Theoretical Computer Science

Administrative Information

Lecture 6 - Manuel Mazzara

Mid-term Exam

When : Wednesday, 15 March 2023, ~12:40-14:10

Where : 106, 107, 108 (precise instructions later)

What: **Formal Languages, FSA, Pumping Lemma, PDA**

List of topics for mid-term

- Finite State Automata
- Finite State Transducers
- Operations on FSA
- Regular Languages
- Pumping Lemma
- Pushdown Automata

Assignment 1

FSA Validator

Design and coding exercise where you have to demonstrate an understanding of the basic principles and functioning of FSAs

In Moodle – please respect the deadline!

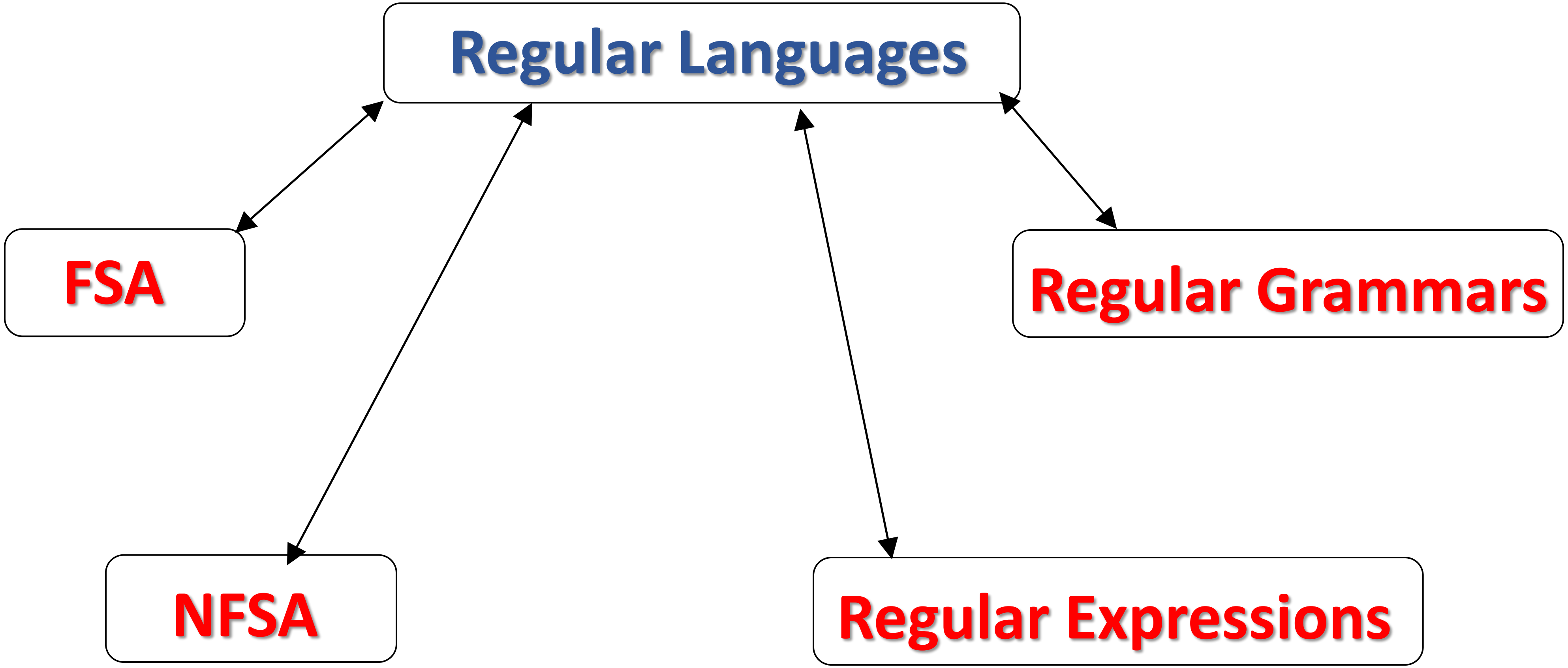
There will be Assignment 2 around Week 12 (release) and 14 (submission)

Theoretical Computer Science

Regular Languages and Pumping Lemma

Lecture 6 - Manuel Mazzara

Representations of Regular Languages



Examples of non regular languages

$$\{a^n b^n : \; n \geq 0\}$$

$$\{ww^R : \; w \in \{a,b\}^*\}$$

- **How can we prove that a language is not regular?**

- Can we prove that there is no FSA that accepts it?

- This is not easy to prove!

Pumping Lemma

Pumping Lemma: <u>**formal statement**</u>

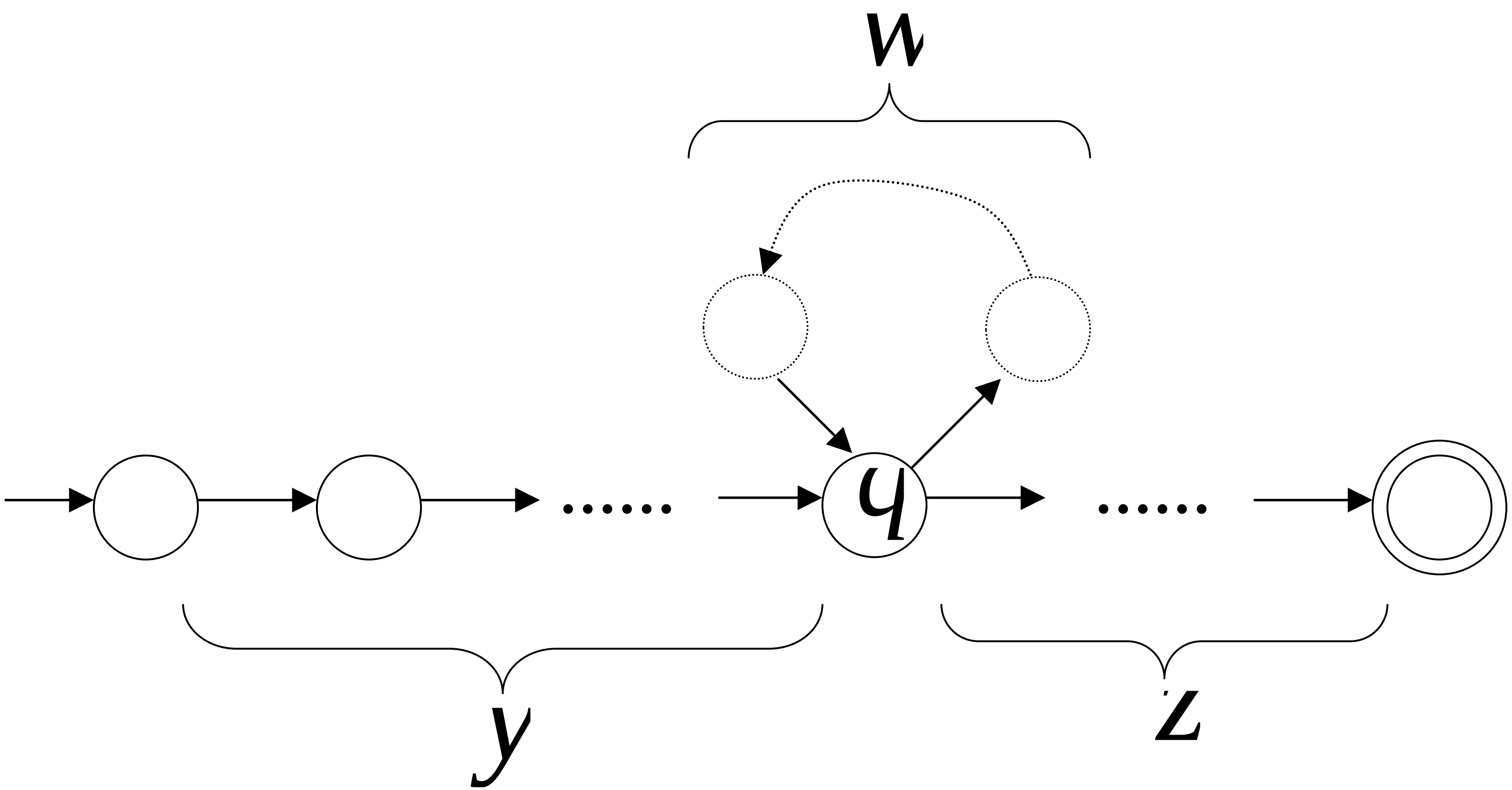
- Given a *regular language* **L**, If **x** $\in$ **L** and |**x**| $\geq$ |**Q**|, then there exists a q $\in$ Q and a **w** $\in$ I^+ such that:
  - $x = ywz$
  - $\delta^* (q_0,y) = q$
  - $\delta^* (q,z) = q' \in F$
  - $\delta^* (q,w) = q$
  - $|yw| \leq |\mathbf{Q}|$
  - $yw^nz \in L, \forall n \geq 0$

For languages that are not regular these conditions does NOT have to apply. However, some at time may apply.

- This is the *Pumping Lemma* (one can "pump" **w**)

The pumping lemma is a <u>necessary but not sufficient</u> condition for a language to be regular

Pumping Lemma, graphically



Levels of expressiveness

- In order to "count" an ***arbitrary n*** we need an infinite memory!

