

Turing Machines

Part One

What problems can we solve with a computer?

A Venn diagram illustrating the relationship between different classes of languages. It features a large yellow oval representing the set of all languages, and a smaller yellow circle inside it representing the set of regular languages. The text 'Regular Languages' is centered within the circle, and 'Languages recognizable by any feasible computing machine' is centered within the oval. The label 'All Languages' is positioned at the bottom right of the image.

**Regular
Languages**

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

All Languages

That same drawing, to scale.

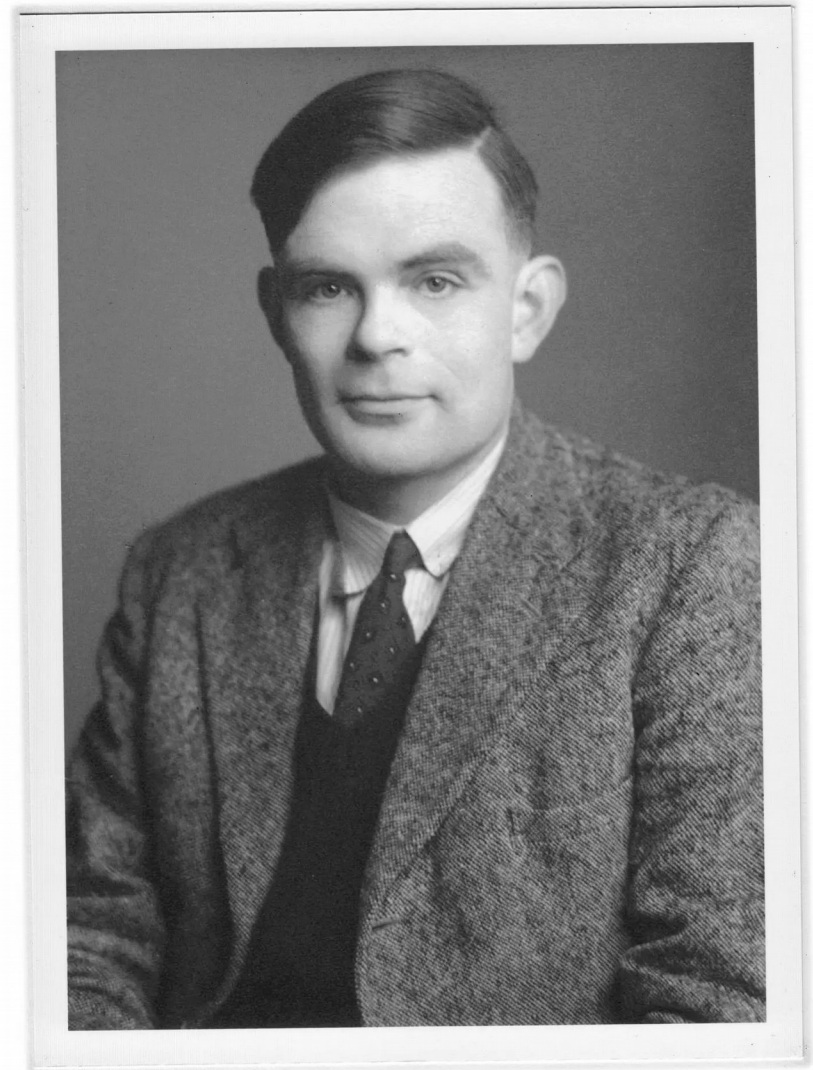
The Problem

- Finite automata accept precisely the regular languages.
- We've already seen cases where we may need unbounded memory to recognize languages.
 - e.g. $\{ a^n b^n \mid n \in \mathbb{N} \}$ requires unbounded counting.
- How do we model a computing device that has unbounded memory?

A Brief History Lesson

Turing Machines

- In March 1936, Alan Turing (aged 23!) published a paper detailing the ***a-machine*** (for ***automatic machine***), an automaton for computing on real numbers.
- They're now more popularly referred to as ***Turing machines*** in his honor.
- He also later made contributions to computational biology, artificial intelligence, cryptography, etc. Seriously, Google this guy.



$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline \end{array}$$

$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline 7 \end{array}$$

$$\begin{array}{r} 27182818284^1 590 \\ + 31415926535^1 897 \\ \hline 27182818284 590 \\ 31415926535 897 \\ \hline 5859874477187 \end{array}$$

$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline 27182818284590 \\ + 31415926535897 \\ \hline 58598744825487 \end{array}$$

$$\begin{array}{r} 27182818 \mathbf{2} 84590 \\ + 31415926 \mathbf{5} 35897 \\ \hline 27182818 \mathbf{8} 20487 \end{array}$$

$$\begin{array}{r} 2718281\mathbf{8}284590 \\ + 3141592\mathbf{6}535897 \\ \hline 2718281\mathbf{4}820487 \end{array}$$

$$\begin{array}{r} 271828\mathbf{1}8284590 \\ + 314159\mathbf{2}6535897 \\ \hline 271828\mathbf{4}4820487 \end{array}$$

$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline 27182818284590 \\ 31415926535897 \\ \hline 744820487 \end{array}$$

$$\begin{array}{r} 2718 \mathbf{2}818284590 \\ + 3141 \mathbf{5}926535897 \\ \hline \mathbf{8}744820487 \end{array}$$

$$\begin{array}{r} 2718 \\ + 3141 \\ \hline 2718 \\ 3141 \\ \hline 5859 \end{array}$$

Carry digits: 1 1 1 1 1 1

$$\begin{array}{r} 27\mathbf{1}82818284590 \\ + 31\mathbf{4}15926535897 \\ \hline \mathbf{5}98744820487 \end{array}$$

Carry digits: 1 1 1 1 1 1

$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline 8598744820487 \end{array}$$

Carry values: 1 1 1 1 1 1

$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline 58598744820487 \end{array}$$

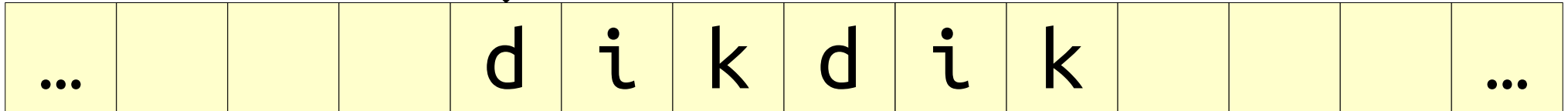
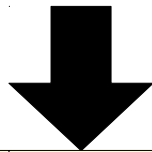
Carry digits: 1 1 1 1 1 1

$$\begin{array}{r} 27182818284590 \\ + 31415926535897 \\ \hline 58598744820487 \end{array}$$

Key Idea: Even if you need huge amounts of scratch space to perform a calculation, at each point in the calculation you only need access to a small amount of that scratch space.

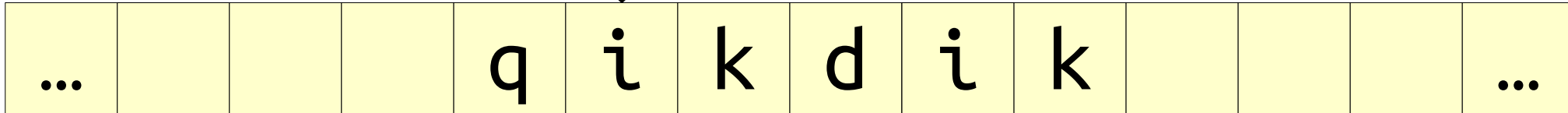
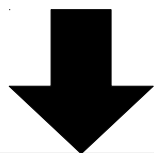
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



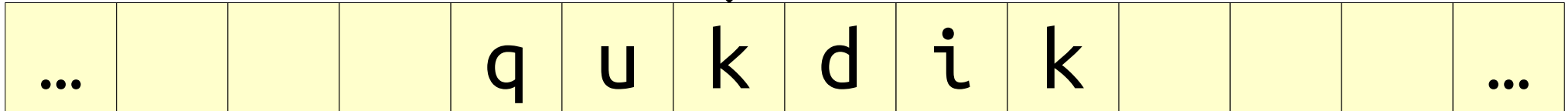
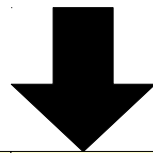
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



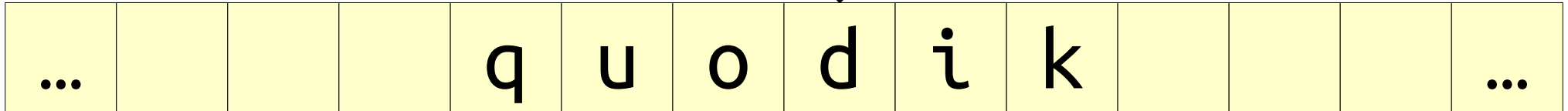
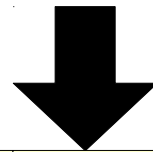
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



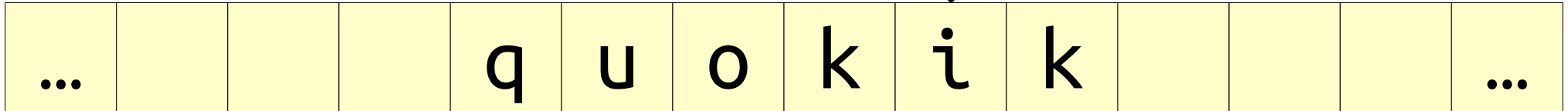
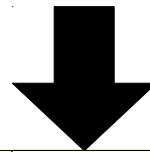
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



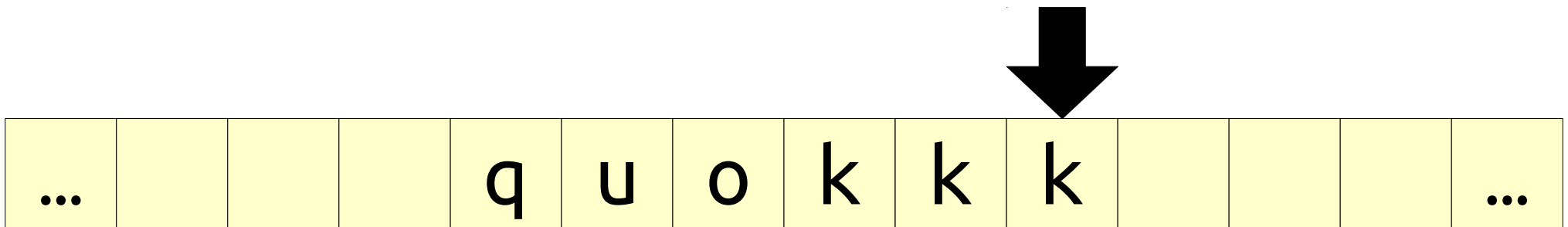
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



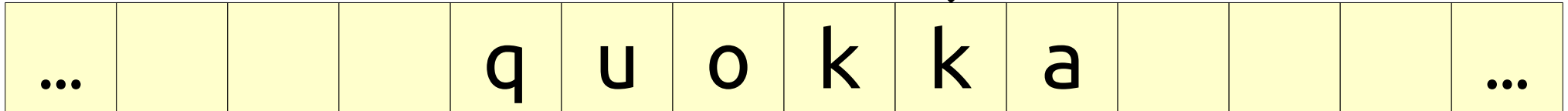
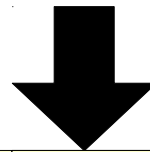
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



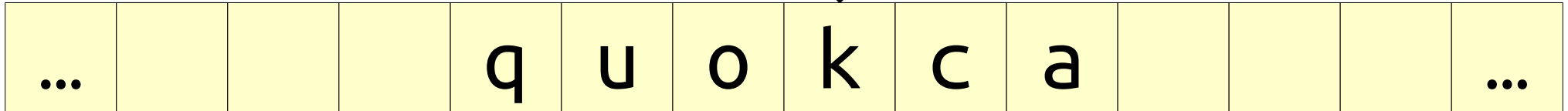
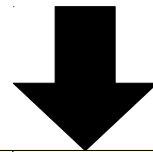
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



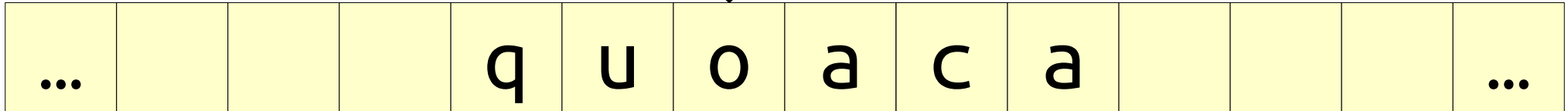
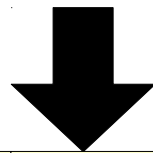
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



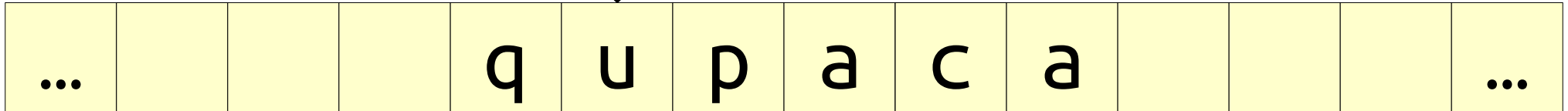
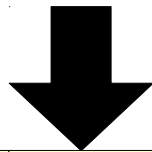
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



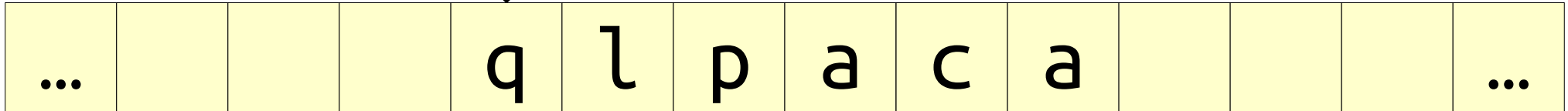
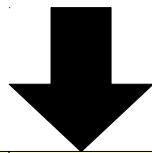
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



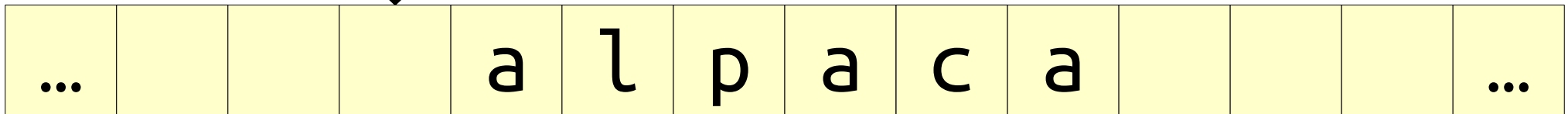
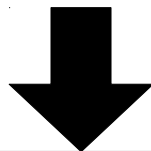
Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



Turing Machines

- To provide his machines extra memory, Turing gave his machines access to an *infinite tape* subdivided into a number of *tape cells*.
- A Turing machine can only see one tape cell at a time, the one pointed at by the *tape head*.
- The Turing machine can
 - read the cell under the tape head,
 - (possibly) change which symbol was written under the tape head, and
 - move its tape head to the left or to the right.



Turing Machines

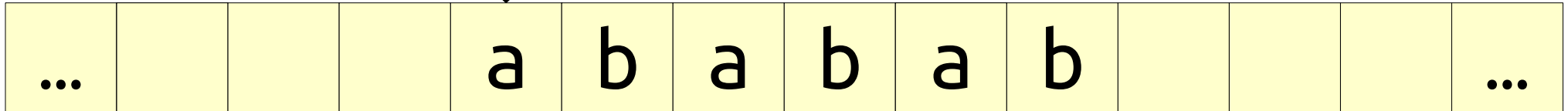
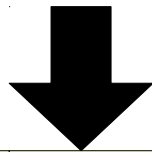
- Over the years, there have been many simplifications and edits to Turing's original automata.
 - In practice, electronic computers are written in terms of individual instructions rather than states and transitions.
 - Turing's original paper deals with computing individual real numbers; we typically want to compute functions of inputs.
- What we're going to present as "Turing machines" in this class differ significantly from Turing's original description, while retaining the core essential ideas.
 - (Our model is closer to Emil Post's *Formulation 1* and Hao Wang's *Basic Machine B*, for those of you who are curious.)
- If you'd like to learn more about Turing's original version of the Turing machine, come chat with me after class!

Turing Machines

- A TM is a series of instructions that control a tape head as it moves across an infinite tape.
- The tape begins with the input string written somewhere, surrounded by infinitely many blank cells.
 - Rule: The input string cannot contain blank cells.
- The tape head begins above the first character of the input. (If the input is ϵ , the tape head points somewhere on a blank tape.)

Start:

```
If Blank Return True
If 'b' Return False
Write 'x'
Move Right
If Not 'b' Return False
Write 'x'
Move Right
Goto Start
```

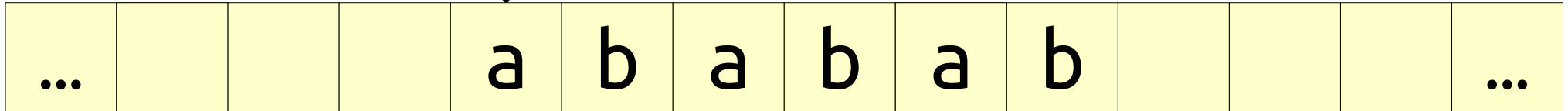
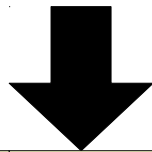


Turing Machines

- We begin at the Start label.
- Labels indicate different sections of code. The name Start is special and means “begin here.”
- Labels have no effect when executed. We just move to the next line.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

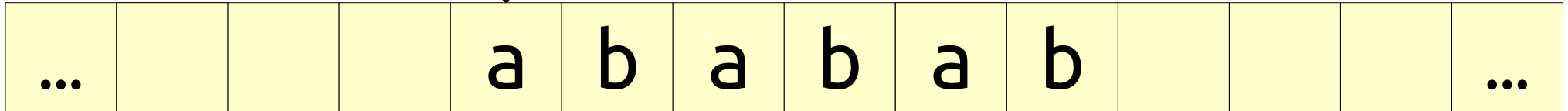
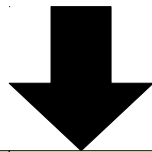


Turing Machines

- We begin at the Start label.
- Labels indicate different sections of code. The name Start is special and means “begin here.”
- Labels have no effect when executed. We just move to the next line.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```



Turing Machines

- We begin at the Start label.
- Labels indicate different sections of code. The name Start is special and means “begin here.”
- Labels have no effect when executed. We just move to the next line.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

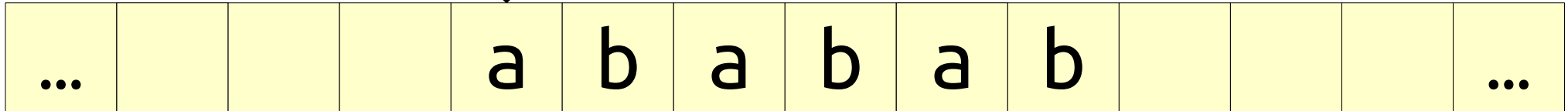
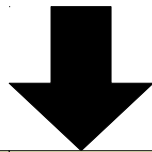
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

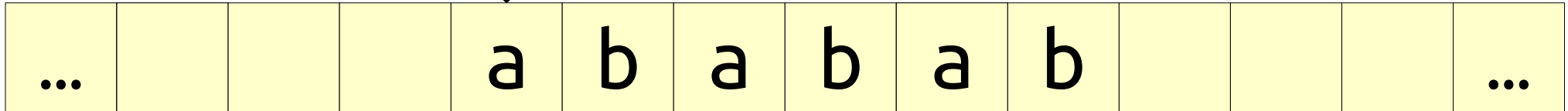
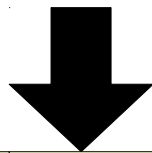
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

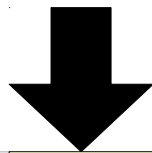
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

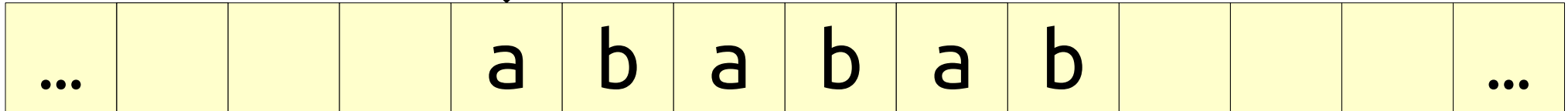
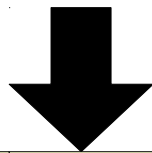
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

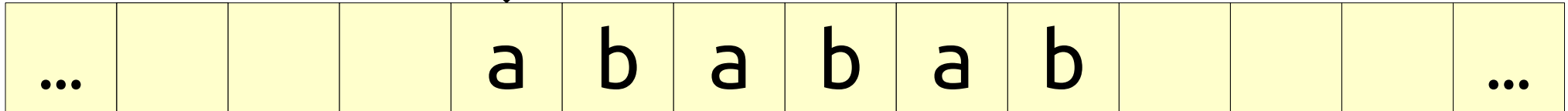
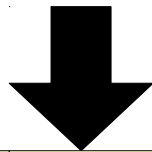
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

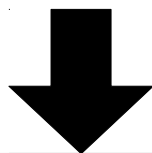
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

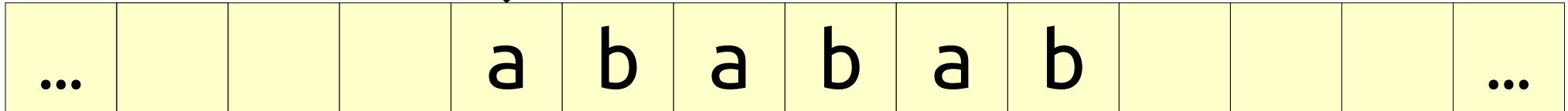
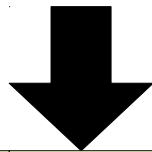
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form
If *symbol* *command*
checks if the character under the tape head is *symbol*.
- If so, it executes *command*.
- If not, nothing happens.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

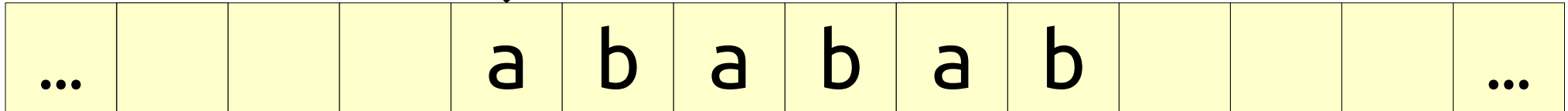
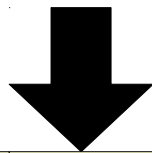
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The statement
Write *symbol*
writes *symbol* to the
cell under the tape
head.
- The *symbol* can
either be Blank or a
character in quotes.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

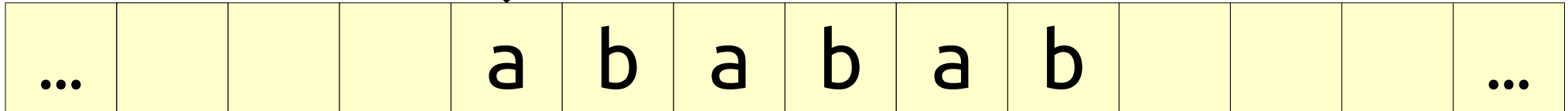
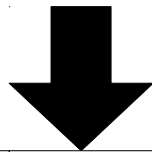
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The statement
Write *symbol*
writes *symbol* to the
cell under the tape
head.
- The *symbol* can
either be Blank or a
character in quotes.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

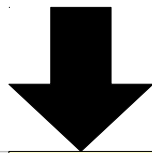
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The statement
Write *symbol*
writes *symbol* to the
cell under the tape
head.
- The *symbol* can
either be Blank or a
character in quotes.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

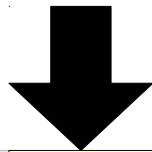
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The statement
Write *symbol*
writes *symbol* to the
cell under the tape
head.
- The *symbol* can
either be Blank or a
character in quotes.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

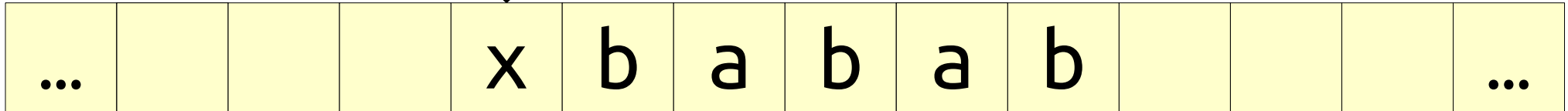
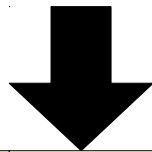
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The statement
Write *symbol*
writes *symbol* to the
cell under the tape
head.
- The *symbol* can
either be Blank or a
character in quotes.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

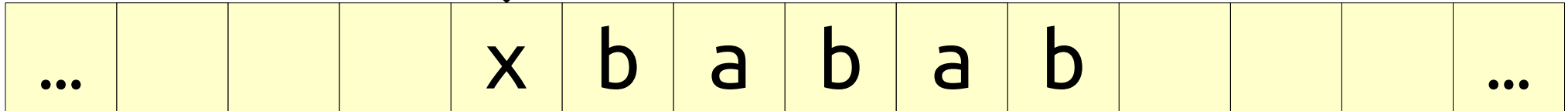
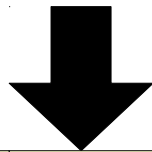
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The command
Move *direction*
moves the tape head one step in the indicated direction (either Left or Right).

Start:

If Blank Return True

If 'b' Return False

Write 'x'

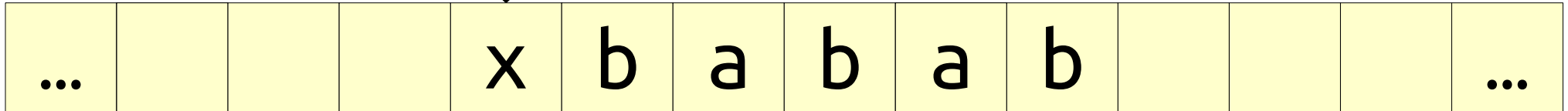
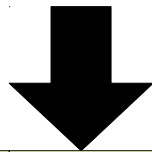
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The command **Move *direction*** moves the tape head one step in the indicated direction (either Left or Right).

Start:

If Blank Return True

If 'b' Return False

Write 'x'

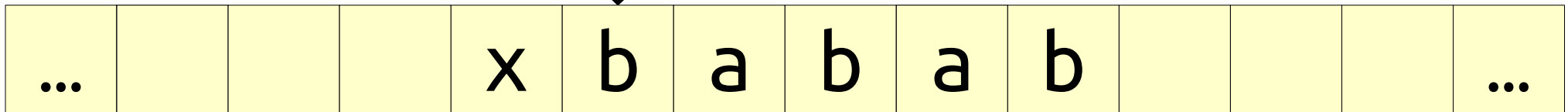
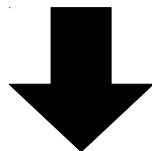
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- The command
Move *direction*
moves the tape head one step in the indicated direction (either Left or Right).

