# Introduction to Natural Language Processing

## Part VI: NLP using Regular Expressions

Henning Wachsmuth

https://ai.uni-hannover.de

# Learning Objectives

## Concepts

- Different types of formal grammars in NLP
- Sequential patterns in language
- Main operators of regular expressions

## Methods

- Identification of numeric entities with regular expressions

## Covered tasks

- Time expression recognition

# Outline of the Course


I. Overview

II. Basics of Linguistics

III. NLP using Rules

IV. NLP using Lexicons

V. Basics of Empirical Methods

VI. NLP using Regular Expressions

- Introduction
- Regular Grammars
- Regular Expressions in NLP

VII. NLP using Context-Free Grammars

VIII. NLP using Language Models

IX. Practical Issues

# Introduction

# Woodchuuucks

**HOW MUCH WOOD WOULD A WOODCHUCK CHUCK,
IF A WOODCHUCK COULD CHUCK WOOD?**


https://commons.wikimedia.org

So much wood as a woodchuck could,
if a woodchuck would chuck wood.

It would chuck, if it would, as much wood as
it could, if a woodchuck could chuck wood.

A woodchuck would chuck no amount of wood,
since woodchucks can't chuck wood.

*But if Woodchucks Could and Would Chuck Some Wood,
What Amount of Wood Would each Woodchuck Chuck?*

... and so forth

If you don't know, maybe you know the difference between
woodchucks and groundhogs?

There is none. A groundhog is a woodchuck.

# Woodchuuucks
## Mining Woodchucks from Text

### Woodchucks in the text above

- “Woodchuuucks”
- “WOODCHUCK”
- “woodchuck”
- “woodchucks”
- “Woodchucks”
- “Woodchuck”
- “groundhogs”
- “groundhog”

along with “It” and “it”

https://commons.wikimedia.org

### Questions

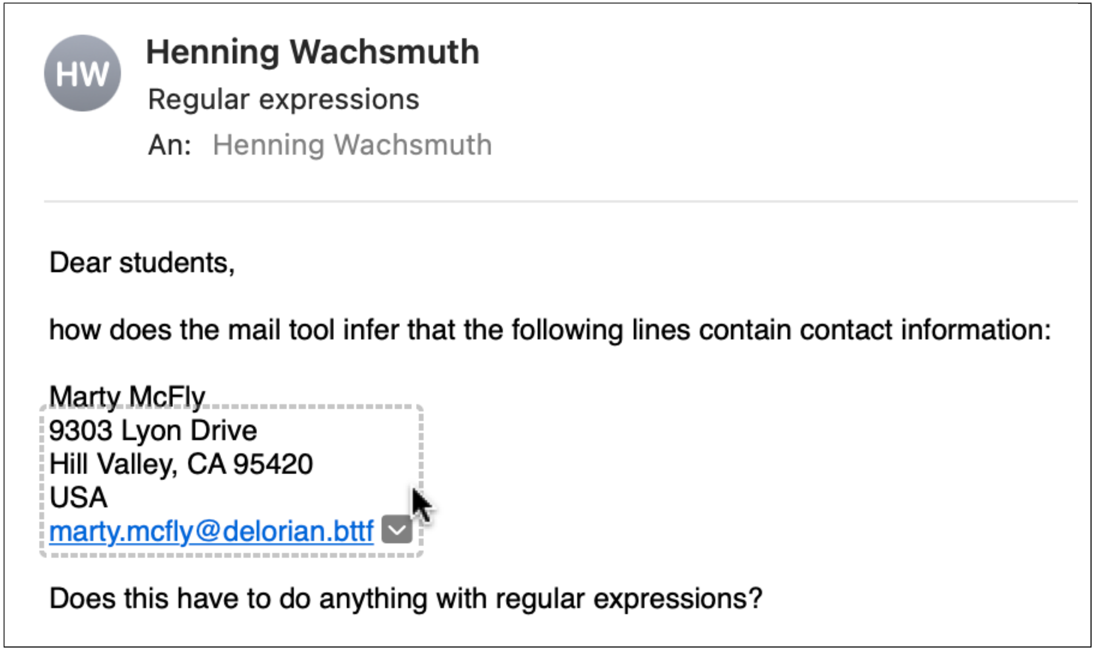
- How to find these and other references to woodchucks automatically?
- What makes woodchuck mining challenging?

  Examples: Ambiguous cases, missing whitespaces, varying writings, ...

# Contact Information

## Finding contact information

- See the following e-mail:


HW

**Henning Wachsmuth**

Regular expressions

An: Henning Wachsmuth

Dear students,

how does the mail tool infer that the following lines contain contact information:

Marty McFly
9303 Lyon Drive
Hill Valley, CA 95420
USA
marty.mcfly@delorian.bttf

Does this have to do anything with regular expressions?


https://pixabay.com

## Questions

- How can we describe patterns in text sequences formally?
- What can be described well, what not?

# NLP using Regular Expressions

## Regular expression (regex)

- A sequence of characters that describes sequential text patterns
- A text can be matched against a regex to find all pattern instances

## Use in NLP

- Effective for finding information that follows clear sequential structures
- Examples. Numeric entities, structural entities (e.g., e-mail addresses), lexico-syntactic relations (e.g., “<NN> is a <NN>”), ...

## Numeric (and alphanumeric) entities

- Values, quantities, proportions, ranges, or similar
- This includes time periods, dates, monetary values, phone numbers, ...

