

Formal Grammars

CHRISTOPHE SERVAN, PHD

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

Formal Grammar

Once upon a time...

- 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.

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

- 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.

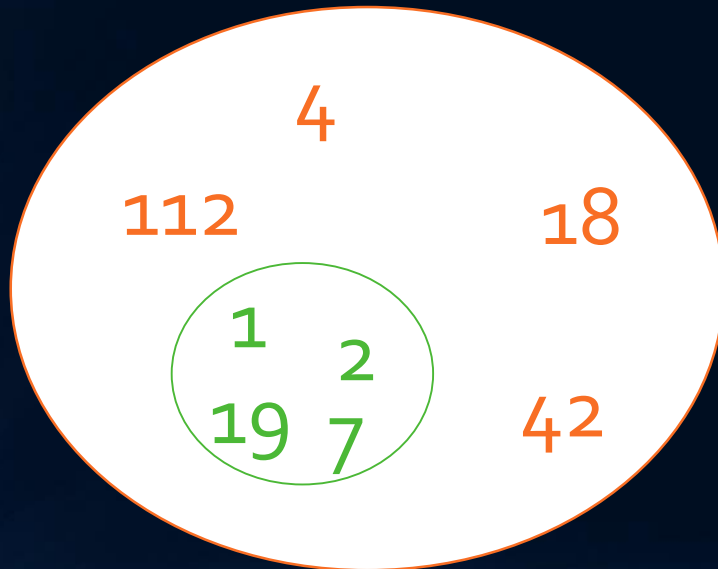
hello!, 42, 3.141592, Douglas and Sally

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

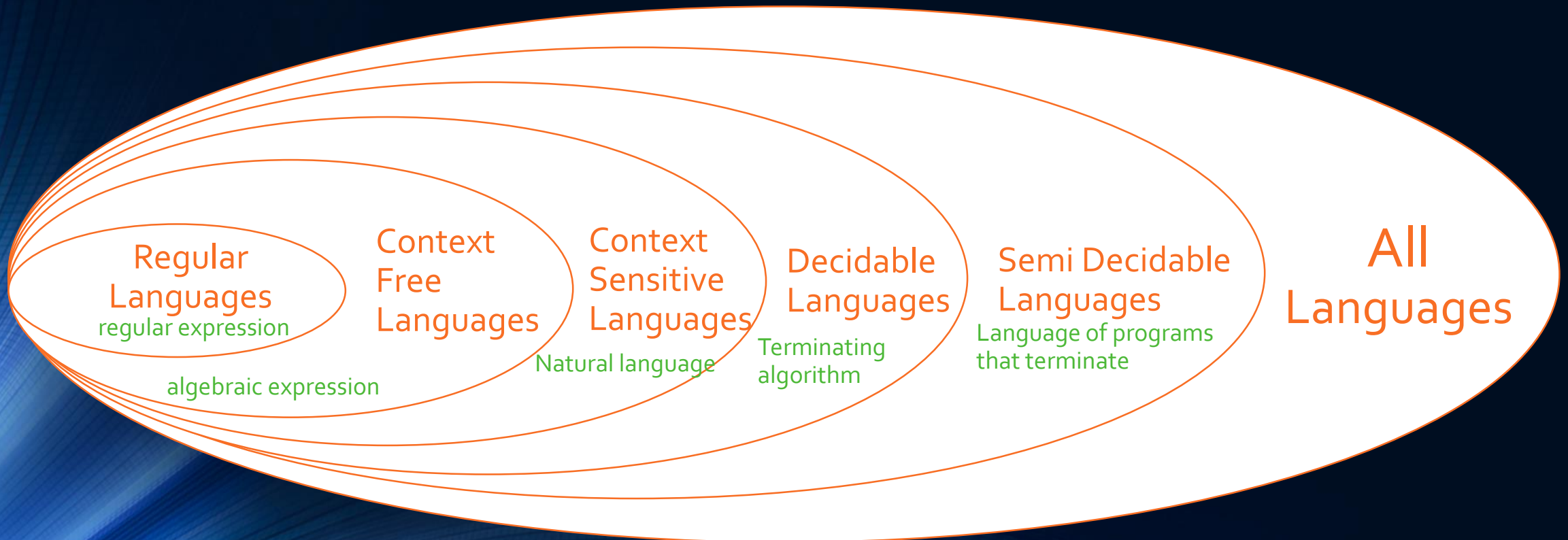
Decision problem

- 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

$$\left(\sum U^N\right)^* N \left(\sum U^N\right)^* \rightarrow \left(\sum U^N\right)^*$$

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

- Example:

$N = \{S, NP, VP\}$

$s = S$

$\Sigma = \{\text{they, run, love, you}\}$

$P = \{S \rightarrow NP VP, NP \rightarrow \text{they}, NP \rightarrow \text{you}, VP \rightarrow \text{love}, VP \rightarrow \text{run}\}$

