# Formal Grammars

CHRISTOPHE SERVAN, PHD

- Formal Languages
- Regular Languages
- Context-free languages
- Context-sensitives languages
- Recursively enumerable languages

# Formal Grammar

Once upon a time…

# Alphabet, word

- An alphabet is a set of symbols.

  A={a,b,c,d,e,f,0,1,2,3,4,5,6,7,8,9}

- We will use as alphabet always implicitly the set of all unicode characters.

  A={0,…9,a…,z,A…Z,!,?,…}

- A word over an alphabet A is a sequence of symbols from A.
- Since we use the alphabet of unicode characters, words are just strings.

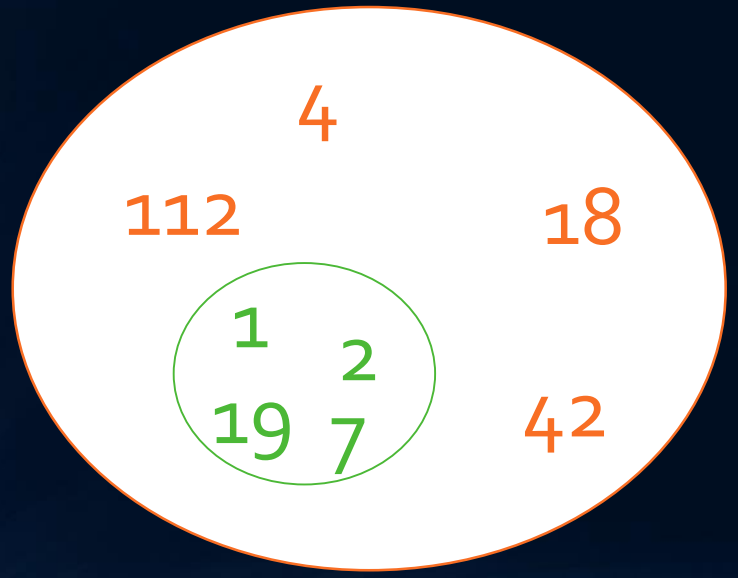
  hello!, 42, 3.141592, Douglas and Sally

# Language

- A language over an alphabet S is a set of words over S.
- Examples of interesting languages:
  - The language of French words: {aimer, chat, sur, …}
  - The language of French sentences: {J'aime les chats, Le chat chante,…}
  - The language of valid Java programs: {public static void main(…)…, …}
  - The language of terminating Java programs: {public…,…}
  - The language of algebraic expressions: {(1+2)*3, (7-2)+(7+2), …}
  - The language of prime numbers: {2, 3, 5, 7, 11, …}
  - The language of person names: {Thomas Paine, Bertrand Russel,…}
- We want to do two things with languages:
  - decide whether a given word belongs to the language
  - generate the words of the language

