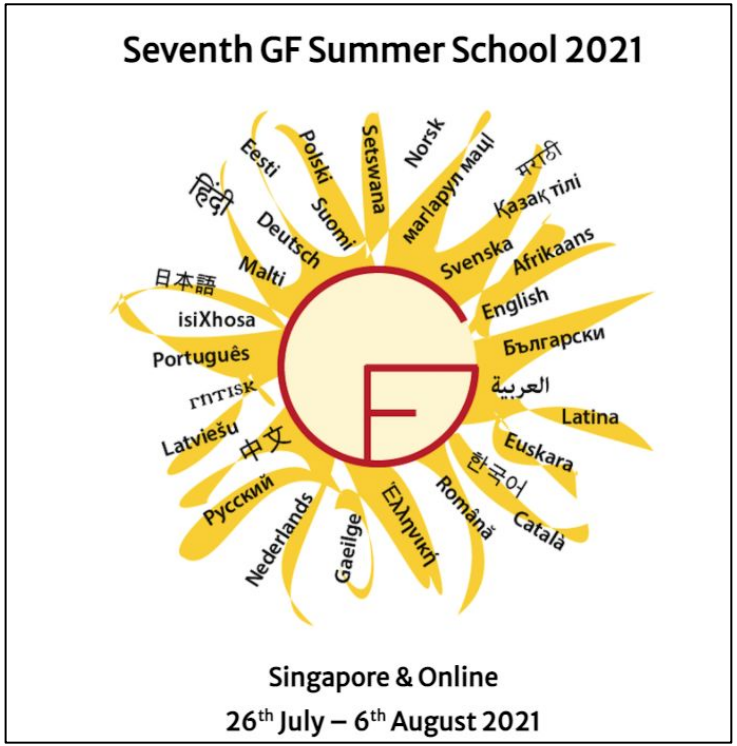
Grammatical Framework and Universal Dependencies

Aarne Ranta



Volume 46, Issue 2

June 2020

June 01 2020

Abstract Syntax as Interlingua: Scaling Up the Grammatical Framework from Controlled Languages to Robust Pipelines

Aarne Ranta, Krasimir Angelov, Normunds Gruzitis, Prasanth Kolachina

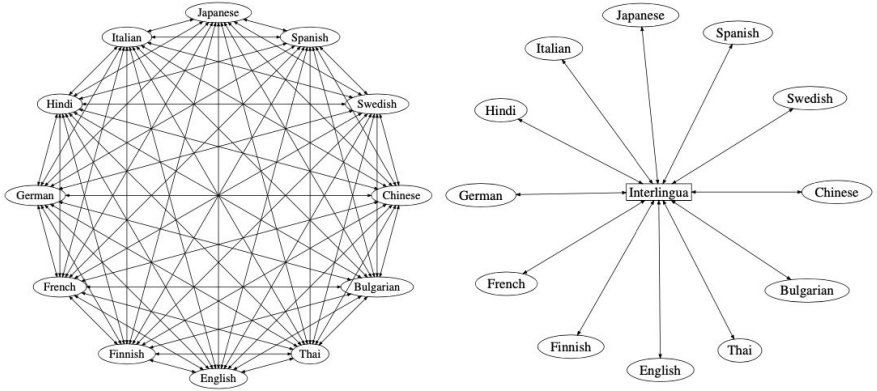
> Author and Article Information

Computational Linguistics (2020) 46 (2): 425–486.

https://doi.org/10.1162/coli_a_00378 **Article history**

Computational Grammar

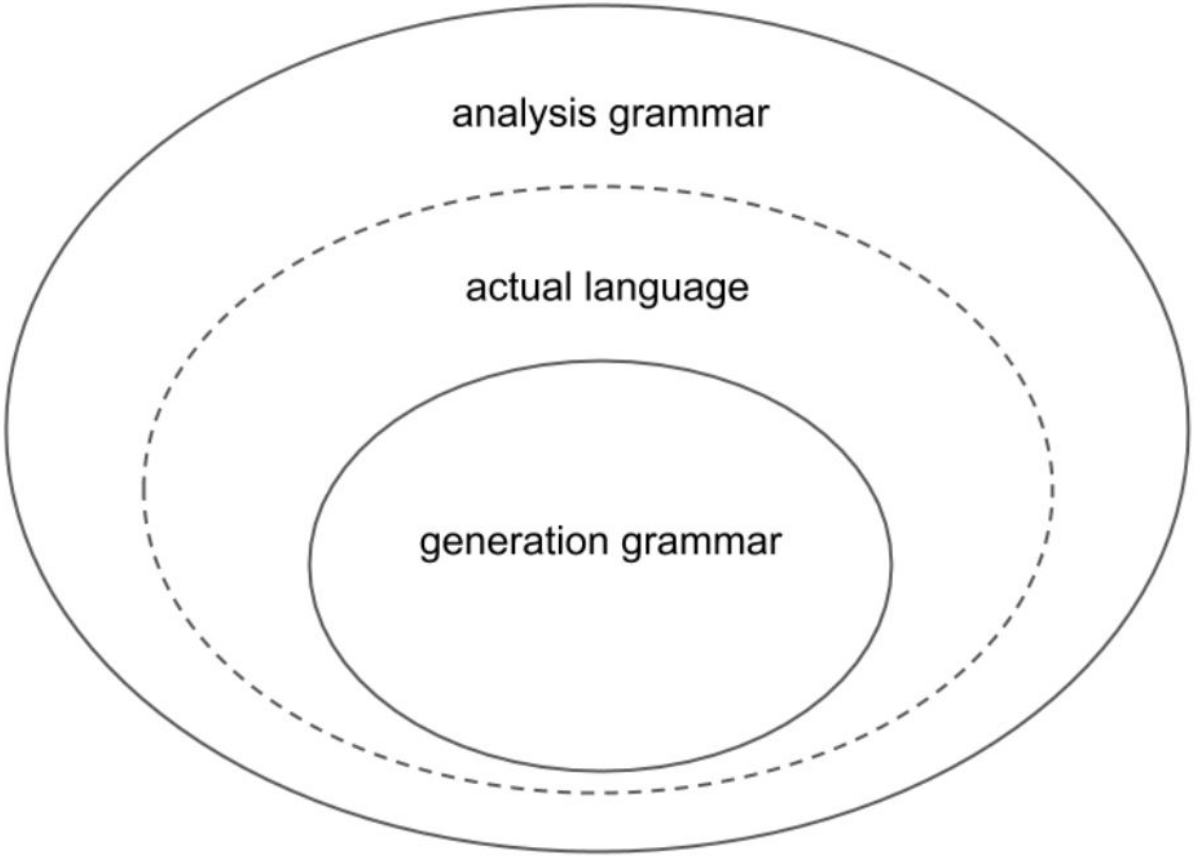
An Interlingual Perspective



Aarne Ranta

February 25, 2021

http://www.cse.chalmers.se/~aarne/grammarbook.pdf


analysis grammar
actual language
generation grammar

UD = Universal Dependencies

https://universaldependencies.org/

the

black

cat

sees

us

now

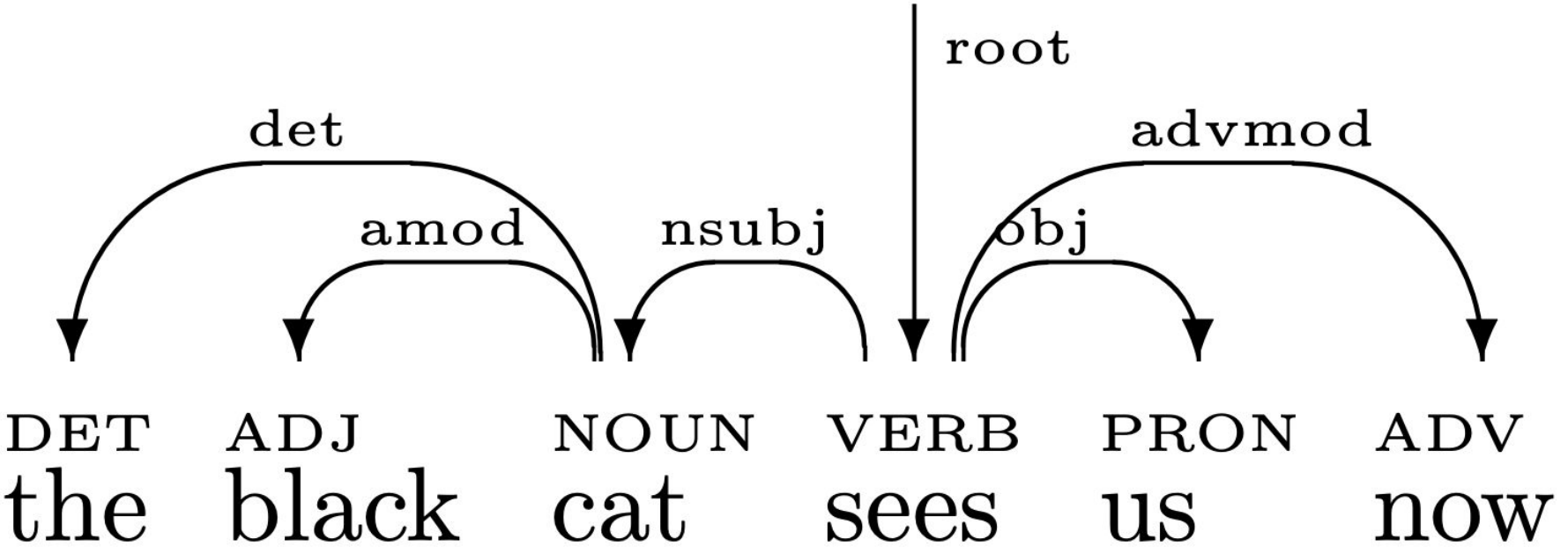
the	the
black	black
cat	cat
sees	see
us	we
now	now

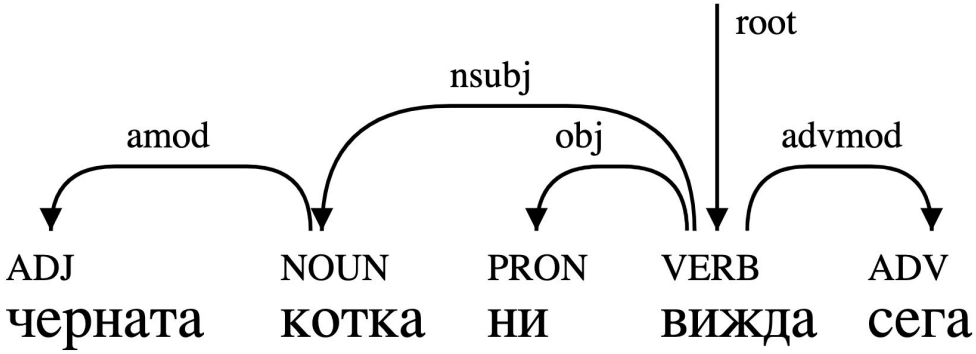
the	the	DET
black	black	ADJ
cat	cat	NOUN
sees	see	VERB
us	we	PRON
now	now	ADV

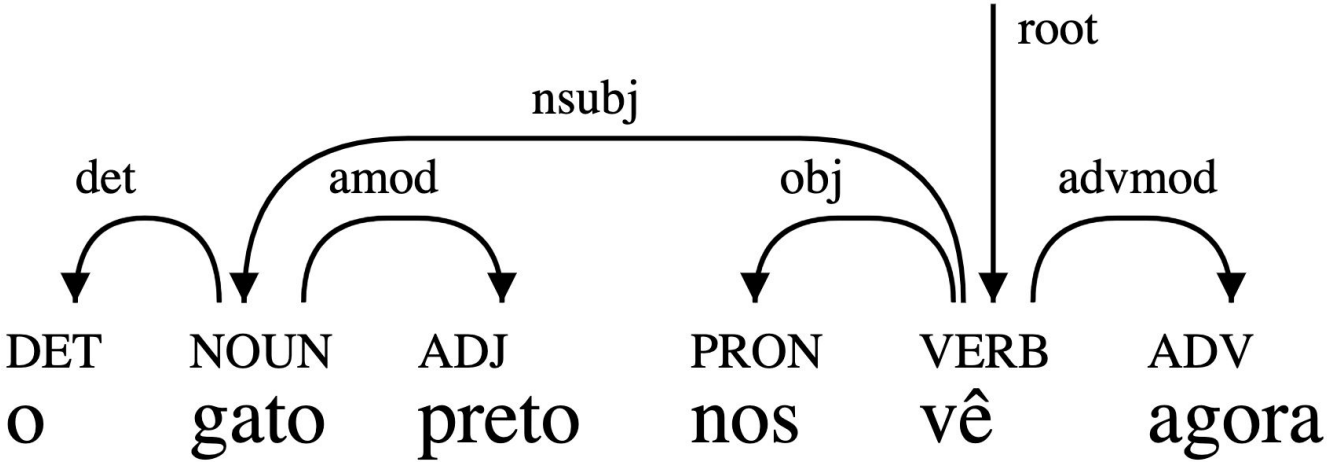
the	the	DET	_
black	black	ADJ	Posit
cat	cat	NOUN	Sg
sees	see	VERB	P3
us	we	PRON	Acc
now	now	ADV	_

1	the	the	DET	_	3
2	black	black	ADJ	Posit	3
3	cat	cat	NOUN	Sg	4
4	sees	see	VERB	P3	0
5	us	we	PRON	Acc	4
6	now	now	ADV	_	4

1	the	the	DET	_	3	det
2	black	black	ADJ	Posit	3	amod
3	cat	cat	NOUN	Sg	4	nsubj
4	sees	see	VERB	P3	0	root
5	us	we	PRON	Acc	4	obj
6	now	now	ADV	_	4	advmod


root
det
advmod
amod
nsubj
obj
DET ADJ NOUN VERB PRON ADV
the black cat sees us now


root
nsubj
amod
obj
advmod
ADJ
NOUN
PRON
VERB
ADV
черната
котка
ни
вижда
сега


root
nsubj
det
amod
obj
advmod
DET
NOUN
ADJ
PRON
VERB
ADV
o
gato
preto
nos
vê
agora

Manning's Law

1. UD needs to be satisfactory for analysis of individual languages.
2. UD needs to be good for linguistic typology.
3. UD must be suitable for rapid, consistent annotation.
4. UD must be suitable for computer parsing with high accuracy.
5. UD must be easily comprehended and used by a non-linguist.
6. UD must provide good support for downstream NLP tasks.

https://en.wikipedia.org/wiki/Manning%27s_Law

More about UD labels

grammarbook.pdf, Chapter 3

gf-ud

https://github.com/GrammaticalFramework/gf-ud

- analysing and converting dependency trees:
  - test things from README.md

- converting between UD and GF
  - after that

gf2ud

In action:

- `gfud gf2ud <path> <lang> <startcat>`
- gf shell, command `visualize_dependencies`
- minibar, click at language-specific syntax tree icon

The algorithm:

- after that

Applications:

- visualization
- treebank synthesis
- treebank augmentation

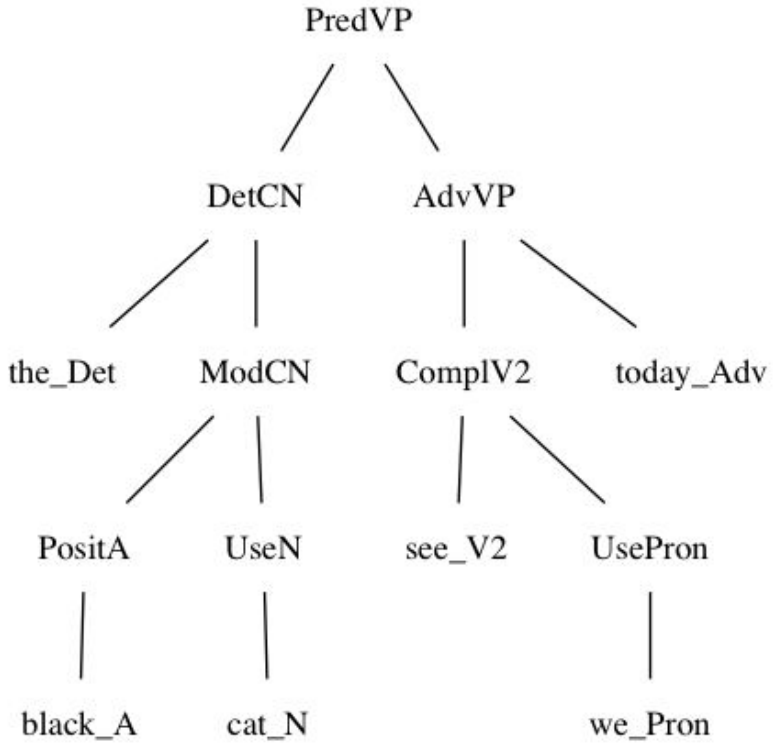
abstract syntax

```
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
```

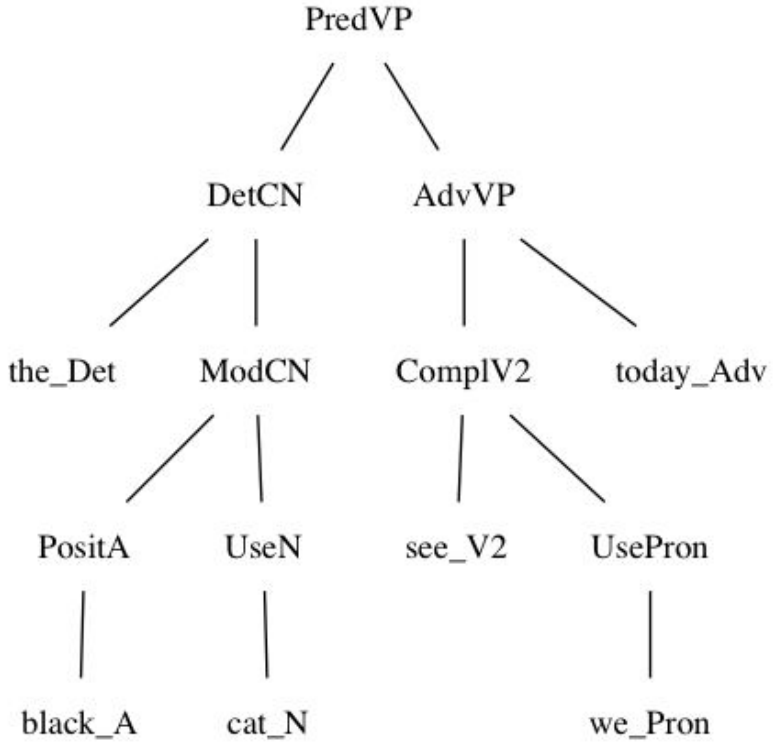
abstract syntax

```
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
```

the black cat sees us today



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`



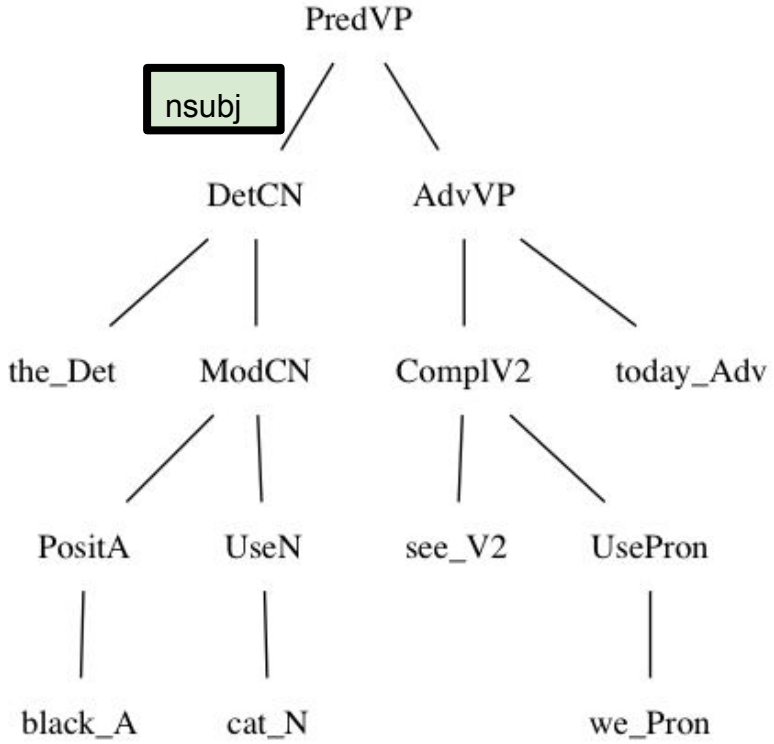
Kolachina & Ranta, From Abstract Syntax to Universal Dependencies, LiLT 2016.

abstract syntax

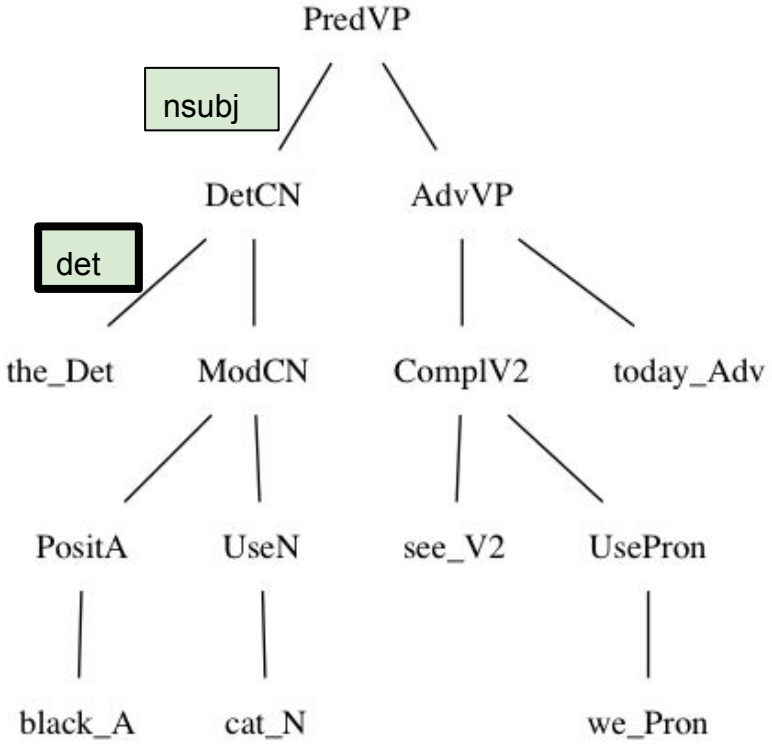
```
PredVP   : NP   -> VP   -> Cl
ComplV2  : V2   -> NP   -> VP
AdvVP    : VP   -> Adv  -> VP
DetCN    : Det  -> CN   -> NP
ModCN    : AP   -> CN   -> CN
UseN     : N    -> CN
UsePron  : Pron -> NP
PositA   : A    -> AP
```

dependency configuration

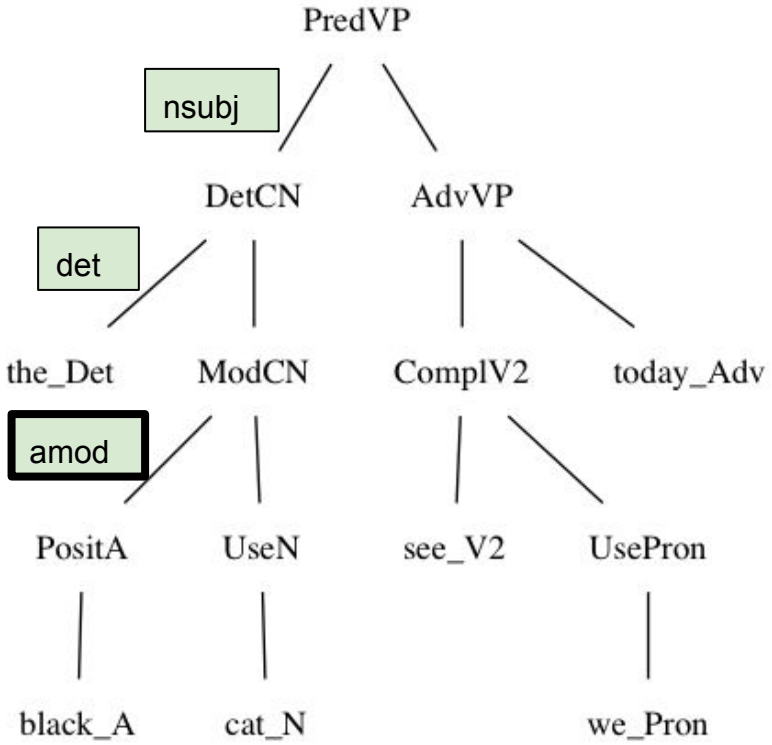
```
nsubj  head
head   dobj
head   advmod
det    head
amod   head
head
head
head
```



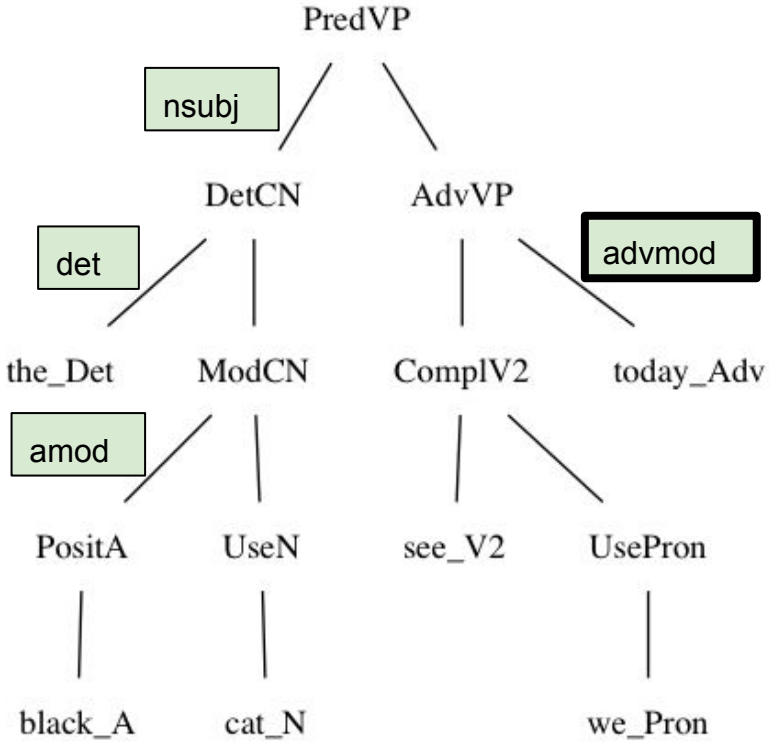
abstract syntax	dependency configuration
`PredVP : NP -> VP -> Cl`	`nsubj head`
`ComplV2 : V2 -> NP -> VP`	`head dobj`
`AdvVP : VP -> Adv -> VP`	`head advmod`
`DetCN : Det -> CN -> NP`	`det head`
`ModCN : AP -> CN -> CN`	`amod head`
`UseN : N -> CN`	`head`
`UsePron : Pron -> NP`	`head`
`PositA : A ->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

