LING/C SC/PSYC 438/538

Lecture 24

Sandiway Fong

Adminstration

538 Presentations

- Last Homework will be out Wednesday!

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		
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		

Last Time

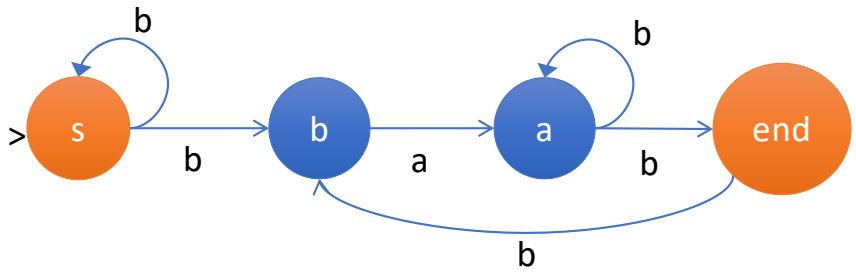
- SWI-Prolog DCG: grammar rules
  - n –––> RHS (*RHS* = sequence (,) of terminal and non-terminal symbols)
  - n (non-terminal symbol)
  - `[t]` (terminal symbol)
- Chomsky Hierarchy type 3 (equiv. FSA and regex):
  - if RHS limited to either `[t],` n or `[t]`
- **Key Concepts so far**:
  - recursive rules
    - examples: factorial, Σ* and grammar rules, e.g. multiple defs. for nonterminal n
  - backtracking: *explore multiple possible paths of execution*
    - *Prolog remembers* (*FOR YOU*) *which grammar rules have been tried*

Prolog Derivations

- Prolog's computation rule:
  - Try first matching rule in the database
  - Backtrack if matching rule leads to failure
  - undo and try next matching rule
- For grammars, this means:
  - Top-down left-to-right derivations
  - **left-to-right** = expand leftmost nonterminal first
  - Leftmost expansion done recursively = **depth-first**

bab2.prolog

```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], b.
7. a --> [b], a.
```



Enumeration

Using bab2.prolog

4. the set of all strings from the alphabet a, b such that each a is immediately preceded by and immediately followed by a b;

- What happens with the query `s(List, []).`?

```
?- [bab2].
true.
?- s(List, []).
List = [] ;
List = [b, a, b] ;
List = [b, a, b, a, b] ;
List = [b, a, b, a, b, a, b] ;
List = [b, a, b, a, b, a, b, a, b] ;
List = [b, a, b, a, b, a, b, a, b|...] [write]
List = [b, a, b, a, b, a, b, a, b, a, b] ;
List = [b, a, b, a, b, a, b, a, b, a, b, a, b]
```

; another answer?

press w turns off abbreviating

This is not all strings in the language!
λ|b(ab)+
b+ is in the language!

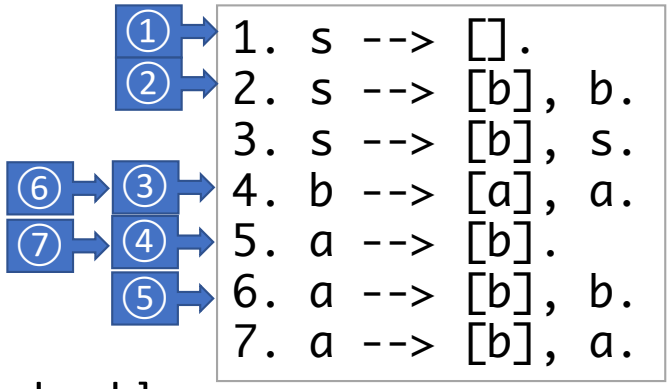
```
s([b, b], []).
true.
```

Enumeration

Why?

- **Answer**: Prolog tries first matching rule
- Order of application:

①②③④	⑤⑥⑦	⑧⑨⑩
1, 2, 4, 5,	6, 4, 5	6, 4, 5
[] [b,a,b]	[b,a,b,a,b]	[b,a,b,a,b,a,b]



Rules 3 and 7 never fire!

Enumeration

- We can swap the order of rules!

```
?- [bab2swap].
true.
?- s(List, []).
List = [] ;
List = [b, a, b] ;
List = [b, a, b, b] ;
List = [b, a, b, b, b] ;
List = [b, a, b, b, b, b]
```

Now we have
λ|bab+
cf. λ|b(ab)+
before the swap

```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], b.
7. a --> [b], a.
```

```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], a.
7. a --> [b], b.
```

Enumeration

What happens if we swap rules 2 and 3?

```
?- [bab2swap2].
true.
?- s(List, []).
List = [] ;
List = [b] ;
List = [b, b] ;
List = [b, b, b] ;
List = [b, b, b, b] ;
List = [b, b, b, b, b]
```

