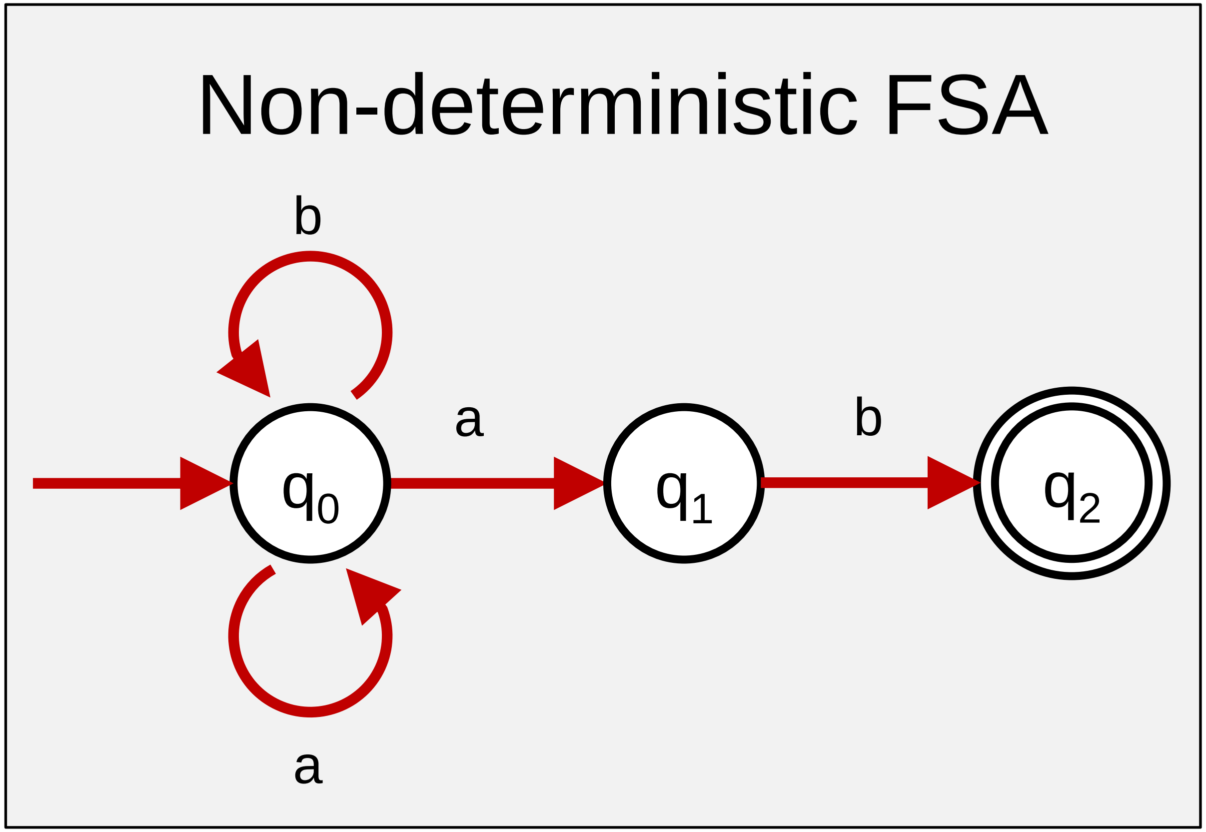
Mathematical Concepts (G6012)

Lecture 9

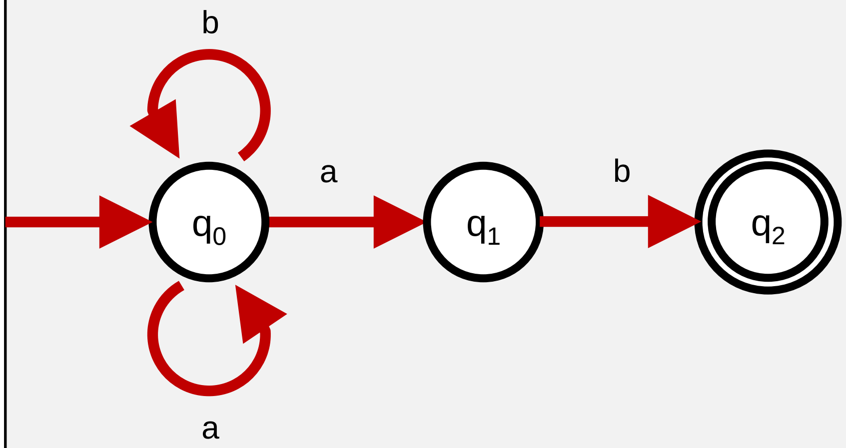
Thomas Nowotny
Chichester I, Room CI-105
Office hours: Tuesdays 15:00-16:45
T.Nowotny@sussex.ac.uk

BB: Translate the following non-deterministic FSA systematically into a deterministic FSA

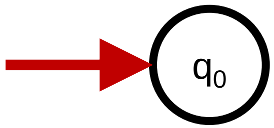


BB Systematic translation of non-deterministic to deterministic FSA

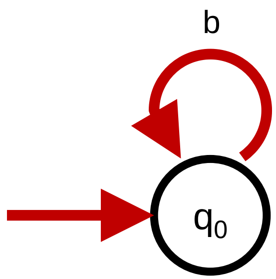
Non-deterministic FSA



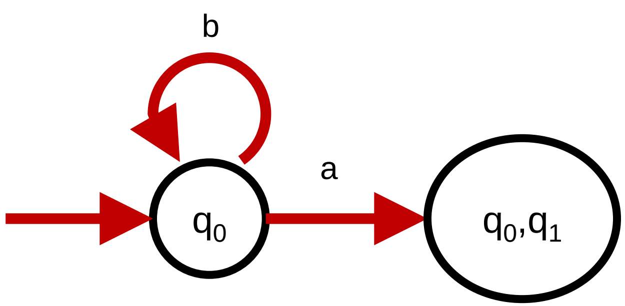
1. Start:



2. “b” transistions out of q_0:

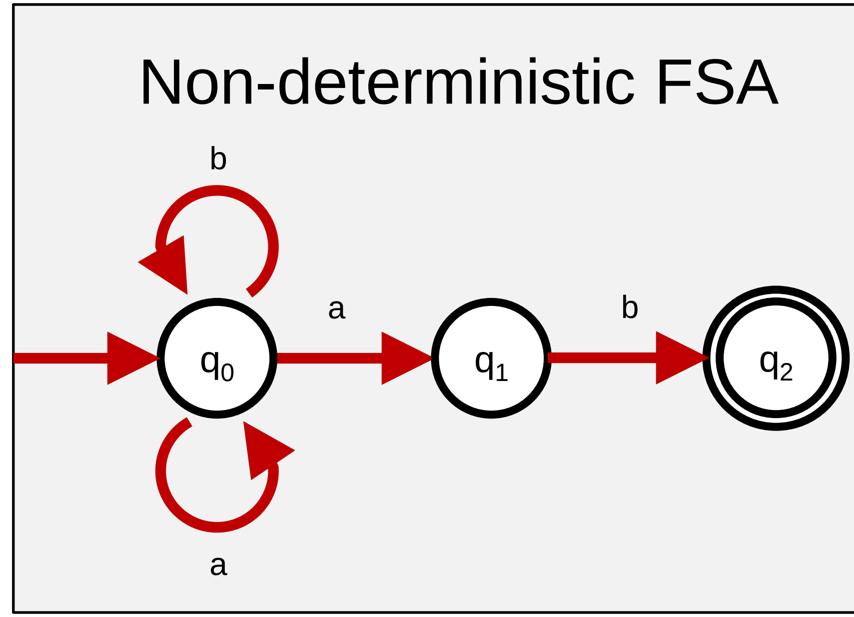


3. “a” transistions out of q_0:



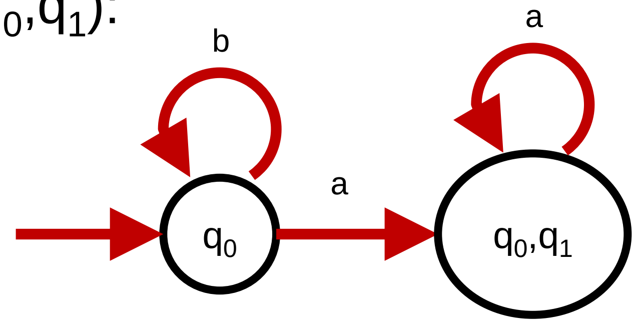
We can get with a from q_0 to q_0 or q_1

BB Systematic translation of non-deterministic to deterministic FSA



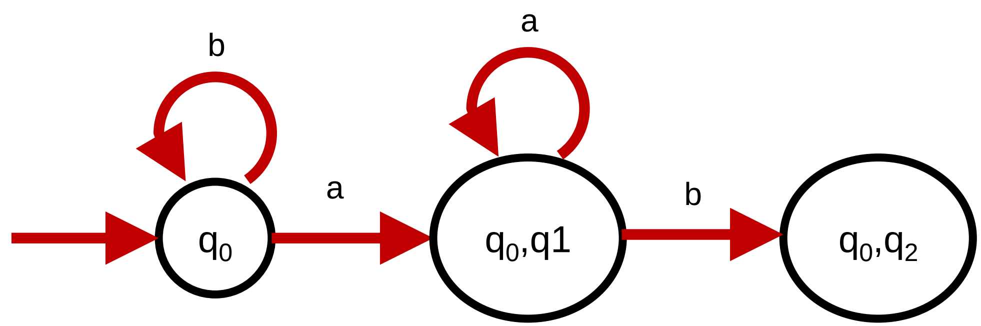
4. ‘a’ transitions from (q_0,q_1):

We can get with ‘a’ from q_0 to q_0 or q_1; we don’t get anywhere with ‘a’ from q_1; hence, we have total target set of (q_0,q_1):



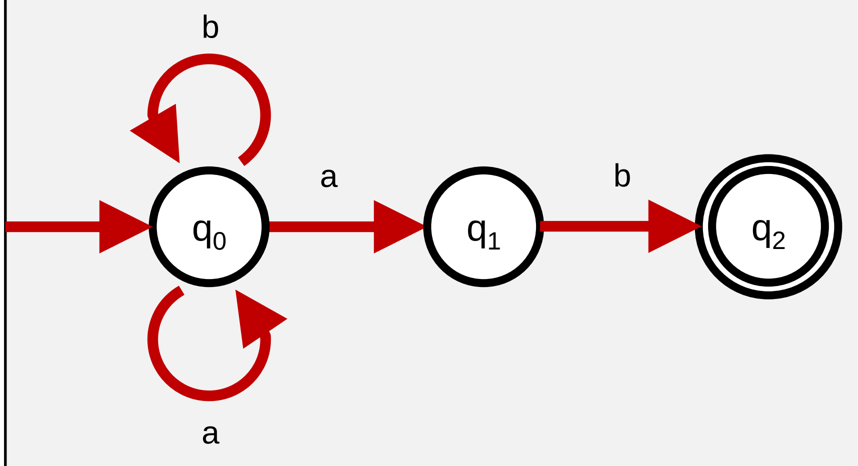
5. ‘b’ transitions from (q_0,q_1):

We can get with ‘b’ from q_0 to q_0; we get with ‘b’ from q_1 to q_2; hence, total target set of (q_0,q_2):



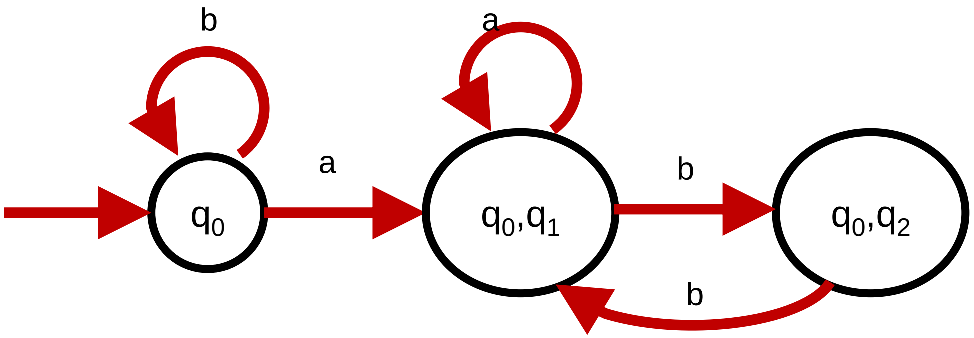
BB Systematic translation of non-deterministic to deterministic FSA

Non-deterministic FSA



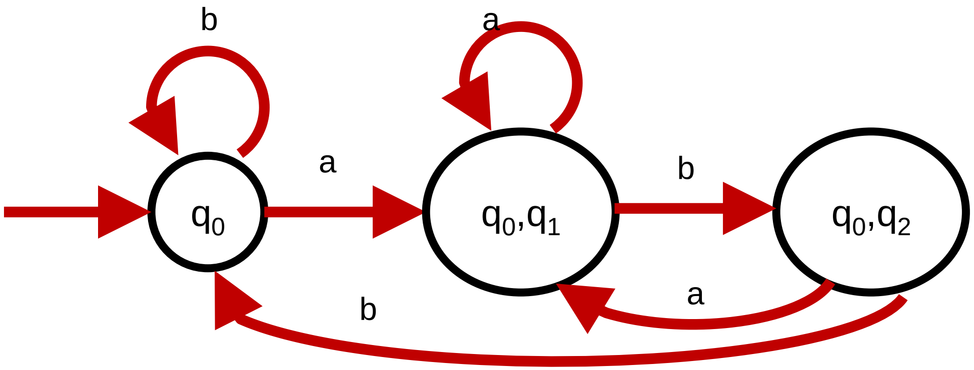
6. ‘a’ transitions from (q_0,q_2):

We can get with ‘a’ from q_0 to q_0 or q_1; we don’t get anywhere with ‘a’ from q_2; hence, we have total target set of (q_0,q_1):

