Theoretical Computer Science

PDA, recap++

Lecture 8 - Manuel Mazzara

Our itinerary

Again now we **go from informal/intuition into examples and then to the formal definition**

We need to be able to master all the levels back and forth

This is the job of a **Computer Scientist and Software Engineer**

Your job

Advancing **software correctness** means making tools and methods available for standard off-the-shelf software and average users

Tools need **simplicity and friendly interface** for their use to be **scalable**, at the moment often PhDs-level researchers are necessary

Recap questions

There are languages that are not regular. What does it mean?

Do you remember some of the consequences of Pumping Lemma?

Do you remember the difference between **fixed and finite memory**?

What are the acceptance criteria for a PDA?

Fixed vs finite memory (1)

- **Regular languages are languages which can be recognized by an automaton with <u>fixed memory</u>**
  - Fixed memory is more restrictive than finite!
  - Finite vs. unlimited
  - Think about FSA (states only) and PDA (stack can grow)

- FSA is a model of computation with fixed memory
- PDA has finite but not fixed

Fixed vs finite memory (2)

- **Many languages cannot be recognized using only fixed memory**
  - For example $a^n b^n$
  - FSA cannot count an unlimited *n*
  - <u>Number of states is fixed, stack can grow</u>

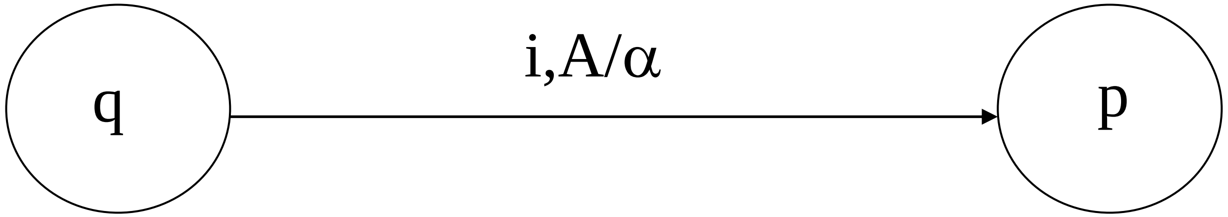
A PDA, formally

A PDA is a tuple **$<Q, I, \Gamma, \delta, q_0, Z_0, F>$**

- Q is a finite set of states
- I is the input alphabet
- Γ is the stack alphabet
- δ is the transition function
- $q_0 \in Q$ is the initial state
- $Z_0 \in \Gamma$ is initial stack symbol
- $F \subseteq Q$ is the set of final states

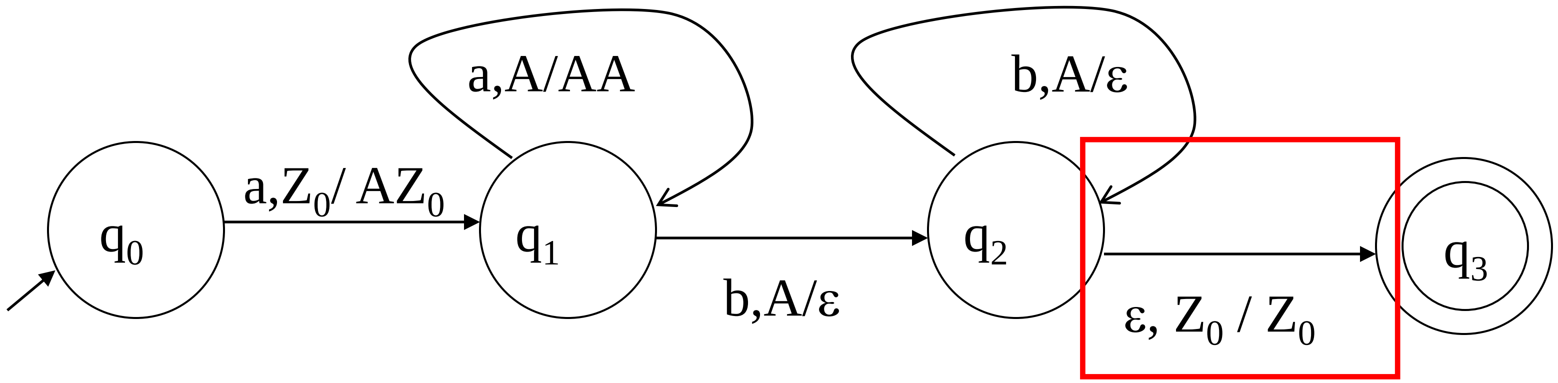
Transition function

- δ is the **transition function**
- δ: $Q\times(I\cup\{\varepsilon\})\times\Gamma\rightarrow Q\times\Gamma^*$
  - $\delta(q, i, A)=<p, \alpha>$
- Graphical notation:



Example

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



Configuration, informally

- A configuration is a generalization of the notion of state

- A configuration shows
  - the current state of the control device
  - the portion of the input string that starts from the head
  - the stack

