Abstract Syntax and Universal Dependencies

Aarne Ranta
Joint work with Prasanth Kolachina

University of Malta, 4 April 2017

Structural representations

- defining the level of abstraction

In computational linguistics, everyone needs structures

- to manipulate symbolically
- to analyse statistically

token strings

POS tagged lemma sequences

token strings

parse trees

POS tagged lemma sequences

token strings

parse trees **dependency trees**

POS tagged lemma sequences

token strings

logical forms

parse trees **dependency trees**

POS tagged lemma sequences

token strings

logical forms

abstract syntax trees

parse trees **dependency trees**

POS tagged lemma sequences

token strings

Abstract syntax tree (AST)

between parse trees and logical forms

pure constituency

parts + how they are put together

Abstract syntax tree (AST)

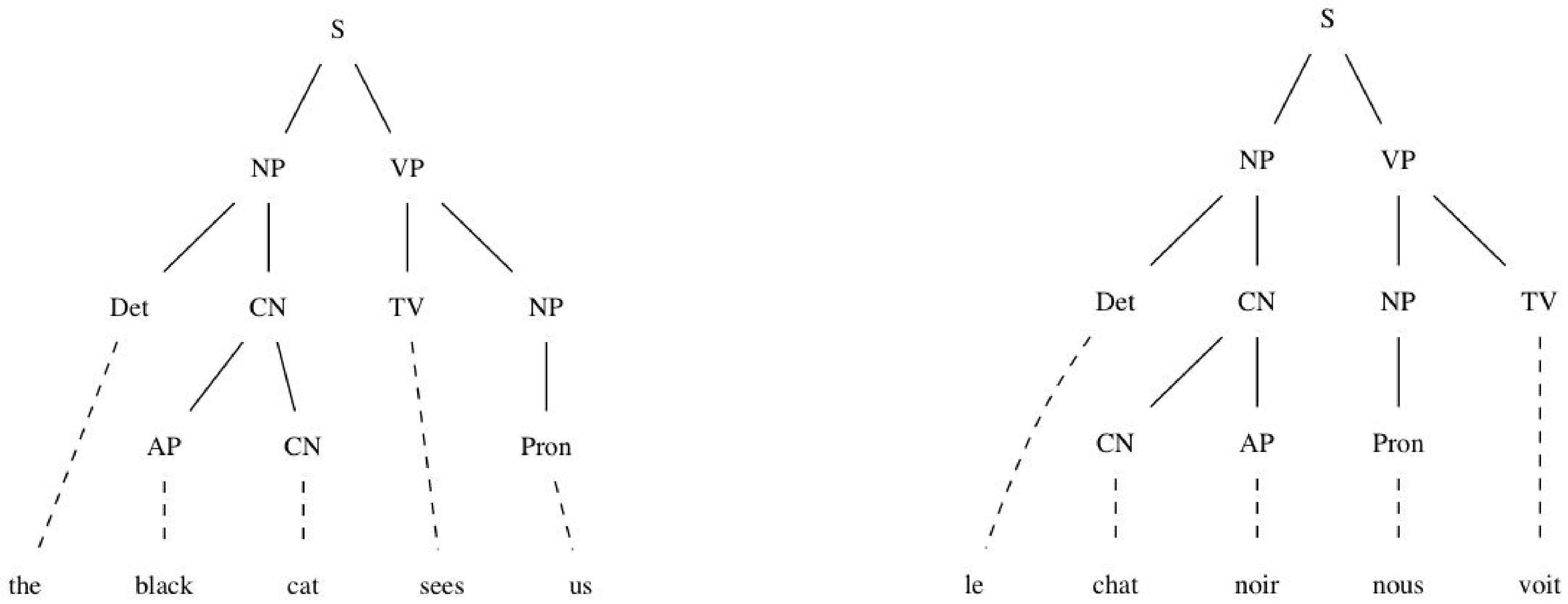
between parse trees and logical forms

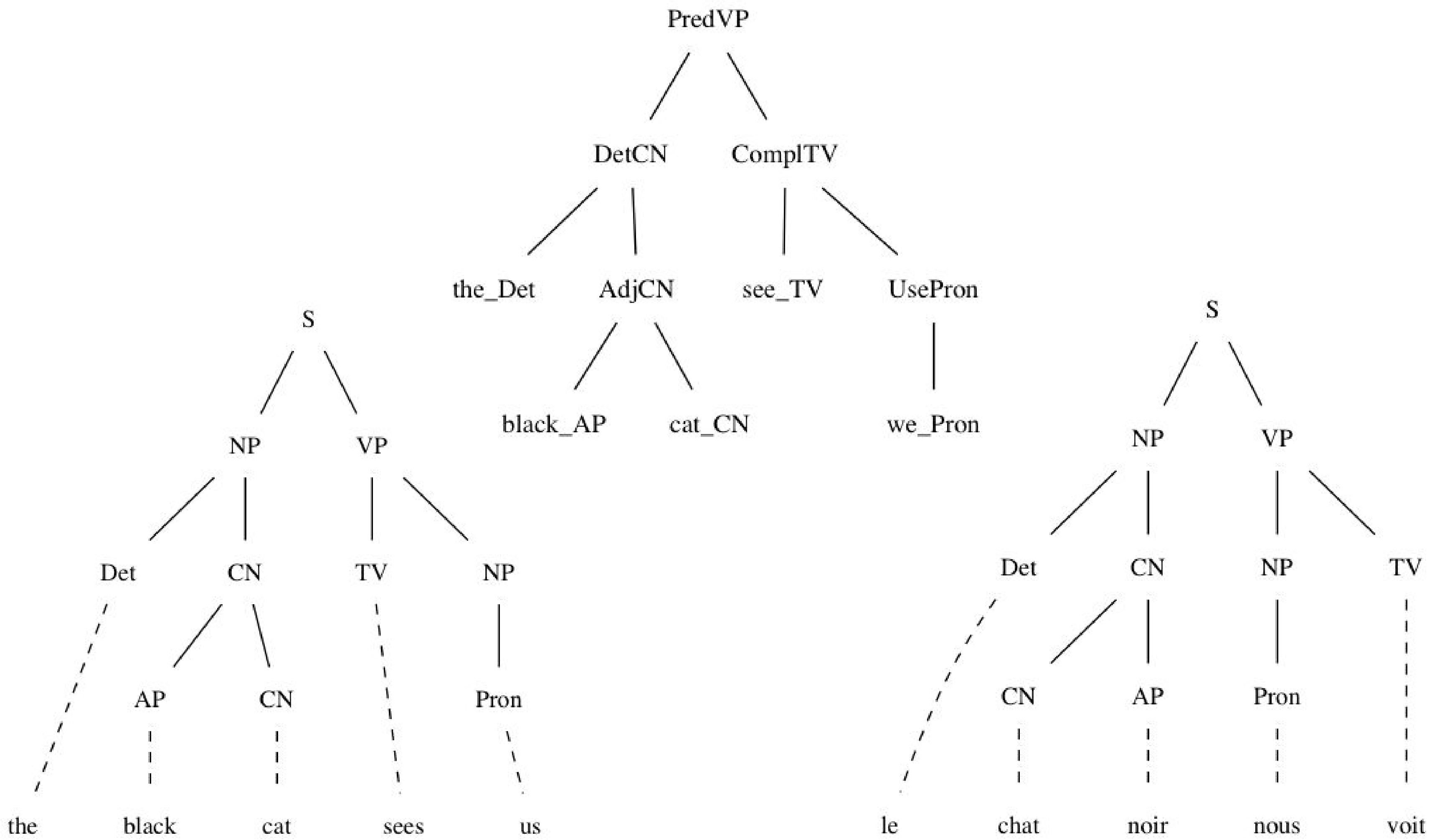
pure constituency

parts + how they are put together

not: order of parts

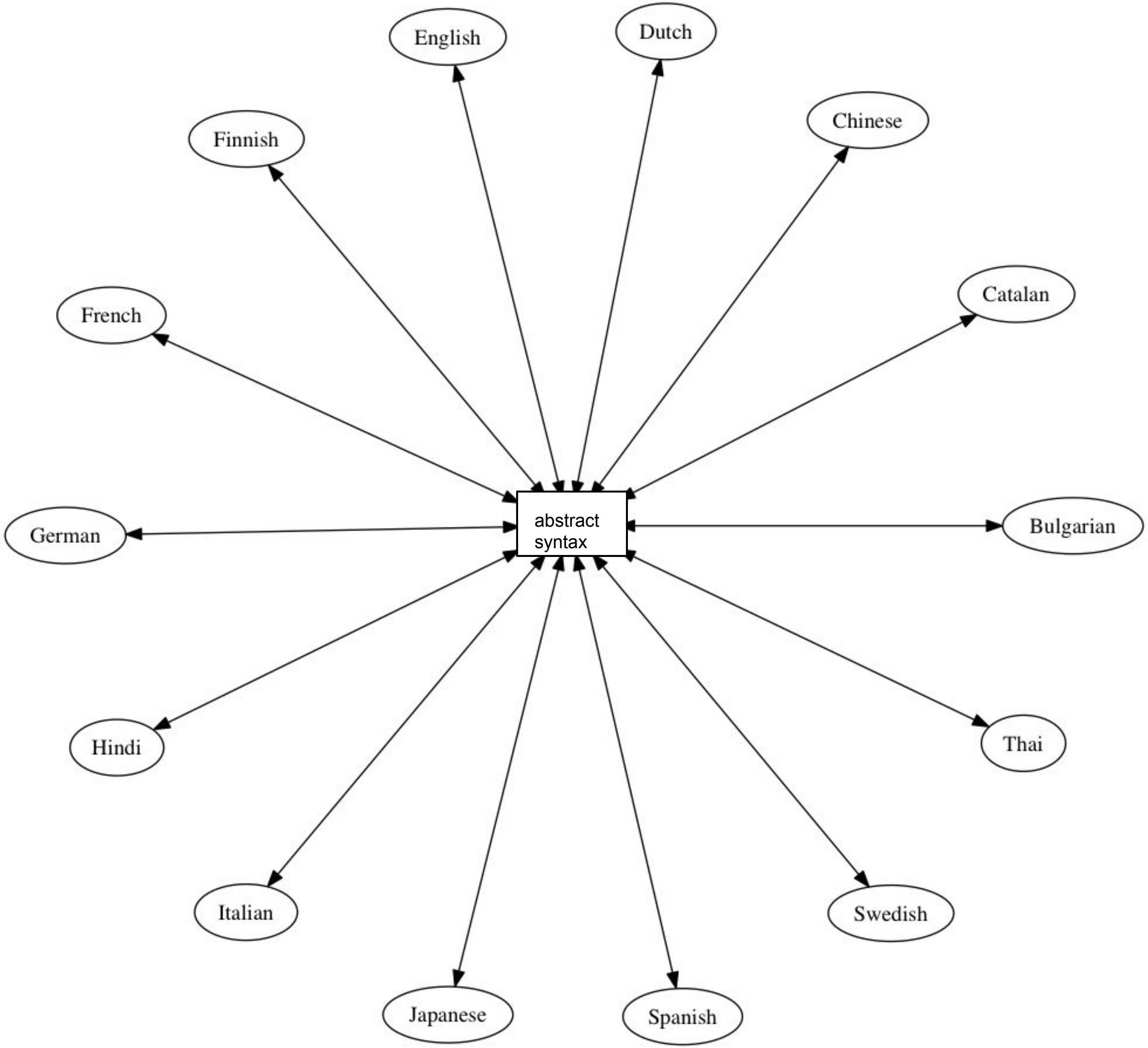
not: what the parts look like


S
NP
VP
Det
CN
TV
NP
AP
CN
Pron
the
black
cat
sees
us
S
NP
VP
Det
CN
NP
TV
CN
AP
Pron
le
chat
noir
nous
voit


PredVP
DetCN
ComplTV
the_Det
AdjCN
see_TV
UsePron
black_AP
cat_CN
we_Pron
S
NP
VP
Det
CN
TV
NP
AP
CN
Pron
the
black
cat
sees
us
S
NP
VP
Det
CN
NP
TV
CN
AP
Pron
le
chat
noir
nous
voit

Translation interlingua

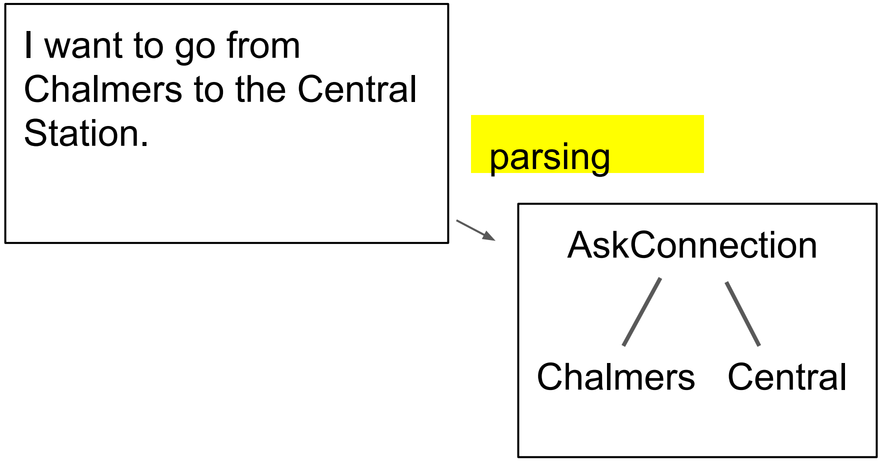
- defining the meaning to be preserved

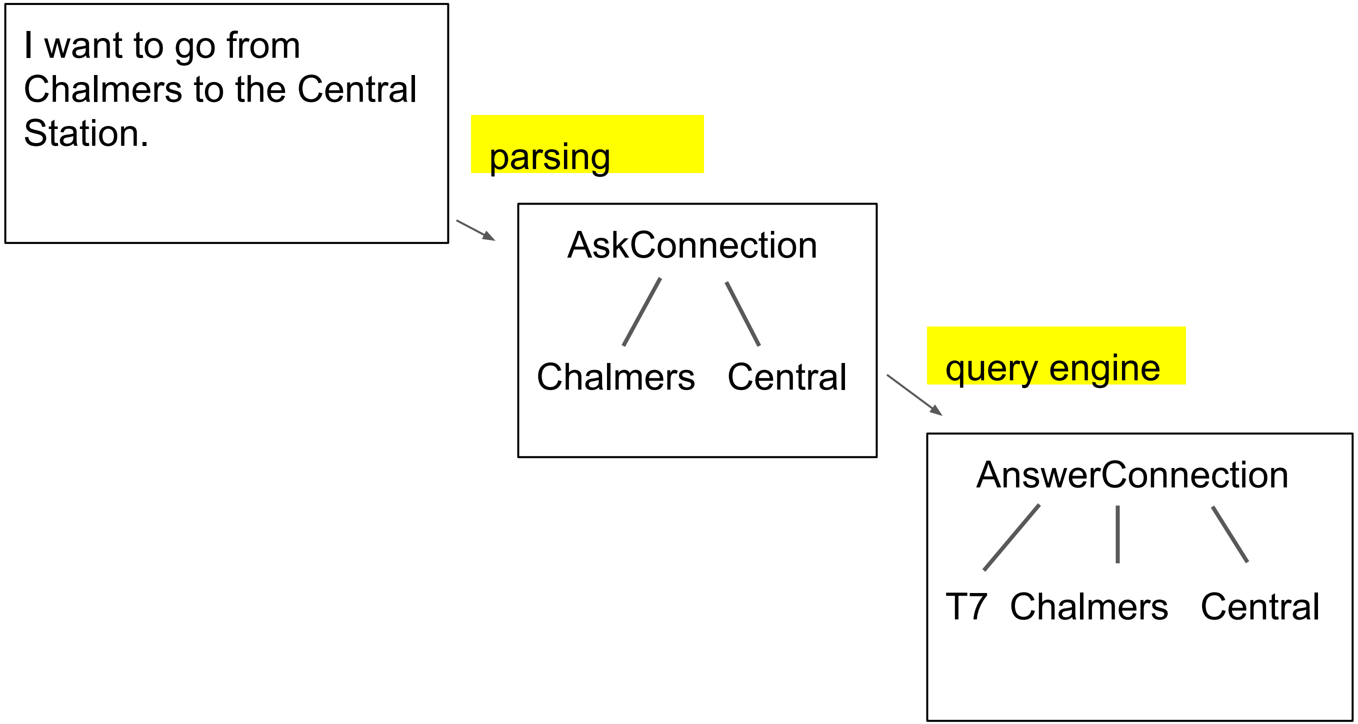

English
Dutch
Finnish
Chinese
French
Catalan
German
abstract
syntax
Bulgarian
Hindi
Thai
Italian
Swedish
Japanese
Spanish

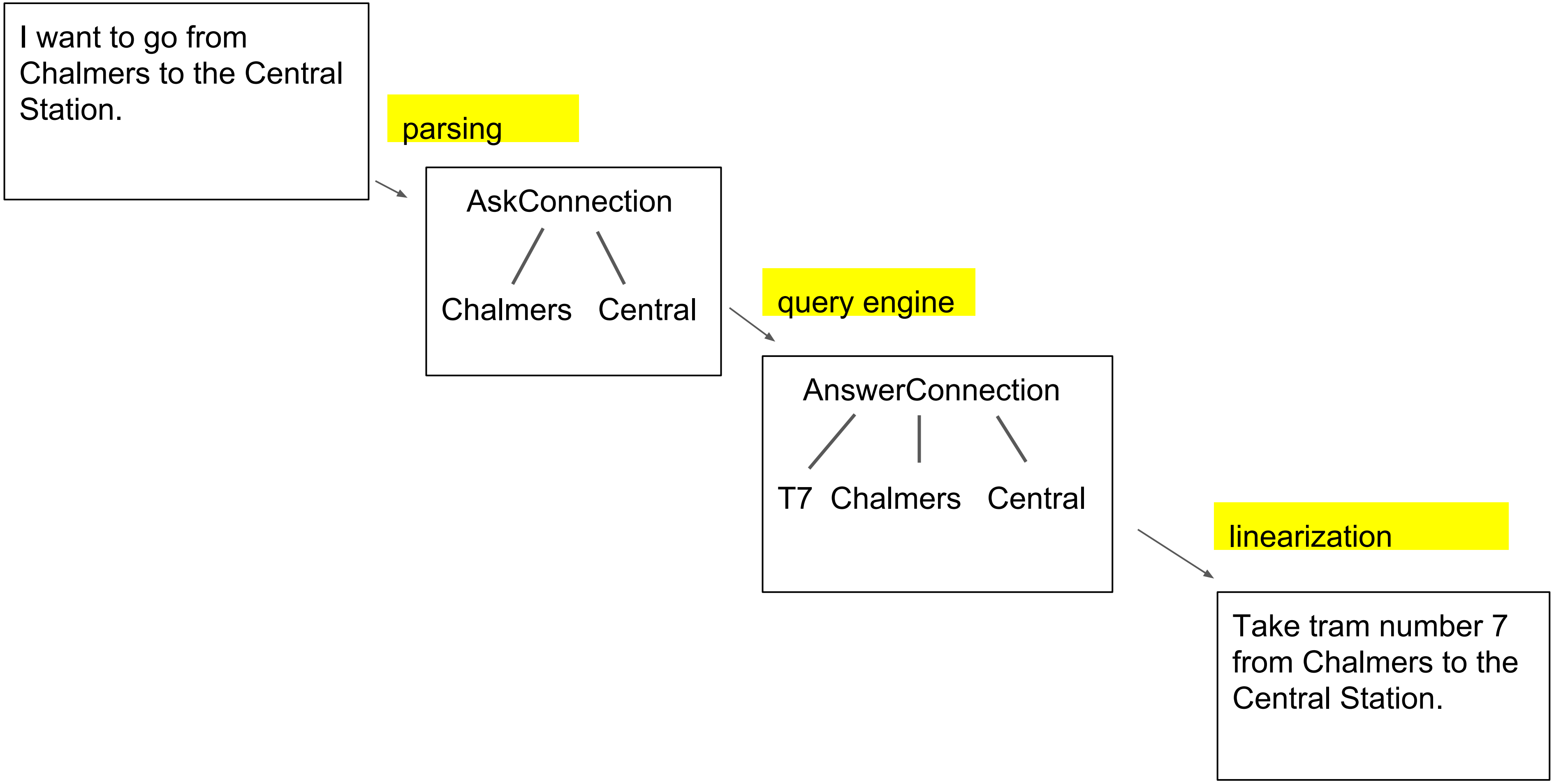
Shared semantics

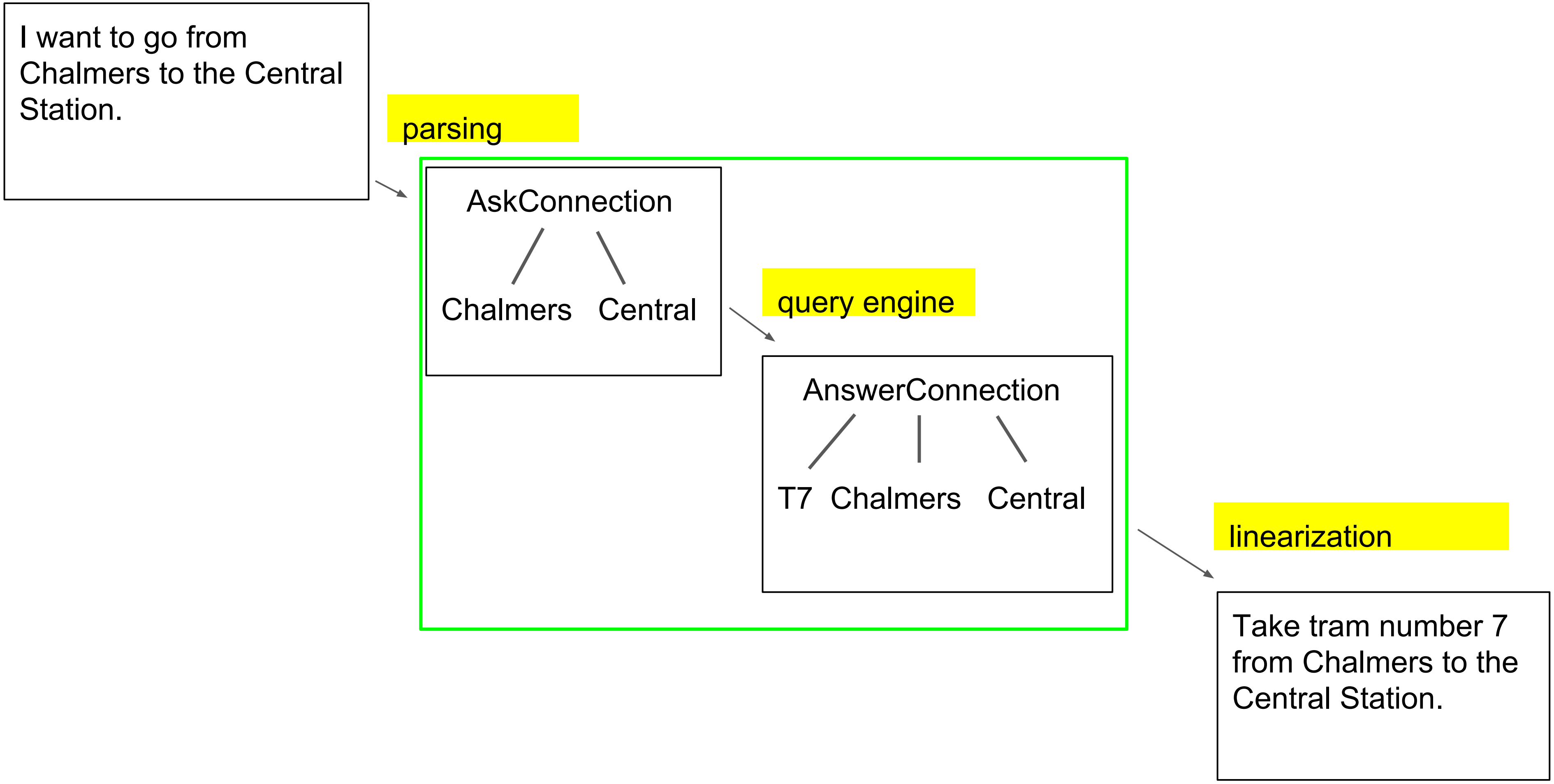
- to do semantics for many languages at once

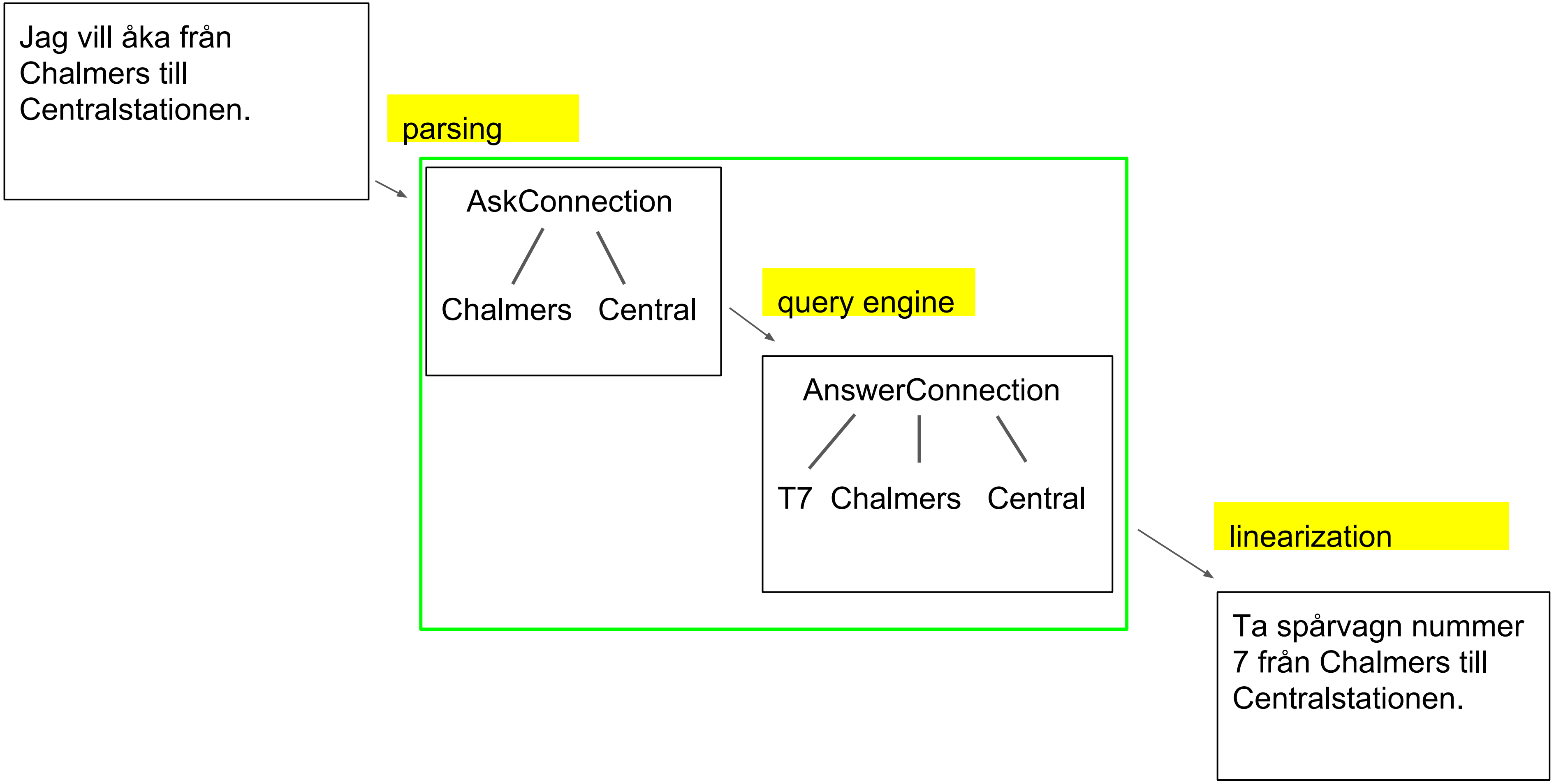
I want to go from Chalmers to the Central Station.