Language of a grammar

- A string of a grammar (N, s, Σ, P) is an element of $(\Sigma \cup N)^*$ (we use Greek letters for strings). The grammar derives a string $\alpha\beta\gamma$ from a string $\alpha\delta\gamma$ in one step, if there is a production rule $\delta \rightarrow \beta$. It derives a word (of terminal symbols) if there is a sequence of one or more 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:

$N = \{S, NP, VP\}$

$s = S$

$\Sigma = \{\text{they, run, love, you}\}$

$P = \{S \rightarrow NP VP, NP \rightarrow \text{they}, NP \rightarrow \text{you}, VP \rightarrow \text{love}, VP \rightarrow \text{run}\}$

$S \rightarrow NP VP \rightarrow \text{you VP} \rightarrow \text{you run}$

- Formal Languages
- Regular Languages
- Context-free languages
- Context-sensitive 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
 - $B \rightarrow a$, where “B” is a non-terminal and “a” is a terminal
 - $B \rightarrow aC$, where “B” and “C” are non-terminals and “a” is a terminal
 - $B \rightarrow \varepsilon$, where is a non-terminal and “ ε ” is the empty string

Example:

$N = \{S, A, B\}$

$s = S$

$\Sigma = \{a, b\}$

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

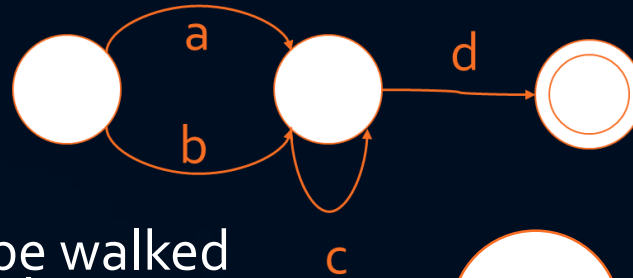
$S \rightarrow A \rightarrow aA \rightarrow aaA \rightarrow aa$

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:

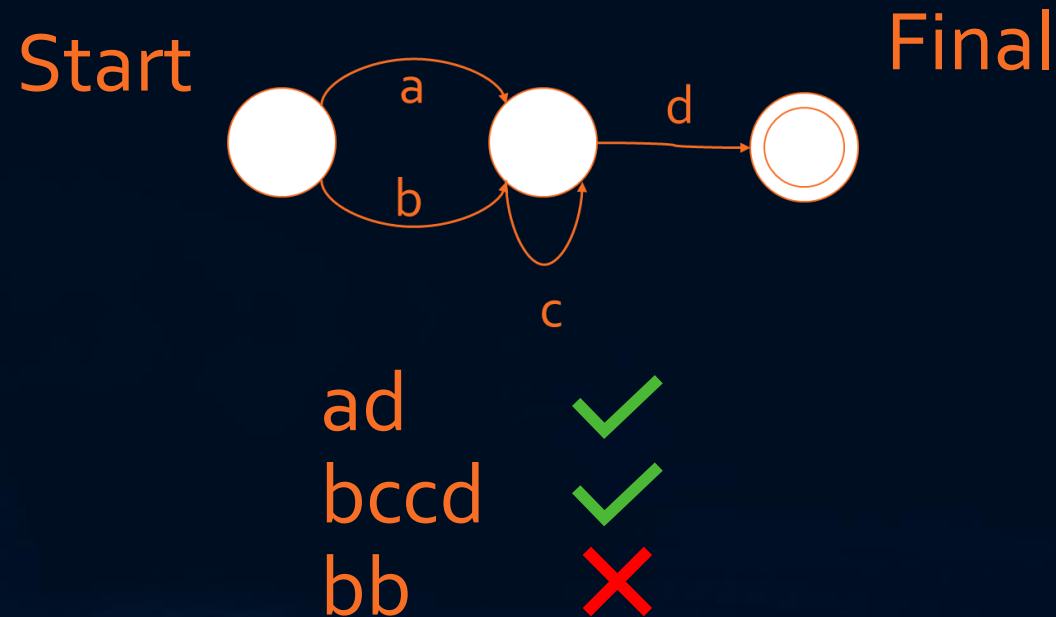


- Multiple edges are written as:



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.



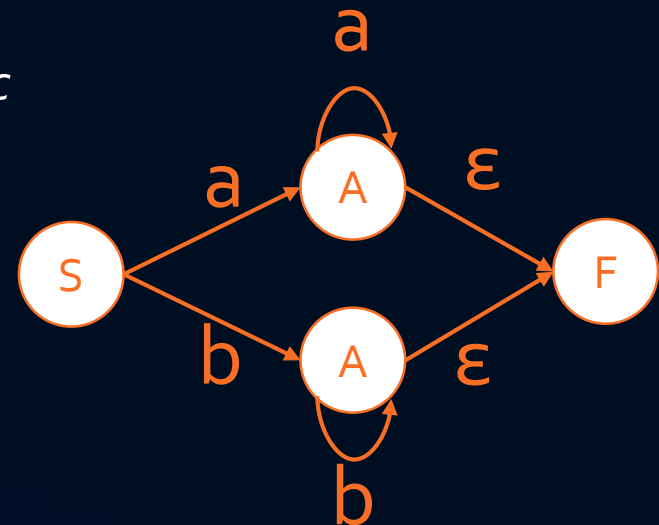
FSM & regular languages

- 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 $A \rightarrow a$, add a transition from A to F with symbol a
 - for every rule $A \rightarrow aC$, add a transition from A to C with symbol a

