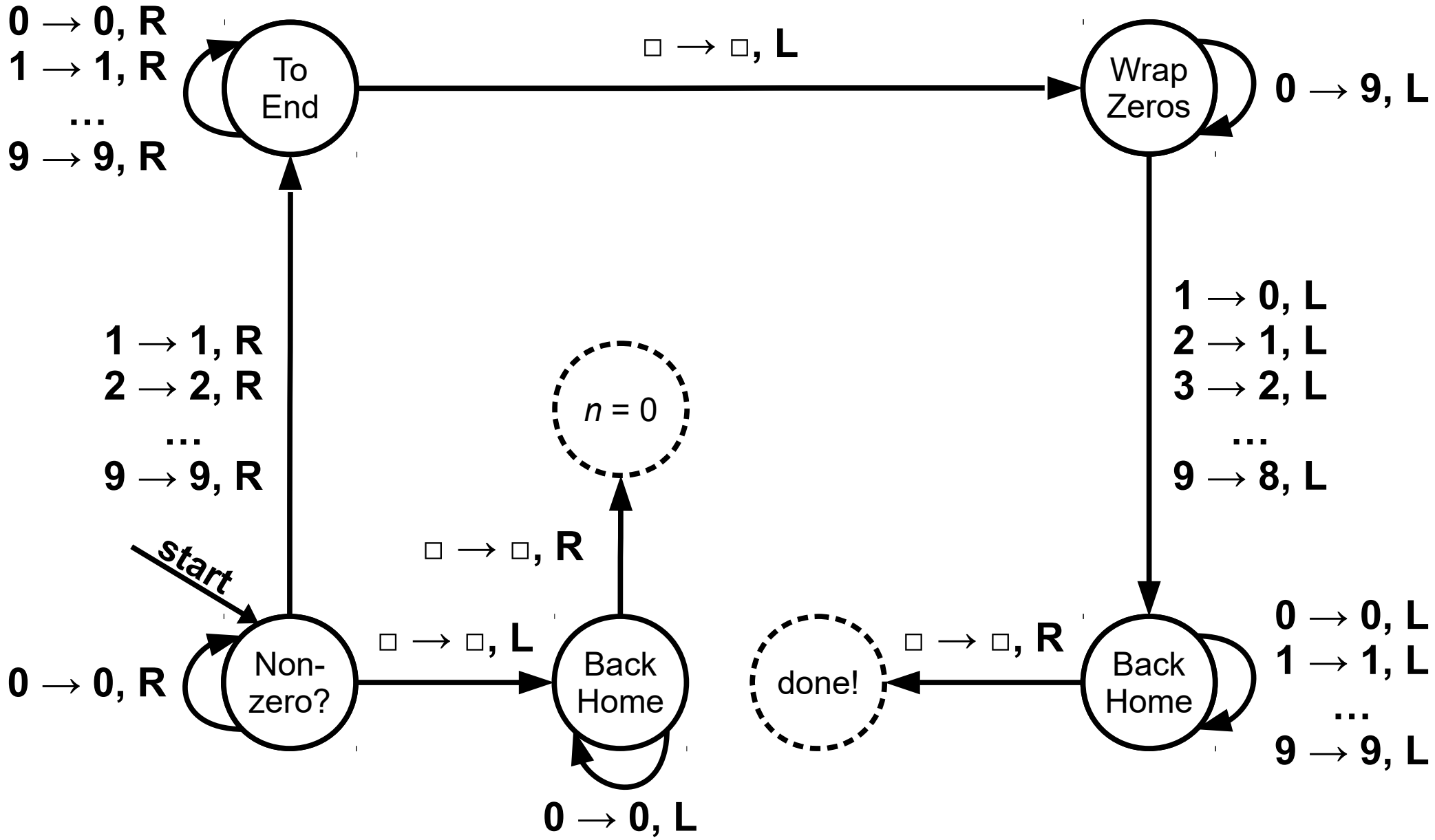
Decrementing Numbers

- 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;
```

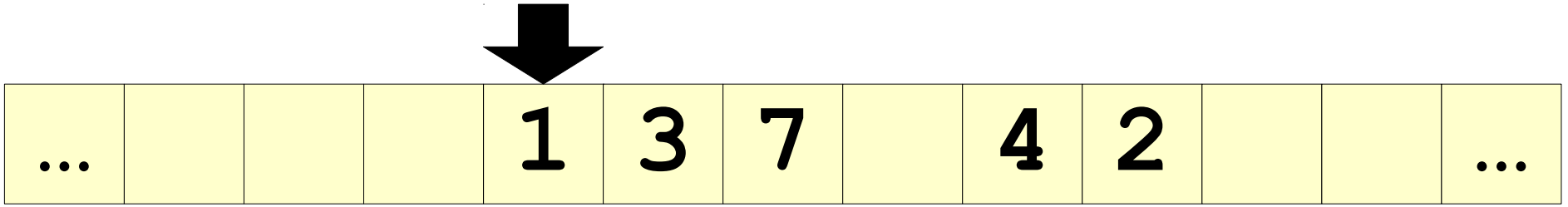

0 → 0, R
1 → 1, R
…
9 → 9, R
To End
□ → □, L
Wrap Zeros
0 → 9, L
1 → 0, L
2 → 1, L
3 → 2, L
…
9 → 8, L
1 → 1, R
2 → 2, R
…
9 → 9, R
n = 0
□ → □, R
start
□ → □, L
Non-zero?
0 → 0, R
Back Home
0 → 0, L
□ → □, R
done!
□ → □, R
Back Home
0 → 0, L
1 → 1, L
…
9 → 9, L

TM Subroutines

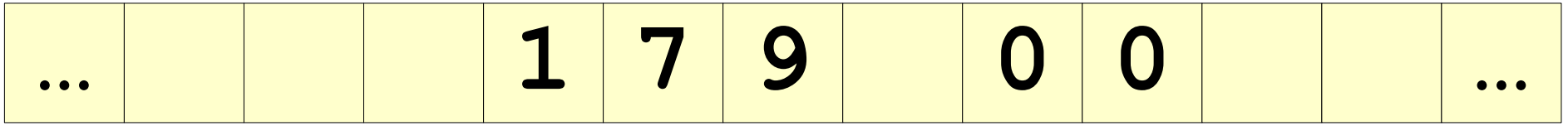
- Sometimes, a subroutine needs to report back some information about what happened.
- Just as a function can return multiple different values, we'll allow subroutines to have different “done” states.
- Each state can then be wired to a different state, so a TM using the subroutine can control what happens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

Putting it All Together

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

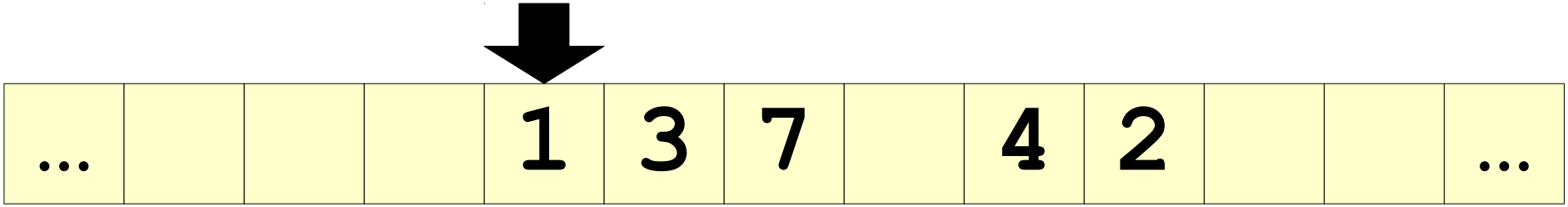


- After:



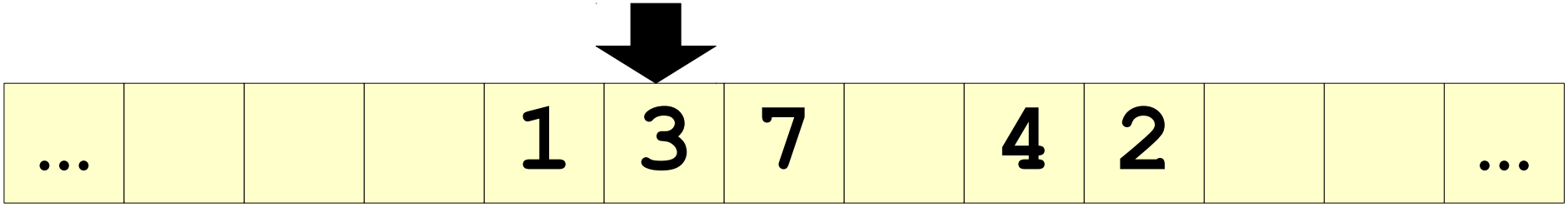
Using Our Subroutines

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



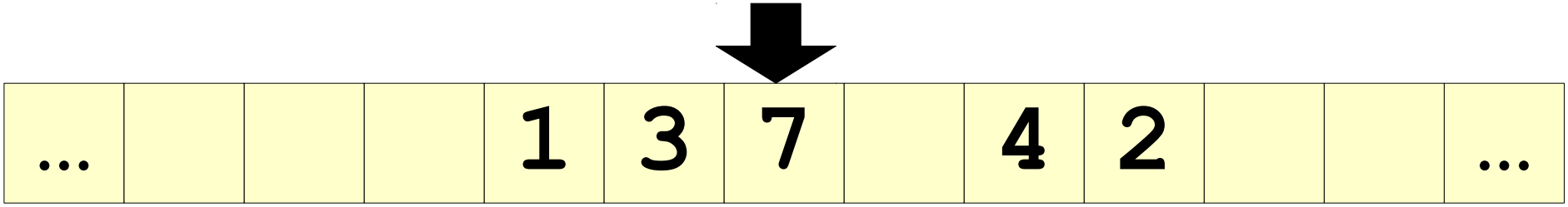
Using Our Subroutines

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



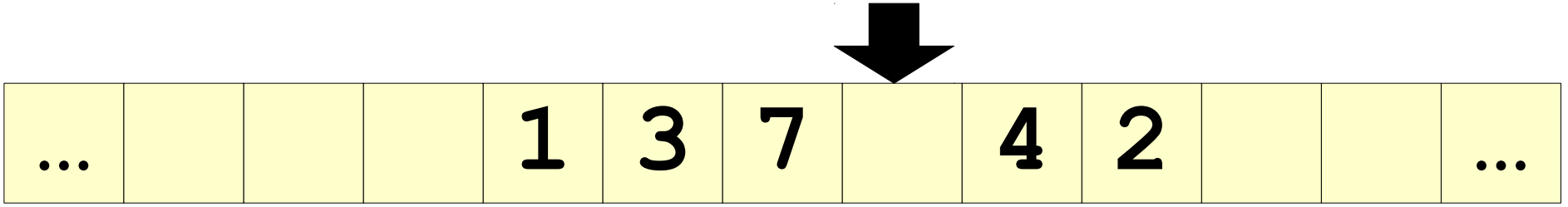
Using Our Subroutines

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



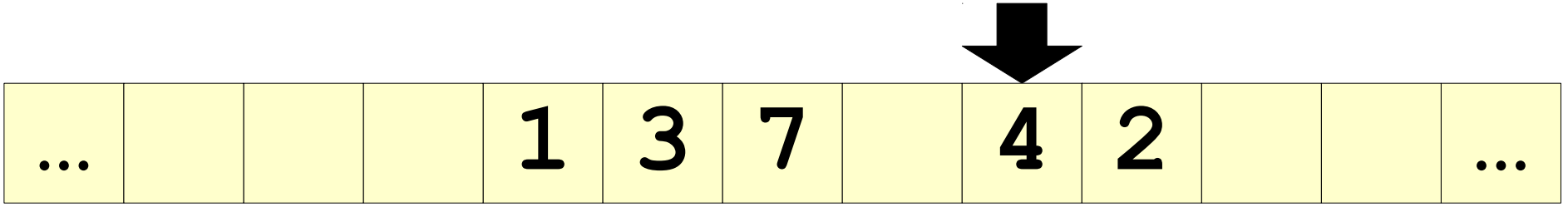
Using Our Subroutines

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



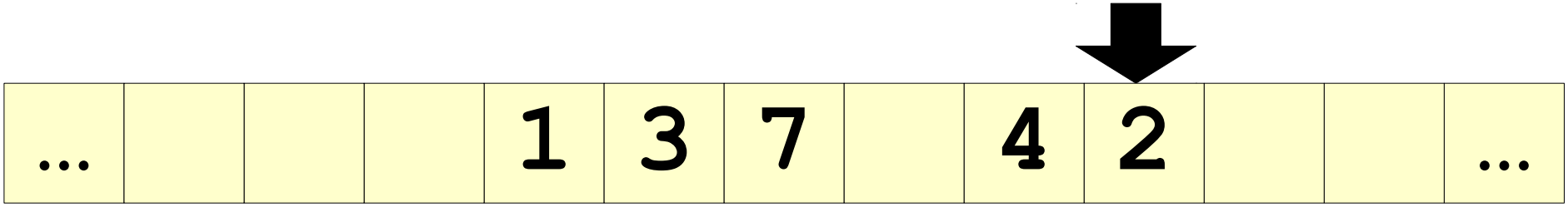
Using Our Subroutines

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



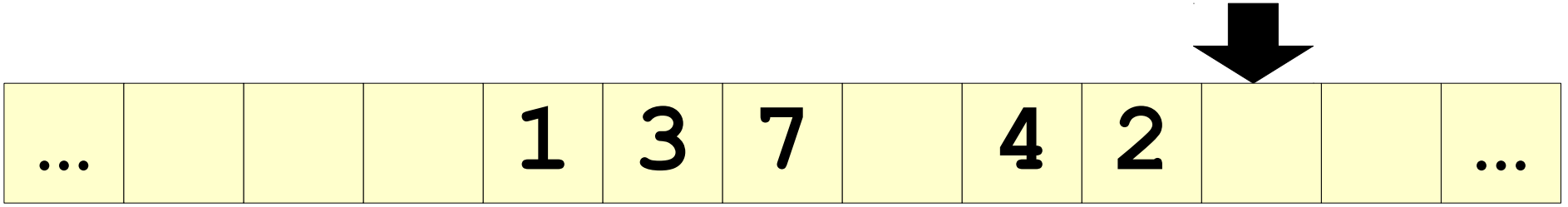
Using Our Subroutines

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



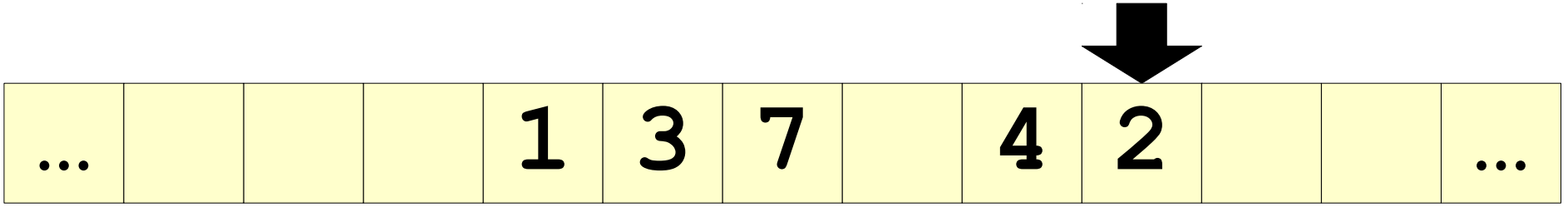
Using Our Subroutines

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



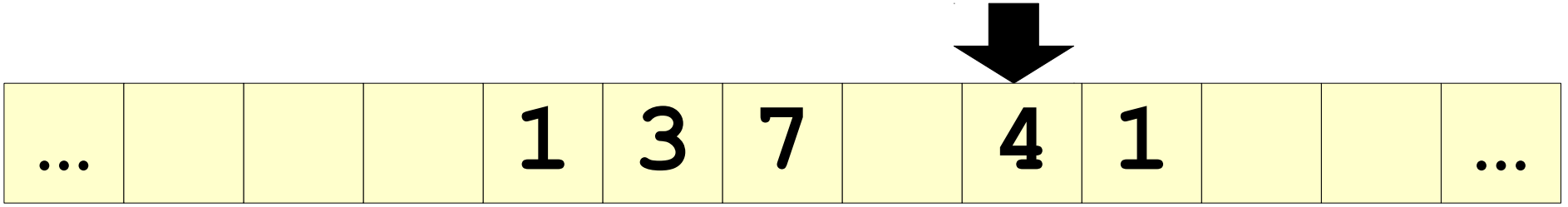
Using Our Subroutines

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



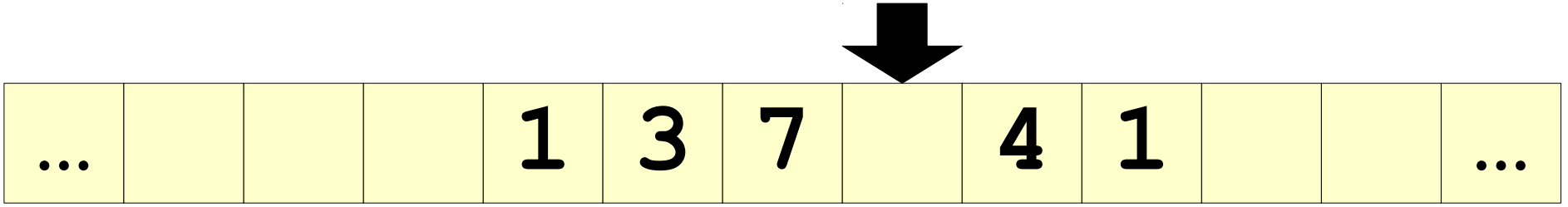
Using Our Subroutines

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



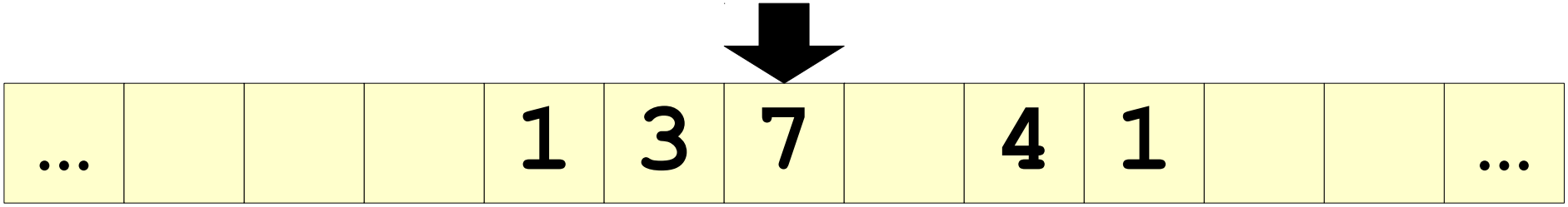
Using Our Subroutines

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



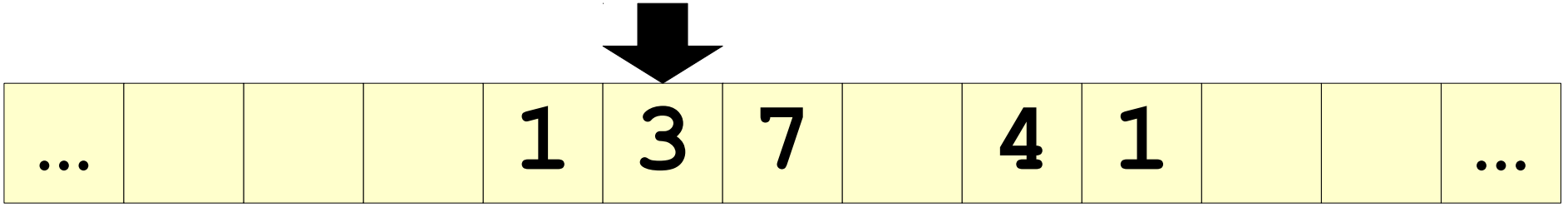
Using Our Subroutines

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



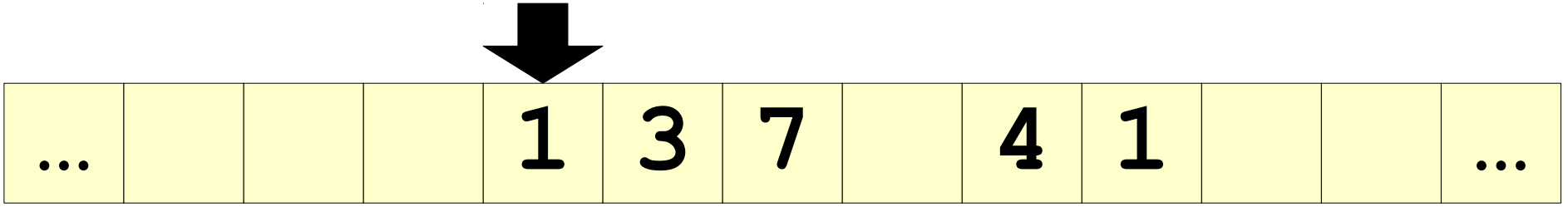
Using Our Subroutines

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



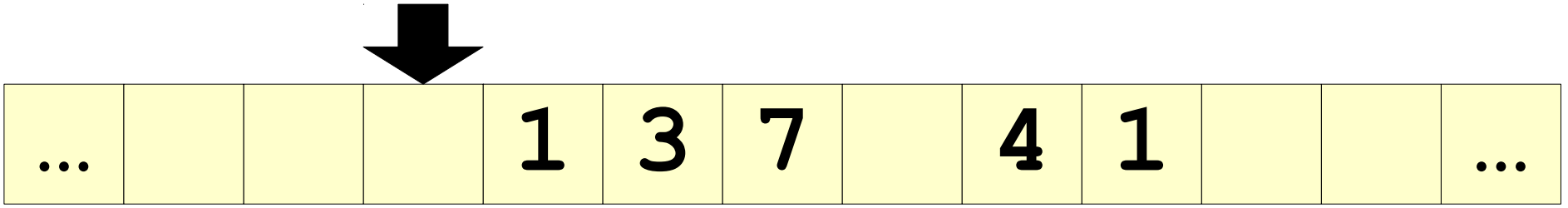
Using Our Subroutines

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



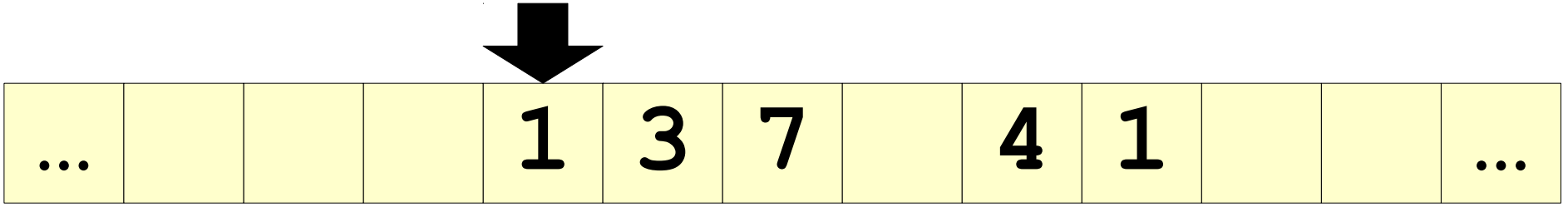
Using Our Subroutines

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



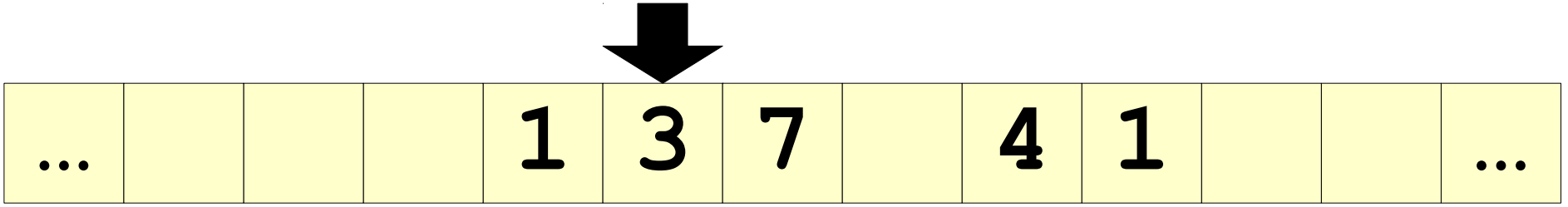
Using Our Subroutines

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



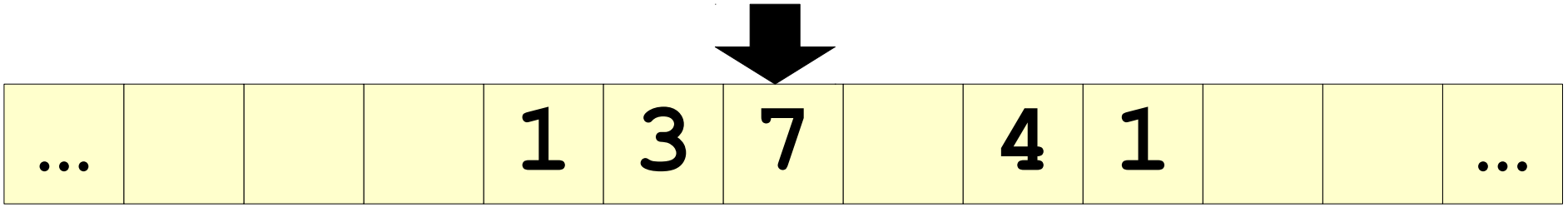
Using Our Subroutines

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



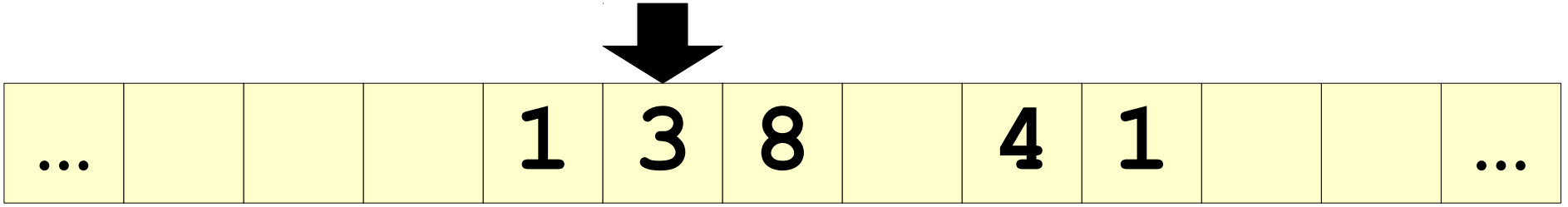
Using Our Subroutines

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



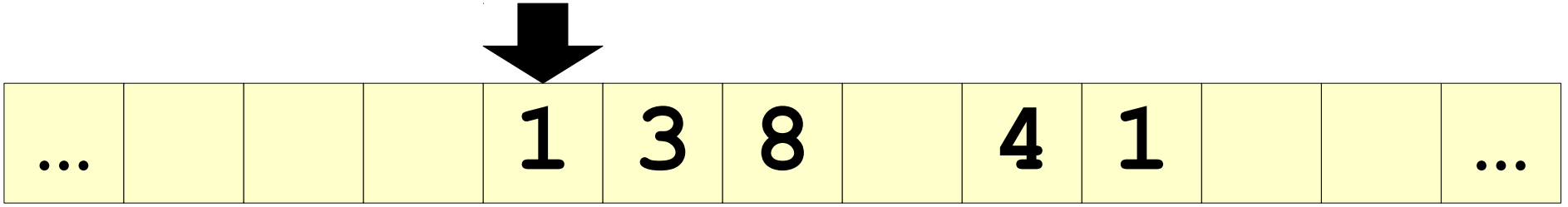
Using Our Subroutines

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



