Universiteit Leiden

Master Computer Science

Polarity Marking Dutch Parse Trees

Name:	Lars Ruigrok
Student ID:	2292998
Date:	Draft 30/06/2024
Specialisation:	Computer Science
1st supervisor:	Gijs Wijnholds
2nd supervisor:	Akrati Saxena

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Monotonicity reasoning allows humans and computers alike to efficiently draw inferences from natural language statements. For example, in the sentence "Not all ducks quack", we can replace "ducks" with the more generic term "birds" to get a valid inference "Not all birds quack"; similarly, we may replace "quack" with a more specific phrase: "Not all ducks quack loudly". Whether we can substitute a more specific or more generic phrase depend on the *polarity* of the position. In this example, "ducks" occurs with positive polarity ↑, while "quack" occurs with negative polarity ↓:

"Not↑ all↓ ducks↑ quack↓".

Based on Chen & Gao's Udep2Mono [CG21], we built a system to automatically mark polarity in Dutch sentences using Alpino dependency structures. Alpino is a syntactic annotation scheme and parser specifically tailored to Dutch, as opposed to Universal Dependencies which make some compromises for cross-linguistic consistency.

Contents

1 Introduction

There is a lot of excitement around Large Language Models (LLMs) as of late. These neural networks trained on self-supervised tasks with huge piles of data can perform a wide range of natural language tasks (zero-shot or after some finetuning). But while their performance on natural language reasoning tasks appears impressive, these results often do not generalize between benchmarks [TC19]. Commonly-used benchmarks often inflate the results of the kind of systems they were designed to evaluate, because they ignore aspects of human reasoning not covered by those systems [Ber02].

One area where the dominant transformer models still struggle is in Monotonicity Reasoning, particularly when downward entailing (negative polarity) contexts are involved [YMB+19a, Wij23].

For example, you and BERT[1] will likely have no problem recognising the following as a valid inference:

"Some ducks quack" $\implies$ "Some birds quack"

However, while you would recognise the following with similar ease, BERT performs significantly worse on cases like this[2]:

"All birds quack" $\implies$ "All ducks quack"

In the first example, "ducks" occurred with positive polarity, so replacing it with a more general term "birds" yields a valid inference. In the second, "birds" occurs with negative polarity, such that instead replacing it with a more specific term yields a valid inference.

Part of the problem is that positive polarity is more common. Data augmentation may help here [YMB+19b]. But adding more examples with negative polarity can actually reduce accuracy on positive polarity statements, indicating the real problem is that neural models do not 'understand' monotonicity reasoning [YMB+19a] [3].

Symbolic systems like MonaLog do understand monotonicity, but don't handle syntactic variation well. MonaLog on its own does not have particularly impressive accuracy. But it can moderately improve on BERT performance in a hybrid sytem, which uses MonaLog when it gives a definite prediction, but defers to BERT when it is unsure. MonaLog can also improve BERT by generating additional training data [HCR+20].

NeuralLog [CGM21] is a more advanced hybridization of BERT with a monotonicity reasoning component. It automatically searches for a sequence of neural or symbolic inference steps from the premise to the conclusion.

[1] BERT here referring to a suitably fine-tuned variant of Devlin et al.'s transformer model [DCLT19]

[2] this particular case is completely fabricated for illustratory purposes, but based on table 9, cells 'Up Lexical, -HELP' and 'Down Lexical, -Help' of Yanaka et al. [YMB+19a]

[3] at least in 2019. A recent (May 2024) preprint [LNT+23] suggests GPT-4 single-shot outperforms fine-tuned BERT with data augmentation on the Monotonicity Entailment Dataset and comes close to human average accuracy. Since this preprint has not been reviewed and we know little about the internals of GPT-4, I'd take that with a pinch of salt. It's possible MED was part of GPT-4's training data, for example. Results on other NLI datasets were less impressive, and the authors conclude "Logical reasoning remains challenging for ChatGPT and GPT-4, especially on out-of-distribution and natural language inference datasets".

Both generating additional negative polarity training data and hybrid systems benefit from an accurate polarity marking component. MonaLog uses CCG2Mono [HM18], which was also employed in creating the MED [YMB+19a] dataset, while NeuralLog uses Udep2Mono [CG21], the current state-of-the-art in polarity marking.

Though Udep2Mono uses the multilingual Universal Dependencies [NdG+16] with a language-neutral transformation [RTC+16] to represent natural language statements, it is English-specific. Our goal is to bring the state-of-the-art in polarity marking to Dutch, by

1. Translating Udep2Mono to Dutch UD.
2. Building an alternative system using Alpino dependencies.

Alpino dependencies [TNH02] provide a slightly more sophisticated annotation than Universal Dependencies specifically tailored to Dutch. We want to see if this can be leveraged to improve polarity marking accuracy.

