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 (Σ, 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 ε 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 $(Q, \Sigma, q_0, F, \delta)$

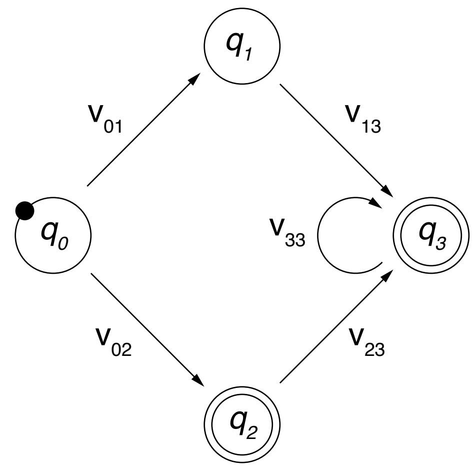
Q A finite set of $n > 0$ states, $Q = \{q_0, ..., q_n\}$.

Σ 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, $F \subseteq Q$.

δ 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 Σ as a sequence of characters (from Σ) and metacharacters.
- Metacharacters denote disjunction, negation, repetition, ... (next pages).
- The example FSA from the previous slide is defined by the following regex.

$$\mathrm{v}_{02} \quad | \quad (\mathrm{v}_{01}\mathrm{v}_{13} \,|\, \mathrm{v}_{02}\mathrm{v}_{23})\,\mathrm{v}_{33}{}^{*}$$

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”, “wooodchuck”, “woooodchuck”, ...

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

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

- 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 “too”, ...

- 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	`woooo*dchuck`
`+`	One or more repetitions	`wooo+dchuck`
`?`	Zero or one repetition	`woodchucks?`

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)

    List<TimeExpression> matches ← ()
    for each sentence ∈ sentences do
        int index ← 0
        while index < sentence.length - 1 do

            // ...


            index ← index + 1
    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)

    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
```

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|
```

