LING/C SC 581:

Advanced Computational Linguistics

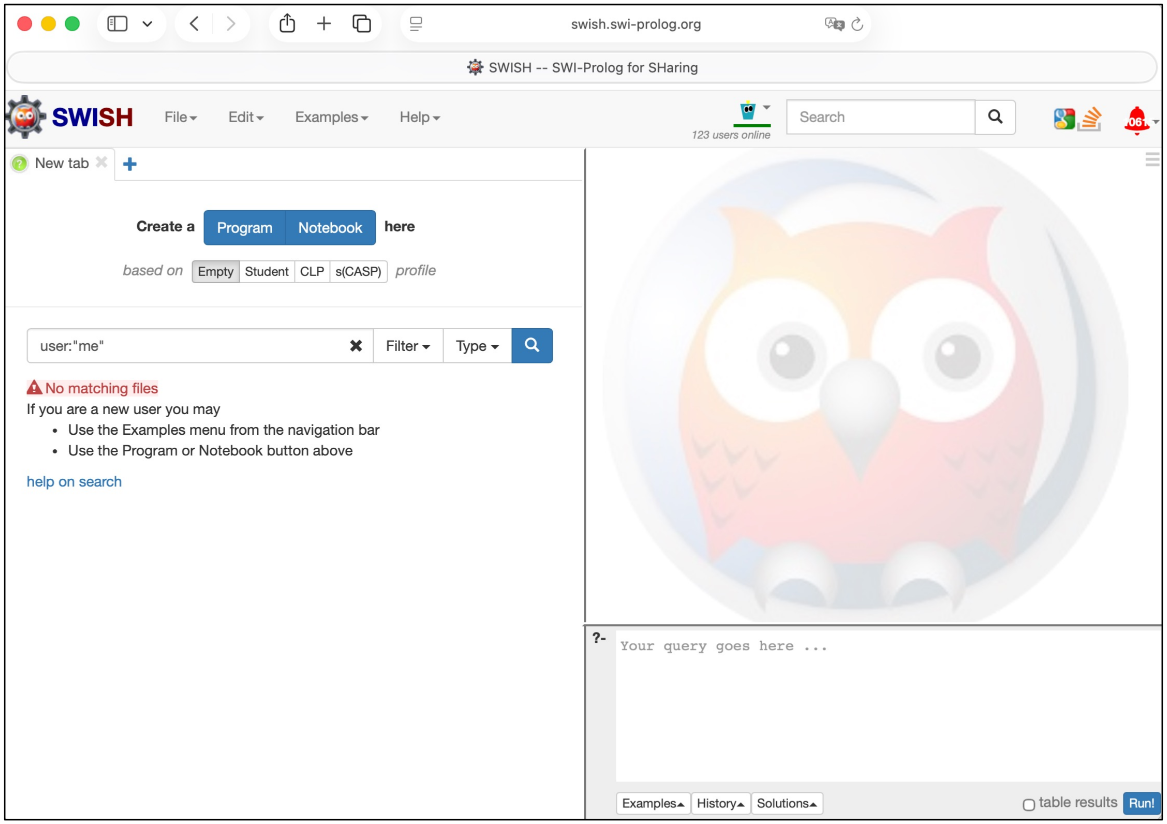
Lecture 10

Prof. Sandiway Fong

Today's Topic

- *Leaving the topic of context-sensitive languages*
- Turn to writing our own CFGs for natural language:
  1. A note on SWISH (SWI-Prolog for Sharing)
  2. agreement in natural language
  3. the problem with Prolog & left recursion
  4. a grammar transformation:
     - left recursive to right recursive **BUT** structure preserving
- Homework 6 (*note deadline change*)

SWISH = SWI Prolog for SHaring



Create a Program Notebook here

based on Empty Student CLP s(CASP) *profile*

user:"me"

No matching files

If you are a new user you may

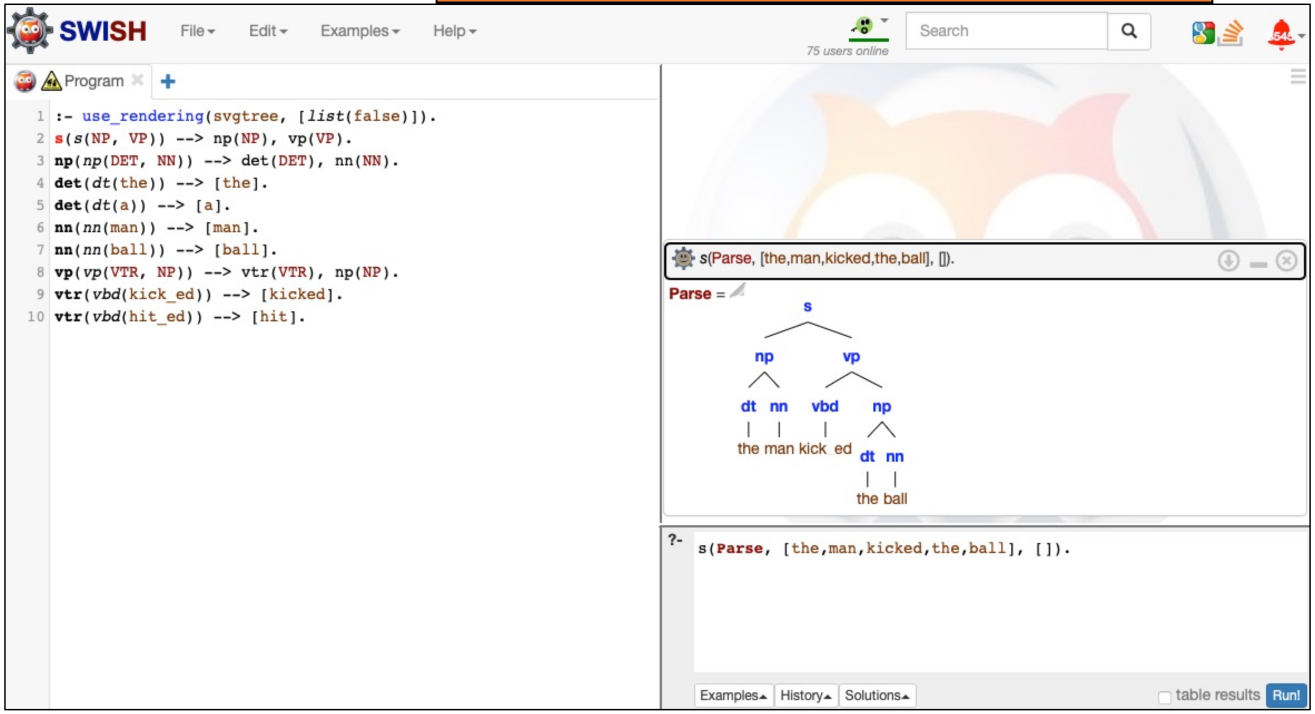
- Use the Examples menu from the navigation bar
- Use the Program or Notebook button above

