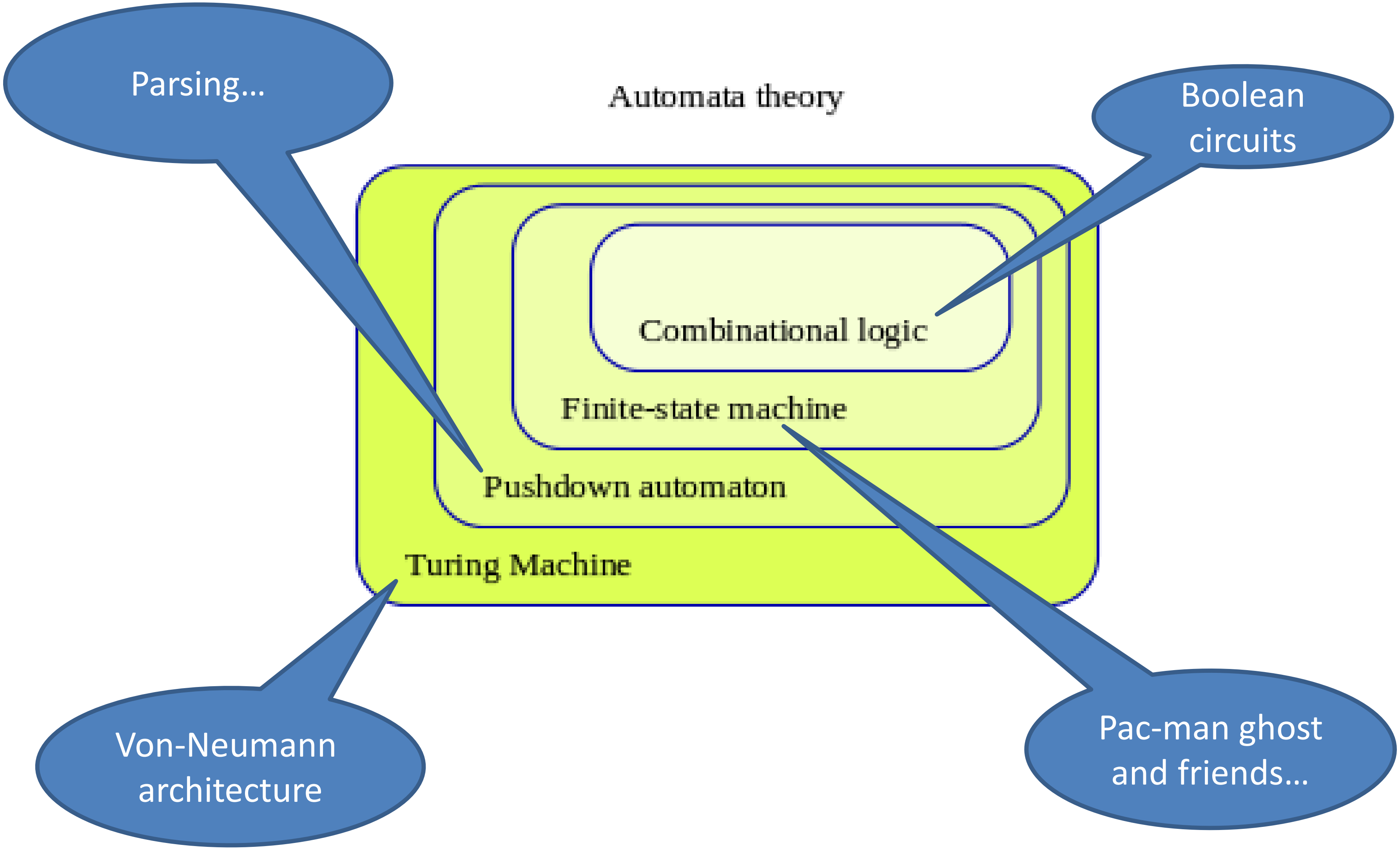
Theory of Computation

More on Turing Machines

Lecture 7a - Manuel Mazzara

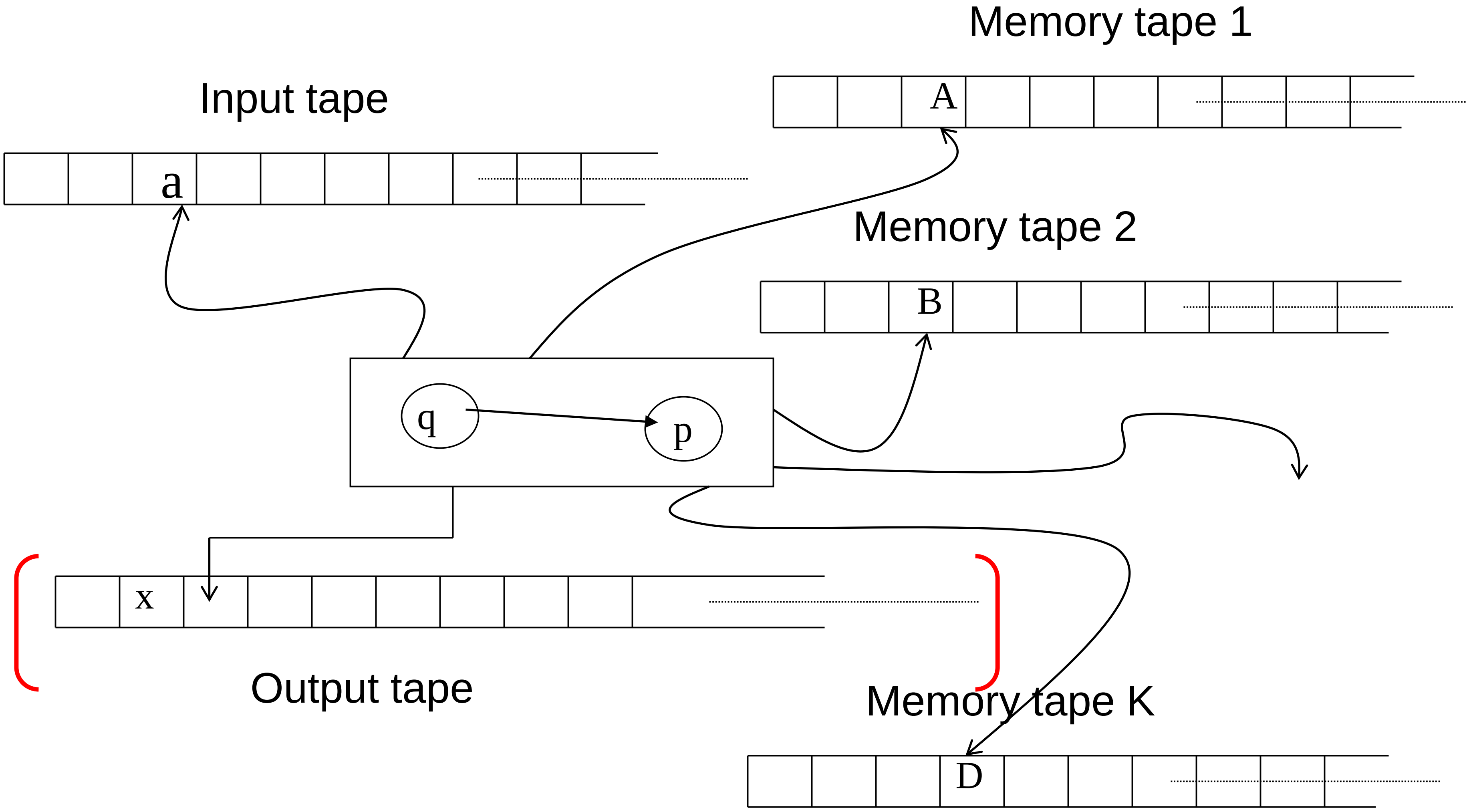
A bit of context



Turing machine

- The Turing machine (TM) is the **historical model of "computer"**
  - simple
  - conceptually important
- TMs use tapes as memory
  - **Tapes are not destructive**
  - They can be read many times

The general model



History

- The Turing machine was invented **in 1936 by Alan Turing**, (*a-machine*, automatic machine)

- It is a mathematical description of a very simple device for <u>arbitrary computations</u>

Church–Turing thesis

- The intuitive notion of "effectively calculable" is captured by
  - the functions computable by a Turing machine
  - by those expressible in the lambda calculus