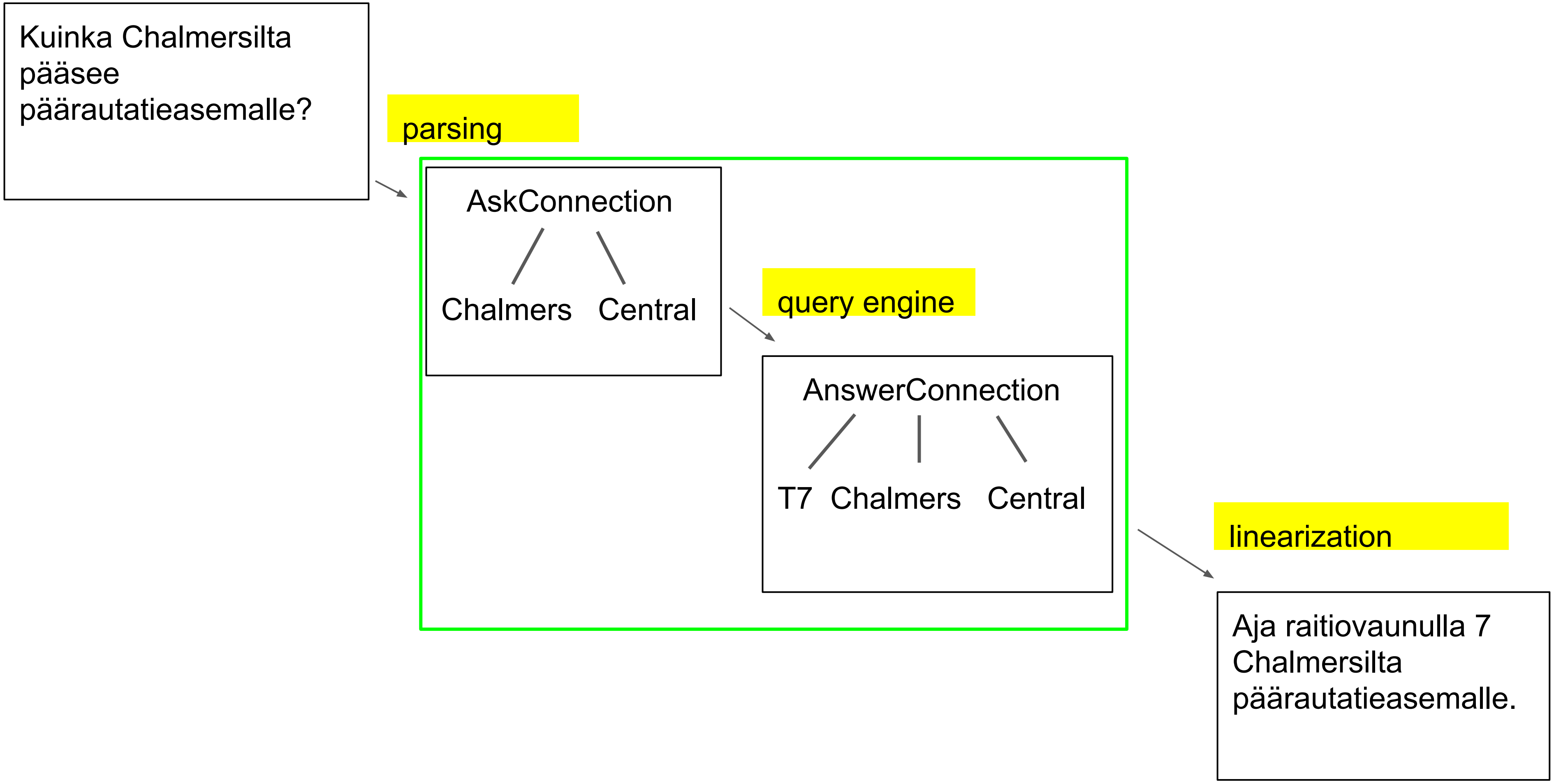

I want to go from Chalmers to the Central Station.
parsing
AskConnection
Chalmers
Central


I want to go from Chalmers to the Central Station.
parsing
AskConnection
Chalmers
Central
query engine
AnswerConnection
T7
Chalmers
Central


I want to go from Chalmers to the Central Station.
parsing
AskConnection
Chalmers
Central
query engine
AnswerConnection
T7
Chalmers
Central
linearization
Take tram number 7 from Chalmers to the Central Station.


I want to go from Chalmers to the Central Station.
parsing
AskConnection
Chalmers
Central
query engine
AnswerConnection
T7
Chalmers
Central
linearization
Take tram number 7 from Chalmers to the Central Station.


Jag vill åka från Chalmers till Centralstationen.
parsing
AskConnection
Chalmers
Central
query engine
AnswerConnection
T7
Chalmers
Central
linearization
Ta spårvagn nummer 7 från Chalmers till Centralstationen.


Kuinka Chalmersilta pääsee päärautatieasemalle?
parsing
AskConnection
Chalmers
Central
query engine
AnswerConnection
T7
Chalmers
Central
linearization
Aja raitiovaunulla 7 Chalmersilta päärautatieasemalle.

Defining abstract and concrete syntax

- GF, a dedicated formalism for this task

GF = Grammatical Framework

LF = Logical Framework = Constructive Type Theory

GF = LF + Concrete Syntax

Xerox XRCE 1998, now open source (GPL/LGPL/BSD)

Abstracting away from...

words

word order

Abstracting away from...

words

word order

morphology

```
abstract Grammar

cat

  CN

  AP

fun

  AdjCN : AP -> CN -> CN

  cat_CN : CN

  black_AP : AP
```

```
abstract Grammar

cat

  CN

  AP

fun

  AdjCN : AP -> CN -> CN

  cat_CN : CN

  black_AP : AP
```

```
concrete GrammarEng

lincat
CN = Number => Str
AP = Str

lin
AdjCN ap cn = \\n => ap ++ cn ! n
cat_CN = table {Sg => “cat” ; Pl => “cats”}
black_AP = “black”
```

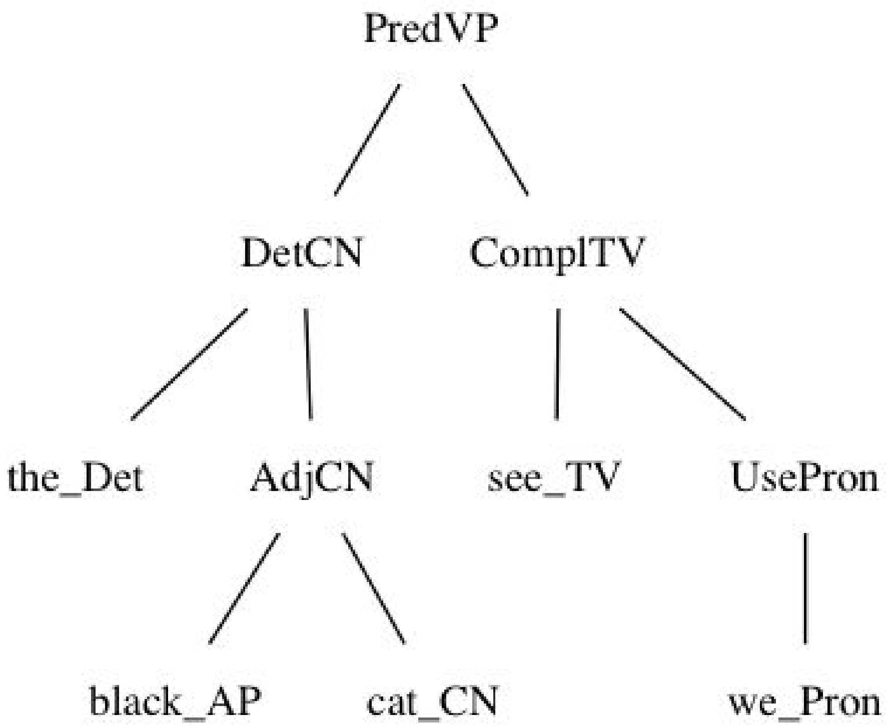
```
abstract Grammar

cat

  CN

  AP

fun

  AdjCN : AP -> CN -> CN

  cat_CN : CN

  black_AP : AP
```

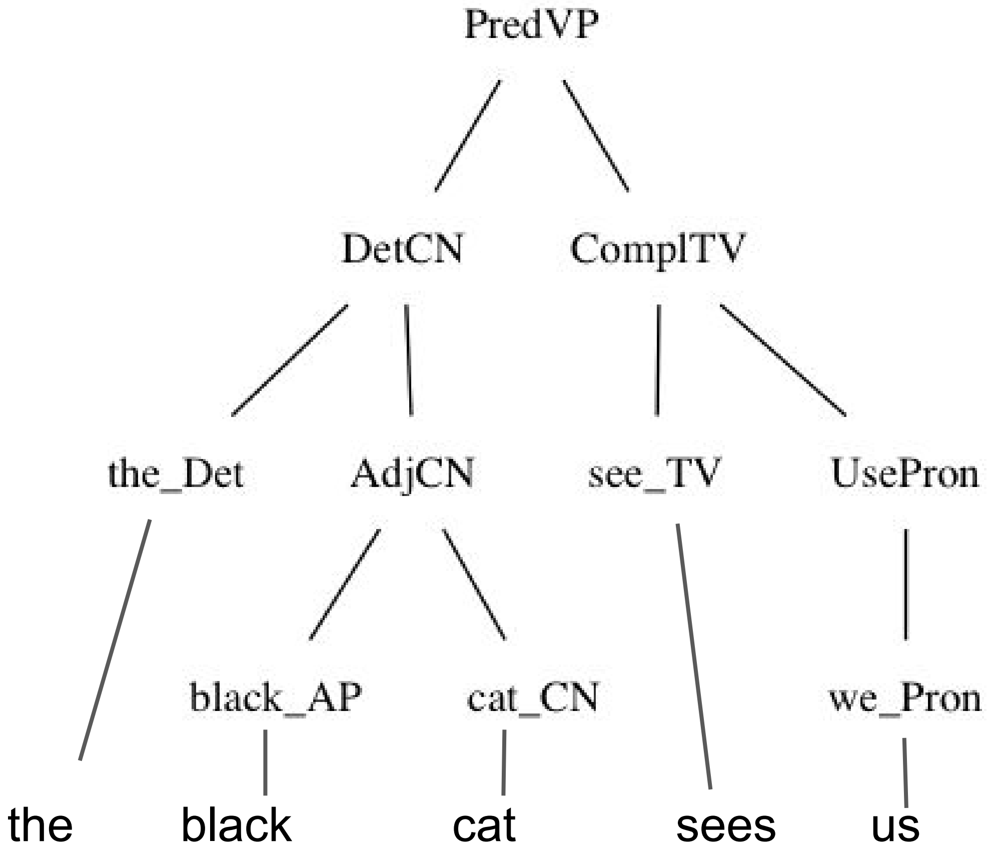
```
concrete GrammarEng

lincat
CN = Number => Str
AP = Str

lin
AdjCN ap cn = \\n => ap ++ cn ! n
cat_CN = table {Sg => “cat” ; Pl => “cats”}
black_AP = “black”
```

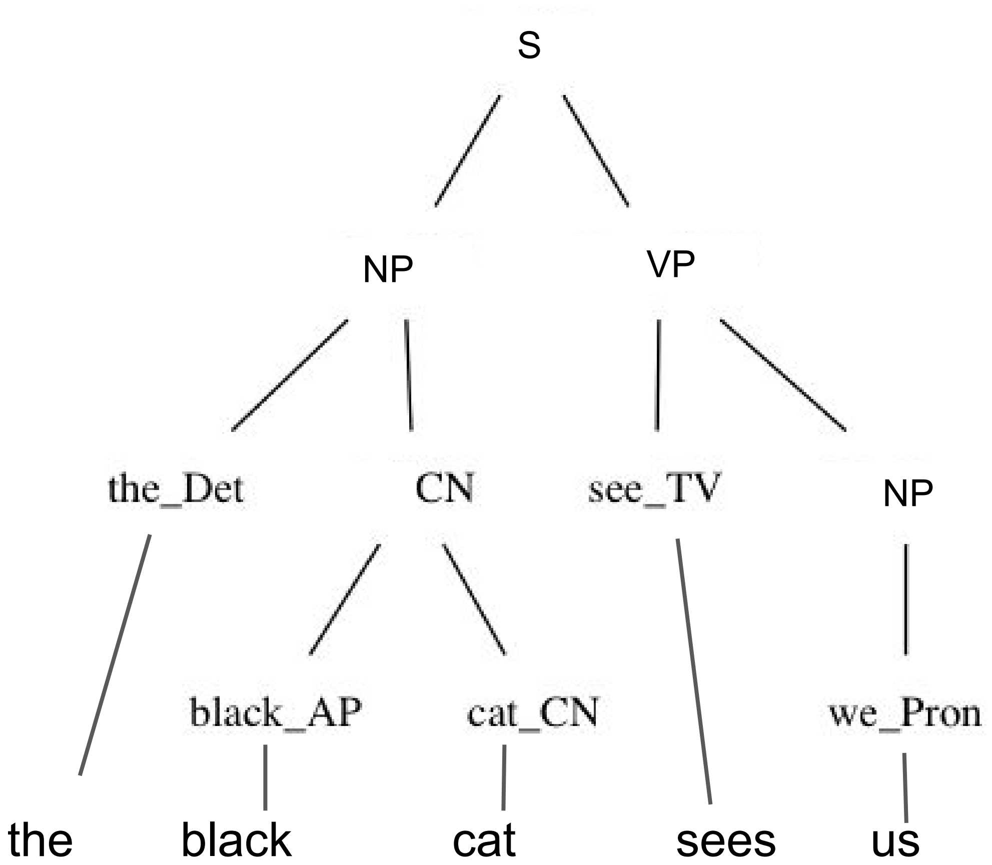
```
concrete GrammarFre

lincat
CN = {s : Number => Str ; g : Gender}
AP = Gender => Number => Str

lin
AdjCN ap cn =
{s =\\n => cn ! n ++ ap ! cn.g ! n ; g = cn.g}
cat_CN =
{s = table {Sg => “chat” ; Pl =>“chats”} ; g = Masc}
black_AP = s = table {Masc => table {Sg => “noir” ; ...
```

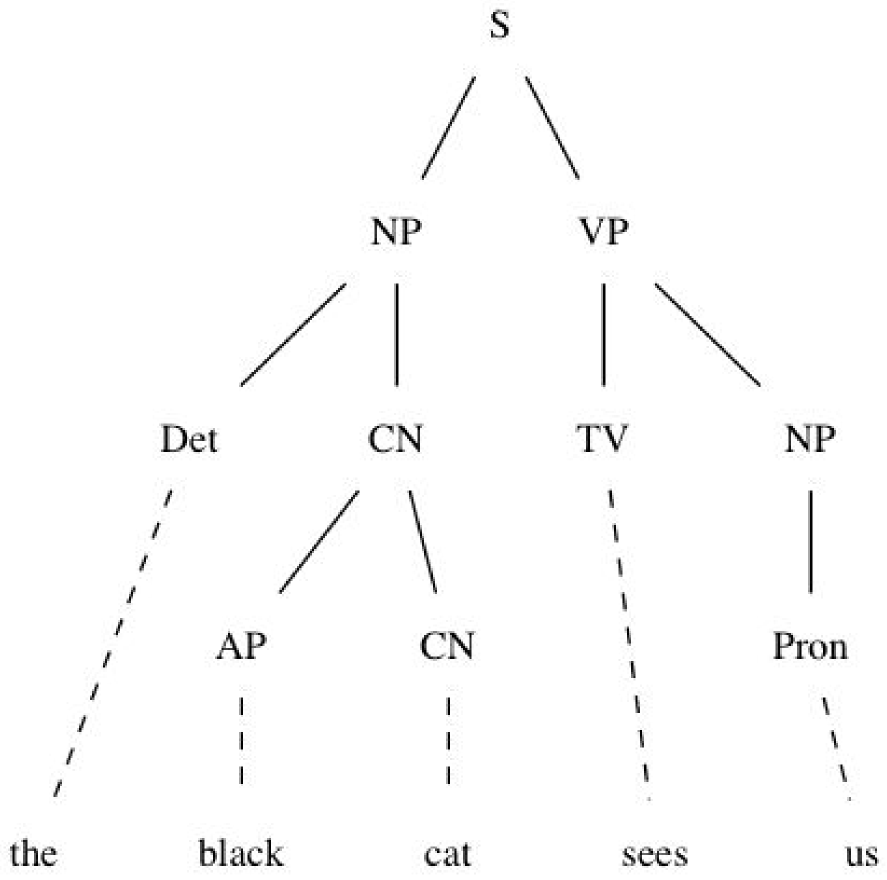
From AST to parse tree

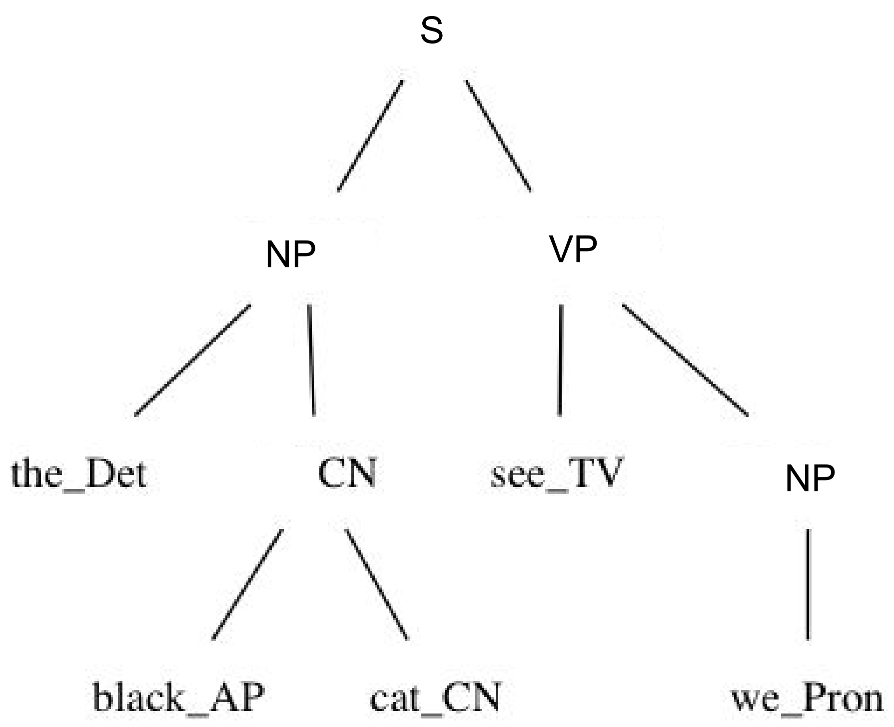
- link words to subtrees and hide functions

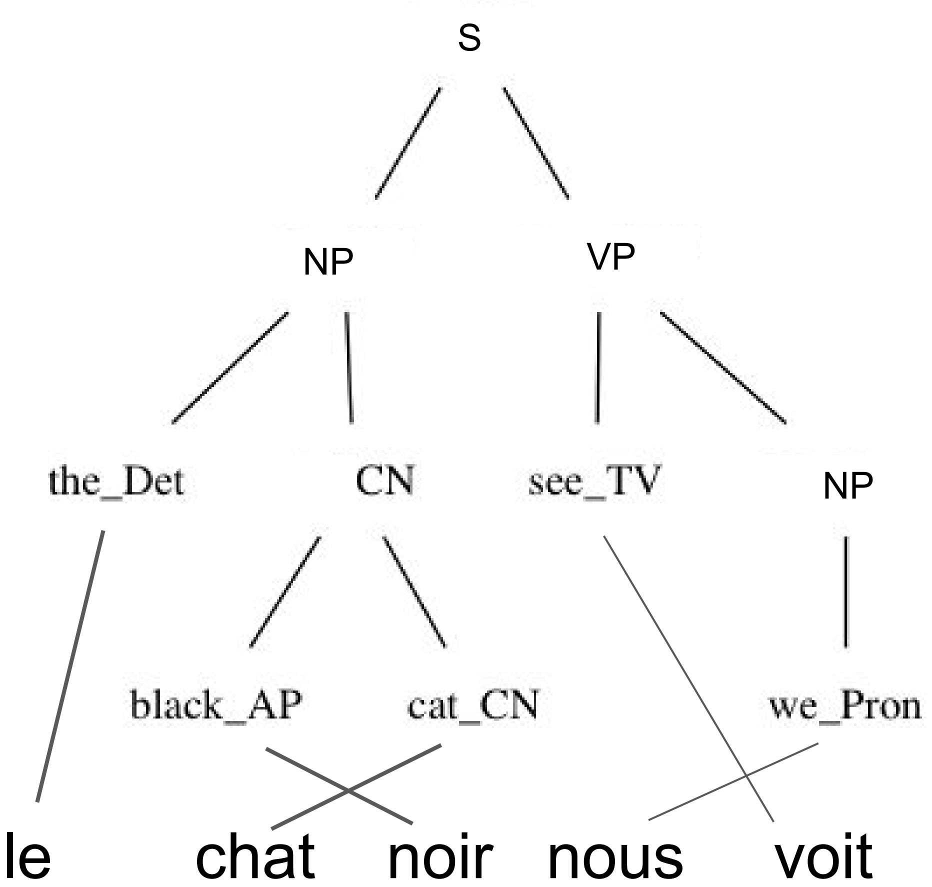

PredVP
DetCN
ComplTV
the_Det
AdjCN
see_TV
UsePron
black_AP
cat_CN
we_Pron

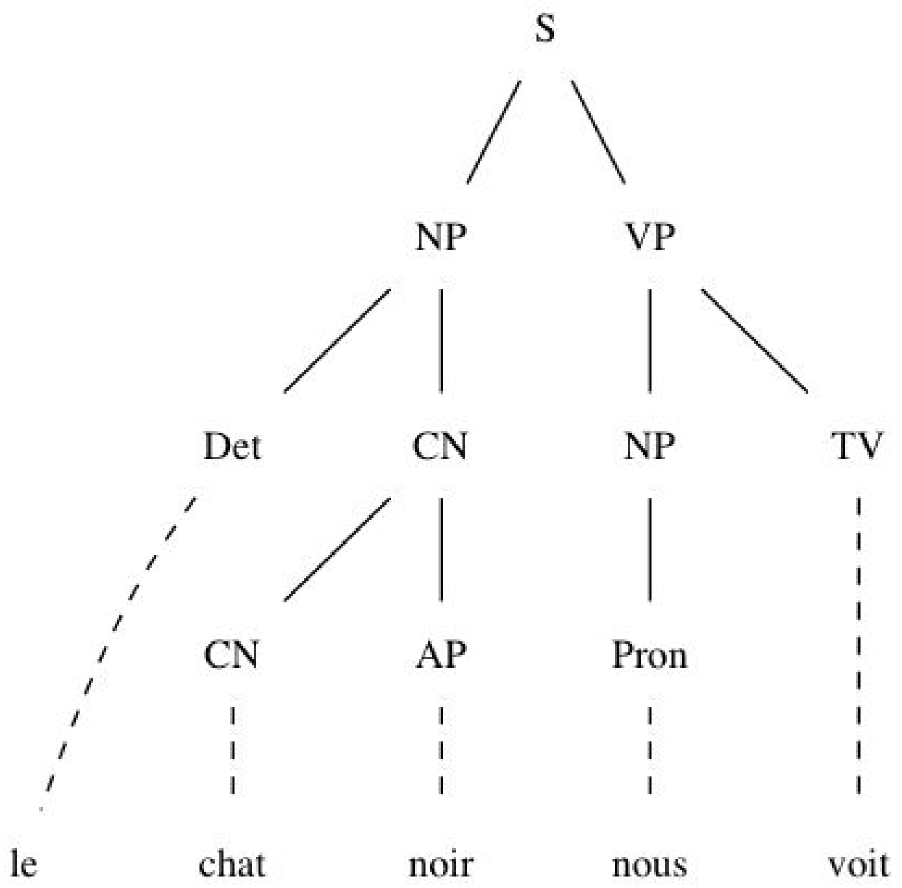

PredVP
DetCN
ComplTV
the_Det
AdjCN
see_TV
UsePron
black_AP
cat_CN
we_Pron
the
black
cat
sees
us