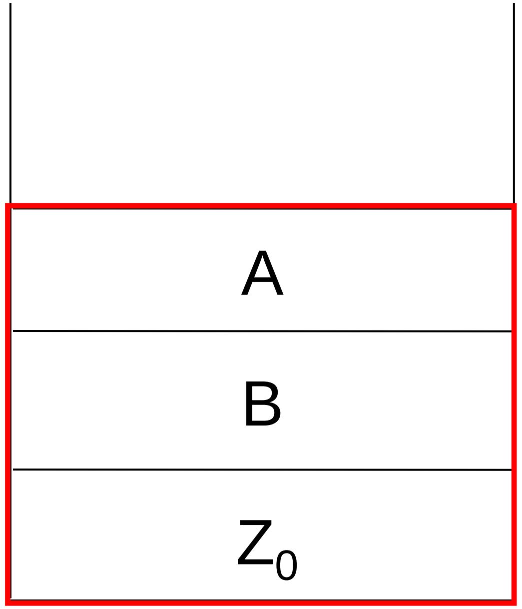
- It is a **snapshot** of the PDA

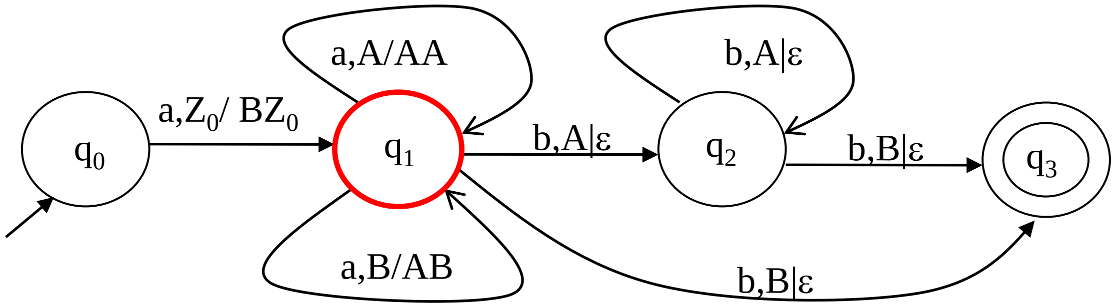
Configuration, formally

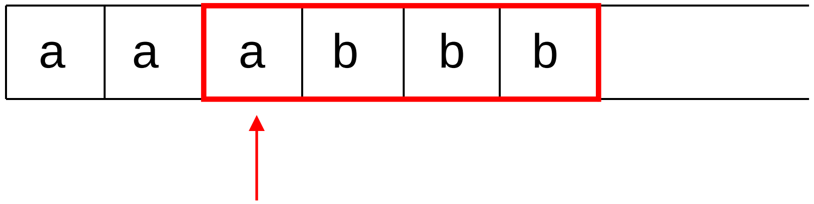
- A configuration c is **$<q, x, \gamma>$**
  - $q \in Q$ is the current state of the control device
  - $x \in I^*$ is the unread portion of the input string
  - $\gamma \in \Gamma^*$ is the string of symbols in the stack
- Conventions:
  - The stack grows bottom-up
  - The input strings is read left to right
  - The other way around is possible, but is important to be coherent!

Example of configuration

$c=\langle q_1, abbb, ABZ_0\rangle$







Transitions

- Transitions between configurations (|--) depend on the transition function
  - The transition function shows **how to move from a PDA snapshot to another**
- There are two cases:
  - The transition function *is defined for an input symbol*
  - The transition function *is defined for an ε move*

Spontaneous moves and nondeterminism

- An ε move is a <u>spontaneous move</u>
  - If $\delta(q,\varepsilon,A)\neq\perp$ and A is the top symbol on the stack, the transition can always be performed
- If $\delta(q,\varepsilon,A)\neq\perp$, then $\delta(q,i,A)=\perp \;\; \forall i\in I$
  - If this property was not satisfied, both the transitions would be allowed
  - <u>**Nondeterminism**</u>

It means "undefined

We will see this aspect soon

Acceptance condition

- Let $|\text{-}^{*}\text{-}$ be the reflexive transitive closure of the relation $|\text{--}$

<u>Acceptance condition</u>:

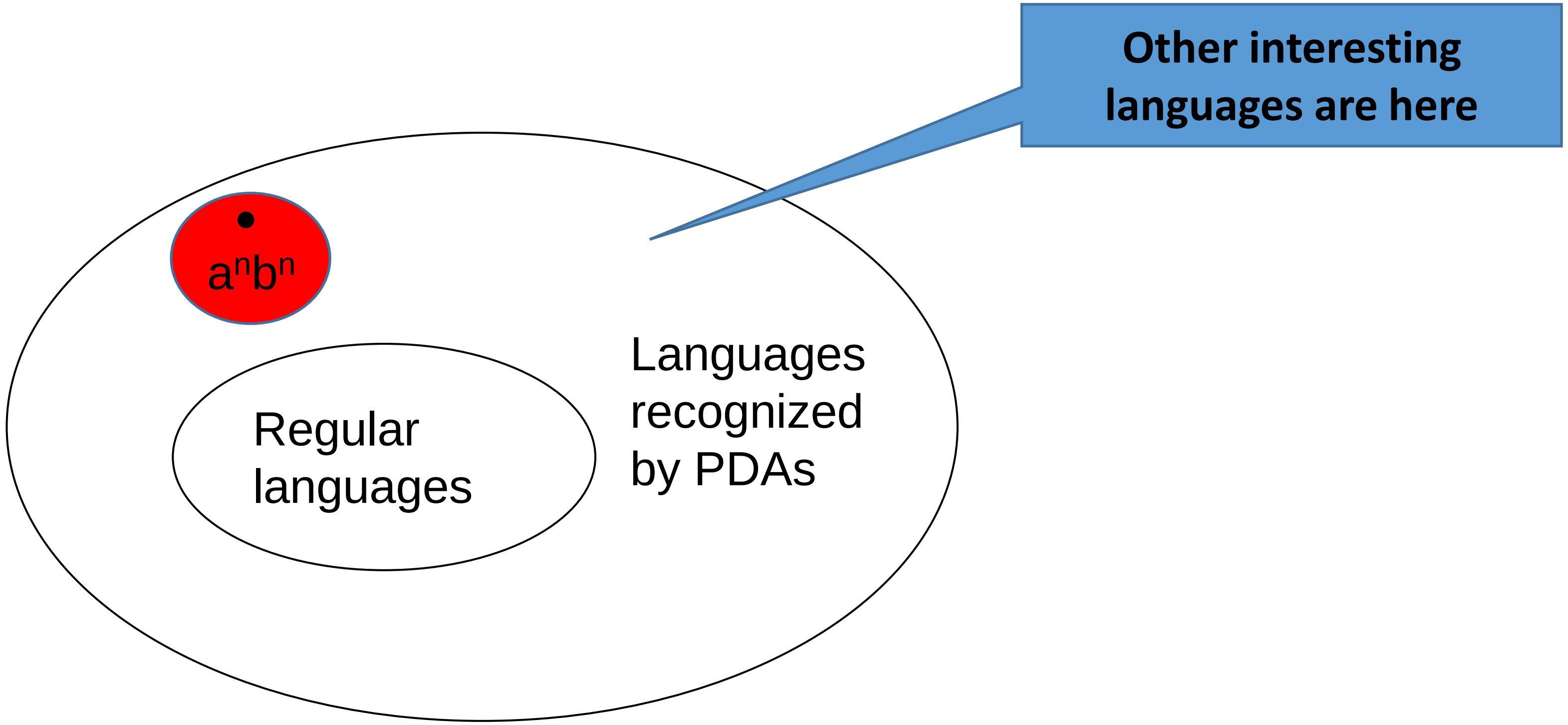
$\forall x \in I^{*}\ (x \in L \Leftrightarrow c_0 = \langle q_0, x, Z_0 \rangle\ |\text{-}^{*}\text{-}\ c_F = \langle q, \varepsilon, \gamma \rangle$ and $q \in F)$