Start:

If Blank Return True

If 'b' Return False

Write 'x'

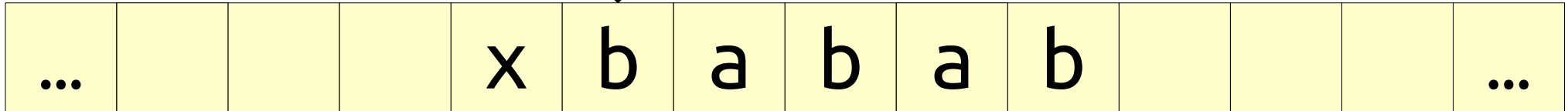
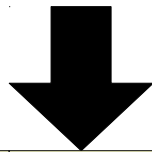
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form **If Not** *symbol command* sees if the cell under the tape head holds *symbol*.
- If so, nothing happens.
- If not, it executes *command*.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

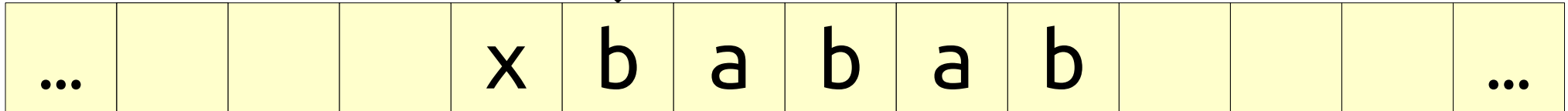
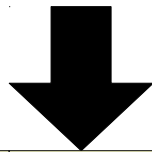
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start

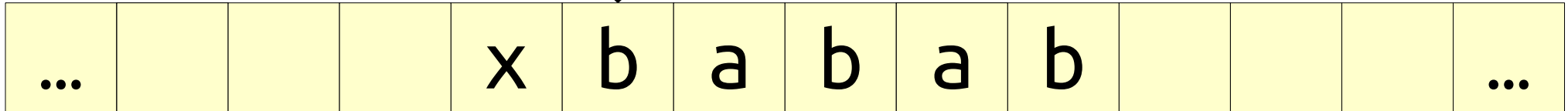
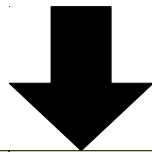


Turing Machines

- A statement of the form **If Not** *symbol command* sees if the cell under the tape head holds *symbol*.
- If so, nothing happens.
- If not, it executes *command*.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```



Turing Machines

- A statement of the form **If Not** *symbol command* sees if the cell under the tape head holds *symbol*.
- If so, nothing happens.
- If not, it executes *command*.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

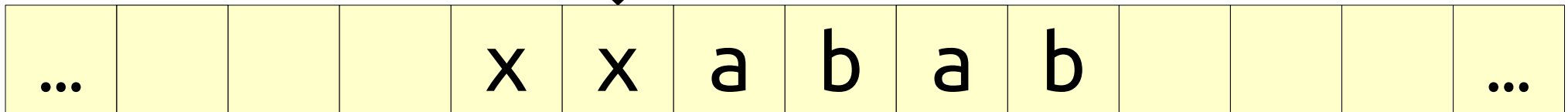
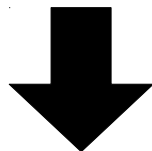
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A statement of the form **If Not *symbol command*** sees if the cell under the tape head holds *symbol*.
- If so, nothing happens.
- If not, it executes *command*.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

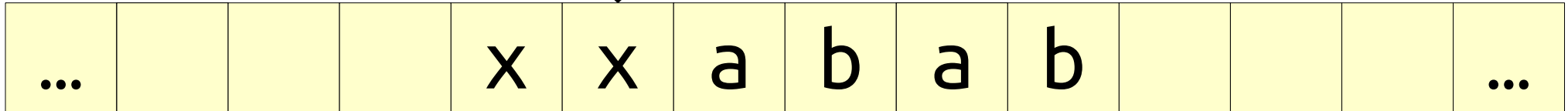
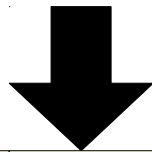
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start

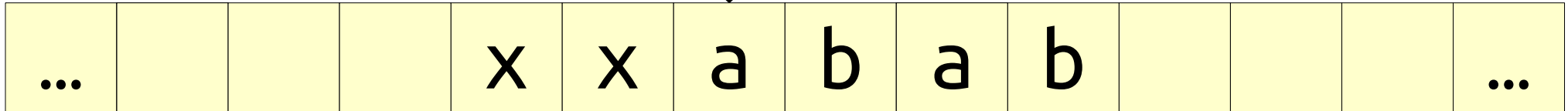
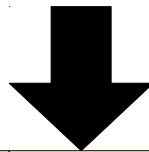


Turing Machines

- A statement of the form **If Not** *symbol command* sees if the cell under the tape head holds *symbol*.
- If so, nothing happens.
- If not, it executes *command*.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

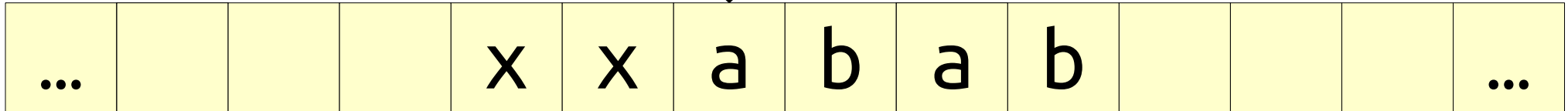
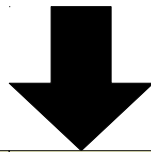


Turing Machines

- A statement of the form **If Not** *symbol* *command* sees if the cell under the tape head holds *symbol*.
- If so, nothing happens.
- If not, it executes *command*.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

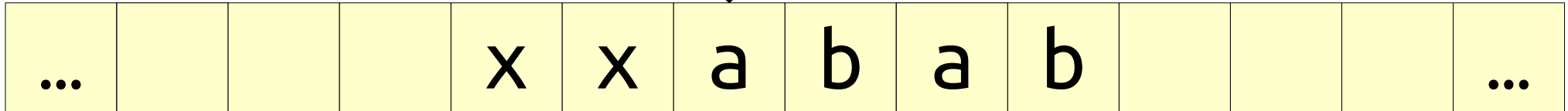
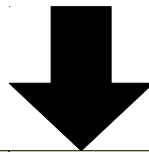


Turing Machines

- The command
Goto *label*
jumps to the indicated label.
- This program just has a Start label, but most interesting programs have other labels beyond this.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

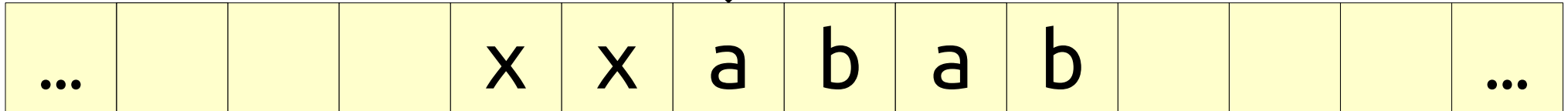
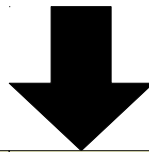


Turing Machines

- The command
Goto *label*
jumps to the indicated label.
- This program just has a Start label, but most interesting programs have other labels beyond this.

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

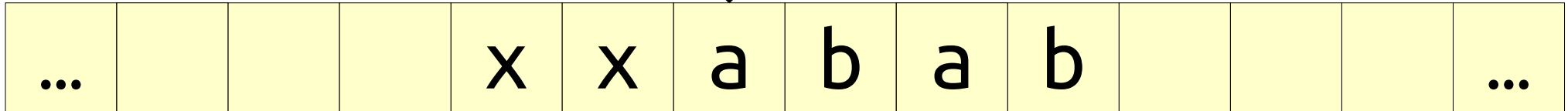
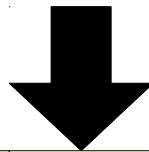


Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

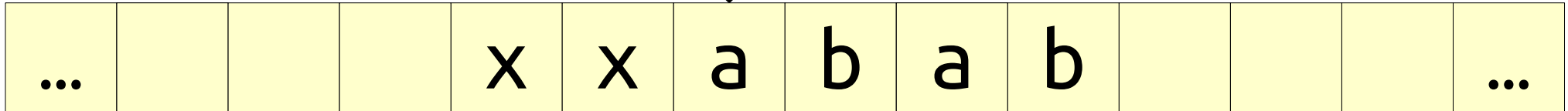
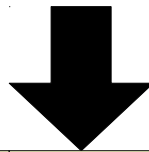
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

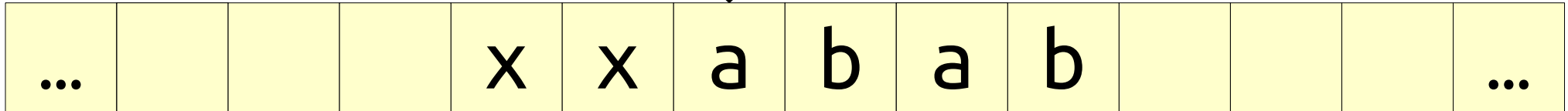
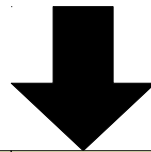
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

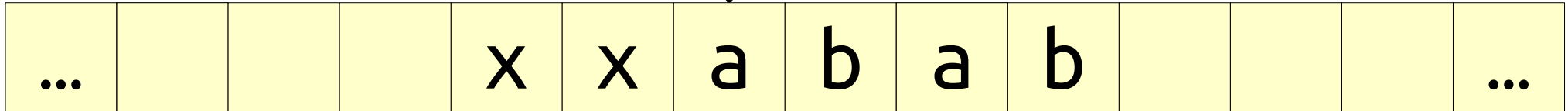
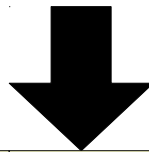
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

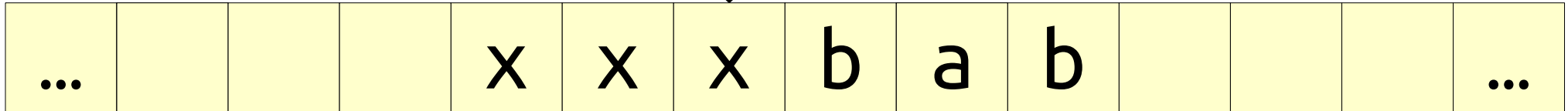
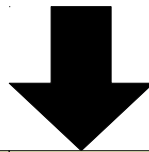
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

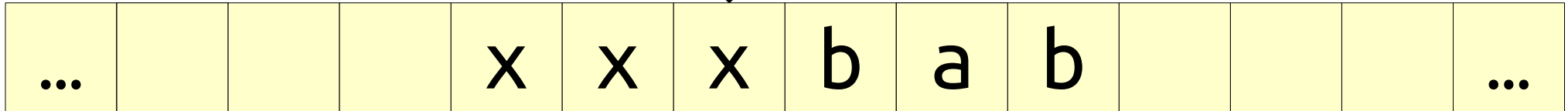
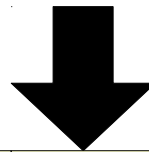
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

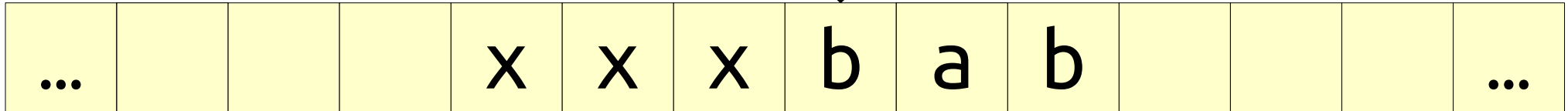
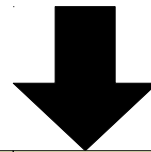
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

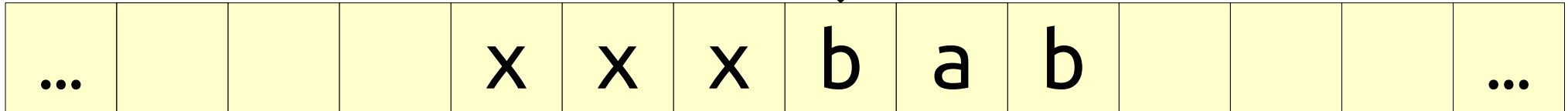
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

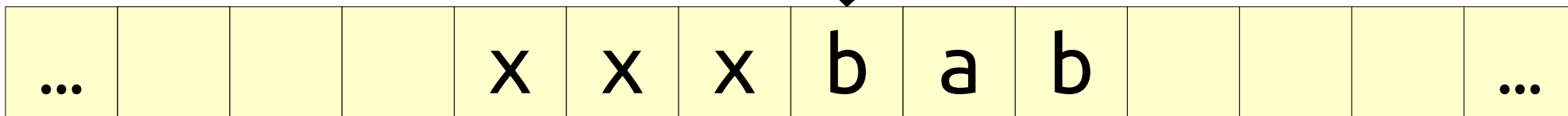
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

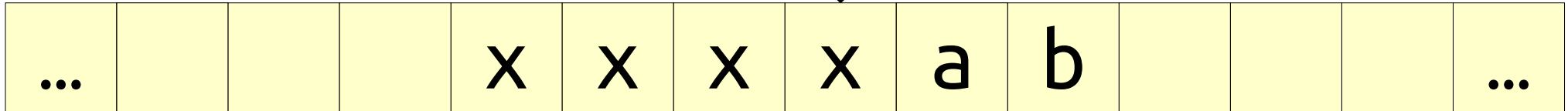
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

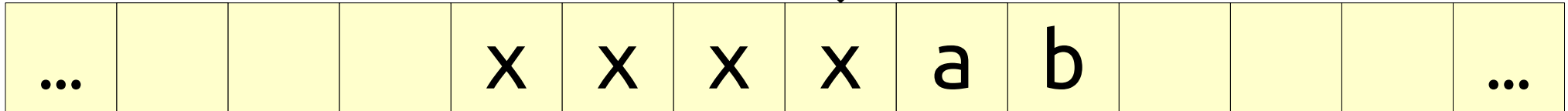
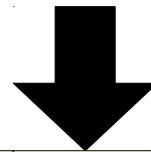
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

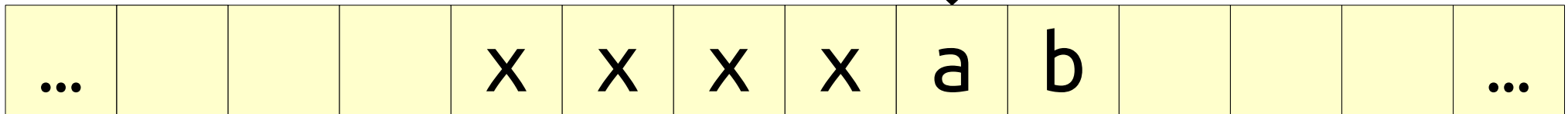
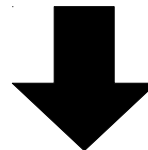
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start

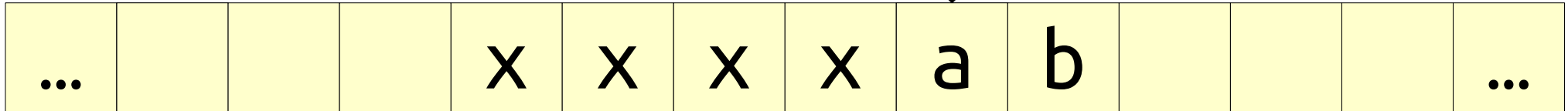
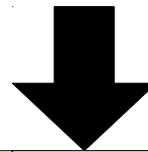


Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

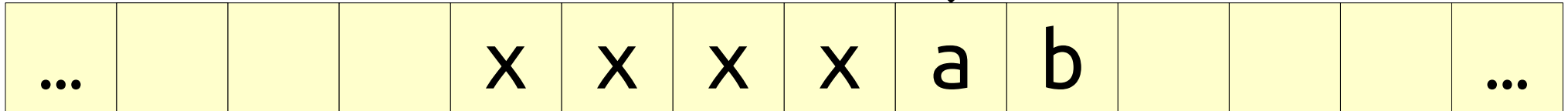
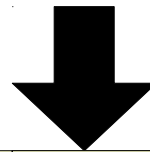


Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

```
If Blank Return True
If 'b' Return False
Write 'x'
Move Right
If Not 'b' Return False
Write 'x'
Move Right
Goto Start
```



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

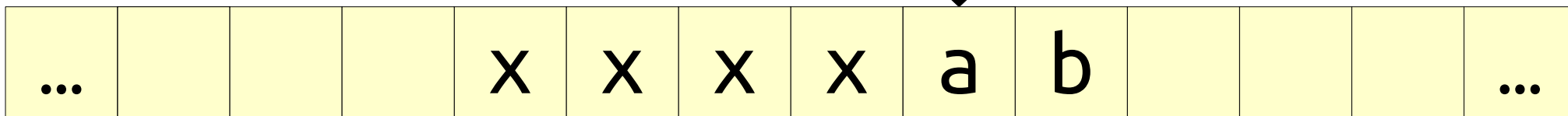
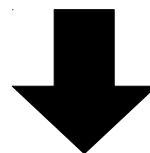
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

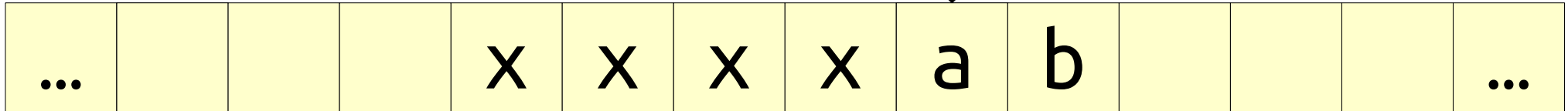
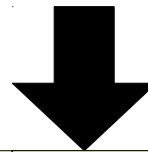
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

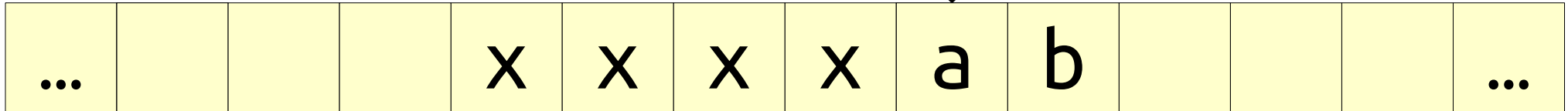
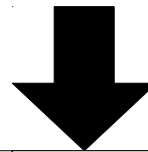
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

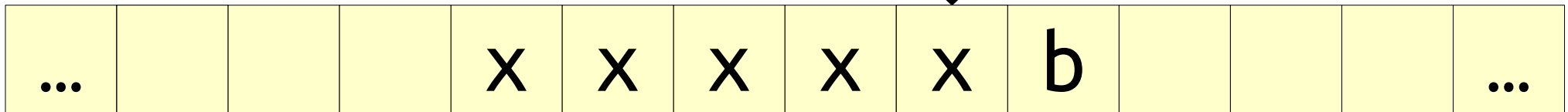
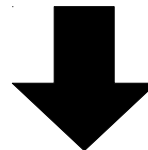
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the **Return *result*** command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

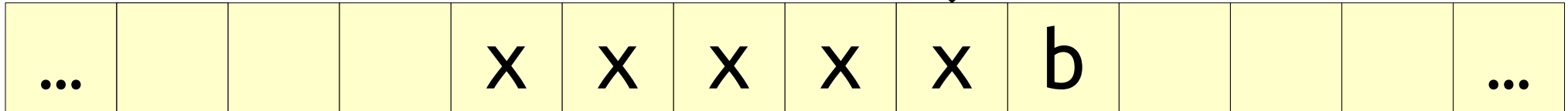
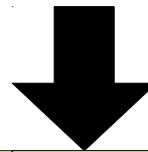
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

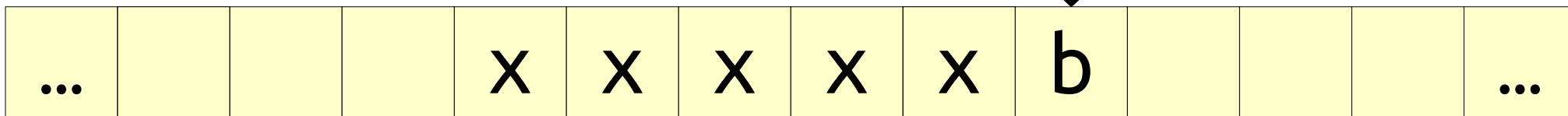
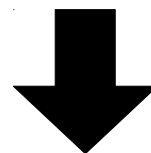
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

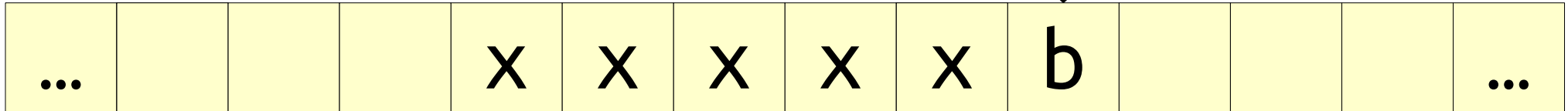
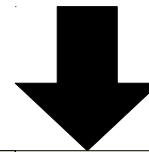
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

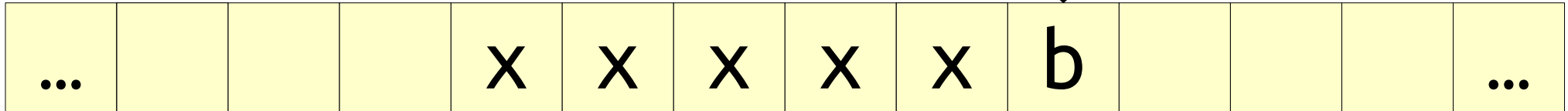
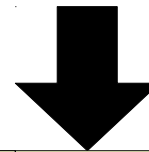
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

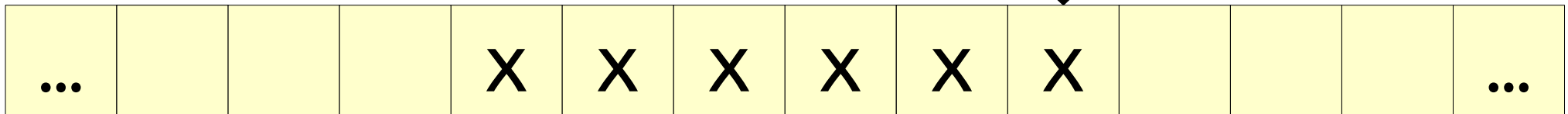
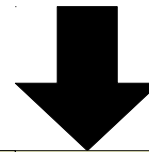
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

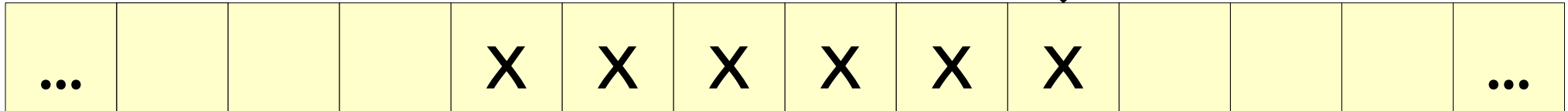
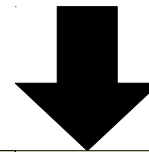
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

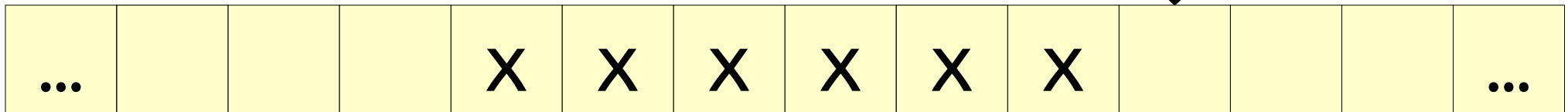
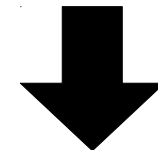
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start

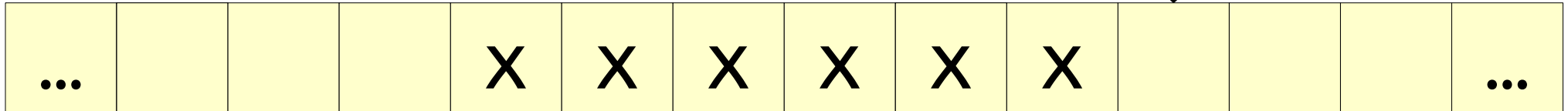


Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

```
If Blank Return True  
If 'b' Return False  
Write 'x'  
Move Right  
If Not 'b' Return False  
Write 'x'  
Move Right  
Goto Start
```

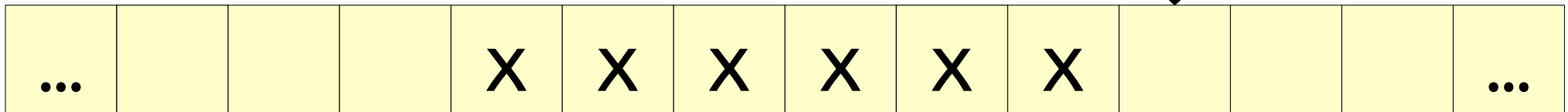
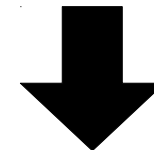


Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

```
If Blank Return True
If 'b' Return False
Write 'x'
Move Right
If Not 'b' Return False
Write 'x'
Move Right
Goto Start
```



Turing Machines

- A TM stops when executing the
Return *result*
command.
- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

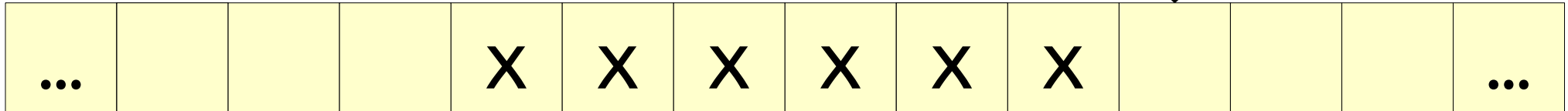
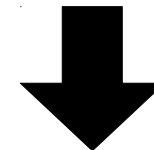
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- A TM stops when executing the

Return *result*

command.

- Here, *result* can be either True or False.
- (If we “fall off” the bottom of the program, the TM acts as though it executes the Return False command.)

Start:

If Blank Return True

If 'b' Return False

Write 'x'

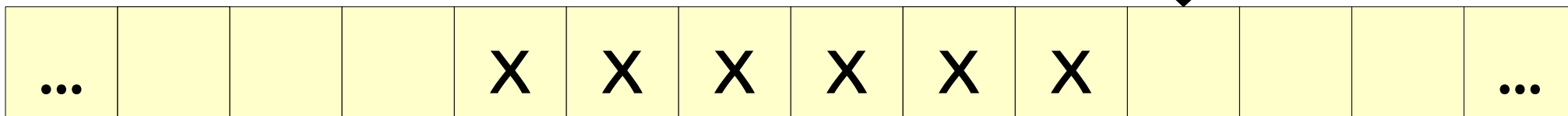
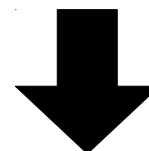
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start



Turing Machines

- This TM initially started up with the string ababab on its tape, so this means that TM returns true on the input ababab, not xxxxxx.
- An intuition for this: we gave this program an input. It therefore returned true with respect to that input, not whatever internal data it generated in making its decision.

Start:

If Blank Return True

If 'b' Return False

Write 'x'

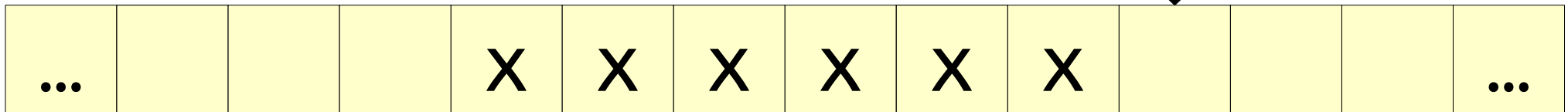
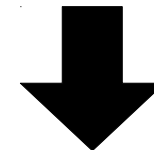
Move Right

If Not 'b' Return False

Write 'x'

Move Right

Goto Start

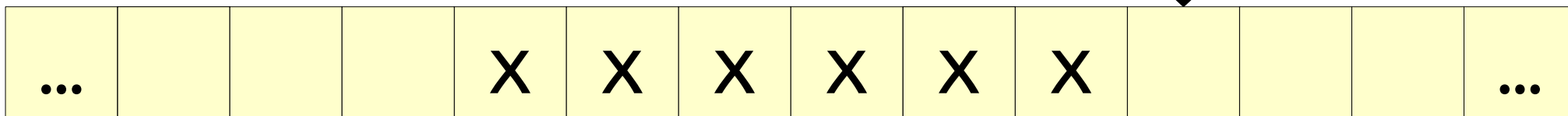
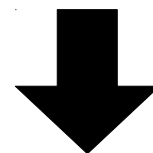


Turing Machines

- To summarize, we only have six commands:
 - Move *direction*
 - Write *symbol*
 - Goto *label*
 - Return *result*
 - If *symbol command*
 - If Not *symbol command*
- Despite their simplicity, TMs are *surprisingly* powerful. The rest of this lecture explores why.

Start:

```
If Blank Return True
If 'b' Return False
Write 'x'
Move Right
If Not 'b' Return False
Write 'x'
Move Right
Goto Start
```



Programming Turing Machines

Our First Challenge

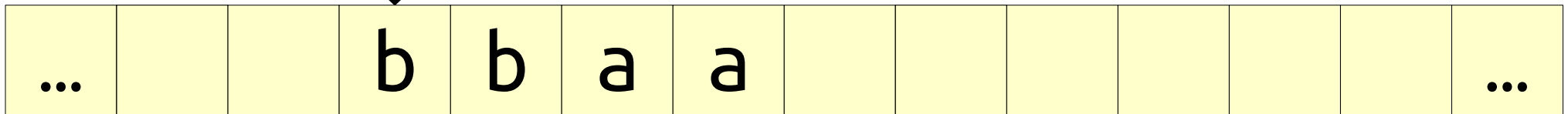
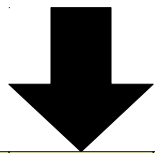
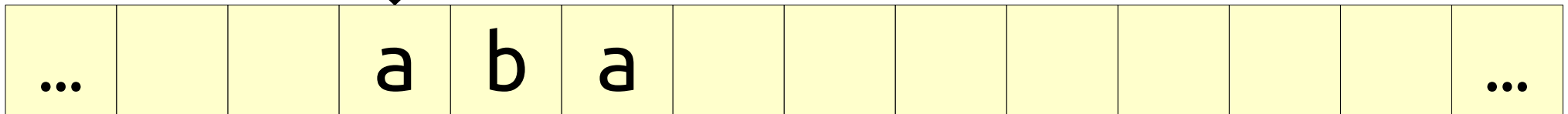
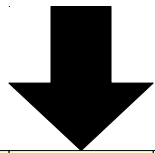
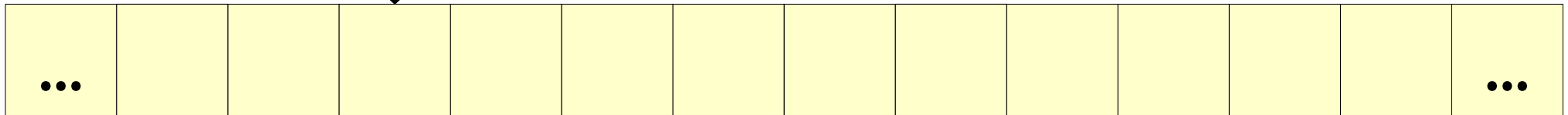
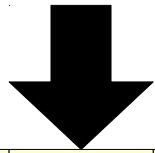
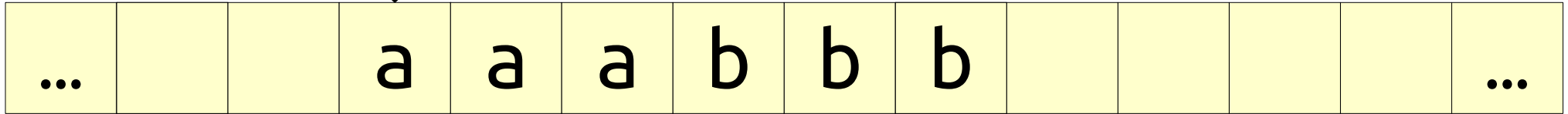
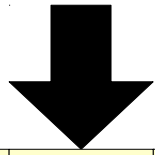
- The language

$$\{ a^n b^n \mid n \in \mathbb{N} \}$$

is a canonical example of a nonregular language. It's not possible to check if a string is in this language given only finite memory.

- Turing machines, however, are powerful enough to do this. Let's see how.