- **Fixed vs finite**

- From the toy example $\{a^n b^n\}$ to more concrete cases:
  - Checking **well-balancing of brackets** (typically used in programming languages) cannot be done with fixed memory

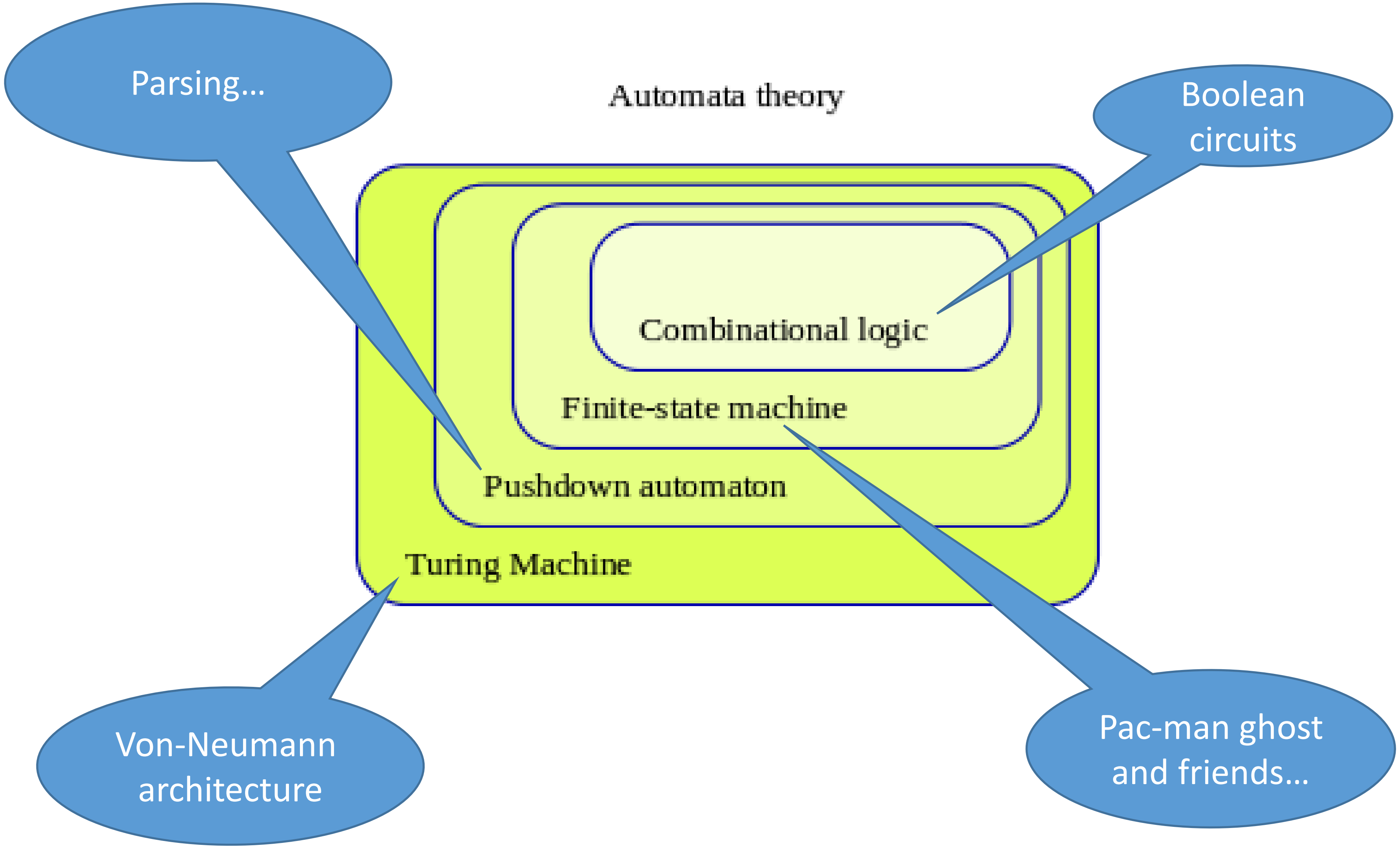
- We therefore need more powerful models (**PDA**)

Theoretical Computer Science

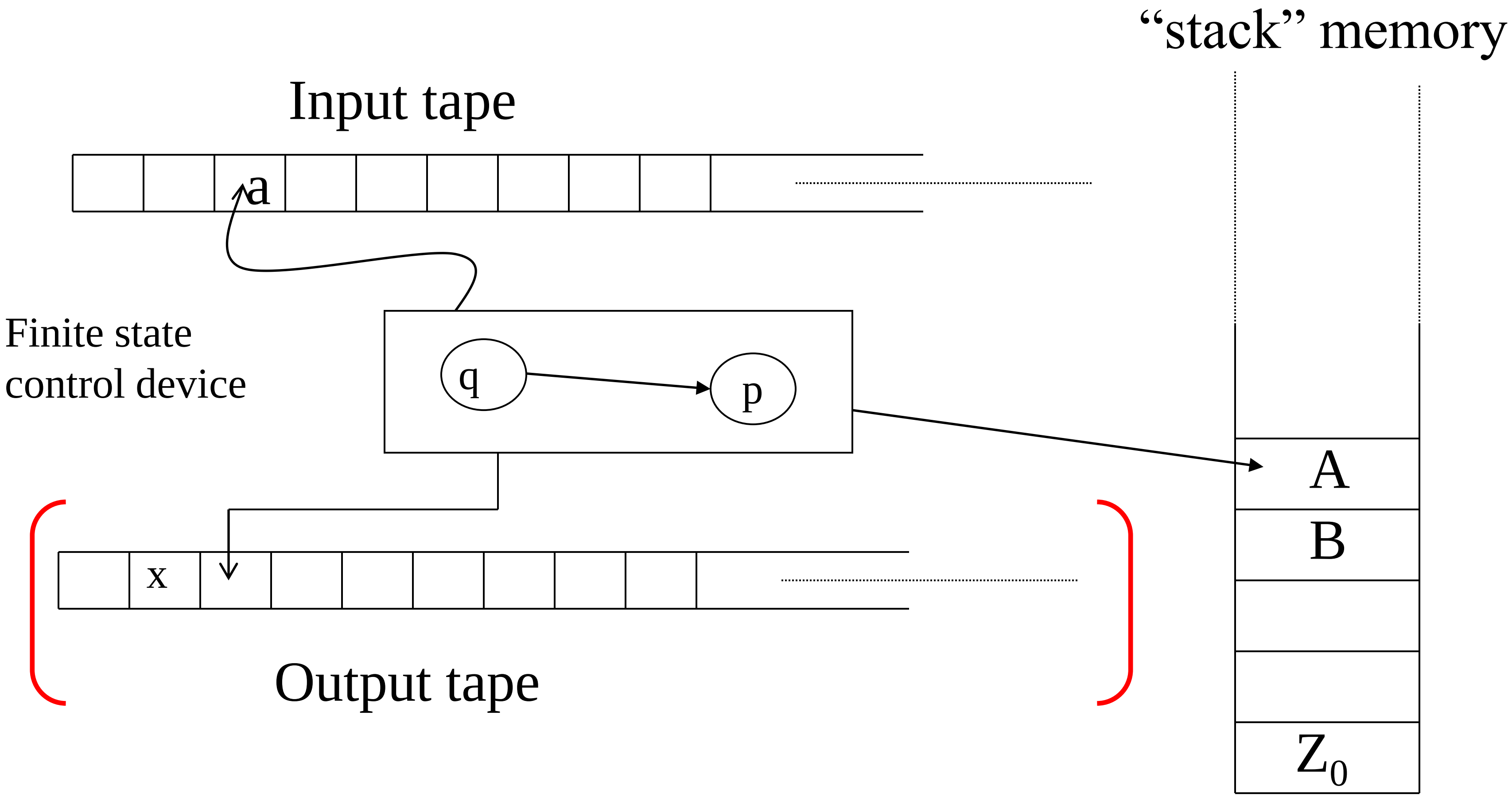
Pushdown automata

Lecture 6 - Manuel Mazzara

A bit of context



Adding a (destructive) external memory



Pushdown automata

- Finite state automata can be enriched with a stack

  → Pushdown Automata (PDA)
- PDAs differ from finite state machines in two ways:
  - They can **use the top of the stack to decide which transition has to be made**
  - They can **manipulate the stack** as part of performing a transition

Moves of a PDA

Depending on

- the symbol read from the input (but it could also read nothing)
- the symbol read from the top of the stack
- the state of the control device

the PDA

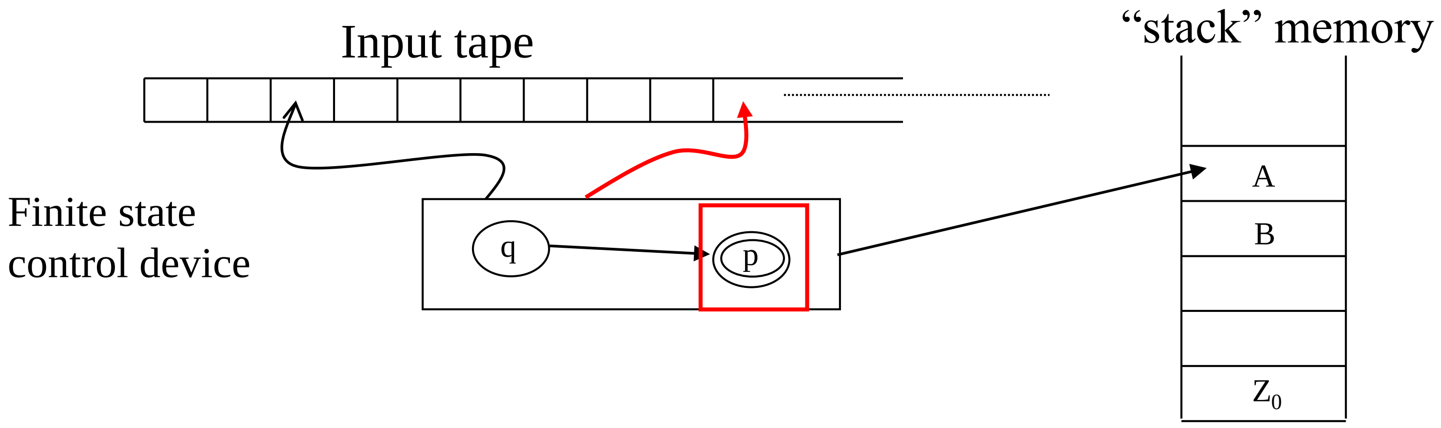
- *changes its state*
- *moves ahead the scanning head*
- *changes the symbol read from the stack with a string α (possibly empty)*

Acceptance

The input string **x** is recognized (accepted) if

- the PDA scans it completely
- upon reaching the end of **x**, it is in an **accepting state**

This is one of the possible definitions of "acceptance"



Acceptance, in general

- Acceptance by **“Final State”**
  - Input is consumed and PDA is in a final state
- Acceptance by **“Empty Stack”**
  - Input is consumed and stack is empty
- **Not equivalent** for the deterministic pushdown automaton
- They are equivalent for the *non-deterministic* pushdown automaton

