LING/C SC/PSYC 438/538

Lecture 16

Sandiway Fong

Today's Topics

- Homework 9 review
- Prime number testing using Perl regex

Homework 9 Review

- First, notice I said you may assume the patterns:
  - the *noun*$_1$ *verb* the *noun*$_2$ ➠ *verb*(*noun*$_1$, *noun*$_2$)
  - the *noun*$_1$ who *verb* the *noun*$_2$ ➠ *verb*(*noun*$_1$, *noun*$_2$)
- Perl regex patterns:
  - `/the (\w+) (\w+) the (\w+)/` ➠ `print "$2($1, $3)"`
  - `/the (\w+) who (\w+) the (\w+)/` ➠ `print "$2($1, $3)"`
- Perl regex patterns:
  - `perl -le '$_ = qq/@ARGV/; /the (\w+) (\w+) the (\w+)/; print "$2($1, $3)"' the woman saw the boy who saw the girl`
  - `saw(woman, boy)`
  - `perl -le '$_ = qq/@ARGV/; /the (\w+) who (\w+) the (\w+)/; print "$2($1, $3)"' the woman saw the boy who saw the girl`
  - `saw(boy, girl)`

Homework 9 Review

- Next thing to notice is the regex overlap:
  - `the woman noticed the boy who saw the girl who found the man`
  - `the (\w+) (\w+) the (\w+)`
  - `the (\w+) who (\w+) the (\w+)`
  - `the (\w+) who (\w+) the (\w+)`
- Recall regex matching goes from left to right, moving a pointer.
- We want to iterate this matching using the g (global) flag
- One solution is simply to make the overlapping part a lookahead so the pointer is not advanced
  - i.e. `(?=the (\w+))`

Homework 9 Review

Example:

- `perl -le '$_ = qq/@ARGV/; while (/the (\w+) (who )?(\w+) (?=the (\w+))/g) {print "$3($1, $4)"}' the woman saw the boy who saw the girl`
- `saw(woman, boy)`
- `saw(boy, girl)`

You can also use a non- capturing group
`(?:regexp)`

Example:

- `perl -le '$_ = qq/@ARGV/; while (/the (\w+) (who )?(\w+) (?=the (\w+))/g) {print "$3($1, $4)"}' the woman saw the boy`
- `saw(woman, boy)`

Homework 9 Review

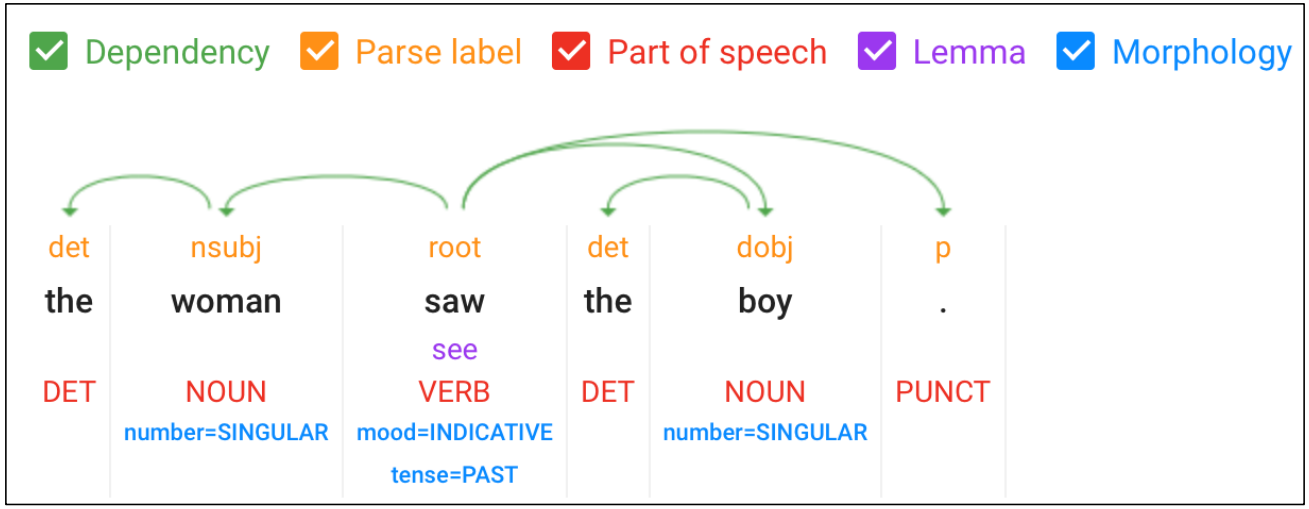
Example:

- `perl -le '$_ = qq/@ARGV/; while (/the (\w+) (who )?(\w+) (?=the (\w+))/g) {print "$3($1, $4)"}' the woman saw the boy who saw the girl who found the man`
- `saw(woman, boy)`
- `saw(boy, girl)`
- `found(girl, man)`