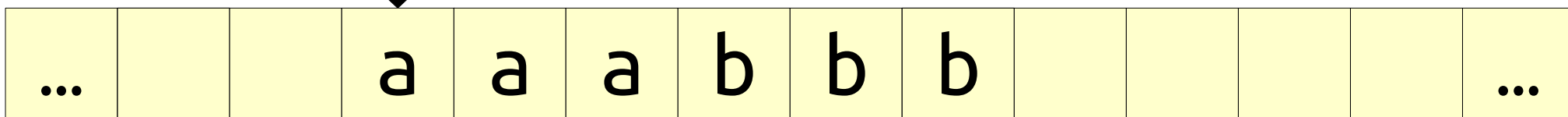
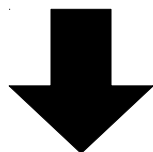
$$L = \{ \mathbf{a}^n \mathbf{b}^n \mid n \in \mathbb{N} \}$$



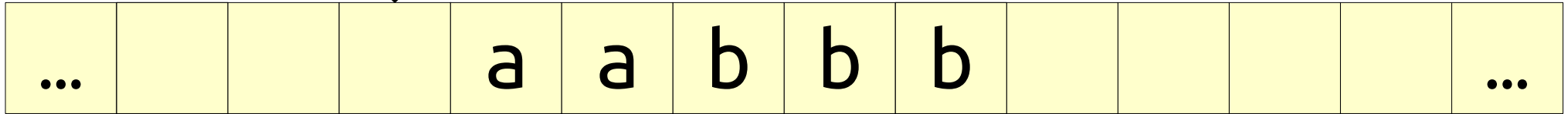
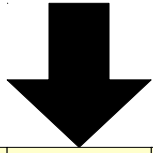
A Recursive Approach

- We can process our string using this recursive approach:
 - The string ε is in L .
 - The string **a** w **b** is in L if and only if w is in L .
 - Any string starting with **b** is not in L .
 - Any string ending with **a** is not in L .
- All that's left to do now is write a TM that implements this.

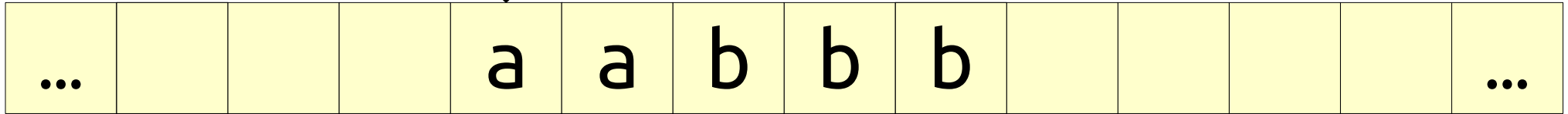
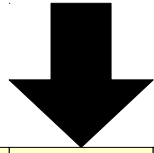
A Sketch of our TM



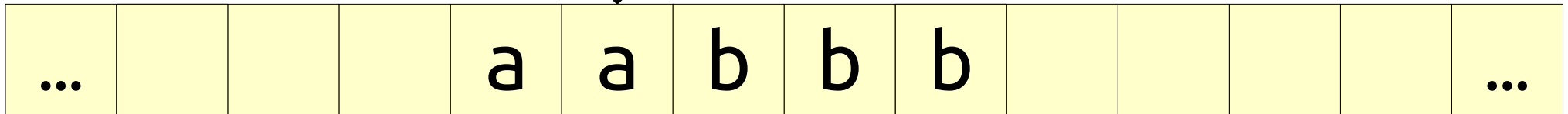
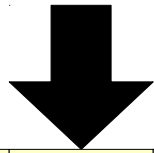
A Sketch of our TM



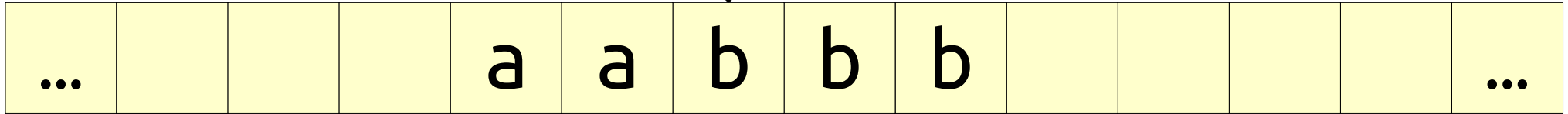
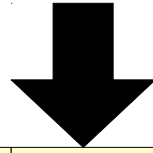
A Sketch of our TM



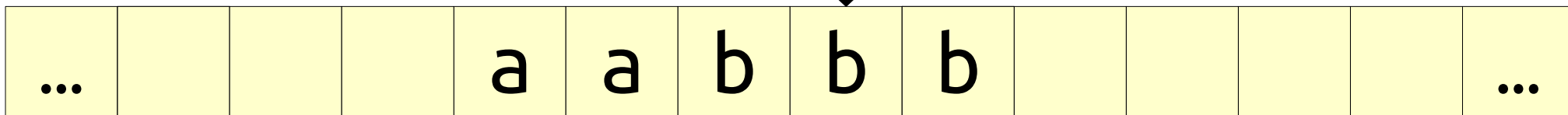
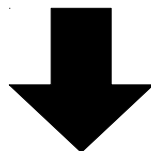
A Sketch of our TM



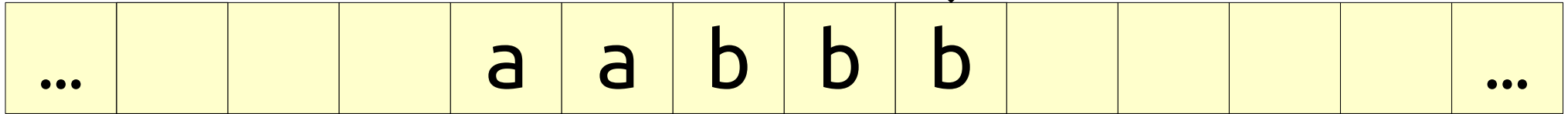
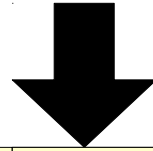
A Sketch of our TM



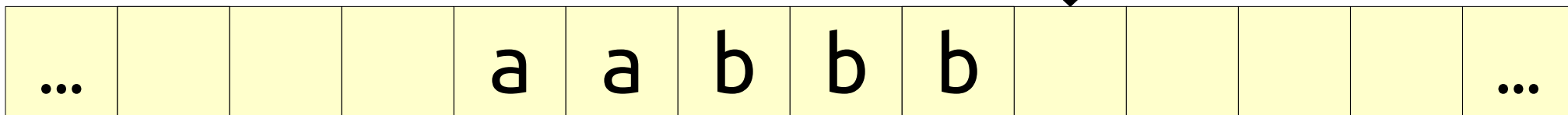
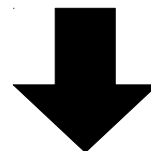
A Sketch of our TM



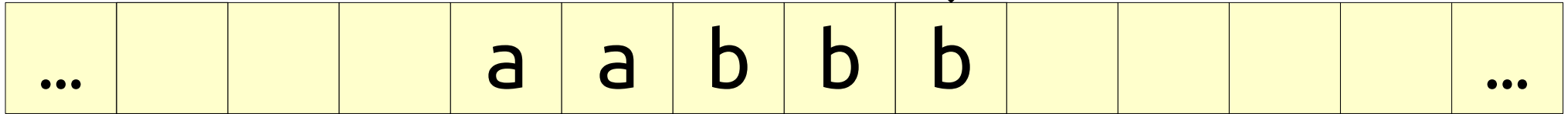
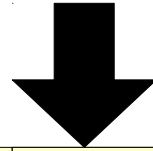
A Sketch of our TM



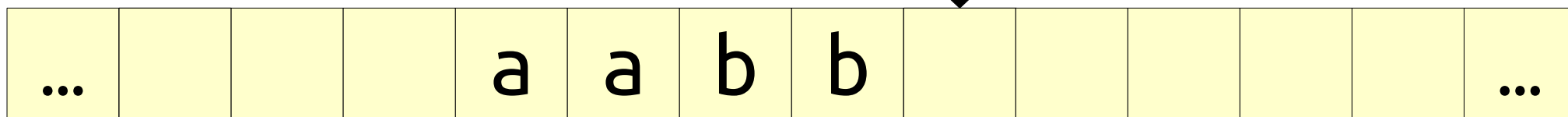
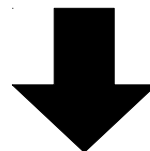
A Sketch of our TM



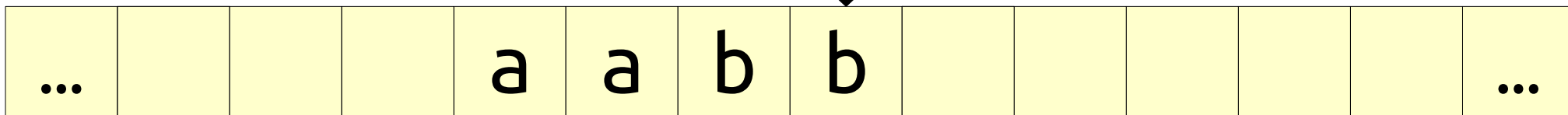
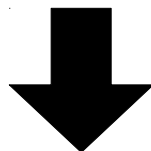
A Sketch of our TM



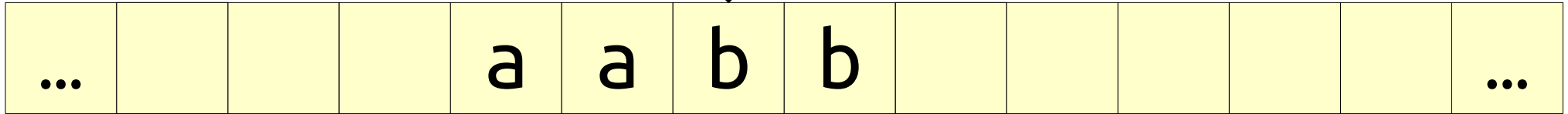
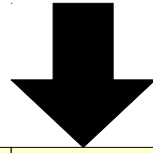
A Sketch of our TM



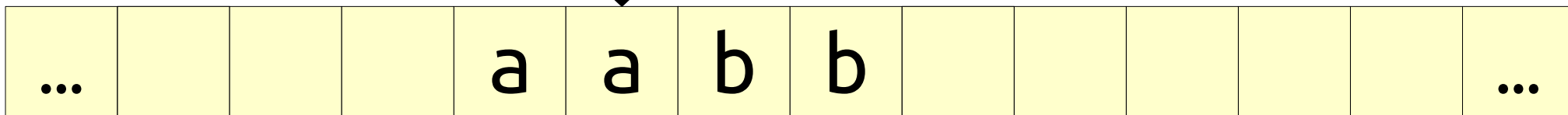
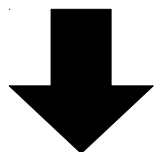
A Sketch of our TM



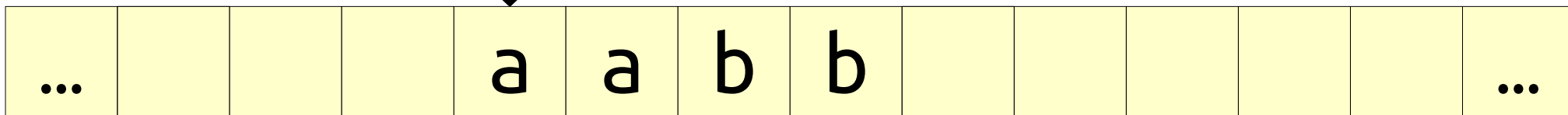
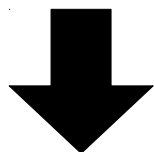
A Sketch of our TM



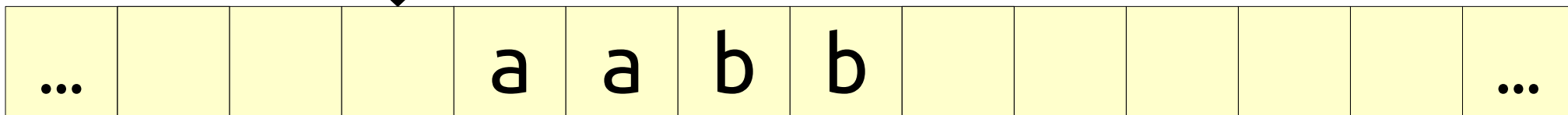
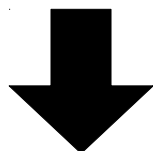
A Sketch of our TM



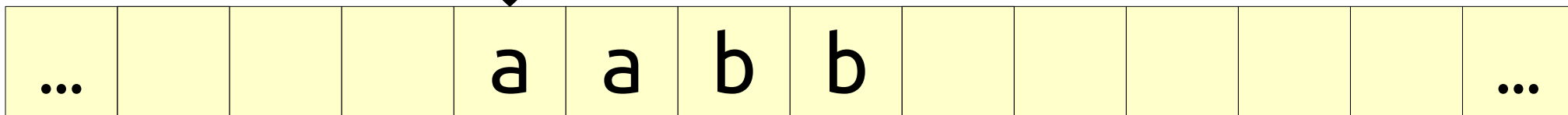
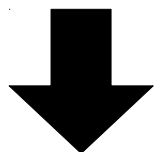
A Sketch of our TM



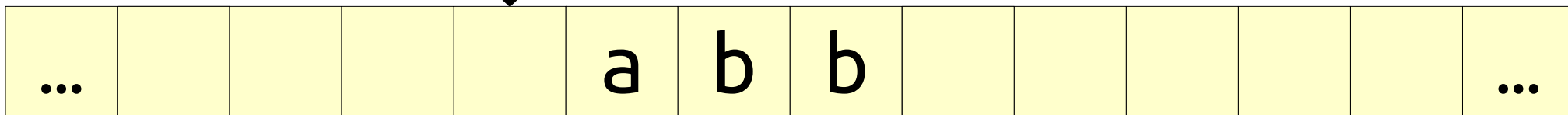
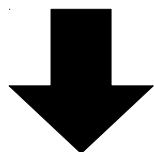
A Sketch of our TM



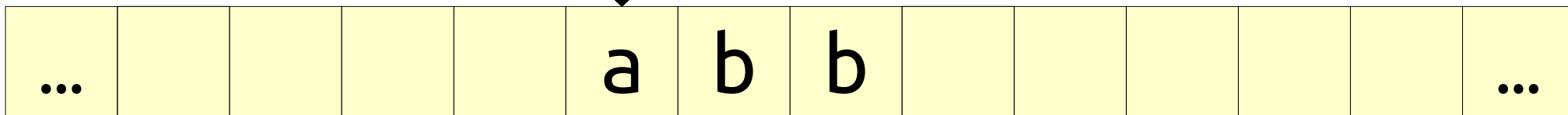
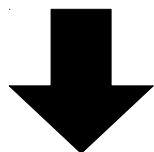
A Sketch of our TM



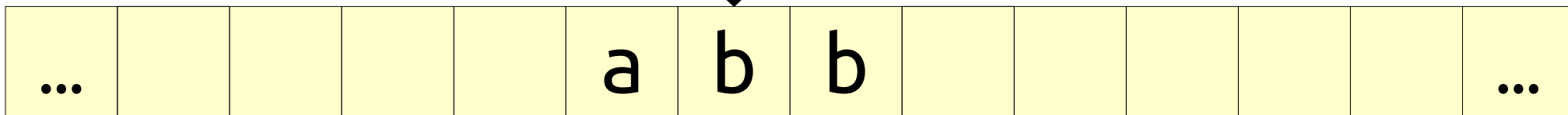
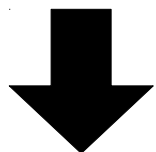
A Sketch of our TM



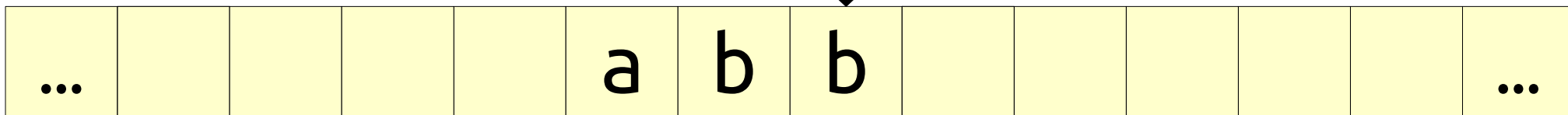
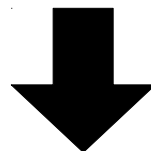
A Sketch of our TM



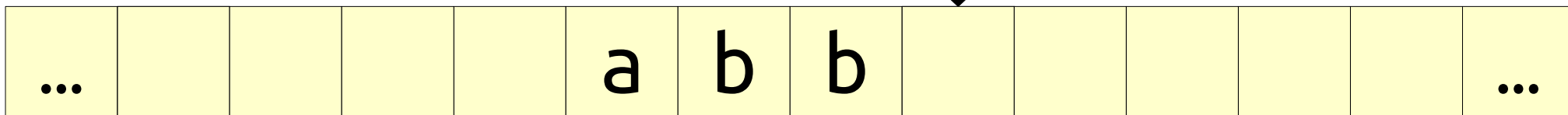
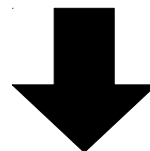
A Sketch of our TM



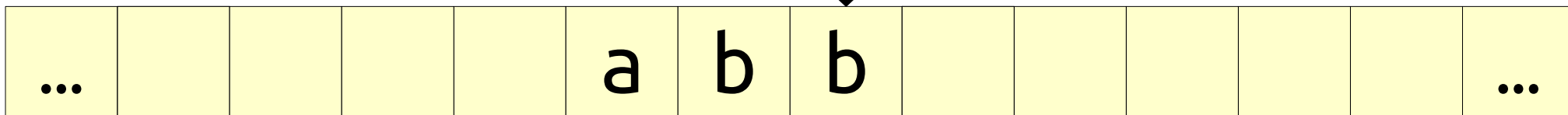
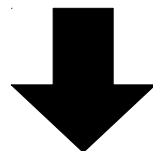
A Sketch of our TM



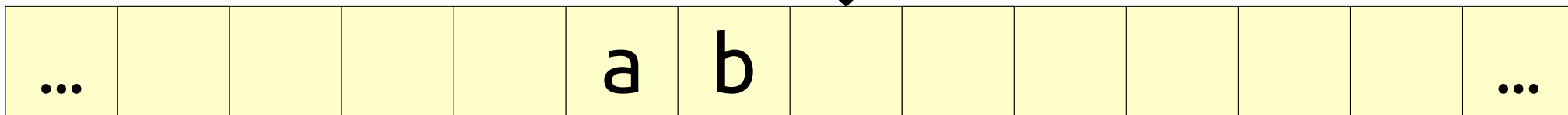
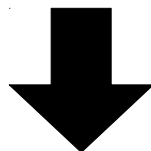
A Sketch of our TM



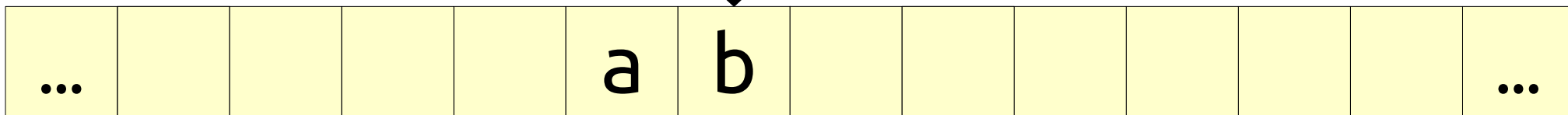
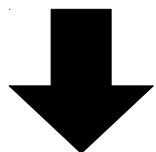
A Sketch of our TM



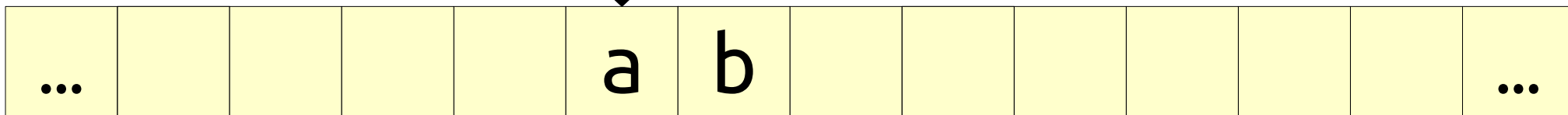
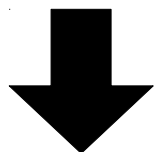
A Sketch of our TM



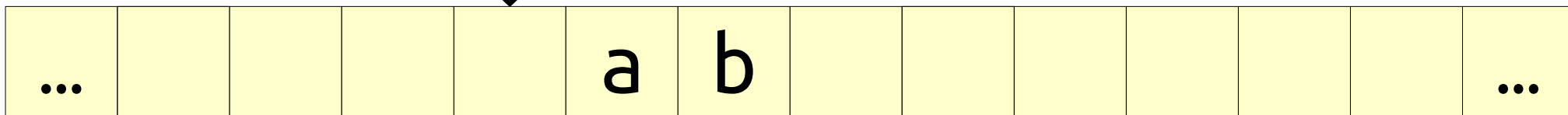
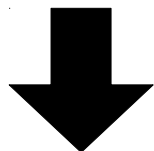
A Sketch of our TM



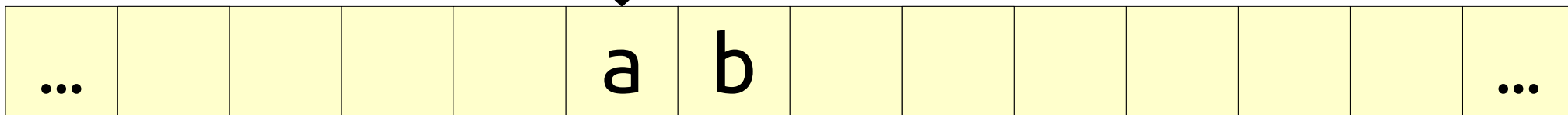
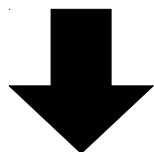
A Sketch of our TM



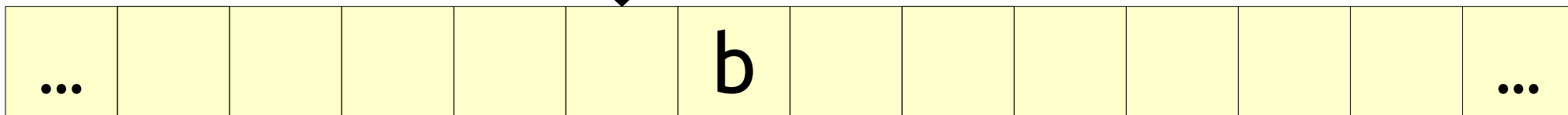
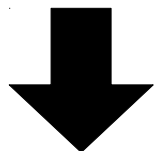
A Sketch of our TM



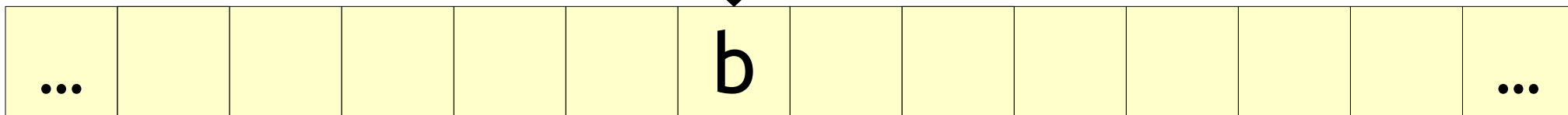
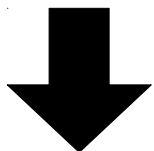
A Sketch of our TM



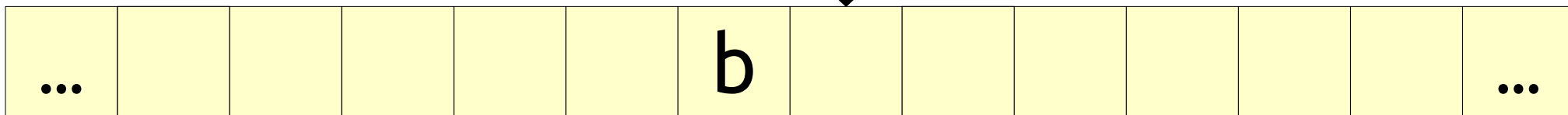
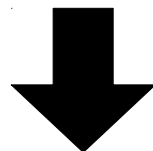
A Sketch of our TM



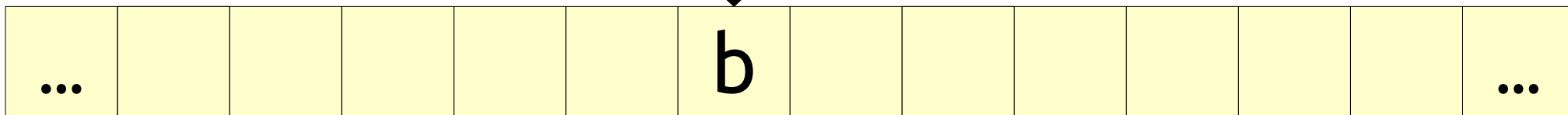
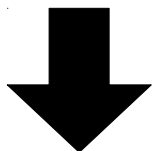
A Sketch of our TM



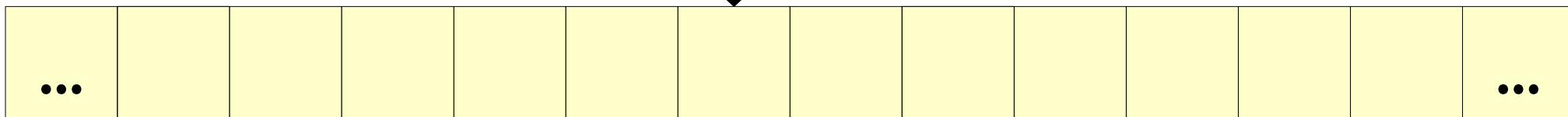
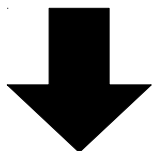
A Sketch of our TM

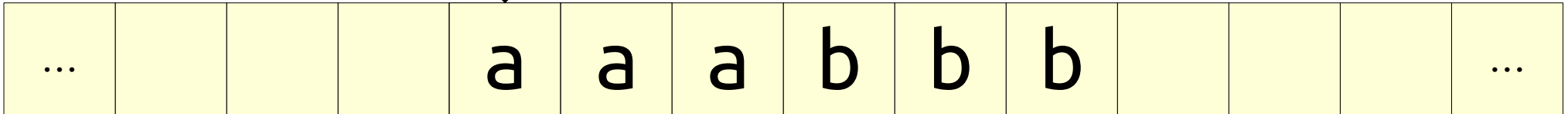
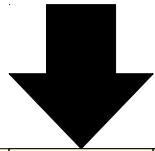


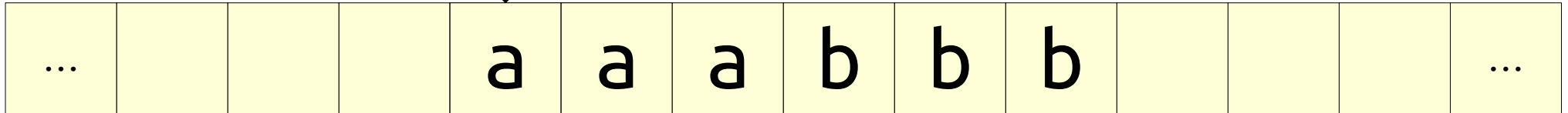
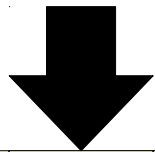
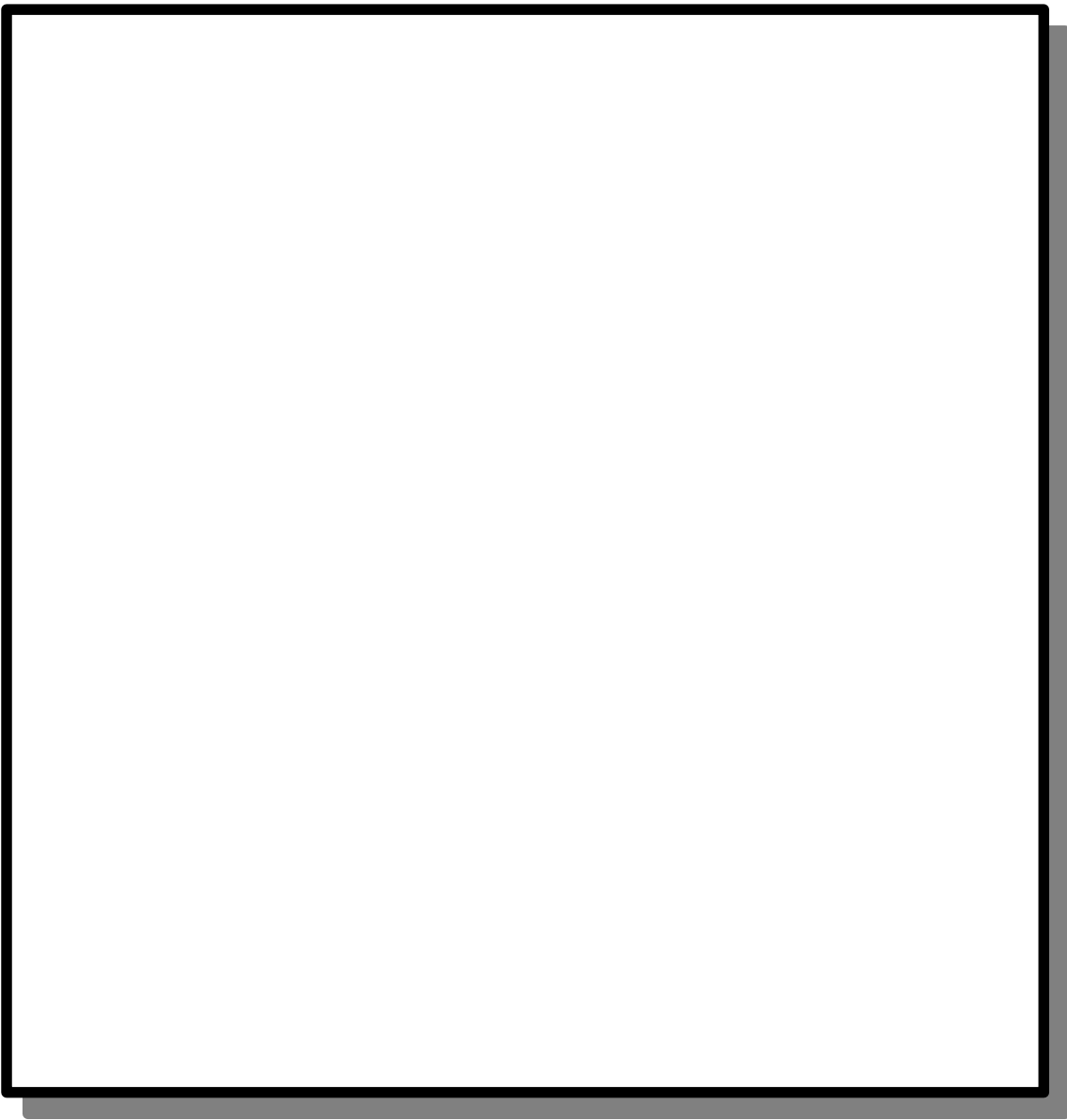
A Sketch of our TM