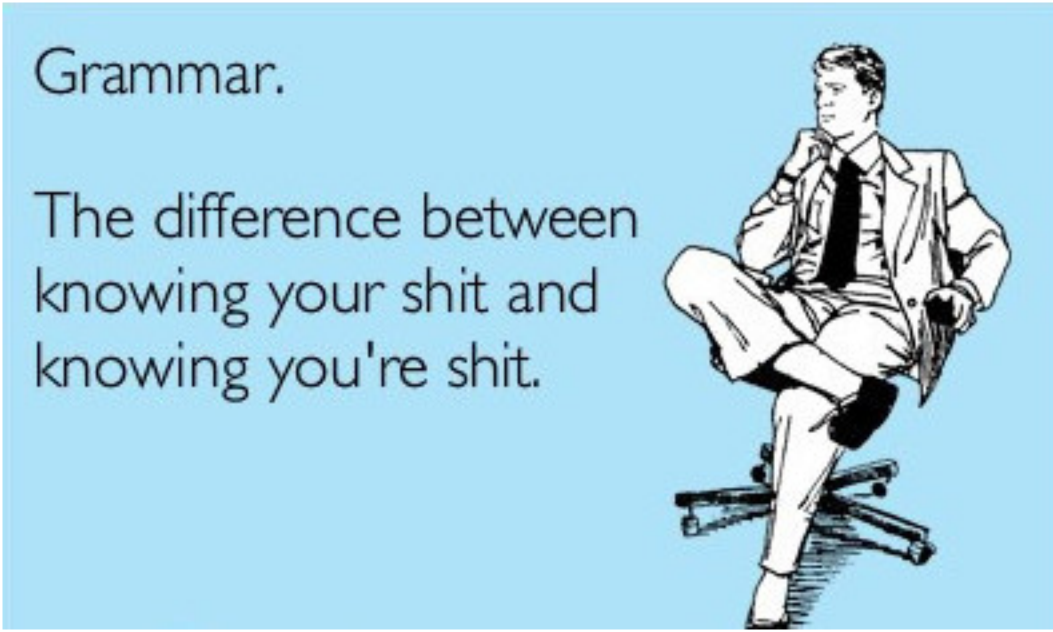
“in this year” “2024-06-13” “$ 100 000” “762-123 77”

## Numeric entity recognition

- The text analysis that mines numeric entities from text
- Used in NLP within many information extraction tasks

# Regular Grammars

# Grammar




https://someecards.com

# Grammar

## Grammars

### Grammars

- A grammar is a description of the valid structures of a language.
- One of the most central concepts of linguistics is *formal grammars*.

### Formal grammars

- A formal grammar specifies a set of rules consisting of terminal and non-terminal symbols.
- Terminals. "Words" that cannot be rewritten any further
- Non-terminals. Clusters or generalizations of terminals

### Grammar $(\Sigma, N, S, R)$

$\Sigma$ An alphabet, i.e., a finite set of terminal symbols, $\Sigma = \{v_1, v_2, \ldots\}$

$N$ A finite set of non-terminal symbols, $N = \{W_1, W_2, \ldots\}$

$S$ A start non-terminal symbol, $S \in N$

$R$ A finite set of production rules, $R \subseteq (\Sigma \cup N)^+ \setminus \Sigma^* \times (\Sigma \cup N)^*$

Here, $\Sigma^* := \Sigma \times \Sigma \times \ldots$

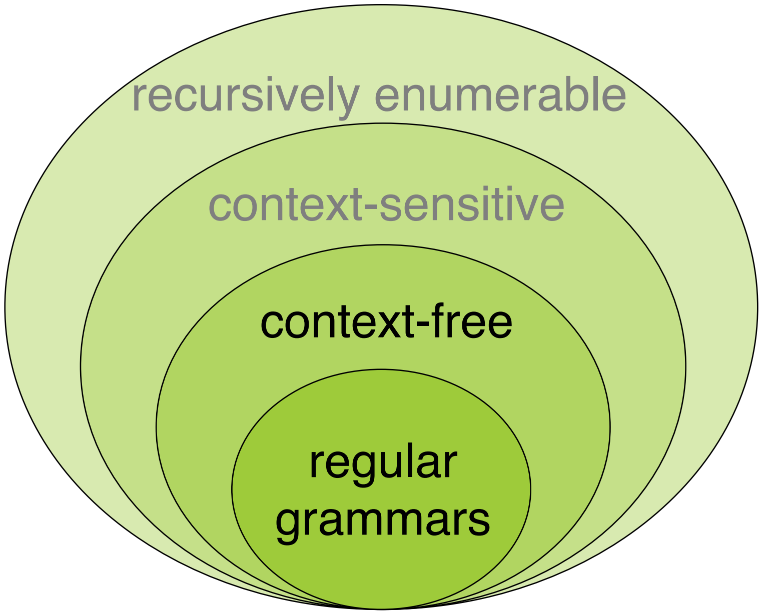
# Grammar

## Chomsky Grammars

### Four types of formal grammars

- Chomsky-0 (recursively enumerable). Any $(\Sigma, N, S, R)$ as defined
- Chomsky-1 (context-sensitive). Only rules $U \rightarrow V$ with $|U| \leq |V|$
- Chomsky-2 (context-free). Only rules $U \rightarrow V$ with $U \in N$
- Chomsky-3 (regular). Only rules $U \rightarrow V$ with $U \in N$ and $V \in \{\varepsilon, v, vW\}$, $v \in \Sigma$, $W \in N$

  $\varepsilon$ is the empty word.




### Grammars in NLP

- Only regular and context-free grammars are commonly used.
- Regular grammars are covered here, context-free grammars in the next part.

# Regular Grammars

## Regular grammar

- A grammar $(\Sigma, N, S, R)$ where all rules in $R$ are of the form $U \to V$ with $U \in N$ and $V \in \{\varepsilon, v, vW\}$, such that $v \in \Sigma$ and $W \in N$
- Extended. A regular grammar is *extended*, if $v \in \Sigma^*$
  Below, we refer to this also as *regular grammar* only.
- Right-regular. Intuitively, a structure defined by a regular grammar can be constructed from left to right.

## Regular language

- A language is regular, if there is a regular grammar that defines it.

## Representation

- Every regular grammar can be represented by a *finite-state automaton*.
- Every regular grammar can be represented by a *regular expression*.
  And vice versa

# Regular Grammars

## Finite-State Automata

### Finite-state automaton (FSA) (recap)

- An FSA is a state machine that reads a string from a regular language.
- It represents the set of all strings belonging to the language.

