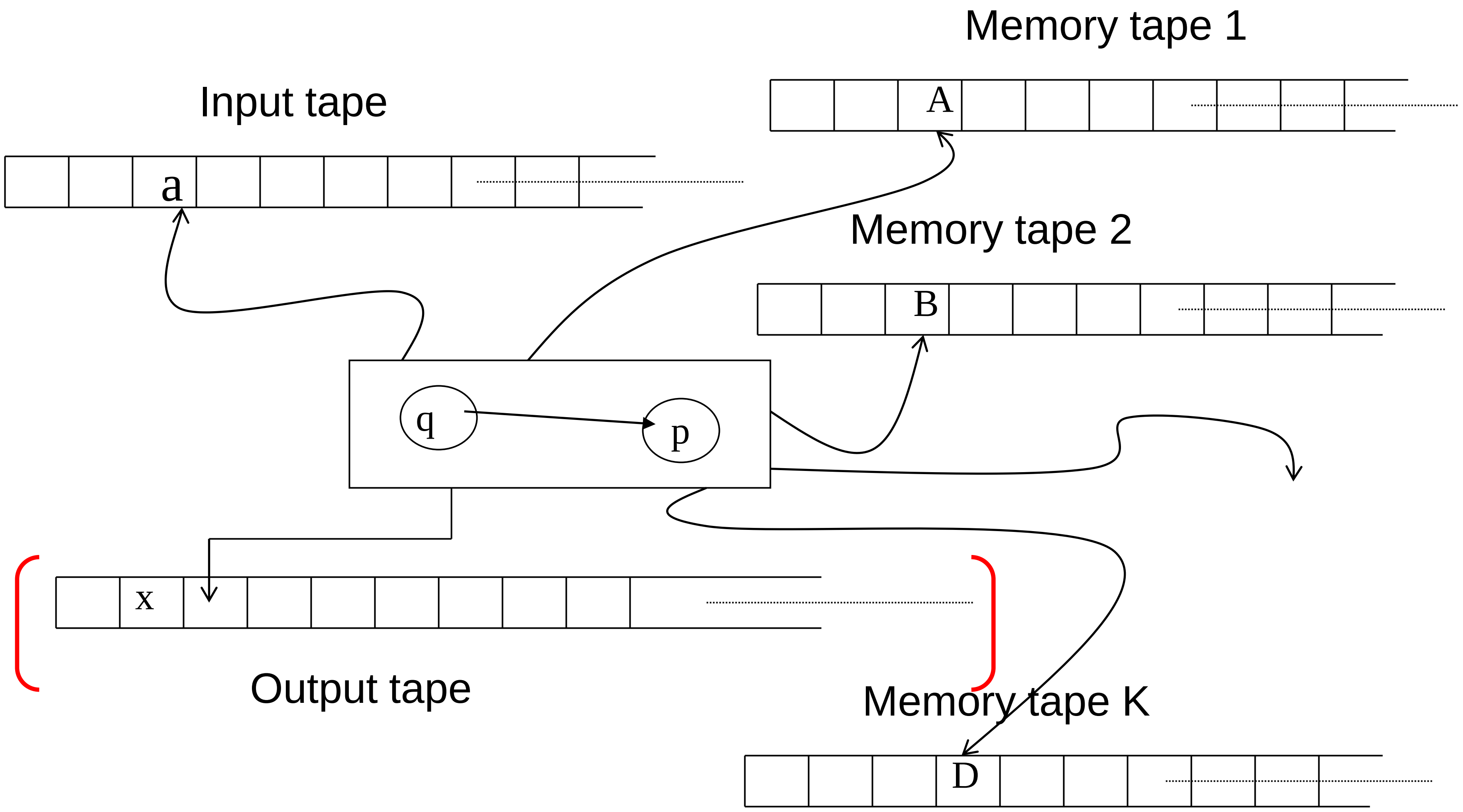
Theoretical Computer Science

Turing Machines

Lecture 10 - Manuel Mazzara

Turing Machines



Informal description

- States and alphabet are as in the other automata
  - Input
  - Output
  - Control device
  - Memory alphabet
- Tapes are represented as infinite cell sequences with a special “blank” symbol (or barred ‘b’ or ‘_’ or ‘-’)
  - Tapes contain only a finite number of non-blank symbols

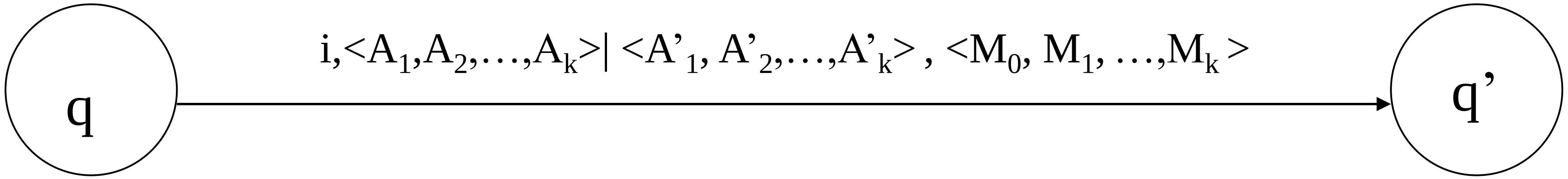
Moves

- Moves are based on
  - one symbol read from the input tape
  - K symbols, one for each memory tape
  - state of the control device
- Actions
  - Change state
  - Write a symbol replacing the one read on each memory tape
  - Move the K+1 heads

Moves of the heads

- Memory and input scanning heads can be “moved” in three ways
  - One position right (R)
  - One position left (L)
  - Stand still (S)
- The direction of each head must be specified explicitly