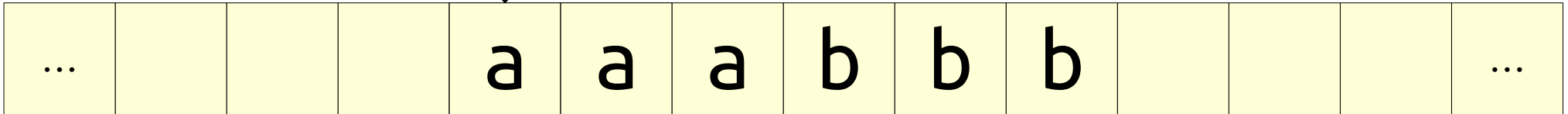
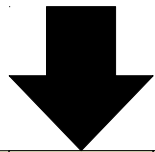
A Sketch of our TM



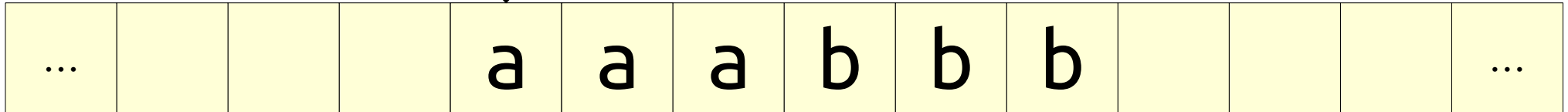
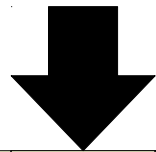




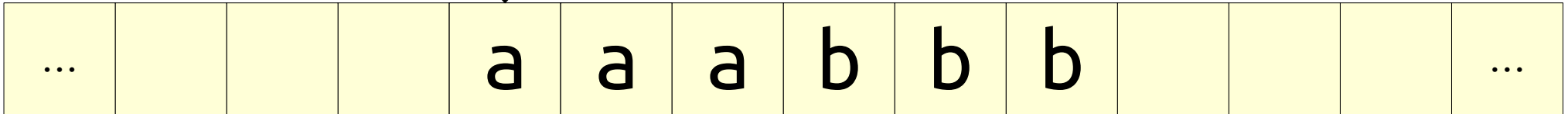
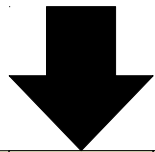
Start:



Start:

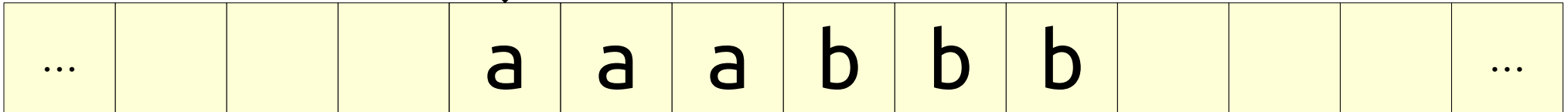
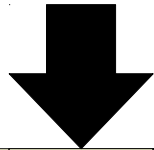


Start:

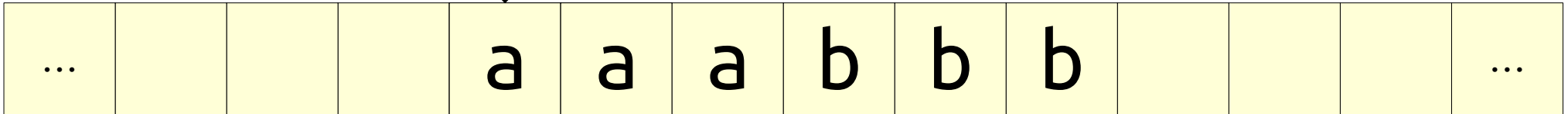
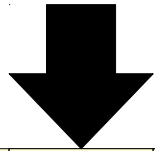


Start:

If Blank Return True



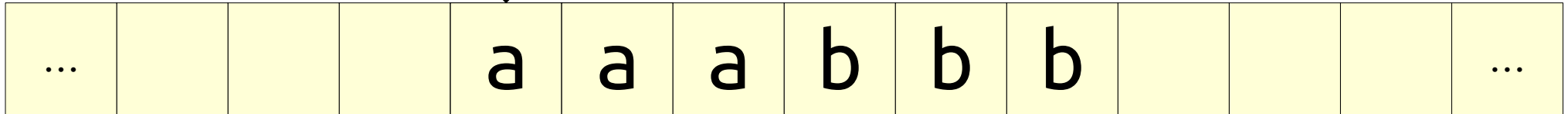
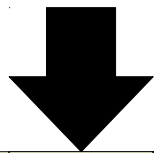
Start:
If Blank Return True



Start:

If Blank Return True

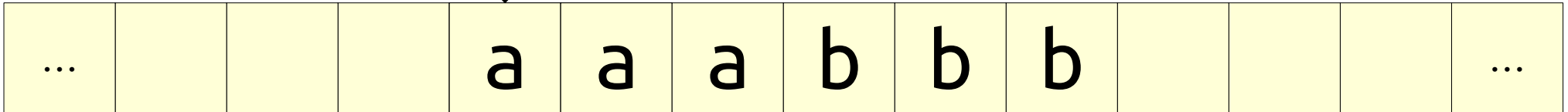
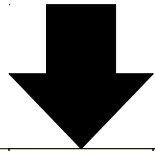
If 'b' Return False



Start:

If Blank Return True

If 'b' Return False

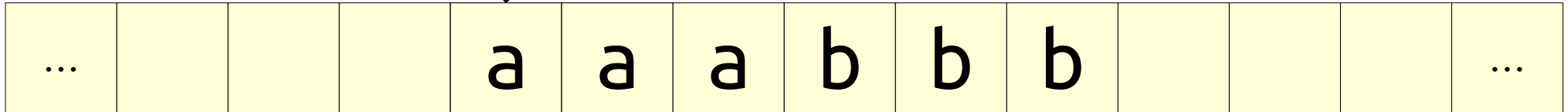
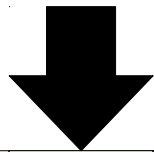


Start:

If Blank Return True

If 'b' Return False

Write Blank

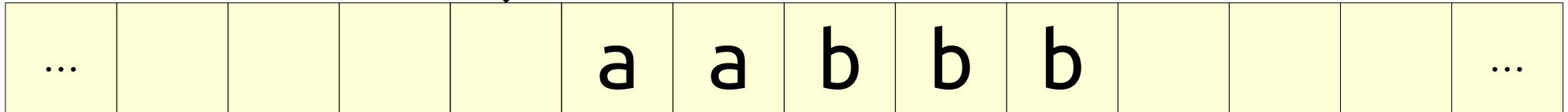
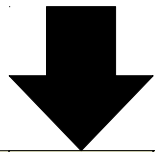


Start:

If Blank Return True

If 'b' Return False

Write Blank

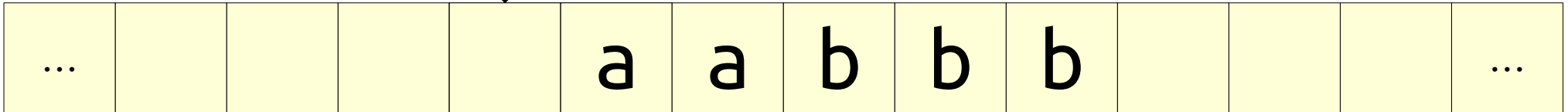
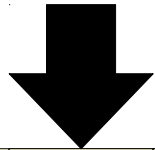


Start:

If Blank Return True

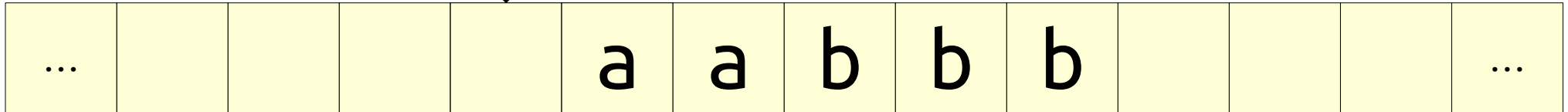
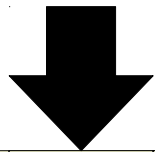
If 'b' Return False

Write Blank



```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:



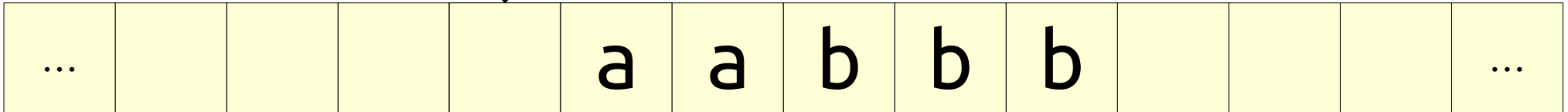
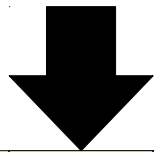
Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:



Start:

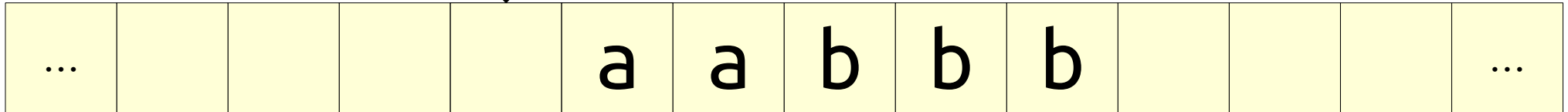
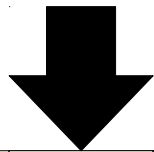
If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right



Start:

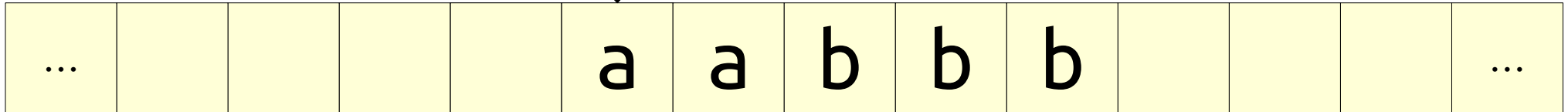
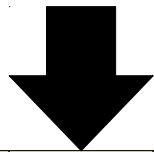
If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right



Start:

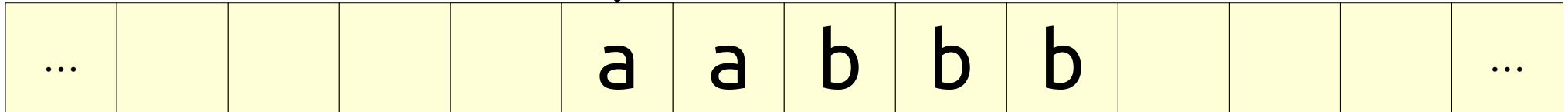
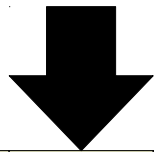
If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right



Start:

If Blank Return True

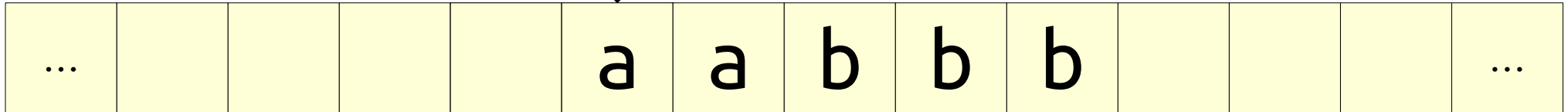
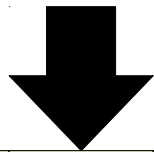
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

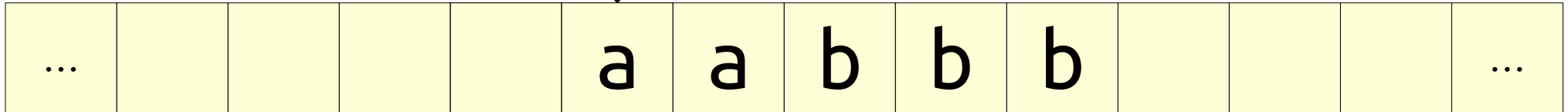
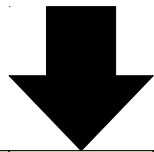
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

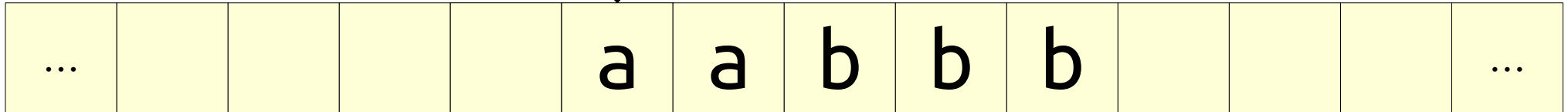
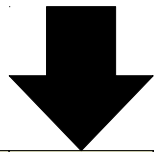
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

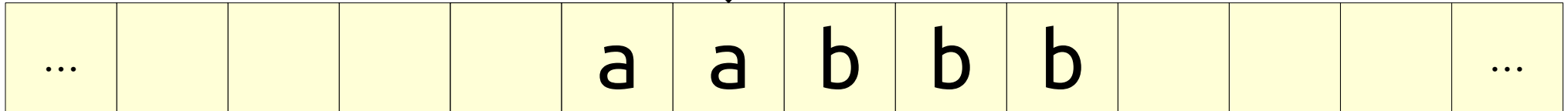
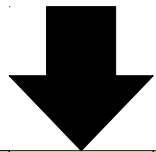
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

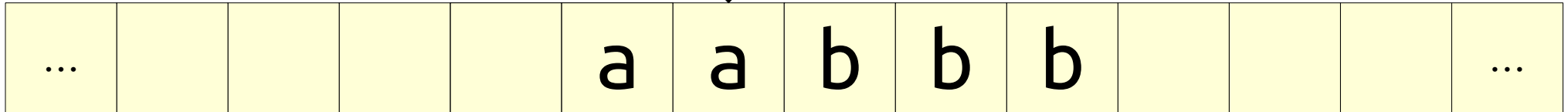
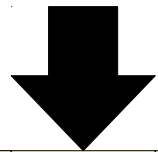
If 'b' Return False

Write Blank

ZipRight:

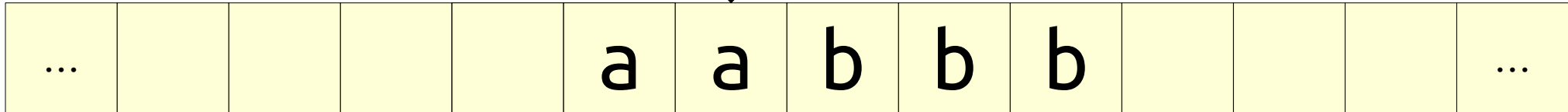
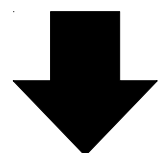
Move Right

If Not Blank Goto ZipRight



```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

```
ZipRight:  
  Move Right  
  If Not Blank Goto ZipRight
```



Start:

If Blank Return True

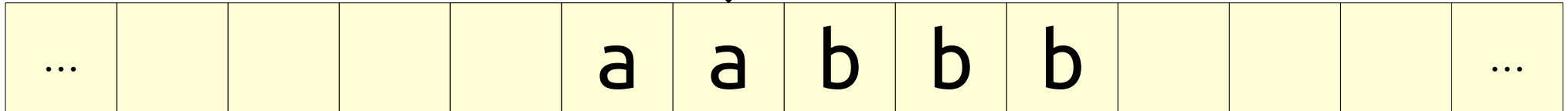
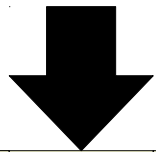
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

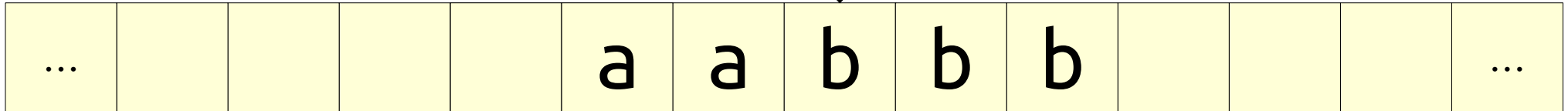
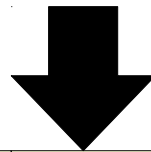
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

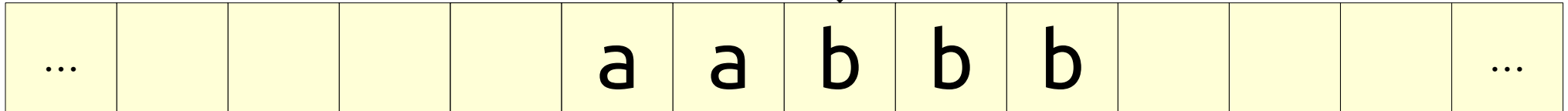
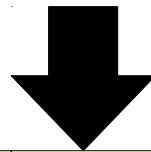
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

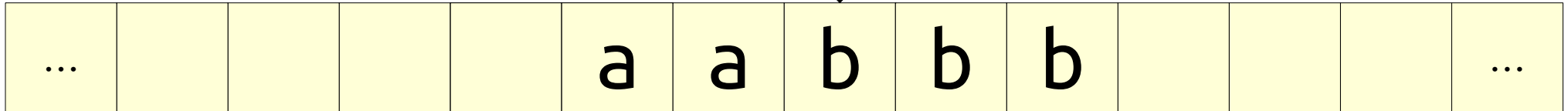
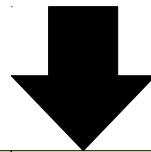
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

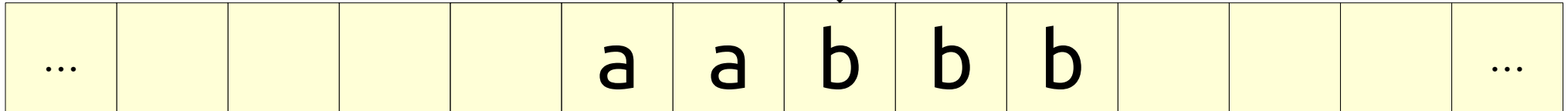
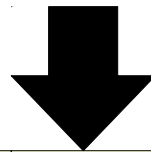
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

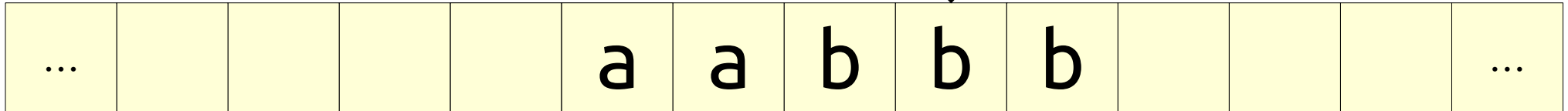
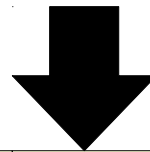
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

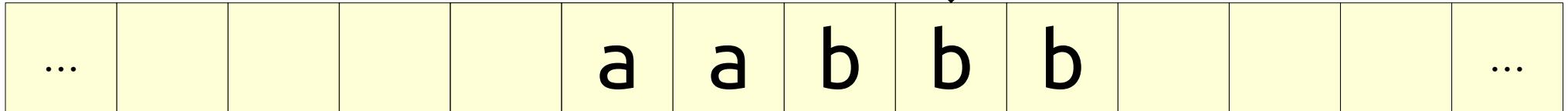
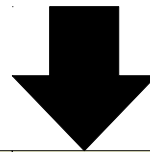
If 'b' Return False

Write Blank

ZipRight:

Move Right

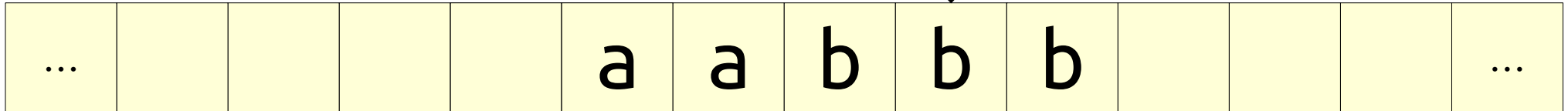
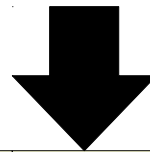
If Not Blank Goto ZipRight



```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

```
ZipRight:
```

```
  Move Right  
  If Not Blank Goto ZipRight
```



Start:

If Blank Return True

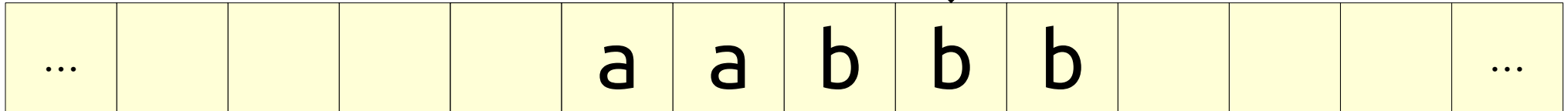
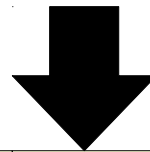
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

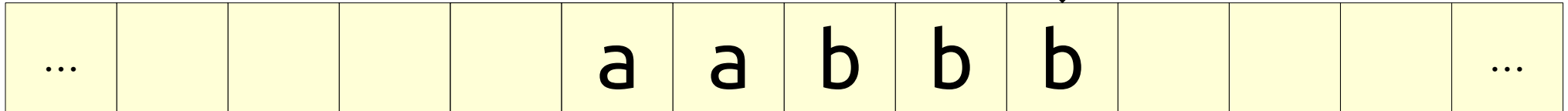
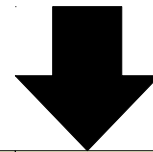
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

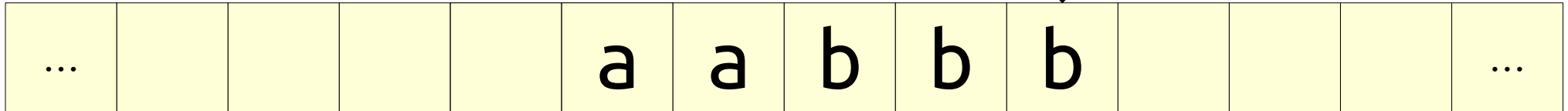
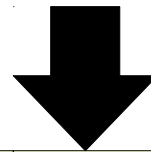
If 'b' Return False

Write Blank

ZipRight:

Move Right

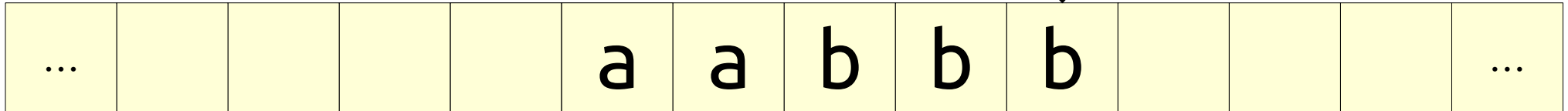
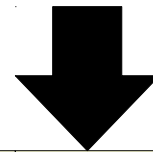
If Not Blank Goto ZipRight



```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:

```
  Move Right  
  If Not Blank Goto ZipRight
```



Start:

If Blank Return True

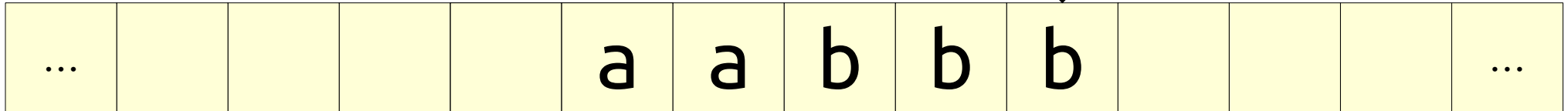
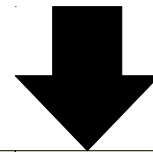
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

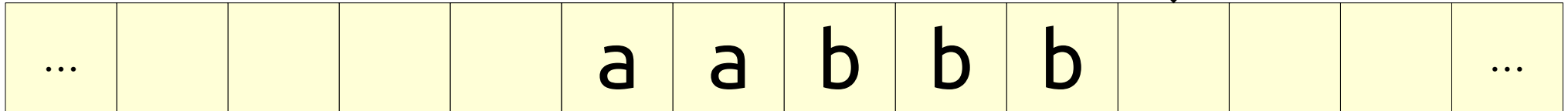
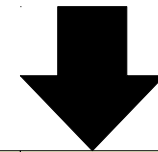
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

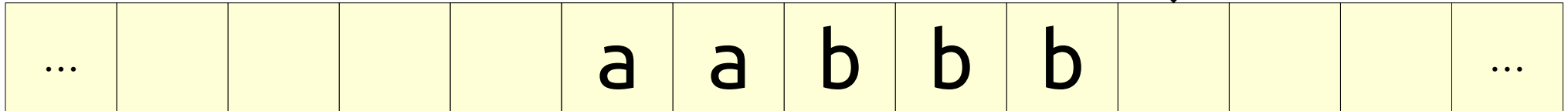
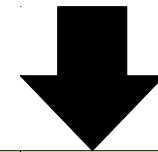
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

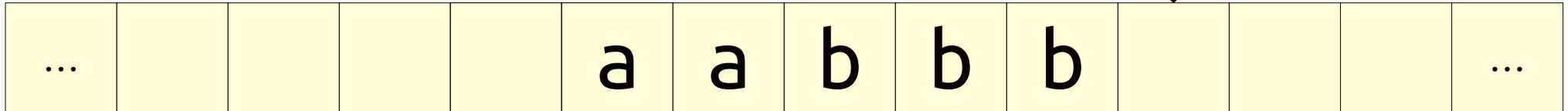
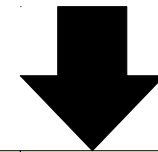
If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight



Start:

If Blank Return True

If 'b' Return False

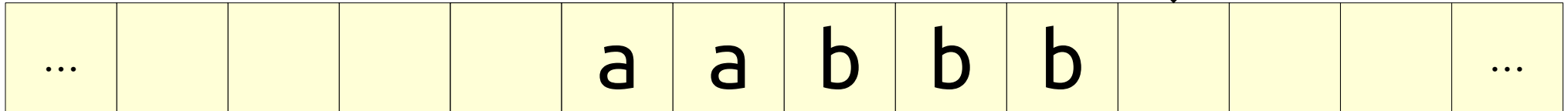
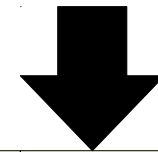
Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left



Start:

If Blank Return True

If 'b' Return False

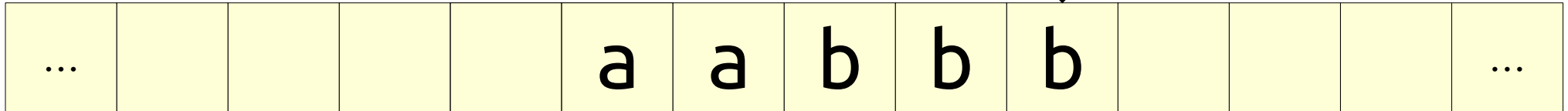
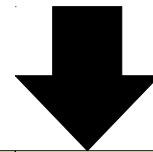
Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left



Start:

If Blank Return True

If 'b' Return False

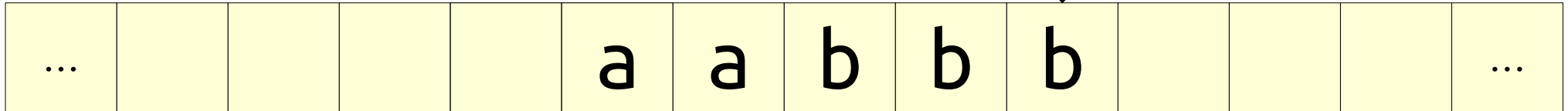
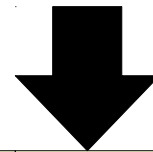
Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left



Start:

If Blank Return True

If 'b' Return False

Write Blank

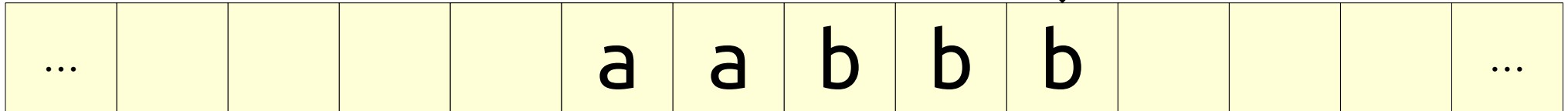
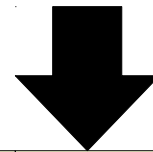
ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False



Start:

If Blank Return True

If 'b' Return False

Write Blank

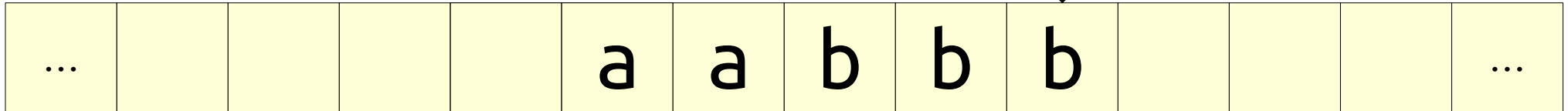
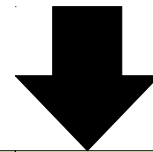
ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