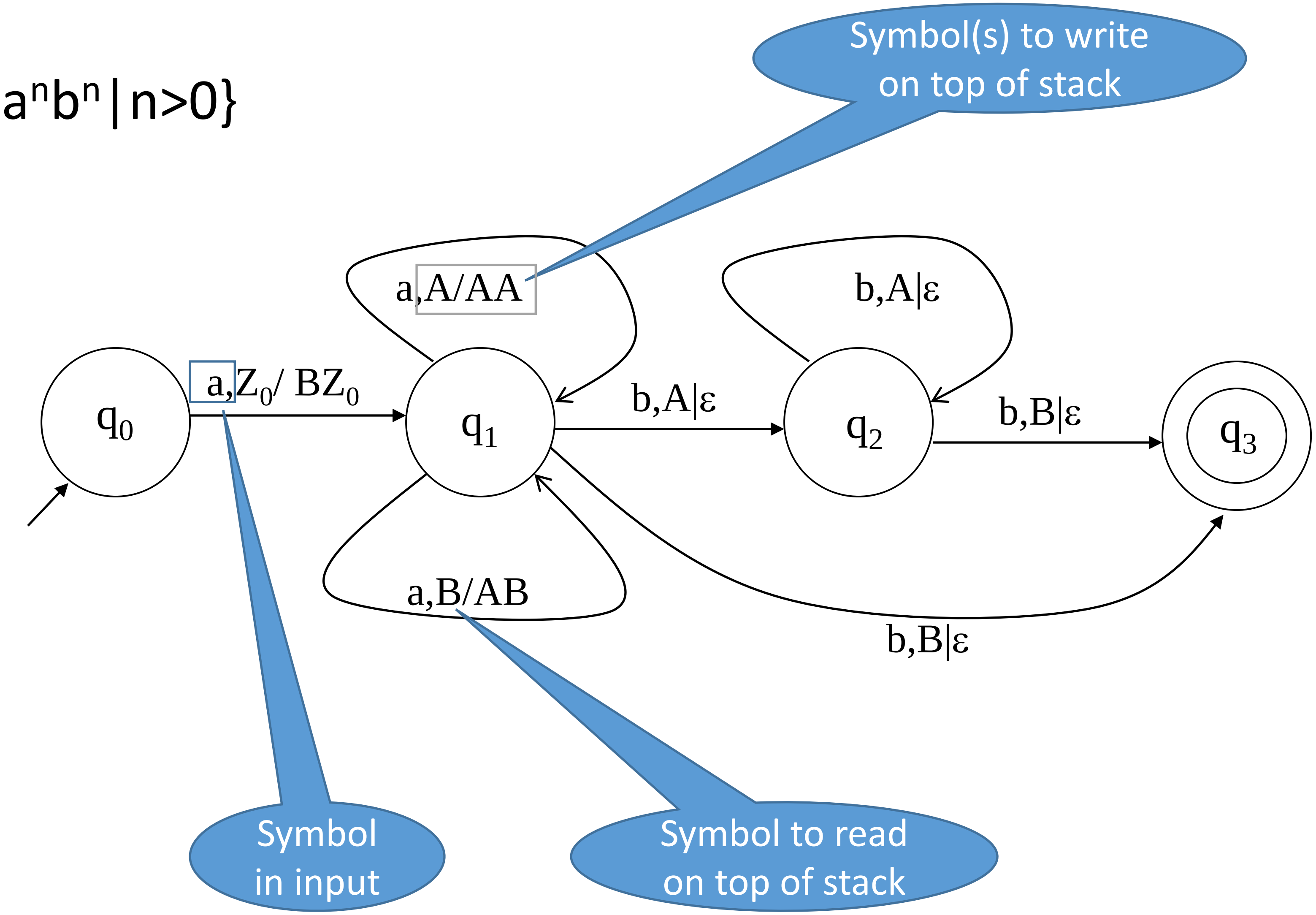
We will see this later

A little “hint”

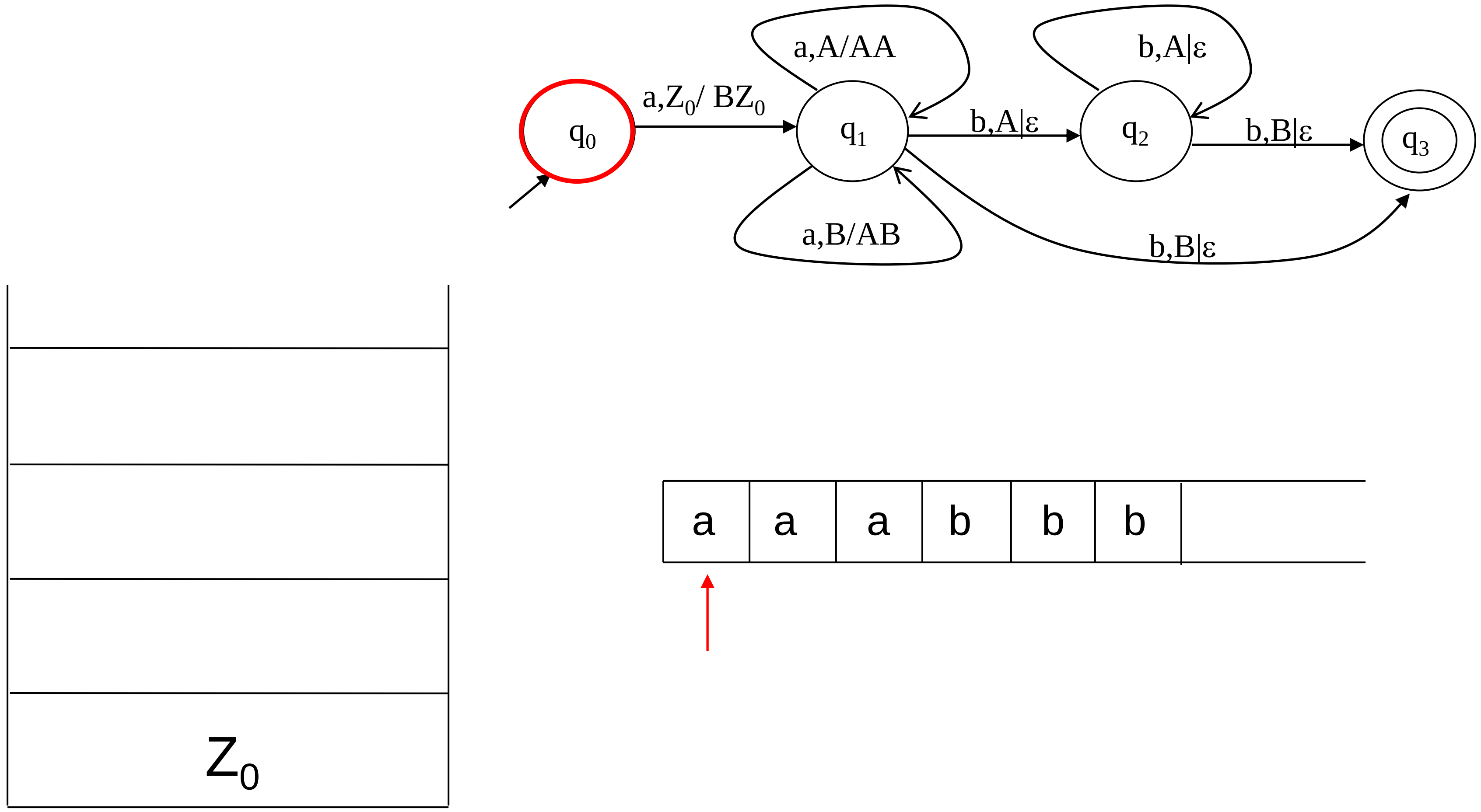
- The languages accepted by *empty stack* are:
  - the languages that are accepted by *final state*
  - have no word in the language that is the prefix of another word in the language

PDA: a first example

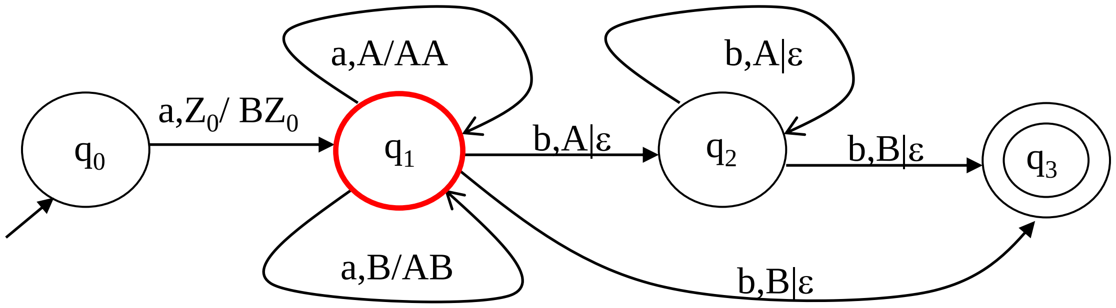
$L=\{a^n b^n \mid n>0\}$

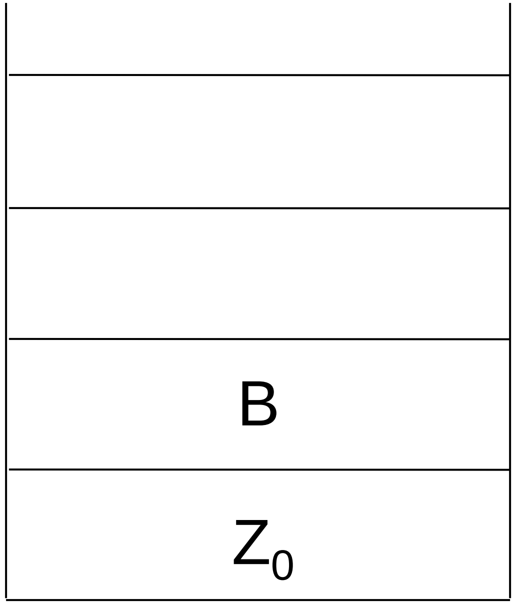


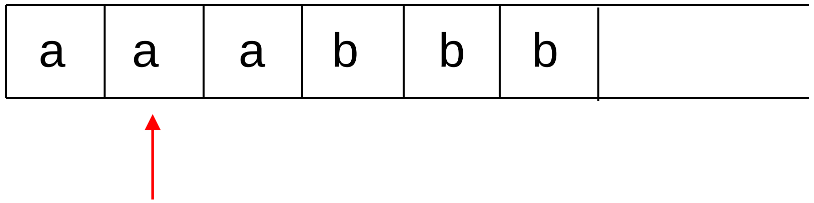
Example (animated)