Note: used in a configuration the meaning is not the same than epsilon-move – means the input string has been entirely "consumed"

- Informally, **a string is accepted by a PDA if <u>there is a path coherent with x on the PDA that goes from the initial state to the final state</u>**
  - The input string has to be read completely

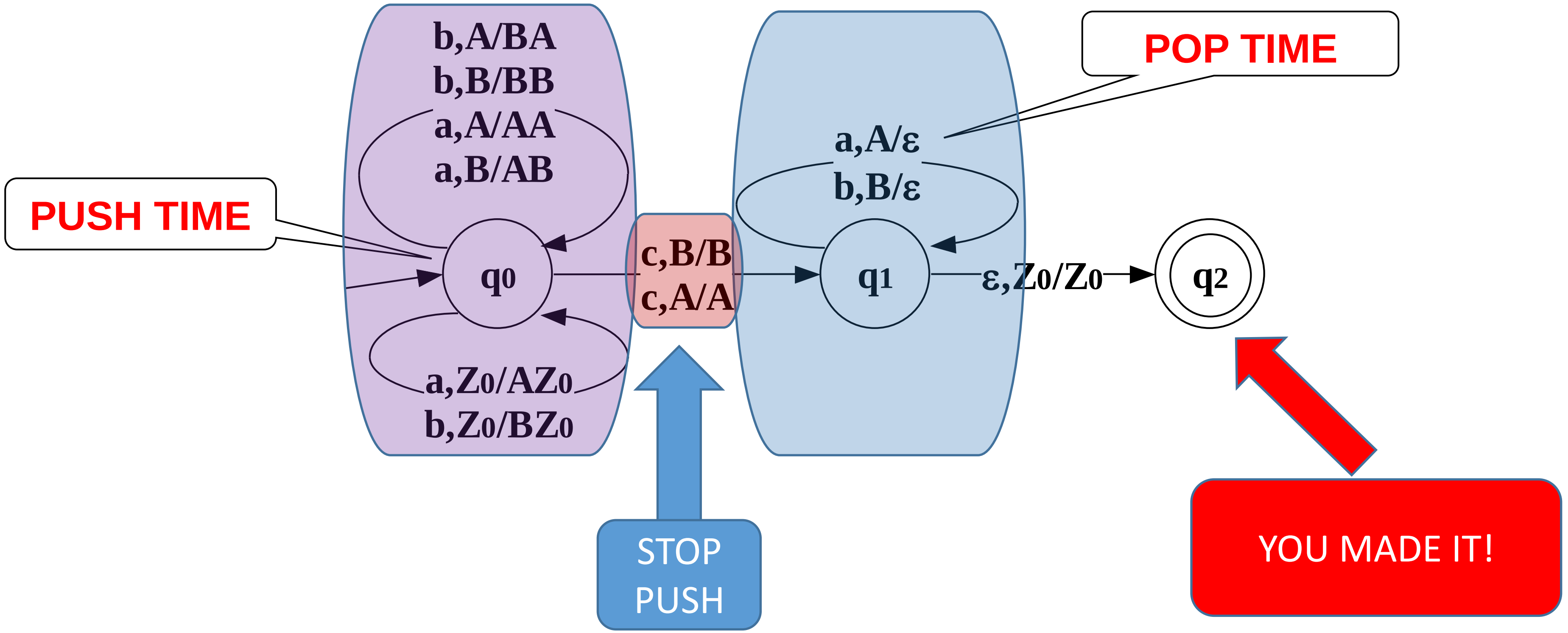
PDA vs FSA

- **PDAs are <u>more expressive</u> than FSAs**



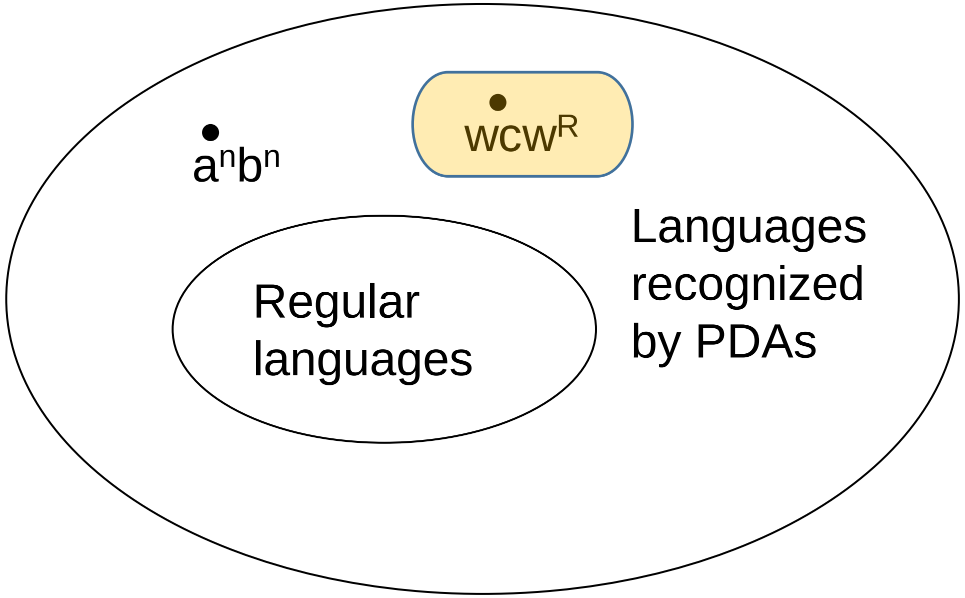
Example

- L = {wcw^R | $w \in \{a,b\}^+$}
  - We need to use a **LIFO policy to memorize w**



