LING/C SC/PSYC 438/538

Lecture 25

Sandiway Fong

Adminstration

538 Presentations

- Homework 12 out today!
  - Due day Sunday midnight

538 Presentations

Name	Presentation	Date
Alkuraydis,Ahmed		
Barner,Jacob Ryan		
Bejarano,Cielo S		
Bell,Jack T		
Converse,Amber Charlotte	8.4 HMM Part-of-Speech Tagging	7
Cox,Samantha Ann	18 Word Senses and WordNet, 18.1-18.3	7
Davis,Katherine Nicole		
Dharmala,Bayu	12.3 Some Grammar Rules for English	7
Hopper,Ashlyn Danielle		
Jain,Varshit Chirag	23 Question Answering, 23.4-23.6	5
Kankia,Kevin Pinakin		
Kleczewski,Alison	15.4 Event and State Representations, 15.4, 15.4.1, 15.4.2	7
LaScola Ochoa,Logan Michelle		
Logan,Haley Brooke	21.1 Coreference Phenomena: Linguistic Background	5
Maibach,Marcus Wile		
Mangkang,Tinnawit		
Mangla,Sourav	3 N-gram Language Models, 3.1,3.4	5
McLaughlin,Matthew	15.2 Model-theoretic Semantics	5
Mehta,Deep Paresh	17 Extracting Times and Events, 17.3-17.4	5
Mendoza,Freddy		
Murphy III,Michael LaMotte		
Pinto,Aayush Bernard	20 Lexicons for Sentiment, Affect, and Connotation, 20.1-20.3	5
Pipatanangkura,Leighanna D	23.2 IR-based Factoid Question Answering	5
Raju,Anish	Lexical and Vector Semantics, 6.1-6.2	7
Reeve,Keegan Austin	Coreference Resolution: Mention and Architectures 21.3-21.4	7
Ruparel,Deep Anil	6 : Vector Semantics and Embeddings, 6.3-6.5	5
Shakyam Shreya Nupur	24.2 Chatbots	7
Shu,Qiyu		
Shukla,Kartikey	23.1 Information Retrieval	5
Thompson,Brendan S		
Warrick,Baylor M		
West,Georgia Alexandra		
Willittes,Taylor		
Yuan,Alan		

538 Presentations

- Slides due to me to in Powerpoint or PDF format:
  - midnight before your presentation
  - we will use my laptop (*no switching of laptops*)

Last Time

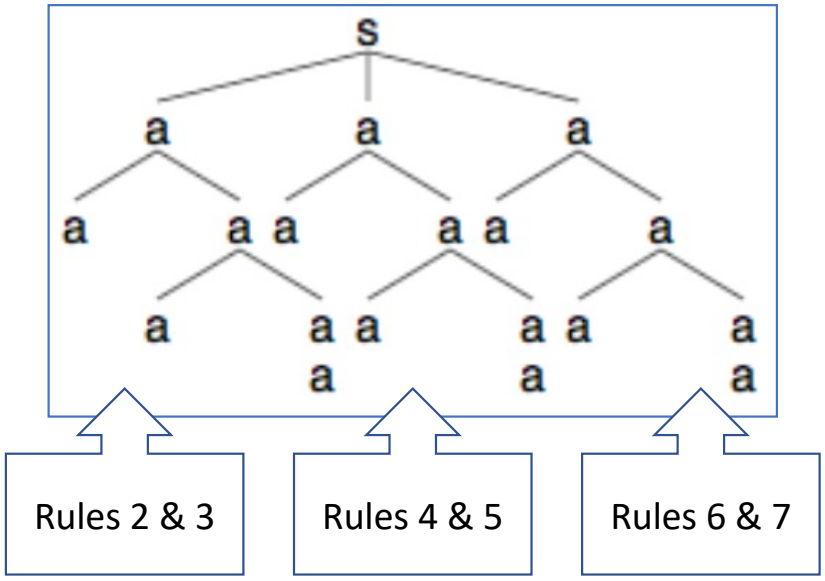
- Extra arguments are powerful
  - they allow us to impose (grammatical) constraints and change the expressive power of the system
    - if used as read-able memory (cf. *Turing Machine discussion*)

- **Example**:
  - $a^n b^n c^n$ n>0 is not a context-free language (type-2)
  - *i.e. you cannot write rules of the form* n –––> RHS *to generate this language*
  - in fact, it's context-sensitive (type-1)