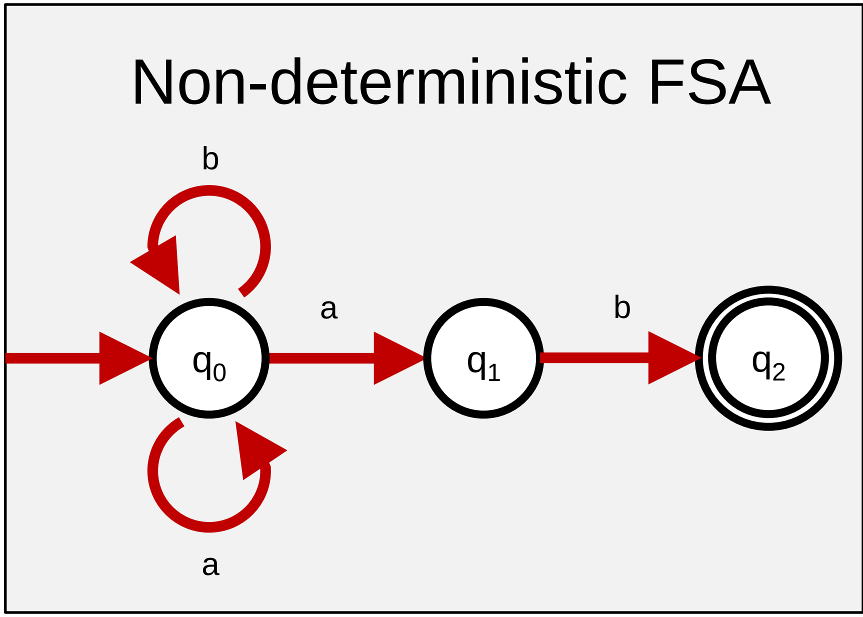


4. ‘b’ transitions from (q_0,q_2):

We can get with ‘b’ from q_0 to q_0; we get nowhere with ‘b’ from q_2; hence, total target set of q_0:

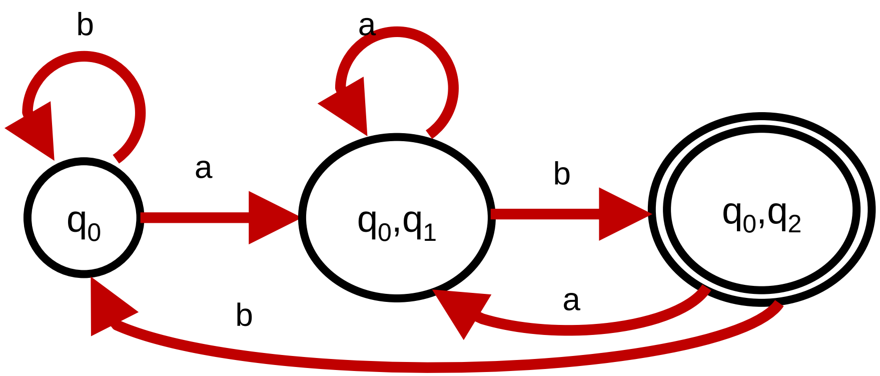


BB Systematic translation of non-deterministic to deterministic FSA



6. Final state:

Every state that contains a final state is final: Here (q_0,q_2):



That's the final deterministic automaton done!

Properties of Pushdown Automata

- Family of languages:
  - PDA accept the same family of languages as can be expressed by **Context Free Grammars**
  - In other words they accept exactly the **Context Free Languages**
  - Context Free Grammars are used to describe (define) programming language syntax
  - (Also equivalent to BNF and syntax charts)

Context-Free Grammar

- Is define by “productions” or “production rules”:

  1. $S \mapsto aSb$
  2. $S \mapsto ab$

- Are applied repeatedly, e.g

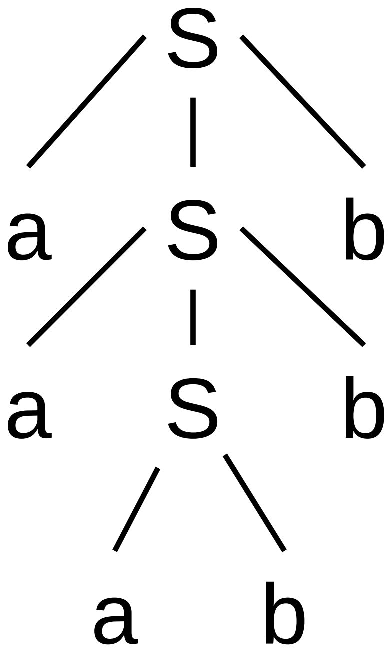
  $S \mapsto aSb \mapsto aaSbb \mapsto aaabbb$

  (rules applied: 1., 1., 2.)

- Generates a’s followed by same number of b’s

Derivation tree

- Generating a word can be visualised as a tree:



Other example

- Productions:

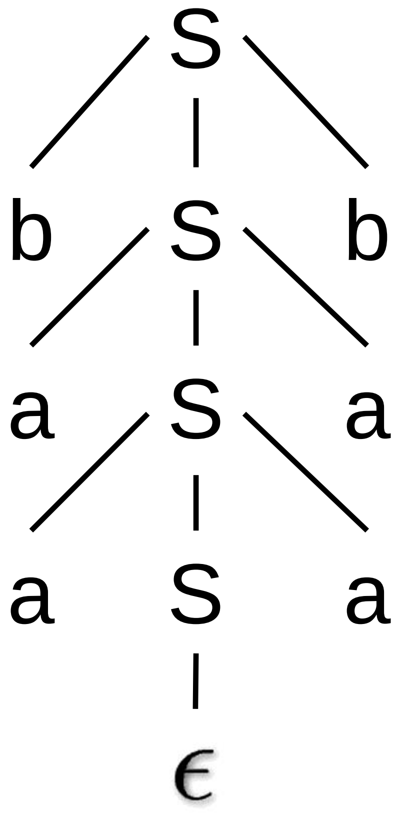
  S $\mapsto$ aSa

  S $\mapsto$ bSb

  S $\mapsto$ ϵ

- Generates the palindrome language $\{SS^R \,|\, S \in \{a,b\}^n, n \geq 0\}$ where the ^{R} denotes the reverse of the string

Derivation tree example



Pushdown Automata: Limits of Power

- Can be achieved:
  - Language of palindroms
  - Counting two symbols
  - Programming languages (deterministically)
  - Natural Languages?
- What can't be done:
  - Copy language $\{SS \mid S \in \{a,b\}^n, n \geq 0\}$
  - Counting symbols beyond 2
  - (Crossing dependencies)

Performance consideration

- When syntax-checking programs,
  - PDA based checking can take $O(n^3)$ time
  - This can be very slow for large programs
  - However, if the PDA is deterministic, time is only $O(n)$

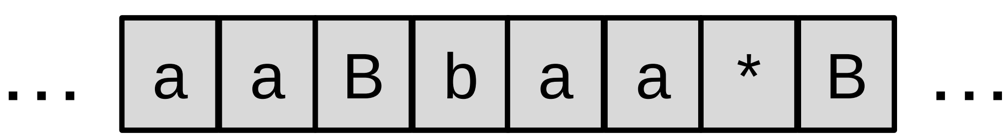
Stacks vs Pushdowns

- Most people would not make a distinction
- If a distinction is made
  - Pushdown strictly push-pop
  - Stack can be inspected read-only
- Stack automata are a more powerful but little known type of machine

