Parseme-NL

January 18, 2024

1 Converting Dutch Universal Dependencies data to Parseme

There are currently no Parseme corpora for Dutch. Most existing Parseme corpora build on UD, as explained in Savary et al 2023.

In this project, we enrich Dutch UD data with Parseme annotation automatically. This annotation should then be corrected and validated manually, as not all distinctions made by Parseme are reflected in the UD annotation and some verbal MWEs might not have been annotated as such in UD. (The latter happens when syntax is regular and only semantics is idionatic/non-compositional.)

1.1 Operationalizing Parseme guidelines

We follow the discussion in Savary et al 2023 on the relationship between UD and Parseme annotation, and operationalize it as a set of heuristics that will identify specific Parseme categories on the basis of (Dutch) UD annotation as accurately as possible. Grew-match queries for Dutch UD are formulated to check whether the heuristic accurately identifies the Parseme category. To do the actual automatic annotation of a UD conllu file, we use the parseme.cupt python library.

1.1.1 Inherent reflexives (IRV)

The reflexive pronoun dependent of an inherently reflexive verb is labeled with the dependency relation expl:pv. This Grew-match query on UD Dutch Alpino finds relevant instances. According to Parseme guidelines, verbal expressions containing an inherent reflexive as one of its parts (ie there also is another dependent of the verb that is part of the verbal expression), are labeled VID. This query finds inherent reflexives that are part of such a VID. As we can accurately identify IRV cases on the basis of UD annotation, no manual verification is necessary.

1.1.2 Verb-Particle Construction (VPC)

The VPC class refers to verbs that combine with a separable particle (ie we both have '*ik bel haar op*' and '*dat ik haar op moet bellen*' as well as '*ik moet haar opbellen*'). This is a frequent phenomenon in Germanic languages. In UD, the dependency label compound:prt is used to identify a particle dependent of a verb. However, not all compound:prt cases are Parseme VPCs. We only annotate verb-particle combinations as VPC where the particle has upos ADP or ADJ or ADV, and where the node labeled compound:prt does not have dependents of its own (Grew-match query). Other instances involving a compound:prt are discussed below.

If the particle immediately precedes the verb, the two are written as a single word. Nevertheless, these are also labeled as VPC. In the Dutch UD corpora, such cases can be identified as verbs where the lemma consists of prefix, a '_' and a lemma, ie 'uit_schakelen' ('eliminate, switch-off') (and there is no compound:prt dependent in the sentence that matches the prefix), Grew-match

Issue Parseme distinguises between VPC.full, VPCs with a fully non-compositional meaning, and VPC.semi, VPCs where the meaning of the constructions can be derived to some extent from the meaning of its parts. As UD is a syntactic annotation framework, and this is a semantic distinction, this distinction requires manual checking. As most VPCs seem to be non-compositional, and approximately 75% of the VPCs in EN and DE are VPC.full, we label all VPCs as VPC.full for now.

Issue Note that this query also matches cases where the verb is a participle modifying a noun (amod) or cases of nominalisation (*het afgelopen weekeinde, de afgetreden bestuursleden*). The guidelines suggest that such cases are not to be included in the annotation. DE does not have such cases (maybe because participles used as nominal modifiers are not marked as such?), EN does contain a few cases (a marked-up draft, fucked up kids). On the other hand, there are also cases where the verb has a compound dependent but the result is not marked as MWE (above referenced counterparty, freeze dried food, radiation-induced cancers). For now, we have decided to not mark these as VPC (by excluding particple verbs that have the deprel amod).