Graphically

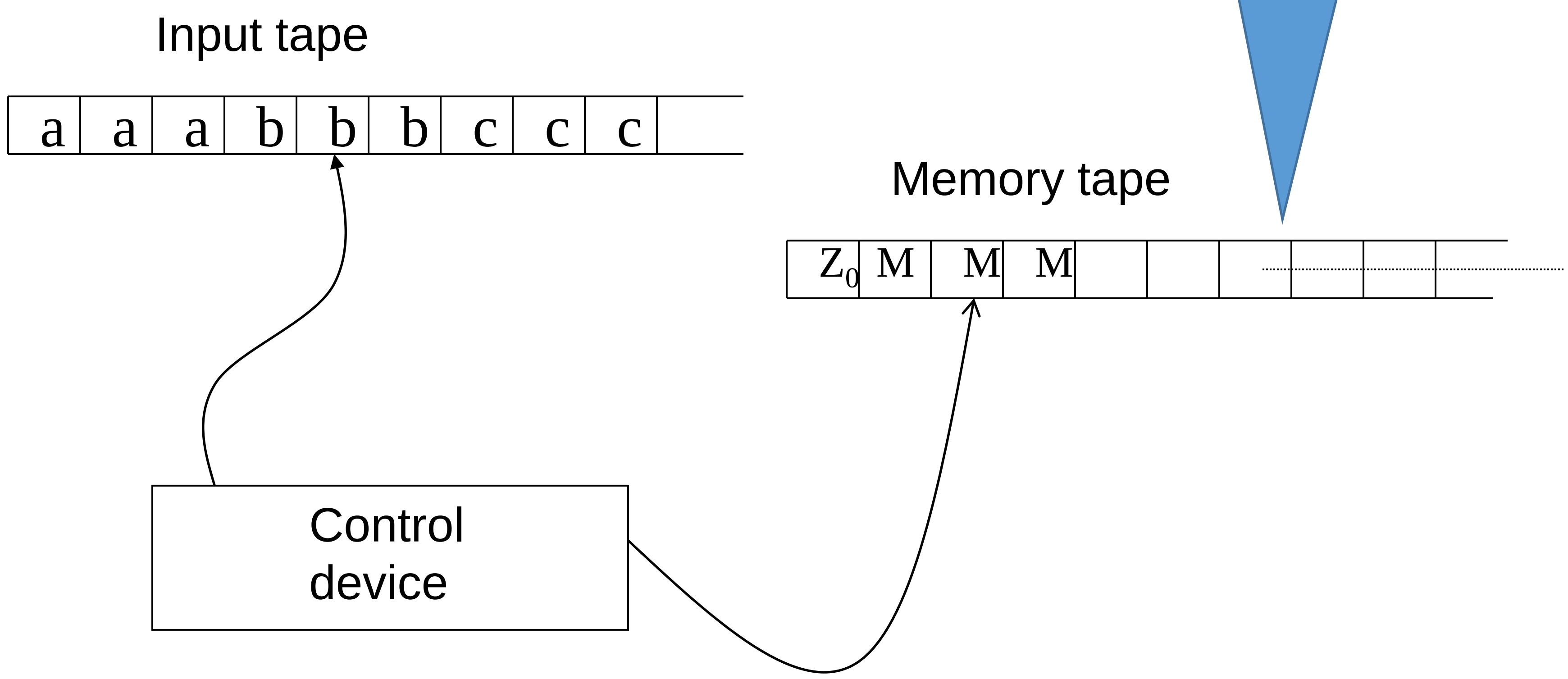


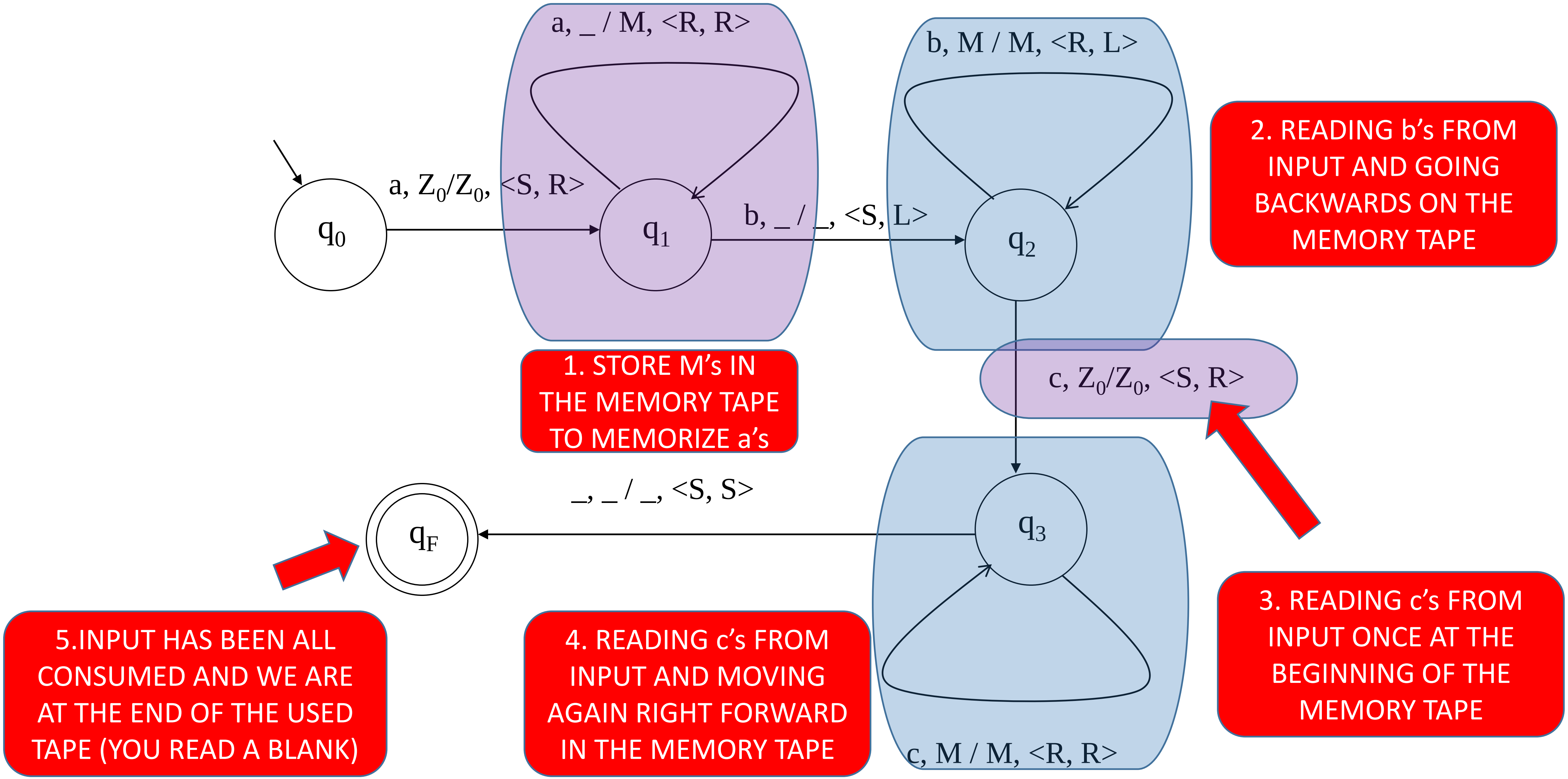
- ‘i’ is the input symbol
- A_j is the symbol read from the j^{th} memory tape
- A_j’ is the symbol replacing A_j
- M_0 is the direction of the head of the input tape
- M_j ($1 \leq j \leq k$) is the direction of the head of the j^{th} memory tape

Example 1

$L=\{a^n b^n c^n \mid n > 0\}$

PDAs have a <u>destructive</u> external memory
TMs have <u>persistent</u> memory tapes




a, _ / M, <R, R>
b, M / M, <R, L>
a, Z0/Z0, <S, R>
b, _ / _, <S, L>
q0
q1
q2
q3
qF
2. READING b's FROM INPUT AND GOING BACKWARDS ON THE MEMORY TAPE
1. STORE M's IN THE MEMORY TAPE TO MEMORIZE a's
c, Z0/Z0, <S, R>
_, _ / _, <S, S>
3. READING c's FROM INPUT ONCE AT THE BEGINNING OF THE MEMORY TAPE
5.INPUT HAS BEEN ALL CONSUMED AND WE ARE AT THE END OF THE USED TAPE (YOU READ A BLANK)
4. READING c's FROM INPUT AND MOVING AGAIN RIGHT FORWARD IN THE MEMORY TAPE
c, M / M, <R, R>

Formally

- A TM with K tapes is a tuple of 7 elements $M=<Q,I,\Gamma,\delta,q_0,Z_0,F>$
  - Q is a <u>finite set of states</u>
  - I is the <u>input alphabet</u>
  - Γ is the <u>memory alphabet</u>
  - δ is the <u>transition function</u>
  - $q_0 \in Q$ is the <u>initial state</u>
  - $Z_0 \in \Gamma$ is the <u>initial memory symbol</u>
  - $F \subseteq Q$ is the <u>set of final states</u>

Transition function

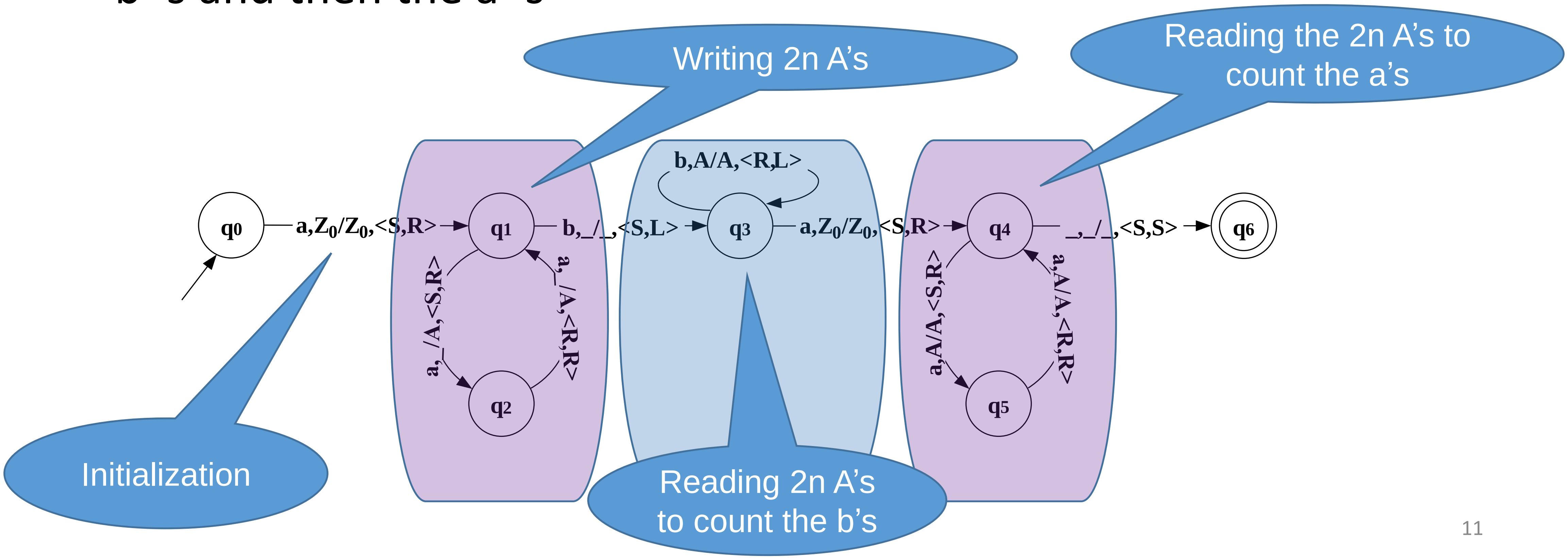
- The transition function is defined as

  δ: (Q-F) $\times$ I $\times$ Γ^k $\rightarrow$ Q $\times$ Γ^k $\times$ $\{R,L,S\}^{k+1}$
- Remarks
  - The function can be partial
  - No transitions outgoing from the final states

Example 2

$L = \{a^n b^{2n} a^n \mid n \geq 1\}$

– We write 2n A's on a memory tape and we use them to check the b's and then the a's



Configuration, Informally

- A configuration of a TM is a **snapshot of the machine**
- A configuration should include:
  – state of the control device
  – string on the input tape and the position of the head
  – string and position of the head for each memory tape

Definition

A <u>configuration</u> *c* of a TM with K memory tapes is the following (K+2)-tuple:

$$c=<q,\ x\uparrow iy,\ \alpha_1\uparrow A_1\beta_1,\ \ldots,\ \alpha_K\uparrow A_K\beta_K>$$

where

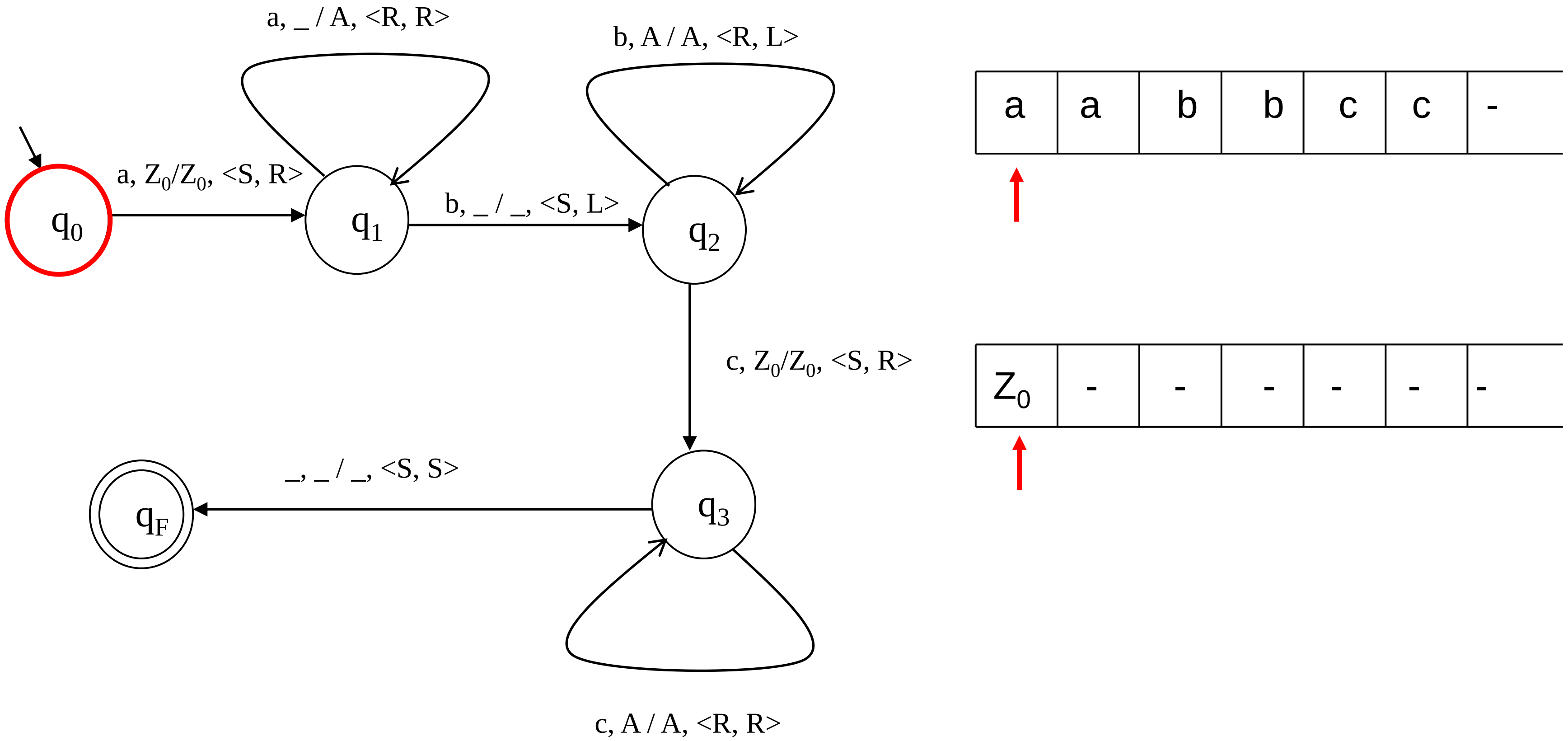
- $q \in Q$
- $x,y \in I^*$, $i \in I$
- $\alpha_r, \beta_r \in \Gamma^*$, $A_r \in \Gamma$ $\forall r$ $1 \le r \le K$
- $\uparrow \notin I \cup \Gamma$