Now we have
b*
cf. λ|bab+ and λ|b(ab)+
But

```
s([b,a,b,b,a,b], []).
true
```

```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], a.
7. a --> [b], b.
```

```
1. s --> [].
2. s --> [b], s.
3. s --> [b], b.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], a.
7. a --> [b], b.
```

Breadth-first search

- Depth-first search (DFS) is **unfair** – not all rules get their turn
- Breadth-first search (BFS) is **fair**
  - computation paths are explored in order of length
  - we can simulate BFS using DFS (*at some cost to efficiency*)
  - called **Iterative Deepening** (ID) (*explanation an advanced topic*)
  - `id_meta.prolog` (*meta-level Prolog program*)

BFS: enumeration is possible

- ?– [id_meta].
- **true.**
- ?– [bab2].
- **true.**
- ?– id(s(List, [])).

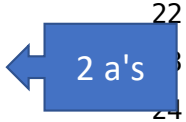
```
1.   List = [] ;
2.   List = [b] ;
3.   List = [b, a, b] ;
4.   List = [b, b] ;
5.   List = [b, a, b, b] ;
6.   List = [b, b, a, b] ;
7.   List = [b, b, b] ;
8.   List = [b, a, b, a, b] ;
9.   List = [b, a, b, b, b] ;
10.  List = [b, b, a, b, b] ;
11.  List = [b, b, b, a, b] ;
12.  List = [b, b, b, b] ;
13.  List = [b, a, b, a, b, b] ;
14.  List = [b, a, b, b, a, b] ;
15.  List = [b, a, b, b, b, b] ;
16.  List = [b, b, a, b, a, b] ;
17.  List = [b, b, a, b, b, b] ;
18.  List = [b, b, b, a, b, b] ;
19.  List = [b, b, b, b, a, b] ;
20.  List = [b, b, b, b, b] ;
21.  List = [b, a, b, a, b, a, b]
22.  List = [b, a, b, a, b, b, b] ;
23.  List = [b, a, b, b, a, b, b] ;
24.  List = [b, a, b, b, b, a, b] ;
25.  List = [b, a, b, b, b, b, b] ;
26.  List = [b, b, a, b, a, b, b] ;
27.  List = [b, b, a, b, b, a, b] ;
28.  List = [b, b, a, b, b, b, b] ;
29.  List = [b, b, b, a, b, a, b] ;
30.  List = [b, b, b, a, b, b, b] ;
31.  List = [b, b, b, b, a, b, b] ;
32.  List = [b, b, b, b, b, a, b] ;
33.  List = [b, b, b, b, b, b]
```

BFS: enumeration is possible

- How many a's in a string of length n?
  - Suppose we have 3 a's in a string
  - Every *a* needs a *b* on either side, so the minimum length of the string has to be 7.
  - 1 a, minimum length 3; 2 a's, minimum length is 5.
- **General Formula**:
  - len_{min} = #a * 2 + 1, for #a ≥ 1 because
  - len_{min} = #a * 3 – (#a – 1), for #a ≥ 1
  - Notice that len_{min} must always be odd
  - When length is even: must be all b's or any odd string (of length one less) in the language + one additional b.

BFS: enumeration is possible

1. List = [] ;
2. List = [b] ;
3. List = [b, a, b] ;
4. List = [b, b] ;
5. List = [b, a, b, b] ;
6. List = [b, b, a, b] ;
7. List = [b, b, b] ;
8. List = [b, a, b, a, b] ;
9. List = [b, a, b, b, b] ;
10. List = [b, b, a, b, b] ;
11. List = [b, b, b, a, b] ;
12. List = [b, b, b, b] ;
13. List = [b, a, b, a, b, b] ;
14. List = [b, a, b, b, a, b] ;
15. List = [b, a, b, b, b, b] ;
16. List = [b, b, a, b, a, b] ;
17. List = [b, b, a, b, b, b] ;
18. List = [b, b, b, a, b, b] ;
19. List = [b, b, b, b, a, b] ;
20. List = [b, b, b, b, b] ;
21. List = [b, a, b, a, b, a, b] ;
22. List = [b, a, b, a, b, b, b] ;
23. List = [b, a, b, b, a, b, b] ;
24. List = [b, a, b, b, b, a, b] ;
25. List = [b, a, b, b, b, b, b] ;
26. List = [b, b, a, b, a, b, b] ;
27. List = [b, b, a, b, b, a, b] ;
28. List = [b, b, a, b, b, b, b] ;
29. List = [b, b, b, a, b, a, b] ;
30. List = [b, b, b, a, b, b, b] ;
31. List = [b, b, b, b, a, b, b] ;
32. List = [b, b, b, b, b, a, b] ;
33. List = [b, b, b, b, b, b] ;
34. List = [b, a, b, a, b, a, b, b]