- This assumption is now known as the **Church–Turing thesis**

Informally

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

Definition

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

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

where

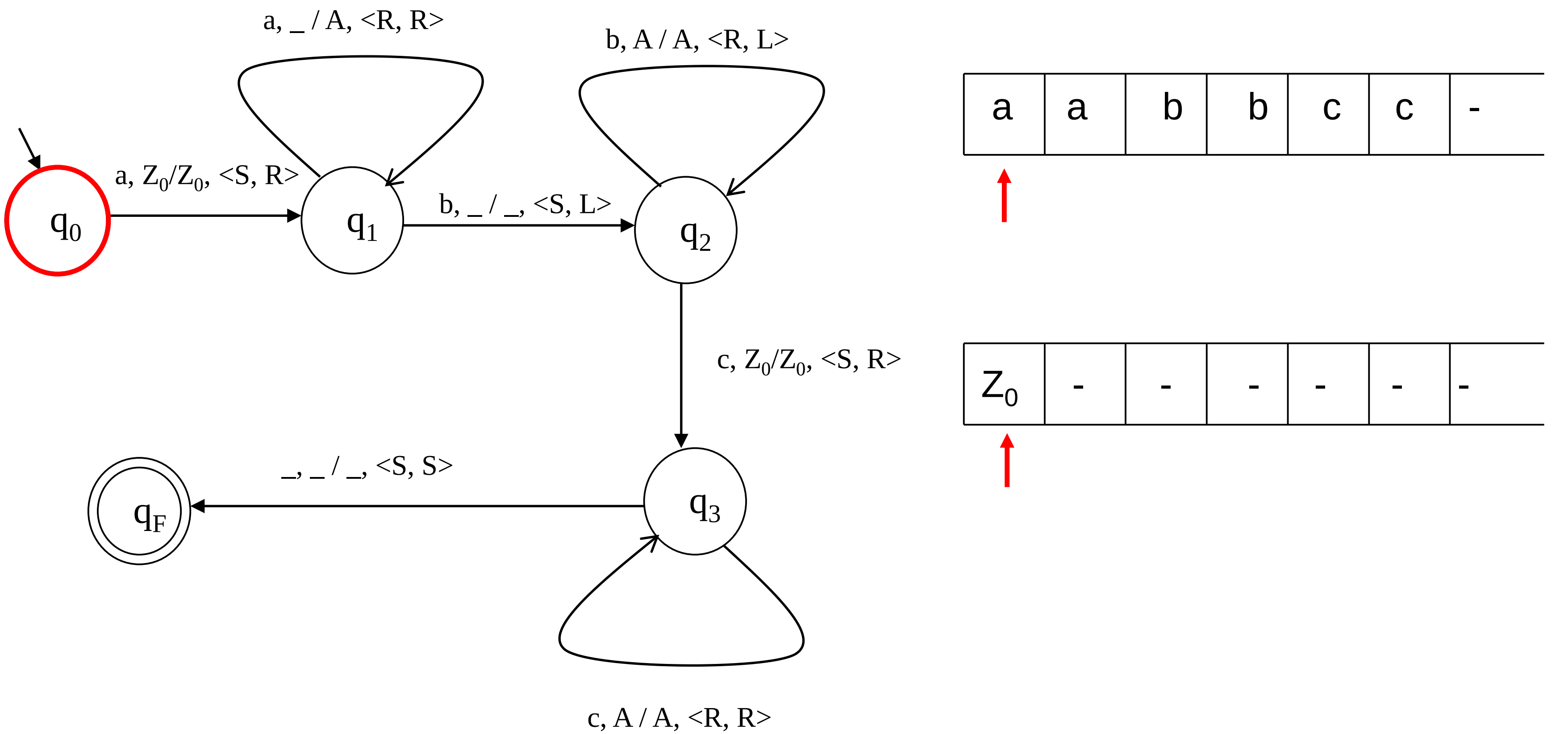
- $q \in Q$
- $x,y \in I^*$, $i \in I$
- $\alpha_r,\beta_r \in \Gamma^*$, $A_r \in \Gamma$ $\forall r\ 1 \leq r \leq K$
- $\uparrow \notin I \cup \Gamma$

Initial configuration

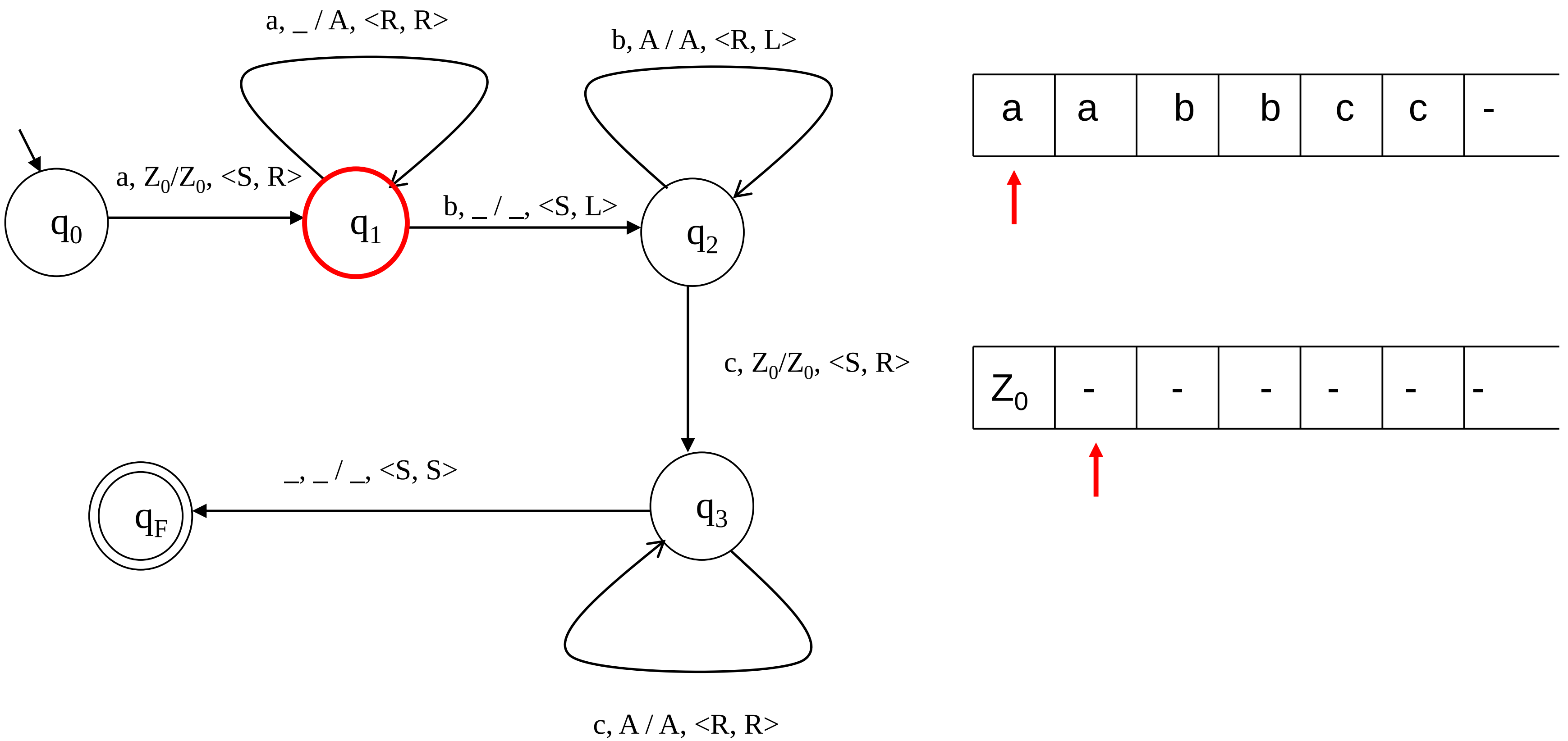
$$c_0 = \langle q_0, \uparrow iy, \uparrow Z_0, \ldots, \uparrow Z_0 \rangle$$

- 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

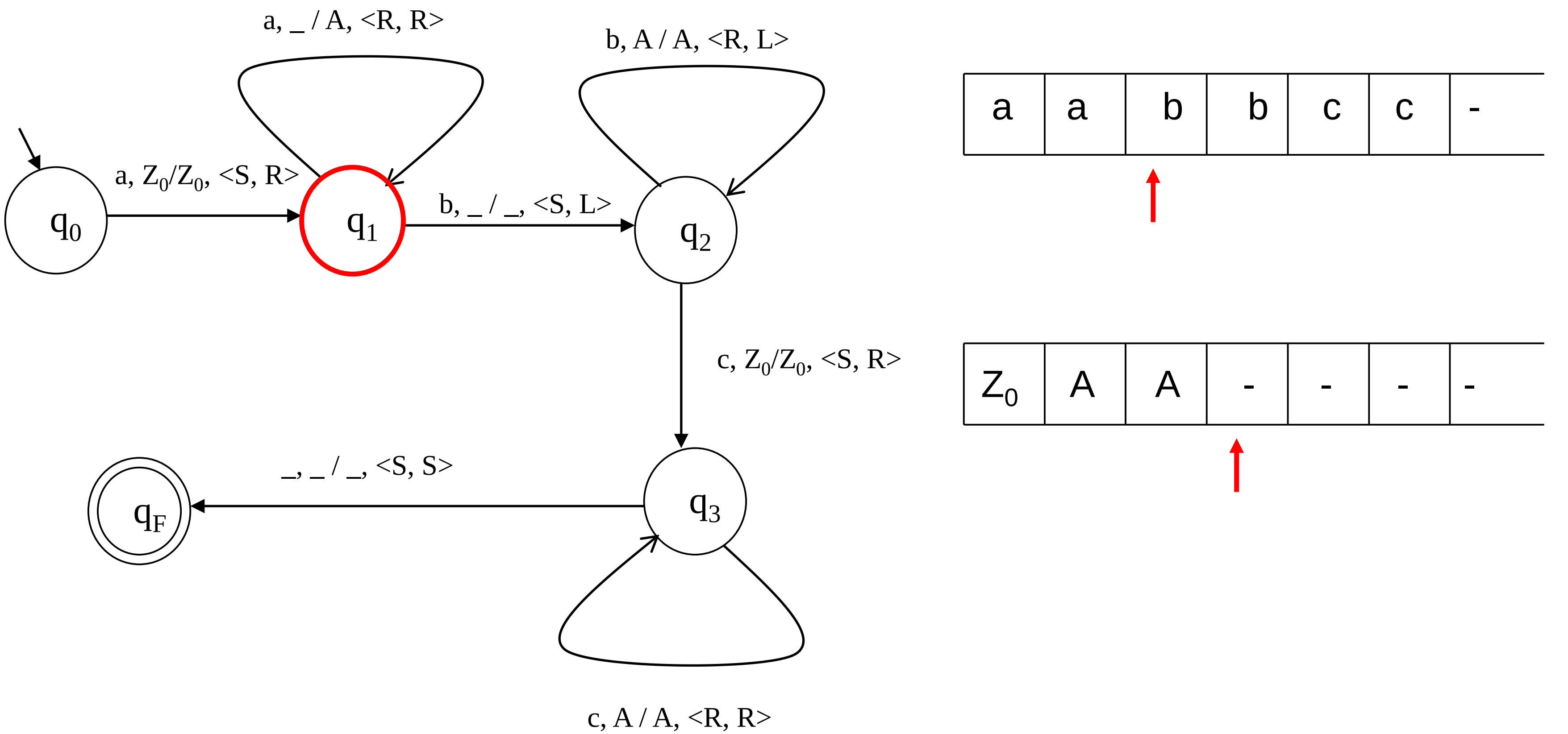
Example



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



$c = \langle q_1, \uparrow aabbcc, Z_0 \uparrow \rangle$



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

Acceptance condition

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

$$c_0 = <q_0, \uparrow x, \uparrow Z_0, \ldots, \uparrow Z_0> \quad |\!-^*\!-_M$$