Initial configuration

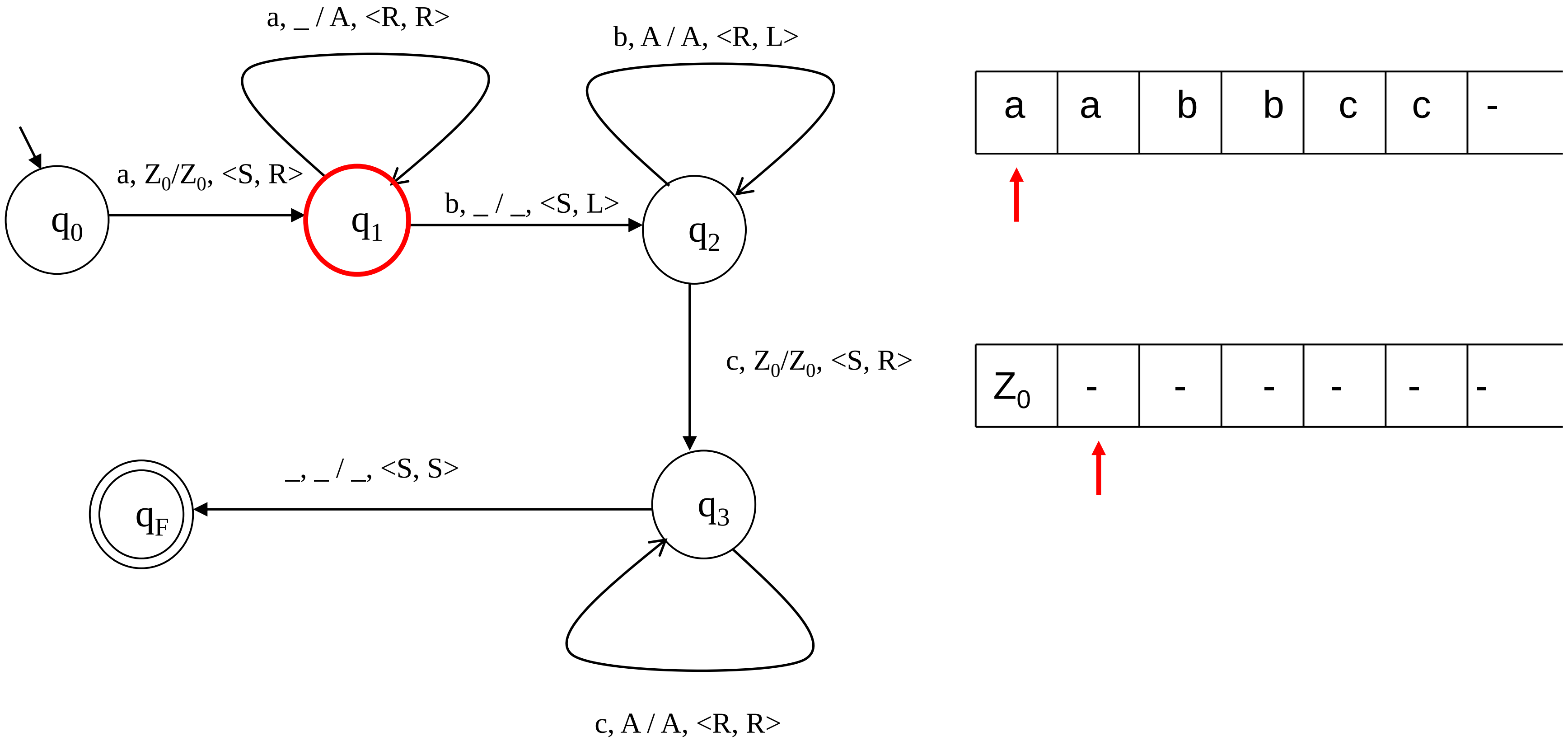
$$c_0=\langle q_0, \uparrow iy, \uparrow Z_0, \ldots, \uparrow Z_0\rangle$$

- Formally:
  - $x=\varepsilon$
  - $\alpha_r,\beta_r=\varepsilon$, $A_r=Z_0\ \forall r\ 1\leq r\leq K$
- Informally:
  - The control device is in the initial state
  - All the heads are at the beginning of the corresponding tape

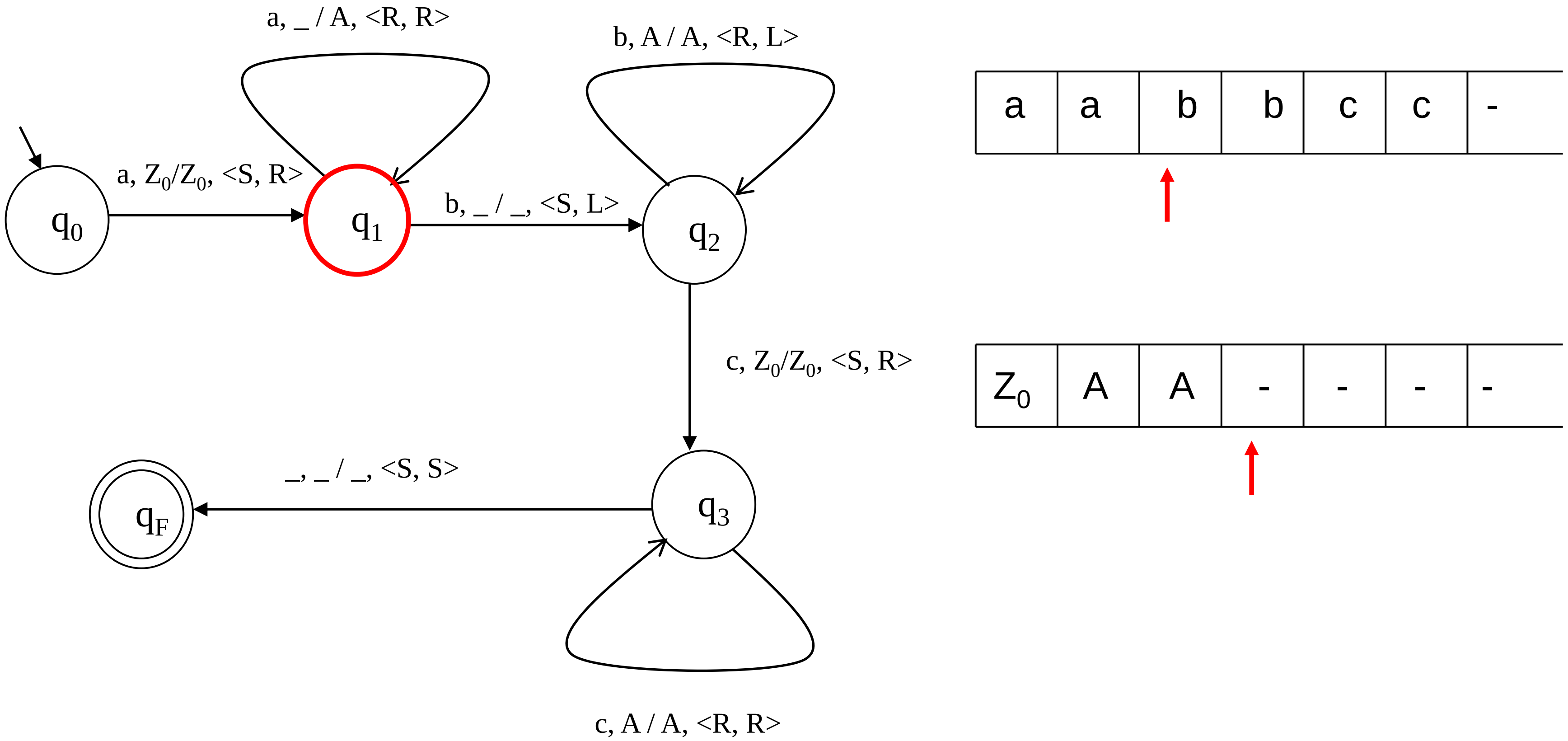
Example



$c = \langle q_0, \uparrow aabbcc, \uparrow Z_0 \rangle$



$$c=<q_1, \uparrow aabbcc, Z_0\uparrow>$$



$$c=\langle q_1,\ aa\uparrow bbcc,\ Z_0AA\uparrow\rangle$$

Acceptance condition (1)

- A string $x \in I^*$ is accepted by a TM M with K memory tapes if and only if:

$$\mathbf{c_0 = \langle q_0, \uparrow x, \uparrow Z_0, \ldots, \uparrow Z_0 \rangle \quad |\text{-}^*\text{-}_M \; c_F = \langle q, x' \uparrow iy, \alpha_1 \uparrow A_1 \beta_1, \ldots, \alpha_K \uparrow A_K \beta_K \rangle}$$

with $q \in F$ (and $x = x'iy$)