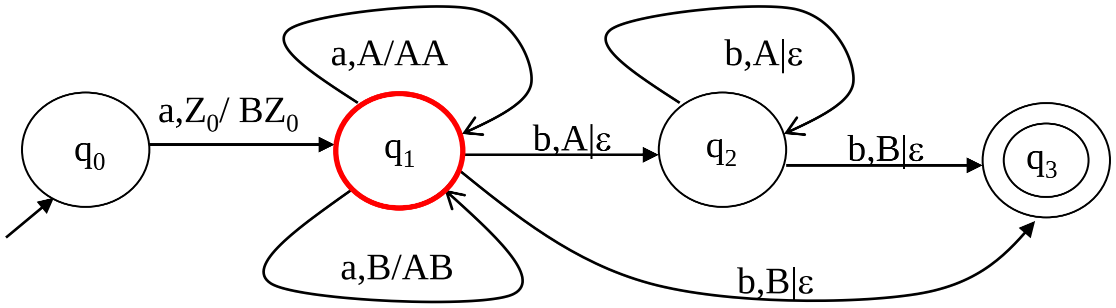
Example (animated)

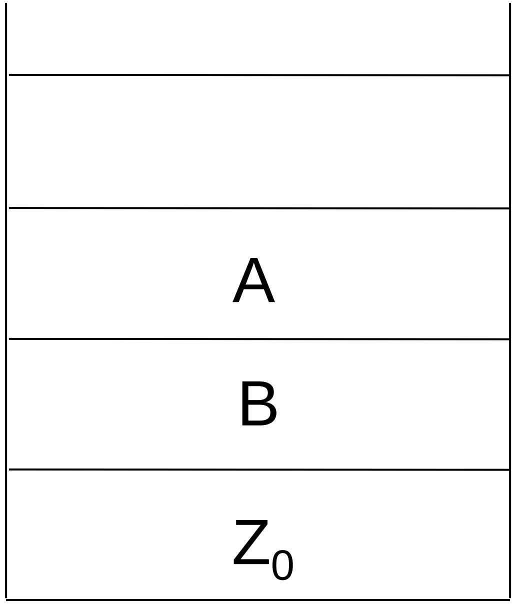


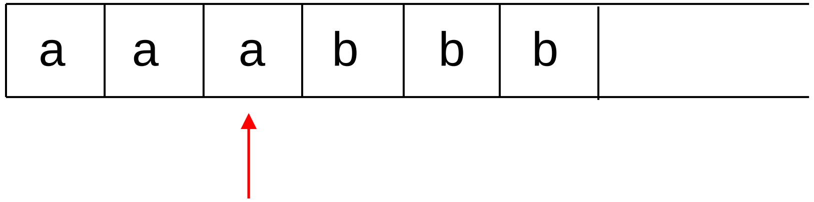




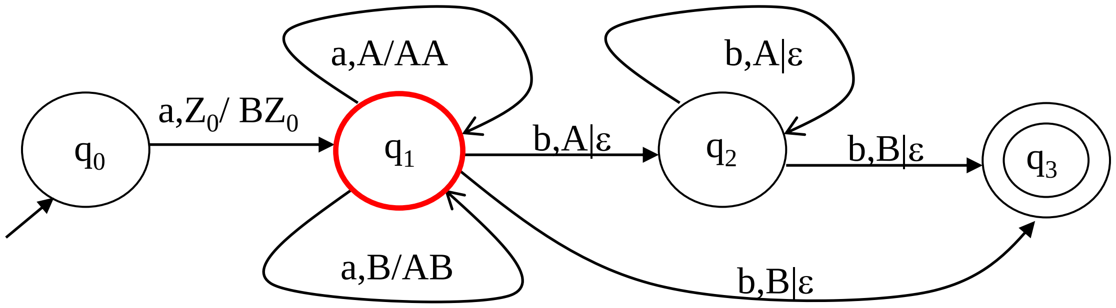
Example (animated)

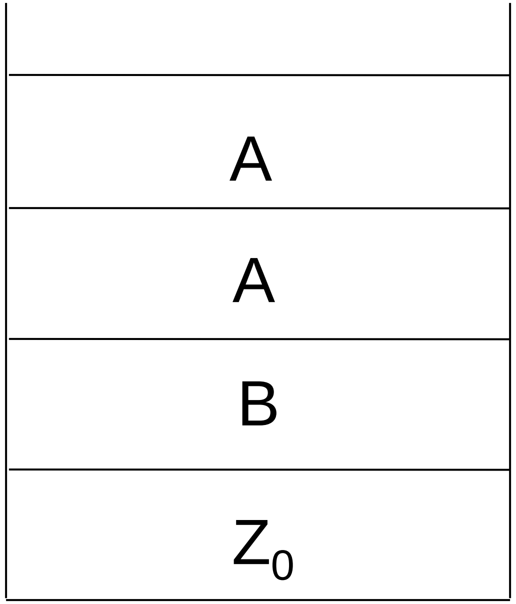


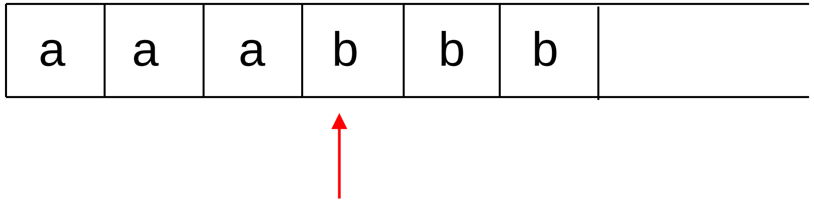




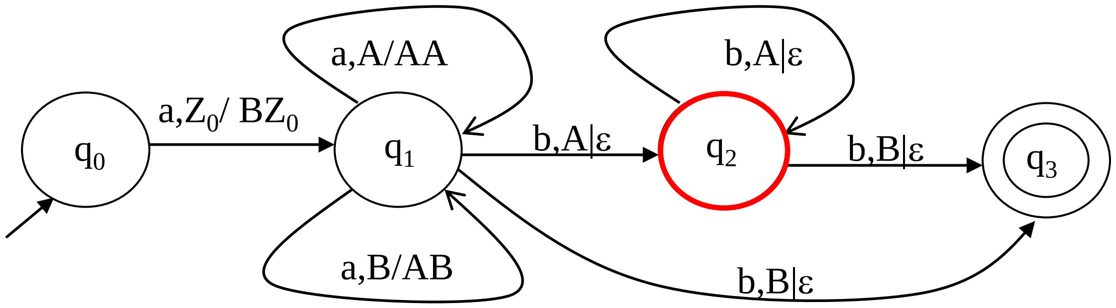
Example (animated)

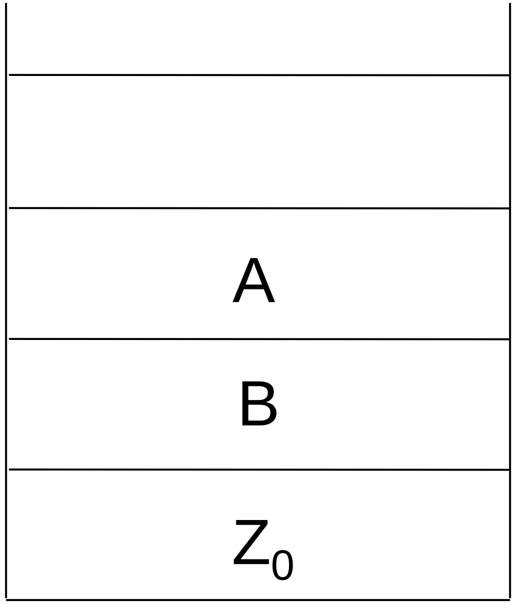


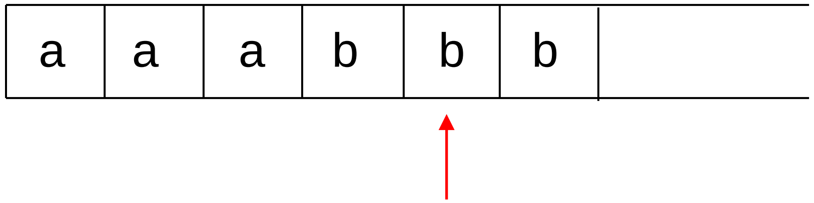




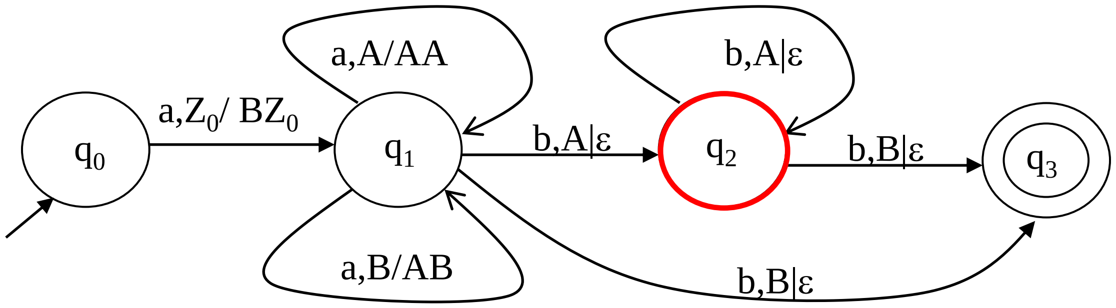
Example (animated)