1.1 Thesis Overview

Section 2 lists some prior work on polarity marking and other related work. Section 2.1 makes the notion of polarity ↓↑ more precise by introducing the concept of monotonicity. Section 2.2 compares different syntactic parsing paradigms used in this and prior work. Section 2.3 shows how such a parse is marked with polarities. A more concrete description of our implementation is given in Section 3. In Section 4, we measure accuracy of our system on a translation of the small test set used by Chen&Gao [CG21] and originally from Hu&Moss [HM20]. In Section 5 we see whether Alpino's features actually provide a polarity-marking advantage over Universal Dependencies. Section 7 concludes.

2 Background and Related Work

Given some natural language statement, we want to find polarity marks for each phrase, e.g. "All↑ ducks↓ quack↑". These marks indicate whether a phrase can be replaced by a more general (in case of positive polarity ↑) or more specific (in case of negative polarity ↓) phrase to yield a valid inference.

This presupposes a number of things, which the following sections aim to explain. First, in section 2.1 we formalise what it means to be 'more specific' or 'more general'. In section 2.2 we look at various ways to think about what constitutes a phrase and how these phrases are composed to form a sentence. Finally section 2.3 describes how polarity is actually computed.

Some early work on natural language polarity marking includes Van Benthem [VB86] and Sanchez-Valencia's external monotonicity marking algorithm for Lambek Grammars [San91].

A number of alternative algorithms and extensions to different grammars have since been proposed. Sanchez-Valencia's algorithm consists of two passes: external monotonicity marking and polarity marking. Dowty [Dow94] proposes a single-pass approach, marking polarities as the parse tree is constructed. Dowty's approach introduces a lot of additional syntactic categories. Van Eijck's algorithm [van07] instead annotates categories with polarity mappings, which determine the polarity

of arguments from the polarity of the function. This requires two passes again: a bottom-up pass adding the mappings and a top-down pass computing polarity. Lavalle-Martinez et al [LMV+18] show Van Benthem, Sanchez-Valencia, Dowty and Van Eijck's algorithms are equivalent.

Hu&Moss extend Van Benthem's polarity marking algorithm to CCG and produce a practical software implementation [HM18].

Accuracy of parsers for categorial grammars had been falling behind dependency parsers, prompting Chen&Gao [CG21] to develop their system for polarity marking Universal Dependencies. They adapt the obliqueness hierarchy of Reddy et al. [RTC+16] to transform UD trees into binary trees more suitable for polarity marking.

Not all natural language reasoning is monotonicity reasoning. MacCartney and Manning [MM14] extend the monotonicity calculus to a projectivity calculus, enabling reasoning about not only inclusion but also exclusion relations. They also implement an algorithm for marking projectivity into Stanford CoreNLP [MM08].

Nairn et al. [NCK06] investigate the relations between main and complement clauses. They do not focus on monotonicity, but arrive at a compatible notion of polarity.

2.1 Monotonicity

To clarify the concepts of polarity and monotonicity, I would recommend Icard III and Moss' *Recent Progress on Monotonicity* [IIM14].[4] I will briefly summarise their account below.

A function $f\colon X \to Y$ is *monotone increasing* if whenever $a \sqsubseteq b$, also $f(a) \sqsubseteq f(b)$. If instead $f(a) \sqsupseteq f(b)$, f is *monotone decreasing*. Here $\sqsubseteq$ is some (partial) ordering on the domain X or codomain Y. For example, for functions on real numbers $\mathbb{R}$ we may take the standard $\leq$ ordering for both domain and codomain. Then $f(x) = x+1$ is monotone increasing, $g(x) = -x$ is monotone decreasing, but $h(x) = x^2$ is non-monotone.

Composing functions of identical monotonicities produces a monotone increasing function, while opposite monotonicities produce a monotone decreasing function. So for example, $k(x) = g(f(x)) = -(x+1)$ is monotone decreasing. Composing with a non-monotone function always results in a non-monotone function.

Expressions which are the argument of a monotone increasing (/decreasing) function have positive (/negative) polarity. Thus a has negative polarity in $-(a+1) = g(f(a))$. We mark this with a $\downarrow$ symbol after the expression: $-(a\downarrow + 1)$. Likewise with $\uparrow$ for positive polarity and $=$ for neither.

Knowing the polarity of an expression allows us to draw some simple inferences by substituting $x\uparrow$ expressions with $x' \sqsupseteq x$ or $x\downarrow$ with $x' \sqsubseteq x$: If $-(a\downarrow + 1) < 6$, and we know $a < b$, it follows that $-(b+1) < 6$.