- c_F is called **final configuration**

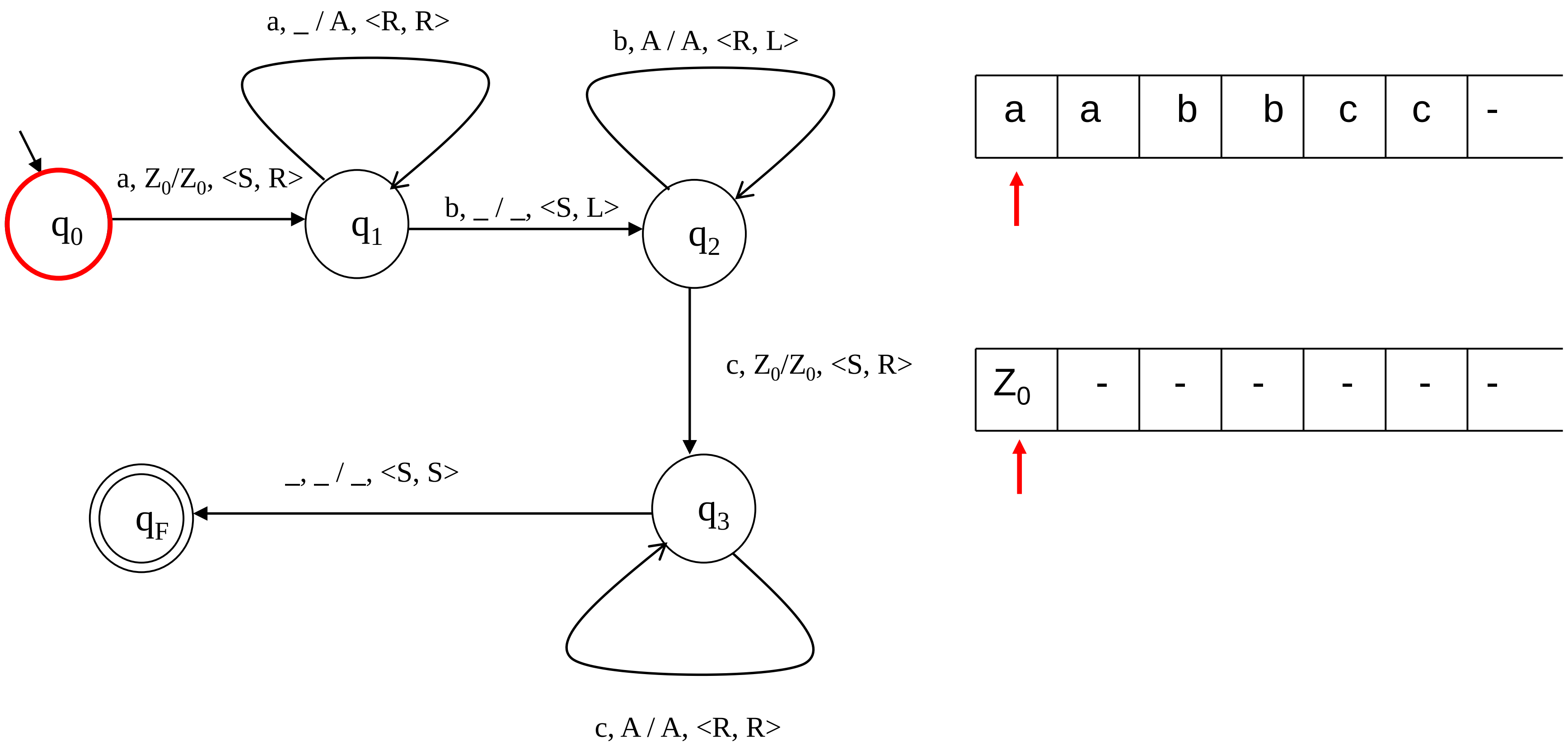
- $|\text{-}^*\text{-}_M$ is the reflexive transitive closure of the $|\text{--}_M$ relation

Acceptance condition (2)

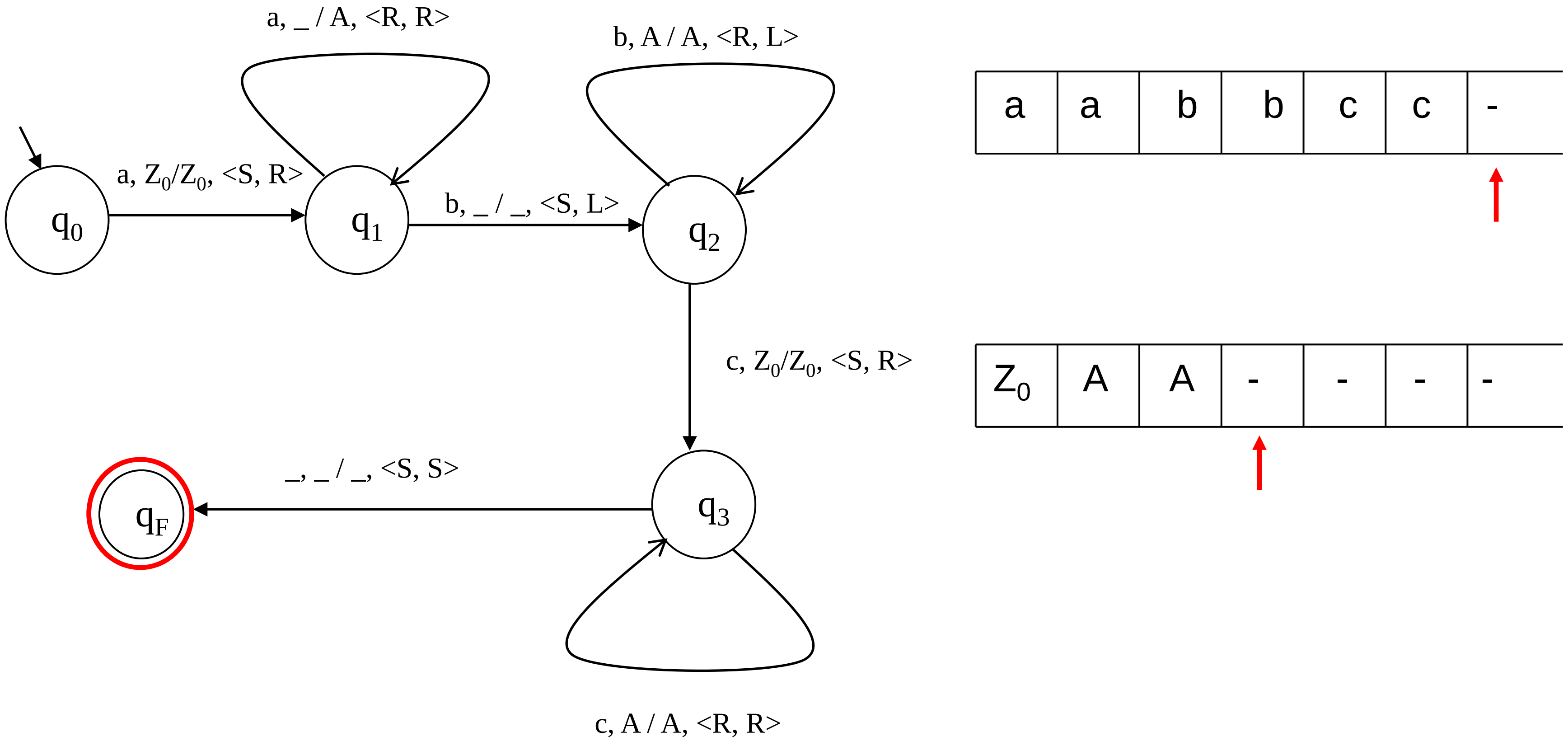
- The initial tape content is said to be accepted by M if it **eventually halts in a final state**

- L(M)={x| x$\in$I* and x is accepted by M}

Example



c_0=<q_0, ↑aabbcc, ↑Z_0>



$$c_0=<q_0,\ \uparrow aabbcc,\ \uparrow Z_0>$$

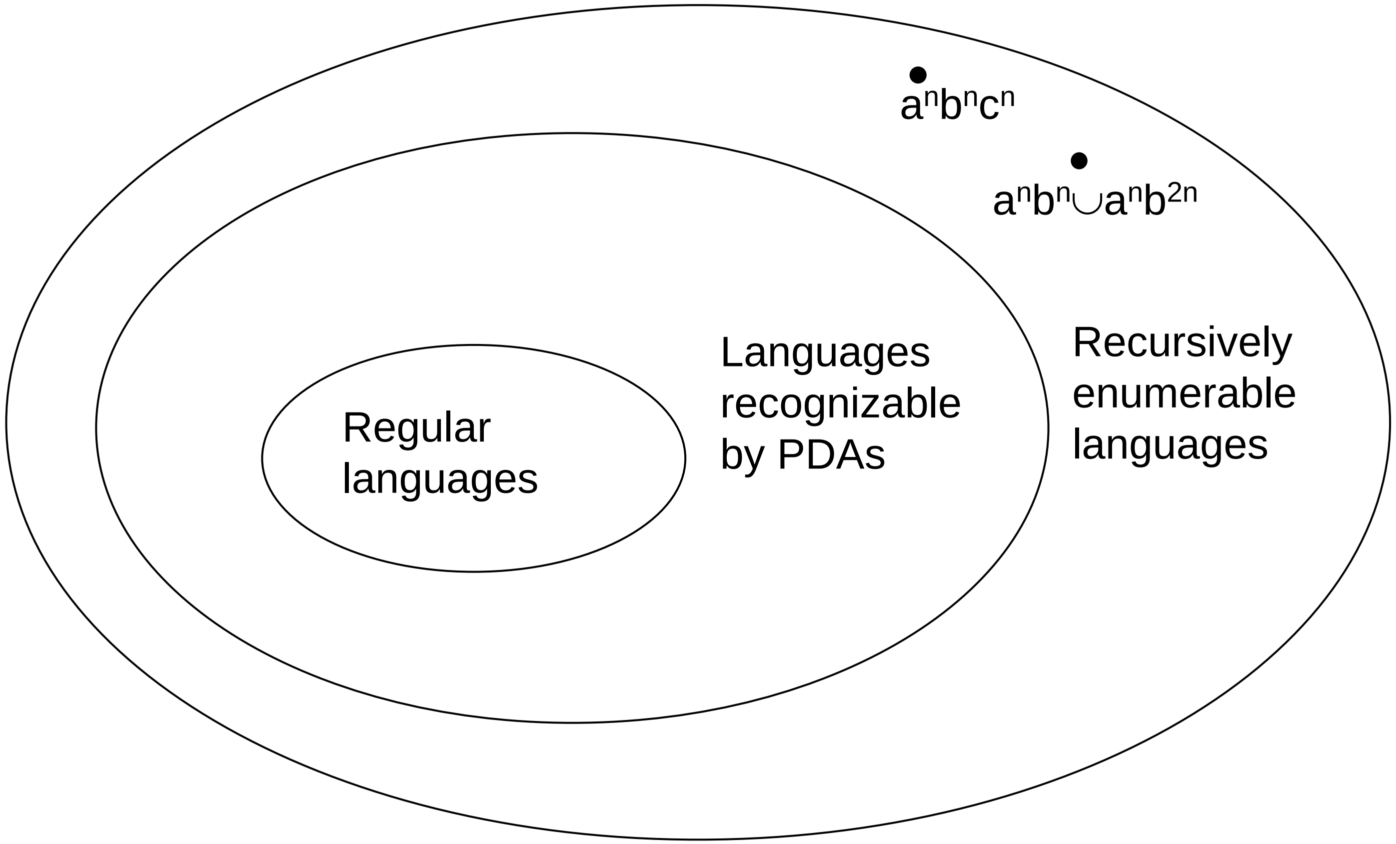
$$c_F=<q_F,\ aabbcc\uparrow,\ Z_0AA\uparrow>$$

