LING/C SC 581:
Advanced Computational Linguistics

Lecture 6

- *grateful acknowledgements to Riny Huybregts for his help and contributions*
- *all remaining errors are mine*

Today's Topics

- Homework 3 Review
- An example of context-sensitivity in natural language

Homework 3 Review

- `abc_count.prolog` for $\{a^n b^n c^n \mid n>0\}$

```
1s --> a(X), b(X), c(X).
2a(1) --> [a].
3a(N) --> [a], a(M), {N is M+1}.
4b(1) --> [b].
5b(N) --> [b], b(M), {N is M+1}.
6c(1) --> [c].
7c(N) --> [c], c(M), {N is M+1}.
```

- Question 1:
  - Suggested in class that a *more efficient* grammar (measured using time/1) could be built to reject strings not in the grammar by counting down the b's and c's.
  - Compared to `abc_count.prolog` for inputs:
    - `[a,a,b,b,b,b,b,b,b,b,c,c]` (8 *b*'s)
    - `[a,a,b,b,b,b,b,b,b,b,b,b,b,b,b,b,c,c]` (16 *b*'s)

Homework 3 Review

```
?- [abc_count].
true.
• ?- time(s([a,a,b,b,c,c],[])).
• % 9 inferences, 0.000 CPU in 0.000 seconds (79% CPU, 219512 Lips)
• true ;
• % 6 inferences, 0.000 CPU in 0.000 seconds (67% CPU, 333333 Lips)
• false
?- time(s([a,a,b,b,b,b,b,b,b,b,c,c],[])).
% 45 inferences, 0.000 CPU in 0.000 seconds (78% CPU, 548780 Lips)
false.

?- time(s([a,a,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,c,c],[])).
% 145 inferences, 0.000 CPU in 0.000 seconds (83% CPU, 1705882 Lips)
false.
```

Homework 3 Review

```
?- [abc_count2].
true.
?- time(s([a,a,b,b,c,c],[])).
% 9 inferences, 0.000 CPU in 0.000 seconds (71% CPU, 409091 Lips)
true ;
% 6 inferences, 0.000 CPU in 0.000 seconds (65% CPU, 333333 Lips)
false.

?- time(s([a,a,b,b,b,b,b,b,b,b,c,c],[])).
% 11 inferences, 0.000 CPU in 0.000 seconds (68% CPU, 458333 Lips)
false.
?- time(s([a,a,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,c,c],[])).
% 11 inferences, 0.000 CPU in 0.000 seconds (70% CPU, 392857 Lips)
false.
?-
```

Homework 3 Review

- Another example:

```
?- [abc_count3].
true.

?- time(s([a,a,b,b,c,c],[])).
% 10 inferences, 0.000 CPU in 0.000 seconds (69% CPU, 555556 Lips)
true ;
% 8 inferences, 0.000 CPU in 0.001 seconds (4% CPU, 235294 Lips)
false.
?- time(s([a,a,b,b,b,b,b,b,b,b,c,c],[])).
% 13 inferences, 0.000 CPU in 0.000 seconds (77% CPU, 393939 Lips)
false.

?- time(s([a,a,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,c,c],[])).
% 13 inferences, 0.000 CPU in 0.000 seconds (73% CPU, 619048 Lips)
false.
```

Homework 3 Review

- Question 2:
  - Give a counting DCG for $\{a^n b^{2n} c^{n+1} \mid n>0\}$
  - Accept:
    - abbcc
    - aabbbbccc
    - aaabbbbbbcccc
  - Reject:
    - aabbcc
    - aabbbccc
    - aaabbbbbbccc

Homework 3 Review

Availability: *built-in*

string_chars(*?String, ?Chars*)

Bi-directional conversion between a string and a list of characters. At least one of the two arguments must be instantiated.

See also: atom_chars/2.