S
NP
VP
the_Det
CN
see_TV
NP
black_AP
cat_CN
we_Pron
the
black
cat
sees
us


S
NP
VP
Det
CN
TV
NP
AP
CN
Pron
the
black
cat
sees
us

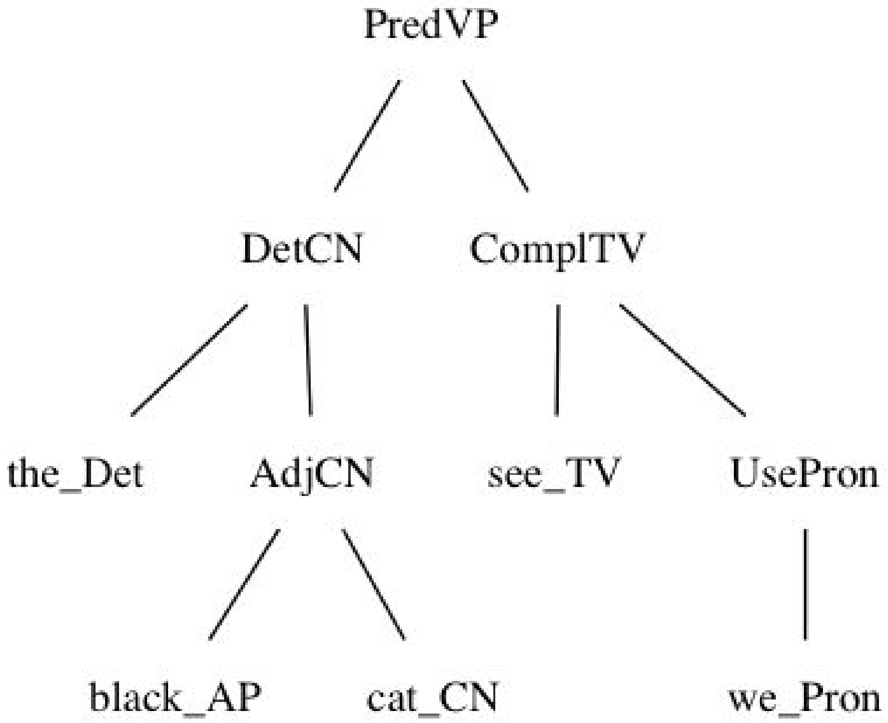

S
NP
VP
the_Det
CN
see_TV
NP
black_AP
cat_CN
we_Pron

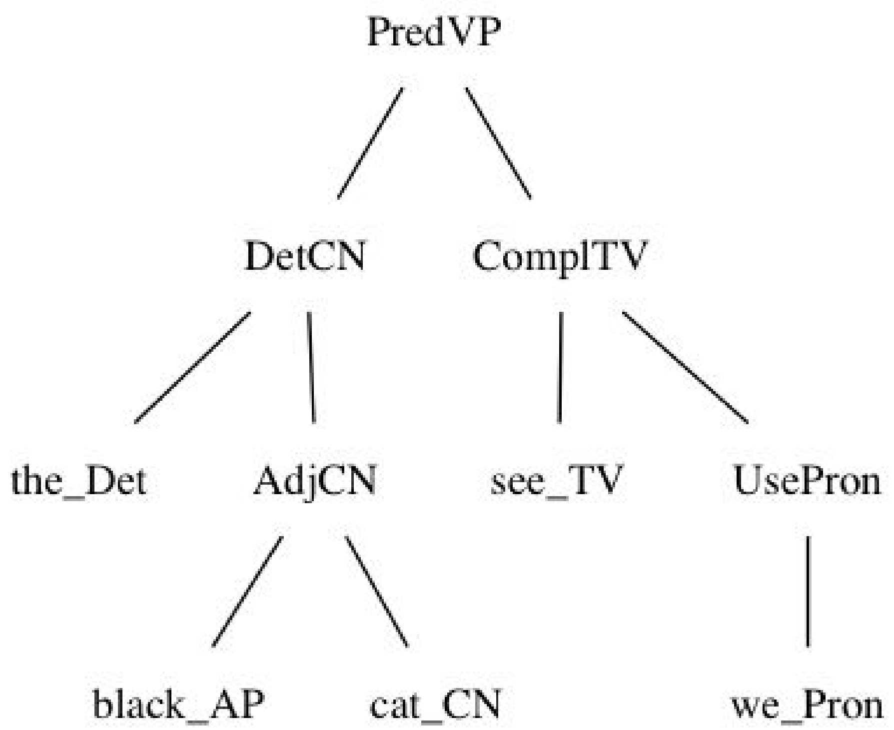

S
NP
VP
the_Det
CN
see_TV
NP
black_AP
cat_CN
we_Pron
le
chat
noir
nous
voit


S
NP
VP
Det
CN
NP
TV
CN
AP
Pron
le
chat
noir
nous
voit

From AST to dependency tree

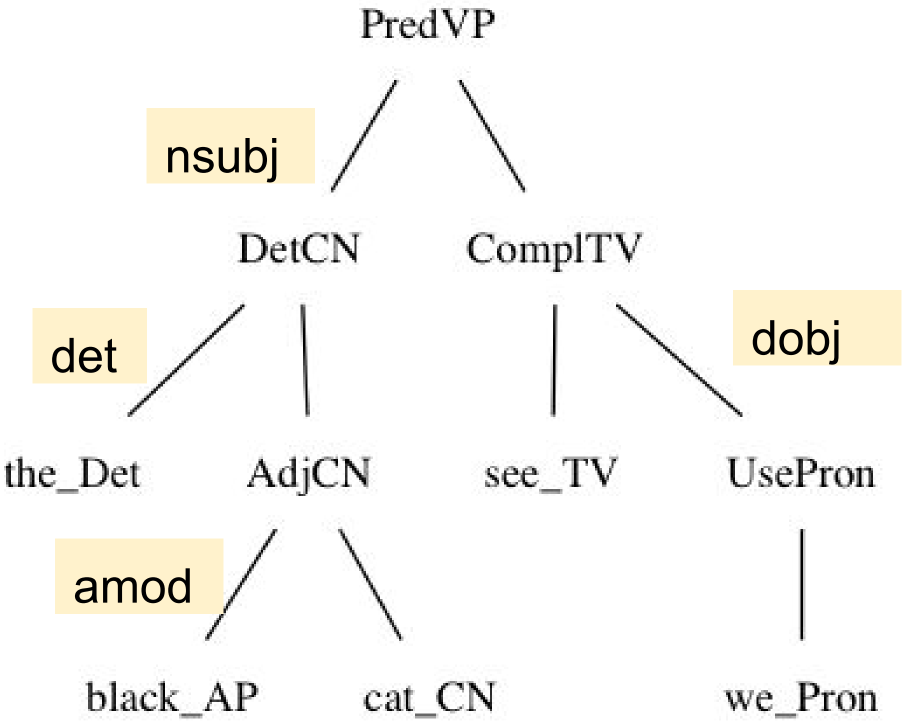
- put labels on subtrees and hide functions


PredVP
DetCN
ComplTV
the_Det
AdjCN
see_TV
UsePron
black_AP
cat_CN
we_Pron



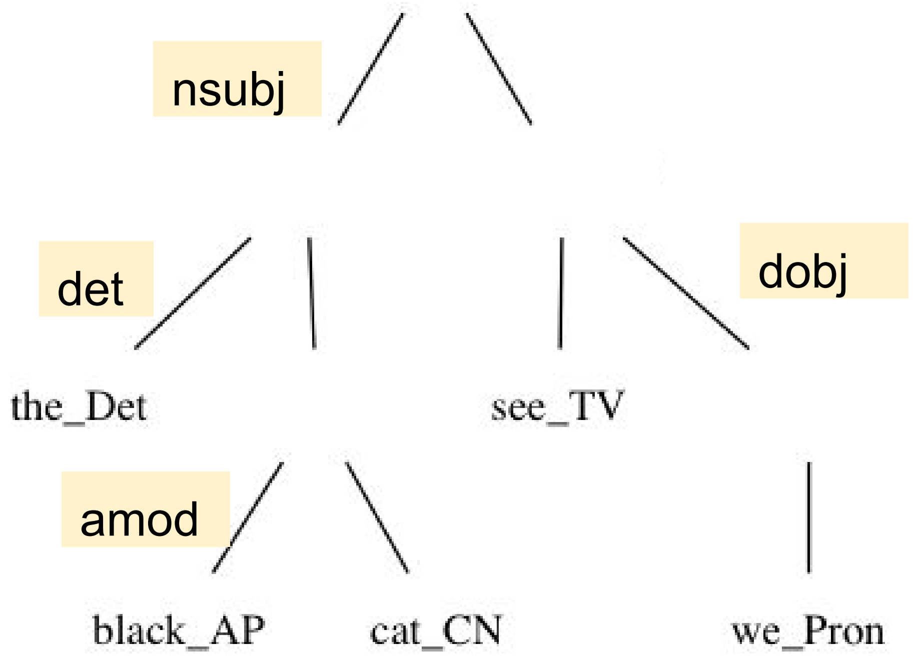
Dependency configuration

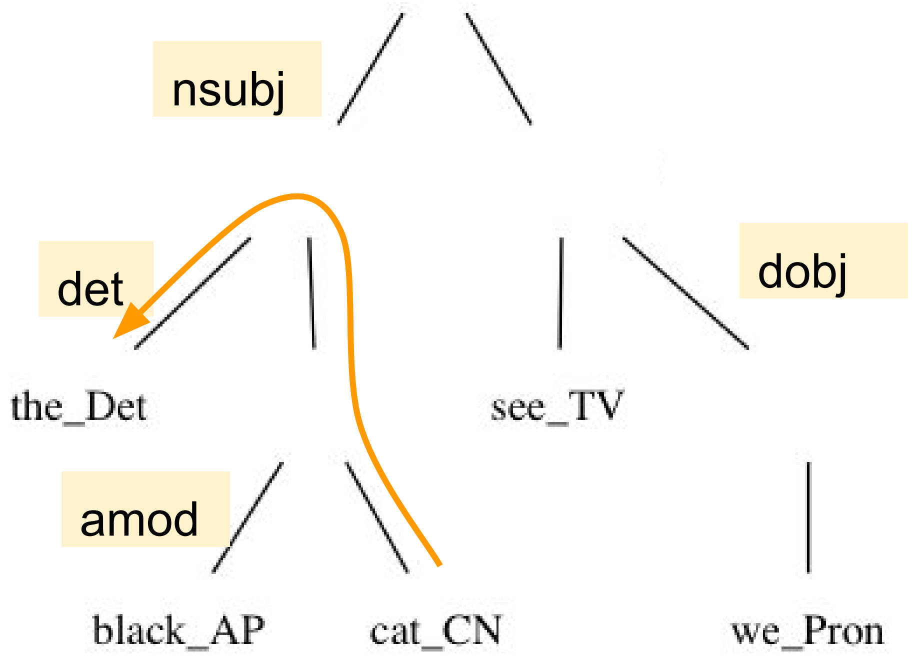
```
PredVP     nsubj head
ComplTV    head  dobj
DetCN      det   head
AdjCN      amod  head
```

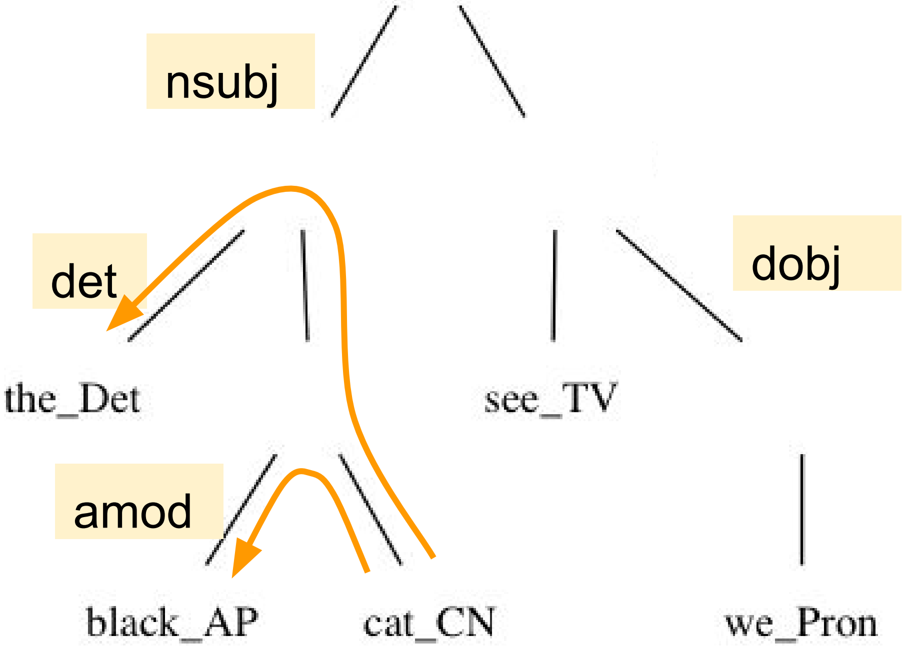


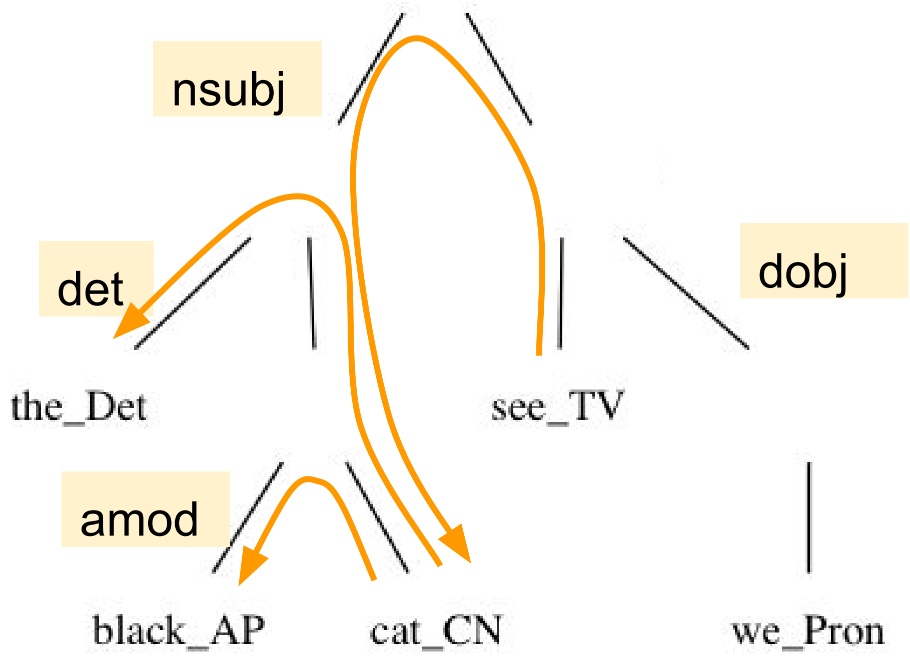
Dependency configuration

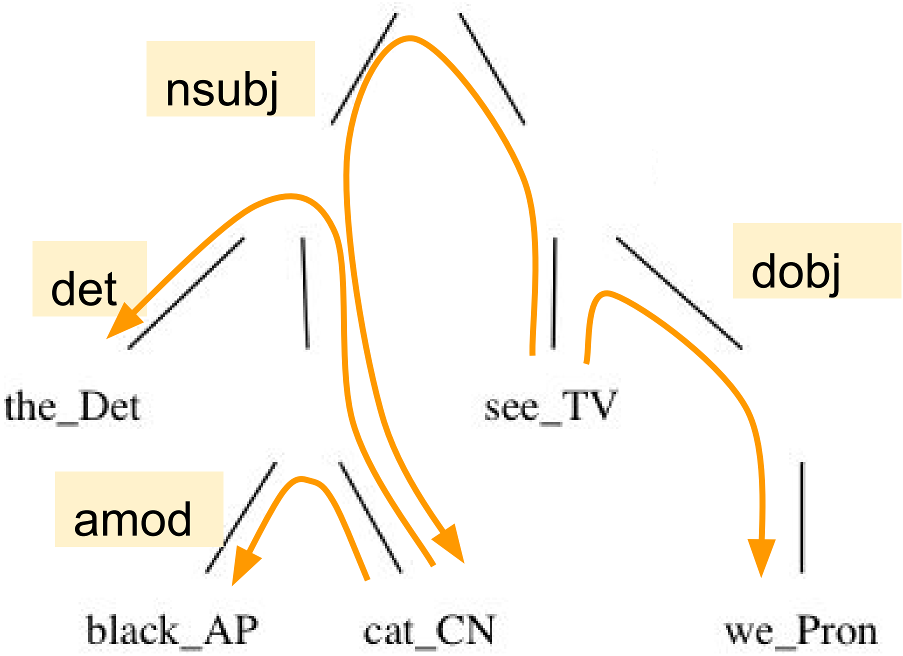
PredVP	nsubj	head
ComplTV	head	dobj
DetCN	det	head
AdjCN	amod	head

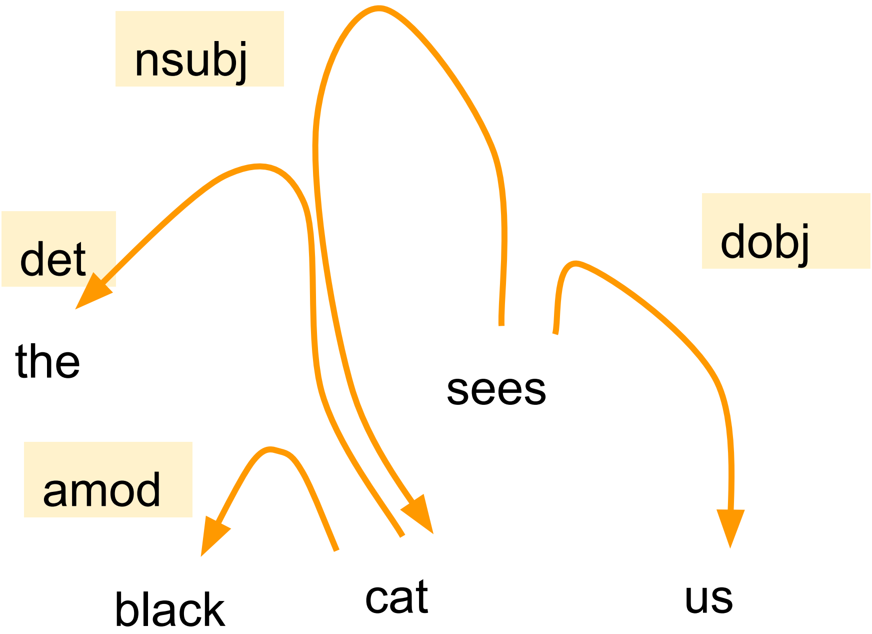

nsubj
det
dobj
the_Det
see_TV
amod
black_AP
cat_CN
we_Pron

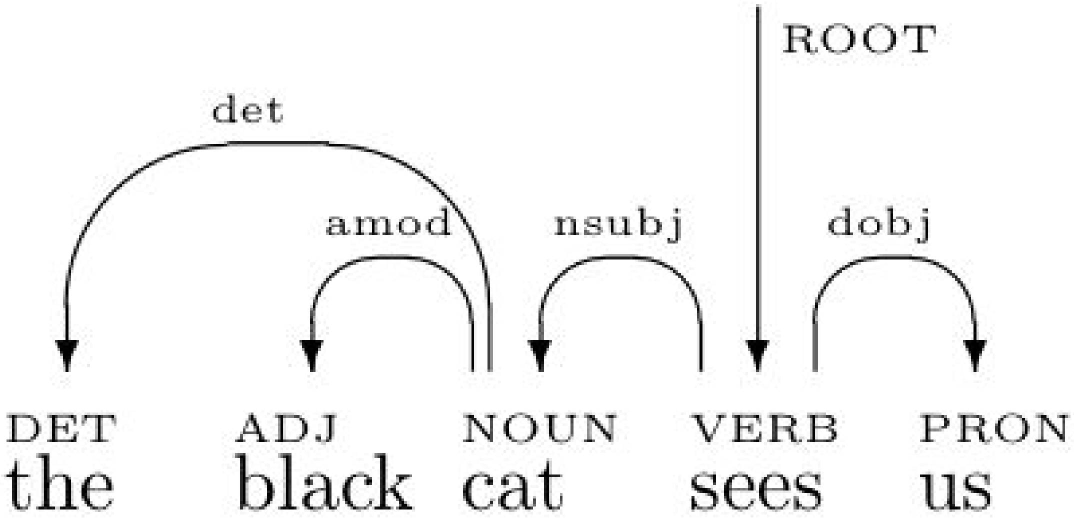

nsubj
det
dobj
the_Det
see_TV
amod
black_AP
cat_CN
we_Pron


nsubj
det
dobj
the_Det
see_TV
amod
black_AP
cat_CN
we_Pron


nsubj
det
dobj
the_Det
see_TV
amod
black_AP
cat_CN
we_Pron

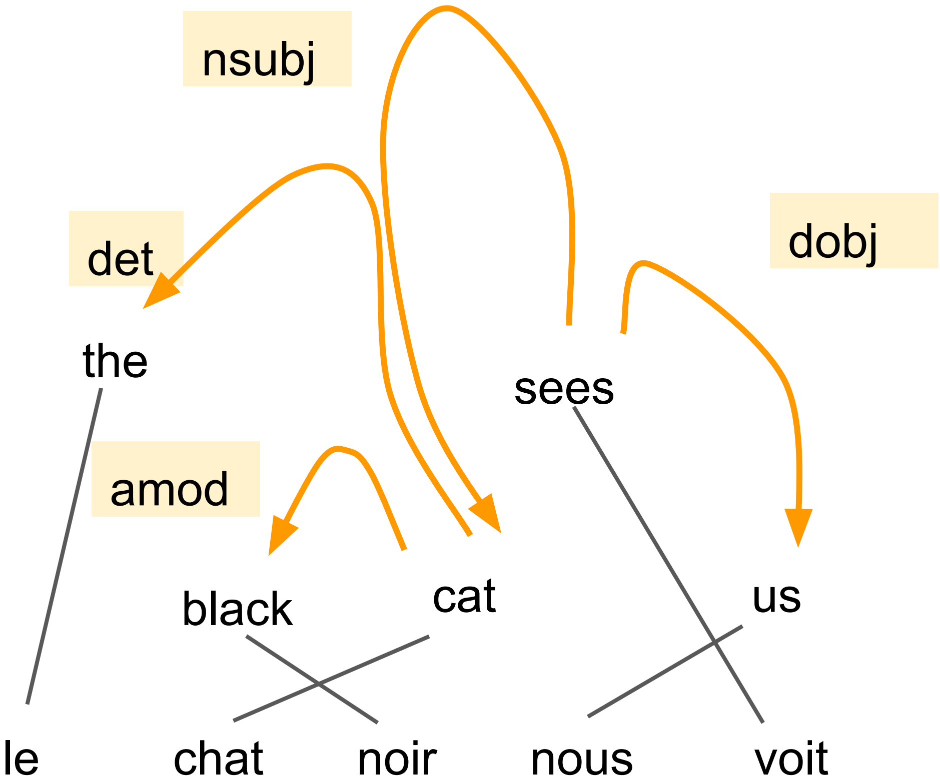

nsubj
det
dobj
amod
the_Det
see_TV
black_AP
cat_CN
we_Pron

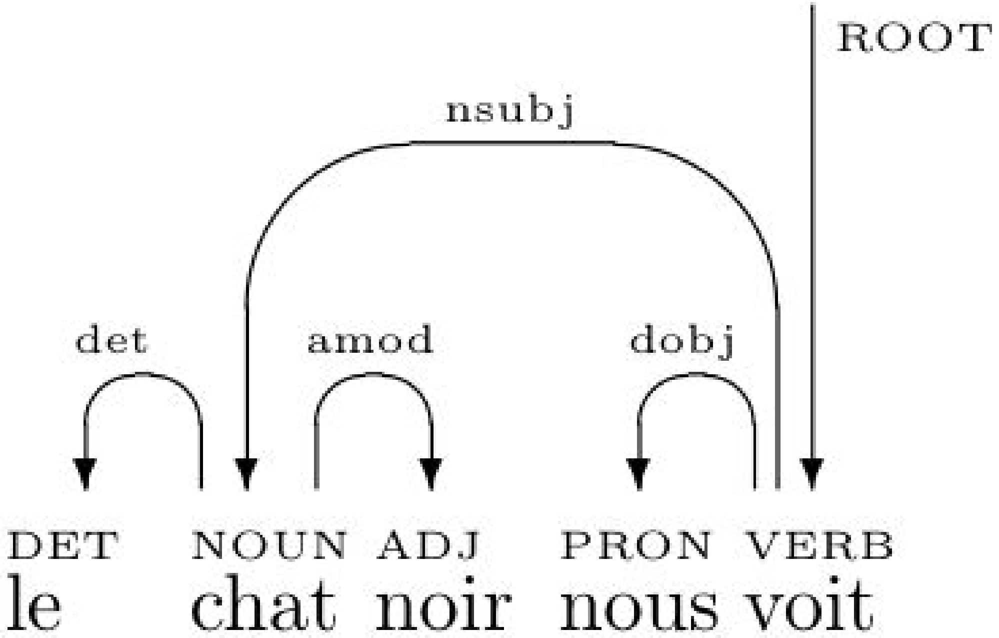

nsubj
det
the
sees
dobj
amod
black
cat
us



POS configuration

Det	DET
AP	ADJ
CN	NOUN
TV	VERB
Pron	PRON


nsubj
det
dobj
the
sees
amod
black
cat
us
le
chat
noir
nous
voit


ROOT
nsubj
det
amod
dobj
DET NOUN ADJ PRON VERB
le chat noir nous voit

Trees: summary

abstract syntax tree: language-independent

- built from functions
- lossless representation: functions determine categories and dependencies

Trees: summary

abstract syntax tree: language-independent

- built from functions
- lossless representation: functions determine categories and dependencies

parse tree: language-dependent

- built from categories and words
- lossy: does not determine dependencies (let alone functions)

Trees: summary

abstract syntax tree: language-independent

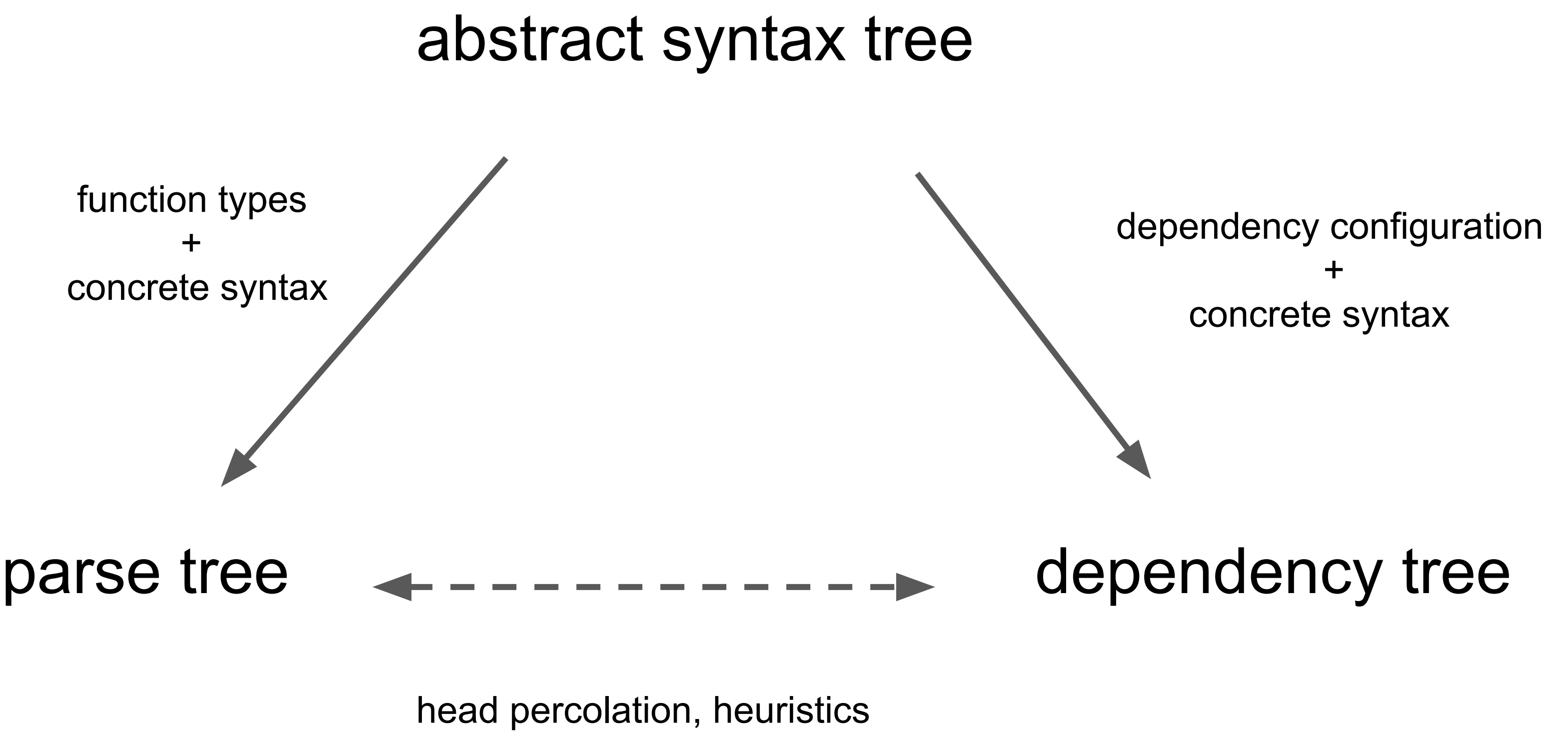
- built from functions
- lossless representation: functions determine categories and dependencies

parse tree: language-dependent

- built from categories and words
- lossy: does not determine dependencies (let alone functions)

dependency tree: language-dependent

- built from dependencies and words
- lossy: does not determine categories (let alone functions)


abstract syntax tree
function types
+
concrete syntax
dependency configuration
+
concrete syntax
parse tree
dependency tree
head percolation, heuristics

