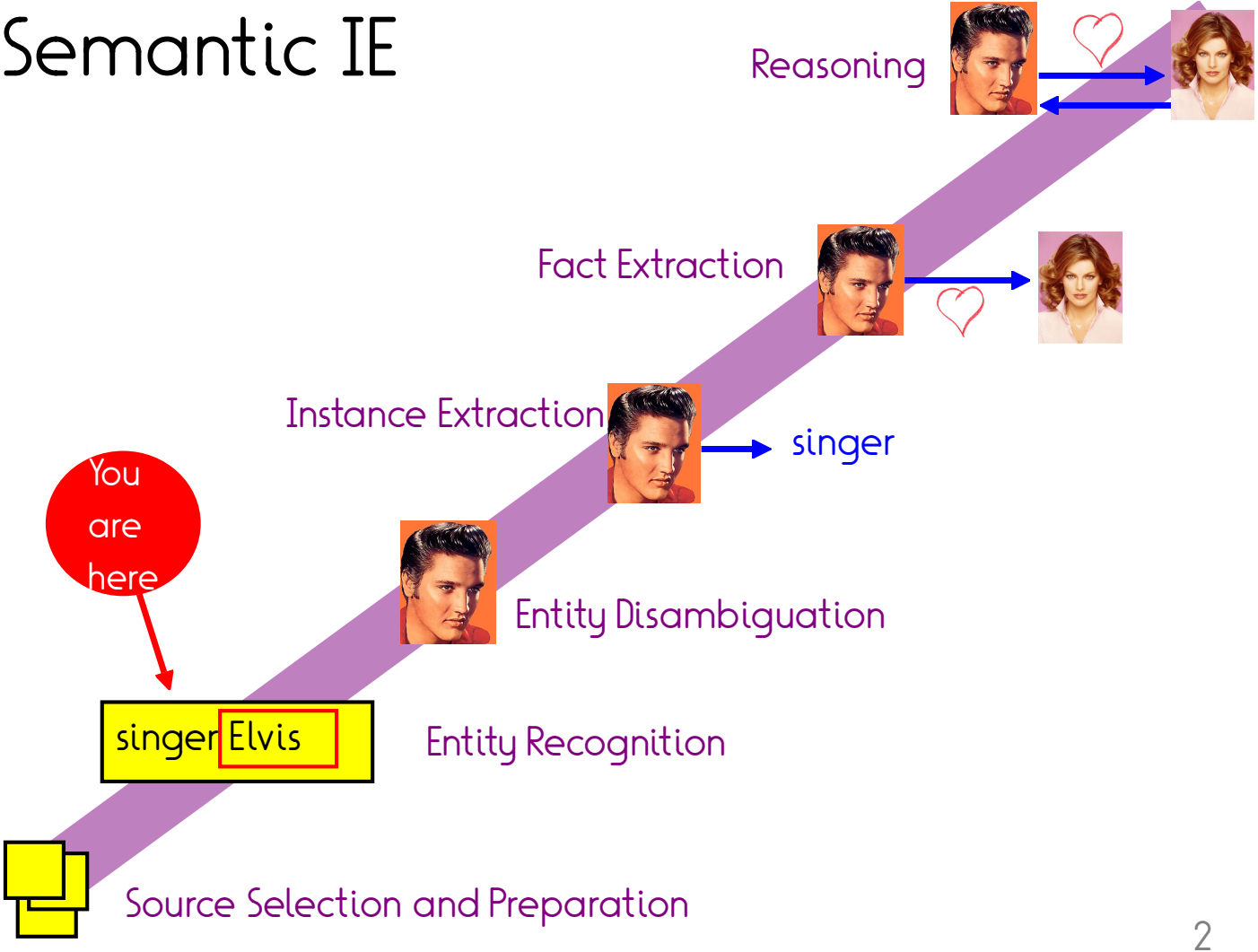
# Named Entity Recognition

Nada Mimouni

Based on Slides by:
Fabian M. Suchanek

# Semantic IE


Reasoning
Fact Extraction
Instance Extraction
singer
You are here
Entity Disambiguation
singer Elvis
Entity Recognition
Source Selection and Preparation

# Overview

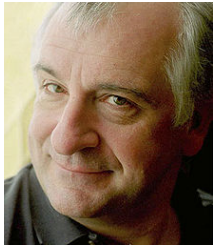
- Named Entity Recognition
- ...by dictionary
- ...by regex

# Named Entity

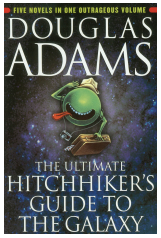
A named entity is an instance that has a name.

This typically concerns people, organizations, and artifacts.


Télécom ParisTech



Douglas Adams




Hitchhiker's guide to the galaxy

# Def: Named Entity Recognition

Named entity recognition (NER) is the task of finding entity names in a corpus.

In homage to Douglas Adams' book "Hitchhiker's Guide to the Galaxy" Google replies to the question of "life, the universe, and everything" with the number "42".

Try it out

(In its basic form, NER does not care to which entity the name belongs. It just finds names.)

# NER is difficult

Douglas Adam wrote the book The Hitchhiker's guide to the galaxy in...

In the beginning the Universe ? was created. This has made a lot of people very angry and been widely regarded as a bad move.

Capital Letters Were Always ✗ The Best Way Of Dealing With Things You Didn't Have A Good Answer To.

Cable and Wireless, a major company...

Microsoft and Dell both agreed...

✗

Alexander the Great ? ?

Quotes form Douglas Adams.

# Def: Dictionary

A dictionary (also: gazetteer, lexicon) is a set of entity names.

NER by dictionary finds only names of the dictionary.