help on search

```
Your query goes here ...
```

SWISH https://swish.swi-prolog.org

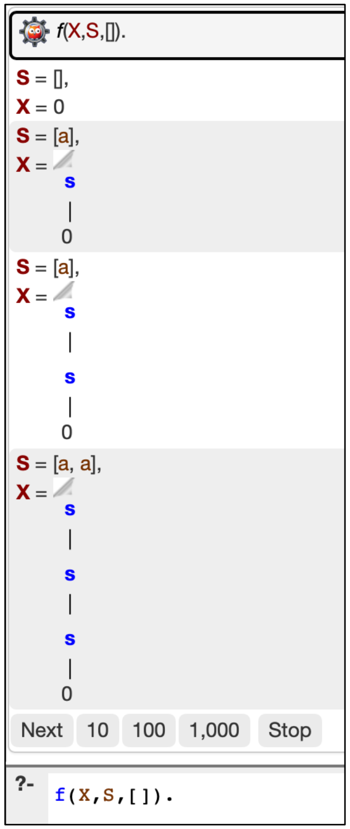
:- use_rendering(svgtree, [list(false)]).



```
:- use_rendering(svgtree, [list(false)]).
s(s(NP, VP)) --> np(NP), vp(VP).
np(np(DET, NN)) --> det(DET), nn(NN).
det(dt(the)) --> [the].
det(dt(a)) --> [a].
nn(nn(man)) --> [man].
nn(nn(ball)) --> [ball].
vp(vp(VTR, NP)) --> vtr(VTR), np(NP).
vtr(vbd(kick_ed)) --> [kicked].
vtr(vbd(hit_ed)) --> [hit].
```

```
s(Parse, [the,man,kicked,the,ball], []).
Parse =
```

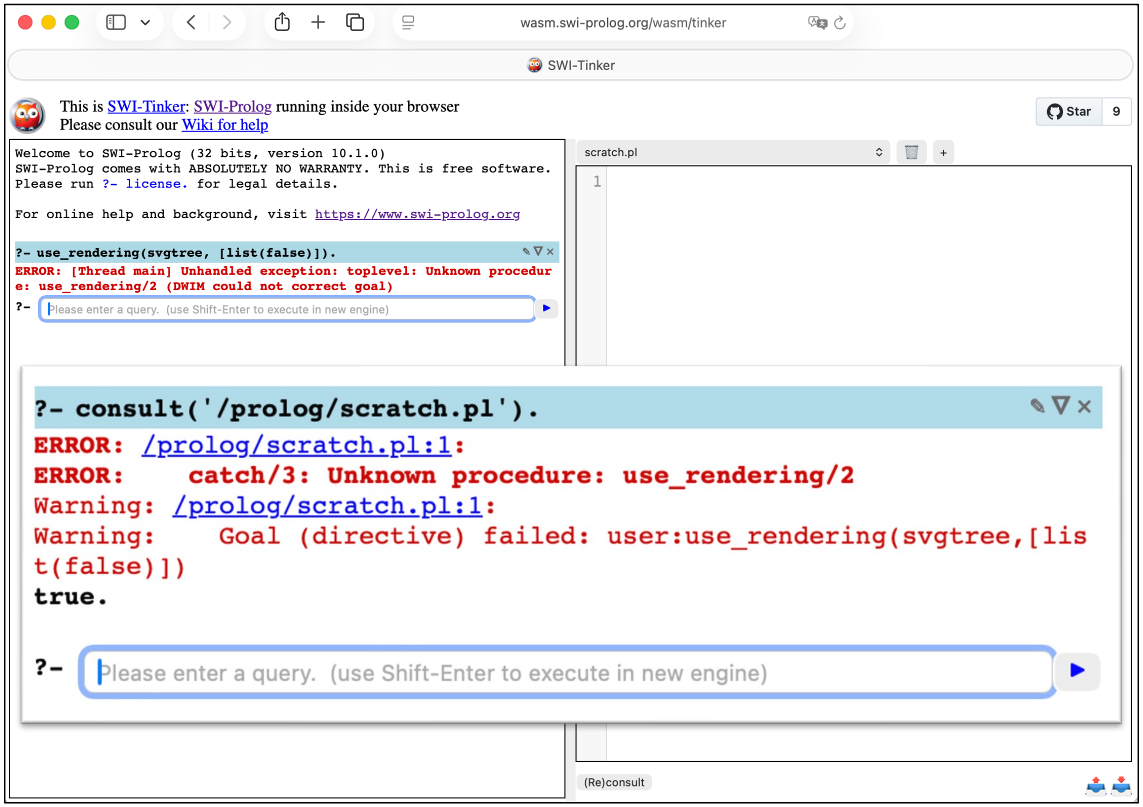
```
?- s(Parse, [the,man,kicked,the,ball], []).
```



```
f(X,S,[]).
S = [],
X = 0
S = [a],
X =
S = [a],
X =
S = [a, a],
X =
```

```
?- f(X,S,[]).
```

SWI-Tinker (*inside your browser*)


wasm.swi-prolog.org/wasm/tinker
SWI-Tinker
Star 9
scratch.pl
+
1
(Re)consult
Please enter a query. (use Shift-Enter to execute in new engine)

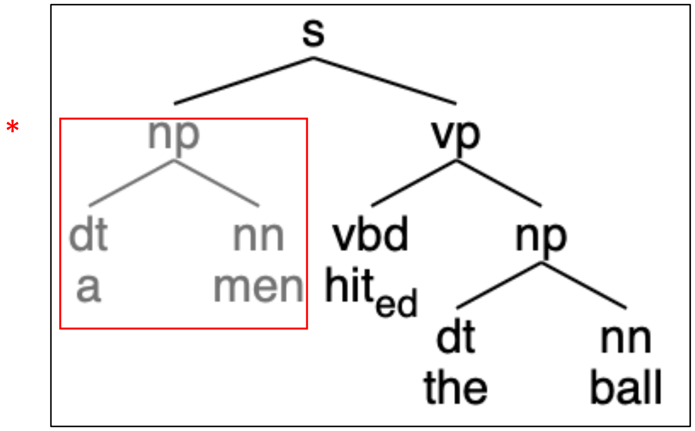
This is SWI-Tinker: SWI-Prolog running inside your browser
Please consult our Wiki for help

```
Welcome to SWI-Prolog (32 bits, version 10.1.0)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org

?- use_rendering(svgtree, [list(false)]).
ERROR: [Thread main] Unhandled exception: toplevel: Unknown procedur
e: use_rendering/2 (DWIM could not correct goal)
?-
```

```
?- consult('/prolog/scratch.pl').
ERROR: /prolog/scratch.pl:1:
ERROR:    catch/3: Unknown procedure: use_rendering/2
Warning: /prolog/scratch.pl:1:
Warning:    Goal (directive) failed: user:use_rendering(svgtree,[lis
t(false)])
true.

?-
```