Syncategorematic words

- pinpointing a difference in the ways of thinking:
  - dependency grammar is about words,
  - GF is about meanings

categorematic: word with its own category and function

```
fun cat_CN : CN
lin cat_CN = “cat”
```

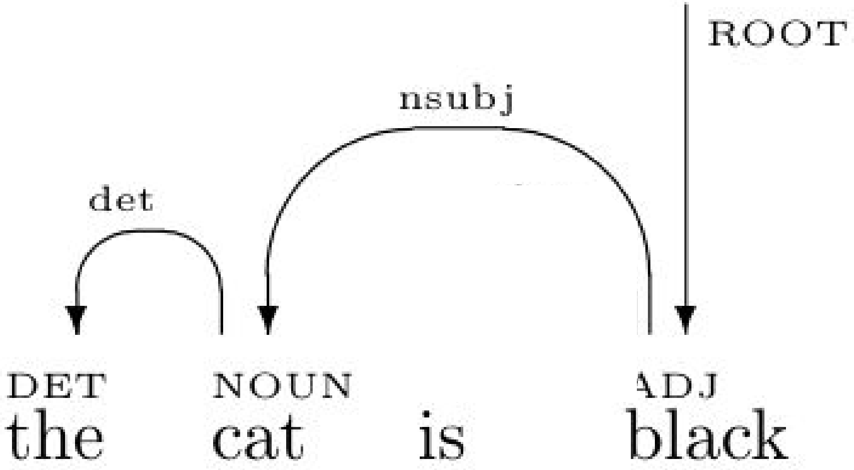
categorematic: word with its own category and function

```
fun cat_CN : CN
lin cat_CN = “cat”
```

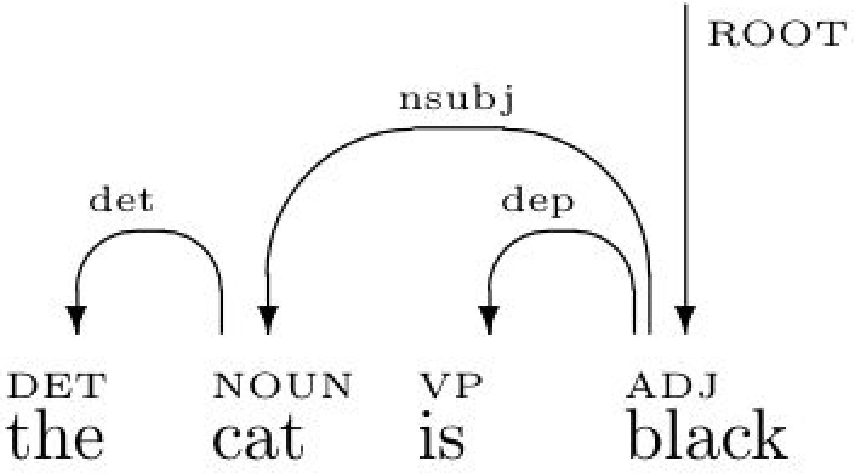
syncategorematic: word that is “between categories”

```
fun ComplAP : AP -> VP
lin ComplAP ap = “is” ++ AP
```

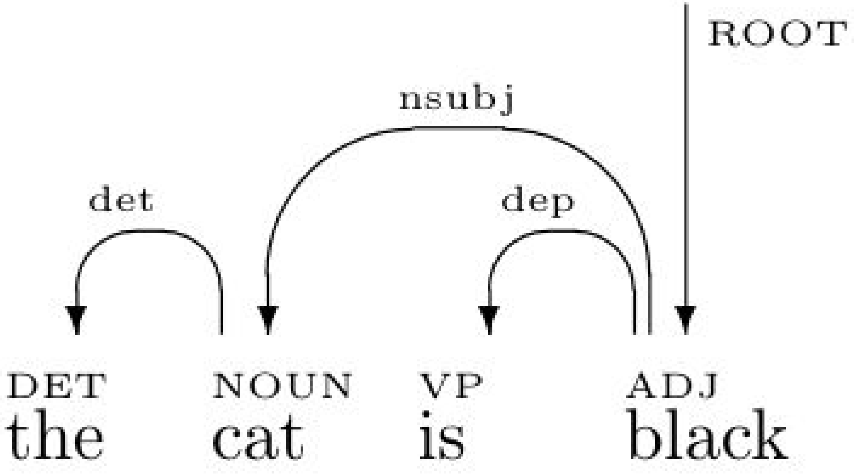
No semantics (`fun`) of its own. Not an argument. No label.


ROOT
nsubj
det
DET
NOUN
ADJ
the
cat
is
black

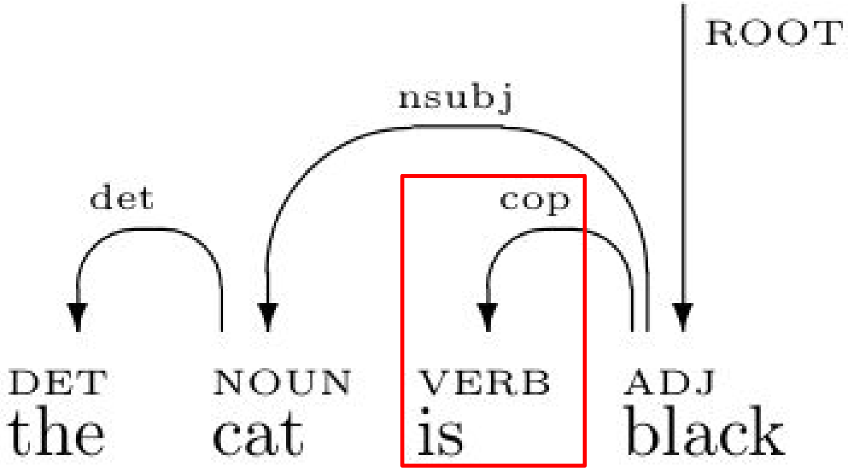
adding default labels



we get



UD wants



Solution 1: rewrite the grammar

```
cat Cop                                    -- VERB
fun ComplAP : Cop -> AP -> VP              -- cop  head
fun be_Cop : Cop

lincat Cop = Str
lin ComplAP cop ap = cop ++ ap
lin be_Cop = “is”
```

Solution 2: extend the dependency configuration

abstract → abstract | concrete

local → local | nonlocal

- more complicated, not universal
+ less work than rewriting the grammar anyway
+ more flexible (if UD should change…)
+ abstract syntax can be kept more abstract

Other syncategorematic words

- negation words
- tense auxiliaries
- infinitive marks
- (sometimes) prepositions

Typically words eliminated in **collapsing** or **flattening** gf2ud

The GF Resource Grammar Library

- 32 dependency treebanks for free

86 categories

131 syntactic combination functions

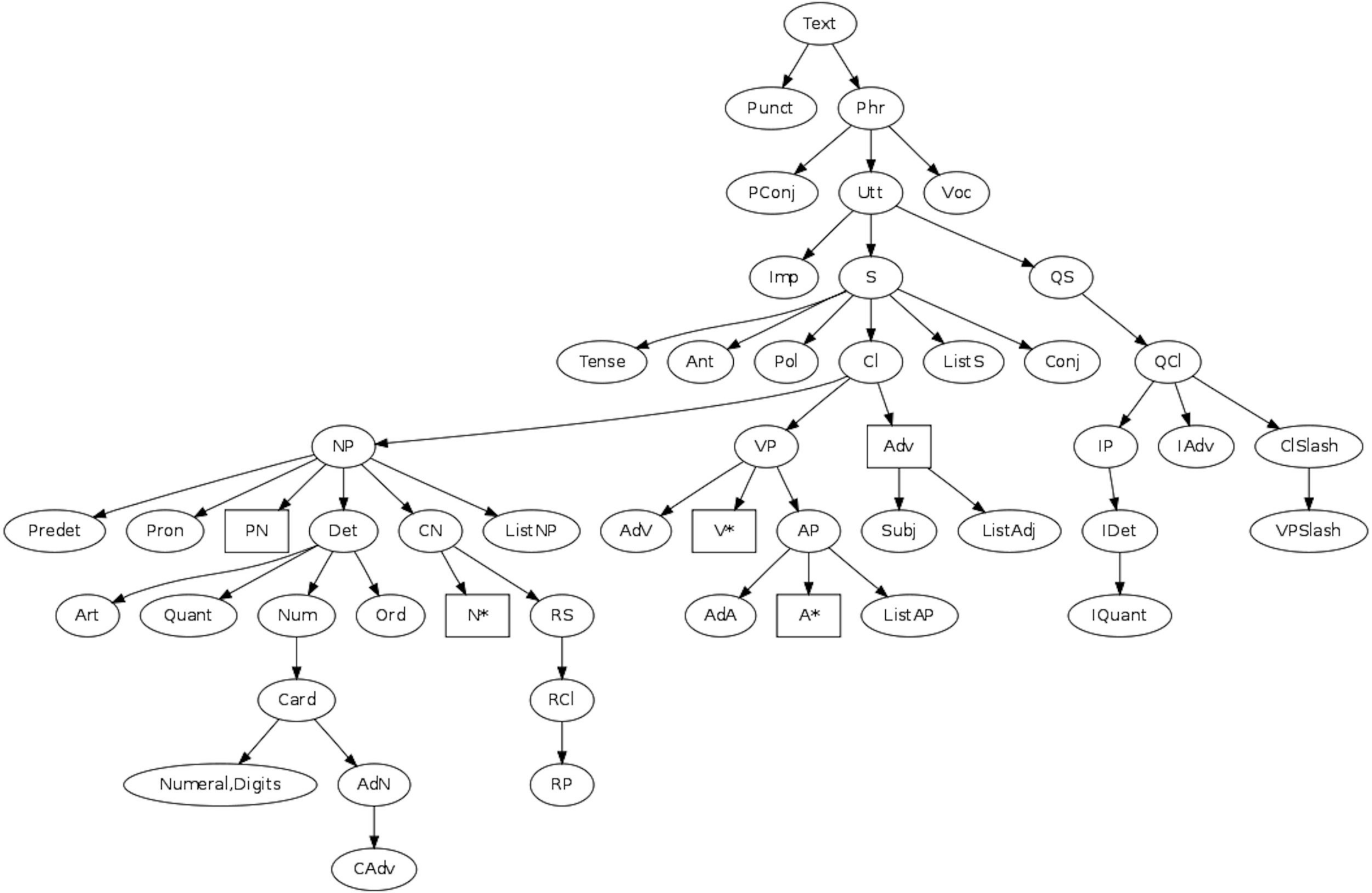
500 lexical items (32 languages)

20,000-70,000 lexical items (16 languages)

15-20 person years of work by 50+ persons

open source (LGPL/BSD)

Norwegian Danish Afrikaans
English Swedish German Dutch
French Italian Spanish Catalan
Russian Bulgarian Finnish Estonian
Japanese Thai Chinese Hindi
Romanian
Polish
Icelandic
Latvian Mongolian Urdu Punjabi Sindhi
Greek Maltese Nepali Persian
Latin Turkish Hebrew Arabic Amharic Swahili


Text
Punct
Phr
PConj
Utt
Voc
Imp
S
QS
Tense
Ant
Pol
Cl
ListS
Conj
QCl
NP
VP
Adv
IP
IAdv
ClSlash
Predet
Pron
PN
Det
CN
ListNP
AdV
V*
AP
Subj
ListAdj
IDet
VPSlash
Art
Quant
Num
Ord
N*
RS
AdA
A*
ListAP
IQuant
Card
RCl
Numeral,Digits
AdN
RP
CAdv

Universal Dependencies (2016)

A community-driven effort to annotate multilingual texts

Cross-lingual consistency in annotations across languages

17 Part-of-Speech tags ; 40 dependency labels ; lexical features

Annotated corpora released every 6 months;

recent release has 33 languages

Content words as head => to maximize cross-lingual sharedness

Clausal Predicates

nsubj csubj
dobj iobj
ccomp xcomp

Coordination

conj cc punct

Passive voice

nsubjpass csubjpass
auxpass

Adverbials

advmod nmod
advcl

Copulas and special marker

mark cop

Auxiliary verbs and negation

aux neg

Noun dependents

det nummod
amod appos
neg nmod
acl

case

Compounding

compound mwe
name

Other

root dep

Unknowns

list dislocated parataxis remnant reparandum

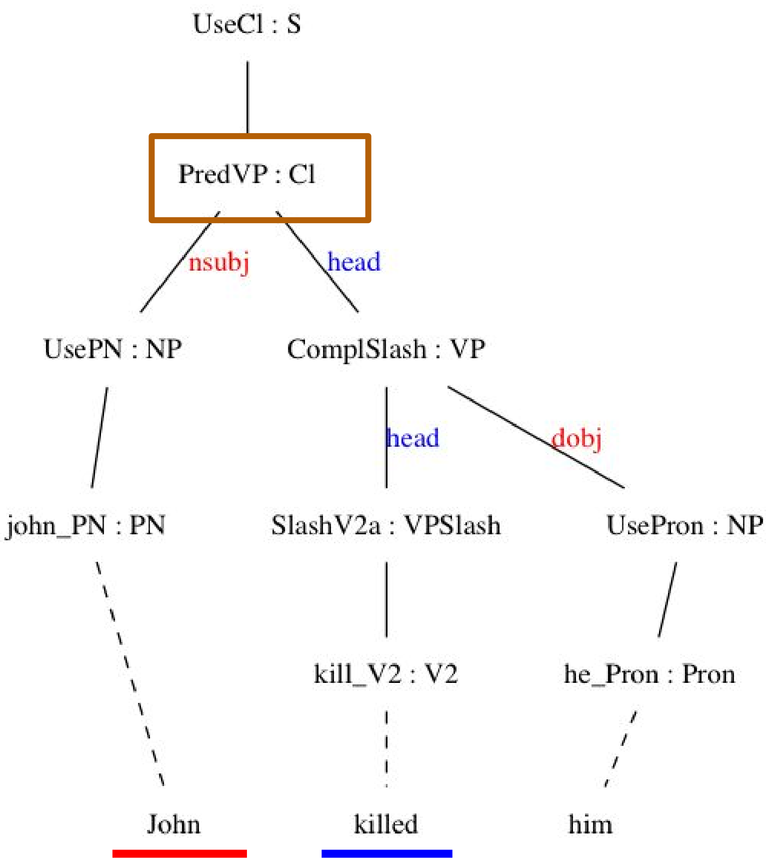
The full UD mappings for English

	Local	Non-local
Abstract	132	11
Concrete	17	29

Local abstract configurations

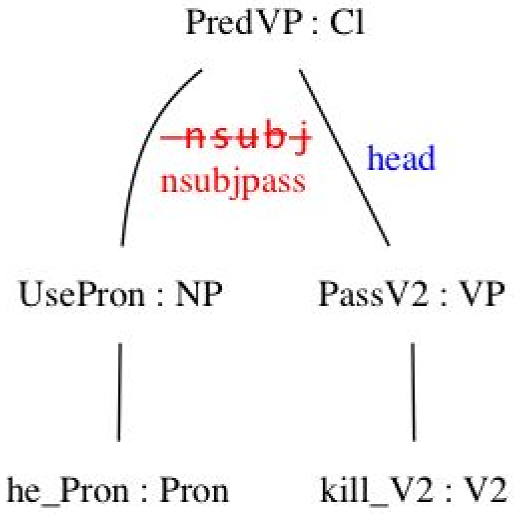
PredVP	NP->VP->Cl	nsubj head

nominal subjects



Non-local abstract configurations

(PredVP ? (PassV2 ?))	nsubjpass	head
PredVP	nsubj	head



Local Concrete configurations

English

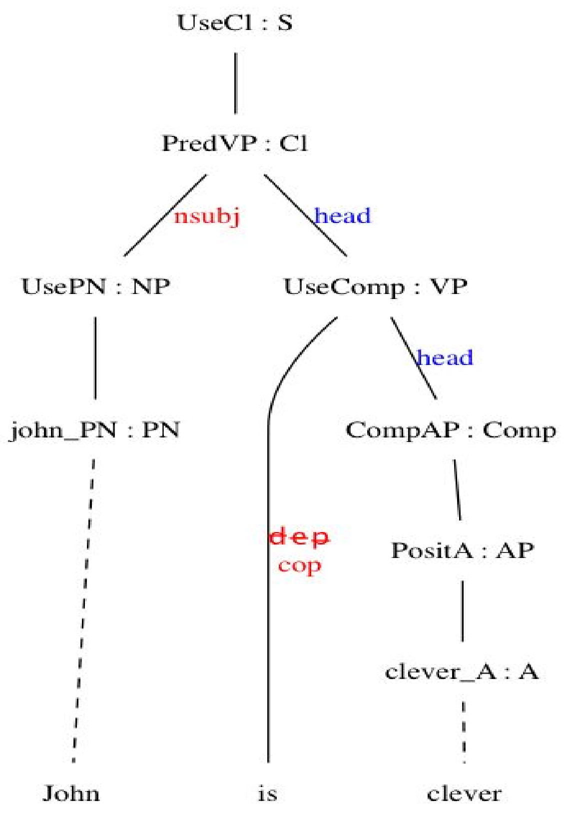
UseComp head

{“is”, “was”, “be”, “are”} cop head

Swedish

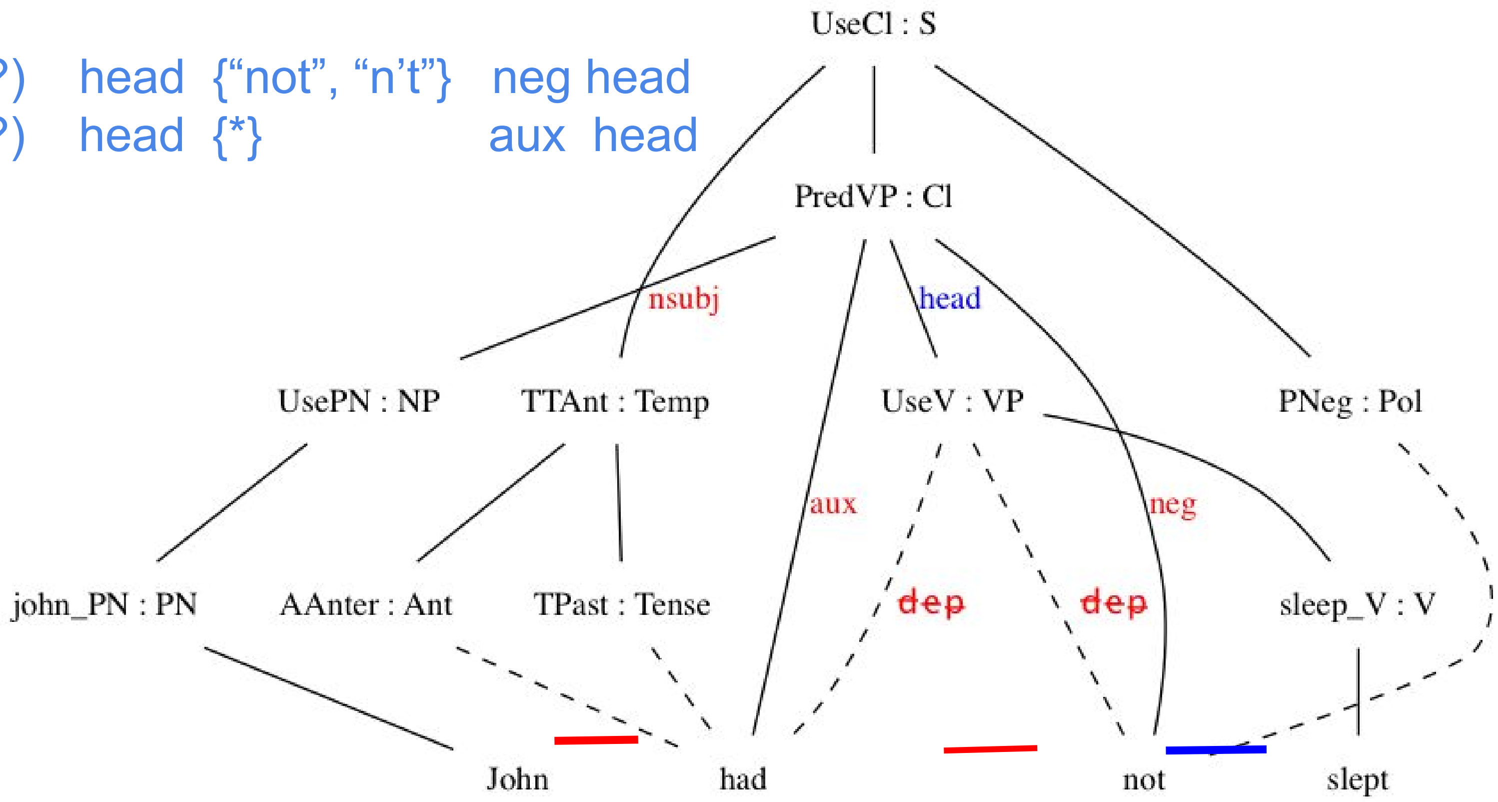
UseComp head

{“är”, “var”, “vara”, “varit”} cop head



Non-local concrete configurations

(UseCl ? PNeg ?)   head {“not”, “n’t”}   neg head
(UseCl ?   ?   ?)   head {*}   aux head



Experiments

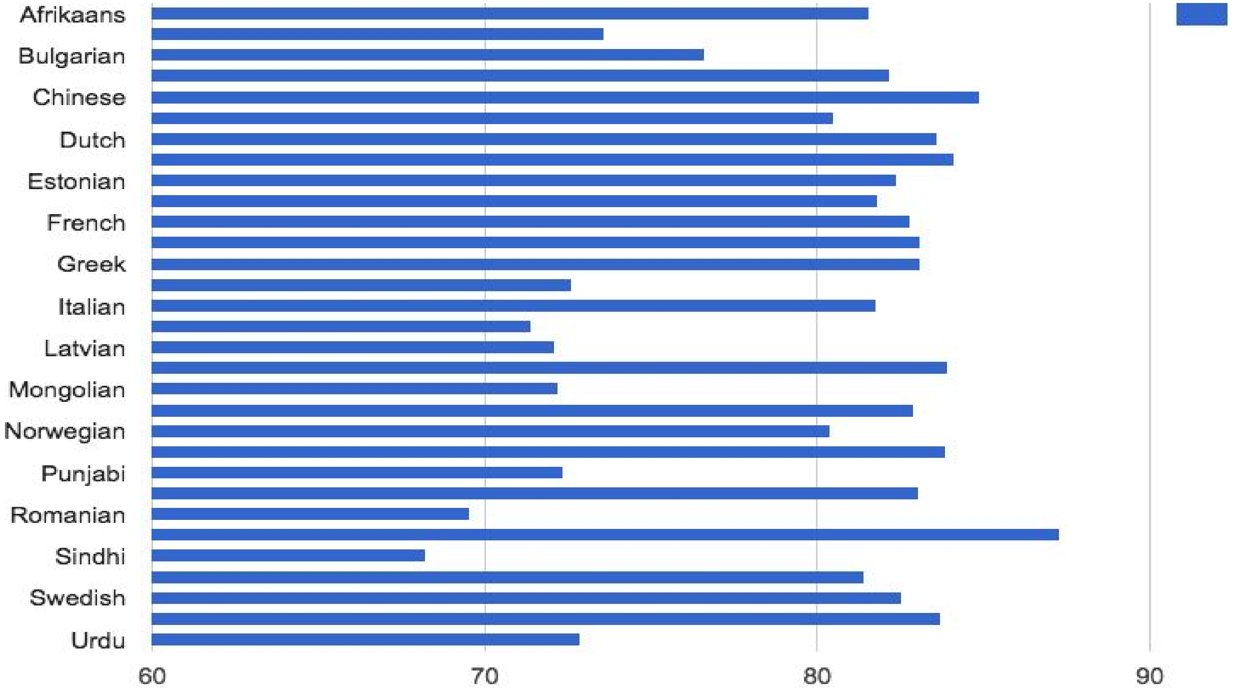
Cross-lingual bootstrapping

Given a treebank in GF, using the mappings defined on abstract syntax it is possible to bootstrap dependency tree for new languages

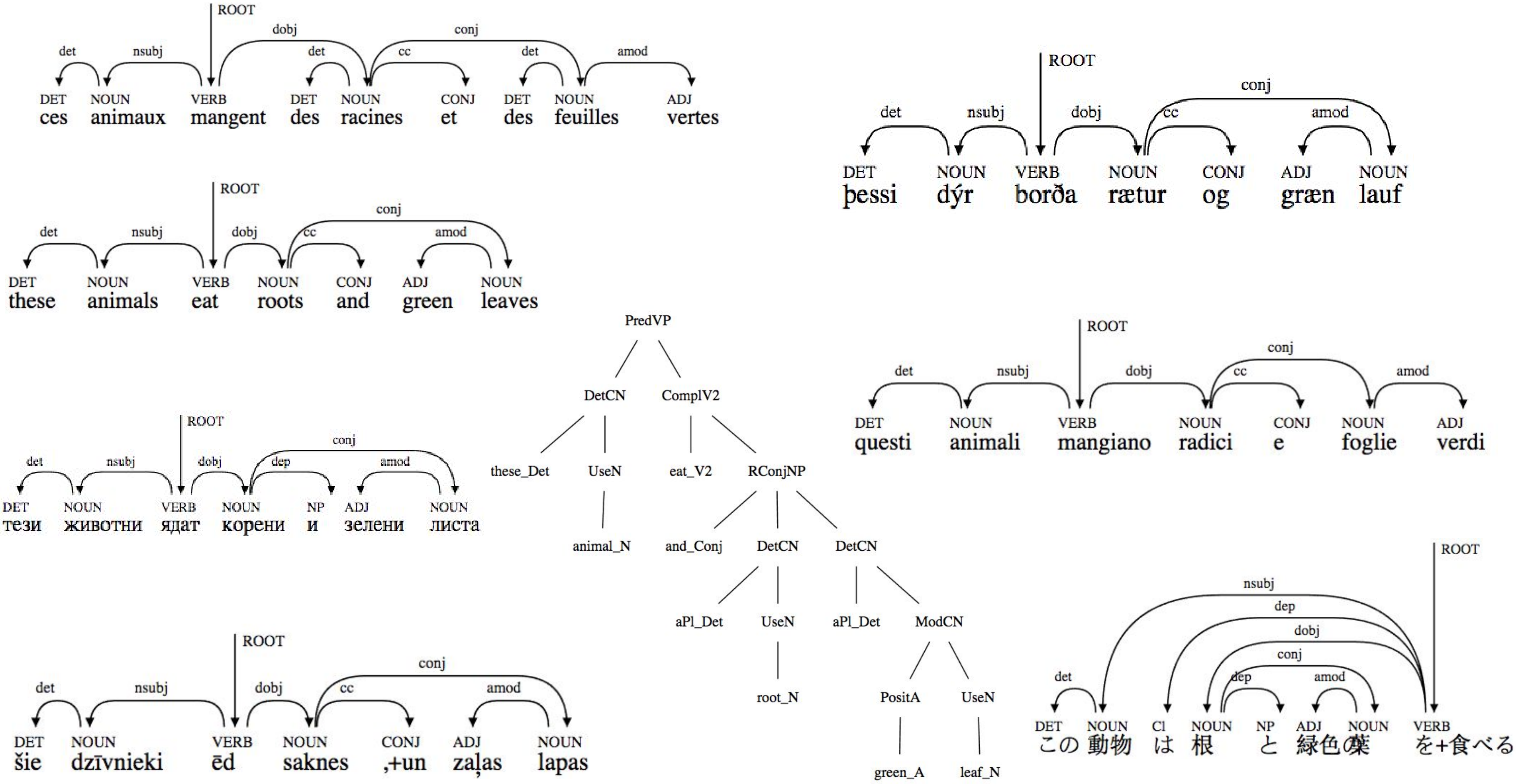
Experiments with bootstrapping to all 36 languages in the RGL