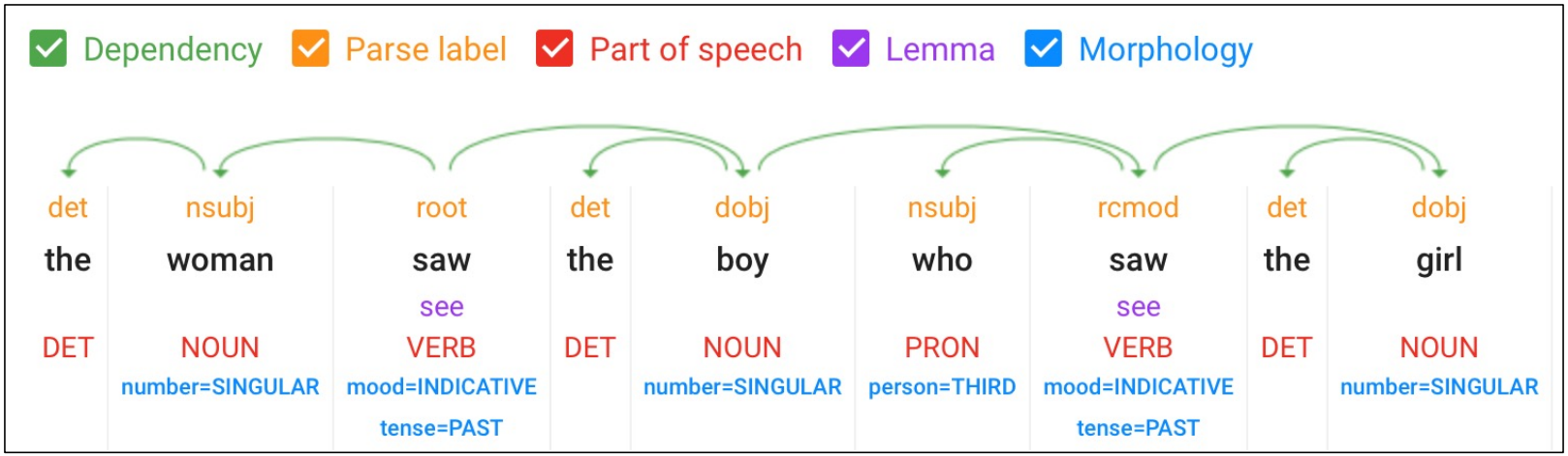
Example:

- `perl -le '$_ = qq/@ARGV/; while (/the (\w+) (who )?(\w+) (?=the (\w+))/g) {print "$3($1, $4)"}' the woman saw the boy who saw the girl who found the man who chased the cat`
- `saw(woman, boy)`
- `saw(boy, girl)`
- `found(girl, man)`
- `chased(man, cat)`

https://cloud.google.com/natural-language

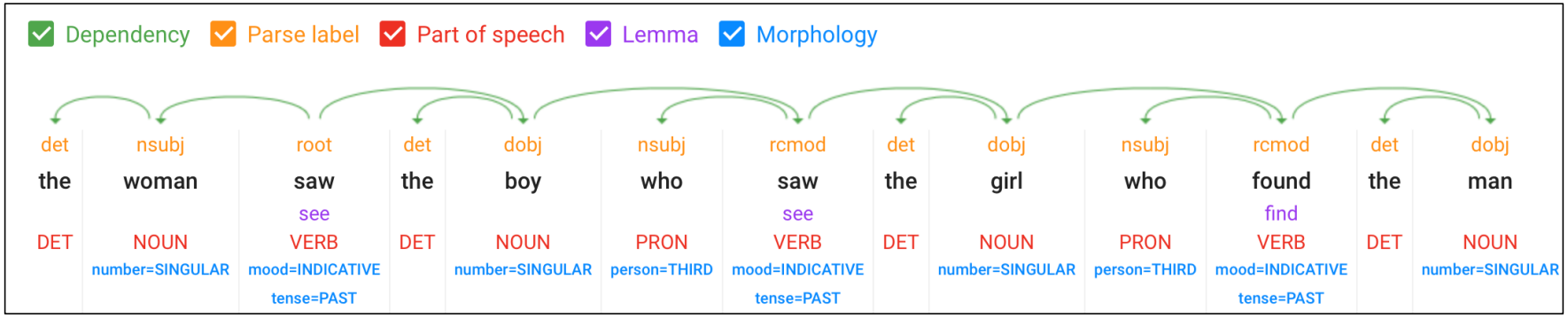


https://cloud.google.com/natural-language

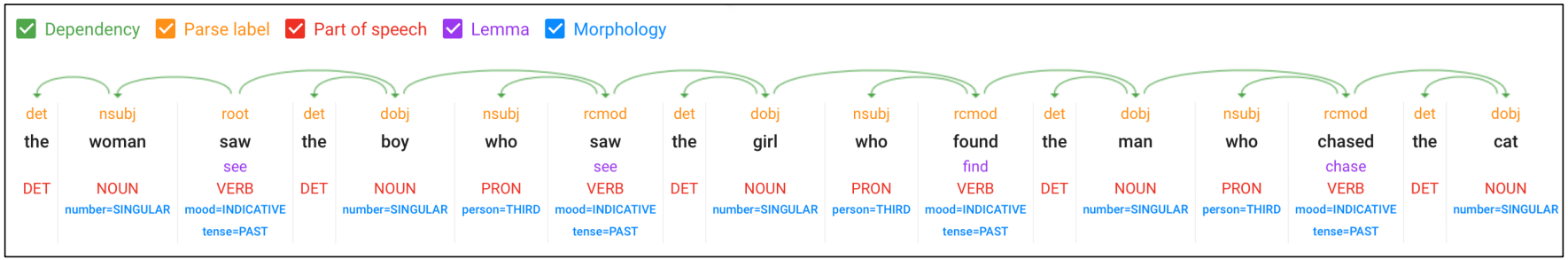


RCMOD points back to NOUN
VERB(PRON, DOBJ)
VERB(NOUN, DOBJ)
saw(boy, girl)

https://cloud.google.com/natural-language



https://cloud.google.com/natural-language



Homework 9 Review

- Part 1 (`nsubj` relativization):
  - the woman saw the boy who saw the girl (*relative pronoun is obligatory*)
  - `saw(woman, boy) saw(boy, girl)`
  - the woman saw the boy who saw the girl who found the man
  - `saw(woman, boy) saw(boy, girl) found(girl, man)`
- Part 2 (`dobj` relativization):
  - the woman sensed the boy the girl saw
  - the woman sensed the boy who the girl saw (*relative pronoun is optional*)
  - `saw(woman, boy) saw(girl, boy)` boy is the implict `dobj`
  - the woman sensed the boy the girl the man found saw
  - the woman sensed the boy who the girl who the man found saw
  - `sensed(woman, boy) saw(girl, boy) found(man, girl)`