Extra Arguments: Agreement

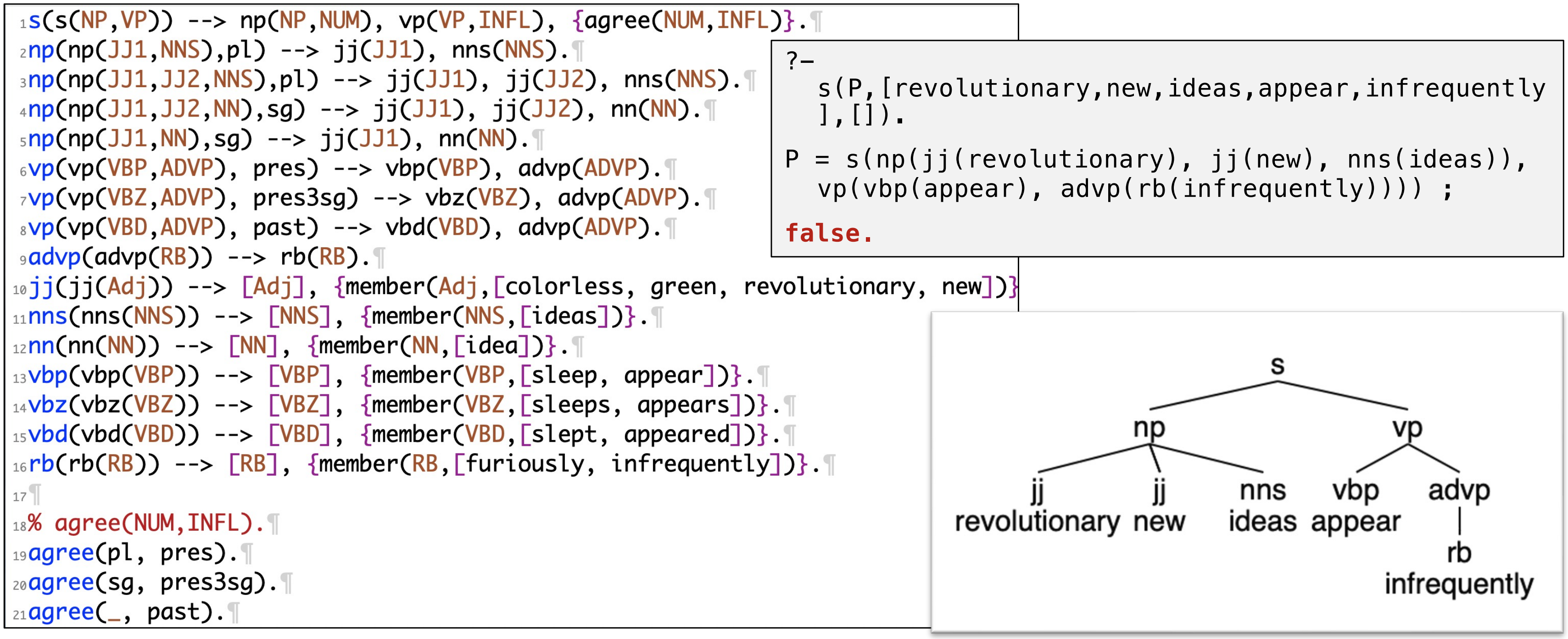
- **Idea**:
  - We can also use an extra argument to impose constraints between constituents within a DCG rule



- **Example**:
  - English number agreement between DT and NN
  - Data:
    - the man   the men
    - a man   *a men
  - Lexical Features (Number):
    - *man* value singular (`sg`)
    - *men* value plural (`pl`)
    - *the* value singular or plural (`sg/pl`)
    - *a* value singular (`sg`)

* means ***ungrammatical***

Recall a Class Exercise: `agree/2`



```
s(s(NP,VP)) --> np(NP,NUM), vp(VP,INFL), {agree(NUM,INFL)}.
np(np(JJ1,NNS),pl) --> jj(JJ1), nns(NNS).
np(np(JJ1,JJ2,NNS),pl) --> jj(JJ1), jj(JJ2), nns(NNS).
np(np(JJ1,JJ2,NN),sg) --> jj(JJ1), jj(JJ2), nn(NN).
np(np(JJ1,NN),sg) --> jj(JJ1), nn(NN).
vp(vp(VBP,ADVP), pres) --> vbp(VBP), advp(ADVP).
vp(vp(VBZ,ADVP), pres3sg) --> vbz(VBZ), advp(ADVP).
vp(vp(VBD,ADVP), past) --> vbd(VBD), advp(ADVP).
advp(advp(RB)) --> rb(RB).
jj(jj(Adj)) --> [Adj], {member(Adj,[colorless, green, revolutionary, new])}
nns(nns(NNS)) --> [NNS], {member(NNS,[ideas])}.
nn(nn(NN)) --> [NN], {member(NN,[idea])}.
vbp(vbp(VBP)) --> [VBP], {member(VBP,[sleep, appear])}.
vbz(vbz(VBZ)) --> [VBZ], {member(VBZ,[sleeps, appears])}.
vbd(vbd(VBD)) --> [VBD], {member(VBD,[slept, appeared])}.
rb(rb(RB)) --> [RB], {member(RB,[furiously, infrequently])}.

% agree(NUM,INFL).
agree(pl, pres).
agree(sg, pres3sg).
agree(_, past).
```

```
?-
  s(P,[revolutionary,new,ideas,appear,infrequently
  ],[]).

P = s(np(jj(revolutionary), jj(new), nns(ideas)),
  vp(vbp(appear), advp(rb(infrequently)))) ;

false.
```

Determiner-Noun Agreement

- Example (`nl3.prolog`):

```
1. np(np(DET, NN)) --> det(DET, NUM), nn(NN, NUM).
2. det(dt(the), sg) --> [the].
3. det(dt(the), pl) --> [the].
4. det(dt(a), sg) --> [a].
5. nn(nn(man), sg) --> [man].
6. nn(nn(men), pl) --> [men].
7. nn(nn(ball), sg) --> [ball].
8. vp(vp(VTR, NP)) --> vtr(VTR), np(NP).
9. vtr(vbd(kick_ed)) --> [kicked].
10.vtr(vbd(hit_ed)) --> [hit].
```

Extra Argument: Agreement

Note:

- extra argument NUM for agreement here is basically “syntactic sugar” and ***lends no more expressive*** power to the grammar rule system:
- i.e. *we can enforce agreement without the use of the extra argument at the cost of writing more rules*

