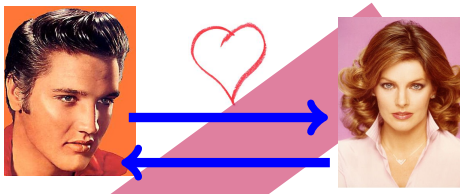
Information Extraction

Lecture 2: Named Entity Recognition

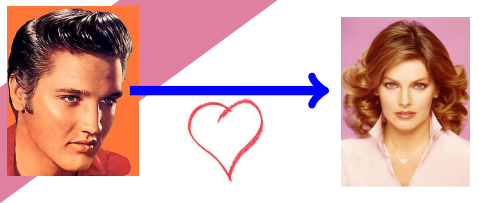
Fabian M. Suchanek

Semantic IE

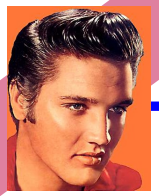
Reasoning



Fact Extraction

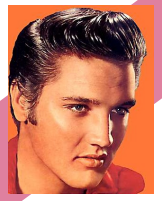


Instance Extraction



singer

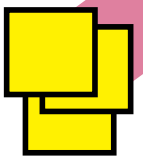
You are here



Entity Disambiguation

singer Elvis

Entity Recognition



Source Selection and Preparation

Overview

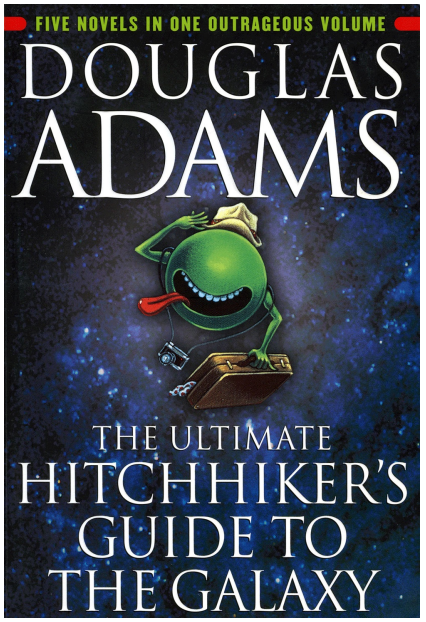
- Source selection
- Character encodings
- Named Entity Recognition (NER)
  - by dictionary
  - with regexes

Def: Corpus

The corpus is the set of digital text documents from which we want to extract information.

Where do we get our corpus from?

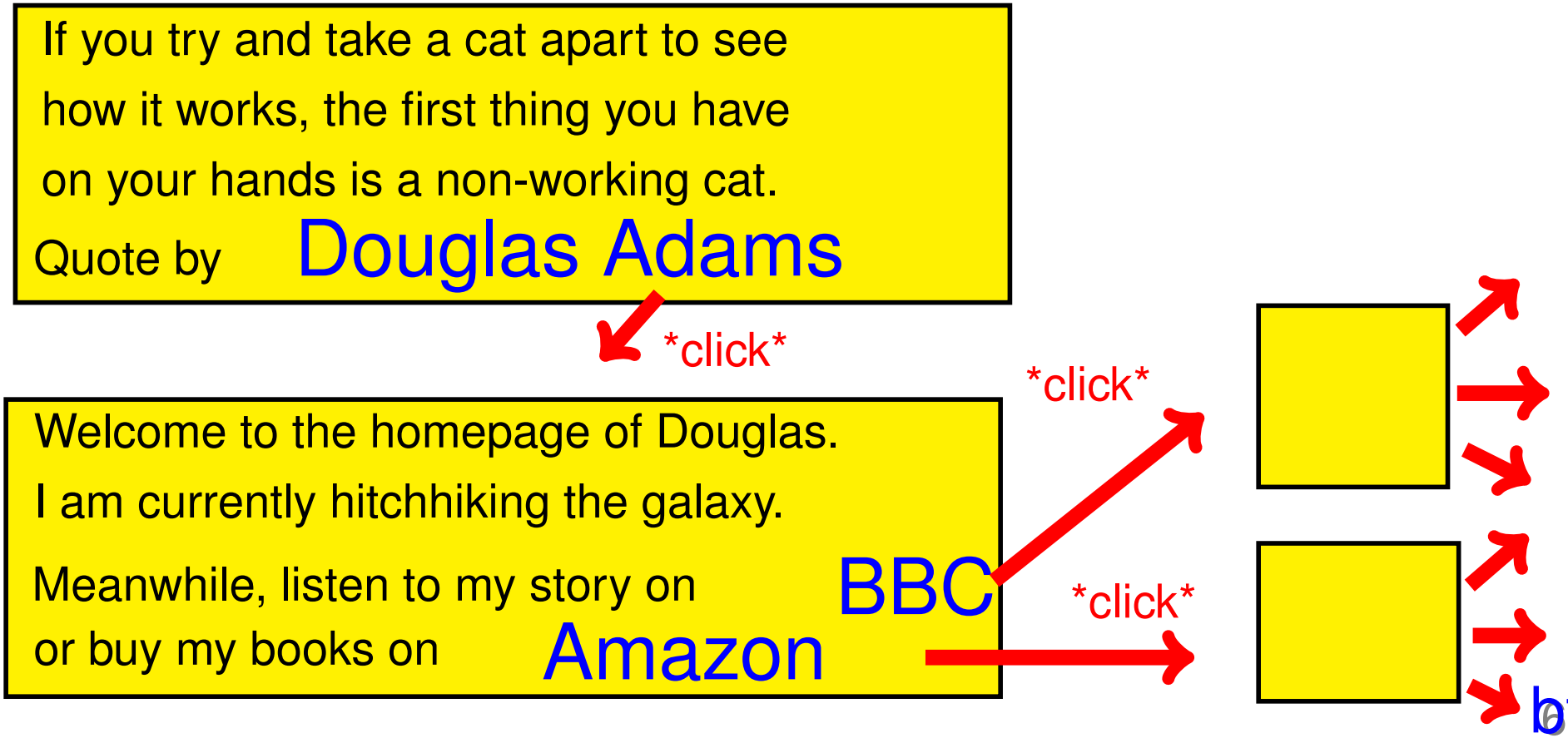
We can use a given corpus

- 

- The New York Times nyt
- Enron Email corpus
- Sarah Palin's emails

We can crawl the Web

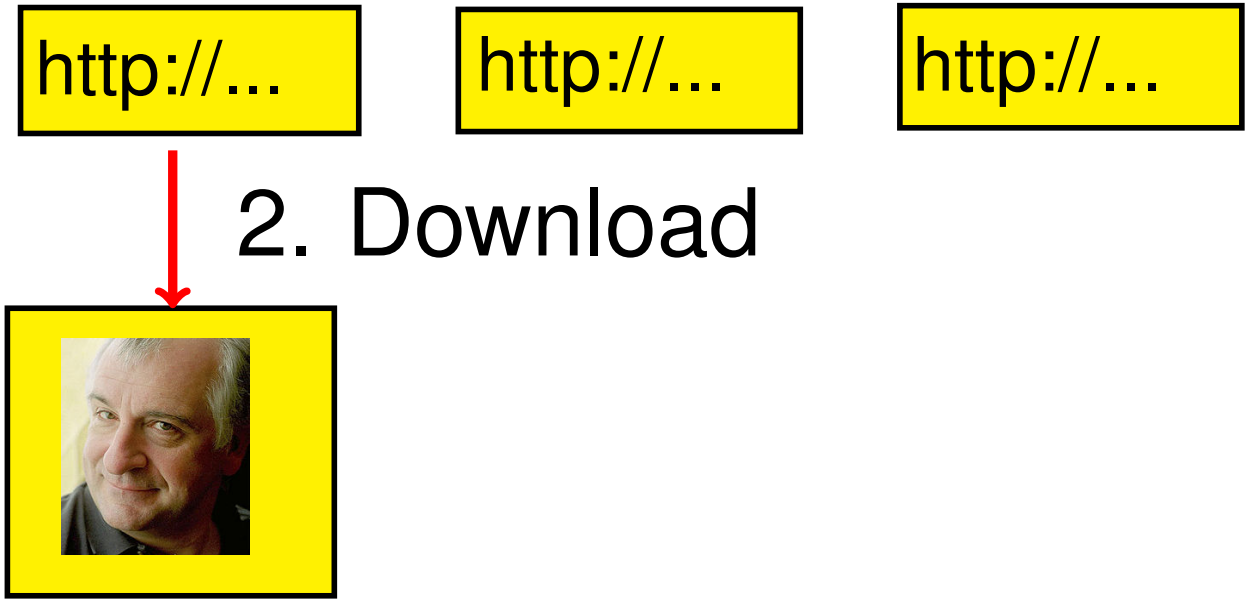
A Web crawler is a system that follows hyperlinks, collecting all pages on the way.



A crawler does BFS on URLs

1. Start with queue of important URLs

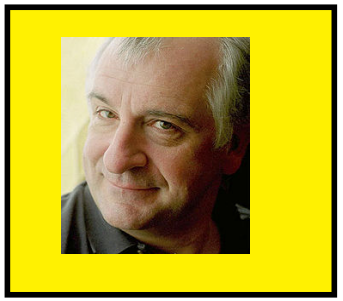
A crawler does BFS on URLs

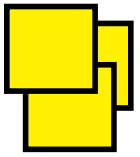


A crawler does BFS on URLs

http://...

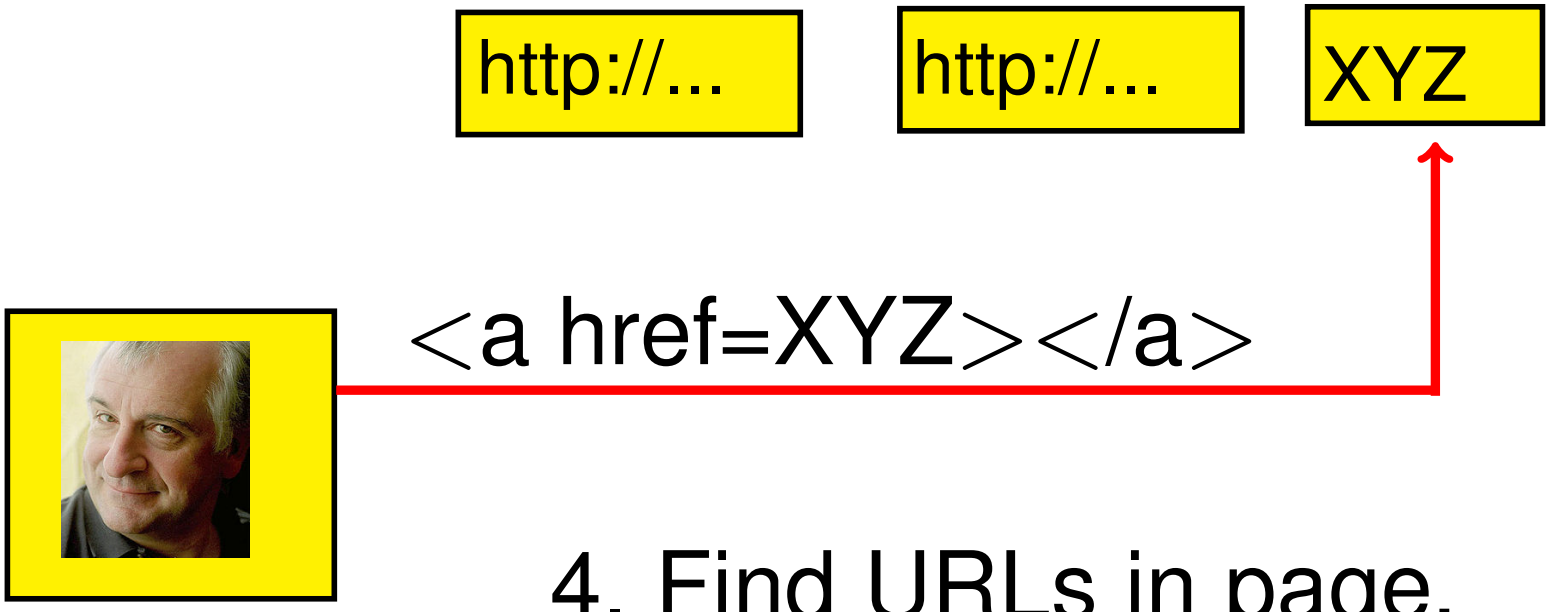
http://...



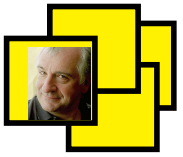


3. If page is “good”, add it to corpus

A crawler does BFS on URLs



4. Find URLs in page, enqueue them



A crawler does BFS on URLs

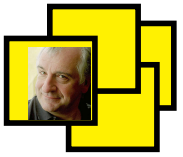
http://...

http://...

XYZ

5. repeat

the process



When does crawling stop?