TURING MACHINES

Turing Machines (TM)

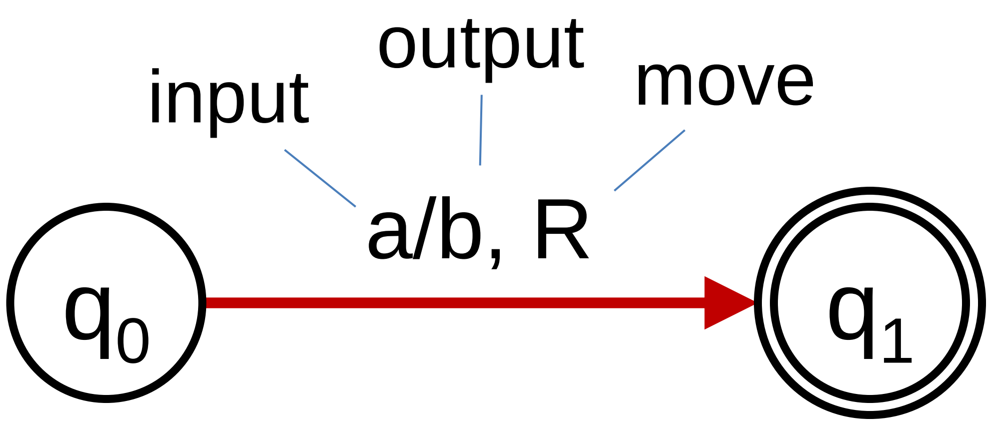
- Are a very simple extension to finite state machines
- The main change is to allow **editing the input tape**
- No limit on the size of the tape
- Tape 2-way infinite (like the integers)



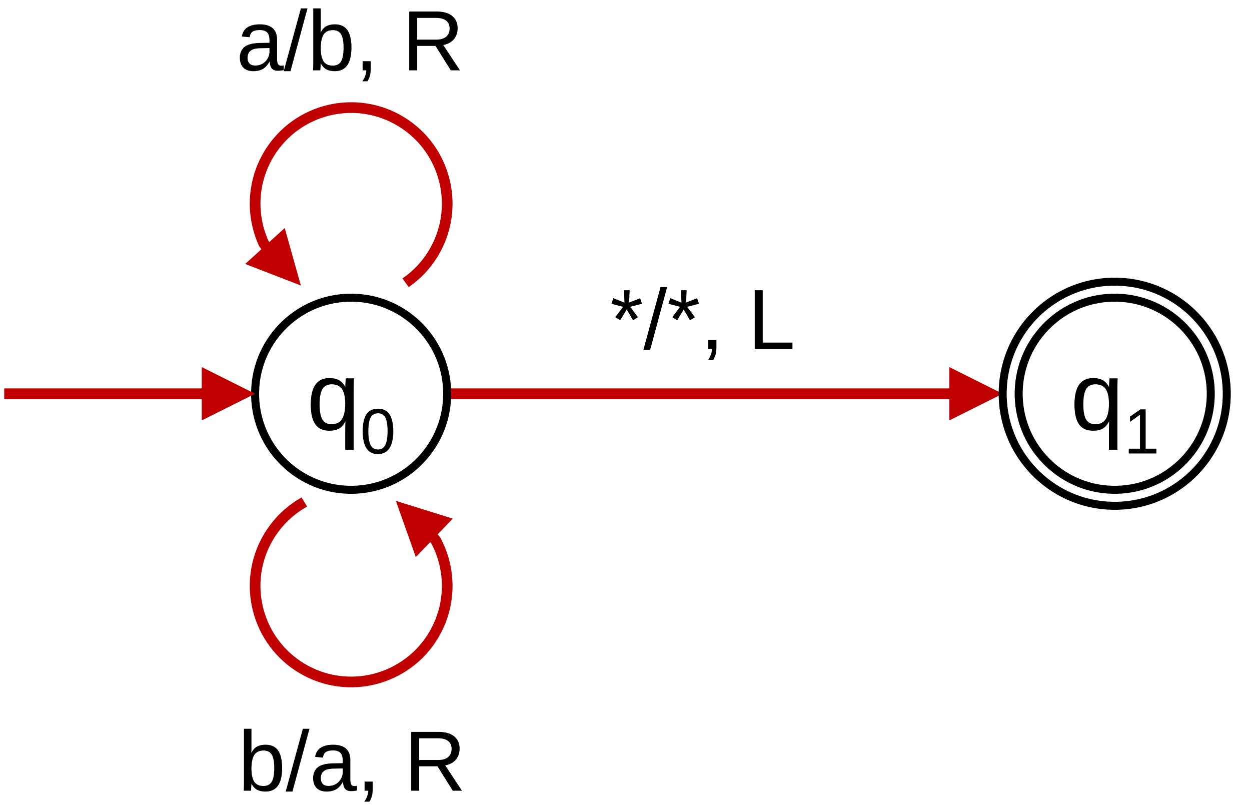
- (We will use the symbol B for blank positions)