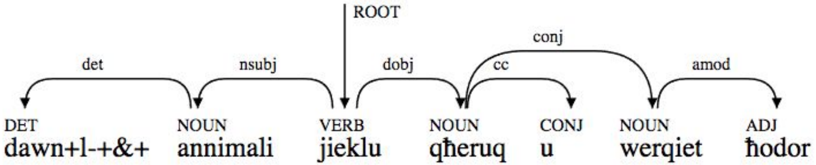
- a GF treebank from the examples presented in UD annotation manuals
- about 80% of edges in the treebank can be labelled using the abstract rules

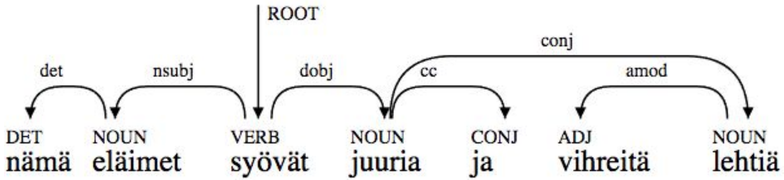
Worst case: 70-75% (Japanese, Latvian, Punjabi)

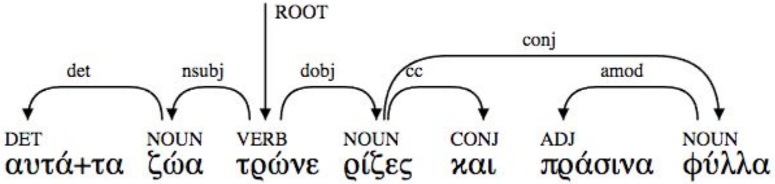


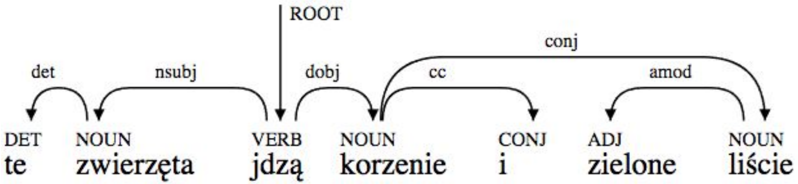
Afrikaans
Bulgarian
Chinese
Dutch
Estonian
French
Greek
Italian
Latvian
Mongolian
Norwegian
Punjabi
Romanian
Sindhi
Swedish
Urdu
60
70
80
90

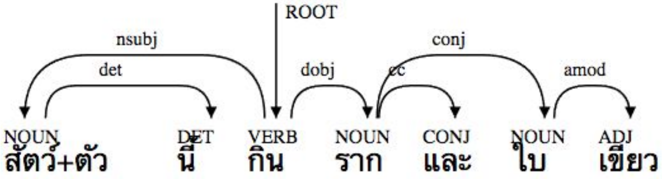

ROOT
det nsubj dobj det cc conj det amod
DET NOUN VERB DET NOUN CONJ DET NOUN ADJ
ces animaux mangent des racines et des feuilles vertes
ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
these animals eat roots and green leaves
ROOT
det nsubj dobj dep conj amod
DET NOUN VERB NOUN NP ADJ NOUN
тези животни ядат корени и зелени листа
ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
šie dzīvnieki ēd saknes ,+un zaļas lapas
PredVP
DetCN
ComplV2
these_Det
UseN
eat_V2
RConjNP
animal_N
and_Conj
DetCN
DetCN
aPl_Det
UseN
aPl_Det
ModCN
root_N
PositA
UseN
green_A
leaf_N
ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
þessi dýr borða rætur og græn lauf
ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ NOUN ADJ
questi animali mangiano radici e foglie verdi
ROOT
nsubj dep dobj conj det dep amod
DET NOUN Cl NOUN NP ADJ NOUN VERB
この 動物 は 根 と 緑色 葉 を+食べる

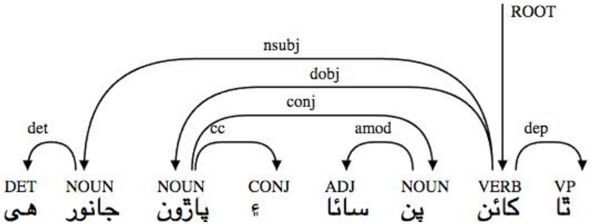

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ NOUN ADJ
dawn+l-+&+ annimali jieklu qħeruq u werqiet ħodor

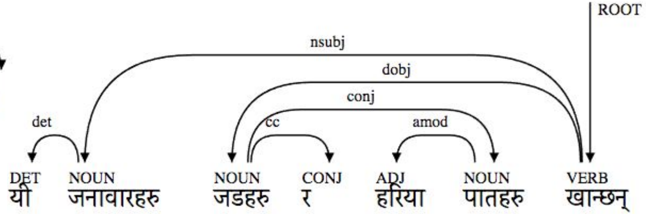

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
nämä eläimet syövät juuria ja vihreitä lehtiä


ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
αυτά+τα ζώα τρώνε ρίζες και πράσινα φύλλα


ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
te zwierzęta jdzą korzenie i zielone liście


ROOT
nsubj det dobj cc conj amod
NOUN DET VERB NOUN CONJ NOUN ADJ
สัตว์+ตัว นี้ กิน ราก และ ใบ เขียว


ROOT
nsubj dobj conj det cc amod dep
DET NOUN NOUN CONJ ADJ NOUN VERB VP
هی جانور پاڙون ۽ سائا پن کائن ٿا


ROOT
nsubj dobj conj det cc amod
DET NOUN NOUN CONJ ADJ NOUN VERB
यी जनावारहरु जडहरु र हरिया पातहरु खान्छन्

Converting the GF Penn treebank

GF converted version of the PTB

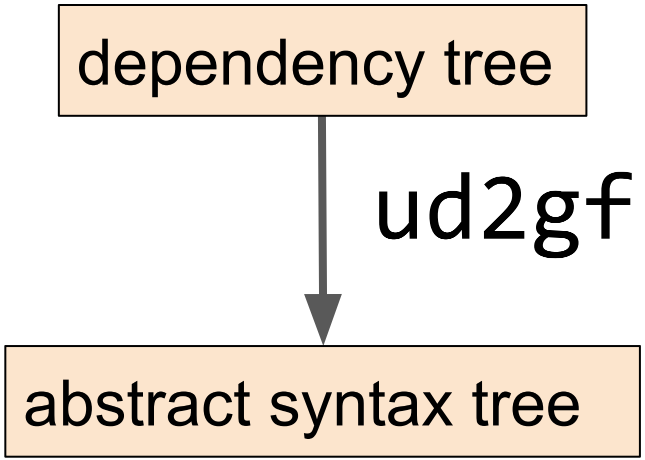
About 5% of the nodes in this treebank are missing/incomplete

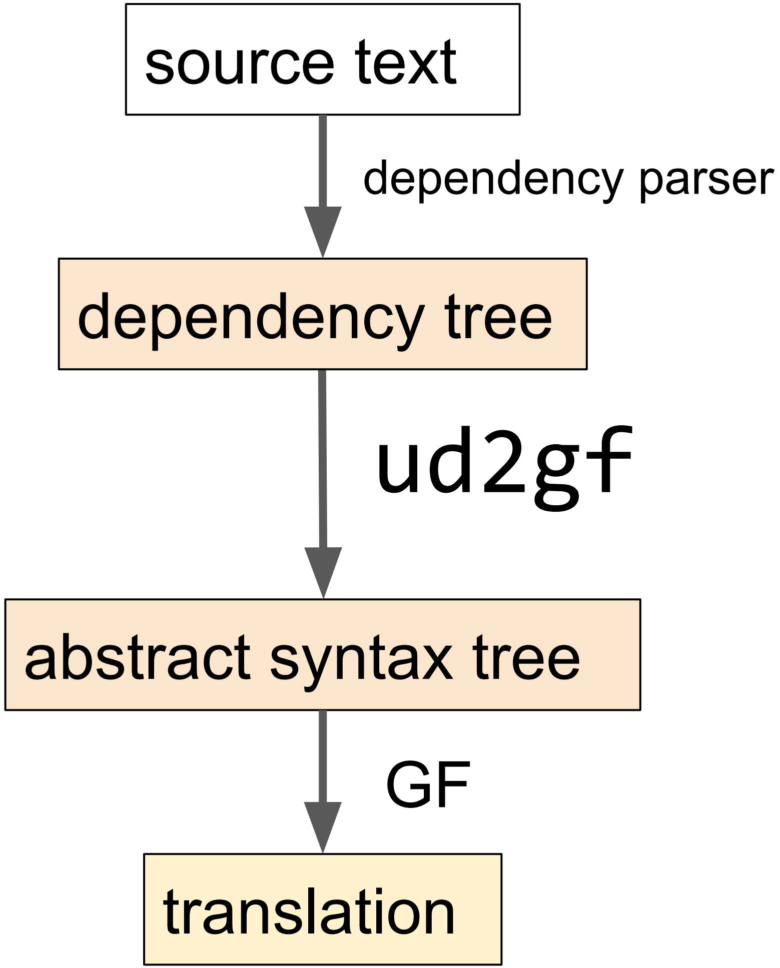
- almost 10% of nodes are assigned the default `dep’ label

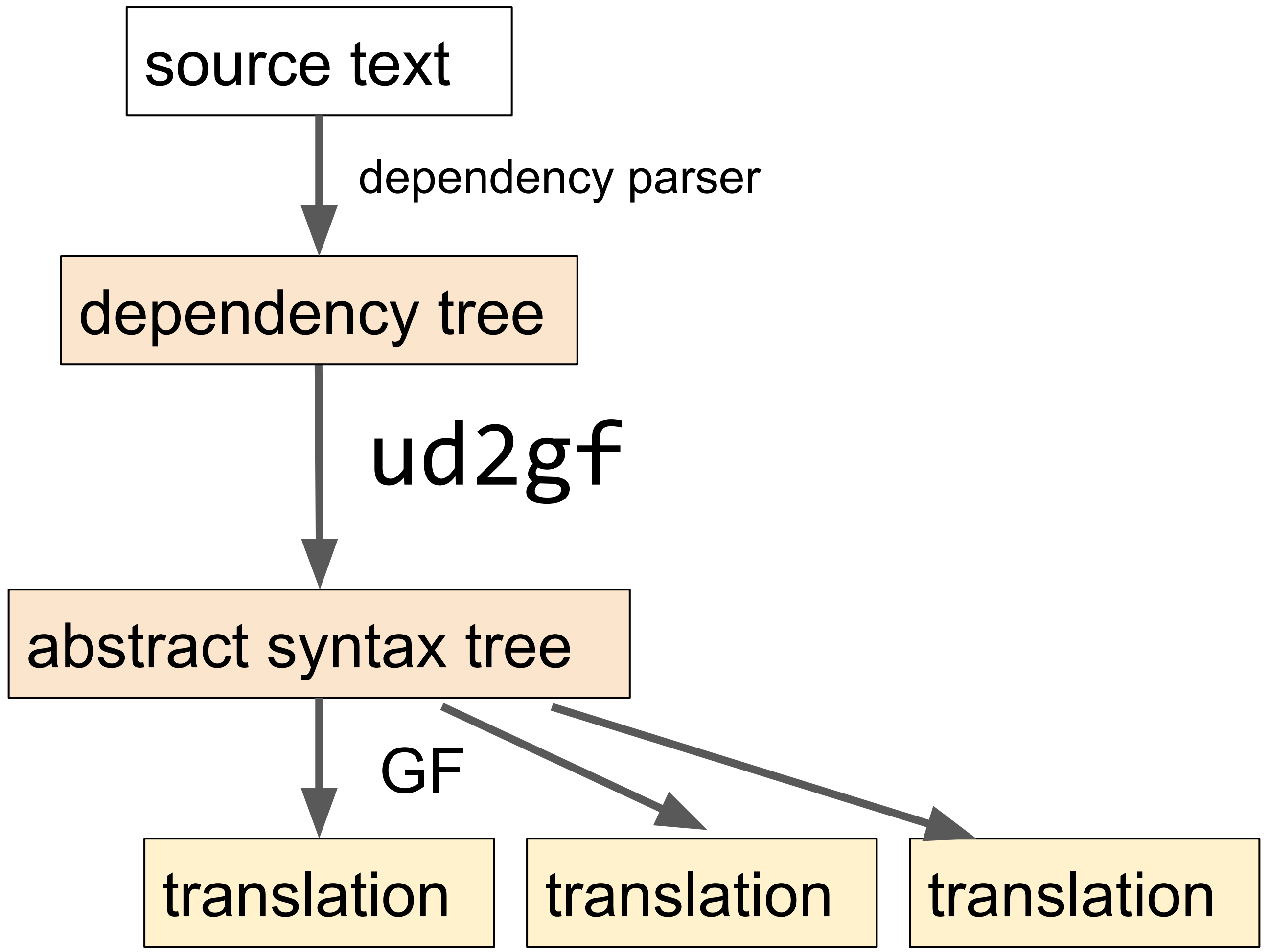
UD version of the PTB

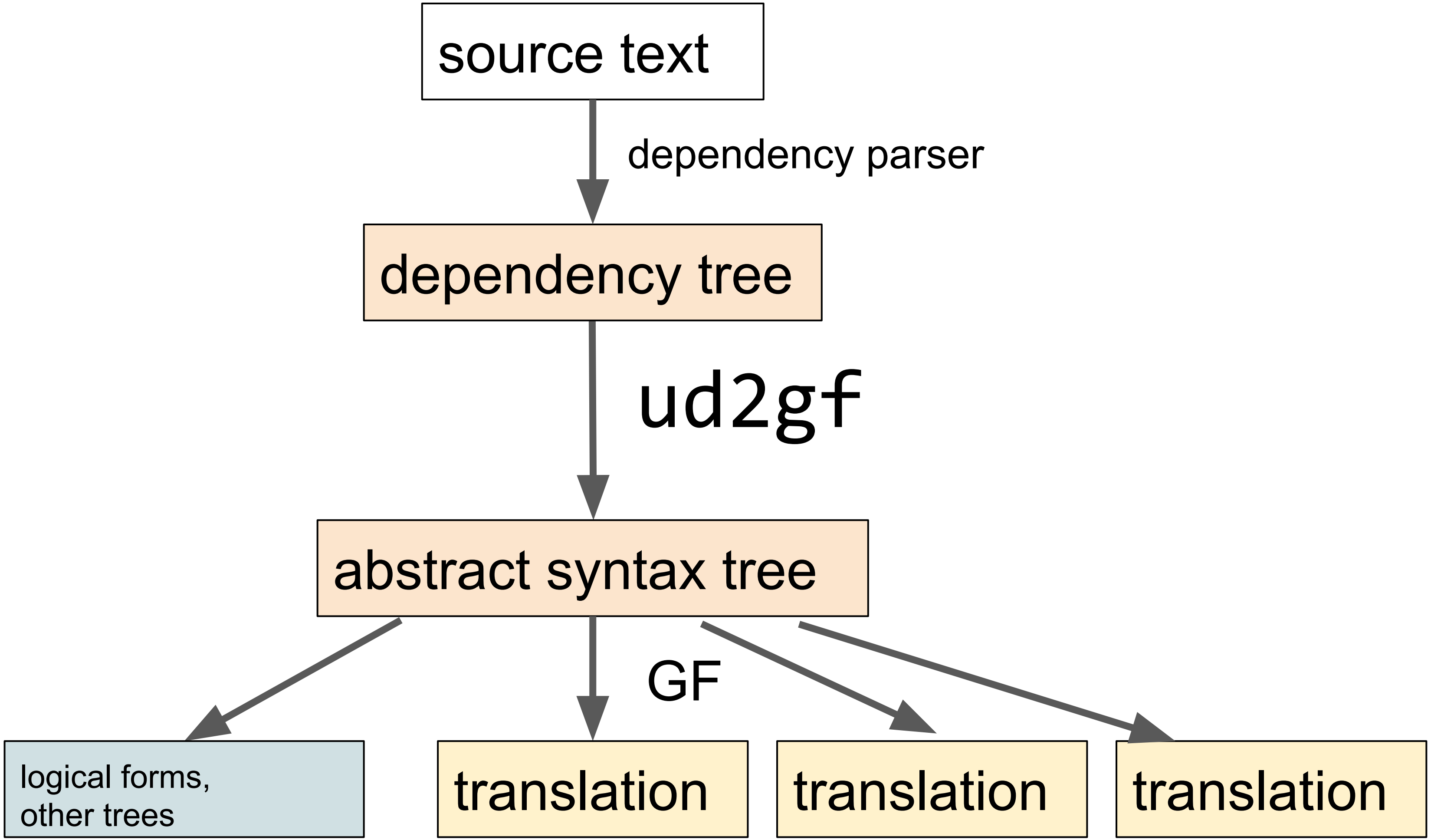
- labelled accuracies (LAS) of 79.32% on WSJ-22
- requires handling modal verbs and compounds

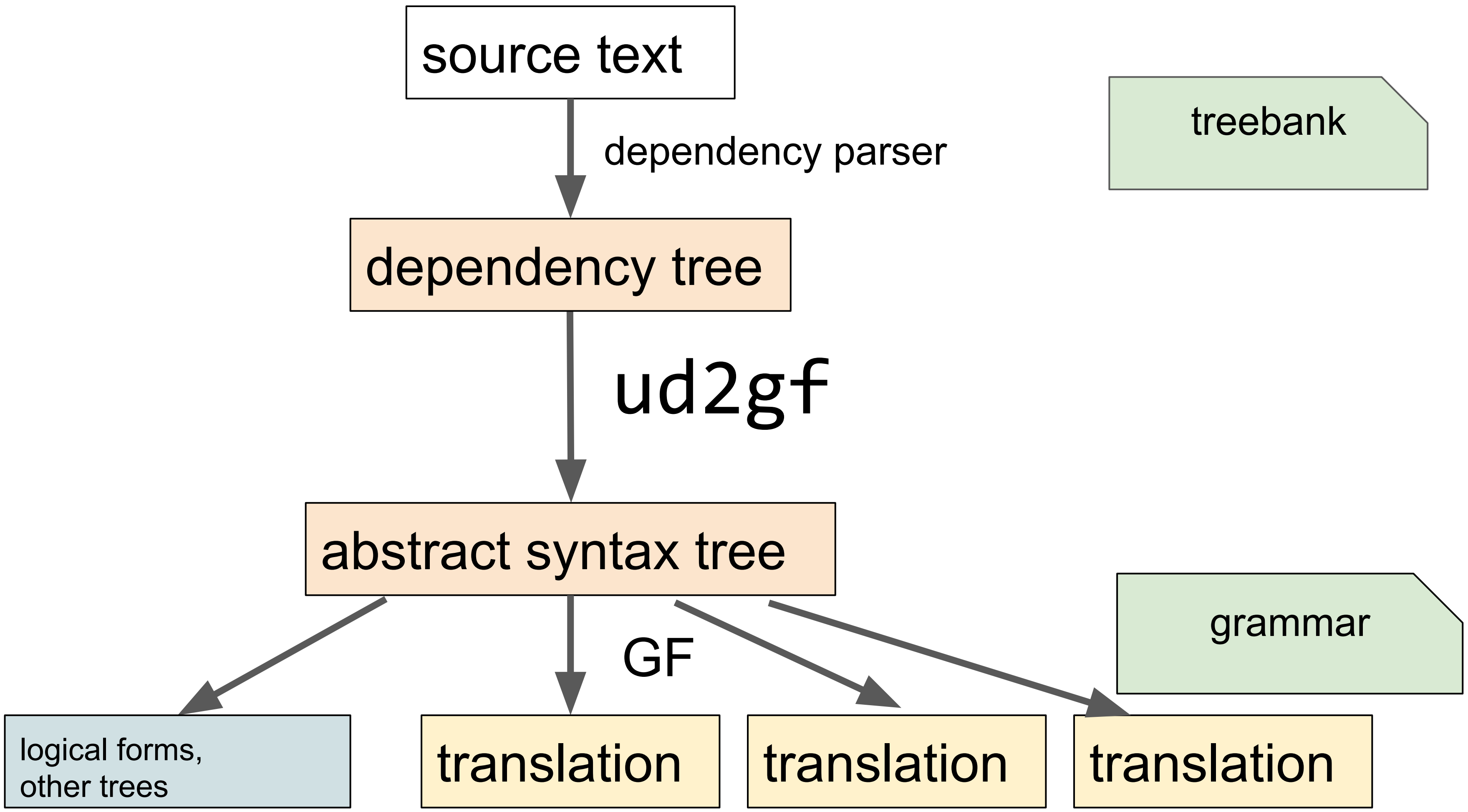
ud2gf

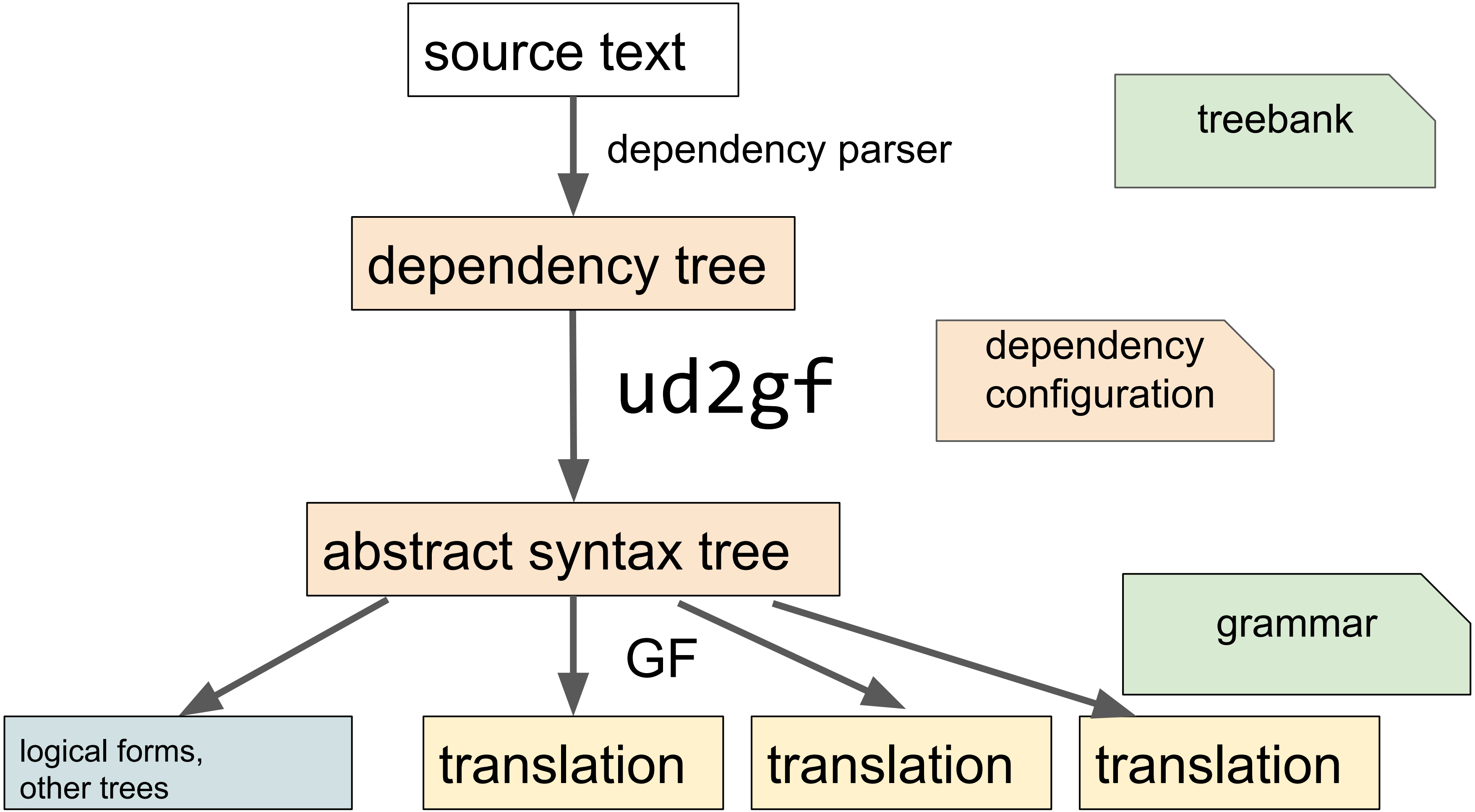

dependency tree
ud2gf
abstract syntax tree


source text
dependency parser
dependency tree
ud2gf
abstract syntax tree
GF
translation


source text
dependency parser
dependency tree
ud2gf
abstract syntax tree
GF
translation
translation
translation

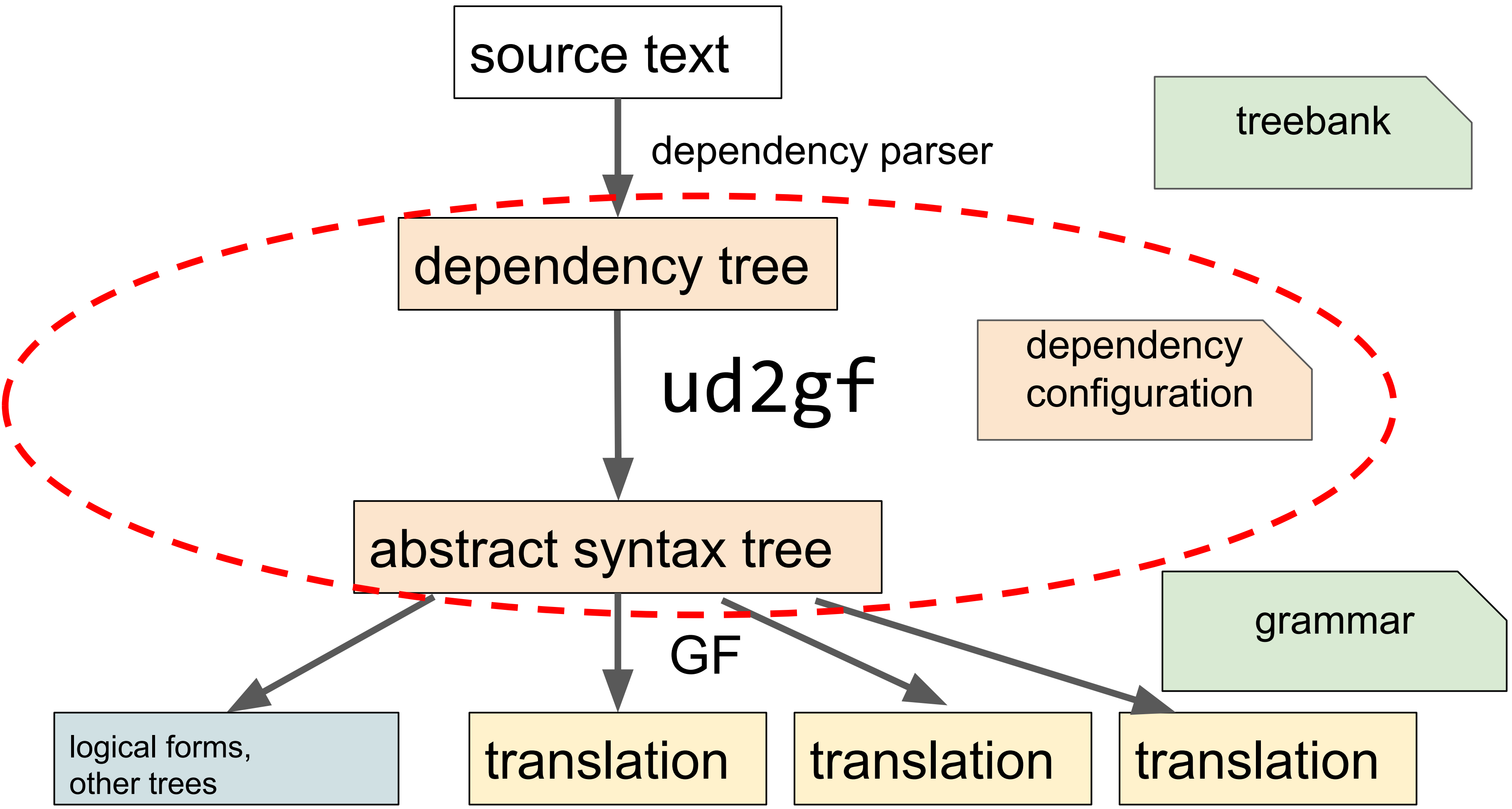

source text
dependency parser
dependency tree
ud2gf
abstract syntax tree
GF
logical forms,
other trees
translation
translation
translation


source text
dependency parser
treebank
dependency tree
ud2gf
abstract syntax tree
grammar
GF
logical forms,
other trees
translation
translation
translation


source text
dependency parser
treebank
dependency tree
ud2gf
dependency configuration
abstract syntax tree
GF
grammar
logical forms, other trees
translation
translation
translation