Homework 9 Review

Center-embedding (*distance problem*):

2. the woman sensed the boy the girl saw

   the woman sensed [the boy_{DOBJ} the $\text{girl}_{\text{NSUBJ}}$ $\text{saw}_{(\text{girl, boy})}$]

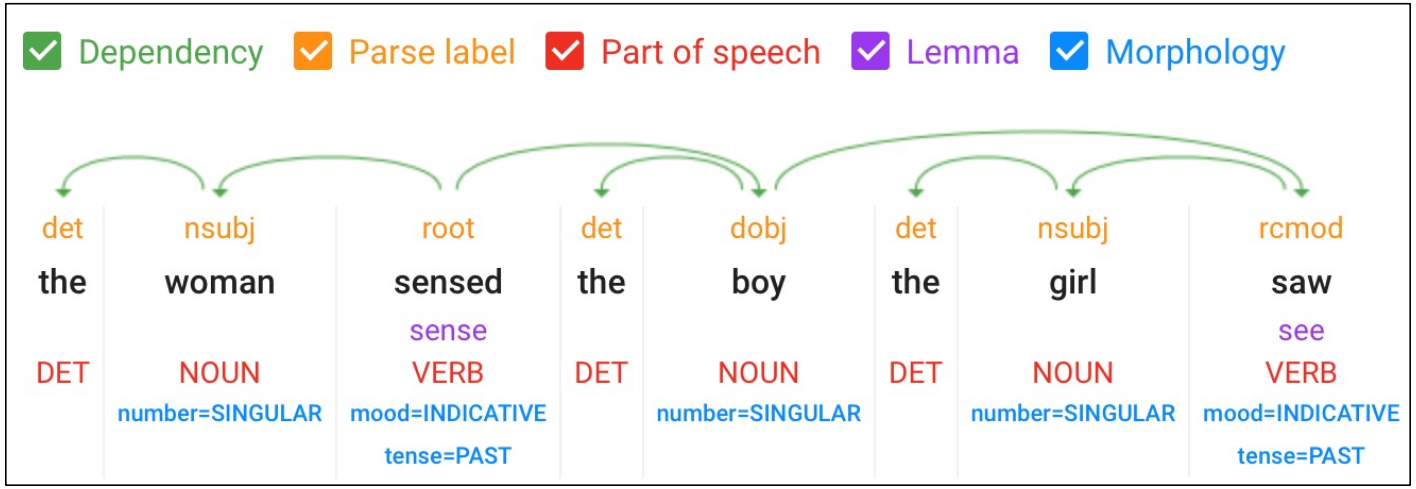
3. the woman sensed the boy the girl the man found saw

   the woman sensed [the boy_{DOBJ} [the $\text{girl}_{\text{DOBJ}}$ the $\text{man}_{\text{NSUBJ}}$ $\text{found}_{(\text{man, girl})}$] $\text{saw}_{(\text{girl, boy})}$]

4. the woman sensed the boy the girl the man the cat chased found saw

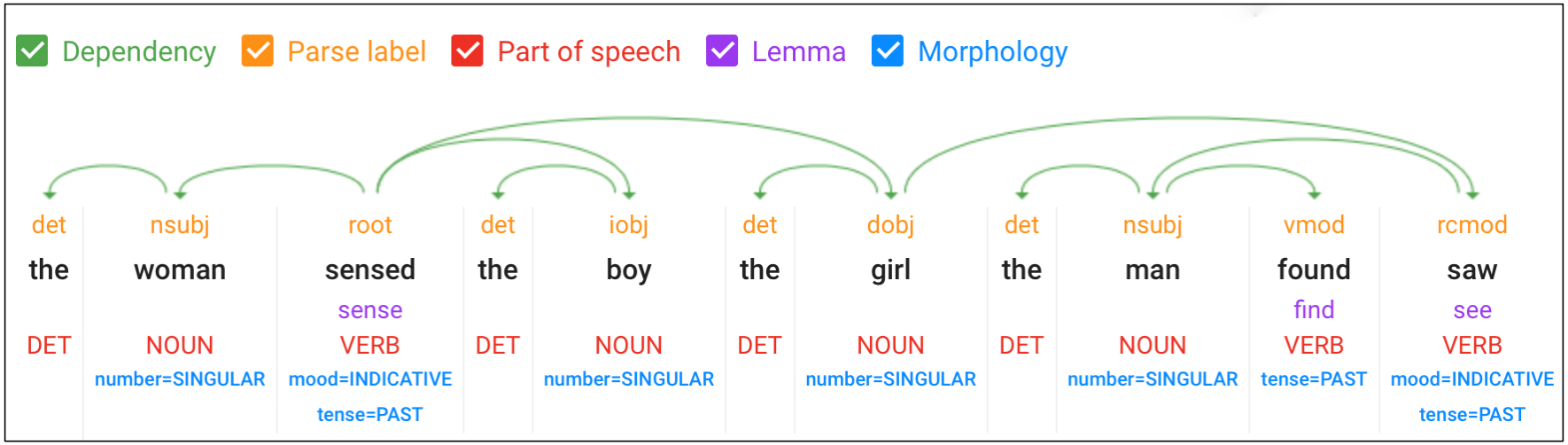
   the woman sensed [the boy [the girl [the man_{DOBJ} the $\text{cat}_{\text{NSUBJ}}$ $\text{chased}_{(\text{cat, man})}$] $\text{found}_{(\text{man, girl})}$] $\text{saw}_{(\text{girl, boy})}$]