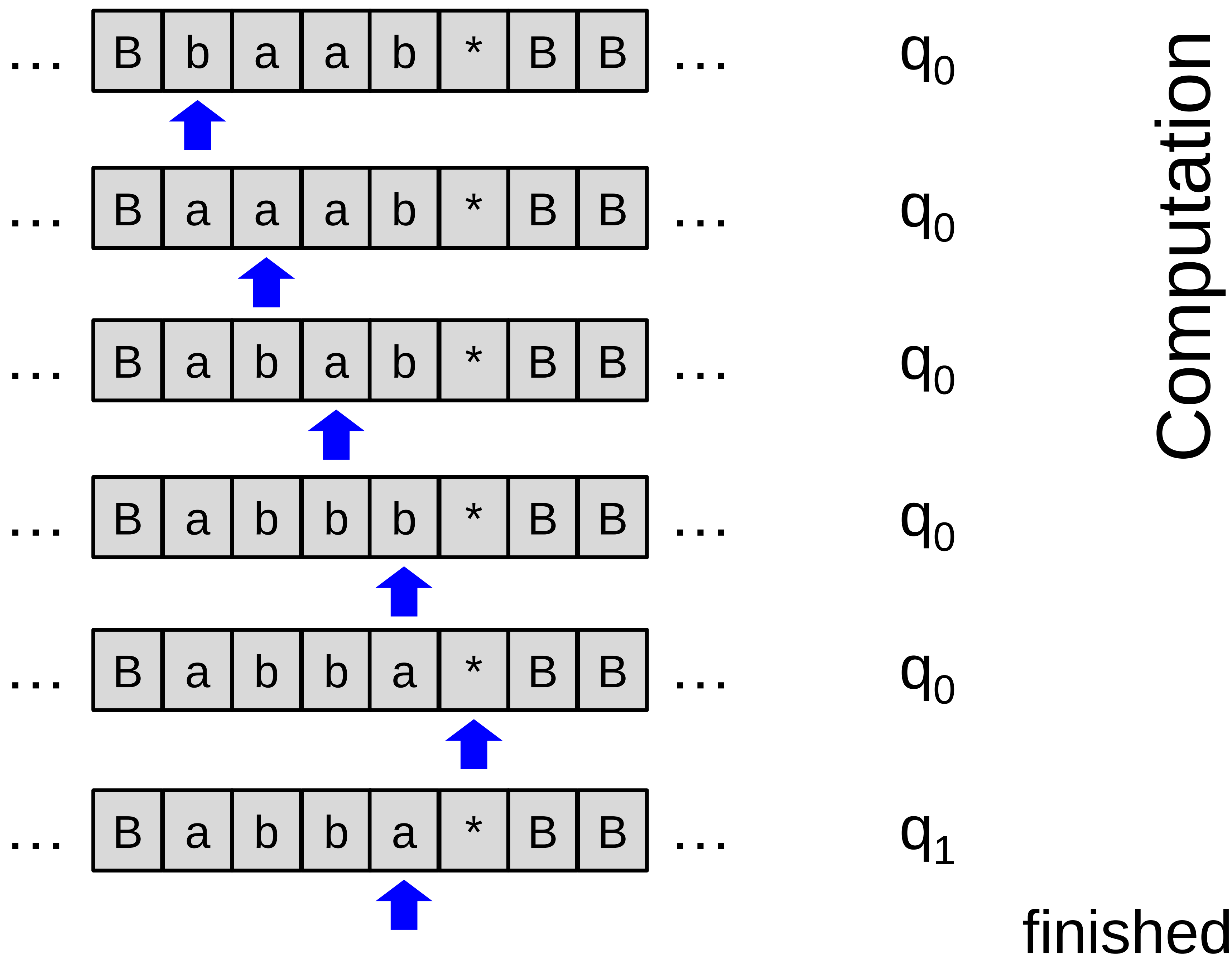
Transitions in TM

- Current state
- New state
- Symbol currently read
- New symbol to replace the read symbol
- Direction to move the tape head (left (L), right (R), stay (S))



Example



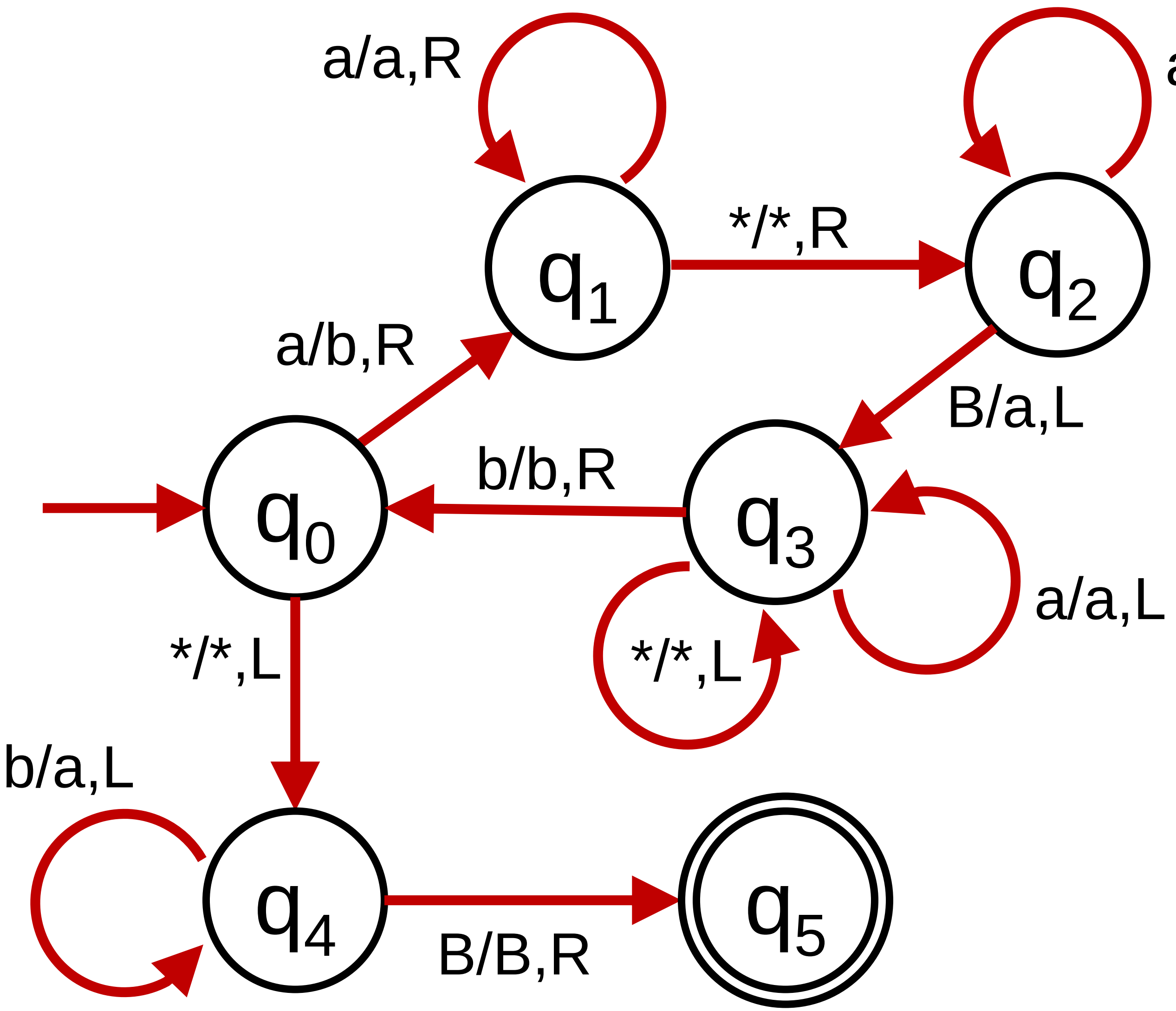


… B b a a b * B B … q_0
… B a a a b * B B … q_0
… B a b a b * B B … q_0
… B a b b b * B B … q_0
… B a b b a * B B … q_0
… B a b b a * B B … q_1
Computation
finished