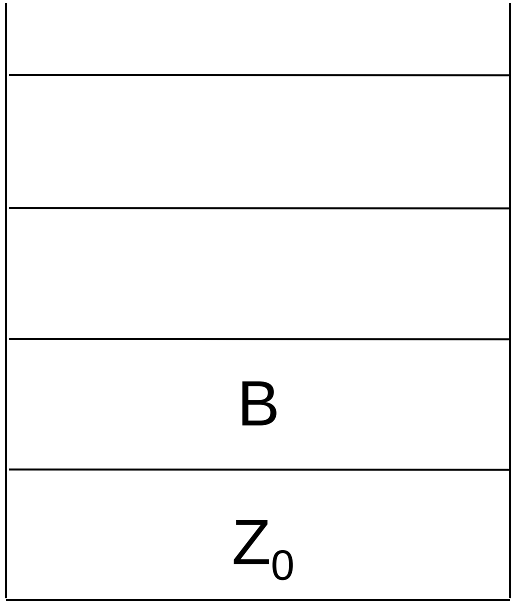


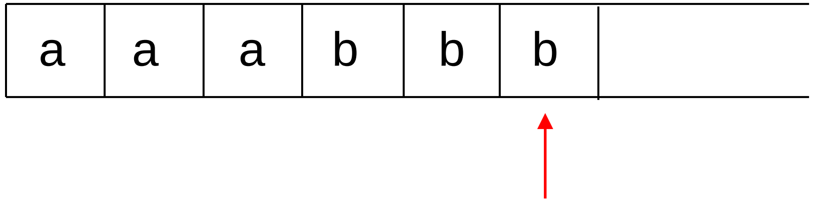




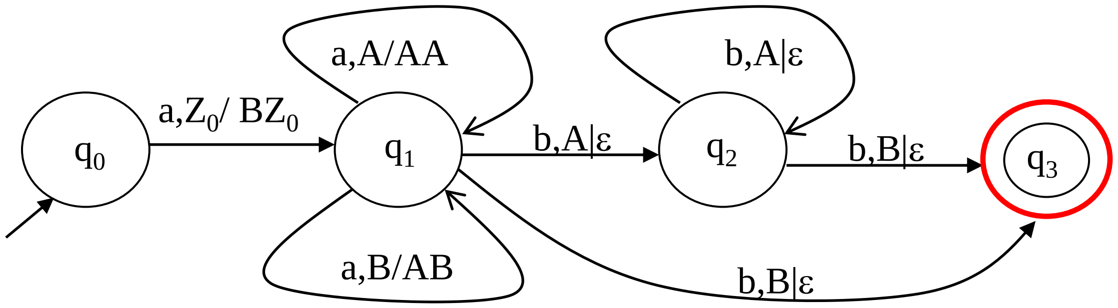
Example (animated)

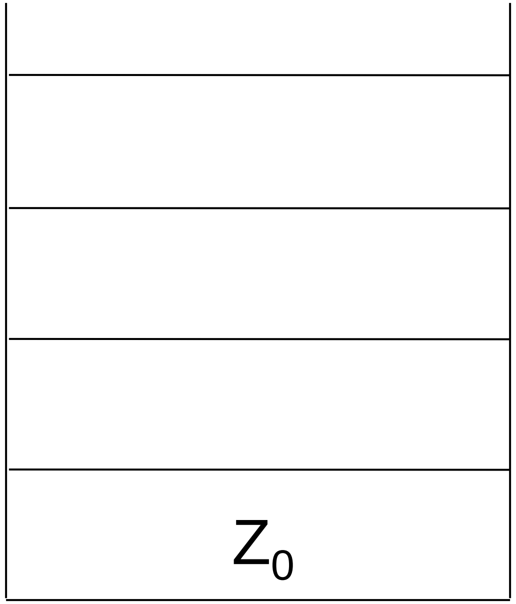


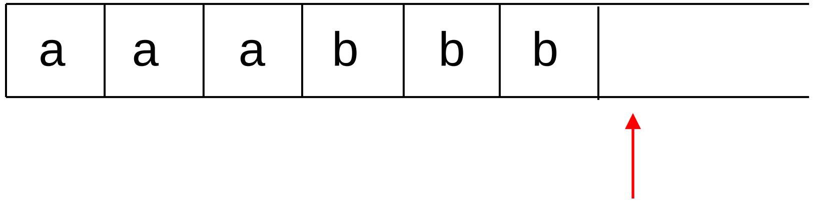




Example (animated)







Theoretical Computer Science

PDA in context

Lecture 6 - Manuel Mazzara

PDA vs FSA (1)

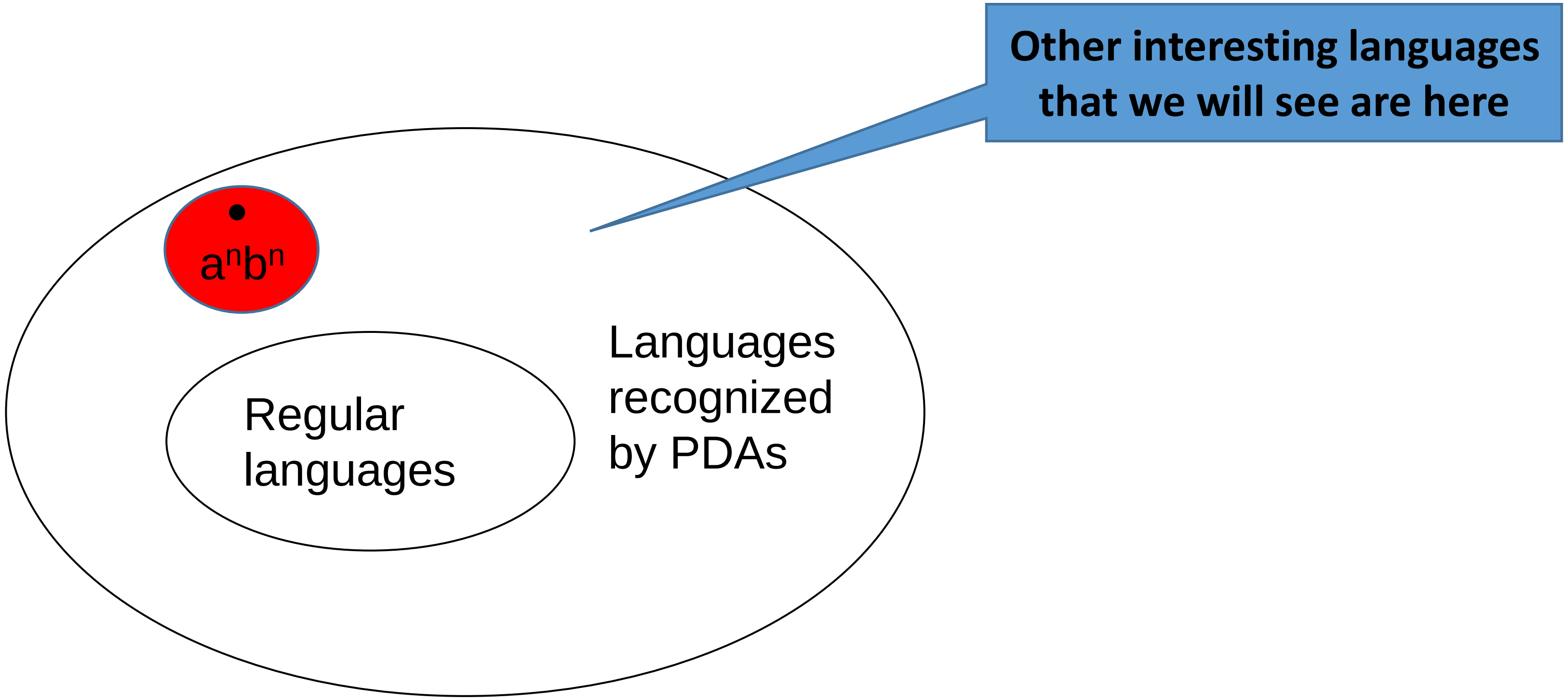
- We know that $a^n b^n$ cannot be recognized by any FSA
  - **Pumping Lemma**

  but <u>it can be recognized by a PDA</u>

- **Every regular language can be recognized by a PDA**
  - Given an FSA $A=<Q,I,\delta,q_0,F>$ it is straightforward to build a PDA $A'=<Q',I',\Gamma',\delta',q_0',Z_0',F'>$ such that $L(A)=L(A')$

PDA vs FSA (2)

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



PDA vs FSA (3)

- **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 <u>fixed memory</u>**
- **PDA has <u>finite</u> but not fixed –**
- **It is using an <u>unbounded</u> amount of memory**

PDA vs FSA (4)

- **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 with no bound</u>

PDA and compilers

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

Balanced Parentheses

Intuitively, a string of parentheses is *balanced* if each left parenthesis has a matching right parenthesis and the matched pairs are well nested. The set PAREN of balanced strings of parentheses [ ] is the prototypical context-free language and plays a pivotal role in the theory of CFLs.