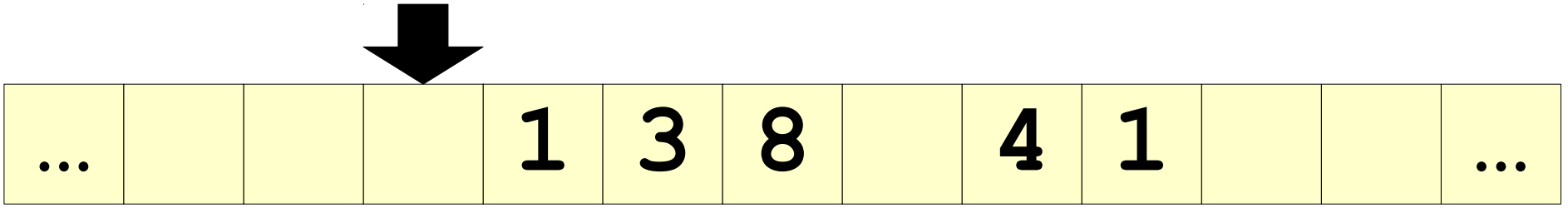
Using Our Subroutines

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



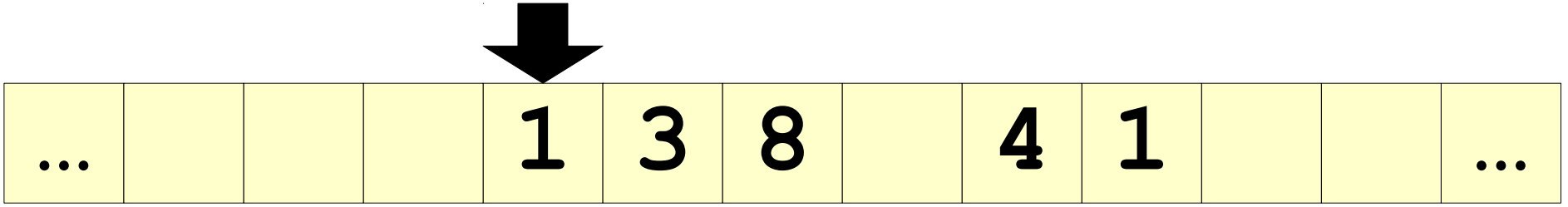
Using Our Subroutines

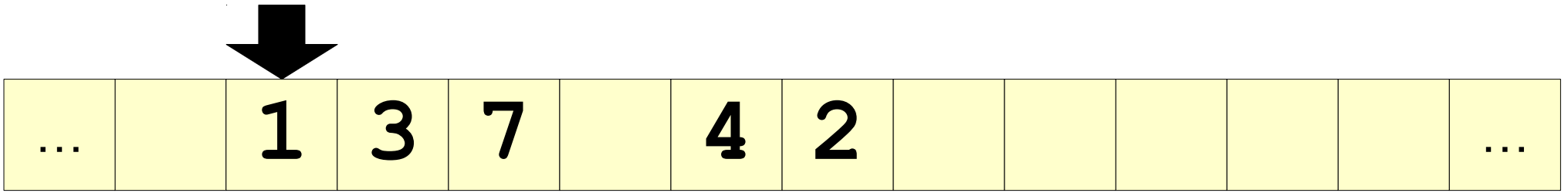
- We'll build our new machine using our existing increment and decrement subroutines:

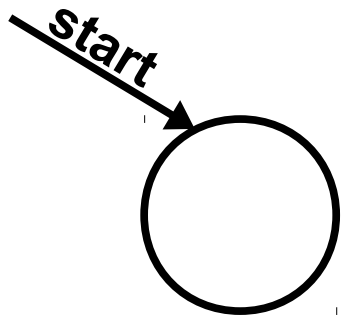


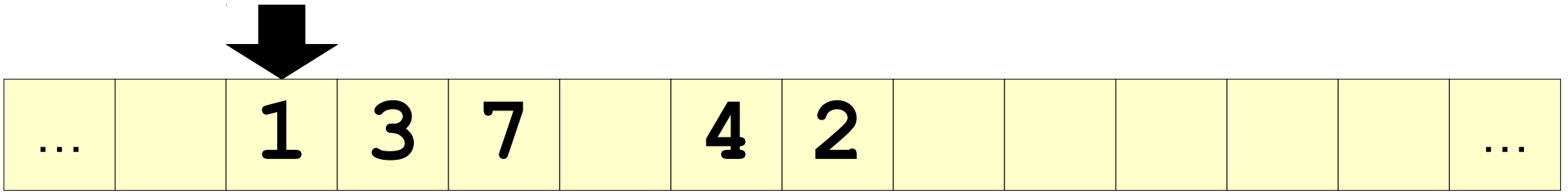
Using Our Subroutines

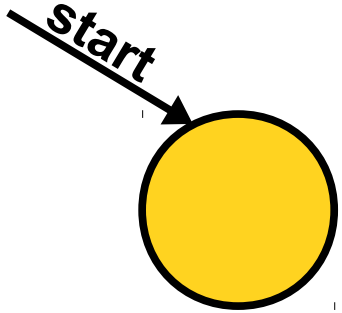
- We'll build our new machine using our existing increment and decrement subroutines:

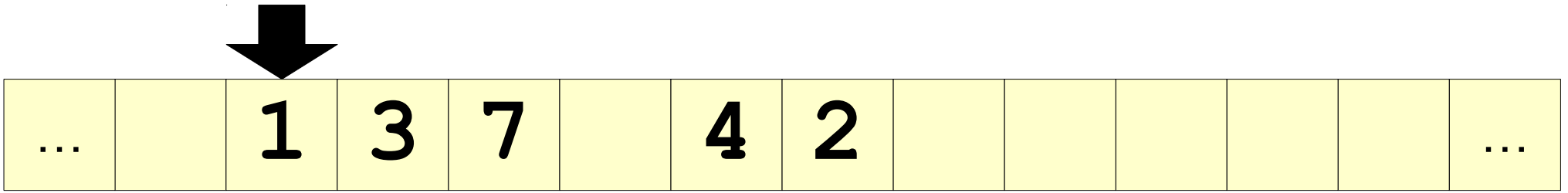


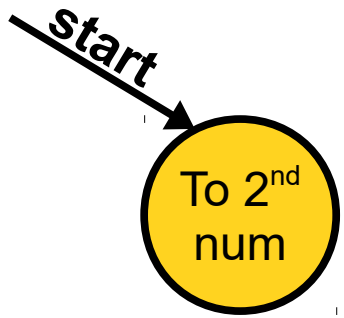

…
1
3
7
4
2
…

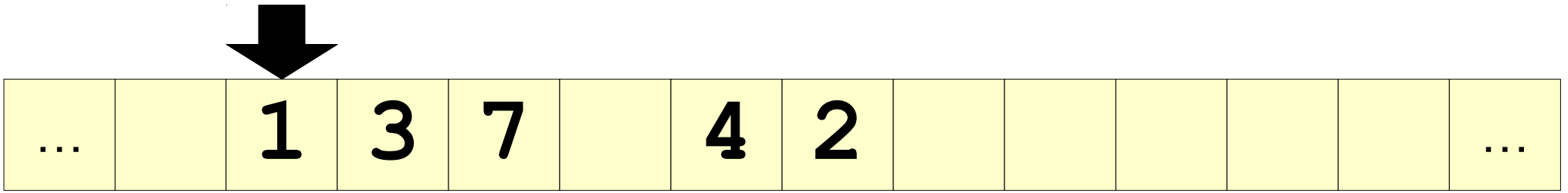

start

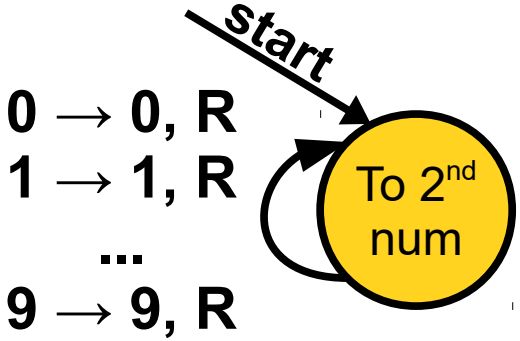

…
1
3
7
4
2
…

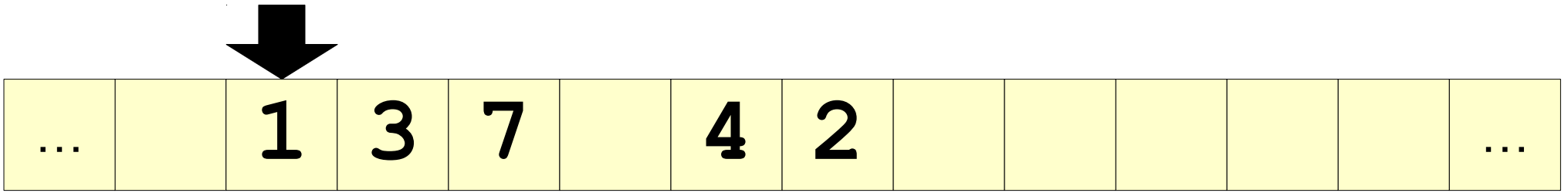

start

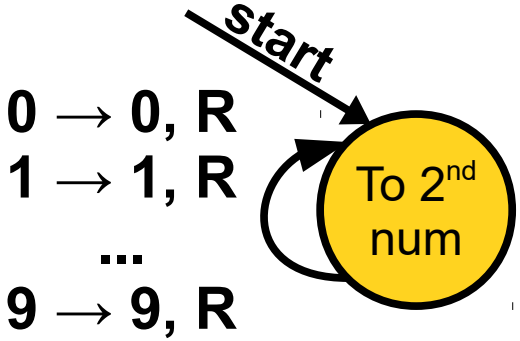

…
1
3
7
4
2
…

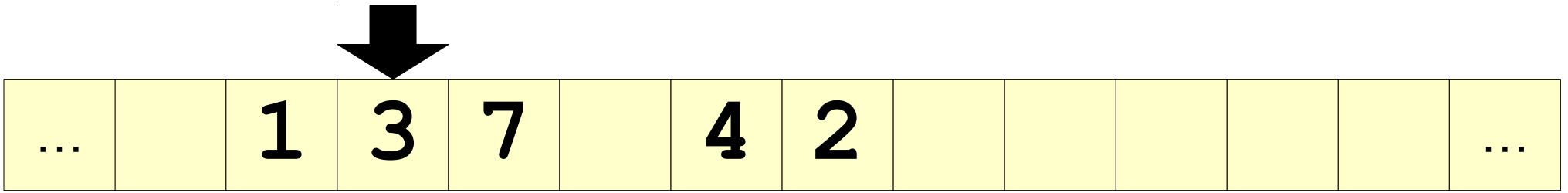

start
To 2nd num

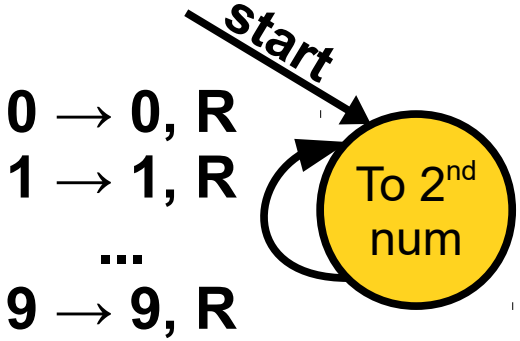

…
1
3
7
4
2
…

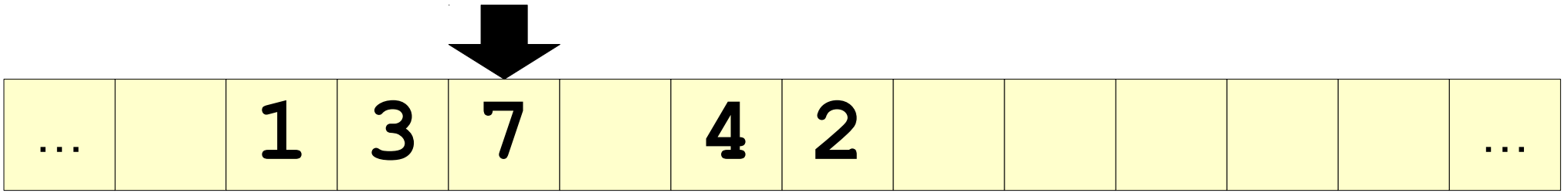

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num

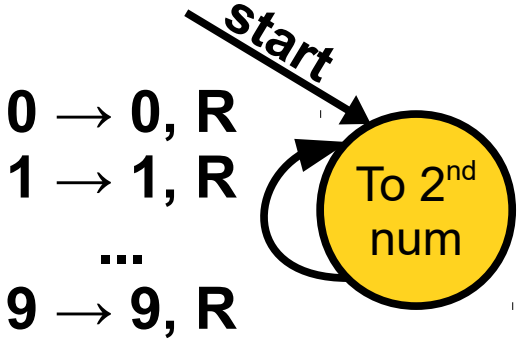

…
1
3
7
4
2
…

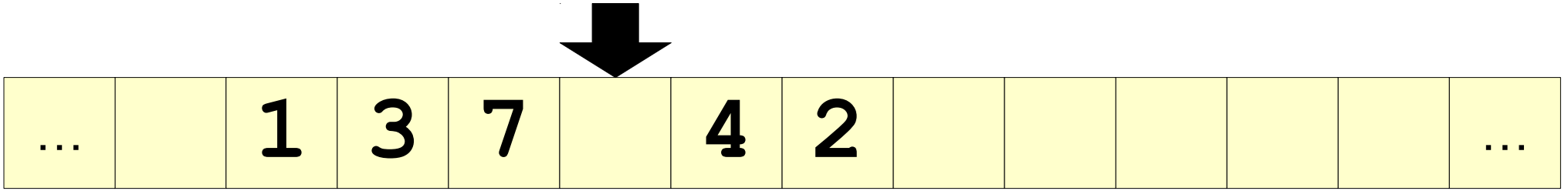

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num

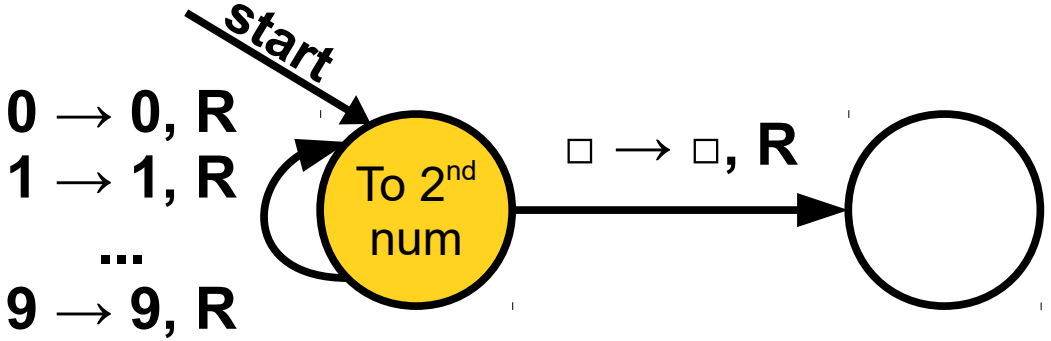

…
1
3
7
4
2
…

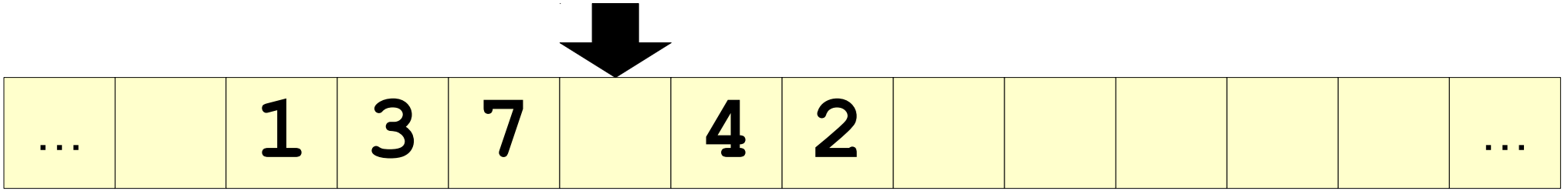

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num

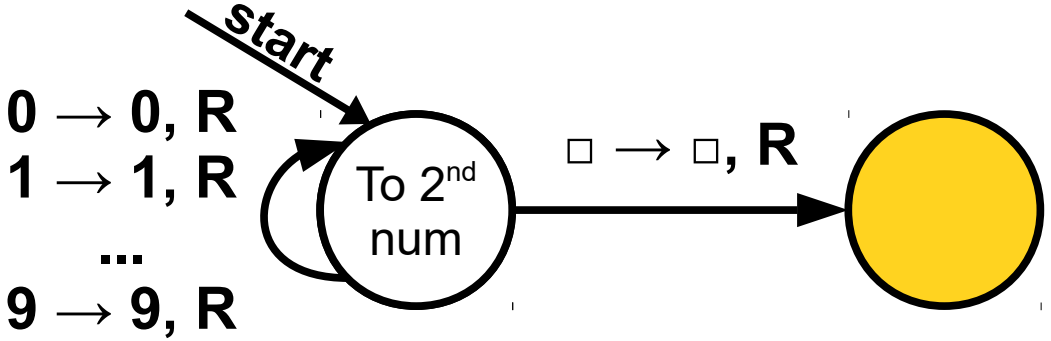

…
1
3
7
4
2
…

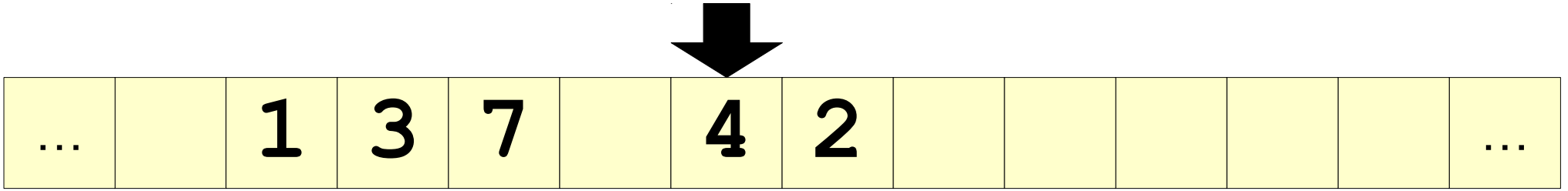

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num

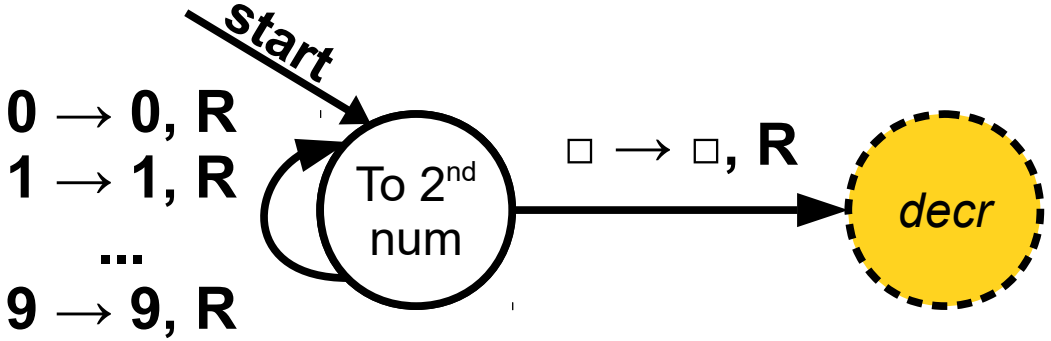

…
1
3
7
4
2
…

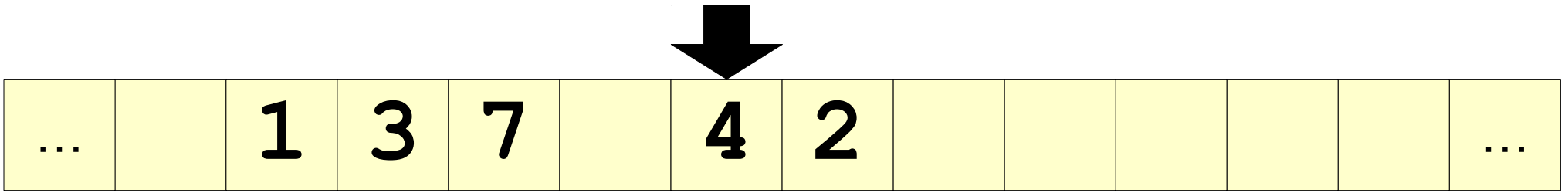

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R

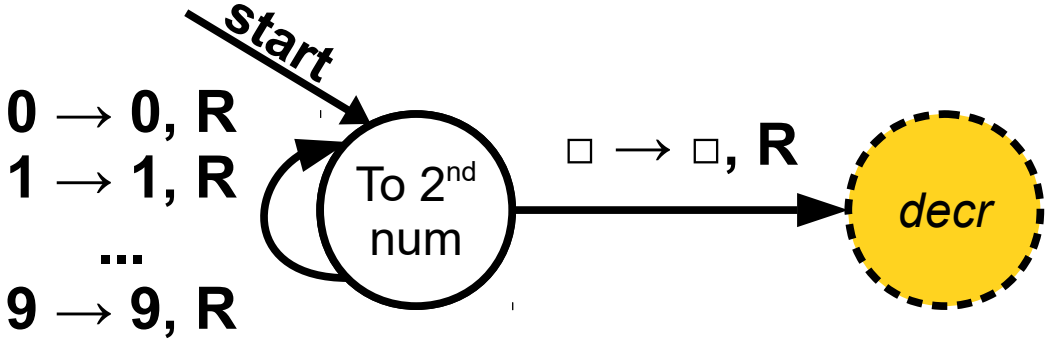

…
1
3
7
4
2
…

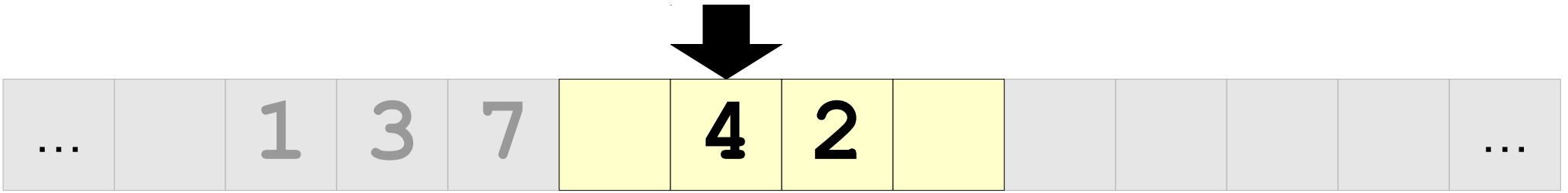

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R

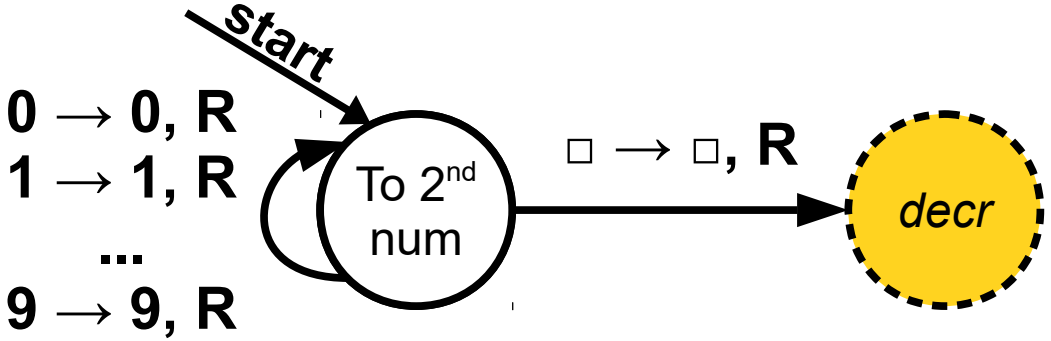

…
1
3
7
4
2
…

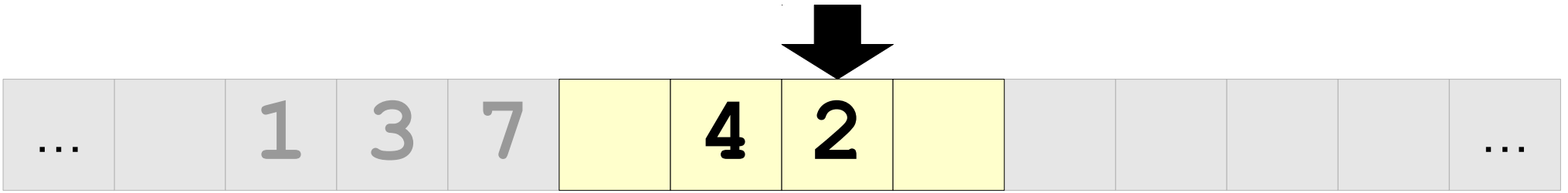

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

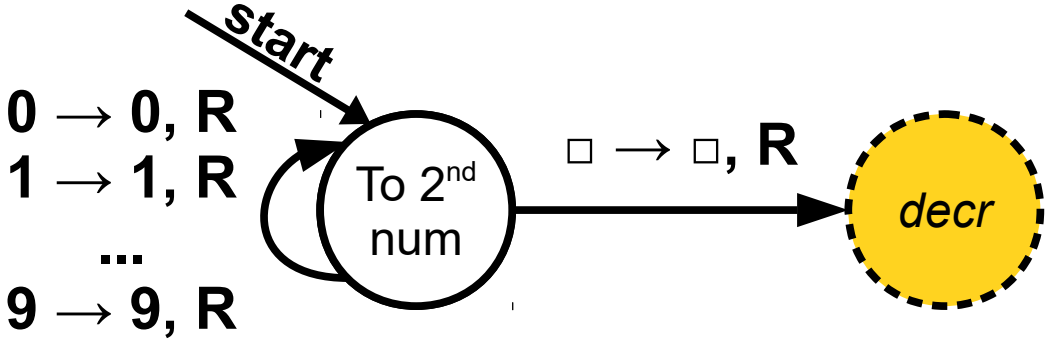

…
1
3
7
4
2
…

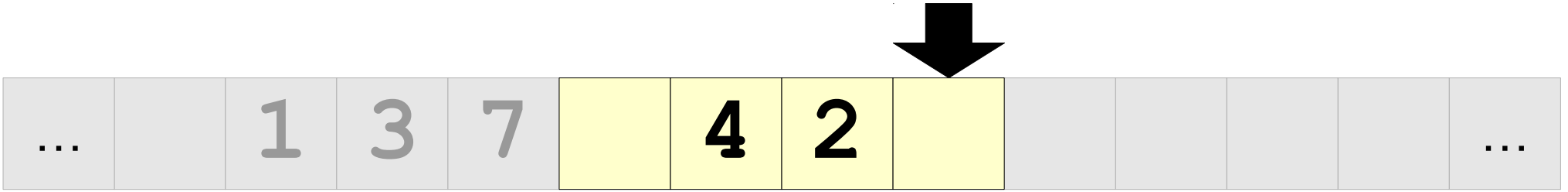

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

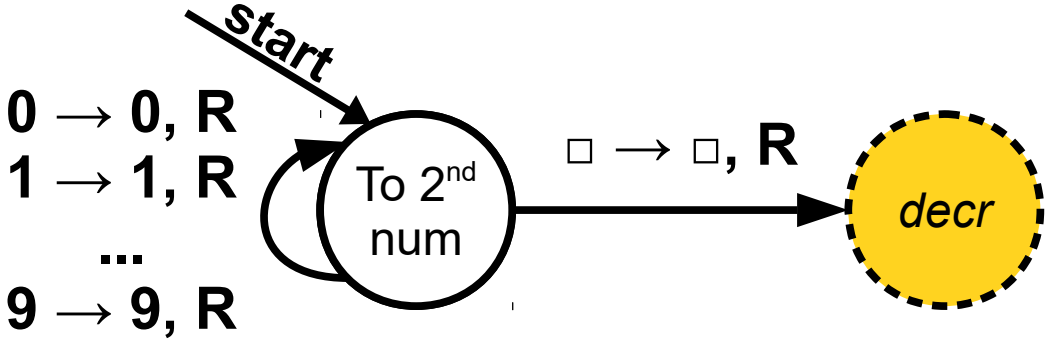

...
1
3
7
4
2
...

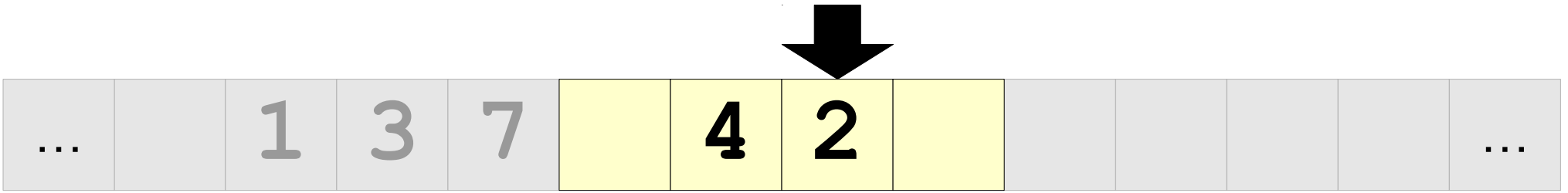

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

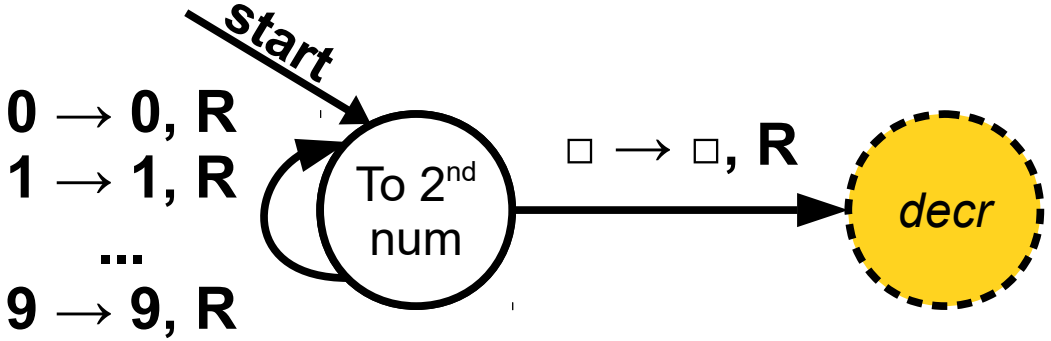

...
1
3
7
4
2
...

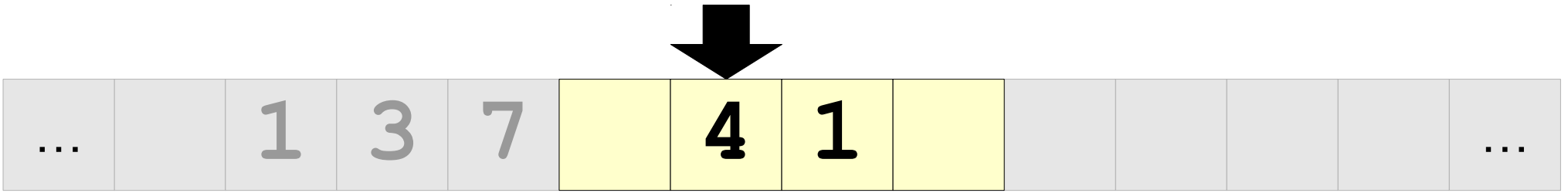

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

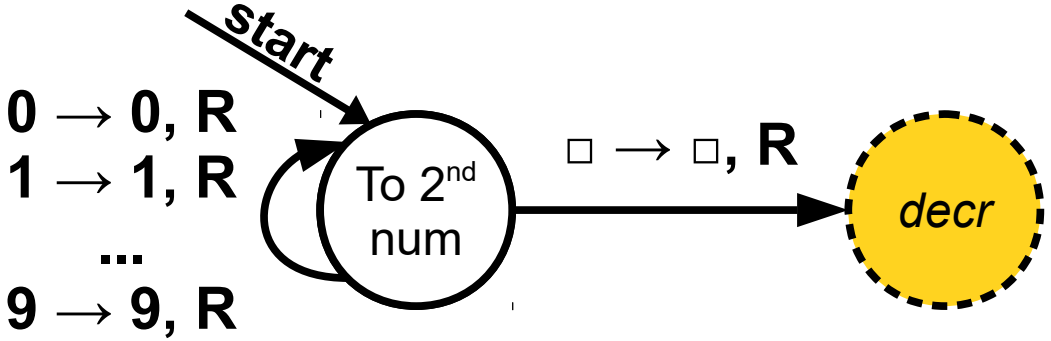

...
1
3
7
4
2
...

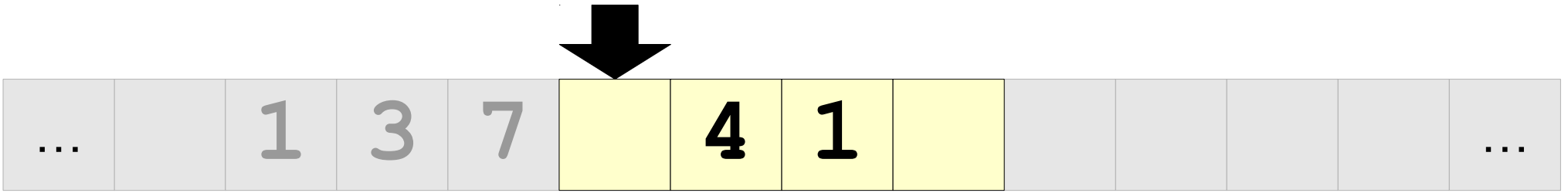

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

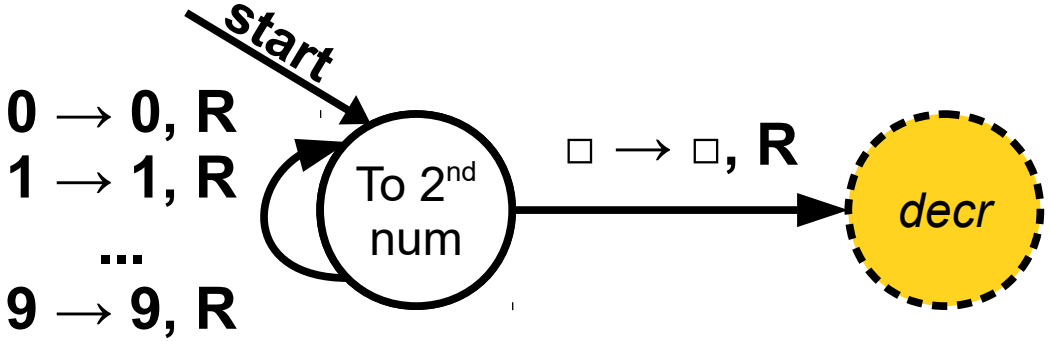

...
1
3
7
4
2
...

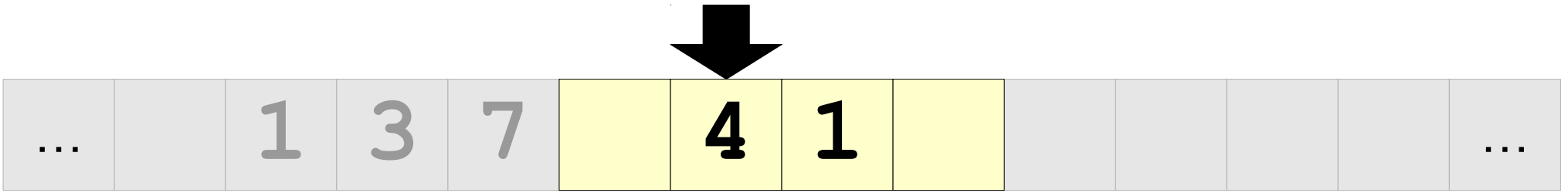

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

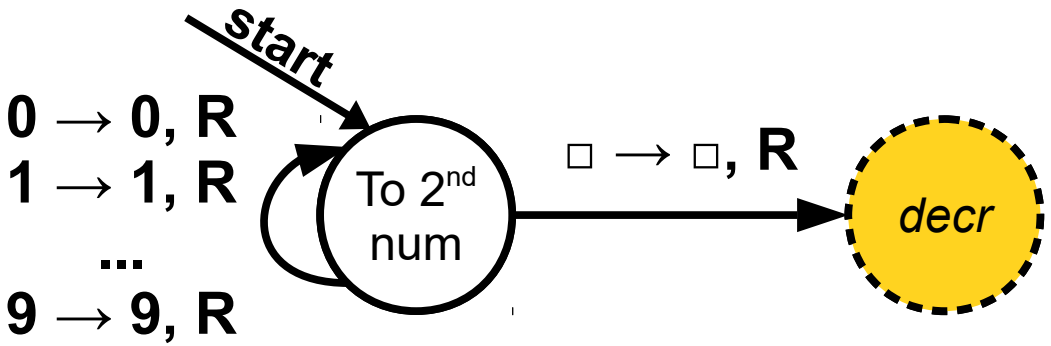

...
1
3
7
4
1
...

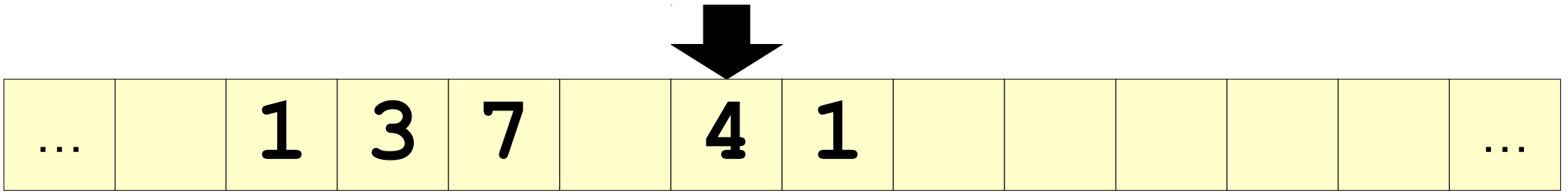

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

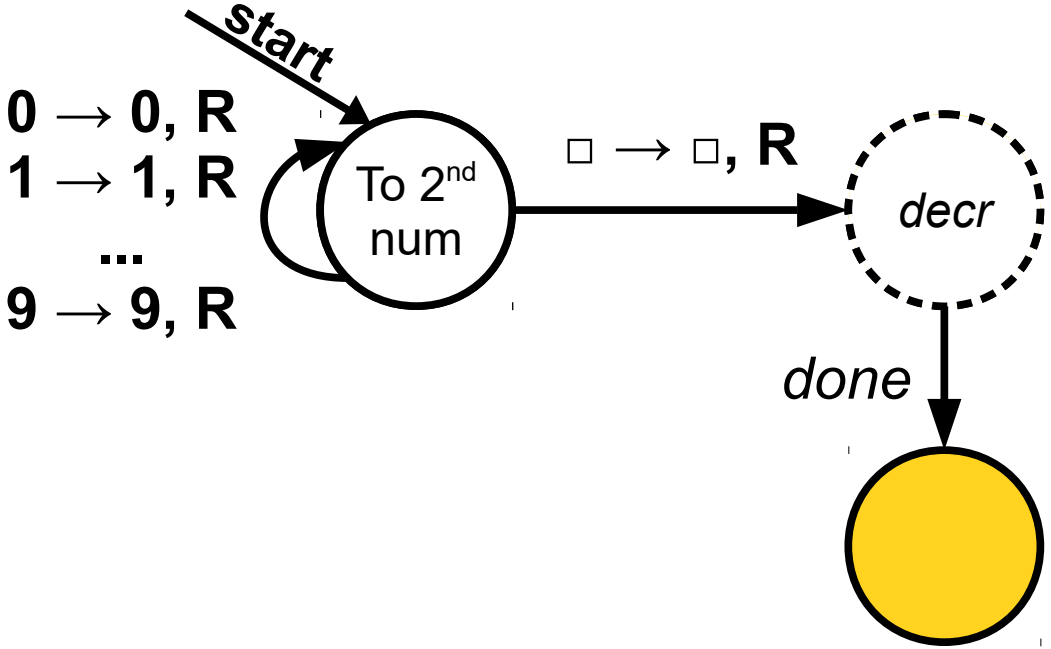

...
1
3
7
4
1
...

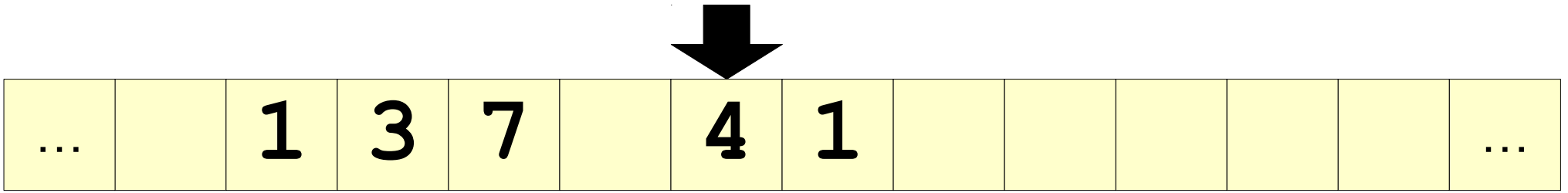

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

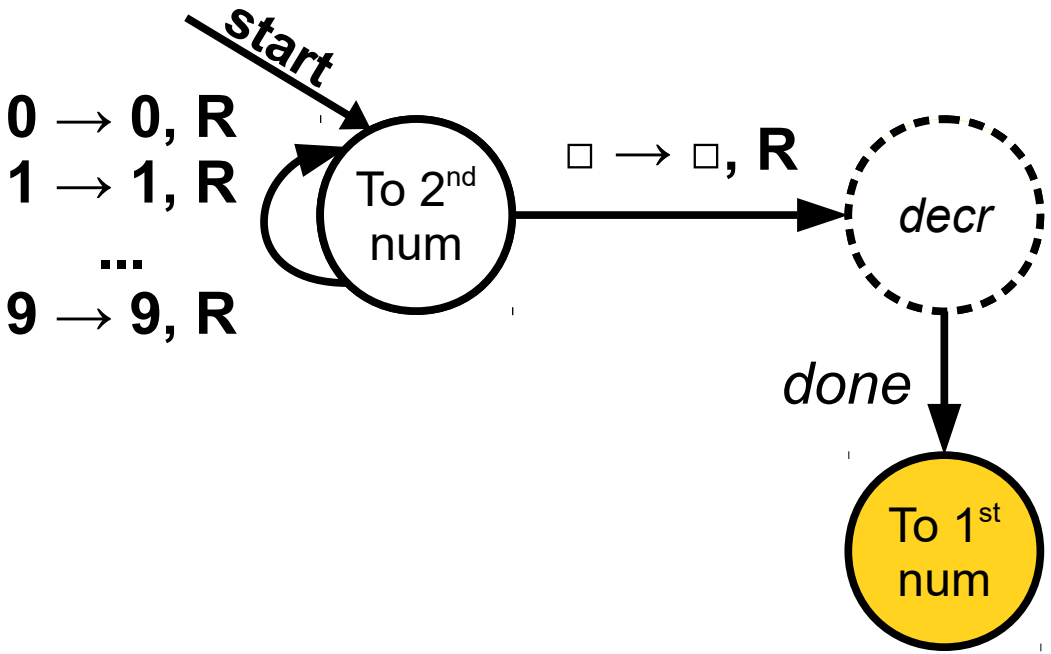

...
1
3
7
4
1
...