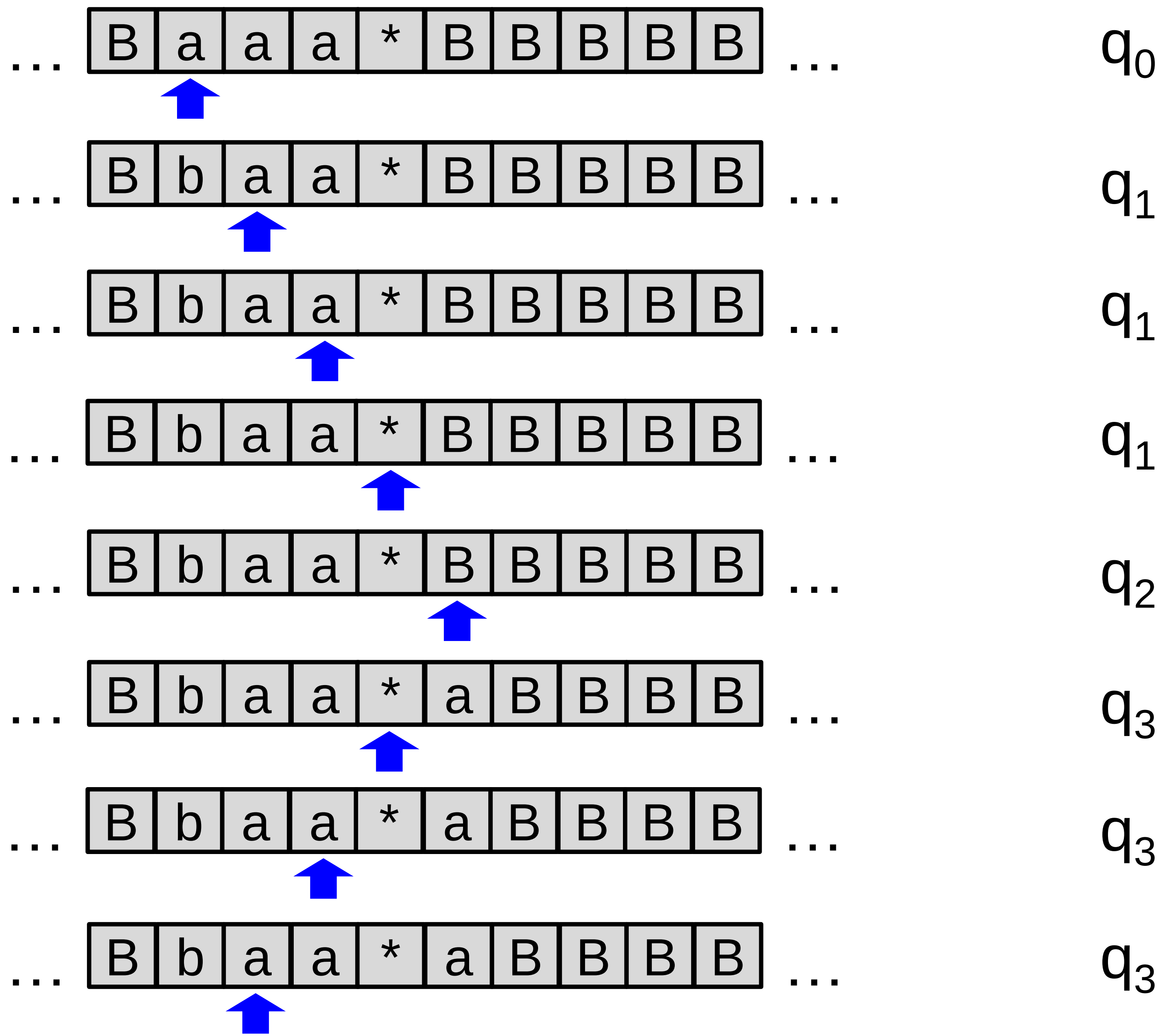
What did it do?

- *baab* became *abba*
- This machine swaps *a* to *b* and *b* to *a* until it finds a *