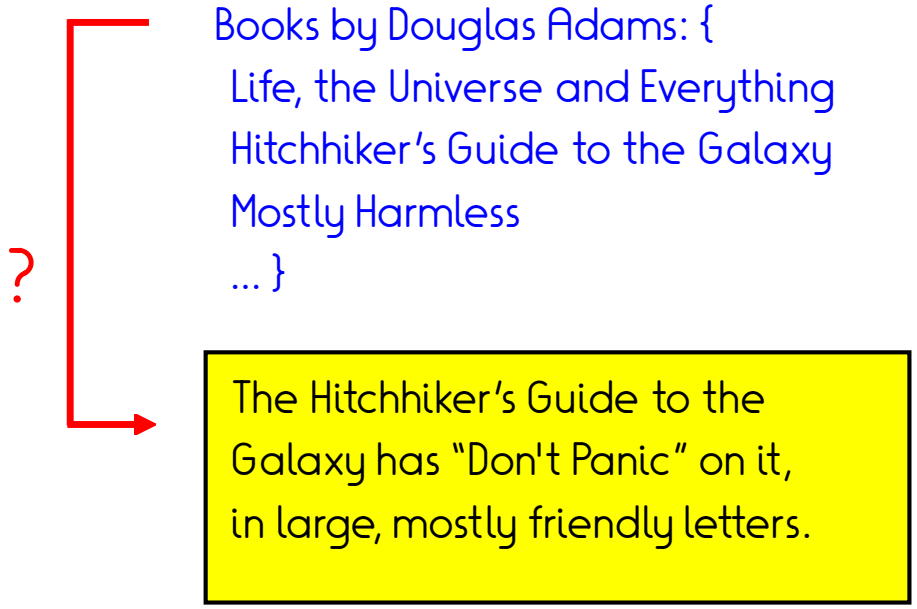
It can be used if the entities are known upfront.

US states: {Alabama, Alaska, California, ...}

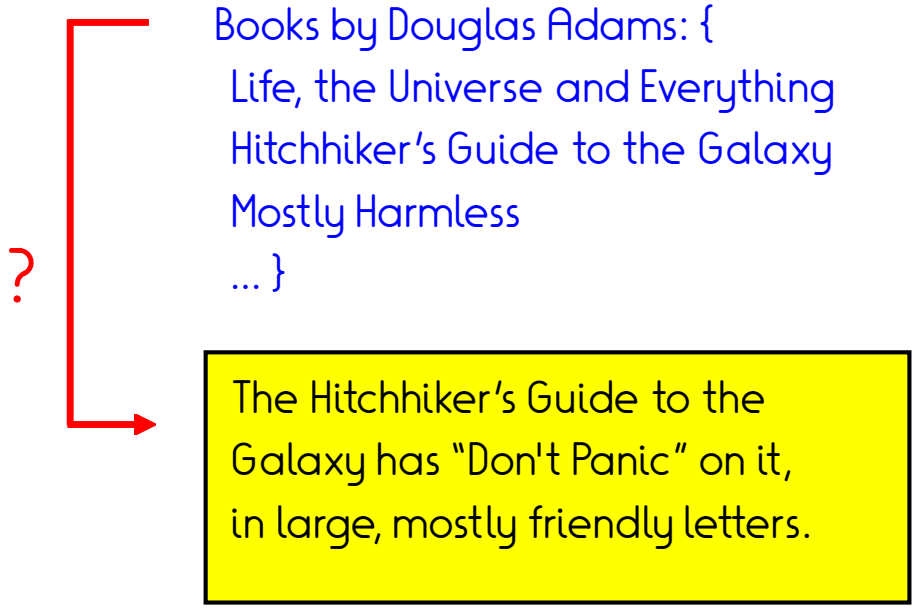
...lived in Los Angeles, California, while...

- US states
- countries
- Dow Jones companies
- Actors of a given movie
- ...

# Naive Dictionary NER is slow



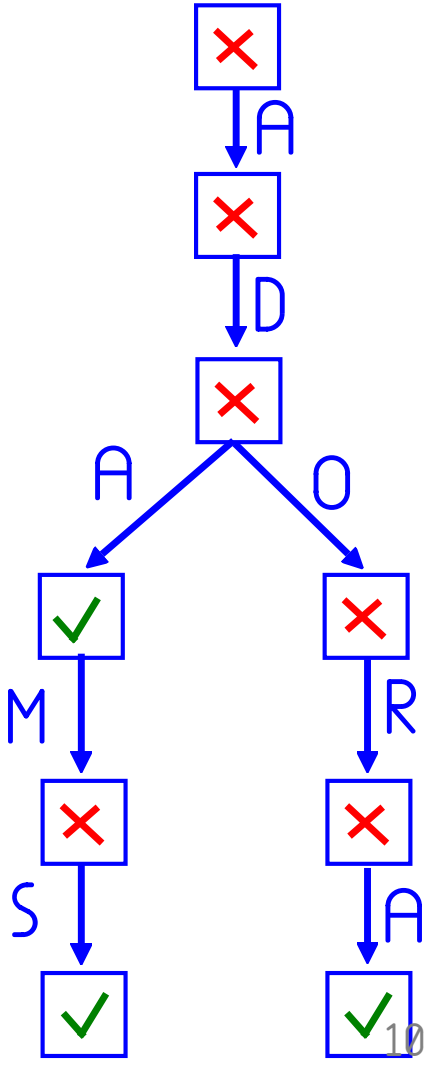
# Naive Dictionary NER is slow




$$O(textLength \times dictSize \times maxWordLength)$$

# Def: Trie

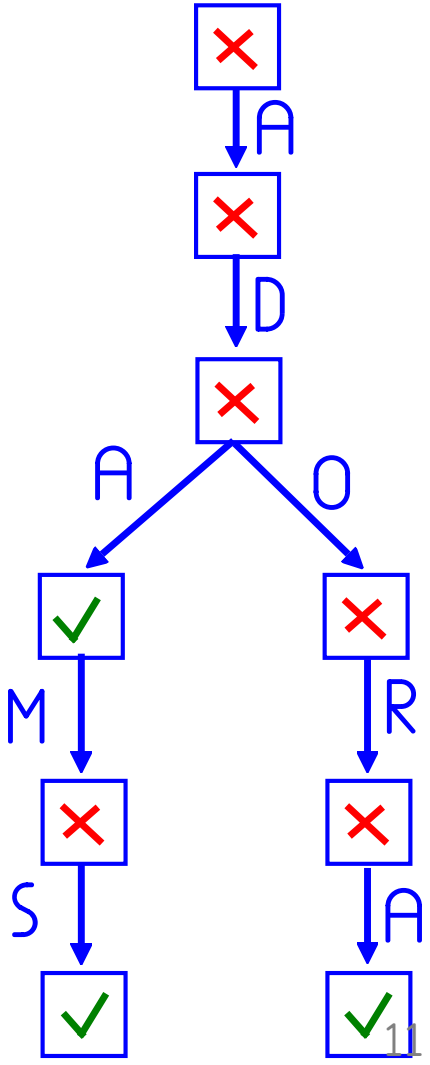
A trie is a tree, where nodes are labeled with booleans and edges are labeled with characters.