https://cloud.google.com/natural-language



RCMOD points back to NOUN
VERB(NSUBJ, *missing* DOBJ)
VERB(NSUBJ, NOUN)
saw(girl, boy)

https://cloud.google.com/natural-language



https://cloud.google.com/natural-language

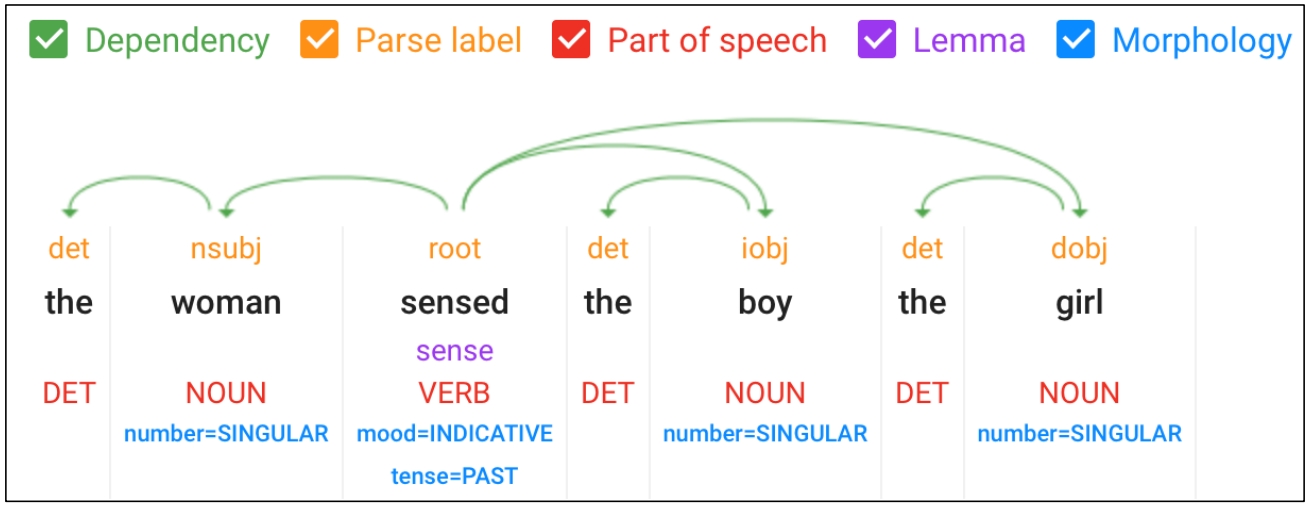
https://downloads.cs.stanford.edu/nlp/software/dependencies_manual.pdf

***vmod*: reduced non-finite verbal modifier**

A reduced non-finite verbal modifier is a participial or infinitive form of a verb heading a phrase (which may have some arguments, roughly like a VP). These are used to modify the meaning of an NP or another verb. They are not core arguments of a verb or full finite relative clauses.

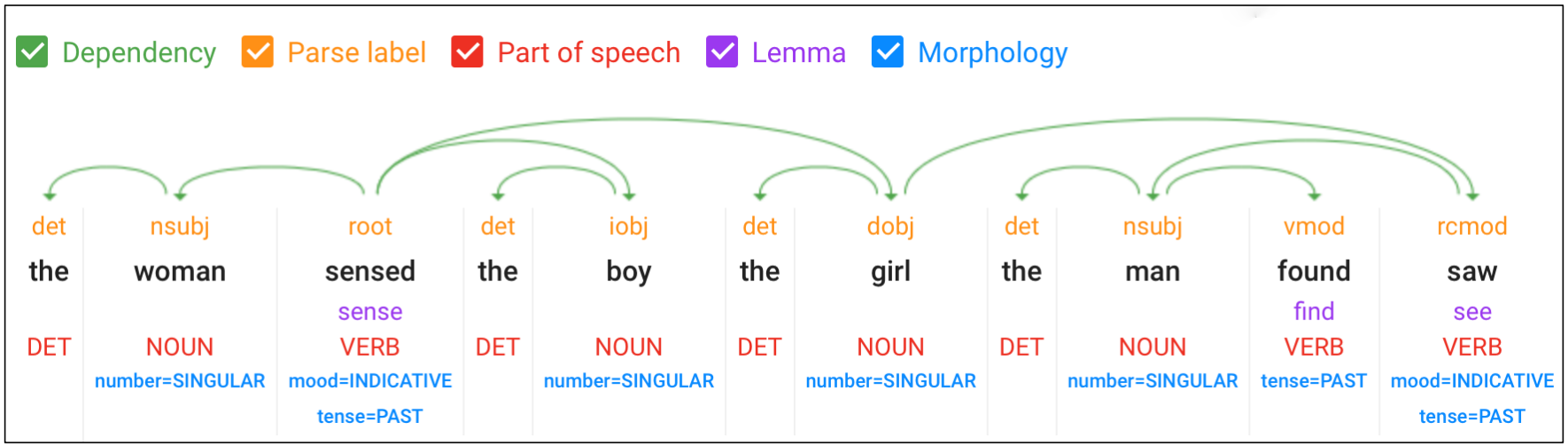
"Points to establish are . . ."	*vmod*(points, establish)
"I don't have anything to say to you"	*vmod*(anything, say)
"Truffles picked during the spring are tasty"	*vmod*(truffles, picked)
"Bill tried to shoot, demonstrating his incompetence"	*vmod*(shoot, demonstrating)

