

# LING/C SC 581:

## Advanced Computational Linguistics

Lecture 9

Prof. Sandiway Fong

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

# Today's Topics

- Homework 5 Review
- An example of context-sensitivity in natural language

# Homework 5 Review

- Question 1:

- Give a revised version grammar that counts a's up, b's and c's down.

- Original: `abc_count.prolog`

- `s --> a(X), b(X), c(X).`

- `a(1) --> [a].`

- `a(N) --> [a], a(M), {N is M+1}.`

- Note: X is valued by a/1, b/1 (and c/1) receive a natural number

- `b(1) --> [b].`

- Subtract 1 from N, but make sure we don't over-subtract.

- `b(N) --> [b], {M is N-1, M>=1}, b(M).`

# Homework 5 Review

```
?- [abc_count].
```

```
true.
```

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

```
% 63 inferences, 0.000 CPU in 0.000 seconds (67% CPU, 1235294 Lips)
```

```
false.
```

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

```
% 217 inferences, 0.000 CPU in 0.000 seconds (77% CPU, 1486301 Lips)
```

```
false.
```

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

```
% 220 inferences, 0.000 CPU in 0.000 seconds (77% CPU, 1366460 Lips)
```

```
false.
```

# Homework 5 Review

```
?- [hw5q1].
```

```
true.
```

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

```
% 11 inferences, 0.000 CPU in 0.000 seconds (81% CPU, 647059 Lips)
```

```
false.
```

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

```
% 10 inferences, 0.000 CPU in 0.000 seconds (88% CPU, 476190 Lips)
```

```
false.
```

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

```
% 13 inferences, 0.000 CPU in 0.000 seconds (54% CPU, 520000 Lips)
```

```
false.
```

# Homework 5 Review

- Question 2: enumeration.

?- s(S, []).

S = [a, b, c] ;

S = [a, a, b, b, c, c] ;

S = [a, a, a, b, b, b, c, c, c] ;

S = [a, a, a, a, b, b, b, b, c|...] ;

S = [a, a, a, a, a, b, b, b, b|...] ;

S = [a, a, a, a, a, a, b, b, b|...] [write]

S = [a, a, a, a, a, a, b, b, b, b, b, b, c, c, c, c, c, c] .

# Relevant Slide from an earlier lecture

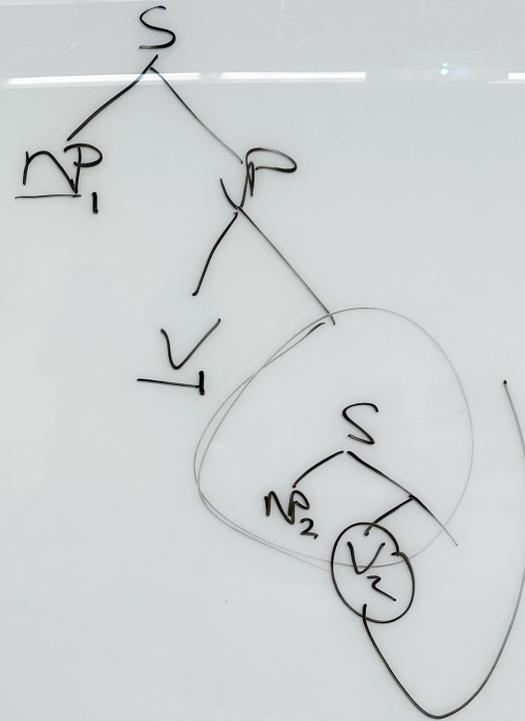
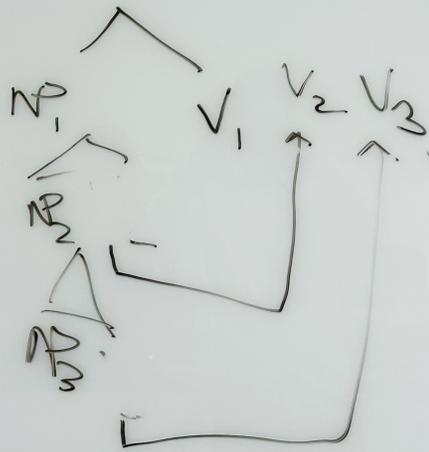
- Extra arguments are powerful
  - they allow us to impose (grammatical) constraints and change the expressive power of the system
    - *if used as read-able memory*
- **Example:**
  - $a^n b^n c^n$   $n > 0$  is not a context-free language (type-2, Chomsky hierarchy)
  - *i.e. you cannot write rules of the form  $X \rightarrow \text{RHS}$  that generate this language*
  - in fact, it's context-sensitive (type-1, Chomsky hierarchy)