# A trie contains strings

A trie contains a string, if the string denotes a path from the root to a node marked with "true".

{ADA, ADAMS, ADORA}

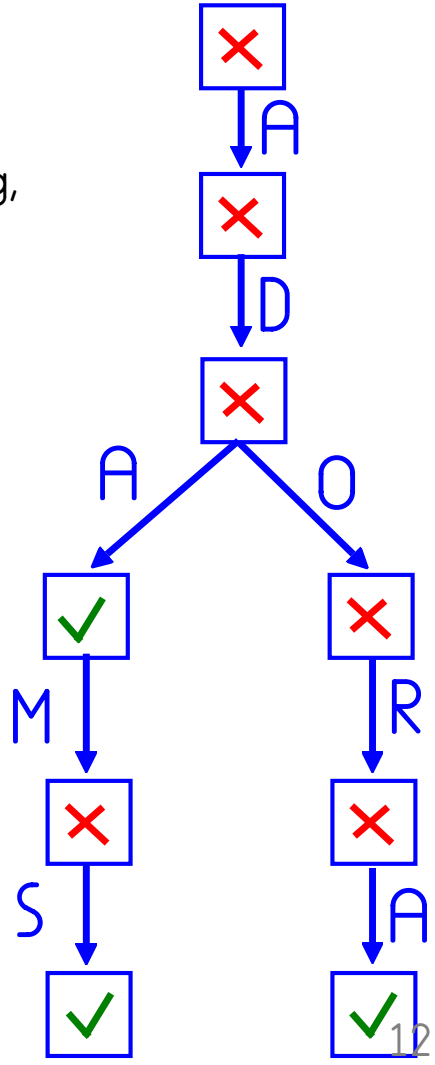


# Adding strings (1)

To add a string that is a prefix of an existing string, switch node to "true".

{ADA, ADAMS, ADORA}

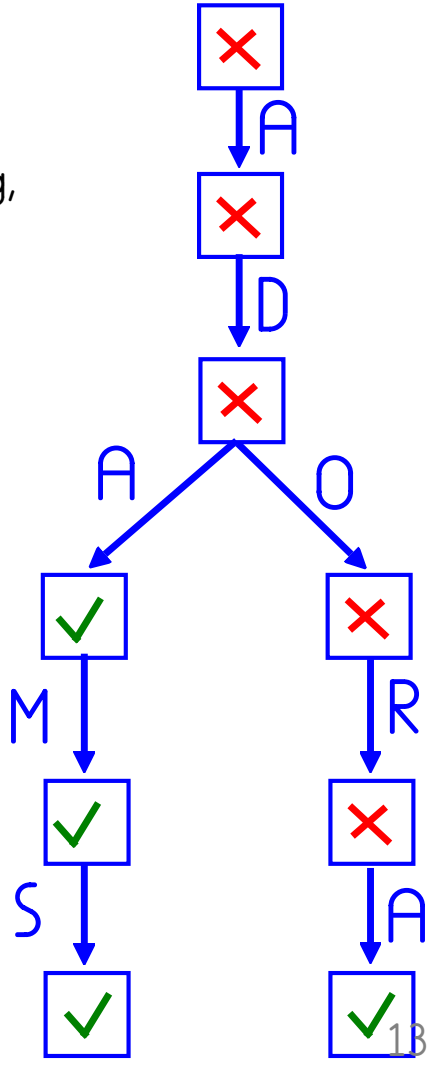
+ ADAM



# Adding strings (1)

To add a string that is a prefix of an existing string, switch node to "true".

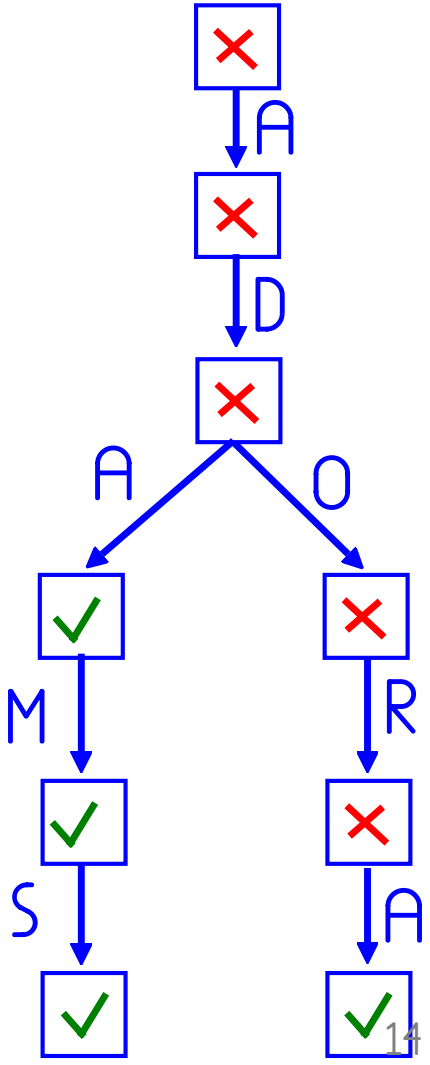
{ADA, ADAM, ADAMS, ADORA}



# Adding strings (2)

To add another string, make a new branch.

