# Chapter NLP:IV

## IV. Syntax

# Regular Grammars

## Woodchucks

**How much wood would a woodchuck chuck, if a woodchuck could chuck wood?**

# Regular Grammars

## Woodchucks

**How much wood would a woodchuck chuck, if a woodchuck could chuck wood?**

- So much wood as a woodchuck chuck would, if a woodchuck could chuck wood.
- A woodchuck would chuck as much wood as a he could, if a woodchuck could chuck wood.
- He would chuck, he would, as much as he could, and chuck as much wood as a woodchuck would, if a woodchuck could chuck wood.
- A woodchuck would chuck no amount of wood, since a woodchuck can't chuck wood.
- But if a woodchuck could and would chuck some wood, what amount of wood would a woodchuck chuck?
- Even if a woodchuck could chuck wood and even if a woodchuck would chuck wood, should a woodchuck chuck wood?
- A woodchuck should chuck if a woodchuck could chuck wood, as long as a woodchuck would chuck wood.

Remark: Yes, not all are really insightful examples ;-)

# Regular Grammars

## Mining Woodchucks from Text

**How can we find all of all these in a text?**

- “woodchuck”
- “Woodchuck”
- “woodchucks”
- “Woodchucks”
- “WOODCHUCK”
- “WOODCHUCKS”
- “wooooodchuck”
- “groundhog” (synonym)
- ... and so on

# Regular Grammars

## Regular Grammars to the Rescue

- A grammar $(\Sigma, N, S, R)$ is called regular if all rules in $R$ are of the form $U \to V$ with $U \in N$ and $V \in \{\varepsilon, v, vW\}$, where $\varepsilon$ is the empty word, $v \in \Sigma$, and $W \in N$.
- In an extended regular grammar, $v \in \Sigma^*$. We just refer to all as regular grammar.
- Intuitively, a structure defined by a regular grammar can be constructed from left to right (right-regular). From right to left would also be possible (left-regular).
- A language is regular, if there is a regular grammar that defines it.

### Representation of regular grammars

- Every regular grammar can be represented by a finite-state automaton.
- Every regular grammar can be represented by a regular expression.
- And vice versa. This should all already be known from your basic courses.

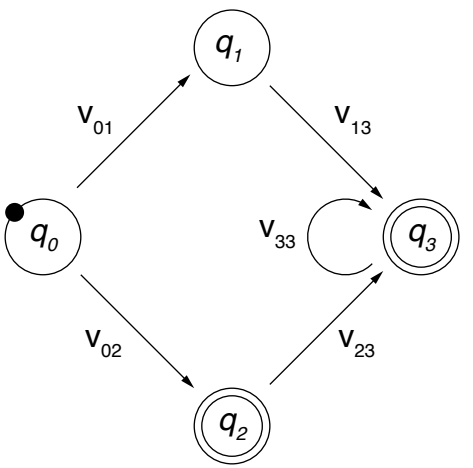
# Regular Grammars

## Finite-State Automata

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

**An 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), $\Sigma \cap Q = \emptyset$.
- $q_0$ 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 Grammars

## Regular Expressions (aka 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, ... (next pages).
- The example FSA from the previous slide is defined by the following regex.

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

### Use of regular expressions

- Definition of patterns that generalize over structures of a language.
- The patterns match all spans of text that contain any of the structures.

### Regular expressions in NLP

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

# Regular Grammars

## 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 metacharacters to efficiently encode specific regular-language constructions, such as negation and repetition.
- The main metacharacters are presented below in Python notation:

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

  The used metacharacters 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 when given token information.

# Regular Grammars

## Regular Expressions: Disjunction of Patterns

- Brackets `[]` specify a character class.

  `[wod]` matches “w” or “o” or “d” `[wW]` matches “w” or “W”

- 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”

- Combinations of different disjunctions are often useful.

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

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

  `[wod.]` matches “w”, “o”, “d”, and “.”

# Regular Grammars

## 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”

- Outside brackets, the caret `^` is interpreted as a normal character.

  `woodchuck^` matches “woodchuck^”

### Free choice

- The period `.` matches any character.