- Instead of

```
np(np(DET, NN)) --> det(DET, NUM), nn(NN, NUM).
```

could have encoded NUM into the nonterminal name:

```
np(np(DET, NN)) --> det_sg(DET), nn_sg(NN).
np(np(DET, NN)) --> det_pl(DET), nn_pl(NN).
det_sg(dt(the)) --> [the].
det_pl(dt(the)) --> [the].
det_sg(dt(a)) --> [a].
nn_sg(nn(man)) --> [man].
nn_pl(nn(men)) --> [men].
nn_sg(nn(ball)) --> [ball].
```

`nl4.prolog`

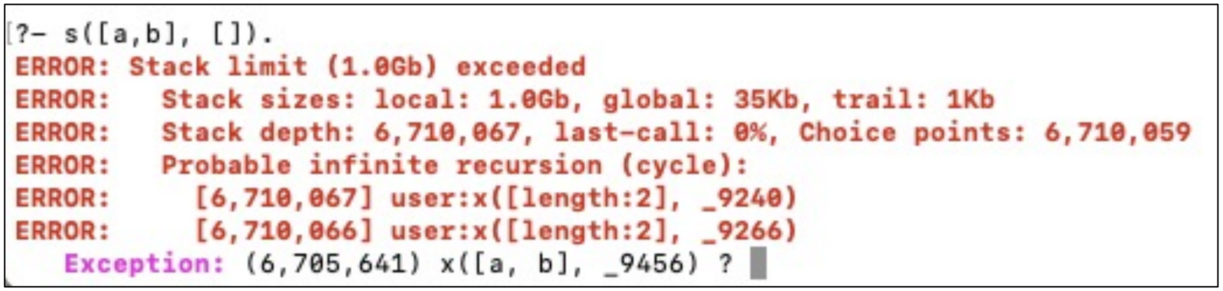
Left recursion and Prolog DCGs

Left recursive grammars:

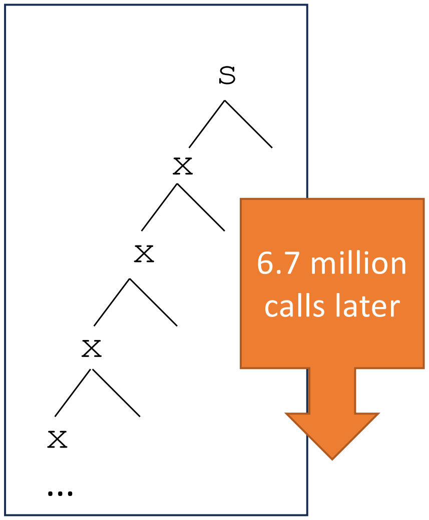
- given Prolog's simple *left-to-right depth-first computation rule*, left recursive rules are a *no-no* …
- Example (`left.prolog`):

```
1. s --> x, y.
2. x --> x, [a].
3. x --> [a].
4. y --> [b].
```

rule for nonterminal x immediately calls x again!



```
[?- s([a,b], []).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 1.0Gb, global: 35Kb, trail: 1Kb
ERROR:   Stack depth: 6,710,067, last-call: 0%, Choice points: 6,710,059
ERROR:   Probable infinite recursion (cycle):
ERROR:     [6,710,067] user:x([length:2], _9240)
ERROR:     [6,710,066] user:x([length:2], _9266)
   Exception: (6,705,641) x([a, b], _9456) ? 
```



6.7 million calls later

Left recursion and Prolog DCGs

- **Aside**: *what is the language of this grammar*?
- `left.prolog`:

```
1. s --> x, y.
2. x --> x, [a].
3. x --> [a].
4. y --> [b].
```

Rule ordering

- Example (`left.prolog`):

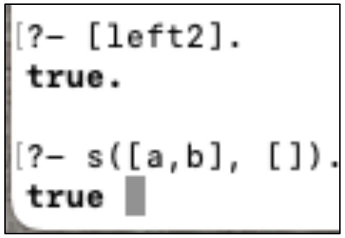
```
1. s --> x, y.
2. x --> x, [a].
3. x --> [a].
4. y --> [b].
```

- An idea (*swap rules 2 and 3*):

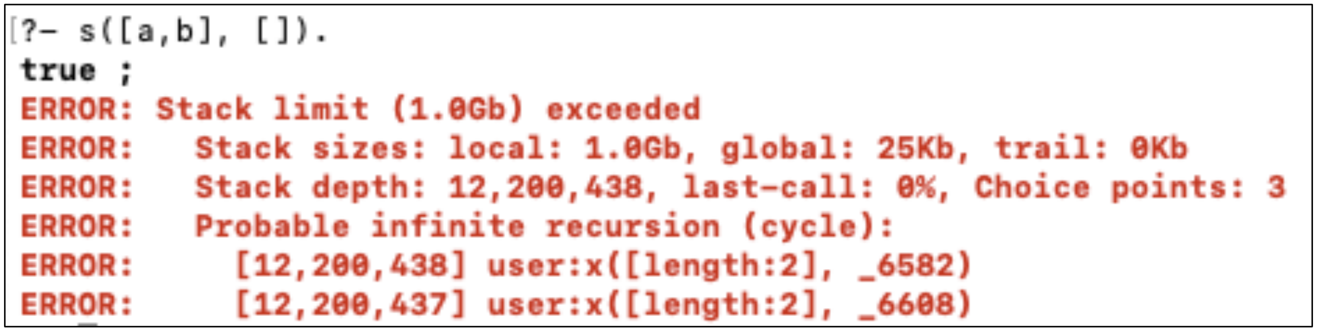
```
1. s --> x, y.
2. x --> [a].
3. x --> x, [a].
4. y --> [b].
```

; eventually calls for stacking rule 3. *12 million deep*

- (`left2.prolog`)



```
[?- [left2].
 true.

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