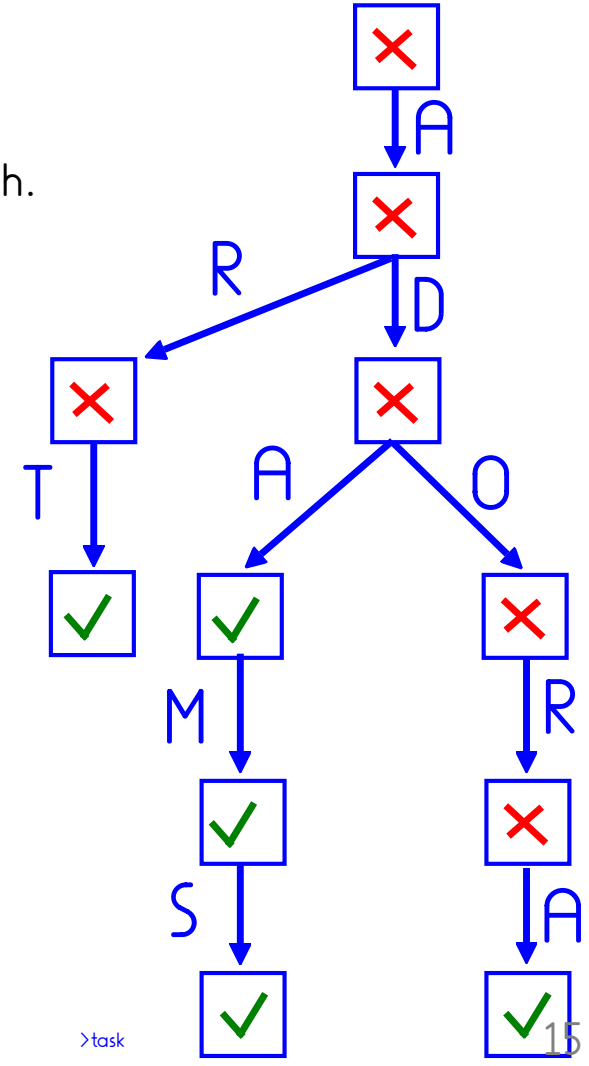
{ADA, ADAM,
ADAMS, ADORA} + ART



# Adding strings (2)

To add another string, make a new branch.

{ADA, ADAM,
ADAMS, ADORA, ART}



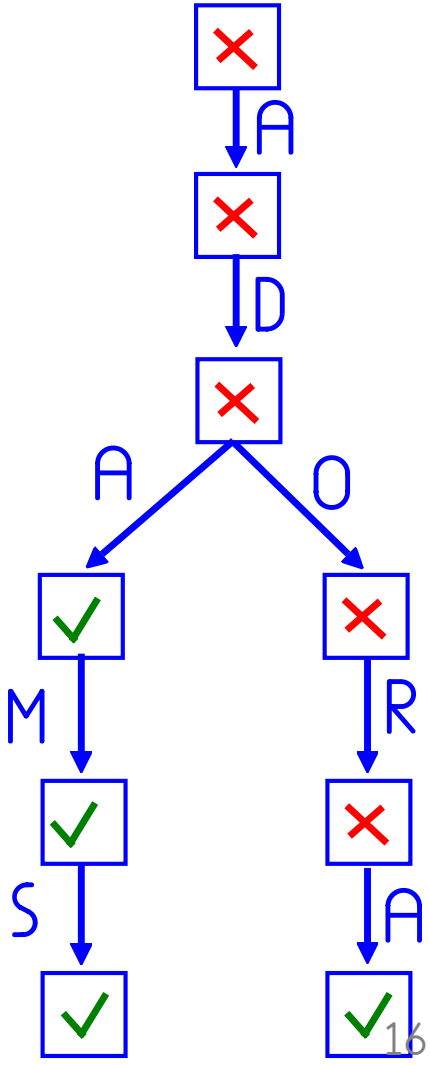

>task

# Task: Tries

Start with an empty trie.

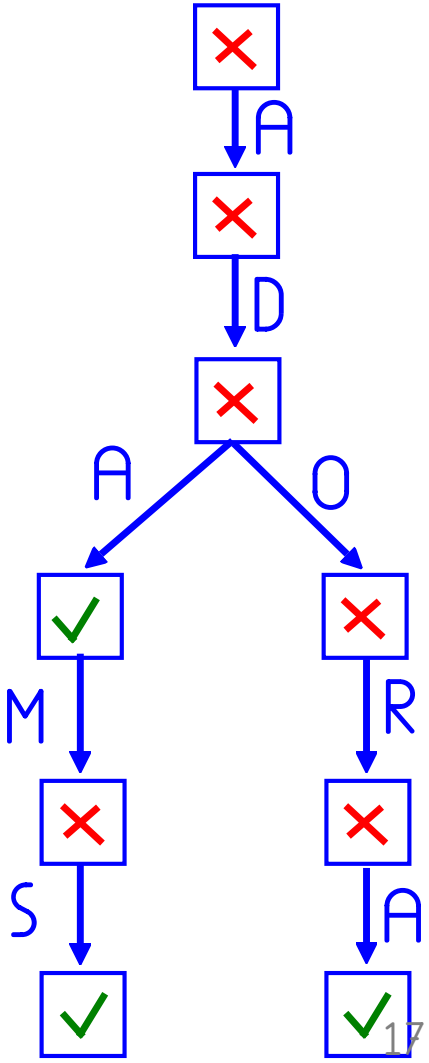
Add

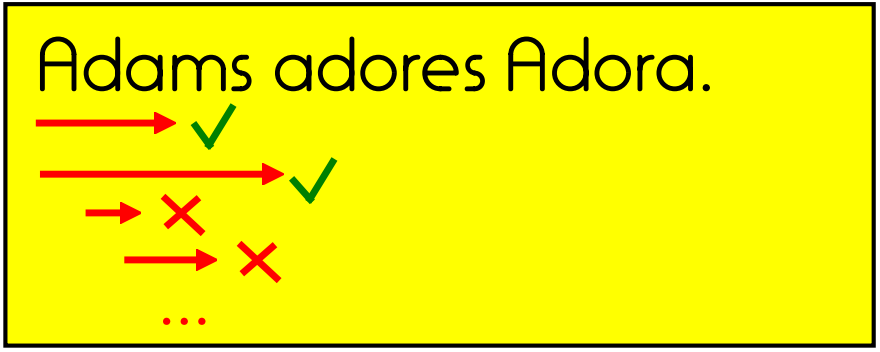
- bon
- bonbon
- on



# Tries can be used for NER

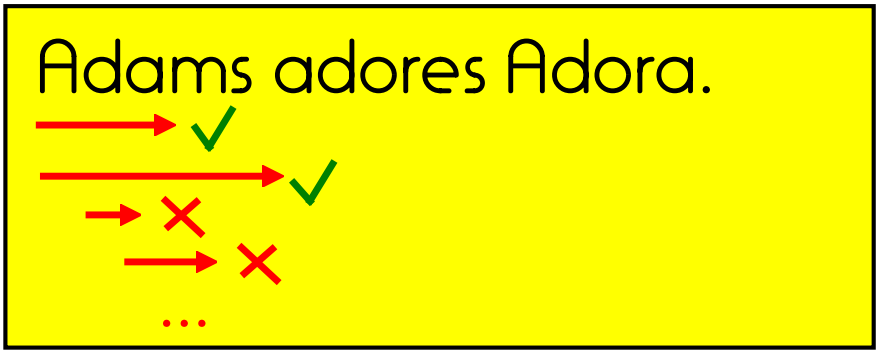
For every character in the doc
- advance as far as possible in the trie and
- report match whenever you meet a "true" node