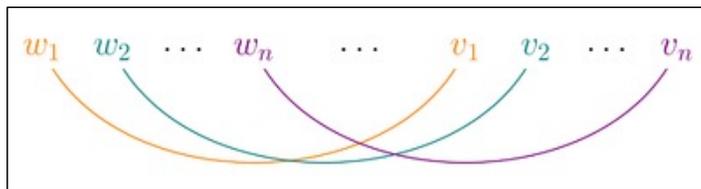


# From the whiteboard

$\theta$   
 $1 = \text{Successor}(\theta)$   
 $s(\theta)$   
 $2 = s(s(\theta))$   
 $\vdots$



# Cross Serial Dependencies



[https://en.wikipedia.org/wiki/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:

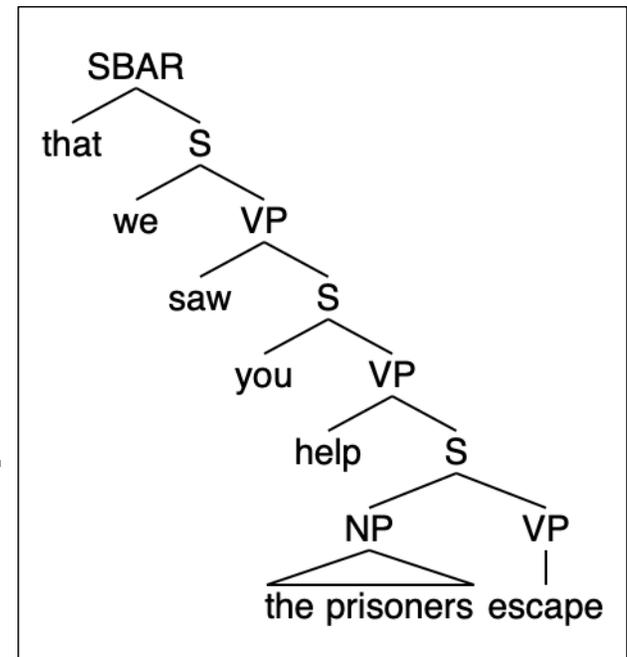
- In linguistics, **cross-serial dependencies** (also called **crossing dependencies** by some authors) occur when the lines representing the dependency relations between two series of words cross over each other.
- By this fact, **Dutch** and **Swiss-German** have been proven to be non-context-free.

# 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<sub>θ-saw</sub> you<sub>θ-help</sub> the prisoners<sub>θ-escape</sub> 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.
```



*CFG can't do Dutch surface word order!*

# 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.**

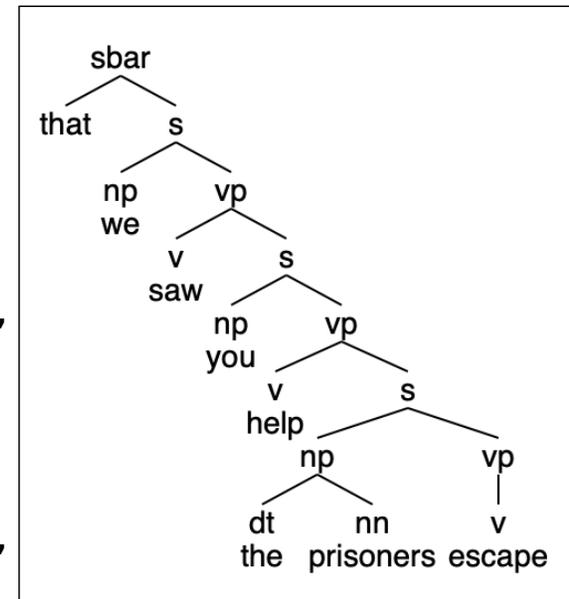
?- sbar(Parse, [that,we,saw,you,help,the,prisoners,escape], []).