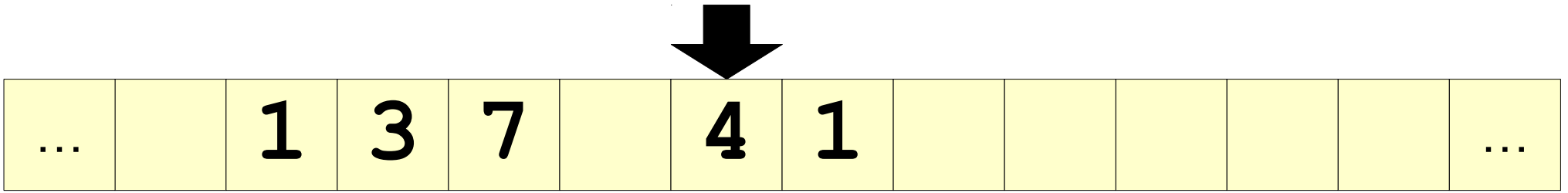

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr

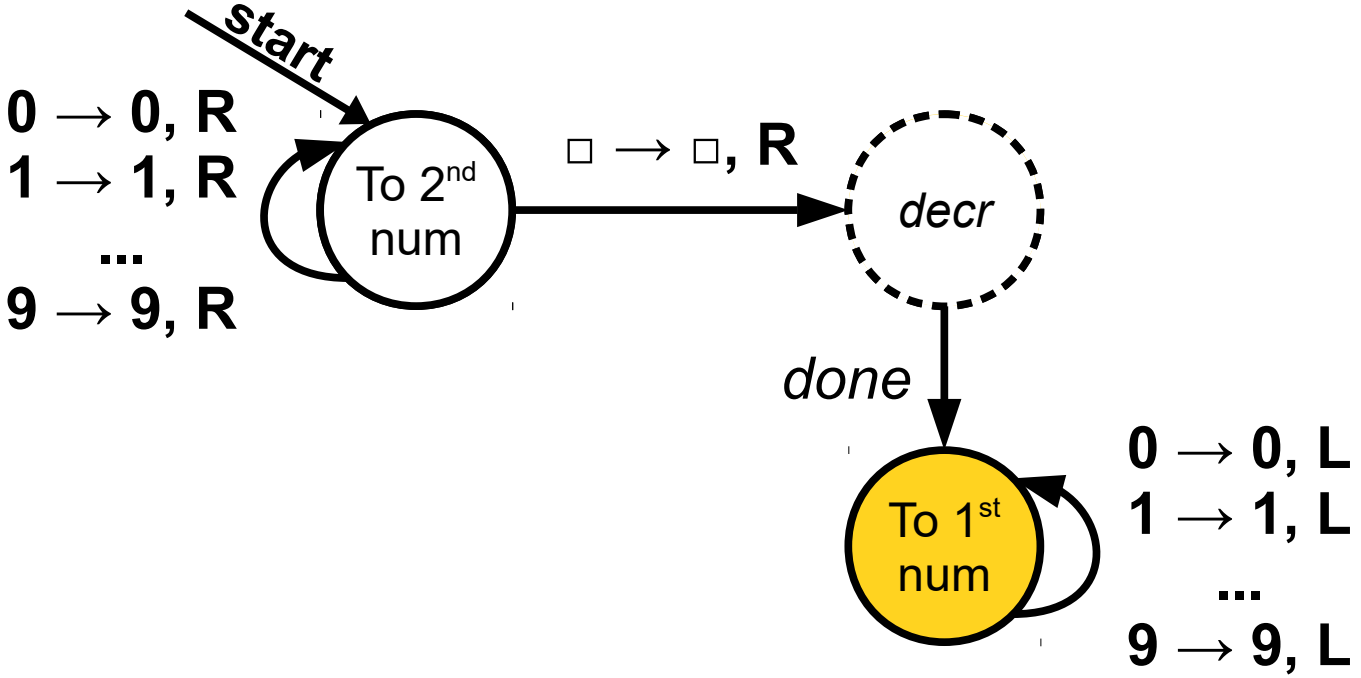

…
1
3
7
4
1
…

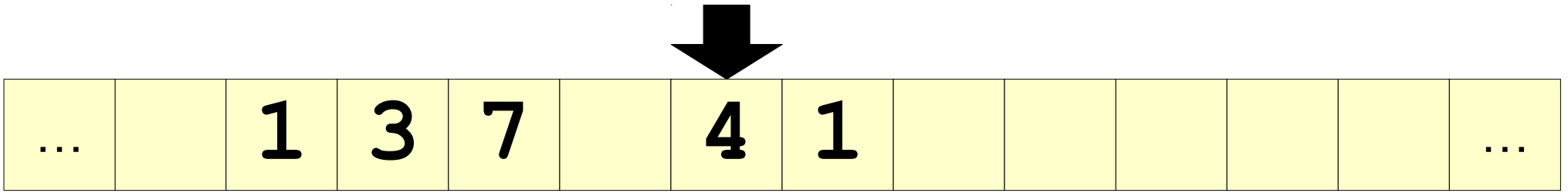

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done

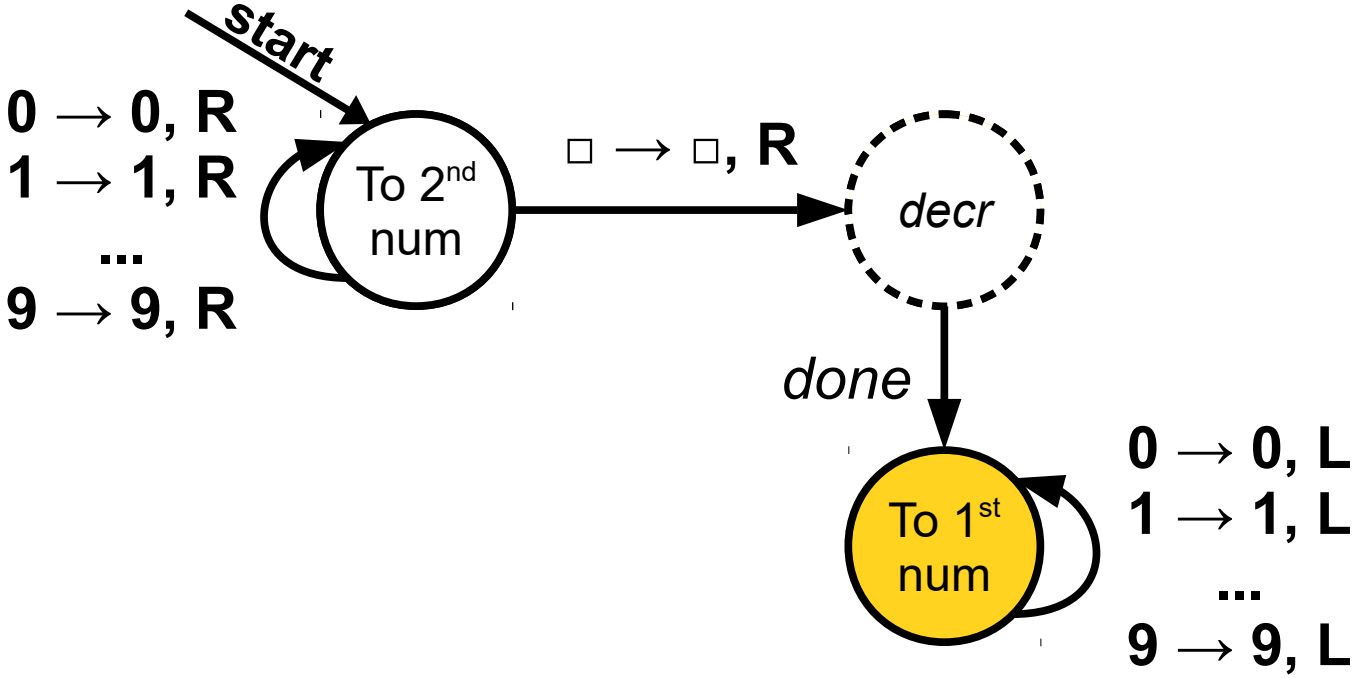

…
1
3
7
4
1
…

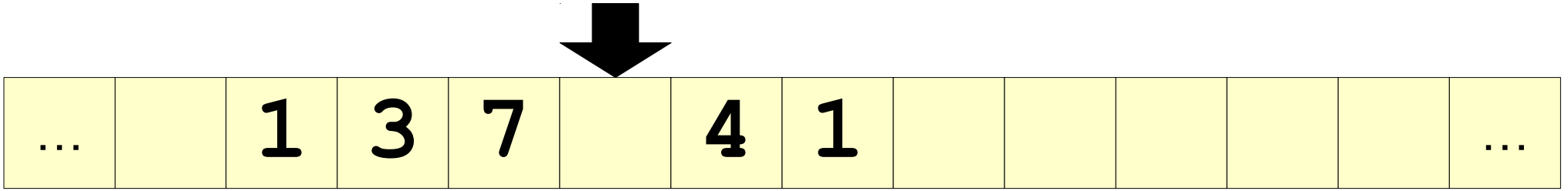

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num

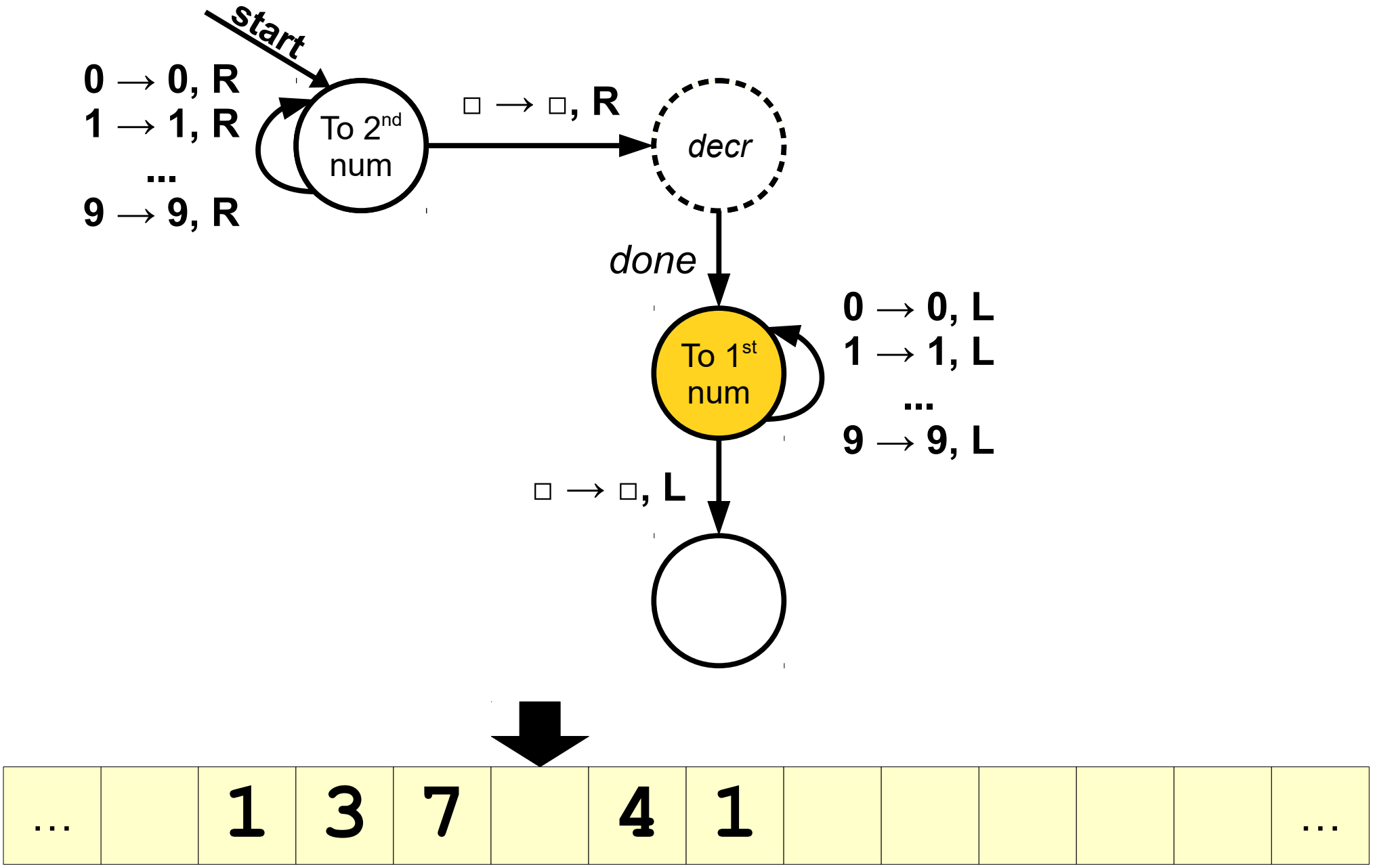

…
1
3
7
4
1
…

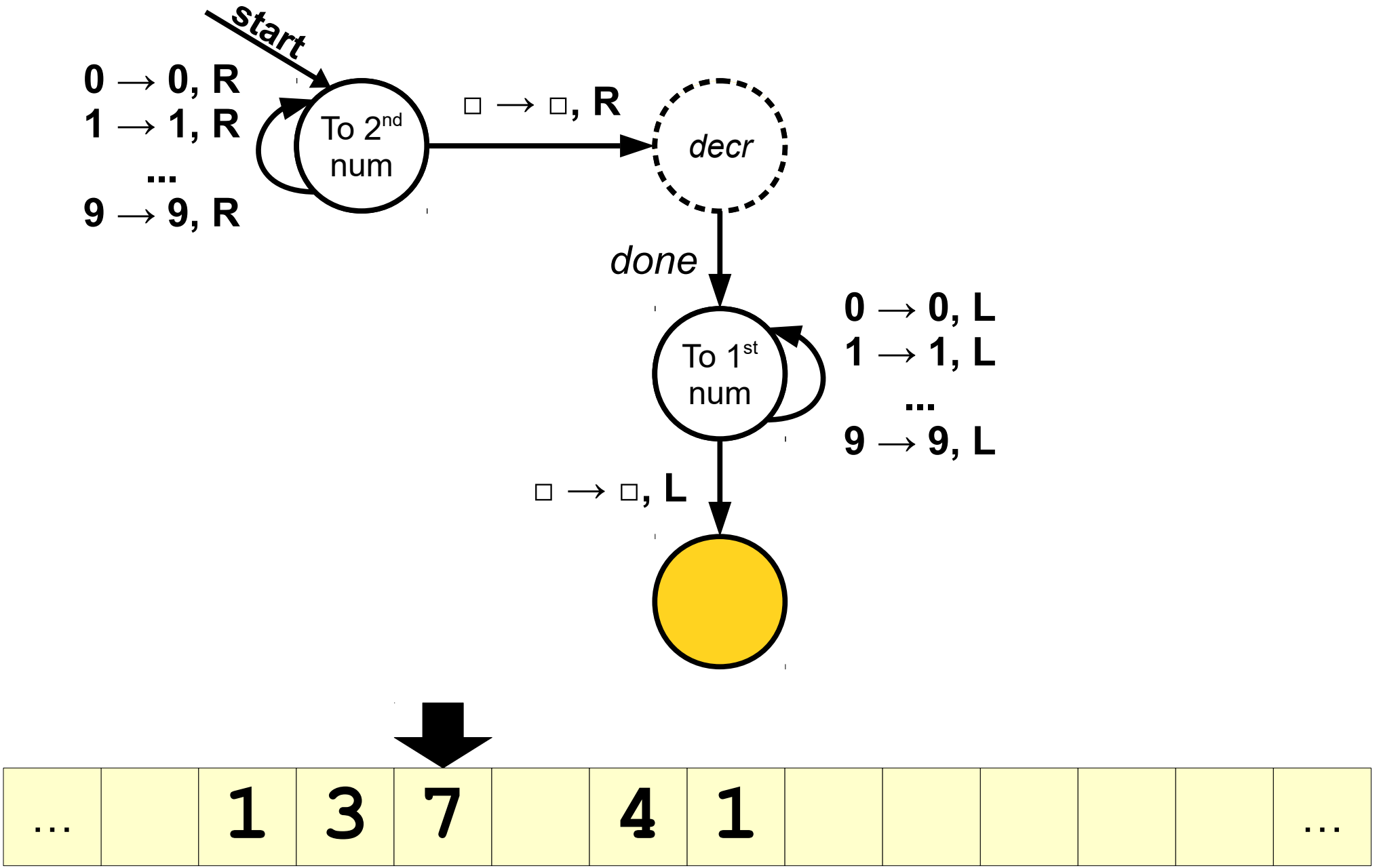

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L

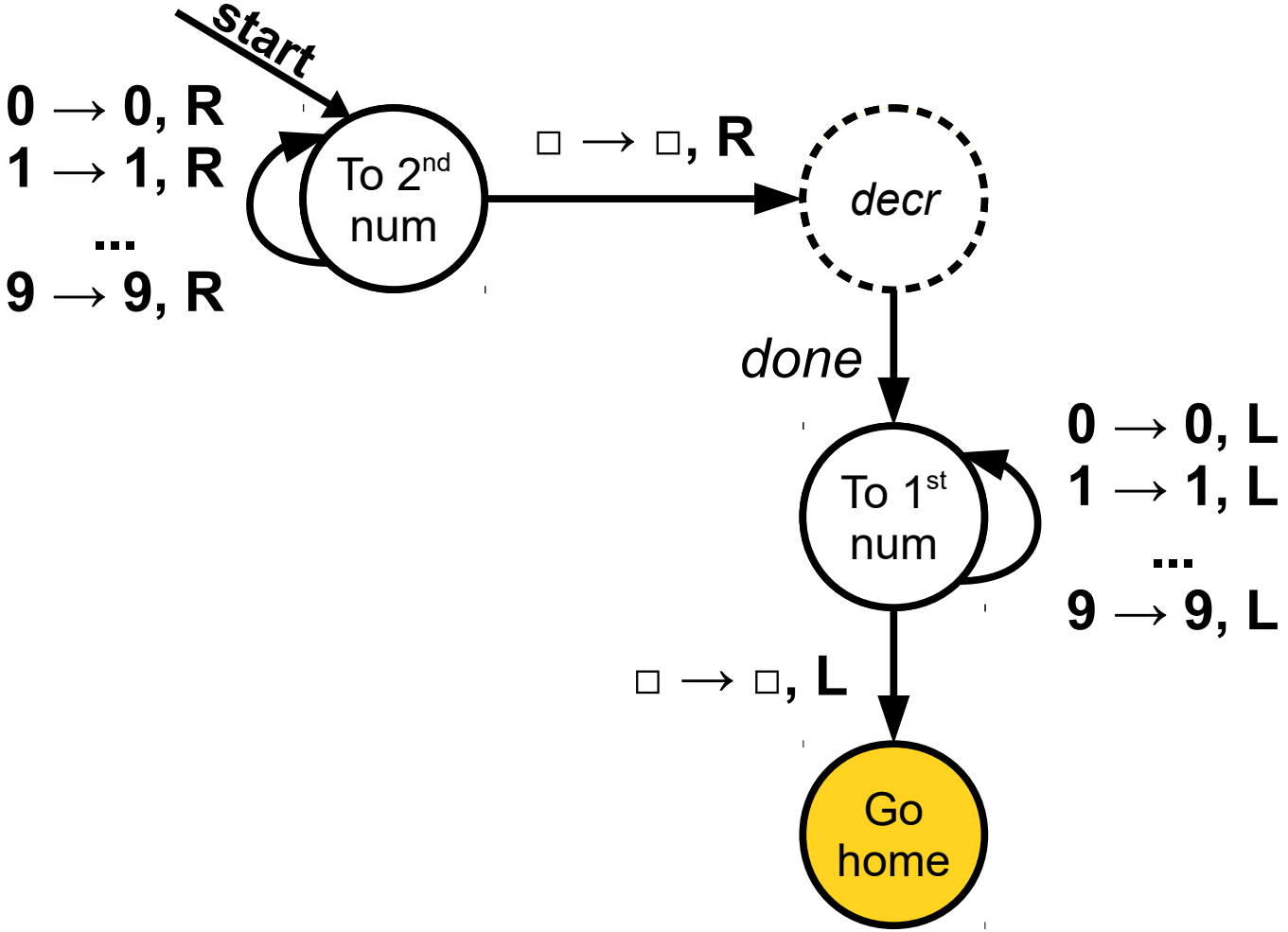

…
1
3
7
4
1
…

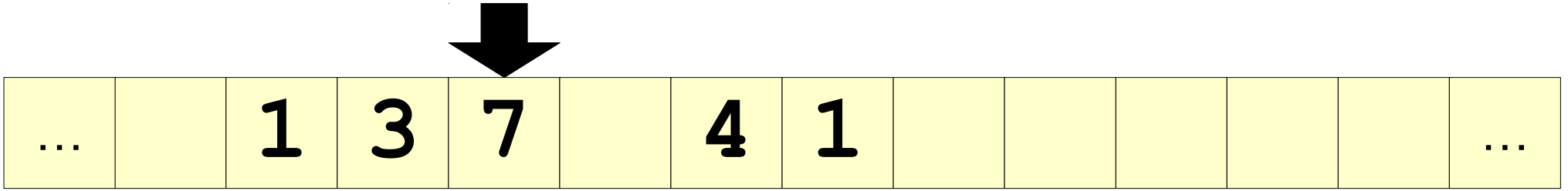

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L

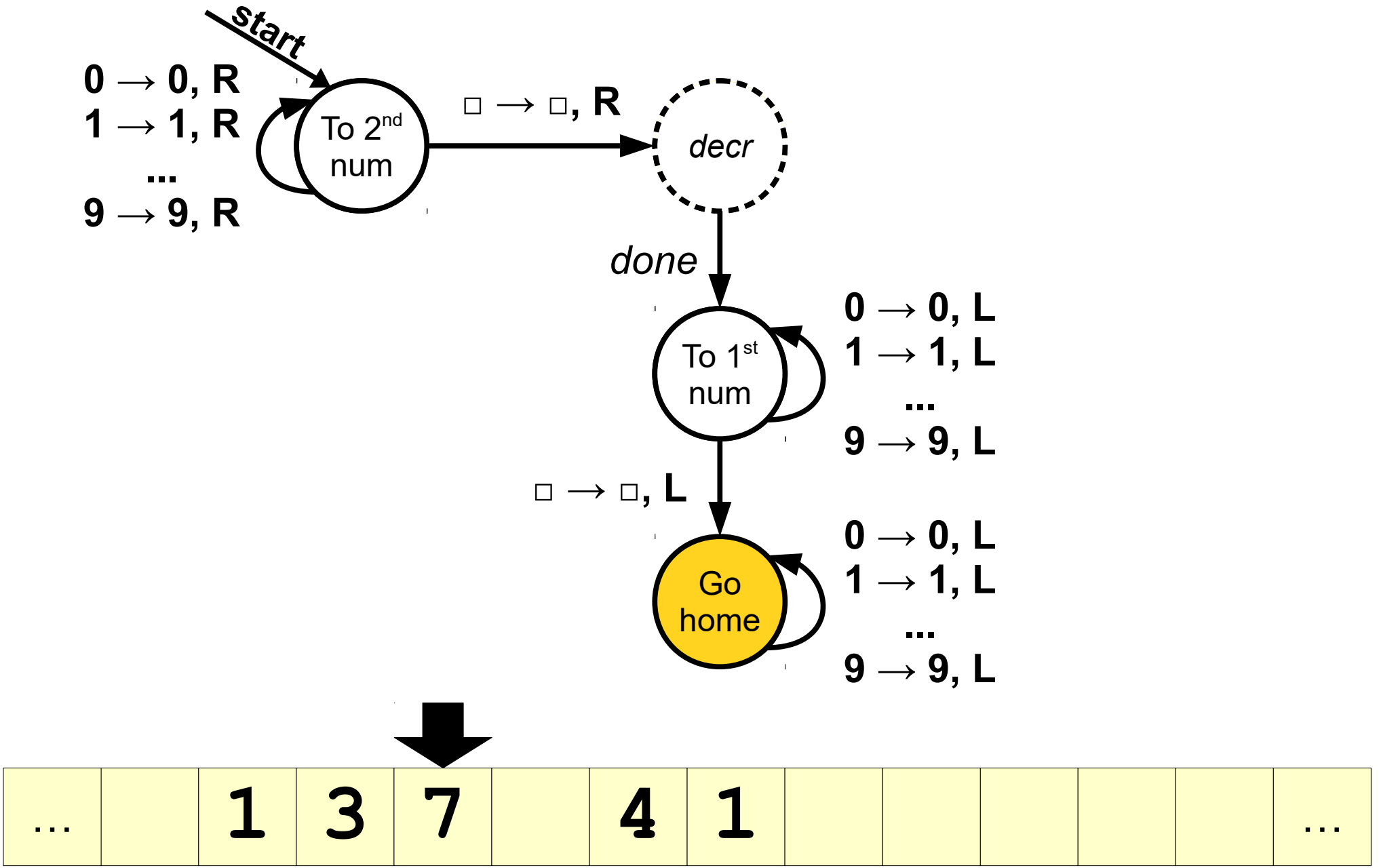

…
1
3
7
4
1
…

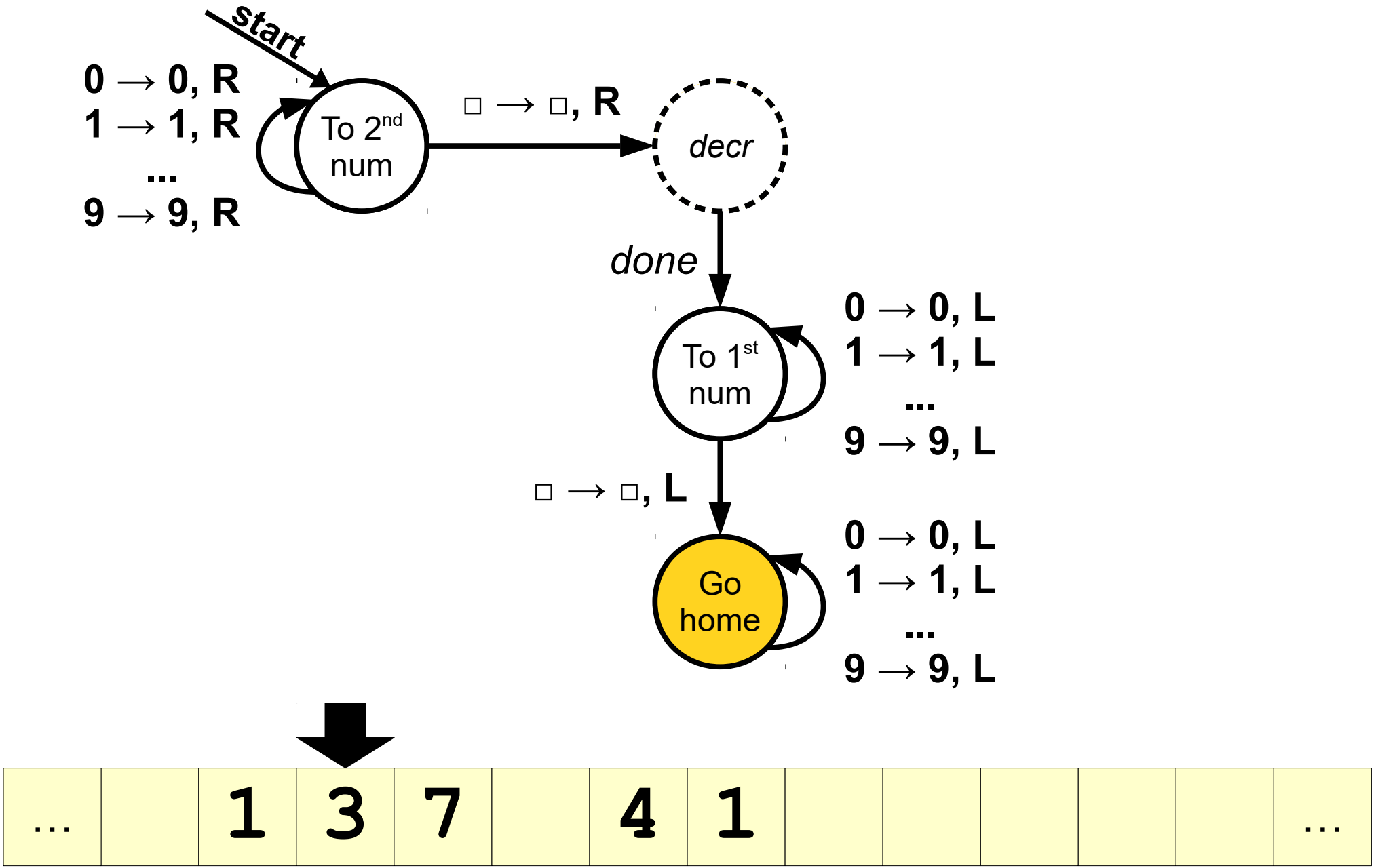

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
… 1 3 7 4 1 …

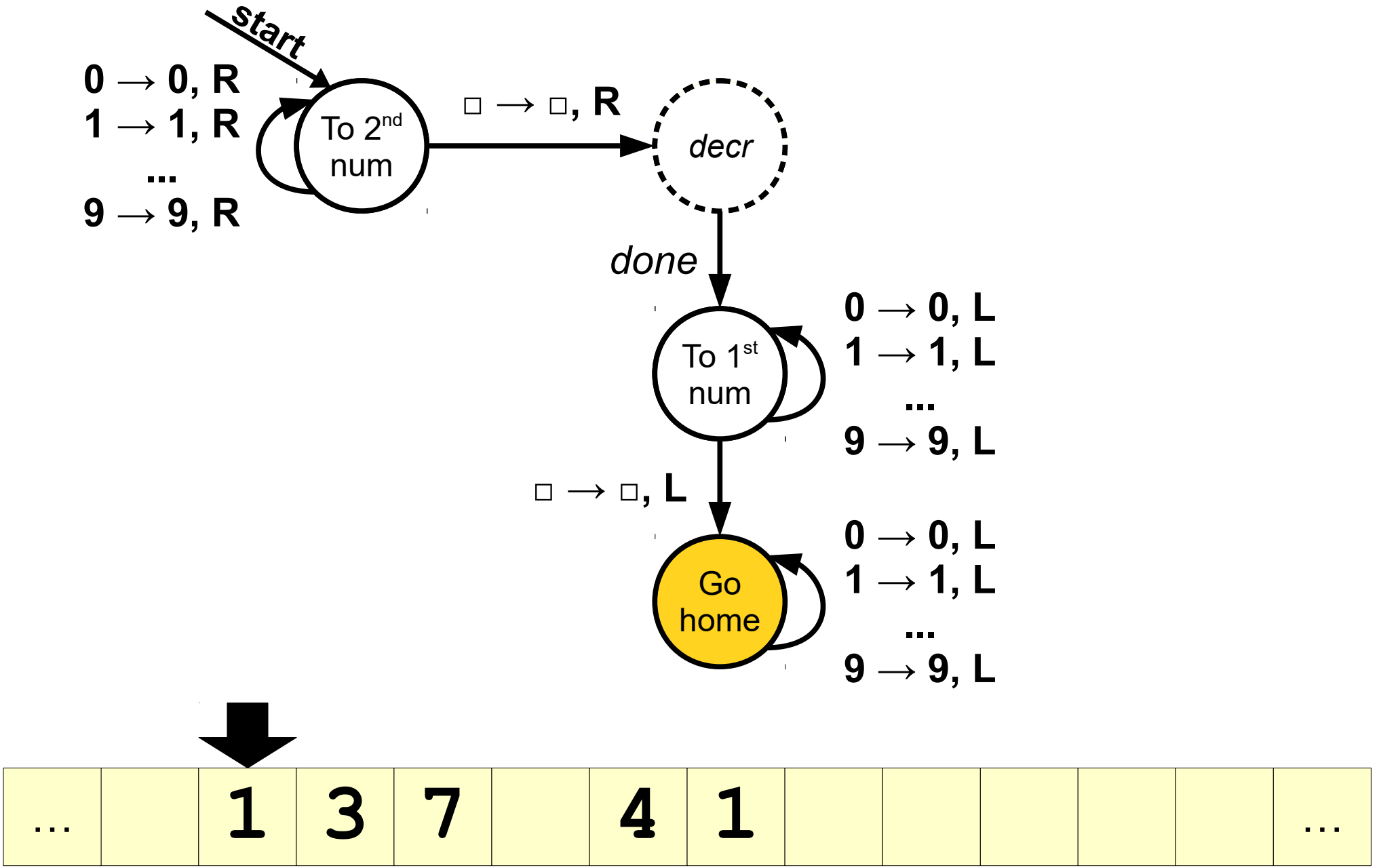

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
… 1 3 7 4 1 …

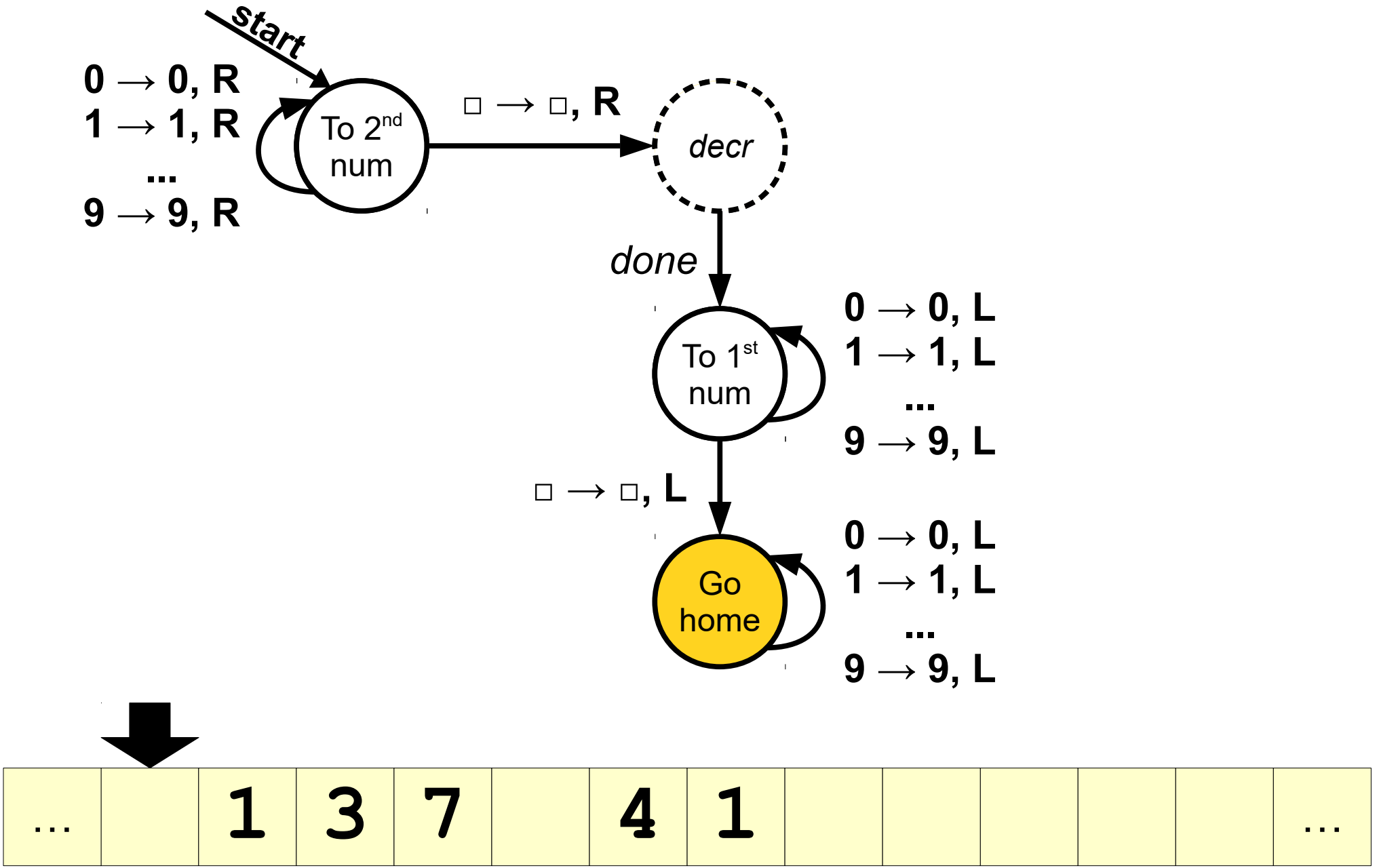

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home

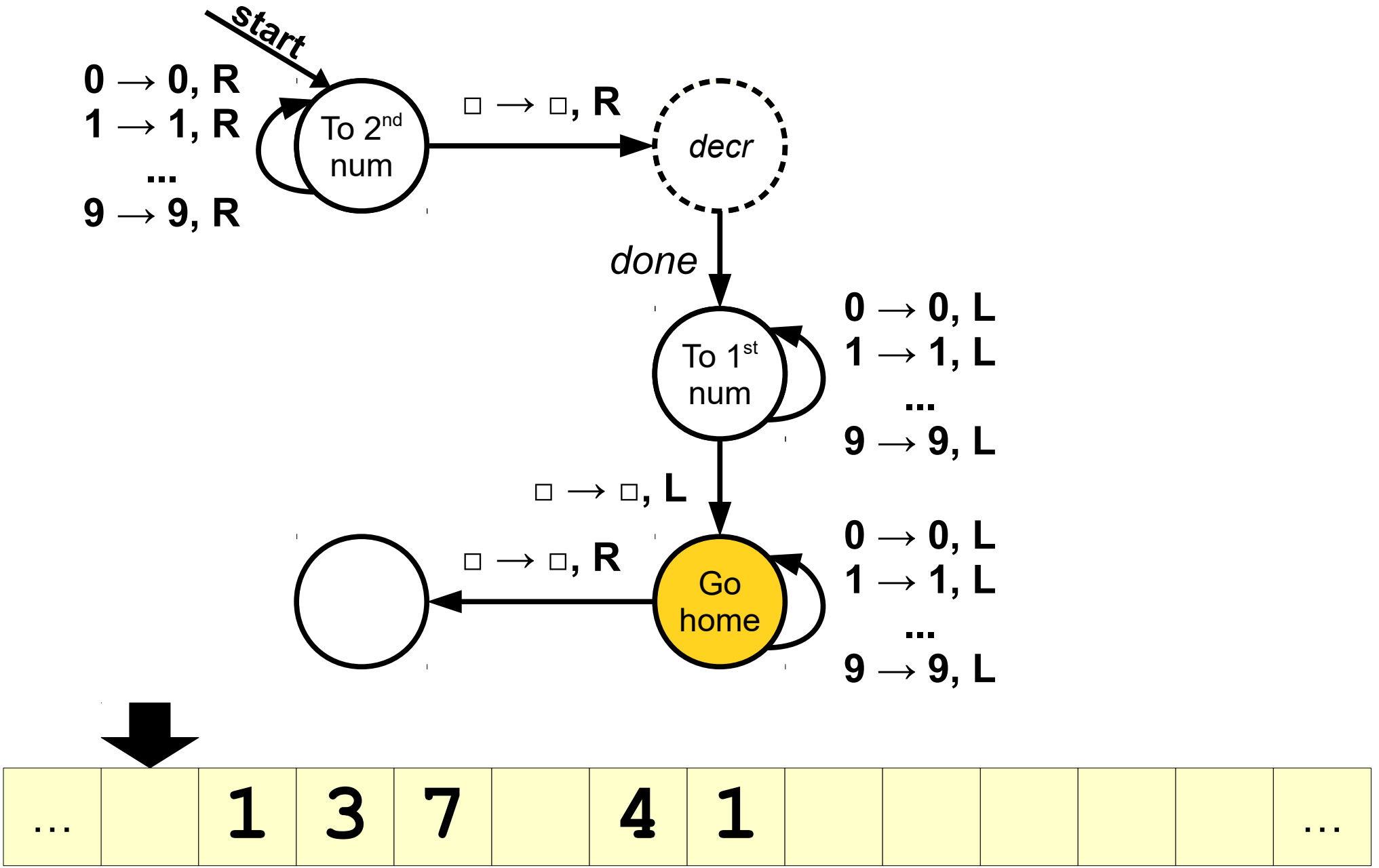