### FSA as a 5-tuple $(Q, \Sigma, q_0, F, \delta)$

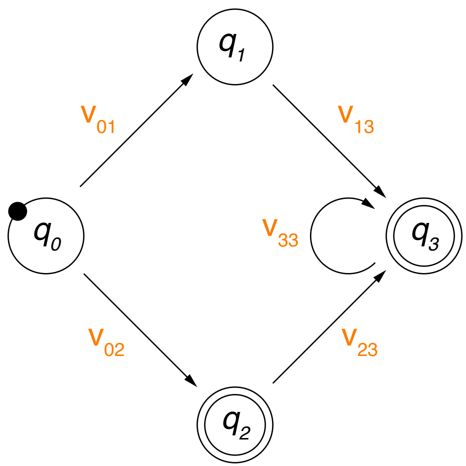
$Q$ A finite set of $n > 0$ states, $Q = \{q_0, ..., q_n\}$

$\Sigma$ An alphabet, i.e., a finite set of terminal symbols, $\Sigma = \{v_1, v_2, \ldots\}$, $\Sigma \cap Q = \emptyset$

$q_0$ A start state, $q_0 \in Q$

$F$ A set of final states, $F \subseteq Q$

$\delta$ A transition function between states, triggered based on $v \in \Sigma$, $\delta : Q \times \Sigma \to Q$



# Regular Expressions in NLP

# Regular Expressions

## Regular expression (regex)

- A regex defines a regular language over an alphabet $\Sigma$ as a sequence of characters from $\Sigma$ and *metacharacters*
- Metacharacters denote disjunction, negation, repetition, ... (see below)
- Example. The FSA above is defined by the following regex:

$$v_{02} \;\; | \;\; (v_{01}v_{13} \,|\, v_{02}v_{23})\; v_{33}{}^{*}$$

## Use of regular expressions

- Definition of patterns that generalize over structures of a language
- A pattern matches all spans of text that contain any of the structures.

## Regular expressions in NLP

- Regexes are a widely used technique in NLP, particularly for the extraction of numeric and similar entities.
- In statistical NLP, regexes often take on the role of features.

# Regular Expressions

## Characters and Metacharacters

### Regular characters

- The default interpretation of a character sequence in a regex is a concatenation of each single character.

`woodchuck` matches "woodchuck"

### Metacharacters

- A regex uses specific characters to encode specific regular language constructions, such as negation and repetition.
- The main metacharacters are the following (in Python notation):

`[]   -   |   ^   .   ( )   \   *   +   ?`

The characters used partly differ across literature and programming languages.

- Some languages also include certain *non-regular* constructions, e.g., `\b` matches if a word boundary is reached.

Regexes can solve this case only if given token information.

# Regular Expressions

## Disjunction

### Disjunction of patterns

- Brackets `[]` specify a character class.

  `[wW]` matches "w" or "W"

  `[wod]` matches "w" or "o" or "d"

- Disjunctive ranges of characters can be specified with a hyphen `-`.

  `[a-zA-Z]` matches any letter

  `[0-8]` matches any digit except for "9"

- The pipe `|` specifies a disjunction of string sequences.

  `groundhog|woodchuck` matches "groundhog" and "woodchuck"

### Notes on disjunctions

- Different disjunctions can be combined.

  `[gG]roundhog|[wW]oodchuck` matches "groundhog", "Woodchuck", ...

- In Python, many metacharacters are not active within brackets.

  `[wod.]` matches "w", "o", "d", and "."

# Regular Expressions

## Negation, Choice, Grouping

### Negation

- The caret `^` inside brackets complements the specified character class.

`[^0-9]` matches anything but digits

`[^wo]` matches any character but “w”, “o”

### Free choice

- The period `.` matches any character.

  To match a period, it needs to be escaped as: `\.`

`w..dchuck` matches “woodchuck”, “woudchuck”, ...

### Grouping

- Parentheses `()` can be used to group parts of a regex. A grouped part is treated as a single character.

`w(oo|uo|ou)dchuck` matches “woodchuck”, “wuodchuck”, and “woudchuck”

# Regular Expressions

## Whitespaces and Predefined Character Classes

### Whitespaces

- Different whitespaces are referred to with different special characters.
- For instance, `\n` is the regular new-line space.

### Predefined character classes

- Several specific character classes are referred to by a backslash \ followed by a specific letter.

| | | |
|---|---|---|
| `\d` | Any decimal digit | equivalent to `[0-9]` |
| `\D` | Any non-digit character | equivalent to `[^0-9]` |
| `\s` | Any whitespace character | equivalent to `[\t\n\r\f\v]` |
| `\S` | Any non-whitespace character | equivalent to `[^\t\n\r\f\v]` |
| `\w` | Any alphanumeric character | equivalent to `[a-zA-Z0-9]` |
| `\W` | Any non-alphanumeric character | equivalent to `[^a-zA-Z0-9]` |

- These classes can be used within brackets.

`[\s0-9]` matches any space and digit.

# Regular Expressions

## Repetition

### Repetition

- The asterisk `*` repeats the previous character zero or more times.

  `woo*dchuck` matches "wodchuck", "woodchuck", "wooodchuck", ...

- The plus `+` repeats the previous character one or more times.

  `woodchu+ck` matches "woodchuck", "woodchuuck","woodchuuuck", ...

- The question mark `?` repeats the previous character zero or one time.

  `woodchucks?` matches "woodchuck" and "woodchucks"

### Notes on repetitions

- Repetitions are implemented in a greedy manner in many programming languages, i.e., longer matches are preferred over shorter ones.

  `to*` matches "too", not "too", ...

- This may actually violate the regularity of the defined language.

  "woodchuck" needs to be processed twice for the regex `wo*odchuck`

# Regular Expressions

## Summary of Metacharacters

| Char | Concept | Example |
|---|---|---|
| `[ ]` | Disjunction of characters | `[Ww]oodchuck` |
| `-` | Ranges in disjunctions | `There are [0-9]+ woodchucks\.` |
| `\|` | Disjunction of regexes | `woodchuck\|groundhog` |
| `^` | Negation | `[^0-9]` |
| `.` | Free choice | `What a (.)* woodchuck` |
| `( )` | Grouping of regex parts | `w(oo)+dchuck` |
| `\` | Special (sets of) characters | `\swoodchuck\s` |
| `*` | Zero or more repetitions | `wooo*dchuck` |
| `+` | One or more repetitions | `woodchu+ck` |
| `?` | Zero or one repetition | `woodchucks?` |

# Regular Expressions
## Examples

### The

- Regex for all variations of “the” in news article text:

`the` (misses capitalized cases, matches “theology”, ...)

`[^a-zA-Z][tT]he[^a-zA-Z]` (requires a character before and afterwards)

### Woodchucks

- Regex for all woodchuck cases from above (and for similar):

```
( [wW][oO][oO][dD][cC][hH][uU]+[cC][kK] | [Gg]roundhog ) [sS]?
```

### E-mail addresses

- All e-mail addresses with the German top-level domain, whose text segments contain no special characters:

```
[a-zA-Z0-9]+ @ ([a-zA-Z0-9]+\.)* [a-zA-Z0-9][a-zA-Z0-9]+ \.de
```

# Time Expression Recognition with Regular Expressions

## Time expression

- An alphanumeric entity that represents a date or a period

> "Cairo, August 25th 2010 — Forecast on Egyptian Automobile industry
> [...] In the next five years, revenues will rise by 97% to US-$ 19.6 bn. [...]"