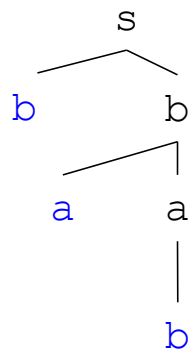


Extra Argument: Parse Tree

- **Recovering a parse tree**
  - in case of **true**, we can compute a syntax tree representation of the derivation
  - simple transformation: adding an extra argument to **all** nonterminals
  - applies to all grammar rules (not just regular grammars)

Example: `bab2.prolog` **again**

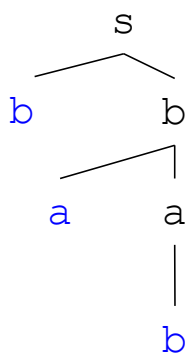
- string: bab




```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], b.
7. a --> [b], a.
```

Extra Argument: Parse Tree

- **Tree:**



`s(b,b(a,a(b)))`

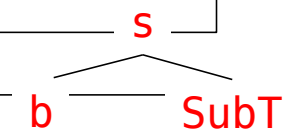
- **Prolog term data structure**:
  - hierarchical
  - allows sequencing of arguments
  - `functor(`arg_1`,..,`arg_n`)`
  - each arg_i could be another term or simple atom

Extra Argument: Parse Tree

- Transform each rule:

```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], b.
7. a --> [b], a.
```

```
1. s(Tree) --> [].
2. s(Tree) --> [b], b(SubTree).
…
5. a(Tree) --> [b].
```




```
1. s(s([])) --> [].
2. s(s(b,SubT)) --> [b], b(SubT).
…
5. a(a(b)) --> [b].
```

Extra Arguments: Parse Tree

- **Prolog grammar**

bab2.prolog

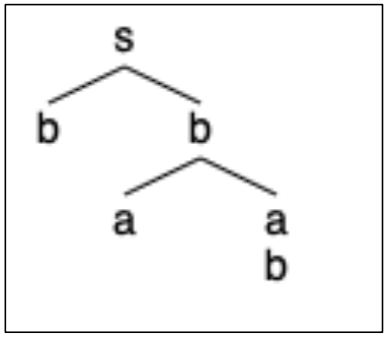
```
1. s --> [].
2. s --> [b], b.
3. s --> [b], s.
4. b --> [a], a.
5. a --> [b].
6. a --> [b], b.
7. a --> [b], a.
```

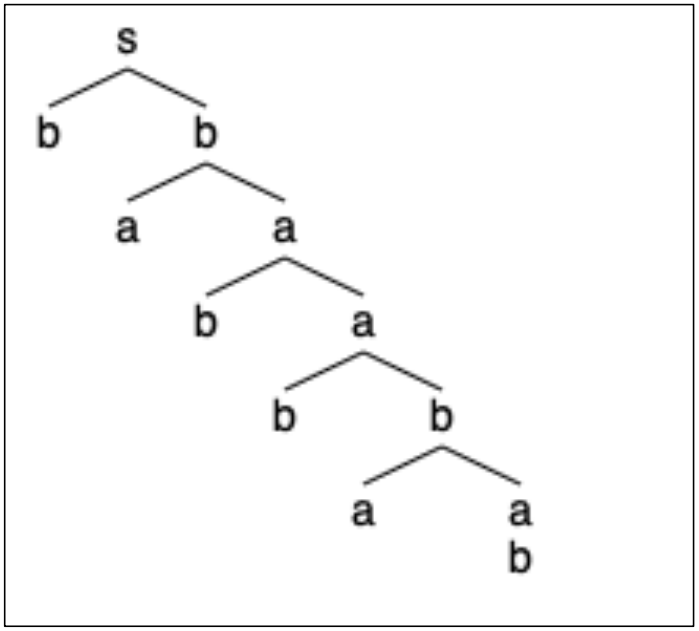
- **Prolog grammar computing a parse**

bab2tree.prolog

Extra Arguments: Parse Tree

```
?- [bab2tree].
true.
?- s(Tree, [b,a,b], []).
Tree = s(b, b(a, a(b))) ;
false.
?- s(Tree, [b,a,b,b,a,b], []).
Tree = s(b, b(a, a(b, a(b, b(a, a(b)))))) ;
false.
```



