Computability: Automata & Turing machines

alberto.suarez@uam.es

Computer Science Department

Universidad Autónoma de Madrid

Some material from:

- Kenneth H. Rosen, ”Discrete Mathematics and its applicaions”, Fifth edition ,McGraw-Hill, Boston, 2003

Finite automaton: Regular grammar

Language symbols

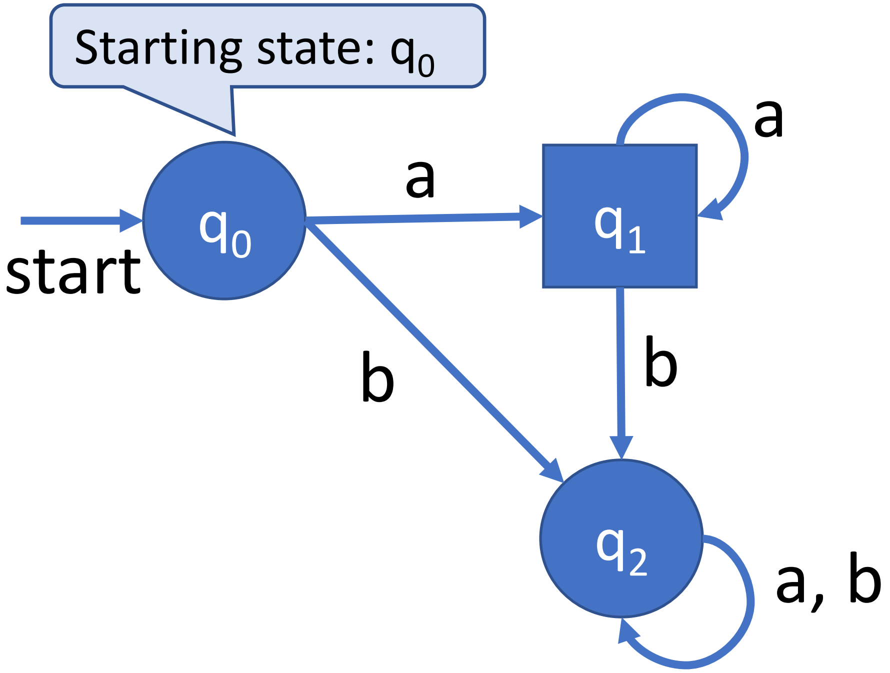
Build a finite automaton that accepts the regular language $L = \{a^n; n > 0;\ \Sigma = \{a, b\}\}$

Language alphabet

Accepting state: whole word processed + q_1

Reject state: whole word processed + q_0 / q_2

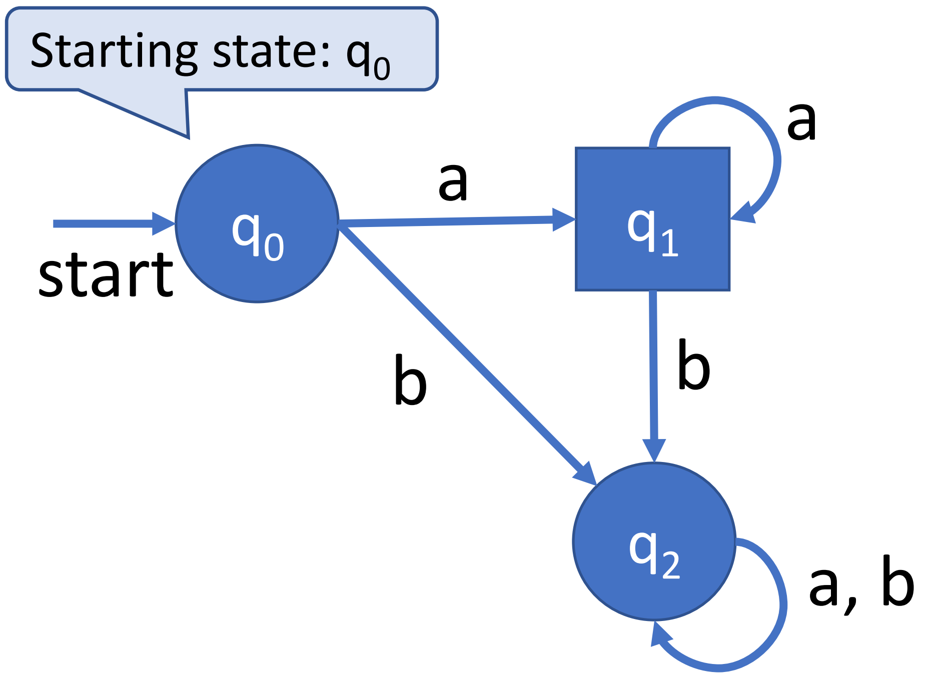
Initial state	Symbol	Final state
q_0	a	q_1
q_0	b	q_2
q_1	a	q_1
q_1	b	q_2
q_2	a	q_2
q_2	b	q_2



Finite automaton: Regular grammar

Build a finite automaton that accepts the regular language $L = \{a^n; n > 0;\ \Sigma = \{a, b\}\}$

Accepting state: whole word processed + q_1



a	a	a	a		
q_0					

b	a	a	a		
q_0					

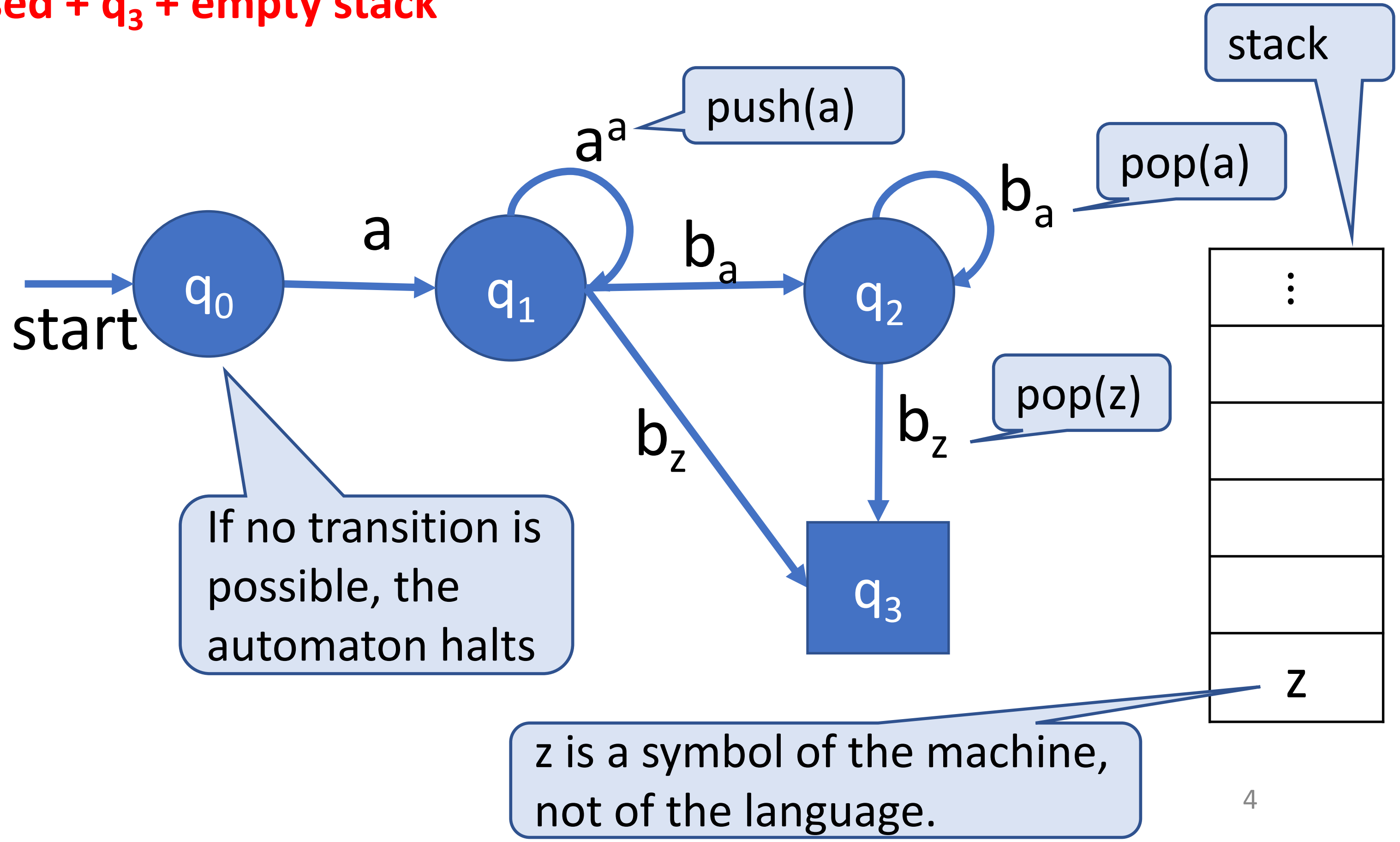
a	a	b	a		
q_0					

Pushdown automaton: Context-free grammar

Build a finite automaton that accepts the regular language $L = \{a^n b^n; n > 0;\ \Sigma = \{a, b\}\}$