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

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

### Grouping

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

  `w[^(oo)]dchuck` matches any variation of the two o’s in “woodchuck”

# Regular Grammars

## 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 Grammars

## Regular Expressions: Repetition

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

  `woo*dchuck` matches “wodchuck”, “woodchuck”, “woooodchuck”, “wooooodchuck”, ...

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

  `woo+dchuck` matches “woodchuck”, “woooodchuck”,“wooooodchuck”, ...

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

  `woo?dchuck` matches “wodchuck” and “woodchuck”

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

  `to*` matches “too”, not “toо”, ...

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

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

# Regular Grammars

## 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 | `woo+dchuck` |
| `?` | Zero or one repetition | `woodchucks?` |

# Regular Grammars

## Regular Expressions: Examples

### The

- Regex for all instances 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][sS]? | groundhog
```

### Email Adresses

- All email addresses from a selection of top-level domains, which contain no special character (besides periods and “@”).

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

# Regular Grammars

## Time Expression Recognition with Regular Expressions

- A time expression is 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 for event and temporal relation extraction.

### Approach in a nutshell

- Models phrase structure of time expressions with a sophisticated regex.
- Include lexicons derived from a training set to identify closed-class terms, such as month names and prepositions.
- Match regex with sentences of a text.

  The matching approach can easily be adapted to any other type of information.

# Regular Grammars

## 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)
 1.     List<TimeExpression> matches ← ()
 2.     for each sentence ∈ sentences do
 3.         int index ← 0
 4.         while index < sentence.length - 1 do
 5.
 6.             // ...
 7.
 8.
 9.             index ← index + 1
10.     return matches
```

# Regular Grammars

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

Remark: Most programming languages provide explicit matching classes.

# Regular Grammars

## Time Expression Recognition with Regular Expressions: Complete Regex 1/2

```
((((((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]a
y|[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))|((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]egin|[Ee]nd|[eE]nd|[Mm]idth|[mM]idth)((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*([tT]he|[tT]his|[tT]h
ese|[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|[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]leventh)))?((\s+(\r(\n)?|\n)?|(\r(\n)?|\n))\s*((1|2|3|4|5|6|7|8|9)\d?|([oO]ne|[sS]everal|[s
S]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]our
ty|[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]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]int
h|[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]hre
e|[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]ixt
y|[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|[A
a]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]ctobe
r|[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]epo
rted\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]p
an|[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|[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]undred|[a
A]\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|[e
E]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]wenty|[tT]hirty|[fF]ourty|[fF]orty|[
fF]ifty|[sS]ixty|[sS]eventy|[eE]ighty|[nN]inety|[hH]undred|
```