https://cloud.google.com/natural-language



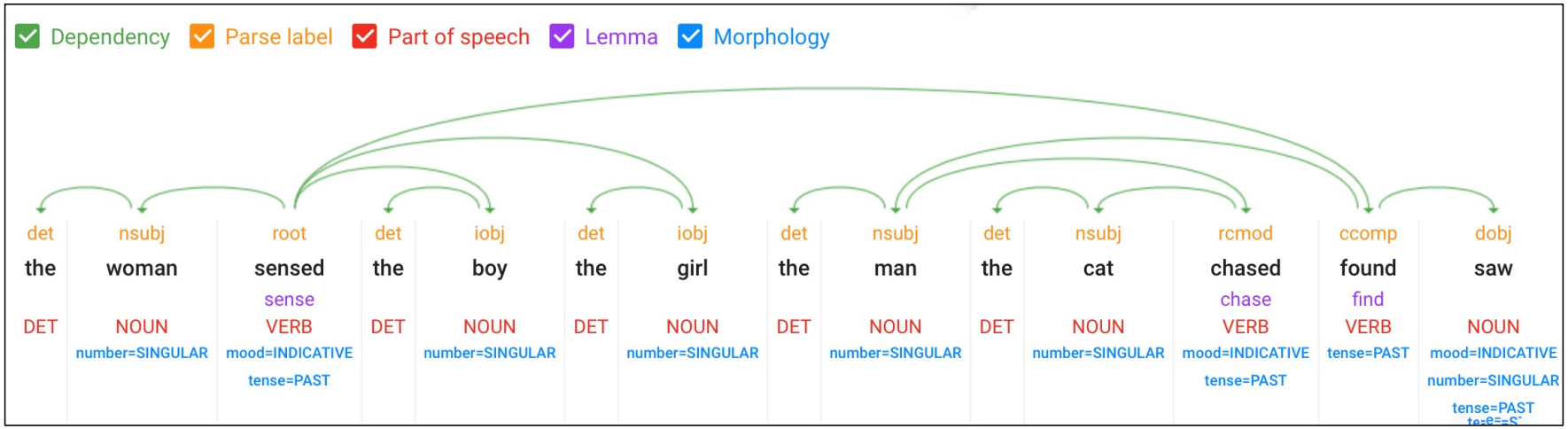
sensed(NSUBJ, ***IOBJ***, DOBJ)

https://cloud.google.com/natural-language



- sensed(woman, ***boy***, girl)
- saw(man, girl)
- "the found man"

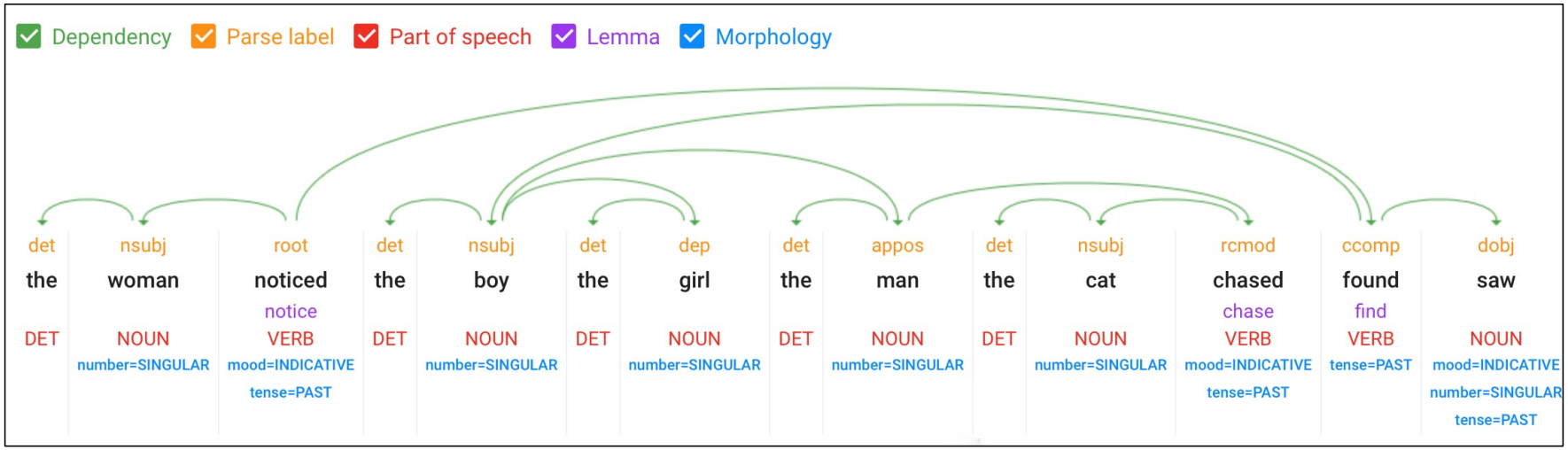
https://cloud.google.com/natural-language