Availability: *built-in*

atom_chars(*?Atom, ?CharList*) *[ISO]*

Similar to atom_codes/2, but *CharList* is a list of *character*s (atoms of length 1) rather than a list of *character codes* (integers denoting characters).

```
?- atom_chars(hello, X).
X = [h, e, l, l, o]
```

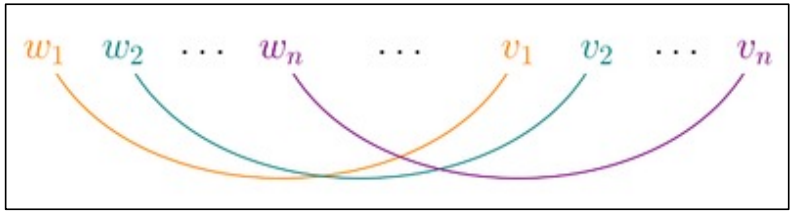
Homework 3 Review

- Desired behavior:

```
?- ['anb2ncn+1'].
true.

?- string_chars('abbcc', Cs).
Cs = [a, b, b, c, c].


?- string_chars('abbcc',L), s(L,[]).
L = [a, b, b, c, c] ;
false.

?- string_chars('aabbbbccc',L), s(L,[]).
L = [a, a, b, b, b, b, c, c, c] ;
false.
```

```
?- string_chars('aaabbbbbbcccc',L), s(L,[]).
L = [a, a, a, b, b, b, b, b, b|...] [write]
L = [a, a, a, b, b, b, b, b, b, c, c, c, c] ;
false.

?- string_chars('aabbcc',L), s(L,[]).
false.

?- string_chars('aabbbccc',L), s(L,[]).
false.

?- string_chars('aaabbbbbbccc',L), s(L,[]).
false.
```

Cross Serial Dependencies



https://en.wikipedia.org/wiki/Cross-serial_dependencies

References (*Wikipedia ones are inadequate*)

M.A.C. Huybregts, 1976, Overlapping dependencies in Dutch. *Utrecht Working papers in Linguistics* 1: 24-65.

M.A.C. Huybregts, 1984. The weak inadequacy of context-free phrase structure grammars. In: G. de Haan, M. Trommelen, & W. Zonneveld (eds.), *Van Periferie naar Kern*, 81-99. Dordrecht: Foris.

- Wikipedia:
  - In linguistics, **cross-serial dependencies** (also called **crossing dependencies** by some authors) occur when the lines representing the dependency relations between two series of words cross over each other.
  - By this fact, **Dutch** and **Swiss-German** have been proven to be non-context-free.

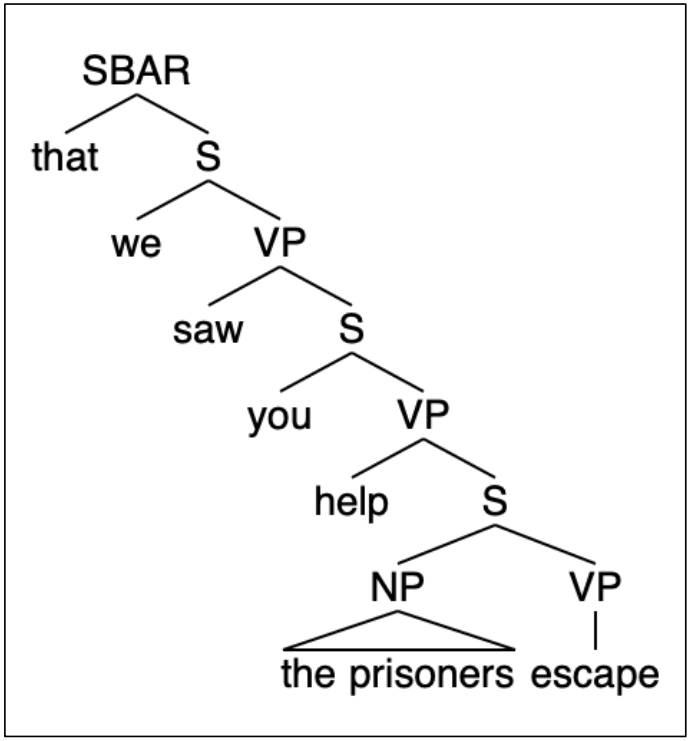
Cross Serial Dependencies

- First discussed in (Huybregts 1976), Dutch has Verb Raising (**VR**):
  - … dat we je de gevangenen zagen helpen ontsnappen
  - … that we you the prisoners saw help escape
  - … that we$_{\theta\text{-saw}}$ you$_{\theta\text{-help}}$ the prisoners$_{\theta\text{-escape}}$ saw help escape
  - argument to predicate dependencies do not appear to respect nesting.
- **VR** is obligatory for modal verbs:
  - *let, see, hear, feel, learn, teach, help.*
- Verbs like *help, teach, learn* can take bare VP (**VR**) or to-VP complements (*undergo Extraposition*).
- **Swiss German** has the same word order as **Dutch** (different from German), but Case differences are visible (unlike in Dutch, only nominative/non-nominative Case marking).
  - *that* Arg-V1 Arg-V2 Arg-V3 V1 V2 V3
  - but these associations are not visible in the surface sentence