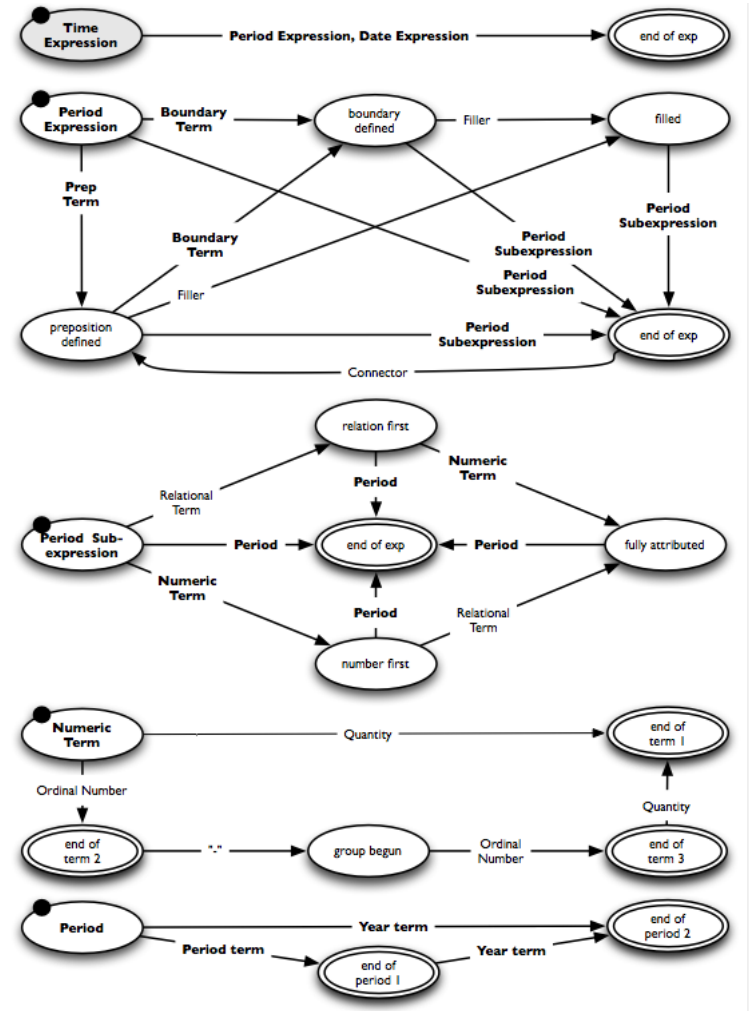
# Regular Grammars

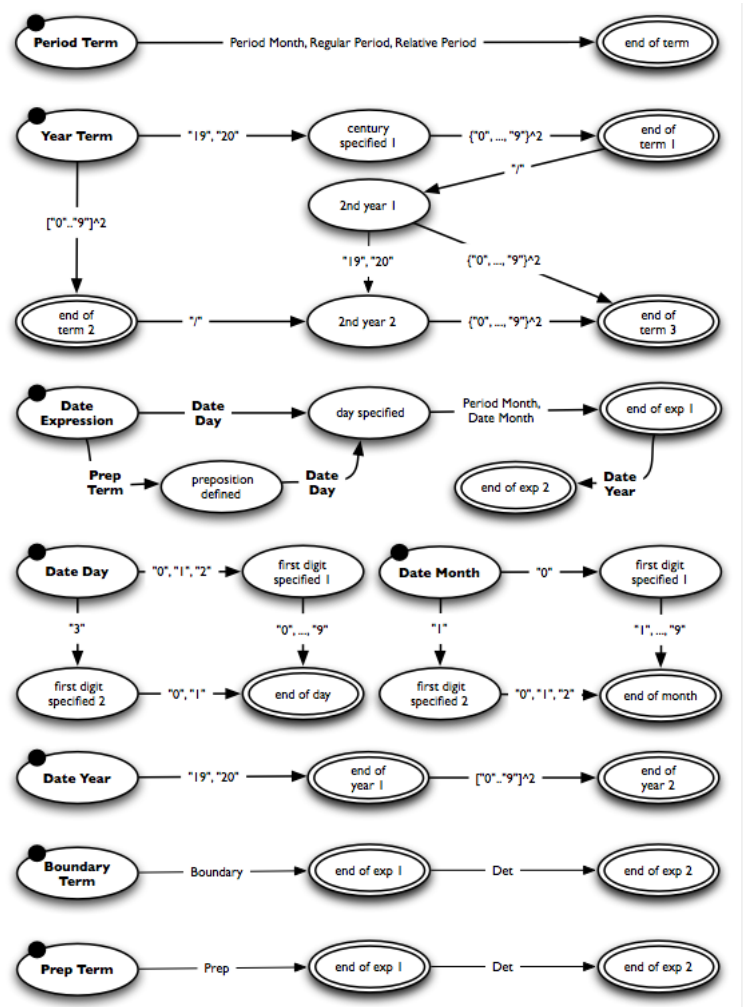
## Time Expression Recognition with Regular Expressions: Complete Regex 2/2

```
[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]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]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]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]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|[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|[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]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]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]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]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]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]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))))*))
```

# Regular Grammars

## Time Expression Recognition with Regular Expressions: Complete Regex as FSA



# Regular Grammars

## Time Expression Recognition: FSA Top-level


### Notice

- Bold edge labels indicate sub-FSAs, regular ones 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 usually rather works bottom-up.