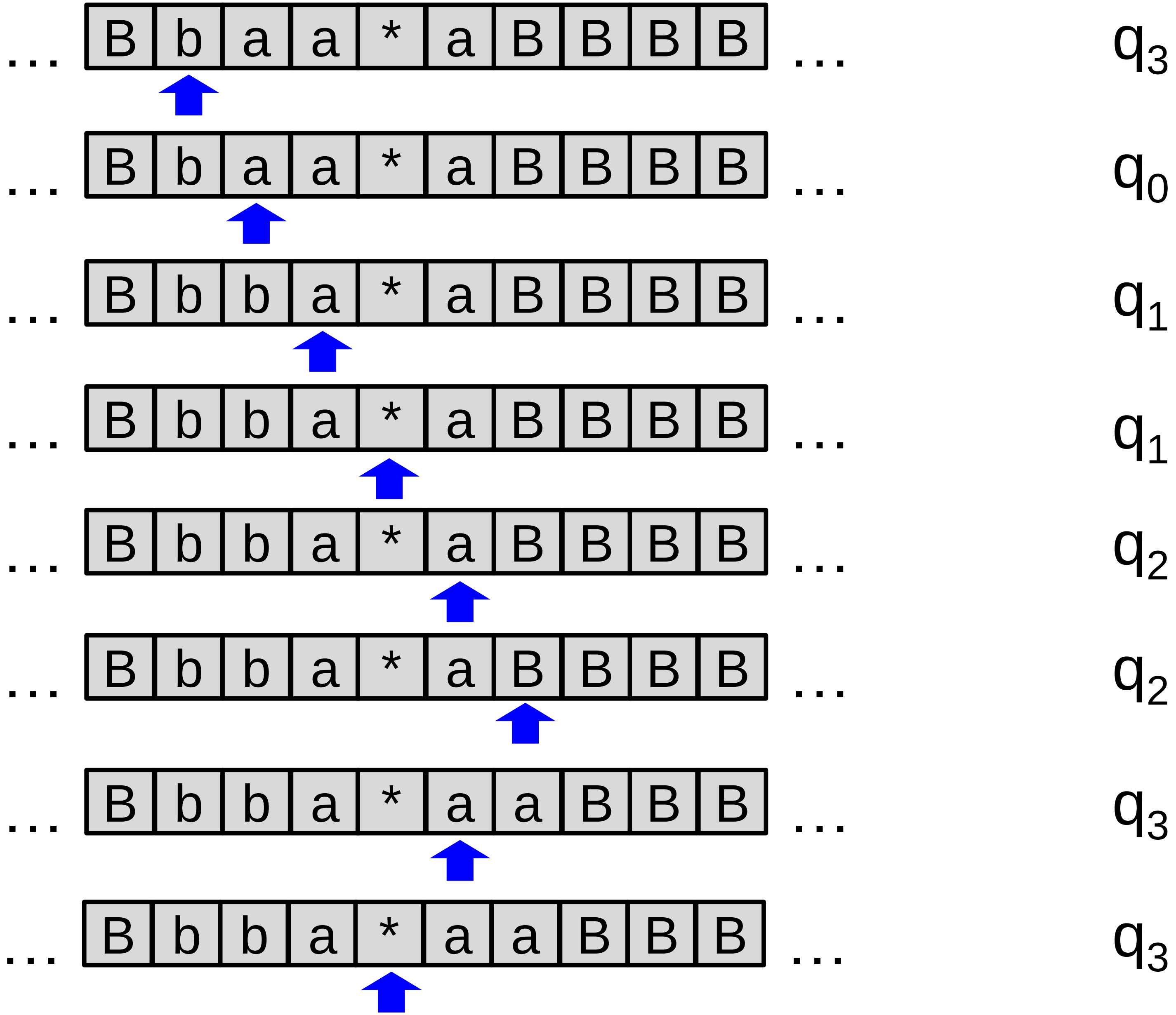
Another example

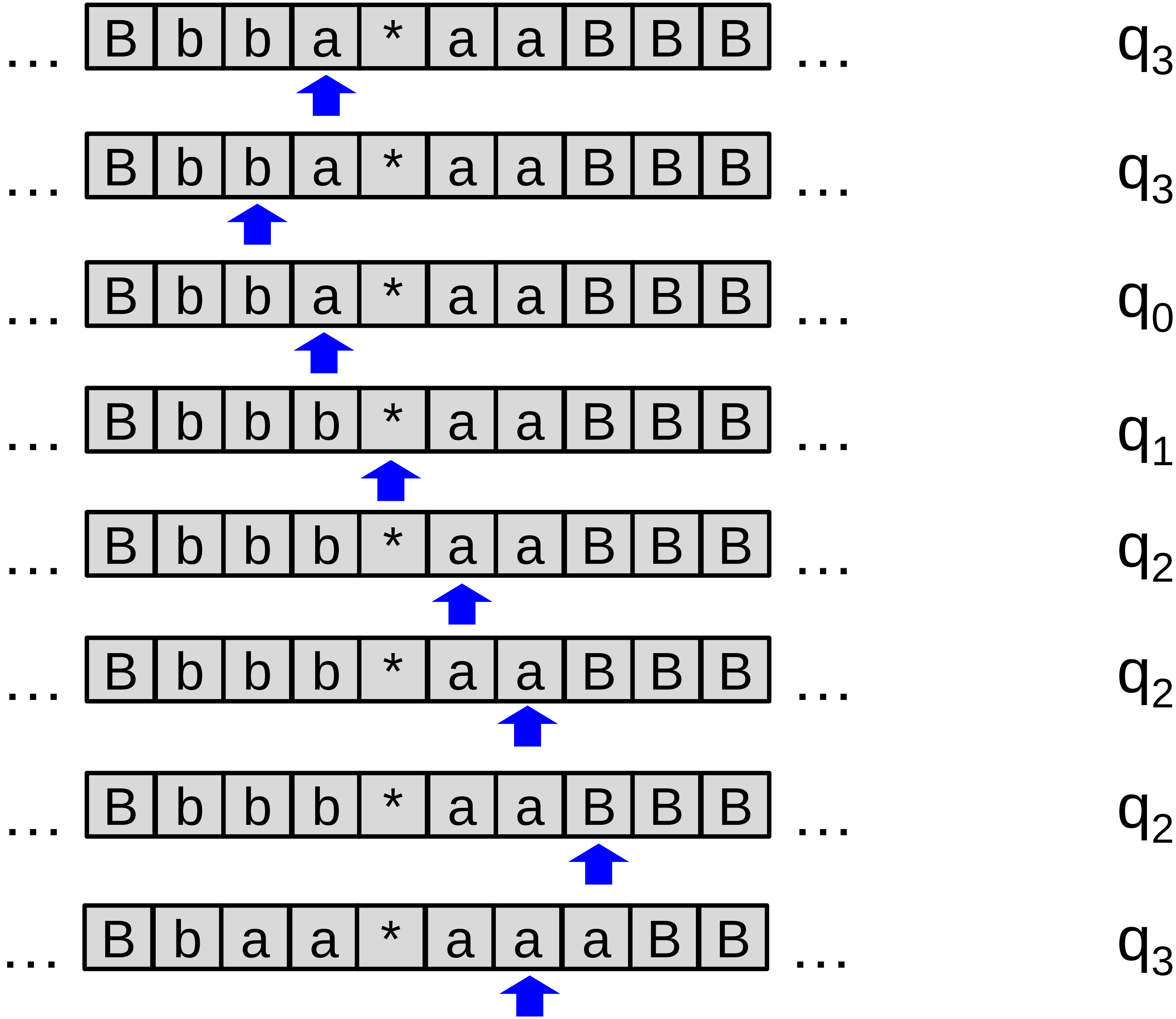


What does it do?