Underlying form and surface form

- Surface form (*anglicized*):
  - dat we je de gevangenen zagen helpen ontsnappen
  - that we you the prisoners saw help escape
- Underlying hierarchical form:
  - [that [we [saw [you [help [the prisoners escape]]]]]]
- Simplified grammar (*context-free*) g1.prolog:

```
1. sbar --> [that], s.
2. s --> subject, vp.
3. subject --> [we] | [you] | [the, prisoners].
4. vp --> verb, s.
5. vp --> iverb.
6. verb --> [saw] | [help].
7. iverb --> [escape].
```

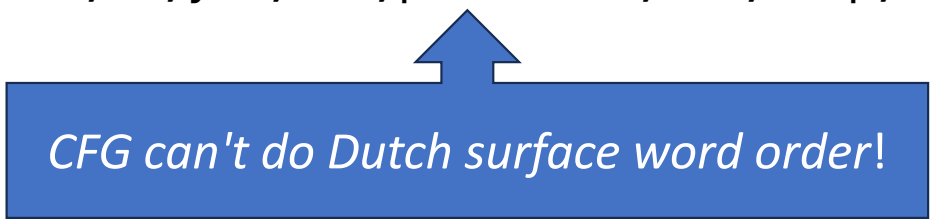


To be more precise, *the prisoners* is the dative argument of Control verb *help*

g1.prolog

- Grammar (*context-free*) g1.prolog:

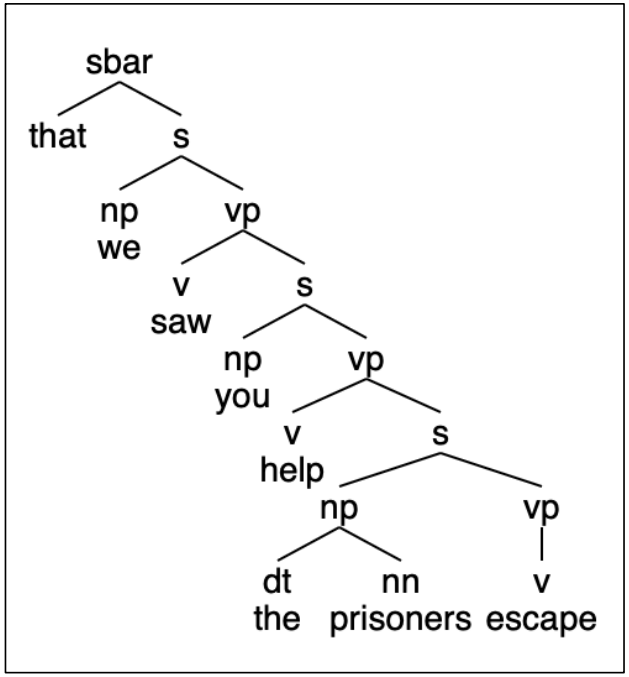
```
?- [g1].
true.

?- sbar([that,we,saw,you,help,the,prisoners,escape], []).
true ;
false.

?- sbar([that,we,you,the,prisoners,saw,help,escape], []).
false.
```



g2.prolog

- Context-Free Grammar (*CFG*) g2.prolog:
  - *same as g1.prolog but reporting a parse tree*
  - dat we je de gevangenen zagen helpen
  - ontsnappen[that,we,saw,you,help,the,prisoners,escape]

```
?- [g2].
true.

?- sbar(Parse, [that,we,saw,you,help,the,prisoners,escape],
[]).
Parse = sbar(that, s(np(we), vp(v(saw), s(np(you),
vp(v(help), s(np(dt(the), nn(prisoners)),
vp(v(escape)))))))) ;
false.

?- sbar(Parse, [that,we,you,the,prisoners,saw,help,escape],
[]).
false.
```