Rationale

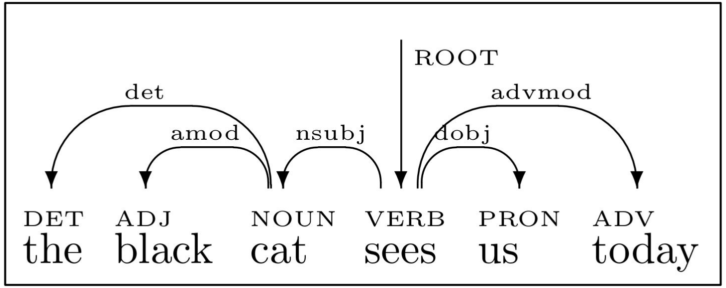
	dependencies	GF
parsing robustness	**robust**	brittle
parsing speed	**fast**	slow
disambiguation	**context-sensitive**	context-free
semantics	loose	**compositional**
generation	?	**accurate**
adding languages	low-level work	**high-level work**


source text
dependency parser
treebank
dependency tree
ud2gf
dependency configuration
abstract syntax tree
grammar
GF
logical forms,
other trees
translation
translation
translation

the black cat sees us today

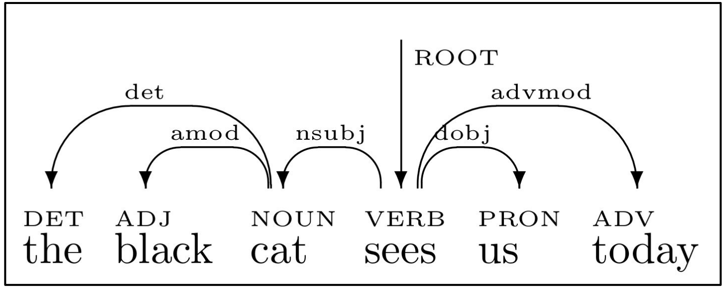
the black cat sees us today

dependency parser

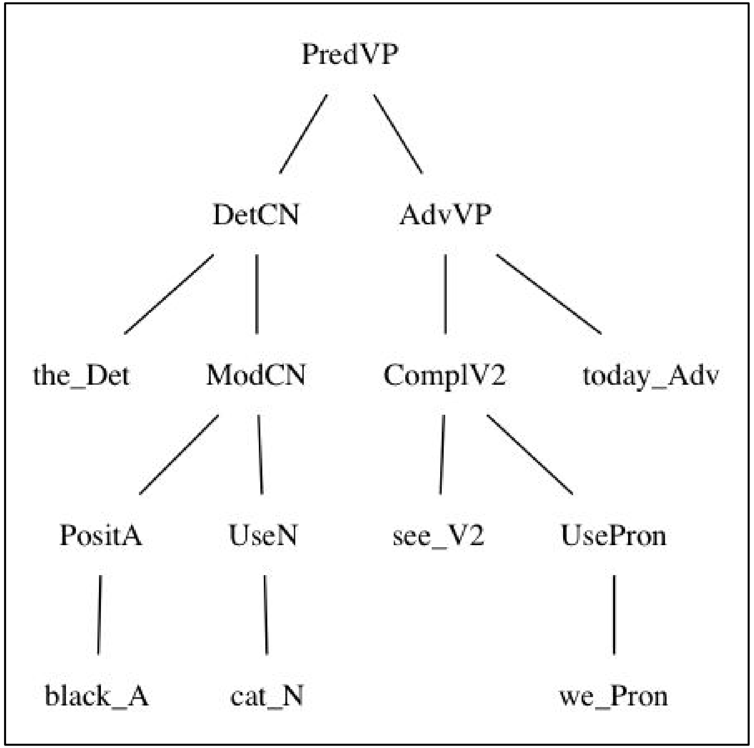


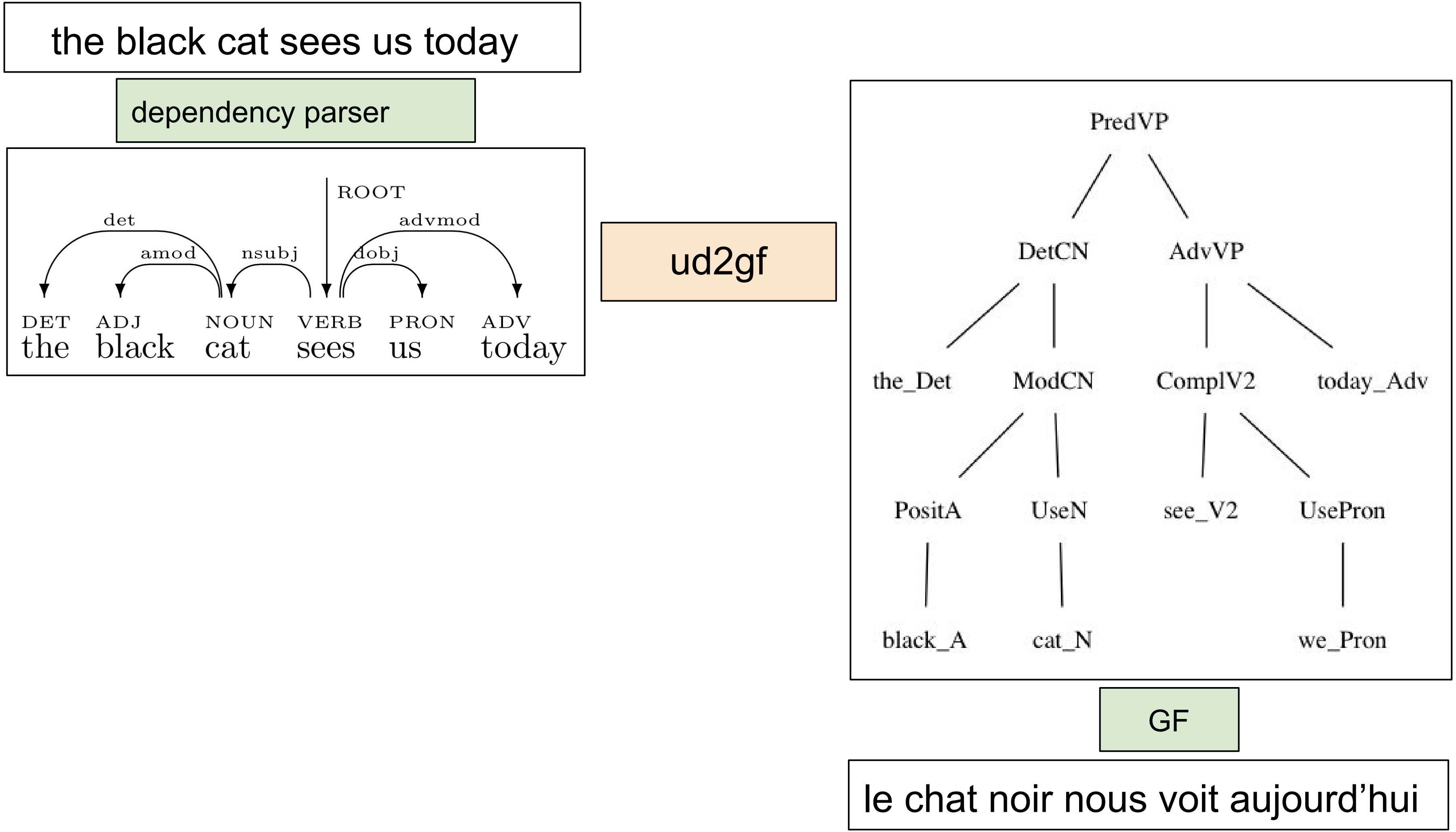
the black cat sees us today

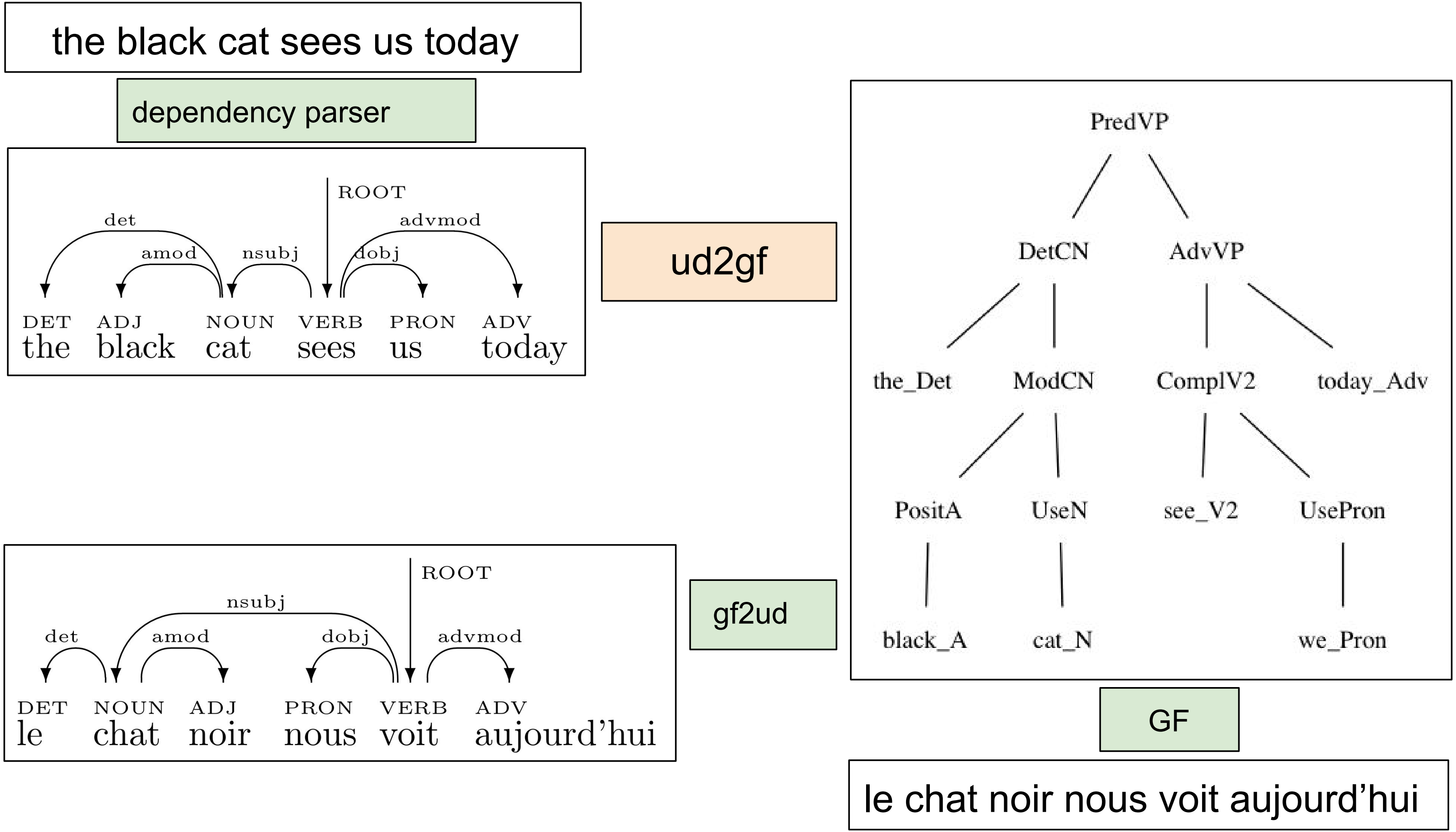
dependency parser



ud2gf




the black cat sees us today
dependency parser
ROOT
det
amod
nsubj
dobj
advmod
DET ADJ NOUN VERB PRON ADV
the black cat sees us today
ud2gf
PredVP
DetCN
AdvVP
the_Det
ModCN
ComplV2
today_Adv
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron
GF
le chat noir nous voit aujourd'hui



the black cat sees us today
dependency parser
ROOT
det
amod
nsubj
dobj
advmod
DET ADJ NOUN VERB PRON ADV
the black cat sees us today
ud2gf
PredVP
DetCN
AdvVP
the_Det
ModCN
ComplV2
today_Adv
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron
GF
le chat noir nous voit aujourd’hui
gf2ud
ROOT
nsubj
det
amod
dobj
advmod
DET NOUN ADJ PRON VERB ADV
le chat noir nous voit aujourd’hui

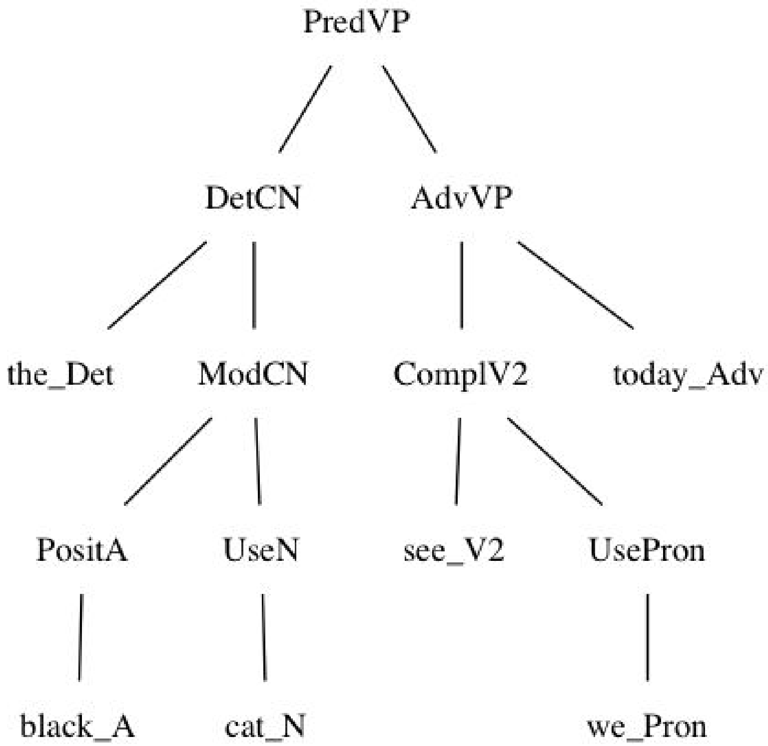
gf2ud

abstract syntax

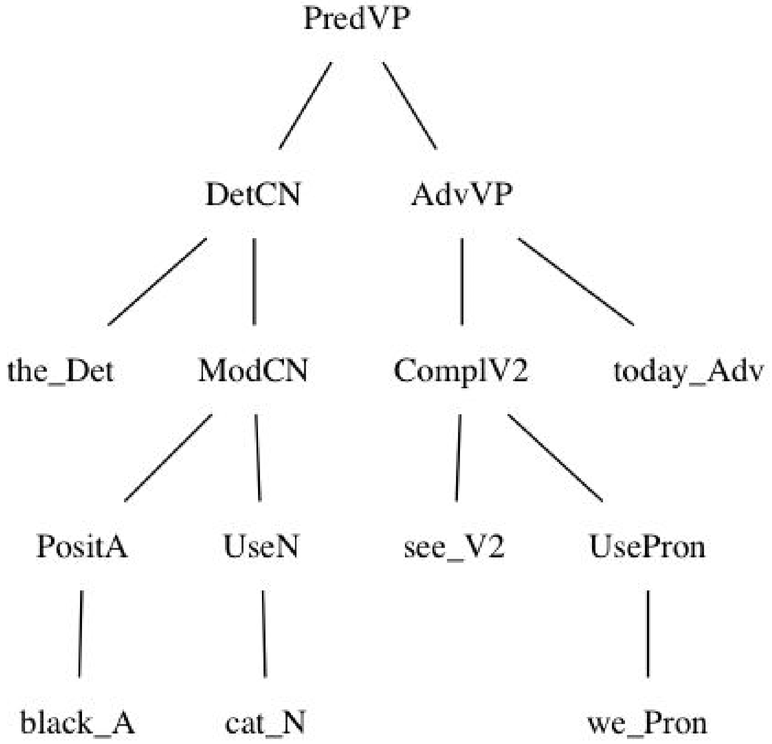
```
PredVP  : NP  -> VP  -> Cl

ComplV2 : V2  -> NP  -> VP

AdvVP   : VP  -> Adv -> VP

DetCN   : Det -> CN  -> NP

ModCN   : AP  -> CN  -> CN

UseN    : N    -> CN

UsePron : Pron -> NP

PositA  : A    -> AP
```

abstract syntax

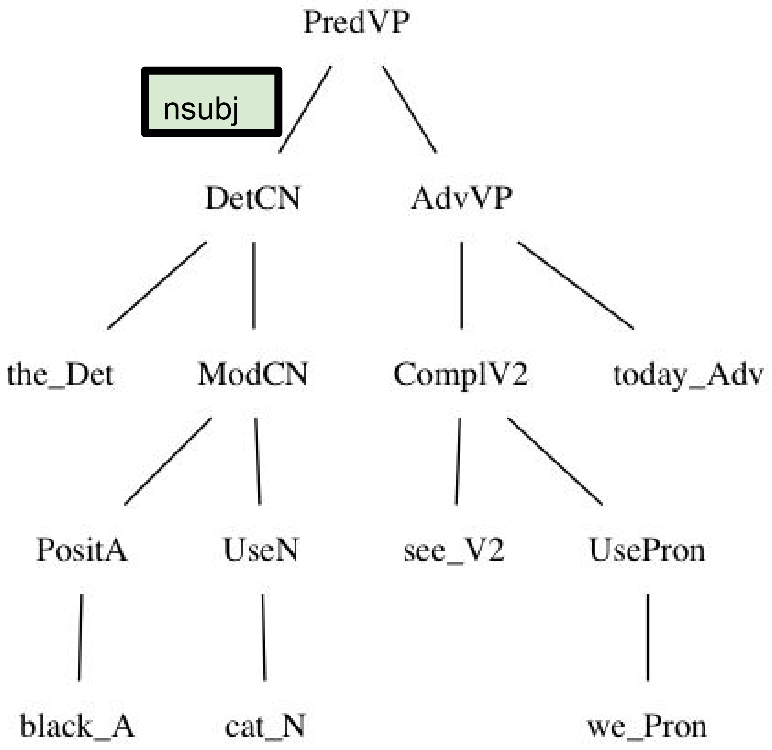
```
PredVP  : NP   -> VP   -> Cl

ComplV2 : V2   -> NP   -> VP

AdvVP   : VP   -> Adv -> VP

DetCN   : Det -> CN   -> NP

ModCN   : AP   -> CN   -> CN

UseN    : N     -> CN

UsePron : Pron -> NP

PositA  : A     -> AP
```



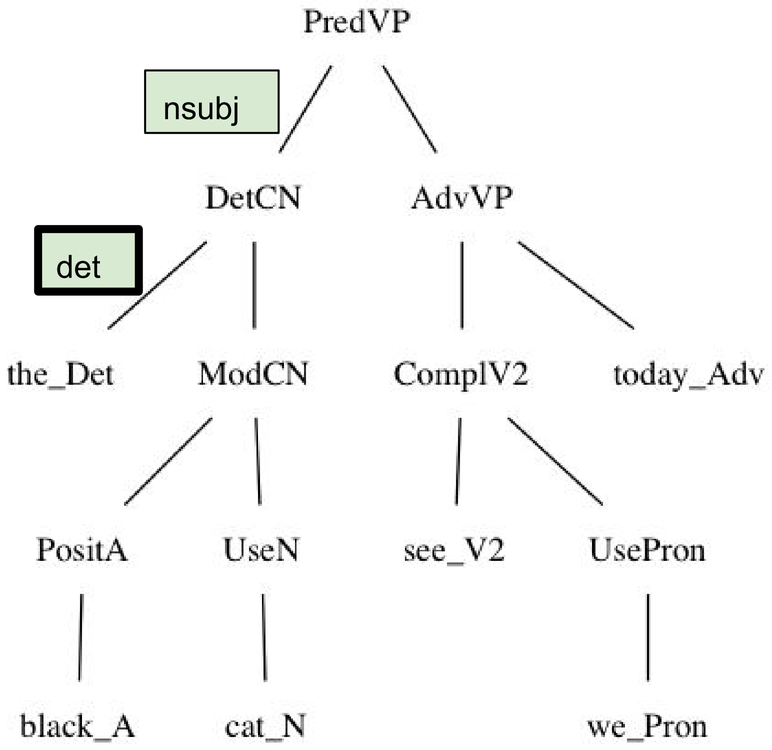
abstract syntax	**dependency configuration**
`PredVP  : NP   -> VP  -> Cl`	`nsubj  head`
`ComplV2 : V2   -> NP  -> VP`	`head   dobj`
`AdvVP   : VP   -> Adv -> VP`	`head   advmod`
`DetCN   : Det  -> CN  -> NP`	`det    head`
`ModCN   : AP   -> CN  -> CN`	`amod   head`
`UseN    : N    -> CN`	`head`
`UsePron : Pron -> NP`	`head`
`PositA  : A    -> AP`	`head`



abstract syntax	dependency configuration
`PredVP  : NP   -> VP  -> Cl`	nsubj head
`ComplV2 : V2   -> NP  -> VP`	head dobj
`AdvVP   : VP   -> Adv -> VP`	head advmod
`DetCN   : Det  -> CN  -> NP`	det head
`ModCN   : AP   -> CN  -> CN`	amod head
`UseN    : N    -> CN`	head
`UsePron : Pron -> NP`	head
`PositA  : A    -> AP`	head

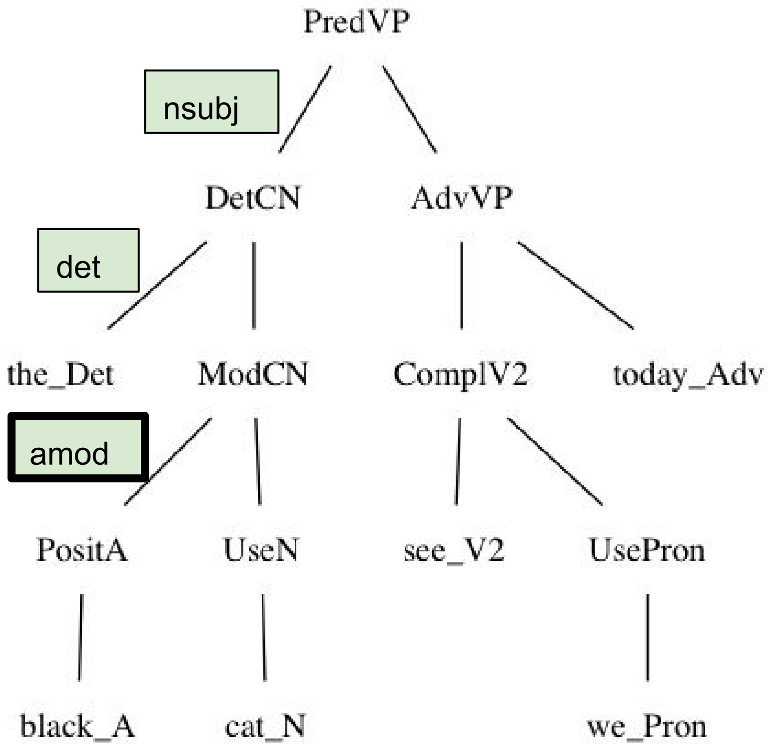


abstract syntax	**dependency configuration**
`PredVP  : NP   -> VP   -> Cl`	`nsubj  head`
`ComplV2 : V2   -> NP   -> VP`	`head   dobj`
`AdvVP   : VP   -> Adv  -> VP`	`head   advmod`
`DetCN   : Det  -> CN   -> NP`	`det    head`
`ModCN   : AP   -> CN   -> CN`	`amod   head`
`UseN    : N    -> CN`	`head`
`UsePron : Pron -> NP`	`head`
`PositA  : A    -> AP`	`head`

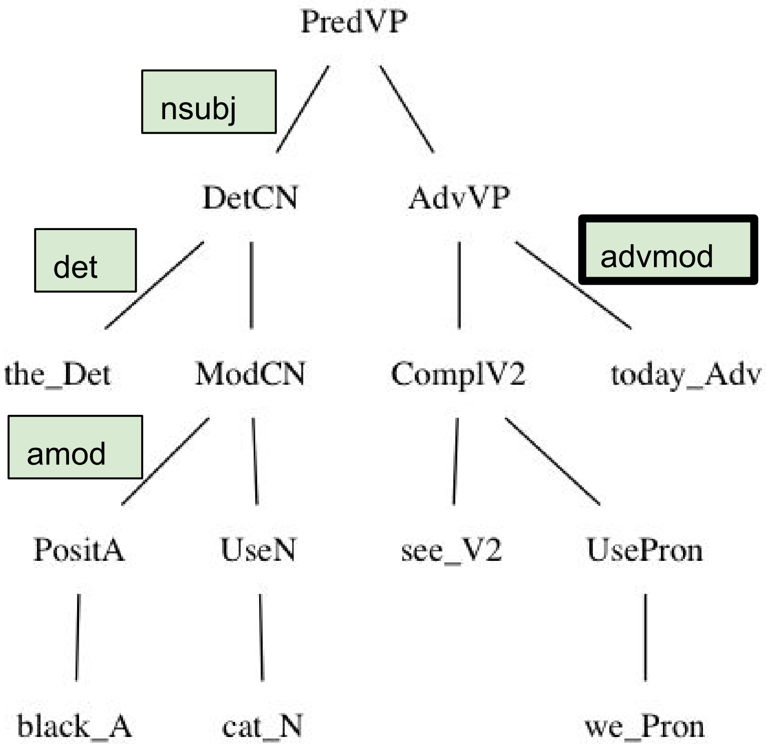


abstract syntax | **dependency configuration**

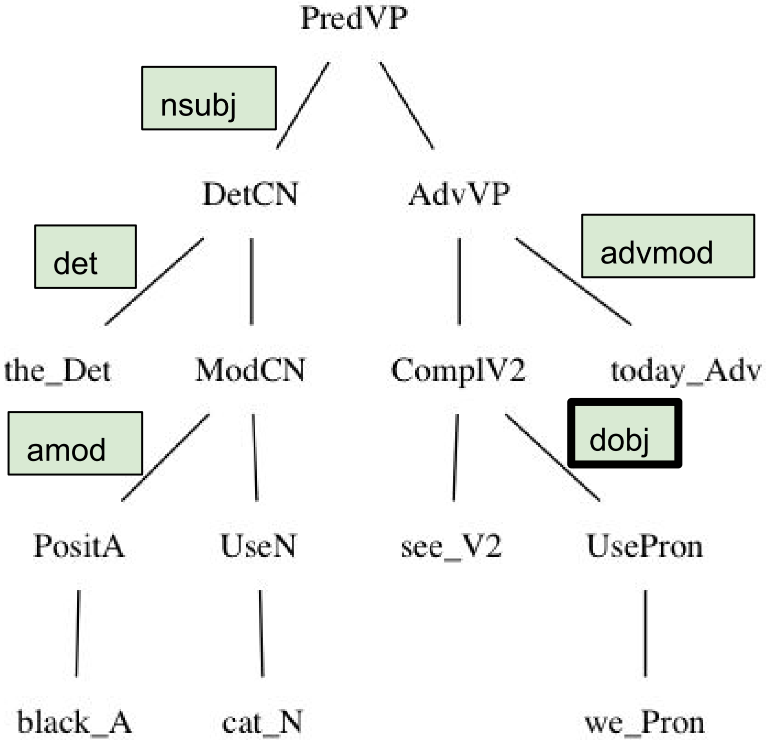
abstract syntax	dependency configuration
PredVP : NP -> VP -> Cl	nsubj head
ComplV2 : V2 -> NP -> VP	head dobj
AdvVP : VP -> Adv -> VP	head advmod
DetCN : Det -> CN -> NP	det head
ModCN : AP -> CN -> CN	amod head
UseN : N -> CN	head
UsePron : Pron -> NP	head
PositA : A -> AP	head