```
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].`



Context-sensitive rules!

# 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

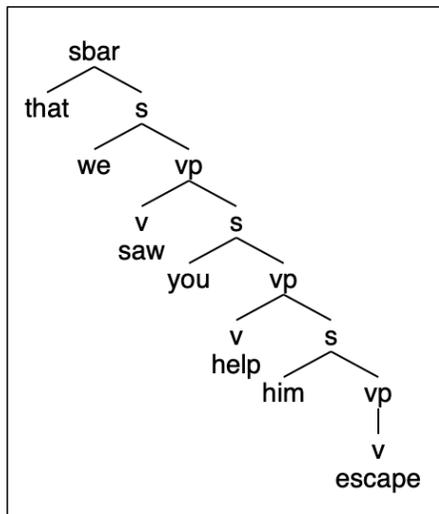
**Exit:** (10) sbar(sbar(that, s(we, vp(v(saw), s(you, vp(v(help), s(him, vp(v(escape))))))), [that, we, you, him, saw, help, escape], []) ? creep

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

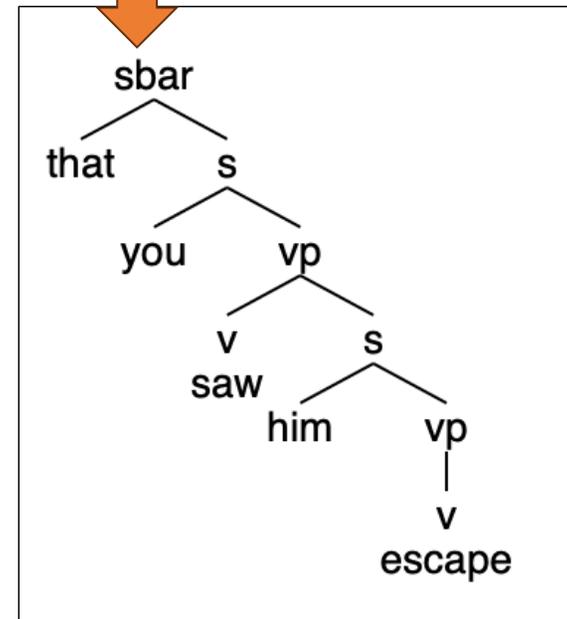
**Problem:** two surface word orders, one parse:

- *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)))))).
```

Why two ways to derive this?



## 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:

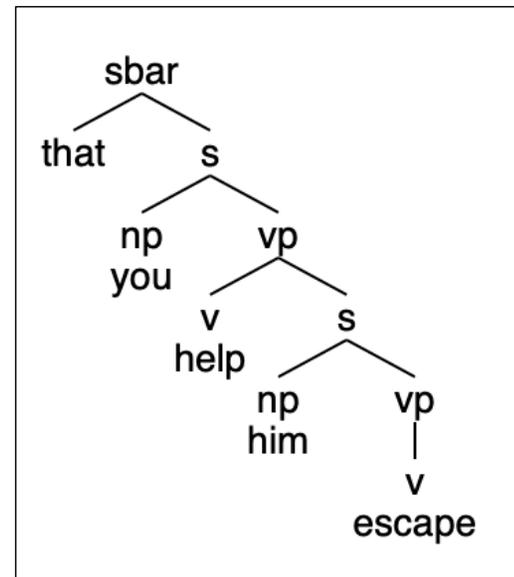
- dat je hem helpt ontsnappen
- [that, you, him, help, escape]

```
?- 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]