Accepting state: whole word processed + q_3 + empty stack

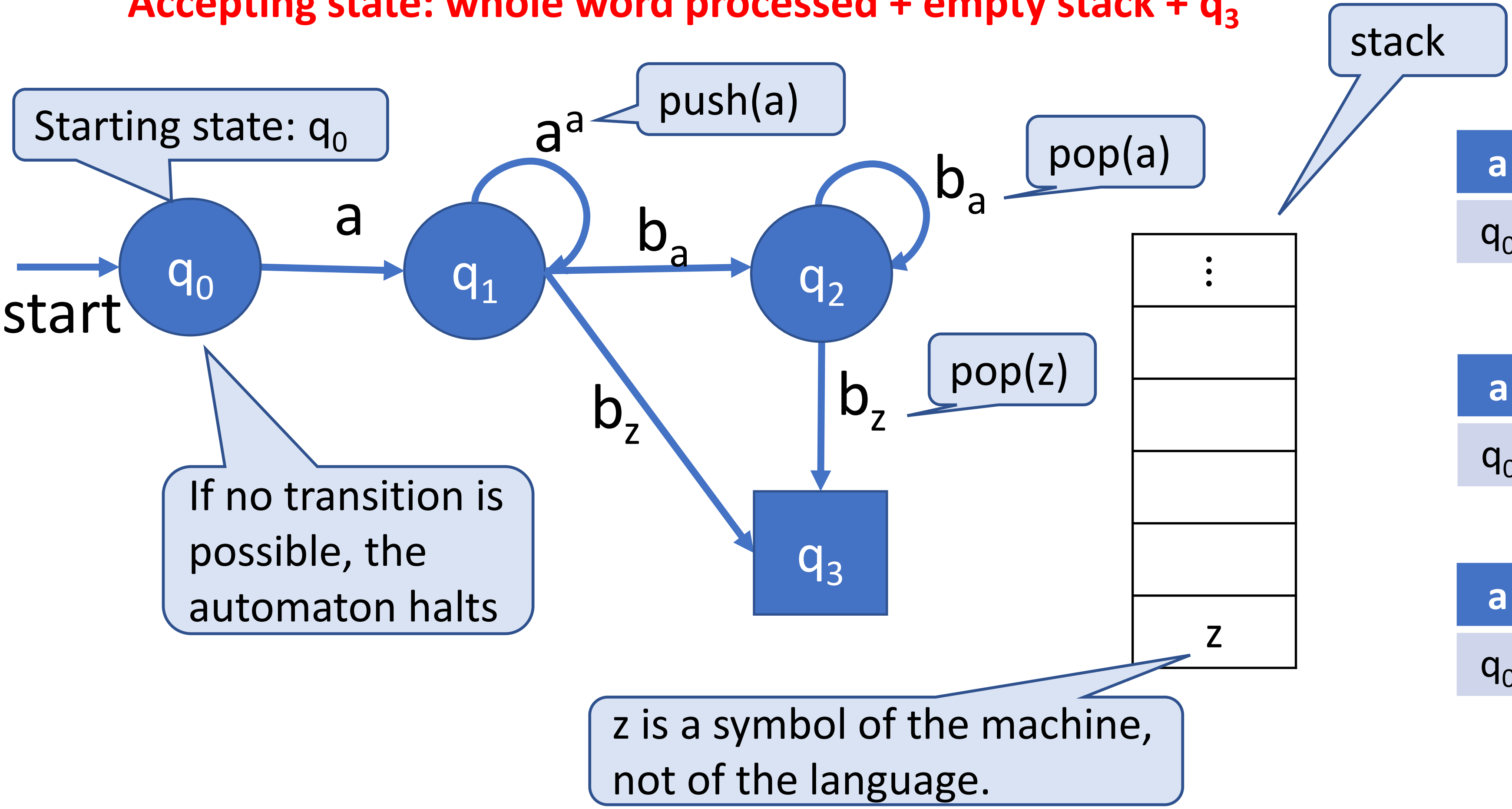
Initial state	Symbol	Top stack	Final state	Stack oper.
q_0	a	z	q_1	-
q_1	a	z	q_1	push(a)
q_1	b	a	q_2	pop(a)
q_1	b	z	q_3	pop(z)
q_2	b	a	q_2	pop(a)
q_2	b	z	q_3	pop(z)



Pushdown automaton: Context-free grammar

Build a finite automaton that accepts the regular language $L = \{a^n b^n; n > 0;\ \Sigma = \{a, b\}\}$

Accepting state: whole word processed + empty stack + q_3



a	a	a	b	b	b	
q_0						

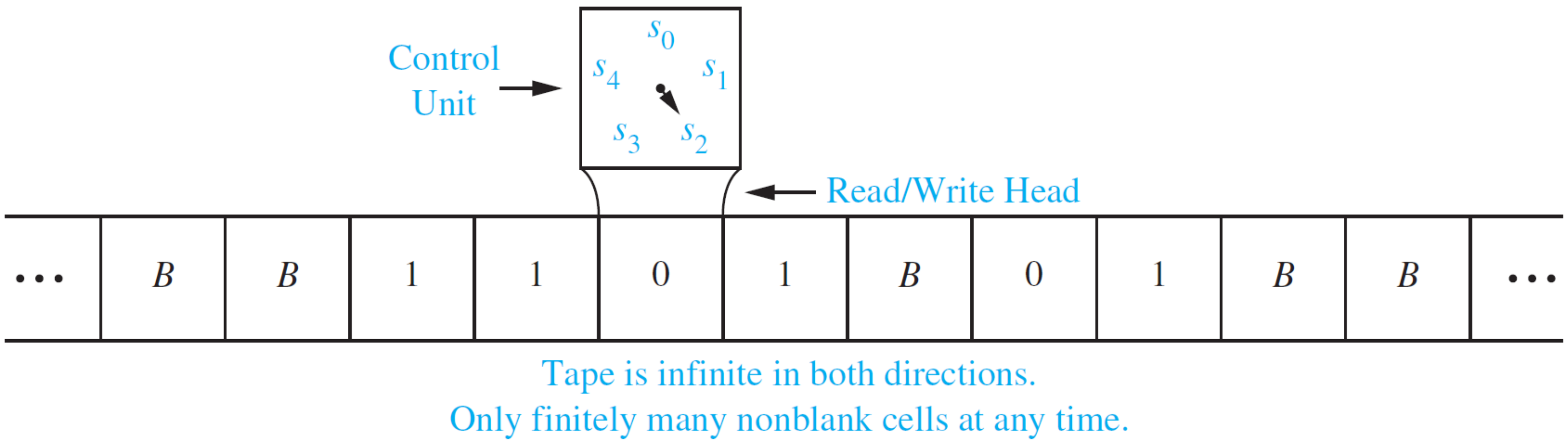
a	a	b	b	b	a	
q_0						

a	a	a	b	b	a	
q_0						

Turing machine

Computational device composed of

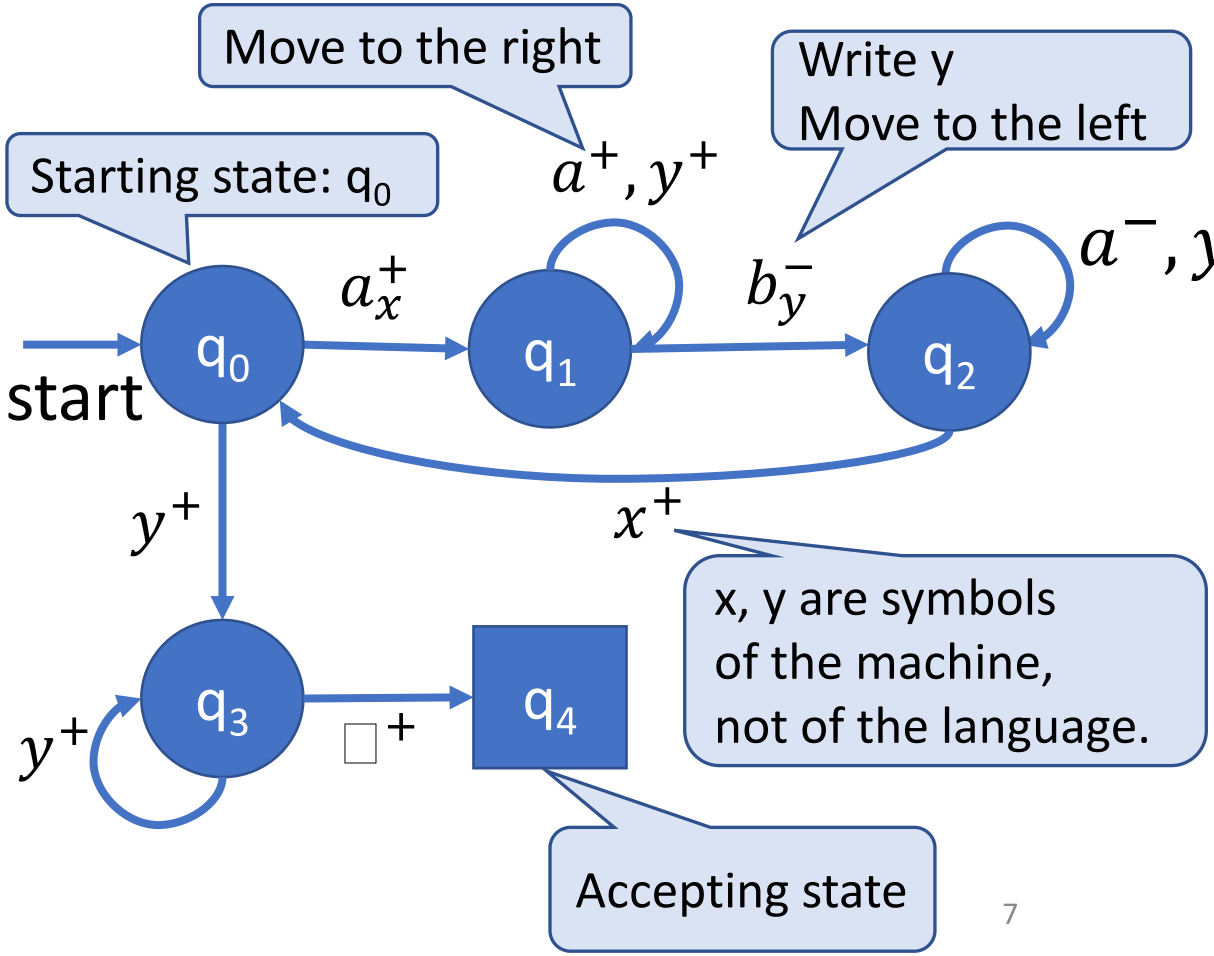
- Control unit (programmable)
- Read/Write head
- Tape (memory)



