Computational Syntax Exam

LT2214

May 24, 2023

This exam has four questions. Each question is worth 15 points. You need 30 points for the mark G, 45 for VG.

As your help, we provide two tables: UD labels and GF constructs.

relation	explanation	dependent-head	example
acl	clausal modifier of noun	S-N	*the moon as we* ***see*** *it*
acl:relcl	relative clause modifier	S-N	*the moon that we* ***see***
advcl	adverbial clause modifier	S-C	*I leave if she* ***goes***
advmod	adverbial modifier	ADV-C	*he sleeps* ***now***
amod	adjectival modifier	ADJ-N	***black*** *cat*
appos	appositional modifier	N-N	*Macron, the* ***president***
aux	auxiliary	AUX-S	***does*** *he sing*
case	case marking	ADP-N	***on*** *the moon*
cc	coordinating conjunction	CCONJ-C	***and*** *dogs*
ccomp	clausal complement	S-C	*I know that he* ***runs***
compound	compound	N-N	***data*** *science*
conj	conjunct	C-C	*cats and* ***dogs***
cop	copula	AUX-S	*he* ***is*** *old*
csubj	clausal subject	S-S	*that is* ***moves*** *is clear*
dep	unspecified dependency	C-C	(if nothing else works)
det	determiner	DET-N	***the*** *cat*
expl	expletive	PRON-S	***there*** *is hope*
fixed	fixed multiword expression	ADP-C	*because* ***of***
flat	flat multiword expression	PROPN-PROPN	*Adam* ***Smith***
iobj	indirect object	N-VERB	*she gave* ***us*** *a hint*
mark	marker	PART/SCONJ-S	***to*** *go*
nmod	nominal modifier	NOUN-NOUN	*man on the* ***moon***
nmod:poss	possessive modifier	N-NOUN	***my*** *cat*
nsubj	nominal subject	N-S	***John*** *walks*
nsubj:pass	nominal subject of passive	N-VERB	***John*** *was seen*
nummod	numeric modifier	NUM-N	***five*** *cats*
obj	object	N-VERB	*she sees* ***us***
obl	oblique nominal	N-S	*she comes with* ***us***
parataxis	parataxis	VERB-VERB	*I said:* ***come*** *here*
punct	punctuation	PUNCT-S	*I see* ***!***
root	root	S-	*John **walks*
xcomp	open clausal complement	S-S	*I want to* ***go***

Syntactic relations used in UD standard 2, together with their typical uses. In the examples, the dependent is **boldfaced** and its head underlined. S means a sentence-like, N a noun-like, and C any kind of phrase.

Construct	Notation	Example
Abstract syntax module	`abstract`	`abstract Lang =...`
Concrete syntax module	`concrete`	`concrete LangEng of Lang Lang =...`
Resource module	`resource`	`resource ResEng Lang =...`
Module extension	`**`	`abstract Lang = Noun,Verb **...`
Module opening	`open`	`resource ResEng = open Prelude in...`
Abstract syntax category	`cat`	`cat NP`
Abstract syntax function	`fun`	`fun Pred :  NP -> VP -> S`
Linearization type	`lincat`	`lincat N = {s :  Number => Str}`
Linearization rule	`lin`	`lin Pred np vp = np.s!Nom ++ vp!np.a`
Parameter type	`param`	`param Case = Nom | Acc`
Auxiliary operation	`oper`	`oper addS : Str -> Str = \x -> x + "s"`
String concatenation	`++`	`"loves" ++ "Mary"`
Token concatenation	`+`	`"Maria" +"m" ⇓ "Mariam"`
Function type	`->`	`NP -> VP -> S`
Function application	`f a b`	`Pred np vp`
Function abstraction	`\ ->`	`\x,y -> x + y`
Table type	`=>`	`Case => Str`
Table	`table`	`table {Nom =>"she" ; Acc =>"her"}`
Selection from table	`!`	`she_NP.s !  Acc ⇓ "her"`
Table with one branch	`\\ =>`	`\\p,q => np ! q ! p`
Record type	{...:...}	{s : Str ; g : Gender}
Record	{...=...}	{s = "doctor" ; g = Fem}
Projection from record	`.`	`{s = "doctor"}.s ⇓ "doctor"`
Record update	`**`	`doctor_N ** {g = Masc}`
Case expression	`case`	`case np.a of {Ag n _ => cn.s ! n}`
Tuple type	`*`	`Number * Case`
Tuple	`<,>`	`<Sg,Dat>`
Comment	`--`	`-- comment till the end of line`
Comment	`{- -}`	`{- comment of any length -}`

This is a reading guide for GF notation. The first five rows are about modules, the next six list the different kinds of rules. The rest are expressions for types and objects. The notation $e \Downarrow v$ means that expression e is computed to value v.

Question 1

Show the Universal Dependency tree of the following sentence:

Universal Dependencies is a framework for consistent annotation of grammar across different human languages.

Show it in two formats:

- as CoNLLu with fields ID, FORM, LEMMA, POS, DEPREL, HEAD
- as a graphical figure with arrows and dependency labels.

Question 2

Draw a phrase structure tree for the example sentence of Question 1, maintaining as much as possible of the same structure as your UD tree.

Write a context-free grammar that recognizes the sentence with the tree that you drew.

Question 3

Here is a fragment of an abstract syntax of phrases:

```
abstract Noun = {
cat
  CN ; NP ; AP ; Det ;
fun
  AdjCN : AP -> CN -> CN ;
  DetNP : Det -> CN -> NP ;
}
```

In many languages (e.g. Germanic, Latin, Slavic, Semitic), the categories have the following features:

- CN has variable number and case, inherent gender
- NP has variable case, inherent number and gender
- AP has veriable gender, number, and case
- Det has variable gender and case, inherent number

Define the linearization types of these two categories in accordance to these features. You don't need to define what number, gender and case are; they are different in different languages even if their use is similar.

Also define the linearization functions of the two functions, so that the types match in the expected way and the results become type-correct.

- in AdjCN, the AP agrees to the gender of the CN, and the resulting CN inherits it
- in DetCN, the CN agrees to the number of the Det, and the resulting NP inherits the number and the gender from its constituents

You can select the constituent order in any way you want. It differs between languages even when the agreement structure remains the same.

Here are some examples from Latin, using the same word order as English (which is possible in Latin although not always the best choice):

- *omnis altus mons* "every high mountain" (singular, masculine, nominative)
- *omni alto monte* "every high mountain" (singular, masculine, ablative)
- *omne altum castellum* "every high castle" (singular, neuter, nominative)
- *tres alti montes* "three high mountains" (plural, masculine, nominative)

Question 4

The formal language

$$\{a^n b^n c^n \mid n = 0, 1, 2, \ldots\}$$

consists of strings of a's, b's and c's, in this order, and an equal number of each. Thus the following strings belong to the language:

```
a b c
a a b b c c
a a a b b b c c c
```

and so on for any number n. Also the empty string is included (with n=0). But the following do not:

```
a a b c c
b b a a c c
```

This language is known to be non-context-free. Therefore, it is impossible to write a context-free grammar that covers it exactly. But GF is stronger than context-free, and it is easy to write a GF grammar for it.

Your task is hence to write a GF grammar (abstract and concrete syntax) for this language.

Hint: use discontinuous constituents.