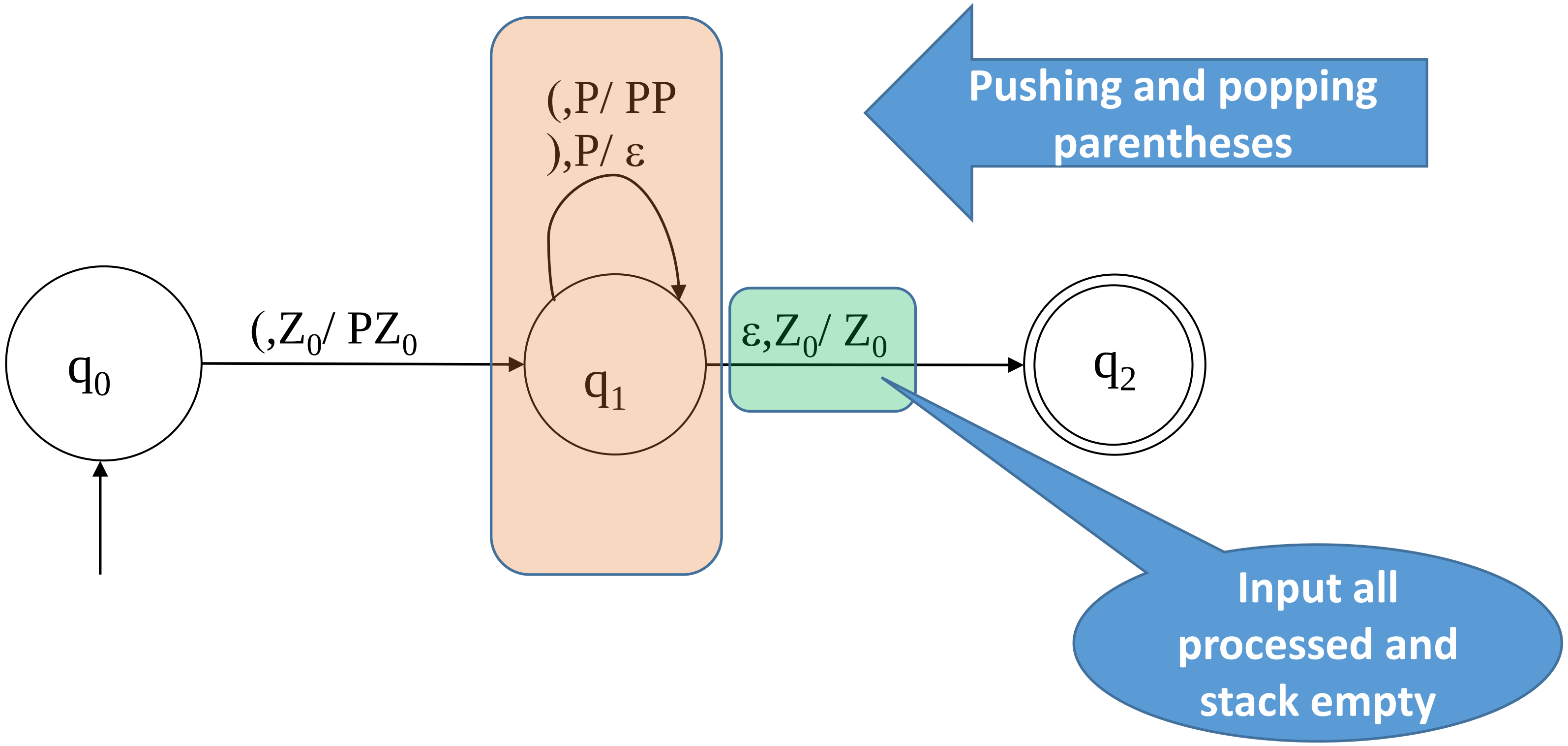
Historical Notes

The pivotal importance of balanced parentheses in the theory of context-free languages was recognized quite early on.

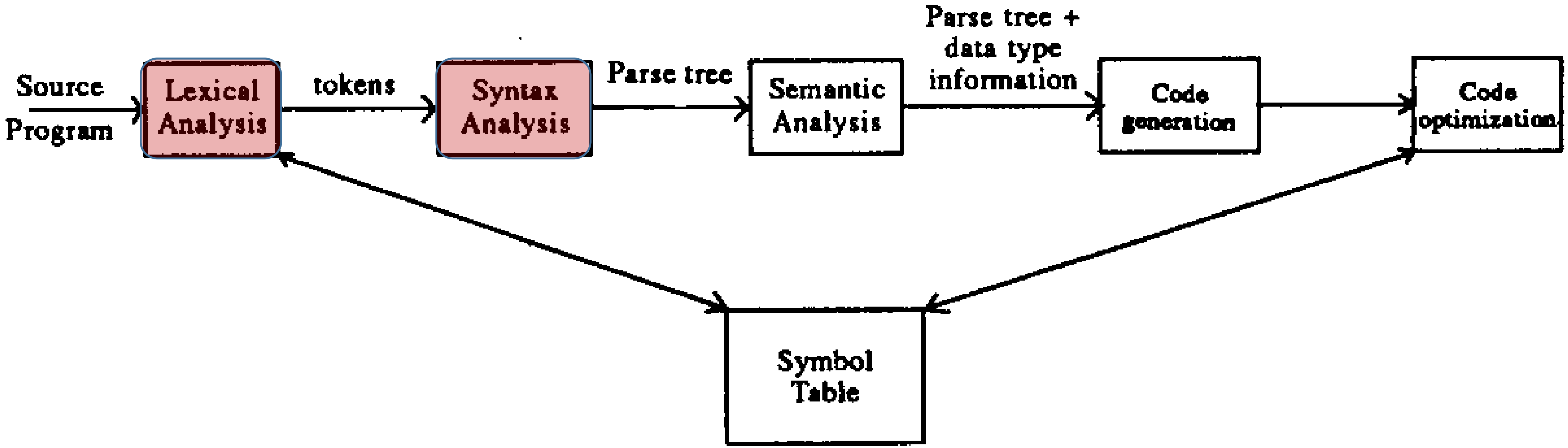
Dexter Kozen, *Automata and Computability*, Springer-Verlag , 1997

Well parenthesized strings

- PDA to recognize well parenthesized strings
  - (()((()()))) OK
  -)()))())) NO



General Structure of a Compiler



You will study this in Compilers course

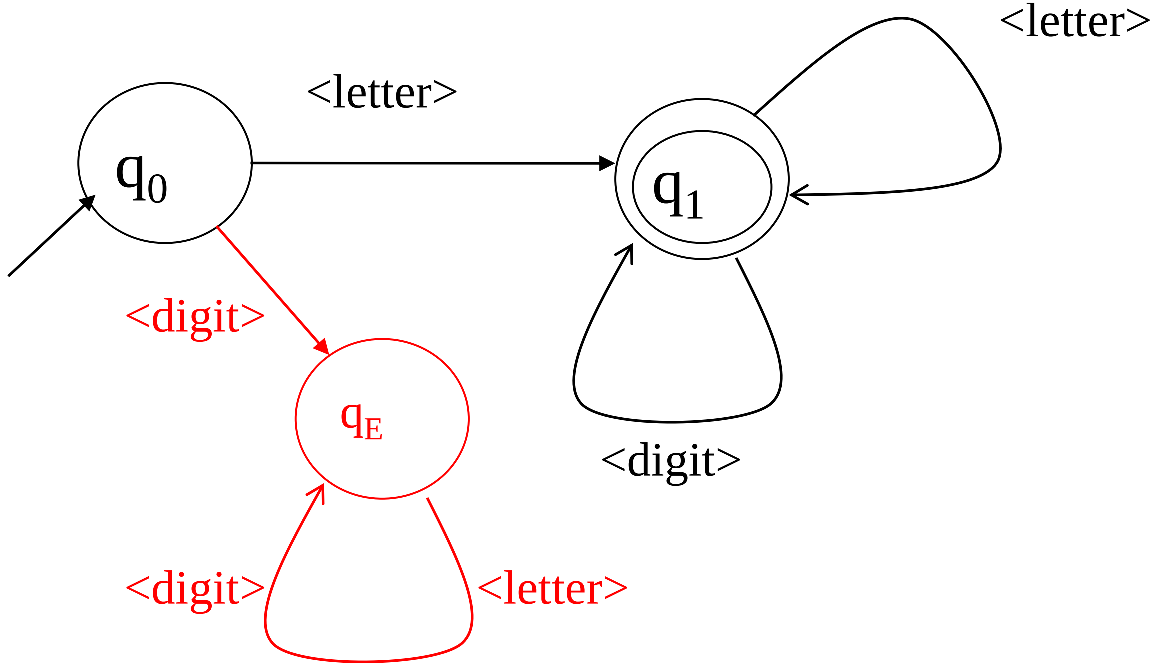
Lexical Analysis

- **Lexical analysis** (lexing/scanning) breaks the source code text into small pieces
  - *Tokens*
  - Single atomic units of the language
  - Keywords, identifiers ...

From Greek *lexikos* ‘of words’ (from *lexis* ‘word’)

- **The token syntax is typically a regular language**
  - **Finite State Automaton**, Regular expressions
  - This compiler part is called ***lexer***

Pascal identifiers



Syntax Analysis

- PDA is the most important class of automata between FSA and TM
- FSA cannot even recognize a simple language such as $a^n b^n$
- **Nested structures are the key of programming languages**
- Specific (nondeterministic) PDAs are used in ***Syntax Analysis/parsing***

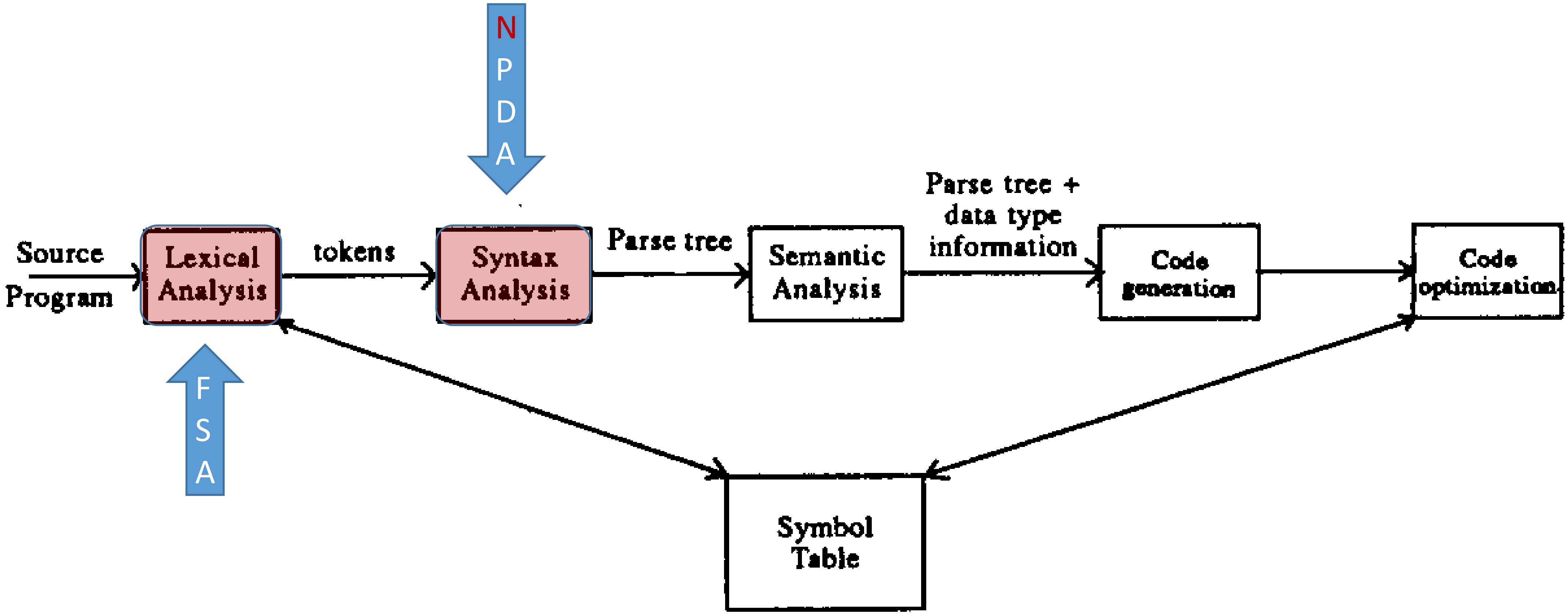
From Latin *pars* (*orationis*): part (of speech)

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.