We can crawl

- up to a certain depth
- only a certain domain
- only pages with certain topics
- “everything”

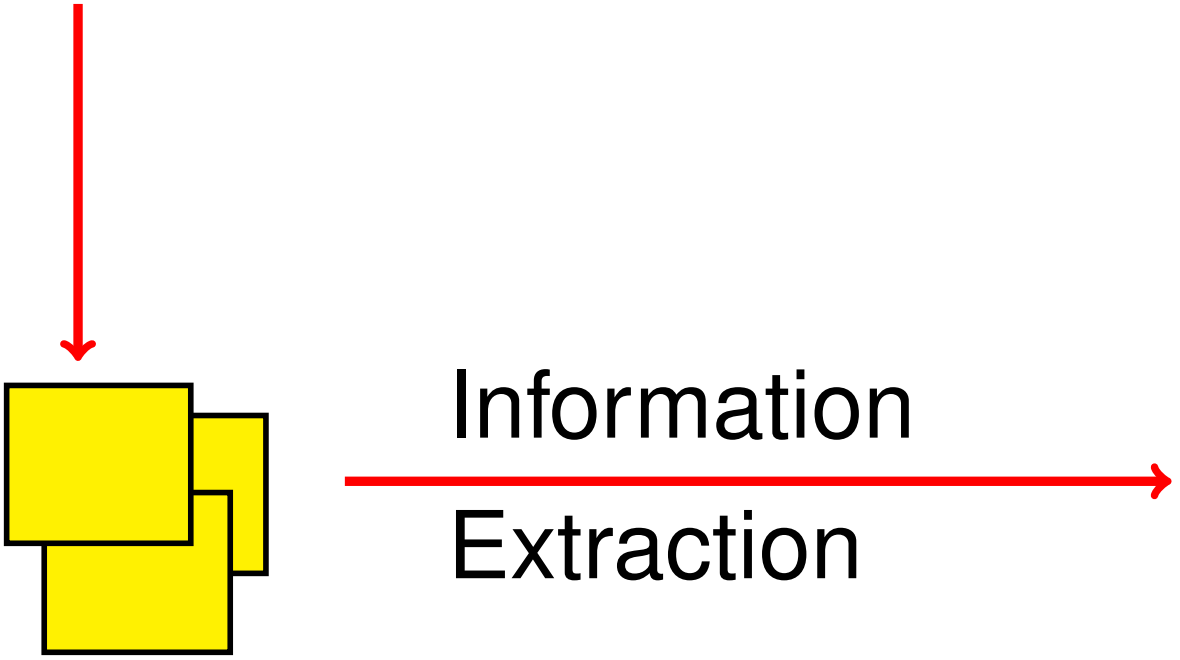
We can use an existing Web crawl

	pages	size	
ClueWeb	1b	25 TB	$$$
CommonCrawl	6b	100TB	free
Internet Archive	2b	80TB	free

We can find pages on demand

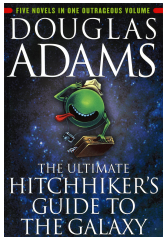
Google

Are vogons vegans?



(At least) 3 ways to build corpus

1. Use given corpus



2. Crawl the Web

http://... http://... http://...

3. Find Web pages on demand

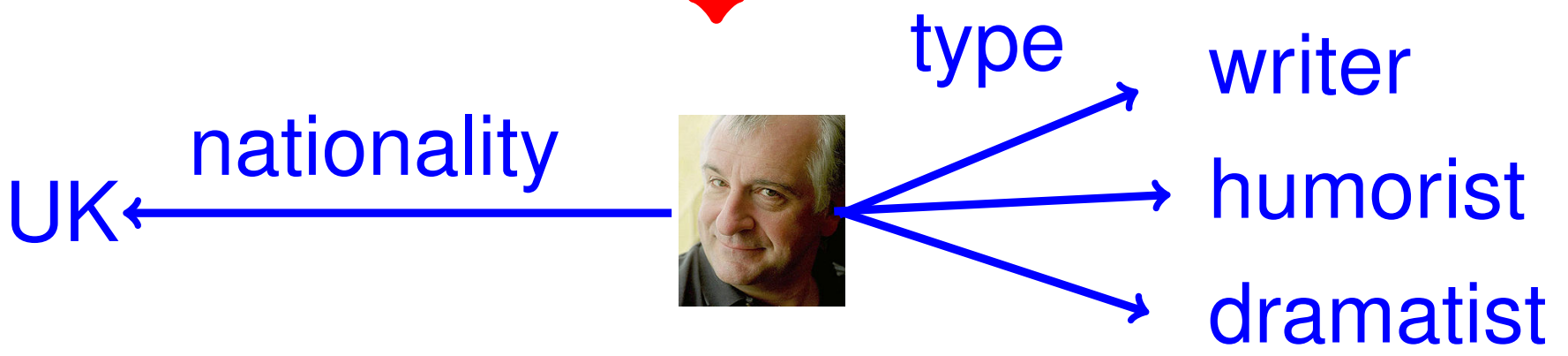
Google

Def: Document-centric IE

Document-centric IE aims to extract every fact from a given document.

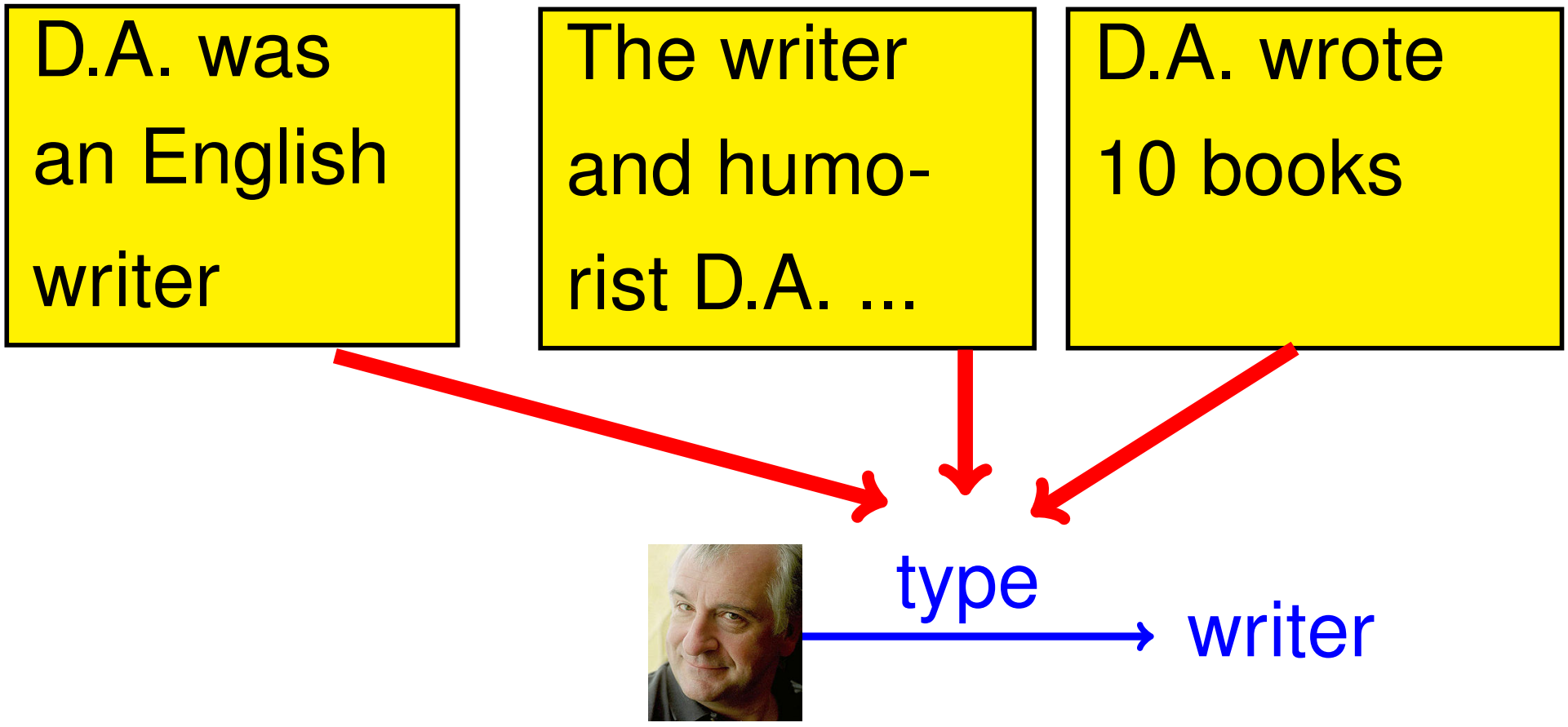
Wikipedia - Douglas Adams:

Douglas Adams was an English writer,humorist,and dramatist.

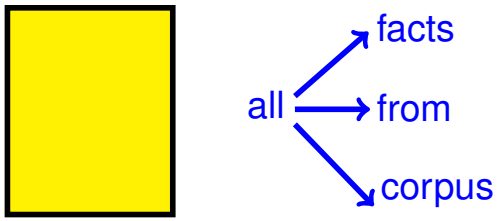


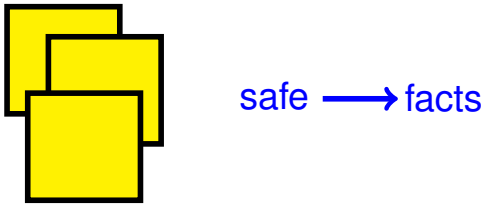
Def: Fact-centric IE

Fact-centric IE aims to extract facts that appear often in the documents.



Document- vs. Fact-centric IE





Doc-centric IE

- cares about every single fact in the corpus
- serves, e.g., for summarization
- is difficult

Fact-centric IE

- cares about evidence for extracted facts
- serves, e.g., to build up a KB
- is “easier”

Overview

- Source selection
- Character encodings
- Named Entity Recognition (NER)
  - by dictionary
  - with regexes

chars>40

Scripts

Thanks for all the fish

Scripts

Thanks for all the fish

(Latin)

(“Simplified” Chinese)

(Hebrew)

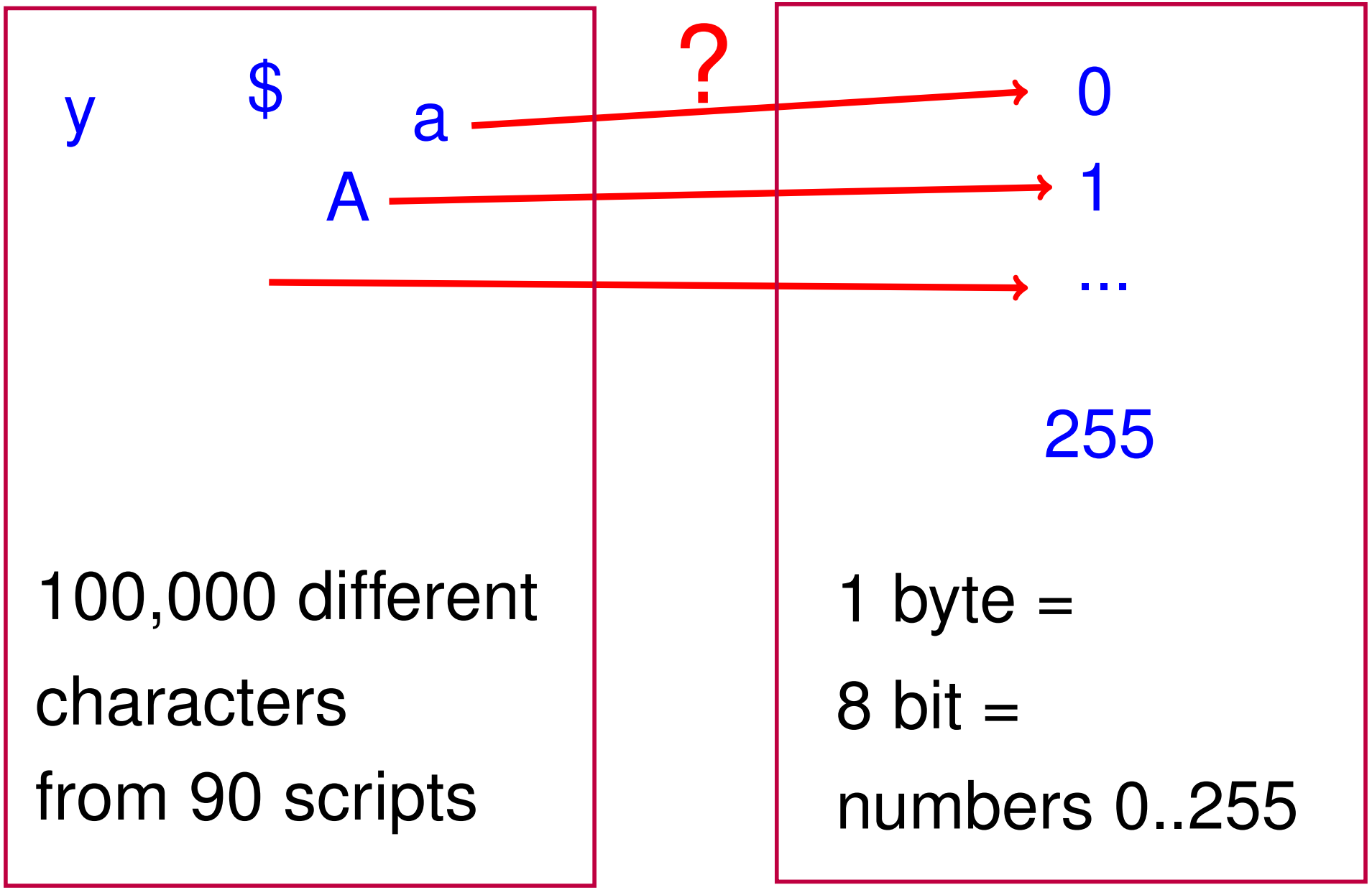
(Arabic)

(Korean)

(Thai)