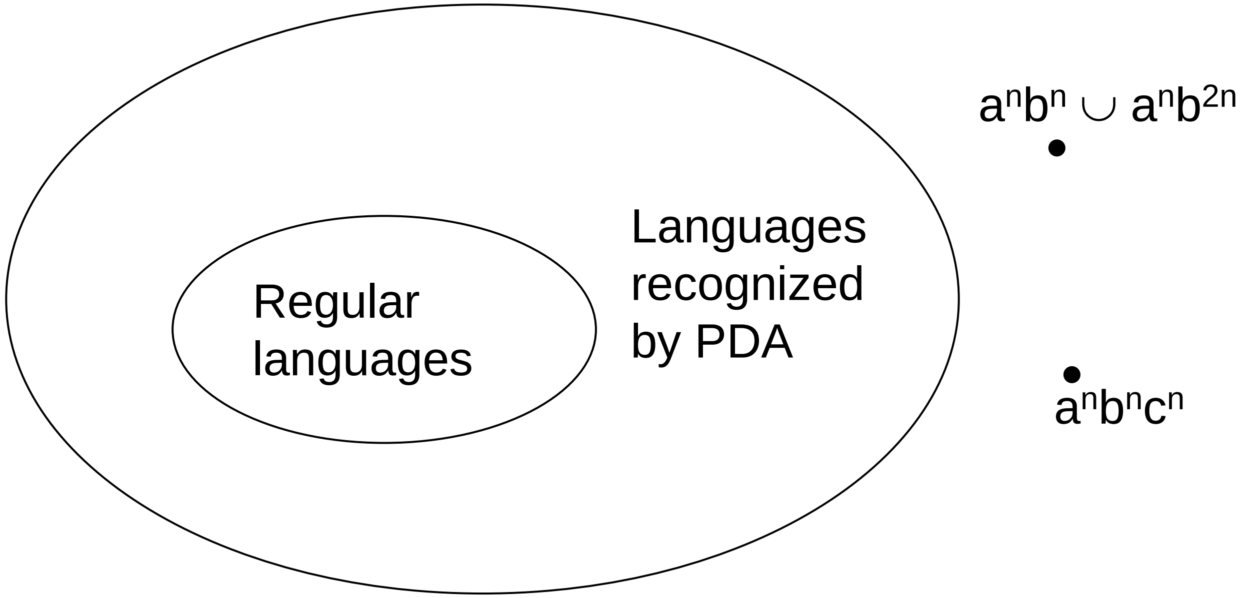
PDA vs FSA

- **PDAs are <u>more expressive</u> than FSAs**



Are there languages that cannot be recognized by a (deterministic) PDA?

Languages



What are the limits of PDAs?

PDA and compilers

- PDA are **at the heart of compilers**
- Stack memory has a LIFO policy
- LIFO is suitable to analyze **nested syntactic structures**
  - Arithmetical expressions
  - Begin/End
  - Activation records
  - Parenthesized strings
  - …

What are context-free languages?

Context-free languages and PDA

Context-free grammars have played a central role in compiler technology since the 1960s .... There is an automaton-like notation, called the "pushdown automaton", that also describes all and only the context-free languages.

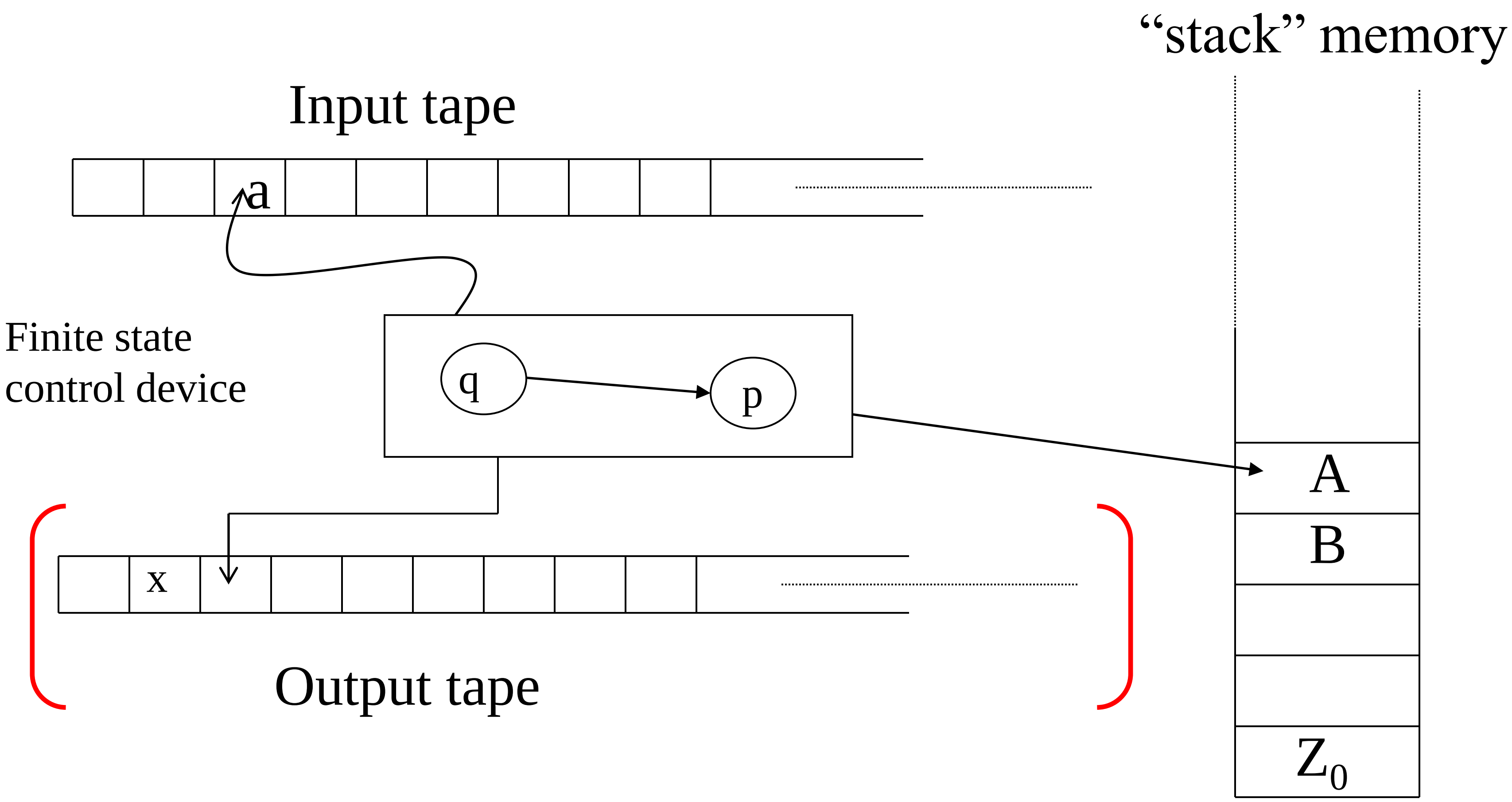
John E. Hopcroft, Rajeev Motwani
and Jeffrey D. Ullman