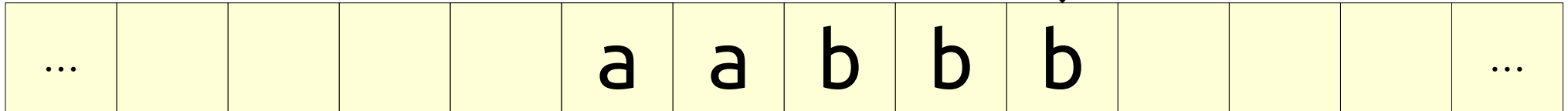
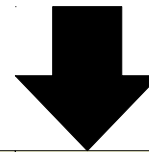
Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

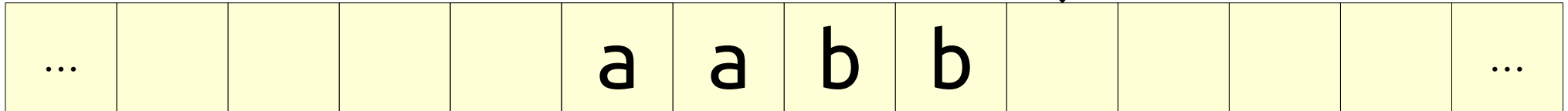
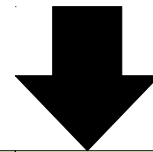
Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

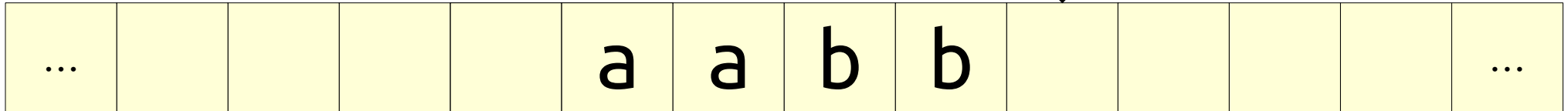
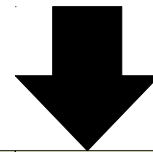
Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank



Start:

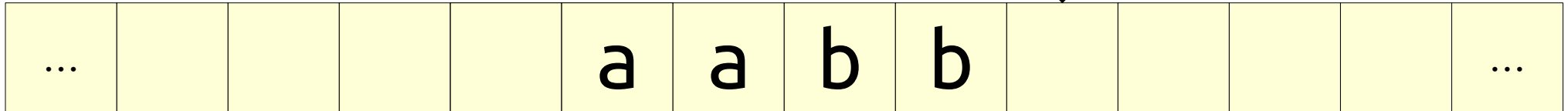
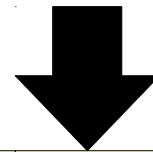
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

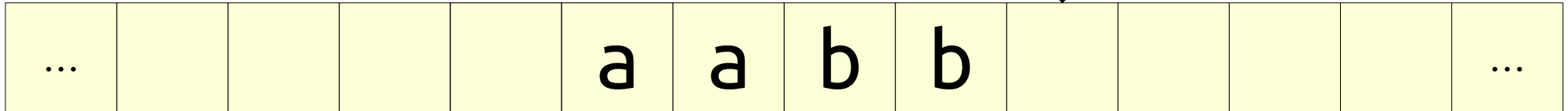
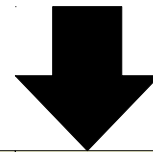
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

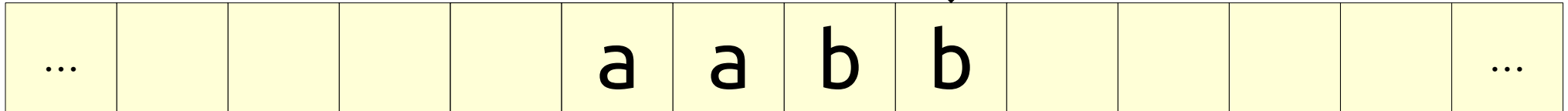
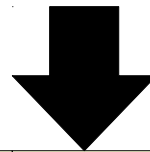
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

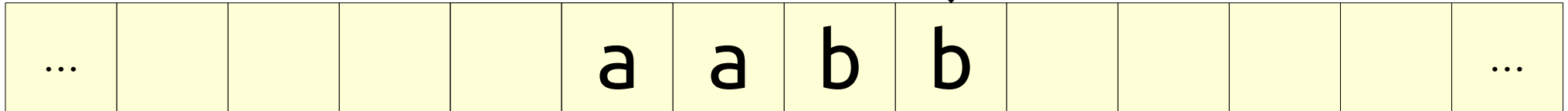
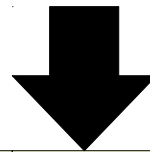
If Not 'b' Return False

Write Blank

ZipLeft:

Move Left

If Not Blank Goto ZipLeft



Start:

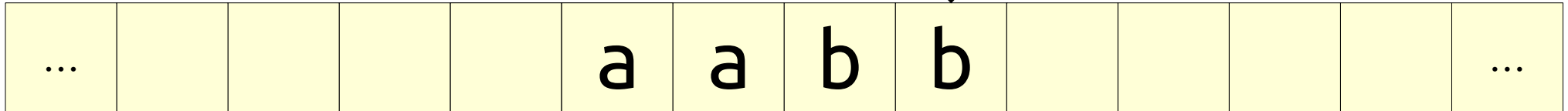
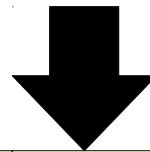
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

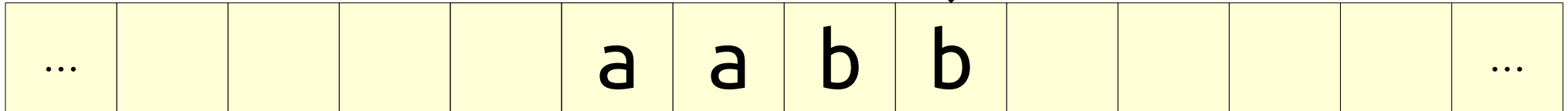
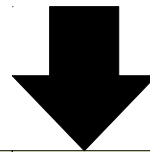
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

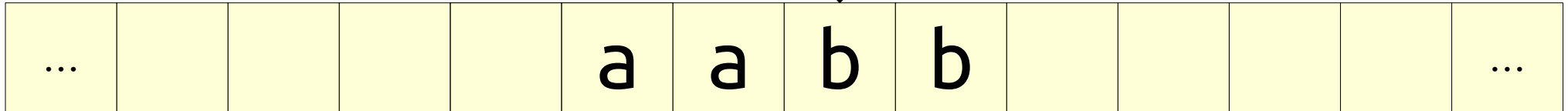
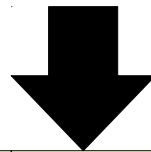
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

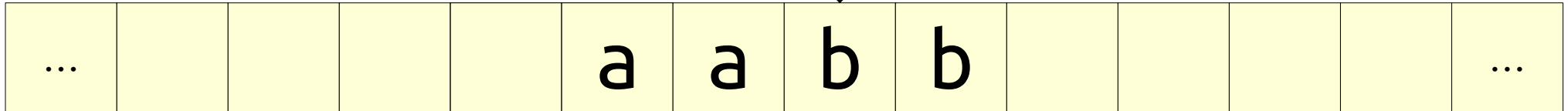
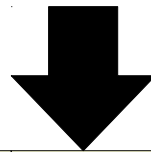
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

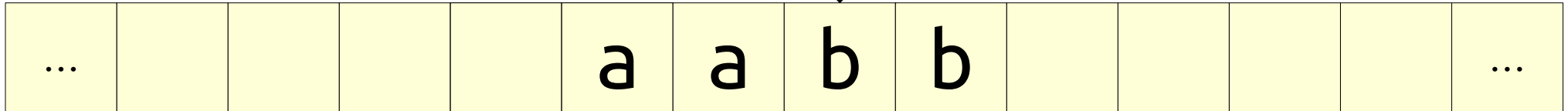
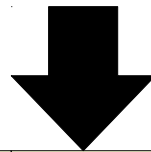
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

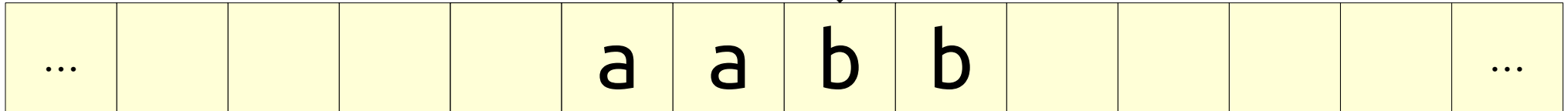
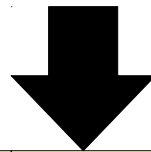
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

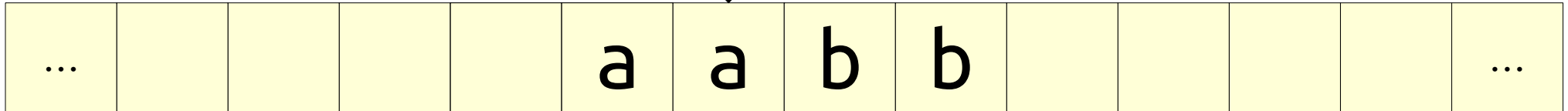
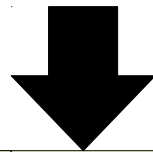
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

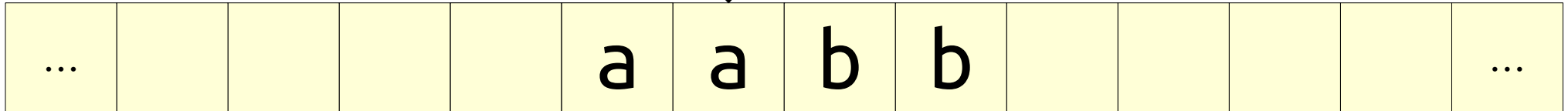
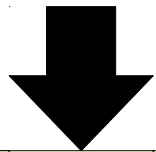
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

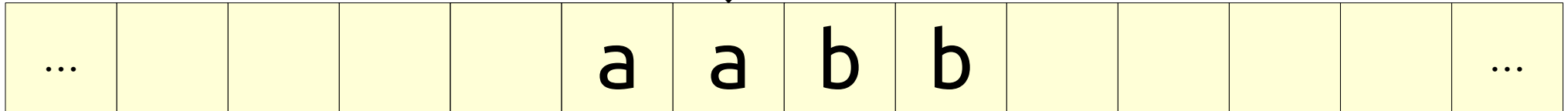
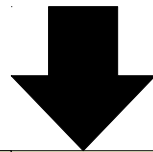
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

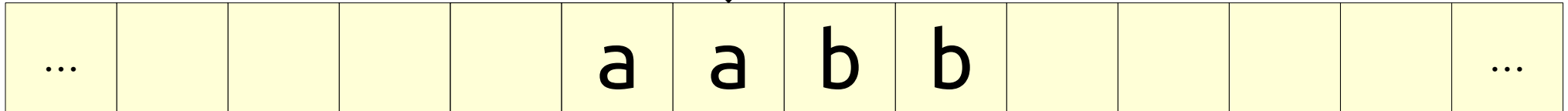
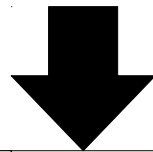
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

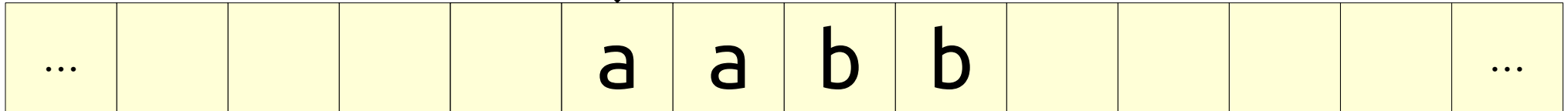
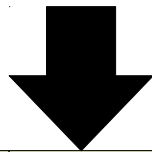
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

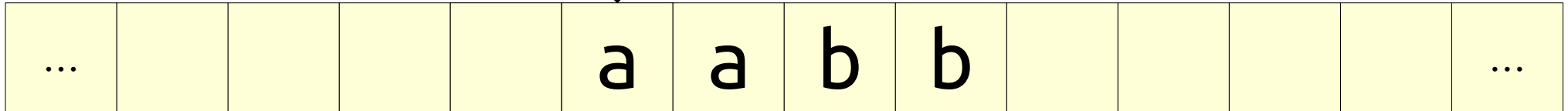
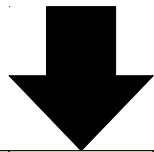
If Not 'b' Return False

Write Blank

ZipLeft:

Move Left

If Not Blank Goto ZipLeft



Start:

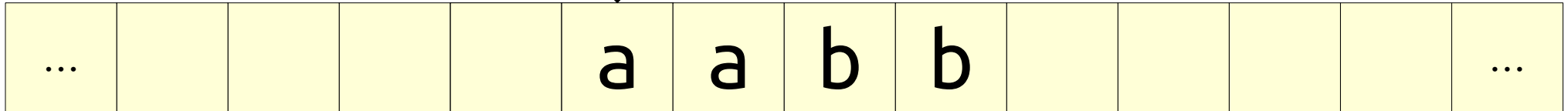
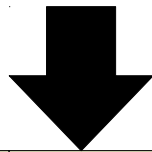
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

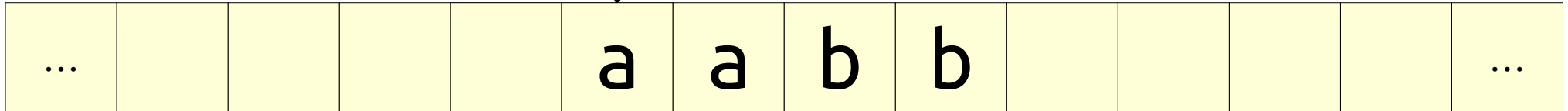
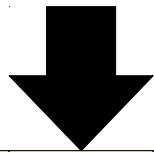
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

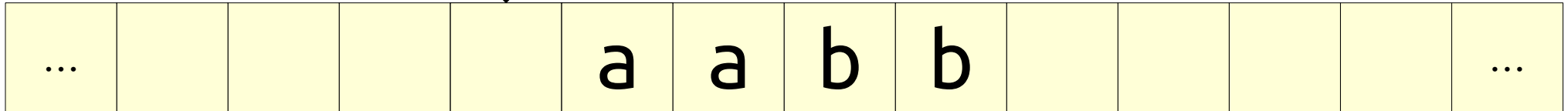
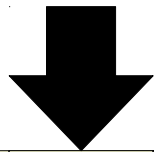
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

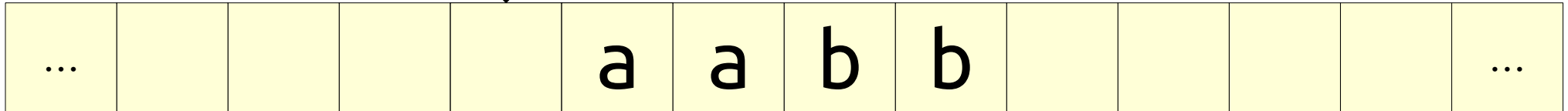
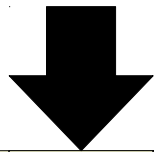
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

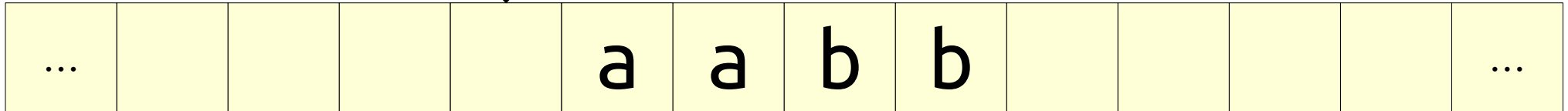
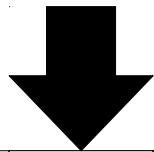
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft
```



Start:

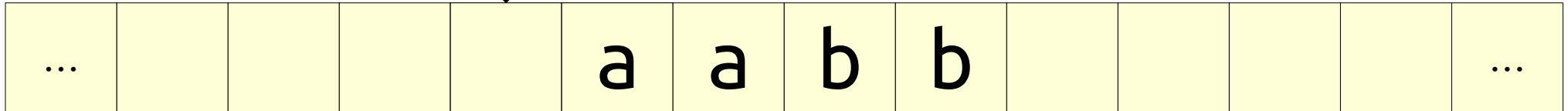
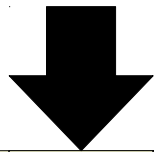
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right
```



Start:

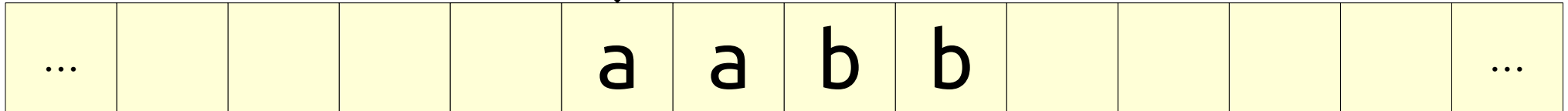
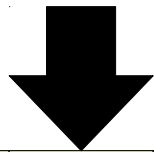
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right
```



Start:

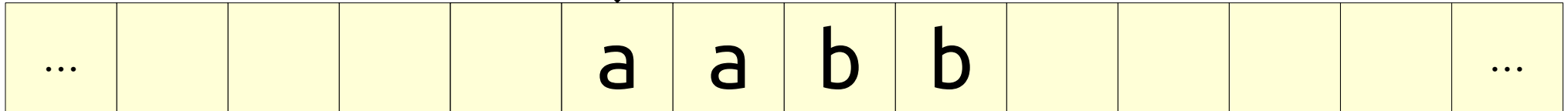
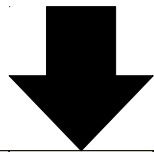
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right
```



Start:

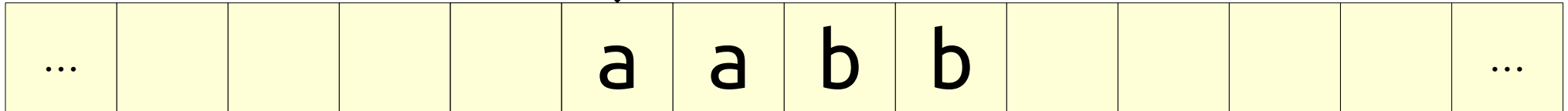
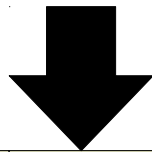
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

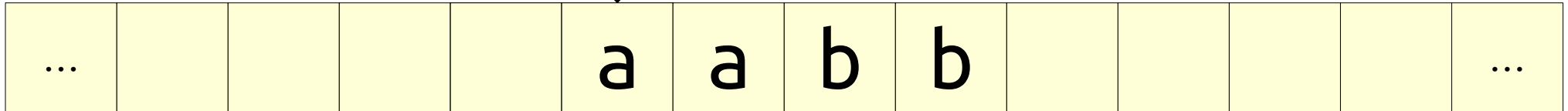
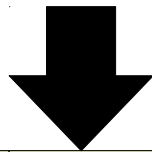
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

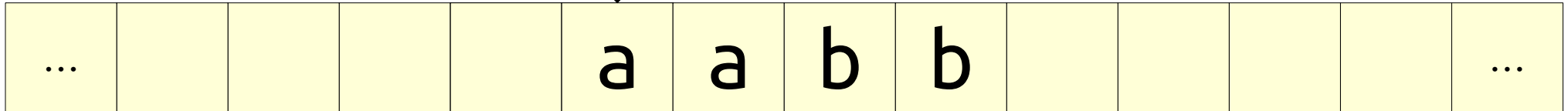
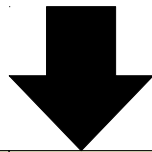
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

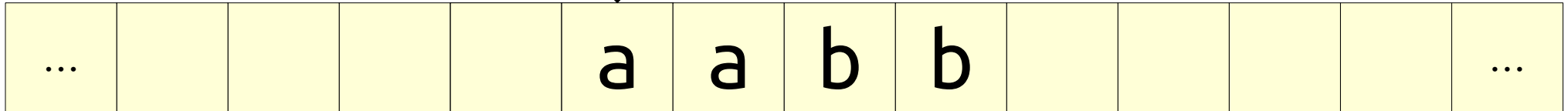
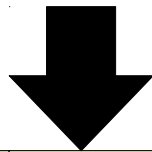
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

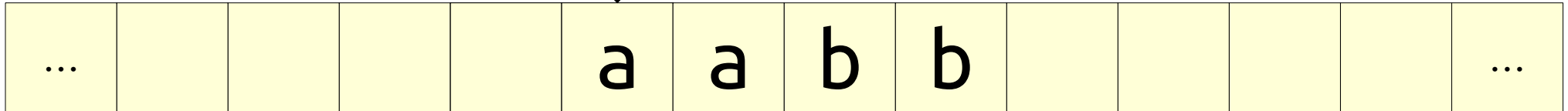
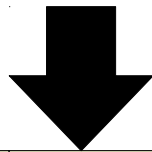
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

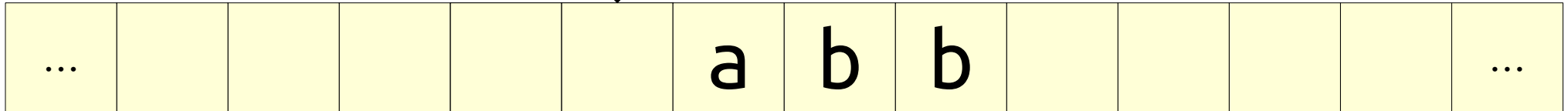
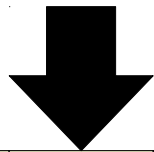
Goto Start



```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

```
ZipRight:  
  Move Right  
  If Not Blank Goto ZipRight  
  Move Left  
  If Not 'b' Return False  
  Write Blank
```

```
ZipLeft:  
  Move Left  
  If Not Blank Goto ZipLeft  
  Move Right  
  Goto Start
```



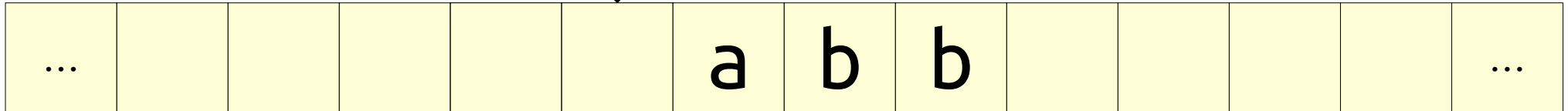
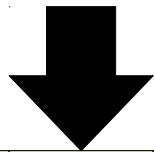
```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

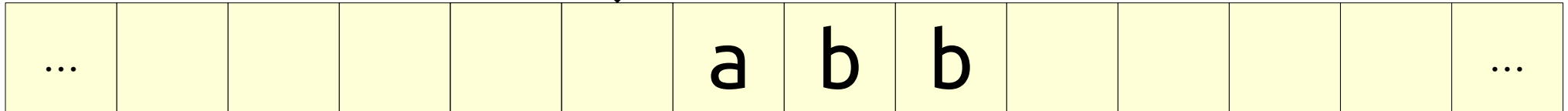
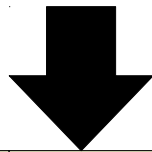
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

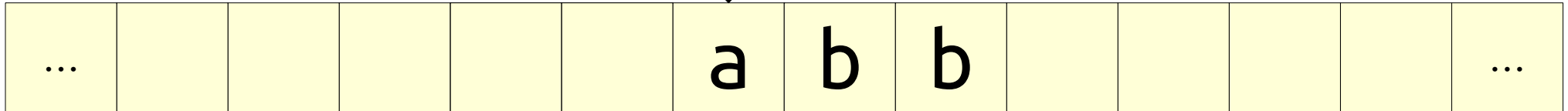
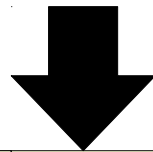
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

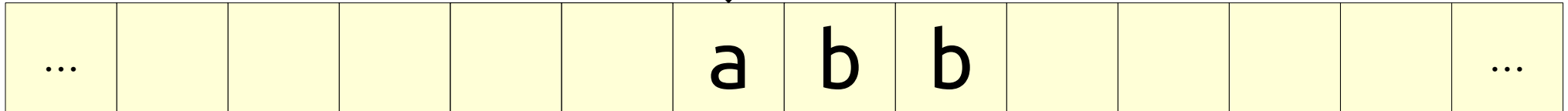
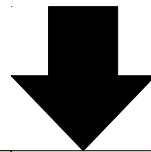
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```

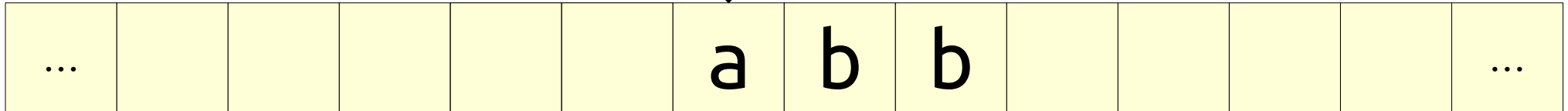
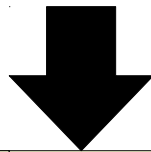


```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

```
ZipLeft:  
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

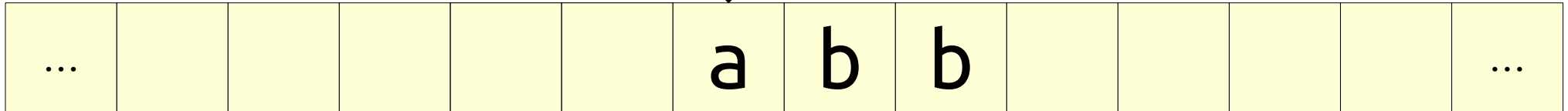
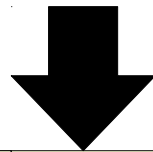
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

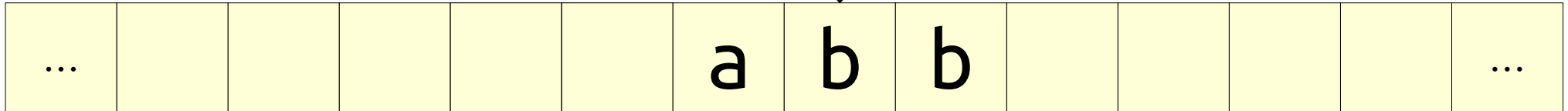
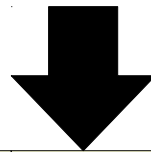
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

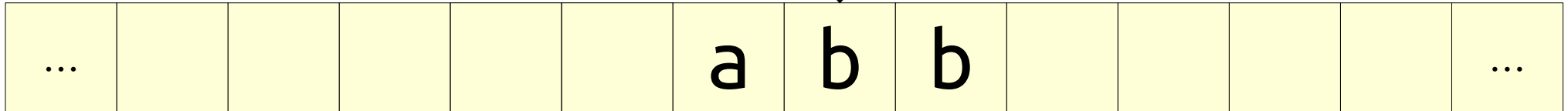
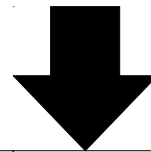
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



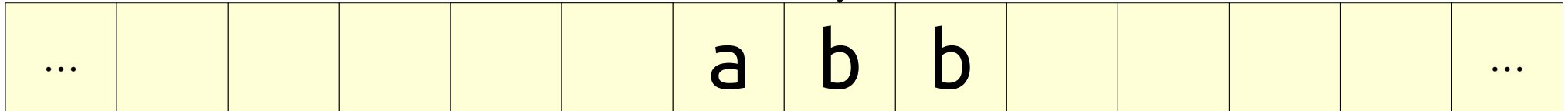
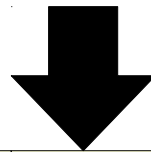
```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

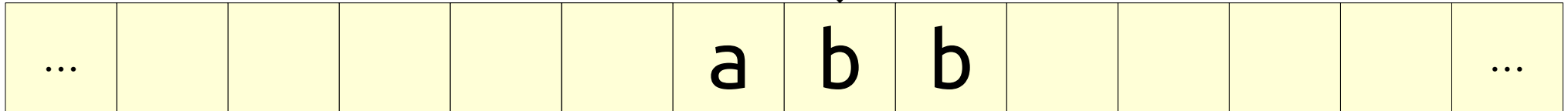
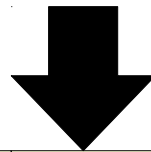
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

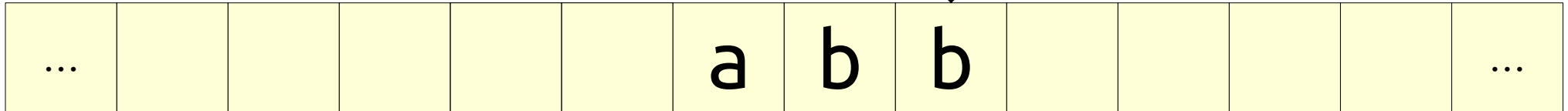
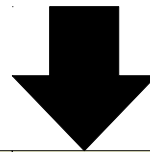
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

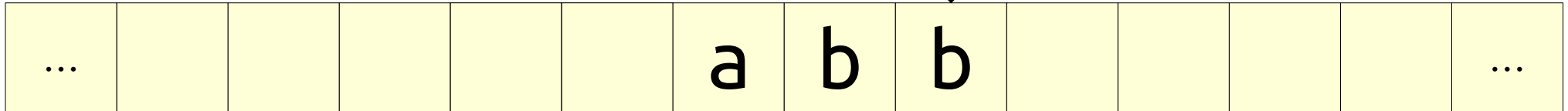
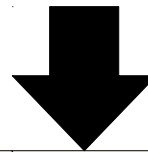
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start

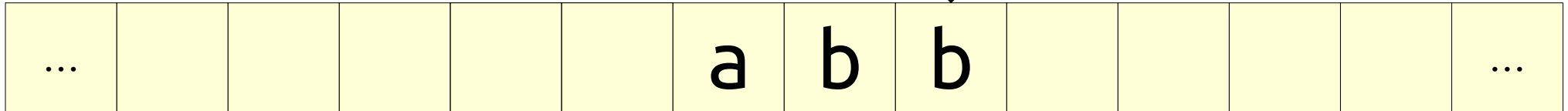
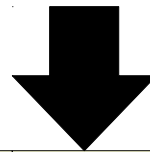