TM vs PDA

- **$a^n b^n c^n$ or $a^n b^n \cup a^n b^{2n}$ cannot be recognized by any PDA**
- **They can be recognized by a TM**
  - We have seen a TM for $a^n b^n c^n$
- Every language recognizable by a PDA can be recognized by a TM
  - A TM can always be built to use (one of) its memory tape(s) as a stack
- The languages accepted by TMs are called **recursively enumerable**

Languages

- TMs have a higher expressive power than PDAs



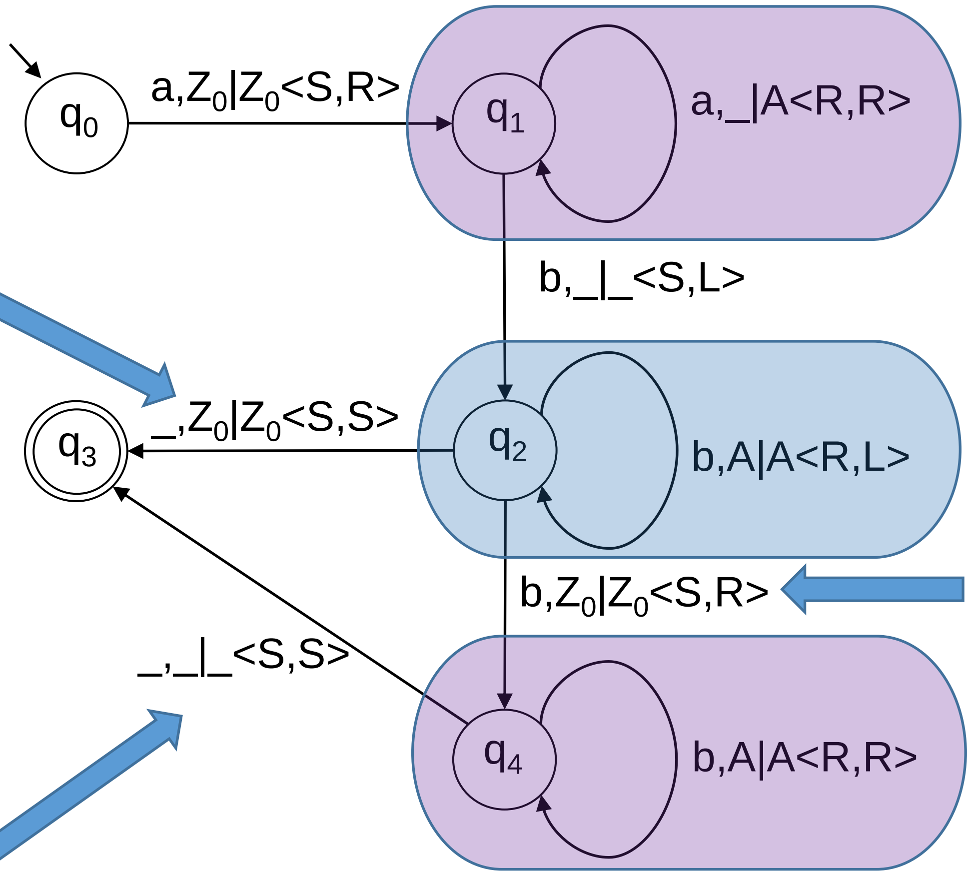
Example 3: $a^n b^n \cup a^n b^{2n}$

1. STORE A's IN THE MEMORY TAPE

2. COUNT THE FIRST n B's

3a. YOU REACHED THE BEGINNING OF THE TAPE AND THE END OF INPUT: $a^n b^n$

3b. YOU REACHED THE BEGINNING OF THE TAPE BUT THERE IS MORE INPUT, YOU CAN NAVIGATE THE TAPE AGAIN RIGHTWARDS TO COUNT n MORE B's : VERIFY IF IT IS $a^n b^{2n}$



4. YOU REACHED THE END OF BOTH MEMORY AND INPUT TAPE: $a^n b^{2n}$

Operations on TMs (1)

- TMs are closed under
  - Intersection
  - Union
  - Concatenation
  - Kleene star
- **TMs are not closed under complement**
  - They are not closed under difference either (why?)
  - $A \setminus B = A \cap C(B)$

Operations on TM (2)

- Closure for union, intersection, … : positive answer
- A TM can easily simulate two other TMs
  - In series or
  - In parallel

… and this explains the closure

Complement

- Are **loop-free TMs** closed under complement?
  - **Yes**: it suffices to define the set of halting states and partition it into accepting and non-accepting states

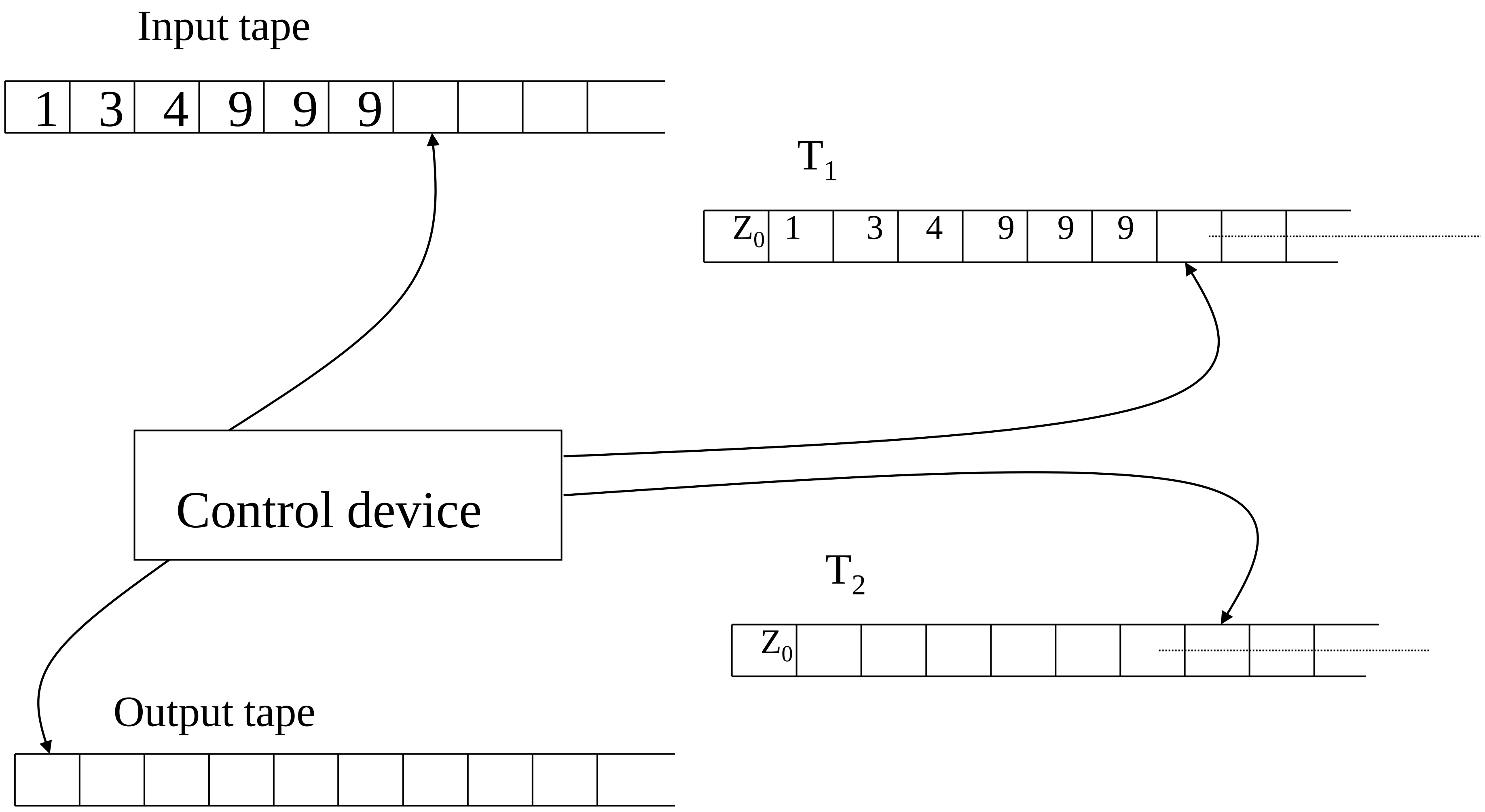
- Problems arise from **nonterminating** computations (to be discussed later)

How do we use TMs?

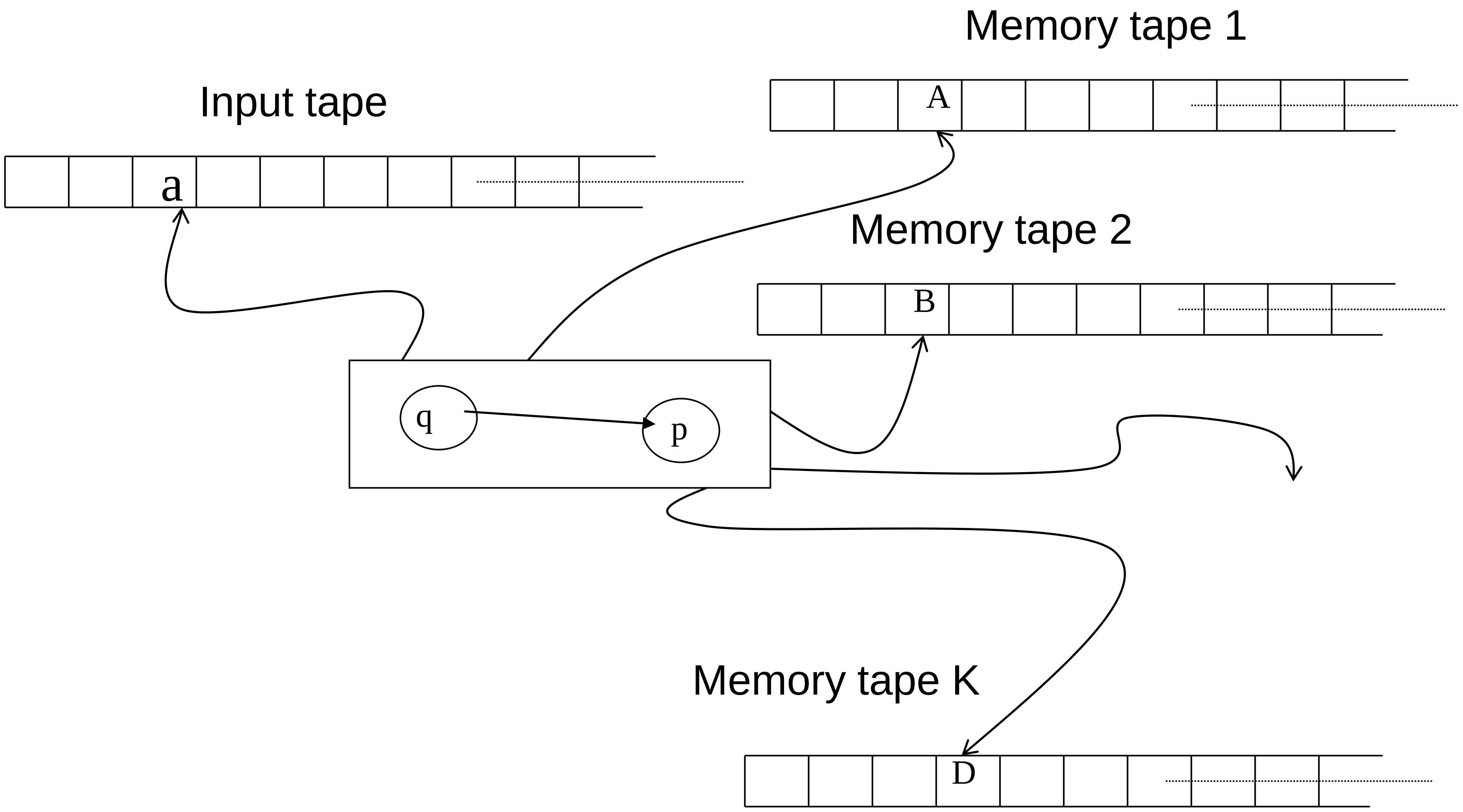
- TMs can
  - Recognize languages (Acceptors)
  - Translate accepted languages (Transducers)
- … but also compute functions
  - They are equivalent to VNMs