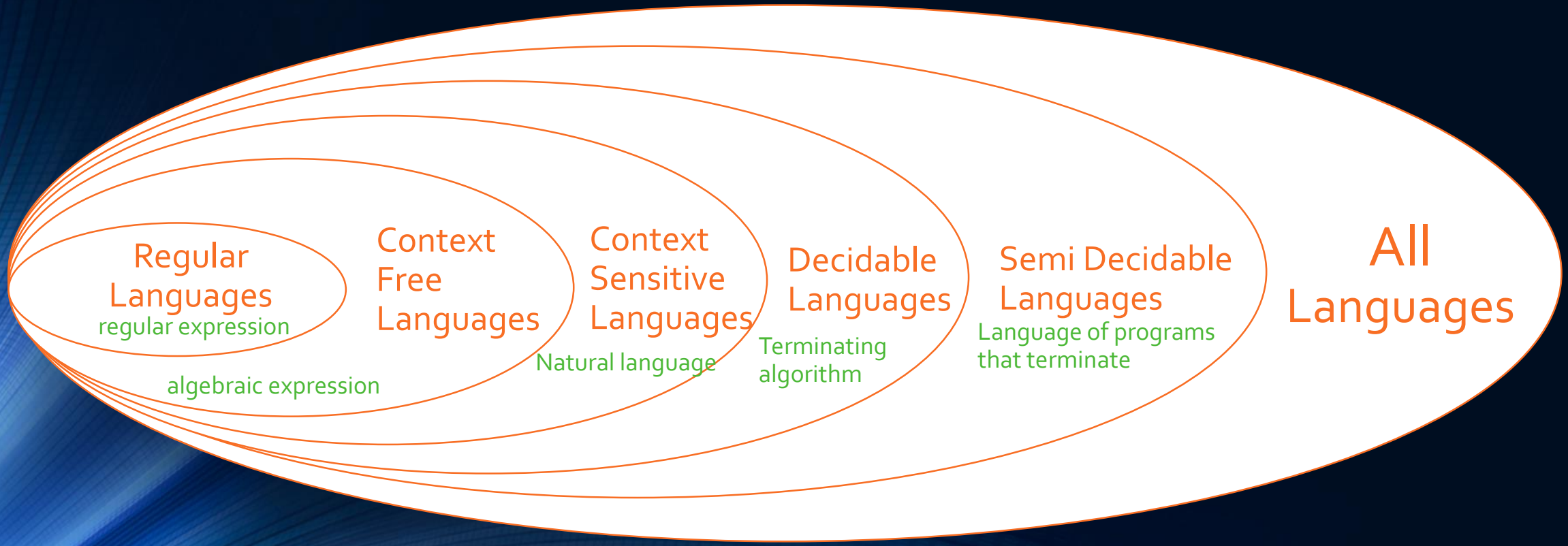
# Decision problem

- A  decision problem  is the question of whether a given string (*e.g.* a natural number ) belongs to a language (*e.g.* the language of prime numbers ).



# Chomsky hierarchy

- It turns out that the decision problem is more difficult for certain types of languages than for others. Languages are grouped as follows:



# Recognizing languages

- There are several formalisms that to recognize/generate the words of a language:
  - grammars (e.g., context-free grammars)
  - simple automata (e.g., finite state automata)
  - other formalisms (Turing machines, regular expressions)
- The main properties are:
  - each level in the Chomsky hierarchy has its own type of grammar (the decidable and semi-decidable ones have the same type)
  - for each type of grammar, there is usually an equivalent automaton and equivalent other formalisms
  - most formalisms can not just recognize the languages, but also generate them

# Formal grammar

- A (formal) grammar is a tuple of
- A finite set of nonterminal symbols (we write them in upper case)
- A start symbol
- A finite set of terminal symbols that is disjoint from (lower case)
- A finite set of production rules, each rule of the form

$$\left(\sum \cup N\right)* N\left(\sum \cup N\right)* \rightarrow \left(\sum \cup N\right)*$$

Where * is the Kleene star, which allows repetition of the elements.

- Example:

N={S,NP,VP}
s=S
∑={they, run, love, you}
P ={ S → NP VP, NP → they, NP → you, VP → love, VP → run }

# Language of a grammar

- A string of a grammar (N, s, ∑, P) is an element of $(\sum \cup N)*$ (we use Greek letters for strings). The grammar derives a string αβγ from a string αδγ in one step, if there is a production rule δ -> β.
  It derives a word (of terminal symbols) if there is a sequence of one s tep derivations from the start symbol to the word.
  The language of a grammar is the set of all derivable words.

The grammar can both generate
the words of the language (by derivation)
and recognize them (by parsing, i.e., by finding
the derivations for the word).

- Example:

N={S,NP,VP}
s=S
∑={they, run, love, you}
P ={ S → NP VP, NP → they, NP → you, VP → love, VP → run }

S → NP VP → you VP → you run

- Formal Languages
- Regular Languages
- Context-free languages
- Context-sensitives languages
- Recursively enumerable languages

# Formal Grammar

Once upon a time…

# Regular Grammar

- We will see 3 equivalent formalisms for regular languages: regular grammars, finite state automata, and regular expressions.
- A (right) regular grammar has only productions of the following form
  - B → a, where "B" is a non-terminal and "a" is a terminal
  - B → aC, where "B" and "C" are non-terminals and "a" is a terminal
  - B → ε, where is a non-terminal and "ε" is the empty string

Example:

N={S,A,B}
s=S
∑={a,b}
P ={S → aA, S → bB, A → aA, B → bB, A → ε, B → ε}

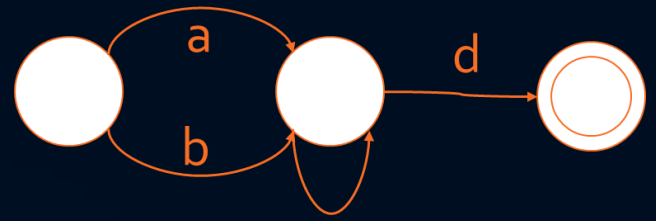
S → A → aA → aaA → aa

# Regular Languages

- The languages that can be generated/recognized by a regular grammar are the  regular languages  of the Chomsky Hierarchy.
- Example of regular languages:
  - Search/replace patterns in editors
  - Simple name entities
  - Number, dates, quantities
- Regular languages are closed under:
  - Union
  - Intersection
  - Complement
  - Concatenation
  - Iteration (Kleene star)