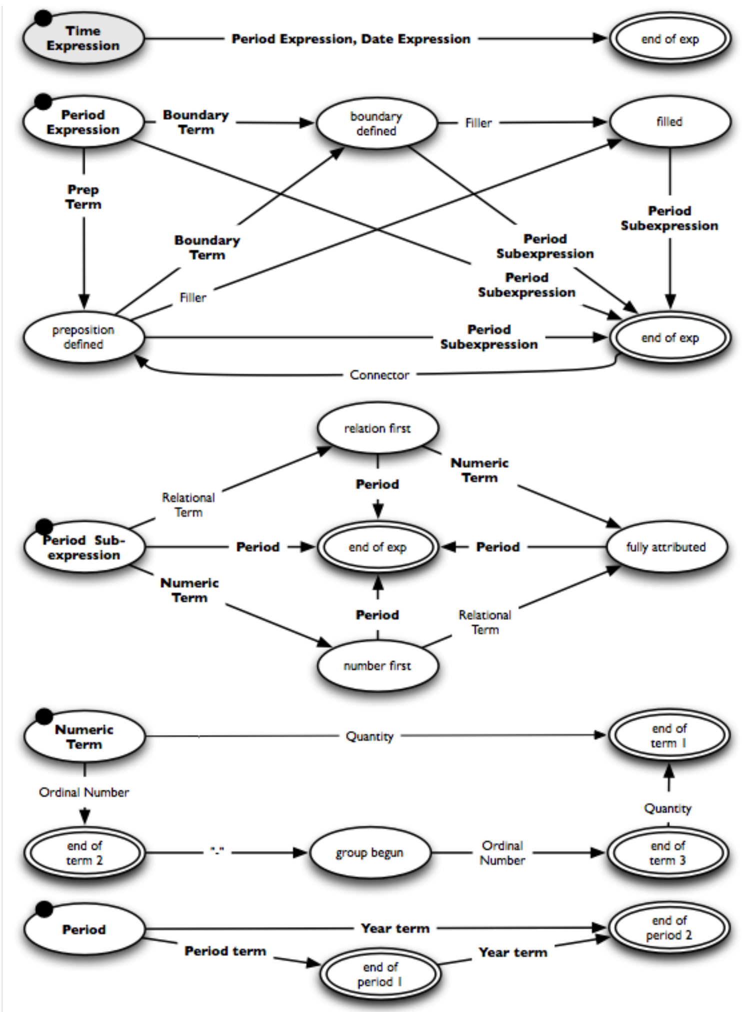
Regular Grammars

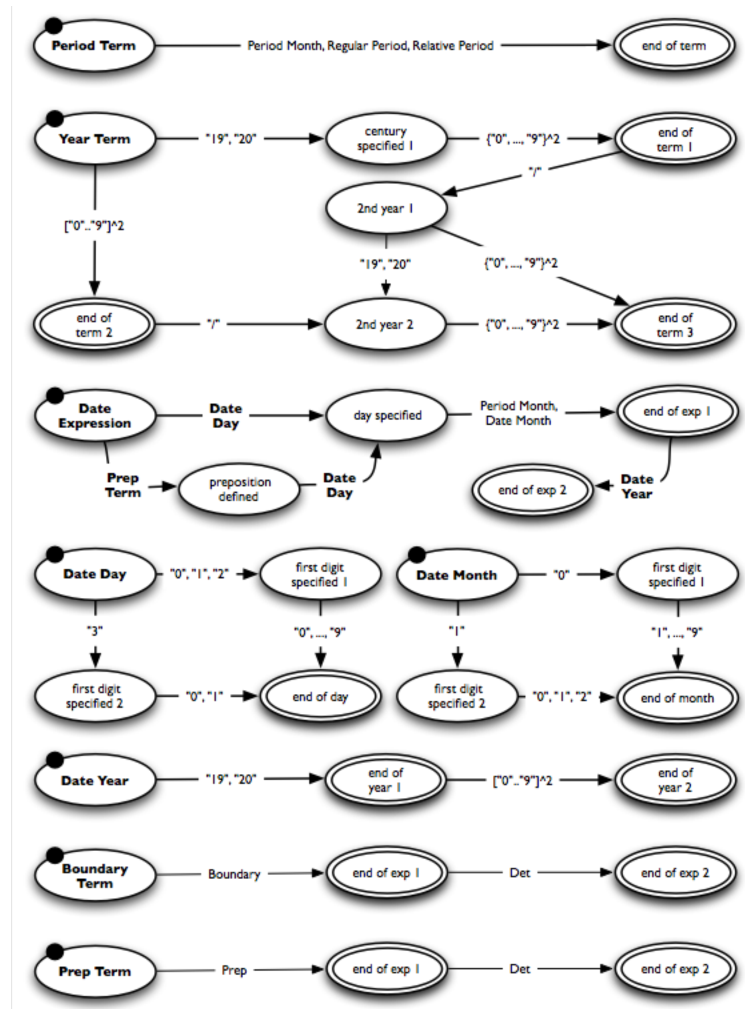
Time Expression Recognition with Regular Expressions: Complete Regex 2/2

```
[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]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))|((1[012]?|2|3|4|5|6|7|8|9)(\.|())|([fF]irst|[sS]econd|[tT]hird|[fF]our
th|[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\.|[D
d]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?t
ime|[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]o
llowing|[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]econ
d|[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]lev
en|[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]ight
y|[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]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]welve|[tT]wenty|[tT]h
irty|[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\.|[F
f]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]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]eported\s\s?tim
e|[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

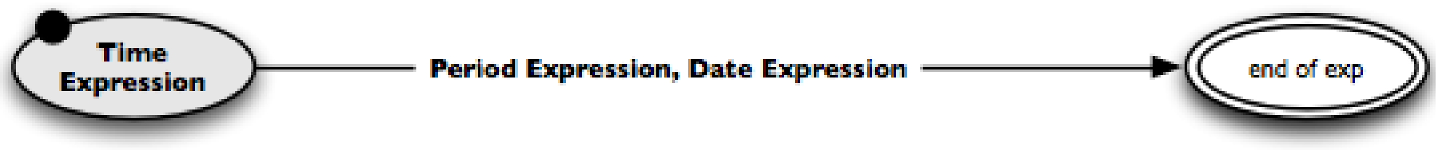
Time Expression Recognition with Regular Expressions: Complete Regex as FSA





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.

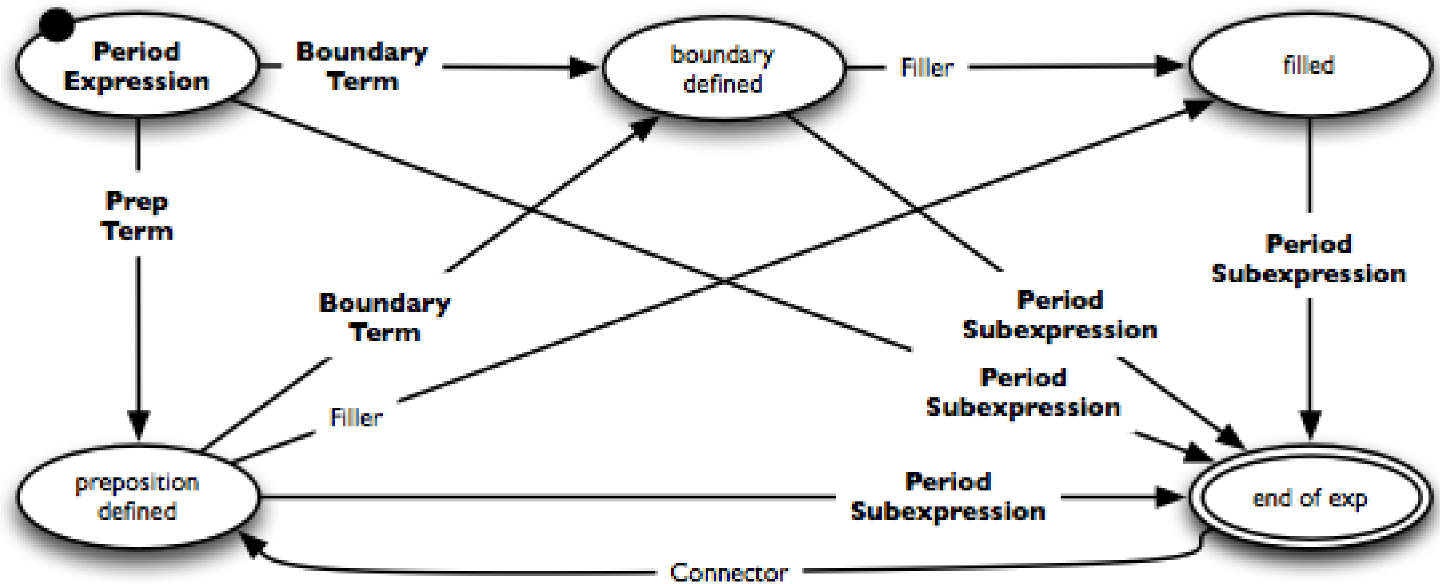
Example

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

“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

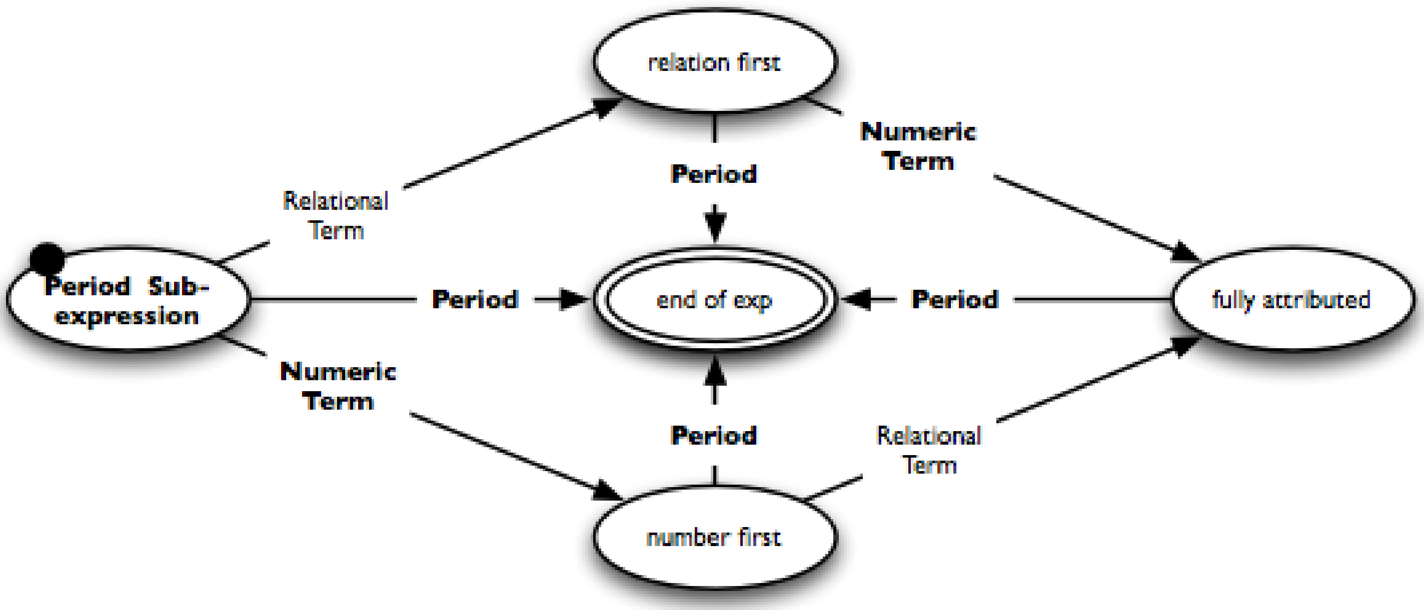


Lexicons

- **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

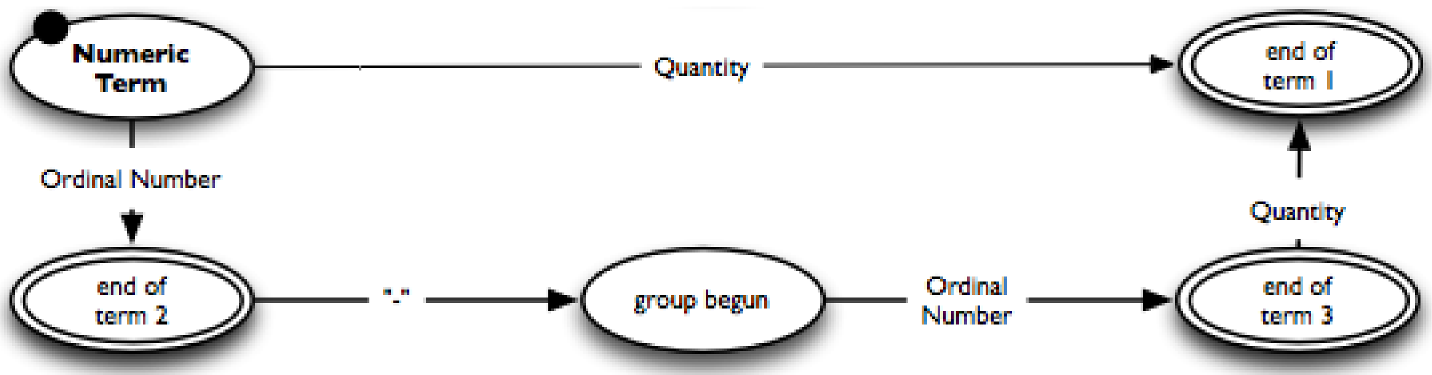


Lexicons

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

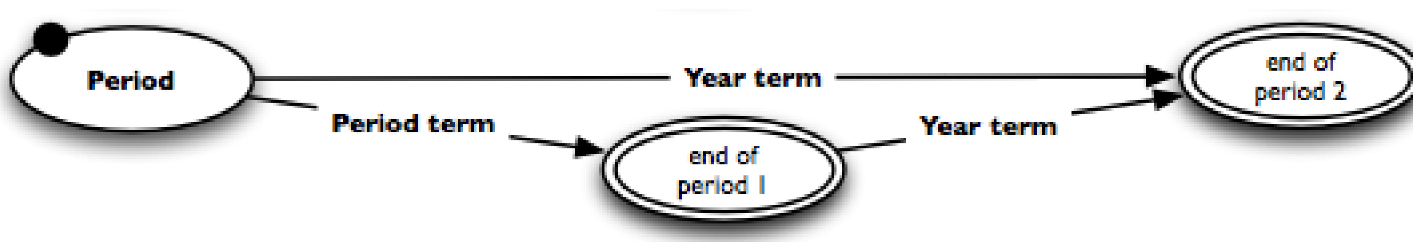
Regular Grammars

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