- A TM is an <u>**abstract model of "computer"**</u> with sequential memory access

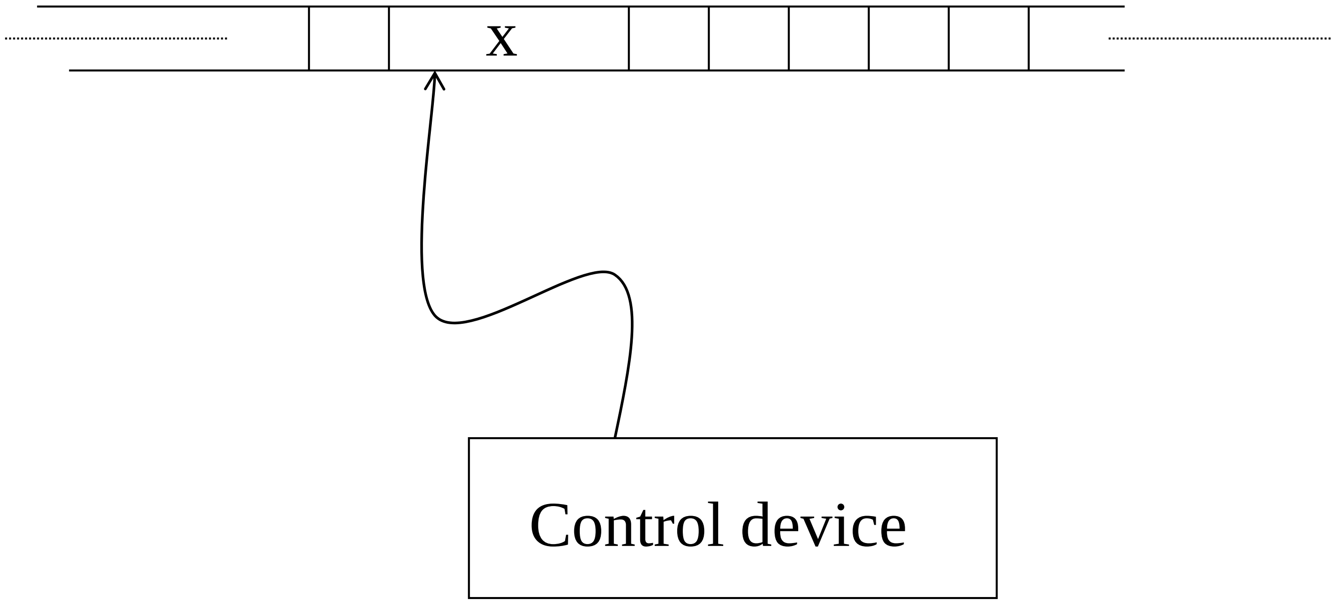
Mechanical view



TM generic model



Single tape TM

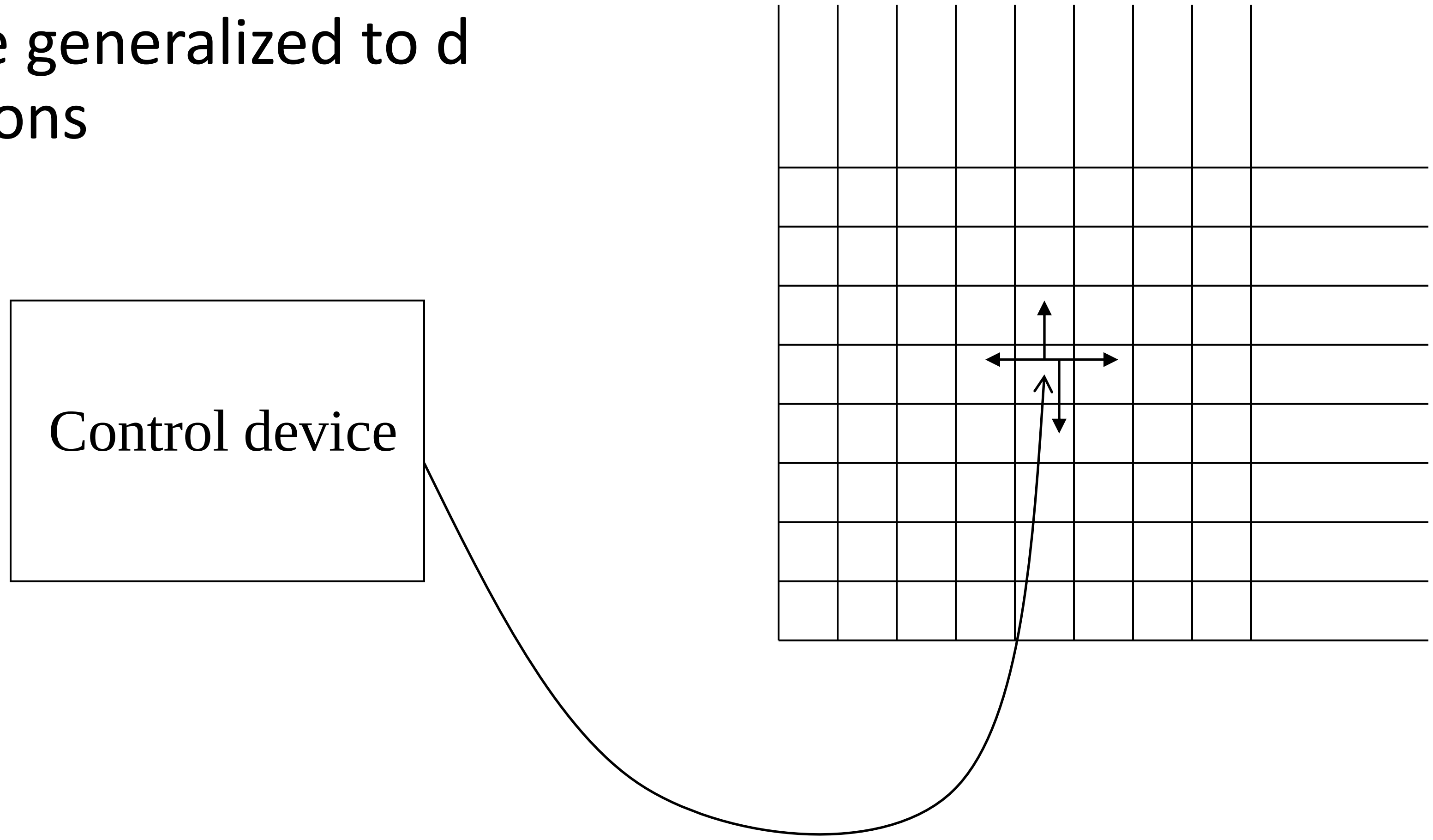


Single tape

- It can be unlimited in both directions
- it serves as input, memory and output tape

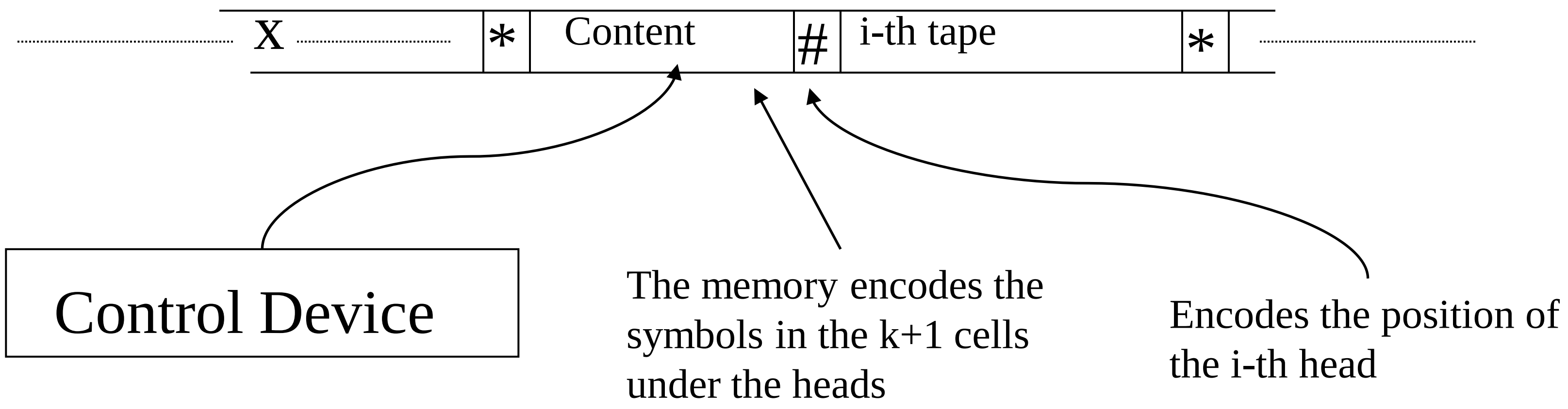
Bidimensional tape TM

- A head for each dimension
- It can be generalized to d dimensions



Relation among different models

- Both single and multi-tape TMs can be equipped with d-dimensional tapes
- <u>**All these TM models are equivalent**</u>
  - They can recognize the same class of languages