Turing machine as a language accepter

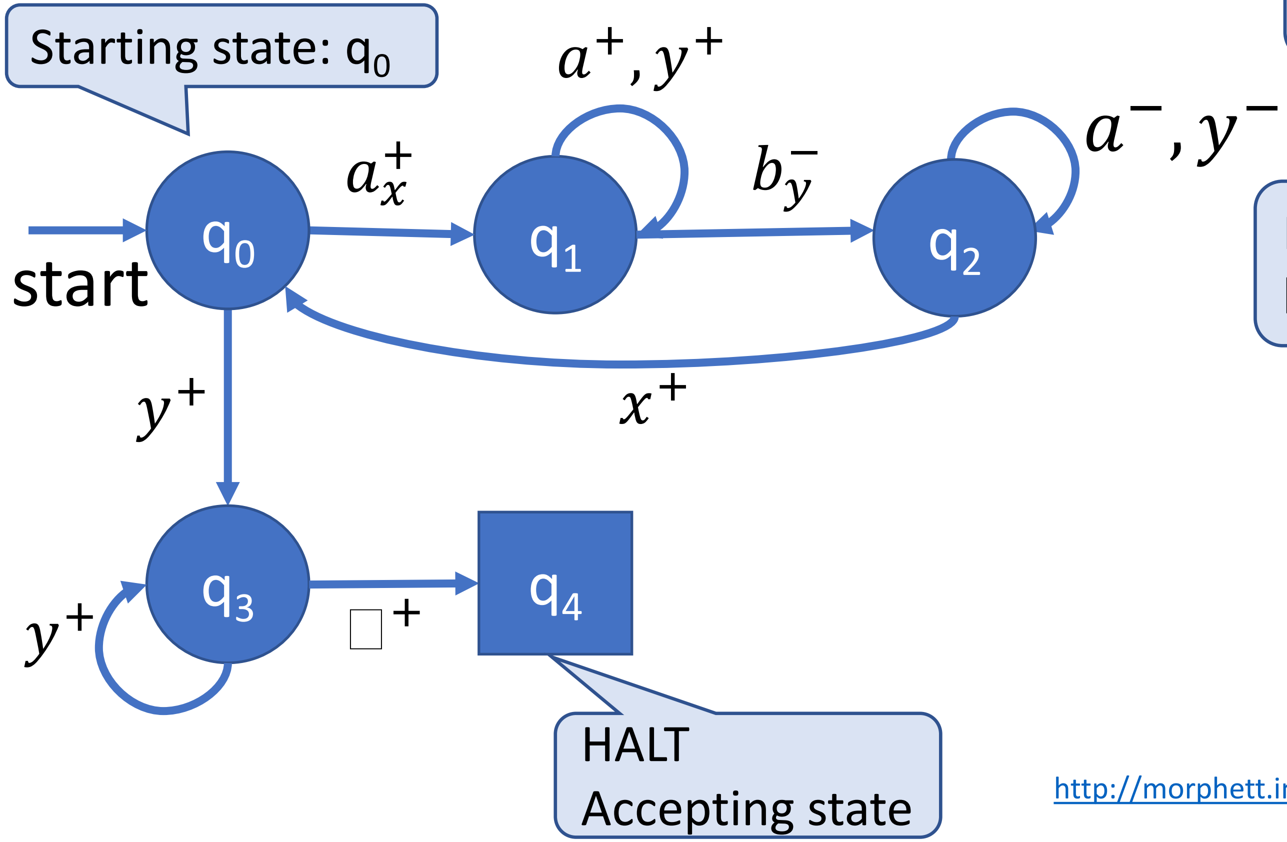
Turing machine that accepts the regular language $L = \{a^n b^n; n > 0;\ \Sigma = \{a, b\}\}$

Initial state	Symbol read	Final state	Write symbol	Move
q_0	a	q_1	x	R
q_1	a	q_1	a	R
q_1	y	q_1	y	R
q_1	b	q_2	y	L
q_2	y	q_2	y	L
q_2	a	q_2	a	L
q_2	x	q_{10}	x	R
q_0	y	q_3	y	R
q_3	y	q_3	y	R
q_3	□	q_4	□	R



Turing machine: $L = \{a^n b^n; n > 0;\ \ \Sigma = \{a, b\}\}$

Turing machine that accepts the regular language $L = \{a^n b^n; n > 0;\ \ \Sigma = \{a, b\}\}$



Tape of the Turing machine

Read / Write head

[]	a	a	b	b	[]	[]
	q_0					

[]	a	a	b	a	[]	[]
	q_0					

[]	a	b	b	a	[]	[]
	q_0					

http://morphett.info/turing/?6ad68516c992ce4bd45bc6c9f3cb2c62

Turing machine: Pseudocode

Turing machine that accepts the regular language $L = \{a^n b^n; n > 0;\ \Sigma = \{a, b\}\}$

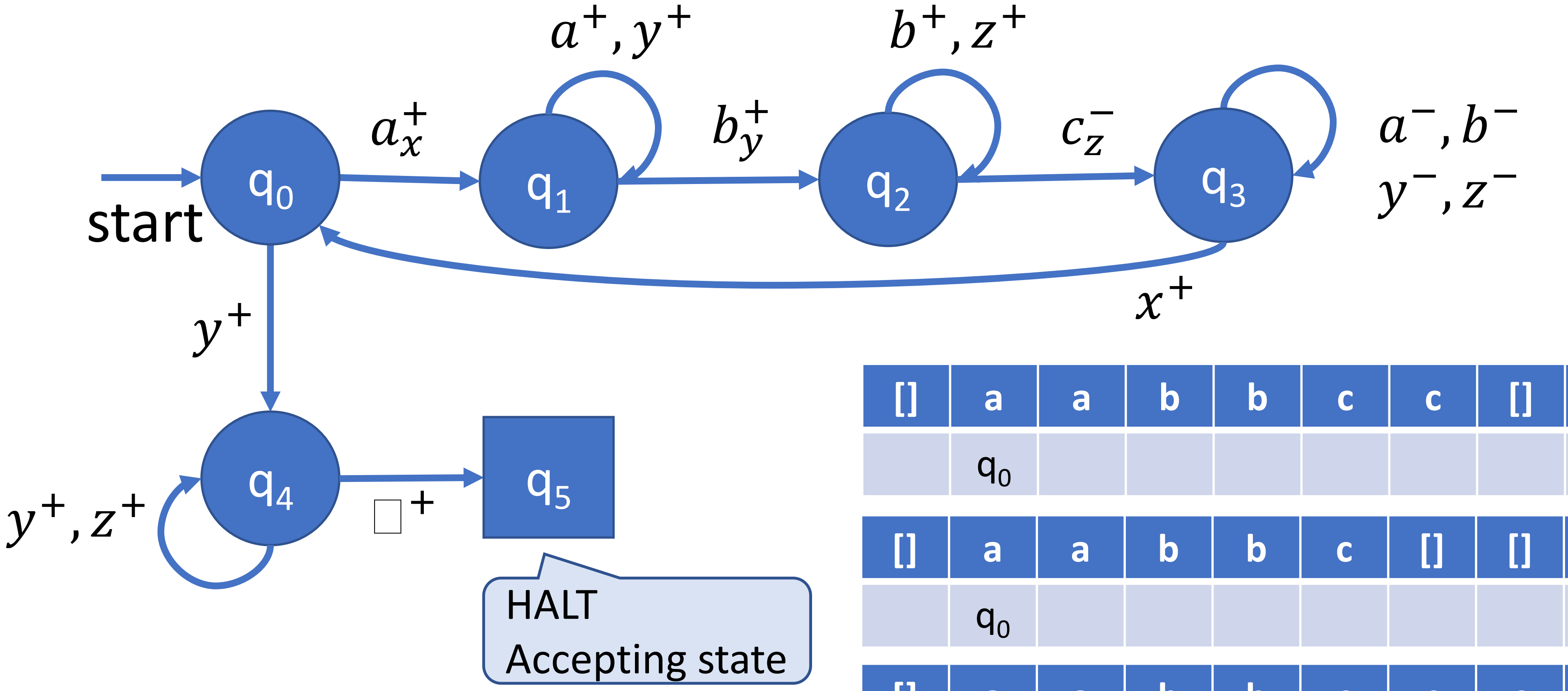
Initial state	Symbol read	Final state	Write symbol	Move
q_0	a	q_1	x	R
q_1	a	q_1	a	R
q_1	y	q_1	y	R
q_1	b	q_2	y	L
q_2	y	q_2	y	L
q_2	a	q_2	a	L
q_2	x	q_0	x	R
q_0	y	q_3	y	R
q_3	y	q_3	y	R
q_3	□	q_4	□	R

1. Replace leftmost ‘a’ with ‘x’.
2. Move R/W head to the right until the first ‘b’ is found.
3. Replace ‘b’ with ‘y’.
4. Enter state q_2 [an ‘a’ has been paired with a ‘b’].
5. Move to the left ignoring the symbols ‘y’ and ‘a’ until an ‘x’ is found.
6. Enter state q_0 and move the R/W head to the right.
    6.1 Process the next ‘a’ if any remains.
    6.2 Otherwise, check whether all the symbols ‘a’ ‘b’ have been paired.