When moving on to linguistic uses we will also want to order more complex expressions. We define the ordering on booleans $\mathbf{F} \sqsubseteq \mathbf{T}$, such that $\sqsubseteq$ is implication. Functions $X \to Y$ are ordered pointwise: $f \sqsubseteq g \iff \forall x \in X\colon f(x) \sqsubseteq g(x)$. Sets are identified by their membership function

[4] their notation is somewhat different from ours, which follows Chen&Gao and Hu&Moss. Where they use x^+ and x^- to denote polarities, we use $x\uparrow$ and $x\downarrow$, and additionally $x=$ to mark nonpolarity.

$U \to \{\mathbf{T}, \mathbf{F}\}$, so their ordering is the subset relation $\subseteq$:
$A \sqsubseteq B \iff \forall x \in U\colon (x \in A) \sqsubseteq (x \in B) \iff \forall x\colon ((x \in A) \implies (x \in B)) \iff A \subseteq B$.
Since sets are functions, they also have monotonicity. A set $A \subseteq U$ is upward monotone iff $a \in A \iff \forall a' \sqsubseteq a\colon a' \in A$, and similarly for downward monotone.

To extend these concepts to natural language we need to define the semantics of linguistic expressions and their orderings $\sqsubseteq$. Below is just an outline of one way to do that.

Example Natural Language semantics for Monotonicity Reasoning Here we present a semantics which serves as a guideline for the implementation described in section 3. We use set notation to make the orderings more obvious, but we could equally well have used lambda notation to emphasize monotonicities: $\{e \in E \mid p(e)\} \cong \lambda e(Ee \wedge pe)$

Assume some model which defines denotations of tokens ($[\![\text{“word”}]\!]$) in terms of entities E, booleans $\{\mathbf{T}, \mathbf{F}\}$ and functions thereof. We will write (dep $d\ h$) for a binary dependency relation with label[5] dep between dependent d and head h.

Statements are either $\mathbf{T}$ or $\mathbf{F}$.

Nouns denote sets of entities:

$$[\![\text{“green turtles”}]\!] = \{e \in E \mid isGreen(e) \wedge isTurtle(e)\}$$

Verb phrases also denote sets of entities, those with which they would form a true statement:

$$[\![\text{“swim”}]\!] = \{e \in E \mid swims(e)\}$$

Verbs may take additional arguments to form a verb phrase:

$$[\![\text{“eat”}]\!]_{\text{“turtles eat crustaceans”}} = \{(e, o) \in E \times E \mid eats(e, o)\}$$

But since objects are often optional, a single verb word can thus have different types depending on use:

$$[\![\text{“eat”}]\!]_{\text{“turtles eat”}} = \{e \in E \mid \exists o\colon eats(e, o)\}$$

Determiners denote a relationship between predicate and subject:

$$[\![\text{“Some”}]\!](A, B) = |A \cap B| \geq 1$$

How these combine to form more complex phrases is mediated by dependency relations.

The det relation combines a noun with a determiner to create a noun phrase, which is a set of sets of entities:

$$\text{det}[\![\text{“Some”}]\!][\![\text{“turtles”}]\!] = \{B \subseteq E \mid [\![\text{“Some”}]\!]([\![\text{“turtles”}]\!], B)\} = \{B \subseteq E \mid |isTurtle \cap B| \geq 1\}$$

[5]Labels used are from Universal Dependencies [NdG+16], as described in section 2.2. See `universaldependencies.org/u/dep/` for the full list.

Noun phrases form a clause when combined with a predicate by a subj relation:

$$\text{nsubj}[\![\text{“Some turtles”}]\!][\![\text{“swim”}]\!] = ([\![\text{“swim”}]\!] \in [\![\text{“Some turtles”}]\!])$$

The polarity of predicate and subject in these simple cases is determined by the determiner as listed in Table 1.

Noun phrases are also used as objects or other arguments of verb phrases:

$$\text{obj}[\![\text{“crustaceans”}]\!][\![\text{“eat”}]\!] = \{e \mid e, o \in E \times [\![\text{“crustaceans”}]\!] \cap [\![\text{“eat”}]\!]\}$$

As with subj, obj, iobj and obl are monotone increasing in the noun phrase.