Extra arguments

- A context-free grammar (CFG) + extra argument (EA) for the context-sensitive language { $a^n b^n c^n$ | n>0}:

```
1. s(s(A,A,A)) --> a(A), b(A), c(A).
2. a(a(a)) --> [a].
3. a(a(a,X)) --> [a], a(X).
4. b(a(a)) --> [b]. % cf. b(b)
5. b(a(a,X)) --> [b], b(X).
6. c(a(a)) --> [c]. % cf. c(c)
7. c(a(a,X)) --> [c], c(X).
```

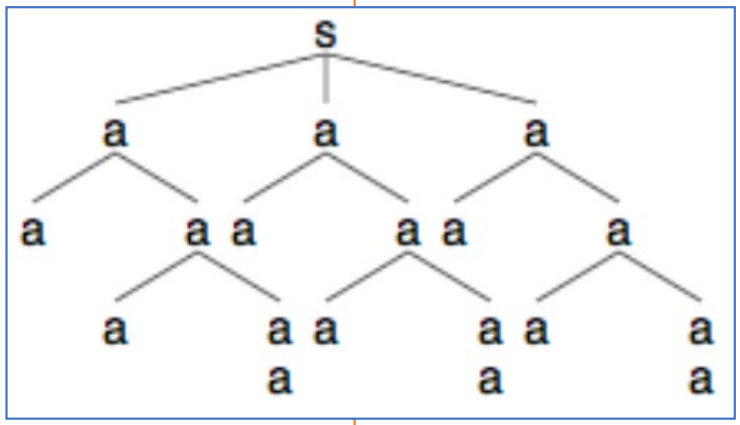


Extra arguments

- A CFG+EA for $a^n b^n c^n$ n>0:

Set membership question

```
[?- [abc_parse].
true.

[?- s(Parse,[a,a,a,b,b,b,c,c,c],[]).
Parse = s(a(a, a(a, a(a))), a(a, a(a, a(a))), a(a, a(a, a(a)))) ;
false.

[?- s(Parse,[a,a,a,b,b,b,c,c],[]).
false.

[?- s(Parse,[a,a,a,b,b,c,c,c],[]).
false.

[?- s(Parse,[a,a,b,b,b,c,c,c],[]).
false.

?-
```



Extra arguments

- A CFG+EA for $a^n b^n c^n$ n>0:

```
?- s(_,[a,a,b,b,c,c,c],[]).
false.

?- s(_,[a,a,b,b,c,c],[]).
true .

?- s(_,[a,a,b,b,c],[]).
false.

?- s(_,[a,a,b,b,c,c,c],[]).
false.

?- s(_,[a,a,a,b,b,b,c,c,c],[]).
true .
```

Set membership question

Context-sensitive Grammar (CSG)

- Type-1:
  - **note**: more than one symbol on the LHS of the rule
- `s --> [a,b,c].`
- `s --> [a],a,[b,c].`
- `a --> [a,b], c.`
- `a --> [a],a,[b],c.`
- `c,[b] --> [b], c.`
- `c,[c] --> [c,c].`

Last Time

- Grammar rule transformation:
  - *produces a parse tree*
  - **Idea**: each nonterminal can take an structural argument

```
s --> [b], b.
```