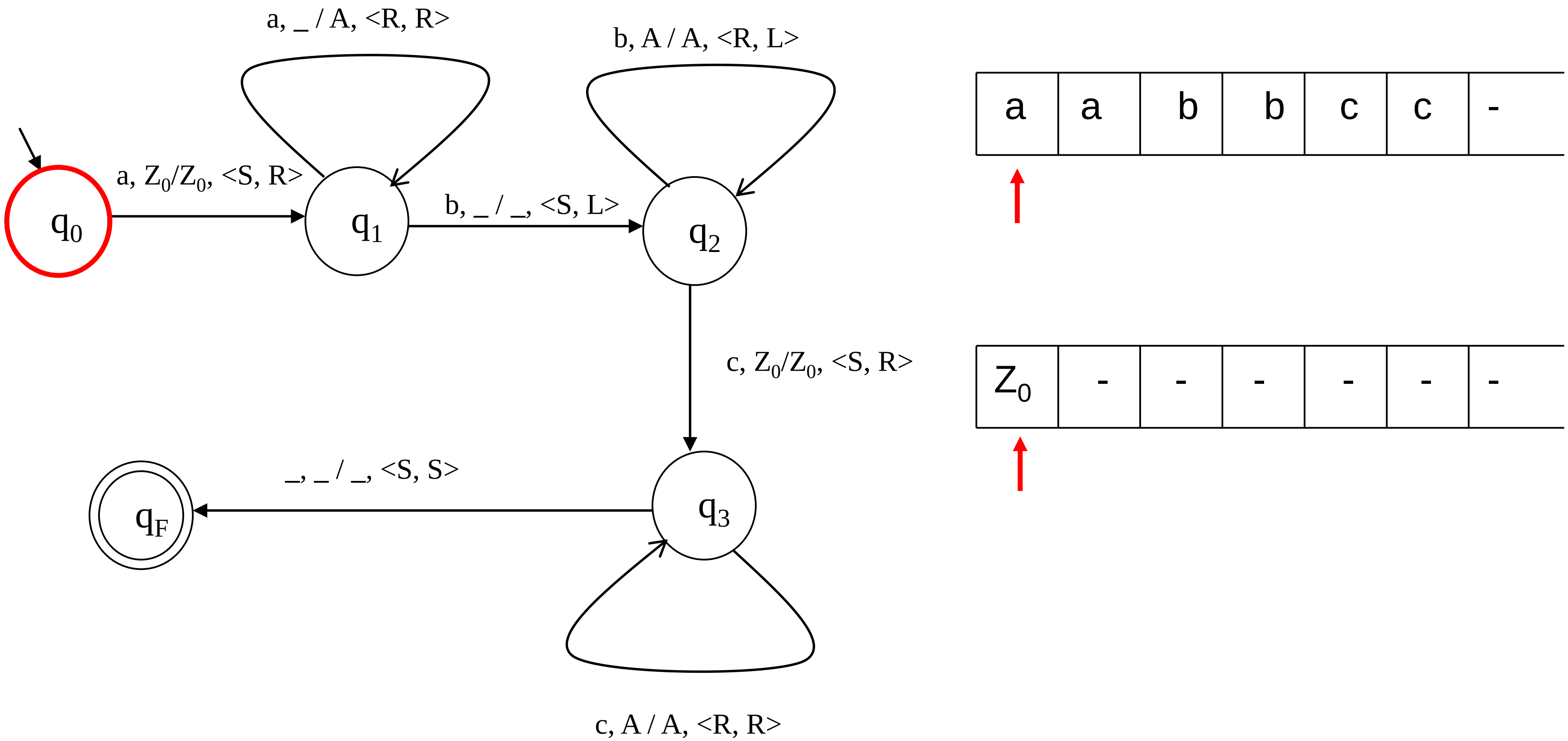
$$c_F = <q, x' \uparrow iy, \alpha_1 \uparrow A_1 \beta_1, \ldots, \alpha_K \uparrow A_K \beta_K> \text{ with } q \in F$$

$$(\text{and } x = x'iy)$$

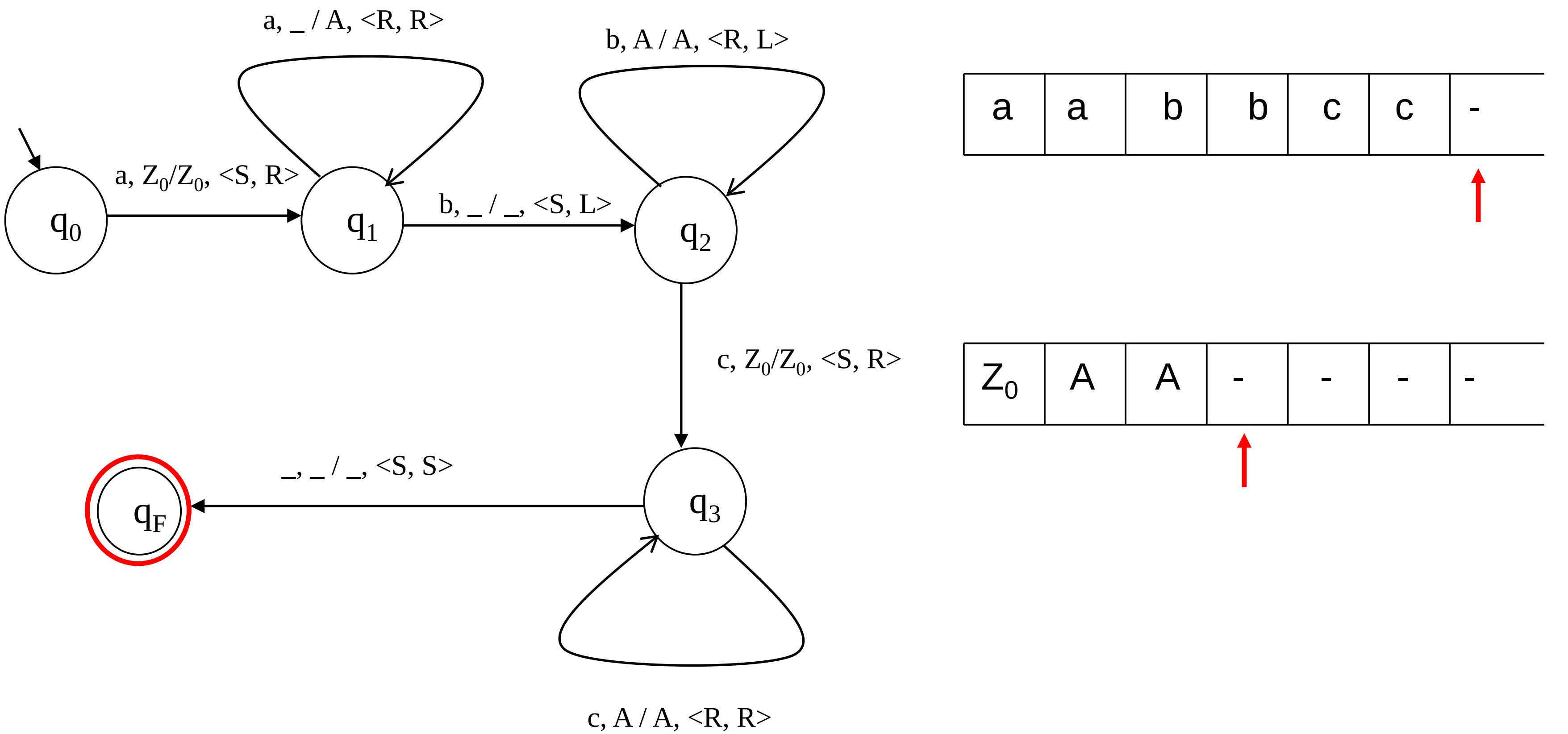
  - c_F is called final configuration
  - $|\!-^*\!-_M$ is the reflexive transitive closure of the $|\!-\!-_M$ relation