g2.prolog

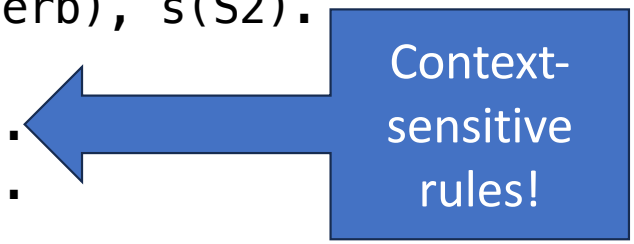
- Extra argument for each non-terminal holds the parse (*for each phrase*) – recall concept from earlier lecture (*and last semester*)
  - `sbar(sbar(that,S)) --> [that], s(S).`
  - `s(s(SBJ,VP)) --> subject(SBJ), vp(VP).`
  - `subject(np(we)) --> [we].`
  - `subject(np(you)) --> [you].`
  - `subject(np(dt(the),nn(prisoners))) --> [the,prisoners].`
  - `vp(vp(V,S)) --> verb(V), s(S).`
  - `vp(vp(V)) --> iverb(V).`
  - `verb(v(saw)) --> [saw].`
  - `verb(v(help)) --> [help].`
  - `iverb(v(escape)) --> [escape].`

So far

- Grammars `g1.prolog` and `g2.prolog` are context-free grammars
    - *limited to a single nonterminal on the LHS of a rule* (-->)
    - they do not encode the word order found in Dutch (and Swiss German)
- Let's introduce `g3.prolog`, a context-sensitive grammar
    - *will have a slightly more complex rule LHS*

g3.prolog

- Context-Sensitive Grammar (CSG) (*ternary branching*):
  - simplified for exposition: replaced *the prisoners* with *him*.
  - `sbar(sbar(that,S2)) --> [that], s(S2).`
  - `s(s(him,vp(Verb))) --> [him], iverb(Verb).`
  - `s(s(we,vp(Verb,S2))) --> [we], verb(Verb), s(S2).`
  - `s(s(you,vp(Verb,S2))) --> [you], verb(Verb), s(S2).`
  - `verb(Verb), [we] --> [we], verb(Verb).`
  - `verb(Verb), [you] --> [you], verb(Verb).`
  - `verb(Verb), [him] --> [him], verb(Verb).`
  - `verb(v(saw)) --> [saw].`
  - `verb(v(help)) --> [help].`
  - `iverb(v(escape)) --> [escape].`



g3.prolog trace

Sentence:

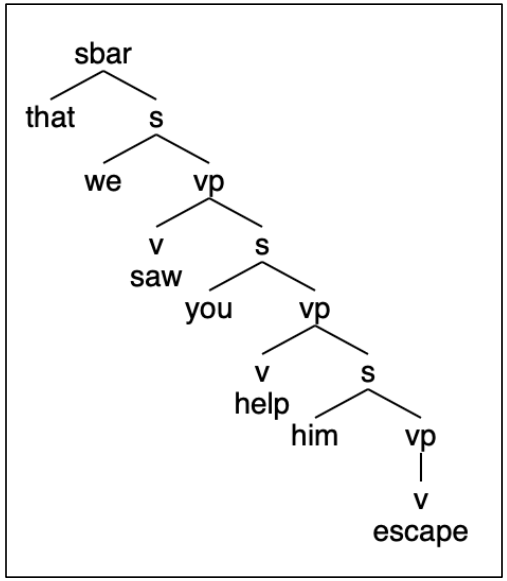
- [that, we, you, him, saw, help, escape]

```
[trace]  ?- sbar(Parse,[that, we, you, him, saw, help, escape], []).
   Call: (10) sbar(_92970, [that, we, you, him, saw, help, escape], [])
? creep
   Call: (11) s(_94400, [we, you, him, saw, help, escape], []) ? creep
   Call: (12) verb(_95224, [you, him, saw, help, escape], _95228) ?
creep
   Call: (13) verb(_95224, [him, saw, help, escape], _96044) ? creep
   Call: (14) verb(_95224, [saw, help, escape], _96860) ? creep
   Exit: (14) verb(v(saw), [saw, help, escape], [help, escape]) ? creep
   Call: (14) _96044=[him, help, escape] ? creep
   Exit: (14) [him, help, escape]=[him, help, escape] ? creep
   Exit: (13) verb(v(saw), [him, saw, help, escape], [him, help,
escape]) ? creep
   Call: (13) _95228=[you, him, help, escape] ? creep
   Exit: (13) [you, him, help, escape]=[you, him, help, escape] ? creep
   Exit: (12) verb(v(saw), [you, him, saw, help, escape], [you, him,
help, escape]) ? creep
   Call: (12) s(_95226, [you, him, help, escape], []) ? creep
   Call: (13) verb(_104196, [him, help, escape], _104200) ? creep
   Call: (14) verb(_104196, [help, escape], _105016) ? creep
   Exit: (14) verb(v(help), [help, escape], [escape]) ? creep
   Call: (14) _104200=[him, escape] ? creep
   Exit: (14) [him, escape]=[him, escape] ? creep
   Exit: (13) verb(v(help), [him, help, escape], [him, escape]) ? creep
   Call: (13) s(_104198, [him, escape], []) ? creep
   Call: (14) iverb(_109912, [escape], []) ? creep
   Exit: (14) iverb(v(escape), [escape], []) ? creep
   Exit: (13) s(s(him, vp(v(escape))), [him, escape], []) ? creep
   Exit: (12) s(s(you, vp(v(help), s(him, vp(v(escape))))), [you, him,
help, escape], []) ? creep
   Exit: (11) s(s(we, vp(v(saw), s(you, vp(v(help), s(him,
vp(v(escape)))))))), [we, you, him, saw, help, escape], []) ? creep
   Exit: (10) sbar(sbar(that, s(we, vp(v(saw), s(you, vp(v(help),
s(him, vp(v(escape))))))))), [that, we, you, him, saw, help, escape],
[]) ? creep

Parse = sbar(that, s(we, vp(v(saw), s(you, vp(v(help), s(him,
vp(v(escape))))))))) ;
```