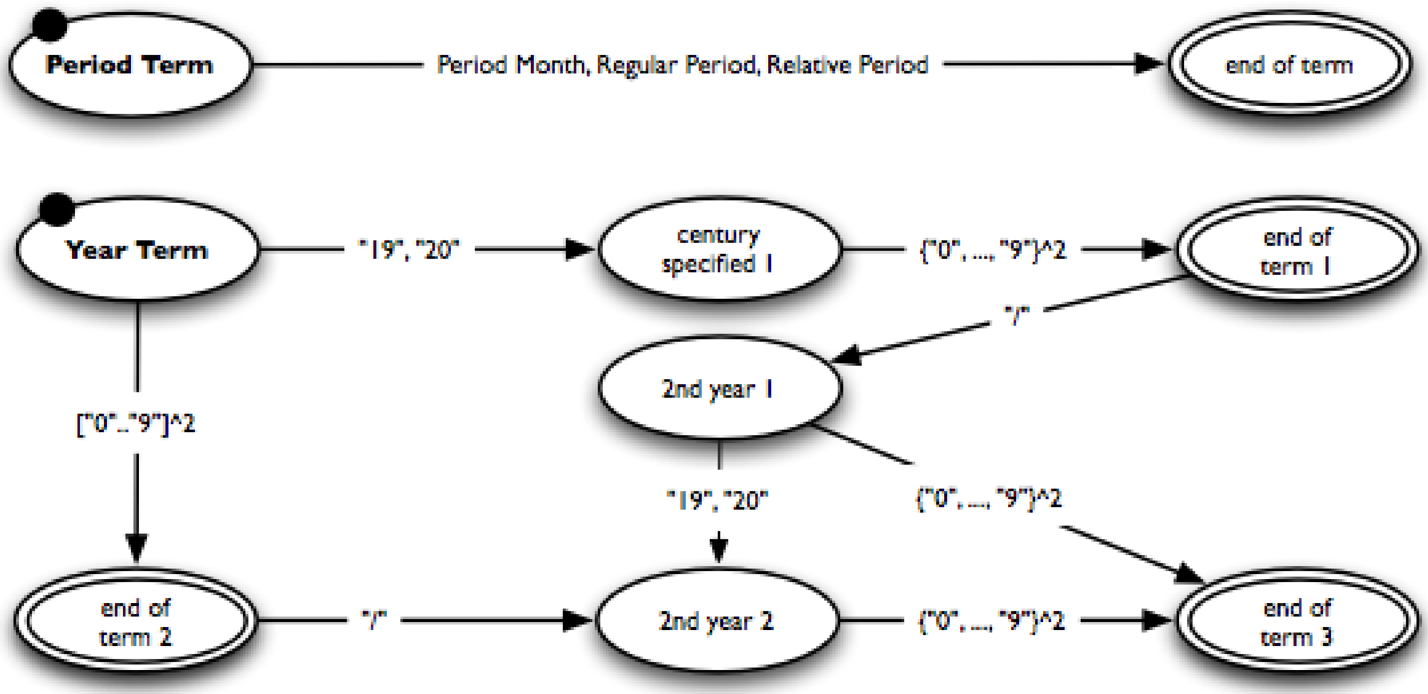
Lexicons

- 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

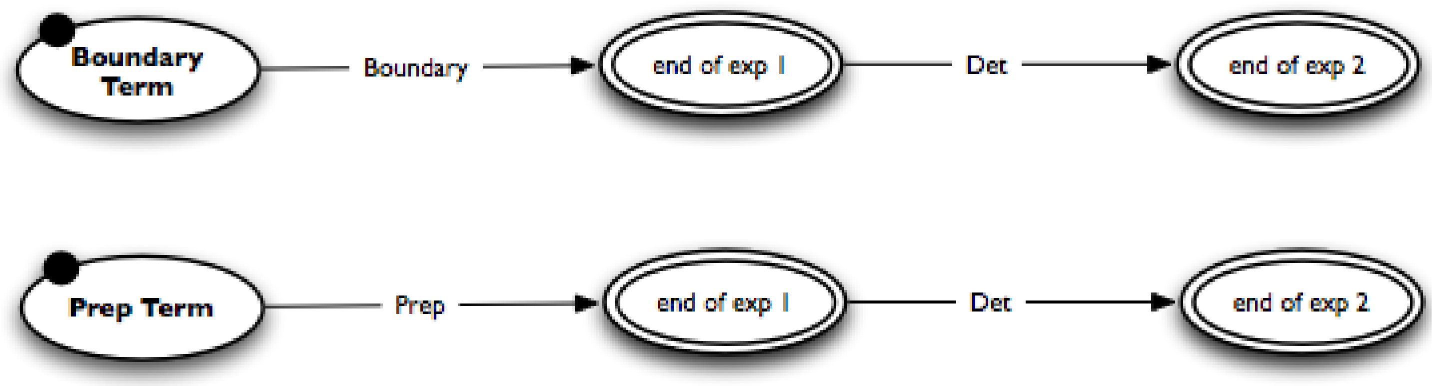


Lexicon

- 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”

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??!

Chapter NLP:IV

IV. Syntax

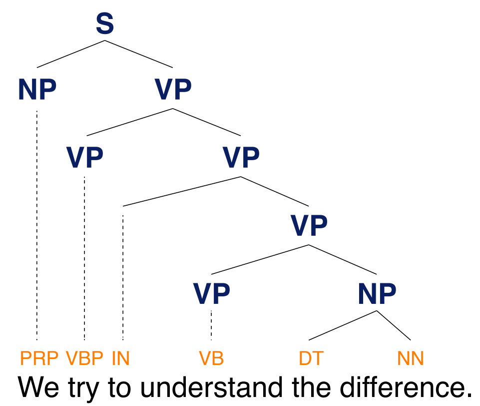
- Introduction
- Regular Grammars
- Probabilistic Context-Free Grammars
- Parsing based on a PCFG
- Dependency Grammars

(Probabilistic) Context-Free Grammars

Phrase grammar vs. Dependency grammar

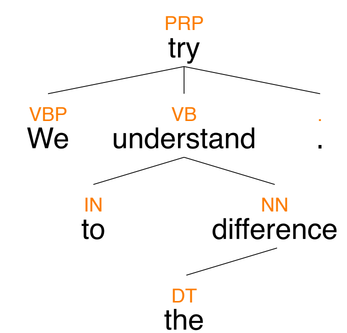
Phrase structure grammar

- Models the constituents of a sentence and how they are composed of each other.
- Constituency (parse) tree. Inner nodes are non-terminals, leafs terminals.



Dependency grammar

- Models the dependencies between the words in a sentence.
- Dependency (parse) tree. All nodes are terminals (root nearly always the main verb).



(Probabilistic) Context-Free Grammars

What is a Phrase Structure Grammar?

- A phrase structure grammar is a context-free grammar (CFG).
- A grammar (Σ, N, S, R) is called context-free if all rules in R are of the form $U \to V$ with $U \in N$ and $V \in (N \cup \Sigma)^*$.
- A language is context-free, if there is a CFG that defines it.

NLP phrase structure grammar $(\Sigma,\ N_{phr} \cup N_{pos},\ S,\ R_{phr} \cup R_{pos})$

N_{phr} A finite set of structural non-terminal symbols (i.e., the phrase types).

N_{pos} A finite set of lexical pre-terminal symbols (i.e., the part-of-speech tags), $N_{phr} \cap N_{pos} = \emptyset$.

R_{phr} A finite set of structure production rules of the form $U \to V$ with $U \in N_{phr}$ and $V \in (N_{phr} \cup N_{pos})^*$.