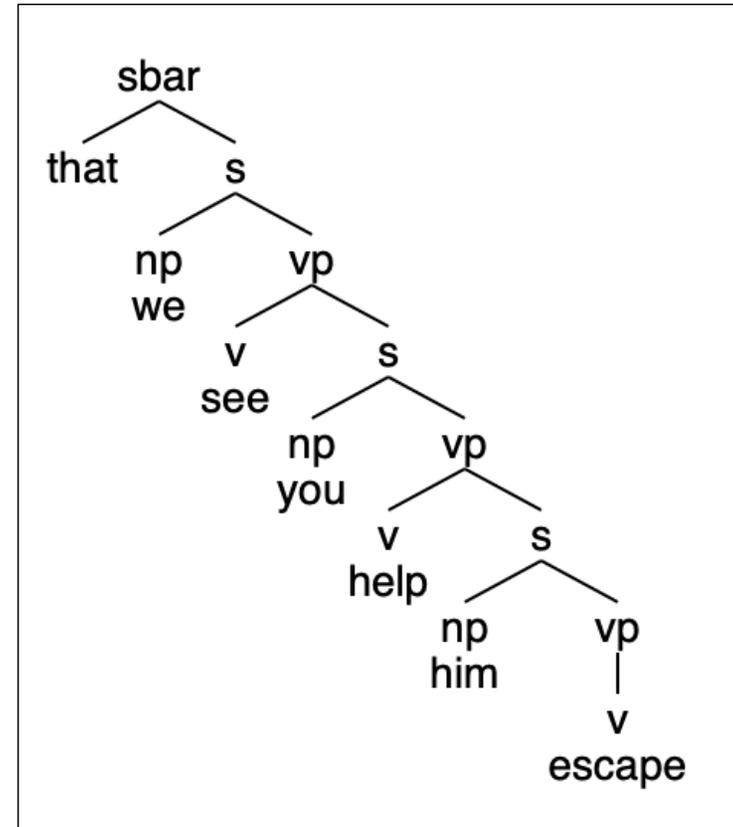
?- sbar(Parse, [that, we, you, him, see, help, escape], []).

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]

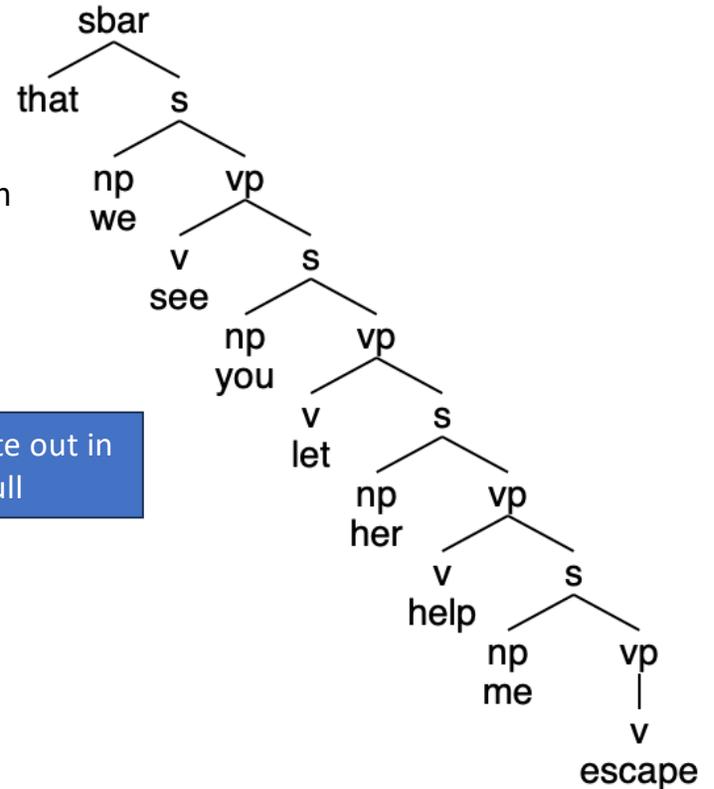
w = write out in full

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.**



# 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.