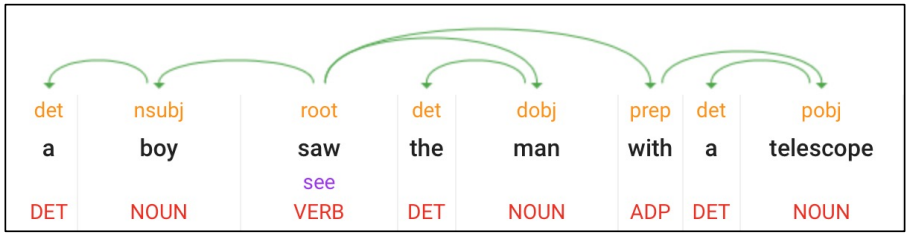
⇩

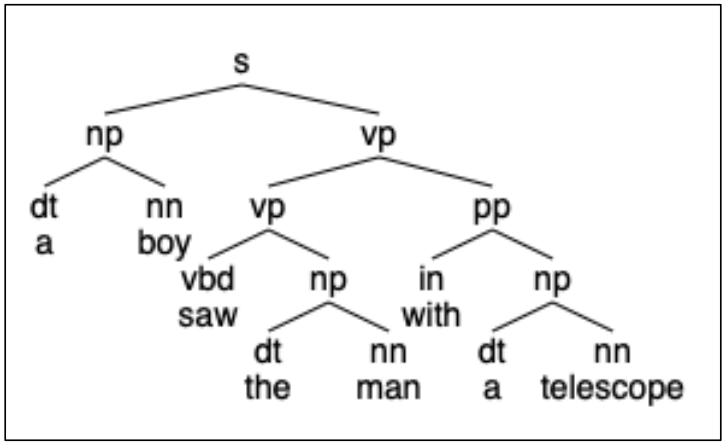
```
s(Tree) --> [b], b(SubTree).
```

⇩

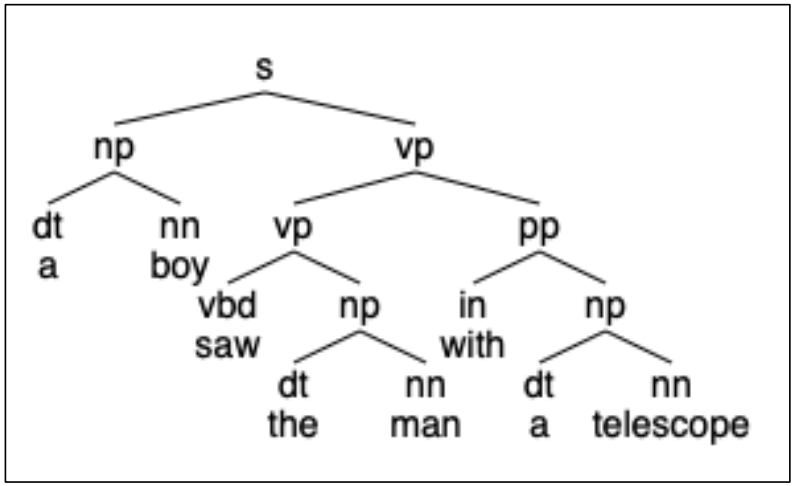
```
s(s(b,Tree)) --> [b], b(Tree).
```

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





Tree ⇨ Grammar



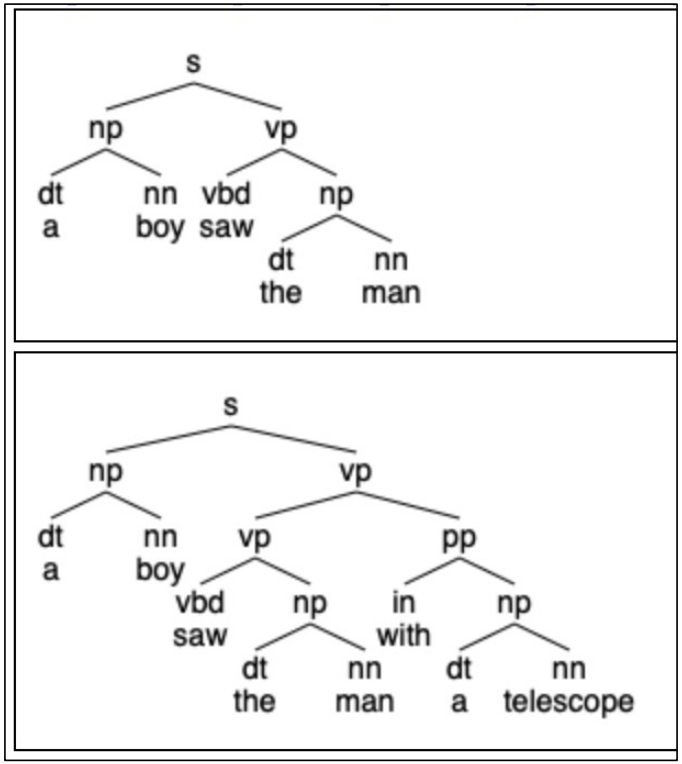
- Context-free Grammar (CFG) `g1.prolog`:

```
1.  s --> np, vp.
2.  np --> dt, nn.
3.  dt --> [a].
4.  nn --> [boy].
5.  vp --> vbd, np.
6.  vbd --> [saw].
7.  dt --> [the].
8.  nn --> [man].
9.  vp --> vp, pp. % left recursive: tricky!
10. pp --> in, np.
11. in --> [with].
12. nn --> [telescope].
```

Homework 12: Question 1

- Transform `g1.prolog` into `g1tree.prolog` so it can parse.
- Examples:
  - *hit* [return] *instead of* ; (*tricky left recursion*)

```
?- s(Tree, [a, boy, saw, the, man], []).

Tree = s(np(dt(a), nn(boy)), vp(vbd(saw),
np(dt(the), nn(man)))) .

 ?- s(Tree, [a, boy, saw, the, man, with,
a, telescope], []).

Tree = s(np(dt(a), nn(boy)),
vp(vp(vbd(saw), np(dt(the), nn(man))),
pp(in(with), np(dt(a), nn(telescope))))) .
```