## Time expression recognition

- The text analysis that finds time expressions in natural language text
- Used in NLP within temporal relation extraction and event extraction

## Approach in a nutshell

- Model sequential structure of time expressions with a complex regex.
- Include lexicons derived from training data to match closed-class terms.
  Example closed classes: Month names, prepositions, ...
- Match regex with sentences of a text.

## Notice

- The approach can easily be adapted to other types of information.

# Time Expression Recognition with Regular Expressions

## Pseudocode

### Signature

- Input. A text split into sentences, and a regex
- Output. All time expressions in the text

**extractAllMatches(List<Sentence> sentences, Regex regex)**

```
List<TimeExpression> matches ← ()
for each sentence ∈ sentences do
    int index ← 0
    while index < sentence.length - 1 do
        int[] exp ← regex.match(sentence.sub(index))
        if exp ≠ ⊥ then   // ⊥ represents "null"
            matches.add(new TimeExpression(exp[0], exp[1]))
            index ← exp[1]
        index ← index + 1
return matches
```

### Notice

- Most programming languages provide explicit matching classes.

# Time Expression Recognition with Regular Expressions

## Complete Regex (Part 1 out of 2)

```
((((([iI]n|[wW]ithin|[tT]o\s\s?the|[tT]o|[fF]or\s\s?the|[fF]or|[fF]rom|[sS]ince|[aA]fter|[bB]efore|[bB]etween|[aA]t|[oO]n|[
oO]ver|[pP]er)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))
\s*)?(((0?[123456789]|[12]\d|3[01])(\.|/))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)(((month|time(span)?(\s+(\r(\n)?|\n)?|(\r(\n)
?|\n))\s*(from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\.|[Ff]eb|[Mm]arch|[Mm]ar\
.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust|[Aa]ug\.|[Aa]ug|[Ss]epte
mber|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[Dd]ez|[Ss]pring|[Ss]ummer
|[Aa]utumn|[Ff]all|[Ww]inter))|((0?[123456789]|1[012])(\.|/)))(()?((19|20)?\d2))?)|((((([iI]n|[wW]ithin|[tT]o\s\s?the|[tT]o
|[fF]or\s\s?the|[fF]or|[fF]rom|[sS]ince|[aA]fter|[bB]efore|[bB]etween|[aA]t|[oO]n|[oO]ver|[pP]er)((\s+(\r(\n)?|\n)?|(\r(\n)
?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((([sS]tart|[bB]egin|[Ss]tart|[Bb]egi
n|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)
?|\n)?|(\r(\n)?|\n))\s*)?|(([sS]tart|[bB]egin|[Ss]tart|[Bb]egin|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s+(\r(\n)?|\n)?|(\r(\n)?|
\n))\s*([tT]he|[tT]his|[tT]hese|[tT]hose|[iI]ts))?)(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)([[a-z]]+(\s+(\r(\n)?|\n)?|(\r(\n)?|\n)
)\s*)?)?(((((([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext))((\s+(\r(\n)?
|\n)?|(\r(\n)?|\n))\s*(((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[s
S]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]
ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]
ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[
tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\
s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]i
ne|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]u
ndred|[aA]\s\s?hundred)))?))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)|((((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]ot
h|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty
|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|
([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|
3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]l
eventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]
our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS
]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(([lL]ast|[pP]recedi
ng|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext)))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*))?(((Q(1
|2|3|4)|H(1|2)(\/(19|20)?\d2)?|(((\w([a-z])*(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(year|quarter))([a-z])*))|((month|time(span)
?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\
.|[Ff]eb|[Mm]arch|[Mm]ar\.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust
|[Aa]ug\.|[Aa]ug|[Ss]eptember|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[
Dd]ez|[Ss]pring|[Ss]ummer|[Aa]utumn|[Ff]all|[Ww]inter))|(([Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time\s\s?span|[Rr]e
ported\s\s?time|[rR]eported\s\s?time|[Tt]ime\s\s?span|[tT]ime\s\s?span|[Ss]pan|[sS]pan|[Dd]ecade|[dD]ecade)))((\s+(\r(\n)?|\
n)?|(\r(\n)?|\n))\s*((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))?|((19|20)\d2(/(19|20)?\d2)?|\d2/\d2)))|((((([lL]ast|[pP]receding|[p
P]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(((1|2|3|4|5|6|
7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]le
ven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?h
undred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[n
N]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[
sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]
everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]went
y|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|
```

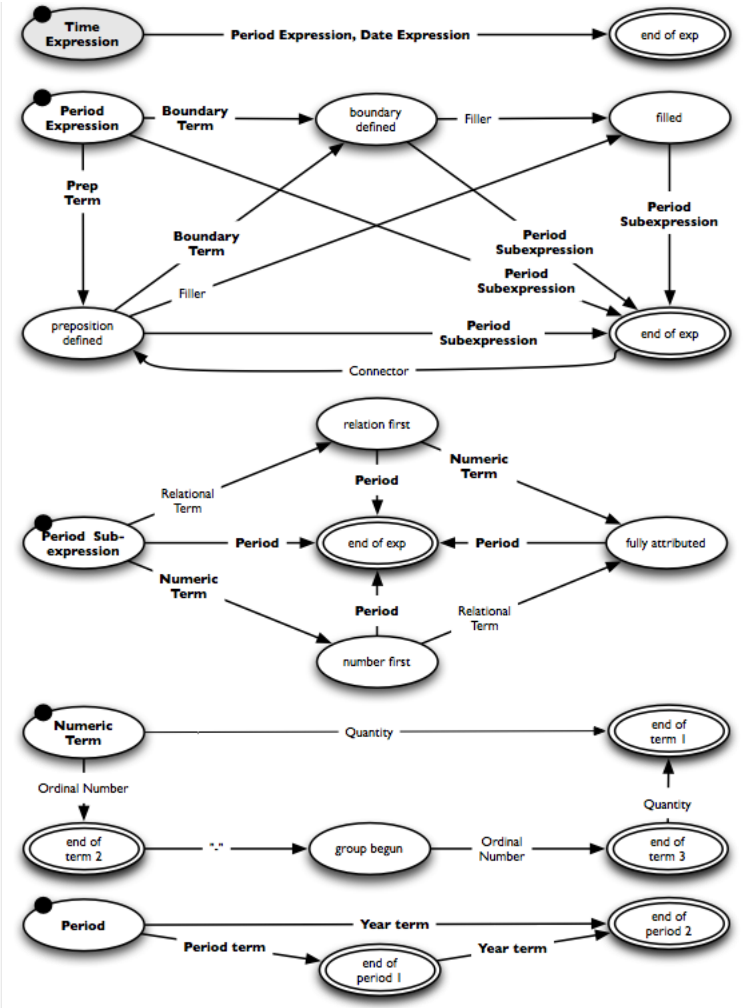
# Time Expression Recognition with Regular Expressions