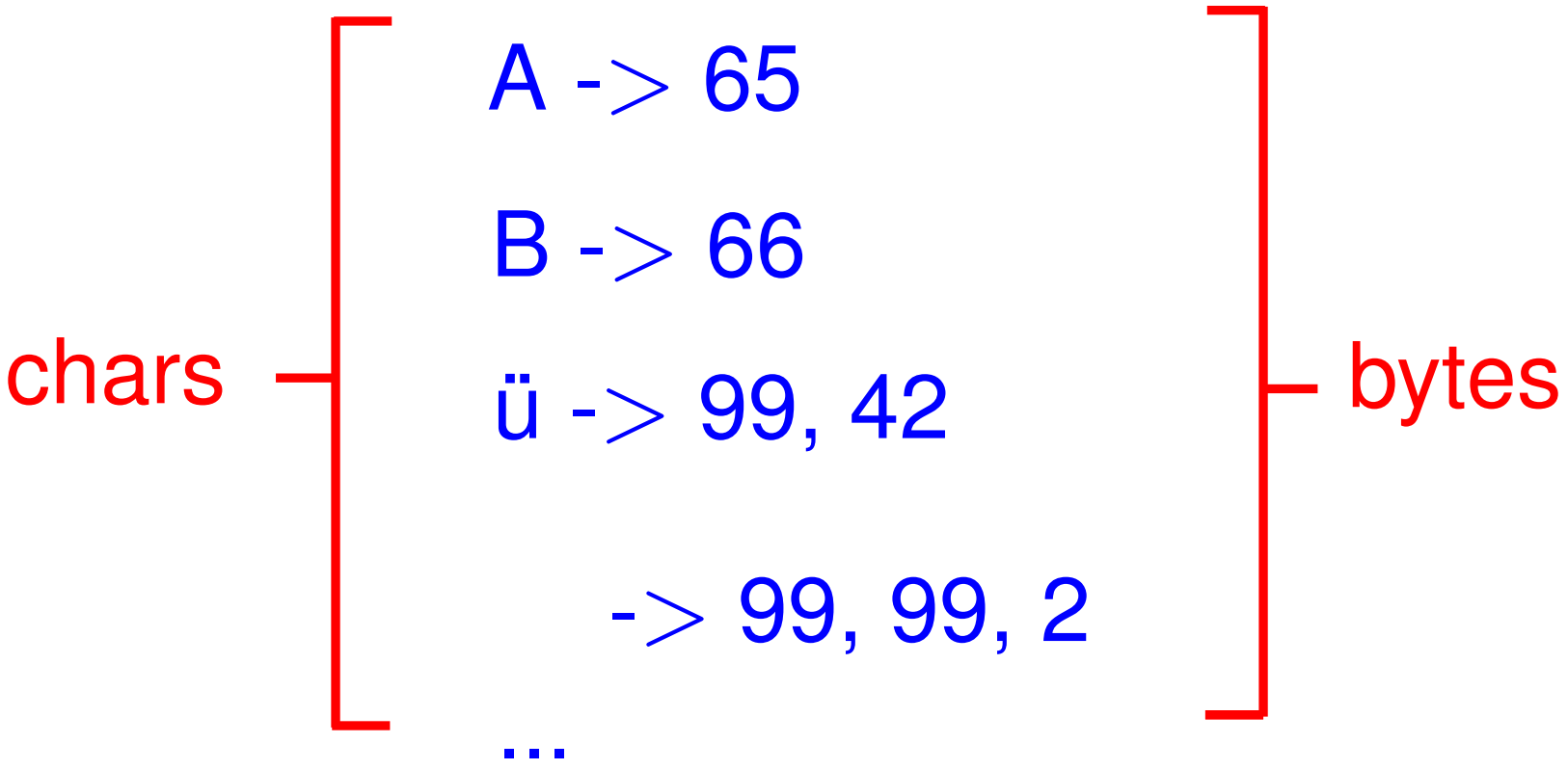
Google Translate. No warranty.

How to map characters to bytes?



Def: Character encoding

A character encoding (also: char encoding) is a bijective mapping from characters to (sequences of) bytes.



Def: ASCII encoding

The ASCII encoding maps certain chars to single bytes, ignores the others.

A -> 65 C -> 67 -> ✗

B -> 66 ... ü -> ✗

26 letters + 26 lowercase letters
+ punctuation 100 characters

Disadvantage: works only for English

Def: Code pages

A code page maps script-specific characters to single bytes.

Greek code page:

A -> 65 -> 224

B -> 66 ...

(0-127 are usually mapped as in ASCII)

Western code page:

A -> 65 à -> 224

B -> 66 ...

Example

Code pages have disadvantages

- We have to know the code page
- We cannot mix scripts
- We cannot represent more than 256 characters

Def: HTML entities

An HTML entity is a string that represents a character (as defined by W3C).

à -> à

ü -> ü

ß -> ß

...

These are sequences of bytes if encoded in ASCII

Example List

Advantage: Works in all browsers

Disadvantage: Very clumsy

Def: Unicode

Unicode maps each character to $[0, 2^{32} - 1]$, i.e., to 4 bytes.

Example1

Example2

A -> 65 -> 0, 0, 0, 65

B -> 66 -> 0, 0, 0, 66

-> 1001 -> 0, 0, 3, 42

-> 2001 -> 0, 0, 4, 17

(not the real mappings)

...

Characters 0-127 are as in ASCII

Advantage: Maps all known characters

Disadvantage: Takes much space

Def: UTF-8

UTF-8 maps Unicode characters to sequences of bytes of different lengths.

A -> 65

B -> 66

-> 128, 42

-> 128, 128, 32

UTF-8: Chars 0-0x7F

Unicode chars 0-0x7F are mapped like in ASCII (i.e., to a single byte).

A -> 65	a -> 96	$ -> 36
B -> 66	b -> 97	! -> 33
...	...	...

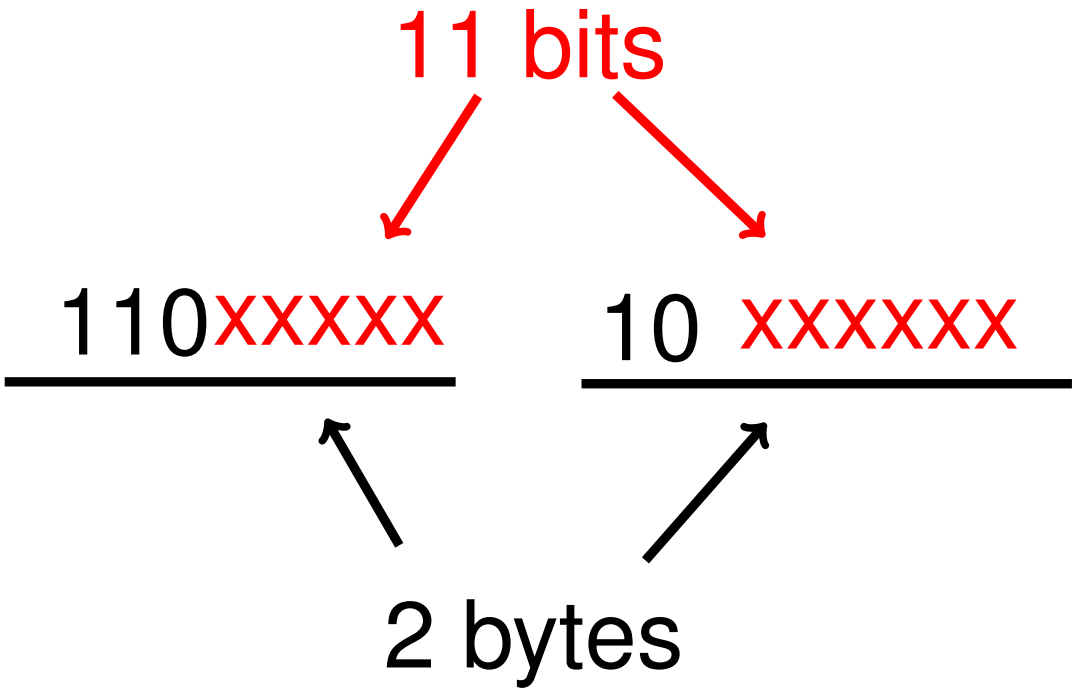
Advantages:

- Compatibility with ASCII and code pages
- Space efficiency for English docs

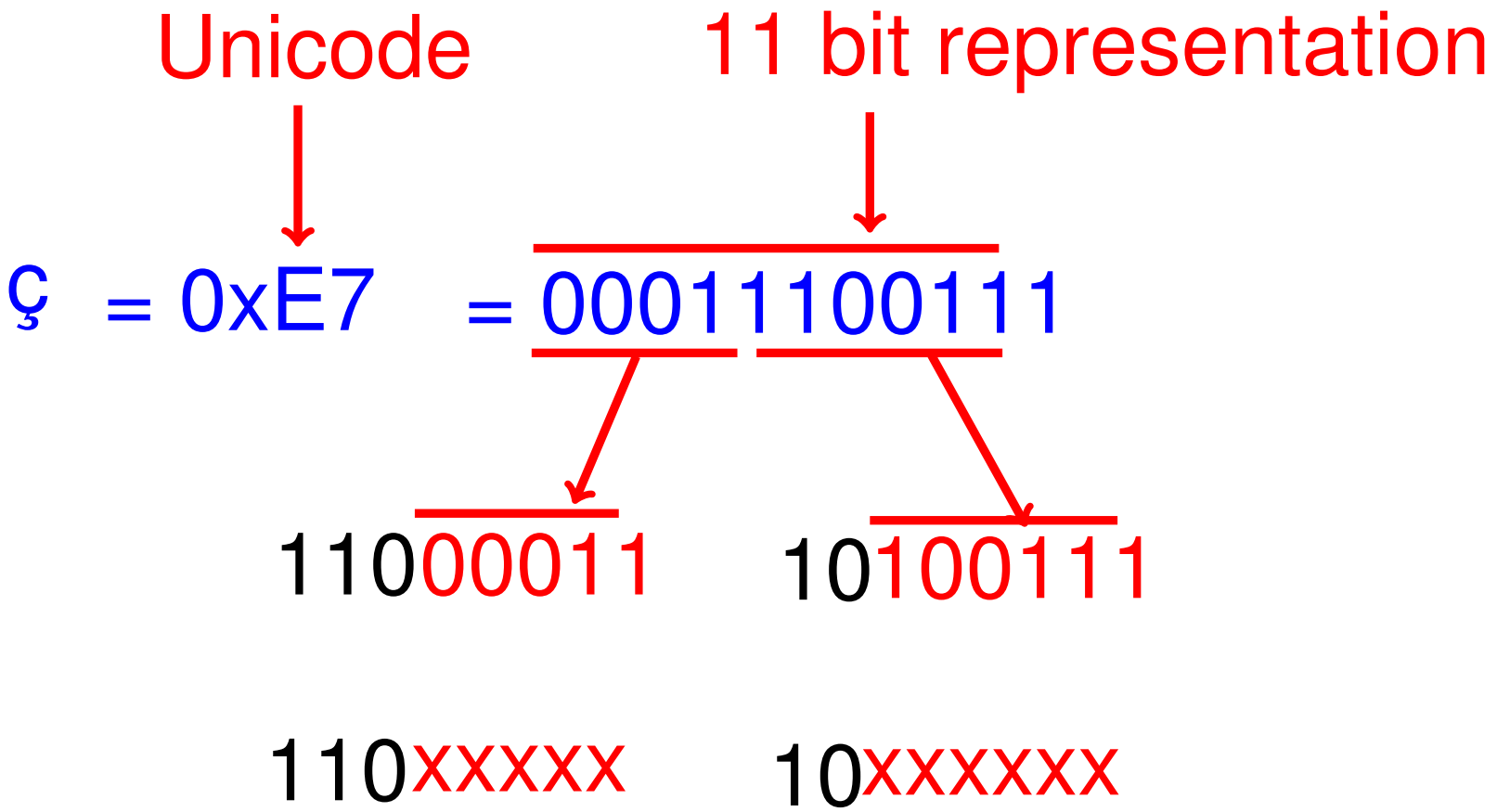
UTF-8: Chars 0x80-0x7FF

Unicode chars 0x80-0x7FF (11 bits)

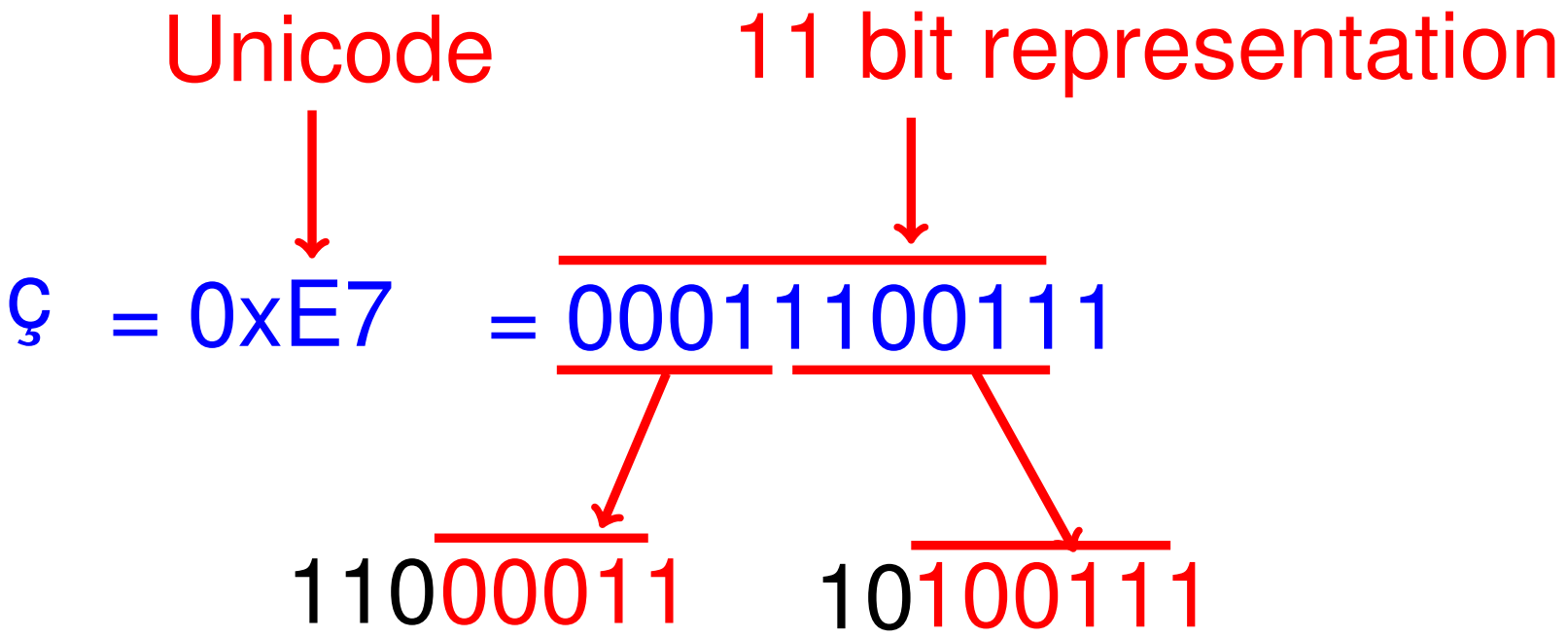
are mapped to two bytes as follows:



UTF-8: Chars 0x80-0x7FF Example



UTF-8: Chars 0x80-0x7FF Example

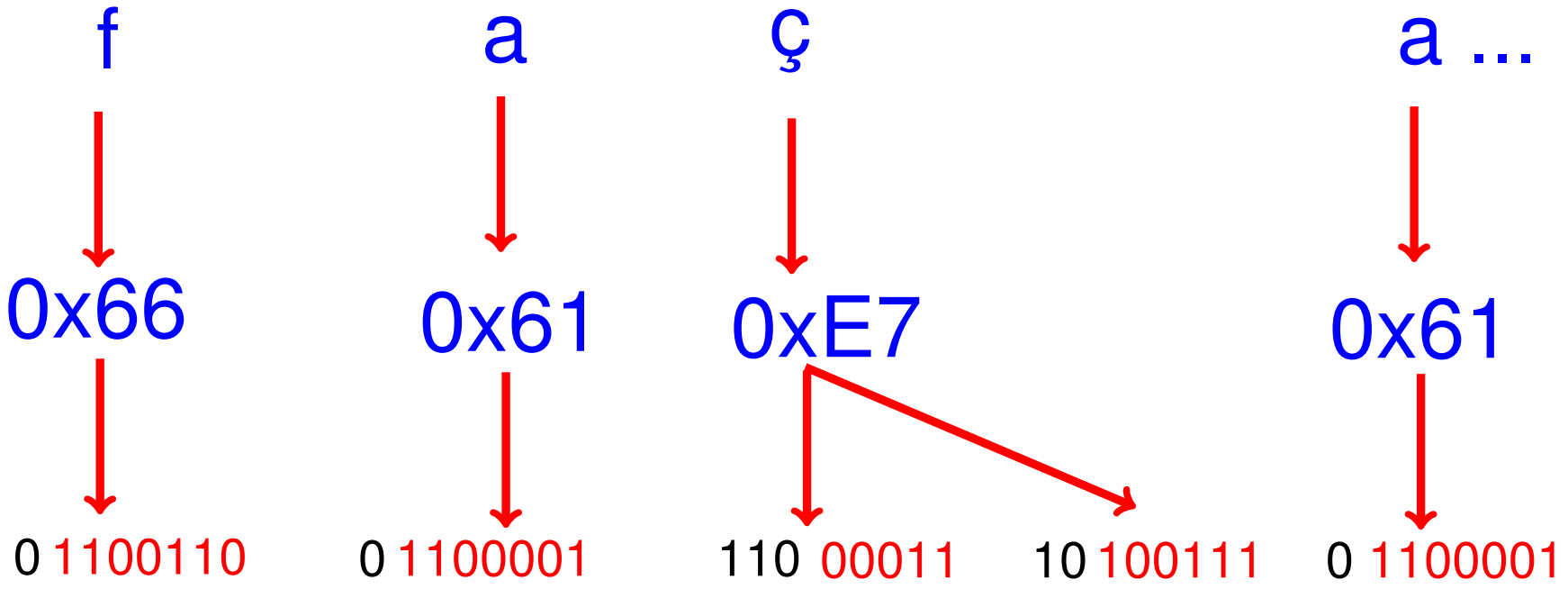


Unicode 9x80-0x7FF are

Greek, Arabic, Hebrew etc.

UTF-8: Chars 0x80-0x7FF Example

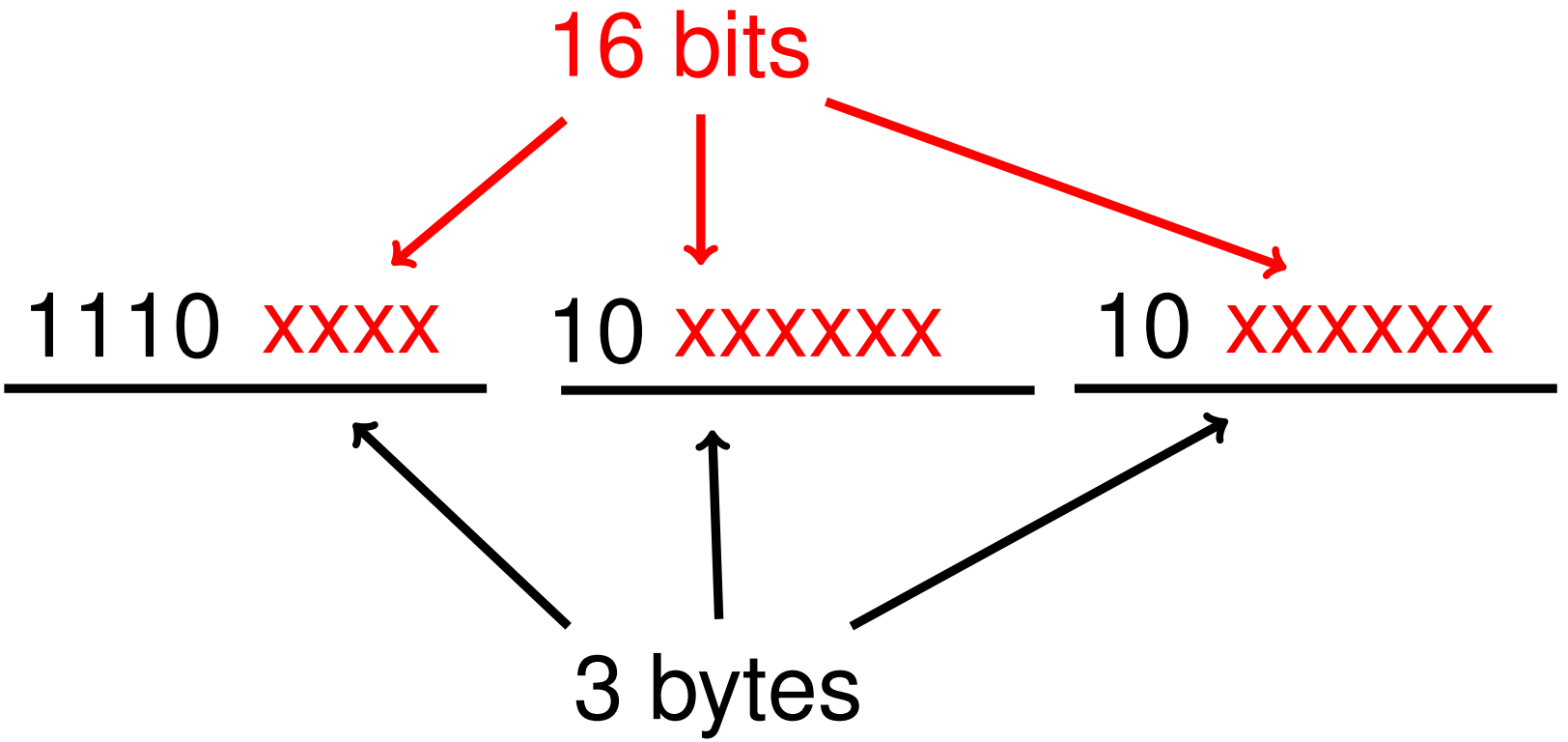
Example: Encoding “façade”



UTF-8: Chars 0x800-0x7FFF

Unicode chars 0x800-0x7FFF (16 bits)

are mapped to three bytes as follows:



Concerns mainly Chinese

Decoding UTF-8

0 1100110	0 1100001	110 00011	10 100111	0 1100001
f	a	ç		a ...

- if the byte starts with 0xxxxxxx
  => it's a "normal" ASCII character
- if the byte starts with 110 xxxxx
  => it's an "extended" char, 1 byte follows
- if the byte starts with 1110 xxxx
  => it's a "Chinese" char, 2 byte follow
- if the byte starts with 10 xxxxxx
  => it's a follower byte, you messed it up!

Summary: UTF-8

UTF-8 maps Unicode chars 0-65535
to 1-4 bytes.

Advantages:

- common Western chars are only 1 byte
- backwards compatibility with ASCII
- stream readability (follower bytes
  cannot be confused with marker bytes)
- sorting compliance

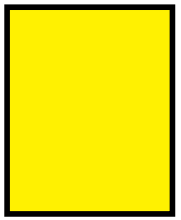
Summary: Char encodings

- ASCII: only English chars
- Code pages: one page per script
- HTML entities: work in browsers
- Unicode: maps all chars
- UTF-8: maps chars to variable # bytes

In most applications, UTF-8 is the encoding of choice.

Example: Char encodings in Java

File f= new File(...);



InputStream s= new FileInputStream(f);

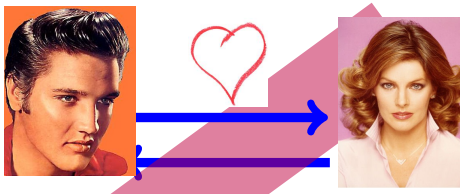
0 1100110 0 1100001 110 00011 10 100111

Reader r= new InputStreamReader(s,“UTF-8”);

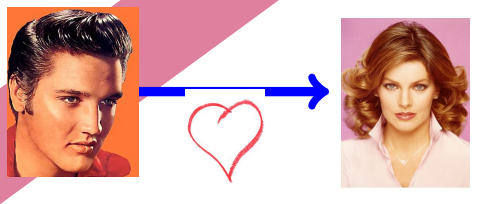
f a ç

Semantic IE

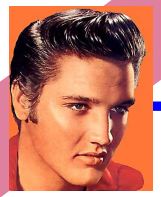
Reasoning



Fact Extraction

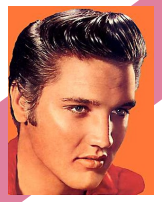


Instance Extraction



→ singer

You are here



Entity Disambiguation

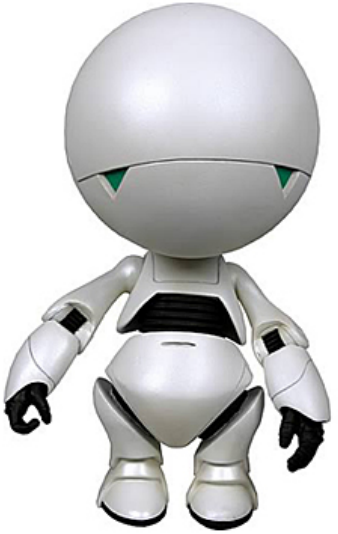
Elvis and Lisa

Entity Recognition

Source Selection and Preparation

Named Entity

A named entity is an entity that has a name (and is not a literal, class, relation, fact id, or reified statement).



Marvin

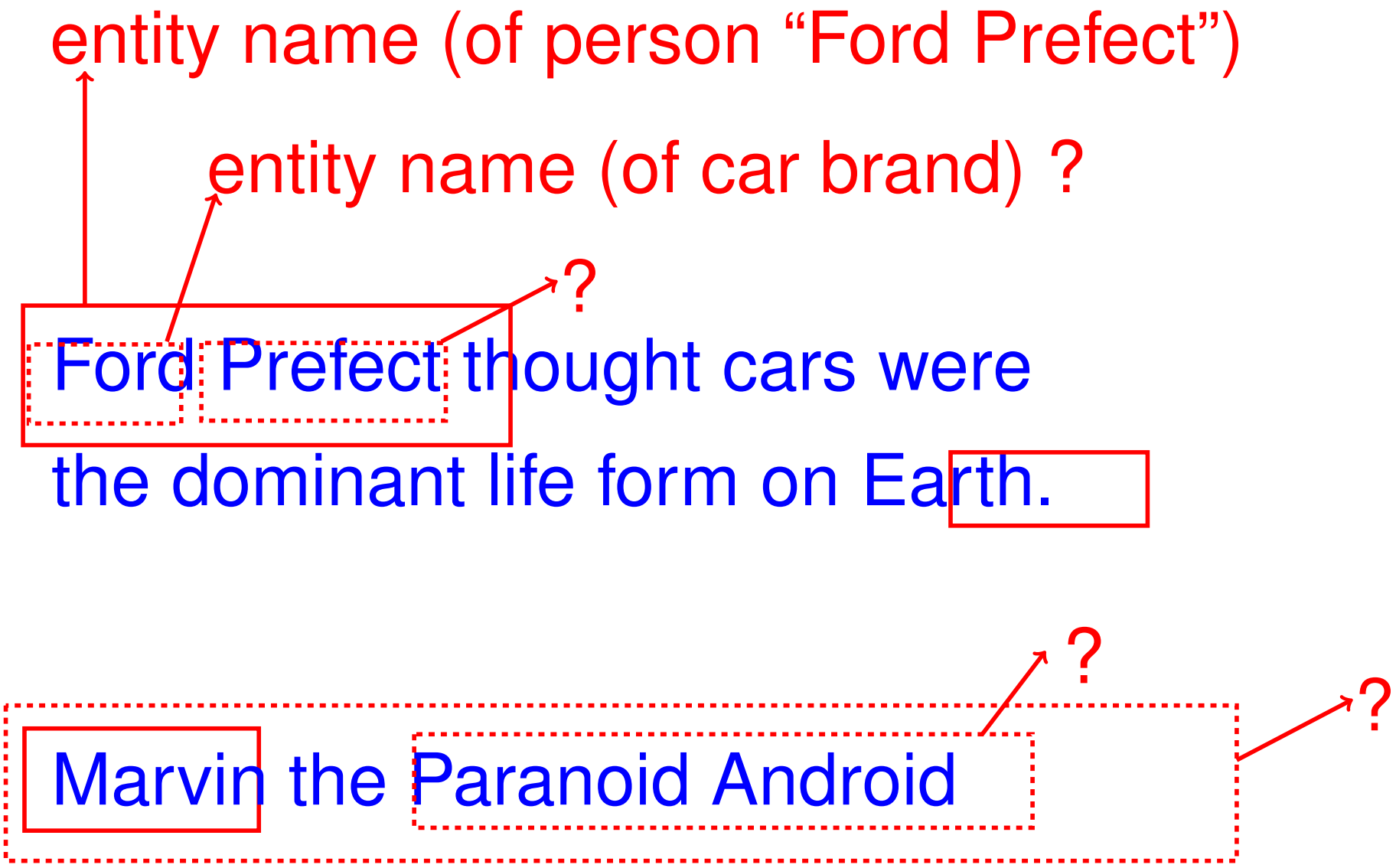
Def: Named Entity Recognition

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

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

Marvin said: “I didn’t ask to be made. No one consulted me or considered my feelings in the matter.“

NER is difficult



NER is difficult

Deposit a penny at the All American Bank...