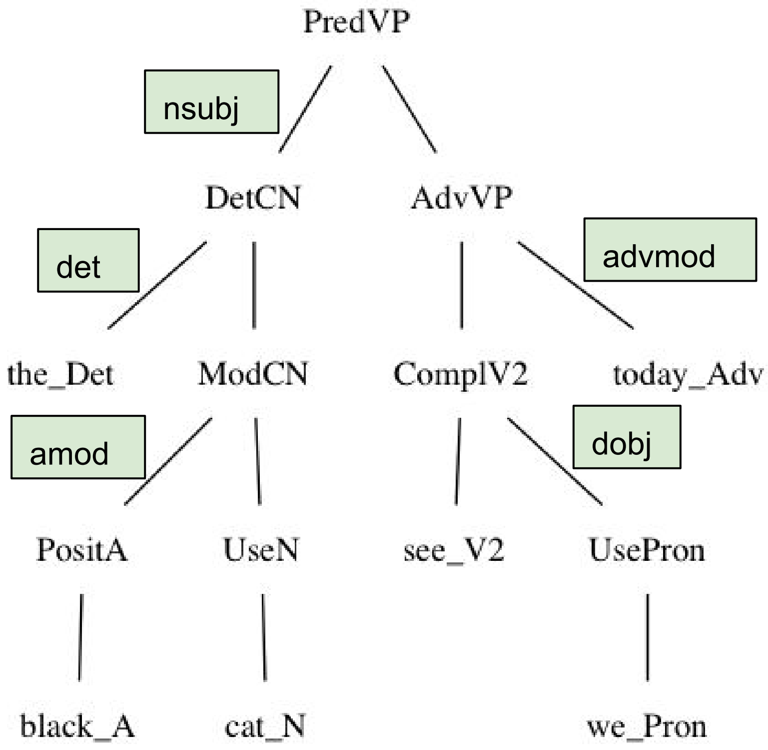


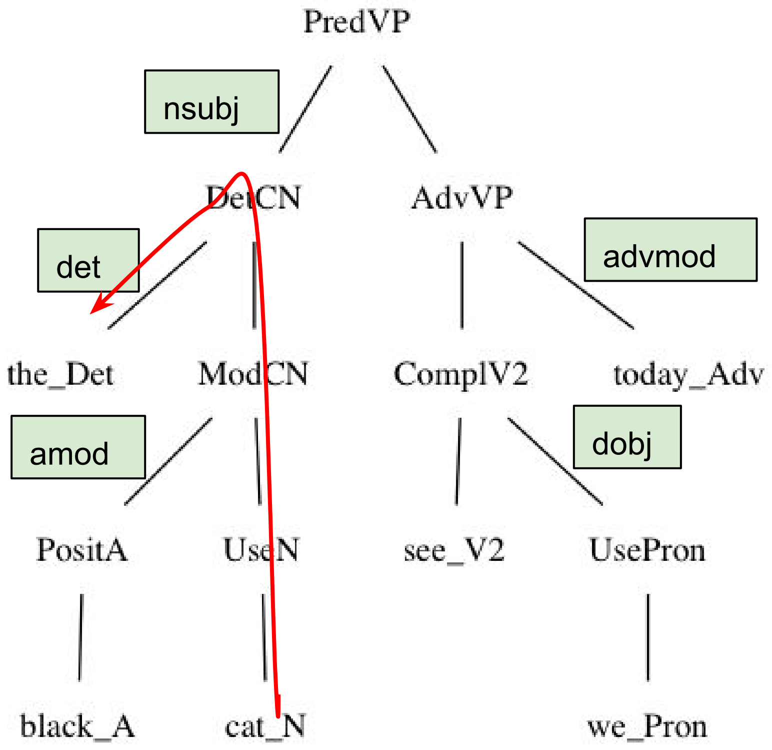
abstract syntax	dependency configuration
`PredVP  : NP   -> VP  -> Cl`	`nsubj head`
`ComplV2 : V2   -> NP  -> VP`	`head dobj`
`AdvVP   : VP   -> Adv -> VP`	`head advmod`
`DetCN   : Det  -> CN  -> NP`	`det head`
`ModCN   : AP   -> CN  -> CN`	`amod head`
`UseN    : N    -> CN`	`head`
`UsePron : Pron -> NP`	`head`
`PositA  : A    -> AP`	`head`

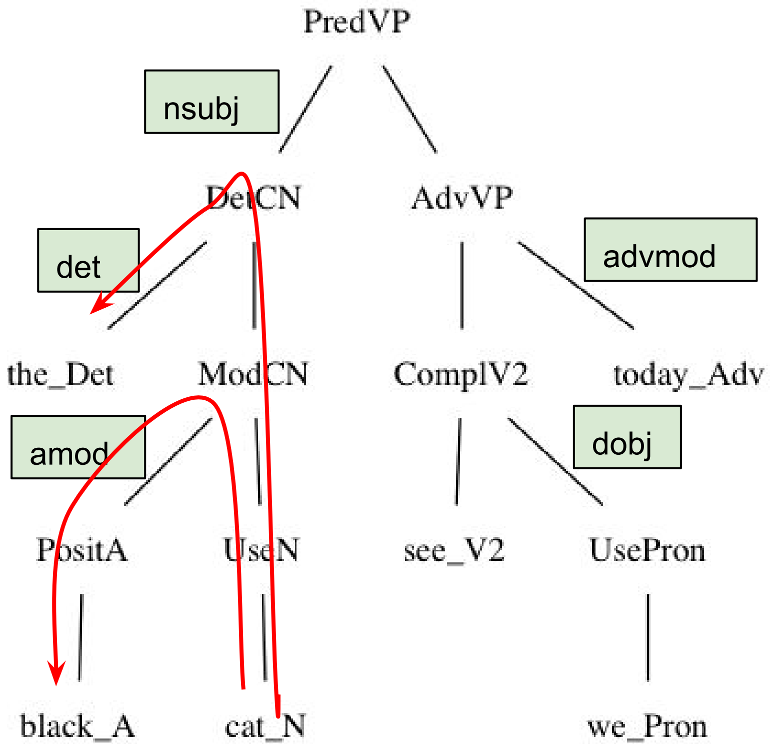


abstract syntax	dependency configuration
`PredVP : NP -> VP -> Cl`	`nsubj head`
`ComplV2 : V2 -> NP -> VP`	`head dobj`
`AdvVP : VP -> Adv -> VP`	`head advmod`
`DetCN : Det -> CN -> NP`	`det head`
`ModCN : AP -> CN -> CN`	`amod head`
`UseN : N -> CN`	`head`
`UsePron : Pron -> NP`	`head`
`PositA : A -> AP`	`head`

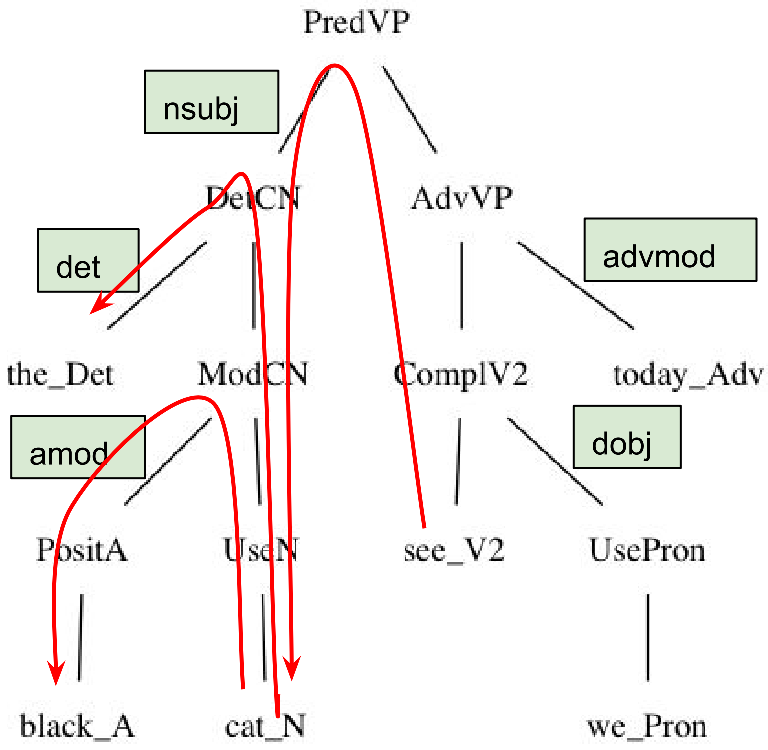


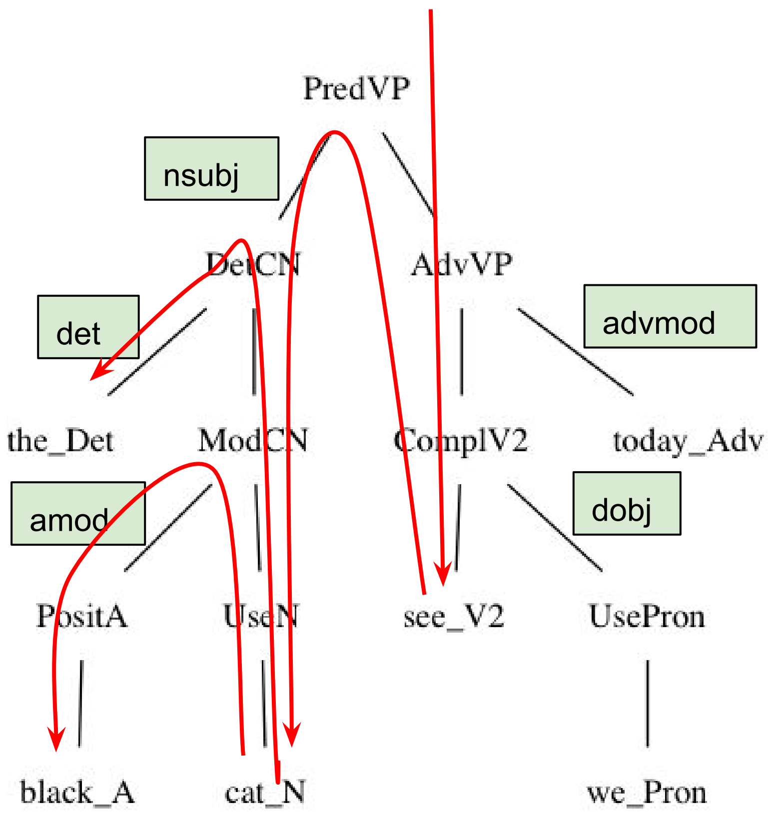

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron


PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

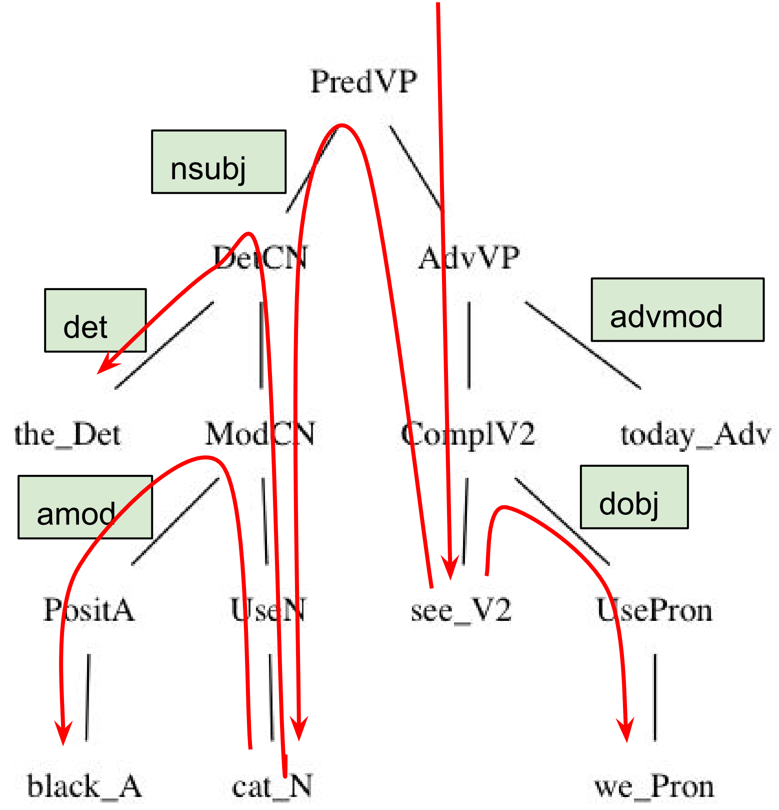


PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

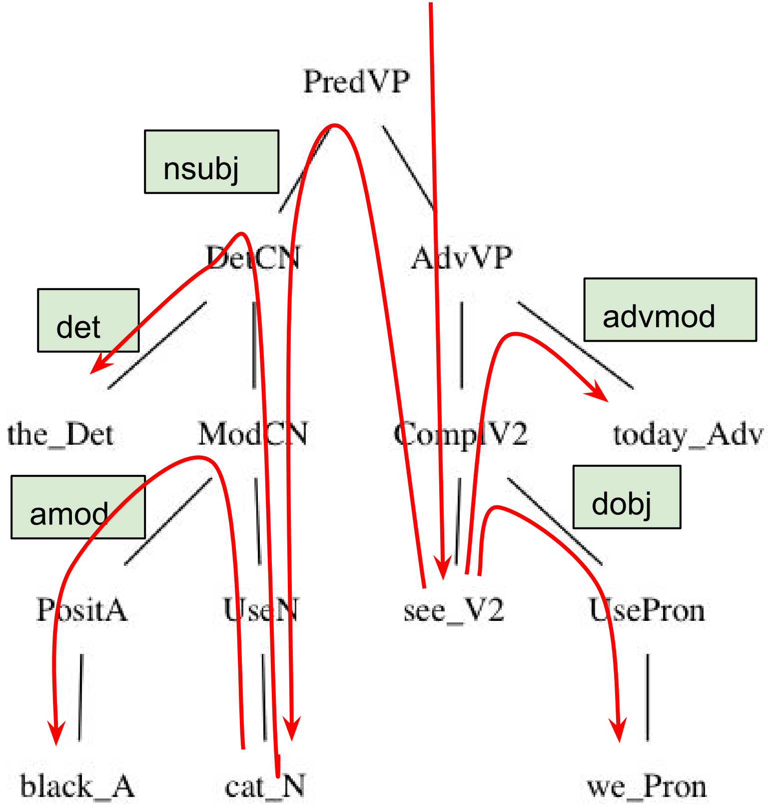

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron



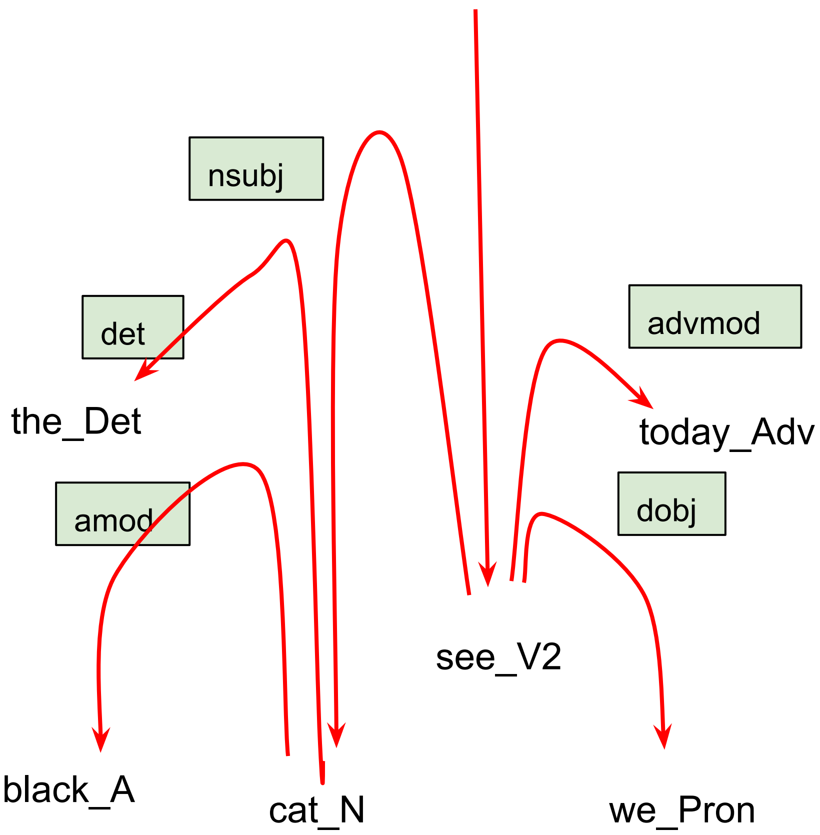
PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron



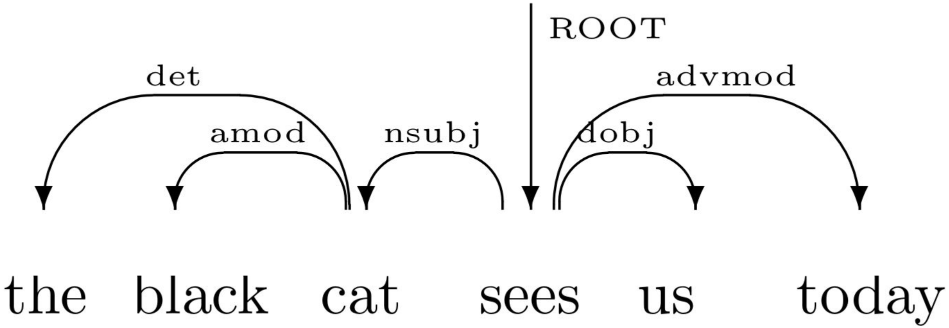
PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

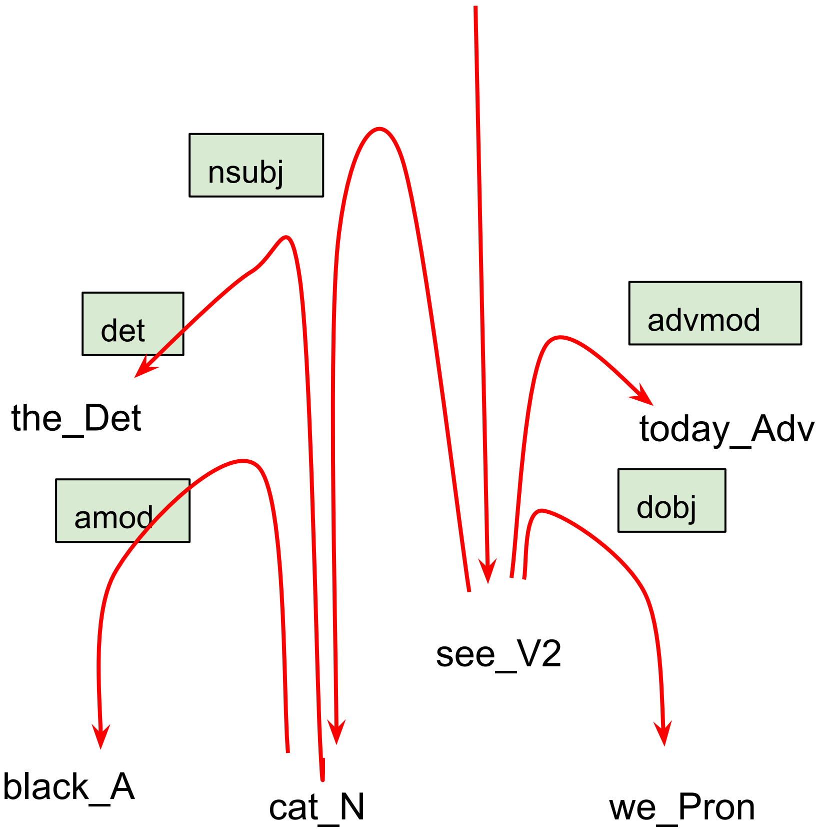


PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

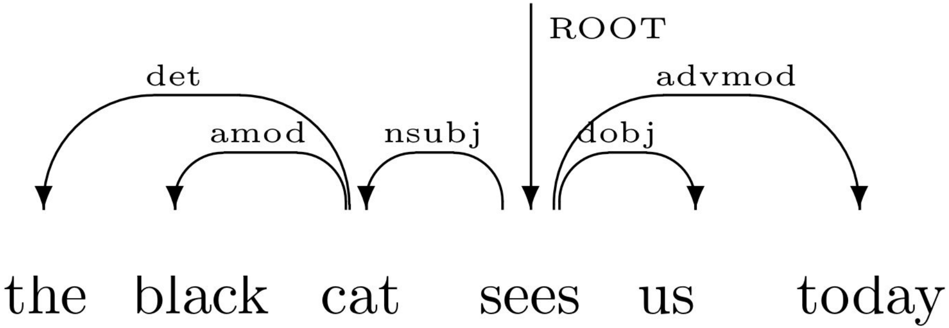


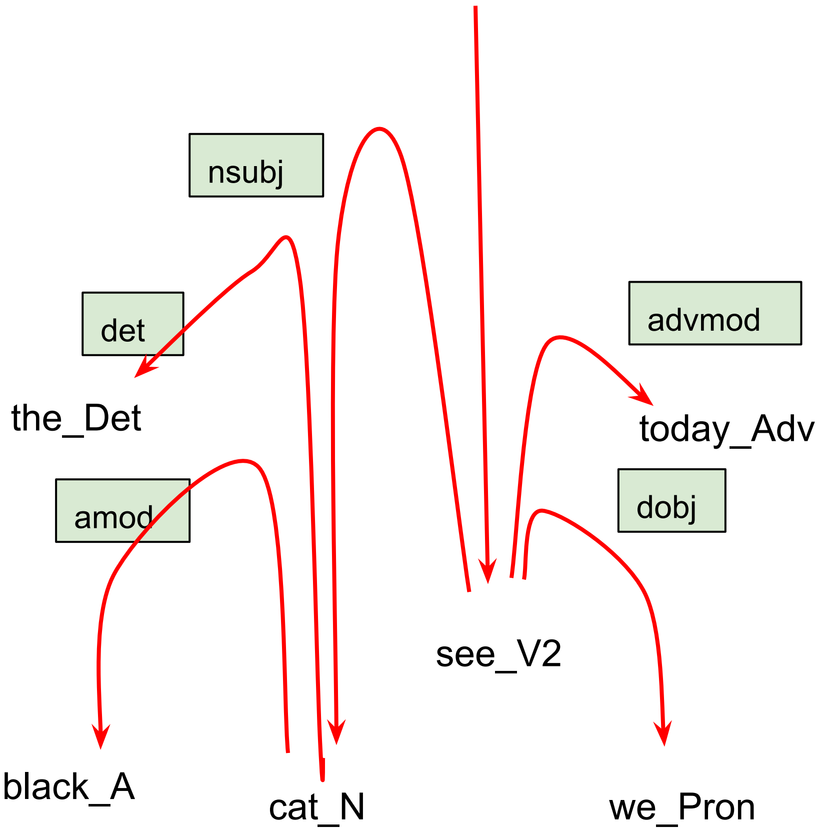
nsubj
det
the_Det
advmod
today_Adv
amod
dobj
see_V2
black_A
cat_N
we_Pron


ROOT
det
amod
nsubj
dobj
advmod
the black cat sees us today

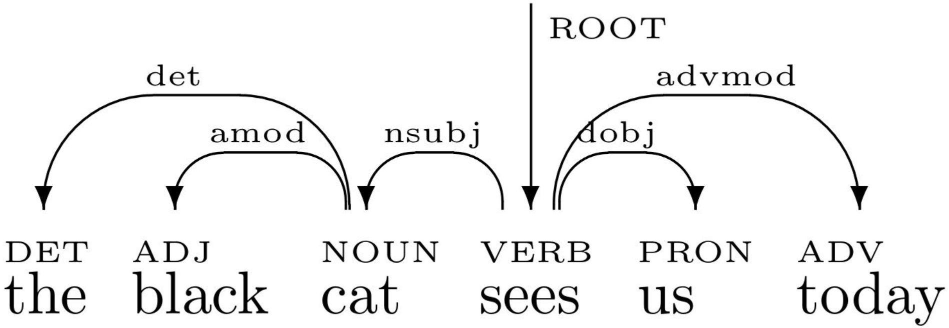

nsubj
det
the_Det
amod
black_A
cat_N
see_V2
advmod
today_Adv
dobj
we_Pron

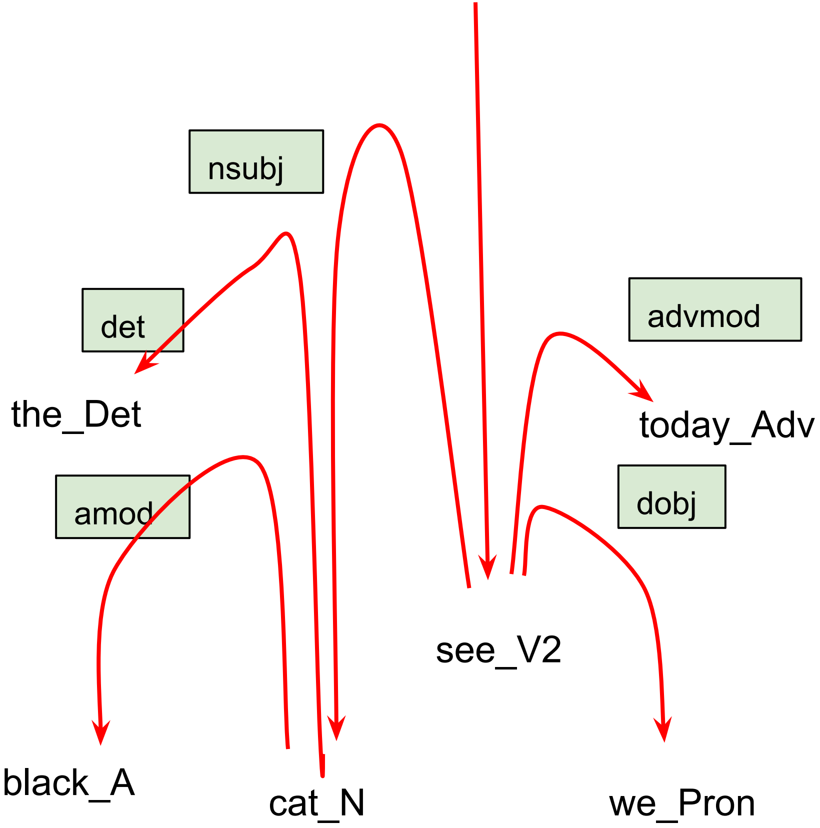
abstract syntax category	configuration
Det	DET
A	ADJ
N	NOUN
V2	VERB
Pron	PRON
Adv	ADV