```
[?- s([a,b], []).
 true ;
 ERROR: Stack limit (1.0Gb) exceeded
 ERROR:   Stack sizes: local: 1.0Gb, global: 25Kb, trail: 0Kb
 ERROR:   Stack depth: 12,200,438, last-call: 0%, Choice points: 3
 ERROR:   Probable infinite recursion (cycle):
 ERROR:     [12,200,438] user:x([length:2], _6582)
 ERROR:     [12,200,437] user:x([length:2], _6608)
```

Big picture question

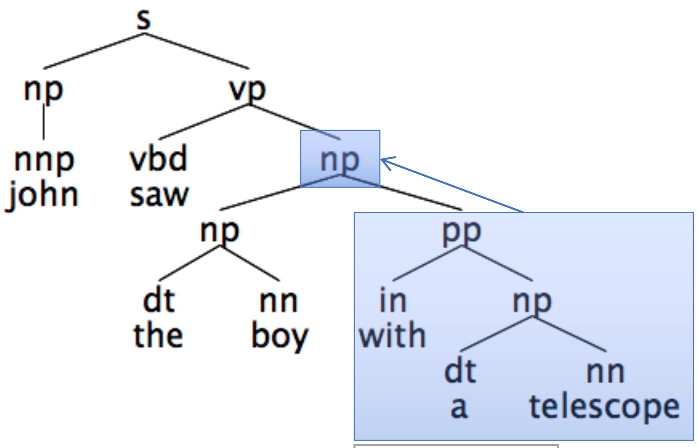
- Is this just a theoretical problem: i.e. not a problem for natural language grammars?
- Unfortunately, it is a problem …
  - *John saw the boy with a telescope*
  - is structurally ambiguous wrt. attachment of the PP *with a telescope*
  - (PP = prepositional phrase)

Preposition Phrase (PP) Attachment

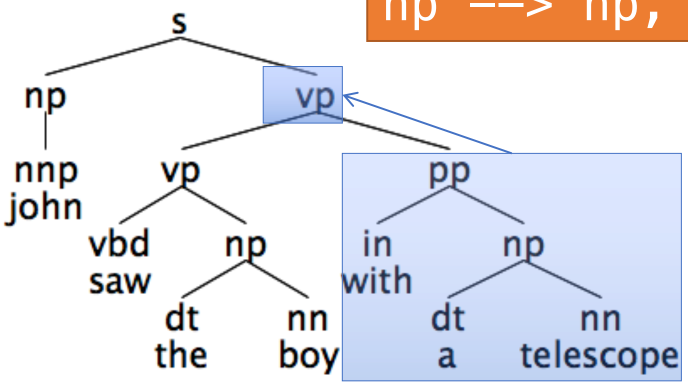
- The preferred syntactic analysis is a left recursive parse
- Example:
  - *John saw the boy with a telescope*

Rules are:

```
vp --> vp, pp.
np --> np, pp.
```



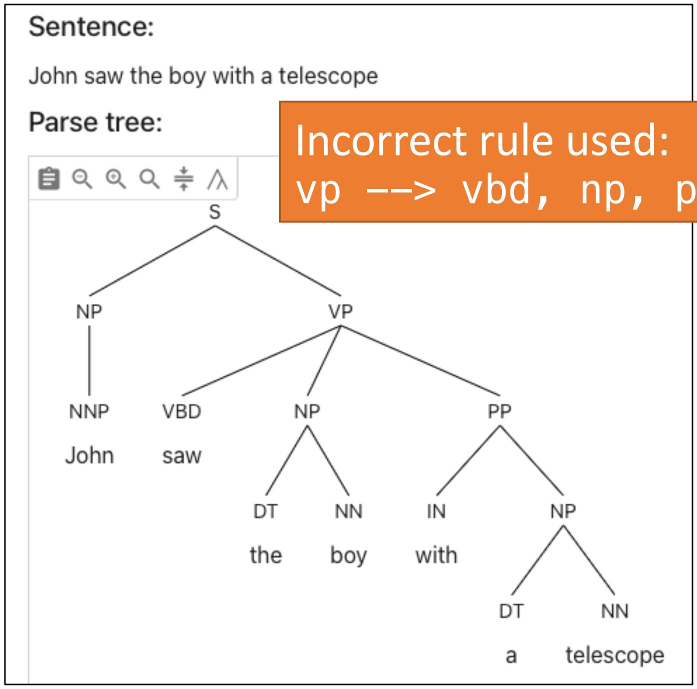
with$_{\text{possessive}}$



with$_{\text{instrument}}$

Preposition Phrase (PP) Attachment

https://parser.kitaev.io



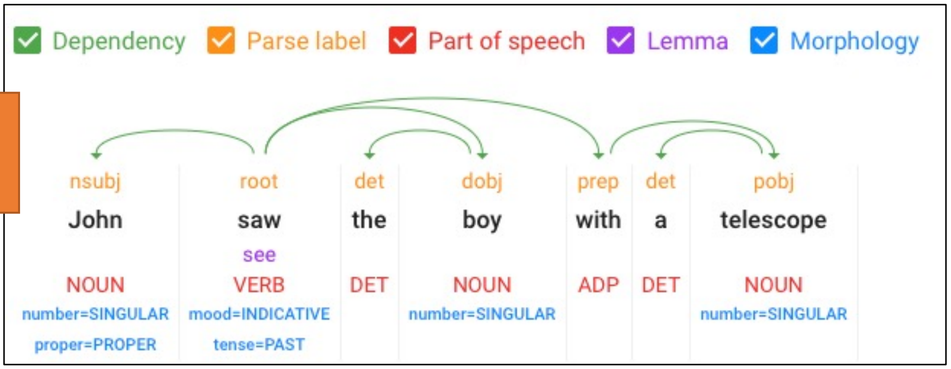
Sentence:

John saw the boy with a telescope

Parse tree:

Incorrect rule used:
`vp --> vbd, np, pp.`

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

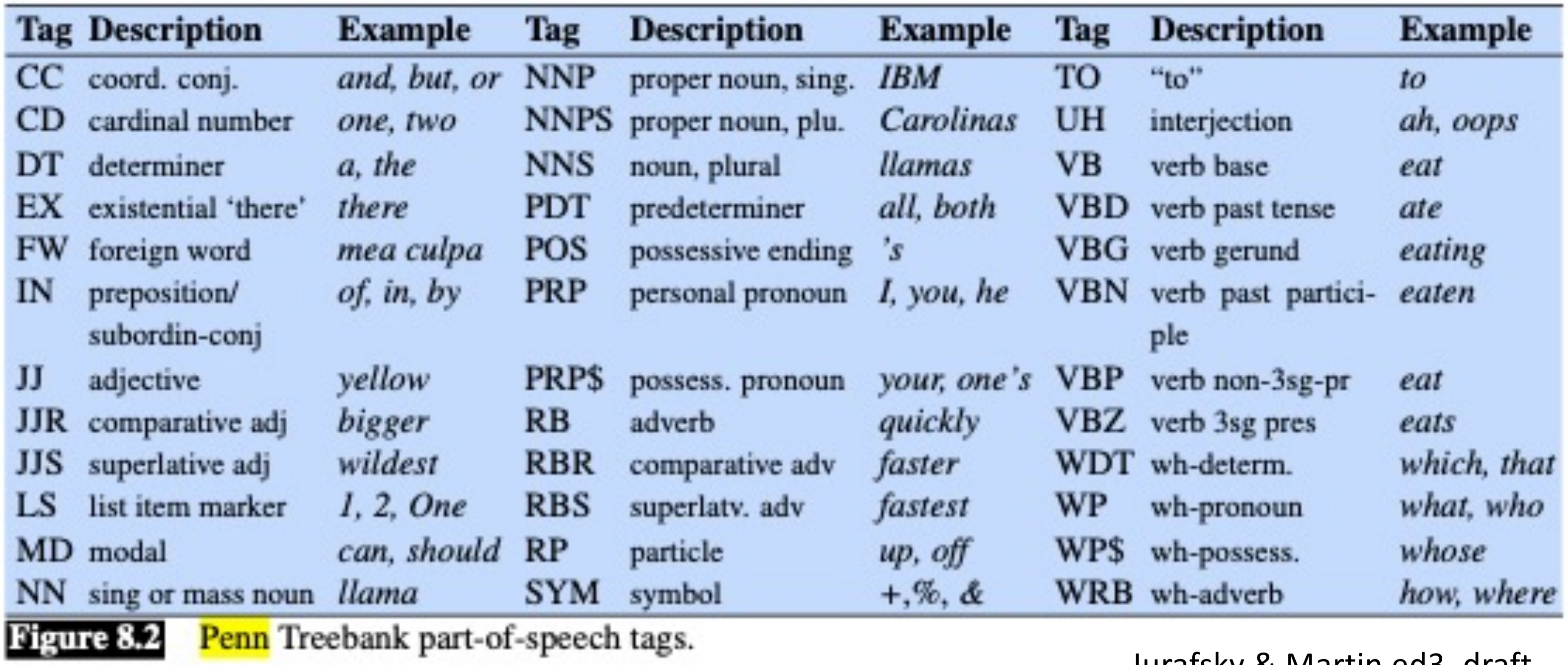


dependency parse (*essentially same problem*):
`root is` *`saw`*
`root --> prep`
`root --> dobj.`

Class Exercise

- Let's add PP-attachment rules mentioned earlier to `nl3.prolog`
  - `vp --> vp, pp.`
  - `np --> np, pp.`
- Need to add:
  - verb (VBD): *saw – past tense* (-ed)
  - preposition (IN): *`with`*
  - singular nouns (NN): *`telescope, boy, limp`*
  - proper noun (NNP): *`john`* ('*John*'), *`mary`* ('Mary')
    - recall: *initial caps = Prolog variable*

Penn Part-of-Speech (POS)Tagset



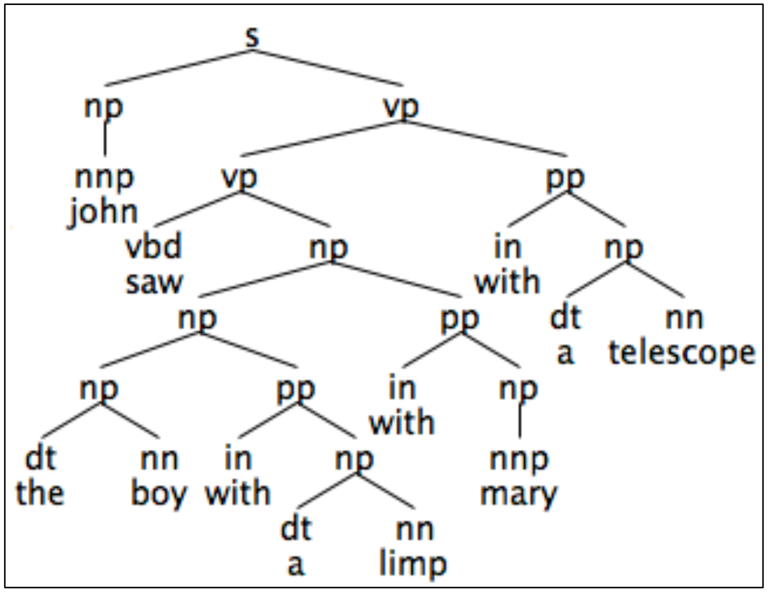
Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+,%, &*	WRB	wh-adverb	*how, where*

Figure 8.2 Penn Treebank part-of-speech tags.

Jurafsky & Martin ed3. draft

Preposition Phrase (PP) Attachment

- The preferred syntactic analysis is a left recursive parse
  - notice PPs can be can "stacked", as in:
  - *John saw the boy with a limp with Mary with a telescope*
  - *with*-ambiguity:
    - *with*possessive ,
    - *with*accompaniment,
    - *with*instrument



Preposition Phrase Attachment

- Linguistically:
  - PP (recursively) adjoins to NP or VP
  - `np(np(NP,PP)) --> np(NP), pp(PP).`
  - `vp(vp(VP,PP)) --> vp(VP), pp(PP).`
- Left recursion gives Prolog problems
- Derivation (top-down, left-to-right):