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

Example



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



$$c_0=\langle q_0, \uparrow aabbcc, \uparrow Z_0\rangle$$

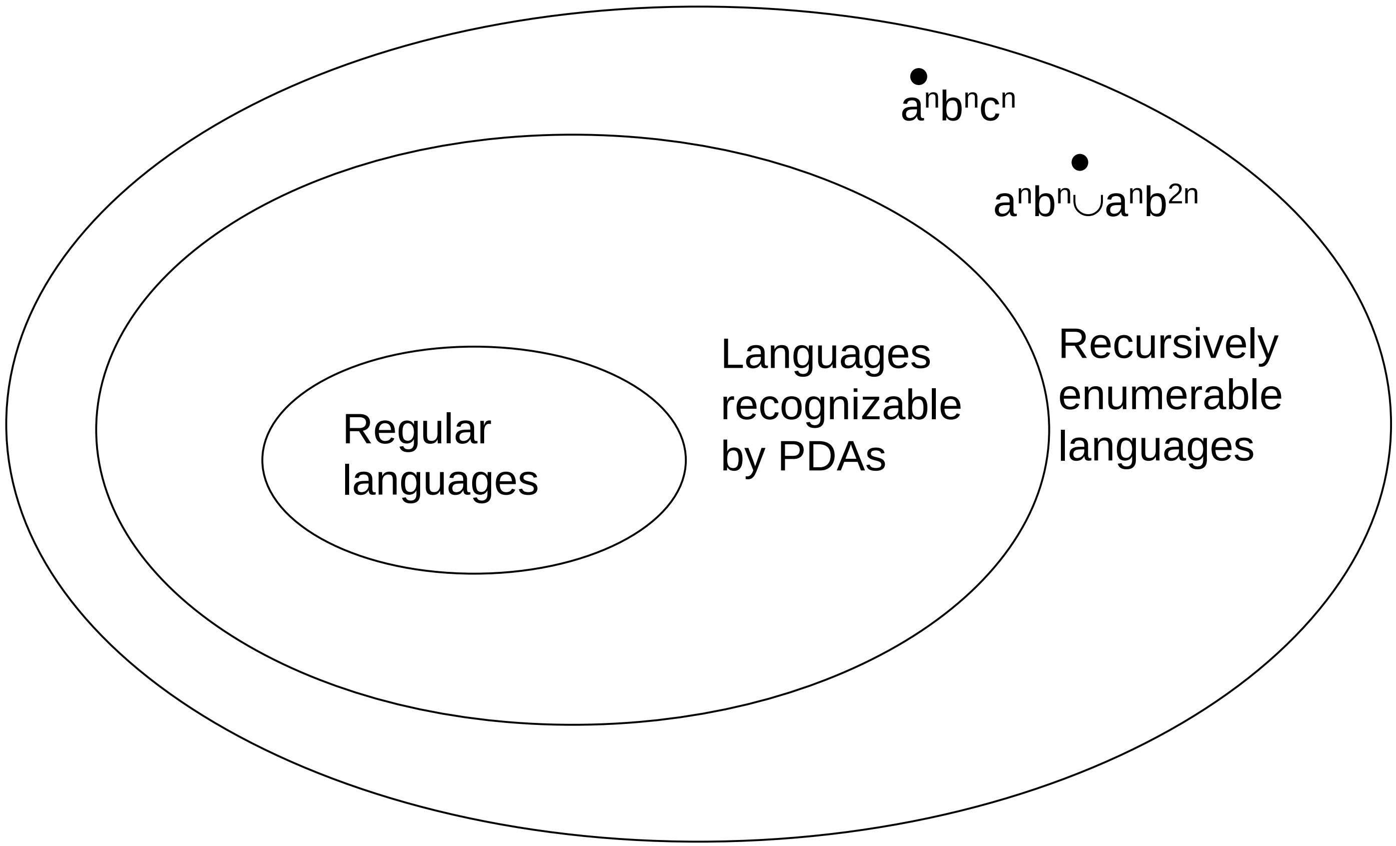
$$c_F=\langle q_F, aabbcc\uparrow, Z_0AA\uparrow\rangle$$

TM vs PDA

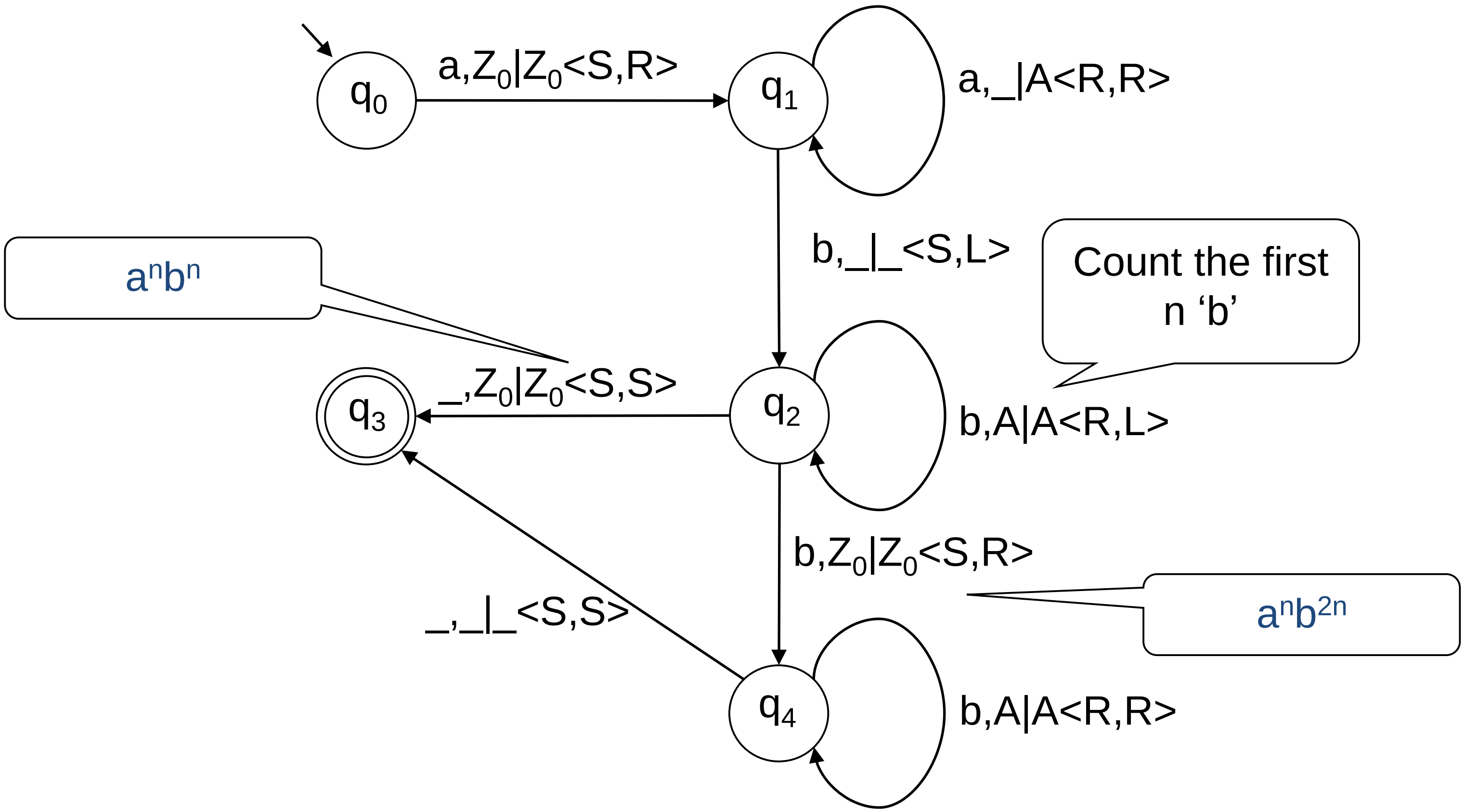
- We know that $a^n b^n c^n$ or $a^n b^n \cup a^n b^{2n}$ cannot be recognized by any PDA
- but they can be recognized by a TM
  - We have seen a TM for $a^n b^n c^n$
- <u>Every language recognizable by a PDA can be recognized by a TM</u>
  - A TM can always be built to use (one of) its memory tape(s)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

Languages

- TMs have a higher expressive power than PDAs



Example: $a^nb^n \cup a^nb^{2n}$



TM vs Von Neumann Machines

- TMs can simulate a Von Neumann machine (VNM)
  - It is an **abstract model of computers**
- TM differs from VNM wrt. memory access
  - TM: sequential
  - VNM: direct
- **The memory access type does not affect the expressive power of a machine**
  - It does not change the class of problems solvable with a machine (**with this specific difference**)
  - It may affect the complexity

Operations on TMs (1)

- TMs are closed under
  - **Intersection**
  - **Union**
  - **Concatenation**
  - **Kleene star**
- TMs are **not close under complement**
  - They are not close under difference either (**why?**)

Operations on TM (2)

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

  this explains the closure

Complement

- Would loop-free TMs (if they existed) be closed under complement?
  - **Yes**: it would suffice to define the **set of halting states** and partition it into **accepting** and **non-accepting states**

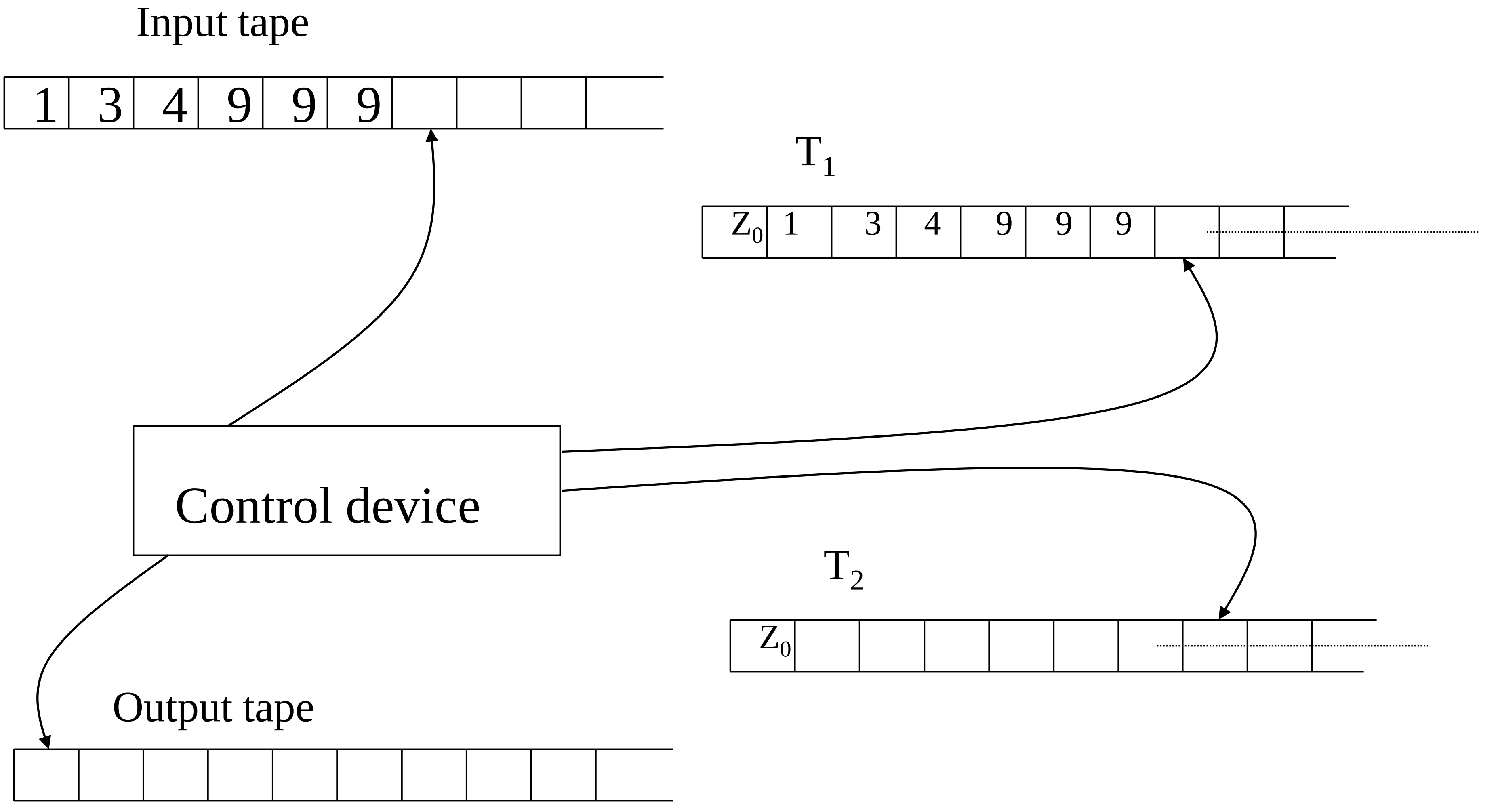
$\rightarrow$ Problems arise from <u>nonterminating</u> computations (to be discussed later on)