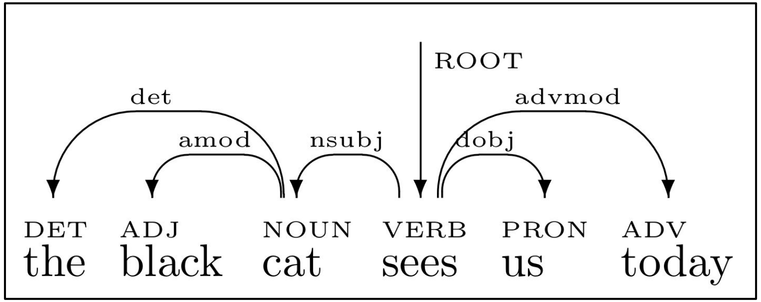


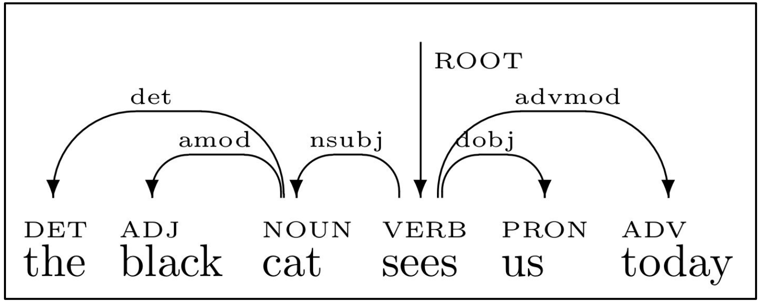
```
abstract syntax category configuration

Det          DET
A            ADJ
N            NOUN
V2           VERB
Pron         PRON
Adv          ADV
```



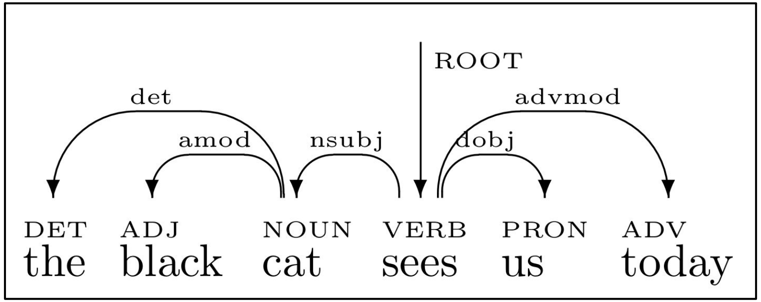


ud2gf


ROOT
det
advmod
amod
nsubj
dobj
DET the
ADJ black
NOUN cat
VERB sees
PRON us
ADV today



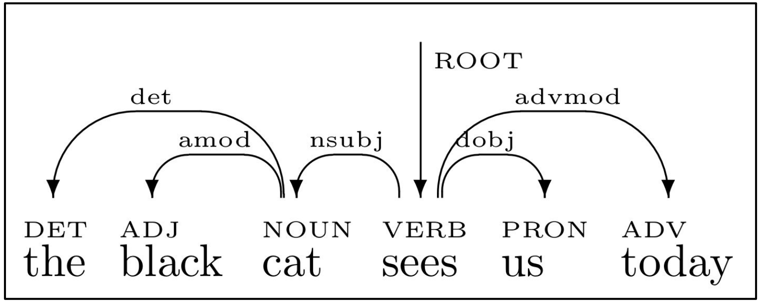
1	the	the	DET	_	3	det
2	black	black	ADJ	_	3	amod
3	cat	cat	NOUN	_	4	nsubj
4	sees	see	VERB	_	0	root
5	us	we	PRON	_	4	dobj
6	today	today	ADV	_	4	advmod



1	the	the	DET	_	3	det
2	black	black	ADJ	_	3	amod
3	cat	cat	NOUN	_	4	nsubj
4	sees	see	VERB	_	0	root
5	us	we	PRON	_	4	dobj
6	today	today	ADV	_	4	advmod

tree

```
root see VERB _ 4

    nsubj cat NOUN _ 3

        det the DET _ 1

        amod black ADJ _ 2

    dobj we PRON _ 5

    advmod today ADV _ 6
```



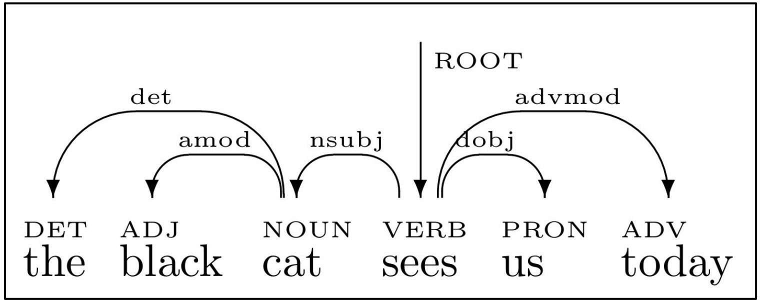
1	the	the	DET	_	3	det
2	black	black	ADJ	_	3	amod
3	cat	cat	NOUN	_	4	nsubj
4	sees	see	VERB	_	0	root
5	us	we	PRON	_	4	dobj
6	today	today	ADV	_	4	advmod

tree

```
root see VERB _ 4

    nsubj cat NOUN _ 3

        det the DET _ 1

        amod black ADJ _ 2

    dobj we PRON _ 5

    advmod today ADV _ 6
```

lexicon

see_V2	“see”
cat_N	“cat”
the_Det	“the”
black_A	“black”
we_Pron	“we”
today_Adv	“today”




```
1    the    the    DET   _   3   det
2    black  black  ADJ   _   3   amod
3    cat    cat    NOUN  _   4   nsubj
4    sees   see    VERB  _   0   root
5    us     we     PRON  _   4   dobj
6    today  today  ADV   _   4   advmod
```

tree

```
root see VERB _ 4
    nsubj cat NOUN _ 3
        det the DET _ 1
        amod black ADJ _ 2
    dobj we PRON _ 5
    advmod today ADV _ 6
```

lexicon

```
see_V2      “see”
cat_N       “cat”
the_Det     “the”
black_A     “black”
we_Pron     “we”
today_Adv   “today”
```

lexically annotated tree

```
root see_V2 V2 4
    nsubj cat_N N 3
        det the_Det Det 1
        amod black_A A 2
    dobj we_Pron Pron 5
    advmod today_Adv Adv 6
```

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod black_A A 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

abstract syntax	dependency configuration
`PredVP  : NP  -> VP  -> Cl`	`nsubj head`
`ComplV2 : V2  -> NP  -> VP`	`head  dobj`
`AdvVP   : VP  -> Adv -> VP`	`head  advmod`
`DetCN   : Det -> CN  -> NP`	`det   head`
`ModCN   : AP  -> CN  -> AP`	`amod  head`
`UseN    : N    -> CN`	`head`
`UsePron : Pron -> NP`	`head`
`PositA  : A    -> AP`	`head`

abstract syntax	dependency configuration		
`PredVP  : NP   -> VP  -> Cl`	nsubj head		
`ComplV2 : V2   -> NP  -> VP`	head dobj		
`AdvVP   : VP   -> Adv -> VP`	head advmod		
`DetCN   : Det  -> CN  -> NP`	det head	**exocentric**	head /= value
`ModCN   : AP   -> CN  -> CN`	amod head	**endocentric**	head = value
`UseN    : N    -> CN`	head		
`UsePron : Pron -> NP`	head		
`PositA  : A    -> AP`	head		

abstract syntax

```
PredVP  : NP  -> VP  -> Cl
ComplV2 : V2  -> NP  -> VP
AdvVP   : VP  -> Adv -> VP
DetCN   : Det -> CN  -> NP
ModCN   : AP  -> CN  -> AP
UseN    : N    -> CN
UsePron : Pron -> NP
PositA  : A    -> AP
```

dependency configuration

```
nsubj  head         exocentric
head   dobj         exocentric
head   advmod       endocentric
det    head         exocentric
amod   head         endocentric
head                exocentric
head                exocentric
head                exocentric
```

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod black_A A 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

```
tree

root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod black_A A 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

proceed bottom-up, i.e. subtrees before their head

tree

```
root see_V2 V2 4
    nsubj cat_N N 3
        det the_Det Det 1
        amod black_A A 2
    dobj we_Pron Pron 5
    advmod today_Adv Adv 6
```

```
tree

root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod black_A A 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

```
tree

root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod black_A A 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

A node is done when no more functions apply

tree	**exo**
root see_V2 V2 4	
nsubj cat_N N 3	
det the_Det Det 1	
amod black_A A 2	PositA 2
dobj we_Pron Pron 5	
advmod today_Adv Adv 6	

no endocentric functions apply
but one exocentric function does

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj we_Pron Pron 5
   advmod today_Adv Adv 6
```

exo

```
UsePron 5
```

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

tree

```
root see_V2 V2 4
   nsubj cat_N N 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

exo

```
UseN 3
```

tree

```
root see_V2 V2 4
   nsubj (UseN 3) [cat_N] CN 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

tree

```
root see_V2 V2 4
   nsubj (UseN 3) [cat_N] CN 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

endo

```
ModCN 2 3
```

exo

```
DetCN 1 3
```

when an endocentric function applies, use it first

```
tree

root see_V2 V2 4
   nsubj (ModCN 2 3) [(UseN 3),cat_N] CN 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

we have used 2 as subtree of 3

tree

```
root see_V2 V2 4
   nsubj (ModCN 2 3) [(UseN 3),cat_N] CN 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

exo

```
DetCN 1 3
```

tree

```
root see_V2 V2 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

tree

```
root see_V2 V2 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

exo

```
ComplV2 4 5
```

tree

```
root (ComplV2 4 5) [see_V2] VP 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

tree

```
root (ComplV2 4 5) [see_V2] VP 4
    nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
        det the_Det Det 1
        amod (PositA 2) [black_A] AP 2
    dobj (UsePron 5) [we_Pron] NP 5
    advmod today_Adv Adv 6
```

endo

```
AdvVP 4 6
```

tree

```
root (AdvVP 4 6) [(ComplV2 4 5),see_V2] VP 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

tree

```
root (AdvVP 4 6) [(ComplV2 4 5),see_V2] VP 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

exo

```
PredVP 3 4
```

tree

```
root (PredVP 3 4) [(AdvVP 4 6),(ComplV2 4 5),see_V2] VP 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
   advmod today_Adv Adv 6
```

The algorithm

1. Convert CoNLL graph to a tree datastructure, where
    - *Tree* ::= (*Node* $Tree_1$ … $Tree_n$)
    - *Node* ::= *Label Lemma POS Position*

2. Replace each (*Lemma,POS*) with (*Function,Cat*) by lexicon lookup.

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:
    - annotate each $Tree_1$,..., $Tree_n$
    - iterate for *Node* ::= *Label AST oldASTs Cat Position*:
        - if an endofunction *f : … Cat … -> Cat* applies,
          replace (*AST, oldASTs*) by ((*f...Label…*), *AST+oldASTs*)
        - else, if an exofunction *f : … Cat … -> Cat'* applies,
          replace (*AST, oldASTs, Cat*) by ((*f...Label…*), *AST+oldASTs,Cat'*)

4. Return the root node AST completed with subtrees following the links.

Problems

1. Convert CoNLL graph to a tree datastructure, where
    - *Tree* ::= (*Node* $Tree_1$ … $Tree_n$)
    - *Node* ::= *Label Lemma POS Position*
2. Replace each (*Lemma,POS*) with (*Function,Cat*) by lexicon lookup
    - there can be several candidate Functions and Cats
3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:
    - annotate each $Tree_1$,..., $Tree_n$
    - iterate for *Node* ::= *Label AST oldASTs Cat Position*:
        - if an endofunction *f : … Cat … -> Cat* applies,
          replace (*AST, oldASTs*) by ((*f...Label…*), *AST+oldASTs*)
            - there can be several endofunctions that apply
        - else, if an exofunction *f : … Cat … -> Cat'* applies,
          replace (*AST, oldASTs, Cat*) by ((*f...Label…*), *AST+oldASTs,Cat'*)
            - there can be several exofunctions that apply
            - an exofunction might only apply to an *oldAST*
4. Return the root node AST completed with subtrees following the links.
    - the tree may have nodes not referenced from the AST

2. Replace each (*Lemma,POS*) with (*Function,Cat*) by lexicon lookup

- there can be several candidate Functions and Cats

```
see_V2     “see”

see_V      “see”

see_VS     “see”



today_1_Adv “today”

today_2_Adv “today”
```

2. Replace each (*Lemma,POS*) with (*Function,Cat*) by lexicon lookup

- there can be several candidate Functions and Cats

```
see_V2      "see"

see_V       "see"

see_VS      "see"


today_1_Adv "today"

today_2_Adv "today"
```

```
root see_V2:V2 [see_V:V, see_VS:VS] 4

   nsubj cat_N N 3

      det the_Det Det 1

      amod black_A A 2

   dobj we_Pron Pron 5

   advmod today_1_Adv:Adv [today_2_Adv:Adv] 6
```

Solution: save all candidates in the AST list

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,..., $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
  - if an endofunction *f : … Cat … -> Cat* applies,
    replace (*AST, oldASTs*) by ((*f...Label…*), *AST+oldASTs*)
    - there can be several endofunctions that apply

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,..., $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
    - if an endofunction *f : … Cat … -> Cat* applies, replace (*AST, oldASTs*) by ((*f...Label…*), *AST+oldASTs*)
        - there can be several endofunctions that apply

```
nsubj (UseN 4) [cat_N] CN 4

   det the_Det Det 1

   amod (PositA 2) [big_A] AP 2

   amod (PositA 3) [black_A] AP 3
```

```
ModCN 2 : 4 -> CN

ModCN 3 : 4 -> CN
```

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,..., $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
    - if an endofunction *f* : … *Cat* … -> *Cat* applies, replace (*AST, oldASTs*) by ((*f...Label…*), *AST+oldASTs*)
        - there can be several endofunctions that apply

```
nsubj (ModCN 2) [(ModCN 3 (UseN 4)),cat_N] CN 4
   det the_Det Det 1
   amod (PositA 2) [big_A] AP 2
   amod (PositA 3) [black_A] AP 3
```

Solution: apply them all

- either, in some order
- or, storing all alternatives

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,..., $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
  - else, if an exofunction *f : … Cat … -> Cat’* applies, replace (*AST, oldASTs, Cat*) by ((*f...Label…*), *AST+oldASTs,Cat’*)
    - there can be several exofunctions that apply
    - an exofunction might only apply to an *oldAST*

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,..., $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
    - else, if an exofunction *f : … Cat … -> Cat’* applies, replace (*AST, oldASTs, Cat*) by ((*f...Label…*), *AST+oldASTs,Cat’*)
        - there can be several exofunctions that apply
        - an exofunction might only apply to an *oldAST*

```
root see_V:V [see_V2:V2, see_VS:VS] 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
```

```
UseV    : V  -> VP
ComplV2 : V2 -> NP -> VP
ComplVS : VS -> S  -> VP
```

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,..., $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
  - else, if an exofunction *f : … Cat … -> Cat’* applies, replace (*AST, oldASTs, Cat*) by ((*f...Label…*), *AST+oldASTs,Cat’*)
    - there can be several exofunctions that apply
    - an exofunction might only apply to an *oldAST*

```
root see_V:V [see_V2:V2, see_VS:VS] 4

  nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3

    det the_Det Det 1

    amod (PositA 2) [black_A] AP 2

  dobj (UsePron 5) [we_Pron] NP 5
```

```
UseV    : V  -> VP
ComplV2 : V2 -> NP -> VP
ComplVS : VS -> S  -> VP
```

Solution:

- a function must find all its arguments!

3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:

- annotate each $Tree_1$,…, $Tree_n$
- iterate for *Node* ::= *Label AST oldASTs Cat Position*:
  - else, if an exofunction *f : … Cat … -> Cat'* applies, replace (*AST, oldASTs, Cat*) by ((*f…Label…*), *AST+oldASTs,Cat'*)
    - there can be several exofunctions that apply
    - an exofunction might only apply to an *oldAST*

```
root (ComplV2 4.2 5):VP [see_V:V, see_V2:V2, see_VS:VS] 4
  nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
    det the_Det Det 1
    amod (PositA 2) [black_A] AP 2
  dobj (UsePron 5) [we_Pron] NP 5
```

```
UseV    : V  -> VP
ComplV2 : V2 -> NP -> VP
ComplVS : VS -> S  -> VP
```

Solution:

- a function must find all its arguments!
- choose the solutions that use the maximal number of arguments

4. Return the root node AST completed with subtrees following the links.
    - the tree may have nodes not referenced from the AST

4. Return the root node AST completed with subtrees following the links.

- the tree may have nodes not referenced from the AST

the black cat sees us ;-) today

```
root (PredVP 3 4) [(AdvVP 4 7),(ComplV2 4 5),see_V2] VP 4
   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3
      det the_Det Det 1
      amod (PositA 2) [black_A] AP 2
   dobj (UsePron 5) [we_Pron] NP 5
      discourse “;-)” [] String 6
   advmod today_Adv Adv 7
```

Solution: wrap them with **backup functions** and attach to their head as an adjunct

4. Return the root node AST completed with subtrees following the links.

- the tree may have nodes not referenced from the AST

the black cat sees us ;-) today

```
root (PredVP 3 4) [(AdvVP 4 7),(ComplV2 4 5),see_V2] VP 4

   nsubj (DetCN 1 3) [(ModCN 2 3),(UseN 3),cat_N] NP 3

      det the_Det Det 1

      amod (PositA 2) [black_A] AP 2

   dobj ModBackupNP 6 5 [(UsePron 5),we_Pron] NP 5

      discourse StringBackup 6 [“;-)”] Backup 6

   advmod today_Adv Adv 7
```

Solution: wrap them with **backup functions** and attach to their head as an adjunct

```
ModBackupNP :
  Backup -> NP -> NP
StringBackup :
  String -> Backup
```

Conclusions so far

General method for dependency-to-abstract conversion

- any dependency scheme
- any GF abstract syntax
- any language (of course)

Maximal use of nodes in grammatical structure

Remaining nodes treated as adjuncts by backups

Room for improvements

Recognizing syncategorematic words

Using morphological tags

Recognizing particle verbs

Introducing word-sense disambiguation

Syncategorematic words

```
fun CompAP : AP -> VP
lin CompAP ap = “is” ++ AP
```

the cat is black

```
root PredVP 2 4 [CompAP 4, UseA 4, black_A] Cl 4
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
   cop be_V [] V 3
```

must be treated by Backup

Syncategorematic words

```
fun CompAP : AP -> VP
lin CompAP ap = “is” ++ AP
```

the cat is black

Solution: **add helper functions**

```
CompAP_ : Cop_ -> AP -> VP cop head
be_Cop_ : Cop_
```

```
root PredVP 2 4 [CompAP 4, UseA 4, black_A] Cl 4
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
   cop be_V [] V 3
```

Syncategorematic words

```
fun CompAP : AP -> VP
lin CompAP ap = “is” ++ AP
```

the cat is black

Solution: **add helper functions**

```
CompAP_ : Cop_ -> AP -> VP cop head
be_Cop_ : Cop_
```

```
root PredVP 2 4 [CompAP_ 3 4, UseA 4, black_A] Cl 4
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
   cop be_Cop_ [] Cop_ 3
```

Syncategorematic words

```
fun CompAP : AP -> VP
lin CompAP ap = "is" ++ AP
```

the cat is black

Solution: **add helper functions**

```
CompAP_ : Cop_ -> AP -> VP cop head
be_Cop_ : Cop_
```

```
root PredVP 2 4 [CompAP_ 3 4, UseA 4, black_A] Cl 4
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
   cop be_Cop_ [] Cop_ 3
```

```
PredVP (DetCN the_Det (UseN cat_N)) (CompAP_ be_Cop (UseA black_A))=
 ?
```

Syncategorematic words

```
fun CompAP : AP -> VP
lin CompAP ap = “is” ++ AP
```

the cat is black

Solution: **add helper functions**

```
CompAP_ : Cop_ -> AP -> VP cop head
be_Cop_ : Cop_
```

Eliminate them by **definitions**:

```
CompAP_ cop ap = CompAP ap
```

```
root PredVP 2 4 [CompAP_ 3 4, UseA 4, black_A] Cl 4
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
   cop be_Cop_ [] Cop_ 3
```

```
PredVP (DetCN the_Det (UseN cat_N)) (CompAP_ be_Cop (UseA black_A))=
```

Syncategorematic words

```
fun CompAP : AP -> VP
lin CompAP ap = “is” ++ AP
```

the cat is black

Solution: **add helper functions**

```
CompAP_ : Cop_ -> AP -> VP cop head
be_Cop_ : Cop_
```

Eliminate them by **definitions**:

```
CompAP_ cop ap = CompAP ap
```

```
root PredVP 2 4 [CompAP_ 3 4, UseA 4, black_A] Cl 4
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
   cop be_Cop_ [] Cop_ 3
```

```
PredVP (DetCN the_Det (UseN cat_N)) (CompAP_ be_Cop (UseA black_A))=
PredVP (DetCN the_Det (UseN cat_N)) (CompAP         (UseA black_A))
```

Morphology

```
PresCl : Cl -> S
PastCl : Cl -> S
```

the cat sleeps
the cat slept

```
root (PredVP 2 3) [UseV 3, sleep_V] Cl 3
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
```

```
Exo

PresCl 3
PastCl 3
```

Morphology

```
PresCl : Cl -> S
PastCl : Cl -> S
```

the cat sleeps
the cat slept

Solution: **add conditions on tags**

```
PresCl : Cl -> S    Tense=Pres
PastCl : Cl -> S    Tense=Past
```

```
root (PredVP 2 3) [UseV 3, sleep_V] Cl 3
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
```

Exo

```
PresCl 3
PastCl 3
```

Morphology

```
PresCl : Cl -> S
PastCl : Cl -> S
```

the cat sleeps
the cat slept

Solution: **add conditions on tags**

```
PresCl : Cl -> S    Tense=Pres
PastCl : Cl -> S    Tense=Past
```

```
root (PredVP 2 3) [UseV 3, sleep_V] Cl Tense=Pres 3
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
```

Exo

```
PresCl 3
PastCl 3
```

```
root (PredVP 2 3) [UseV 3, sleep_V] Tense=Past S 3
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
```

Exo

```
PresCl 3
PastCl 3
```

Morphology

```
PresCl : Cl -> S
PastCl : Cl -> S
```

the cat sleeps
the cat slept

Solution: **add conditions on tags**

```
PresCl : Cl -> S    Tense=Pres
PastCl : Cl -> S    Tense=Past
```

```
root (PresCl 3) [PredVP 2 3, UseV 3, sleep_V] Tense=Pres S 3
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
```

```
root (PastCl 3) [PredVP 2 3, UseV 3, sleep_V] Tense=Past S 3
   nsubj (DetCN 1 2) [UseN 2, cat_N] NP 2
      det the_Det [] Det 1
```

Experiments

GF grammar for 16 languages

- Resource Grammar Library
- Large lexicon (Wordnet, Wiktionary…)

Configuration with UD labels and POS tags

Analysing and translating UD treebanks

- English, Finnish, Swedish

Connecting GF generation to UD parser front-end

First results

language	sentences	Complete parse	Without backup	Nodes covered	Without backup
English	2077	1277	669	93%	62%
Finnish	648	322	37	87%	39%
Swedish	1219	458	124	88%	39%

`UD_English/en-ud-test.conllu`

`UD_Finnish/fi-ud-test.conllu`

`UD_Swedish/sv-ud-test.conllu`

ud2gf

```
$ ud2gf -lEng -t10000 -k3000 -a1 -g1 -Dscamifgtn
    -CUDTranslate.labels,UDTranslateEng.labels
    treebanks/UD_English/en-ud-test.conllu
```

```
Usage: ud2gf <opts> <conll-file>

 -N<int>   -- max number of trees analysed, default no limit

 -G<file>  -- grammar file, default 'UDTranslate.pgf', based on RGL with wide-coverage extensions

 -C<files> -- configuration files, comma-separated, default 'UDTranslate.labels', mapping UD to RGL

 -L<file>  -- read lexicon from file, default 'pgf' i.e. build from the grammar

 -l<lang>  -- source language (ISO-3 code), default Eng

 -k<int>   -- kill: max number of trees considered per sentence, default no limit

 -t<int>   -- timeout: max number of milliseconds per sentence, default no limit

 -a<int>   -- max number of raw abstract trees shown per sentence, default no limit

 -g<int>   -- max number of GF trees shown per sentence, default no limit
```

```
-D<disp>  -- display: what is shown (default sdagt):
              s -- source sentence
              c -- conll tree verbatim
              r -- dependency tree structured, now annotations
              d -- dependency tree, no annotations, showing parser coverage
              a -- dependency tree, annotated with GF lexical functions
              m -- dependency tree, annotated with GF trees
              f -- dependency tree, no annotations, showing interpreter converage
              g -- abstract syntax tree, GF
              t -- translations to languages covered
              i -- statistics of interpreter coverage per tree
              n -- global statistics of coverage
```

Configuration files

Function labellings	`PredVP : NP -> VP -> Cl ; nsubj head`
Helper function definitions	`UseComp_ cop comp = UseComp comp`
Category mappings	`Prep ADP`
Backup functions	`* InterjBackup : Interj -> Backup`
Comments	`-- this is a comment`

Morphological conditions

In functions

```
PosPastCl_ : Cl -> S ;  Tense=Past
PosPastCl_ cl = UseCl (TTAnt TPast ASimul) PPos cl
```

In categories

```
RP    PRON  PronType=Rel
```

On lemmas

```
Cop_  VERB  lemma=be
```

Source code: line counts

```
202 GetConfig.hs      -- parsing and type checking configuration files

198 Translate.hs      -- main conversion by endo- and exocentric functions

278 TreeConv.hs       -- tree data structures, initial conversion with lexicon

210 UD2GF.hs          -- top loop with display options


84 UDTranslate.gf                 247 UDTranslate.labels

17 UDTranslateEng.gf               24 UDTranslateEng.labels

17 UDTranslateFin.gf               14 UDTranslateFin.labels

86 UDTranslateFunctor.gf           21 UDTranslateSwe.labels

17 UDTranslateSwe.gf                                          1415 total
```

Conclusion: from garden to bush

We have learned our botany in a garden.

But it also works when exploring the bush!

Acknowledgements

Krasimir Angelov (Chalmers)

Filip Ginter (Turku)

Richard Johansson (Sprakbanken)

Joakim Nivre (Uppsala)