- found(man, saw)
- sensed(woman, boy, girl, found(…))
- chased(cat, man)

https://cloud.google.com/natural-language

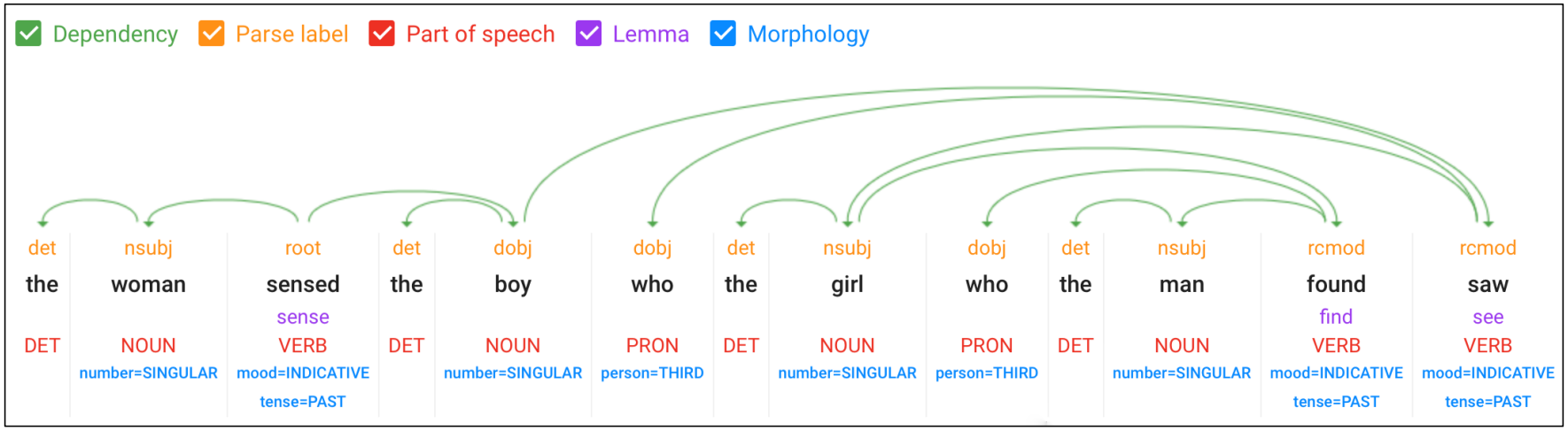
Note: with a slightly different matrix verb *noticed* (vs. *sensed*), a completely different structure



- found(boy, saw)
- noticed(woman, found(…))
- chased(cat, man)
- boy, the man (the cat chased), girl

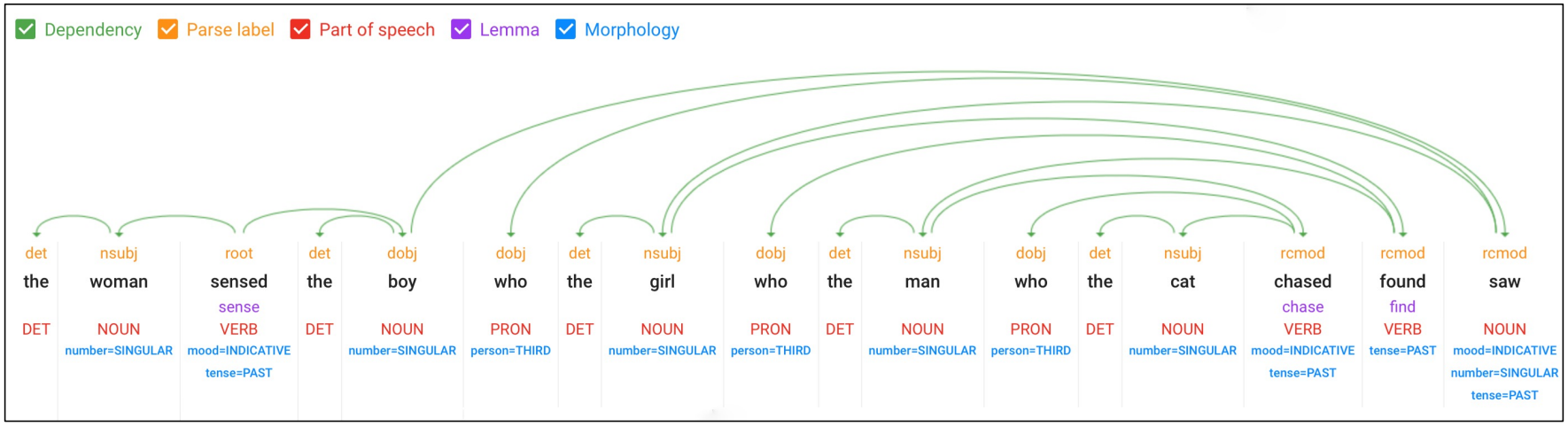
https://cloud.google.com/natural-language

- object relative clause with the *wh*-relative pronoun



https://cloud.google.com/natural-language

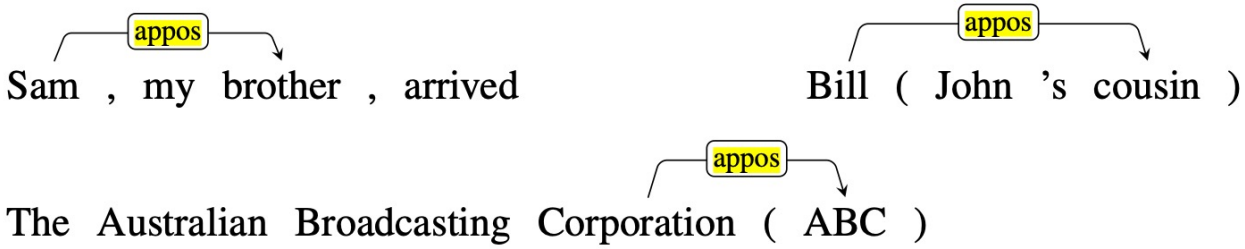
- embedding/nesting plainly visible



https://cloud.google.com/natural-language

***appos*: appositional modifier**

An appositional modifier of an NP is an NP immediately to the right of the first NP that serves to define or modify that NP. It includes parenthesized examples, as well as defining abbreviations in one of these structures.



https://cloud.google.com/natural-language

***dep*: dependent**

A dependency is labeled as *dep* when the system is unable to determine a more precise dependency relation between two words. This may be because of a weird grammatical construction, a limitation in the Stanford Dependency conversion software, a parser error, or because of an unresolved long distance dependency.