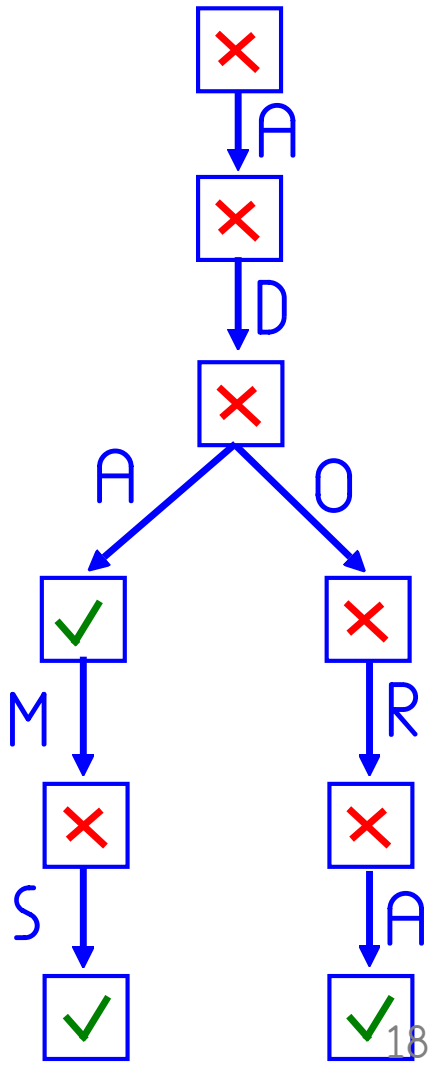


# Tries have good runtime

$O(textLength \times maxWordLength)$



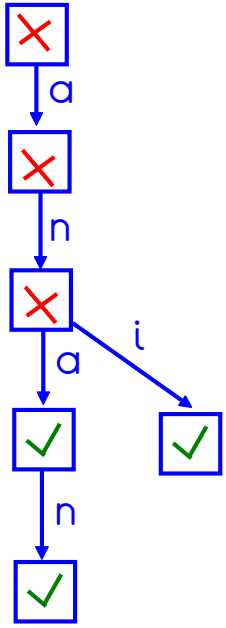

>tasks

# Task: NER with tries

Do NER with the trie from the last task
on the document

on aime un bon bonbon.

# One more example

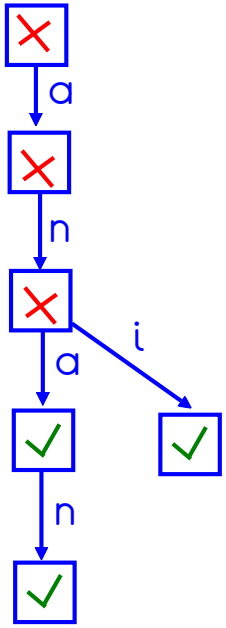



kofi anan eats an ananas in panama, ana!

# Tricks with Tries

In practice, you can

- force tries to respect word boundaries
- ignore upcase/lowcase
- preprocess the string
- ignore nested words
- ...




kofi anan eats an ananas in panama, ana!

# Dictionary NER

Advantages:

- very efficient,

Disadvantages: dictionaries

- have to be given upfront
- have to be maintened to accommodate new names
- cannot deal with name variants
- cannot deal with infinite or unknown sets of names (e.g., people's names)

# Overview

- Named Entity Recognition
- ...by dictionary
- ...by regex

# Some names follow patterns

The trilogy consist of 5 books, written in 1979, 1980, 1982, 1984, and 1992, respectively.

Years

Dr. Frankie and Dr. Benjy have no doctoral title.

People with titles

Main street 42
West Country

Addresses

# Def: 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

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}
L5 = {a, b, aa, bb, aaa, bbb, ...}
L6 = {a, aa, aaa, ...}
L7 = {ab, abab, ababab, ...}
L8 = {c, ca, caa, caaa, ...}
L9 = { , a, aa, aaa, ...}

Set of strings
we want to describe

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}
L5 = {a, b, aa, bb, aaa, bbb, ...}
L6 = {a, aa, aaa, ...}
L7 = {ab, abab, ababab, ...}
L8 = {c, ca, caa, caaa, ...}
L9 = { , a, aa, aaa, ...}  a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}
L5 = {a, b, aa, bb, aaa, bbb, ...}
L6 = {a, aa, aaa, ...}
L7 = { , ab, abab, ababab, ...}
L8 = {c, ca, caa, caaa, ...} ca*
L9 = { , a, aa, aaa, ...} a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}
L5 = {a, b, aa, bb, aaa, bbb, ...}
L6 = {a, aa, aaa, ...}
L7 = { , ab, abab, ababab, ...}  (ab)*
L8 = {c, ca, caa, caaa, ...}  ca*
L9 = { , a, aa, aaa, ...}  a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}
L5 = {a, b, aa, bb, aaa, bbb, ...}
L6 = {a, aa, aaa, ...} aa* = a+
L7 = { , ab, abab, ababab, ...} (ab)*
L8 = {c, ca, caa, caaa, ...} ca*
L9 = { , a, aa, aaa, ...} a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}
L5 = {a, b, aa, bb, aaa, bbb, ...} $a^+ | b^+$
L6 = {a, aa, aaa, ...} $aa^* = a^+$
L7 = { , ab, abab, ababab, ...} $(ab)^*$
L8 = {c, ca, caa, caaa, ...} $ca^*$
L9 = { , a, aa, aaa, ...} $a^*$

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}
L2 = {1900, 1901, 1982, 2013, 2017, ...}
L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
L4 = {a, ab, abb, bbba, aaabbab, ababa, ...} $(a|b)^+$
L5 = {a, b, aa, bb, aaa, bbb, ...} $a^+|b^+$
L6 = {a, aa, aaa, ...} $aa^* = a^+$
L7 = { , ab, abab, ababab, ...} $(ab)^*$
L8 = {c, ca, caa, caaa, ...} $ca^*$
L9 = { , a, aa, aaa, ...} $a^*$

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, Marvin}

L2 = {1900, 1901, 1982, 2013, 2017, ...}

L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}  (0|1|...|9) = [0-9]

L4 = {a, ab, abb, bbba, aaabbab, ababa, ...}  (a|b)+

L5 = {a, b, aa, bb, aaa, bbb, ...}  a+|b+

L6 = {a, aa, aaa, ...}  aa* = a+

L7 = { , ab, abab, ababab, ...}  (ab)*

L8 = {c, ca, caa, caaa, ...}  ca*

L9 = { , a, aa, aaa, ...}  a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, ...}

L2 = {1900, 1901, 1982, 2013, 2017, ...} [0-9][0-9][0-9][0-9] = [0-9]{4}

L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (0|1|...|9) = [0-9]

L4 = {a, ab, abb, bbba, aaabbab, ababa, ...} (a|b)+

L5 = {a, b, aa, bb, aaa, bbb, ...} a+|b+

L6 = {a, aa, aaa, ...} aa* = a+

L7 = { , ab, abab, ababab, ...} (ab)*

L8 = {c, ca, caa, caaa, ...} ca*

L9 = { , a, aa, aaa, ...} a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Def: Language

A language over an alphabet S is a set of words over S.