1.1.3 Light verb constructions (LVC)

Light verb constructions consist of a semantically light verb and a predicative noun. In UD, such cases can be identified by searching for nouns that are compound:prt dependents of a verb and that do not have dependents of their own.Grew-match

Note that this set contains a wide variety of verbs, not all of which should probably be seen as semantically light. It may be possible to filter these out, ie by creating a list of verbal lemma's that can head a light verb construction. Also, a brief inspection of the EN (33 clusters by verb lemma) ,DE (81), and FR (92) parseme corpora suggests that quite a few different verbs are used in the LVC construction, and it does not seem to be a closed class. Grew-match EN, DE FR

Issue A subset of these are cases where a verb has two compound:prt dependents, one a particle and the other a noun. I.e. for the query above, cases with bij_zetten are instances of *'luister bijzetten'* 'add luster to'. We currently analyze these as a single VID (following the guidelines for LVC + IRV). Note, however, that German seems to adopt an approach where these are labeled as overlapping LVC + VPC)

Issue The set of LVC also includes many cases where the noun is an obj dependent of the verb. Assuming predicative nouns in this construction are singular and not introduced by a determiner (last requirement seems too restrictive, ie *geen rekening houden*), this query finds relevant cases (*deel uitmaken (VID?), rekening houden, gebruik maken, ...*) Grew-match Note that the Alpino lexicon contains rich information on idiomatic verb-object constructions. All or a subset of these, where the verb is one of a small set (*hebben, laten, krijgen, ...*) could be labeled as LVC.

Issue As with VPCs, there are 2 kinds of LVC, LVC.full (non-compositional) and LVC.cause (verb describes causal relation to noun, e.g. for German: Grew-match Parseme We initially label all LVC as LVC.full.

1.1.4 Verbal Idioms (VID)

If the node labeled compound:prt has dependents itself (staat in het teken van, aan de orde stellen), the construction is most likely a verbal idiom (VID) in Parseme. Also, if the word that is labeled compound:prt is a predicative noun, it is most likely a light verb construction (LVC). Grew-match

Issue Some verbs have 2 compound:prt dependents. For now, it is assumed these are always VID (er uit zien). Also, some of these compounds have deps themselves: te wensen over laten –> compound:prt(laten,wensen), mark(wensen,te), compound:prt(laten,over) laat zich in de kaart kijken , doe geweld aan , nieuw leven inblazen, luister bij zetten, all OK now

However, one might argue that te wensen over laten is actually a MVC or even that it is a combination of a MVC and VPC (as seems to be done often in the other corpora). Also, see the guidelines for VID+IRV and guidelines for multiple dependents Guidelines suggest that these cases should actually be annotated as a single VID.

Issue van plan zijn de vakantie in t Harde door te brengen : van plan is svp dep of zijn in Alpino, but gets attached to door-brengen in UD , this is an UD conversion mistake (fix: only label zijn as aux if it does not have a svp sister) Grew-match

1.1.5 Multi-verb constructions (MVC)

These are cases where the compound:prt is a verb. The compound verb often has a te mark dependent, these are also included. Examples: laten zien, heeft te maken, geroepen voelen, complex cases: te wensen over laten (2x compound:prt and wensen has a dep), zich laten aan_zien: here the compound:prt is a VPC (and there is a IRV as well), zich in de kaart laten kijken : IRV + VID + MVC ?, raak vuren? (does not feel like a MVC) laat (niets) van zich horen : case(zich,van) Grew-match

1.1.6 Workflow

First, the UD data in CONLLU format are mapped by a shell script to the CUPT format by adding a new column PARSEME:MWE that initially contains '*' for all tokens.

Next we use the cuptlib python library to read the data, match with specific words, and infer the new annotations. The enriched data is then written to an output file.