…
1
3
7
4
1
…

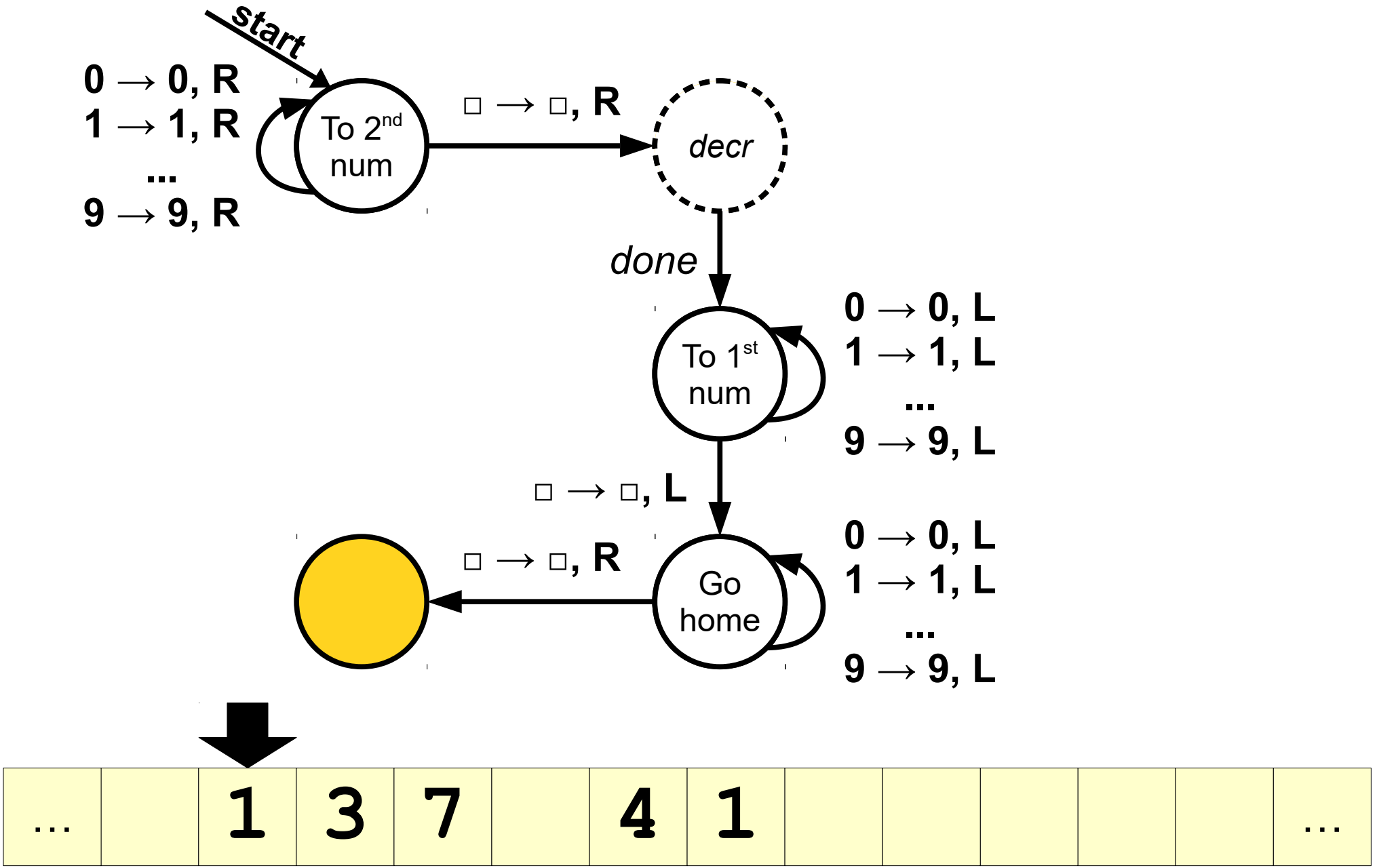

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
… 1 3 7 4 1 …

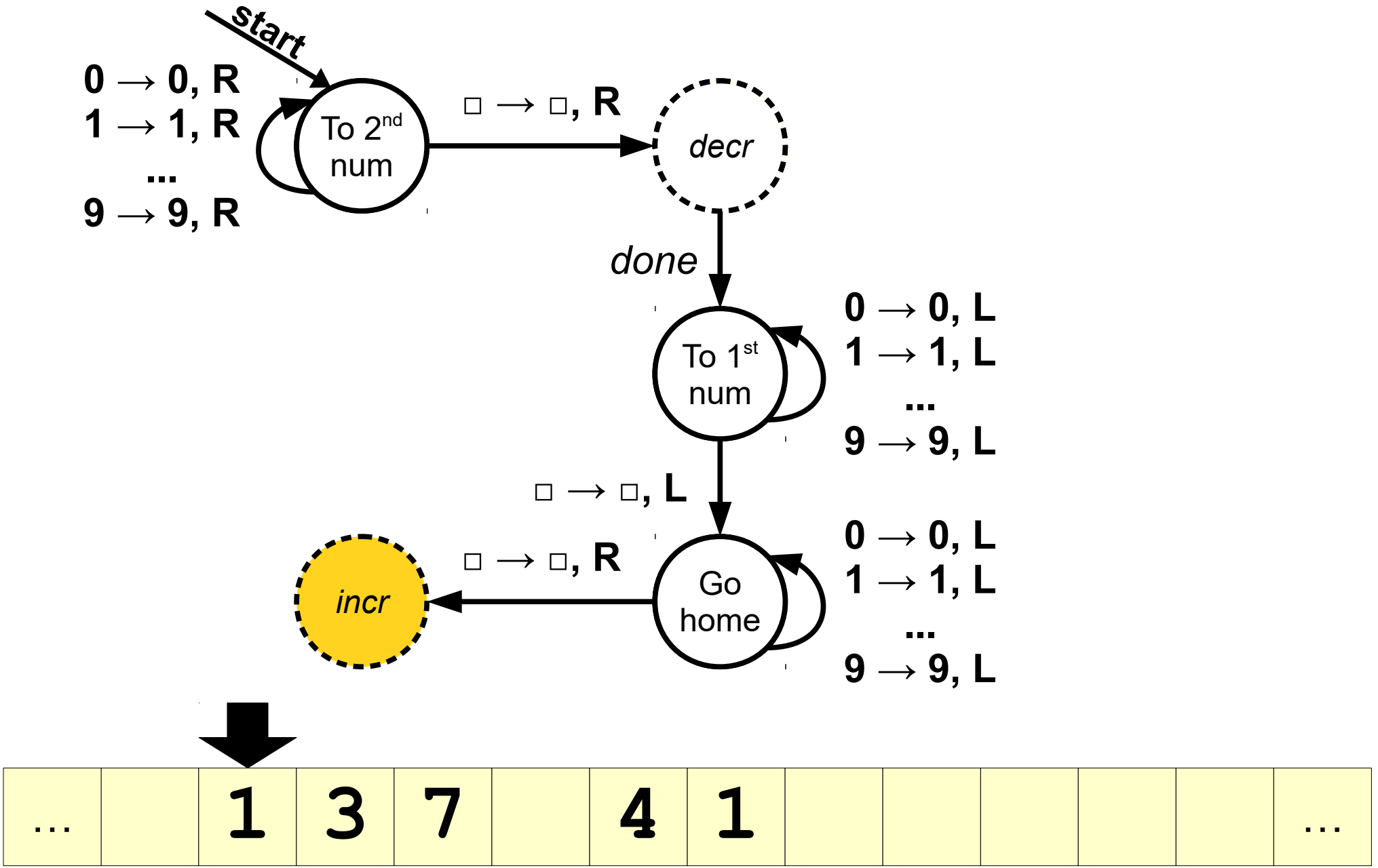

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
… 1 3 7 4 1 …

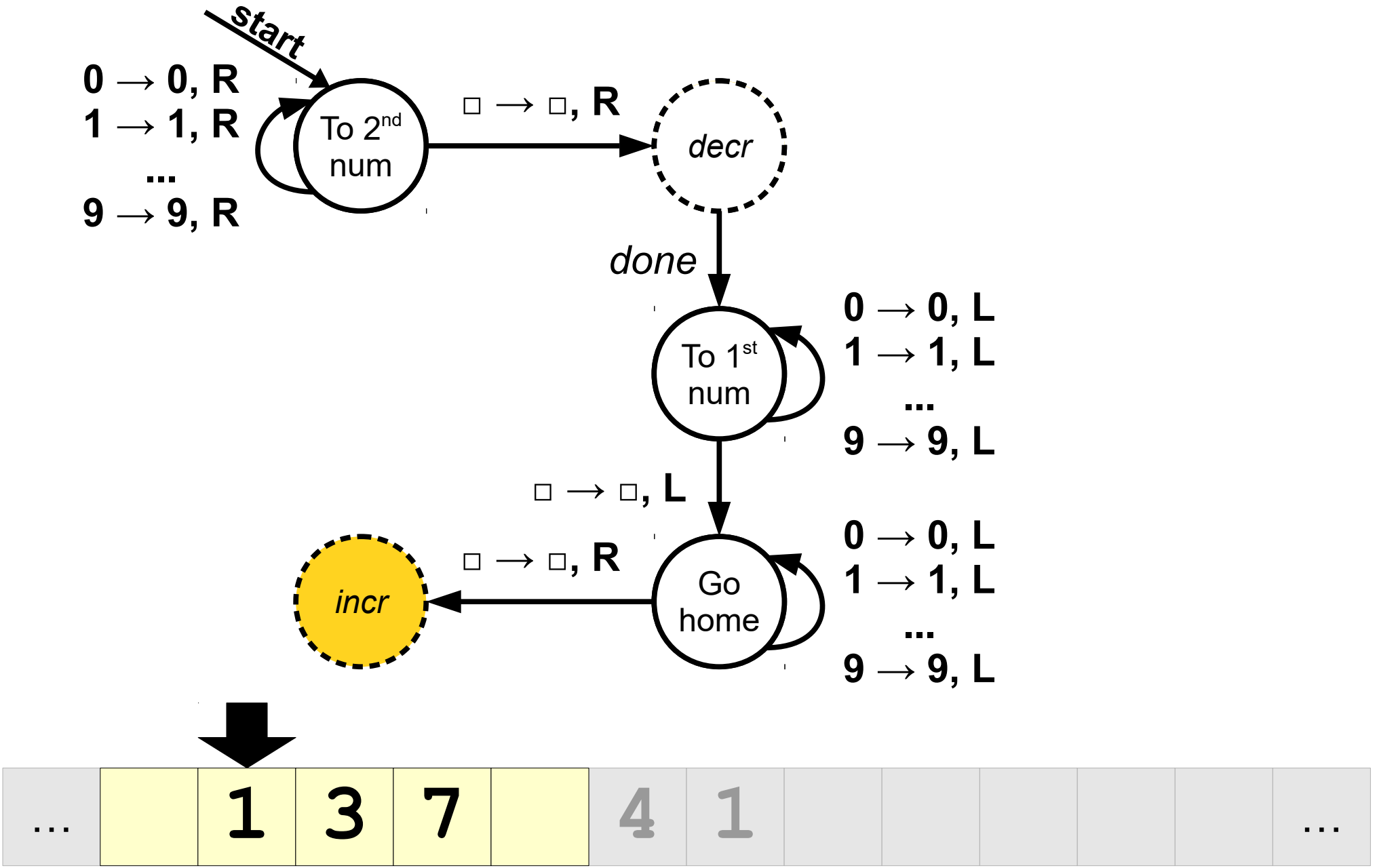

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
… 1 3 7 4 1 …

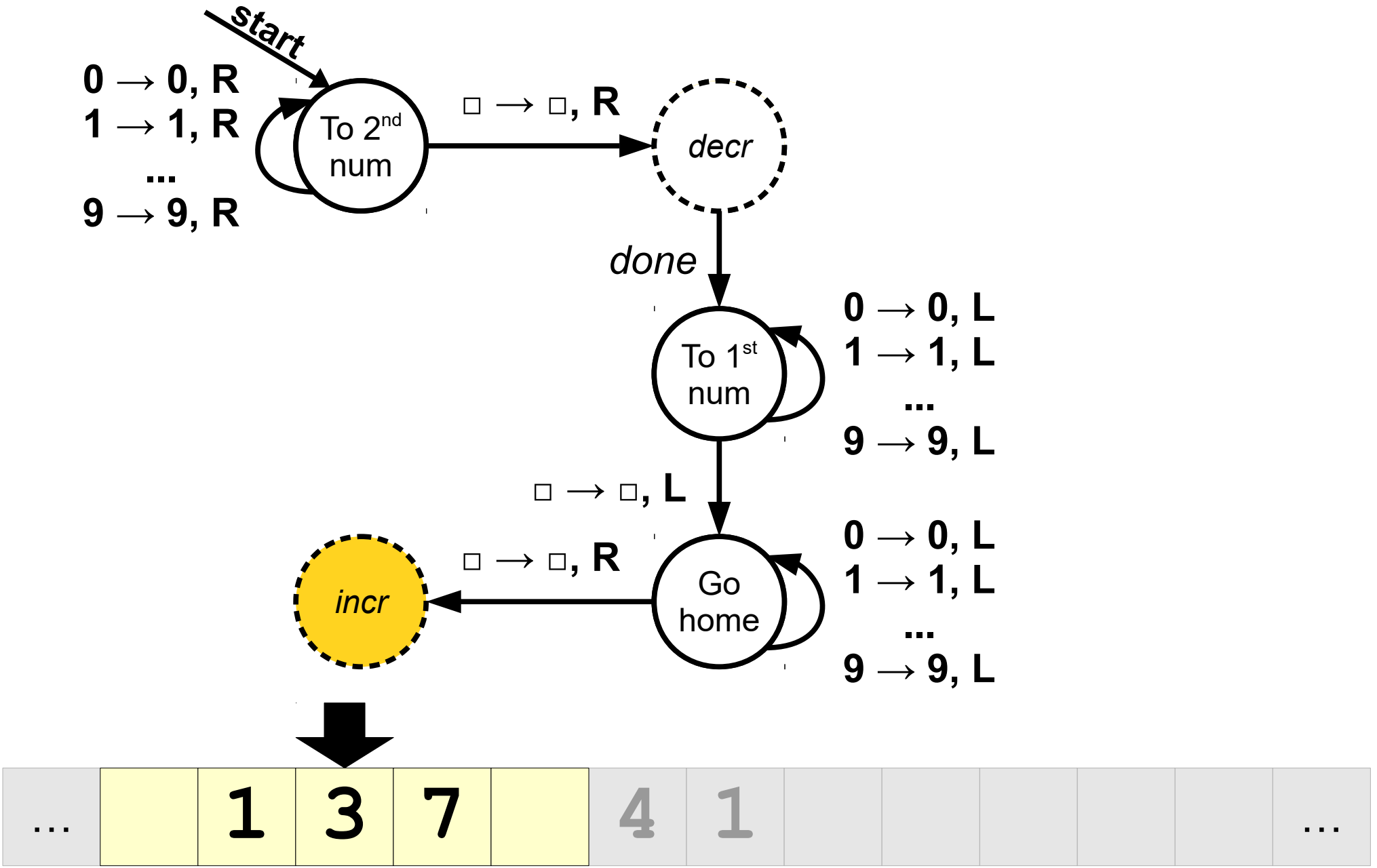

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
… 1 3 7 4 1 …

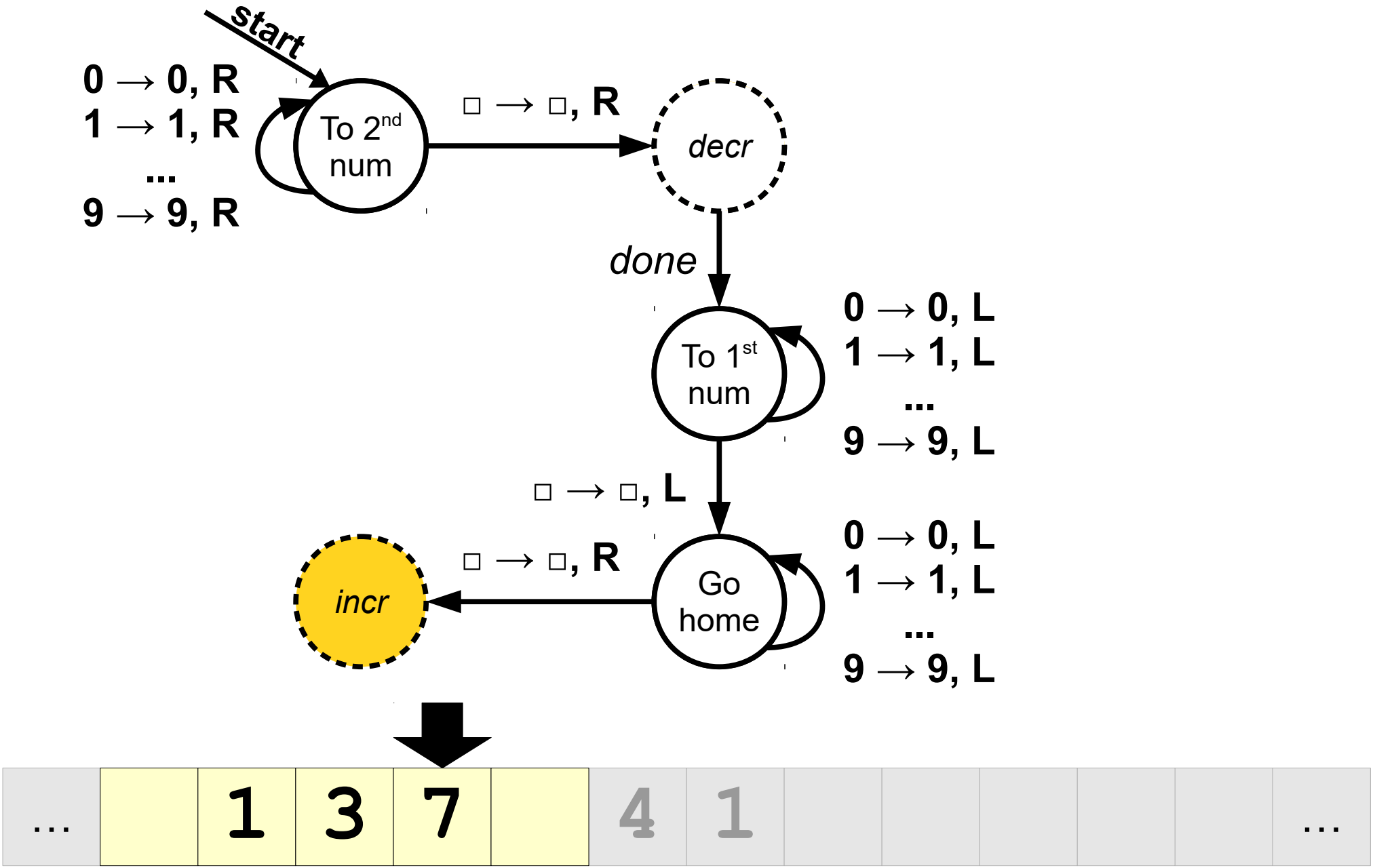

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
□ → □, R
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
… 1 3 7 4 1 …

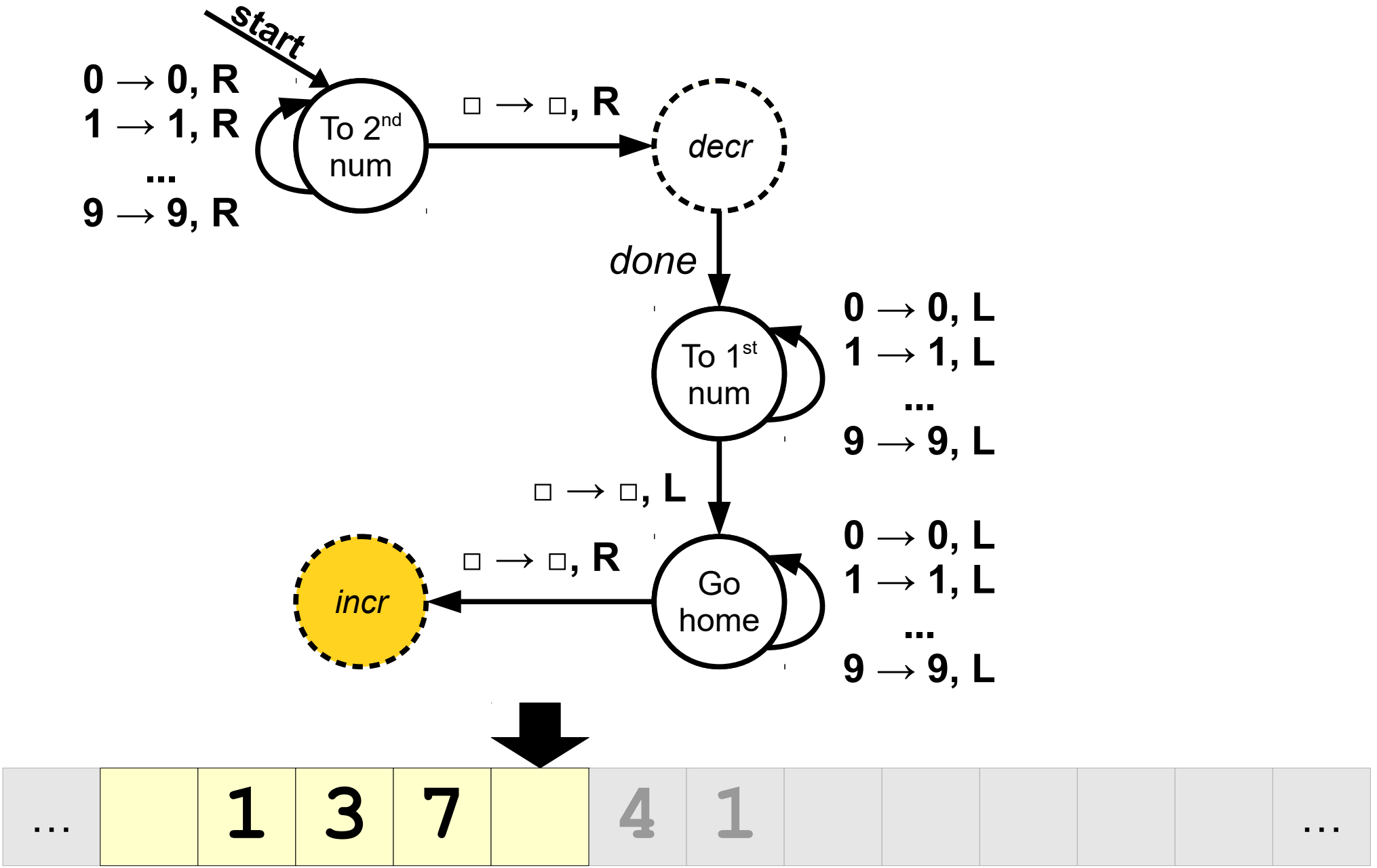

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
□ → □, R
0 → 0, L
1 → 1, L
...
9 → 9, L
… 1 3 7 4 1 …

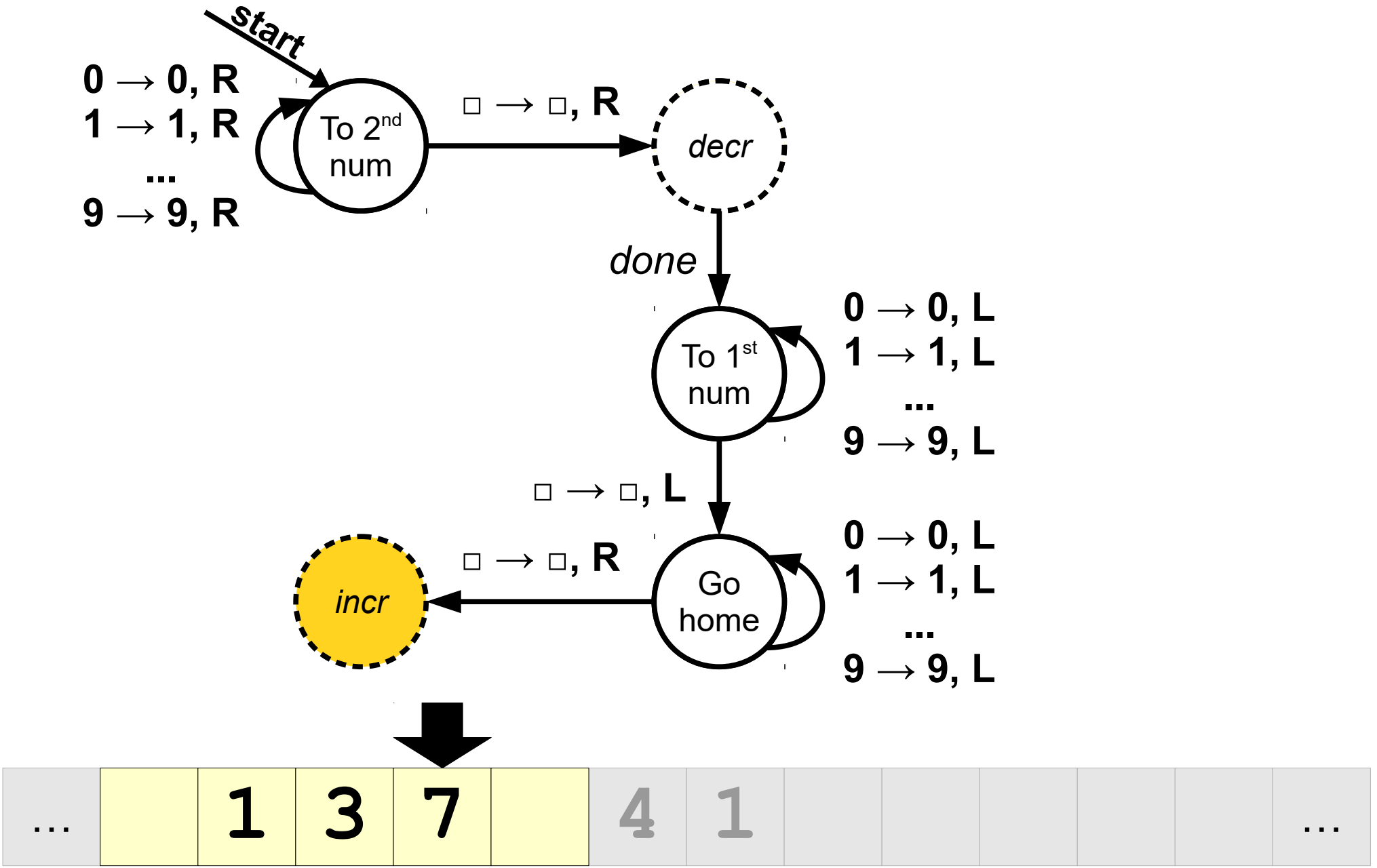

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 7 4 1 …

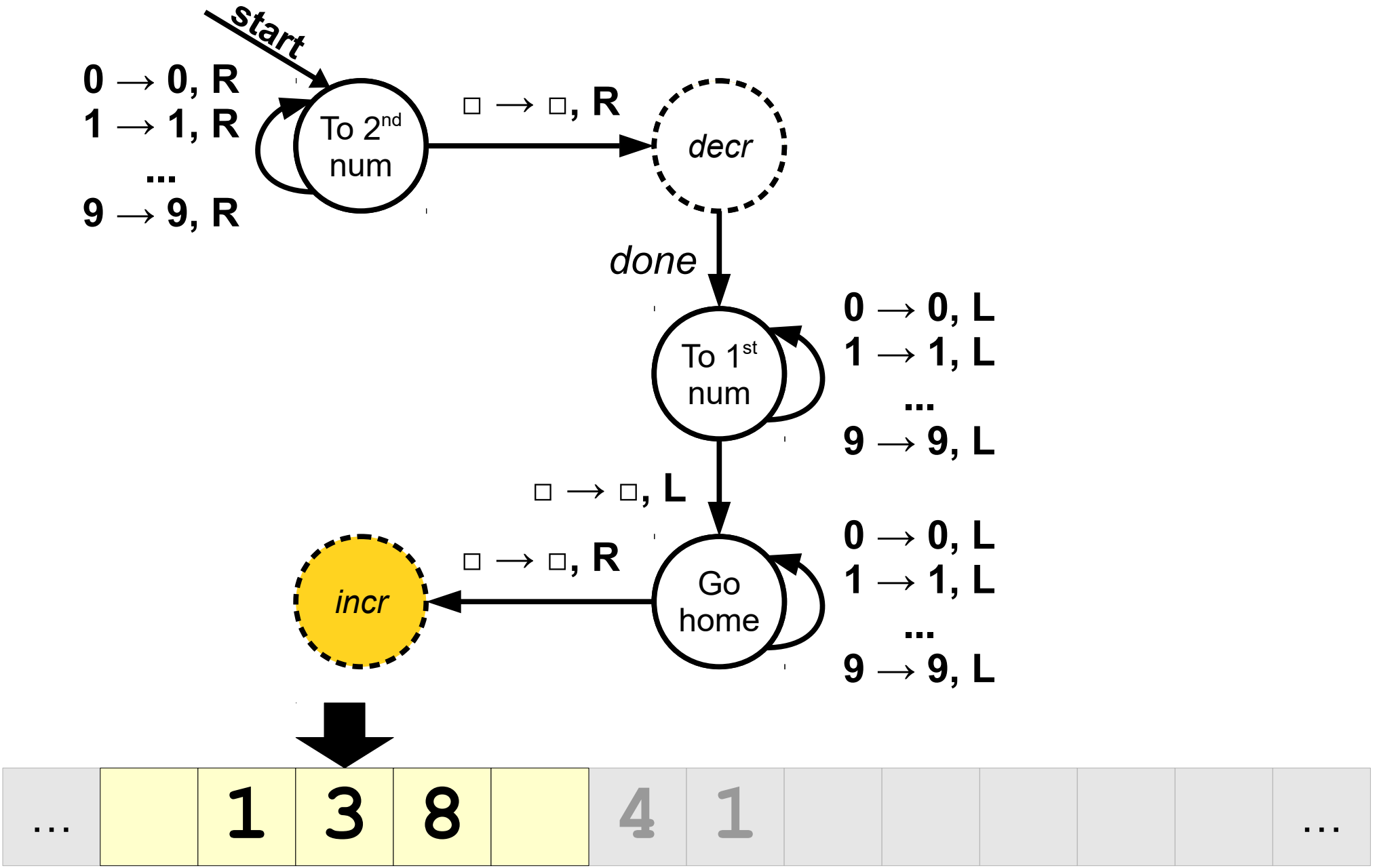

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 7 4 1 …

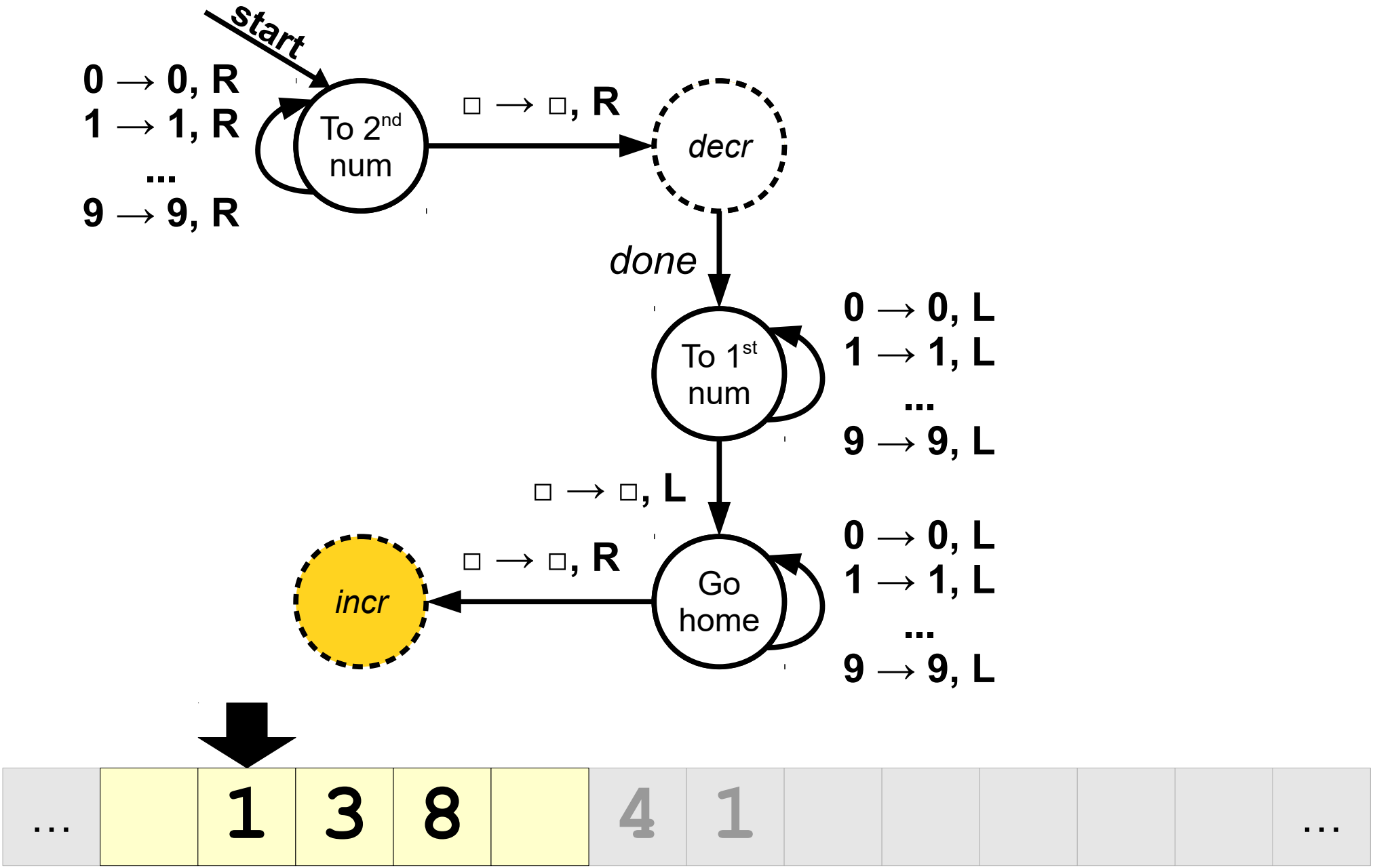

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 7 4 1 …

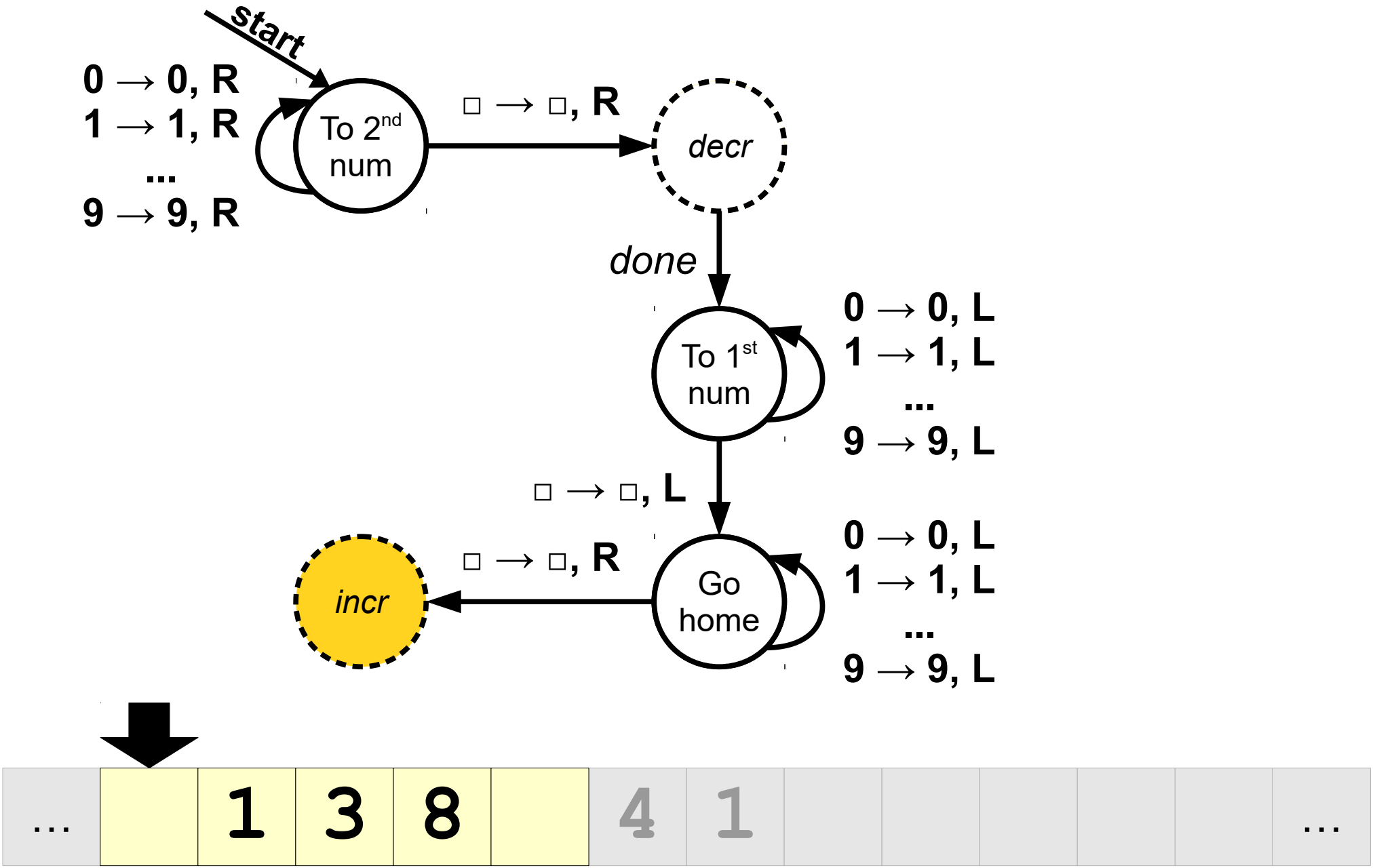