Homework 12: Question 1

- You can ignore warnings about discontiguous grammar rules:

```
Warning:    Clauses of dt/3 are not together in the source-file
Warning:    Earlier definition at
/Users/sandiway/courses/538/ling538-22/g1parse.prolog:3
Warning:    Current predicate: vbd/3
Warning:    Use :- discontiguous dt/3. to suppress this message
```

Left recursion

- Left recursion problem:
  - default: Prolog expands rule RHS left to right.
- g1.prolog:

```
1.   s --> np, vp.
2.   np --> dt, nn.
3.   dt --> [a].
4.   nn --> [boy].
5.   vp --> vbd, np.
6.   vbd --> [saw].
7.   dt --> [the].
8.   nn --> [man].
9.   vp --> vp, pp. % left recursive: tricky!
10.  pp --> in, np.
11.  in --> [with].
12.  nn --> [telescope].
```

① ②

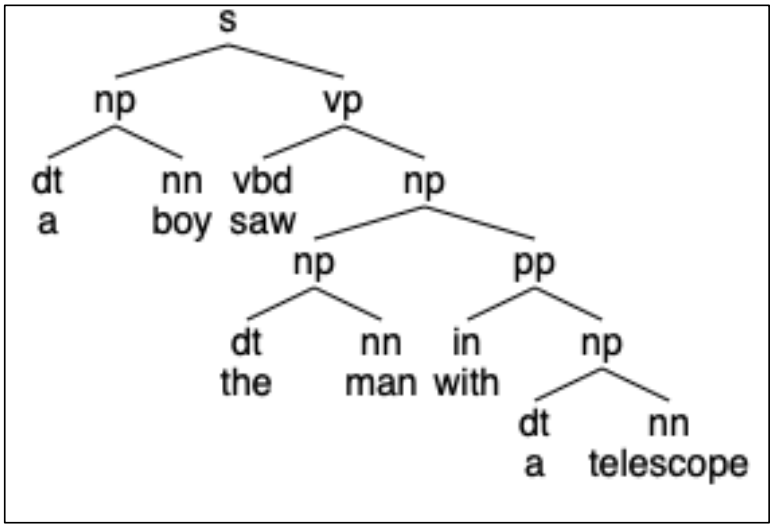
- Example:

```
?- s(Tree, [a, boy, saw, the, man], []).
Tree = s(np(dt(a), nn(boy)), vp(vbd(saw),
np(dt(the), nn(man)))) ;
ERROR: Stack limit (1.0Gb) exceeded
…
   Exception: (7,668,435) vp(_46027412, [saw, the,
man], _46027846) ?
```

- Problem can be fixed:
  - there is a way to transform left recursive rules into right recursive rules that compute left recursive parse trees.
  - *see next semester …*

Homework 12: Question 2

- The parse that Google Natural Language didn't get…



- What new grammar rule does this parse add to `g1.prolog`?
- Currently, your parser only obtains:

```
?- s(Tree, [a, boy, saw, the, man,
with, a, telescope], []).

Tree = s(np(dt(a), nn(boy)),
vp(vp(vbd(saw), np(dt(the), nn(man))),
pp(in(with), np(dt(a),
nn(telescope))))) .
```

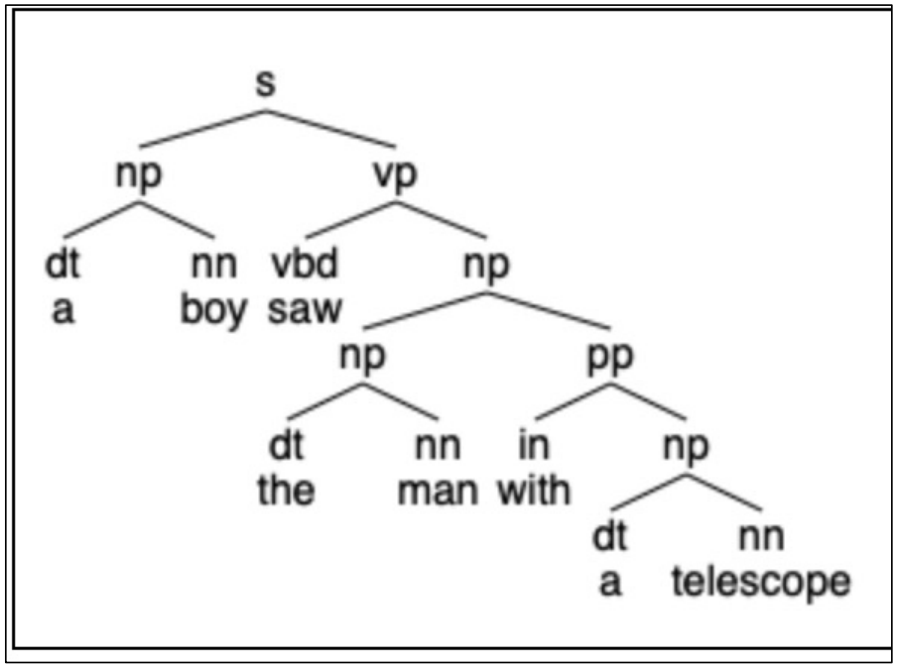
- Add it (*plus the transform*).
- Show it working.