Parse

Dutch surface word order but correct underlying form is retrieved:

- *dat we je hem zagen helpen ontsnappen*
- [that, we, you, him, saw, help, escape]
- sbar(that, s(we, vp(v(saw), s(you, vp(v(help), s(him, vp(v(escape)))))))



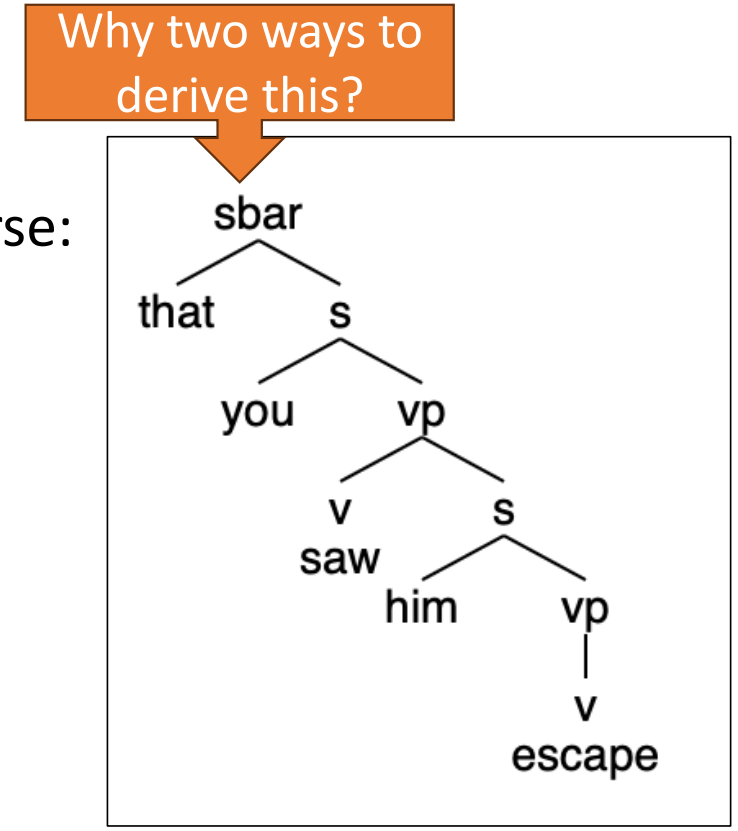
g3.prolog

Verb Raising (**VR**) is optional in g3.prolog

Problem: two surface word orders, one parse:

- *dat je hem zag ontsnappen*
- [that, you, him, saw, escape]

```
?- [g3].
true.

?- sbar(Parse, [that, you, saw, him, escape], []).
Parse = sbar(that, s(you, vp(v(saw), s(him,
vp(v(escape)))))).

?- sbar(Parse, [that, you, him, saw, escape], []).
Parse = sbar(that, s(you, vp(v(saw), s(him,
vp(v(escape)))))).
```



g4.prolog

- Force pronoun verb order swap at least once
  - verb_swap flips pronoun verb order, then calls verb_swap2
  - verb_swap2 optionally swaps
- Furthermore, let's generalize the pronouns used in g3.prolog