# Finite State Machines

- A finite state machine (FSM) is a directed multi-graph, where each edge is labelled with a symbol or the empty symbol "ε" .
- One node is labelled "start" (typically by an incoming arrow).
- Zero or more nodes are labelled "final" (typically by a double circle)

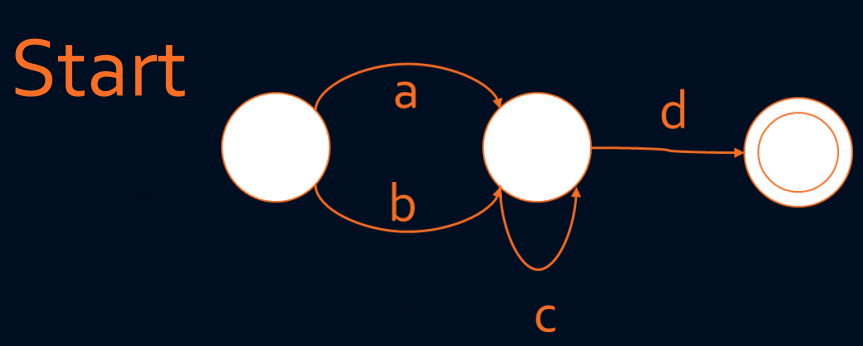



- The empty transition can be walked without accepting a symbol:


- Multiple edges are written as:

# Recognizing words

- An FSM recognizes (also: generates, accepts) a string, if there is a path from the start node to a final node whose edge labels are the string.




ad ✓

bccd ✓

bb ✗

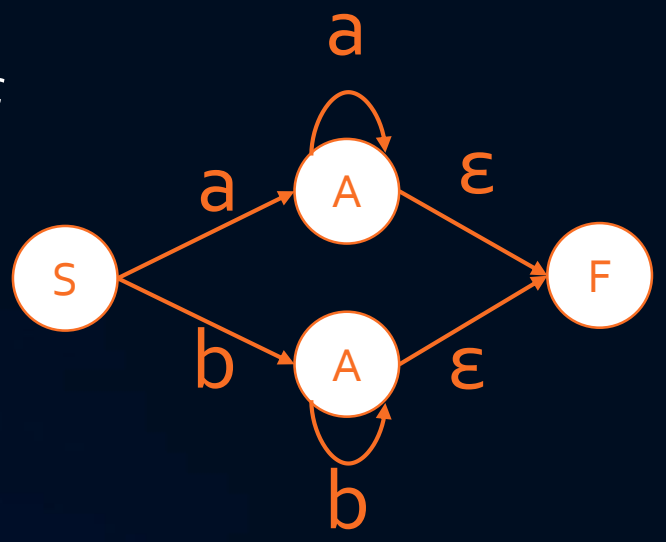
# FSM & regular languages

- Every Right Regular Grammar can be transformed into an equivalent FSM as follows:
  - introduce a state for every non-terminal
  - let the grammar start symbol be the initial state of the automaton
  - introduce a final state *F*
  - for every rule $A \rightarrow \alpha$ , add a transition from *A* to *F* with symbol $\alpha$
  - for every rule $A \rightarrow \alpha C$ , add a transition from *A* to *C* with symbol *c*

Example:

$N=\{S,A,B\}$, $s=S$, $\sum=\{a,b\}$

$P=\{S \rightarrow aA,\ S \rightarrow bB,\ A \rightarrow aA,\ B \rightarrow bB,\ A \rightarrow \varepsilon,\ B \rightarrow \varepsilon\}$




Every FSM can be transformed into a grammar!