R_{pos} A finite set of lexicon production rules of the form $U \to v$ with $U \in N_{pos}$ and $v \in \Sigma$.

Σ, S As before

In addition to S, NLP usually includes an extra node ROOT at the top.

(Probabilistic) Context-Free Grammars

Example CFG

Structural rules			Lexical rules	
s1	S $\rightarrow$ NP VP		l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP		l2	N $\rightarrow$ fish
s3	VP $\rightarrow$ V NP PP		l3	N $\rightarrow$ tanks
s4	NP $\rightarrow$ NP NP		l4	N $\rightarrow$ rods
s5	NP $\rightarrow$ NP PP	// binary	l5	V $\rightarrow$ people
s6	NP $\rightarrow$ N	// unary	l6	V $\rightarrow$ fish
s7	NP $\rightarrow$ ε	// empty	l7	V $\rightarrow$ tanks
s8	PP $\rightarrow$ P NP		l8	P $\rightarrow$ with

Example sentences created by the grammar

- “people fish tanks”
- “people fish with rods”

(Probabilistic) Context-Free Grammars

Chomsky Normal Form

A CFG is in Chomsky Normal Form if all rules in R are in either of the forms:

$$U \rightarrow VW$$
$$U \rightarrow v$$
$$S \rightarrow e$$

where U, V, W: nonterminals, S: start, v: terminal symbol, and e: empty string.

Transformation into normal form

- Binarization: n-ary rules are divided by using new non-terminals, $n > 2$.
- Cleaning: Empties and unaries are removed recursively.
- The transformation does not change the language defined by a grammar, but it may result in different trees.

Why transforming?

- Restricting a CFG in such a way is key to efficient parsing.
- Binarization is crucial for cubic time.
- Cleaning is not mandatory, but makes parsing quicker and cleaner.

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation: Pseudocode

Signature

- Input. The production rules $R = R_{phr} \cup R_{pos}$ of a CFG.
- Output. The production rules R^* of the normalized version of the CFG.

toChomskyNormalForm(Production rules R)