Turing machine:

$$L = \{a^n b^n c^n; n > 0;\ \Sigma = \{a, b, c\}\}$$

Turing machine that accepts the regular language $L = \{a^n b^n c^n; n > 0;\ \Sigma = \{a, b, c\}\}$



[]	a	a	b	b	c	c	[]	[]	[]
	q_0								

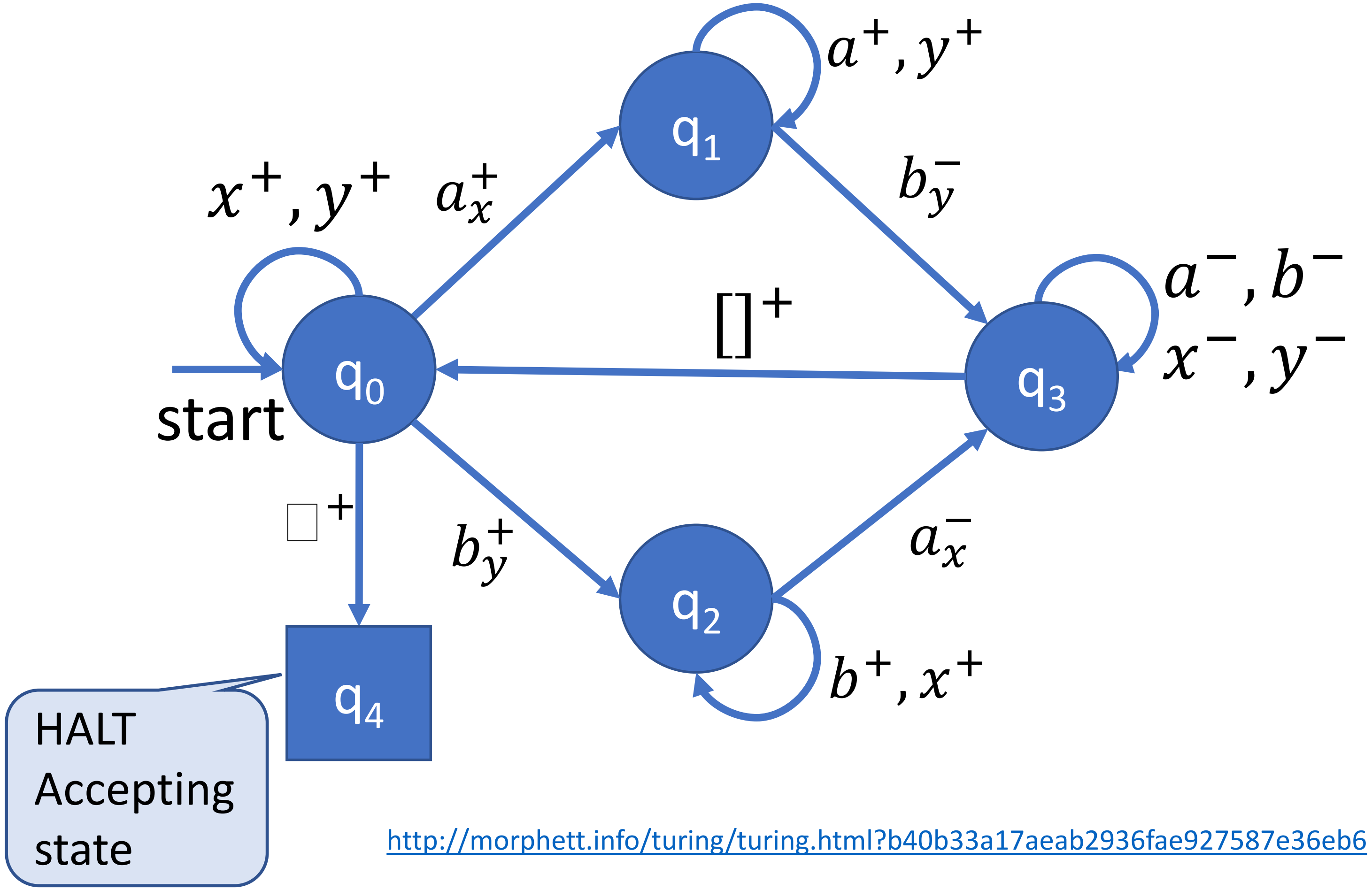
[]	a	a	b	b	c	[]	[]	[]	[]
	q_0								

[]	a	a	b	b	c	c	c	[]	[]
	q_0								

http://morphett.info/turing/turing.html?bc53d03c5e1818157bcd20a07175495a

Turing machine: Same numbers of ‘a’, ‘b’

Turing machine that accepts words with the same number of ‘a’ and ‘b’.