L1 = {Arthur Dent, Ford Prefect, ...} [A-Z][a-z]+ [A-Z][a-z]+

L2 = {1900, 1901, 1982, 2013, 2017, ...} [0-9][0-9][0-9][0-9] = [0-9]{4}

L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (0|1|...|9) = [0-9]

L4 = {a, ab, abb, bbba, aaabbab, ababa, ...} (a|b)+

L5 = {a, b, aa, bb, aaa, bbb, ...} a+|b+

L6 = {a, aa, aaa, ...} aa* = a+

L7 = { , ab, abab, ababab, ...} (ab)*

L8 = {c, ca, caa, caaa, ...} ca*

L9 = { , a, aa, aaa, ...} a*

Set of strings we want to describe

Our description (much shorter than the original set!)

# Regular expressions

A regular expression (regex) describes a language.

L1 = {Arthur Dent, Ford Prefect, ...} [A-Z][a-z]+ [A-Z][a-z]+

L2 = {1900, 1901, 1982, 2013, 2017, ...} [0-9][0-9][0-9][0-9] = [0-9]{4}

L3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} (0|1|...|9) = [0-9]

L4 = {a, ab, abb, bbba, aaabbab, ababa, ...} (a|b)+

L5 = {a, b, aa, bb, aaa, bbb, ...} a+|b+

L6 = {a, aa, aaa, ...} aa* = a+

L7 = { , ab, abab, ababab, ...}

L8 = {c, ca, caa, caaa, ...} ca*

L9 = { , a, aa, aaa, ...} a*

$L(E\,|F) = L(E) \cup L(F)$

$L(EF) = \{w_1\, w_2 : w_1 \in L(E), w_2 \in L(F)\}$

$L(E*) = \{w_1\, w_2 \ldots w_n : w_i \in L(E), n \geq 0\}$

$L(x) = \{x\}$ for symbols $x$

The language of a regular expression

# Def: Regular expressions

A regular expression (regex) over an alphabet $S$ is one of the following:

- the empty string
- an element of $S$
- a string of the form XY
- a string of the form (X|Y)
- a string of the form (X)*

where X and Y are regexes

->formal-grammars

A regex is associated to a language:

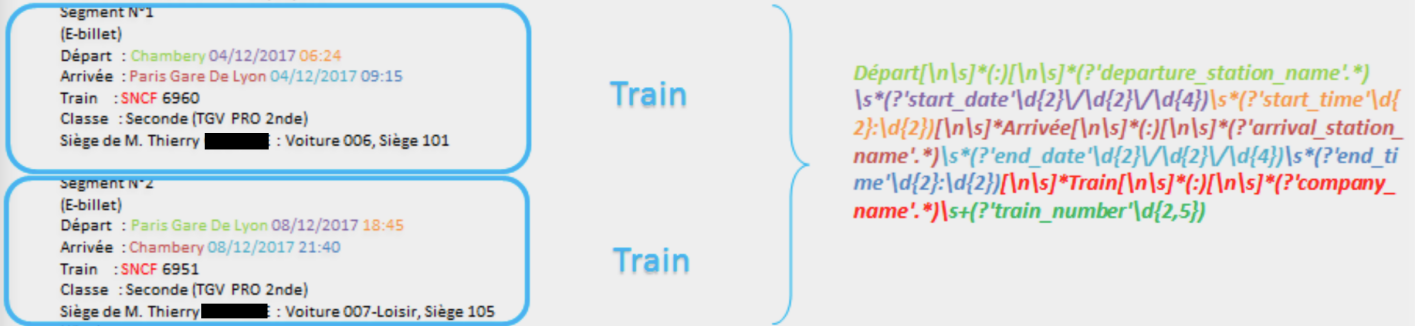
$$L(E\,|F) = L(E) \cup L(F)$$
$$L(EF) = \{w_1\, w_2 : w_1 \in L(E), w_2 \in L(F)\,\}$$
$$L(E\,*\,) = \{w_1\, w_2 \ldots w_n : w_i \in L(E), n \geq 0\}$$
$$L(x) = \{x\} \text{ for symbols } x$$

the regular expression

its language

(∅ is a special regex that cannot appear as part of other regexes. It represents the empty language and does not match any string.)

# Real-world example




Example from Wipolo via Dhouha Bouamor

# Regular expressions cheat sheet

| | |
|---|---|
| L(a) = {a} | Simple symbol |
| L(ab) = {ab} | Concatenation |
| L(a\|b) = {a, b} | Disjunction |
| L(a*) = {, a, aa, aaa, ...} | Kleene star |
| L(a+) := L(aa*) | shorthand for "one or more" |
| L([a-z]) := L(a\|b\|...\|z) | shorthand for a range |
| L(a{2,4}) := L(aa\|aaa\|aaaa) | shortand for a given number |
| L(a{3}) := L(aaa) | shortand for a given number |
| L(a?) := L(\| a) | shorthand for "optional" |
| L(.) := {a\|b\|..\|A\|...\|0\|...\|!\|$\|...} | shorthand for "any symbol" |
| L(\.) := {.} | escape sequence for special symbols |

# Task: Regexes

| | |
|---|---|
| L(a) = {a} | Simple symbol |
| L(ab) = {ab} | Concatenation |
| L(a\|b) = {a, b} | Disjunction |
| L(a*) = {, a, aa, aaa, ...} | Kleene star |
| L(a+) := L(aa*) | shorthand for "one or more" |
| L([a-z]) := L(a\|b\|...\|z) | shorthand for a range |
| L(a{2,4}) := L(aa\|aaa\|aaaa) | shortand for a given number |
| L(a{3}) := L(aaa) | shortand for a given number |
| L(a?) := L(\| a) | shorthand for "optional" |
| L(.) := {a\|b\|..\|A\|...\|0\|...\|!\|$\|...} | shorthand for "any symbol" |
| L(\.) := {.} | escape sequence for special symbols |