Theoretical Computer Science

PDA Transducers

Lecture 8 - Manuel Mazzara

Adding a (destructive) external memory



PD transducer, formally

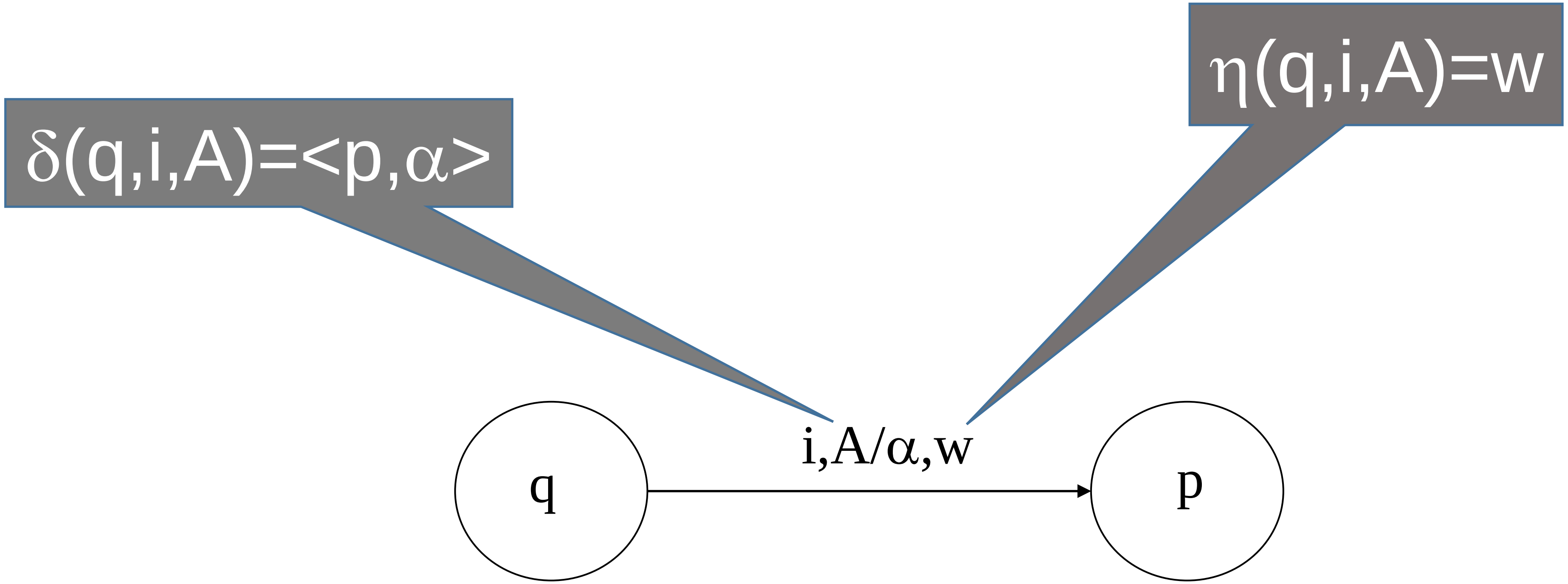
A PD transducer (PDT) is a tuple

$<Q, I, \Gamma, \delta, q_0, Z_0, F, O, \eta>$

Eta

- Q is a finite set of states
- …
- $F \subseteq Q$ is the set of final states
- O is the output alphabet
- $\eta: Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow O^*$

PD transducer, graphically



Remarks

- Q, I, Γ, δ, q_0, Z_0 and F are defined as in “acceptor” PDA
- η is defined only where δ is defined
- The stack can be necessary for two reasons:
  - The language to be recognized requires it
  - The translation requires it

Configuration

A configuration *c* is **$<q, x, \gamma, z>$**

- $q \in Q$ is the current state of the control device
- $x \in I^*$ is the unread portion of the input string
- $\gamma \in \Gamma^*$ is the string of symbols in the stack
- z is the string already written on the output tape

Transitions, formally

- If $\delta(q, \mathbf{i}, A) = <q',\alpha>$ is defined and $\eta(q, \mathbf{i}, A)=w$ then
  - $c=<q, iy, A\gamma, z> \mid\text{--}\ c'=<q', y, \alpha\gamma, zw>$

- If $\delta(q, \boldsymbol{\varepsilon}, A) = <q',\alpha>$ is defined and $\eta(q, \varepsilon, A)=w$ then
  - $c=<q, x, A\gamma, z> \mid\text{--}\ c'=<q', x, \alpha\gamma, zw>$

The difference is here!