Grammar:

```
1.  sbar(sbar(that,S2)) --> [that], s(S2).
2.  s(s(NP,vp(Verb,S2))) --> nom_pronoun(NP), verb_swap(Verb), nnom_s(S2).

3.  nnom_s(s(NP,vp(Verb,S2))) --> nnom_pronoun(NP), verb_swap(Verb), acc_s(S2).
4.  nnom_s(s(NP,vp(Verb))) --> nnom_pronoun(NP), iverb(Verb).

5.  verb_swap(Verb), [Pronoun] -->  pronoun(np(Pronoun)), verb_swap2(Verb).

6.  verb_swap2(Verb), [Pronoun] -->  pronoun(np(Pronoun)), verb_swap2(Verb).
7.  verb_swap2(Verb) --> verb(Verb).
```

g4.prolog

- Verbs:

```
verb(v(see)) --> [see].
verb(v(let)) --> [let].
verb(v(help)) --> [help].

iverb(v(escape)) --> [escape].
```

- Pronouns:

```
pronoun(np(i)) --> [i].
pronoun(np(me)) --> [me].
pronoun(np(we)) --> [we].
pronoun(np(us)) --> [us].
pronoun(np(you)) --> [you].
pronoun(np(he)) --> [he].
pronoun(np(him)) --> [him].
pronoun(np(she)) --> [she].
pronoun(np(her)) --> [her].
pronoun(np(they)) --> [they].
pronoun(np(them)) --> [them].
```

g4.prolog

- Nominative pronouns:

```
nom_pronoun(np(i)) --> [i].
nom_pronoun(np(we)) --> [we].
nom_pronoun(np(you)) --> [you].
nom_pronoun(np(he)) --> [he].
nom_pronoun(np(she)) --> [she].
nom_pronoun(np(they)) --> [they].
```

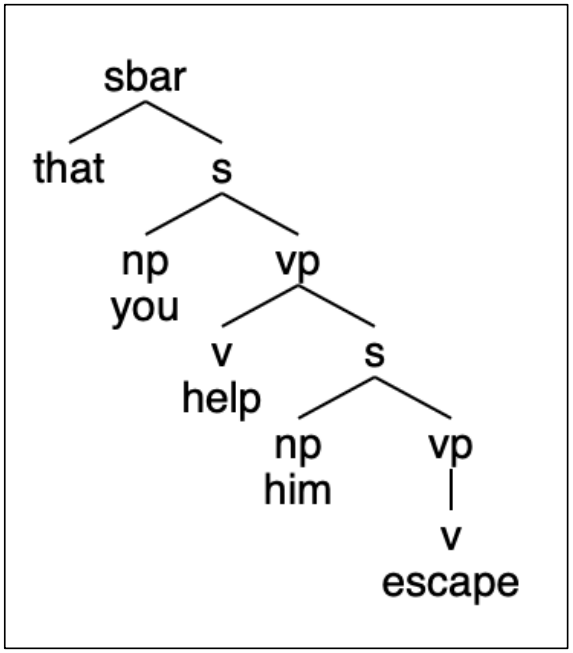
- Non-nominative pronouns:

```
nnom_pronoun(np(me)) --> [me].
nnom_pronoun(np(us)) --> [us].
nnom_pronoun(np(you)) --> [you].
nnom_pronoun(np(him)) --> [him].
nnom_pronoun(np(her)) --> [her].
nnom_pronoun(np(them)) --> [them].
```

g4.prolog

- Example:
  - dat je hem helpt ontsnappen
  - [that, you, him, help, escape]