[]	a	a	b	b	[]	[]
	q_0					

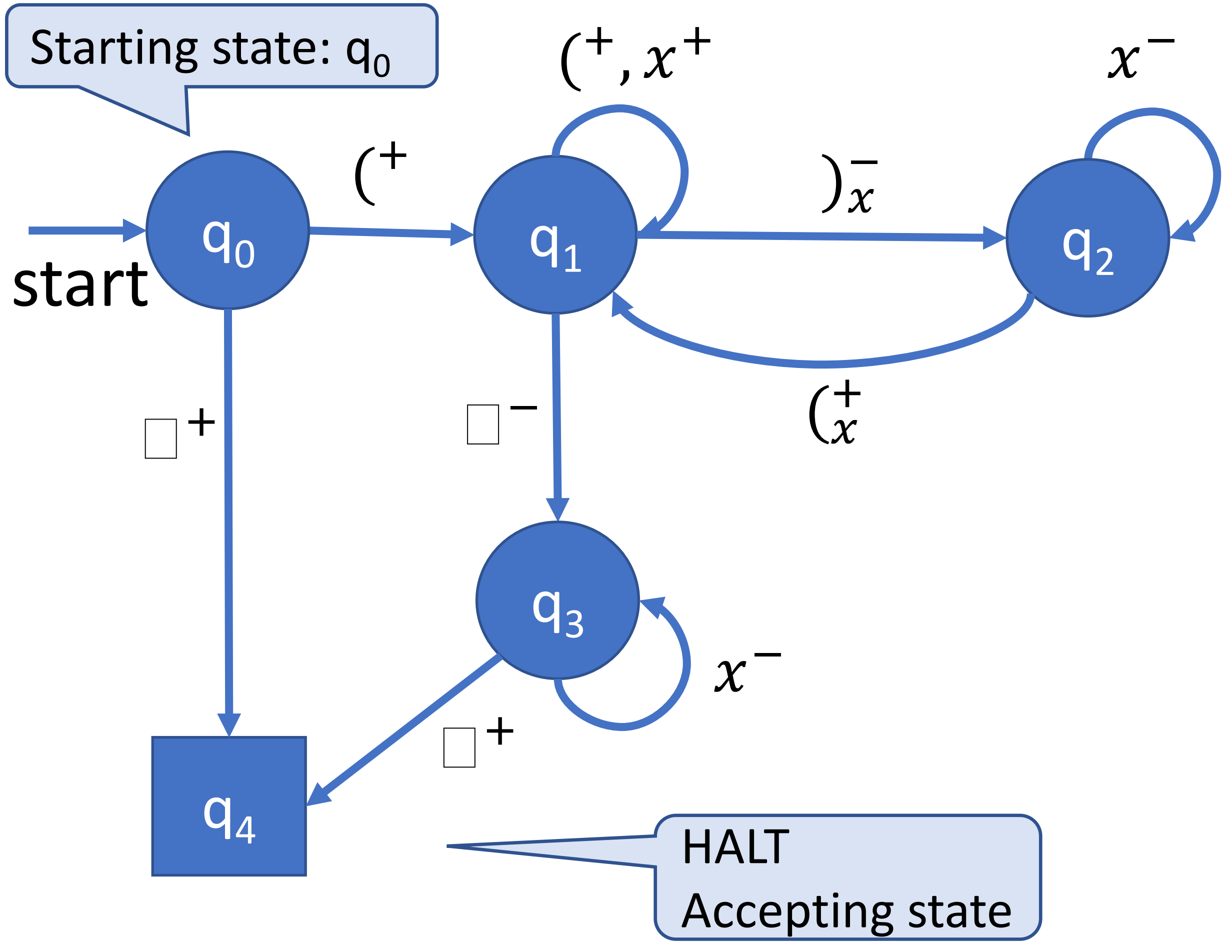
[]	b	a	b	a	[]	[]
	q_0					

[]	a	b	b	a	[]	[]
	q_0					

[]	a	a	b	a	[]	[]
	q_0					

http://morphett.info/turing/turing.html?b40b33a17aeab2936fae927587e36eb6

Turing machine for matching parentheses



[]	(	)	(	)	[]	[]	[]	[]	[]
	q_0								

[]	(	(	)	)	[]	[]	[]	[]	[]
	q_0								

[]	(	(	)	(	)	)	[]	[]	[]
	q_0								

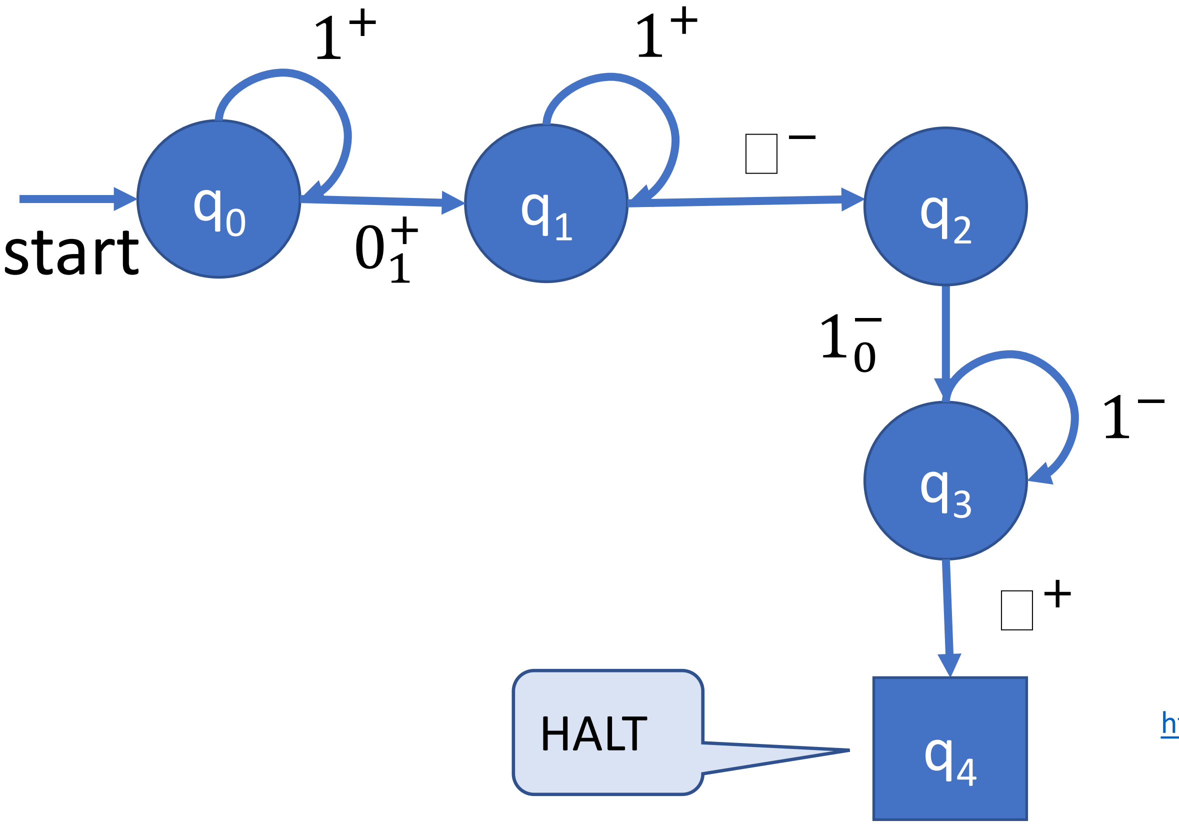
[]	(	(	)	(	)	[]	[]	[]	[]
	q_0								

[]	(	(	)	)	)	[]	[]	[]	[]
	q_0								

http://morphett.info/turing/?c11df429a6cd26e740914f91fed94edf

Turing machine for adding two numbers

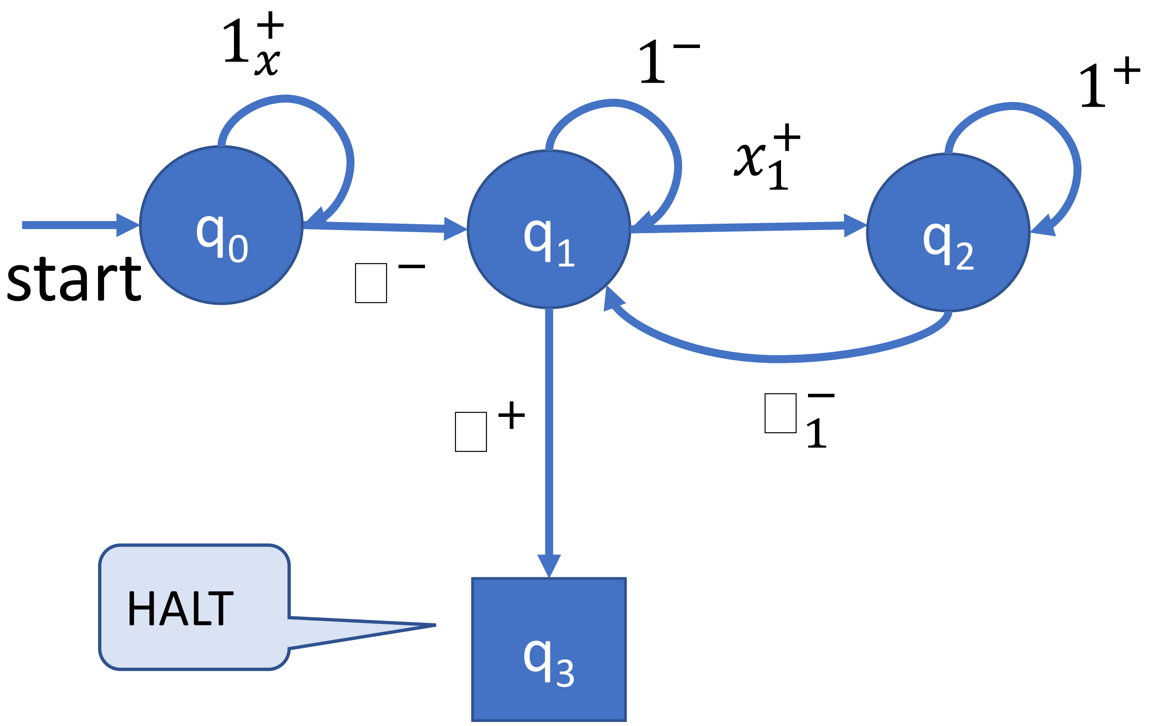
- The numbers are represented in unary notation: Natural number n is represented by a sequence of n consecutive ‘1’ .
- Initially, they are separated by ‘0’.



[]	1	1	0	1	1	1	[]	[]	[]
	q_0								

http://morphett.info/turing/?afe6b337a1e4e0e40df98bcb9f50f091

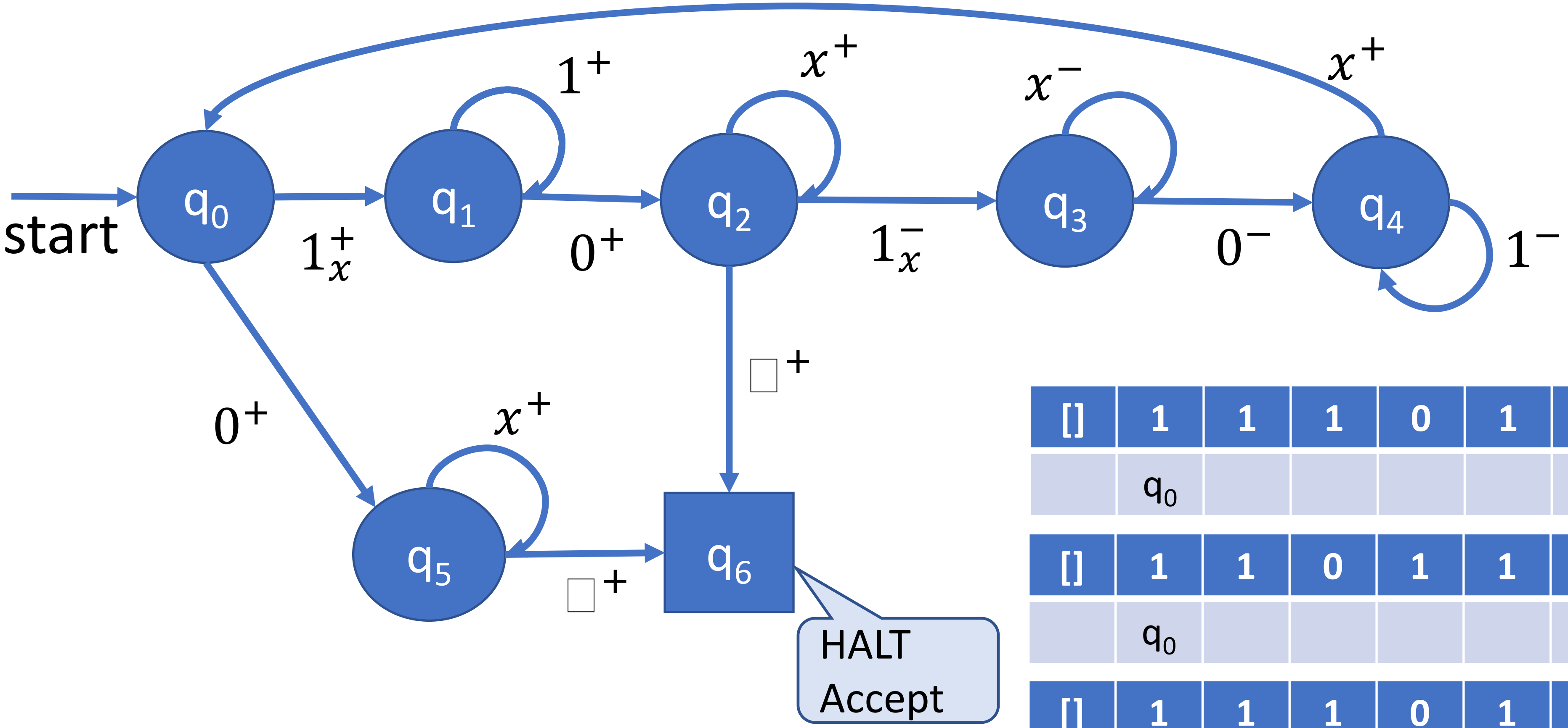
Turing machine for copying unary numbers