Theoretical Computer Science

Recap and spoilers

Lecture 10 - Manuel Mazzara

Our itinerary

- **Automata Theory and Transducer Theory**

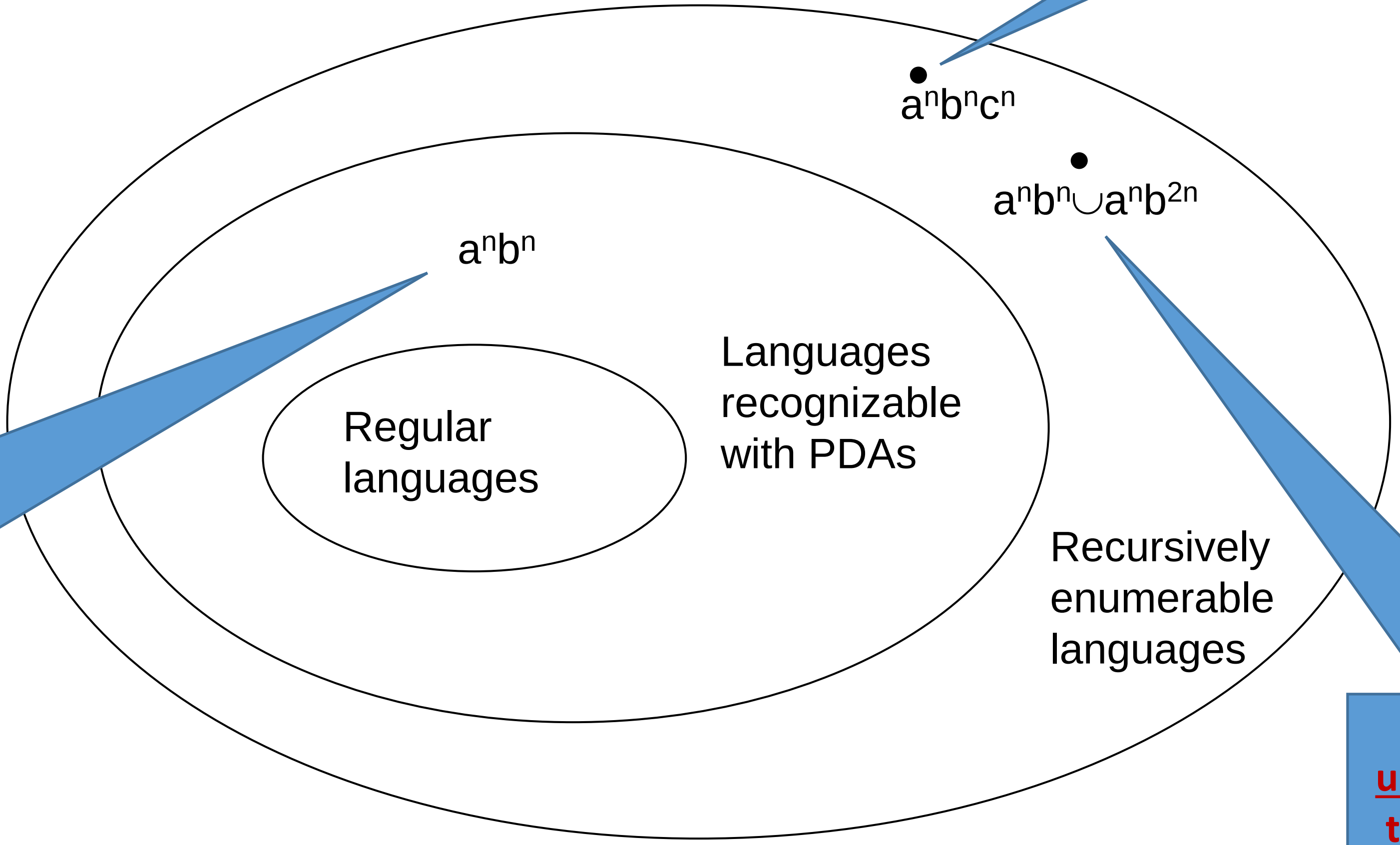
- **Deterministic automata (<u>done</u>)**
  - FSA
  - PDA
  - TM

- Nondeterministic Automata (we will do now)
  - FSA
  - TM
  - **PDA**

Memory Model

- The choice of a **memory model** influences the expressiveness of a computational model
- FSA have a fixed memory: **finite and fixed** number of state
  - You cannot count “more” than the number of states allows
- Stack has an extensible memory, but **destructive**
  - Once a symbol is read, it is destroyed
  - You can count variable *ns,* but when you consume the symbol on stack, they are gone
- TMs have **persistent memory tapes**
  - They behave like the **von Neumann architecture**

Languages



PDAs have a <u>destructive</u> external memory

FSA can only count fixed numbers (finite states), no generic *n*

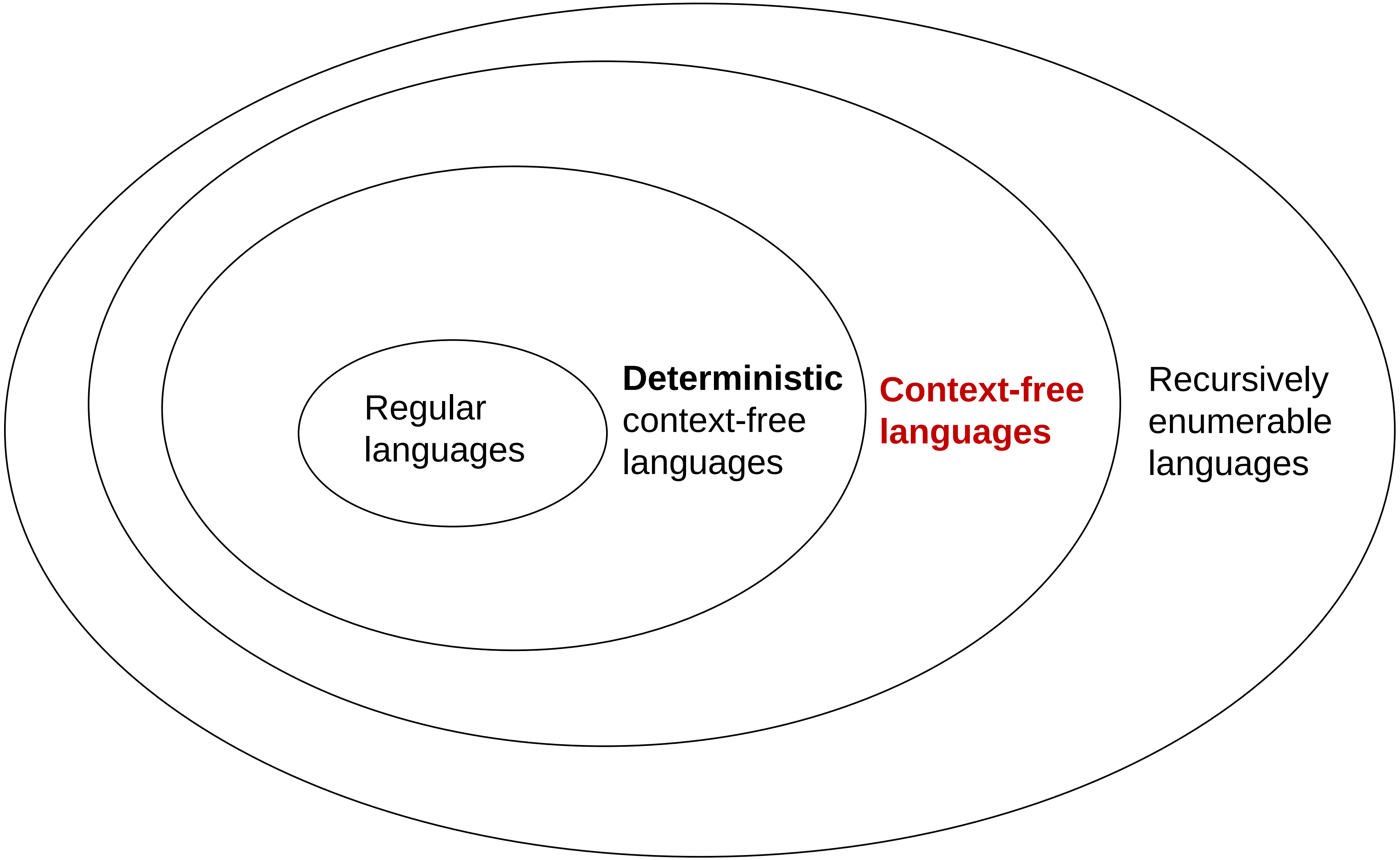
In programming languages we need to syntactically check <u>nested structures</u>

PDAs are <u>not closed under union</u>: stack needs to be prepared for what we are looking for

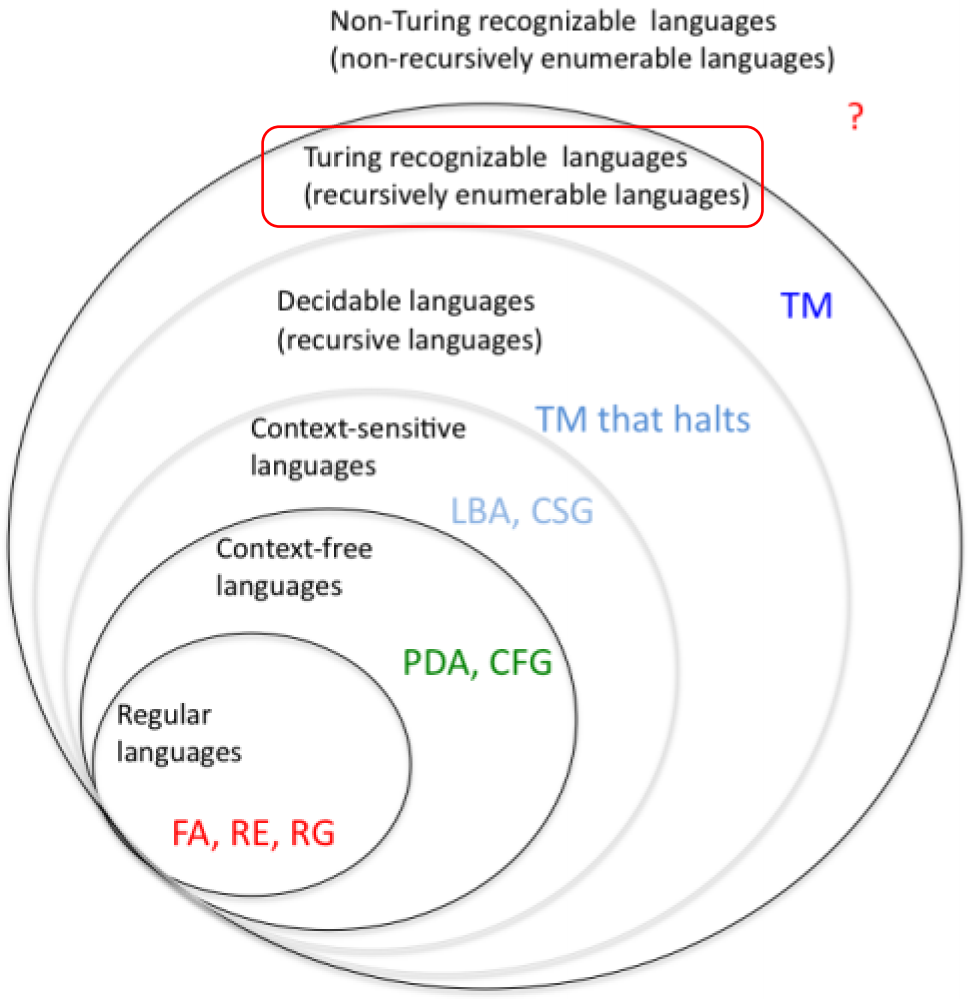
Nondeterminism: spoiler!