Acceptance condition

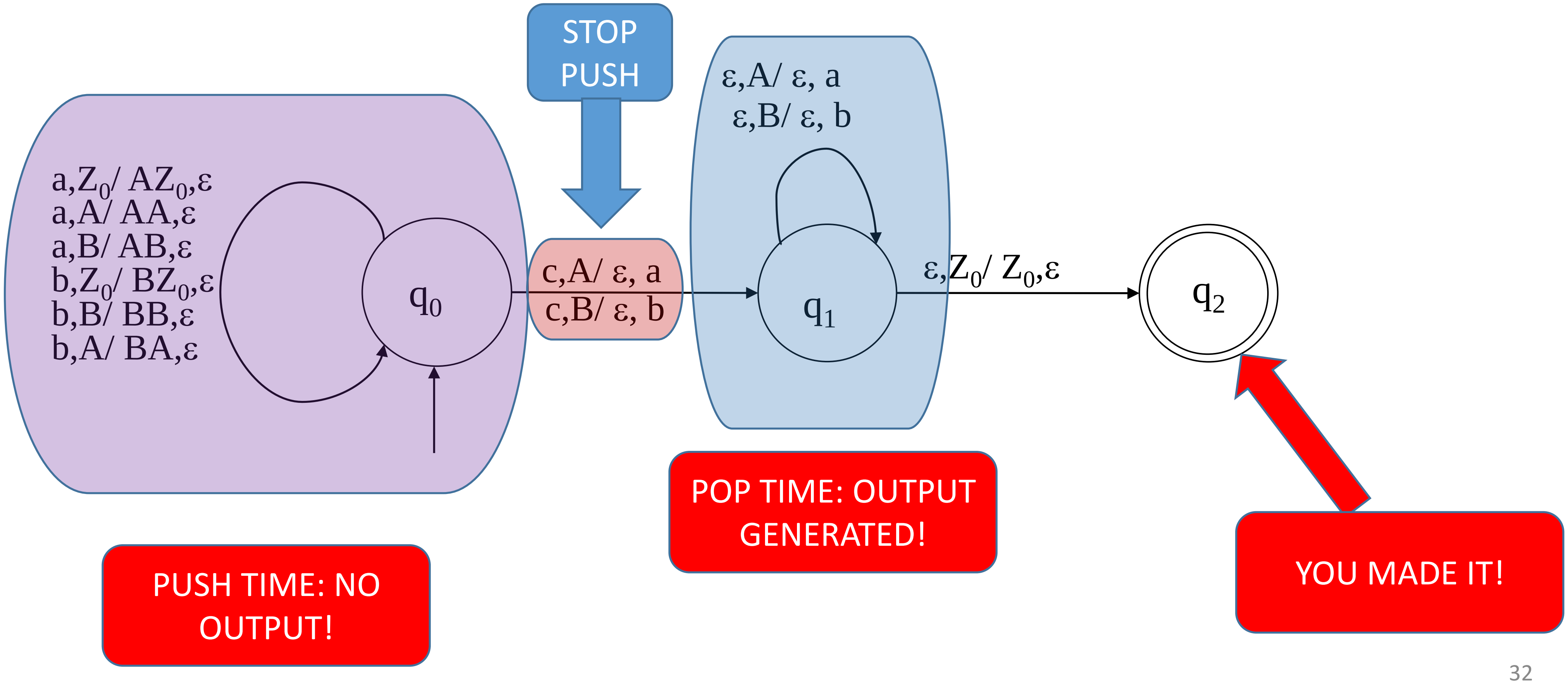
Defined like for FST

- $\forall x \in I^* \; (x \in L \wedge z = \tau(x) \Leftrightarrow c_0 = <q_0, x, Z_0, \varepsilon> \;|\!-^*\!-\; c_F = <q, \varepsilon, \gamma, z> \text{ and } q \in F)$

- Note: **the translation of *x* is defined only if the string *x* is accepted**

Example of Transducer

$L=\{wc \mid w \in \{a,b\}^{+}\}$ and $\tau(wc)=w^{R}$



Theoretical Computer Science

Operations on PDA

Lecture 8 - Manuel Mazzara

Closure properties

- **Closure properties** of languages accepted by deterministic PDA (by final state) are different than those of languages accepted by nondeterministic PDA
  - We will see this later with related implications

- **Deterministic PDA are closed under complementation**
  - The class of **deterministic context-free languages** is closed under complement
  - The class of **non-deterministic context free languages** is **not** closed under complement

- **Deterministic PDA are <u>NOT</u> closed under union**

PDA and complement

- **The class of languages recognized by deterministic PDAs is <u>closed under complement</u>**
- The complement can be algorithmically built by:
  - Eliminating loops
  - Completing δ
  - Swapping final and non-final states

Avoid infinite computations!

Same than FSA!

Acyclic PDA (preliminary definitions)

- Given a PDA we define: **$\langle q,x,\alpha\rangle |-^{*}-_{d} \langle q',y,\beta\rangle$**
- $|-^{*}-_{d}$ is a sequence of moves that needs a symbol to proceed
  - Remember the ε-move!
- $\langle q,x,\alpha\rangle |-^{*}- \langle q',y,\beta\rangle$ and for $\beta=Z\beta'$ $\delta(q',\varepsilon,Z)=\perp$ **(undefined)**
  - You cannot proceed without consuming an input symbol

Acyclic PDA

- A PDA is <u>acyclic</u> if and only if:
  - $\forall x \in I^* \ \langle q_0,x,Z_0\rangle \mid\!-^*\!-_d \langle q,\varepsilon,\gamma\rangle$ for some q and γ
  - **it always reads the whole input string and then stops**
  - **It does not loop after having consumed all the input**

- Every *deterministic* PDA can be transformed into an equivalent acyclic PDA

- The proof is a bit tricky, the idea is to do **loops elimination**
  - We will not show the proof here

PDA and complement

- **The class of languages recognized by deterministic PDAs is <u>closed under complement</u>**
- The complement can be algorithmically built by:
  - Eliminating loops
  - Completing δ
  - Swapping final and non-final states

Avoid infinite computations!

Same than FSA!