All State Police helped evacuate...

Cable and Wireless, a major company...

Microsoft and Dell both agreed...

Overview

- Source selection
- Character Encodings
- Named Entity Recognition (NER)
  - by dictionary
  - with regexes

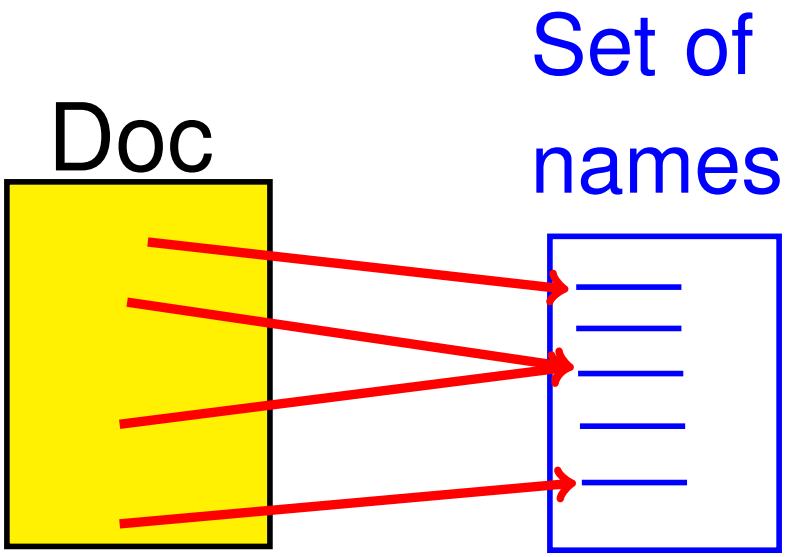
Def: Dictionary

A dictionary (also: gazetteer, lexicon)

is a set of names.

NER by dictionary finds only names

of the dictionary.



- US states
- countries
- DAX companies
- Actors of a given movie
- ...

NER by Dictionary

NER by dictionary can be used if the entities are known upfront.

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

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

Countries (?): {China, Russia, ...}

... while France and Germany were opposed to a 3rd world war...

Naive Dictionary NER is slow

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

The Hitchhiker's Guide to the Galaxy has "Don't Panic" on it, in large, mostly friendly letters.

?

Books by Douglas Adams: {

Life, the Universe and Everything

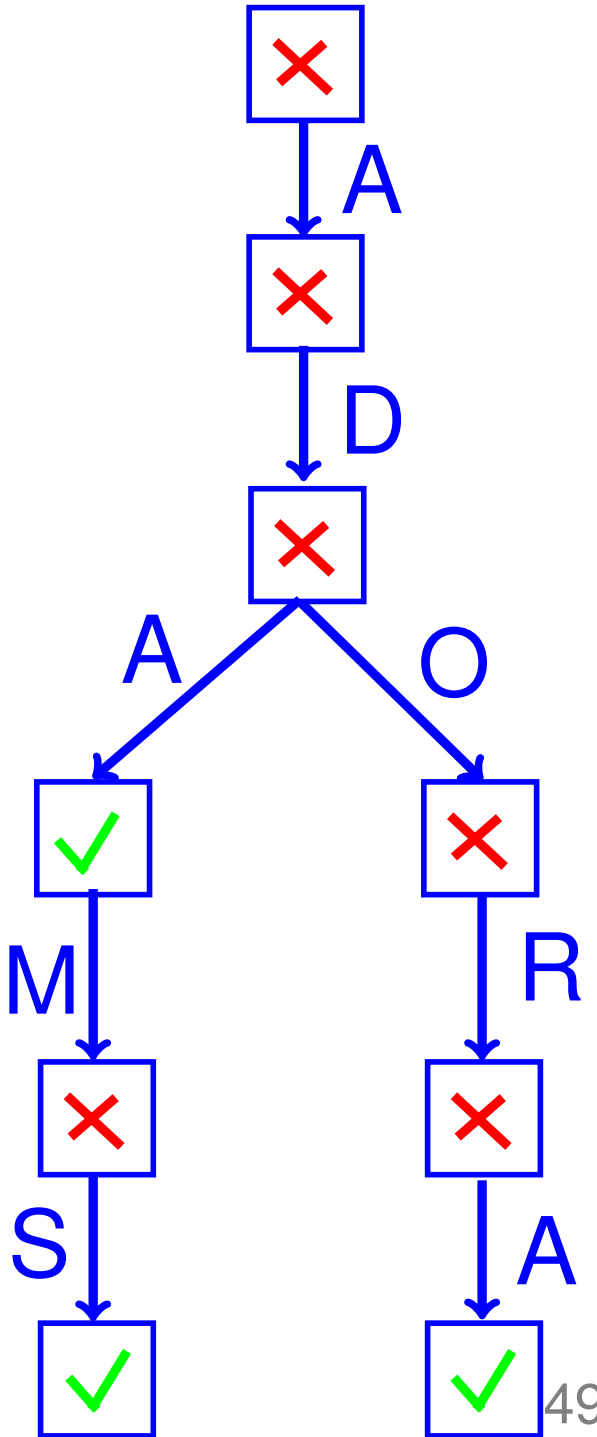
Hitchhiker's Guide to the Galaxy

Mostly Harmless

... }

Def: Trie

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



A trie contains strings

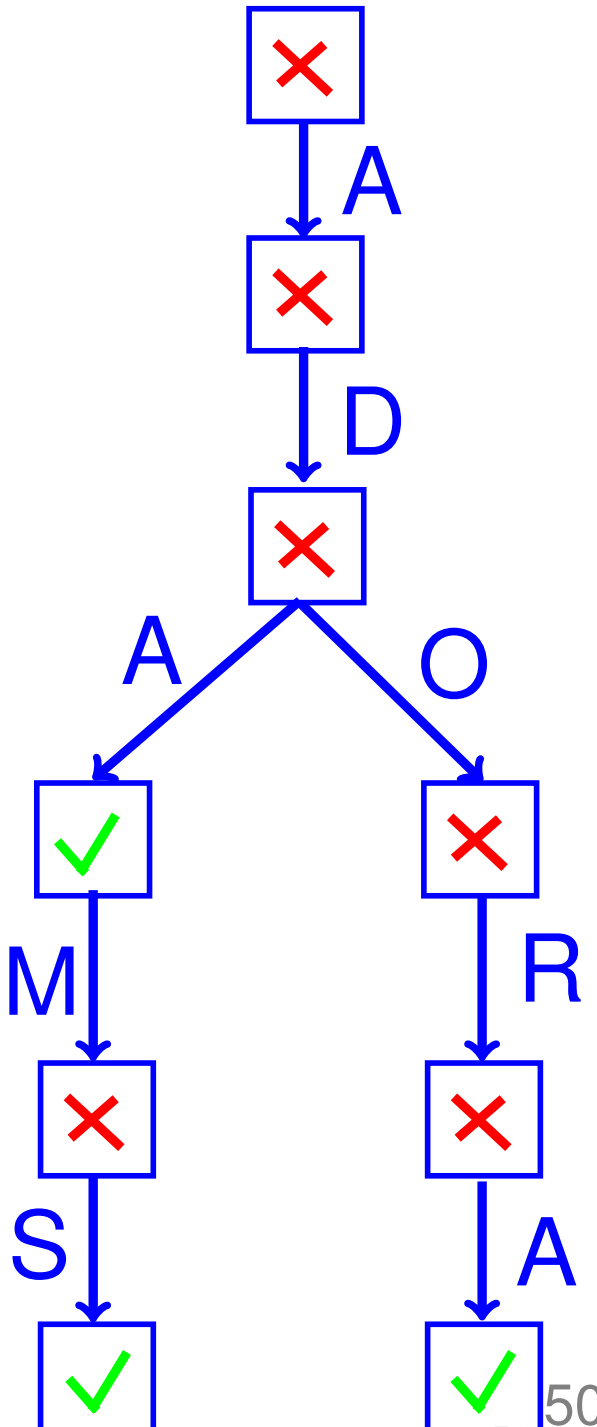
A trie contains a string,

if the string denotes a

path from the root to a

node marked with 'true'.

{ADA, ADAMS, ADORA}

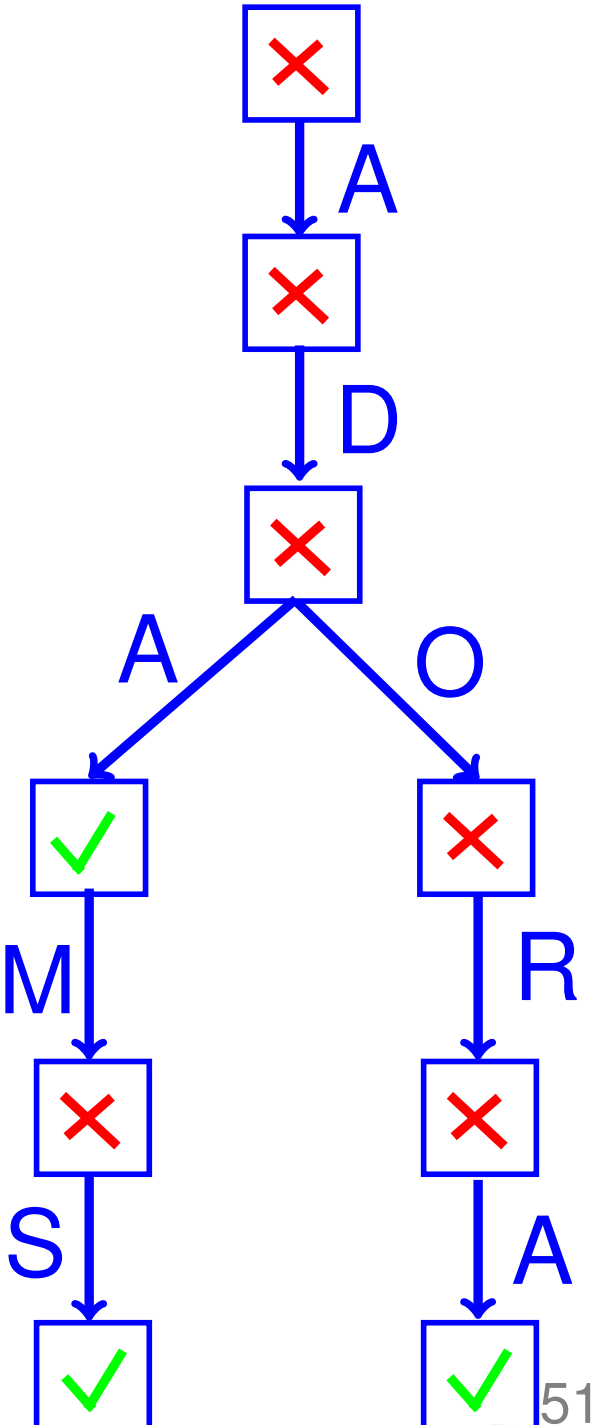


Adding strings (1)

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

{ADA, ADAMS, ADORA}

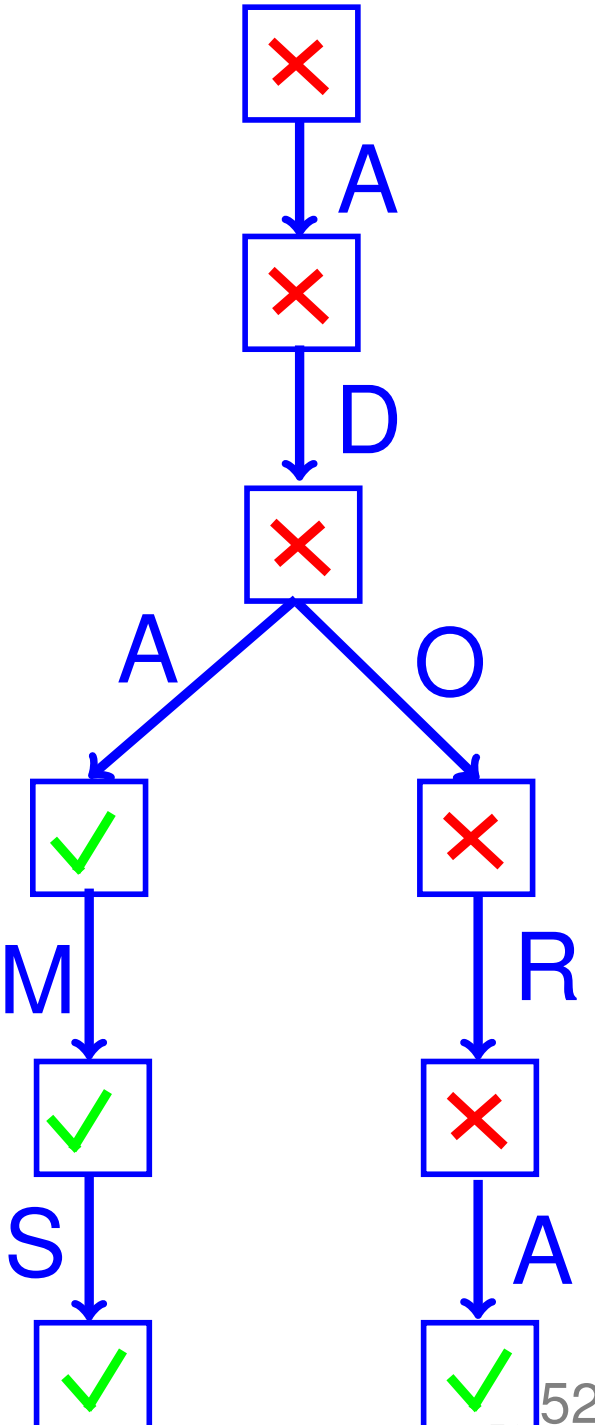
+ ADAM



Adding strings (1)