```
while an empty (U → ε) ∈ R do
    // ...
while a unary (U → V) ∈ R do
    // ...
while an n-ary (U → V_1 ... V_n) ∈ R do // n ≥ 3
    // ...
return R
```

Chomsky Normal Form Transformation: Pseudocode

Signature

- Input. The production rules $R = R_{phr} \cup R_{pos}$ of a CFG.
- Output. The production rules R^* of the normalized version of the CFG.

toChomskyNormalForm(Production rules R)

while an empty $(U \to \varepsilon) \in R$ **do**
    $R \leftarrow R \setminus \{U \to \varepsilon\}$
    for each rule $(V \to V_1 \ldots V_k\, U\, W_1 \ldots W_l) \in R$ **do** // $k, l \geq 0$
        $R \leftarrow R \cup \{V \to V_1 \ldots V_k\;\; W_1 \ldots W_l\}$
while a unary $(U \to V) \in R$ **do**
    // ...
while an n-ary $(U \to V_1 \ldots V_n) \in R$ **do** // $n \geq 3$
    // ...
return R

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation: Pseudocode

Signature

- Input. The production rules $R = R_{phr} \cup R_{pos}$ of a CFG.
- Output. The production rules R^* of the normalized version of the CFG.

toChomskyNormalForm(Production rules R)

while an empty $(U \rightarrow \varepsilon) \in R$ **do**
    $R \leftarrow R \setminus \{U \rightarrow \varepsilon\}$
    for each rule $(V \rightarrow V_1 \ldots V_k\, U\, W_1 \ldots W_l) \in R$ **do** // $k, l \geq 0$
        $R \leftarrow R \cup \{V \rightarrow V_1 \ldots V_k \; W_1 \ldots W_l\}$
while a unary $(U \rightarrow V) \in R$ **do**
    $R \leftarrow R \setminus \{U \rightarrow V\}$
    if $U \neq V$ **then**
        for each $(V \rightarrow V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \cup \{U \rightarrow V_1 \ldots V_k\}$
        if not $(W \rightarrow V_1 \ldots V_k \; V\, W_1 \ldots W_l) \in R$ **then**
            for each $(V \rightarrow V_1 \ldots V_k) \in R$ **do** $R \leftarrow R \setminus \{V \rightarrow V_1 \ldots V_k\}$
while an n-ary $(U \rightarrow V_1 \ldots V_n) \in R$ **do** // $n \geq 3$
    // ...
return R

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation: Pseudocode

Signature

- Input. The production rules $R = R_{phr} \cup R_{pos}$ of a CFG.
- Output. The production rules R^* of the normalized version of the CFG.

toChomskyNormalForm(Production rules R)