Loops elimination

- **Loops elimination** is essential, otherwise the end of the string may never be reached
- What if there are sequences of ε-moves traversing some final and some non-final states (and the input is entirely read)?
- With these precautions, we are sure that either the PDA or its complement will accept the string

Union

- The class of languages recognized by PDA is **not closed under union**

- There is no PDA that recognizes $\{a^nb^n \mid n\geq1\} \cup \{a^nb^{2n} \mid n\geq1\}$, but
  - $\{a^nb^n \mid n\geq1\}$ is recognizable by PDA ← We have seen this
  - $\{a^nb^{2n} \mid n\geq1\}$ is recognizable by PDA ← Straightforward consequence

- Idea: after a sequence of a's, it is not possible to deterministically decide whether start processing single b's or pairs of b's
  - A nondeterministic PDA can do instead

Not a **deterministic** context-free language

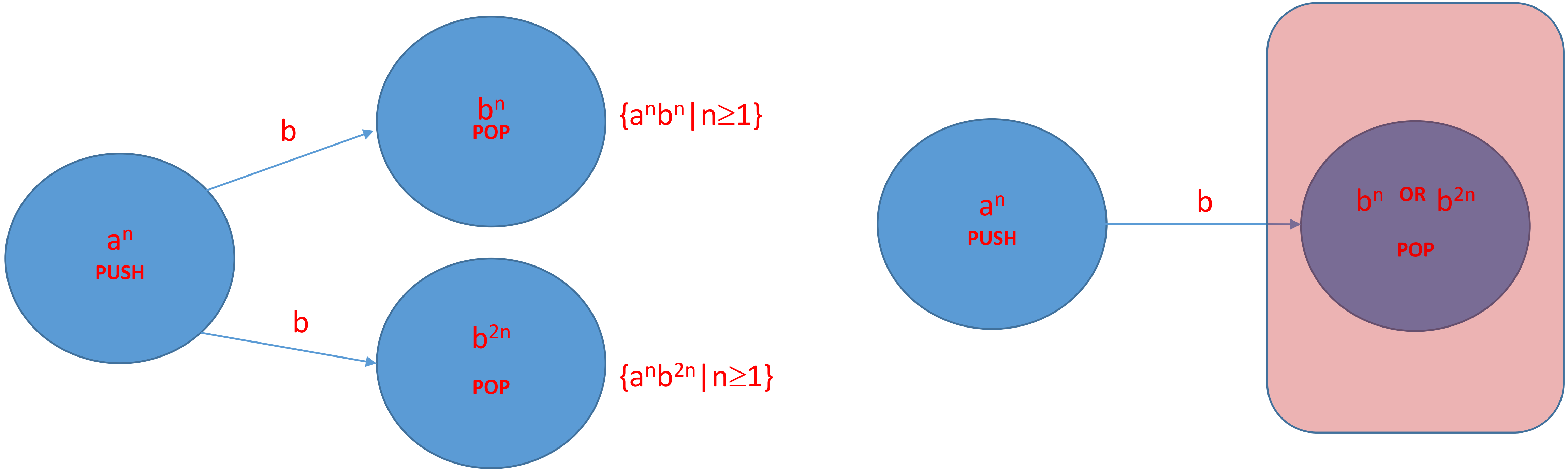
$$L=\{a^nb^n|n \geq 1\} \cup \{a^nb^{2n}|n \geq 1\} \text{ is a CFL and not a DCFL}$$

- We can design a nondeterministic PDA that recognize L
- We **cannot** design a deterministic PDA that recognize L

- It is easy to construct a NPDA for $\{a^nb^n : n \geq 1\}$
- And for $\{a^nb^{2n} : n \geq 1\}$

- These two can be joined by a new start state and epsilon-transitions to create a NPDA for L

Nondeterminism and proof sketch



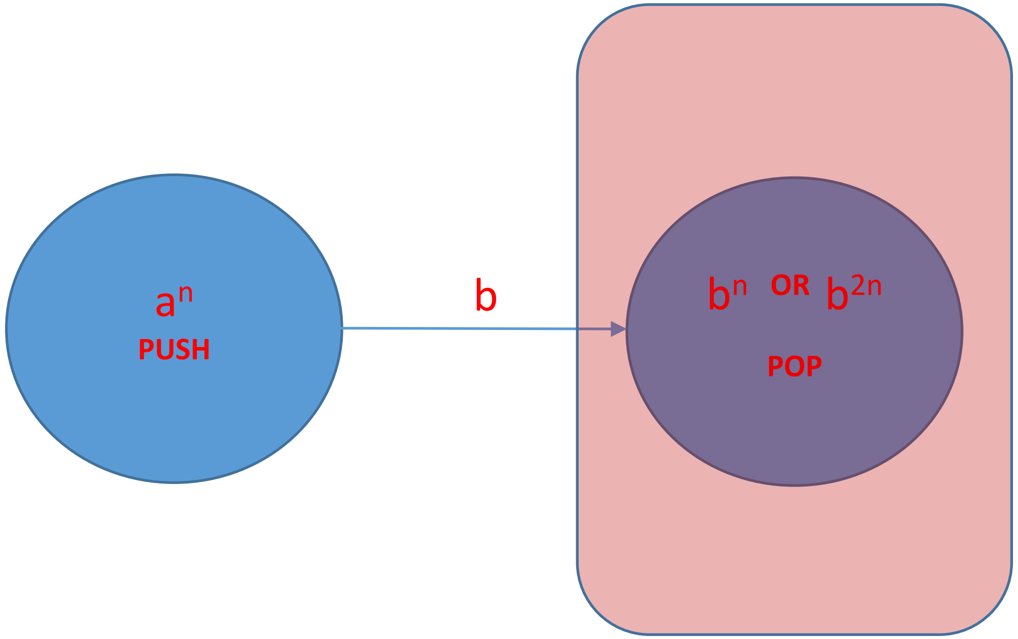
Why?

- **The stack has to be prepared in the push phase to be ready to manage the pop phase**
- If not prepared accordingly we are not able to manage the second part of the input string
- A nondeterministic version would just manage the two possibilities in parallel

Intuitive proof sketch

- The stack is emptied after **n** pops of a*'s* and scanning b's from input:
  - **There are more b's than a's**
  - if there are other *b's* in input: IT IS NOT b^n
  - It may be b^{2n} then but the content of the stack is lost
  - **n** is forgotten – we cannot count anymore!
  - Could be b^{3n} or anything

- The stack is not emptied after popping a's and scanning b's from input:
  - **There are more a's than b's**
  - You previously pushed 2 *a*'s on stack for each *a*'s in input to find a b^{2n} but it is not b^{2n}
  - It may be b^n then but…
  - If there are no more *b*'s in input, it is impossible to know whether the stack contains exactly n symbols



Another example of non-closure

$L_1 = \{a^i b^j c^k, i, j, k \geq 0 \text{ and } i \neq j\}$ (recognizable by PDA)

$L_2 = \{a^i b^j c^k, i, j, k \geq 0 \text{ and } j \neq k\}$ (recognizable by PDA)

$L_3 = L_1 \cup L_2 = \{a^i b^j c^k, i, j, k \geq 0 \text{ and } ((i \neq j) \text{ or } (j \neq k))\}$

Is L_3 recognizable by PDA?