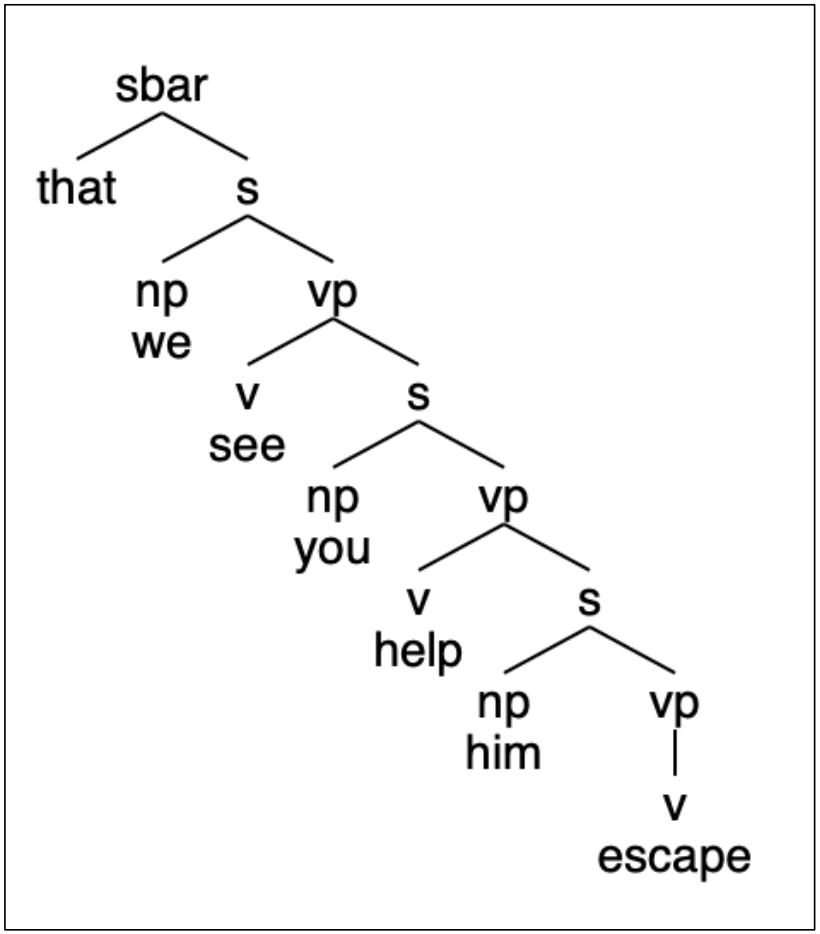
```
?- sbar(Parse, [that, you, him, help, escape], []).

Parse = sbar(that, s(np(you), vp(v(help), s(np(him),
vp(v(escape))))))) ;

false.


?- sbar(Parse, [that, you, help, him, escape], []).

false.
```



g4.prolog

- Example:
  - dat we je hem zien helpen ontsnappen
  - [that, we, you, him, see, help, escape]

```
?- sbar(Parse, [that, we, you, him, see, help,
escape], []).

Parse = sbar(that, s(np(we), vp(v(see),
s(np(you), vp(v(help), s(np(him),
vp(v(escape)))))))) ;

false.

?- sbar(Parse, [that, we, see, you, help, him,
escape], []).

false.
```