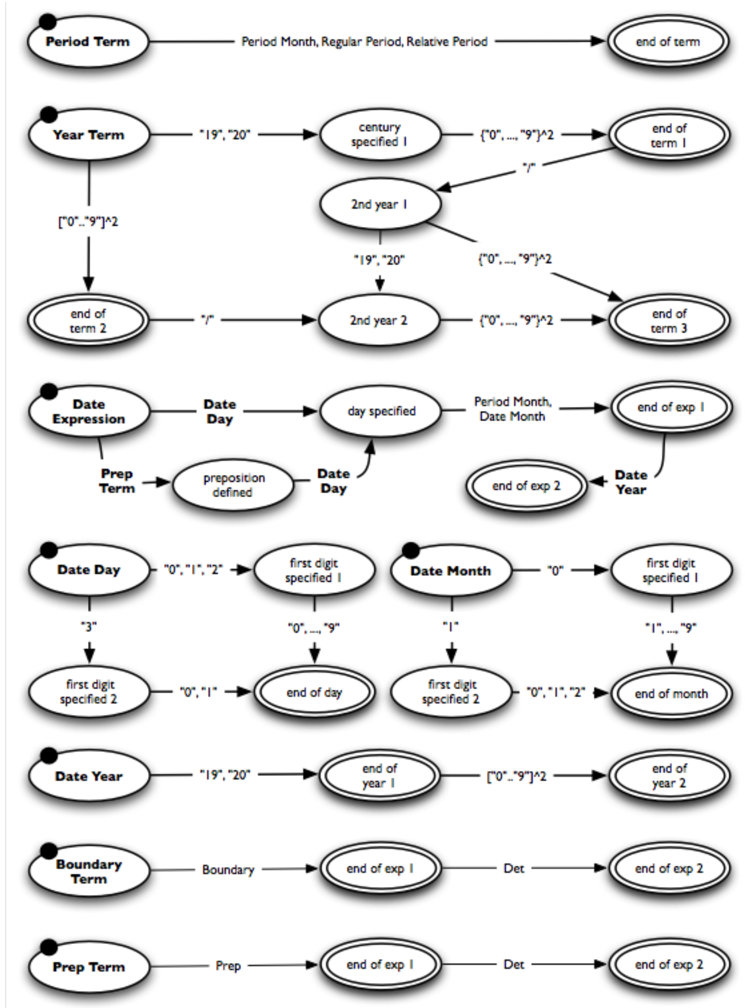
## Complete Regex (Part 2 out of 2)

```
[aA]\s\s?hundred)))?))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)|((((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]
wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]o
rty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]i
rst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|
6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh
)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[f
F]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|
[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(([lL]ast|[pP]receding|[pP
]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext)))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*))?(((Q(1|2|3|4
)|H(1|2)(\/(19|20)?\d2)?|(((\w([a-z])*(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(year|quarter))([a-z])*))|((month|time(span)?(\s+(
\r(\n)?|\n)?|(\r(\n)?|\n))\s*(from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\.|[Ff]
eb|[Mm]arch|[Mm]ar\.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust|[Aa]u
g\.|[Aa]ug|[Ss]eptember|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[Dd]ez|
[Ss]pring|[Ss]ummer|[Aa]utumn|[Ff]all|[Ww]inter))|(([Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time\s\s?span|[Rr]eported\
s\s?time|[rR]eported\s\s?time|[Tt]ime\s\s?span|[tT]ime\s\s?span|[Ss]pan|[sS]pan|[Dd]ecade|[dD]ecade)))((\s+(\r(\n)?|\n)?|(\r
(\n)?|\n))\s*((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))?|((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(
[tT]o|[aA]nd|[oO]r|[oO]n|[aA]t|[oO]f\s\s?the|[oO]f|[tT]he|[tT]his|[iI]ts|[iI]nstead\s\s?of)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\
s*(([sS]tart|[bB]egin|[Ss]tart|[Bb]egin|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|
[tT]hese|[tT]hose|[iI]ts))?))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*[[a-z]]+)?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((((([lL]ast|[p
P]receding|[pP]ast|[cC]urrent|[tT]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((
(1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|
[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undr
ed|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh
|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]if
th|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?
|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]w
elve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))
?))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)|((((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our
|[fF]ive|[sS]ix|[sS]even|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ix
ty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|[aA]\s\s?hundred))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]h
ird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh))(-((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([f
F]irst|[sS]econd|[tT]hird|[fF]ourth|[fF]ifth|[sS]ixth|[sS]eventh|[eE]ighth|[nN]inth|[tT]enth|[eE]leventh)))?((\s+(\r(\n)?|\
n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[sS]ome|[bB]oth|[tT]wo|[tT]hree|[fF]our|[fF]ive|[sS]ix|[sS]e
ven|[eE]ight|[nN]ine|[tT]en|[eE]leven|[tT]welve|[tT]wenty|[tT]hirty|[fF]ourty|[fF]orty|[fF]ifty|[sS]ixty|[sS]eventy|[eE]igh
ty|[nN]inety|[hH]undred|[aA]\s\s?hundred)))?)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*(([lL]ast|[pP]receding|[pP]ast|[cC]urrent|[t
T]his|[uU]pcoming|[fF]ollowing|[sS]ucceeding|[nN]ext)))?(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*))?(((Q(1|2|3|4)|H(1|2)(\/(19|20)?
\d2)?|(((\w([a-z])*(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?(year|quarter))([a-z])*))|((month|time(span)?(\s+(\r(\n)?|\n)?|(\r(\n
)?|\n))\s*(from(\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*)?)?([Jj]anuary|[Jj]an\.|[Jj]an|[Ff]ebruary|[Ff]eb\.|[Ff]eb|[Mm]arch|[Mm]ar
\.|[Mm]ar|[Aa]pril|[Aa]pr\.|[Aa]pr|[Mm]ay|[Jj]une|[Jj]un\.|[Jj]un|[Jj]uly|[Jj]ul\.|[Jj]ul|[Aa]ugust|[Aa]ug\.|[Aa]ug|[Ss]ept
ember|[Ss]ep\.|[Ss]ep|[Oo]ctober|[Oo]ct\.|[Oo]ct|[Nn]ovember|[Nn]ov\.|[Nn]ov|[Dd]ecember|[Dd]ez\.|[Dd]ez|[Ss]pring|[Ss]umme
r|[Aa]utumn|[Ff]all|[Ww]inter))|(([Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time\s\s?span|[Rr]eported\s\s?time|[rR]eport
ed\s\s?time|[Tt]ime\s\s?span|[tT]ime\s\s?span|[Ss]pan|[sS]pan|[Dd]ecade|[dD]ecade)))((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((19|
20)\d2(/(19|20)?\d2)?|\d2/\d2))?|((19|20)\d2(/(19|20)?\d2)?|\d2/\d2))))*))
```