- **DFSA and NFSA have the same expressive power**
- **DTM and NTM have the same expressive power**
- What about NPDA?
  - Deterministic vs nondeterministic context-free languages

Deterministic vs nondeterministic CFL



The bigger picture



FA: finite state automaton
RE: regular expression
RG: regular grammar

CFG: context-free grammar
PDA: pushdown automaton

LBA: linear-bounded automaton
CSG: context-sensitive grammar
TM: Turing machine

Recursively Enumerable Languages: spoiler

- The set of C program-input pairs that **do not** run into an infinite loop is **recursive**

- The set of C programs that contain an infinite loop **is recursively enumerable**

- We will see the details of this

Theory of Computation

Nondeterministic FSA

Lecture 10 - Manuel Mazzara

Nondeterministic models (1)

- Usually, one thinks of an algorithm as a **determined sequence of operations**
  - In a certain **state** with a certain **input** there is no doubt on the next step

- Example: let us compare

```
if x>y then max:=x else max:=y
```

with

```
if
      x>=y then max:=x
      x<=y then max:=y
fi
```

Nondeterministic models (2)

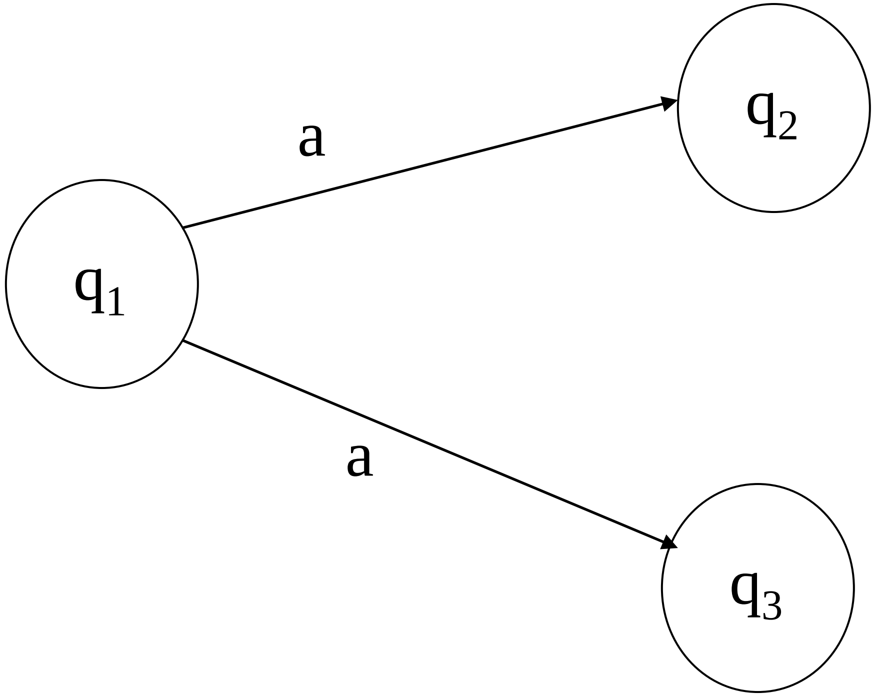
- Let us consider the **`case`** construct of Pascal
- Imagine if we have something like the following:

```
case
      x=y    then S1
      z>y+3  then S2
      …      then
endcase
```

Nondeterministic models (3)

- Nondeterminism (ND) is a model of computation of **parallel computing**
- **Abstraction to describe algorithms**
- It can be applied to various computational models
- ND models must not be confused with stochastic models

Adding nondeterminsm



$$\delta(q_1,a)= \{q_2, q_3\}$$

Nondeterministic FSA

- A nondeterministic FSA (NDFSA) is a tuple

<Q, I, δ , q_0, F>, where

  - Q, I, q_0, F are defined as in (D)FSAs
  - δ: Q×I $\rightarrow \mathcal{P}$(Q)

A set of states

- What happens to δ^*?

Move sequence

- δ^* is **inductively** defined from δ

$$\delta^*(q, \varepsilon) = \{q\}$$

$$\delta^*(q, y.i) = \bigcup_{q' \in \delta^*(q,y)} \delta(q', i)$$

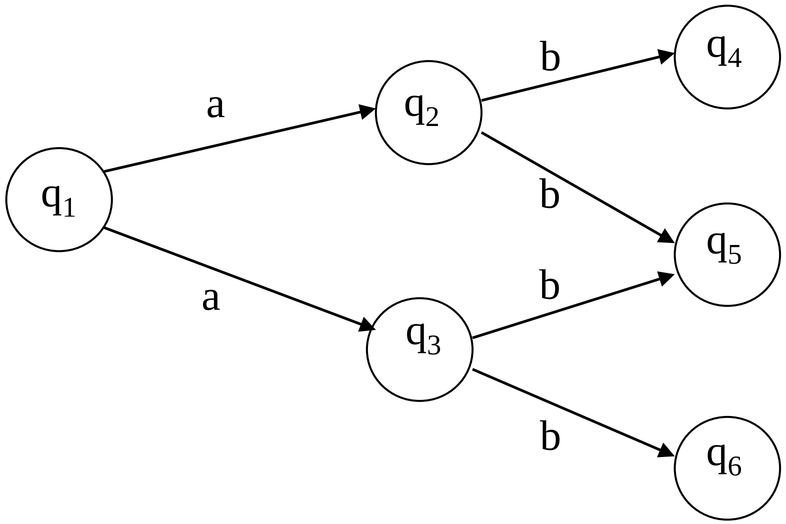
- Example:

  $\delta(q_1,a) = \{q_2, q_3\}$,

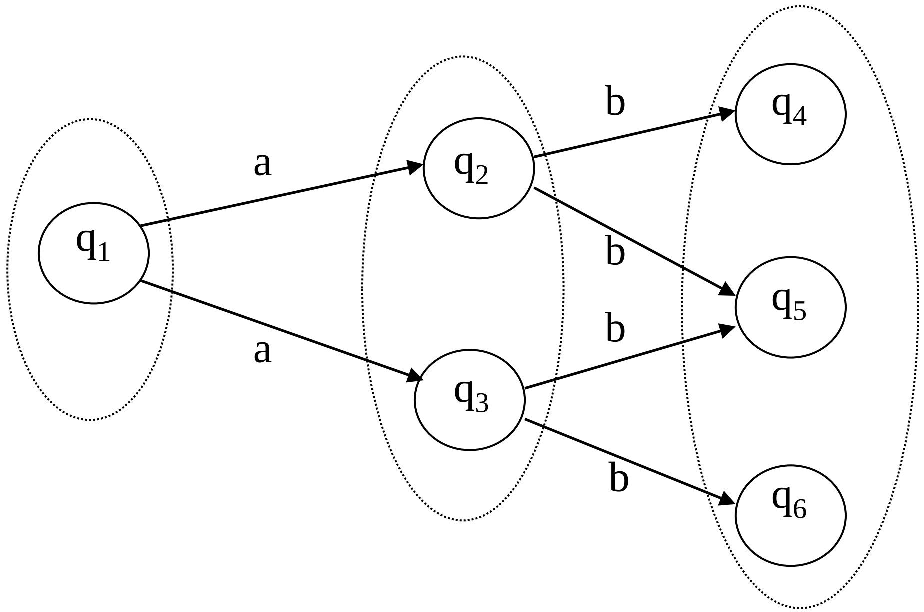
  $\delta(q_2,b) = \{q_4, q_5\}$,

  $\delta(q_3,b) = \{q_5, q_6\}$

$\rightarrow \delta^*(q_1,ab) = \{q_4, q_5, q_6\}$



DFSA vs NDFSA



- Starting from q_1 and reading *ab* the automaton reaches a state that belongs to the set $\{q_4, q_5, q_6\}$
- We call **“state” the set of possible states** in which the NDFSA can be during the run

Acceptance condition

$$x \in L \Leftrightarrow \delta^*(q_0,x) \cap F \neq \emptyset$$

Among the various possible runs (with the same input) of the NDFSA, it is sufficient that **one of them succeeds** to accept the input string

$\rightarrow$**Existential** nondeterminism

- There exists also a **universal interpretation**: $\delta^*(q_0,x) \subseteq F$

NDFSA into DFSA

- **NDFSA have the same power then DFSA**
- Given a NDFSA, an equivalent DFSA can be **automatically** computed as follows:

  If $\mathbf{A_{ND} = \langle Q, I, \delta, q_0, F\rangle}$ then $\mathbf{A_D = \langle Q_D, I, \delta_D, q_{0D}, F_D\rangle}$ with

  - $Q_D = \mathcal{P}(Q)$
  - $\delta_D(q_D, i) = \bigcup_{q \in q_D} \delta(q, i)$
  - $q_{0D} = \{q_0\}$
  - $F_D = \{q_D \mid q_D \in Q_D \wedge q_D \cap F \neq \emptyset\}$

Example (in lab)

- The concept is simple, just take some time to fully go trough an example

- You will see an example in detail during the lab sessions

Why ND?

- **NDFSAs are not more powerful than FSAs**, but they are not useless
  - **It can be easier to design a NDFSA**
  - They can be exponentially smaller w.r.t. the number of states
  - See the example in the lab

- Example: a NDFSA with 5 states becomes in the worst case an FSA with 2^5 states