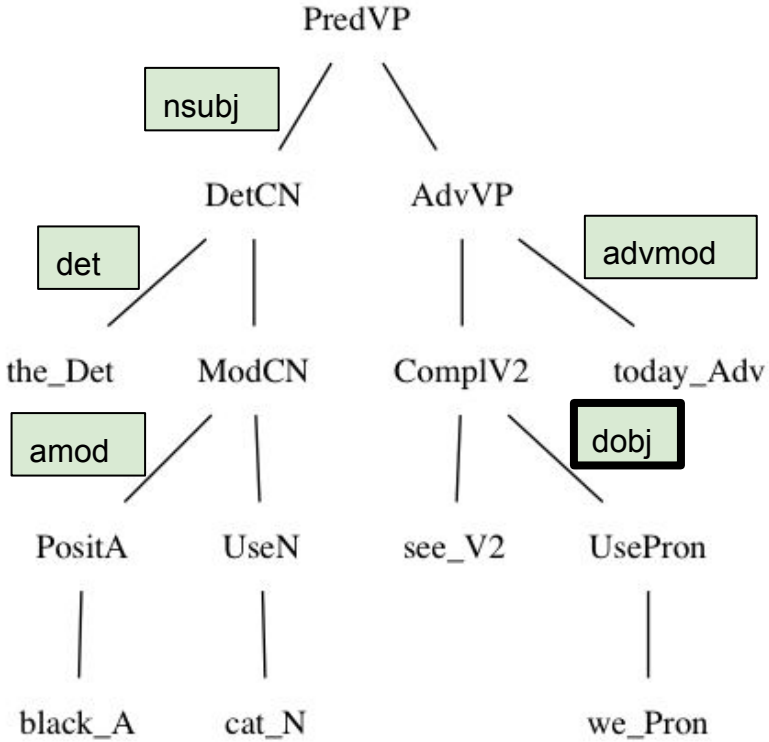


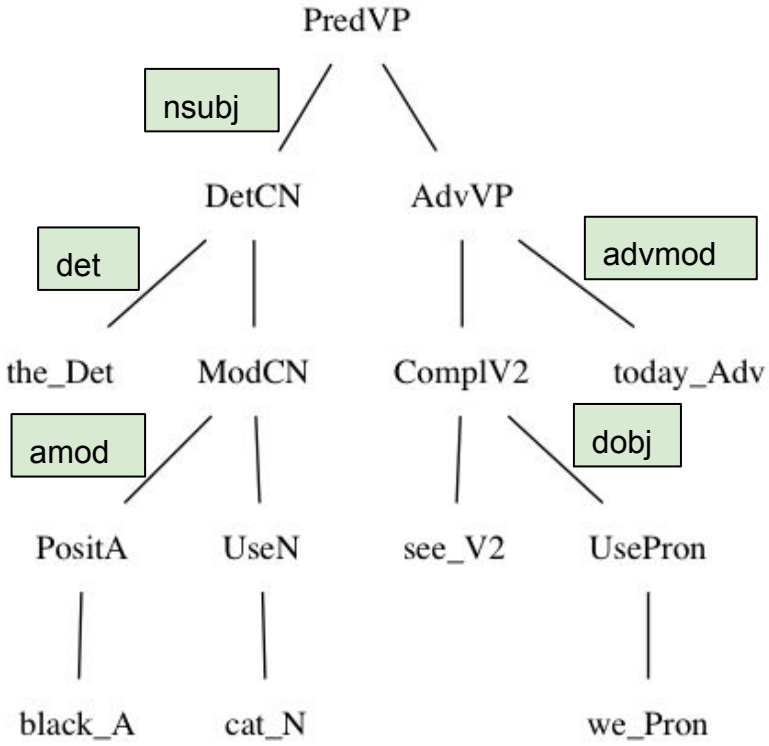
abstract syntax

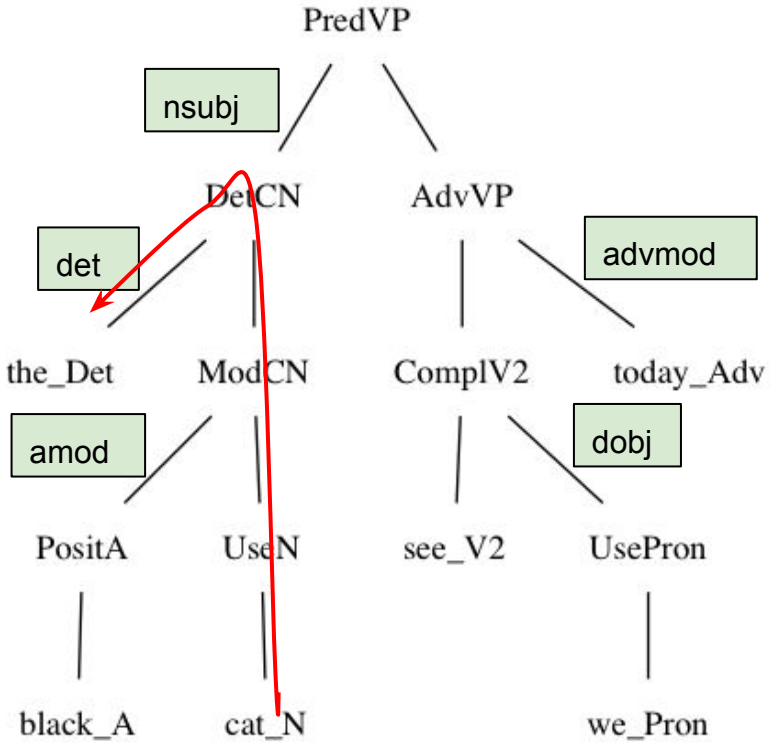
```
PredVP   : NP   -> VP   -> Cl
ComplV2  : V2   -> NP   -> VP
AdvVP    : VP   -> Adv  -> VP
DetCN    : Det  -> CN   -> NP
ModCN    : AP   -> CN   -> CN
UseN     : N    -> CN
UsePron  : Pron -> NP
PositA   : A    -> AP
```

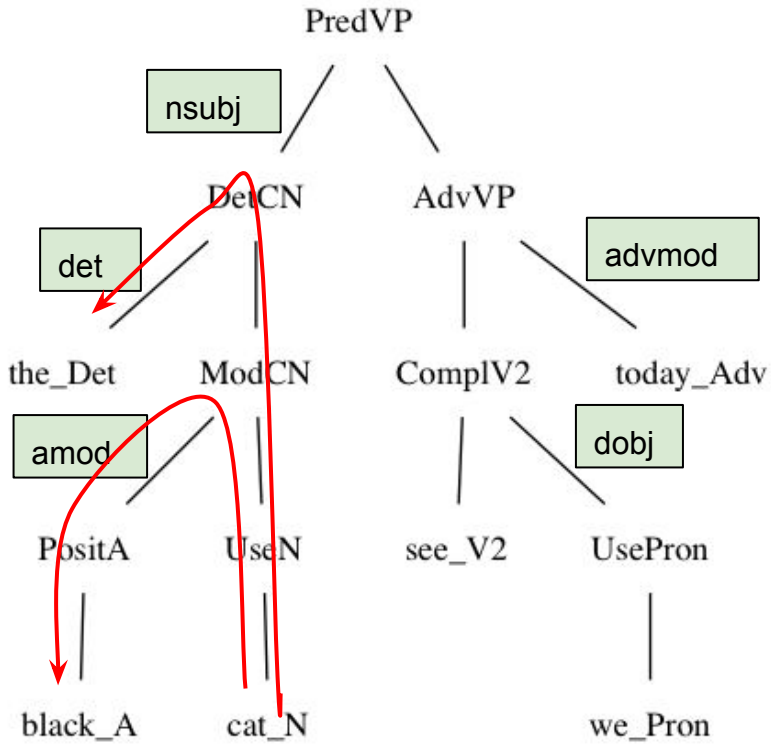
dependency configuration

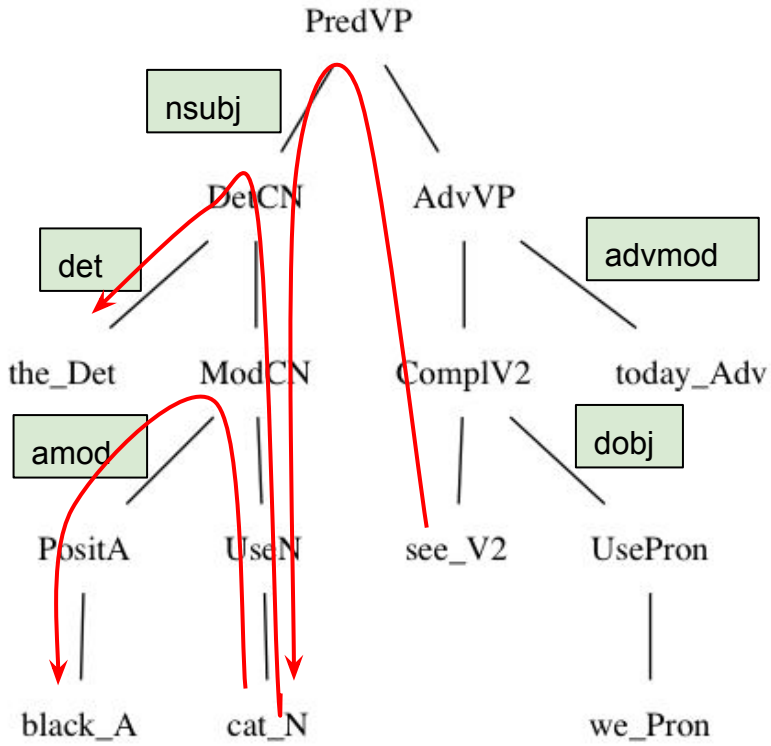
```
nsubj   head
head    dobj
head    advmod
det     head
amod    head
head
head
head
```