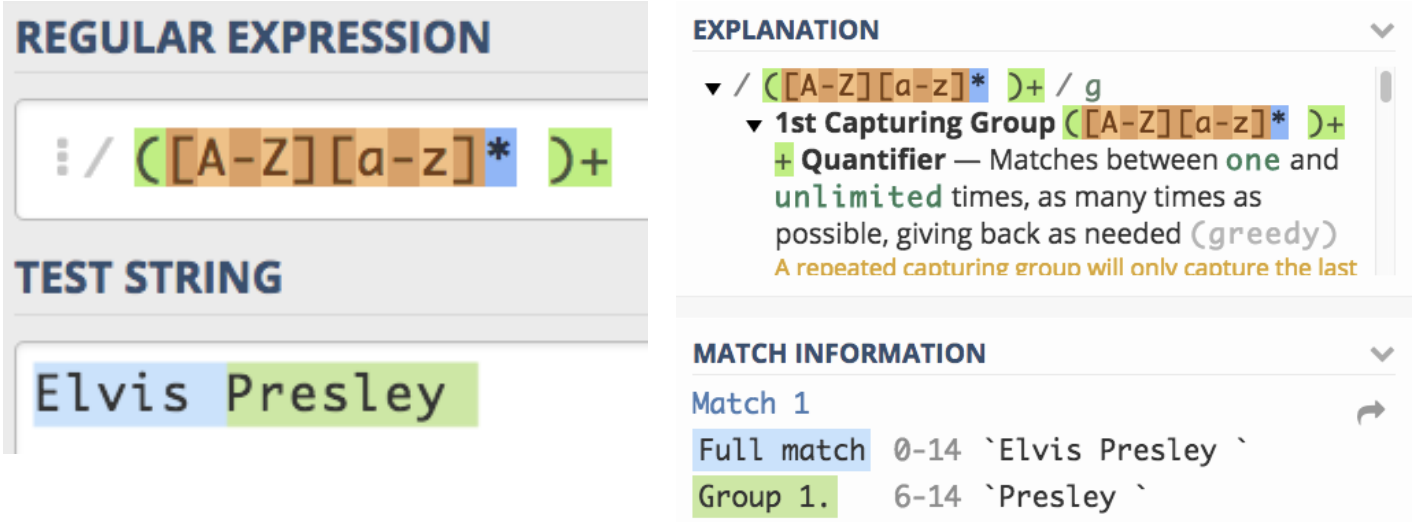
**Define regexes for**

- **numbers**
- **phone numbers**
- HTML tags
- Names of the form "Dr. Blah Blub"

Try it

# Helpful Web page

https://regex101.com/



# Named regexes

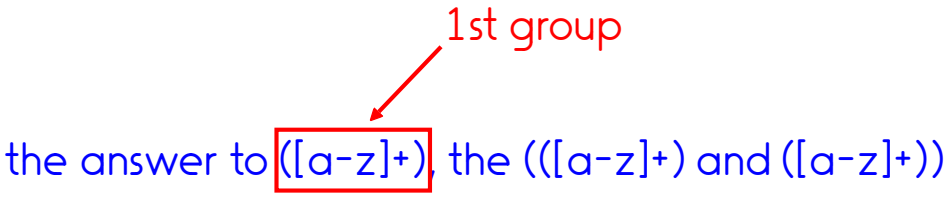
When using regular expressions in a program, it is a good idea to name them:

```
String digits = "[0-9]+";
String separator = "(|-)";
String pattern = digits+separator+digits;
```

But: Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so. (Douglas Adams)

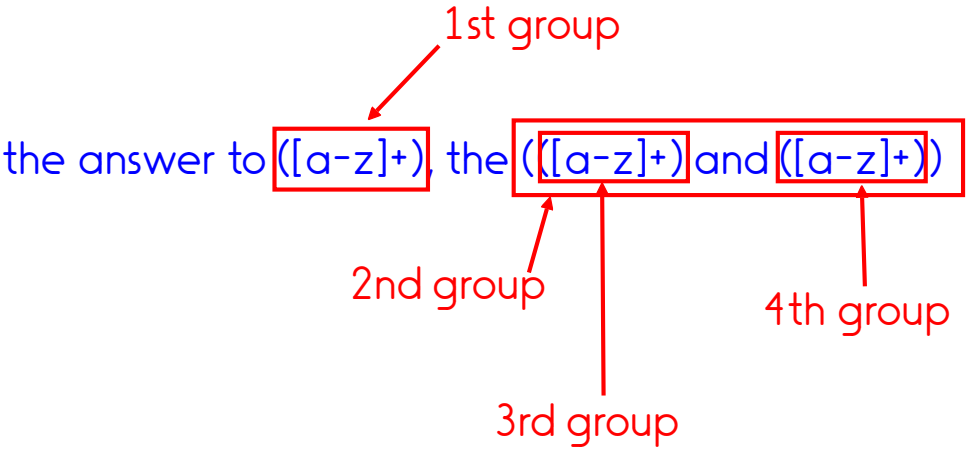
# Regex groups

A regex group is a sequence of the form (...) in a regex.



# Regex groups

A regex group is a sequence of the form (...) in a regex.




Groups are numbered by the order of their opening paranthesis.

# Regex groups

He found the answer to life,
the universe, and everything

the answer to ([a-z]+), the (([a-z]+) and ([a-z]+))

1st group: life
2nd group: universe and everything
3rd group: universe
4th group: everything

Try it out!

# How can we match regexes?

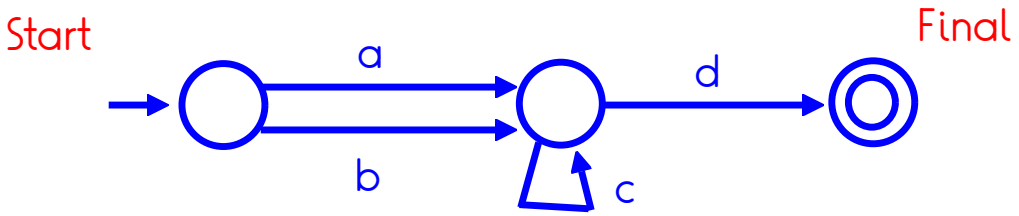
Douglas Adams fictional character names:
[A-Z][a-z]*ch[a-z]*

?