Example:

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

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



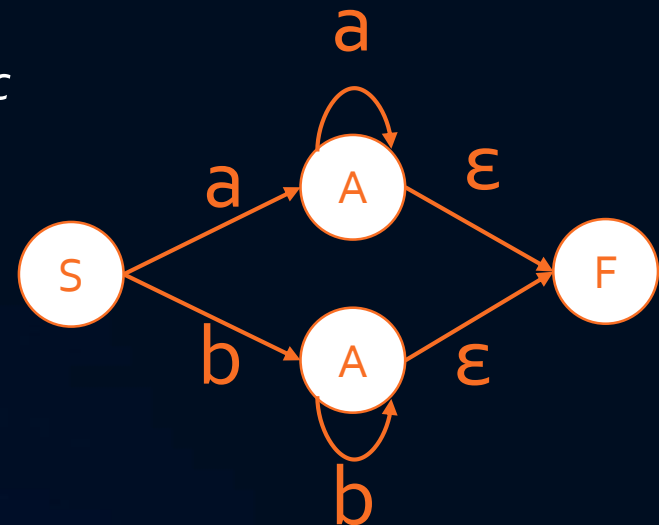
FSM & regular languages

- 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 $A \rightarrow a$, add a transition from A to F with symbol a
 - for every rule $A \rightarrow aC$, add a transition from A to C with symbol c

Example:

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

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



Every FSM can be transformed into a grammar!

Regular expression cheat sheet

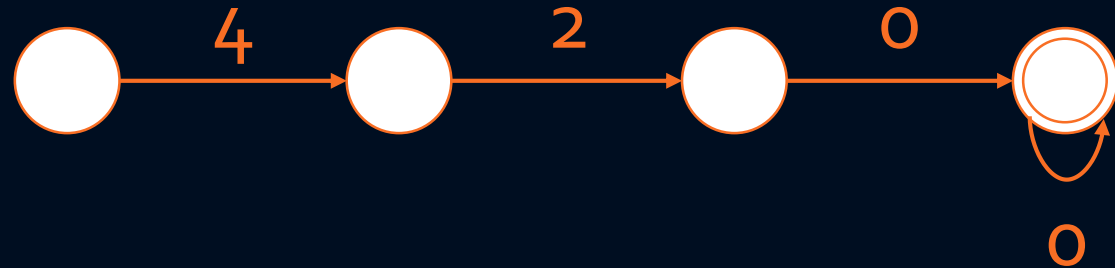
- $L(a) = \{a\}$
- $L(ab) = \{ab\}$
- $L(a \mid b) = \{a, b\}$
- $L(a^*) = \{, a, aa, aaa, \dots\}$
- $L(a^+) := L(aa^*)$
- $L([a-z]) := L(a|b|\dots|z)$
- $L(a\{2,4\}) := L(aa|aaa|aaaa)$
- $L(a\{3\}) := L(aaa)$
- $L(a?) := L(\epsilon \mid a)$
- $L(.) := \{a|b|..|A|\dots|o|\dots|!|\$|\dots\}$
- $L(\backslash.)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

- $42(o)^+$

- $42o(o)^*$

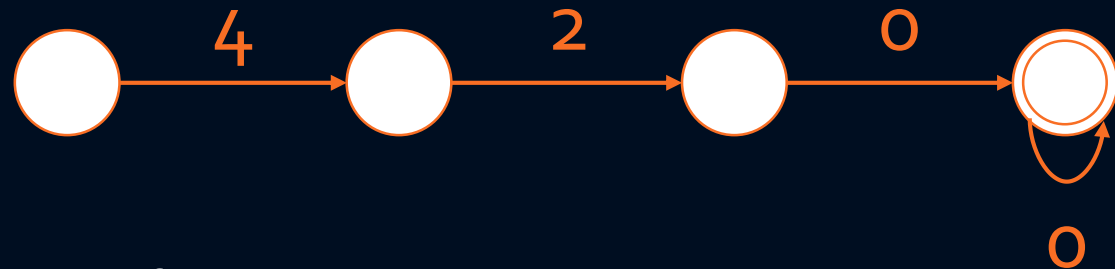


REGEX in programming

- Regex :
- Simplified Regex
- FSM
- Matcher

- $42(o)^+$

- $42o(o)^*$



His favorite numbers are 42, 4200 and 19

Example RegEx in Python

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