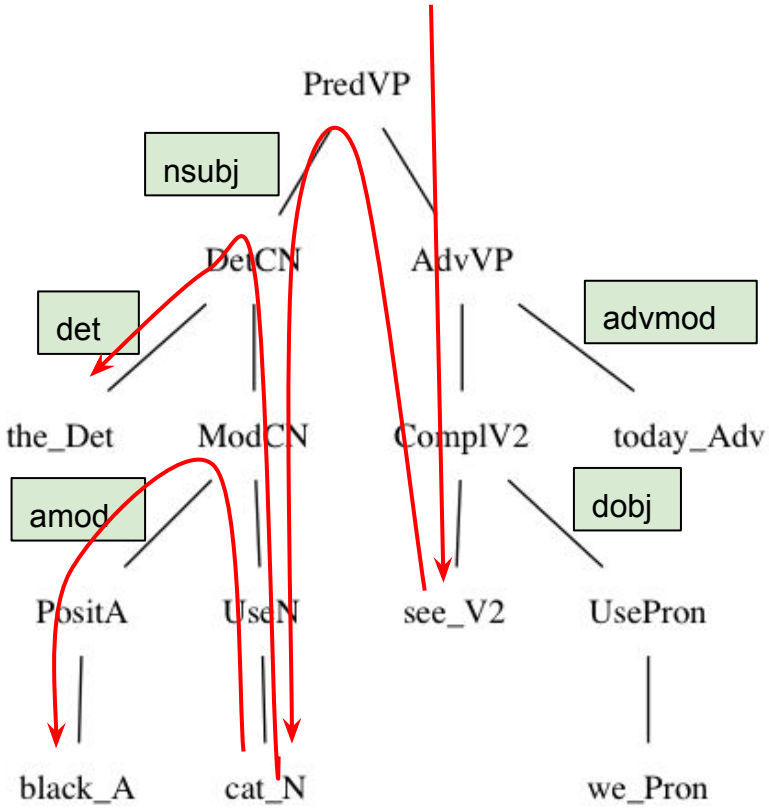


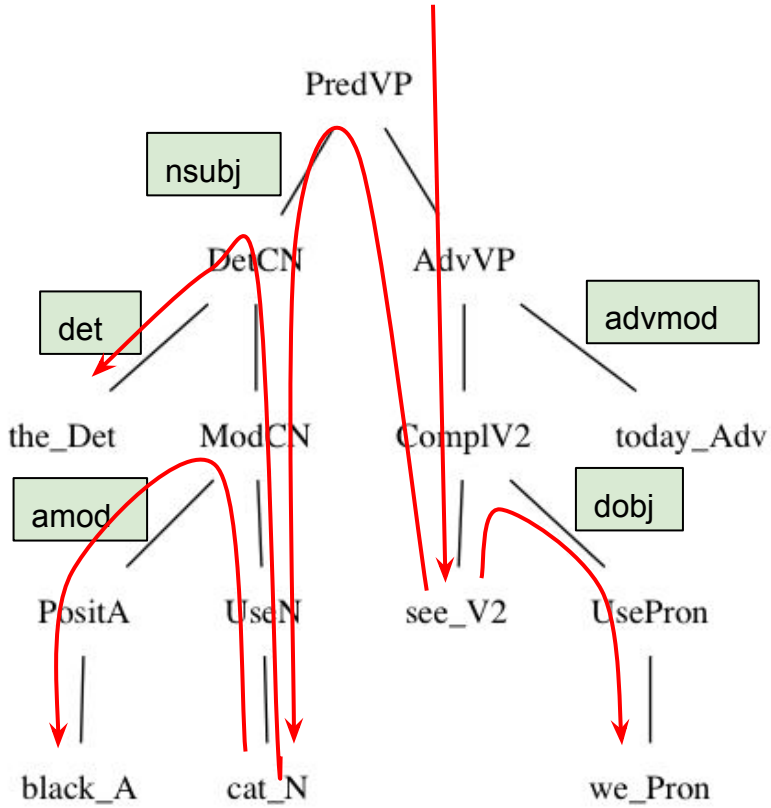

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

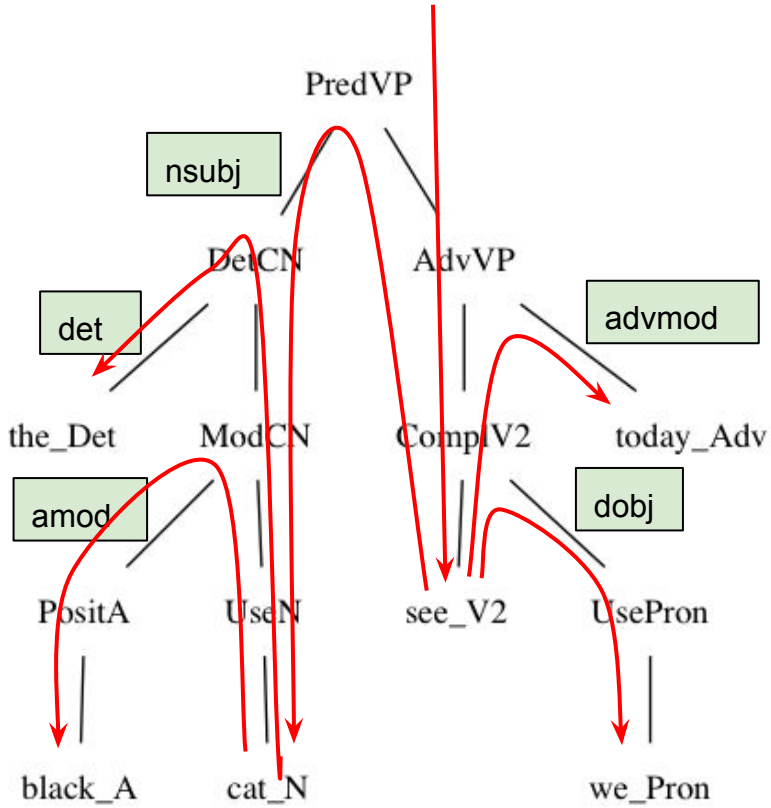

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

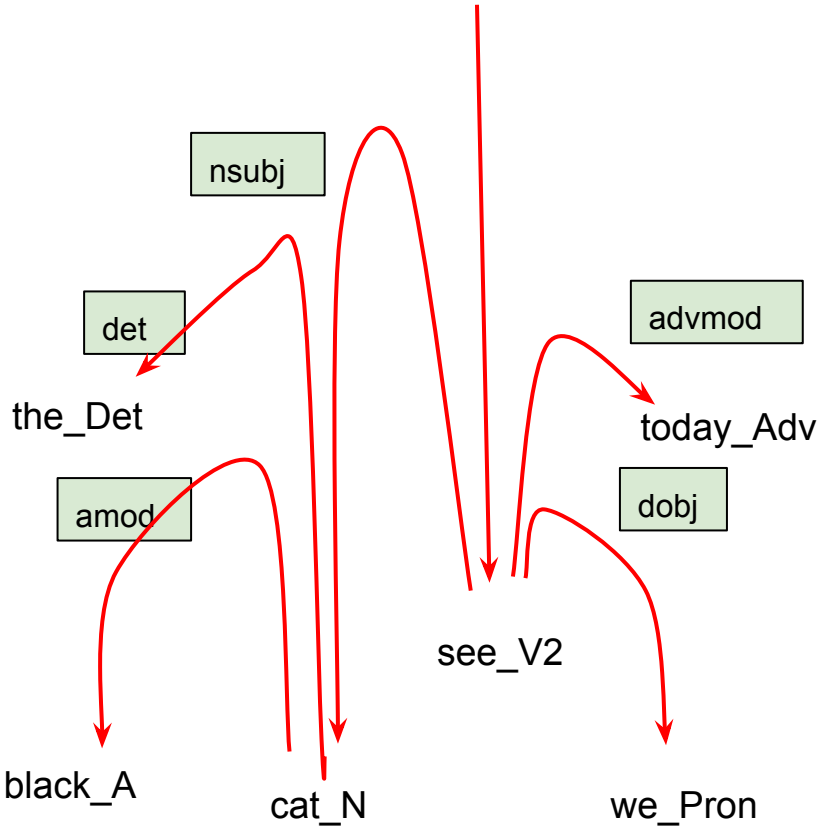

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

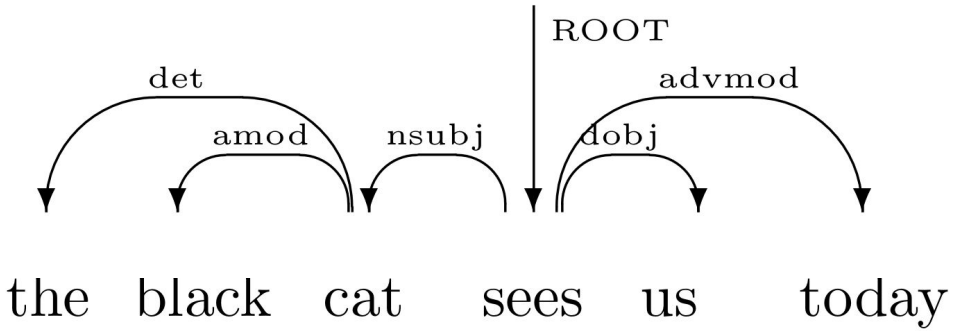

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

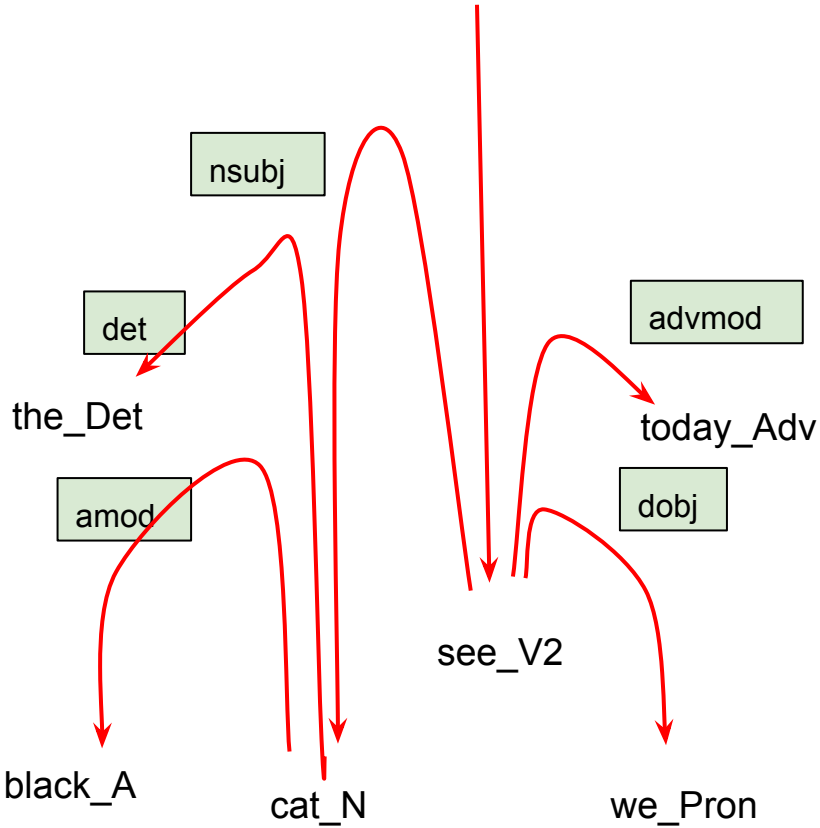

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
ComplV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron


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

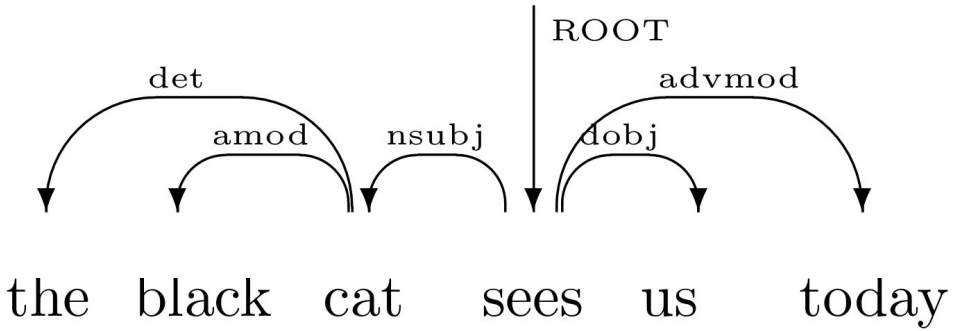

PredVP
nsubj
DetCN
AdvVP
det
advmod
the_Det
ModCN
CompV2
today_Adv
amod
dobj
PositA
UseN
see_V2
UsePron
black_A
cat_N
we_Pron

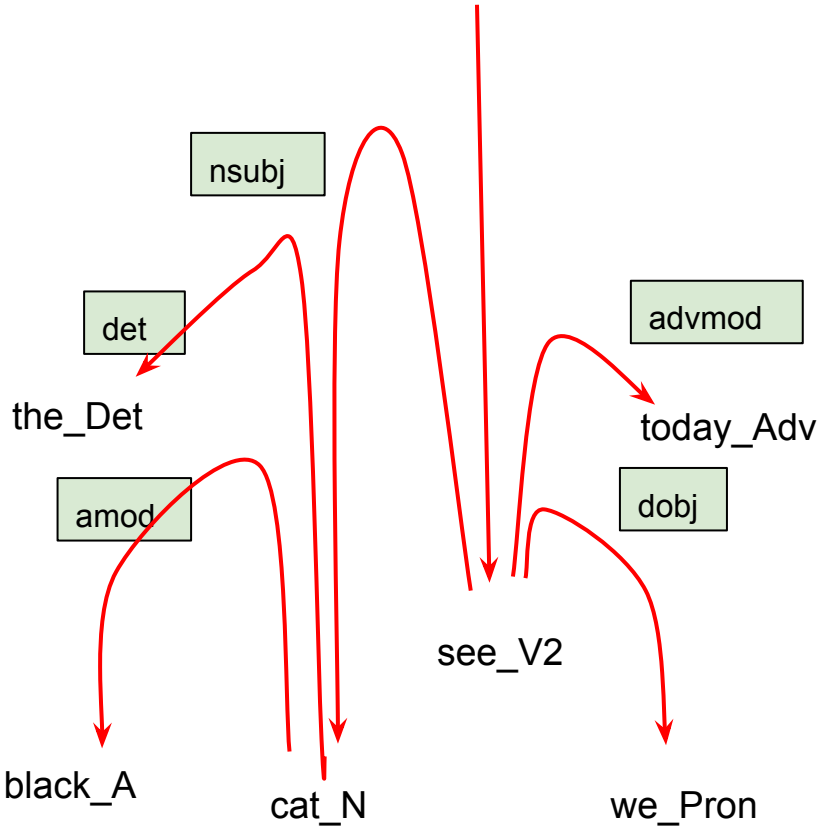

nsubj
det
the_Det
advmod
today_Adv
amod
dobj
see_V2
black_A
cat_N
we_Pron


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


nsubj
det
advmod
the_Det
today_Adv
amod
dobj
see_V2
black_A
cat_N
we_Pron

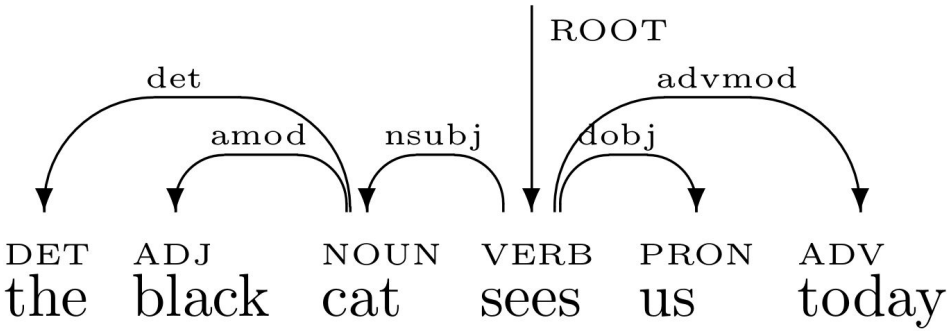
```
abstract syntax category configuration

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

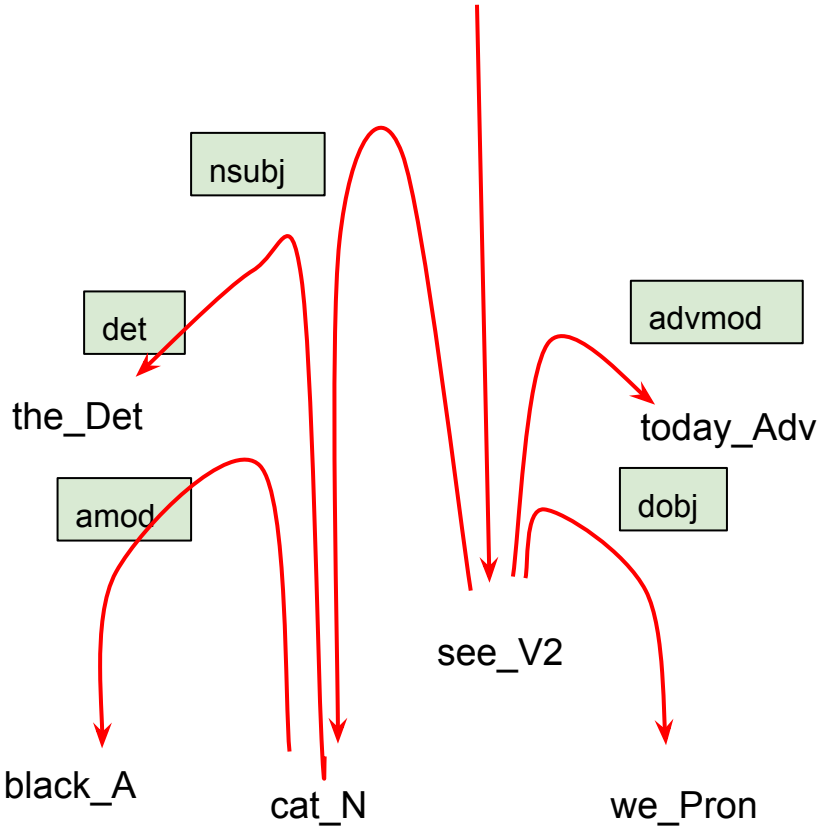


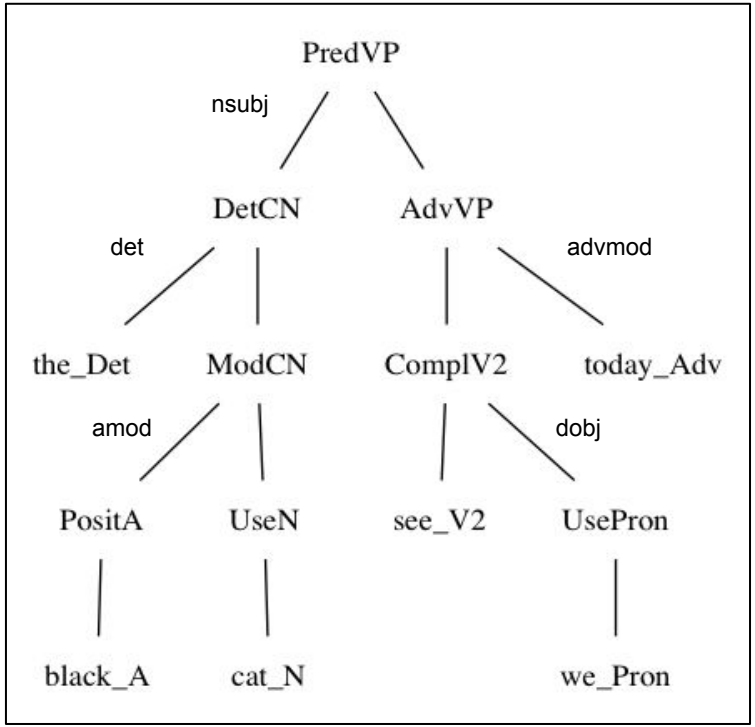


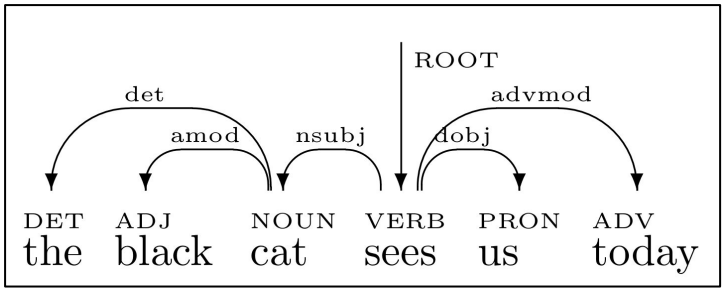
abstract syntax category configuration

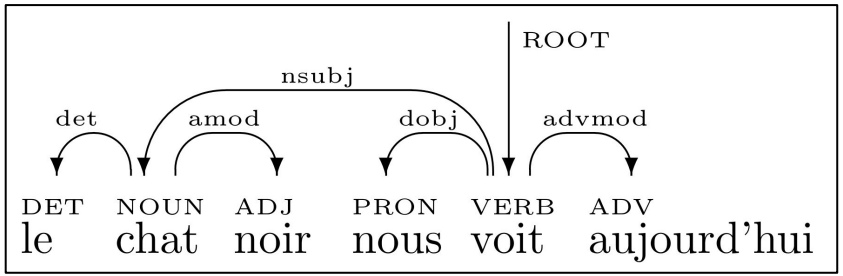
Det	DET
A	ADJ
N	NOUN
V2	VERB
Pron	PRON
Adv	ADV






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


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


ROOT
nsubj
det
amod
dobj
advmod
DET NOUN ADJ PRON VERB ADV
le chat noir nous voit aujourd'hui

Trees: summary

abstract syntax tree: language-independent

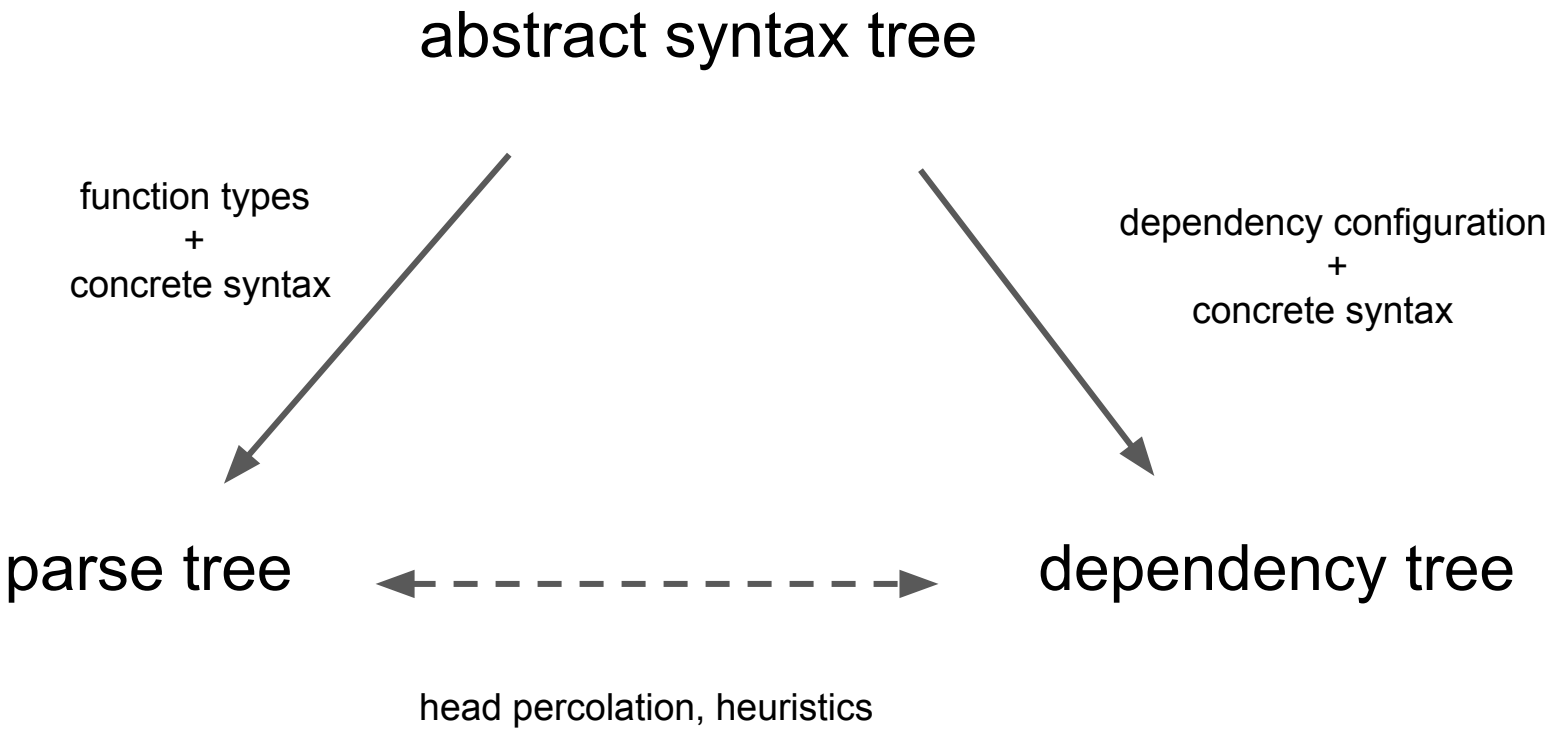
- built from functions
- lossless representation: functions determine categories and dependencies

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”
```