# Regular expression cheat sheet

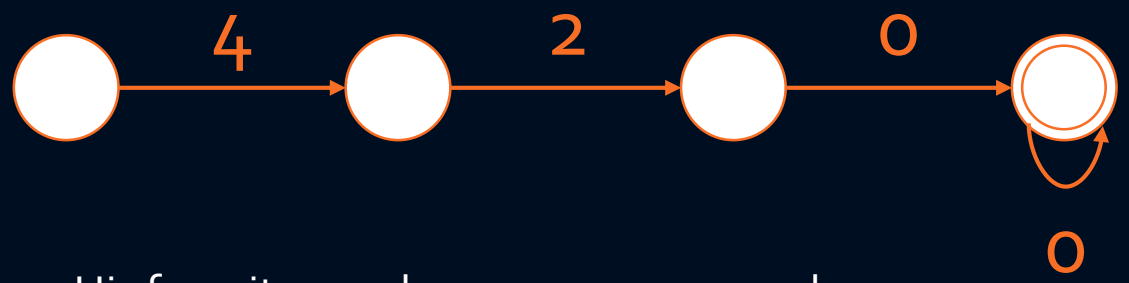
- L(a) = {a}
- L(ab) = {ab}
- L(a | b) = {a, b}
- L(a*) = { , a, aa, aaa, …}
- L(a+) := L(aa*)
- L([a-z]) := L(a|b|…|z)
- L(a{2,4}) := L(aa|aaa|aaaa)
- L(a{3}) := L(aaa)
- L(a?) := L(ε | a)
- L(.) := {a|b|..|A|…|o|…|!|$|…}
- L(\.) := {.}

- Simple symbol
- Concatenation
- Disjunction
- Kleene star
- shorthand for “one or more”
- shorthand for a range
- shorthand for a given number
- shorthand for a given number
- shorthand for “optional”
- shorthand for “any symbol”
- escape sequence for special symbols

# REGEX in programming

- Regex :
- Simplified Regex
- FSM
- Matcher

- 42(0)+
- 420(0)*




His favorite numbers are 42, 4200 and 19

# Example RegEx in Python

```
import re
import sys
pattern = re.compile("42(0)+")
for line in sys.stdin:
        match = re.search(pattern, line)
        if match!=None:
                print(match.group())
```

# Example RegEx in Perl (in command line)

⇒ echo “420000000” | perl –pe ‘s/42[0]+/NUM/g‘

⇒ NUM

- Formal Languages
- Regular Languages
- Context-free languages
- Context-sensitives languages
- Recursively enumerable languages

# Formal Grammar

Once upon a time…

# Context-Free Grammar

A context-free grammar has only production where the left-hand-side is a single non-terminal.

Example:

N={S,NP,VP},
s=S,
∑={they, run,love,you}
P ={S → NP VP, NP → they, NP → you, VP → love, VP → run}

# Context-free Languages

The languages that can be generated/recognized by a context-free grammar are the context-free languages of the Chomsky Hierarchy

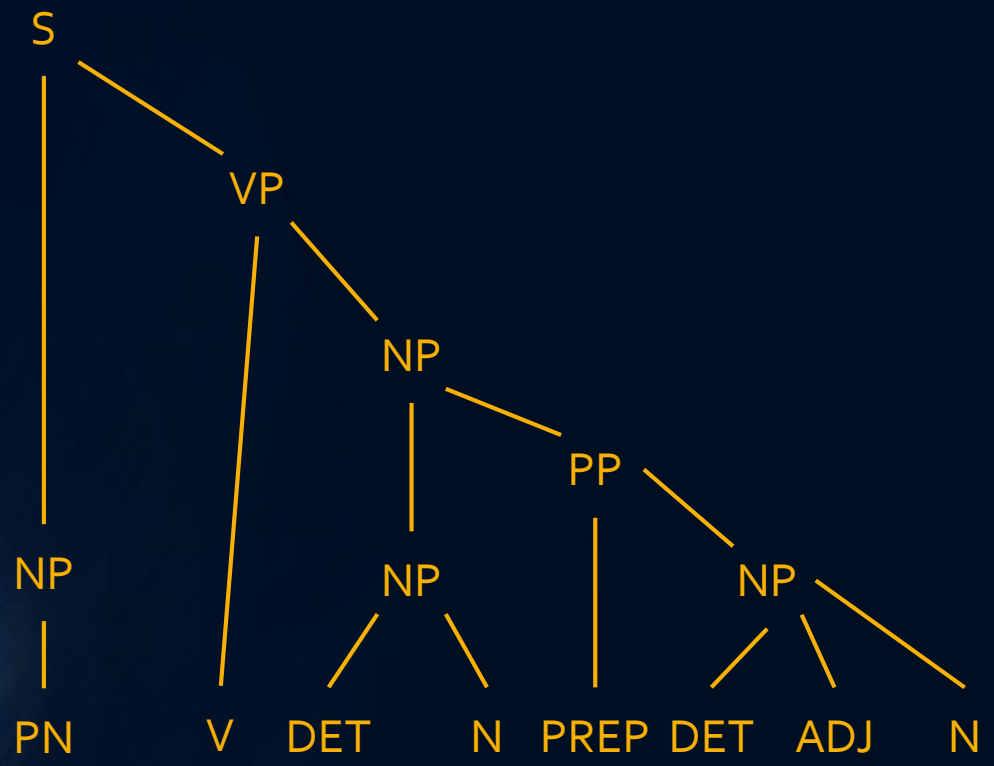
Examples for context-free languages:

- ⇒ algebraic expressions
- ⇒ simple programming languages
- ⇒ simplistic models of natural language sentences
- ⇒ …or anything with nested structures

Context-free languages are closed under:

- ⇒ concatenation
- ⇒Kleene star
- ⇒ …but not under intersection or complement.

# Phrase structure grammars




Howard is the father of the little Tony

S → NP VP

NP → PN

VP → V

NP → DET ADJ N

PN → Howard

DET → the

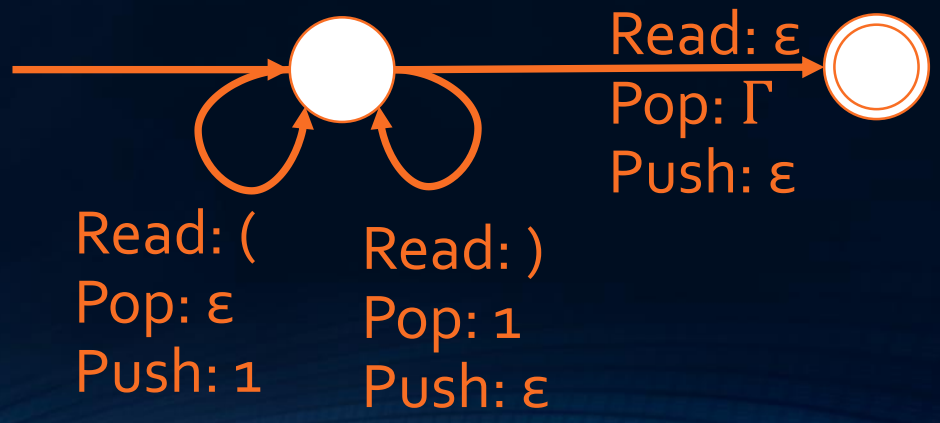
…

# Pushdown Automaton

Given an input alphabet $\in \Sigma$, a stack alphabet $\in \Gamma$, and a start symbol $Z \in \Gamma$, a pushdown automaton (PDA) is a directed multi-graph, were each edge is labeled with :

- a symbol $\in \Gamma^*$ to read from the input (or $\varepsilon$ )
- a symbol $\in \Gamma^*$ to pop from the stack (or $\varepsilon$ )
- a sequence of symbols $\in \Gamma^*$ to push onto the stack

One node is labeled "start", and zero or more nodes are labeled "final".



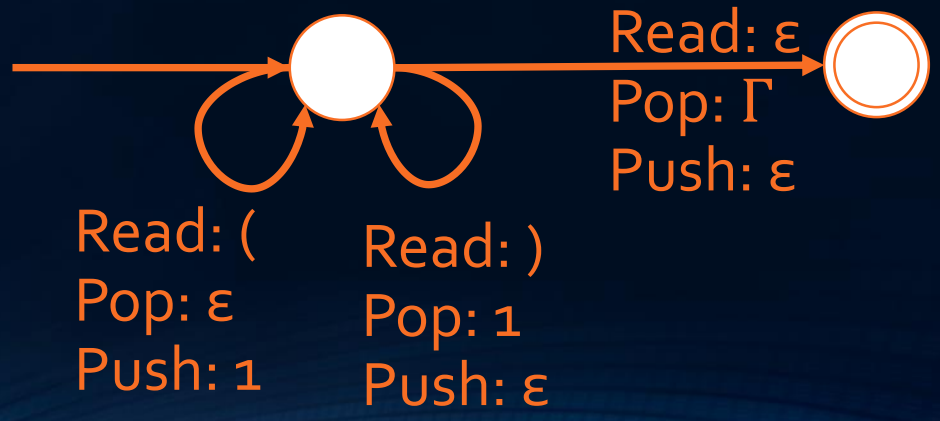
# Running of a PDA

A configuration of a PDA is a triple of a node $q$, an input string $w \in \Sigma^*$, and a stack $\beta \in \Gamma^*$. The step relation is defined for all $w \in \Sigma^*$, $\beta \in \Gamma^*$ as the following relation between configurations:

$\langle q, cw, t\beta \rangle \vdash \langle q', w, \gamma\beta \rangle$

if there is an edge from $q$ to $q'$ that is labeled with "read $c$, pop $t$, push $\gamma$". A word $w$ is recognized by the PDA, if $\exists\, \beta : \langle q_0, w, Z \rangle \vdash^* \langle q, \varepsilon, \beta \rangle$ where $q_0$ is the start node and $q$ is a final node.

The language of a PDA is the set of recognized words

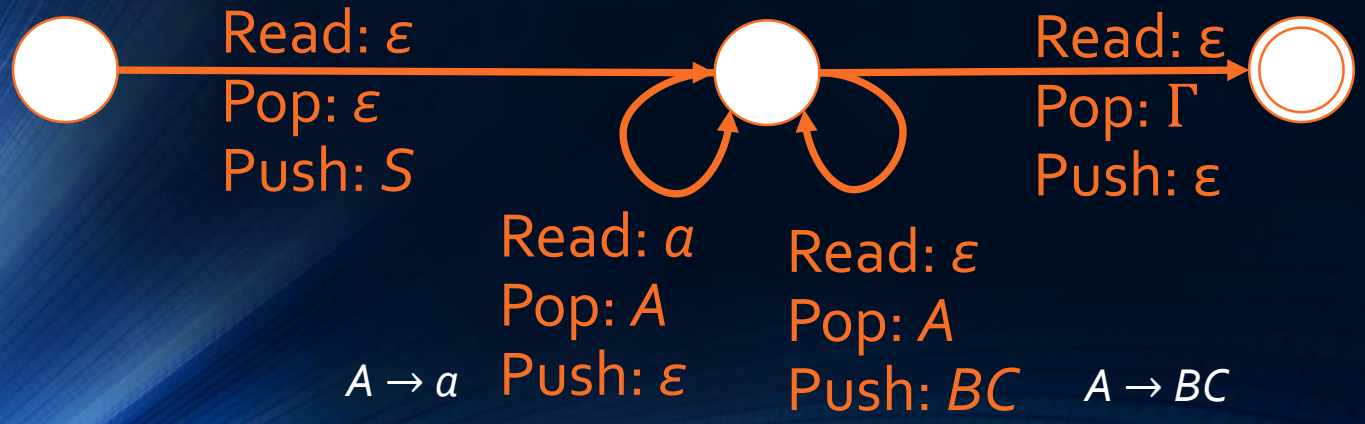


# Context-free grammars & PDAs

For every context-free grammar there is an equivalent pushdown automaton and vice versa. To construct the PDA for a context-free grammar, normalize the rules so that they are all of the following forms:

$A \to \varepsilon$, $A \to a$, $A \to BC$

Be S the start symbol. Then construct the non-deterministic PDA as:



# Limitations of context-free grammars

Context-free grammars need extensive duplication of non-terminal symbols to model dependencies between words:

Sentence -> FirstPersonSingularSentence | SecondPerson…
FirstPersonSingularSentence-> "I" FirstPersonSingularVerbPhrase
FirstPersonSingularVerbPhrase -> "go" | "love" | …

Various remedies have been devised, e.g., using arguments that can be compiled away:

Sentence(*p*,*n*) -> NounPhrase(*p*,*n*) VerbPhrase(*p*,*n*)
NounPhrase(1,s) -> "I"

…or devising "weakly context-sensitive grammars". No context-free grammar can recognize

L=$\{a^n b^n c^n : n > o\}$

# Context-free Languages

The following statements are equivalent for a language $L$ :

- $L$ is context-free
- $L$ can be recognized by a pushdown automaton
- $L$ can be generated by a pushdown automaton
- $L$ can be recognized by a context-free grammar
- $L$ can be generated by a context-free grammar

Recognizing a word of a regular language takes polynomial time.

Every regular language is a context-free language.