Prime Number Testing using Perl Regular Expressions

- Another example:
  - the set of prime numbers is **not** a **regular** language (*can't do It with FSA/regex*)
  - L_{prime} = {2, 3, 5, 7, 11, 13, 17, 19, 23,.. }

Prime number - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Prime_number**
A **prime number** (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. A natural number greater than 1 that is not a ...

Turns out, we can use a short Perl regex to determine membership in this set .. and to factorize numbers

```
/^(11+?)\1+$/
```

Prime Number Testing using Perl regex

- L = {1^n | n is prime} is not a **regular** language

can be proved mathematically using the Pumping Lemma for regular languages (*later*)

- Keys to making this work:
  - \1 backreference
  - unary notation for representing numbers, e.g.
    - 11111 “five ones” = 5
    - 111111 “six ones” = 6
  - unary notation allows us to factorize numbers by repetitive pattern matching
    - (11)(11)(11) “six ones” = 6
    - (111)(111) “six ones” = 6
  - numbers that can be factorized in this way aren’t prime!
    - no way to get nontrivial subcopies of 11111 “five ones” = 5
- Then /^(11+?)**\1**+$/ will match anything that’s greater than 1 that’s not prime

Prime Number Testing using Perl regex

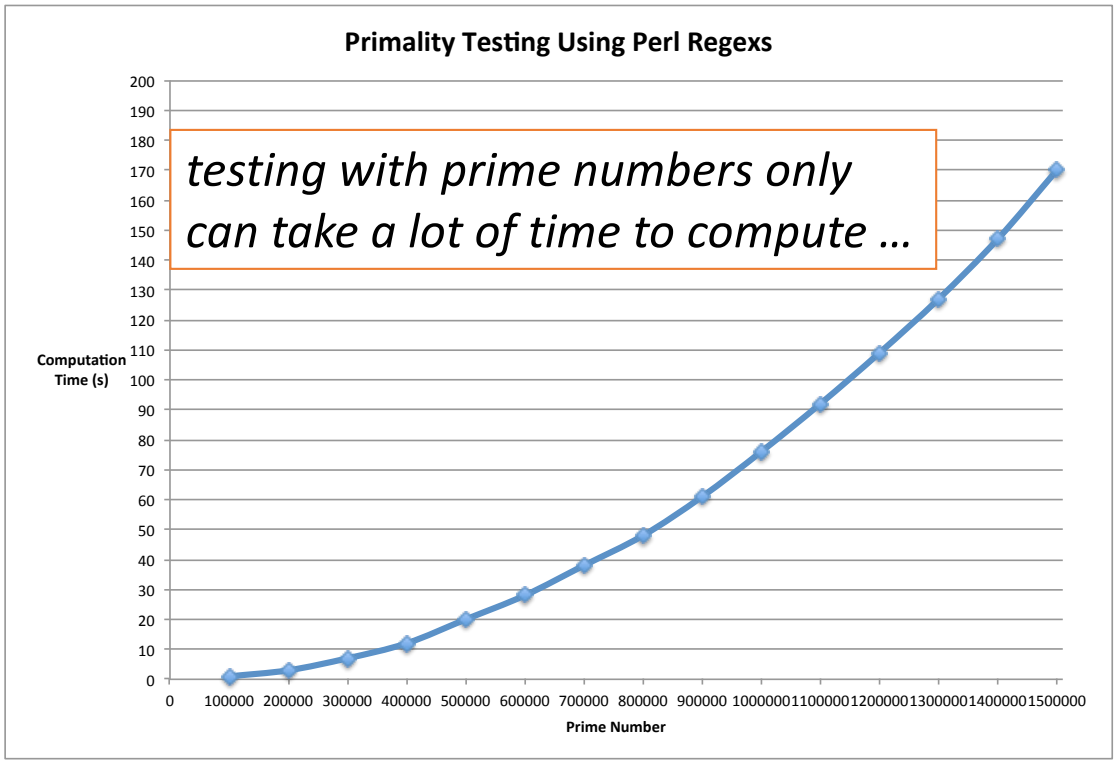
Question: is the non-greedy operator necessary?

- Let's analyze this Perl regex `/^(11+?)\1+$/`
  - `^` and `$` anchor both ends of the strings, forces `(11+?)\1+` to cover the entire string
  - `(11+?)` is the non-greedy (*shortest*) match version of `(11+)`
  - `\1+` provides one or more copies of what we previously matched in `(11+?)`
- Examples:

```
perl –le '$n = shift; $u = "1" x $n; print "$n prime" if $u !~
/^(11+?)\1+$/' 101
101 prime
perl –le '$n = shift; $u = "1" x $n; print "$n prime" if $u !~
/^(11+?)\1+$/' 103
103 prime
perl –le '$n = shift; $u = "1" x $n; print "$n prime" if $u !~
/^(11+?)\1+$/' 105
```

Prime Number Testing using Perl regex

Prime Numbers
100003
200003
300007
400009
500009
600011
700001
800011
900001
1000003
1100009
1200007
1300021
1400017
1500007



Prime Number Testing using Perl regex

- `/^(11+?)\1+$/` vs. `/^(11+)\1+$/`
- i.e. non-greedy vs. greedy matching
- finds smallest factor vs. largest
  - `90021 factored using 3, not a prime` (0 secs)

  vs.
  - `90021 factored using 30007, not a prime` (0 secs)