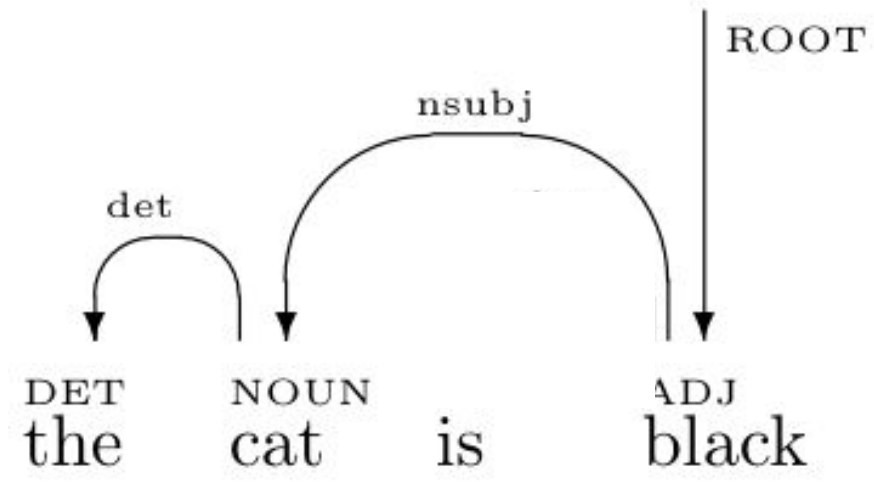
categorematic: word with its own category and function

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

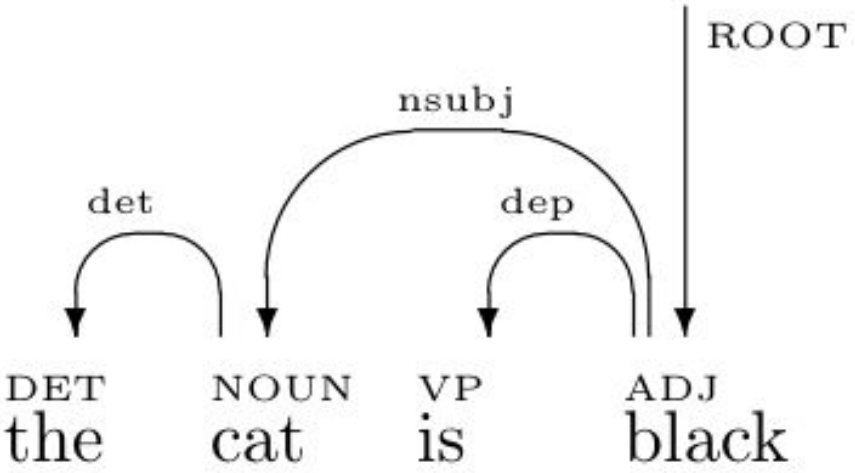
syncategorematic: word that is “between categories”

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

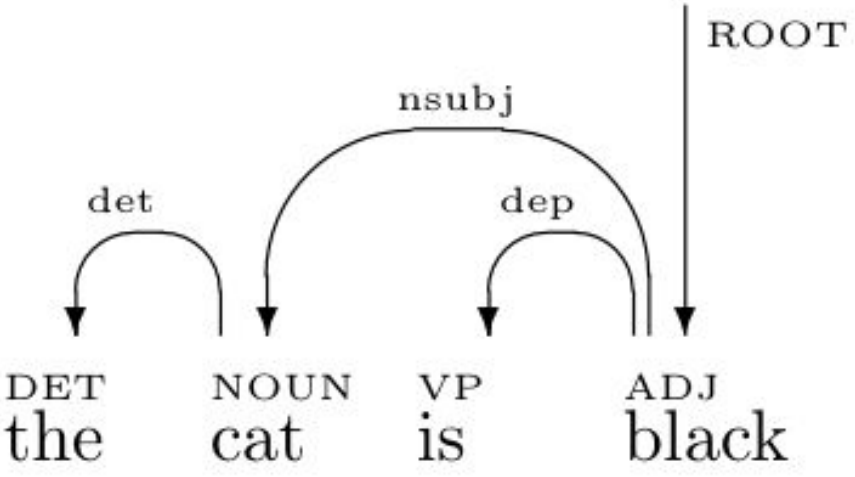
No semantics (`fun`) of its own. Not an argument. No label.


ROOT
nsubj
det
DET
NOUN
ADJ
the
cat
is
black

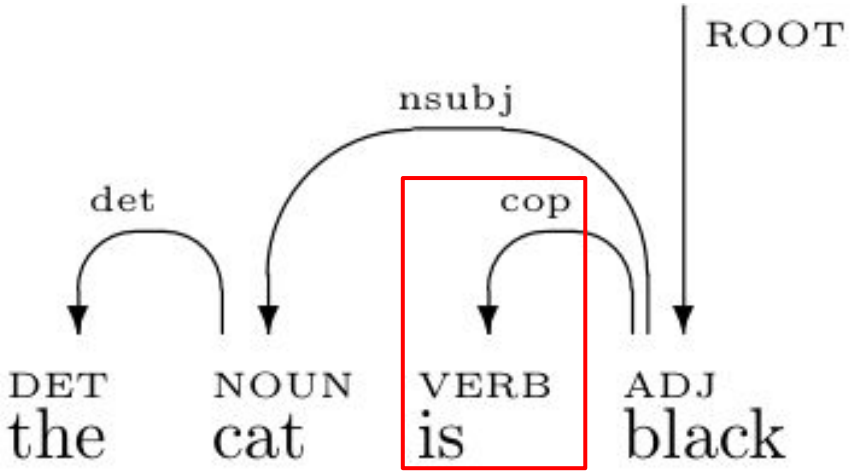
adding default labels



we get



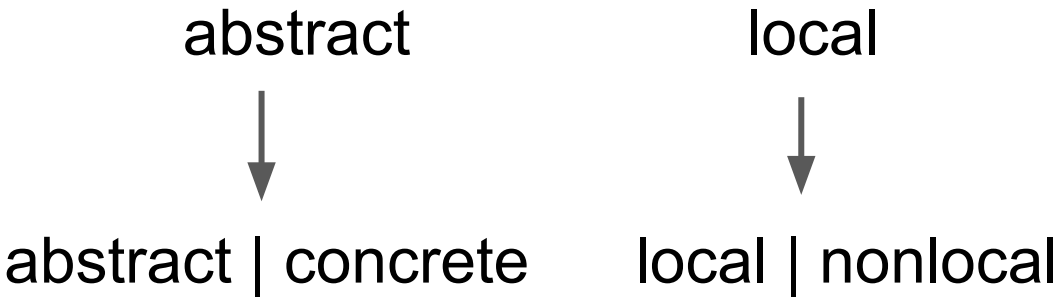
UD wants



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



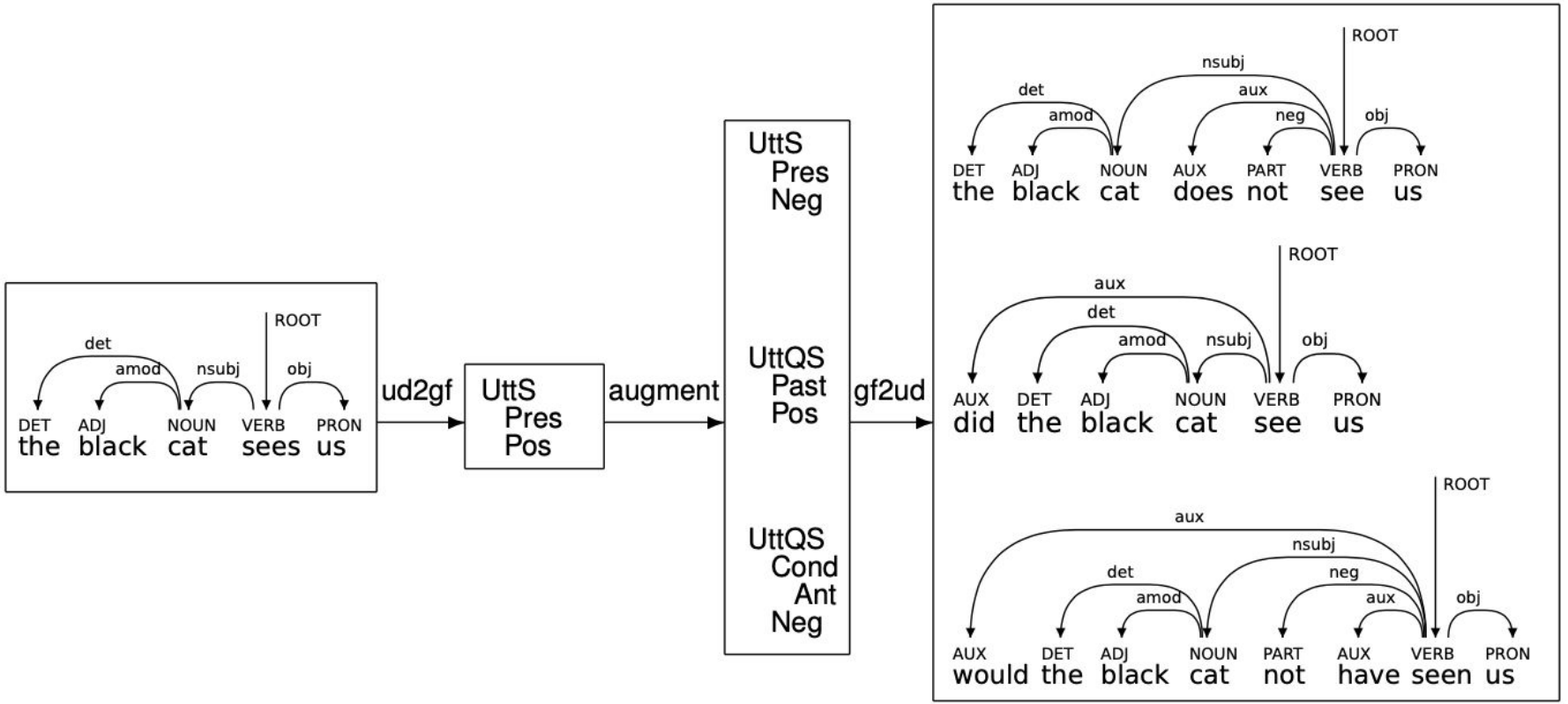
- more complicated, not universal
+ less work than rewriting the grammar anyway
+ more flexible (if UD should change…)
+ abstract syntax can be kept more abstract