# Time Expression Recognition with Regular Expressions

## Complete Regex as a Finite-State Automaton



# Time Expression Recognition with Regular Expressions

## Top-level FSA of Complete Regex


### Notice

- Bold edge labels indicate sub-FSAs, regular labels indicate lexicons.
- Below, the FSA of period expressions is decomposed top-down.
  The regex for date expressions is left out for brevity.
- During development, building a regex rather works bottom-up.

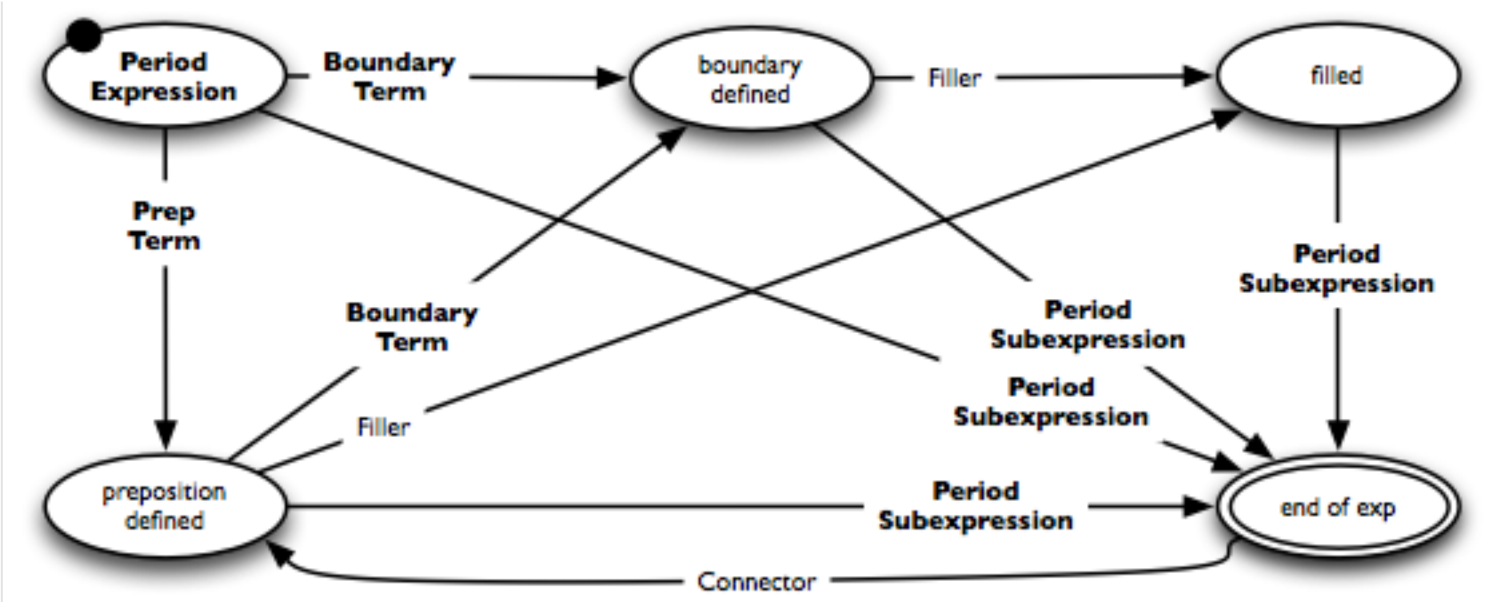
### Example period expression

- “From the very end of last year to the 2nd half of 2019”

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| “From the | very | end of | last | year | to the | 2nd | half of | 2019” |
| prep | filler | boundary | relational | period | connector | ordinal | period | year |