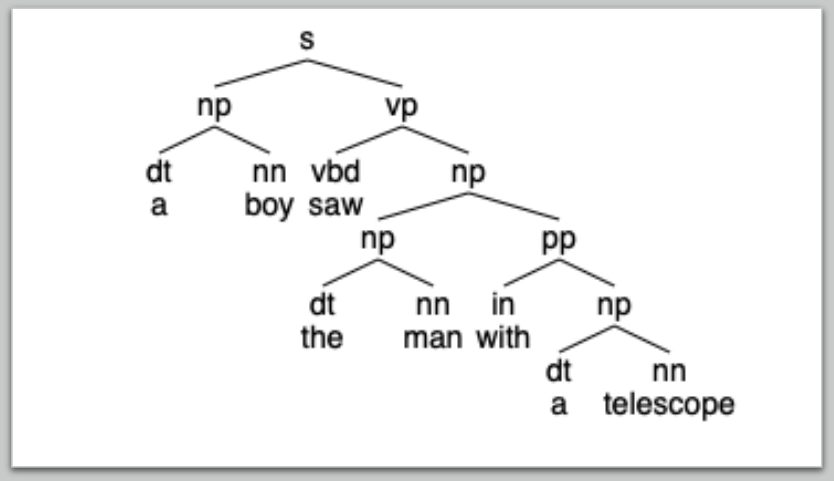


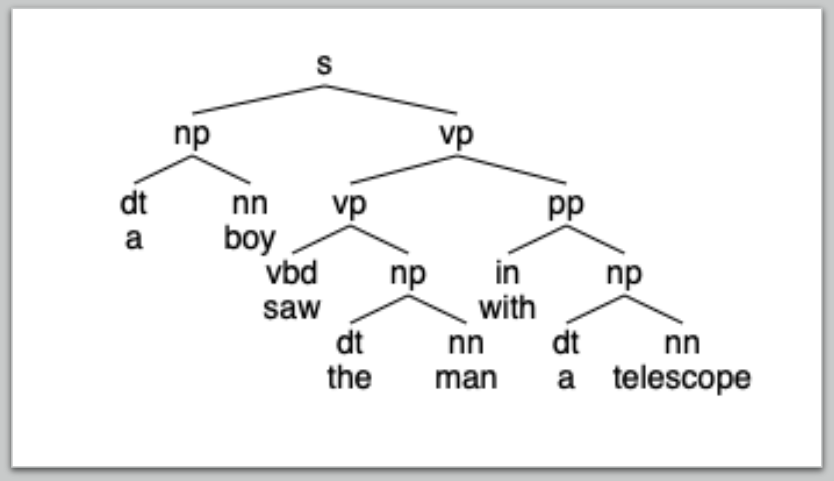
The extra argument *Tree* precedes the string lists in the call to nonterminal S

Extra Arguments: Parse Tree

- Useful in natural language grammars

```
?- [psg2].
true.
?- 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))))))) ;
Tree = s(np(dt(a), nn(boy)),
vp(vp(vbd(saw), np(dt(the), nn(man))),
pp(in(with), np(dt(a), nn(telescope)))))
;
false.
```





Extra Arguments

- 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)

Beyond Regular Languages

- Language
  - $a^n b^n$ = {ab, aabb, aaabbb, aaaabbbb, ... } n>=1
- A regular grammar extended to allow both left and right recursive rules can accept/generate it:
- anbn.prolog

```
1. a --> [a], b.
2. b --> [b].
3. b --> a, [b].
```

Set enumeration

```
?- a(L,[]).
L = [a, b] ;
L = [a, a, b, b] ;
L = [a, a, a, b, b, b] ;
L = [a, a, a, a, b, b, b, b] ;
L = [a, a, a, a, a, b, b, b, b, b]
```

- Example:

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

?- a([a,b],[]).
true ;
false.

?- a([a,a,b],[]).
false.

?- a([a,a,b,b],[]).
true ;
false.

?- a([a,a,b,b,b],[]).
false.

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

Set membership

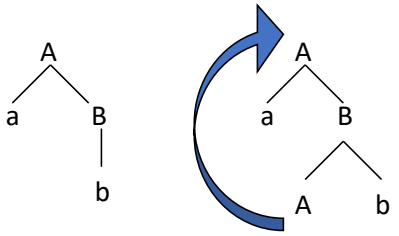
Beyond Regular Languages

- Language
  - $a^n b^n$ = {ab, aabb, aaabbb, aaaabbbb, ... } n>=1

- A regular grammar extended to allow both left and right recursive rules can accept/generate it:

```
1. a --> [a], b.
2. b --> [b].
3. b --> a, [b].
```

- Intuition:
  - grammar implements the stacking of partial trees balanced for a's and b's:



Beyond Regular Languages

- Language
  - $a^n b^n$ = {ab, aabb, aaabbb, aaaabbbb, ... } n>=1
- A regular grammar extended to allow both left and right recursive rules can accept/generate it:

```
1. a --> [a], b.
2. b --> [b].
3. b --> a, [b].
```

- A type-2 or context-free grammar (CFG) has no restrictions on what can go on the RHS of a grammar rule
- **Note**:
  - *CFGs still have a single nonterminal limit for the LHS of a rule*
- Example:

```
1. s --> [a], [b].
2. s --> [a], s, [b].
```