[]	1	1	1	[]	[]	[]	[]	[]	[]
	q_0								

http://morphett.info/turing/?c8ddd8b70b91c78f8cb09c8e266b2ee1

Turing machine for comparing two numbers: Larger or equal Boolean function



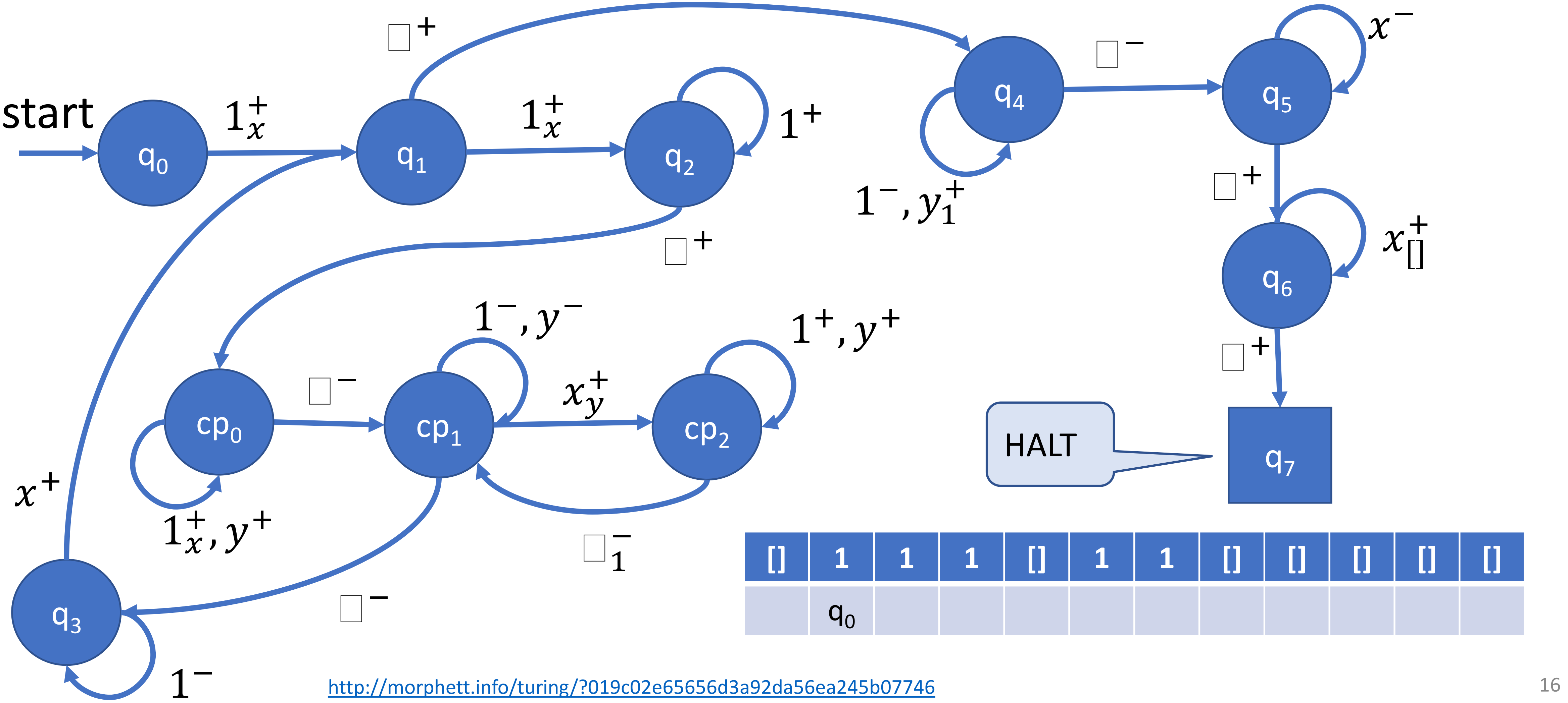
[]	1	1	1	0	1	1	1	[]	[]
	q_0								

[]	1	1	0	1	1	1	[]	[]	[]
	q_0								

[]	1	1	1	0	1	1	[]	[]	[]
	q_0								

http://morphett.info/turing/?53ae290d23b3dc522ba2f05a8a08fd35

Turing machine
for multiplying 2 unary numbers

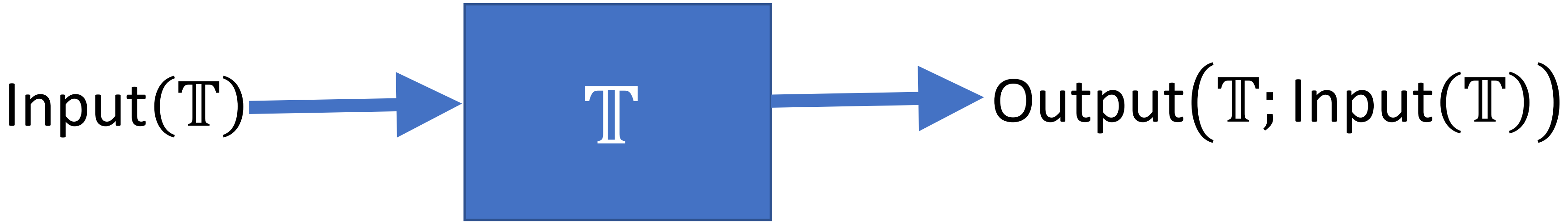


[]	1	1	1	[]	1	1	[]	[]	[]	[]	[]
	q_0										

http://morphett.info/turing/?019c02e65656d3a92da56ea245b07746

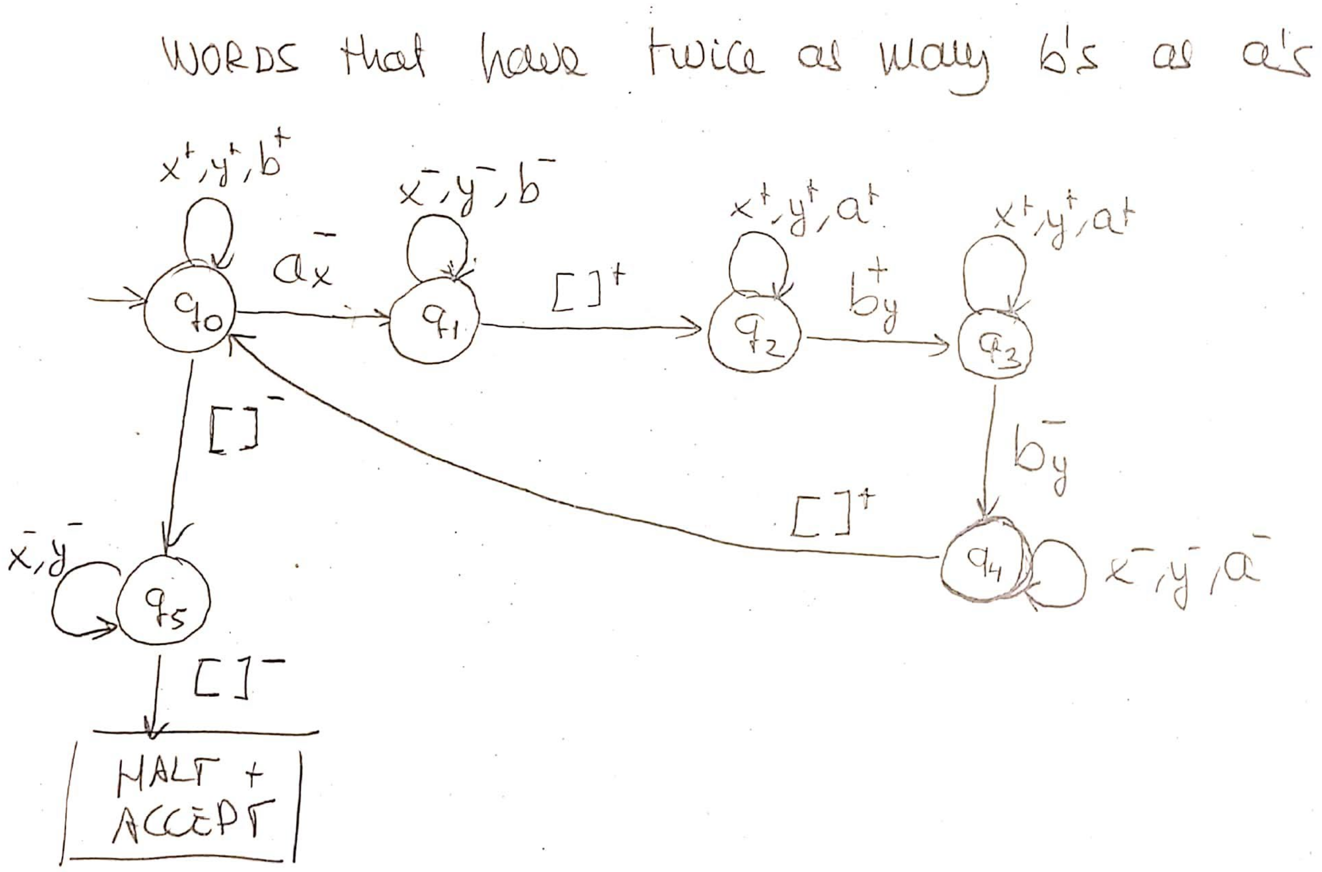
Universal Turing machine

- Up to this point we have designed machines that solve a specific task



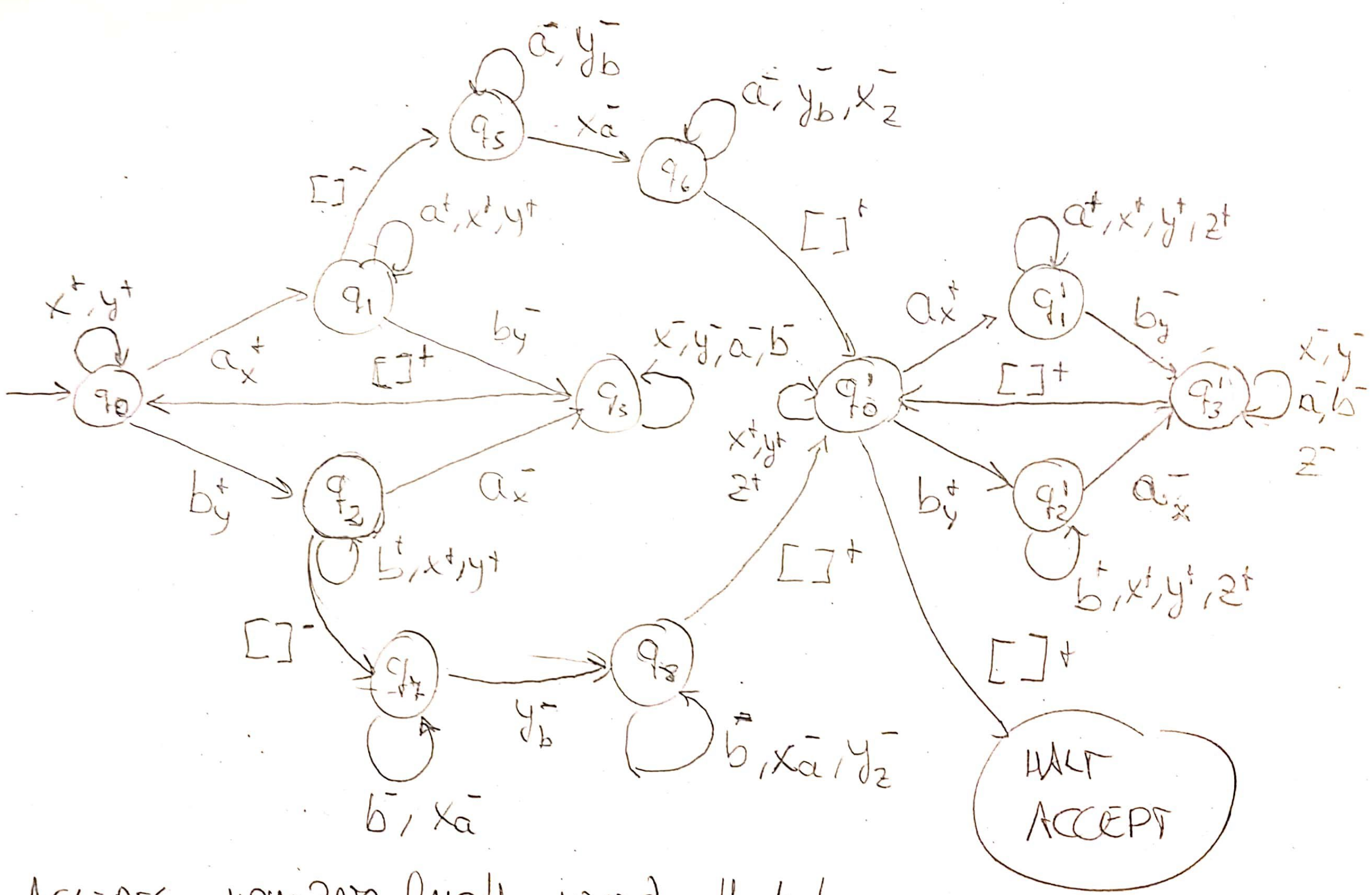
- Goal: Build a universal machine $\mathbb{U}$ that can mimic the operation of any machine $\mathbb{T}$ on any input

Other Turing Machines



- http://morphett.info/turing/turing.html?f83b8f8b059134b3773a72b917fb9bb6

Other Turing Machines

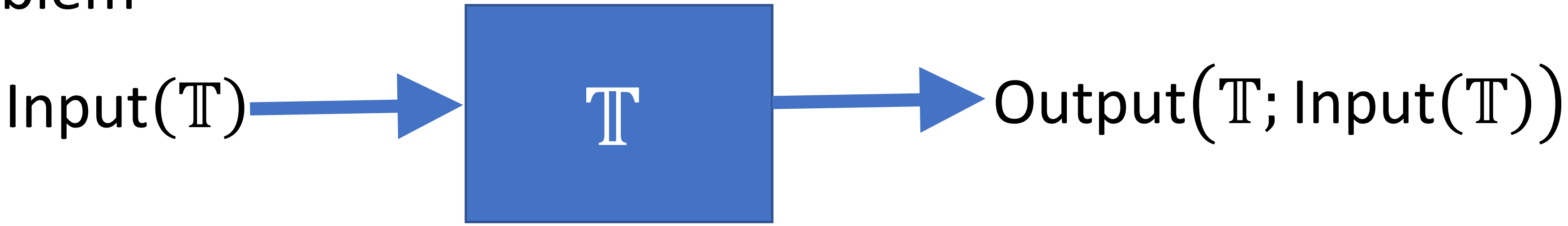


Accepts non-zero length words that have either twice as many a's as b's or twice as many b's as a's

- http://morphett.info/turing/turing.html?ea0eeaa09b5a81618ac88cc336514205

Universal Turing machine

- Up to this point we have designed machines that solve a specific problem



- Goal: Build a universal machine $\mathbb{U}$ that can emulate the operation of any machine $\mathbb{T}$ on any input

$\text{Input}(\mathbb{U})$
$= \big(\text{Description}(\mathbb{T}), \text{Input}(\mathbb{T})\big)$