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 7 4 1 …

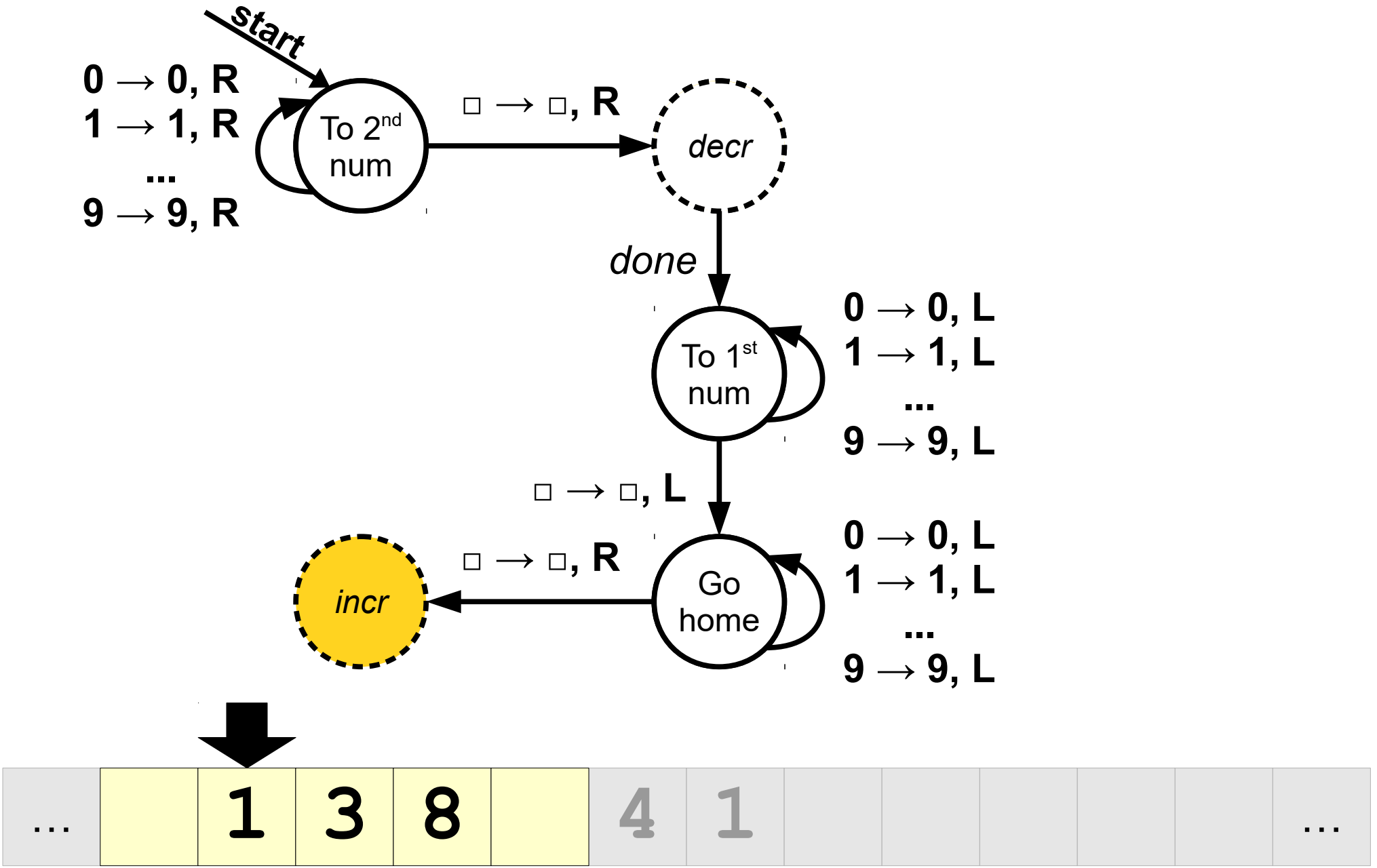

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 7 4 1 …

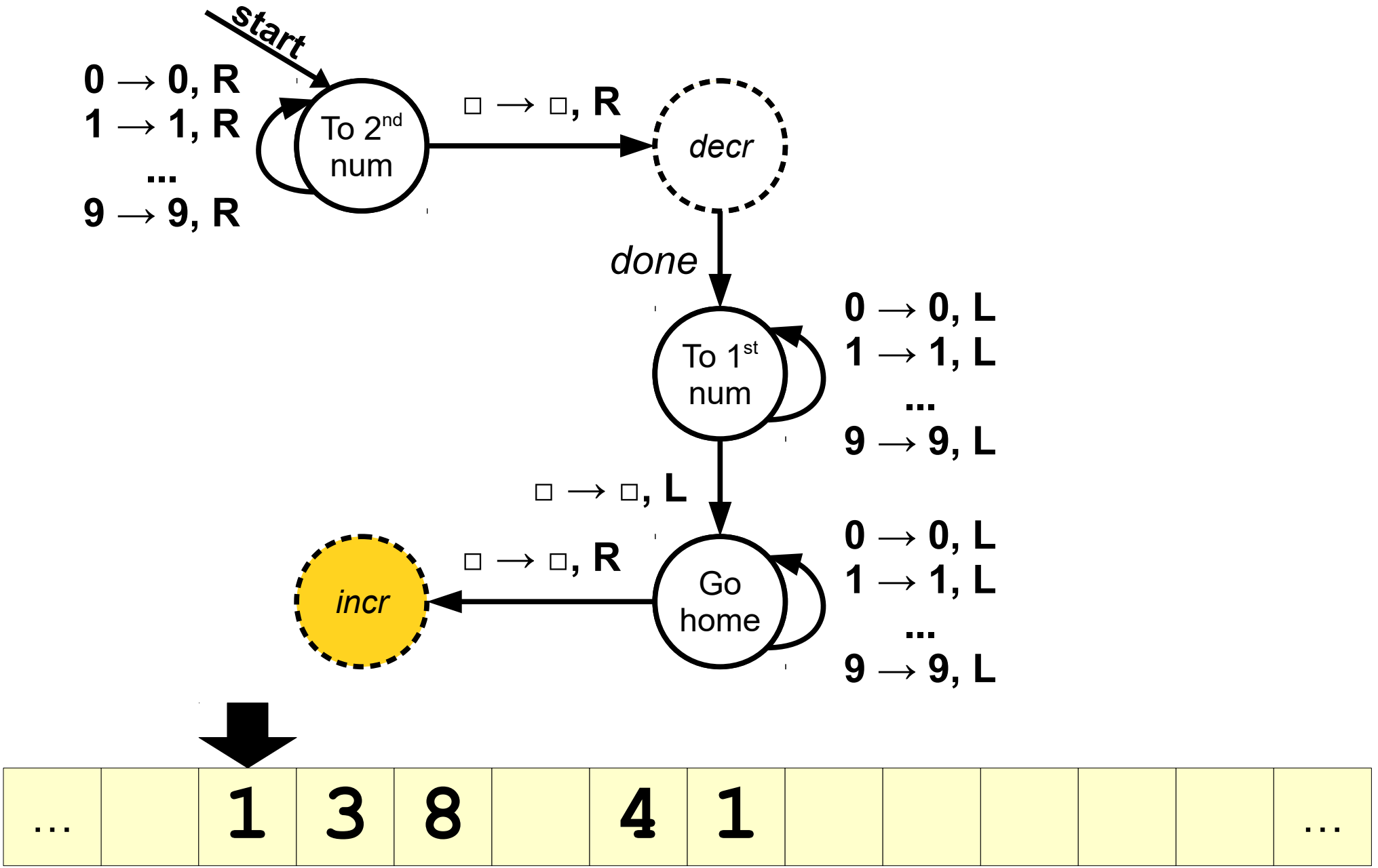

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 7 4 1 …

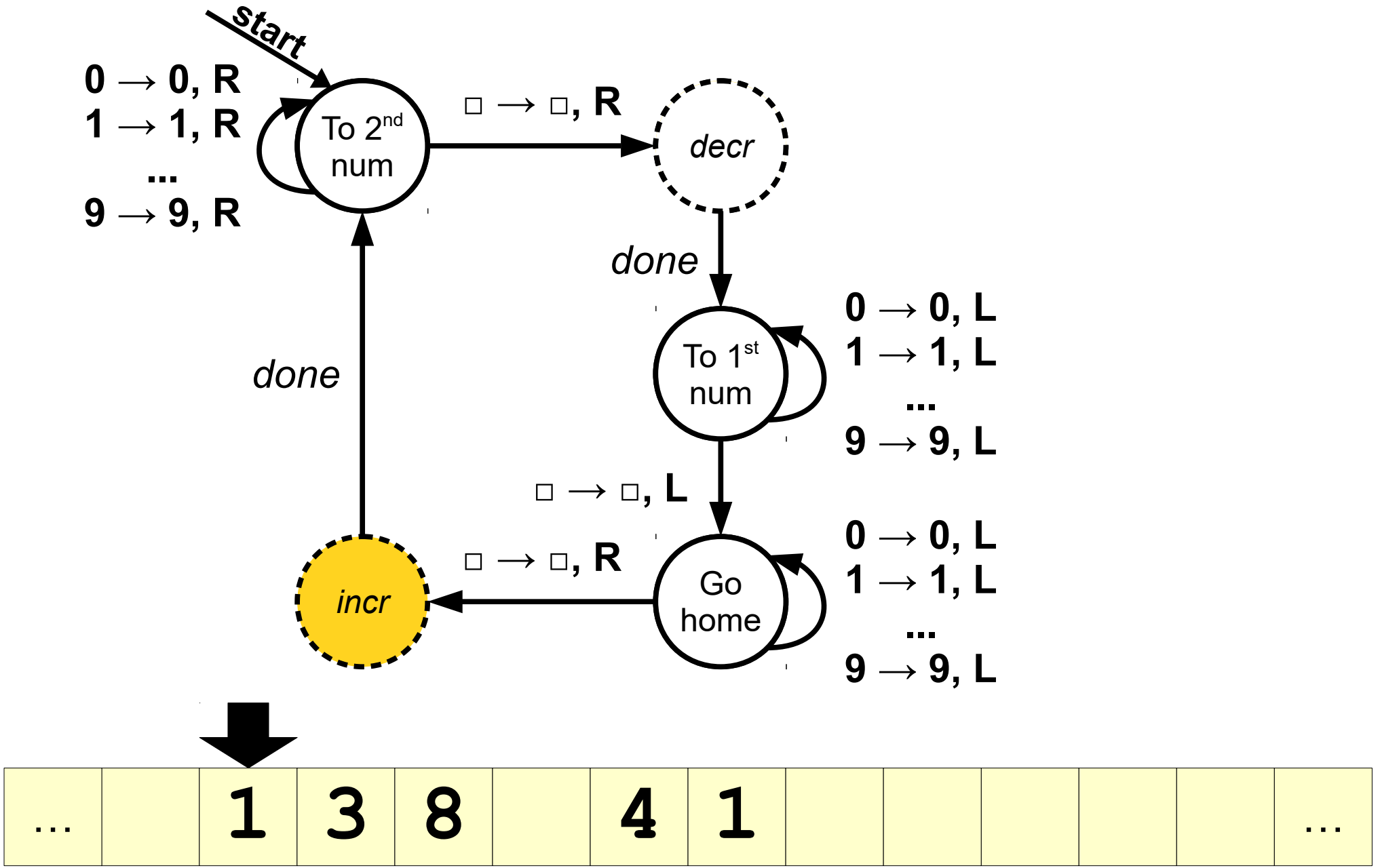

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 8 4 1 …

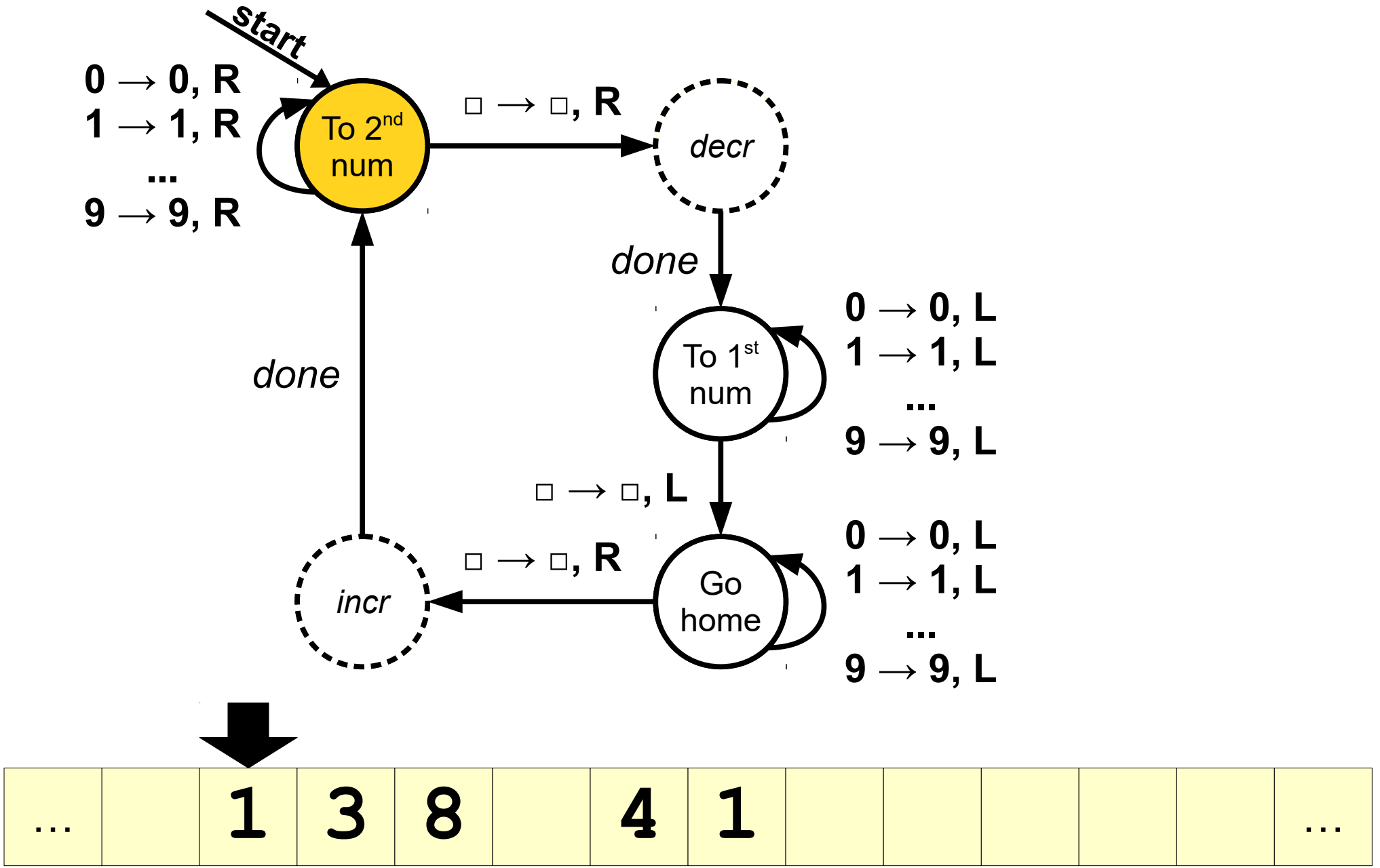

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 8 4 1 …

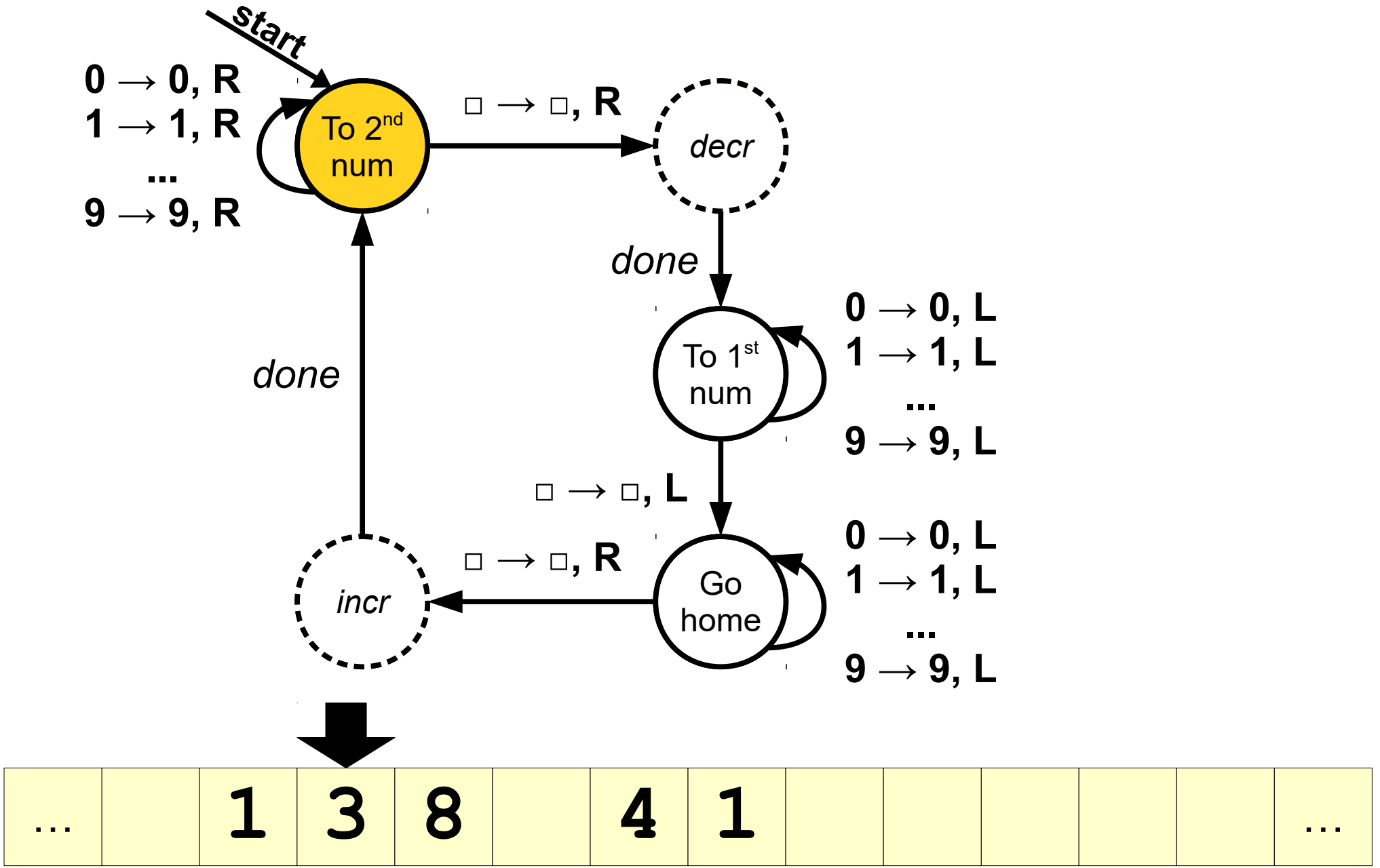

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 8 4 1 …

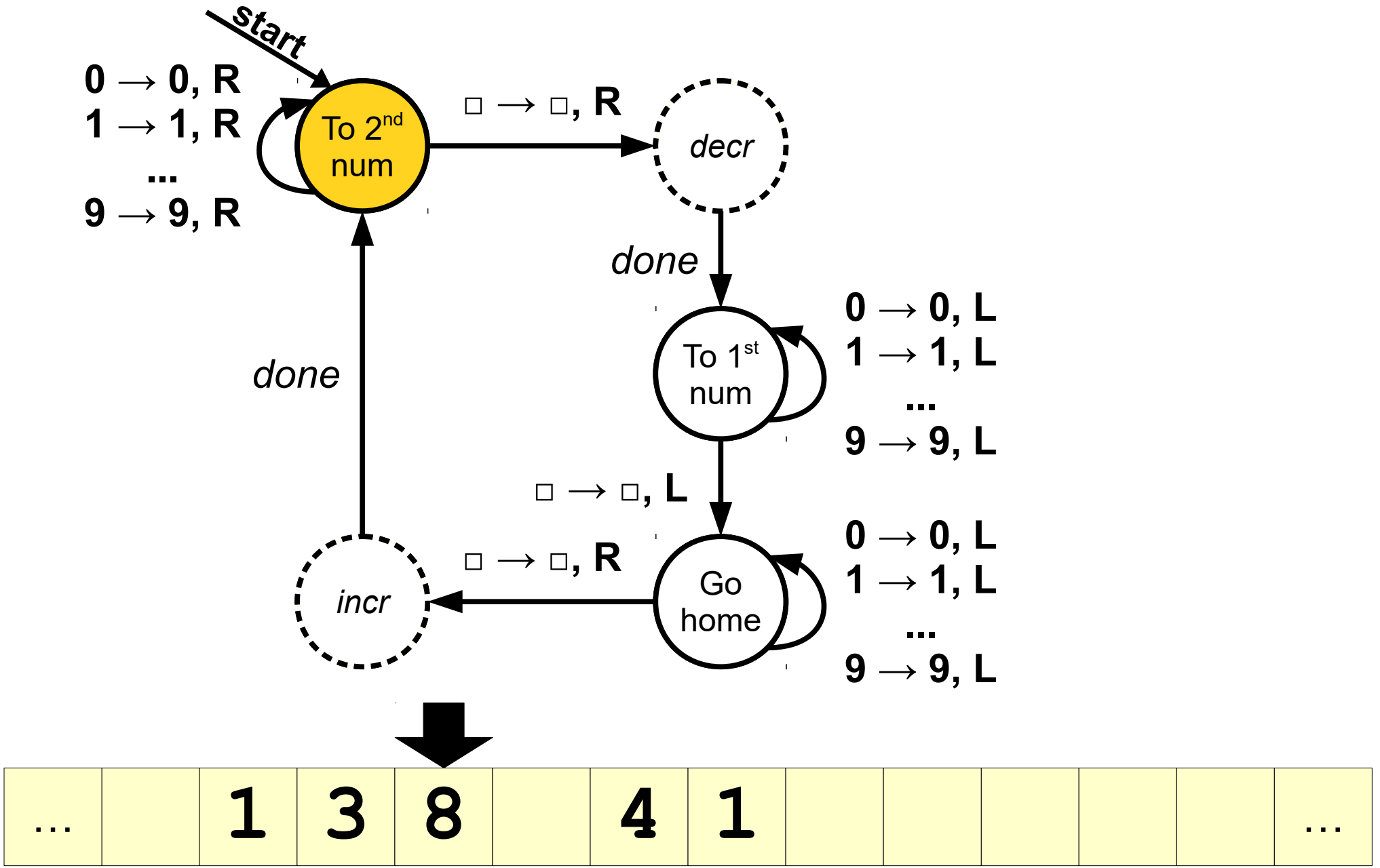

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 8 4 1 …

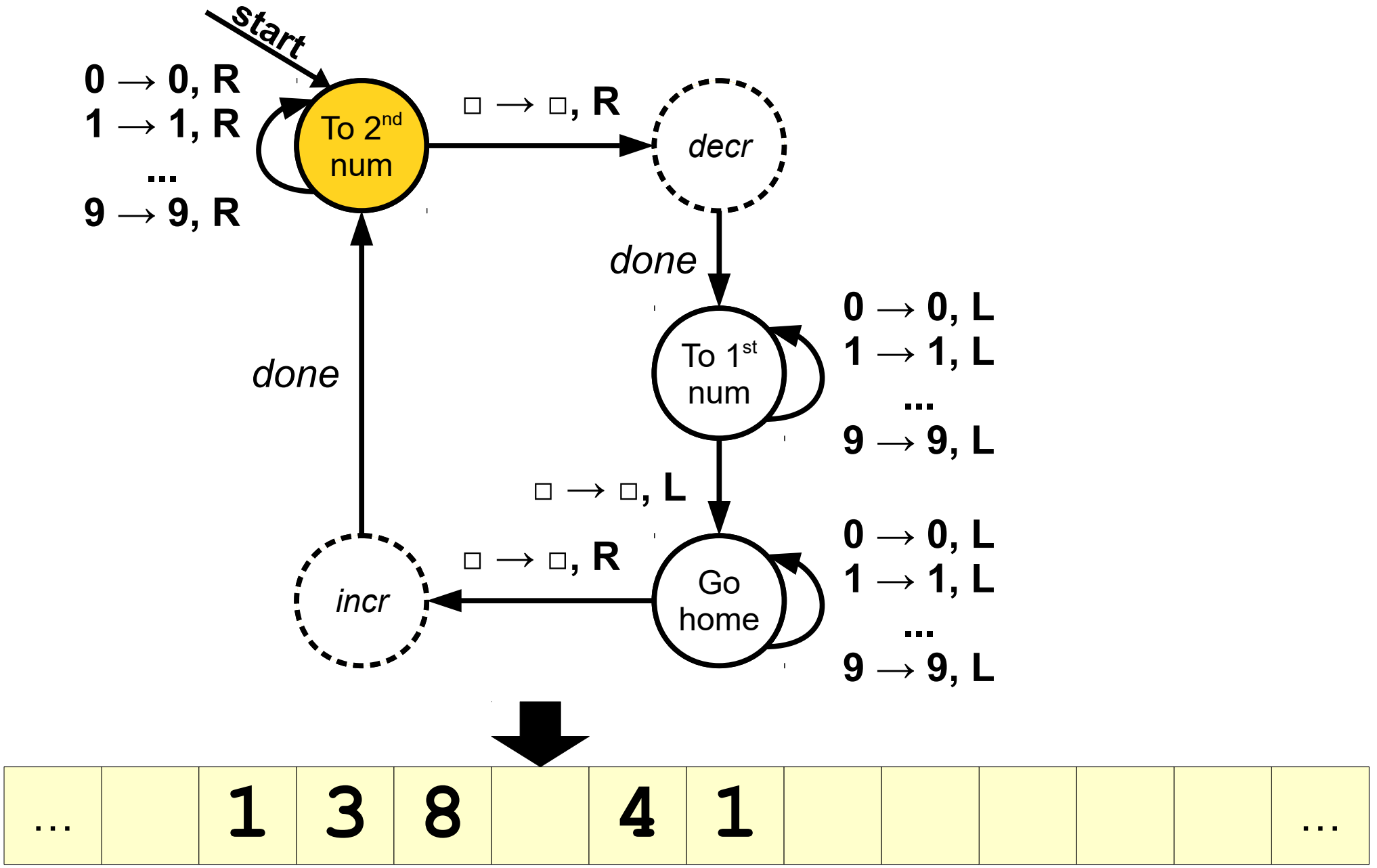

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 3 8 4 1 …

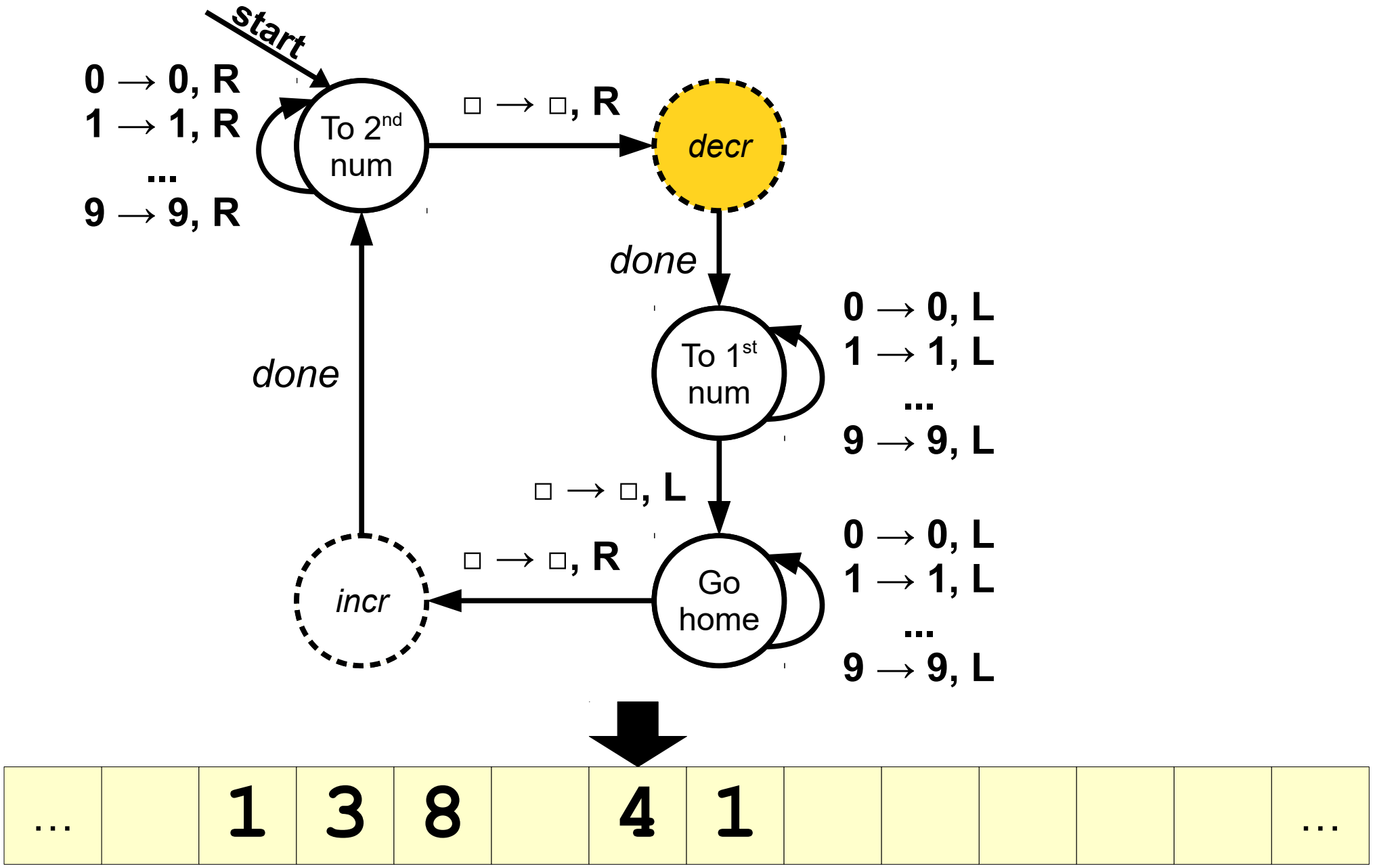

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 3 8 4 1 …

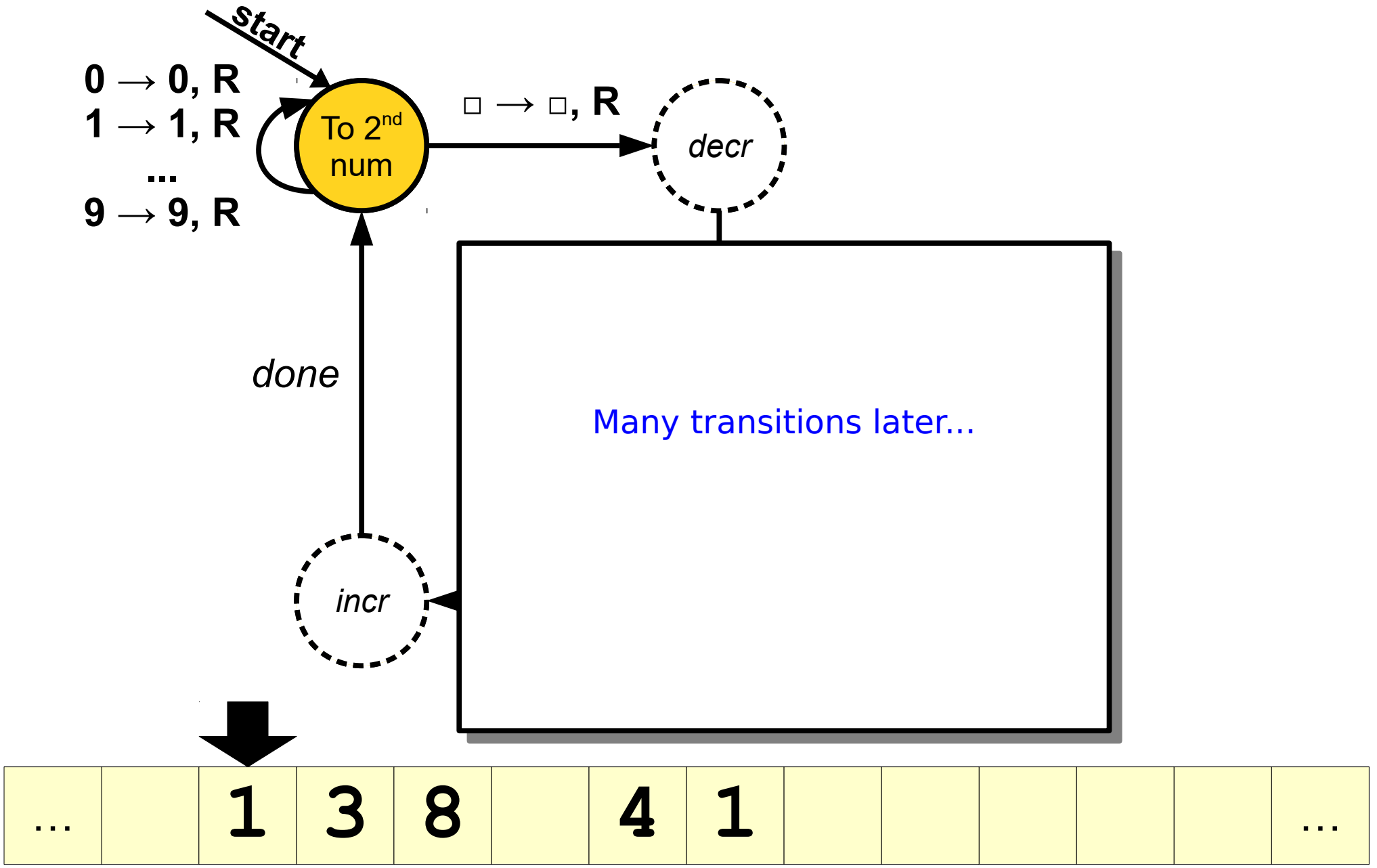

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
... 1 3 8 4 1 ...

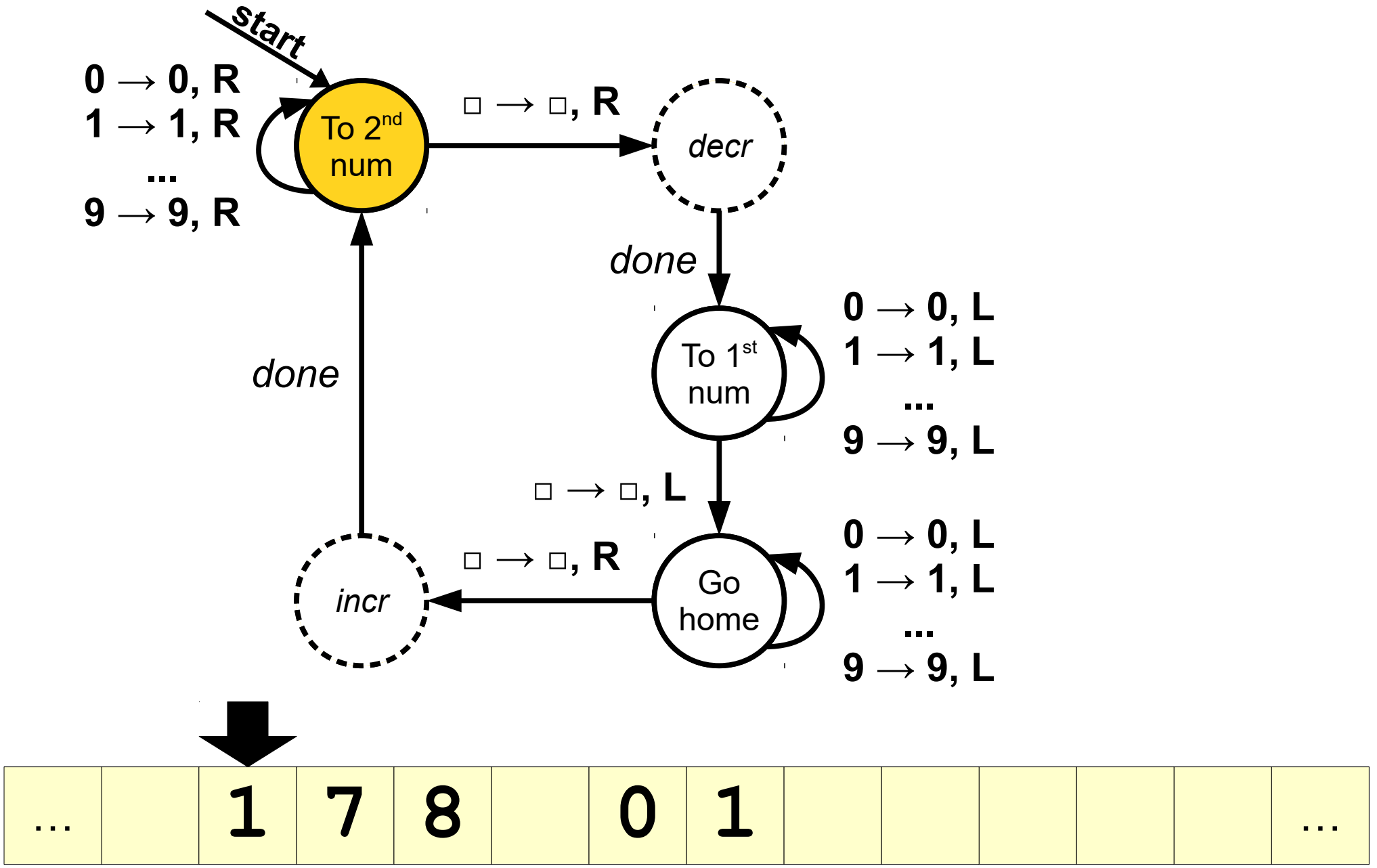

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
... 1 3 8 4 1 ...