The Hitchhiker meets the Golgafrinchan civilisation, but falls in love with a girl named Fenchurch.

# Def: Finite State Machine

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

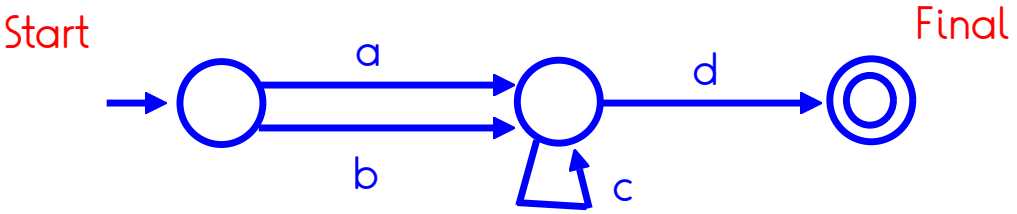



>details

See a more formal definition in Wikipedia

# Def: Acceptance

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



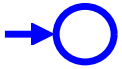

ad ✓

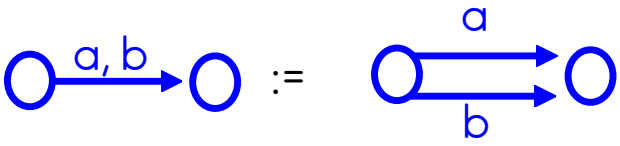
bccd ✓

bb ✗

>details

# Notation

- start node 
- final node 
- empty transition
  (can be walked without
  accepting a symbol) 

- multiple edges 


>details
>task
>trafo

# Task: FSM

Draw an FSM that accepts the following strings

br kbr brbr kbrbr

brbrbr kbrbrbr ...

Draw an FSM that accepts the following strings

ling ping pong long

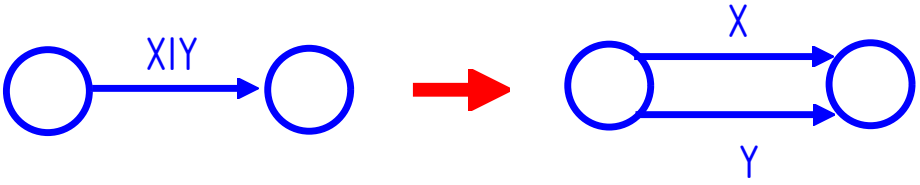
lingping lingpong pingpong

# Transforming a regex to a FSM (1)

1. Simplify the regex to contain only concatenation, alternation, kleene star

2. Start with this configuration:

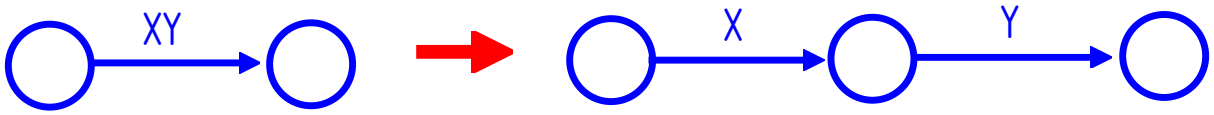

3. Handle alternation:



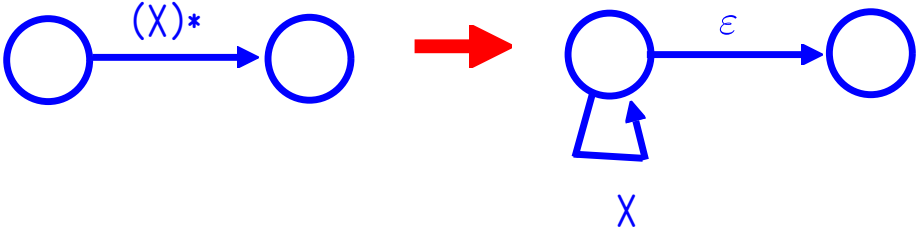

>details

# Transforming a regex to a FSM (2)

4. Handle concatenation:




5. Handle Kleene star:




6. Proceed recursively, until all edges are single symbols

›details

# FSM = Regex

For every regex, there is an FSM that accepts exactly the words of the language of the regex (and vice versa).

Examples:

- k?(br)+
- ((ll|p)(i|o)ng)*
- f{2,3}

>details

# Runtime of regexes

Given a word of length $l$ and given an FSM with $n$ states, determining whether the FSM accepts the word

- takes $O(l)$ time if no state has several outgoing edges with the same label
- $O(l \times n^2)$ else (by making it deterministic on the fly)

There is a looooot more to say about FSMs, e.g.:

- making them deterministic
- compressing them
- making them more powerful
- learning FSMs from examples

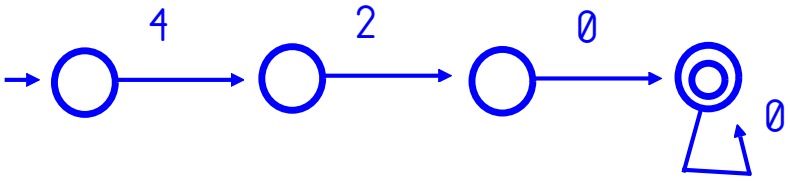
Here, we only use them for IE

# Regexes in programming

Regex `42(0)+`

Simplified regex `420(0)*`

FSM




Matcher

His favorite numbers are 42, 4200, and 19.

you

your programming language

# Example: Regexes in Java

```
Pattern pattern = Pattern.compile("42(0)+");
Matcher matcher = pattern.matcher("His favorite numbers are...");
while(matcher.find())
 System.out.println(matcher.group());
```

→ 4200

His favorite numbers
are 42, 4200, and 19.

# Example: Regexes in Python

```
import re

pattern = re.compile("42(0)+")
match = pattern.search(line)
if match!=None:
   print(match.group())
```

→ 4200

His favorite numbers
are 42, 4200, and 19.

# Entity Recognition

We have seen 2 methods to do entity recognition:

- Tries (if the set of names is known)
- Regexes (if the names follow a pattern)

Douglas N. Adams had the idea for the "Hitchhiker's Guide" while lying drunk in a field near Innsbruck.

References:

Sunita Sarawagi: Information Extraction

->dipre
->disambiguation
->evaluation
->named-entity-annotation