Puzzling behavior: same output non-greedy vs. greedy
`900021 factored using 300007, not a prime` (48 secs vs. 13 secs)

Prime Number Testing using Perl regex

- http://www.xav.com/perl/lib/Pod/perlre.html

The following standard quantifiers are recognized:

```
*      Match 0 or more times
+      Match 1 or more times
?      Match 1 or 0 times
{n}    Match exactly n times
{n,}   Match at least n times
{n,m}  Match at least n but not more than m times
```

(If a curly bracket occurs in any other context, it is treated as a regular character.) The ``*" modifier is equivalent to {0,}, the ``+" modifier to {1,}, and the ``?" modifier to {0,1}. n and m are limited to integral values less than a preset limit defined when perl is built. This is usually 32766 on the most common platforms.

nearest primes to preset limit

3*32749 = 98247 | 32766 | 3*32771 = 98313

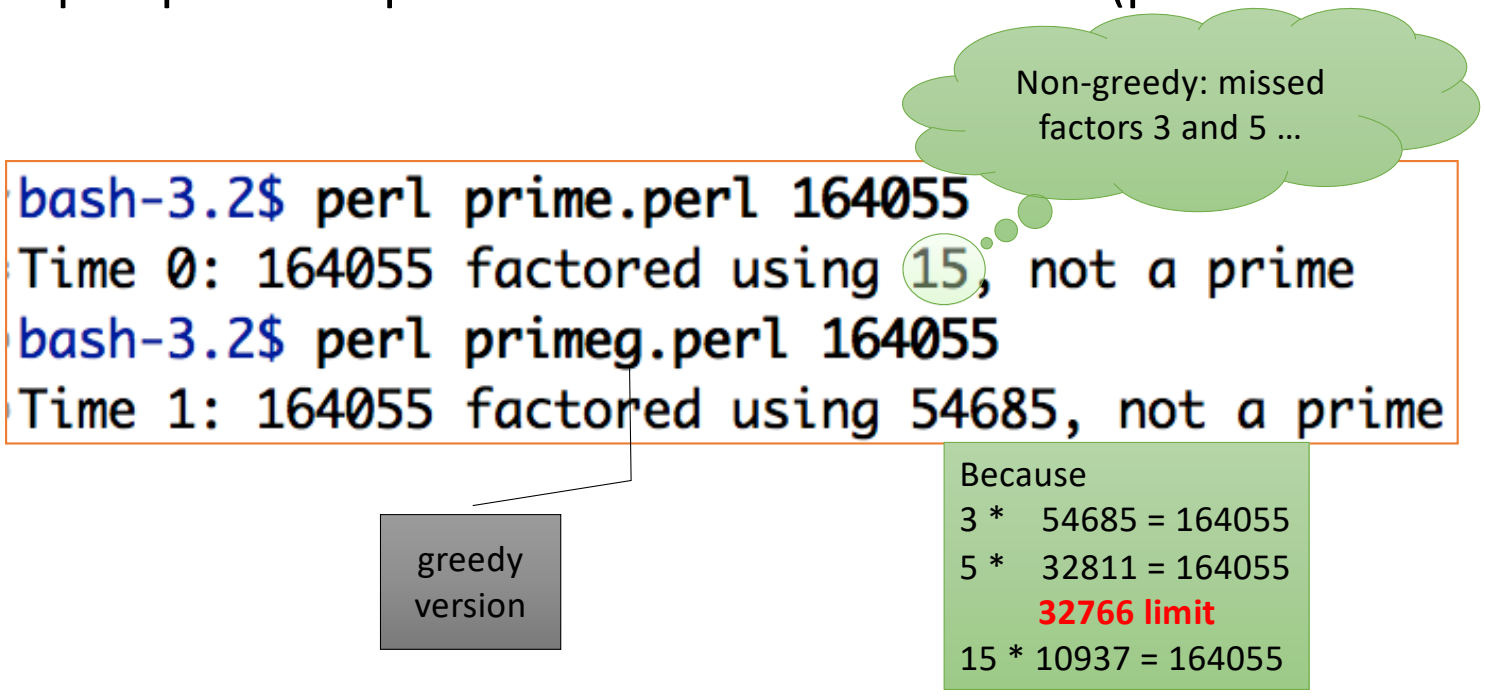
Prime Number Testing using Perl regex

- 32749 x 3 = 98247
- 32771 x 3 = 98313
- *When preset limit is exceeded: Perl's regex matching fails quietly*
- **Why does it report 32771?**

```
bash-3.2$ perl prime.perl 98247
Time 0: 98247 factored using 3, not a prime
bash-3.2$ perl prime.perl 98313
Time 1: 98313 factored using 32771, not a prime
```

Prime Number Testing using Perl Regular Expressions

- Can also get non-greedy to skip several factors
- Example: pick non-prime 164055 = 3 x 5 x 10937 (prime factorization)



Prime Number Testing using Perl Regular Expressions

- Results are still right so far though:
  - *wrt*. prime vs. non-prime
- But we predict it will report an incorrect result for
  - `1,073,938,441`
  - It should claim (incorrectly) that this is prime since $1073938441 = 32771^2$
  - (32766 is the limit for the number of bundles)

32611	32621	32633	32647	32653	32687	32693	32707	32713	32717
32719	32749	32771	32779	32783	32789	32797	32801	32803	32831
32833	32839	32843	32869	32887	32909	32911	32917	32933	32939

https://primes.utm.edu/lists/small/10000.txt