```
1. vp
2. vp pp
3. vp pp pp
4. vp pp pp pp
5. vp pp pp pp pp
```

infinite loop…

Note:
only the parse tree argument shown here
other extra arguments are possible

Transformation

- Apply the general left to right recursive transformation:

```
x(x(X,y)) --> x(X), [y].
x(x(z)) --> [z].
```

```
x(X) --> [z], w(X,x(z)).
x(x(z)) --> [z].
w(W,X) --> [y], w(W,x(X,y)).
w(x(X,y),X) --> [y].
```

Note:
w is a ***fresh*** nonterminal that takes 2 arguments

- to the NP rules:

```
1. np(np(DT,NN)) --> dt(DT,Number),[z]nn(NN,Number).
2. np(np(NP,PP)) --> np(NP), pp(PP).
                                [y]
         x                x
```

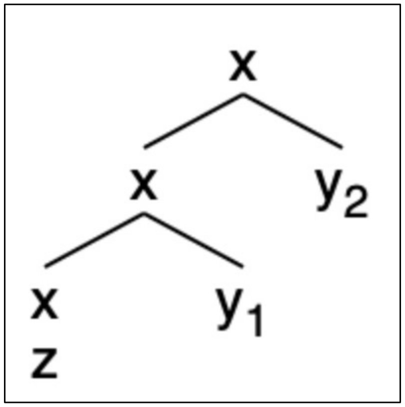
x is the recursive nonterminal

Transformation

• Consider input strings:	Parse:	Transformed rules:
1. [z]	x(z)	2
2. [z, y]	x(x(z),y)	1 + 4
3. [z, y_1, y_2]	x(x(x(z),y_1),y_2)	1 + 3 + 4