# Time Expression Recognition with Regular Expressions

## Sub-FSA for Period Expressions

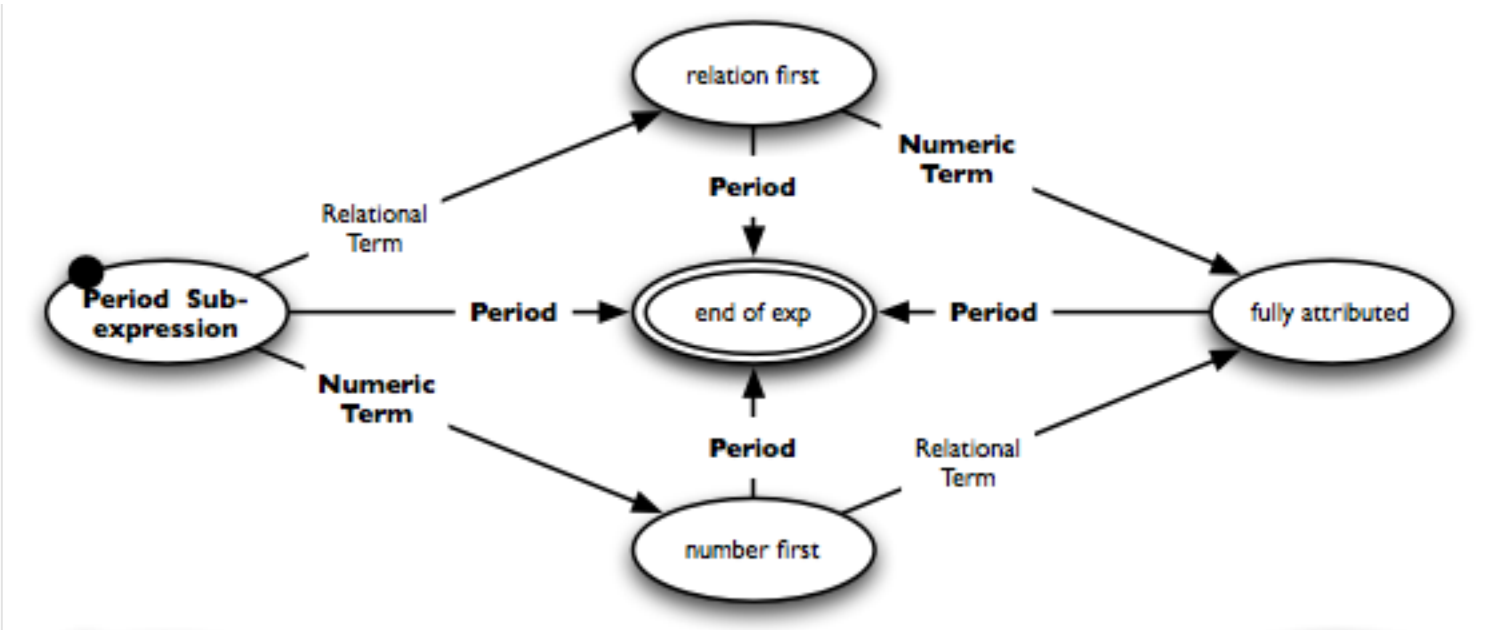



### Lexicons

- Connector lexicon. “to the”, “to”, “and”, “of the”, “of”, ...
- Fillers. *Any* single word, such as “very” in the example above

# Time Expression Recognition with Regular Expressions

## Sub-FSA for Period Subexpressions

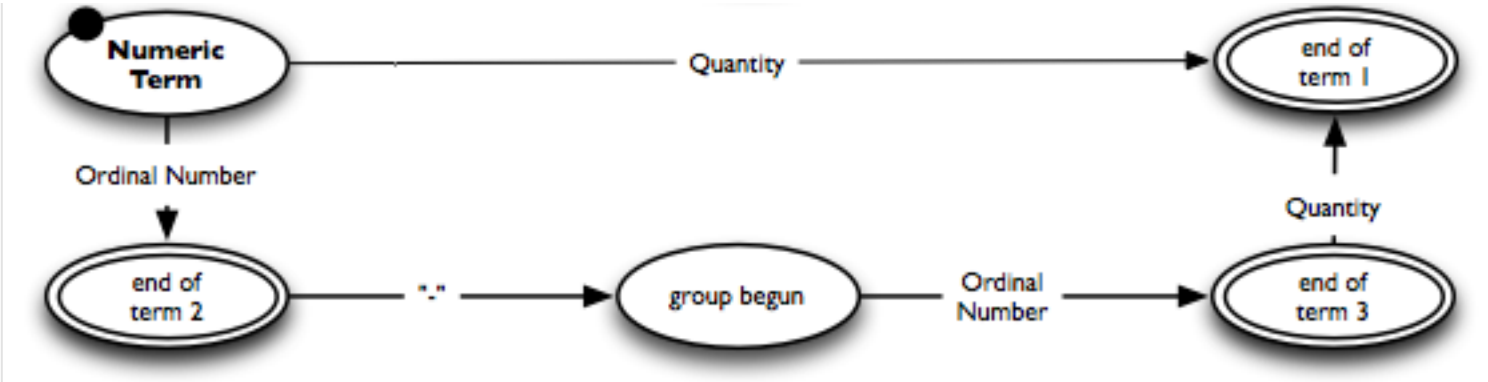



### Lexicons

- Relational term lexicon. “last”, “preceding”, “past”, “current”, “this”, “upcoming”, “next”, ...

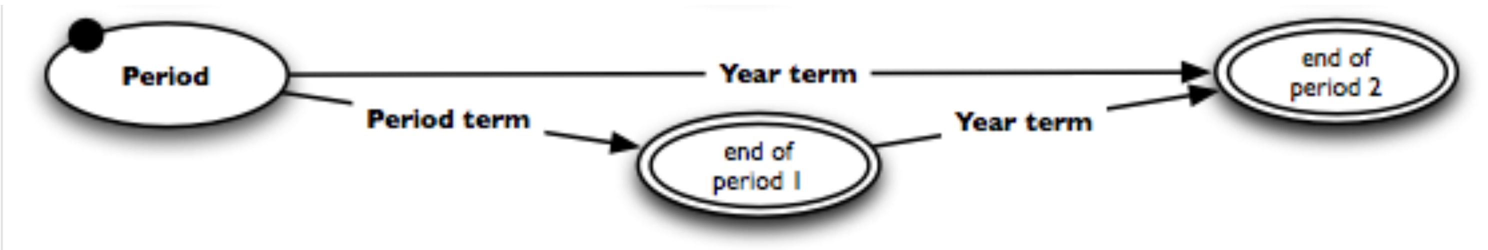
# Time Expression Recognition with Regular Expressions