How do we use TMs?

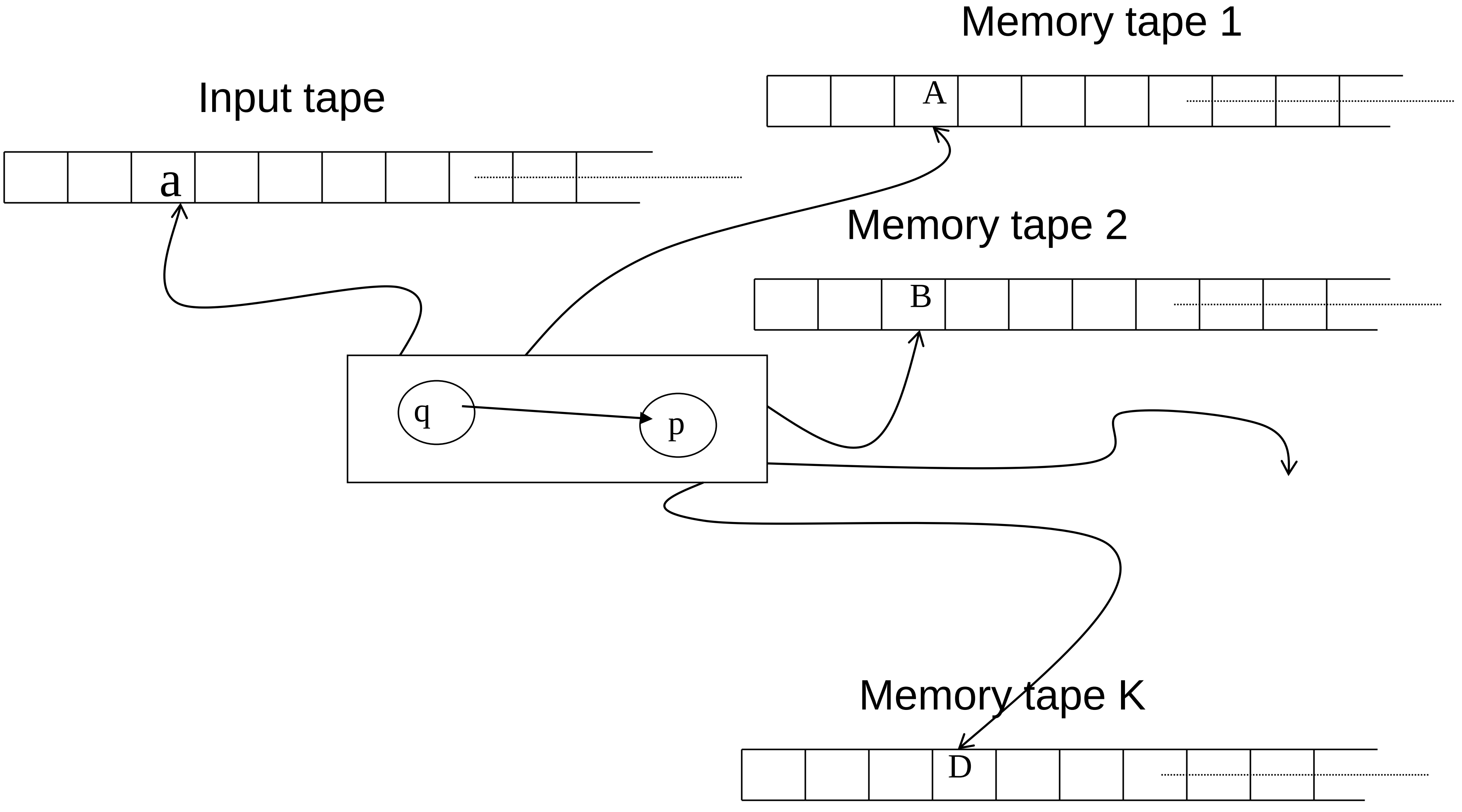
- TMs can
  - Recognize languages (**Acceptors**)
  - Translate accepted languages (**Transducers**)
- … but also compute functions
  - They are **equivalent to VNMs**

$\rightarrow$ We can think of a TM as an **abstract model of "computer" with sequential memory access**