```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

```
ZipLeft:  
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

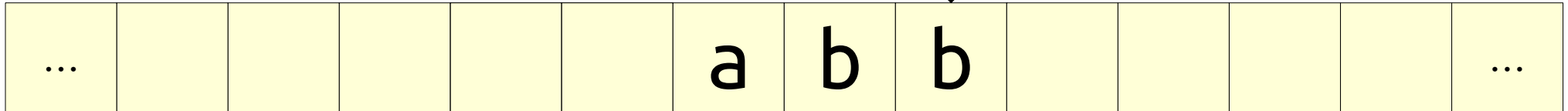
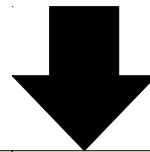
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

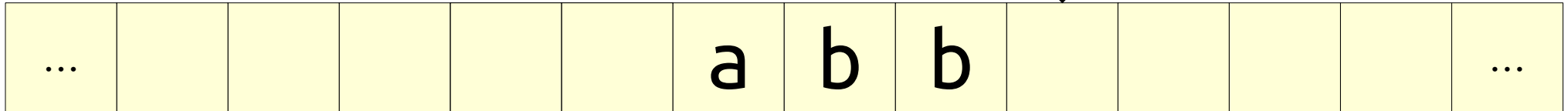
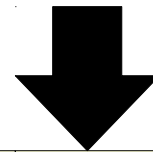
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

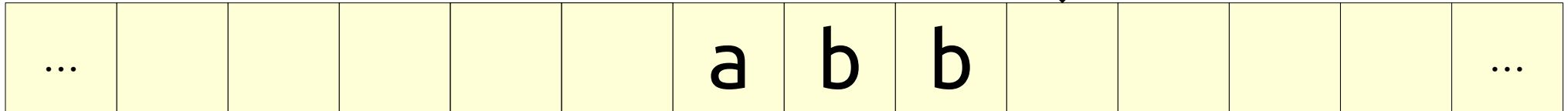
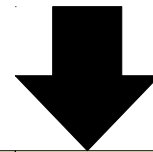
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

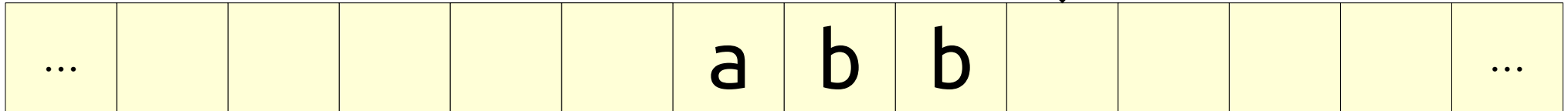
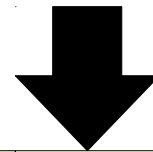
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

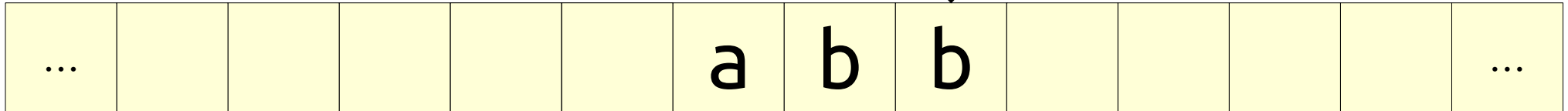
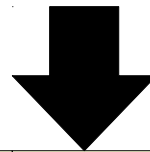
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

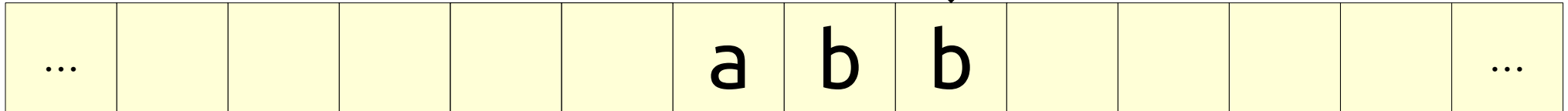
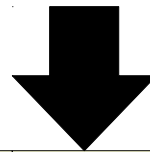
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

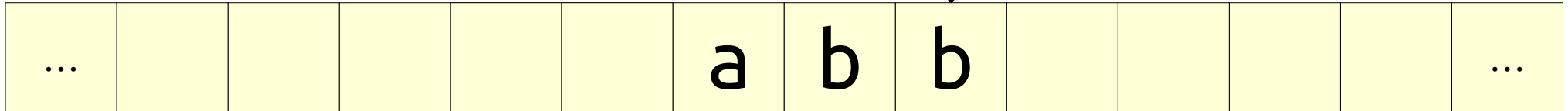
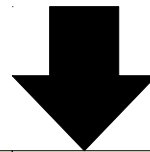
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

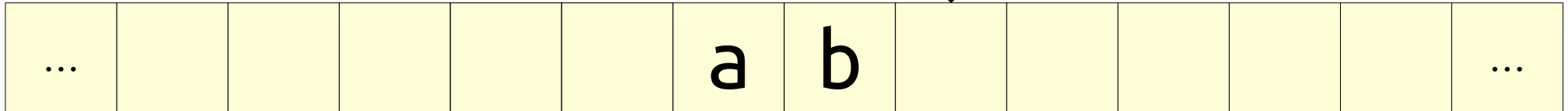
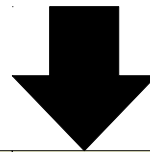
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

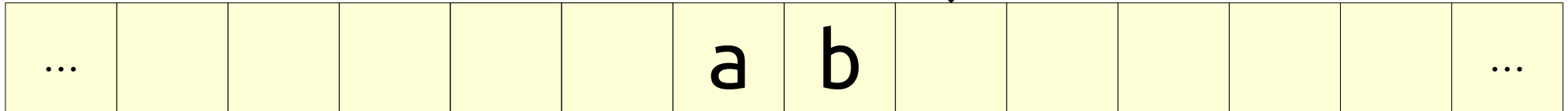
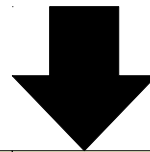
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

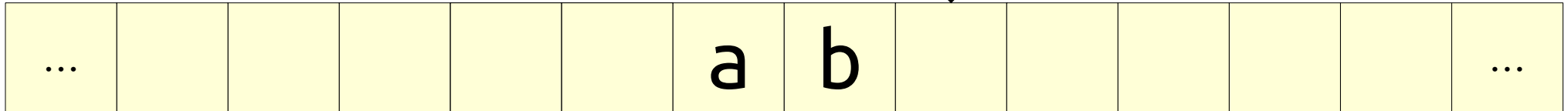
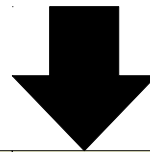
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

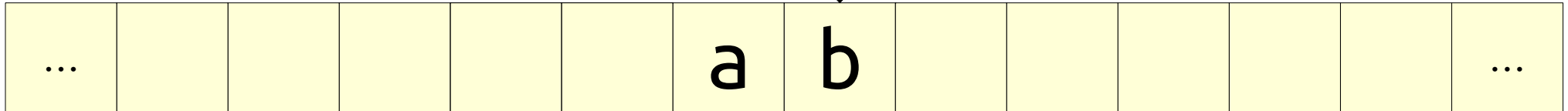
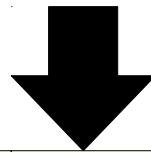
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

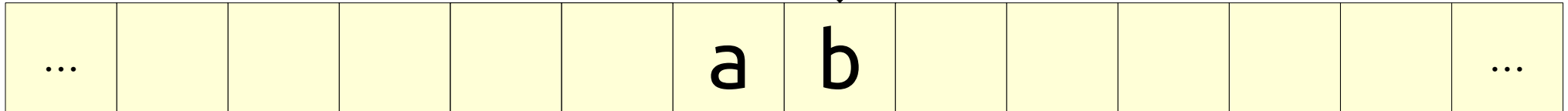
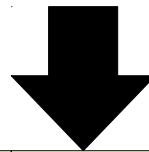
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

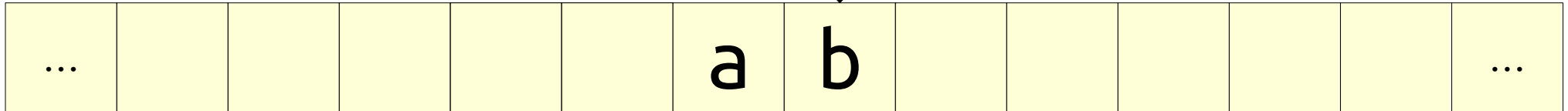
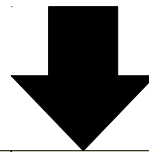
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

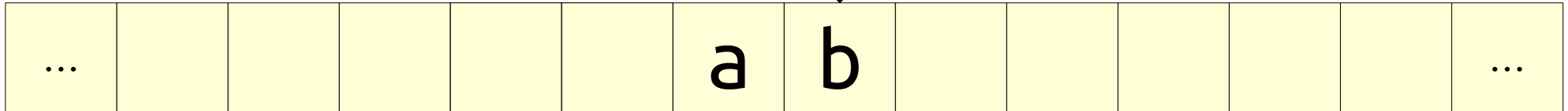
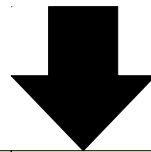
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

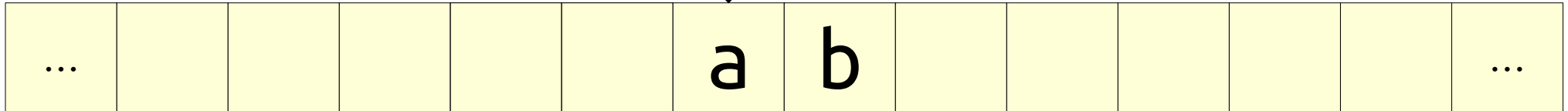
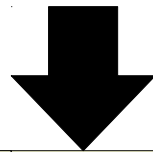
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

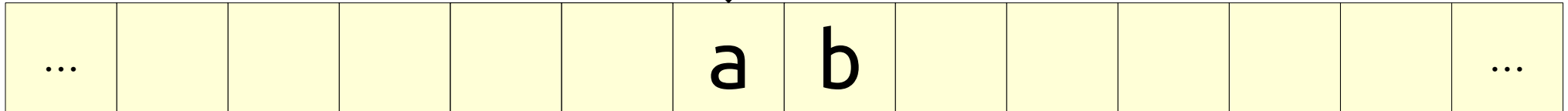
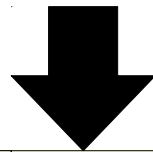
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

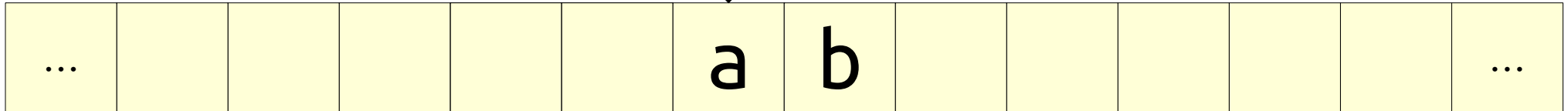
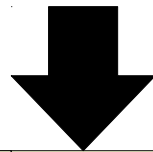
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

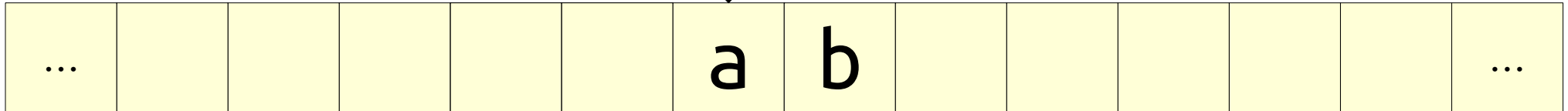
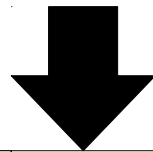
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

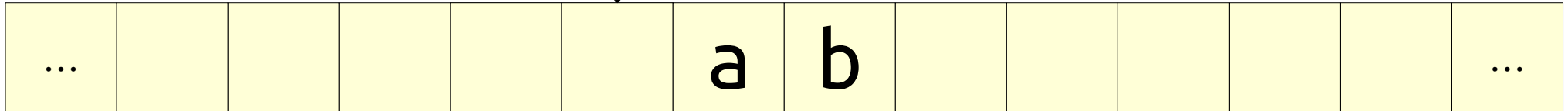
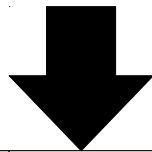
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

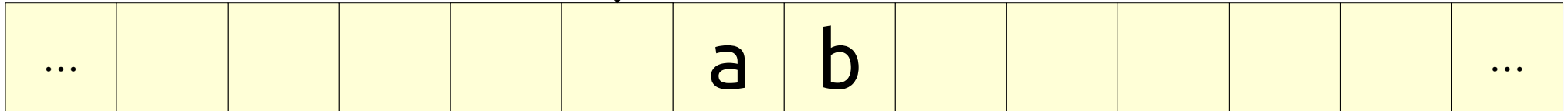
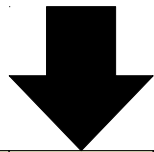
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

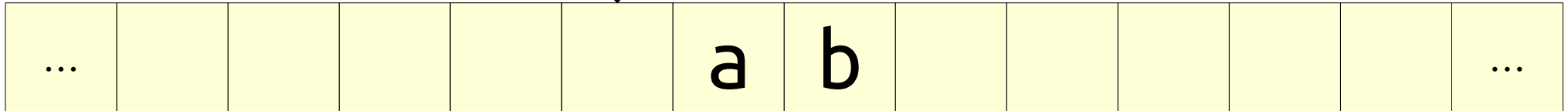
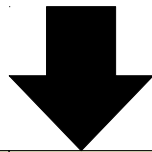
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

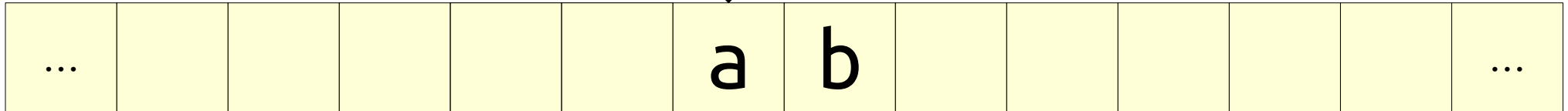
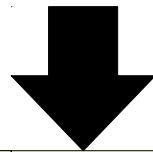
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

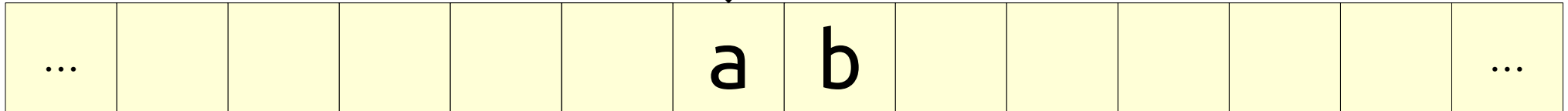
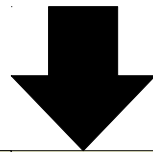
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

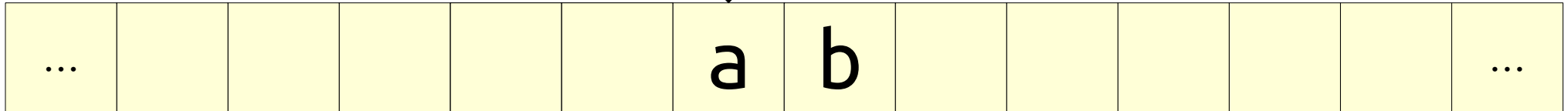
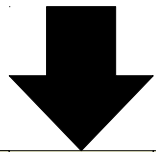
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

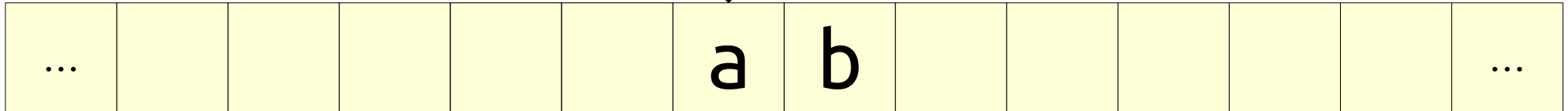
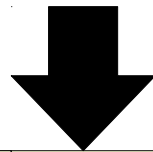
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

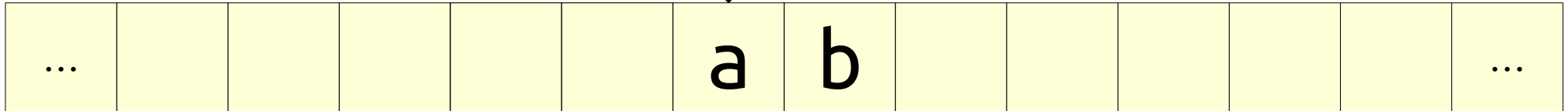
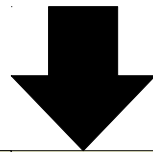
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

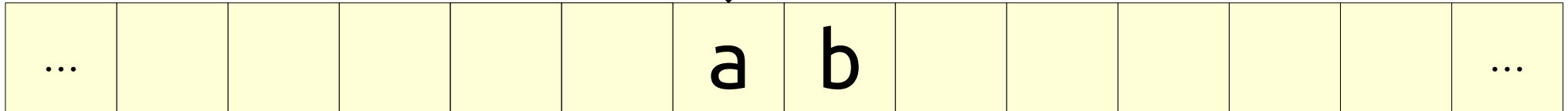
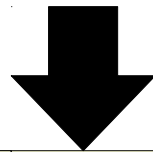
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

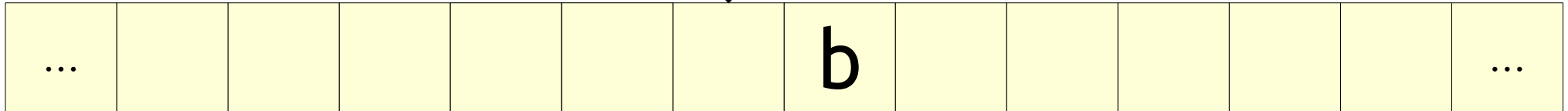
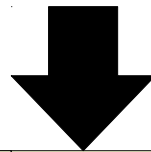
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start

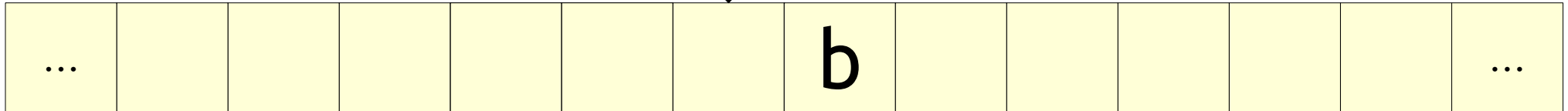
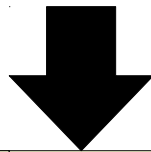


```
Start:  
  If Blank Return True  
  If 'b' Return False  
  Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

```
ZipLeft:  
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

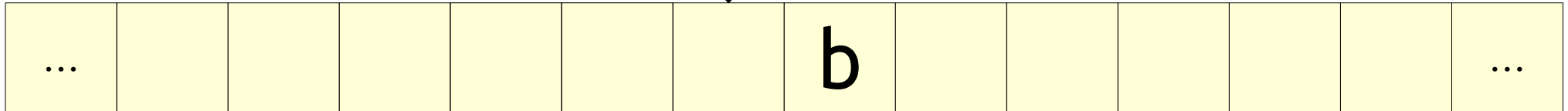
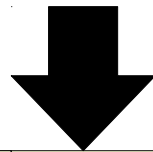
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

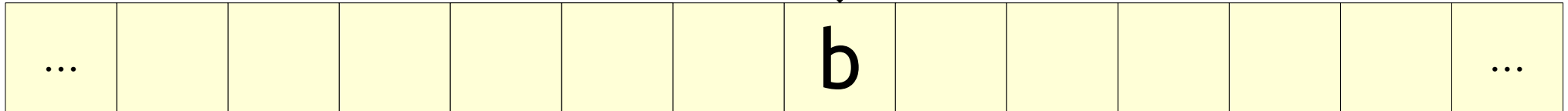
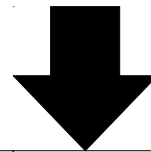
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

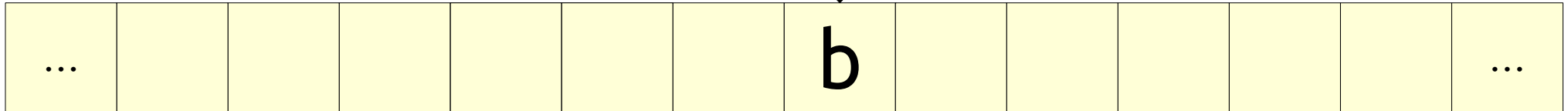
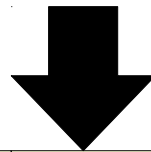
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

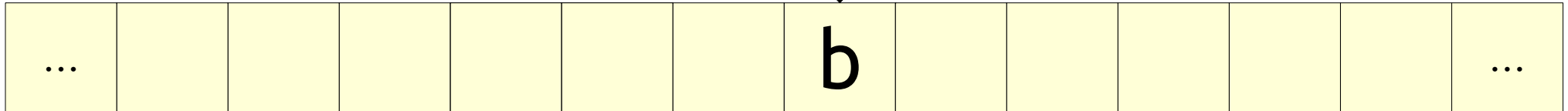
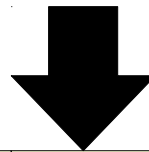
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

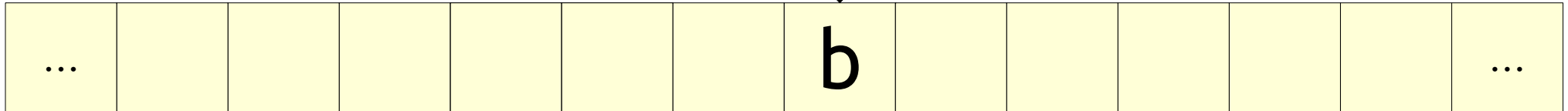
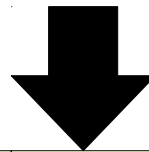
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

```
If Blank Return True  
If 'b' Return False  
Write Blank
```

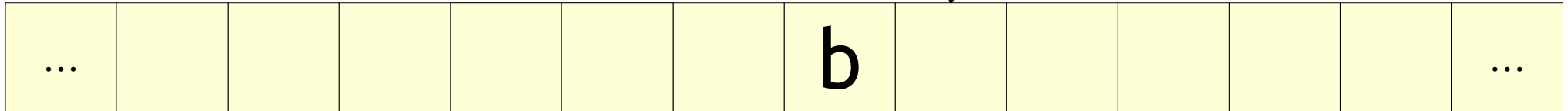
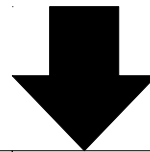
ZipRight:

Move Right

```
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

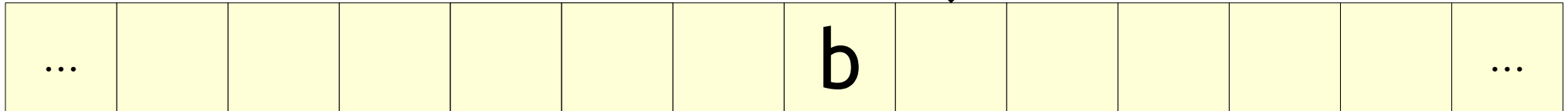
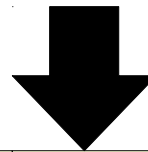
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Start:

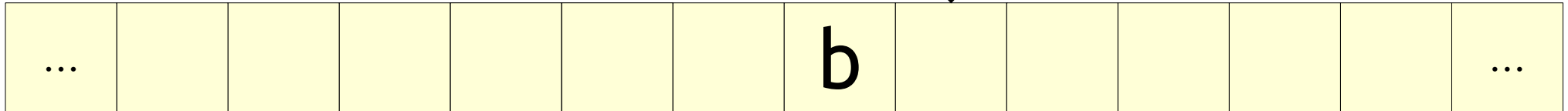
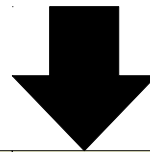
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

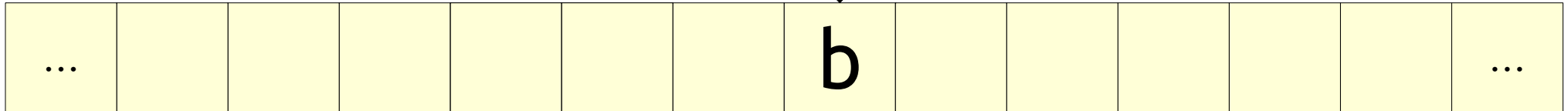
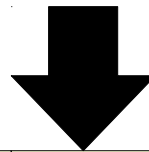
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

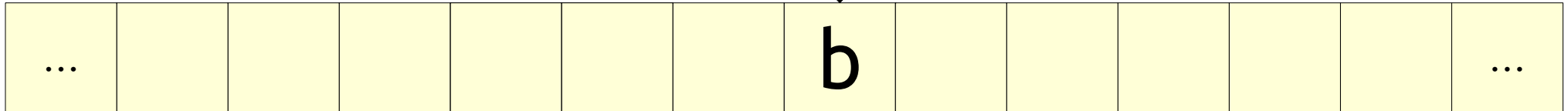
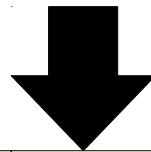
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

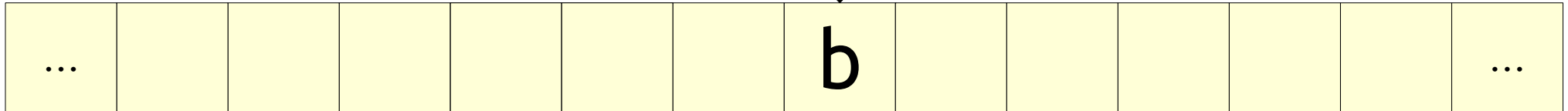
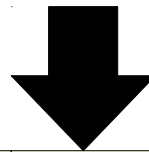
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

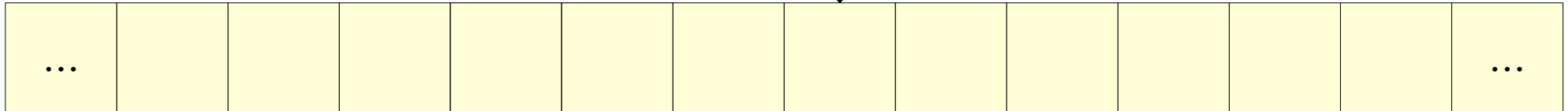
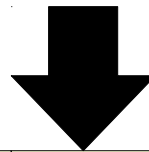
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

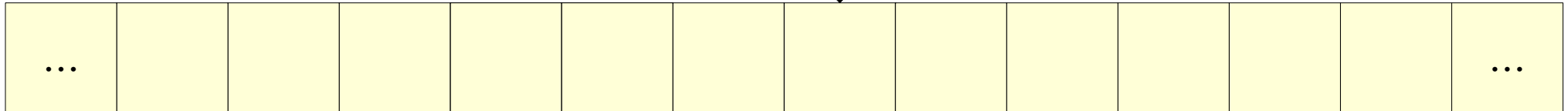
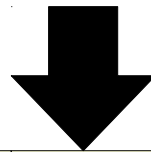
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

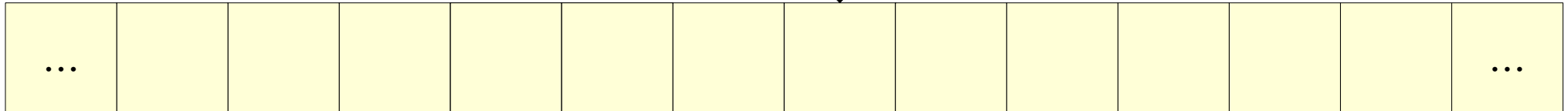
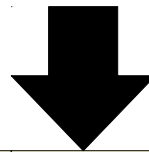
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

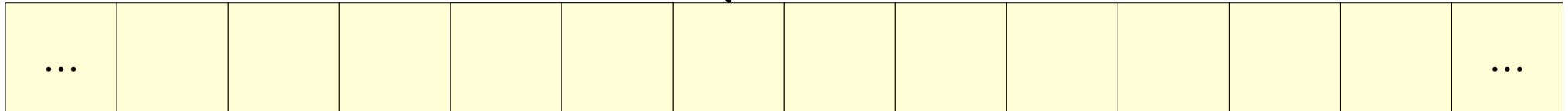
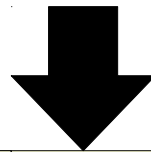
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

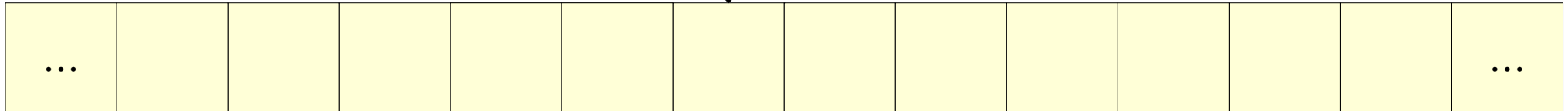
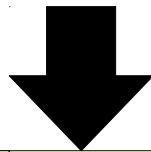
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

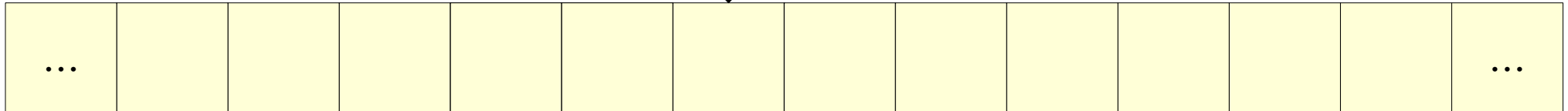
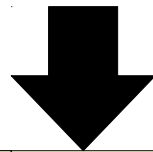
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

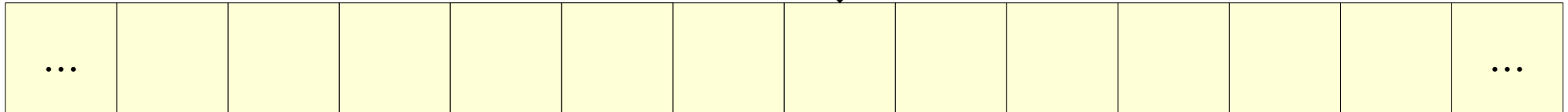
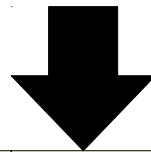
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

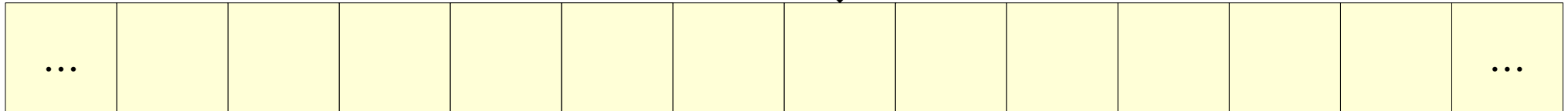
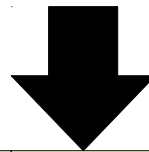
```
If Blank Return True  
If 'b' Return False  
Write Blank
```

ZipRight:

```
Move Right  
If Not Blank Goto ZipRight  
Move Left  
If Not 'b' Return False  
Write Blank
```

ZipLeft:

```
Move Left  
If Not Blank Goto ZipLeft  
Move Right  
Goto Start
```



Start:

If Blank Return True

If 'b' Return False

Write Blank

ZipRight:

Move Right

If Not Blank Goto ZipRight

Move Left

If Not 'b' Return False

Write Blank

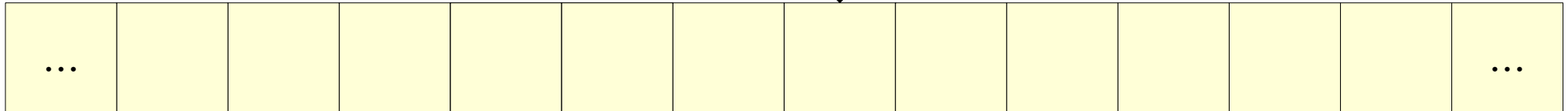
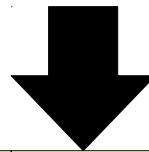
ZipLeft:

Move Left

If Not Blank Goto ZipLeft

Move Right

Goto Start



Let's take a quick break!