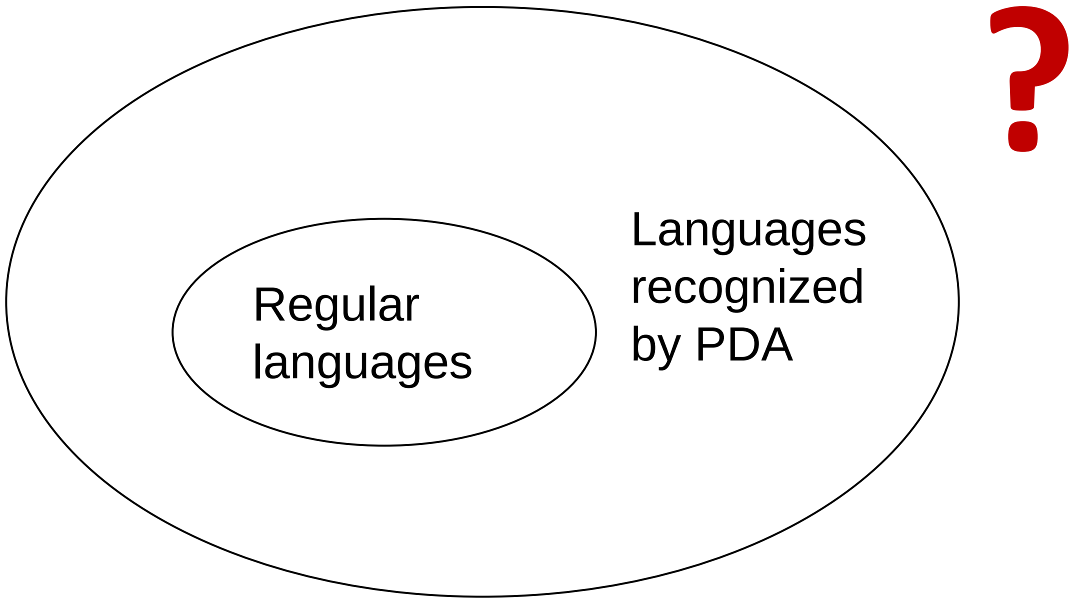
John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman

Very roughly…



You will study this in Compilers course

Everything seems under control…



Are there <u>languages that **cannot** be recognized by PDAs</u>?

The short answer

- The short answer is **yes**:
  - **<u>there are languages that cannot be recognized by PDAs</u>**
- We will look into the details!
- We will also look into the details of PDA formalization
  - Configuration
  - Transitions
  - Transducers

Theoretical Computer Science

PDA, formally

Lecture 6 - 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

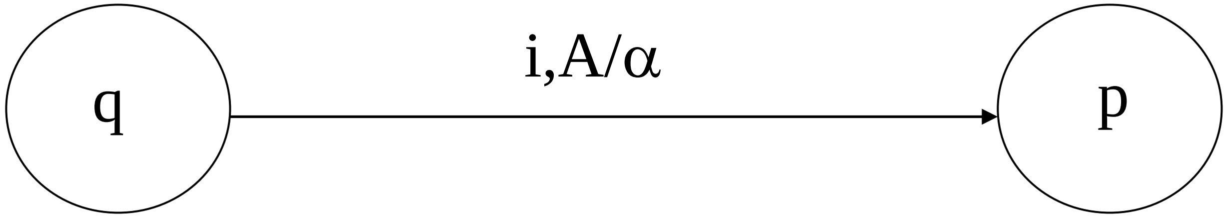
A PDA, formally

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:

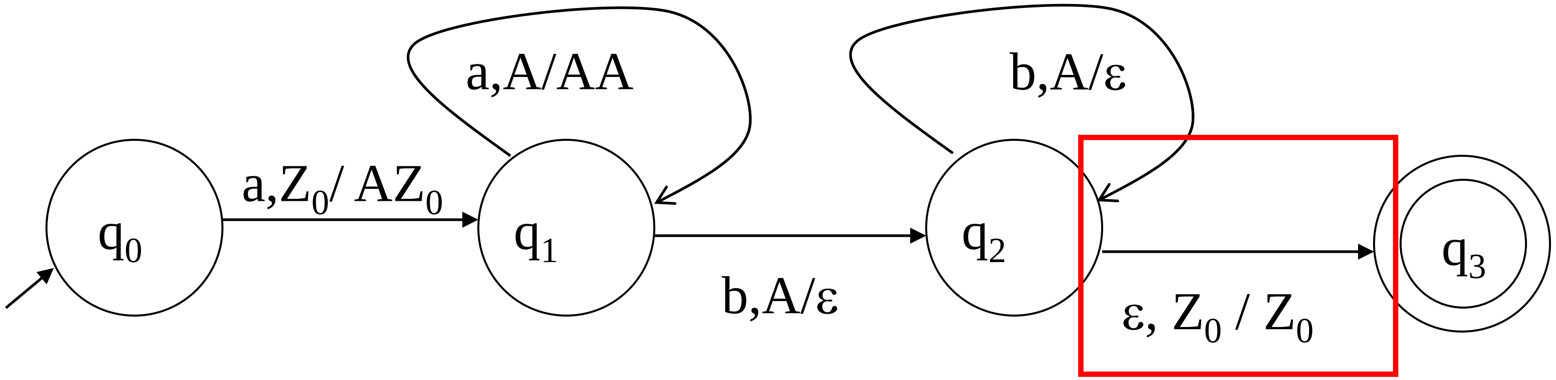


Remarks

- Q, I, q_0 and F are defined as in FSAs
- δ is a <u>partial function</u>
- Z_0 is the initial symbol of the stack, but it is not essential
  - It is useful to simplify definitions
- $\delta(q, \varepsilon, A)=\langle p, \alpha \rangle$
  - An "ε move" is a <u>spontaneous move</u>
  - **ε does not mean that the input is empty!**

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

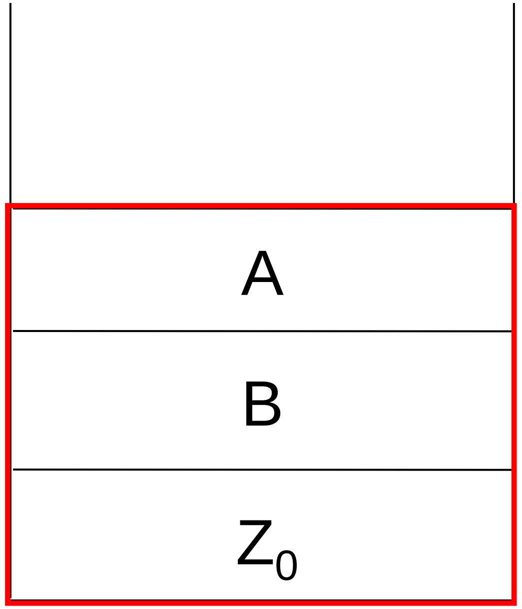
Configuration, formally

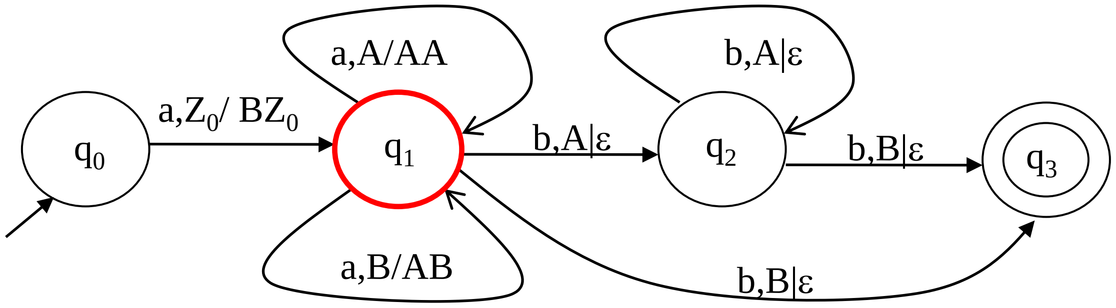
- A configuration c is **$<q, x, \gamma>$**
  - $q \in Q$ is the current state of the control device
  - $x \in I^*$ is the unread portion of the input string
  - $\gamma \in \Gamma^*$ is the string of symbols in the stack

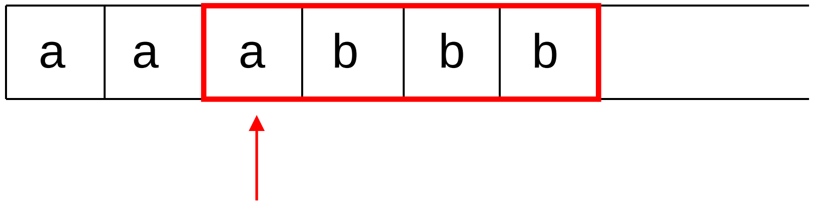
- Conventions:
  - The stack grows bottom-up
  - The input strings is read left to right
  - The other way around is possible, but is important to be coherent!