Other syncategorematic words

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

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

Data augmentation



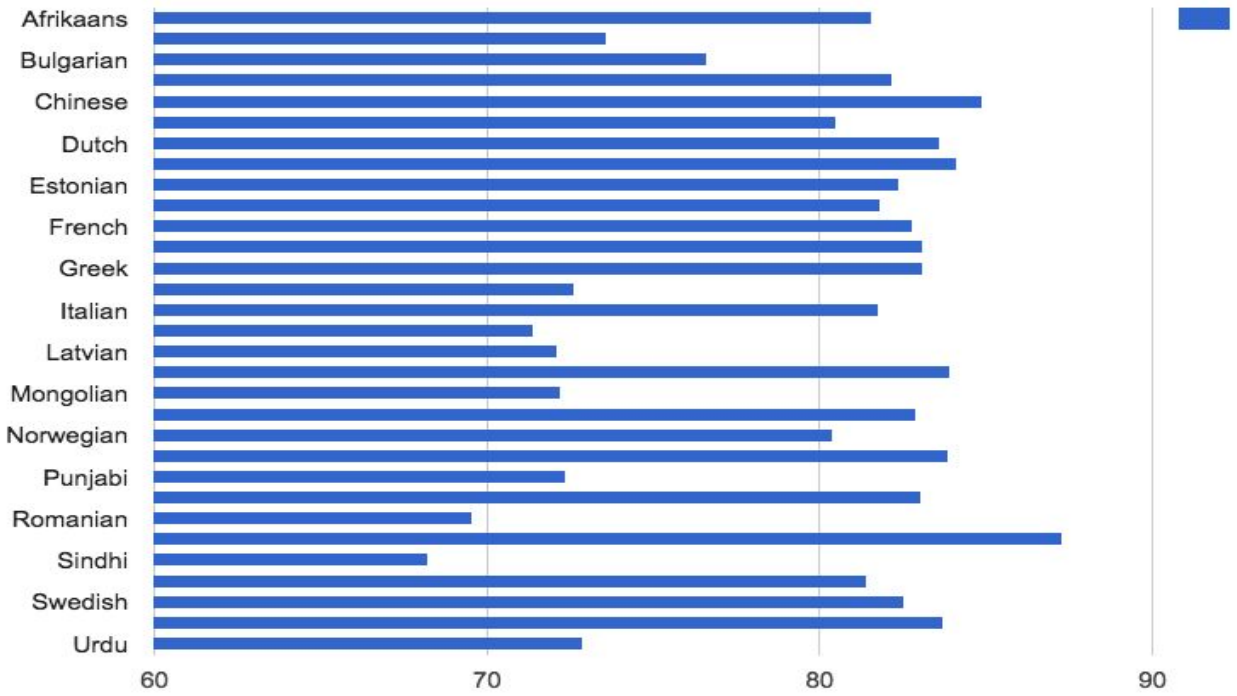
Cross-lingual bootstrapping

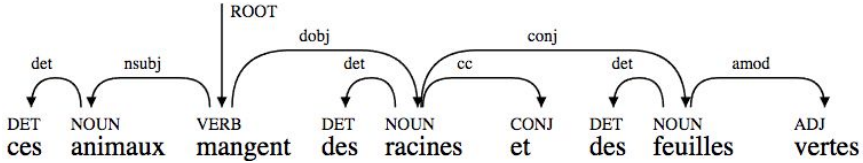
Given a treebank in GF, using the mappings defined on abstract syntax it is possible to bootstrap dependency tree for new languages

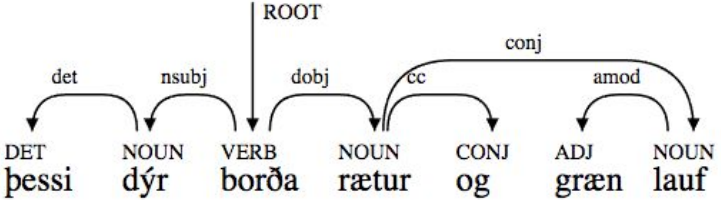
Experiments with bootstrapping to all 36 languages in the RGL

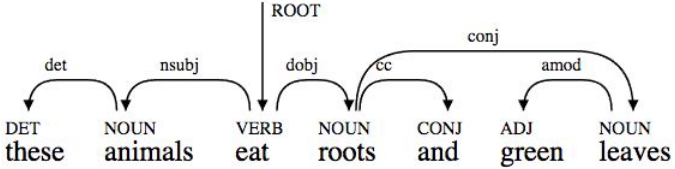
- a GF treebank from the examples presented in UD annotation manuals
- about 80% of edges in the treebank can be labelled using the abstract rules

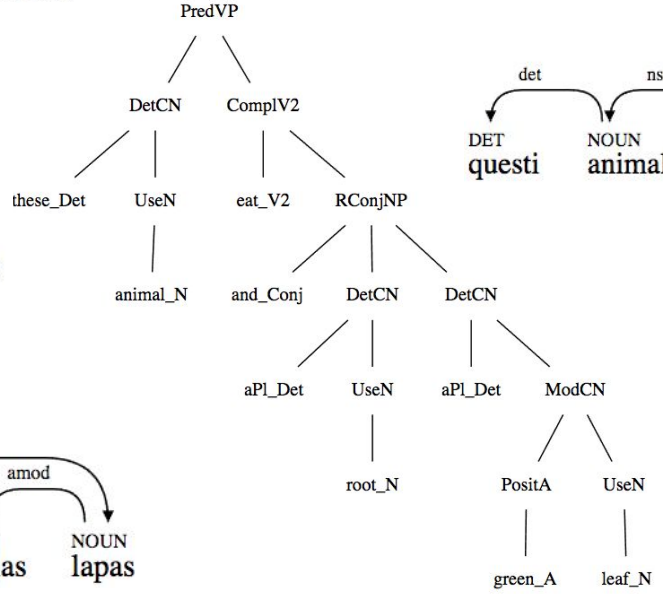
Worst case: 70-75% (Japanese, Latvian, Punjabi)

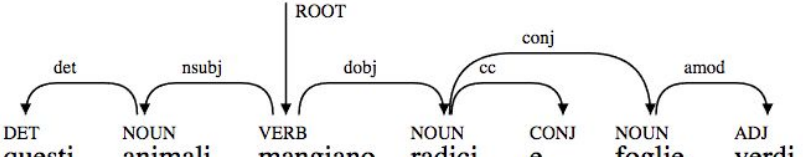

Afrikaans
Bulgarian
Chinese
Dutch
Estonian
French
Greek
Italian
Latvian
Mongolian
Norwegian
Punjabi
Romanian
Sindhi
Swedish
Urdu
60
70
80
90

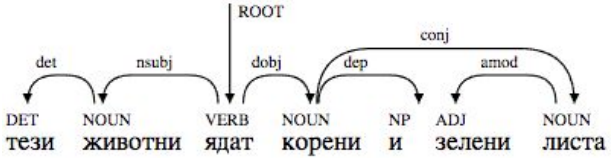

ROOT
det nsubj dobj det cc conj det amod
DET NOUN VERB DET NOUN CONJ DET NOUN ADJ
ces animaux mangent des racines et des feuilles vertes

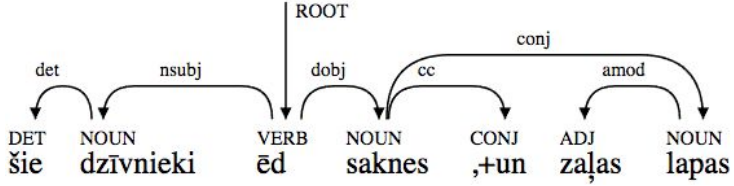

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
þessi dýr borða rætur og græn lauf

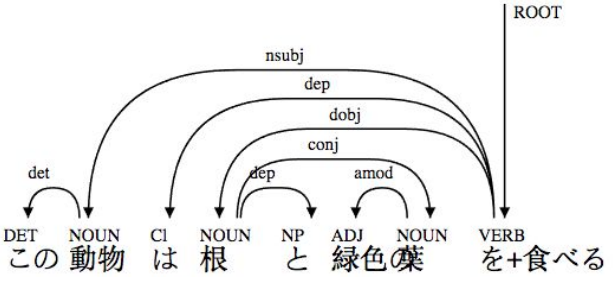

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
these animals eat roots and green leaves

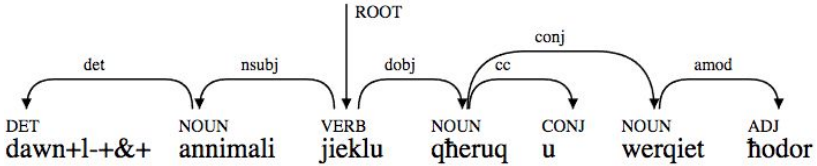

PredVP
DetCN ComplV2
these_Det UseN eat_V2 RConjNP
animal_N and_Conj DetCN DetCN
aPl_Det UseN aPl_Det ModCN
root_N PositA UseN
green_A leaf_N

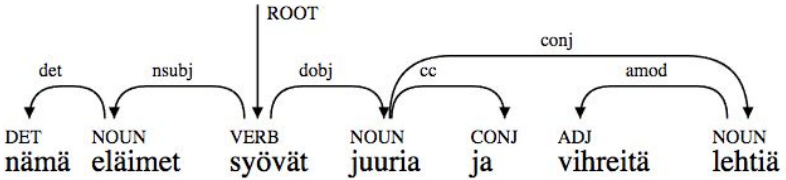

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ NOUN ADJ
questi animali mangiano radici e foglie verdi

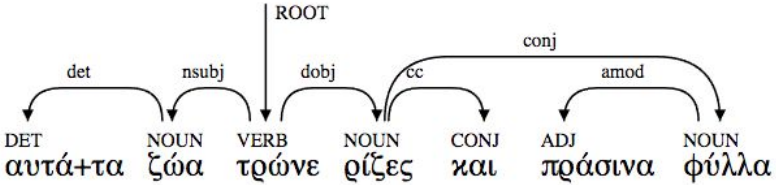

ROOT
det nsubj dobj dep conj amod
DET NOUN VERB NOUN NP ADJ NOUN
тези животни ядат корени и зелени листа

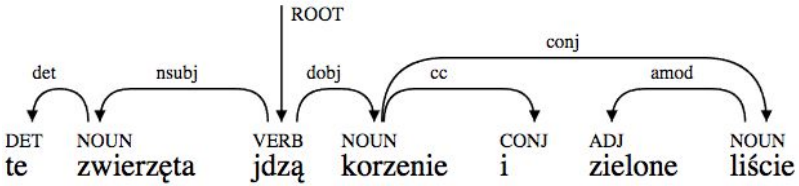

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
šie dzīvnieki ēd saknes ,+un zaļas lapas

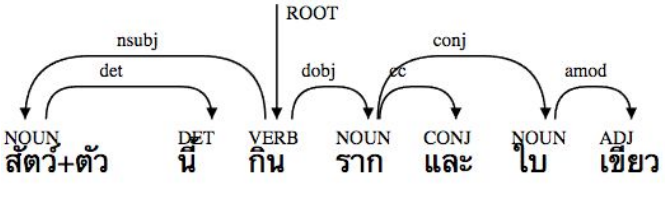

ROOT
nsubj dep dobj conj dep amod
det
DET NOUN Cl NOUN NP ADJ NOUN VERB
この 動物 は 根 と 緑色の葉 を+食べる

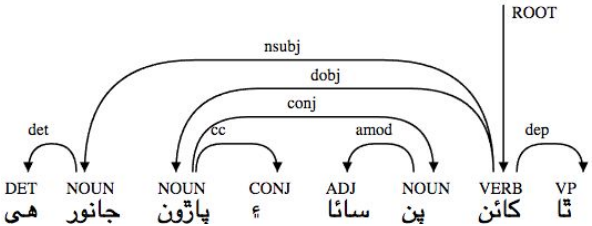

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ NOUN ADJ
dawn+l-+&+ annimali jieklu qħeruq u werqiet ħodor

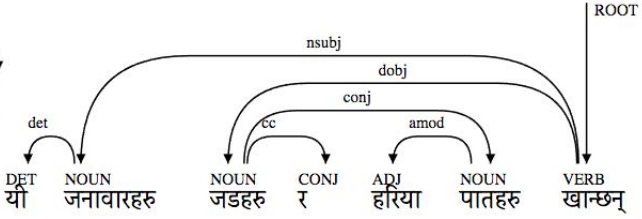

ROOT
det nsubj dobj cc conj amod
DET NOUN VERB NOUN CONJ ADJ NOUN
nämä eläimet syövät juuria ja vihreitä lehtiä


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 cc amod dep
det NOUN NOUN CONJ ADJ NOUN VERB VP
ھی جانور پاڑون ۽ سائا پن کائن ٹا


ROOT
nsubj dobj conj 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

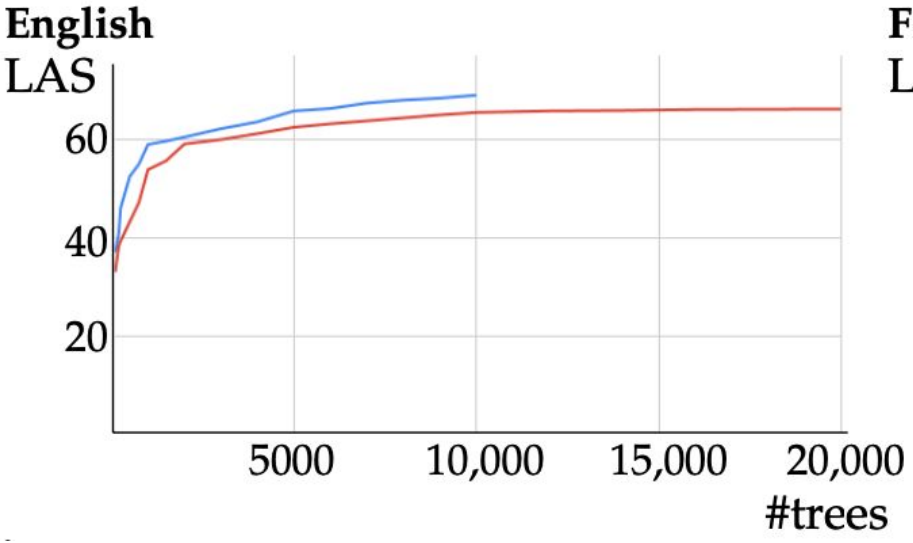
About 5% of the nodes in this treebank are missing/incomplete

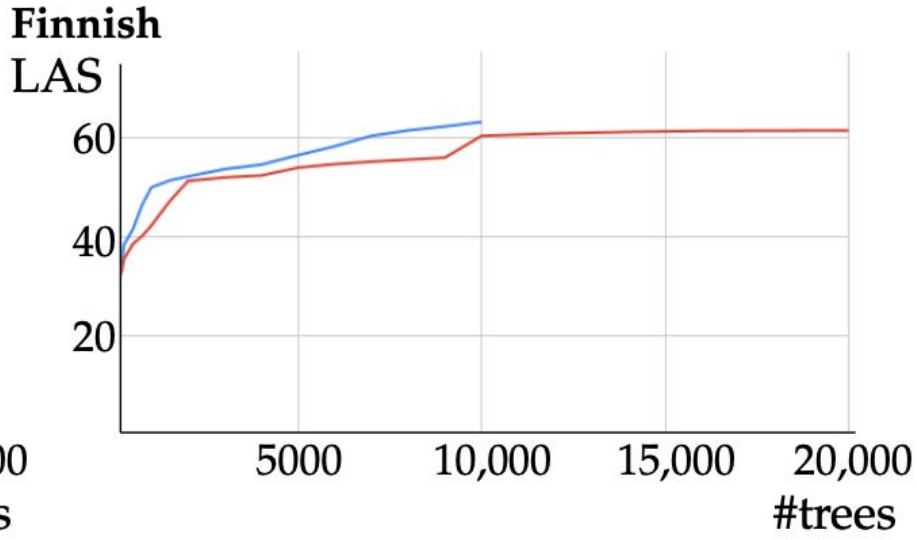
- almost 10% of nodes are assigned the default `dep' label