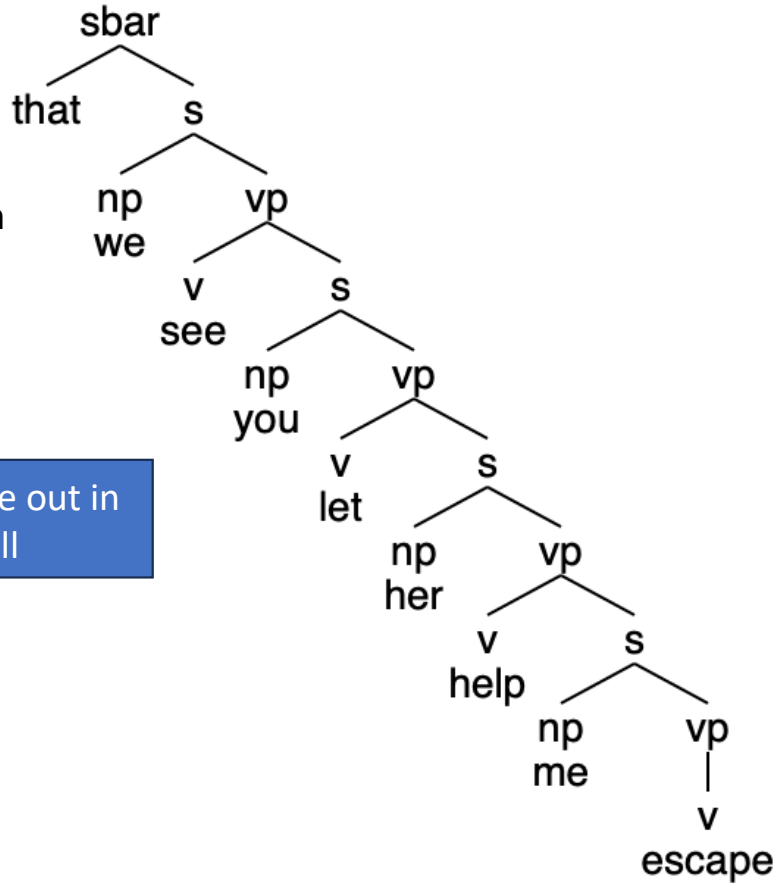


g4.prolog

- Example:
  - dat we je haar mij zien laten helpen ontsnappen
  - [that, we, you, her, me, see, let, help, escape]

```
?- sbar(Parse, [that, we, you, her, me, see, let, help,
escape], []).

Parse = sbar(that, s(np(we), vp(v(see), s(np(you),
vp(v(let), s(np(her), vp(v(help), s(np(me),
vp(v(...)))))))))) [write]

Parse = sbar(that, s(np(we), vp(v(see), s(np(you),
vp(v(let), s(np(her), vp(v(help), s(np(me),
vp(v(escape)))))))))) ;

false.


?- sbar(Parse, [that, we, see, you, let, her, help, me,
escape], []).

false.
```

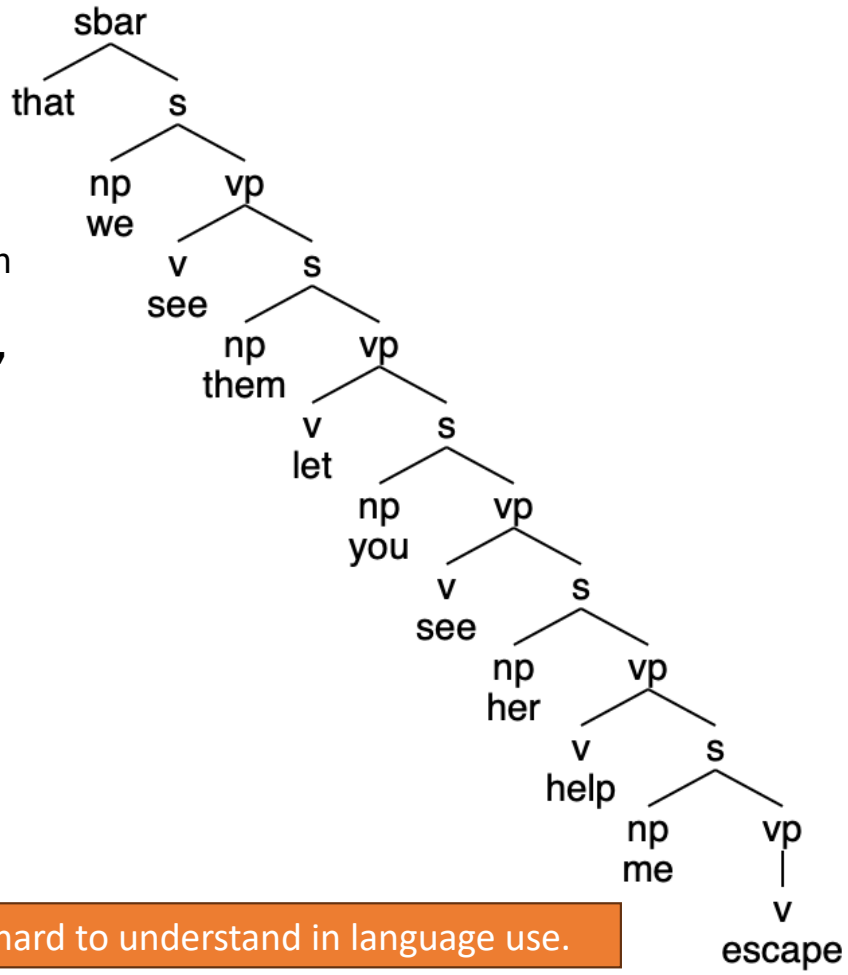
w = write out in full



g4.prolog

- Example:
  - `dat we hen je haar mij zien laten zien helpen ontsnappen`
  - `[that, we, them, you, her, me, see, let, see, help, escape]`

```
?- sbar(Parse, [that, we, them, you, her, me, see, let,
see, help, escape], []).

Parse = sbar(that, s(np(we), vp(v(see), s(np(them),
vp(v(let), s(np(you), vp(v(see), s(np(her), vp(v(help),
s(np(me), vp(v(escape))))))))))) ;

false.


?- sbar(Parse, [that, we, see, them, let, you, help, me,
escape], []).

false.
```



Huybregts: of course, these [last] two cases become very hard to understand in language use.