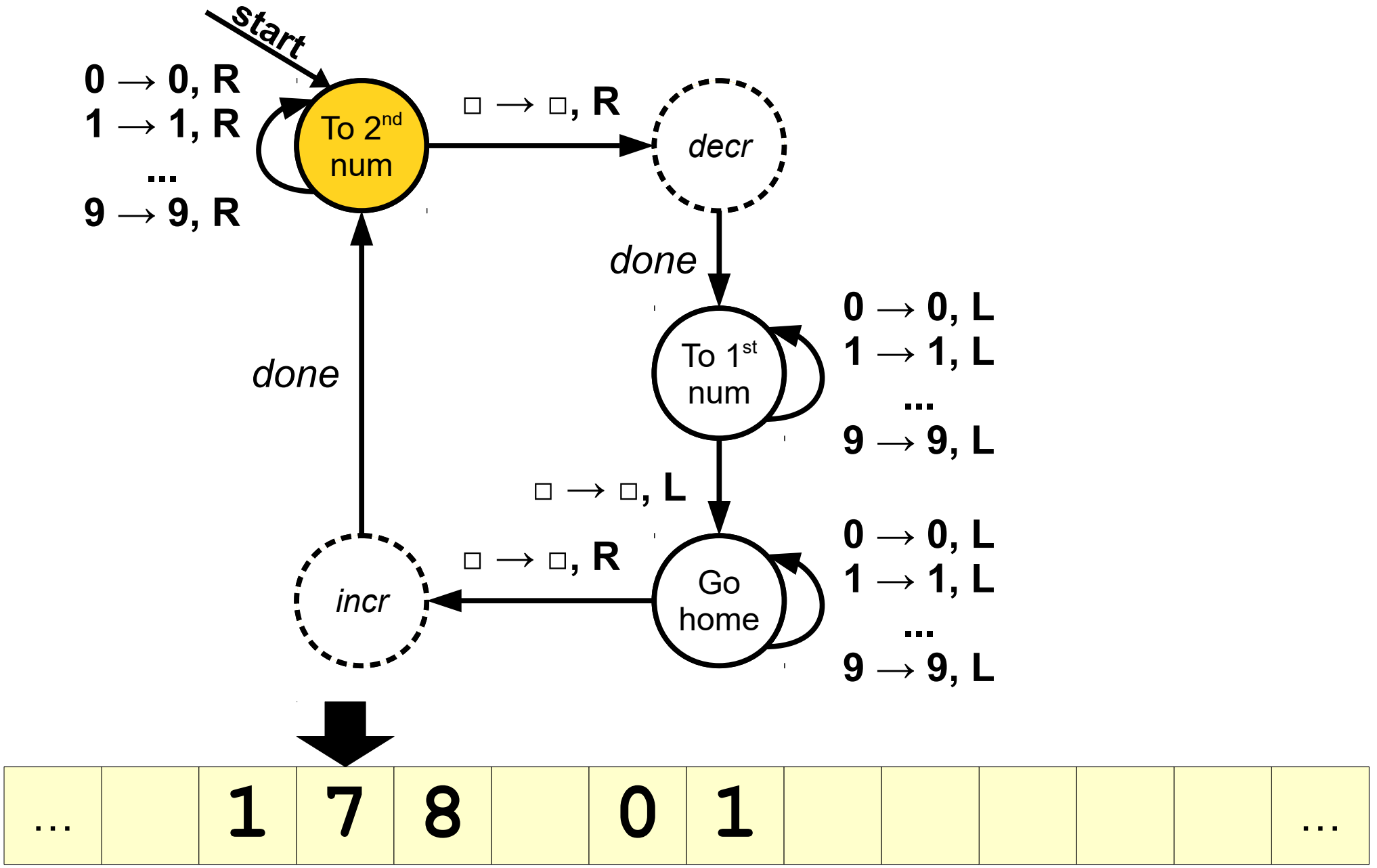

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
... 1 3 8 4 1 ...

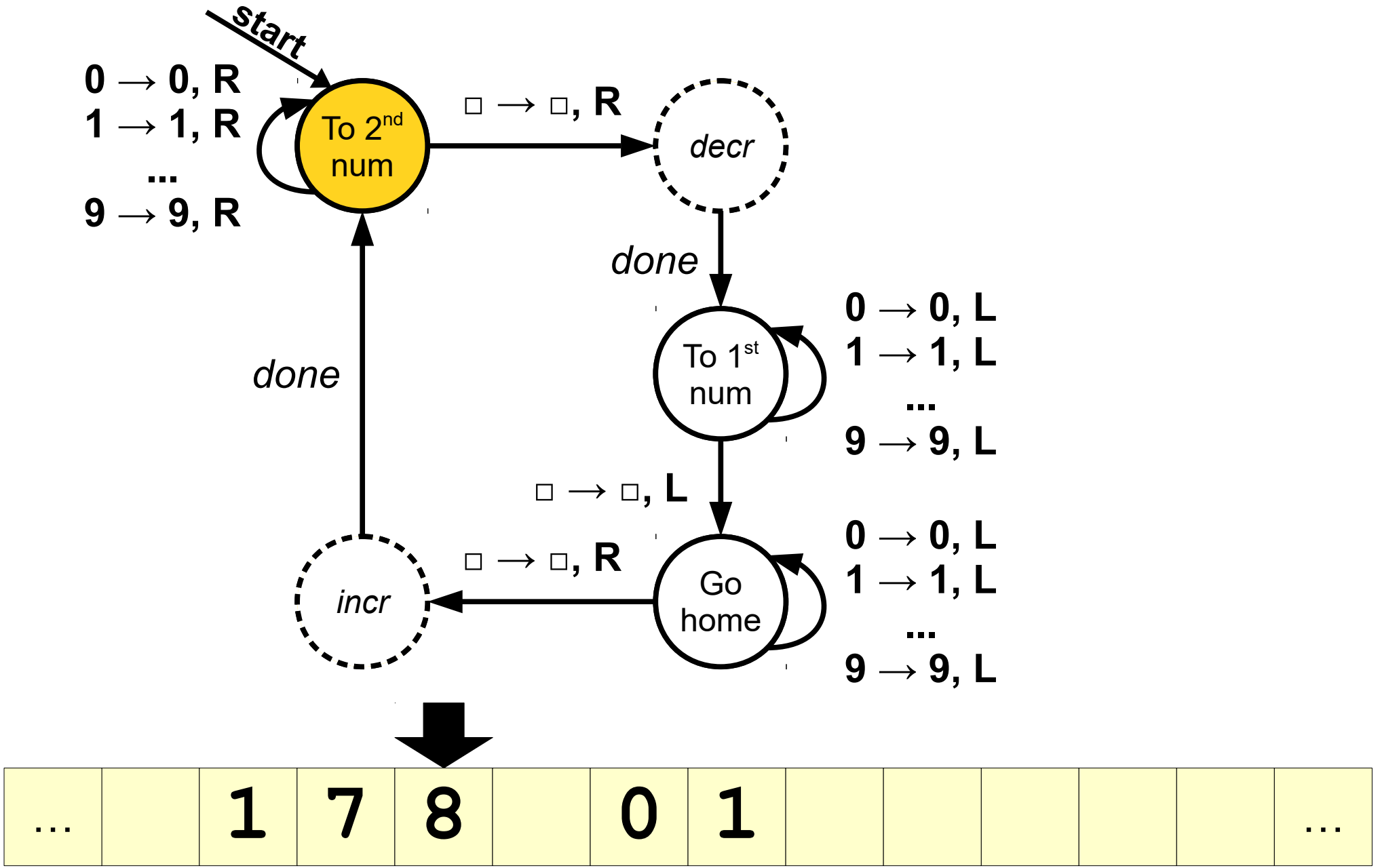

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
... 1 3 8 4 1 ...

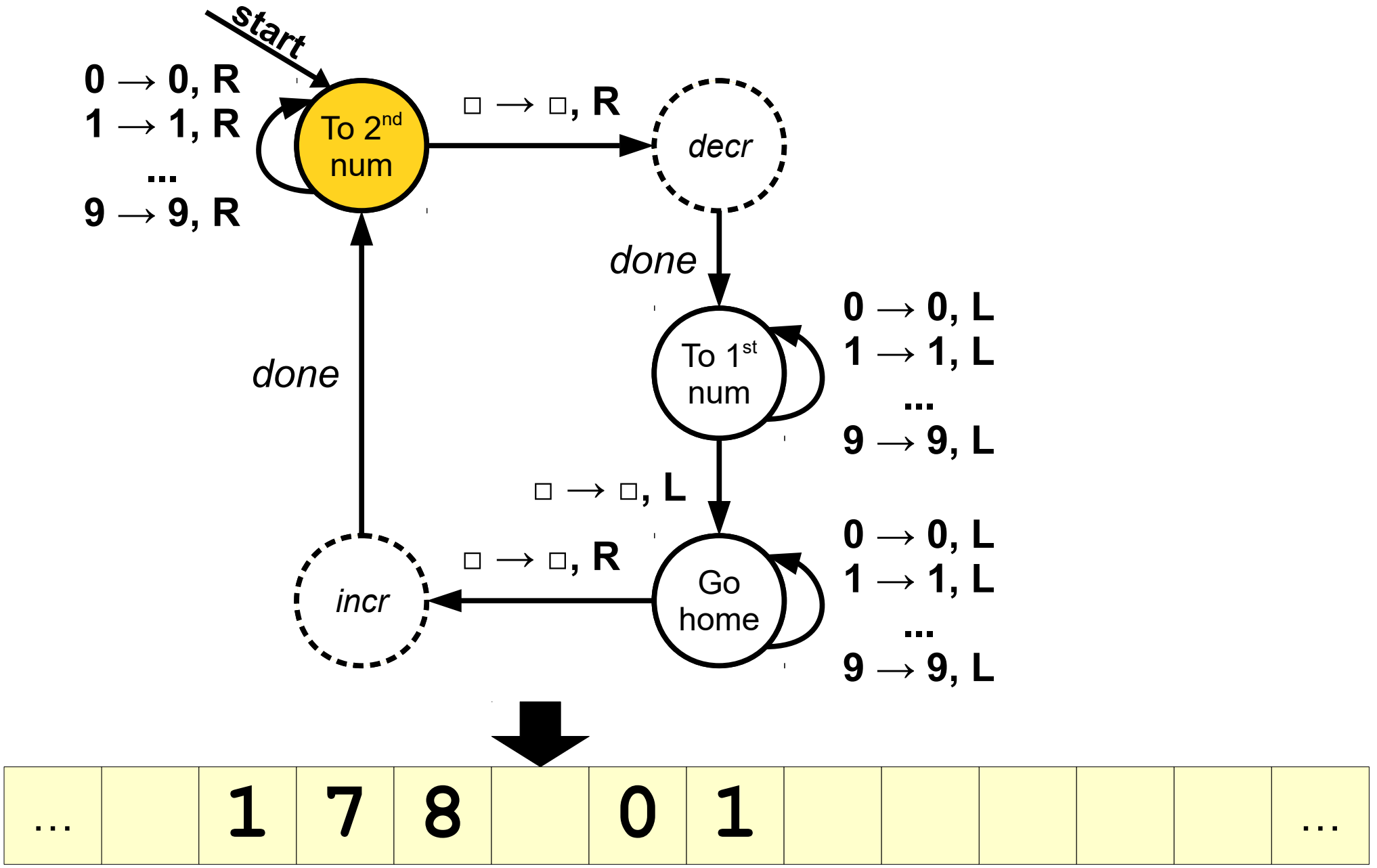

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
done
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
... 1 3 8 4 1 ...

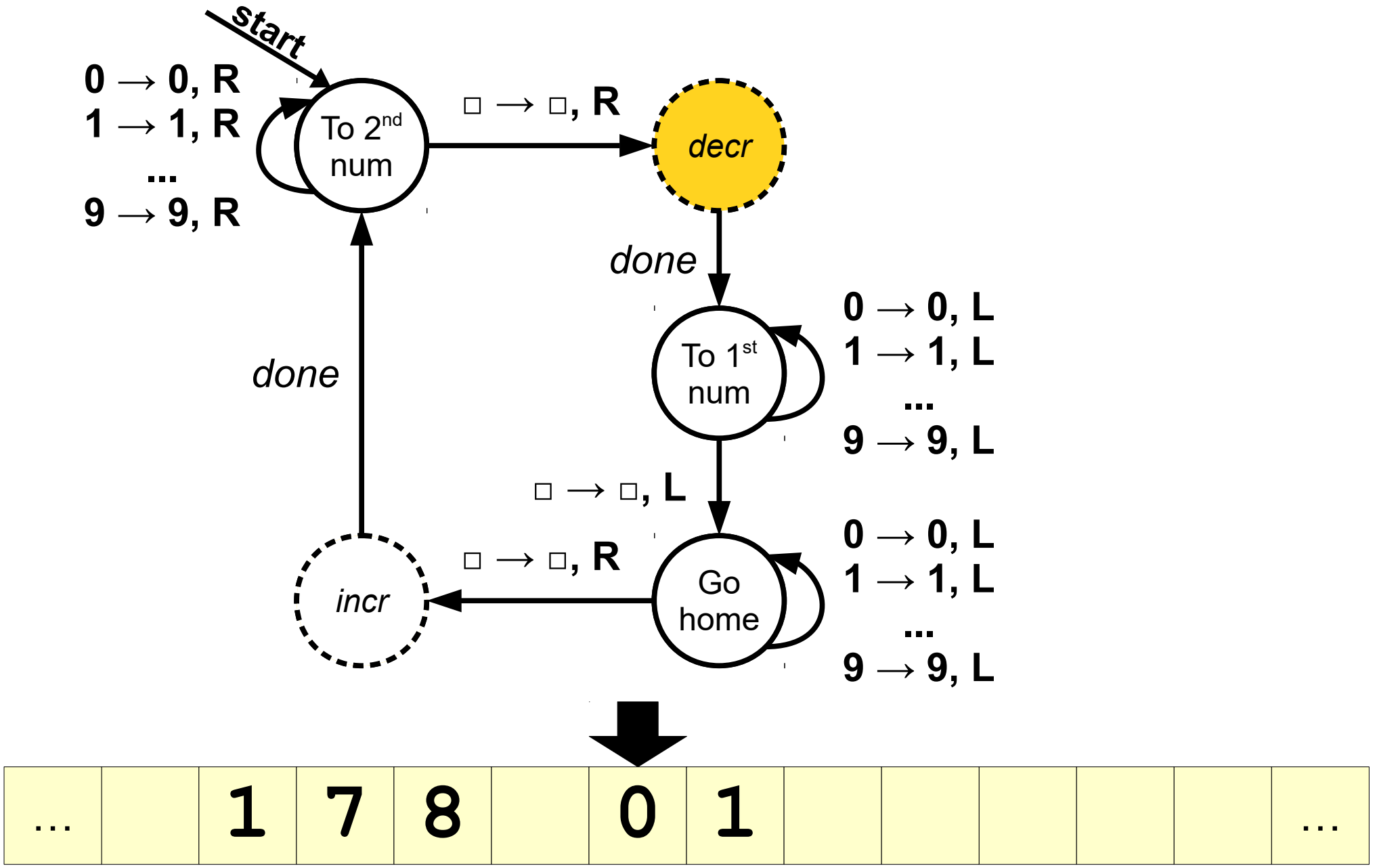

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
Many transitions later...
incr
…
1
3
8
4
1
…

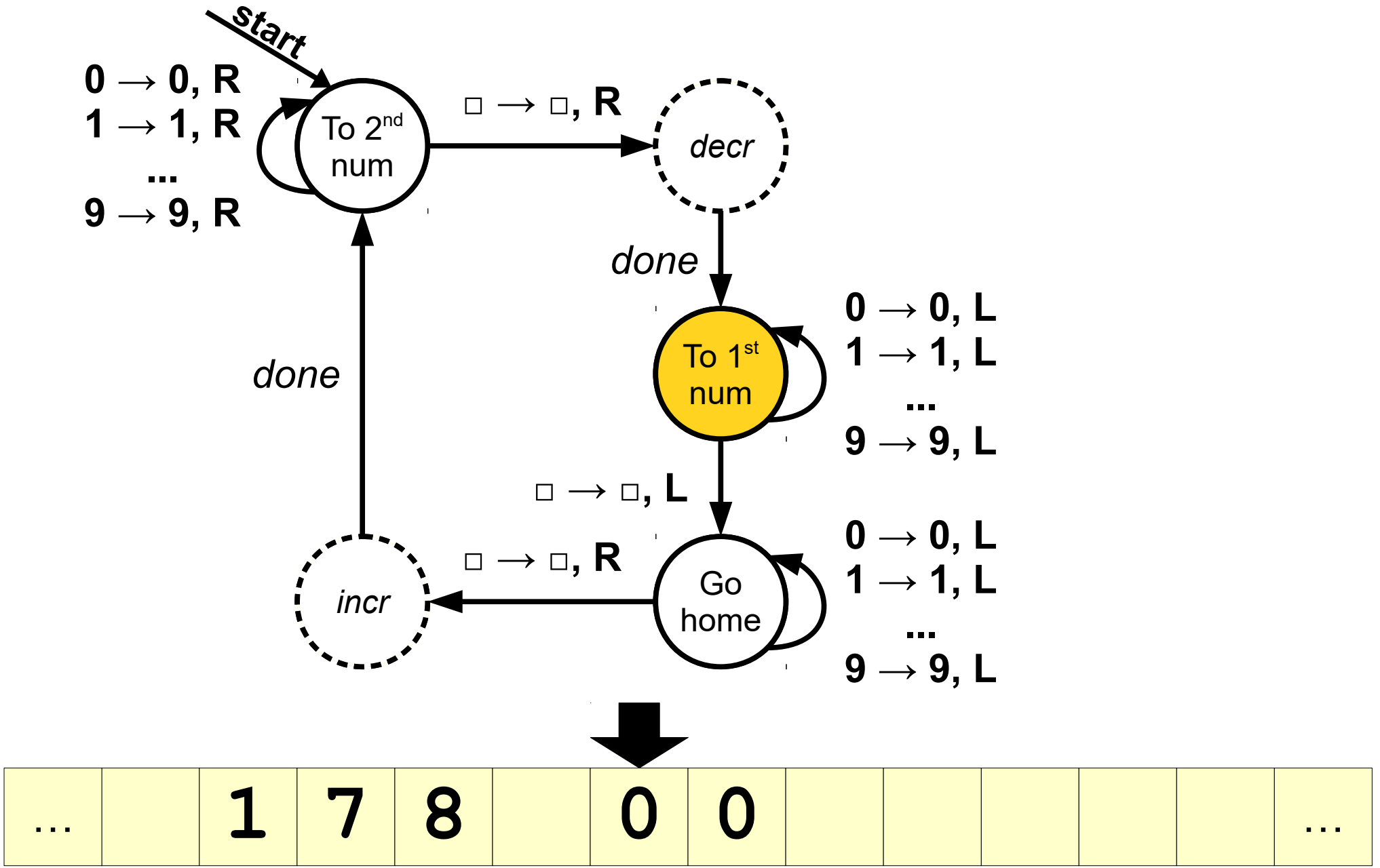

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 1 …

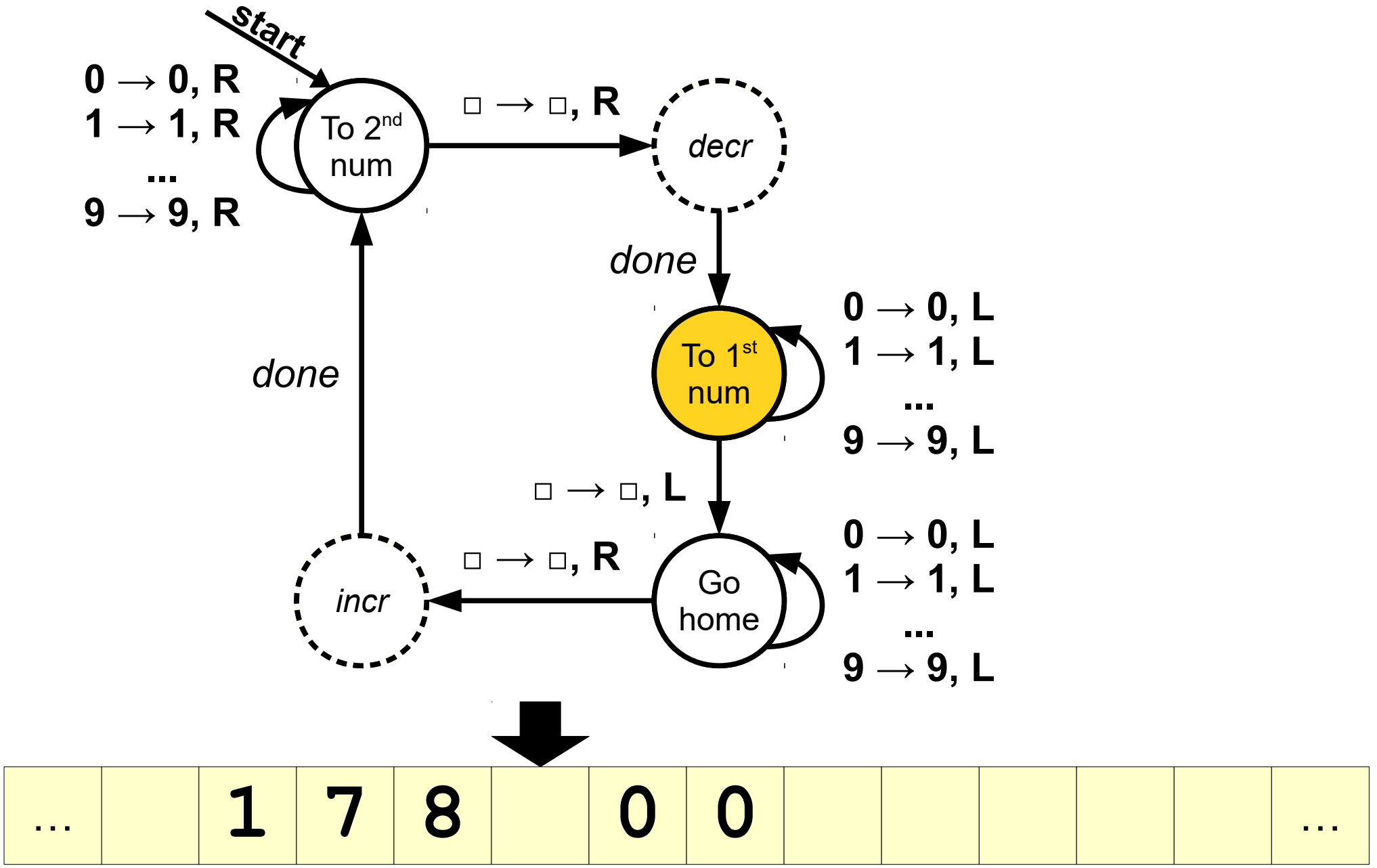

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 1 …

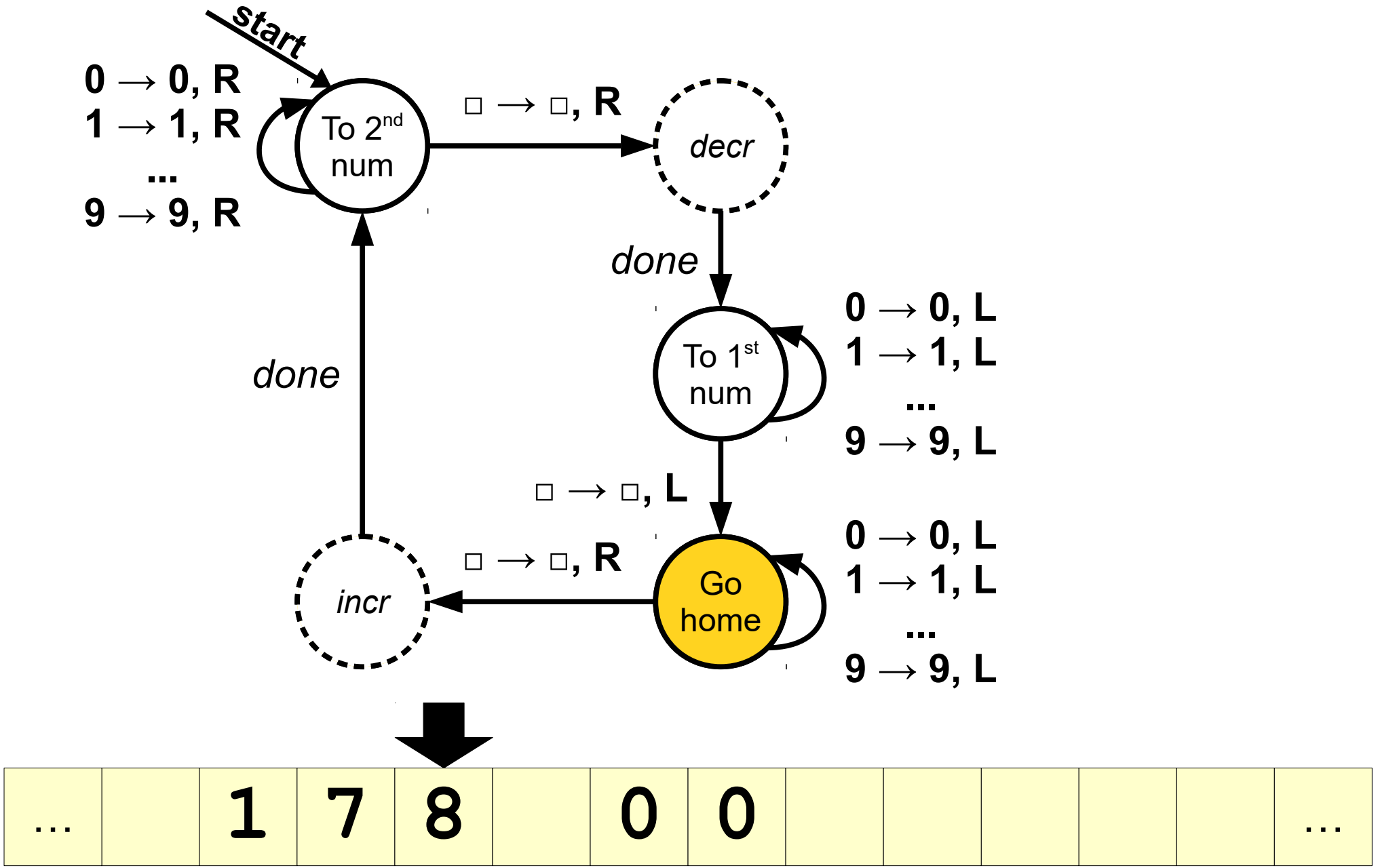

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 1 …

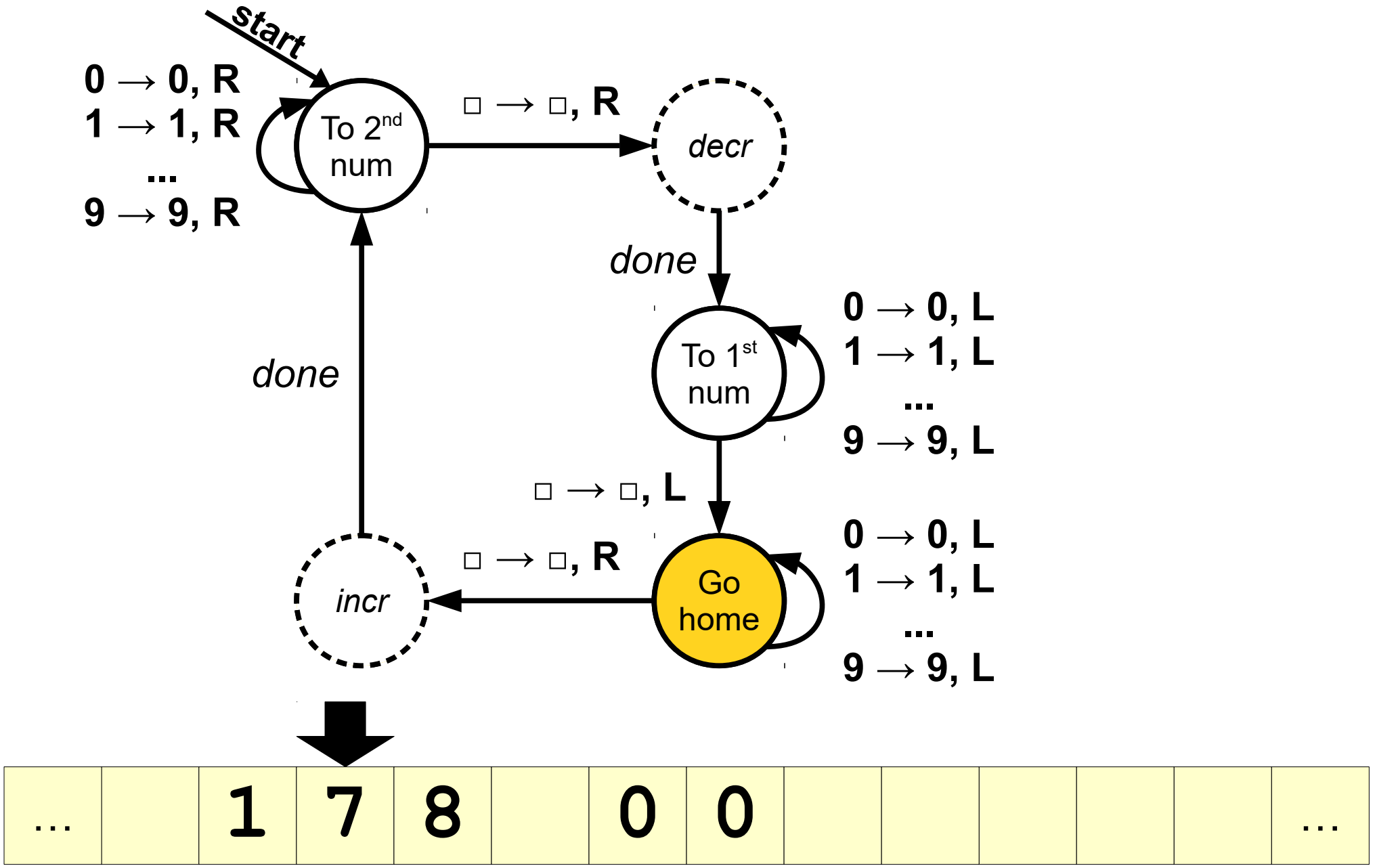

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 1 …

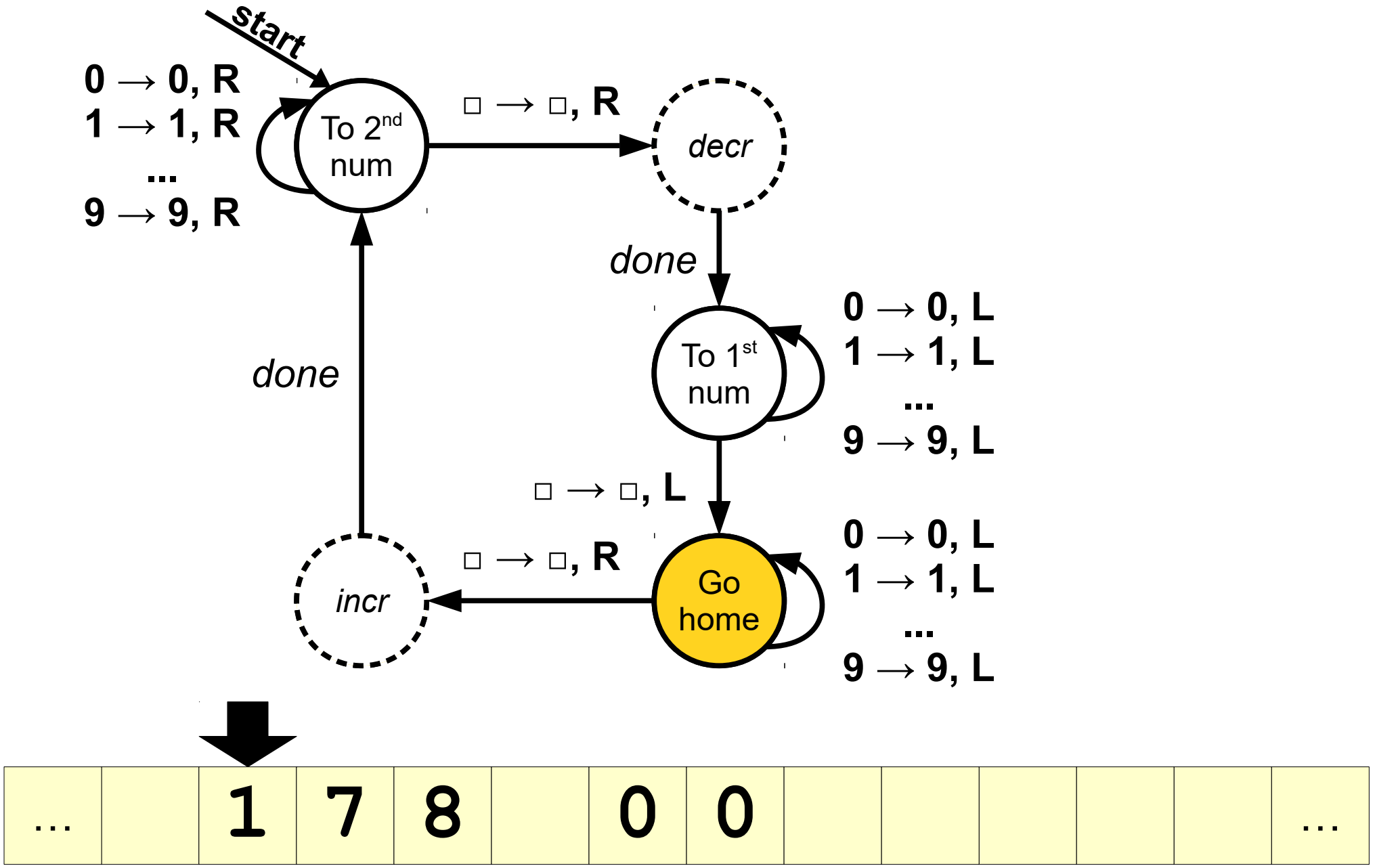

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 1 …

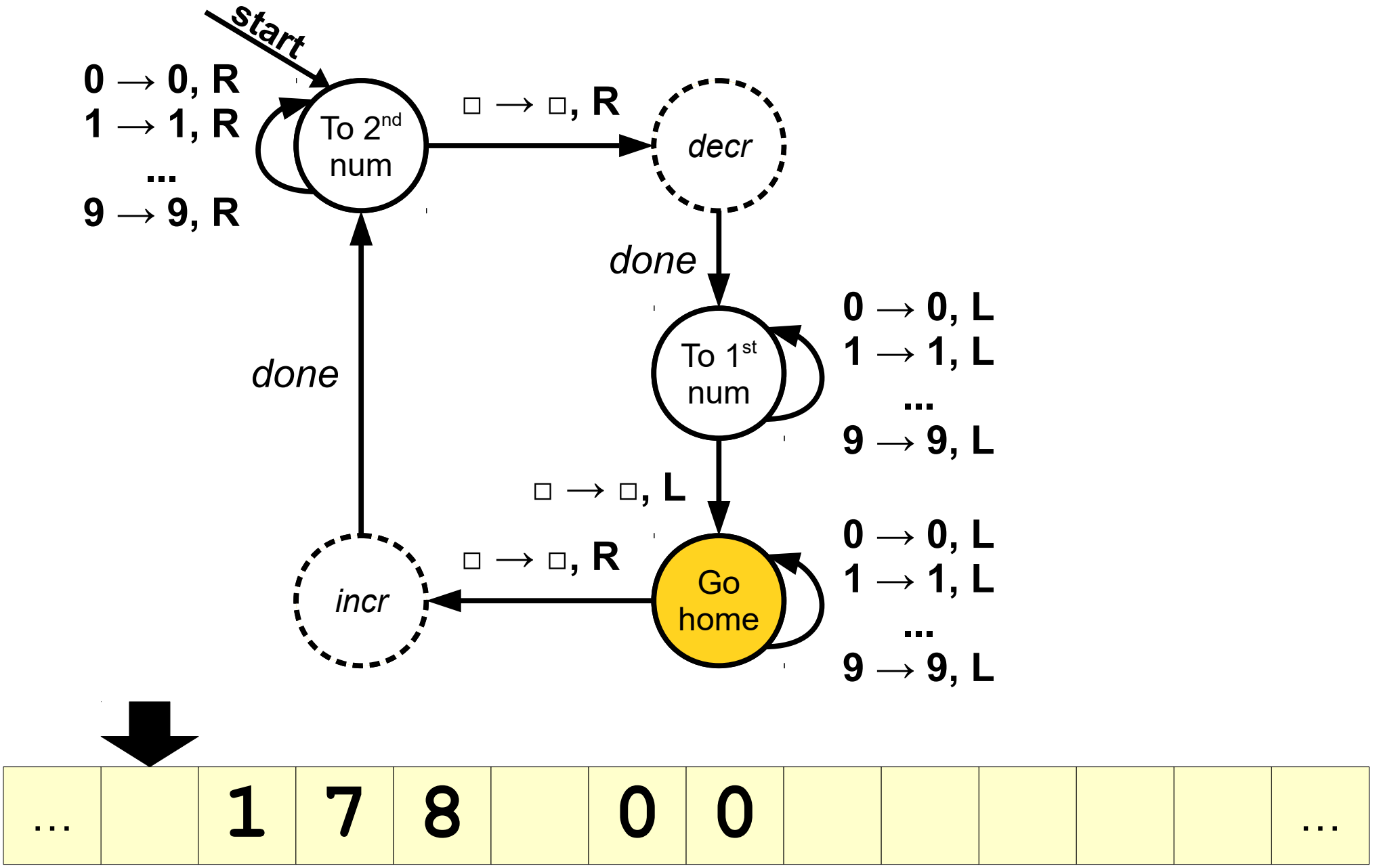

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

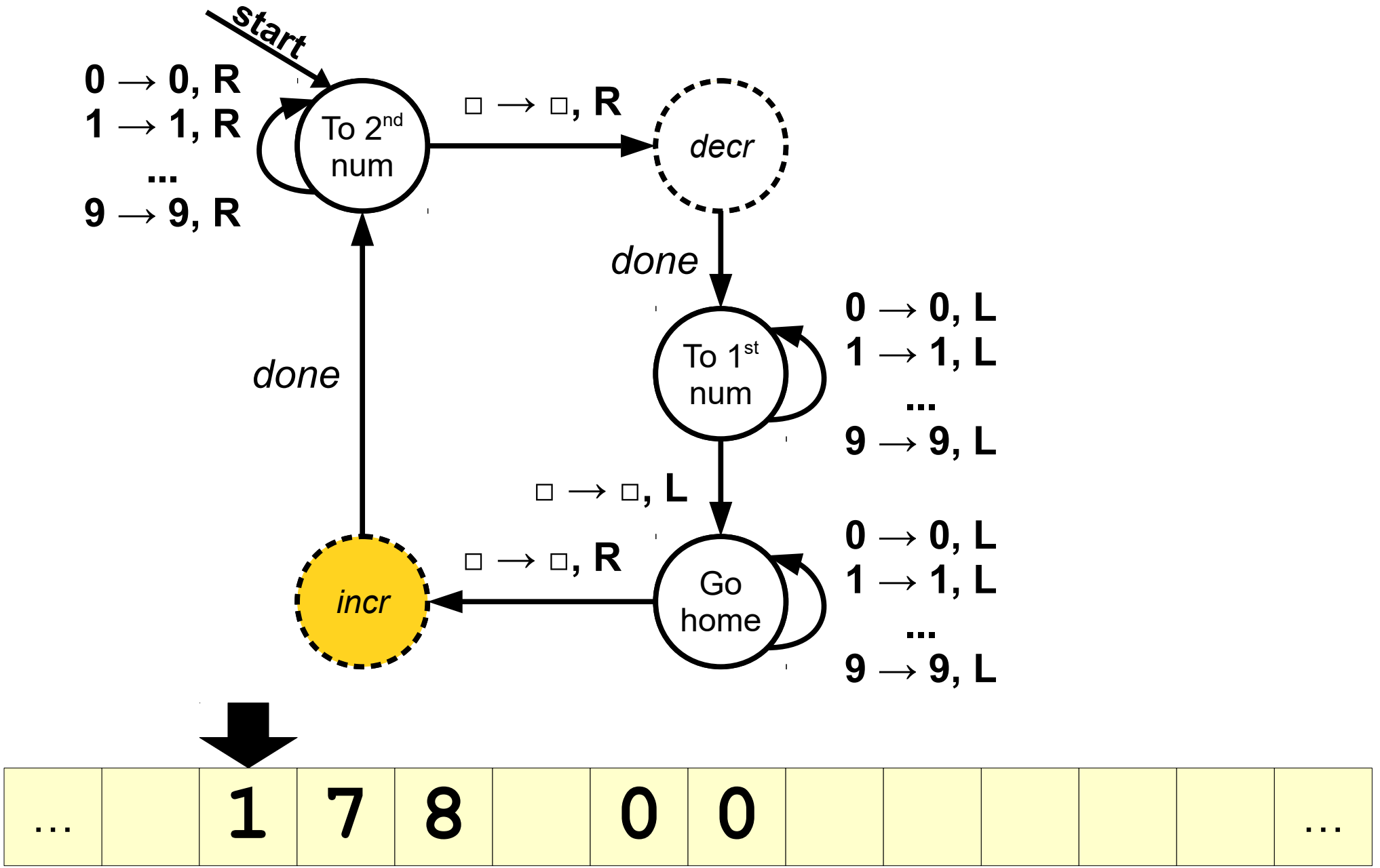

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

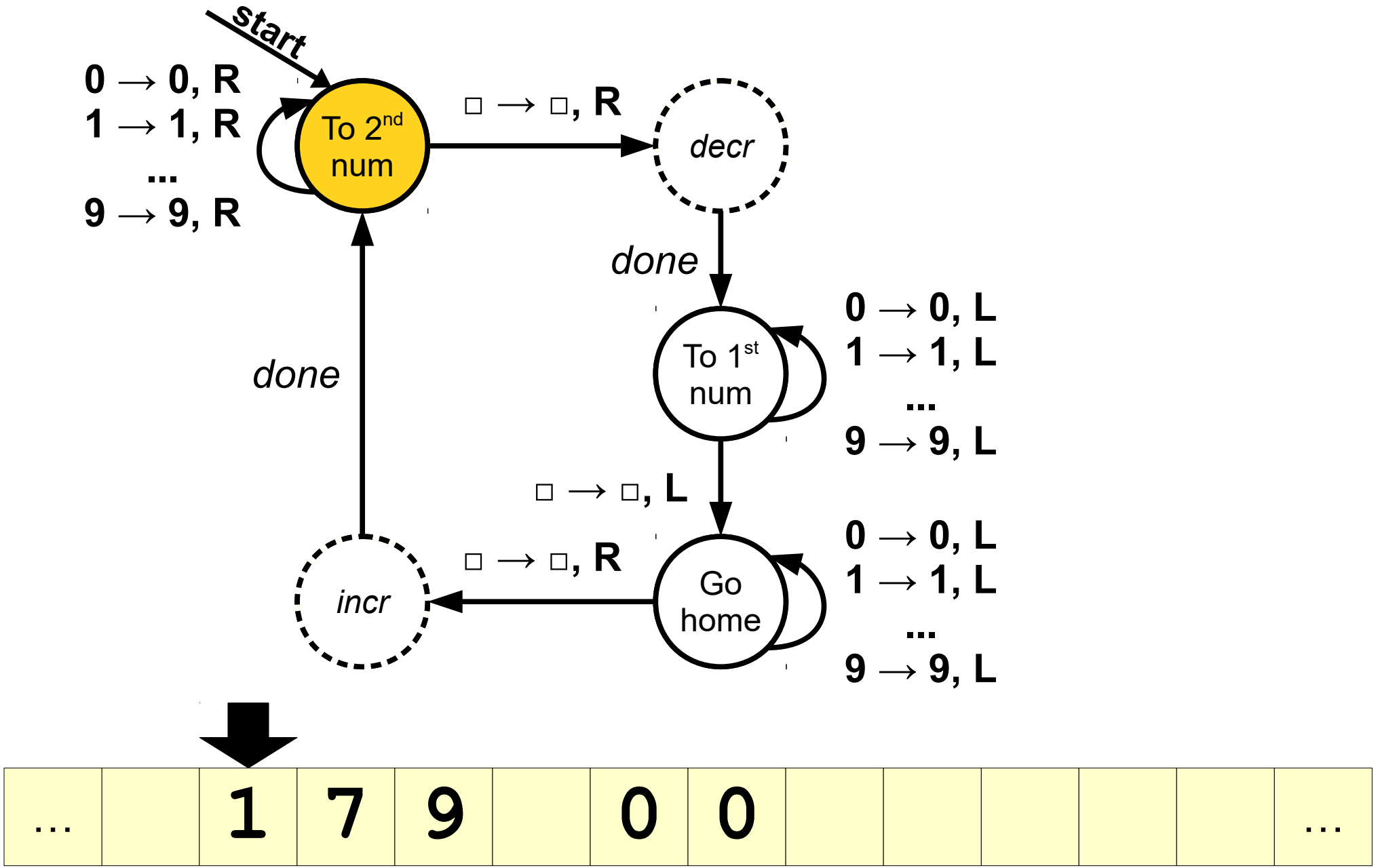

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

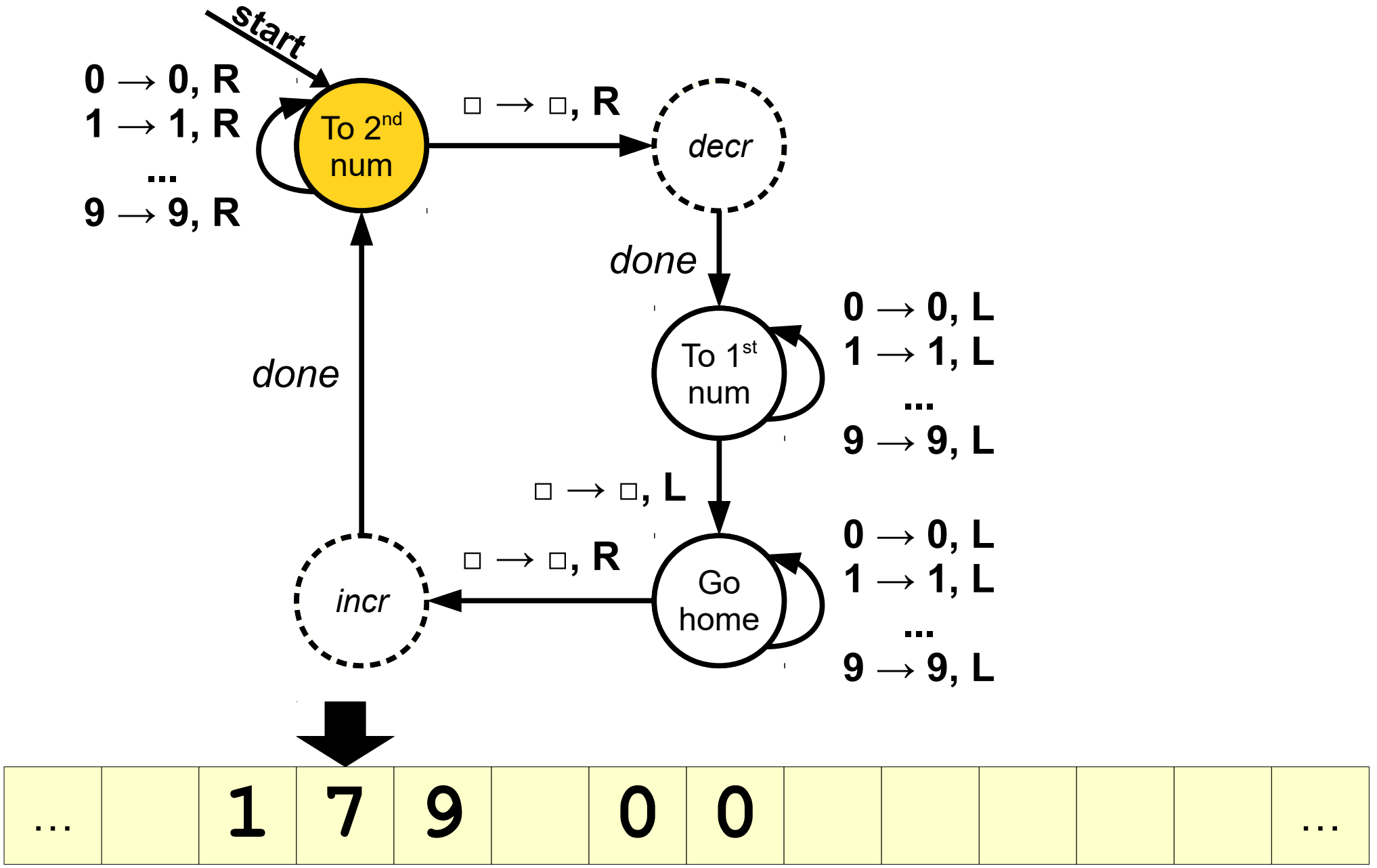

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

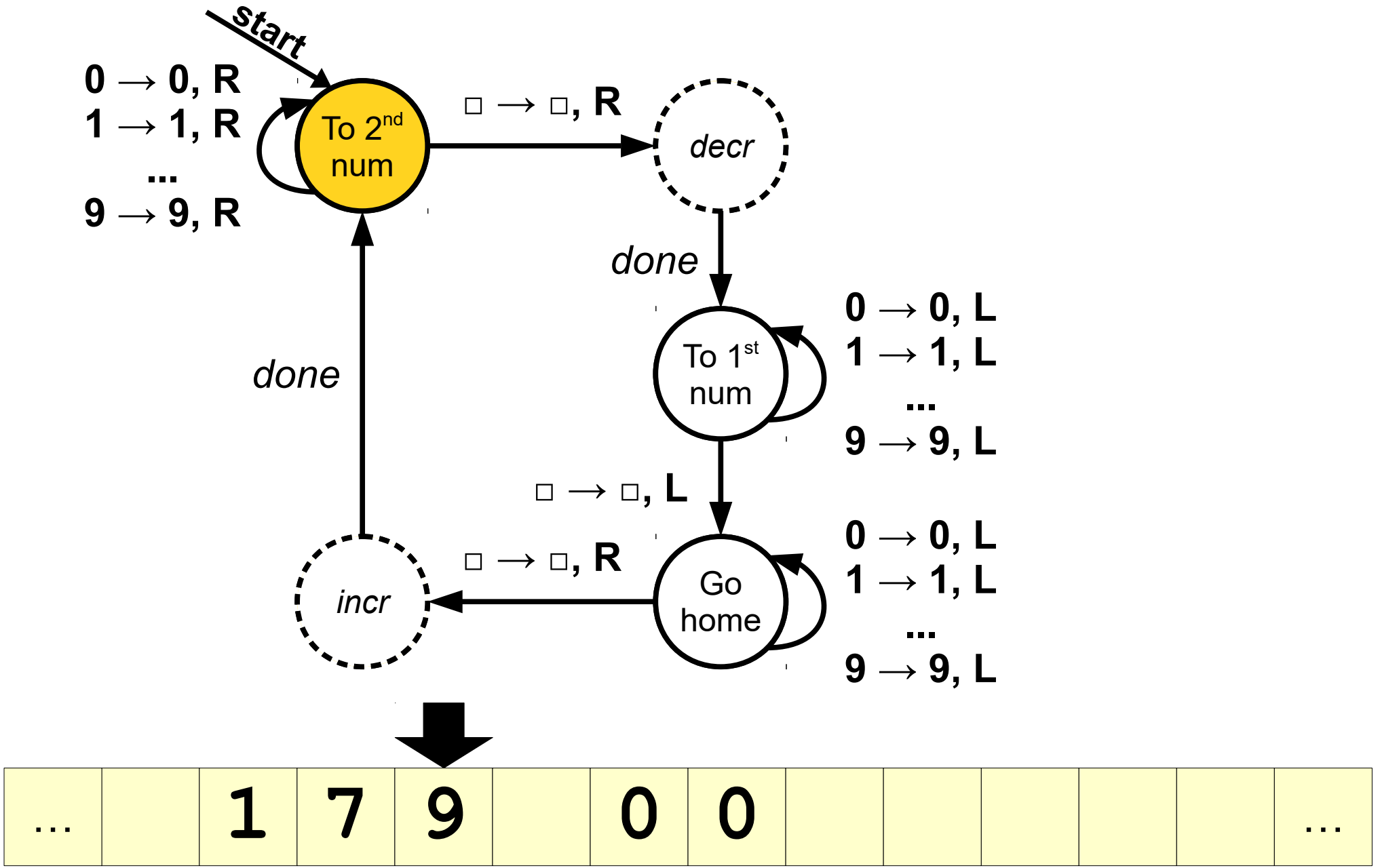

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

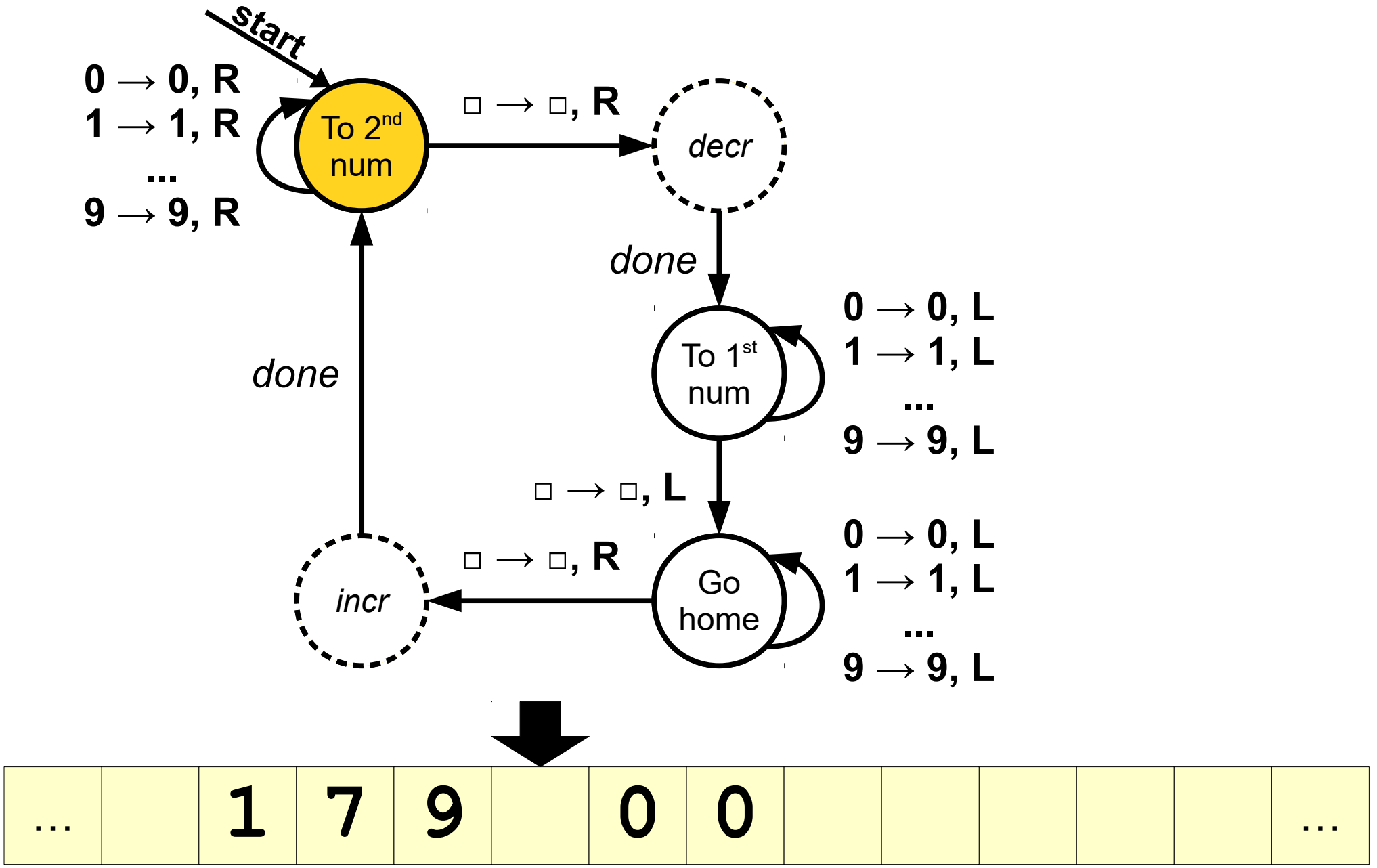

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

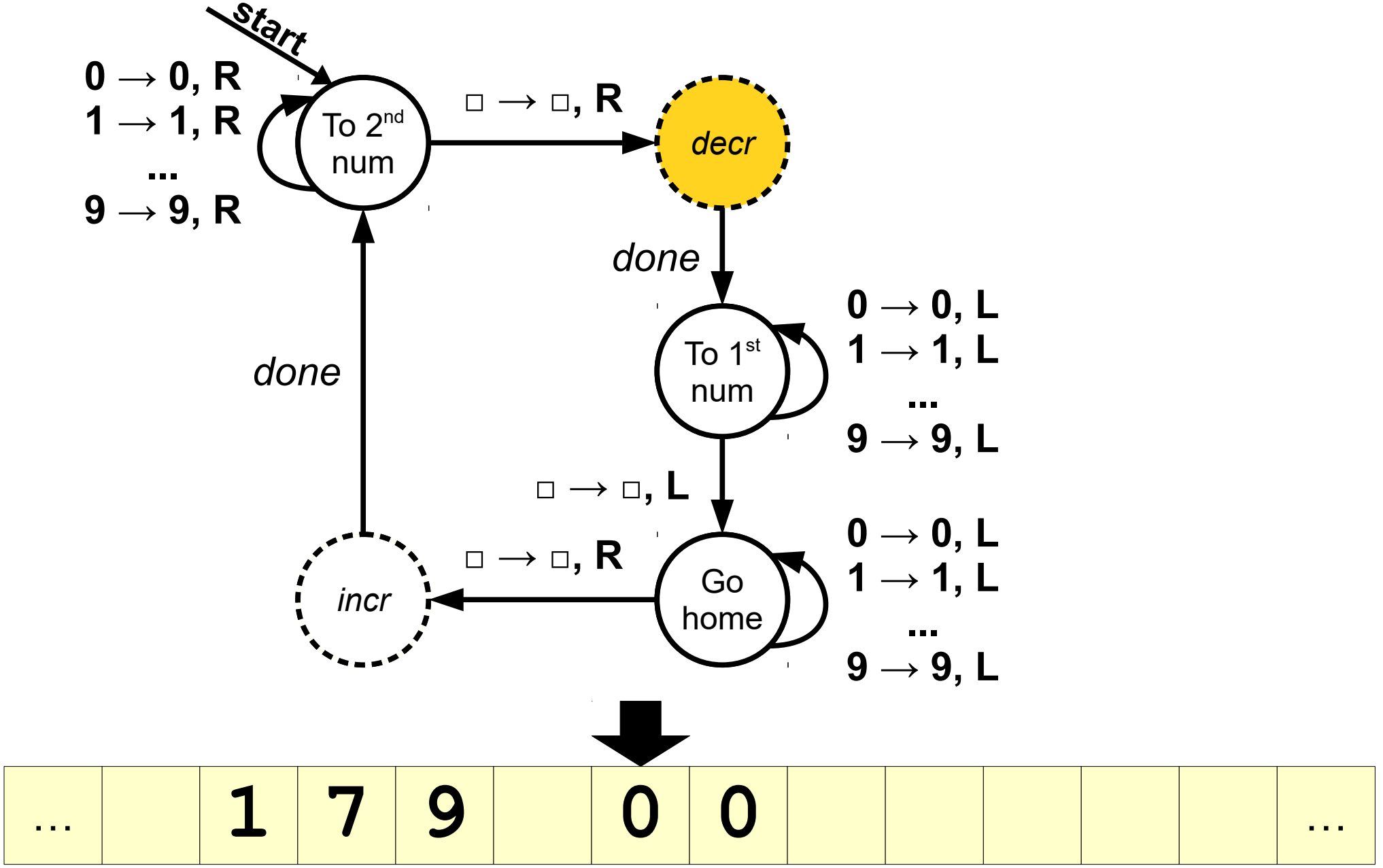

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 8 0 0 …

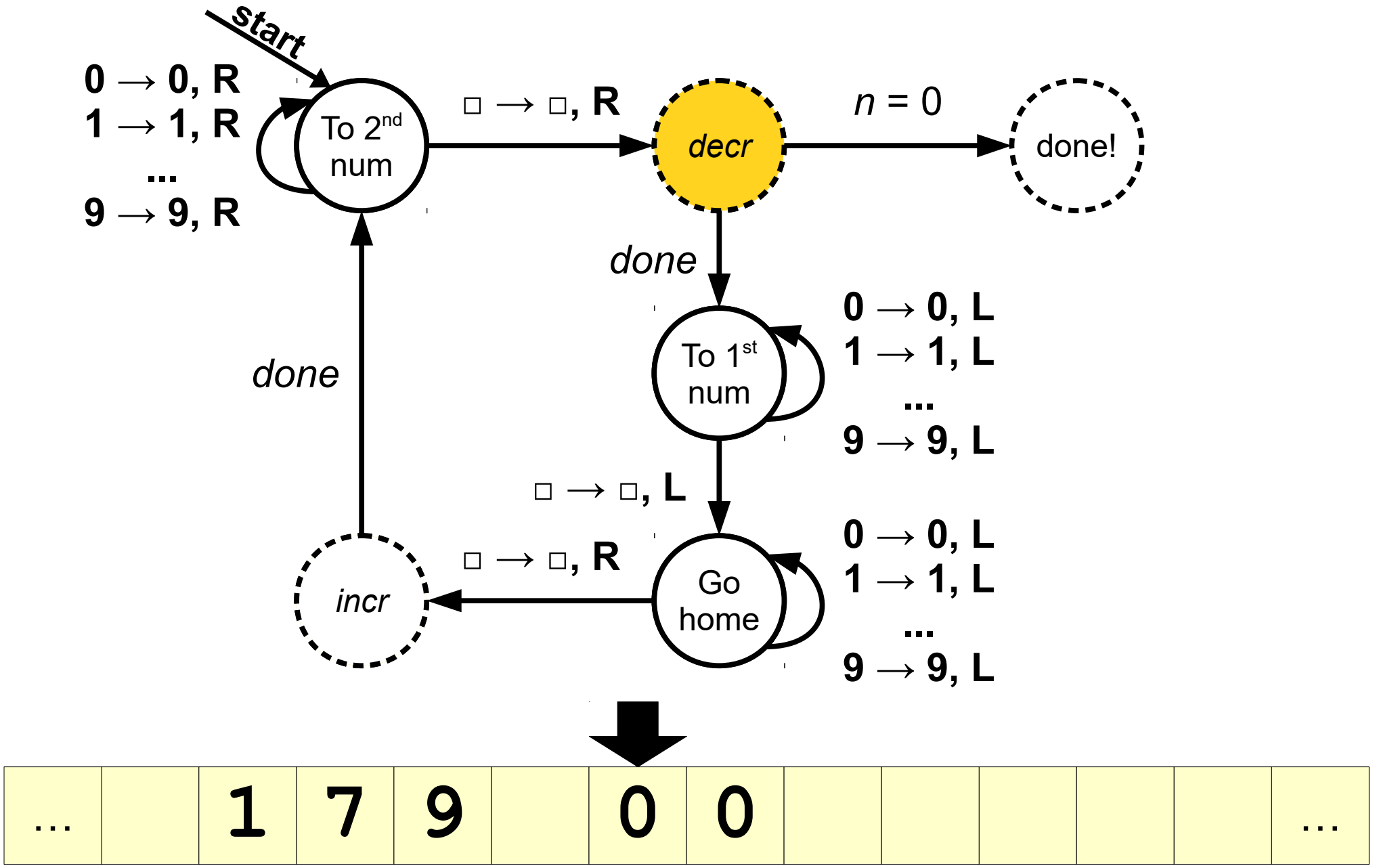

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 9 0 0 …