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

{ADA, ADAM, ADAMS, ADORA}



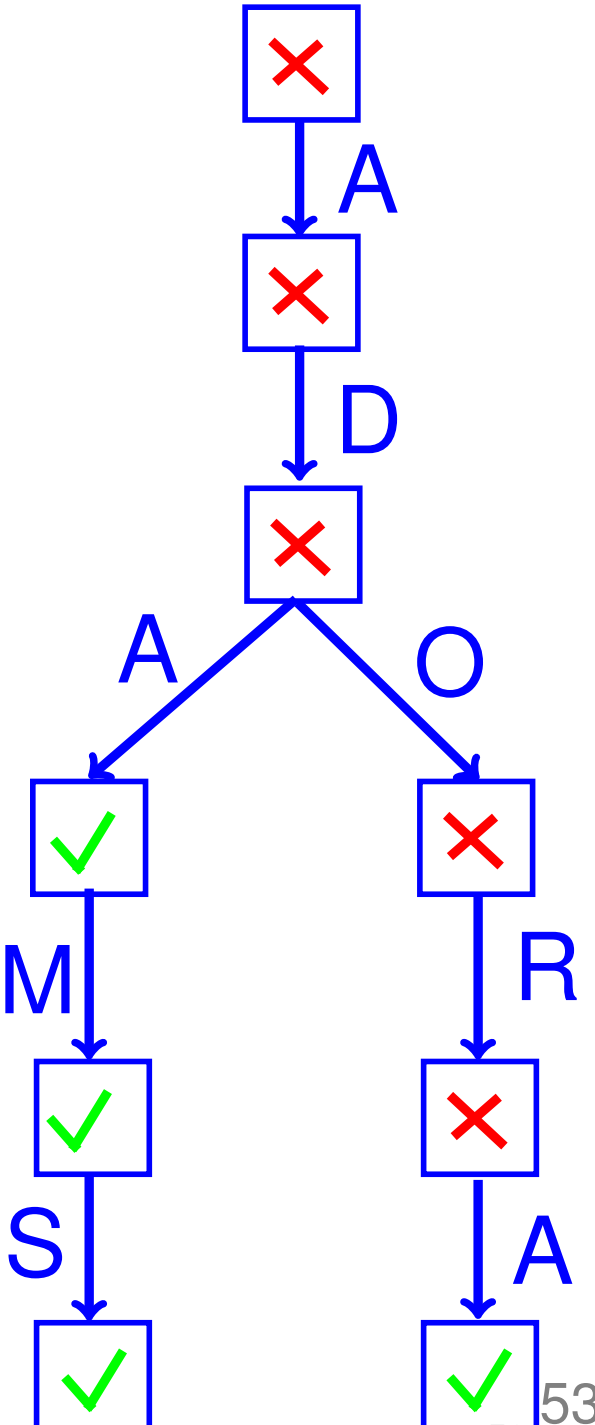
Adding strings (2)

To add another string,

make a new branch.

{ADA, ADAM,

ADAMS, ADORA} + ART



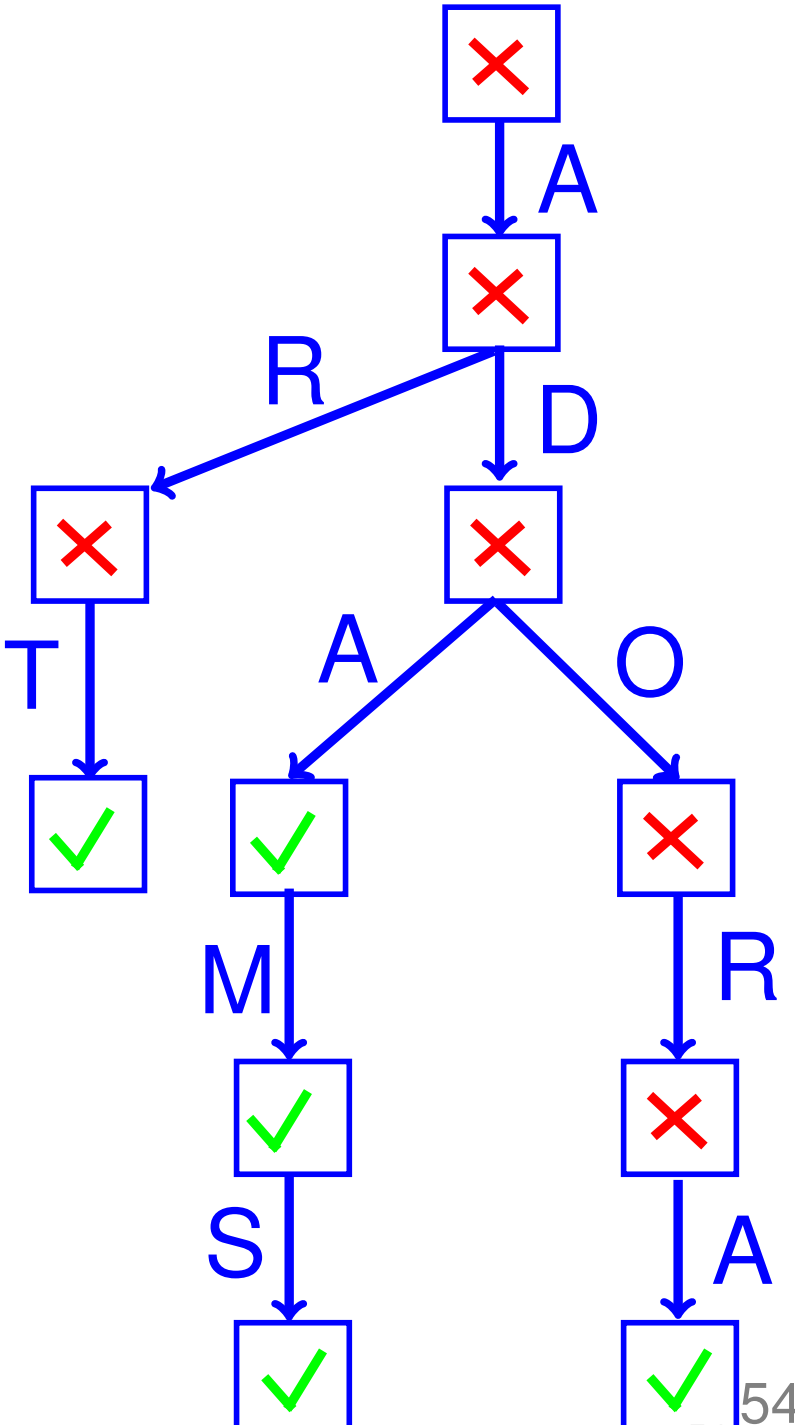
Adding strings (2)

To add another string,

make a new branch.

{ADA, ADAM,

ADAMS, ADORA, ART}

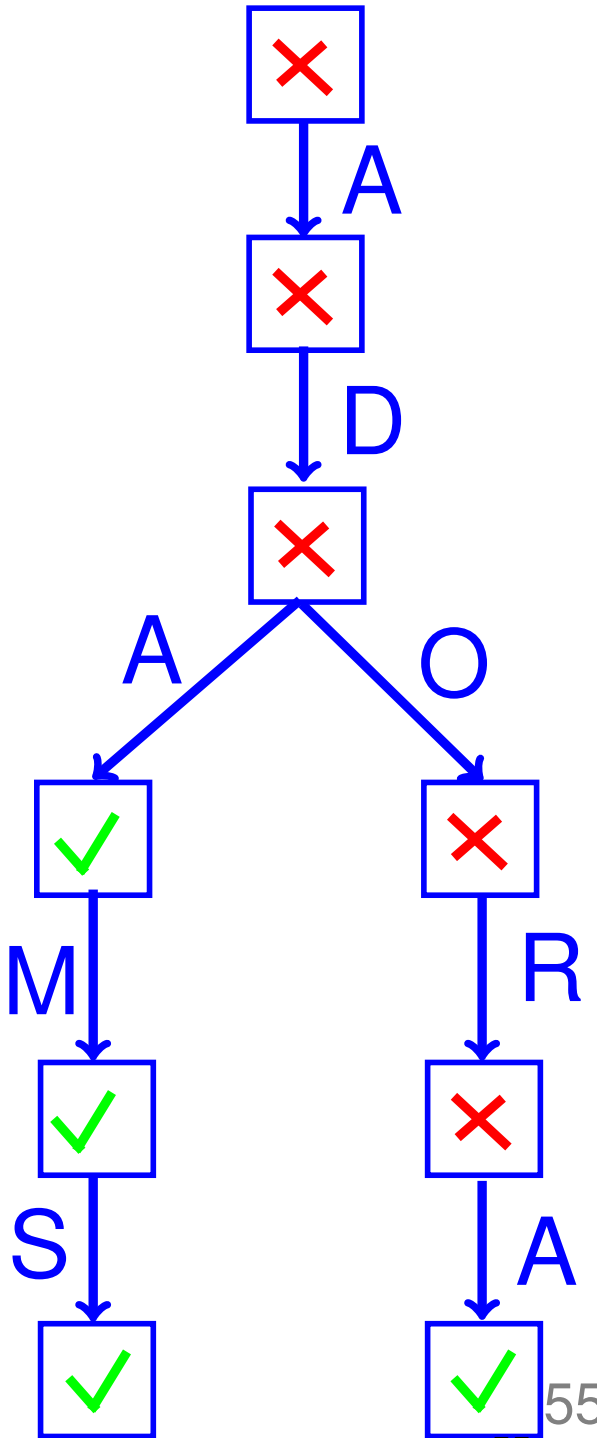


Task: Tries

Start with an empty trie.

Add

- bon
- bonbon
- on

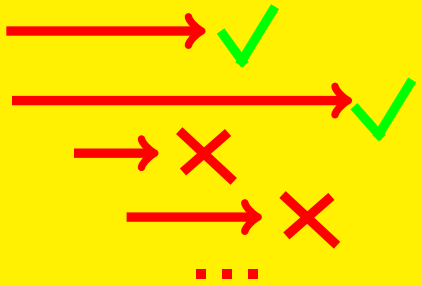


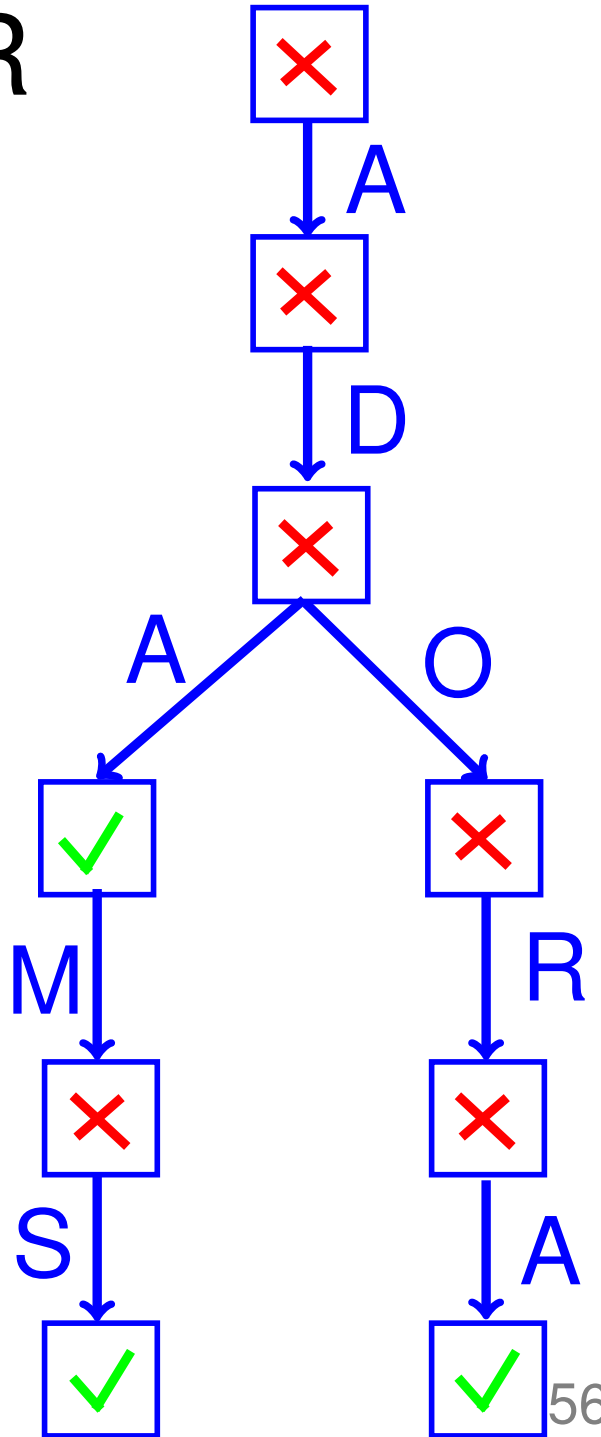
Tries can be used for NER

For every character in the doc

- advance as far as possible in the trie and
  - report match whenever you meet a 'true' node

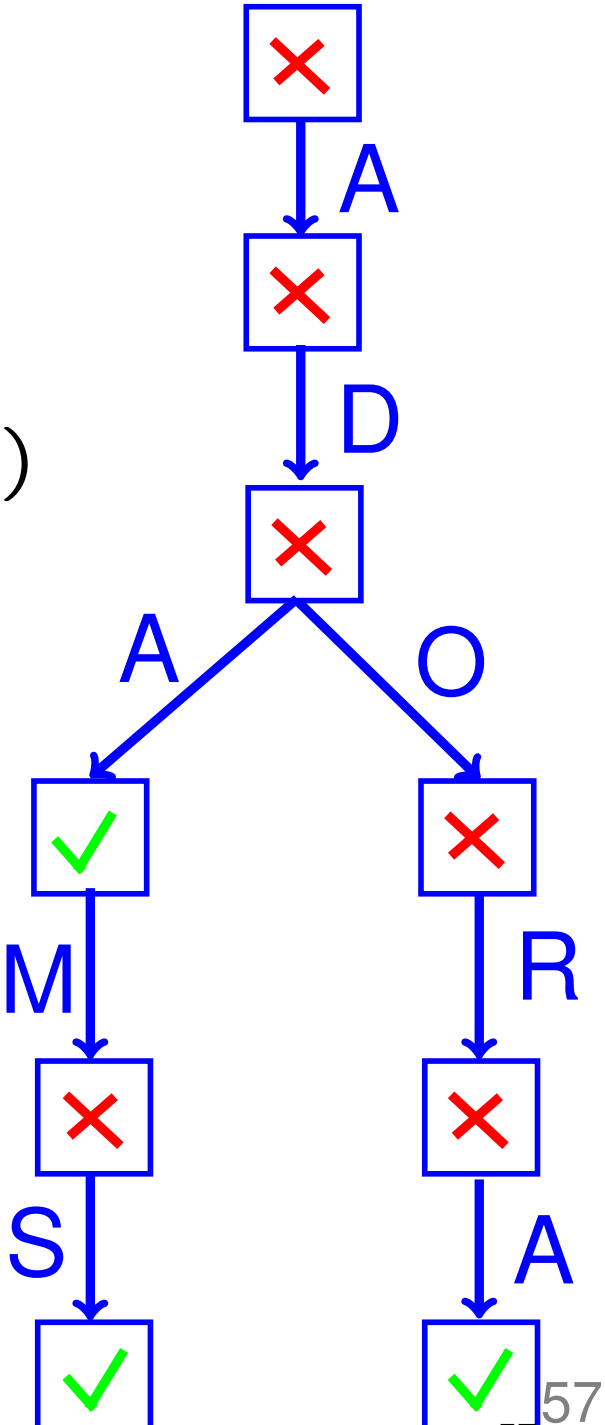
Adams adores Adora.