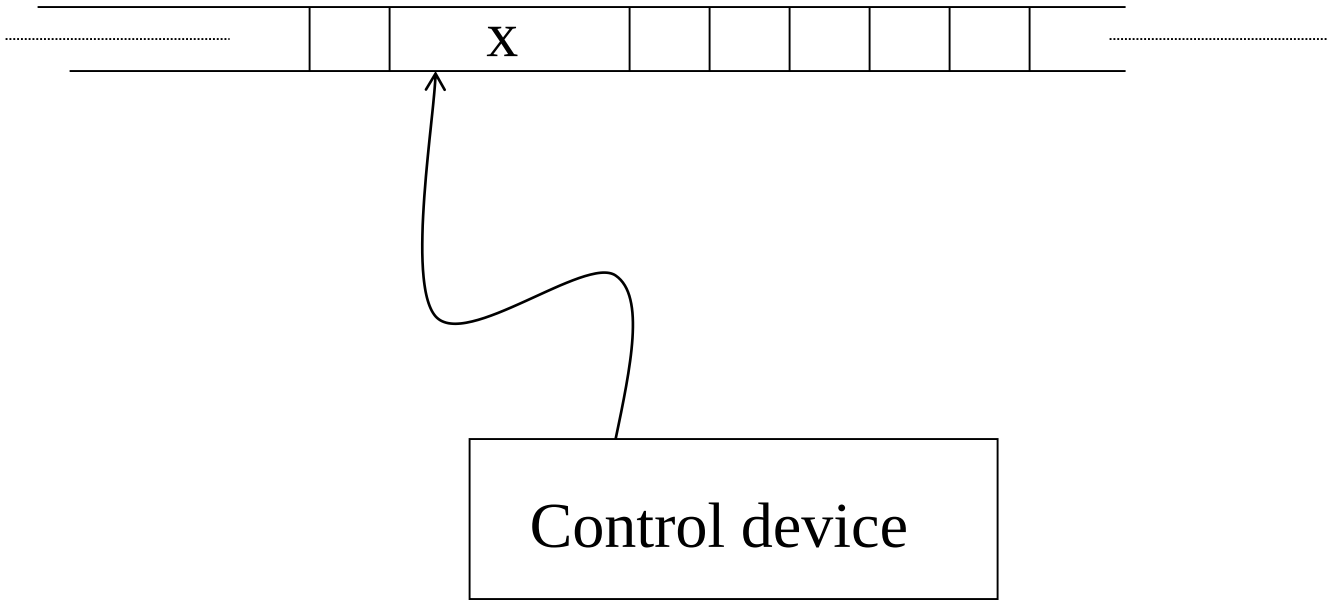
Mechanical view



TM model



Single tape TM

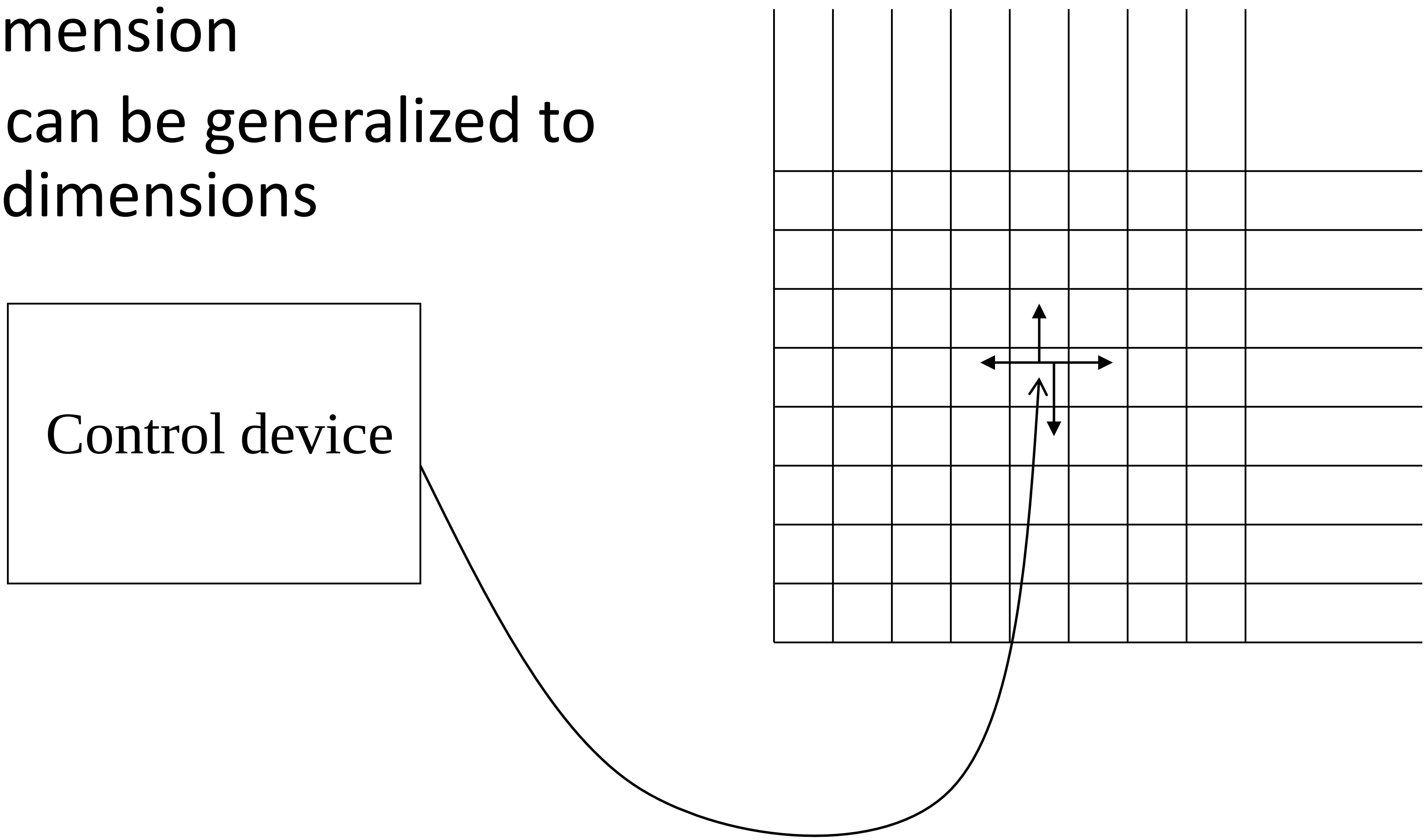


Single tape

- usually **unlimited in both directions**
- it serves as **input**, **memory** and **output** tape

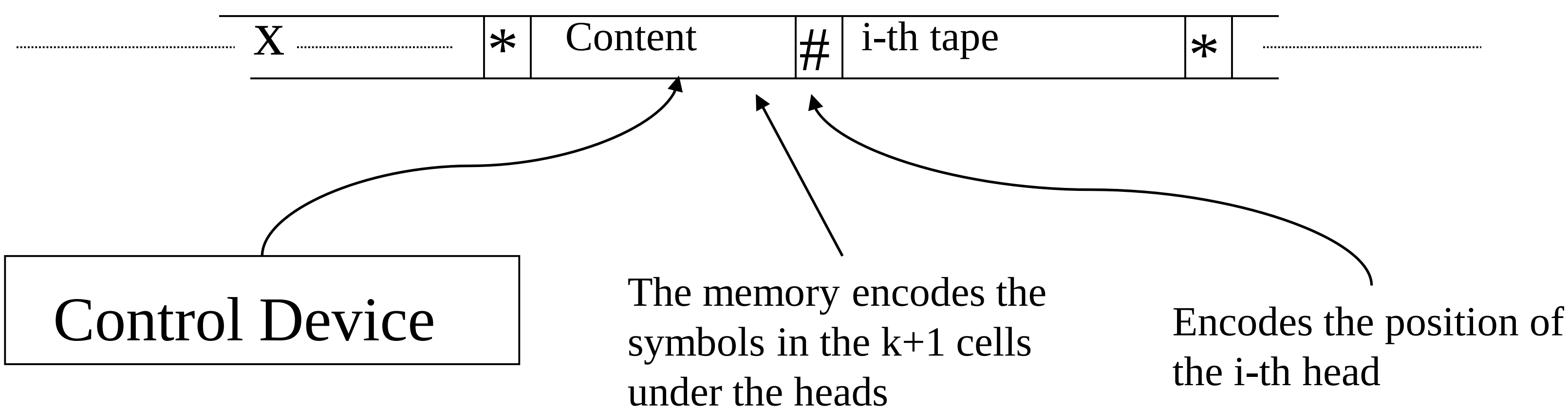
Bidimensional tape TM

- 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
- All these TM models are equivalent
  - They can recognize the same class of languages



Theory of Computation

Nondeterminism

Lecture 7b - 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

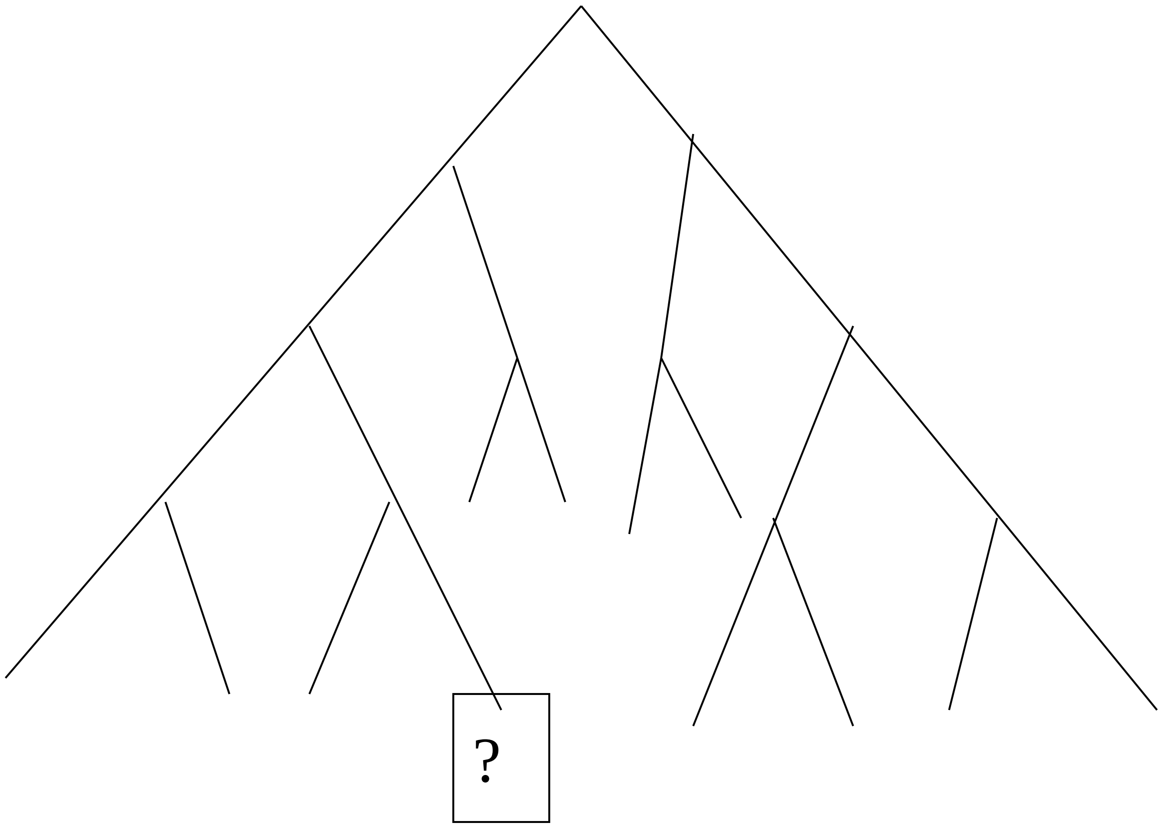
```
if
      x>=y then max:=x
      x<=y then max:=y
fi
```

Nondeterministic models (1)

- Is it only a matter of elegance?
- Let us consider the **`case`** construct of Pascal: why not having something like the following?

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

Blind search



Another form of nondeterminsm that is usually “hidden”

Search algorithms

- Search algorithms are a “simulation” of basically nondeterministic algorithms
  - Is the element searched for in the root?

    If yes, ok

    Otherwise

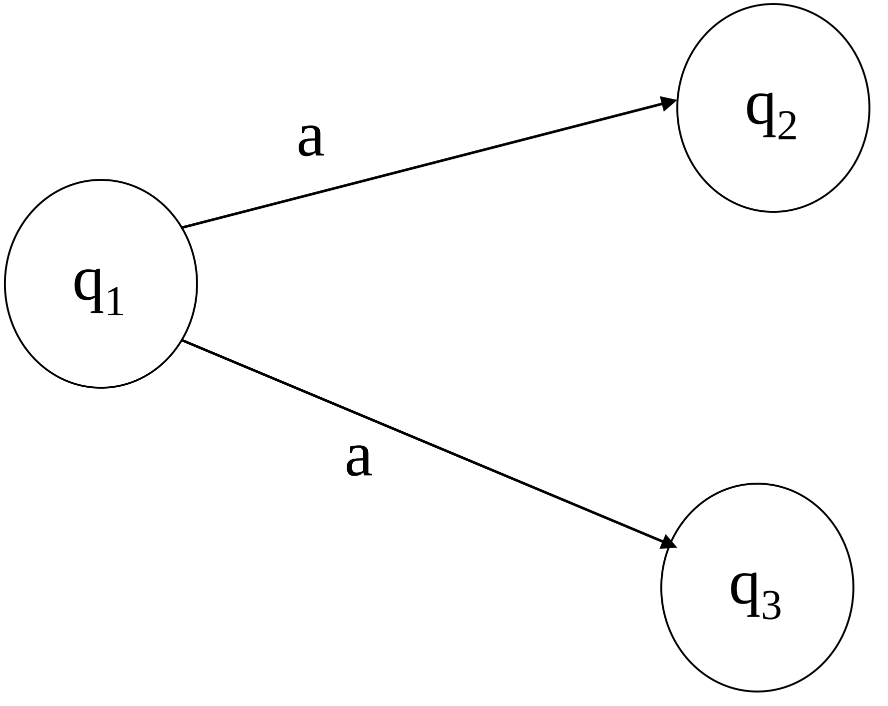
      Search the left subtree

      Search the right subtree
- Choice of priority among paths is often arbitrary

In conclusion

- Nondeterminism (ND) is a model of computation or at least a model of **parallel computing**
  - Ada and other concurrent languages exploit it
- It is a useful abstraction to describe search problems and algorithms
- It can be applied to various computational models
- Important: 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: $Q \times I \rightarrow \mathcal{P}(Q)$
- What happens to δ^*?