```
1. x(x(X,y)) --> x(X), [y].
2. x(x(z)) --> [z].
```

```
1. x(X) --> [z], w(X,x(z)).
2. x(x(z)) --> [z].
3. w(W,X) --> [y], w(W,x(X,y)).
4. w(x(X,y),X) --> [y].
```



the left recursive structure formed by a right recursive parse of [z, y_1, y_2]

Steps for example 3:

[z, ●y_1, y_2] rule 1: call nonterminal w(X,x(z))

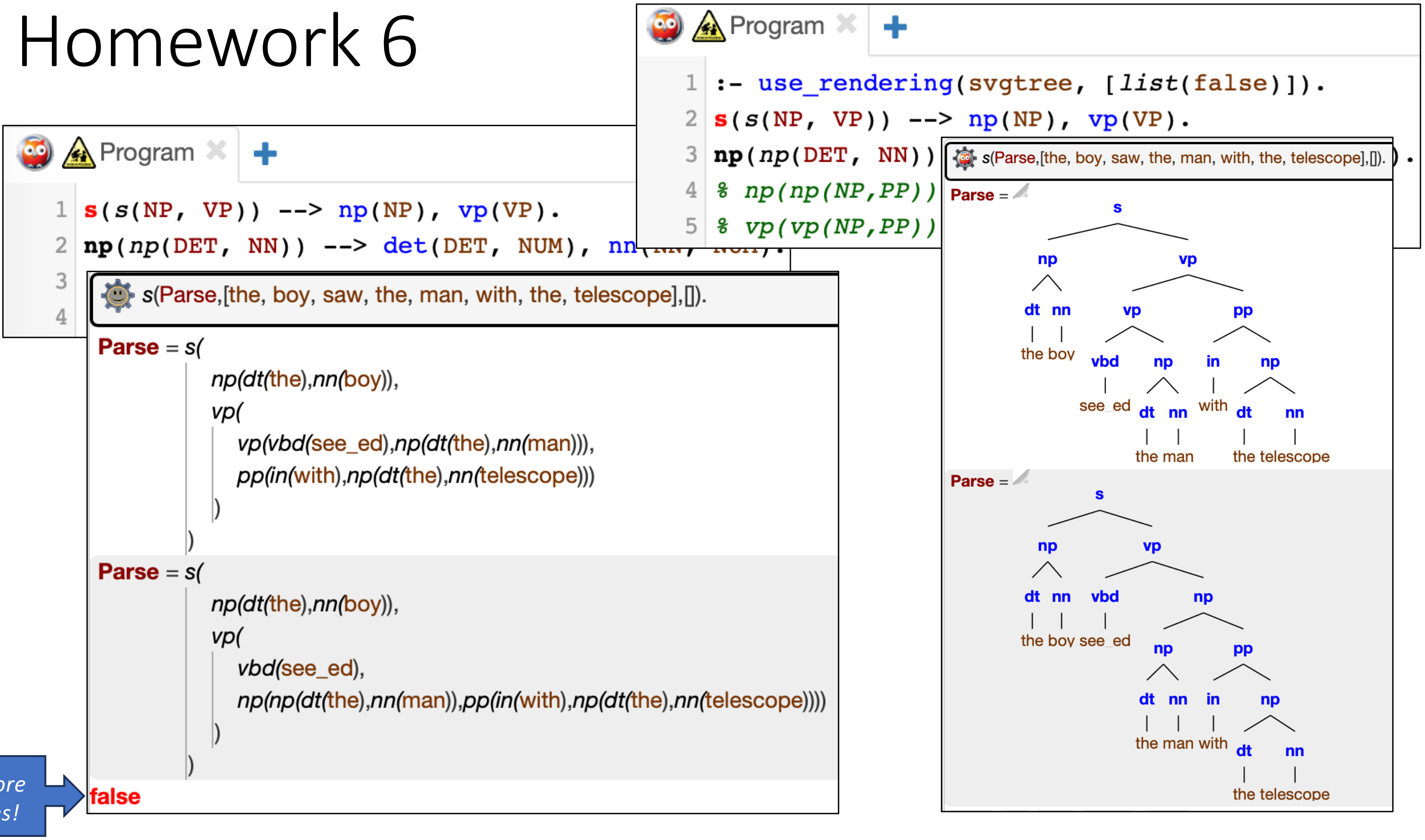
[z, y_1, ●y_2] rule 3: call nonterminal w(X,x(x(z),y_1))

[z, y_1, y_2●] rule 4: answer X = x(x(x(z),y_1),y_2)

Homework 6

- Question 1: apply the transformation to the left recursion starting with `nl3.prolog`:
  - `np(np(NP,PP)) --> np(NP), pp(PP).`
  - `vp(vp(VP,PP)) --> vp(VP), pp(PP).`
- Show your grammar working properly on example sentences:
  1. the boy saw the man with the telescope
  2. the boy with the telescope saw the man
  3. the boy kicked the man with the telescope
  4. the boy with the telescope kicked the man
  5. the boy with the telescope kicked the man with the limp
- Show all possible parses (;) until false in each case

Homework 6



```
s(s(NP, VP)) --> np(NP), vp(VP).
np(np(DET, NN)) --> det(DET, NUM), nn(NN, NUM).
```

```
s(Parse,[the, boy, saw, the, man, with, the, telescope],[]).
```

```
Parse = s(
            np(dt(the),nn(boy)),
            vp(
                vp(vbd(see_ed),np(dt(the),nn(man))),
                pp(in(with),np(dt(the),nn(telescope)))
            )
        )
Parse = s(
            np(dt(the),nn(boy)),
            vp(
                vbd(see_ed),
                np(np(dt(the),nn(man)),pp(in(with),np(dt(the),nn(telescope))))
            )
        )
false
```

```
:- use_rendering(svgtree, [list(false)]).
s(s(NP, VP)) --> np(NP), vp(VP).
np(np(DET, NN))
% np(np(NP,PP))
% vp(vp(NP,PP))
```