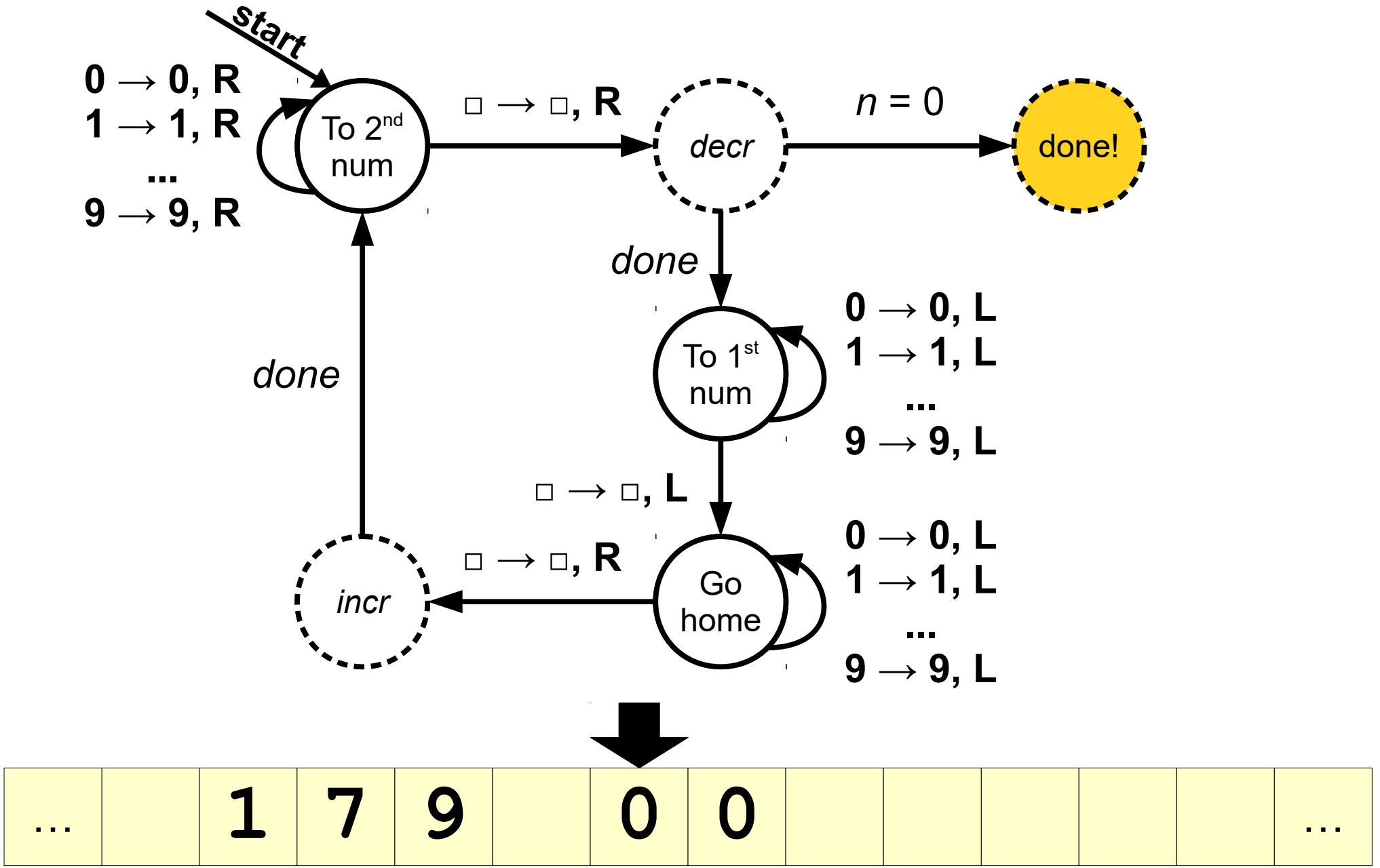

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 9 0 0 …

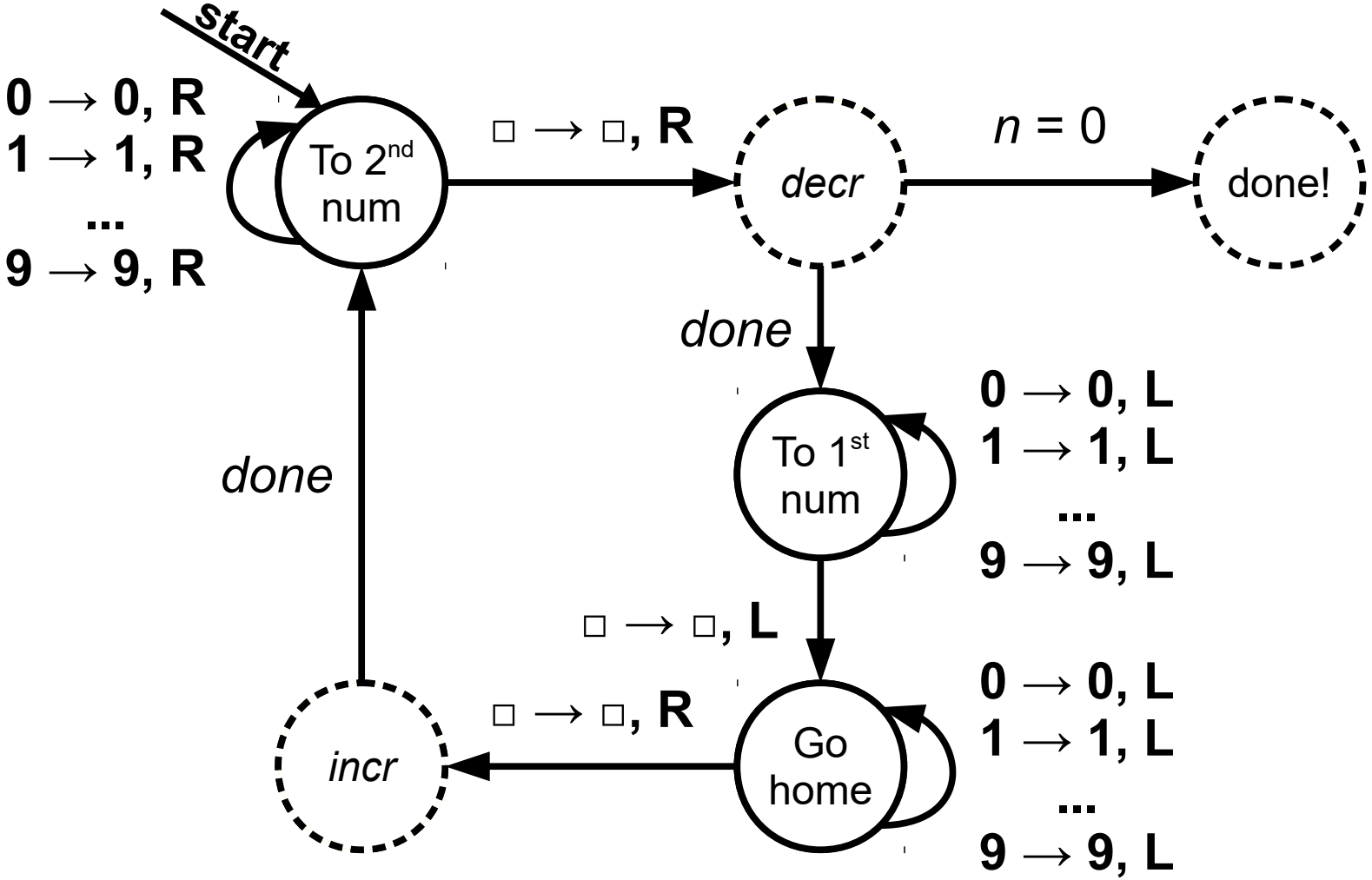

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 9 0 0 …


start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 9 0 0 …


start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done
… 1 7 9 0 0 …

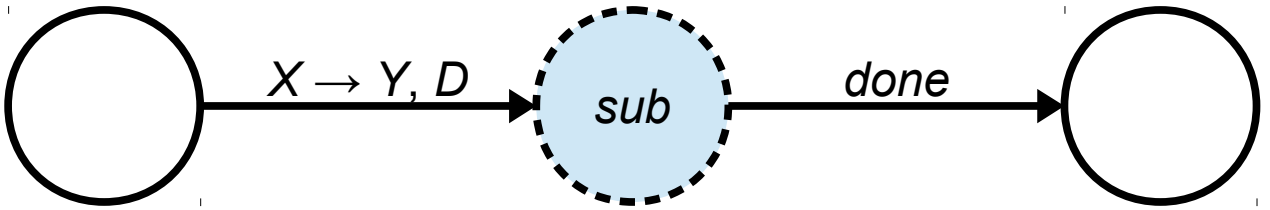

start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
n = 0
done!
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
done
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 7 9 0 0 …


start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
n = 0
done!
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
done
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
… 1 7 9 0 0 …


start
0 → 0, R
1 → 1, R
...
9 → 9, R
To 2nd num
□ → □, R
decr
n = 0
done!
done
To 1st num
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, L
Go home
0 → 0, L
1 → 1, L
...
9 → 9, L
□ → □, R
incr
done

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.

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.

Main Question for Rest of Quarter:

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 what the boundaries (if any) are.
- Let's explore their expressive power.

Three claims we need to believe in order to proceed

1. We can give a TM all kinds of inputs as a string, including groups of inputs and TMs themselves!

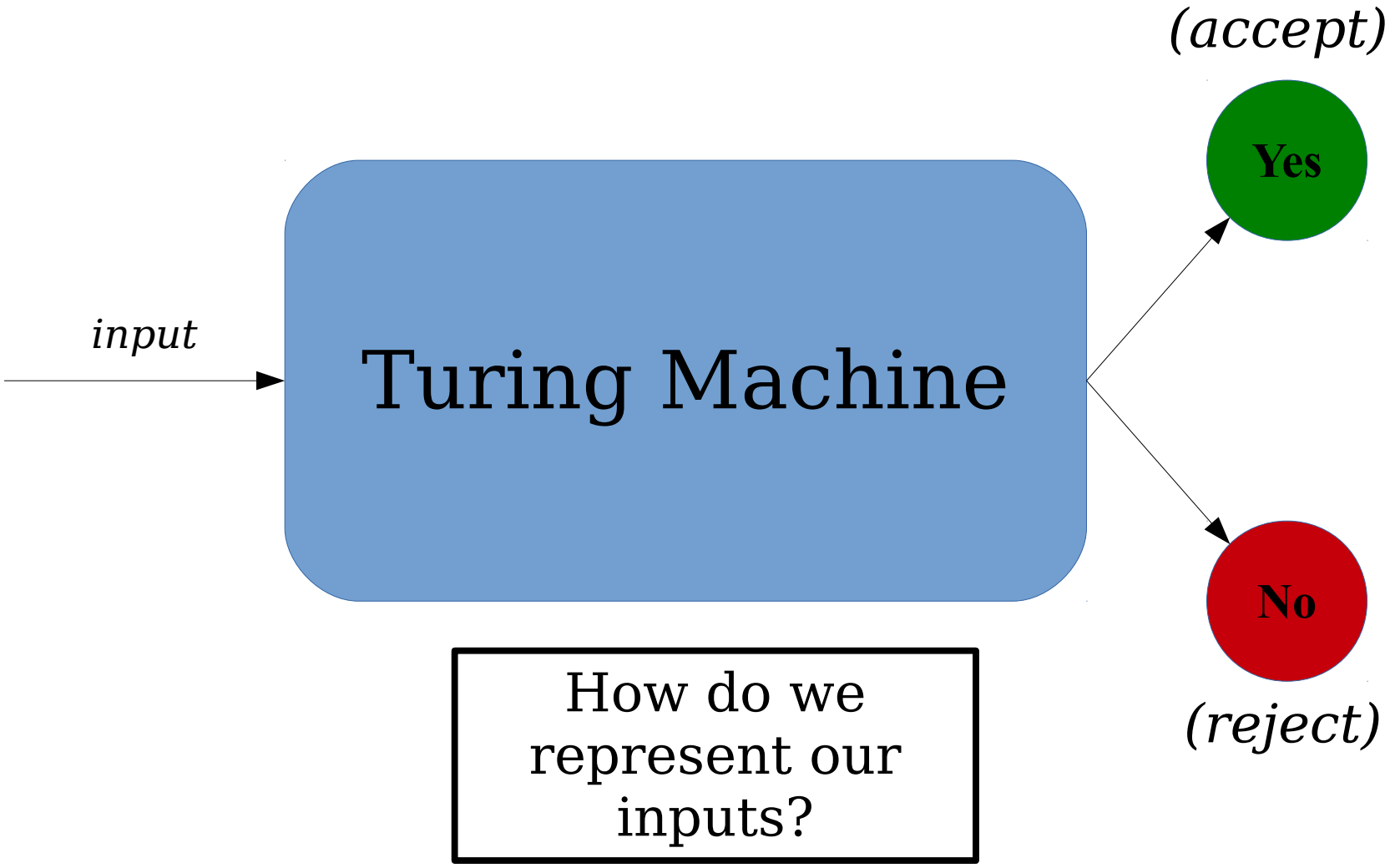
2. There is a TM that can take TMs as input and run them to see what they do. (“Universality” property)

3. That TM (from 2), or any TM that takes other TMs as input, could take itself as input. (“Self-Reference” property)

Claim 1

1. We can give a TM all kinds of inputs as a string, including groups of inputs and TMs themselves!

A Model for Solving Problems



On your computer, *everything* is numbers!