[1]:
```python
import conllu
import parseme.cupt as cupt
from parseme.cupt import MWE

import json
```

[2]:
```python
corpus = 'nl_alpino-ud-all'

cupt0 = corpus + '.cupt.0'

with open(cupt0, encoding="utf-8") as f:
    data = f.read()

sentences = conllu.parse(data)
```

```
# dictionary derived from alpino with fixed expressions (fixed) and␣
↪semi-flexible fixed expressions that are mapped to regular deprels in UD
with open('alpino_dictionary.json') as f:
    dictionary = json.load(f)
```

```
[3]: def compound_children(id,sentence) :   # cases where annotation uses fixed for␣
↪additional deps
    children = []
    for dep in sentence : # ten onder ADP ADP
        if dep['head'] == id and dep['upos'] in ['NOUN','DET','ADP','PRON'] :
            children.append(dep['id'])
    return children

def dictionary_item(headlemma,deprel,lemma) :
    status = False
    for entry in dictionary:
        try :
            if entry['root'] == headlemma and entry['deprels'][deprel] == lemma␣
↪:
                status = True
        except KeyError :
            True
    return status

def case(nounid,sentence) :
    for dep in sentence :
        if dep['head'] == nounid and dep['deprel'] == 'case' :
            return dep['id']

# zie ook https://parsemefr.lis-lab.fr/parseme-st-guidelines/1.3/?
↪page=irv#irv-overlap-vid
# schreef op zijn naam, also incude op zijn? (deps of naam, cmp:prt?) no

# alternative: for a given verb: find all mwe-lexical deps, collect classes,␣
↪collect ids,
# (for one-word-particle cases: add VPC if not already in classes)
# then decide on label on basis of class(es)

#grep VERB nl_lassysmall-ud-all.cupt |cut -f 8 |sortnr
non_verbal_deprel = ['amod','fixed','obl','nmod','obl:arg','nsubj','nsubj:
↪pass','flat','obj','compound:prt','mark']

for sentence in sentences:
    mwes = []
    for token in sentence:  # skip particples modifying a noun and other␣
↪nominalisations and parts of larger idioms
        if token['upos'] == 'VERB' and token['deprel'] not in non_verbal_deprel:
            classes = []
            (verbid,verblemma) = (token['id'],token['lemma'])
            pointers = [verbid]
            for dep in sentence :
                (depid,deprel,depupos,deplemma) =␣
↪(dep['id'],dep['deprel'],dep['upos'],dep['lemma'])
                if dep['head'] == verbid :
                    if deprel == 'expl:pv' :
                        pointers.append(depid)
                        classes.append('IRV')
                    elif deprel == 'compound:prt' :
                        pointers.append(depid)
                        children = compound_children(depid,sentence)
                        if children :
                            pointers = pointers + children
                            if depupos == 'VERB' :
                                classes.append('MVC')
                            else :
                                classes.append('VID')
                        elif depupos == 'NOUN' :
                            classes.append('LVC.full')
                        elif depupos == 'VERB' :
                            classes.append('MVC')
                        else :
                            classes.append('VPC.full')
                    elif deprel == "obj" and␣
↪dictionary_item(verblemma,deprel,deplemma) :
                        pointers.append(depid)
                        classes.append('LVC.full')
                    elif deprel == "nsubj" and␣
↪dictionary_item(verblemma,deprel,deplemma) :
                        pointers.append(depid)
                        classes.append('VID')
                    elif deprel == "obl:arg" :
                        caseid = case(depid,sentence)
                        if caseid :
                            pointers.append(caseid)
                            classes.append('IAV')
                    # elif # obl er case?
            if 'VPC.full' not in classes and '_' in verblemma :
                classes.append('VPC.full')
            if classes :
                if len(classes) > 1 :
                    classes = ['VID']
                mwe = MWE(classes[0],pointers)
                mwes.append(mwe)
        cupt.replace_mwes(sentence,mwes)
```

[4]:
```
outfile = corpus + '.cupt'

with open(outfile, 'w', encoding='utf-8') as f:
    for sentence in sentences:
        f.write(sentence.serialize())
```

1.2 Open Issues and To do

- enhanced deps can reconstruct elided nodes, also annotate?
- expletive subjects + verb ?
- complex syntax cases, ie where obj has a relative clause with the support verb? is that possible?
- in zijn hemd staan, clear VID, not annotated as such
- guidelines suggest that some PPs can also be LVCs

[11]:
```

```