```
s(Parse,[the, boy, saw, the, man, with, the, telescope],[]).
```

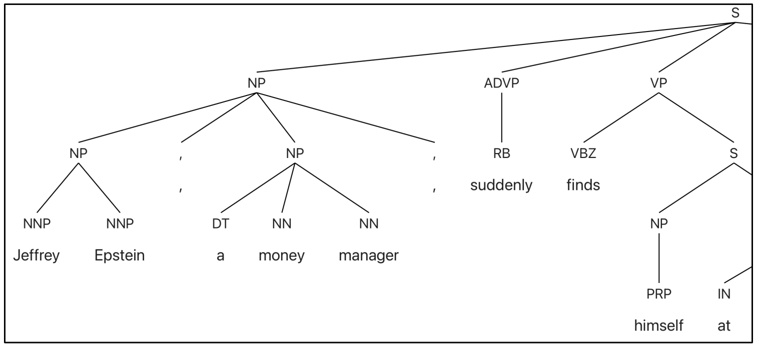
No more parses!

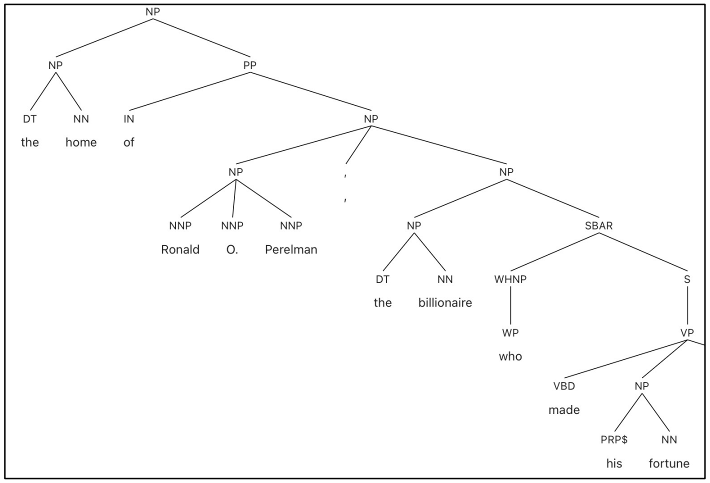
Homework 6

- Hint #1:
  - consider the case when there are multiple base rules for x
- `x(x(X,y)) --> x(X), [y].`
- `x(x(z)) --> [z].`
- `x(x(w)) --> [w].`
- Hint #2:
  - w must be a fresh nonterminal, i.e. cannot be shared between the NP and VP recursions. Why?

Homework 6

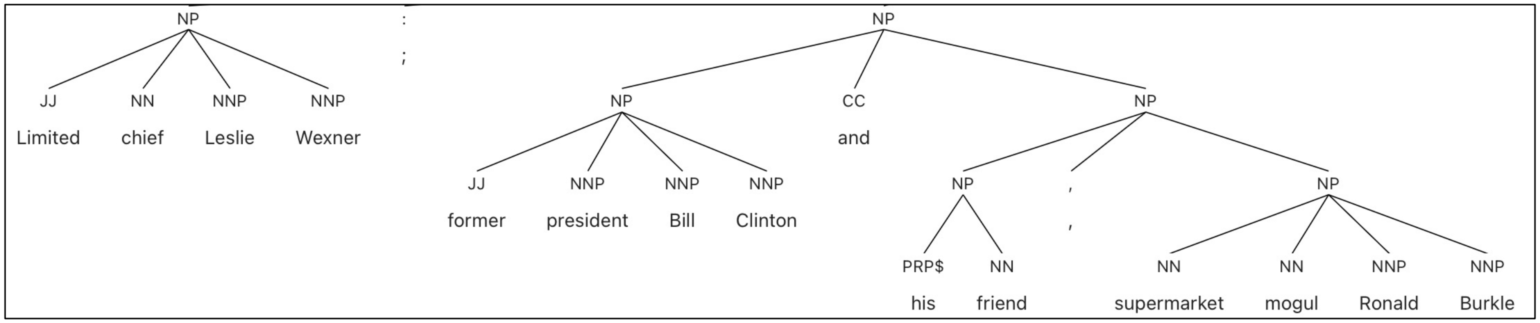
- Question 2: with *appositives*, we also get a left recursive analysis.
- Non-restrictive (*set off by commas*):
  - NP, NP, *Jeffrey Epstein, a money manager,* (source: *NYTimes*)
    Ronald O. Perelman, the billionaire
- Berkeley Neural Parser:
  `parser.kitaev.io`





Homework 6

- **Aside**: there's another kind of apposition, but not left recursive (*according to the Berkeley Neural Parser*):



Homework 6

- Question 2: add the comma-separated kind of appositives to your grammar.
    - Part 1: give the left recursive rule.
    - Part 2: apply the transformation (*to eliminate left recursion*).
- Show how you can parse:
    - the boy , a student , saw the man
    - the boy , a student , saw the man , the astronomer ,
- Note:
    - comma needs to be quoted in the input list or grammar rule: `[',']`
    - `s(Parse,[the, boy, ',', a, student, ',', saw, the, man],[]).`

Homework 6

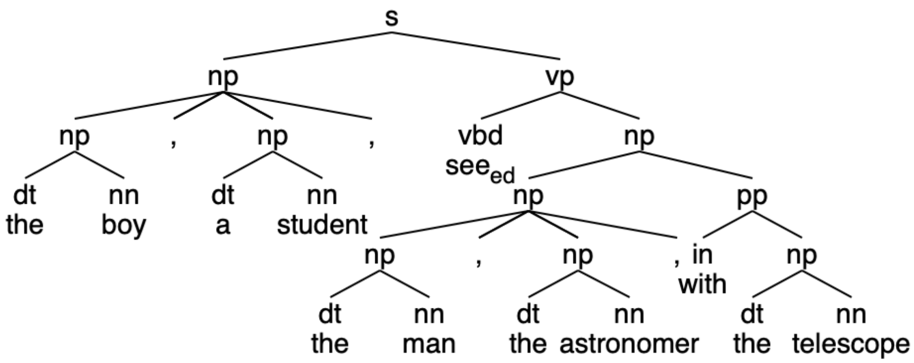
- Question 3: *no need to implement this*, i.e. don't worry about it, but how can you eliminate the superfluous last comma in Question 2?
  - *the boy , a student , saw the man , the astronomer ,
  - the boy , a student , saw the man , the astronomer
- **Hint**: is it context-free?

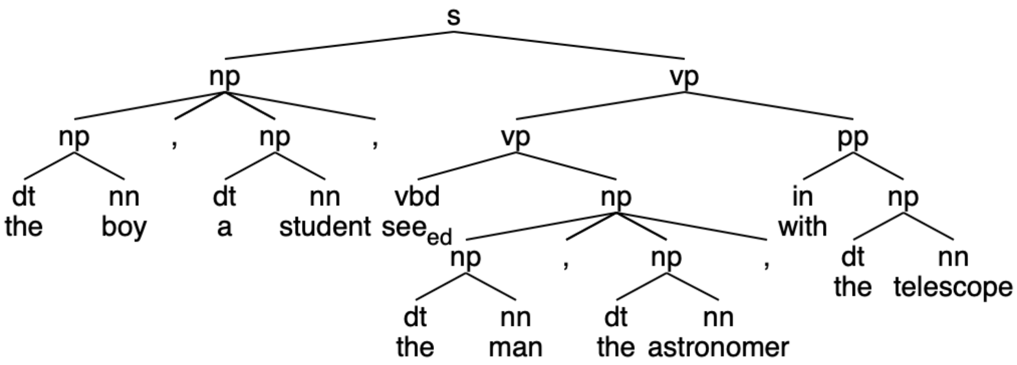
Homework 6

- Question 4: can your grammar handle both of these?
  - the boy , a student , saw the man with the telescope
  - the boy , a student , saw the man , the astronomer , with the telescope
- If not, explain why, then modify your grammar.
- If yes, explain why it can.

Homework 6

- Example parses





Homework 6

- Submit to sandiway@arizona.edu
- SUBJECT: 581 Homework 6 *YOUR NAME*
- One PDF file (for grading)
  - include your grammar code and SWI-Prolog screenshots in your answer
- Attach (if I need to run your code):
  - source code for your grammar
- Deadline:
  - midnight Monday: **no lecture next Monday** (*I'm away*)
  - we will review the homework next Wednesday