Example of configuration

$c=<q_1, abbb, ABZ_0>$







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*

Transitions, formally

- If $\delta(q, \mathbf{i}, A) = \langle q', \alpha \rangle$ is defined then
  - $c=\langle q, \mathbf{iy}, A\beta \rangle \vdash c'=\langle q', \mathbf{y}, \alpha\beta \rangle$

- If $\delta(q, \boldsymbol{\varepsilon}, A) = \langle q', \alpha \rangle$ is defined then
  - $c=\langle q, \mathbf{x}, A\beta \rangle \vdash c'=\langle q', \mathbf{x}, \alpha\beta \rangle$

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"

Acceptance condition

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

Acceptance condition:

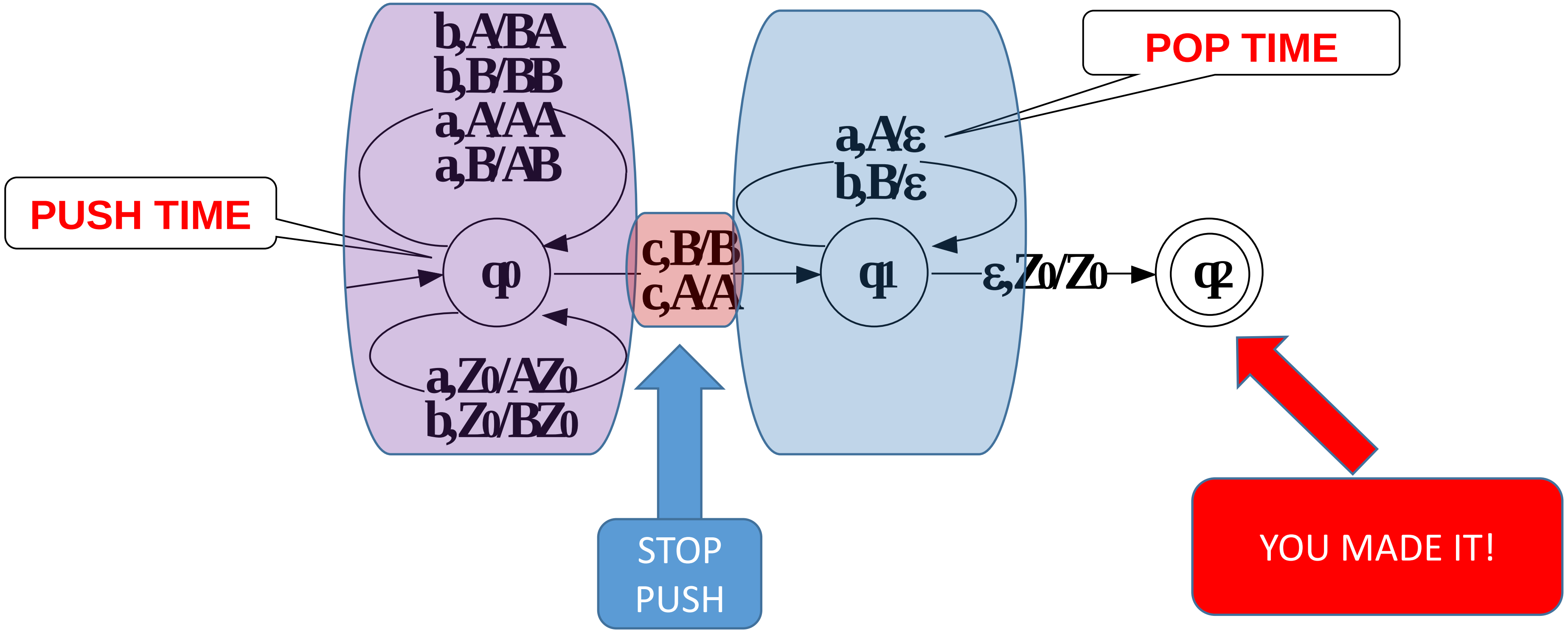
$\forall x \in I^{*}$ $(x \in L \Leftrightarrow c_0=<q_0,x,Z_0>$ $|-^{*}-$ $c_F=<q,\varepsilon,\gamma>$ and $q \in F)$

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 there is a path coherent with x on the PDA that goes from the initial state to the final state**
  - The input string has to be read completely

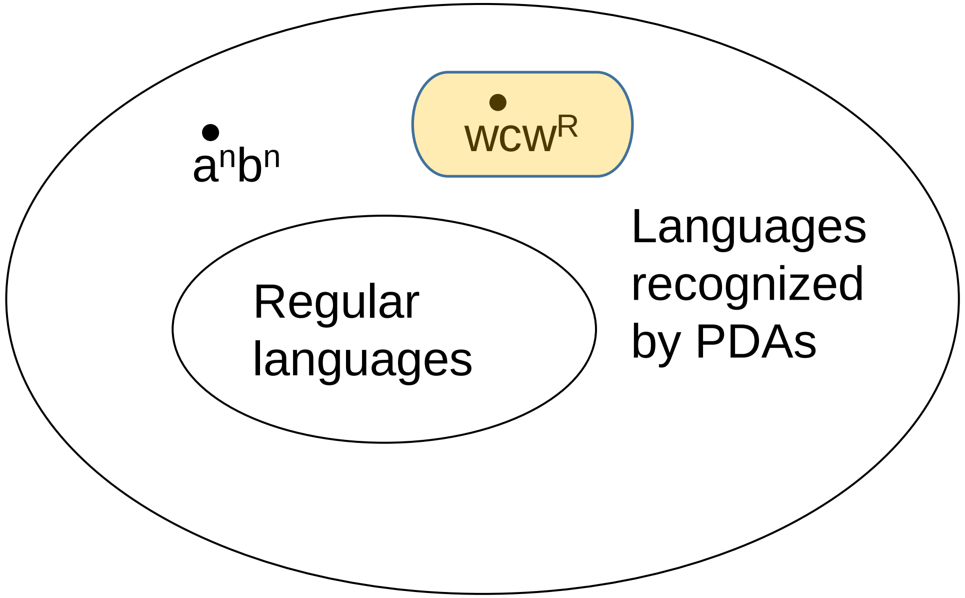
Example

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

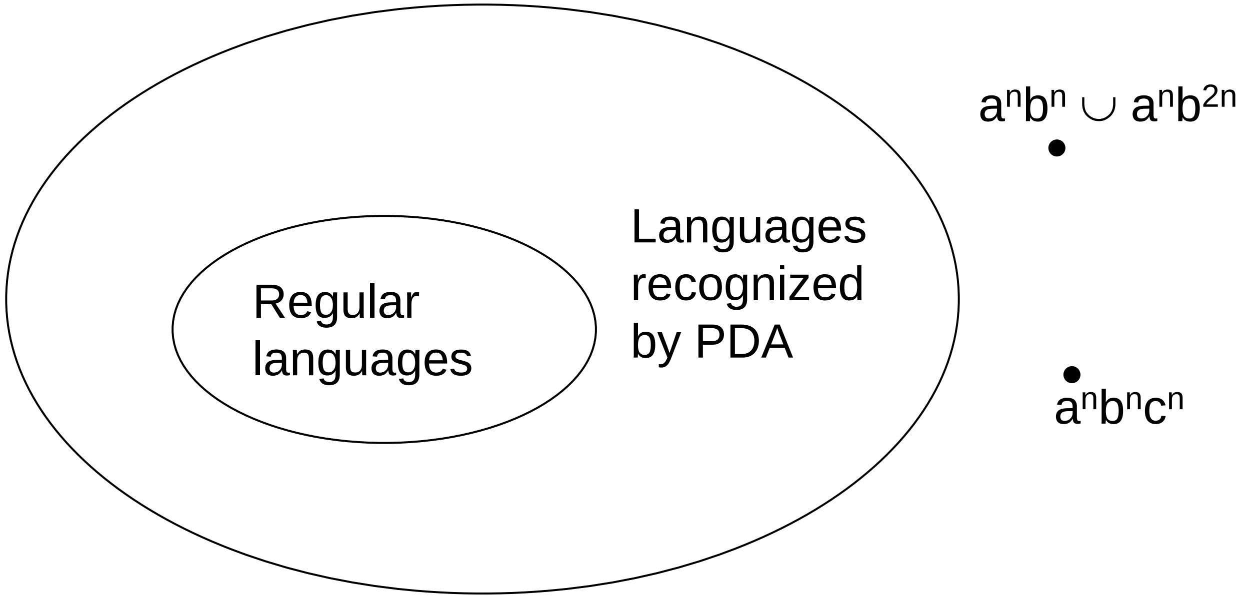


PDA vs FSA

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



Languages



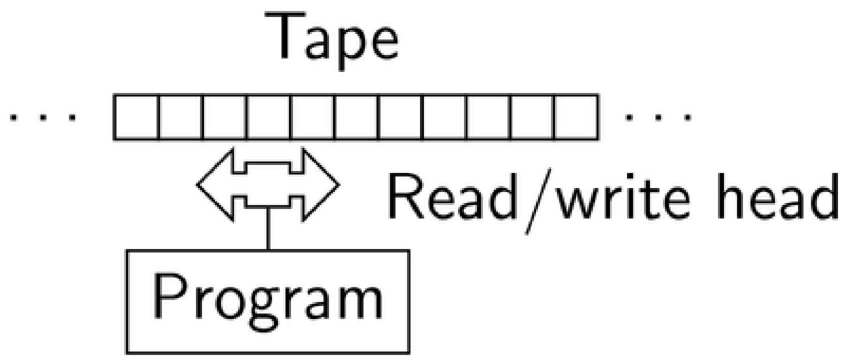
What are the limits of PDAs?

Stack vs. tape

- The stack is a **destructive memory**
  - Once a symbol is read, it is destroyed

- It is necessary to use **persistent memory**

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



Bar-Hillel (hints)

- **There is a generalization of the pumping lemma**
  - Lemma of **Bar-Hillel**
  - The proof is based on the derivation tree (Chomsky generative grammars)

- A property shared by **all context-free languages**

- Not sufficient to guarantee that a language is context-free

- **More on the tutorial**