UD version of the PTB

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

Some treebank synthesis results





A hand-on synthesis experiment

```
# generate training corpus from random GF trees
  echo 'gr -number=500 -depth=6' | gf -run grammars/MiniLangEng.gf | sort -u >mini-trees.gft
# convert to UD trees and inspect
  cat mini-trees.gft | gfud gf2ud grammars/MiniLang Eng Utt ud >mini-trees.conllu
  gfud conll2pdf <mini-trees.conllu
# train a parser with udpipe - takes a long time
  cat mini-trees.conllu | udpipe --train mini-trees.udpipe
# parse input, inspect, convert back to GF
  echo "this is very good" | udpipe --tokenize --tag --parse mini-trees.udpipe | gfud conll2pdf
  echo "this is very good" | udpipe --tokenize --tag --parse mini-trees.udpipe | gfud ud2gf
grammars/MiniLang Eng Utt
  echo "the wolf is very ill" | udpipe --tokenize --tag --parse mini-trees.udpipe | gfud conll2pdf
# generate test corpus
  echo 'gr -number=200 -depth=6' | gf -run grammars/MiniLangEng.gf | sort -u >mini-test.gft
  cat mini-test.gft | gfud gf2ud grammars/MiniLang Eng Utt ud >mini-test.conllu
# parse the test corpus with udpipe
  cat mini-test.conllu | udpipe --parse wordnet.udpipe >out/mini-test-udpipe.conllu
# evaluate the results
  gfud eval macro LAS mini-test.conllu out/mini-test-udpipe.conllu
  gfud eval macro LAS mini-test.conllu out/mini-test-udpipe.conllu units
```

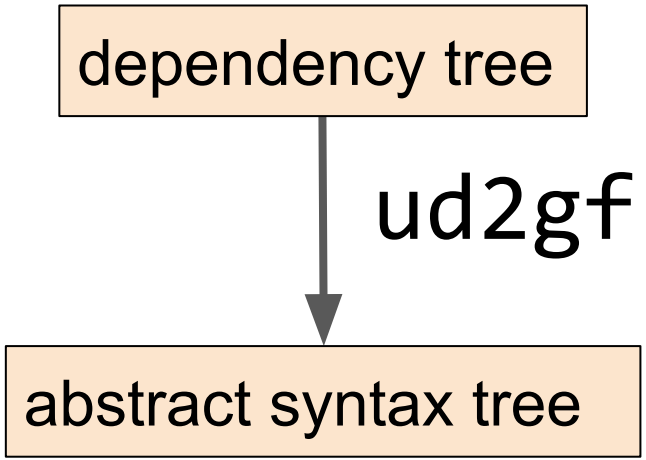
https://github.com/GrammaticalFramework/gf-ud/blob/master/doc/synthetic.md
https://ufal.mff.cuni.cz/udpipe

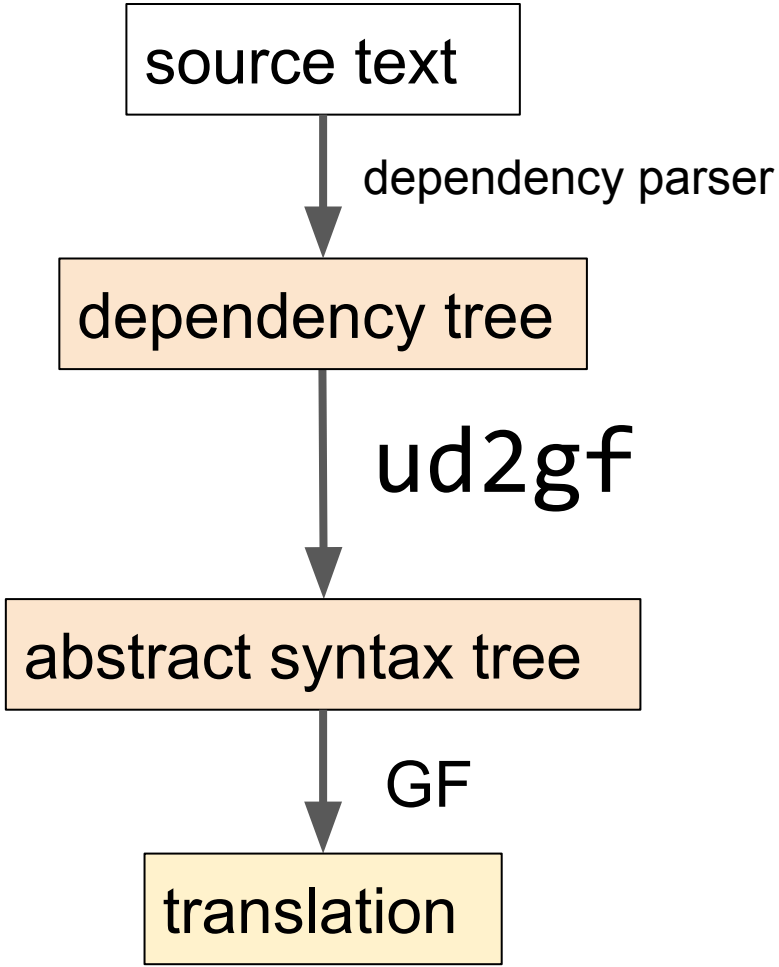
And the same with arithmetic expressions

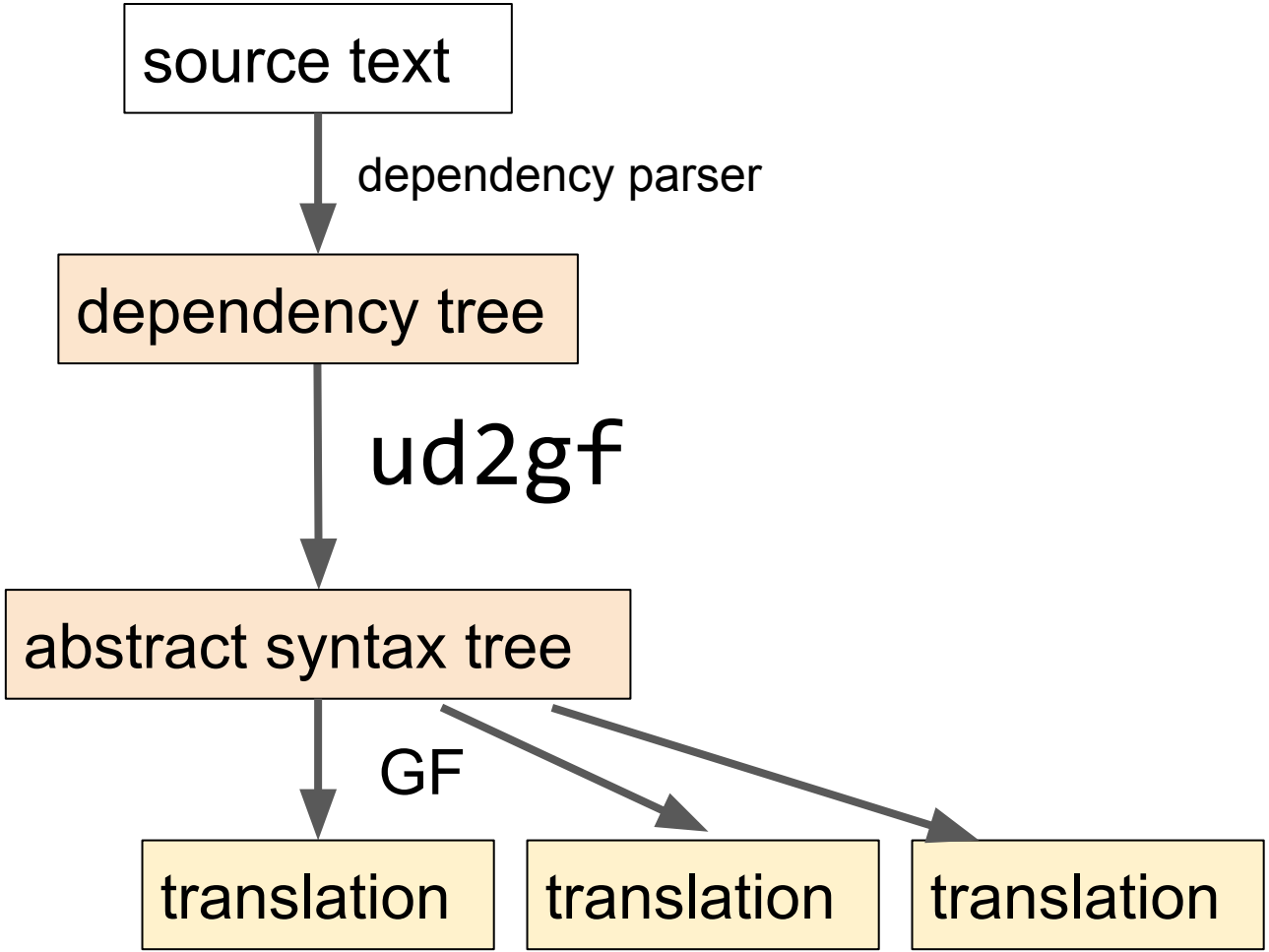
grammars/TermsInfix.gf

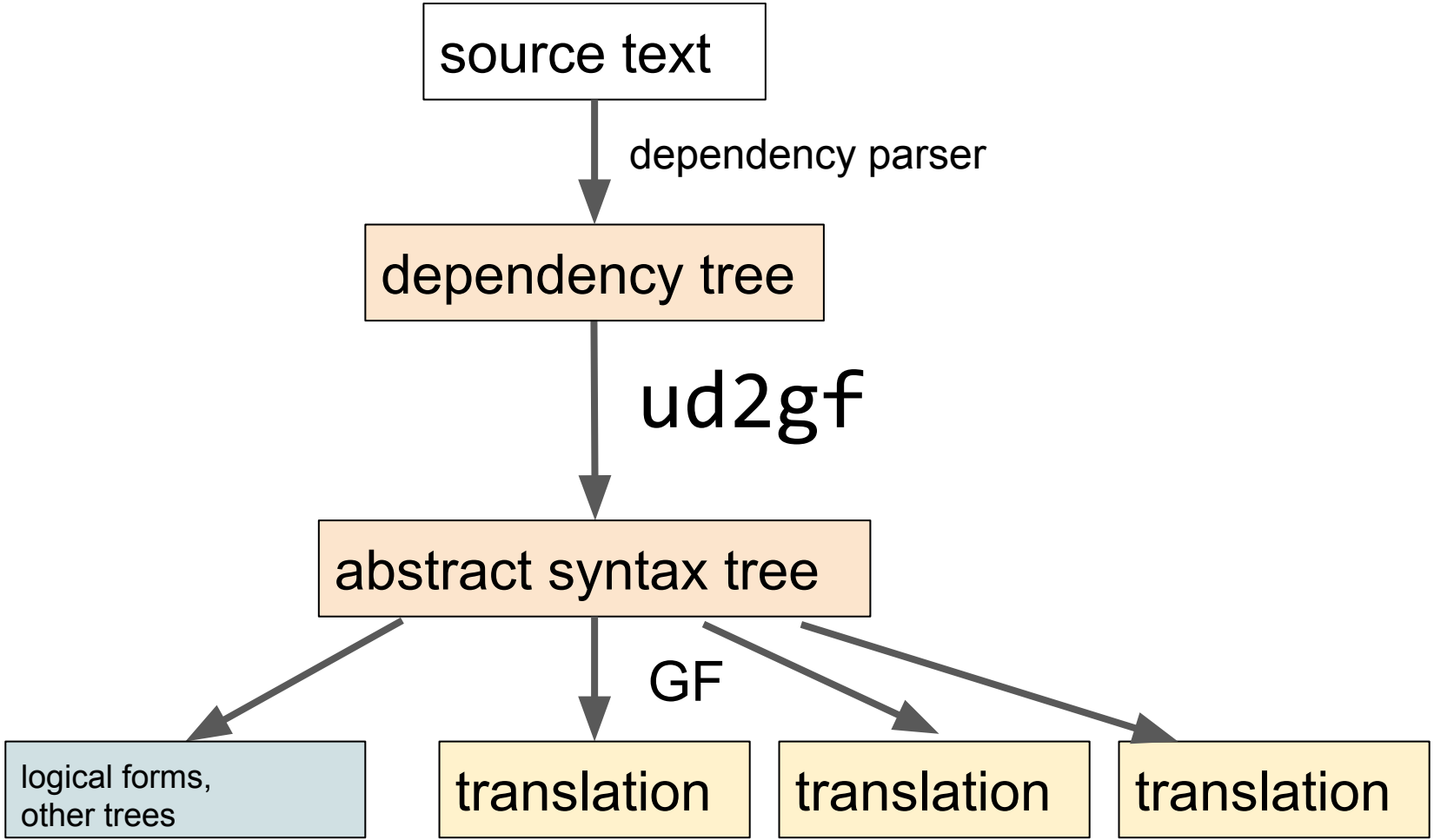
```
cat
  Term ; Factor ; Atom ; TOper ; FOper ;

fun
  OpTerm : TOper -> Term -> Factor -> Term ;   -- head arg1 arg2
  OpFactor : FOper -> Factor -> Atom -> Atom ; -- head arg1 arg2

  FactorTerm : Factor -> Term ;
  AtomFactor : Atom -> Factor ;

  parenth : Term -> Atom ;

  PlusOp : TOper ;
  MinusOp : TOper ;
  TimesOp : FOper ;

  x : Atom ;
```