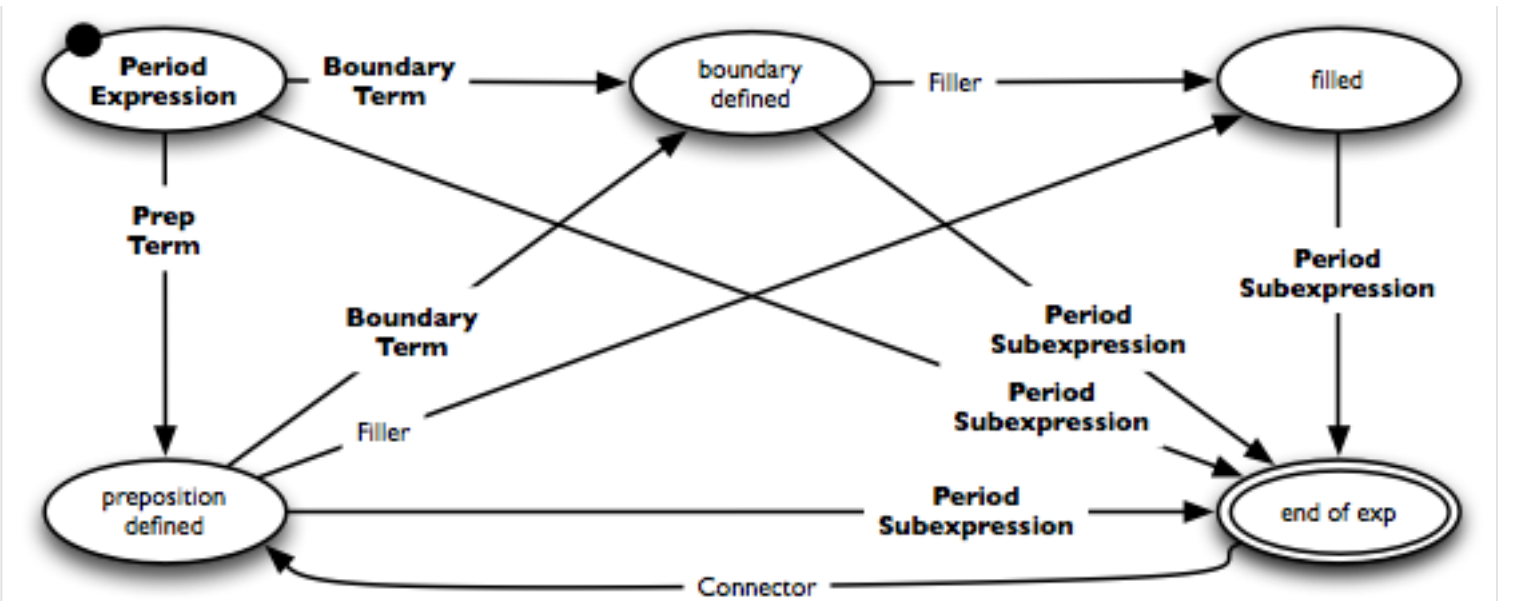
### Example

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

  prep filler boundary relational period connector ordinal period year

# Regular Grammars

## Time Expression Recognition: Sub-FSA for Period Expressions

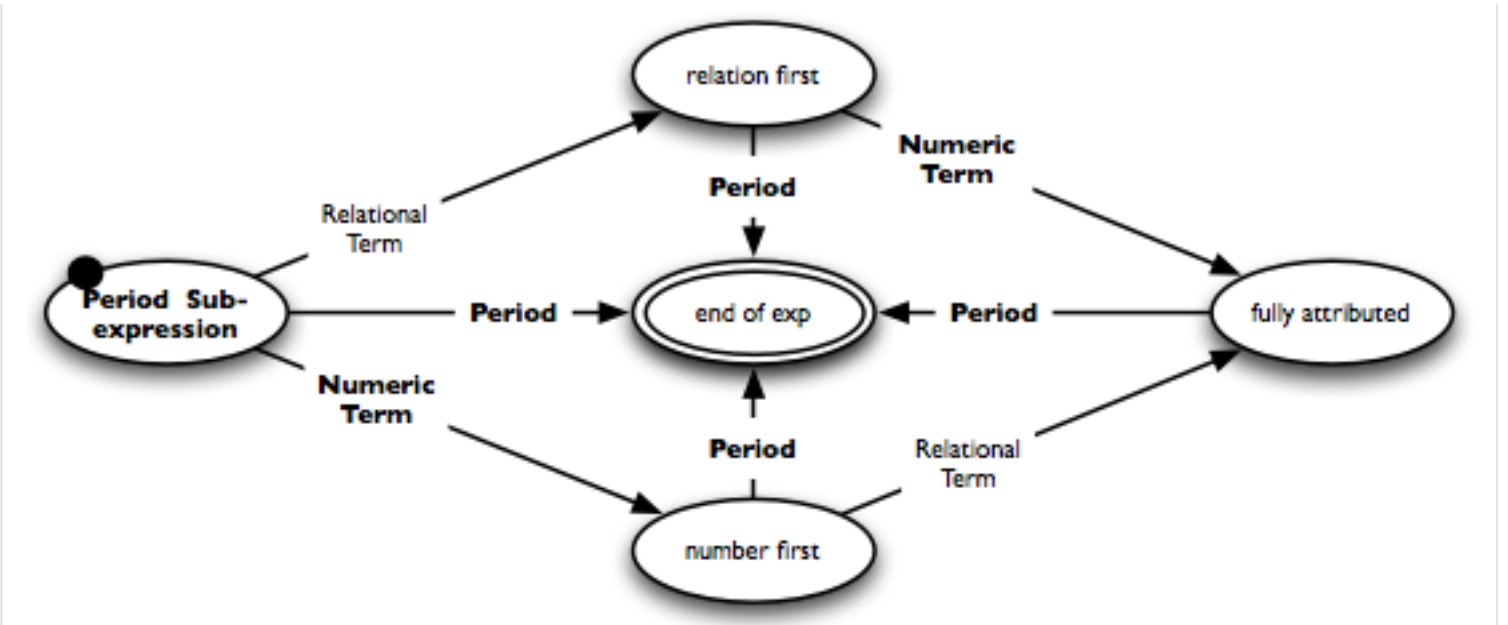



## Lexicons

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

# Regular Grammars

## Time Expression Recognition: Sub-FSA for Period Subexpressions

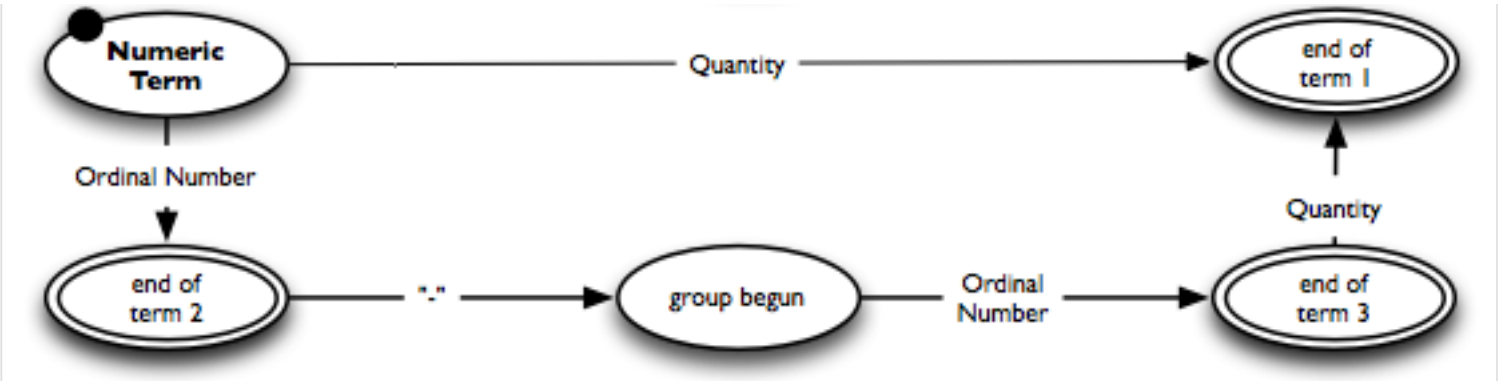