Homework 12: Question 2

- Your program should produce the following output, e.g.:
  - *hit* [return] *instead of* ; (*tricky left recursion again*)

```
?- s(Tree, [a, boy, saw, the, man, with,
a, telescope], []).

Tree = s(np(dt(a), nn(boy)), vp(vbd(saw),
np(np(dt(the), nn(man)), pp(in(with),
np(dt(a), nn(telescope)))))) .
```

- Your program should be able to produce either parse by positioning the rules in order.
  - **recall** Prolog finds 1st matching rule



Homework 12: Question 2

- You can ignore warnings about redefinitions:

```
Warning: /Users/sandiway/courses/538/ling538-22/g2parse.prolog:1:
Warning:    Redefined static procedure s/3
Warning:    Previously defined at /Users/sandiway/courses/538/ling538-
22/g1parse.prolog:1
Warning: /Users/sandiway/courses/538/ling538-22/g2parse.prolog:2:
Warning:    Redefined static procedure np/3
Warning:    Previously defined at /Users/sandiway/courses/538/ling538-
22/g1parse.prolog:2
```

Homework 12: Question 3

- What happens when you try:

```
?- s(Tree, [a, boy, with, a, telescope, saw, the, man], []).
```

- on your answer to Q2.
- Can you explain why?

The Left Corner idea

- The left recursive grammar rule:
  - `vp(vp(X,Y)) --> vp(X), pp(Y). % left recursive: tricky!`
- is transformed into a Prolog rule (*when loaded*):
  - `see listing(vp).`
  - `vp(vp(X, Y), A, B) :- vp(X, A, C), pp(Y, C, B).`
- A, B and C are variables representing list of words.
- `vp(X, A, C)` *means there is a VP between lists* A *and* C, e.g.
  - `?- vp(X, [saw, the, man, with, a, telescope], C).`
  - `X = vp(vbd(saw), np(dt(the), nn(man))),`
  - `C = [with, a, telescope] .`
  - between `[saw, the, man, with, a, telescope]`
  - `[with, a, telescope]`
  - we have `[saw, the, man]` as the difference between the two lists A and C.
  - that's the VP!

The Left Corner idea

- The **left corner** of a rule is the set of terminals that can begin the rule.
- Example (VP rules – *nontransformed*):
  - `vp --> vbd, np.`
  - `vp --> vp, pp. % left recursive: tricky!`

  VP must begin with a verb (*saw*) because:
  - `vbd --> [saw].`

  The query:
  - `?- vp(X, [saw, the, man, with, a, telescope], C).`

  is ok, but not:
  - `?- vp(X, [with, a, telescope], C).`
  - `?- vp(X, [the man], C).`
  - Left corner of VP must be {saw} and not include {with, the}.

The Left Corner idea

- We can replace or substitute:
  - `vp(vp(X,Y)) --> vp(X), pp(Y). % left recursive: tricky!`
- with:
  - `vp(vp(X, Y), A, B) :- A = [saw|_], vp(X, A, C), pp(Y, C, B).`
  - **Note**: Prolog evaluates the RHS of the rule from left to right
    - only if `A = [saw | _ ]` is true will the rest of the RHS be even attempted.
  - **Notation**: [X | Y] means X is the head of the list, and Y is the rest of the list.
  - A = [saw | _] means list A begins with the word *saw* must be true.
  - _ is a variable (*underscore means we don't care about its value*)

Homework 12: Question 4

- Implement the left corner idea for VP.
- Replace:
  - `vp(vp(X,Y)) --> vp(X), pp(Y).`

  by
  - `vp(vp(X, Y), A, B) :- A = [saw|_], vp(X, A, C), pp(Y, C, B).`
- What happens when you try:

```
?- s(Tree, [a, boy, with, a, telescope, saw, the, man], []).
```

- now?

(If you don't see any difference, you may wish to reposition the rule.)

- Can you explain why?

Homework 12

- Due Sunday midnight
- One PDF file writeup.
- Give screenshots where appropriate.
- You can include your code as attachments.