$$\text{Output}\big(\mathbb{U}; \text{Description}(\mathbb{T}), \text{Input}(\mathbb{T})\big) = \text{Output}\big(\mathbb{T}; \text{Input}(\mathbb{T})\big)$$

Universal Turing machine

- The description of $\mathbb{T}$ is the list of instructions that specify the operation of the machine $\mathbb{T}$.
- Therefore, the tape of $\mathbb{U}$ contains the following information:
  - Contents of the tape of $\mathbb{T}$ (initially, Input$(\mathbb{T})$).
  - Description$(\mathbb{T})$: List of instructions (tuples) of $\mathbb{T}$.
  - Current state of $\mathbb{T}$.
  - Position of the Read/Write head of $\mathbb{T}$.
  - Symbol read by the Read/Write head from the tape of $\mathbb{T}$.

Universal Turing machine

- Machine $\mathbb{T}$
  - Current state: q_2 (10)
  - Symbol read: s_R (1)
  - Position of tape: M

Initial state	Symbol read	Final state	Write symbol	Move	description of $\mathbb{T}$
q_0 (00)	1	q_1 (01)	1	L (0)	00 1 01 1 0
q_2 (10)	1	q_3 (11)	0	R (1)	10 1 11 0 1
q_3 (11)	1	q_0 (00)	0	L (0)	11 1 00 0 0
…	…	…	…	…	…

- Current state of the tape of $\mathbb{U}$

…	0	M	0	Y	1	0	1	X	0	0	1	0	1	1	0	X	1	0	1	1	1	0	1	X	1	1	1	0	0	0	0	…
Tape of $\mathbb{T}$					q_2		s_R		Instruction 1 of $\mathbb{T}$								Instruction 2 of $\mathbb{T}$								Instruction 3 of $\mathbb{T}$							

- State of the tape of $\mathbb{U}$ after executing instruction "101"

…	0	0	M	Y	1	1	0	X	0	0	1	0	1	1	0	X	1	0	1	1	1	0	1	X	1	1	1	0	0	0	0	…
Tape of $\mathbb{T}$					q_3		s_R		Instruction 1 of $\mathbb{T}$								Instruction 2 of $\mathbb{T}$								Instruction 3 of $\mathbb{T}$							

Turing’s thesis

Paper-and-pencil method

Any computation that can be carried out by mechanical means can be performed by some Turing machine.