- Images (gif, jpg, png): binary numbers
- Integers (int): binary numbers
- Non-integer real numbers (double): binary numbers
- Letters and words (ASCII, Unicode): binary numbers
- Music (mp3): binary numbers
- Movies (streaming) STRANGER THINGS : binary numbers
- Doge pictures  : binary numbers
- Email messages: binary numbers

Object Encodings

- If *Obj* is some mathematical object that is *discrete* and *finite*, then we'll use the notation ***⟨Obj⟩*** to refer to some way of encoding that object as a string.
- Think of ⟨*Obj*⟩ like a file on disk – it encodes some high-level object as a series of characters.

⟨  ⟩ = 1101110010111011110001001 1…110

Object Encodings

- For the purposes of what we're going to be doing, we aren't going to worry about exactly how objects are encoded.
- For example, we could say $\langle G_{0n1n} \rangle$ to mean "some encoding of a Context-Free Grammar for the language 0^n1^n" without worrying about exactly how a grammar is encoded.
  - ***Intuition check*:** could I type up a grammar and save it as a file on my computer? Yes? Ok then, we know we can put a grammar into a string.
- **As long as we're convinced a thing could be saved in a file**, we don't spend time specifying the exact file format in our proof (some proofs do in certain circumstances where it might be questioned where it's possible, but in this class we won't).

Encoding Groups of Objects

- Given a group of objects Obj_1, Obj_2, …, Obj_n, we can create a single string encoding all these objects.
  - Think of it like a .zip file!
- We'll denote the encoding of all of these objects as a single string by $\langle Obj_1, \ldots, Obj_n \rangle$.
  - This lets us feed a group of inputs into our computational device as a single input.

Claim 2

2. There is a TM that can take TMs as input and run them to see what they do.

An Observation

- When we've been discussing Turing machines, we've talked about designing specific TMs to solve specific problems.
- Does this match your real-world experiences? Do you have one computing device for each task you need to perform?

Can there be a program that simulates what another program does?

- Sure.
- These programs go by many names:
  - An ***interpreter***, like the Java Virtual Machine or most implementations of Python.
  - A ***virtual machine***, like VMWare or VirtualBox, that simulates an entire computer.

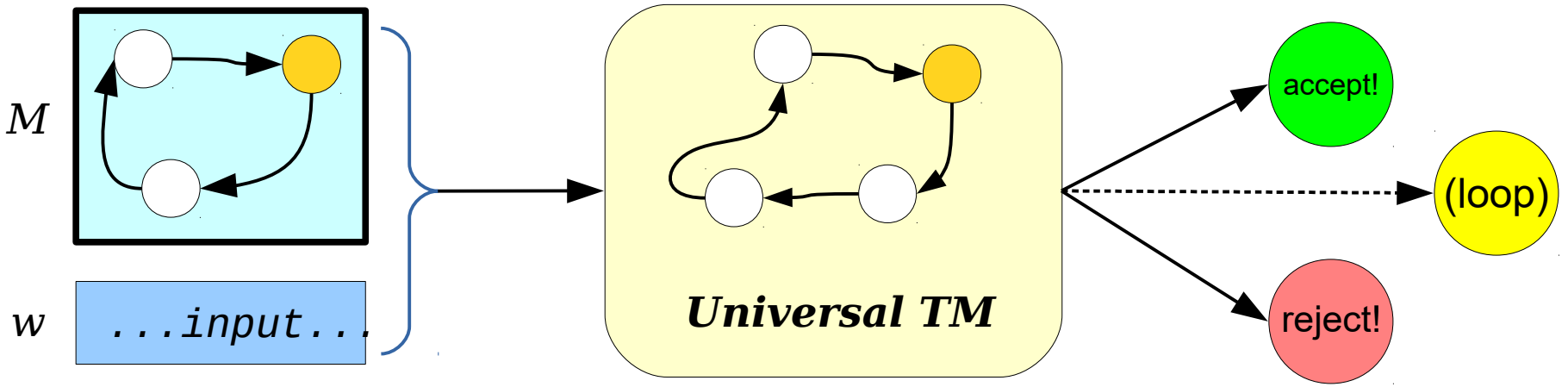
Can there be a TM that simulates what another TM does?

- Sure.
- You could imagine a TM taking a TM table like this as input, and then seeing how it would perform on various sample inputs by simulating its behavior by referring to the table at each step.

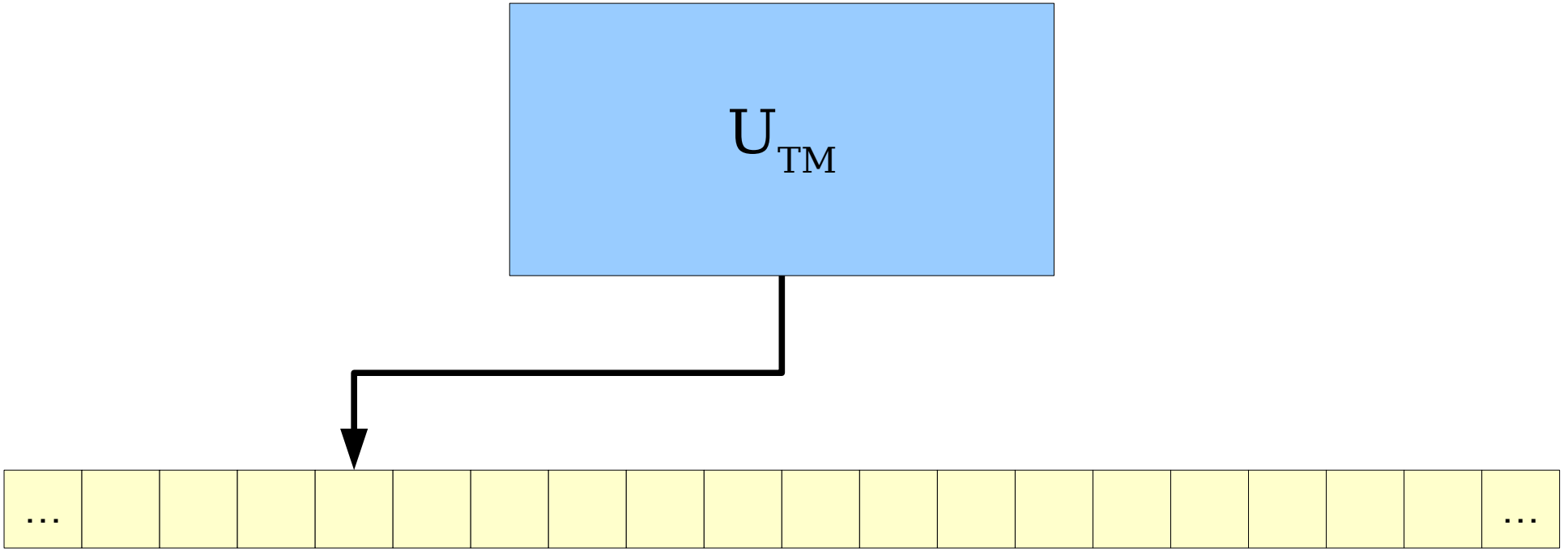
	0			**1**			**□**		
q_0	q_1	□	**R**	q_r	□	**R**	q_a	□	**R**
q_1	q_1	**0**	**R**	q_1	**1**	**R**	q_2	□	**L**
q_2	q_r	**0**	**R**	q_3	□	**L**	q_r	□	**R**
q_3	q_3	**0**	**L**	q_3	**1**	**L**	q_0	□	**R**

The Universal Turing Machine

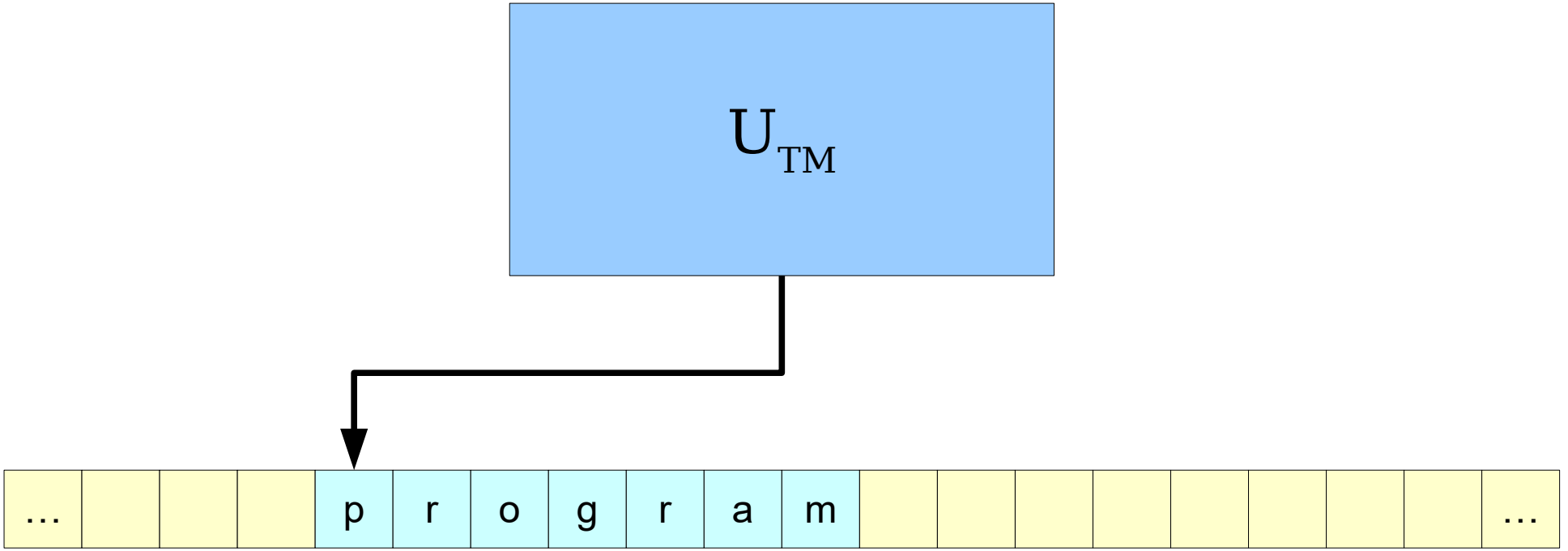
- ***Theorem (Turing, 1936)***: There is a Turing machine $\mathbf{U_{TM}}$ called the ***universal Turing machine*** that, when run on an input of the form $\langle M, w \rangle$, where M is a Turing machine and w is a string, simulates M running on w and does whatever M does on w (accepts, rejects, or loops).
- The observable behavior of U_{TM} is the following:
  - If M accepts w, then U_{TM} accepts $\langle M, w \rangle$.
  - If M rejects w, then U_{TM} rejects $\langle M, w \rangle$.
  - If M loops on w, then U_{TM} loops on $\langle M, w \rangle$.
- **$\mathbf{U_{TM}}$ accepts $\langle M, w \rangle$ if and only if M accepts w.**



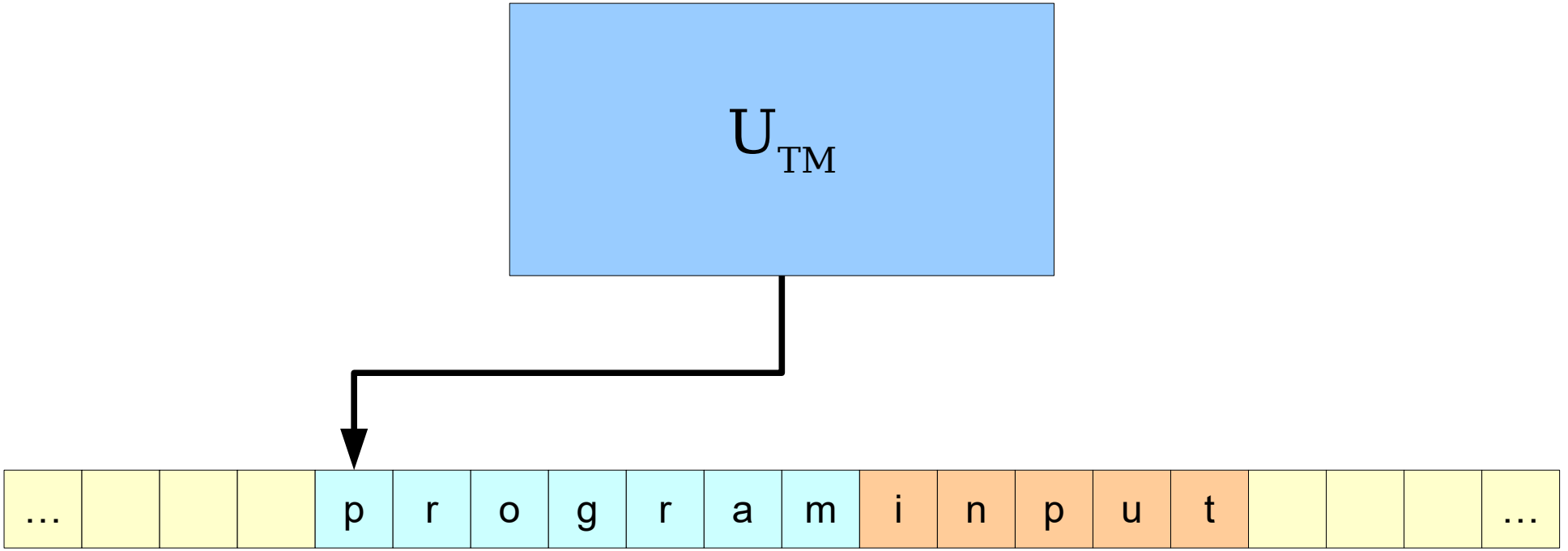
A Universal Machine



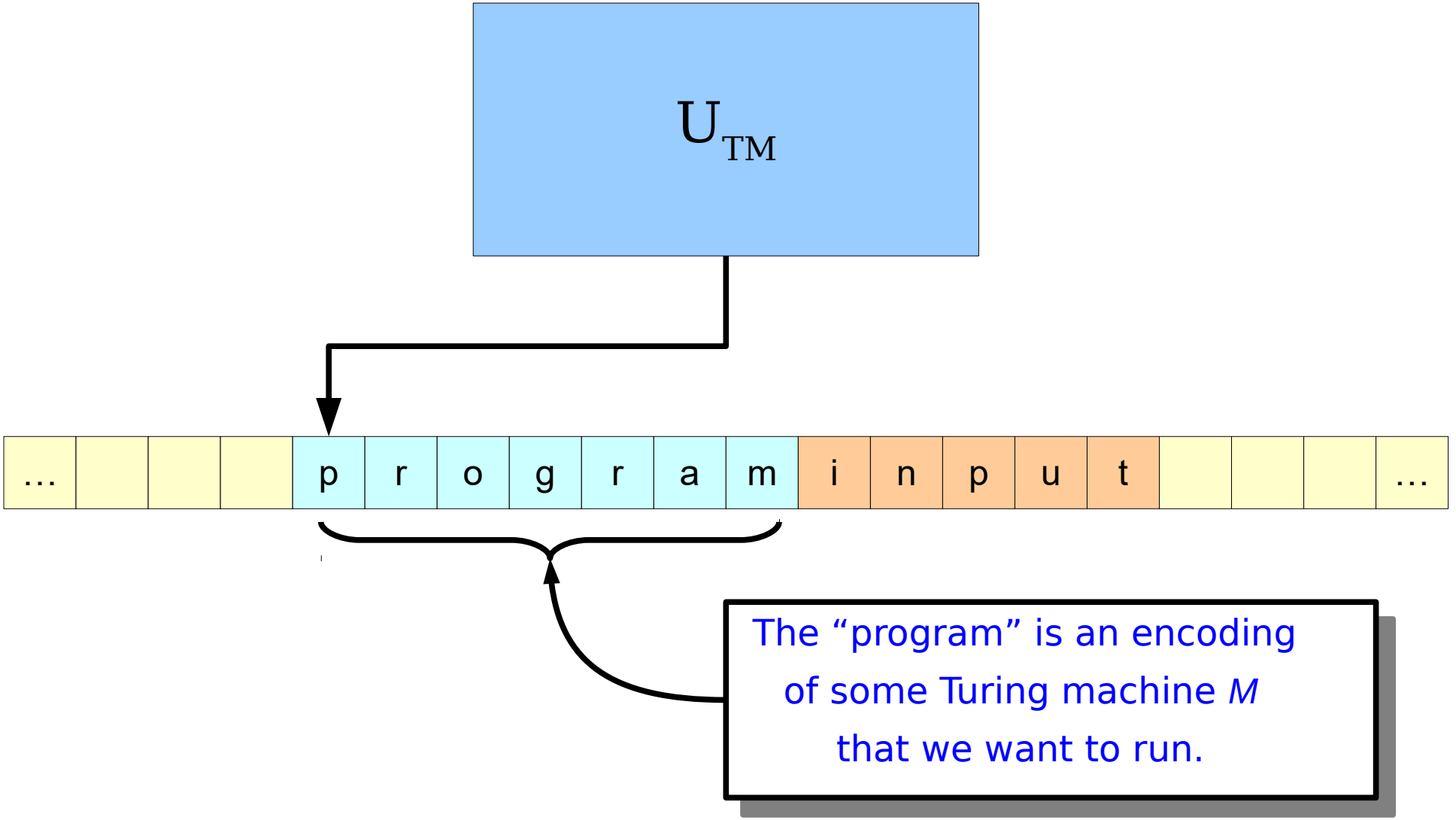
A Universal Machine



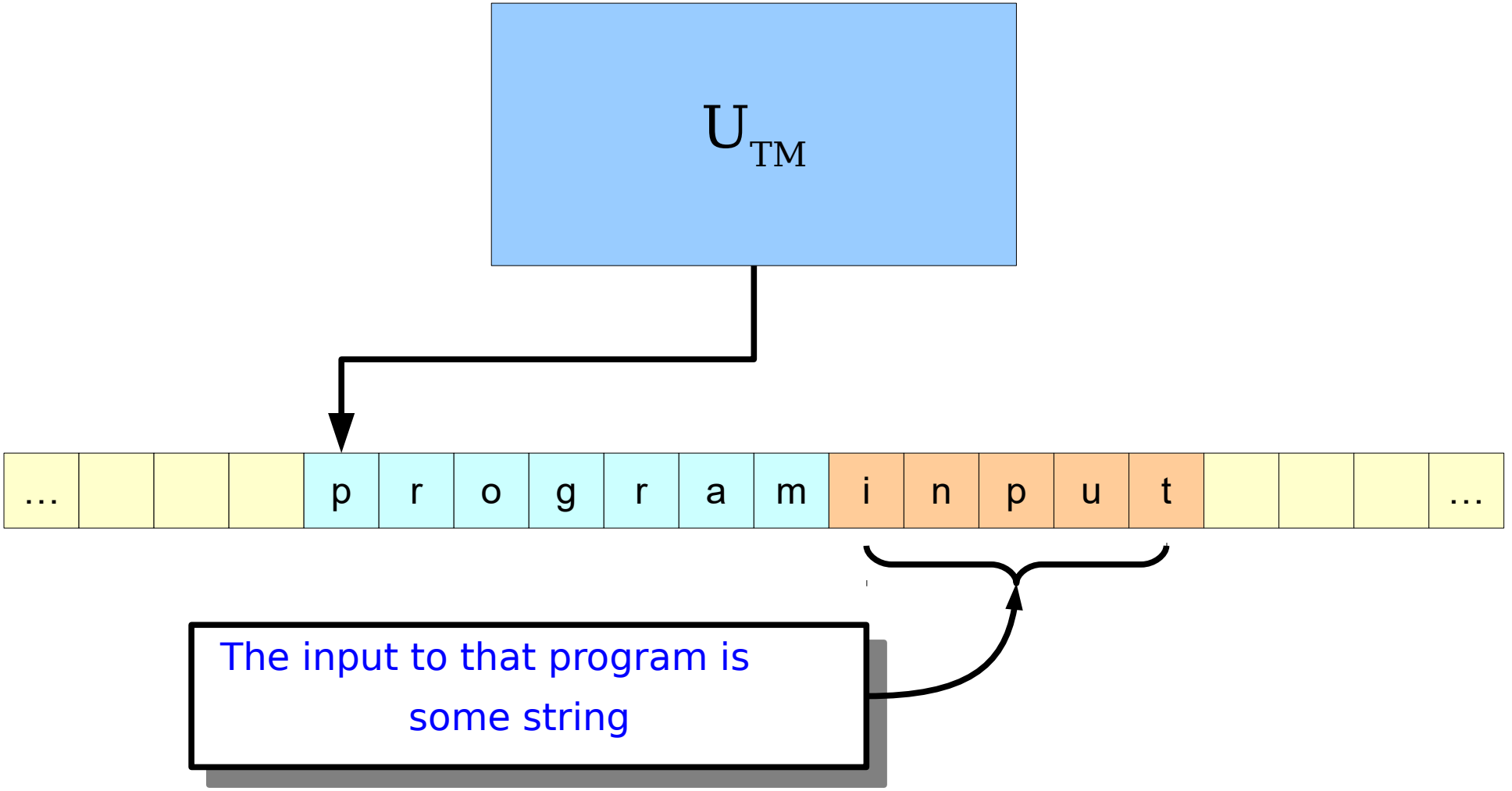
A Universal Machine



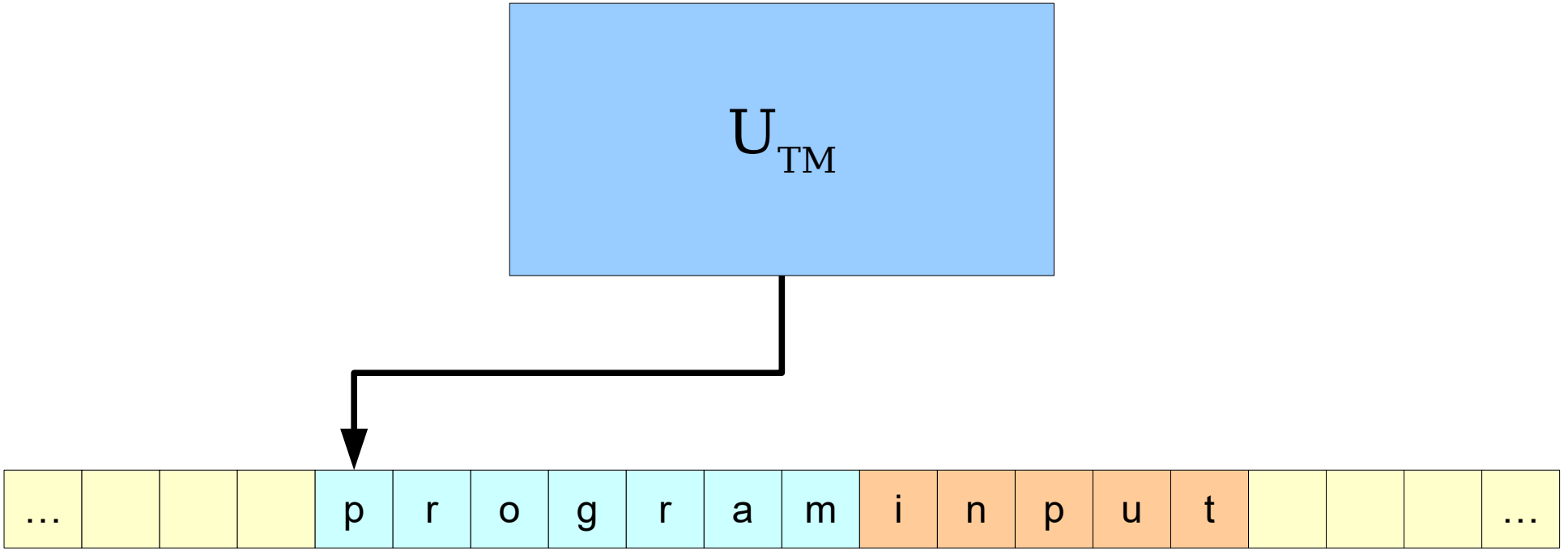
A Universal Machine



A Universal Machine

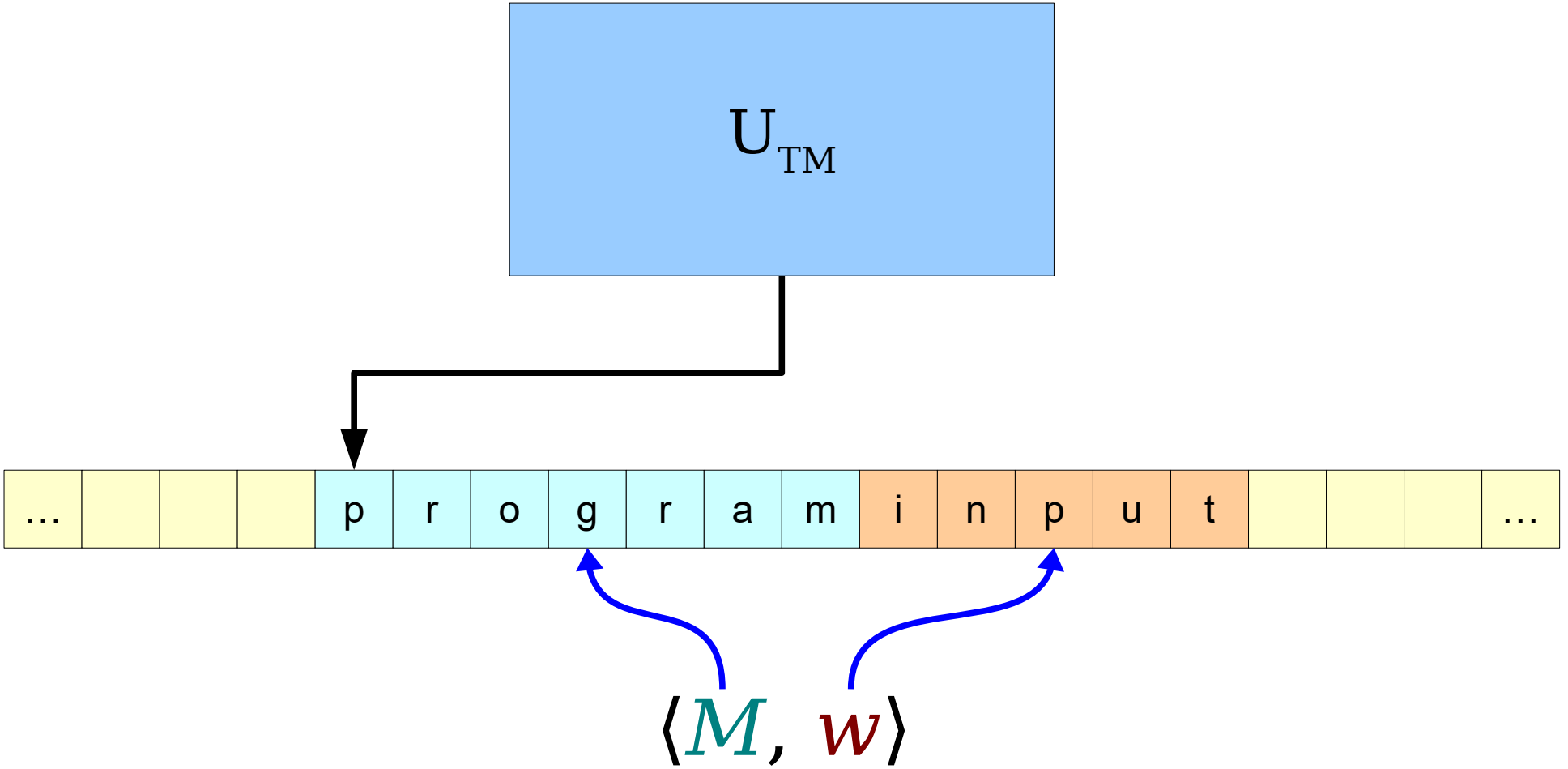


A Universal Machine



The input has the form $\langle M, w \rangle$, where M is some TM and w is some string.

A Universal Machine



Since U_{TM} is a TM, it has a language.

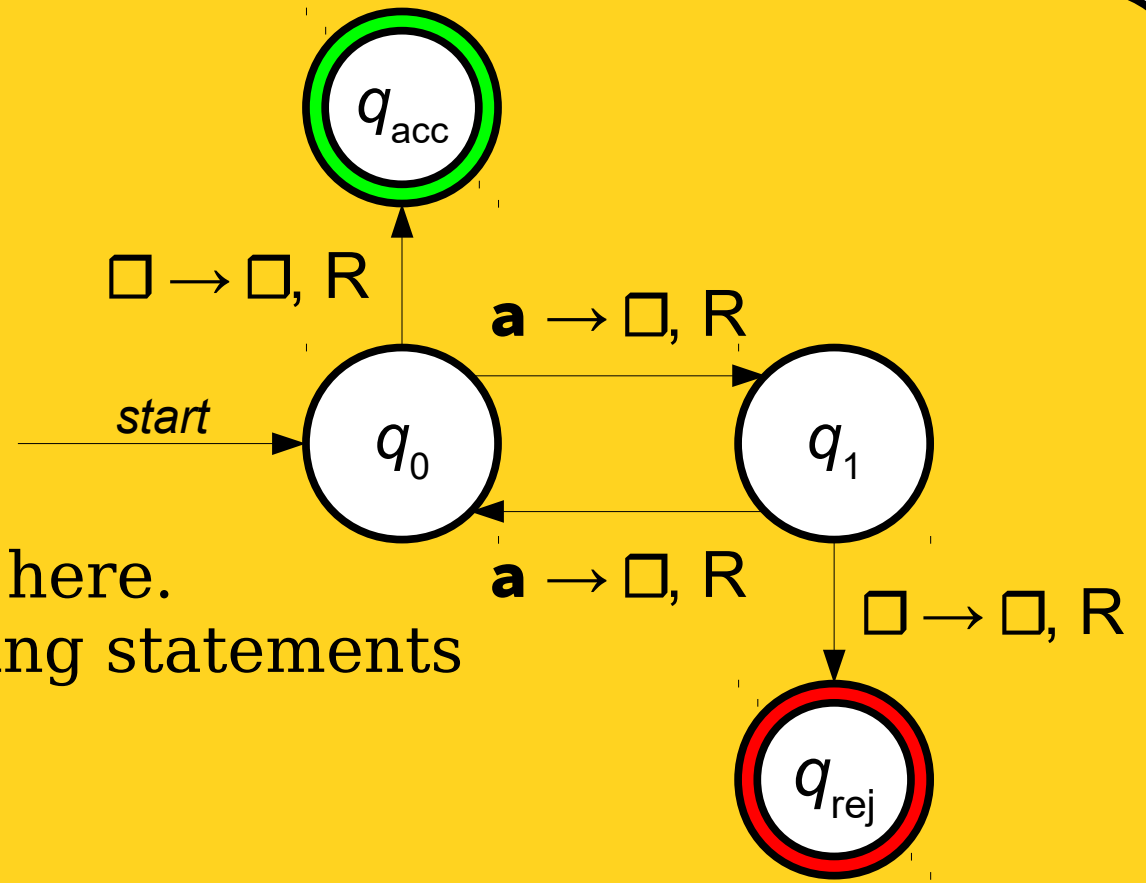
What is the language of the universal Turing machine?

The Language of U_{TM}

- U_{TM} accepts $\langle M, w \rangle$ iff M is a TM that accepts w.
- Therefore:

  $\mathscr{L}(U_{TM}) = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$

- For simplicity, define $A_{TM} = \mathscr{L}(U_{TM})$. This is an important language and we'll see it many times.



Let M be the TM shown here. How many of the following statements are true?

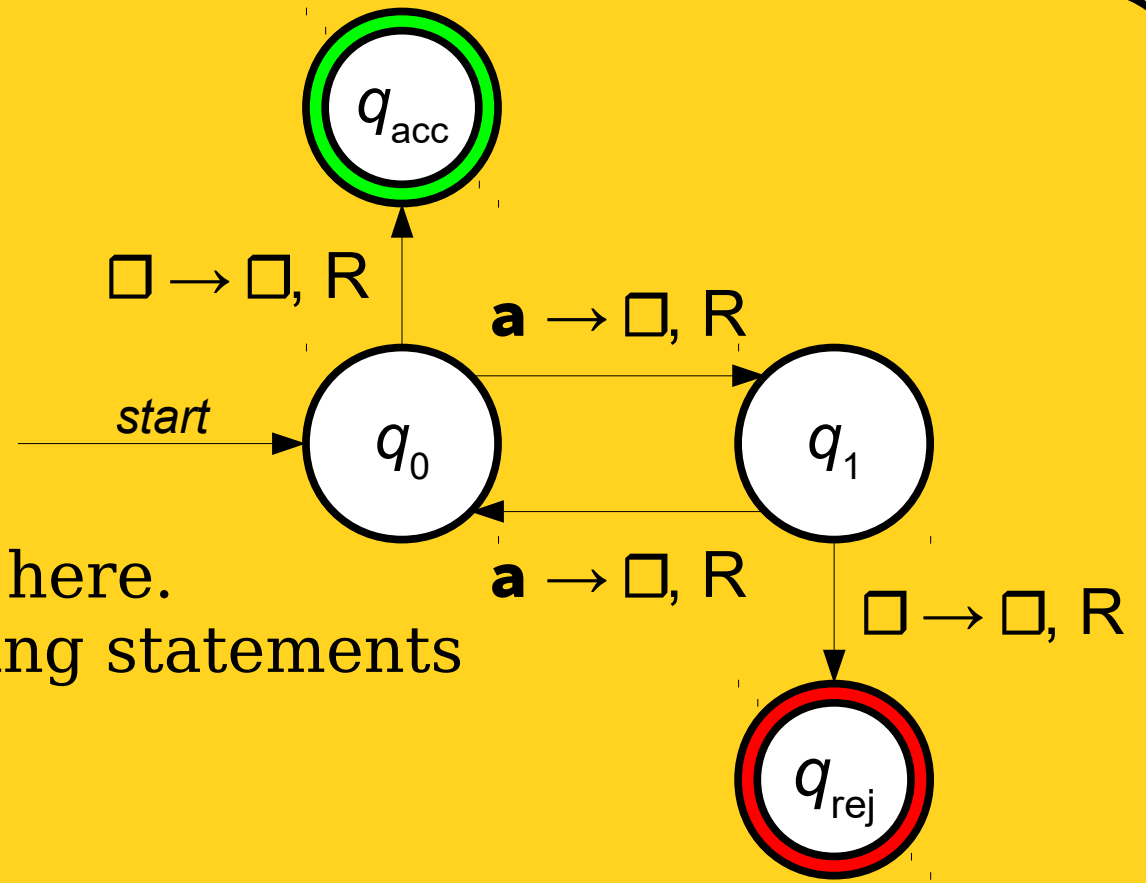
M accepts **aa**

U_{TM} accepts $\langle M,$ **aa**$\rangle$

U_{TM} accepts $\langle M,$ **ε**$\rangle$

U_{TM} accepts $\langle M,$ **a**$\rangle$

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then **a number**.



Let M be the TM shown here.
How many of the following statements are true?

M accepts **aa**

U_{TM} accepts $\langle M,$ **aa**$\rangle$

U_{TM} accepts $\langle M,$ **ε**$\rangle$ is equivalent to $\langle M, \varepsilon\rangle \in A_{TM}$

U_{TM} accepts $\langle M,$ **a**$\rangle$

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then **a number**.

Claim 3

3. That TM (from 2), or any TM that takes other TMs as input, could take itself as input. (“Self-Reference” property)

Equivalence of TMs and Programs

- Here's a sample program we might use to model a Turing machine for { $w \in$ {**a**, **b**}* | w has the same number of **a**'s and **b**'s }:

```
int main() {
    string input = getInput();
    int difference = 0;

    for (char ch: input) {
        if (ch == 'a') difference++;
        else if (ch == 'b') difference--;
        else reject();
    }

    if (difference == 0) accept();
    else reject();
}
```

Equivalence of TMs and Programs

- Now, a new fact: it’s possible to build a method `mySource()` into a program, which returns the source code of the program.
- For example, here's a narcissistic program:

```
int main() {
    string me = mySource();
    string input = getInput();

    if (input == me) accept();
    else reject();
}
```

Equivalence of TMs and Programs

- Sometimes, TMs use other TMs as subroutines.
- We can think of a decider for a language as a method that takes in some number of arguments and returns a boolean.
- For example, a decider for $\{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$ might be represented in software as a method with this signature:

```
bool isAnBn(string w);
```

- Similarly, a decider for $\{ \langle m, n \rangle \mid m, n \in \mathbb{N}$ and m is a multiple of $n \}$ might be represented in software as a method with this signature:

```
bool isMultipleOf(int m, int n);
```

Side note: what does equivalence of TMs and programs mean for YOU?

- In this class (starting Problem Set Nine, and unless directed otherwise) you can write proofs about TMs by just writing normal code (e.g., Java or C++), and never have to painstakingly draw an actual TM ever again!

:: rejoicing ::