```
while an empty (U → ε) ∈ R do
    R ← R \ {U → ε}
    for each rule (V → V_1 ... V_k U W_1 ... W_l) ∈ R do   // k, l ≥ 0
        R ← R ∪ {V → V_1 ... V_k  W_1 ... W_l}
while a unary (U → V) ∈ R do
    R ← R \ {U → V}
    if U ≠ V then
        for each (V → V_1 ... V_k) ∈ R do  R ← R ∪ {U → V_1 ... V_k}
        if not (W → V_1 ... V_k  V W_1 ... W_l) ∈ R then
            for each (V → V_1 ... V_k) ∈ R do  R ← R \ {V → V_1 ... V_k}
while an n-ary (U → V_1 ... V_n) ∈ R do   // n ≥ 3
    R ← (R \ {U → V_1 ... V_n}) ∪ {U → V_1 U_V_1, U_V_1 → V_2 ... V_n}
return R
```

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation: Pseudocode

Essentially:

1. If S occurs on some RHS, create a new start symbol S' and a new production $S' \to S$.
2. Remove null productions
3. Remove unit productions
4. Replace each production $A \to B1 \dots Bn$ where $n > 2$ with $A \to B1C$ where $C \to B2 \dots Bn$; repeat for all productions with two or more symbols on right.
5. If any RHS is in the form $A \to aB$ where a: terminal and A, B: nonterminal, then replace it with $A \to XB$ and $X \to a$; repeat for every production in the form $A \to aB$.

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation: Pseudocode

Essentially:

1. If S occurs on some RHS, create a new start symbol S' and a new production $S' \to S$.
2. Remove null productions
   (a) Find out nullable nonterminal variables which derive ϵ.
   (b) For each production $A \to a$, add all productions $A \to x$ where x is obtained from a by removing one or more nonterminals from step 1 above.
   (c) Add the result of step 2 to the original productions and remove empty productions.
3. Remove unit productions
   (a) To remove $A \to B$, add production $A \to x$ for every occurrence of $B \to x$; x: terminal (also ϵ)
   (b) Delete $A \to B$ from the grammar; repeat 1-2 until all unit productions removed.
4. Replace each production $A \to B1 \ldots Bn$ where $n > 2$ with $A \to B1C$ where $C \to B2 \ldots Bn$; repeat for all productions with two or more symbols on right.
5. If any RHS is in the form $A \to aB$ where a: terminal and A, B: nonterminal, then replace it with $A \to XB$ and $X \to a$; repeat for every production in the form $A \to aB$.

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Empties (Removal)

	Structural rules		Lexical rules
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	N $\rightarrow$ fish
s3	VP $\rightarrow$ V NP PP	l3	N $\rightarrow$ tanks
s4	NP $\rightarrow$ NP NP	l4	N $\rightarrow$ rods
s5	NP $\rightarrow$ NP PP	l5	V $\rightarrow$ people
s6	NP $\rightarrow$ N	l6	V $\rightarrow$ fish
s7	NP $\rightarrow$ ε	l7	V $\rightarrow$ tanks
s8	PP $\rightarrow$ P NP	l8	P $\rightarrow$ with

Removal of empties

- Add new rules for each rule where NP occurs on the right side.

  Pseudocode lines 2–4.

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Empties (Addition)

Structural rules		Lexical rules	
s1	S → NP VP	l1	N → people
s1'	S → VP	l2	N → fish
s2	VP → V NP	l3	N → tanks
s2'	VP → V	l4	N → rods
s3	VP → V NP PP	l5	V → people
s3'	VP → V PP	l6	V → fish
s4	NP → NP NP	l7	V → tanks
s4'	NP → NP	l8	P → with
s5	NP → NP PP		
s5'	NP → PP		
s6	NP → N		
s8	PP → P NP		
s8'	PP → P		

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries (Removal)

Structural rules		Lexical rules	
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s1'	S $\rightarrow$ VP	l2	N $\rightarrow$ fish
s2	VP $\rightarrow$ V NP	l3	N $\rightarrow$ tanks
s2'	VP $\rightarrow$ V	l4	N $\rightarrow$ rods
s3	VP $\rightarrow$ V NP PP	l5	V $\rightarrow$ people
s3'	VP $\rightarrow$ V PP	l6	V $\rightarrow$ fish
s4	NP $\rightarrow$ NP NP	l7	V $\rightarrow$ tanks
s4'	NP $\rightarrow$ NP	l8	P $\rightarrow$ with
s5	NP $\rightarrow$ NP PP		
s5'	NP $\rightarrow$ PP		
s6	NP $\rightarrow$ N		
s8	PP $\rightarrow$ P NP		
s8'	PP $\rightarrow$ P		