Intersection and difference

- The class of languages recognized by PDAs is **not closed under intersection**
  - $A \cup B = (A^c \cap B^c)^c$
  - Since PDA languages are closed under complement, if they were closed under intersection, they should be closed under union as well
- The class of languages recognized by PDAs is **not closed under difference**
  - $A \cap B = A - B^c$
  - Since PDA languages are closed under complement, if they were closed under difference, they should be closed under intersection and union as well

Recap

- We have seen that the union of languages recognized by PDAs cannot be recognized by any PDA
  - **PDAs are not closed under union**
- Example:
  - $L_1=\{a^nb^n|n\geq 1\}$
  - $L_2=\{a^nb^{2n}|n\geq 1\}$
  - … but $L_1 \cup L_2$ is not recognizable by any PDA

Operations	Regular	Context-free	Deterministic CF
union	yes	yes	no
intersection	yes	no	no
complement	yes	no	yes

Closure of operations

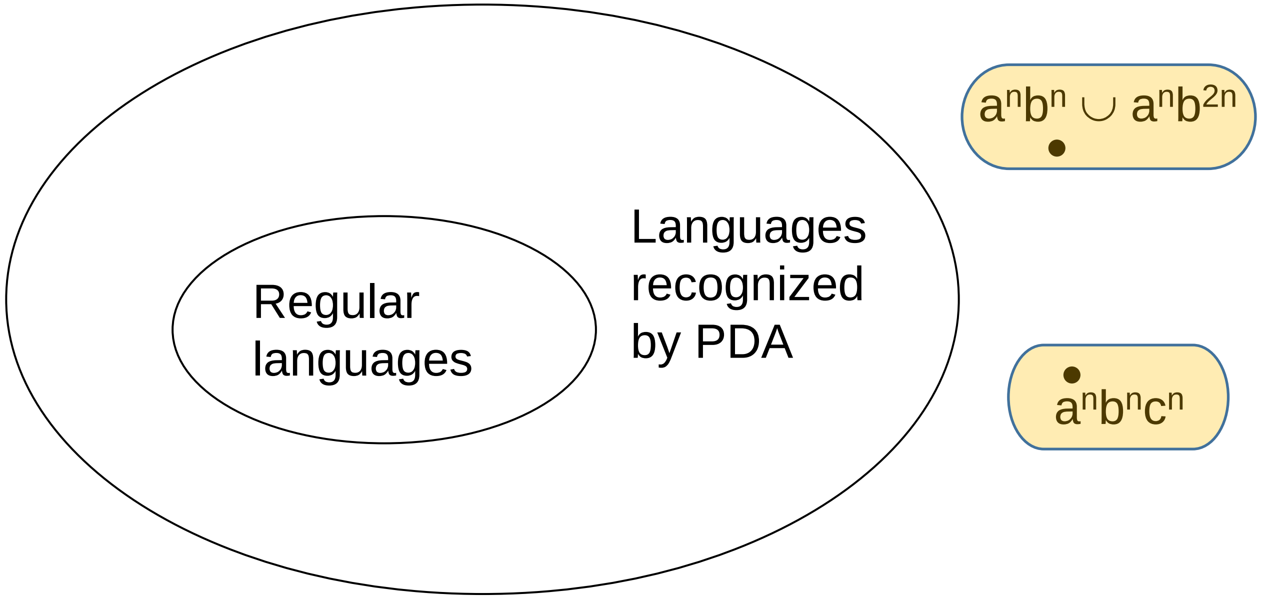
Another language non recognizable by PDA

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

- The stack can be used to count the *a*'s
- The symbols on the stack can be used to check that the number of *b*'s is equal to the number of *a*'s
- How can **n** be remembered so as to check the number of *c*'s?

This language is also used in the formal proof to show that $\{a^n b^n | n \geq 1\} \cup \{a^n b^{2n} | n \geq 1\}$ is not a deterministic context-free language

Languages



What are the limits of PDAs?

Remarks

- The stack is a **destructive memory**
  - Once a symbol is read, it is destroyed
- The limitation of the stack can be proved formally through a **generalization of the pumping lemma** (lemma of Bar-Hillel)
- It is necessary to use **persistent memory**

  $\rightarrow$ **memory tapes and TM**

Theoretical Computer Science

Automata Theory and Models of Computation

Lecture 8 - Manuel Mazzara

Who is him?

Homer

Real character vs. mythological (850 BCE?)

Iliad and *Odyssey*

Believed to be the first and greatest of the epic poets

Author of the first known literature of Europe

Why should we mention him?

Automata Theory

It regards:

- **The study of abstract mathematical machines (automata)**
- **The computational problems that can be solved by them**

Automaton (singular), Automata (plural)

Latinization of the Greek αὐτόματον (automaton): *self-moving*

- something is doing something by itself

The word automaton was first used by *Homer*

- describing automatic door opening
- automatic movement of wheeled tripods
- moving statues…

Why studying Automata Theory?

An automaton is a ***finite*** representation of a formal language that may be ***infinite***

Theoretical models for computing machines to be used for proofs about computability

Model of computation

- A **mathematical model of computation** describes how
  - a set of **outputs are computed** given a set of inputs
  - units of computations, memories, and communications are **organized**

- Theory: **automata theory**, **computability** and **computational complexity**

- Practice: **system specification**, compiler construction…

Different Models of computation

- Sequential
  - **Finite state automata**
  - **Pushdown automata**
  - Turing Machine
- Functional
  - Lambda calculus
- Concurrent
  - Petri nets
  - …
- **This list is not exhaustive**

Example: FSA

- Simple **model of computation**
- **Limited expressiveness**
    - Fixed memory
- Suitable to “brute force” analysis
    - **Model Checking**