## Sub-FSAs for Numeric Terms and Periods



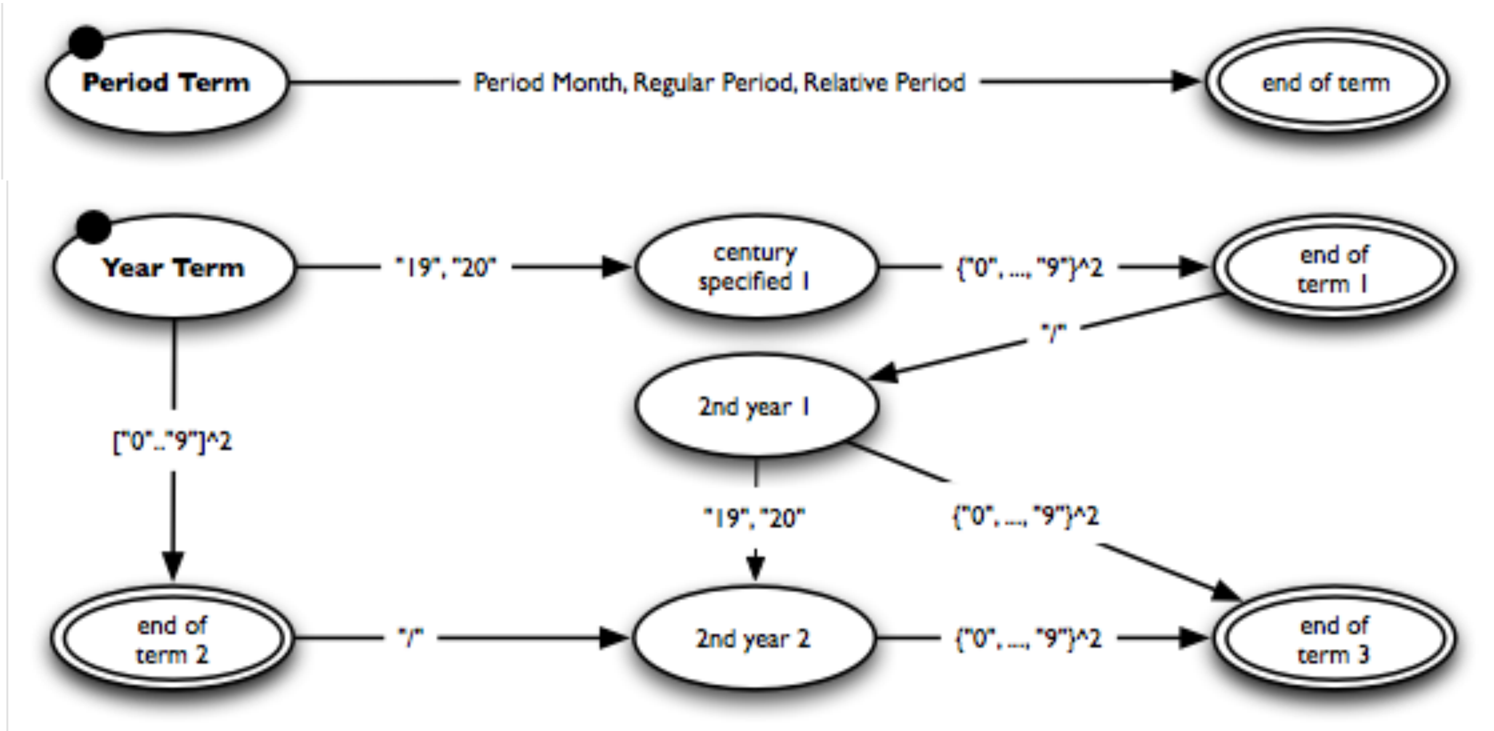

### Lexicons

- Ordinal number lexicon. “first”, “1st”, “second”, “2nd”, “third”, “3rd”, ...
- Quantity lexicon. “one”, “two”, “three”, “both”, “several”, “a hundred”, ...



# Time Expression Recognition with Regular Expressions

## Sub-FSAs for Period Terms and Year Terms




### Lexicon

- Period month lexicon. “March”, “Mar.”, “Mar”, “Fall”, “fall”, “Autumn”, ...
- Regular period lexicon. “year”, “month”, “quarter”, “half”, ...
- Relative period lexicon. “decade”, “reported time”, “time span”, ...

# Time Expression Recognition with Regular Expressions

## Sub-FSAs for Boundary Terms and Prepositional Terms



### Lexicons

- Boundary lexicon. "Beginning", "beginning", "End", "end", "Midth", ...
- Prep lexicon. "in", "within", "to", "for", "from", "since", ...
- Det lexicon. "the", "a", "an"

# Time Expression Recognition with Regular Expressions

## Evaluation

### Effectiveness of the time expressions (Wachsmuth, 2015)

- Originally developed for German texts; only this version was evaluated
- Data. Test set of *InfexBA Corpus* with 6038 business news sentences
- Evaluation metrics. Precision, recall, $F_1$-score, run-time per sentence

  Run-time measured on a standard computer from 2009

### Results compared to other analyses

| Task | Technique | Precision | Recall | $F_1$-score | ms/sent. |
|---|---|---|---|---|---|
| Time expression recognition | Regex | 0.91 | 0.97 | 0.94 | 0.36 |
| Money expression recognition | Regex | 0.99 | 0.95 | 0.97 | 0.68 |
| Forecast event extraction | Linear SVM | 0.87 | 0.93 | 0.90 | 0.81 |
| Time/Money relation extraction | Linear SVM | 0.75 | 0.88 | 0.81 | 10.41 |

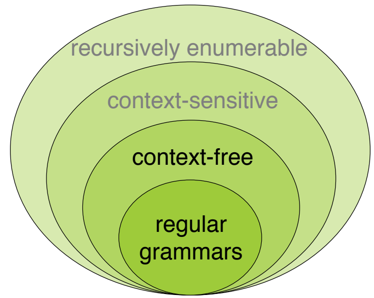
### Observations

- Regexes for semi-closed-class entity types such as time expressions can achieve very high effectiveness and efficiency.
- Their development is complex and time-intensive, though.

# Conclusion

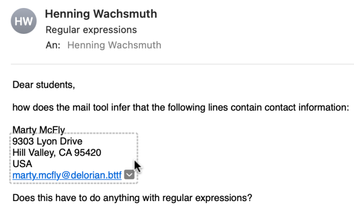
# Conclusion

## Formal grammars



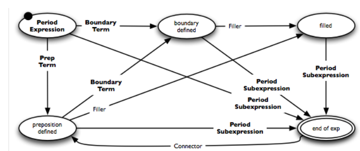

- Descriptions of valid structures in a language
- Production rules map non-terminals to terminals
- Regular grammars model sequential structures

## Regular expressions




- Describe regular grammars with (meta)characters
- Matched against texts to find instances of concepts
- Used in NLP particularly for (alpha)numeric entities

## Benefits and limitations




- Effective for semi-closed classes, often very efficient
- Development complex for sophisticated concepts
- Restriction to sequential patterns limits applicability

# References

## Much content and many examples taken from


- Daniel Jurafsky and Christopher D. Manning (2016). Natural Language Processing. Lecture slides from the Stanford Coursera course. https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html.
- Daniel Jurafsky and James H. Martin (2009). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. Prentice-Hall, 2nd edition.
- Friedhelm Meyer auf der Heide (2010). Einführung in Berechenbarkeit, Komplexität und Formale Sprachen. Begleitmaterial zur Vorlesung. https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Algorithmen/Lehre/Vorlesungsarchiv/WS_2009_10/Einfuehrung_in_die_Berechenbarkeit_K_u_f_S/skript.pdf
- Henning Wachsmuth (2015): Text Analysis Pipelines — Towards Ad-hoc Large-scale Text Mining. LNCS 9383, Springer.