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries (Addition)

Structural rules		Lexical rules	
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	N $\rightarrow$ fish
s2”	S $\rightarrow$ V NP	l3	N $\rightarrow$ tanks
s2’	VP $\rightarrow$ V	l4	N $\rightarrow$ rods
s2”’	S $\rightarrow$ V	l5	V $\rightarrow$ people
s3	VP $\rightarrow$ V NP PP	l6	V $\rightarrow$ fish
s3”	S $\rightarrow$ V NP PP	l7	V $\rightarrow$ tanks
s3’	VP $\rightarrow$ V PP	l8	P $\rightarrow$ with
s3”’	S $\rightarrow$ V PP		
s4	NP $\rightarrow$ NP NP		
s4’	NP $\rightarrow$ NP		
s5	NP $\rightarrow$ NP PP		
s5’	NP $\rightarrow$ PP		
s6	NP $\rightarrow$ N		
s8	PP $\rightarrow$ P NP		
s8’	PP $\rightarrow$ P		

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries 2 (Removal)

Structural rules		Lexical rules	
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	N $\rightarrow$ fish
s2"	S $\rightarrow$ V NP	l3	N $\rightarrow$ tanks
s2'	VP $\rightarrow$ V	l4	N $\rightarrow$ rods
s2"'	S $\rightarrow$ V	l5	V $\rightarrow$ people
s3	VP $\rightarrow$ V NP PP	l6	V $\rightarrow$ fish
s3"	S $\rightarrow$ V NP PP	l7	V $\rightarrow$ tanks
s3'	VP $\rightarrow$ V PP	l8	P $\rightarrow$ with
s3"'	S $\rightarrow$ V PP		
s4	NP $\rightarrow$ NP NP		
s4'	NP $\rightarrow$ NP		
s5	NP $\rightarrow$ NP PP		
s5'	NP $\rightarrow$ PP		
s6	NP $\rightarrow$ N		
s8	PP $\rightarrow$ P NP		
s8'	PP $\rightarrow$ P		

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries 2 (Addition)

Structural rules		Lexical rules	
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	N $\rightarrow$ fish
s2”	S $\rightarrow$ V NP	l3	N $\rightarrow$ tanks
s2”’	S $\rightarrow$ V	l4	N $\rightarrow$ rods
s3	VP $\rightarrow$ V NP PP	l5	V $\rightarrow$ people
s3”	S $\rightarrow$ V NP PP	l5’	VP $\rightarrow$ people
s3’	VP $\rightarrow$ V PP	l6	V $\rightarrow$ fish
s3”’	S $\rightarrow$ V PP	l6’	VP $\rightarrow$ fish
s4	NP $\rightarrow$ NP NP	l7	V $\rightarrow$ tanks
s4’	NP $\rightarrow$ NP	l7’	VP $\rightarrow$ tanks
s5	NP $\rightarrow$ NP PP	l8	P $\rightarrow$ with
s5’	NP $\rightarrow$ PP		
s6	NP $\rightarrow$ N		
s8	PP $\rightarrow$ P NP		
s8’	PP $\rightarrow$ P		

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries 3 (Removal)

Structural rules		Lexical rules	
s1	S → NP VP	l1	N → people
s2	VP → V NP	l2	N → fish
s2”	S → V NP	l3	N → tanks
s2”’	S → V	l4	N → rods
s3	VP → V NP PP	l5	V → people
s3”	S → V NP PP	l5’	VP → people
s3’	VP → V PP	l6	V → fish
s3”’	S → V PP	l6’	VP → fish
s4	NP → NP NP	l7	V → tanks
s4’	NP → NP	l7’	VP → tanks
s5	NP → NP PP	l8	P → with
s5’	NP → PP		
s6	NP → N		
s8	PP → P NP		
s8’	PP → P		

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries 3 (Addition)

Structural rules		**Lexical rules**	
s1	S $\rightarrow$ NP VP	l1	N $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	N $\rightarrow$ fish
s2”	S $\rightarrow$ V NP	l3	N $\rightarrow$ tanks
s3	VP $\rightarrow$ V NP PP	l4	N $\rightarrow$ rods
s3”	S $\rightarrow$ V NP PP	l5	V $\rightarrow$ people
s3’	VP $\rightarrow$ V PP	l5’	VP $\rightarrow$ people
s3”’	S $\rightarrow$ V PP	l5”	S $\rightarrow$ people
s4	NP $\rightarrow$ NP NP	l6	V $\rightarrow$ fish
s4’	NP $\rightarrow$ NP	l6’	VP $\rightarrow$ fish
s5	NP $\rightarrow$ NP PP	l6”	S $\rightarrow$ fish
s5’	NP $\rightarrow$ PP	l7	V $\rightarrow$ tanks
s6	NP $\rightarrow$ N	l7’	VP $\rightarrow$ tanks
s8	PP $\rightarrow$ P NP	l7”	S $\rightarrow$ tanks
s8’	PP $\rightarrow$ P	l8	P $\rightarrow$ with

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries 4–7 (Removal)

Structural rules		**Lexical rules**	
s1	S → NP VP	l1	N → people
s2	VP → V NP	l2	N → fish
s2”	S → V NP	l3	N → tanks
s3	VP → V NP PP	l4	N → rods
s3”	S → V NP PP	l5	V → people
s3’	VP → V PP	l5’	VP → people
s3”’	S → V PP	l5”	S → people
s4	NP → NP NP	l6	V → fish
s4’	NP → NP	l6’	VP → fish
s5	NP → NP PP	l6”	S → fish
s5’	NP → PP	l7	V → tanks