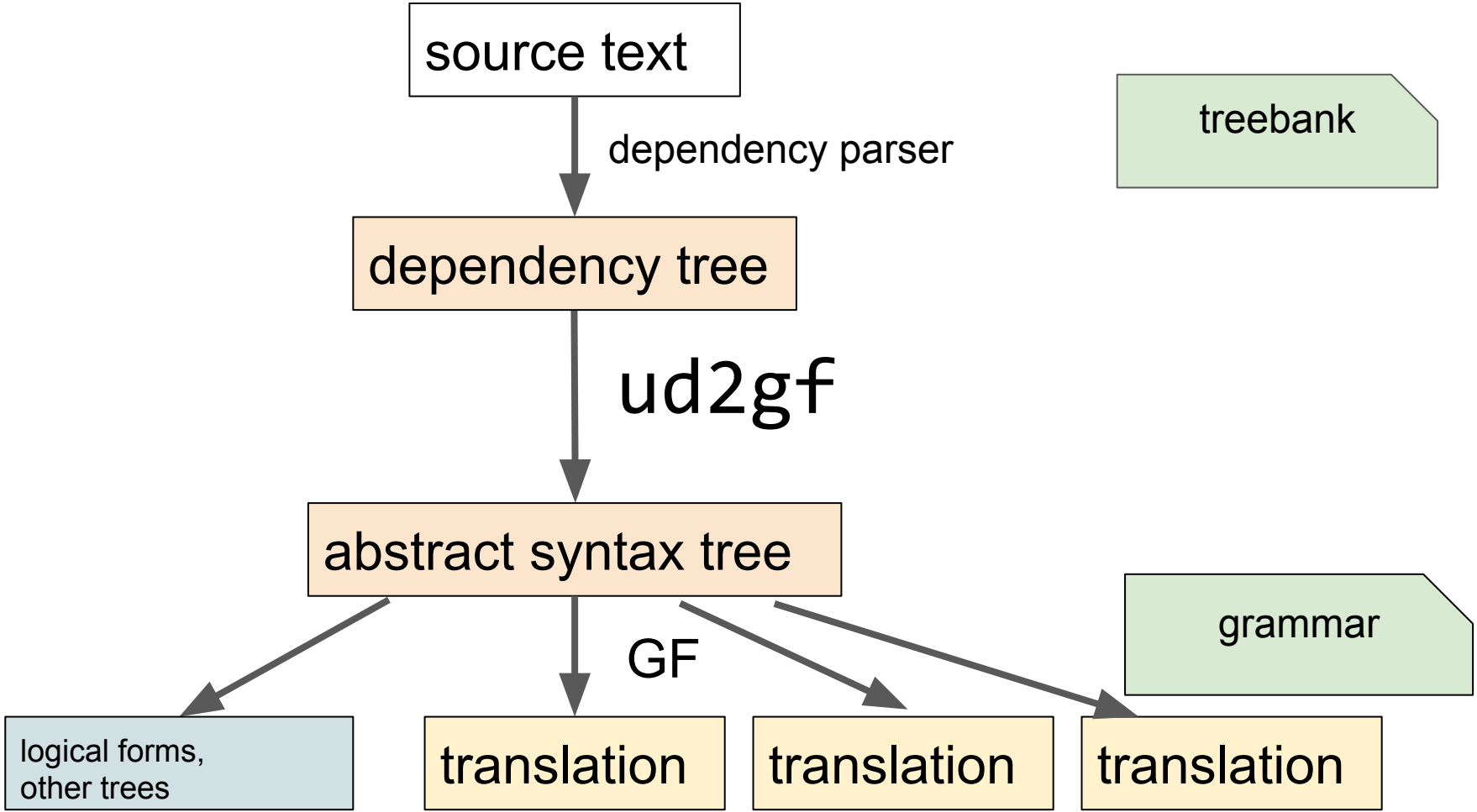
ud2gf

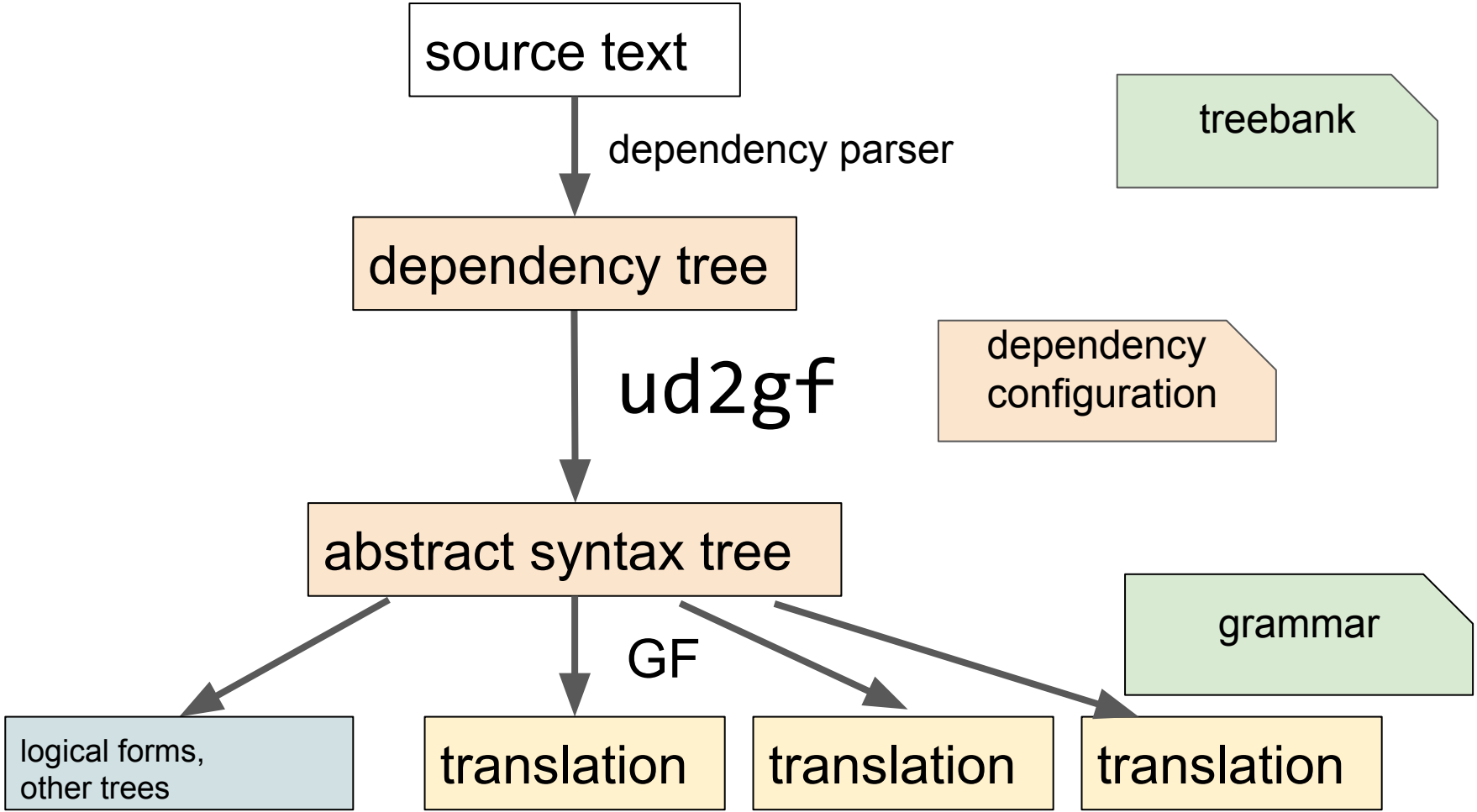

dependency tree
ud2gf
abstract syntax tree


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

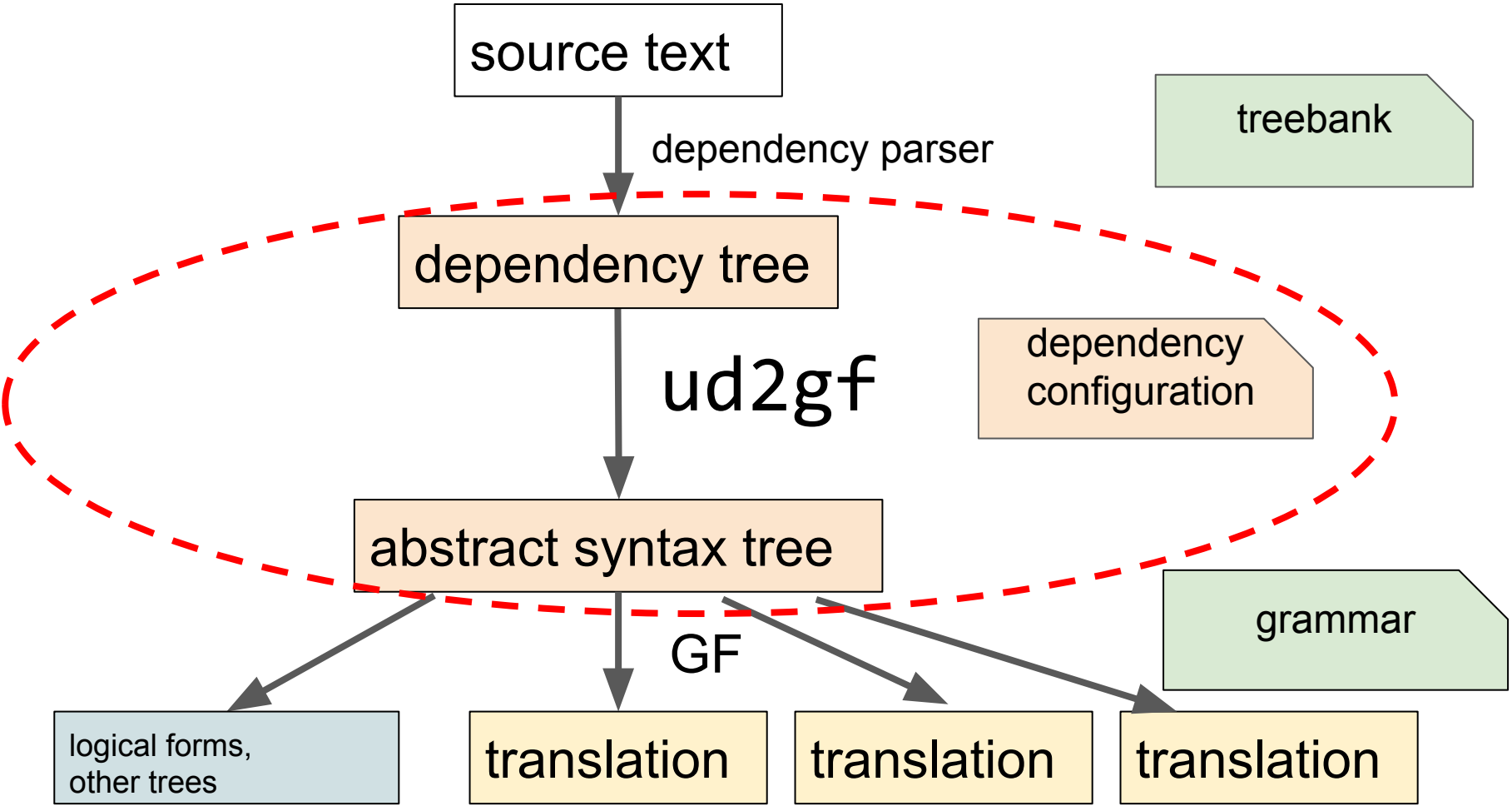

source text
dependency parser
dependency tree
ud2gf
abstract syntax tree
GF
logical forms,
other trees
translation
translation
translation


source text
dependency parser
treebank
dependency tree
ud2gf
abstract syntax tree
GF
grammar
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 algorithm

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

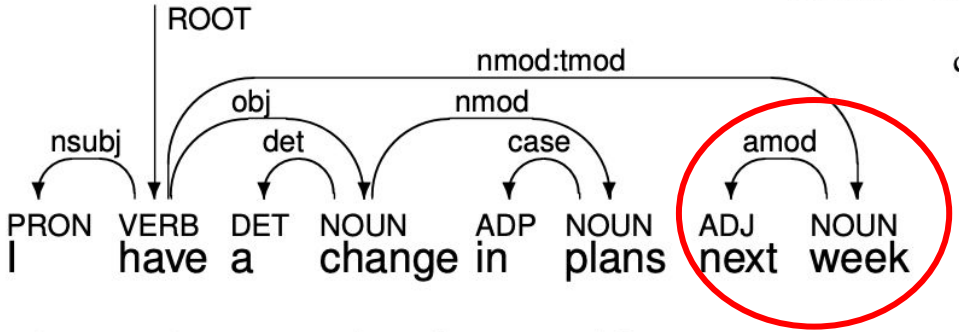
2. Replace each (*Lemma,POS*) with (*Function,Cat*) by lexicon lookup.

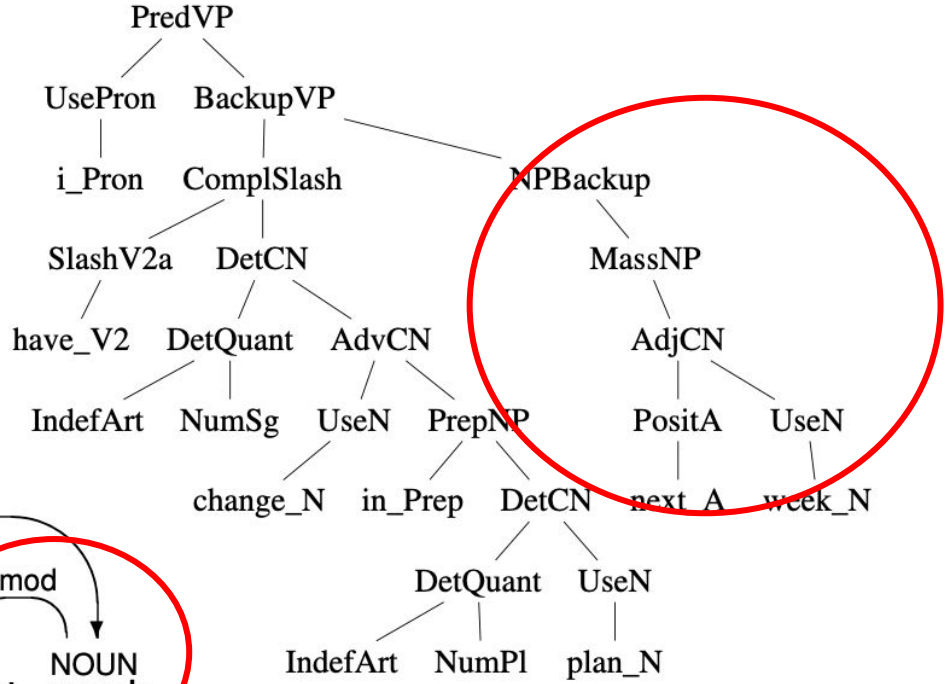
3. Recursively annotate each subtree (*Node* $Tree_1$ … $Tree_n$) as follows:
    - annotate each $Tree_1$,..., $Tree_n$
    - iterate for *Node* ::= *Label AST oldASTs Cat Position*:
        - if an endofunction *f : … Cat … -> Cat* applies,
          replace (*AST, oldASTs*) by ((*f...Label…*), *AST+oldASTs*)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





I have a change in plans [next week]
minulla on muutos suunnitelmissa [seuraava viikko]
jag har en ändring i planer [nästa vecka]

Conclusions

GF and UD share the ambition of interlingual syntax

GF's strength is accurate generation

UD's strength is robust parsing

`gf2ud` works well but needs work on annotations

treebank synthetic is promising but does not reach the quality of natural treebanks

`ud2gf` does not scale well (yet)

`gfud` has useful GF-independent functionalities for UD trees