Move sequence

- δ^* is inductively defined from δ

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

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

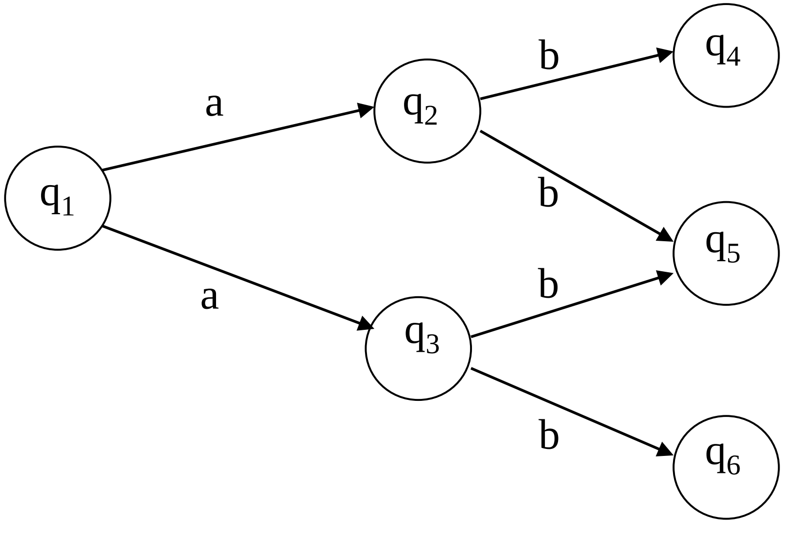
- Example:

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

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

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

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



Acceptance condition

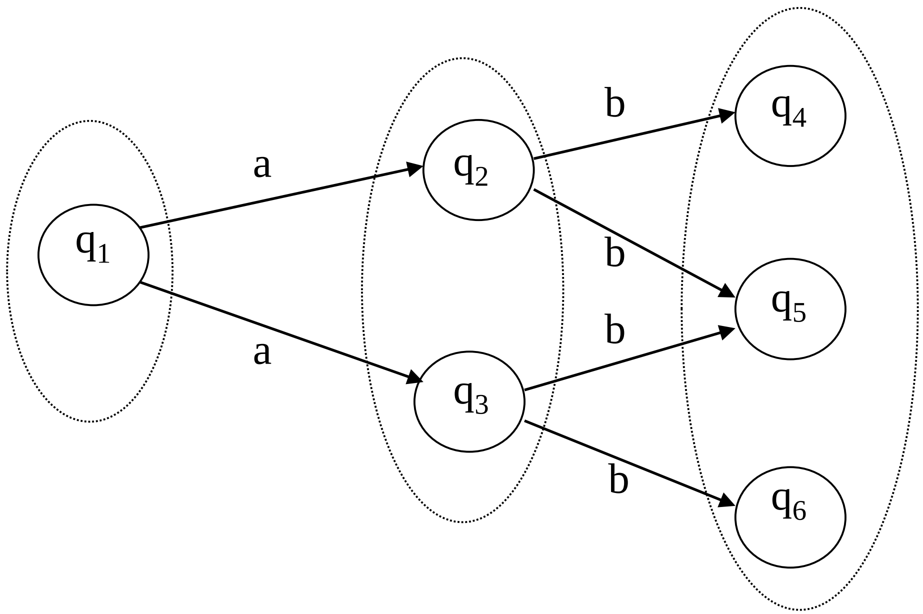
$$x \in L \Leftrightarrow \delta^*(q_0,x) \cap F \neq \emptyset$$

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

→<u>Existential</u> nondeterminism

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

DFSA vs NDFSA



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

Formally

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

  If $A_{ND} = \langle Q, I, \delta, q_0, F \rangle$ then

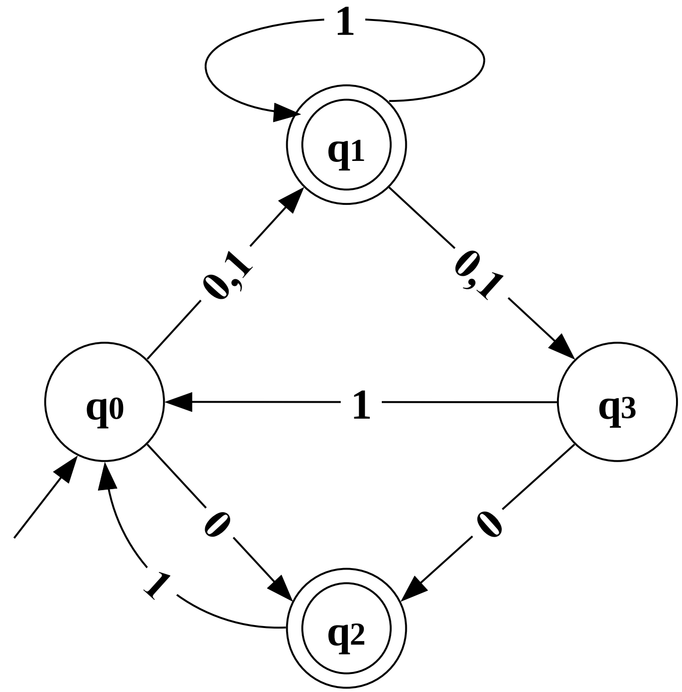
  $A_D = \langle Q_D, I, \delta_D, q_{0D}, F_D \rangle$, where

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

Example (1)

Transform the following NDFSA into the equivalent DFSA

q0	0	q1
q0	0	q2
q0	1	q1
q1	0	q3
q1	1	q3
q1	1	q1
q2	0	⊥
q2	1	q0
q3	0	q2
q3	1	q0



Example (2)

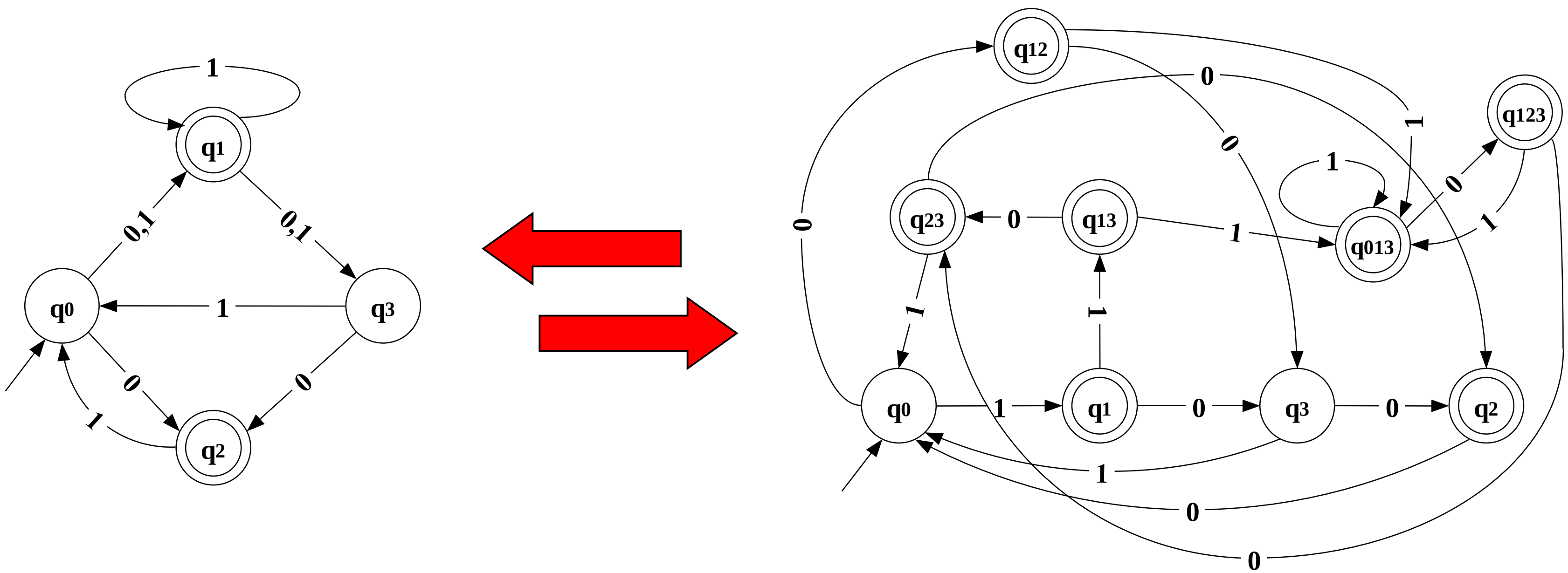
- Let us proceed recursively

q0	0	q1
q0	0	q2
q0	1	q1
q1	0	q3
q1	1	q3
q1	1	q1
q2	0	⊥
q2	1	q0
q3	0	q2
q3	1	q0

q0	0	q12
q0	1	q1
q1	0	q3
q1	1	q13
q2	0	⊥
q2	1	q0
q3	0	q2
q3	1	q0
q12	0	q3
q12	1	q013
q13	0	q23
q13	1	q013
q013	0	q123
q013	1	q013
q23	0	q2
q23	1	q0
q123	0	q23
q123	1	q013

Example (3)

Graphically



Why ND?