s6	NP → N	l7’	VP → tanks
s8	PP → P NP	l7”	S → tanks
s8’	PP → P	l8	P → with

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: Unaries 4–7 (Addition)

Structural rules		**Lexical rules**	
s1	S → NP VP	l1	NP → people
s2	VP → V NP	l2	NP → fish
s2”	S → V NP	l3	NP → tanks
s3	VP → V NP PP	l4	NP → rods
s3”	S → V NP PP	l5	V → people
s3’	VP → V PP	l5’	VP → people
s3”’	S → V PP	l5”	S → people
s4	NP → NP NP	l6	V → fish
s5	NP → NP PP	l6’	VP → fish
s5”	NP → P NP	l6”	S → fish
s8	PP → P NP	l7	V → tanks
		l7’	VP → tanks
		l7”	S → tanks
		l8	P → with
		l8’	PP → with
		l8”	NP → with

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: n-aries 1–2 (Removal)

Structural rules		Lexical rules	
s1	S → NP VP	l1	NP → people
s2	VP → V NP	l2	NP → fish
s2"	S → V NP	l3	NP → tanks
s3	VP → V NP PP	l4	NP → rods
s3"	S → V NP PP	l5	V → people
s3'	VP → V PP	l5'	VP → people
s3"'	S → V PP	l5"	S → people
s4	NP → NP NP	l6	V → fish
s5	NP → NP PP	l6'	VP → fish
s5"	NP → P NP	l6"	S → fish
s8	PP → P NP	l7	V → tanks
		l7'	VP → tanks
		l7"	S → tanks
		l8	P → with
		l8'	PP → with
		l8"	NP → with

(Probabilistic) Context-Free Grammars

Chomsky Normal Form Transformation Example: n-aries 1–2 (Addition)

Structural rules		Lexical rules	
s1	S $\rightarrow$ NP VP	l1	NP $\rightarrow$ people
s2	VP $\rightarrow$ V NP	l2	NP $\rightarrow$ fish
s2"	S $\rightarrow$ V NP	l3	NP $\rightarrow$ tanks
s3""	VP $\rightarrow$ V VP_V	l4	NP $\rightarrow$ rods
s3""'	VP_V $\rightarrow$ NP PP	l5	V $\rightarrow$ people
s3"""	S $\rightarrow$ V S_V	l5'	VP $\rightarrow$ people
s3"""'	S_V $\rightarrow$ NP PP	l5"	S $\rightarrow$ people
s3'	VP $\rightarrow$ V PP	l6	V $\rightarrow$ fish
s3"'	S $\rightarrow$ V PP	l6'	VP $\rightarrow$ fish
s4	NP $\rightarrow$ NP NP	l6"	S $\rightarrow$ fish
s5	NP $\rightarrow$ NP PP	l7	V $\rightarrow$ tanks
s5"	NP $\rightarrow$ P NP	l7'	VP $\rightarrow$ tanks
s8	PP $\rightarrow$ P NP	l7"	S $\rightarrow$ tanks
		l8	P $\rightarrow$ with
		l8'	PP $\rightarrow$ with
		l8"	NP $\rightarrow$ with

Phew, this now is in Chomsky Normal Form!

Probabilistic Context-Free Grammars

Definition

- A probabilistic context-free grammar (PCFG) is a CFG where each production rule is assigned a probablility.

PCFG (Σ, N, S, R, P)

P A probability function $R \rightarrow [0,1]$ from production rules to probabilities, such that

$$\forall U \in N : \sum_{(U \rightarrow V) \in R} P(U \rightarrow V) = 1$$

($\Sigma,\ N = N_{phr} \cup N_{pos},\ S,\ R = R_{phr} \cup R_{pos}$ as before)

Probabilities

- Trees: The probability $P(t)$ of a tree t is the product of the probabilities of the rules used to generate it.
- Strings: The probability $P(s)$ of a string s is the sum of the probabilities of the trees which yield s.

Probabilistic Context-Free Grammars

An example PCFG

Structural rules			**Lexical rules**		
s1	S $\rightarrow$ NP VP	1.0	l1	N $\rightarrow$ people	0.5
s2	VP $\rightarrow$ V NP	0.6	l2	N $\rightarrow$ fish	0.2
s3	VP $\rightarrow$ V NP PP	0.4	l3	N $\rightarrow$ tanks	0.2
s4	NP $\rightarrow$ NP NP	0.1	l4	N $\rightarrow$ rods	0.1
s5	NP $\rightarrow$ NP PP	0.2	l5	V $\rightarrow$ people	0.1
s6	NP $\rightarrow$ N	0.7	l6	V $\rightarrow$ fish	0.6
s7	PP $\rightarrow$ P NP	1.0	l7	V $\rightarrow$ tanks	0.3
			l8	P $\rightarrow$ with	1.0

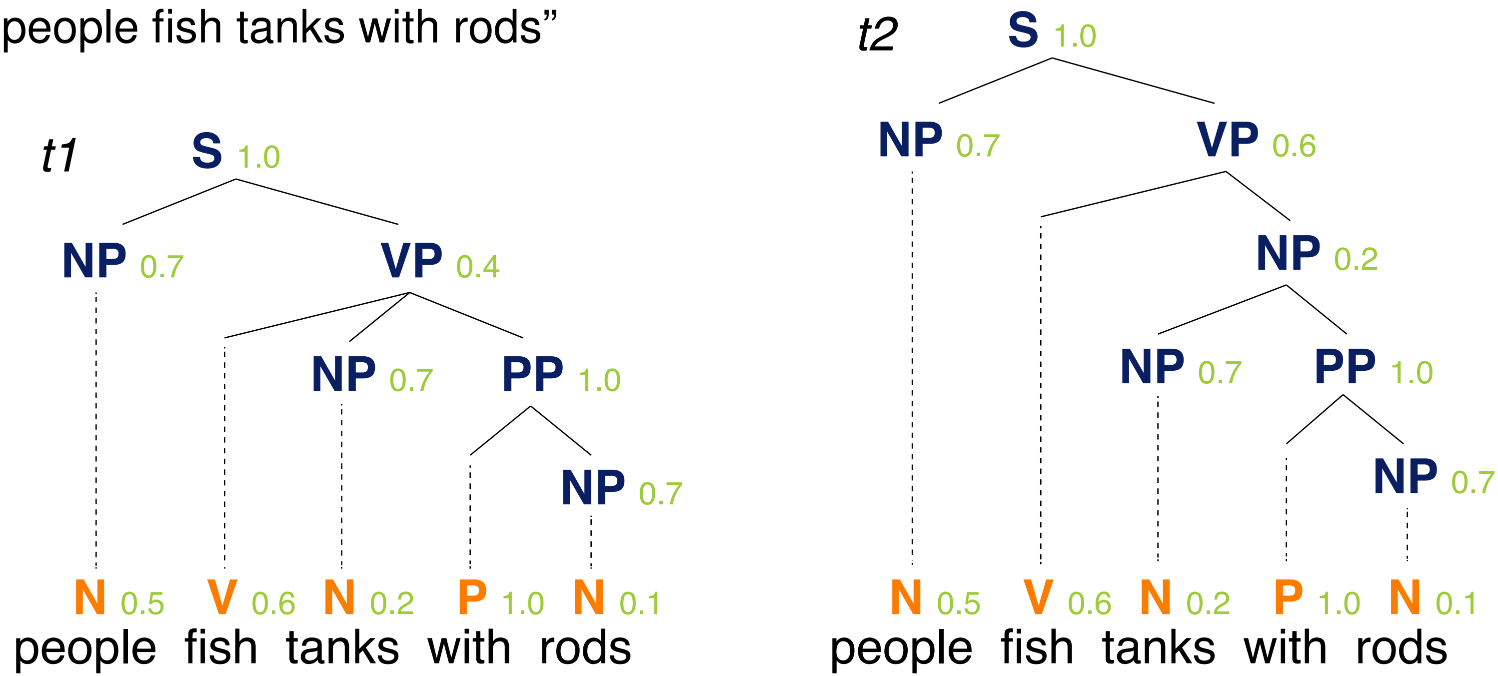
Notice

- For parsing, this PCFG should be transformed to Chomsky Normal Form or at least binarized.

Probabilistic Context-Free Grammars

Example Probabilities

$s =$ "people fish tanks with rods"



Probabilities

$$P(t_1) = 1.0 \cdot 0.7 \cdot 0.4 \cdot 0.5 \cdot 0.6 \cdot 0.7 \cdot 1.0 \cdot 0.2 \cdot 1.0 \cdot 0.7 \cdot 0.1 = 0.0008232$$

$$P(t_2) = 1.0 \cdot 0.7 \cdot 0.6 \cdot 0.5 \cdot 0.6 \cdot 0.2 \cdot 0.7 \cdot 1.0 \cdot 0.2 \cdot 1.0 \cdot 0.7 \cdot 0.1 = 0.00024696$$

$$P(s) = P(t_1) + P(t_2) = 0.0008232 + 0.00024696 = 0.00107016$$