Time-Out for Announcements!

Problem Set

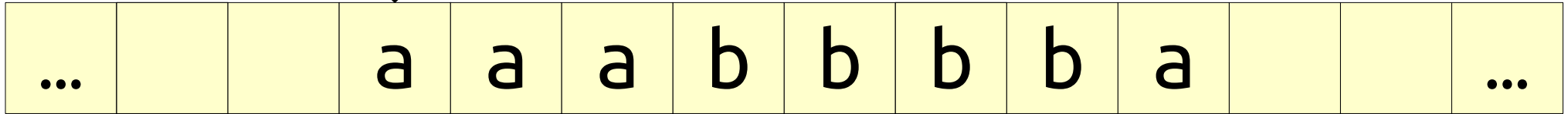
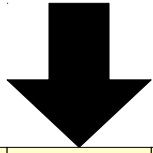
- Problem Set 5 solutions are up on the course website – please take a look at them as soon as possible.
- TAs are working hard on grading your assignments. We're aiming to have those returned to you by Thursday morning.

Back to CS103!

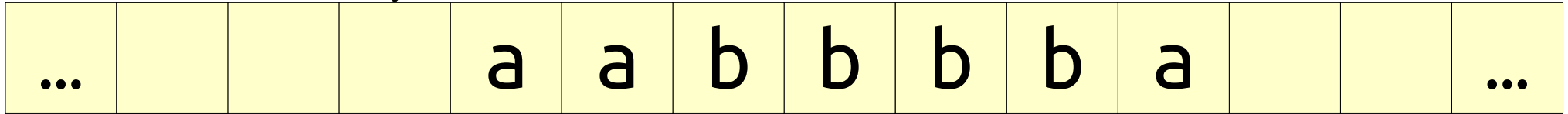
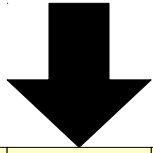
Our Next Challenge

- Let's now take aim at this more general language:
$$\{ w \in \{a, b\}^* \mid w \text{ has an equal number of } a\text{'s and } b\text{'s} \}$$
- This language is not regular (do you see why?)
- Let's see how to design a TM for it.

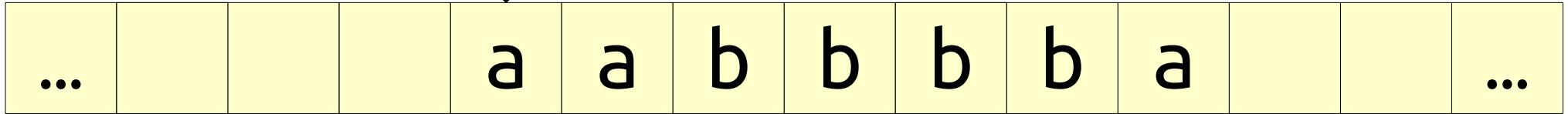
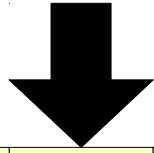
A Caveat



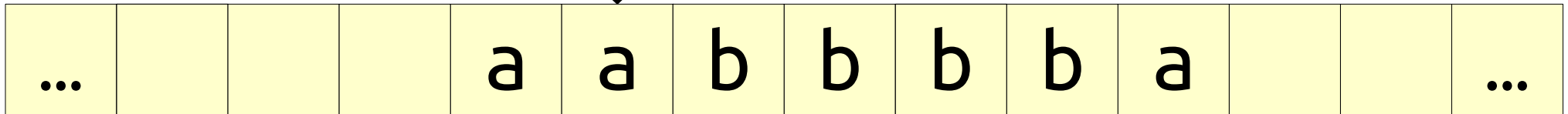
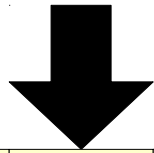
A Caveat



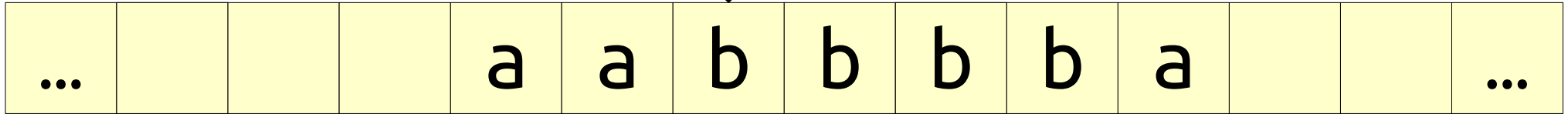
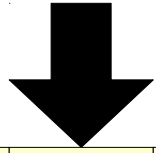
A Caveat



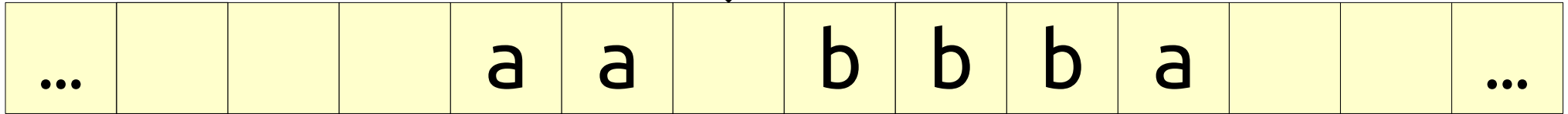
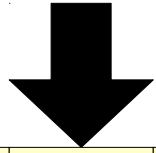
A Caveat



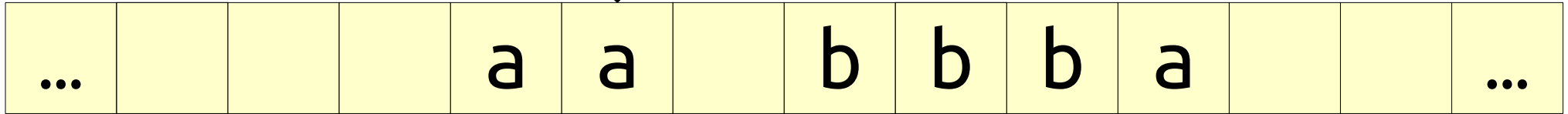
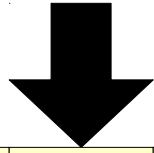
A Caveat



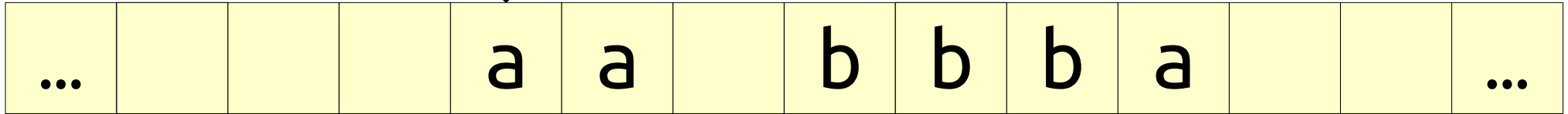
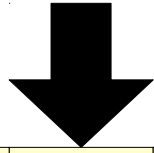
A Caveat



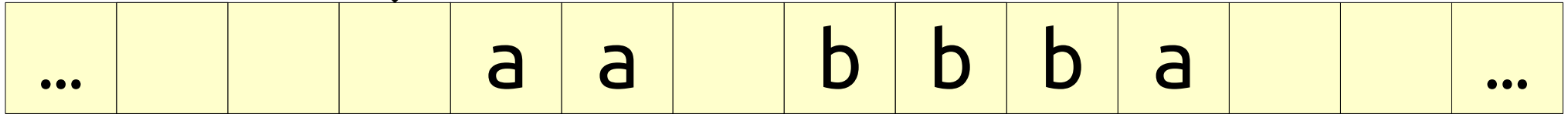
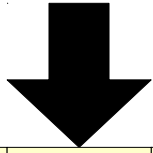
A Caveat



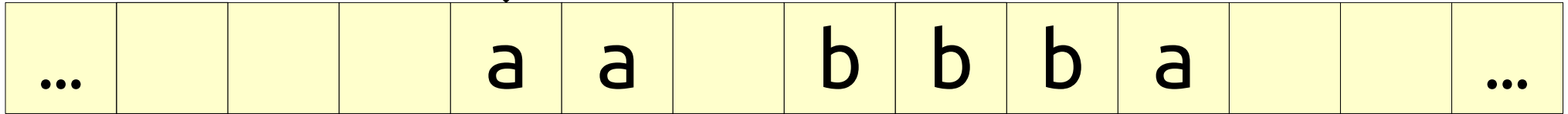
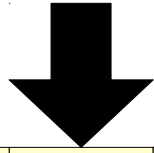
A Caveat



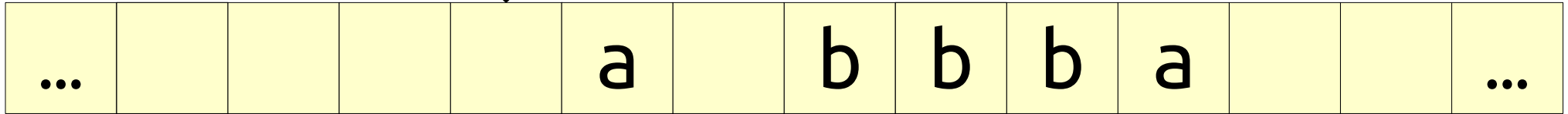
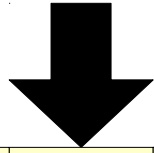
A Caveat



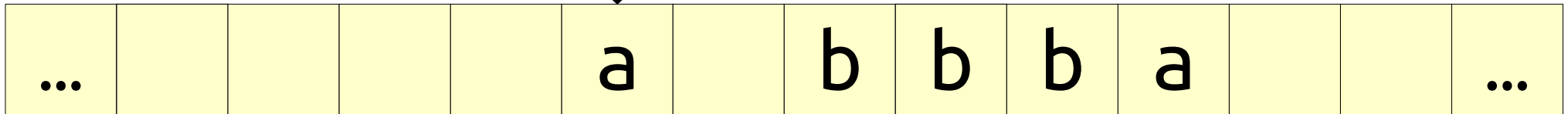
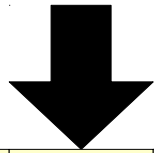
A Caveat



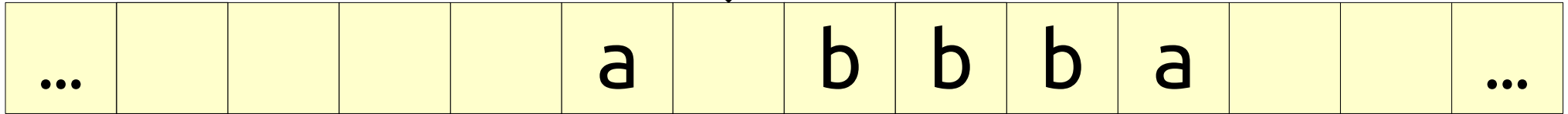
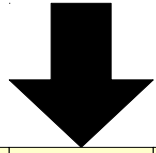
A Caveat



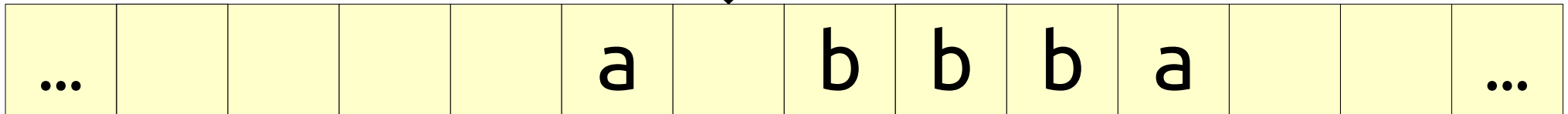
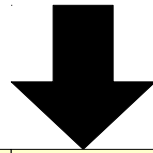
A Caveat



A Caveat

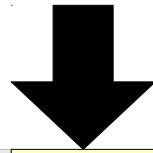


A Caveat



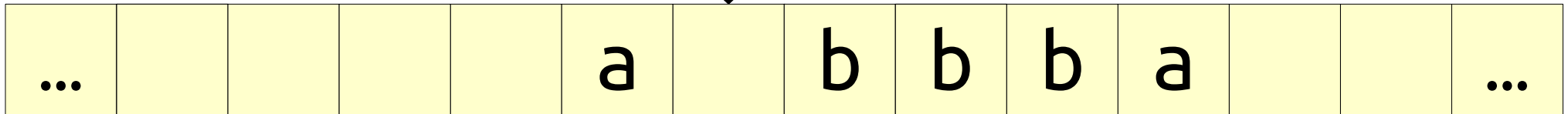
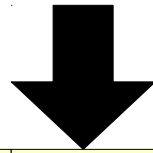
How do we know that this blank isn't one of the infinitely many blanks after our input string?

A Caveat



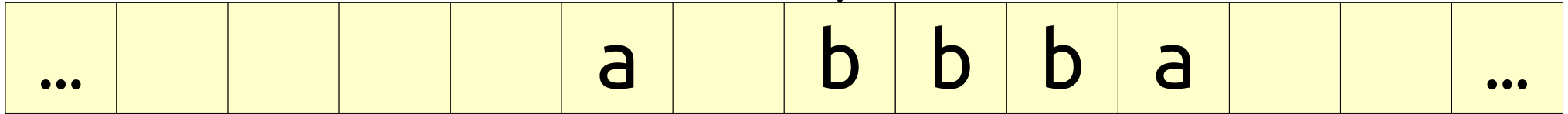
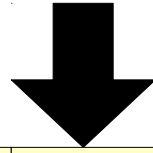
How do we know that
this blank isn't one of
the infinitely many
blanks after our input
string?

A Caveat

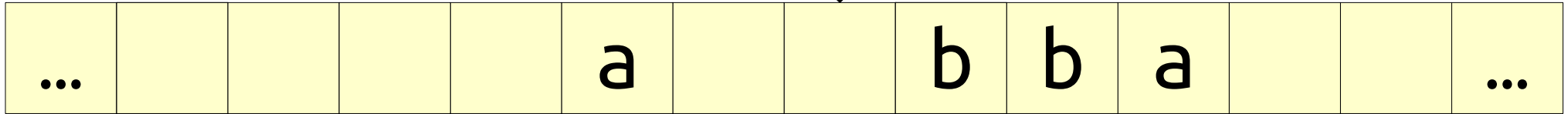
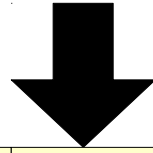


How do we know that
this blank isn't one of
the infinitely many
blanks after our input
string?

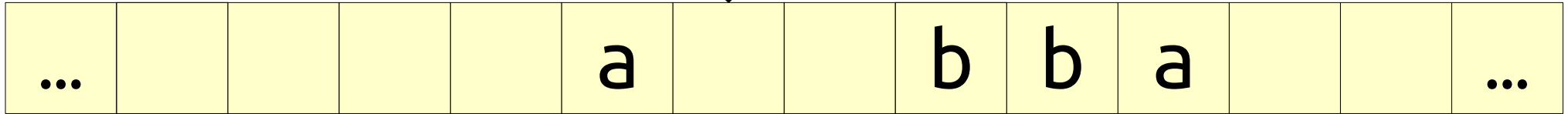
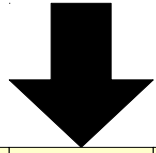
A Caveat



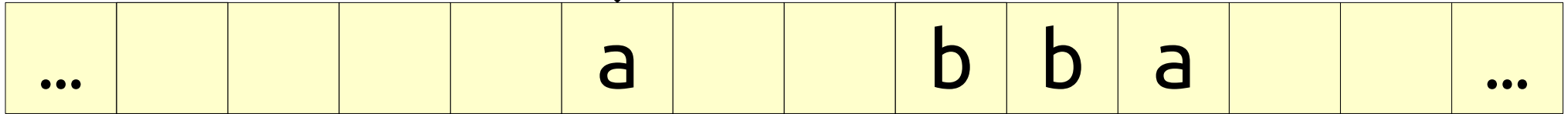
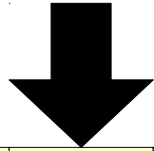
A Caveat



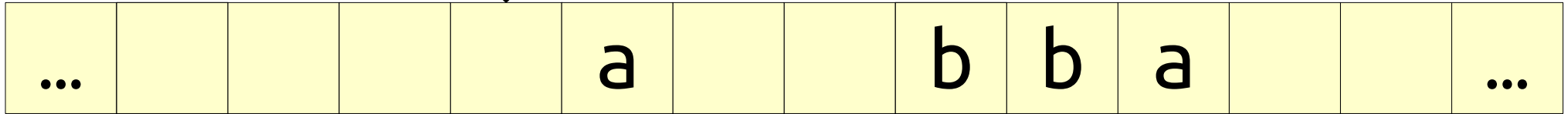
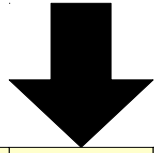
A Caveat



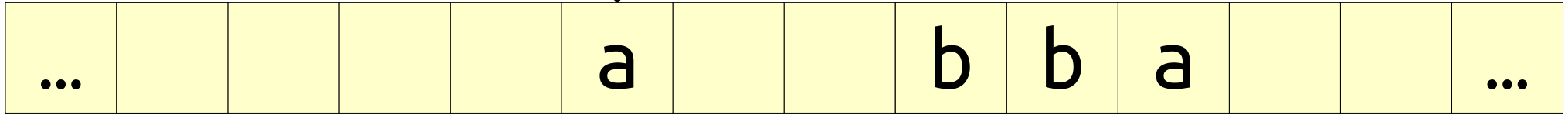
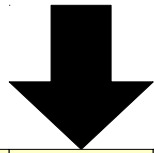
A Caveat



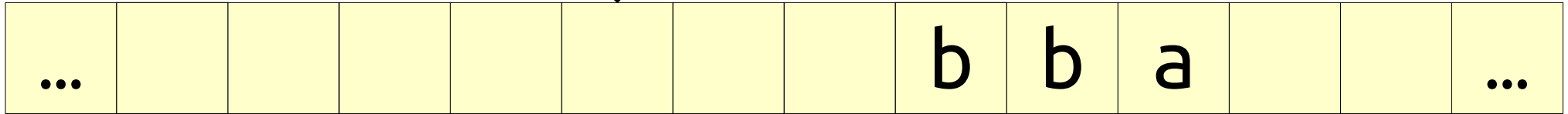
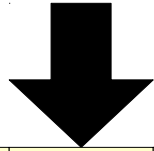
A Caveat



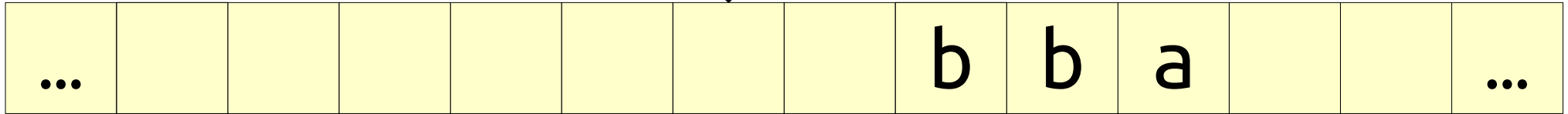
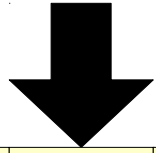
A Caveat



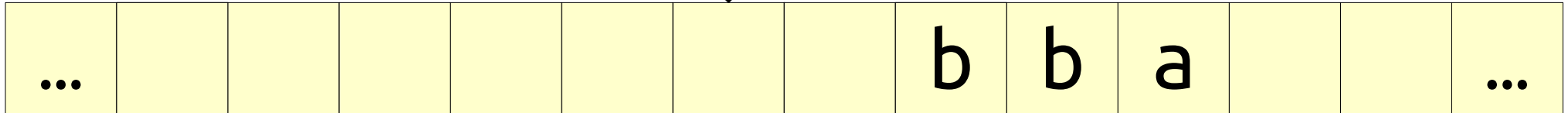
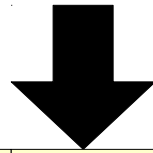
A Caveat



A Caveat

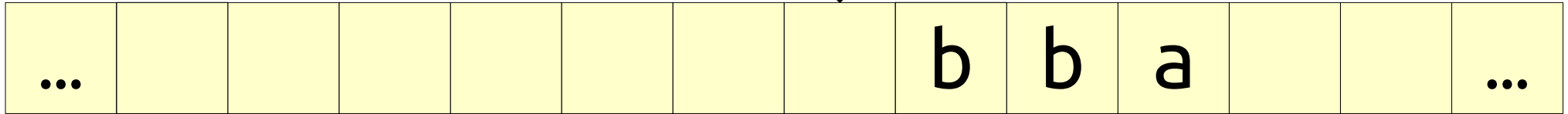
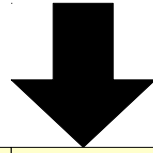


A Caveat

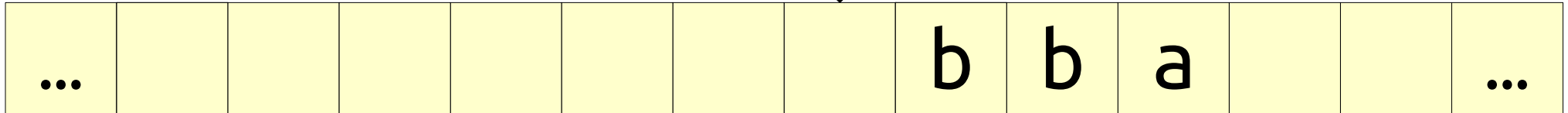
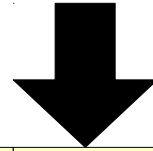


How do we know that
this blank isn't one of
the infinitely many
blanks after our input
string?

A Caveat

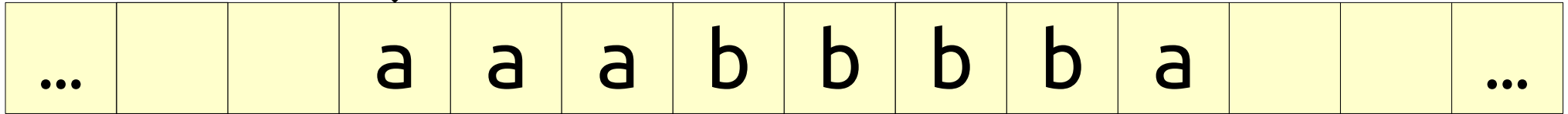
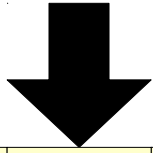


A Caveat

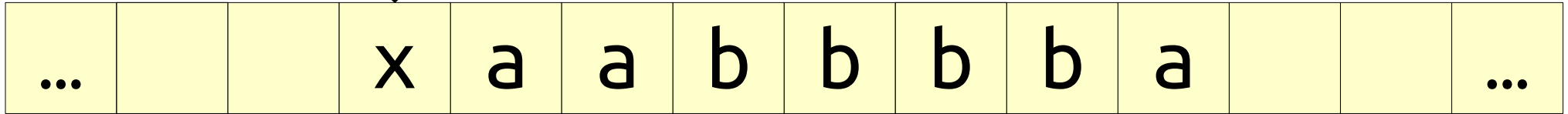
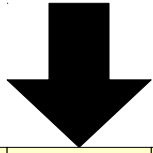


How do we know that this blank isn't one of the infinitely many blanks after our input string?