## Lexicons

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

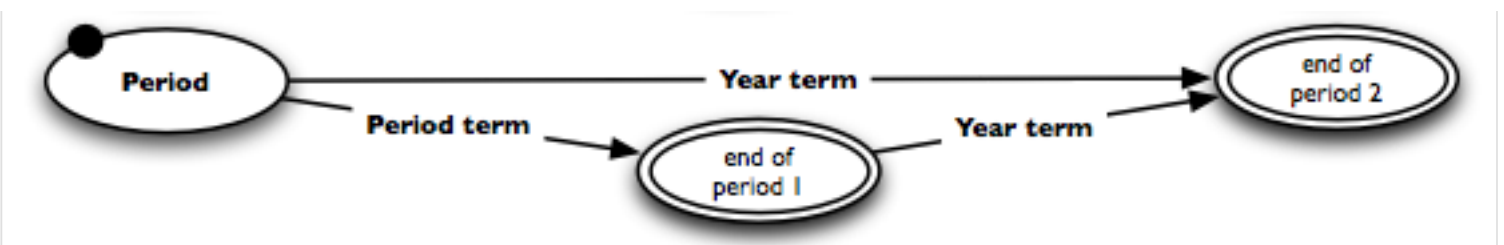
# Regular Grammars

## Time Expression Recognition: Sub-FSAs for Numeric Terms and Periods



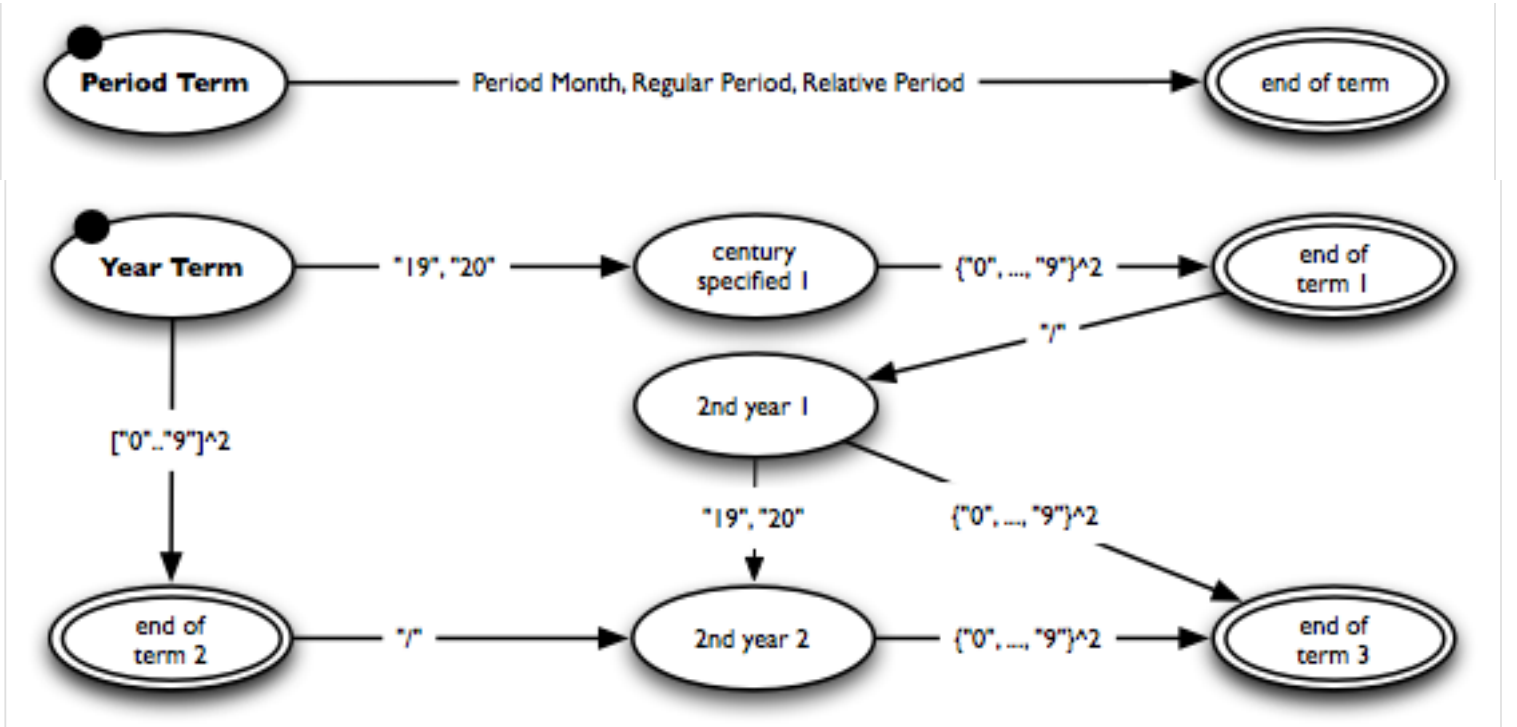

## Lexicons

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



# Regular Grammars

## Time Expression Recognition: Sub-FSAs for Period and Year Terms

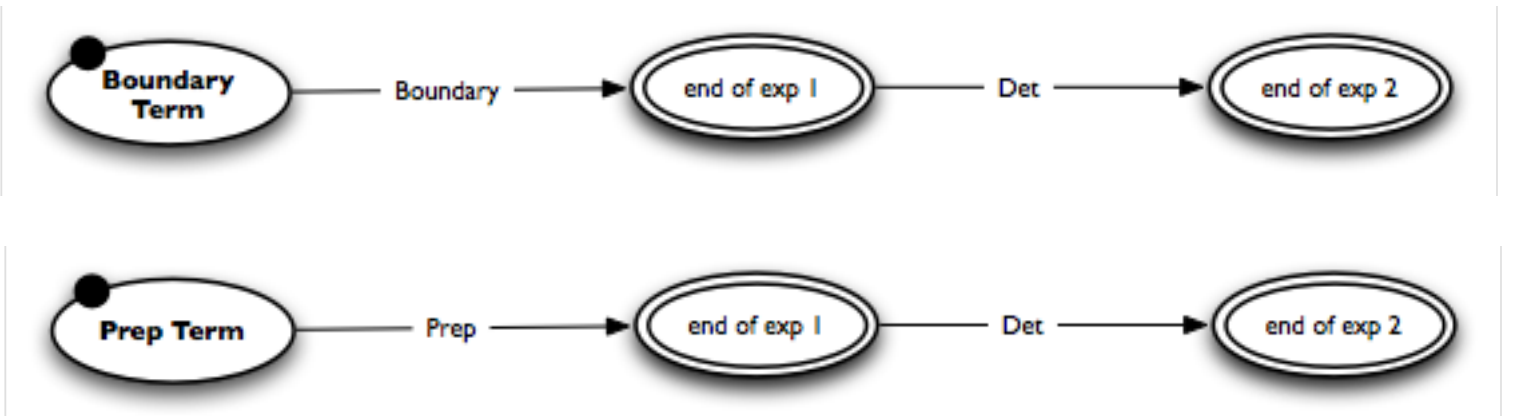



## Lexicon

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

# Regular Grammars

## Time Expression Recognition: Sub-FSAs for Boundary and Prepositional Terms




### Lexicons

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

# Regular Grammars

## Time Expression Recognition with Regular Expressions: Evaluation

### How well does the regex perform?

- Originally developed for German texts; only this version was evaluated.
- Data: Test set of the *InfexBA Revenue corpus* with 6038 sentences from business news articles.
- Evaluation measures: Precision, recall, $F_1$-score, runtime per sentence.

  Runtime measured on a standard computer from 2009.

### Results

| Approach | Precision | Recall | $F_1$-score | ms/sentence |
|---|---|---|---|---|
| Regex | 0.91 | 0.97 | 0.94 | 0.36 |

### Conclusion

- 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.

Who said life would be easy??!