- NDFSAs are no 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
- Example: a NDFSA with 5 states becomes in the worst case a FSA with 2^5 states

Nondeterministic TM

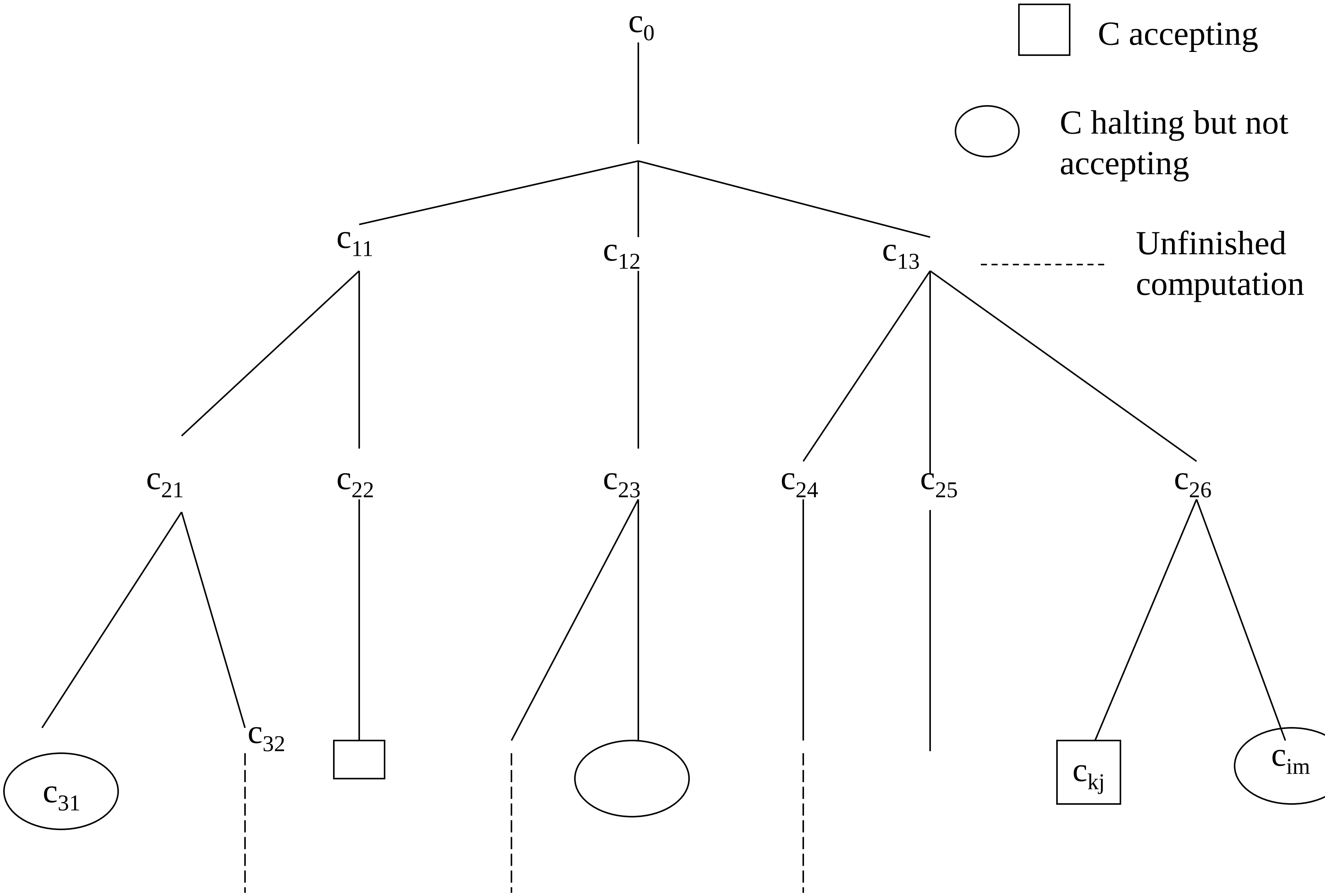
- To define a nondeterministic TM (NDTM), we need to change the **transition function** and the **translation mapping**
- All the other elements remain as in a (D)TM
- The transition function is

$$\delta\text{: (Q-F) x I x } \Gamma^k \rightarrow \mathcal{P}(\text{Q x } \Gamma^k \text{ x } \{\text{R,L,S}\}^{k+1})$$

and the output mapping

$$\eta\text{: (Q-F) x I x } \Gamma^k \rightarrow \mathcal{P}(\text{O x } \{\text{R,S}\})$$

Computation tree



Acceptance condition

- A string $x \in I^*$ is accepted by a NDTM if and only if <u>there exists a computation that terminates in an accepting state</u>
- It would seem that the problem of accepting a string can be reduced to a visit of a computation tree
  - How should we perform the visit?
  - What about the relationship between DTMs and NDTMs?

Visiting the computation tree

- We know different kinds of visits:
  - Depth-first visit
  - Breadth-first visit
- A depth-first visit cannot work in our problem because **the computation tree could have infinite paths** and the algorithm might “get stuck” in it
- We should adopt a breadth-first visit algorithm

DTM vs NDTM

Can we build a DTM that visits a tree level by level?

It is a long (and boring) exercise, but YES

→We can build a DTM that establishes whether a NDTM recognizes a string x

→Given a NDTM, we can build an equivalent DTM

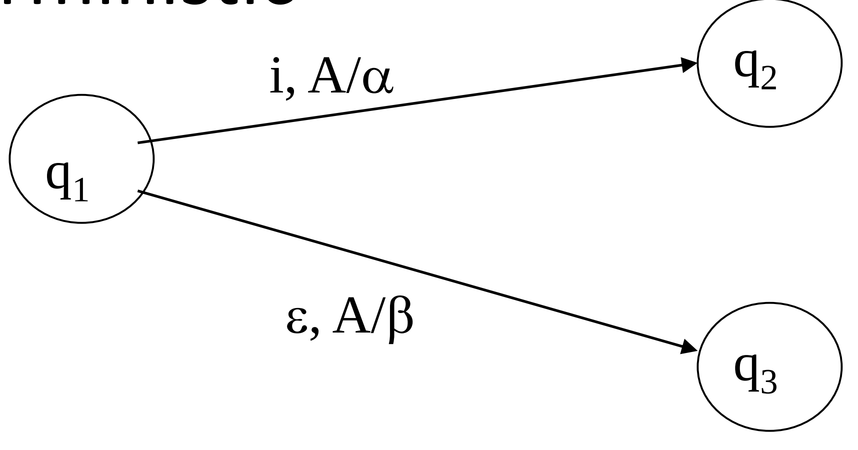
ND does not add power to TMs

ε Moves and PDAs

- ε-moves came with the following constraint:

  If $\delta(q,\varepsilon,A)\neq\perp$, then $\delta(q,i,A)=\perp \;\; \forall i\in I$

- Without this constraint the presence of ε-moves would make PDAs intrinsically nondeterministic



Adding nondeterminism to PDAs

- **Removing the constraint already makes the PDA nondeterministic**
- Additionally, we can have nondeterminism by changing the transition function of a PDA and consequently:
  - transitions among configurations
  - acceptance condition

Definition

A nondeterministic PDA (NDPDA) is a tuple

<Q, I, Γ, δ, q_0, Z_0, F>

- Q, I, Γ, q_0, Z_0, F as in (D)PDA
- δ is the transition function defined as

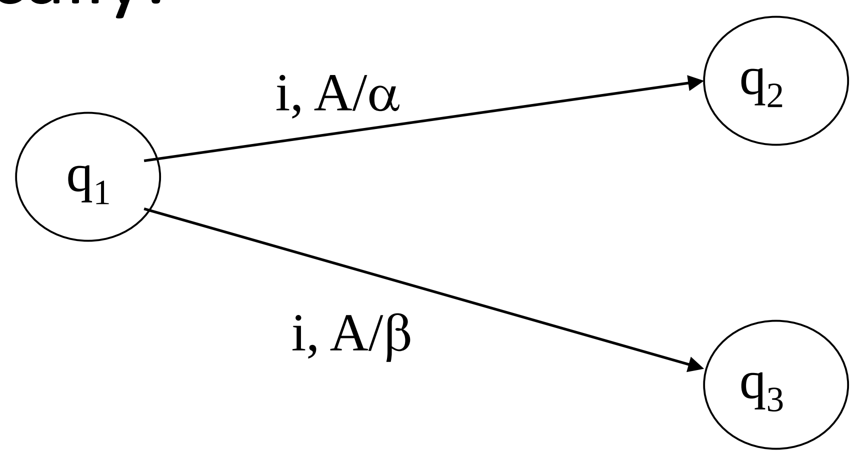
$$\delta\colon Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_{\mathcal{F}}(Q \times \Gamma^*)$$

- What is the $\mathcal{F}$ in $\mathcal{P}_{\mathcal{F}}$?
- Why $\mathcal{F}$?

Transition function

$$\delta: Q\times(I\cup\{\varepsilon\})\times\Gamma\rightarrow\mathcal{P}_{\mathcal{F}}(Q\times\Gamma^*)$$

- $\mathcal{P}_{\mathcal{F}}$ indicates the *finite* subsets of $Q\times\Gamma^*$
  - Why did we not specify it for NDTM?
- Graphically:



Effects of nondeterminism

- ND does not add expressive power to
  - TMs
  - FSAs
- Does ND add expressive power to DPDAs?