Basic law of computer science
(same status as Newton’s law for Physics)

Turing’s thesis: What is computable

Observations on computability:

- Turing’ thesis (also called Church-Turing’s thesis) defines what computation is: A problem is computable iff it can be solved by a TM.
- Nobody has identified a problem that can be solved algorithmically (in finite time) for which a TM cannot be designed.
- Anything that can be done with a digital computer can be done by a TM.
- The only problems that can be solved by other types of machines, such as Quantum Computers, are those that are Turing computable.
- There are alternative models for mechanical computation, such as the λ-calculus developed by Alonzo Church. They can be shown to be equivalent to the TM model.

Limits of computation

- There are problems that cannot be solved by a Turing Machine.
- Therefore, if Turing's thesis is correct, they cannot be solved by any algorithm (mechanical process) that yields an answer in finite time.
- The proof of this theorem is a constructive one: We will define a problem for whose solution no Turing machine can be built.

The halting problem

$\mathbb{T}$ may get stuck (and therefore does not halt) for some inputs

It is not possible to build a Turing Machine $\mathbb{H}$ that, for any Turing Machine, $\mathbb{T}$, and any possible input can determine whether $\mathbb{T}$ halts when it operates on Input$(\mathbb{T})$.

Description$(\mathbb{T})$ → $\mathbb{H}$ → Output$\left(\mathbb{H}; \text{Description}(\mathbb{T}), \text{Input}(\mathbb{T})\right)$

Input$(\mathbb{T})$ → $\mathbb{H}$

Input$(\mathbb{H})$ $= \left(\text{Description}(\mathbb{T}), \text{Input}(\mathbb{T})\right)$

$\mathbb{H}$ always halts (never gets stuck) and yields an output ("Stuck" / " Not Stuck")

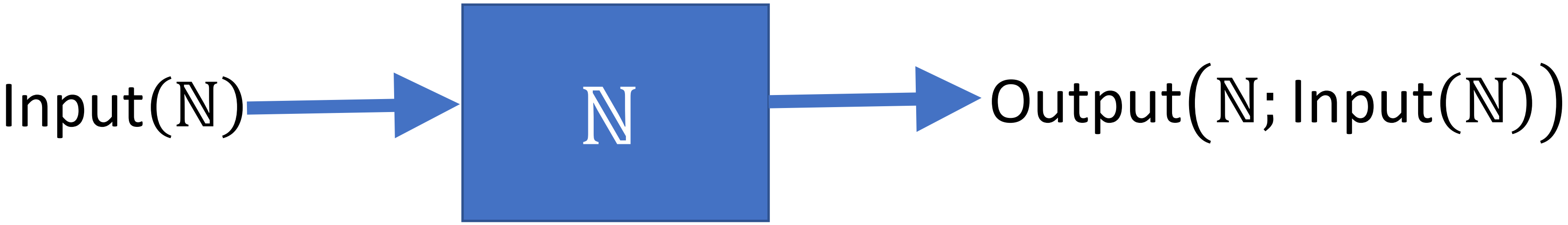
$$\text{Output}\left(\mathbb{H}; \text{Description}(\mathbb{T}), \text{Input}(\mathbb{T})\right) = \begin{cases} \text{"Stuck", if } \mathbb{T} \text{ does not halt (it gets stuck) when it operates on input}(\mathbb{T}) \\ \text{"Not stuck ", if } \mathbb{T} \text{ halts(does not get stuck) when it operates on input}(\mathbb{T}) \end{cases}$$

Proof from https://www.youtube.com/watch?v=92WHN-pAFCs

Auxiliary machines

- The photocopy machine

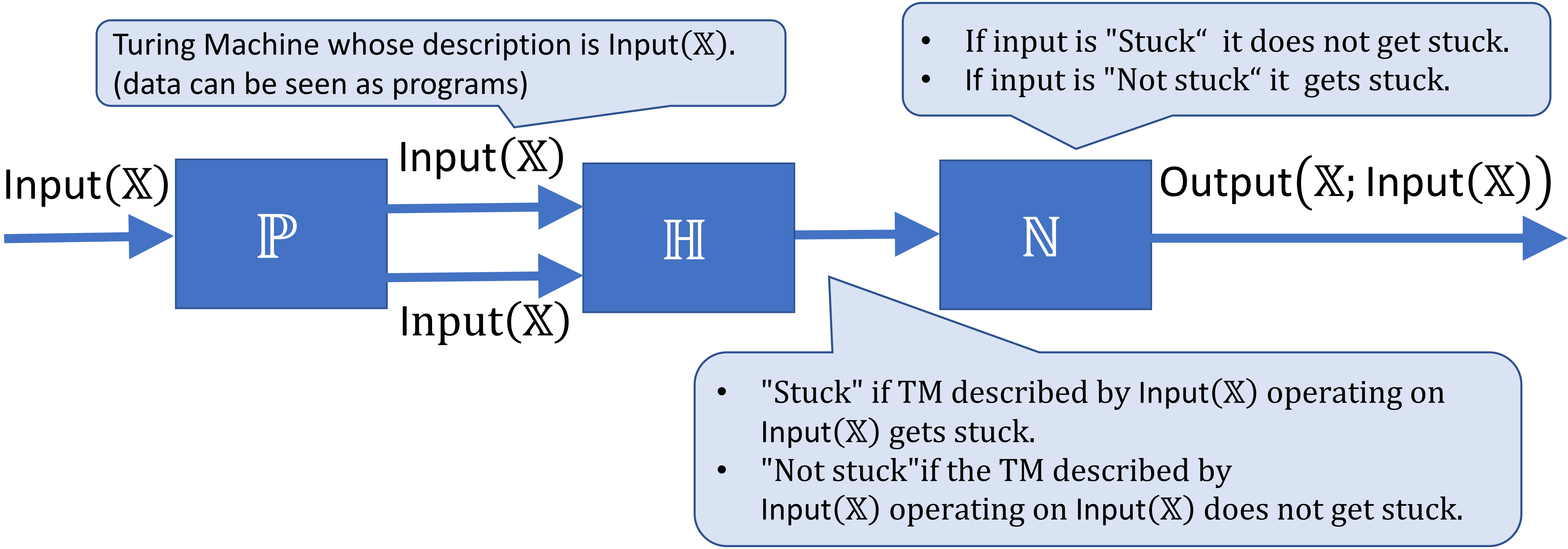
- The negator



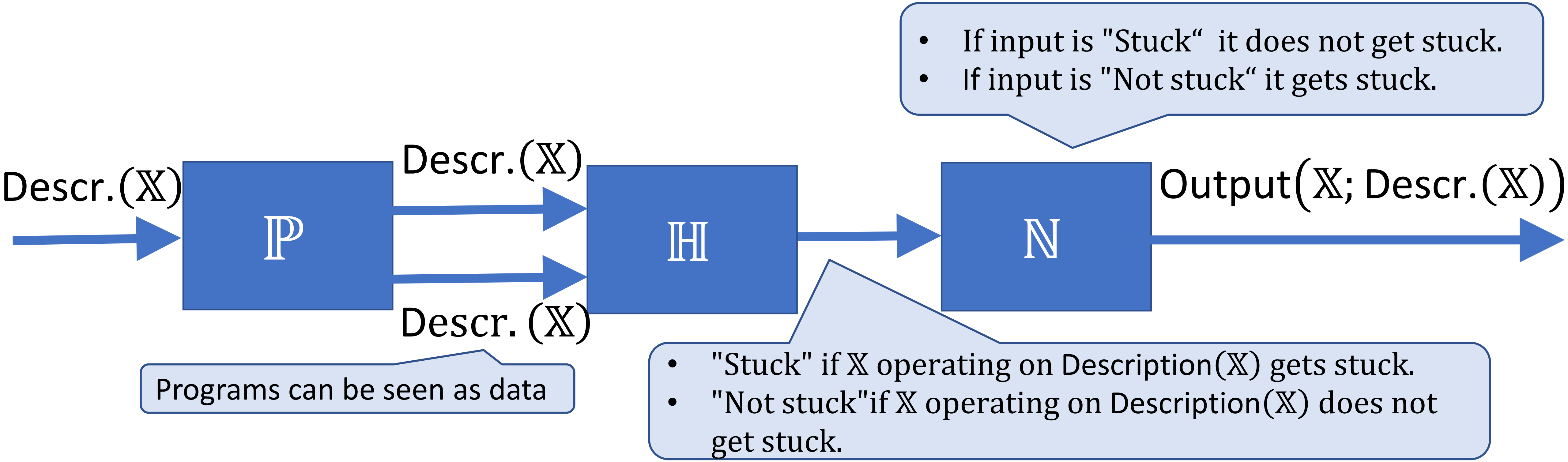
  - If $\text{Input}(\mathbb{N}) =$ "Stuck" then the machine does not get stuck (it halts, and yields some output)
  - If $\text{Input}(\mathbb{N}) =$ "Not stuck" then the machine gets stuck (it does not halt)

The $\mathbb{X}$ machine

Assuming that $\mathbb{H}$ exists, let us build the $\mathbb{X}$ machine



The $\mathbb{X}$ machine operates on Description($\mathbb{X}$)



- If $\mathbb{X}$ operating on Description($\mathbb{X}$) gets stuck, then $\mathbb{X}$ does not get stuck.
- If $\mathbb{X}$ operating on Description($\mathbb{X}$) does not get stuck, then $\mathbb{X}$ gets stuck.

If $\mathbb{H}$ exists, we reach a contradiction. Therefore, $\mathbb{H}$ cannot exist.

This is the end

https://www.youtube.com/watch?v=ayo75QnDnss