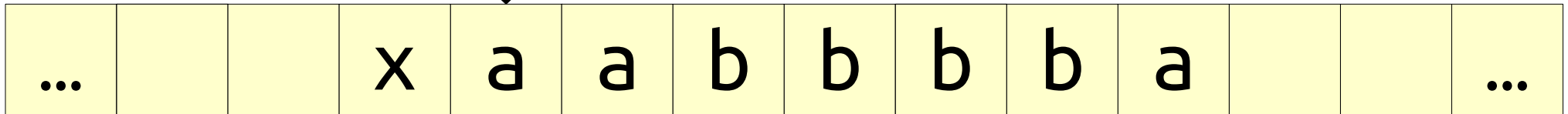
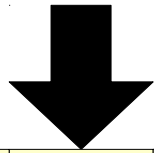
One Solution



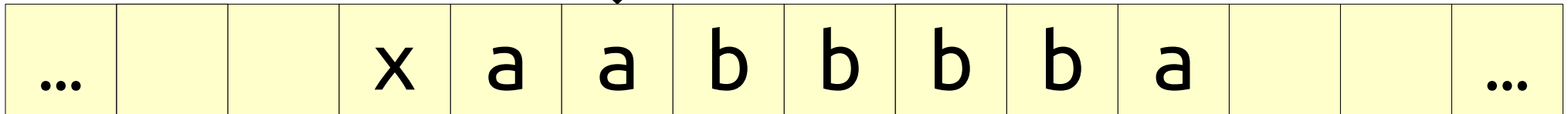
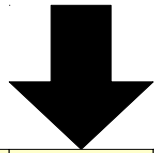
One Solution



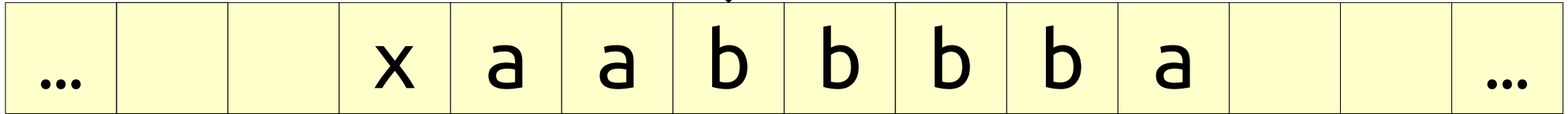
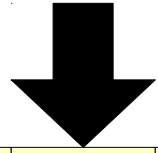
One Solution



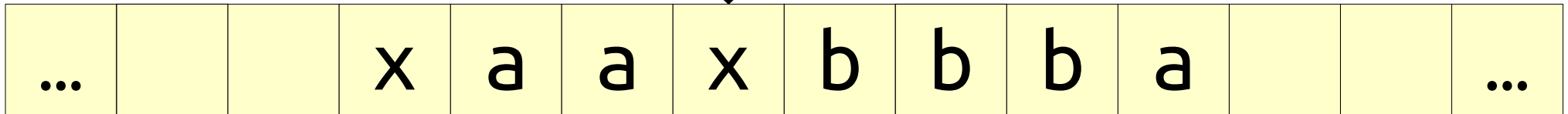
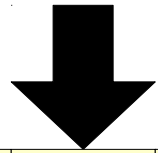
One Solution



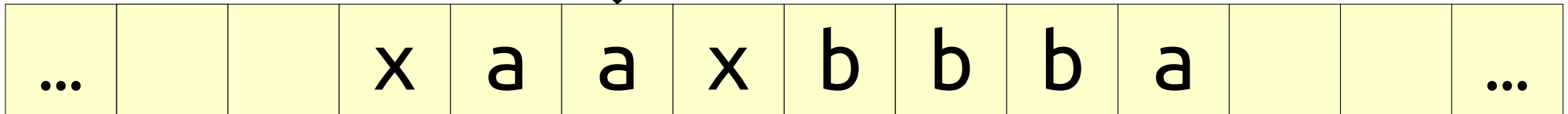
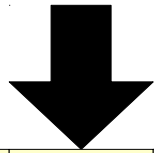
One Solution



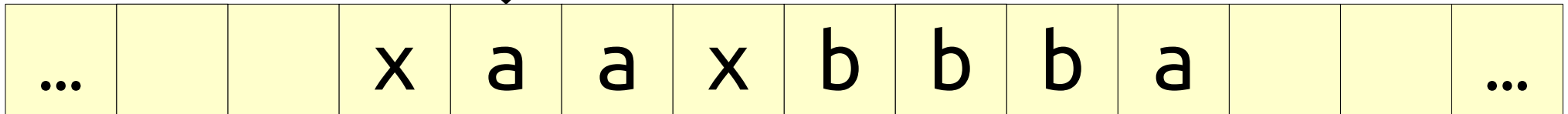
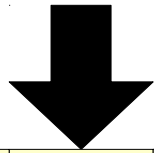
One Solution



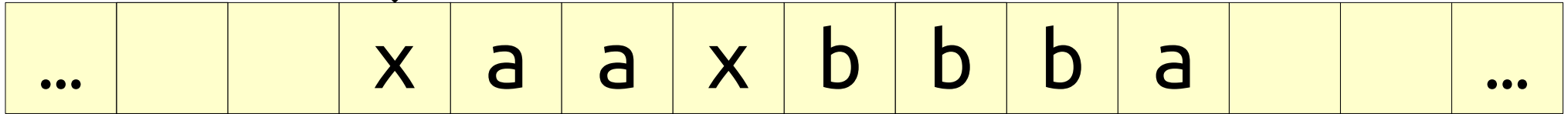
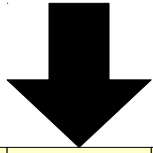
One Solution



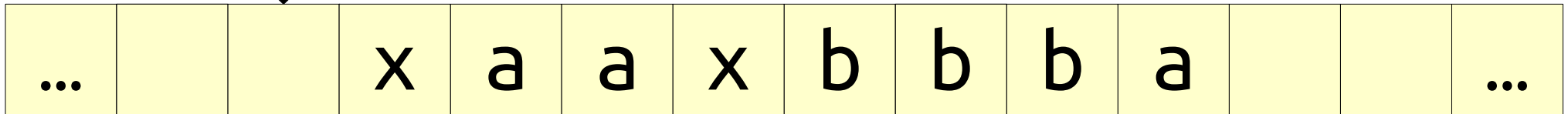
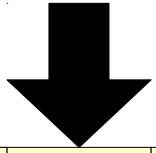
One Solution



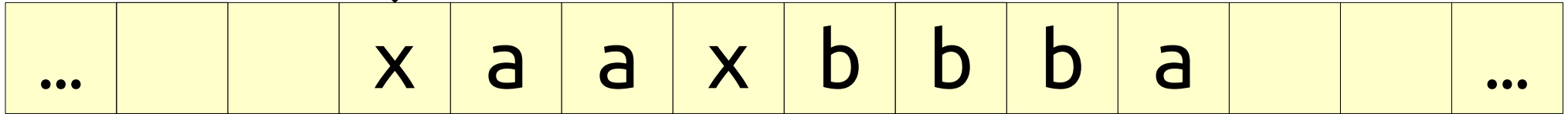
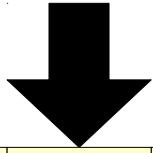
One Solution



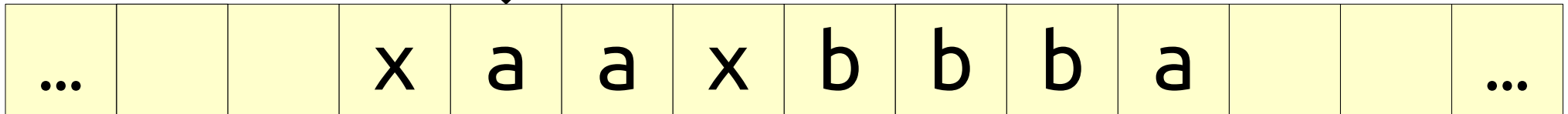
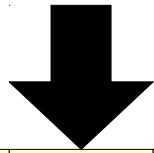
One Solution



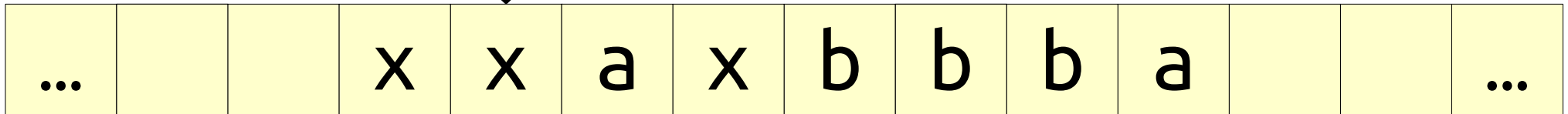
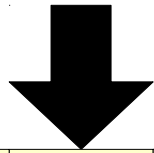
One Solution



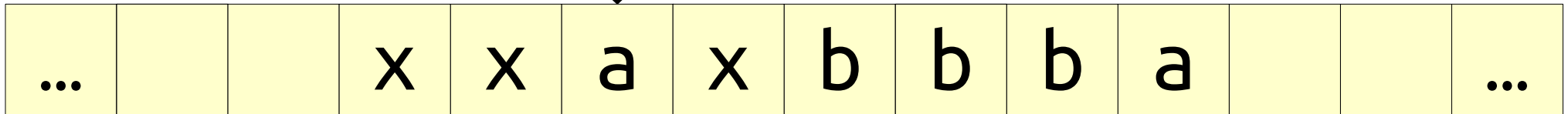
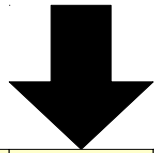
One Solution



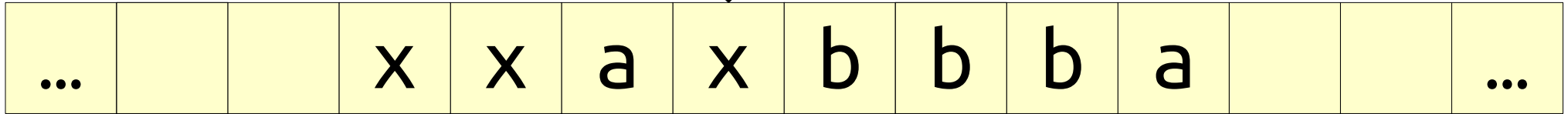
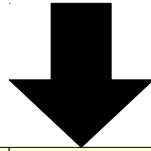
One Solution



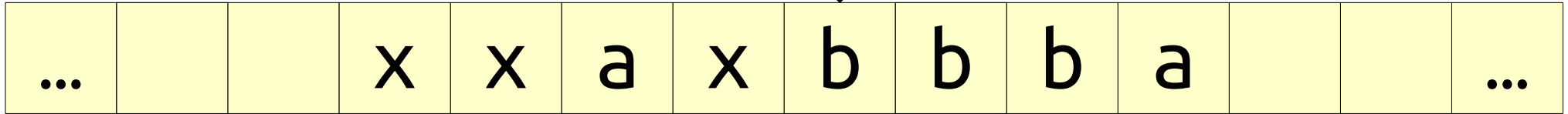
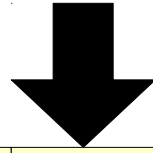
One Solution



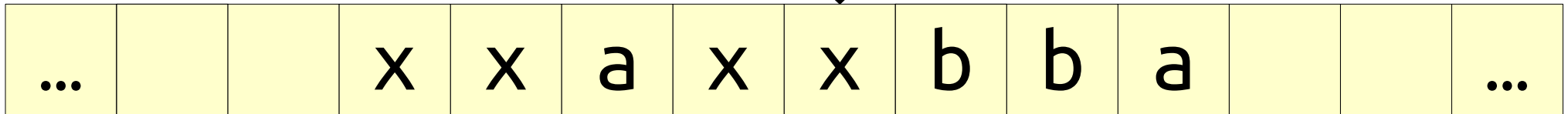
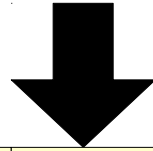
One Solution



One Solution



One Solution

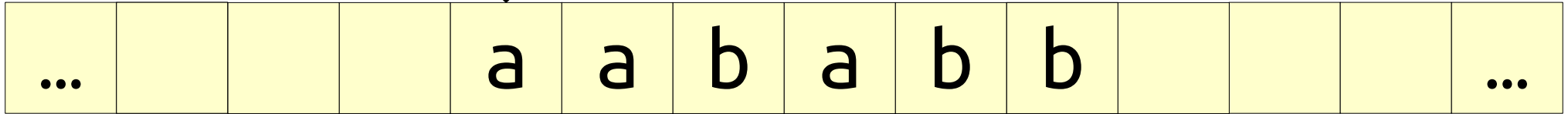
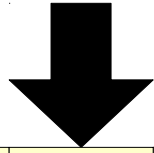


Let's take a look at this in action!

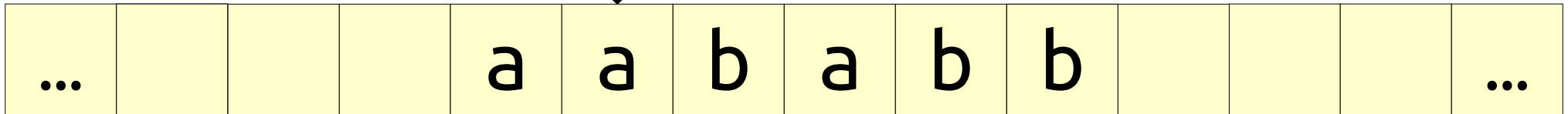
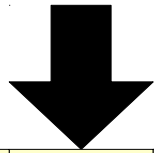
Another Idea

- We just built a TM for the language
 $\{ w \in \{a, b\}^* \mid w \text{ has the same number of } a\text{'s and } b\text{'s} \}$.
- An observation: this would be a *lot* easier to test for if all the **a**'s came before all the **b**'s.
 - In fact, that would turn this into checking if the string has the form $a^n b^n$, which we already know how to do!
- **Idea:** Could we sort the characters of our input string?

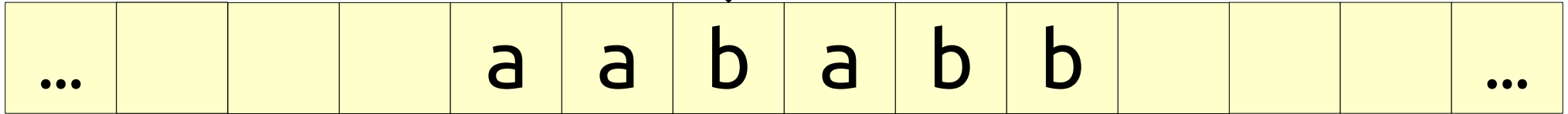
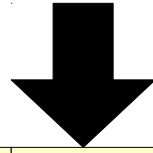
The Idea



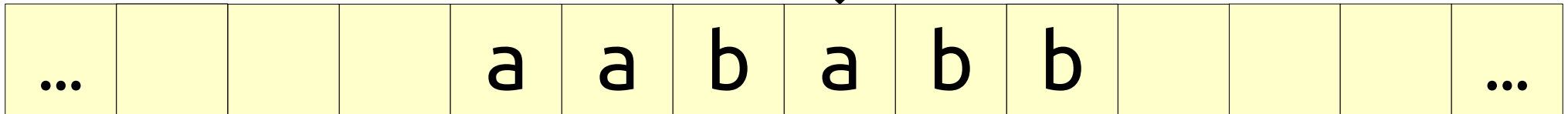
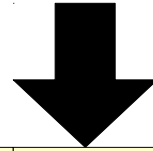
The Idea



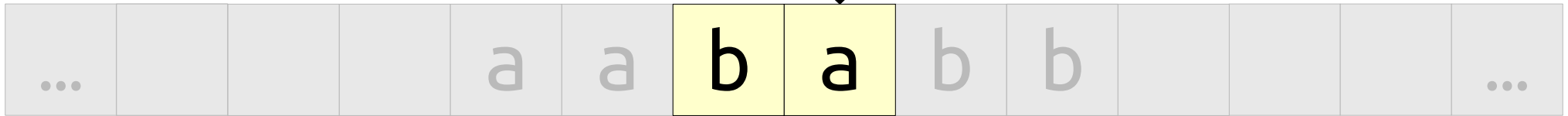
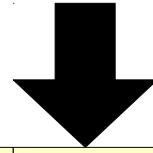
The Idea



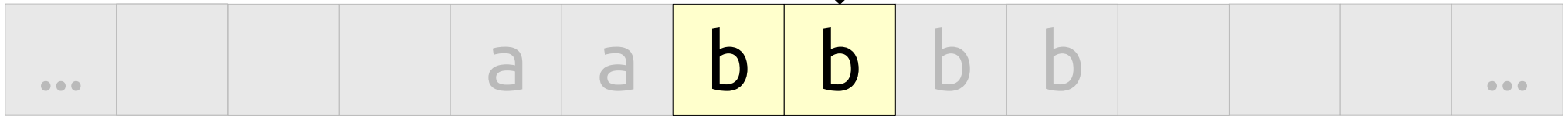
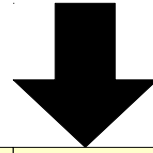
The Idea



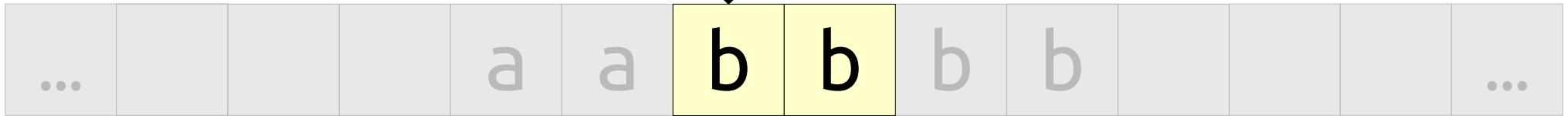
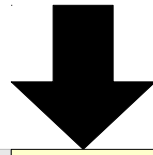
The Idea



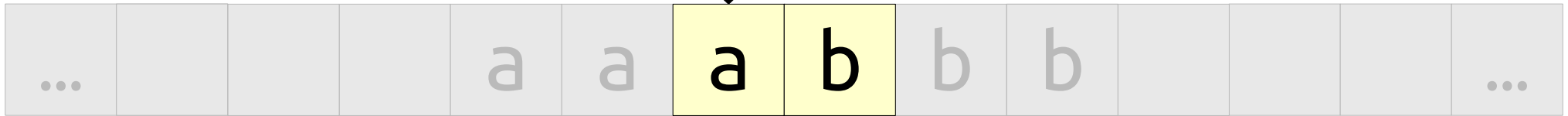
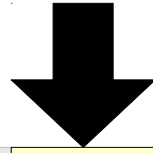
The Idea



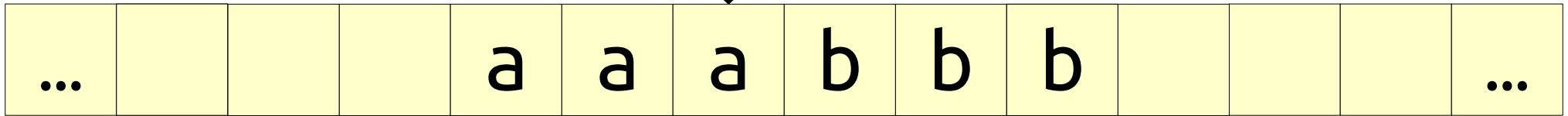
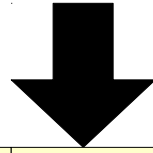
The Idea



The Idea



The Idea



Exploring This Idea

Cool TM Tricks 1: ***Fibonacci Numbers***

Fibonacci Numbers

...		a	a	a	a	a	a	a	a	a	a	a	a			...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	--	--	-----

...		x	y	a	a	a	a	a	a	a	a	a	a			...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	--	--	-----

...		y	y	x	a	a	a	a	a	a	a	a	a			...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	--	--	-----

...		x	x	x	y	y	a	a	a	a	a	a	a			...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	--	--	-----

...		y	y	y	y	y	x	x	x	a	a	a	a			...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	--	--	-----

...		x	x	x	x	x	x	x	x	y	y	y	y	y		...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	---	--	-----

$\{ a^n \mid n \text{ is a Fibonacci number} \}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Cool TM Tricks 2: *Decimal Fibonacci*

Decimal Fibonacci

...		1	3														...
-----	--	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	-----

...		a	a	a	a	a	a	a	a	a	a	a	a	a			...
-----	--	---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	-----

$\{ w \in \{0, 1, 2, \dots, 9\}^* \mid w, \text{ interpreted as a base-10 number, is a Fibonacci number. } \}$

Just how powerful are Turing machines?

Real and “Ideal” Computers

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

Theorem: Turing machines are equal in power to idealized computers. That is, any computation that can be done on a TM can be done on an idealized computer and vice-versa.

Key Idea: Two models of computation are equally powerful if they can simulate each other.

Simulating a TM

- The individual commands in a TM are simple and perform only basic operations:

Move Write Goto Return If

- The memory for a TM can be thought of as a string with some number keeping track of the current index.
- To simulate a TM, we need to
 - see which line of the program we're on,
 - determine what command it is, and
 - simulate that single command.
- **Claim:** This is reasonably straightforward to do on an idealized computer.
 - The “core” logic for the TM simulator is under fifty lines of code, including comments.

Simulating a TM

- Because a computer can simulate each individual TM instruction, a computer can do anything a TM can do.
- ***Key Idea:*** Even the most complicated TM is made out of individual instructions, and if we can simulate those instructions, we can simulate an arbitrarily complicated TM.

Simulating a Computer

- Programming languages provide a set of simple constructs.
 - Think things like variables, arrays, loops, functions, classes, etc.
- You, the programmer, then combine these basic constructs together to assemble larger programs.
- ***Key Idea:*** If a TM is powerful enough to simulate each of these individual pieces, it's powerful enough to simulate anything a real computer can do.

What We've Seen

- We've seen TMs use loops to solve problems.
 - Our $\{ a^n b^n \mid n \in \mathbb{N} \}$ TM repeatedly pulls off the first and last character from the string.
 - Our sorting TM repeatedly finds **ba** and replaces it with **ab**.
- In some sense, the existence of Goto and labels means that TMs have loops.
- Hopefully, it's not too much of a stretch to think that TMs can do while loops, for loops, etc.

What We've Seen

- We've seen TMs that perform basic arithmetic.
 - We can check if two numbers are equal.
 - We can check if a number is a Fibonacci number.
- Hopefully, it's not too much of a stretch to believe we could also do addition and subtraction, compute powers of numbers, do ceilings and floors, etc.

What We've Seen

- We've seen TMs that maintain variables.
 - You can think of our TM for $\{ a^n b^n \mid n \in \mathbb{N} \}$ as storing two variables - one that counts a number of **a**'s, and one that counts a number of **b**'s.
 - Our TM for Fibonacci numbers kinda sorta ish tracks the last two Fibonacci numbers, plus the length of the input string.
- It's a bit larger of a jump to make, but hopefully you're comfortable with the idea that TMs, in principle, can maintain variables.

What We've Seen

- We've seen TMs with helper functions.
 - We saw how to check for equal numbers of **a**'s and **b**'s by first sorting the string, then checking if the string has the form $\mathbf{a}^n\mathbf{b}^n$.
 - We can check if a decimal number is a Fibonacci number by converting it to unary, then running our unary Fibonacci checker.
- Hopefully you're comfortable with the idea that a TM could have multiple "helper functions" that work together to solve some larger problem.

What Else Can TMs Do?

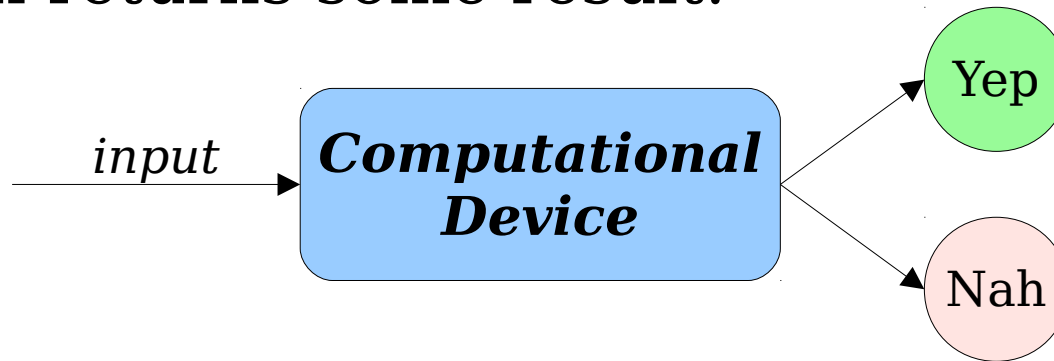
- Maintain strings and arrays.
 - Store their elements separated with some special separator character.
- Support pointers.
 - Maintain an array of what's in memory, where each item is tagged with its "memory address."
- Support function call and return.
 - It's hard, but you can do this if you can do helper functions and variables.

A CS107 Perspective

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

A Leap of Faith

- **Claim:** A TM is powerful enough to simulate any computer program that gets an input, processes that input, then returns some result.



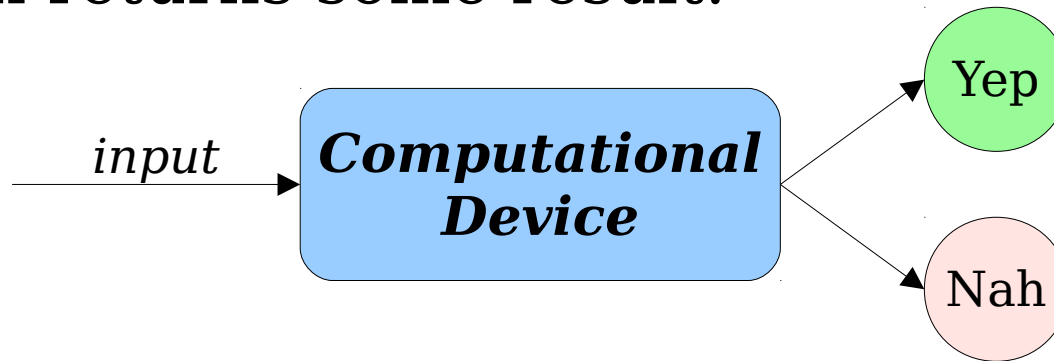
- The resulting TM might be colossal, or really slow, or both, but it would still faithfully simulate the computer.

This is quite a bold claim to make. What's an example of something your computer can do that you're convinced a Turing machine couldn't do?

Respond at pollev.com/cs103

A Leap of Faith

- **Claim:** A TM is powerful enough to simulate any computer program that gets an input, processes that input, then returns some result.



- The resulting TM might be colossal, or really slow, or both, but it would still faithfully simulate the computer.
- We're going to take this as an article of faith in CS103. If you curious for more details, come talk to me after class.

Just how powerful *are* Turing machines?

Effective Computation

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

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

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

- Ryan Williams

**Regular
Languages**

The diagram consists of a large orange oval containing a smaller yellow circle. The yellow circle is labeled 'Regular Languages'. The orange oval is labeled 'Problems Solvable by Any Feasible Computing Machine'. The red background is labeled 'All Languages'.

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

All Languages

A Venn diagram illustrating the relationship between different classes of languages. It features a large orange oval representing 'All Languages' which contains a smaller yellow circle representing 'Regular Languages'. To the right of the yellow circle, within the orange oval, is the text 'Problems solvable by Turing Machines'. The background of the entire image is red.

**Regular
Languages**

**Problems
solvable by
Turing
Machines**

All Languages


TMs and Computation

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

Decidability and Recognizability


What problems can we solve with a computer?

What kind
of
computer?



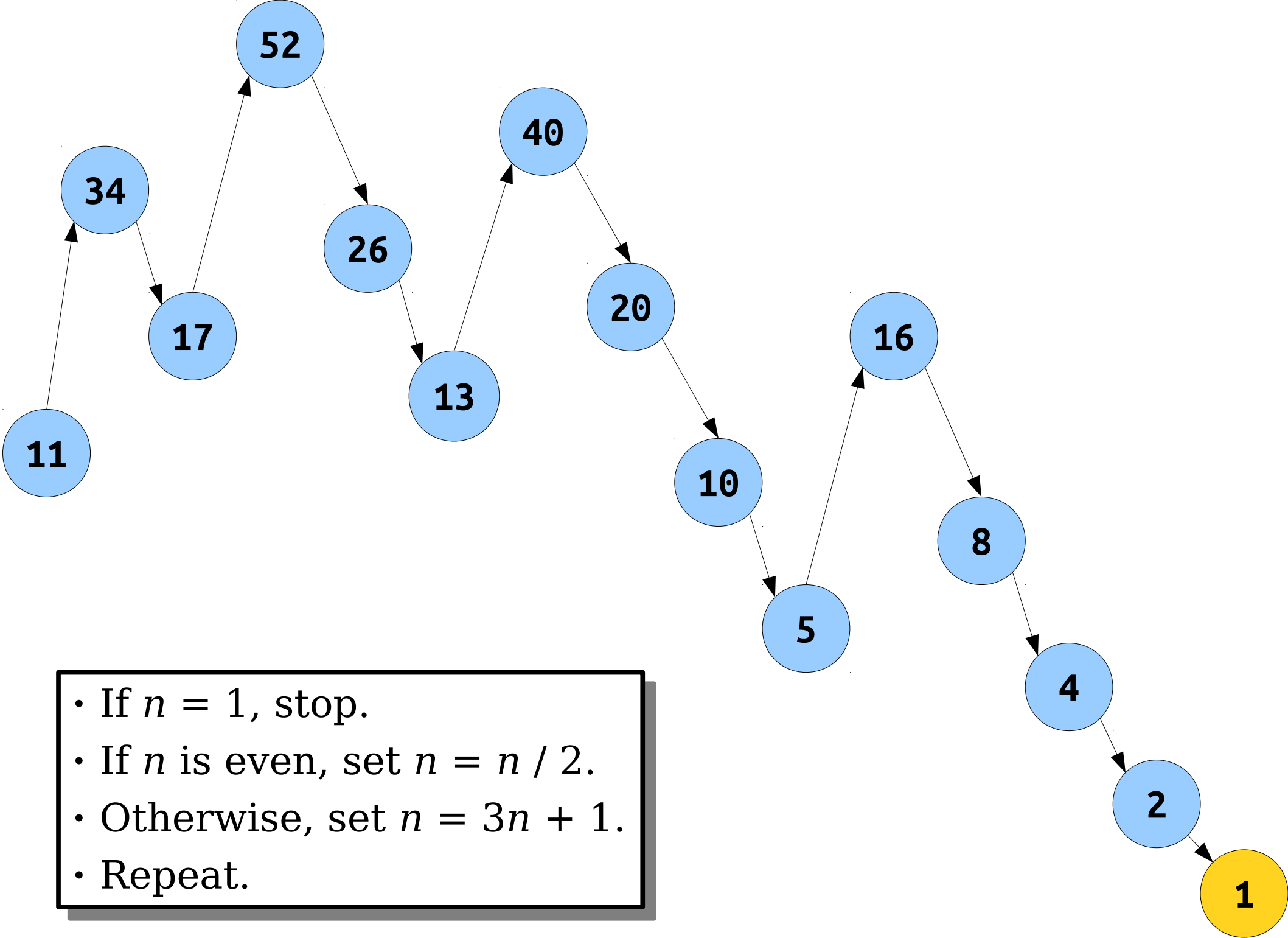
What problems can we solve with a computer?

What does it
mean to
"solve" a
problem?



The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
 - If $n = 1$, you are done.
 - If n is even, set $n = n / 2$.
 - Otherwise, set $n = 3n + 1$.
 - Repeat.
- **Question:** Given a natural number $n > 0$, does this process terminate?



- If $n = 1$, stop.
- If n is even, set $n = n / 2$.
- Otherwise, set $n = 3n + 1$.
- Repeat.

The Hailstone Sequence

- Consider the following procedure, starting with some $n \in \mathbb{N}$, where $n > 0$:
 - If $n = 1$, you are done.
 - If n is even, set $n = n / 2$.
 - Otherwise, set $n = 3n + 1$.
 - Repeat.
- Does the Hailstone Sequence terminate for...
 - $n = 5$?
 - $n = 20$?
 - $n = 7$?
 - $n = 27$?

The Hailstone Sequence

- Let $\Sigma = \{\mathbf{a}\}$ and consider the language
$$L = \{ \mathbf{a}^n \mid n > 0 \text{ and the hailstone sequence terminates for } n \}.$$
- Could we build a TM for L ?

The Hailstone Turing Machine

- We can build a TM that works as follows:
 - If the input is ε , reject.
 - While the string is not **a**:
 - If the input has even length, halve the length of the string.
 - If the input has odd length, triple the length of the string and append a **a**.
 - Accept.

Does this Turing machine accept all
nonempty strings?

The Collatz Conjecture

- It is *unknown* whether this process will terminate for all natural numbers.
- In other words, no one knows whether the TM described in the previous slides will always stop running!
- The conjecture (unproven claim) that the hailstone sequence always terminates is called the ***Collatz Conjecture***.
- This problem has eluded a solution for a long time. The influential mathematician Paul Erdős is reported to have said “Mathematics may not be ready for such problems.”

An Important Observation

- Unlike finite automata, which automatically halt after all the input is read, TMs keep running until they explicitly return true or return false.
- As a result, it's possible for a TM to run forever without accepting or rejecting.
- This leads to several important questions:
 - How do we formally define what it means to build a TM for a language?
 - What implications does this have about problem-solving?

Next Time

- ***Emergent Properties***
 - Larger phenomena made of smaller parts.
- ***Universal Machines***
 - A single, “most powerful” computer.
- ***Self-Reference***
 - Programs that ask questions about themselves.