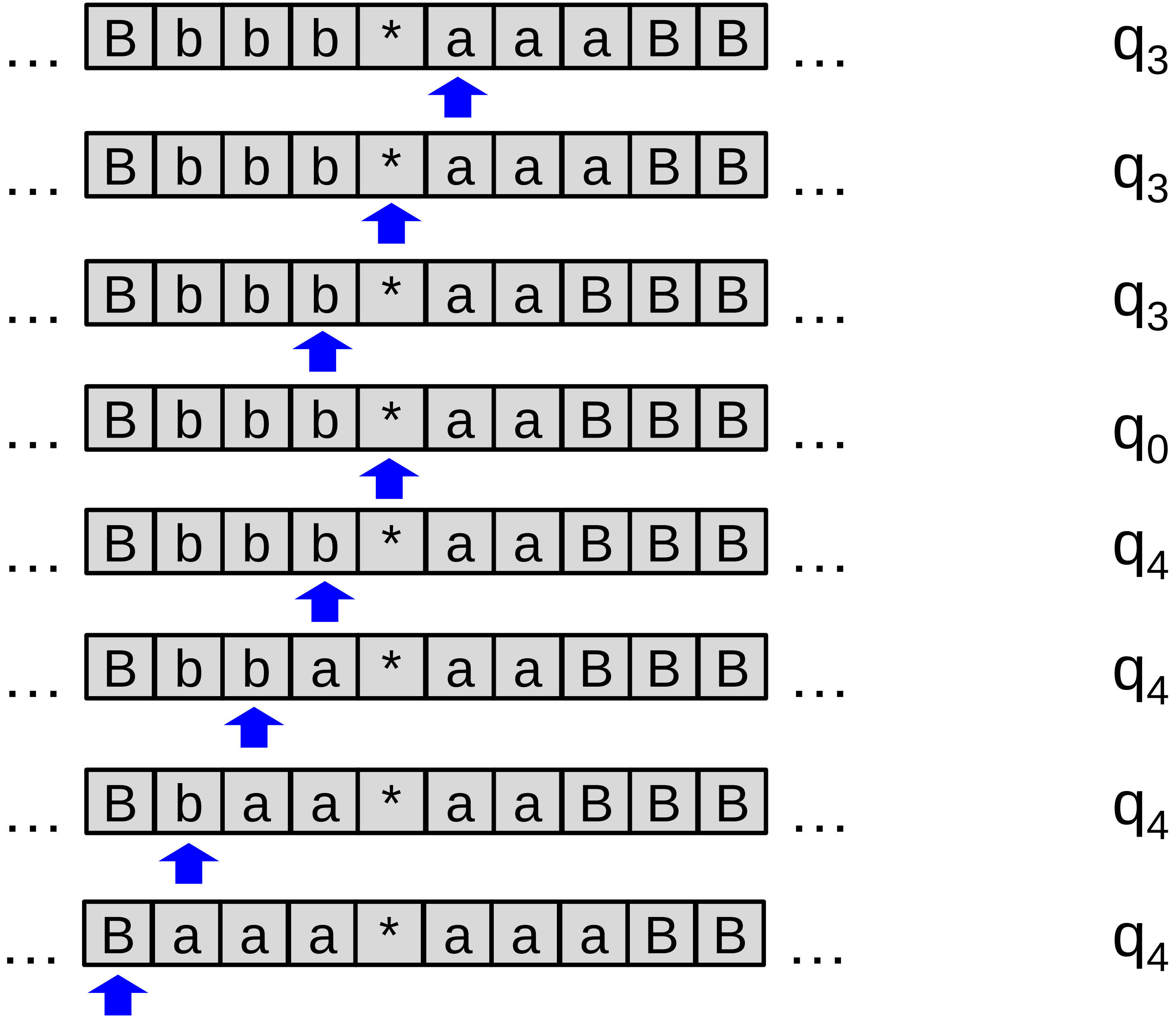


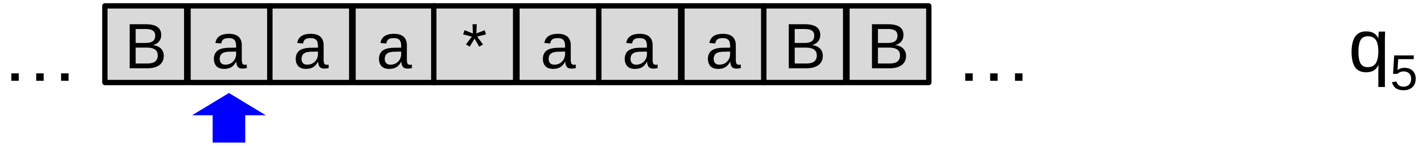
… B a a a * B B B B B … q_0
… B b a a * B B B B B … q_1
… B b a a * B B B B B … q_1
… B b a a * B B B B B … q_1
… B b a a * B B B B B … q_2
… B b a a * a B B B B … q_3
… B b a a * a B B B B … q_3
… B b a a * a B B B B … q_3


… B b a a * a B B B B … q3
… B b a a * a B B B B … q0
… B b b a * a B B B B … q1
… B b b a * a B B B B … q1
… B b b a * a B B B B … q2
… B b b a * a B B B B … q2
… B b b a * a a B B B … q3
… B b b a * a a B B B … q3



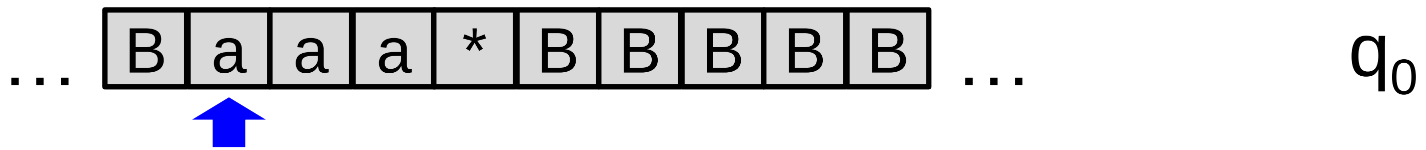
… B b b a * a a B B B … q3
… B b b a * a a B B B … q3
… B b b a * a a B B B … q0
… B b b b * a a B B B … q1
… B b b b * a a B B B … q2
… B b b b * a a B B B … q2
… B b b b * a a B B B … q2
… B b a a * a a a B B … q3


… B b b b * a a a B B … q3
… B b b b * a a a B B … q3
… B b b b * a a B B B … q3
… B b b b * a a B B B … q0
… B b b b * a a B B B … q4
… B b b a * a a B B B … q4
… B b a a * a a B B B … q4
… B a a a * a a a B B … q4



Final State

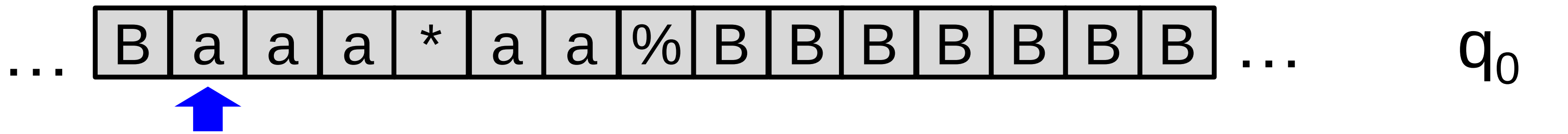
Initial State was:



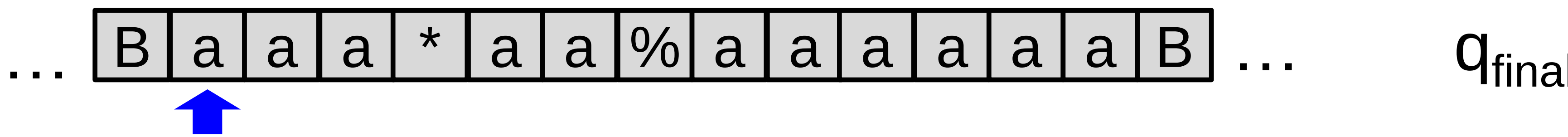
- The machine makes a copy of n a’s and puts them behind the *

Multiplication

- We can use this machine to do “unary multiplication”:



- Multiply “number” before * by “number” between * and % (3 times 2 here):



- Can be done by adapting the discussed machine and using it repeatedly

Church/Turing Thesis

- Every computable function can be computed by a Turing Machine
- I.e.: Turing Machines are universal computing machines
- Every problem that can be solved by an algorithm can be solved by a Turing machine
- Where is the power coming from?

  The read/write input/output tape !

More about TM

- The tape can be used to record any data for later access
- There is always space available after last non-blank location
- There is no limit how often the tape is accessed
- Your PC is less powerful than a TM – why? Because it has finite memory

Efficiency

- TM are universal but not efficient
- Progress can be really slow
- Looking up memory involves sequential access – the opposite of efficiency

Managing complexity

- One can encapsulate useful functionality in “separate” sub-routines
- Collection of states set aside for each subroutine
- (similar to structured programming approach)
- However, TM are mainly useful as a theoretical concept, not for solving real world problems!

Variations

- There are common variants of TM:
  - Multiple tapes
  - Single-side infinite tape
  - Non-deterministic TM
- It can be shown that these have all equivalent power to the TM discussed here.

Example: Non-deterministic TM

- To simulate non-deterministic TM:
  - 3 tapes:
  - One tape for original input
  - One tape for the choice sequence: (2,3,1,2)
  - One tape to run on current choice sequence
- For this to work we need to enumerate all possible sequences of choices (ok, as states are finite)

Another equivalence

- The “2 pushdown” automaton is equivalent to the Turing Machine:
  - One pushdown holds tape contents to left of tape head
  - One pushdown holds tape contents to the right of tape head
  - As tape head moves, symbols shift across from one pushdown to another

More generally …

- Chomsky Hierarchy (for language classes):
  - Type 0: Languages accepted by Turing Machines
  - Type 1: Languages accepted by Turing Machines with linear bounded storage
  - Type 2: Languages accepted by Pushdown Automata
  - Type 3: Languages accepted by Finite State Automata

Alternative Characterization

- Equivalent grammar formalisms:
  - Type 0: Languages generated by unrestricted grammars
  - Type 1: Languages generated by context-sensitive grammars
  - Type 2: Languages generated by context-free grammars
  - Type 3: Languages generated by regular grammars

Equivalence and Inclusions