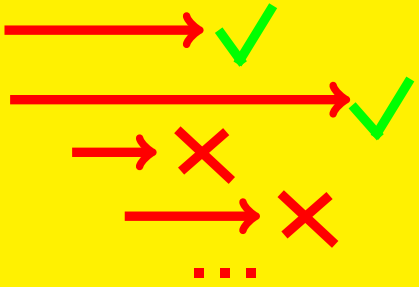


Tries have good runtime

$$O(textLength \times maxWordLength)$$



Adams adores Adora.



Task: NER with tries

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

on aime un bon bonbon.

Dictionary NER

Dictionary NER is very efficient,

but 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., phone numbers)

Overview

- Source selection
- Character Encodings
- Named Entity Recognition (NER)
  - by dictionary
  - with regexes

Dictionaries do not always work

It is unhandy/impossible to come up with exhaustive sets of all years, numbers, people, or books.

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

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 discuss how to best extract the data from Arthur's brain.

People with titles

Main street 42
West Country

Addresses

language>

Describing strings

{0000, 0001, ..., 1980, ... 9999}

Strings that
consist of 4 digits

How can we describe strings systematically?

Def: Alphabet

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.

$A=\{0,...9,a...,z,A...Z,!,?,...\}$

Def: Word

A word over an alphabet A

is a sequence of symbols from A.

Since we use the alphabet of uni-
code 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={AAA,...,USA,...,ZZZ}

L3={1900,1901,1902,...}

Set of strings we want to describe

Def: Regular expression

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

defs>61

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

Example: Regexes

Regular expressions over the set

S={a,b}

a	(single symbol)
ab	(concatenation)
aa	(concatenation again)
(a|b)	(alternation)
(abba|bab)	(concatenation & alternation)
(abba)*	(concatenation & Kleene star)

Language of a Regex

Every regular expression R comes with a language, L(R).

We say that R matches the words in L(R).

(We sometimes also say that the words in L(R) match R.)

R = a|b ⟵ regex

L(R) = {a, b} ⟵ language of the regex

We say that the regex “a|b” matches the word “a” and the word “b”.

Def: Language of a regex (1)

$L(x) = x$ if x is a symbol of the alphabet

L(a) = {a}

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

if E is a regular expression

L(a*) = {, a, aa, aaa, ...}

Def: Language of a regex (2)

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

if E and F are regular expressions

L(a|b) = {a, b}

L(a|(b)*) = {a, ,b, bb, bbb, ...}

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

if E and F are regular expressions

L(ab) = {ab}

L((a|b)c) = {ac, bc}

Task: Regex

Write a regex that matches

10 1010 101010 ...

Write a regex that matches

Deep Thought

Deeep Thought

Deeeep Thought ...

Write a regex that matches

tic tac tuc

We define shorthand regexes (1)

Let S be an ordered set of symbols

- [x-y] := (x|...|y) (for symbols x$<$y)

  [0-9] = [0|1|2|3|4|5|6|7|8|9]
  (meaning “one of”)

- (X)+ := X(X)* (for regex X)

  ([0-9])+ = [0-9]([0-9])*
  (meaning “at least one”)

We define shorthand regexes (2)

- leave away outer parantheses,
  use sequential alternation

  a|b|c = (a|(b|c))

  (we leave out the parantheses)

- . is an arbitrary symbol from S

  sh.t

  matches different symbols

  in place of the dot (e.g, “shot”).

We define shorthand regexes (3)

For integers i, j and regex X, we define

- X{i} := X{i,i} := X...X (i times)

  f{4} = ffff

  f{3,3} = fff

- X{i,j} := (X{i}|X{i+1,j})

  f{4,6} = ffff | fffff | ffffff

- X? := (|X)

  (meaning “an optional X”)

Regex examples

- A digit
- A sequence of digits
- A name

Regex examples

- A digit

  [0-9]

- A sequence of digits

  [0-9]+

- A name

  [A-Z][a-z]+

task>

Task: Regexes

X | Y	Either X or Y
X*	Zero or more X
X+	One or more X
X{i,j}	i to j times X
X?	An optional X
[a-z]	One of the symbols in range
.	An arbitrary symbol

To write C if C is one of {}[]|.?+ use C

Define regexes for

- numbers
- phone numbers
- HTML tags
- Names of the form “Dr. Blah Blub”

Try it

Named regexes

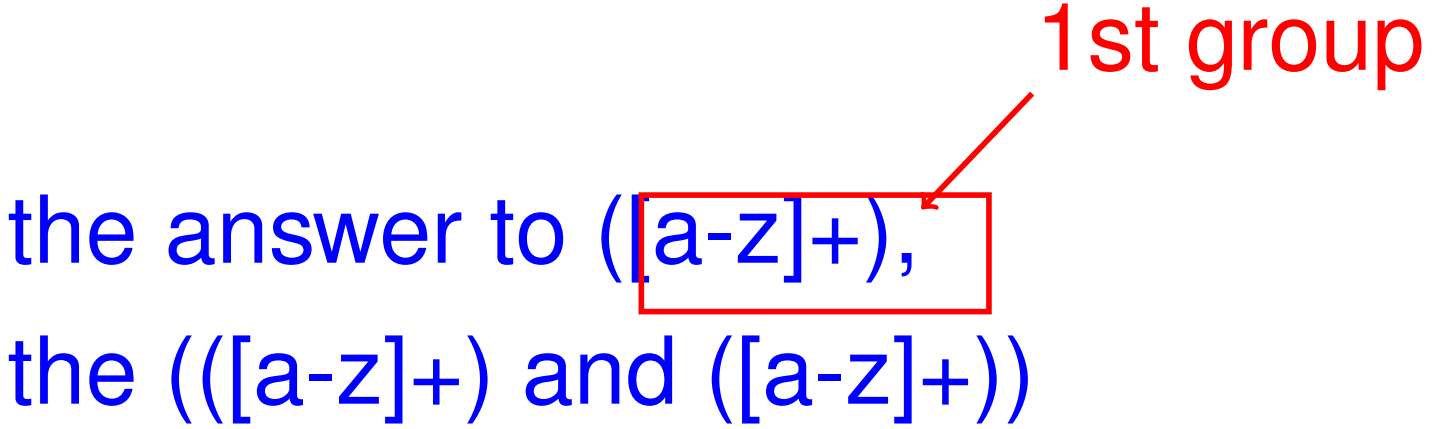
When using regular expressions in a program, it is common to name them:

```
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)

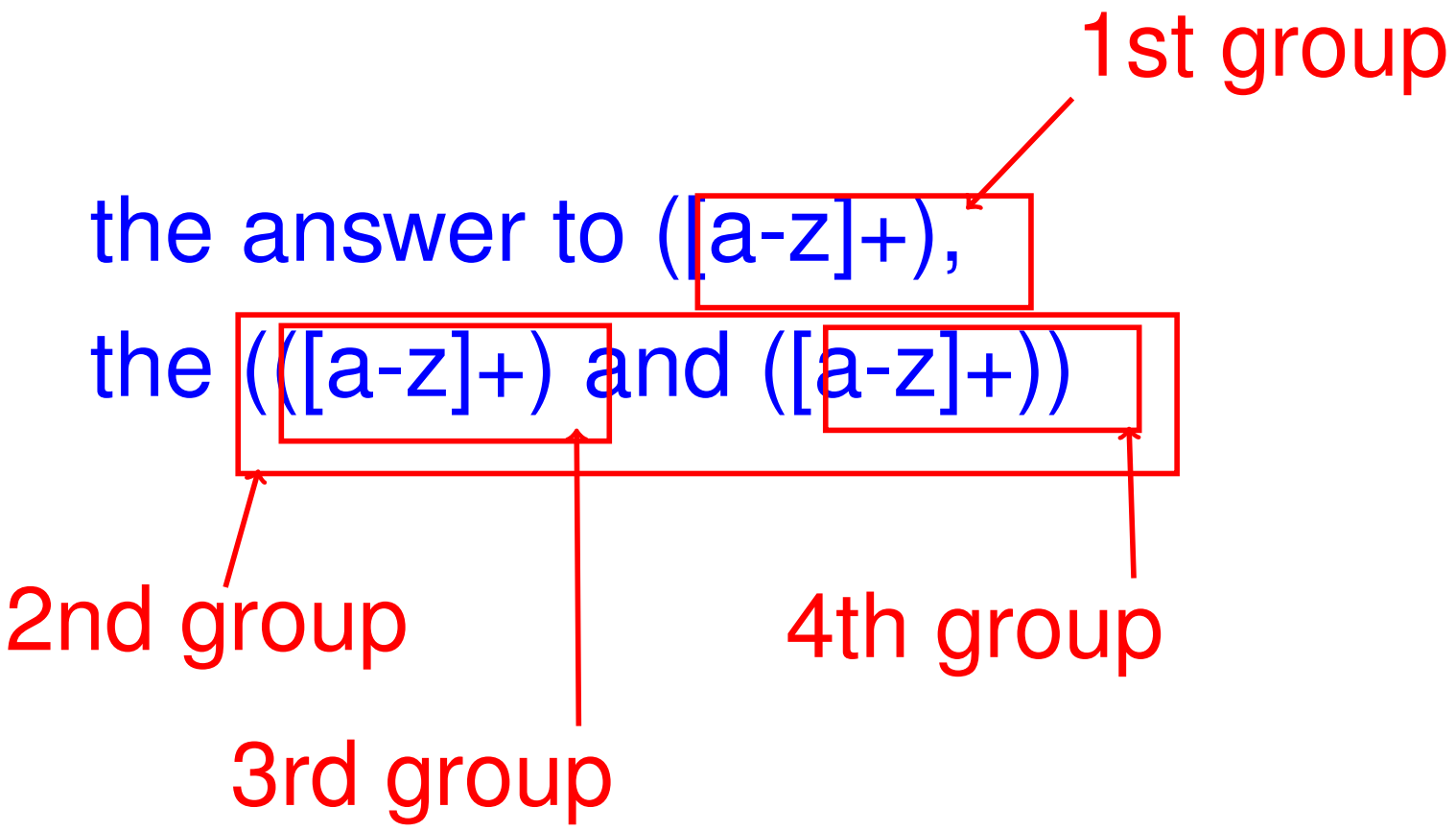
Regex groups

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.



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

fsm>

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.

Finite State Machine

A Finite state machine (FSM) is

a quintuple of

- an input alphabet A
- a finite non-empty set of states S
- an initial state s, an element of S
- a set of final states $F \subseteq S$
- a state transition relation $\delta \subseteq S \times A \times S$

Wikipedia

Def: Finite State Machine

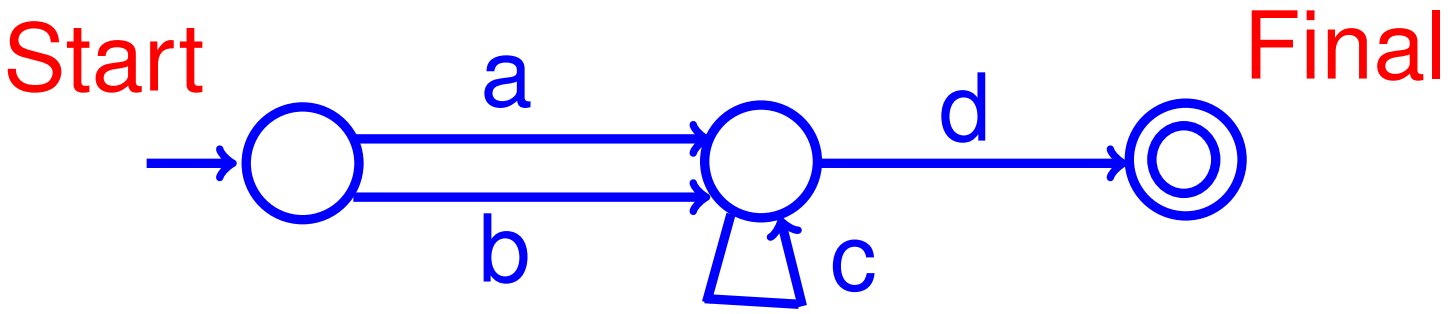
An FSM is a directed multi-graph,

where each edge is labeled with a

symbol or the empty symbol ϵ

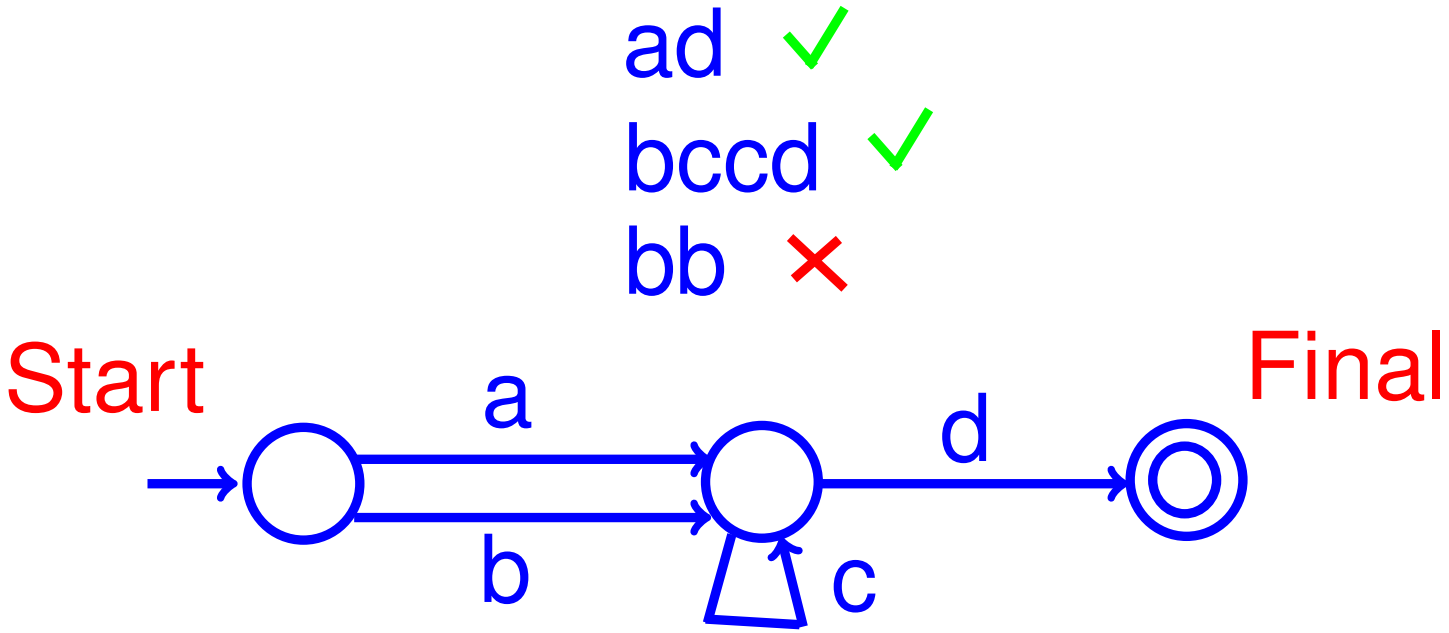
One node is labeled “start”.

Zero or more nodes are labeled “final”.



Def: Acceptance

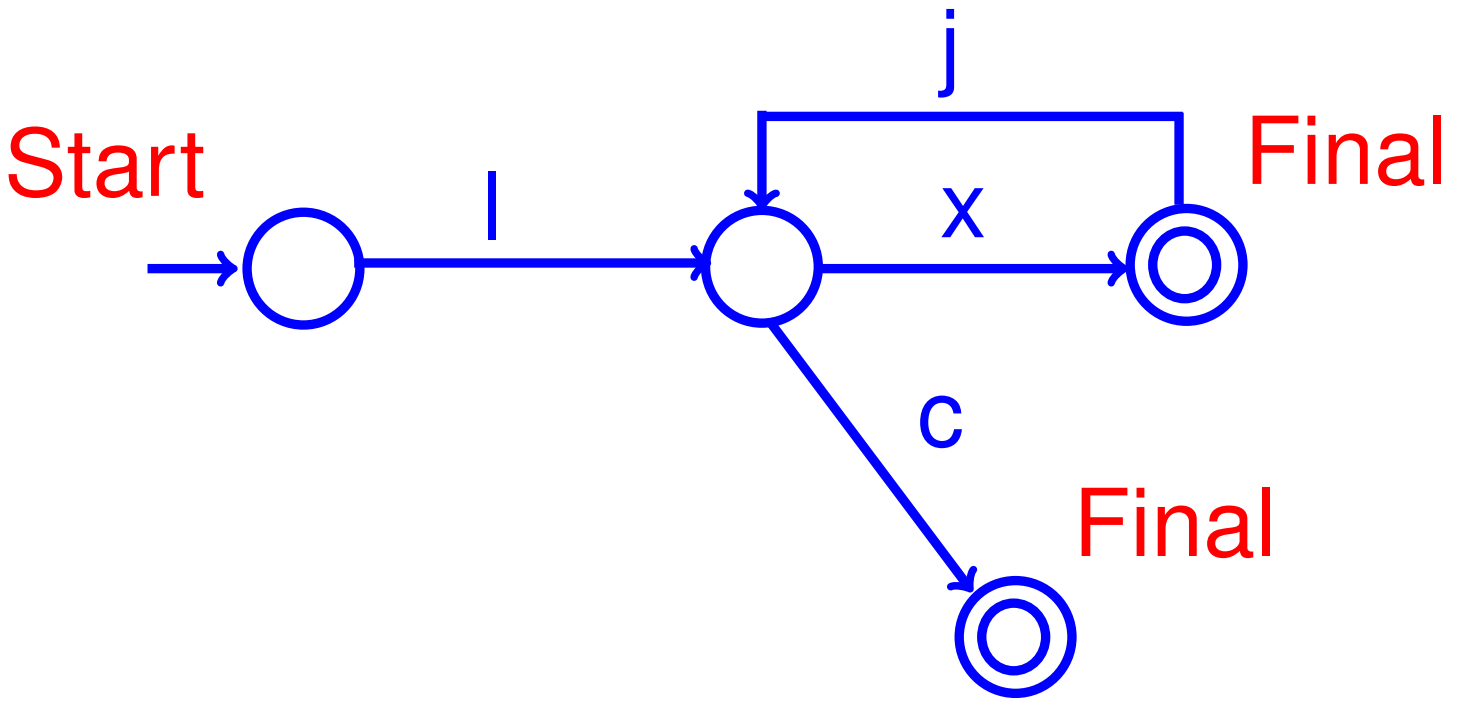
An FSM accepts (also: generates) a string, if there is a path from the start node to a final node whose edge labels are the string.



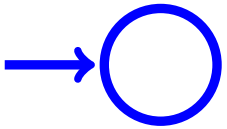
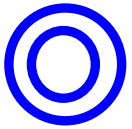
Task: FSM

Find strings generated by the following FSM:

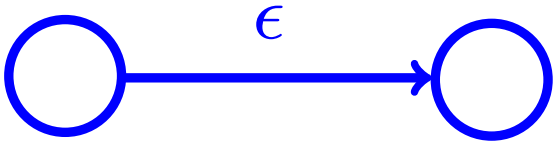
(Betelgeuse 7 language)



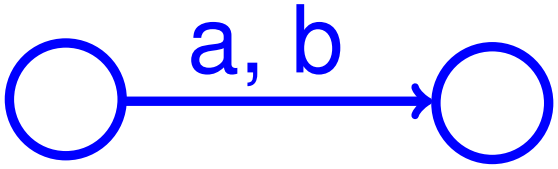
Notation

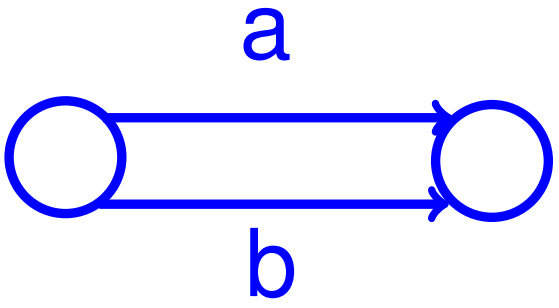
- start node 
- final node 
- empty transition

  (can be walked without

  accepting a symbol) 

- multiple edges



:= 

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

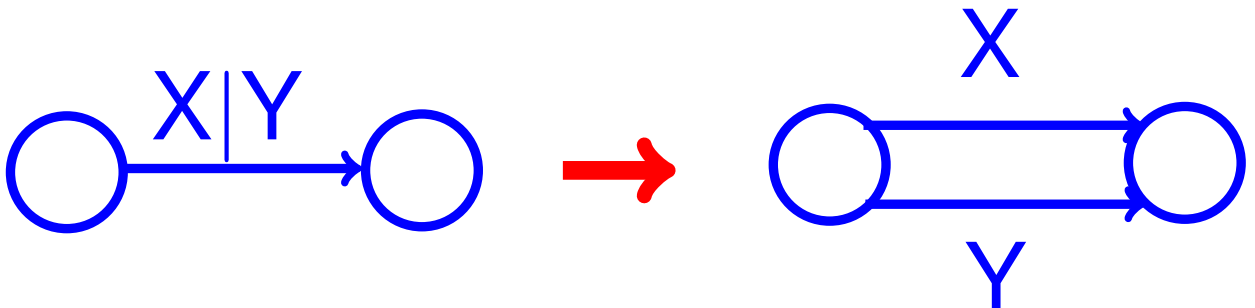
trafo>

Transforming a regex to a FSM (1)

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

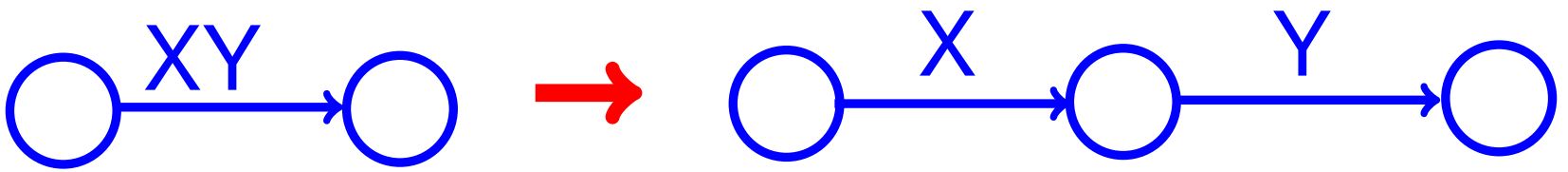
2. Start with this configuration:

3. Handle alternation:

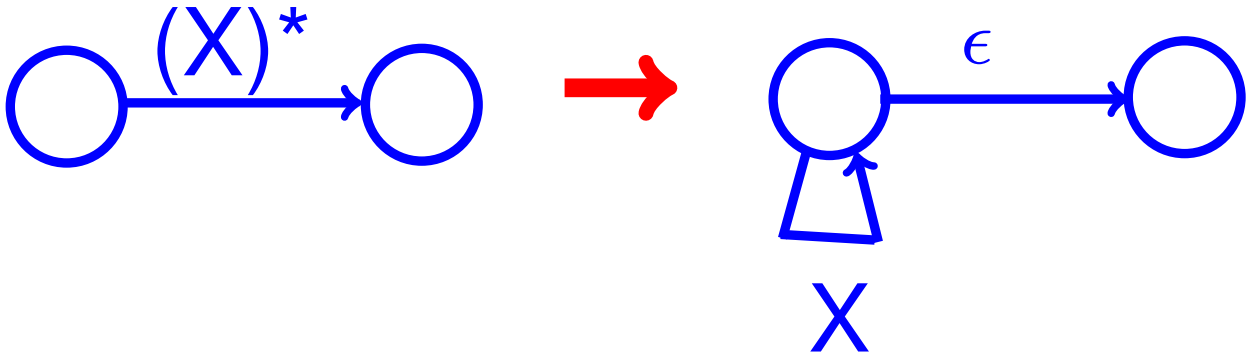


Transforming a regex to a FSM (2)

4. Handle concatenation:



5. Handle Kleene star:



Transforming a regex to a FSM (3)

6. Proceed recursively, until

   all edges are single symbols

Examples:

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

FSM = Regex

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

Every regex corresponds to an FSM

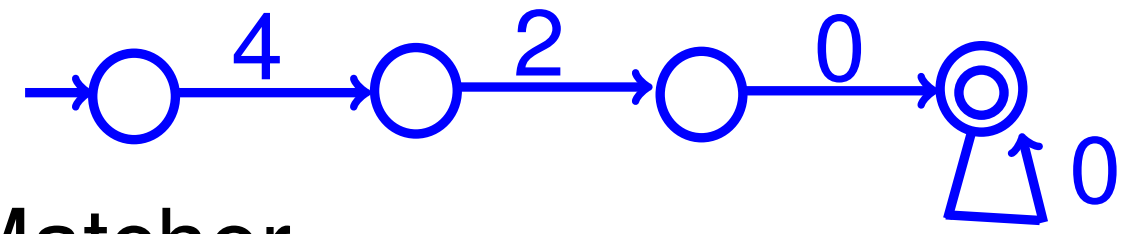
Regex

42(0)+

Simplified regex

420(0)*

FSM



Matcher

His favorite numbers are 42, 4200, and 19.

you

your pro-gramming language

Example: Regexes in Java

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

4200

His favorite numbers
are 42, 4200, and 19.

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
- $O(l \times 2^n)$ else

FSMs

There is a looooot more

to say about FSMs

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

Here, we only use them for IE

Entity Recognition

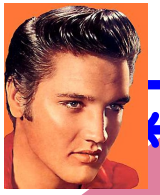
We have seen 2 methods to
do entity recognition:

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

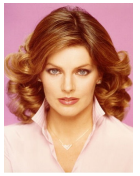
Douglas N. Adams had the idea for the
“Hitchhiker’s Guide” while lying drunk
in a field near Innsbruck.

Semantic IE

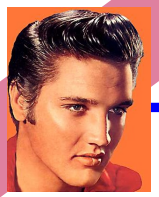
Reasoning

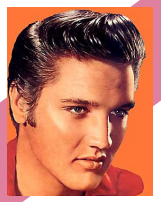



Fact Extraction

Instance Extraction


→ singer


Entity Disambiguation

You are here

singer Elvis

Entity Recognition

Source Selection and Preparation

References

Sunita Sarawagi: Information Extraction