Proper nouns, pronouns and bare plurals are also noun phrases. Let $[\![\text{“John”}]\!]_e \in E$ be the individual denoted by “John”. Then the corresponding proper noun is $[\![\text{“John”}]\!]_{\mathbf{np}} = \{B \subseteq E \mid [\![\text{“John”}]\!]_e \in B\}$ [Pet08b]. Proper nouns are thus monotone increasing. Bare plurals are more complicated, but Peters [Pet08b] gives the following approximation, under which bare plurals are also monotone increasing:

$$[\![\text{“turtles”}]\!]_{\mathbf{np}} = \{B \mid \emptyset \neq [\![\text{“turtles”}]\!] \subseteq B\}$$

Most pronouns are upward monotone, but “nothing” and variants like “nobody”, “nowhere” etc. are downward monotone:

$$[\![\text{“nothing”}]\!] = \{B \subseteq E \mid |B| \leq 0\}$$

Most adjectives are intersective and thus monotone increasing, i.e.

$$\text{amod}[\![\text{“green”}]\!][\![\text{“turtles”}]\!] = isGreen \cap isTurtle$$

However, to be able to model non-intersective adjectives like “former”, “fake”, and superlatives, we model all modifiers as functions $X \to X$:

$$[\![\text{“green”}]\!](x) = isGreen \cap x$$

$$\text{amod}\ \ d\ h = d(h)$$

Intersective adjectives are still ordered by their defining property: $[\![\text{“green”}]\!] \sqsubseteq [\![\text{“colored”}]\!]$

The most obvious example of a downward-monotone modifier is negation “not”:

$$[\![\text{“not”}]\!](x_{\mathbf{vp}}) = x^C$$

$$\text{advmod}[\![\text{“not”}]\!][\![\text{“swim”}]\!] = swims^C = \{e \in E \mid \neg swims(e)\}$$

as well as more specialised forms of negation like “never”, “nowhere”, etc. The denotation above is just a special case, negation can apply to any type with final result type $\mathbb{B} = \{\mathbf{T}, \mathbf{F}\}$, including

Statements: $[\![\text{“not”}]\!](x_{\mathbb{B}}) = \neg x$
Nouns: $[\![\text{“not”}]\!](X_{E\to\mathbb{B}}) = \lambda e\colon \neg X(e) = X^C$
Noun modifiers: $[\![\text{“not”}]\!](f_{(E\to\mathbb{B})\to(E\to\mathbb{B})}) = \lambda X \lambda e\colon \neg f(X)(e) = \lambda X\colon f(X)^C$

In any case, negation is downward monotone.

Conjunctions (in the broad sense including not only "and" but also "or", "but", etc.) are monotone increasing in both conjuncts.

$$\text{conj}(\text{cc}[\![\text{“and”}]\!][\![\text{“turtles”}]\!])[\![\text{“seals”}]\!] = [\![\text{“turtles”}]\!] \cup [\![\text{“seals”}]\!]$$

In case of higher-order conjuncts, if both conjuncts have the same monotonicity the conjunction also has that monotonicity. For example, in "everybody or nobody leaves", the monotonicities of "everybody" and "nobody" are opposite, so the conjunction is non-monotone and the polarisation should be "everybody↑ or↑ nobody↑ leaves=".

Conditional expressions of the form "If X [then] Y" are monotone decreasing in X and increasing in Y. [Mos12] The binary dependency representation of such expressions uses the advcl adverbial clause relation:

$$\text{advcl}(\text{mark}[\![\text{“if”}]\!]X)Y = (X \implies Y)$$

We assume the basic interpretation of numerals used as quantifiers is 'at most', but context changes the meaning of bare numerals to 'at least' or 'exactly'. This strange arrangement does not make much linguistic sense [Spe13], but it allows us to order numerals according to their natural ordering $\leq$:

$$[\![\text{“5”}]\!](A, B) = |A \cap B| \leq 5 \tag{1}$$

$$[\![\text{“6”}]\!](A, B) = |A \cap B| \leq 6 \tag{2}$$

$$\forall A, B\colon [\![\text{“5”}]\!](A, B) \sqsubseteq [\![\text{“6”}]\!](A, B) \tag{3}$$

Modifiers 'at least', 'at most', 'exactly', etc. turn numerals into the quantifiers listed in Table 1. More about how our system infers the intended reading of bare numerals in implementation section 3.2.

We have been assuming all phrases are to be interpreted in the same model. However, once we introduce subclauses this simplification will not hold. For example, in the sentence "Ed believes that the earth is flat", we might feel justified in replacing the phrase "the earth" in "the↑ earth= is↑ flat↑" with the more general "some planet": "The earth is flat" $\sqsubseteq$ "some planet is flat.". This would certainly be a valid inference in our common world (even if the premise is false), but it may not hold in the world of the subclause: Ed might not believe the earth is a planet. If our intention is to draw inferences by applying substitutions from a single knowledge base of real-world facts, we should in general consider verbs to be non-monotone in clausal arguments.

Some constructions do allow inference in subclauses, however. Implicatives carry entailments about their subclauses [NCK06]. For example, the sentence "Ed forgot to lock the door" implies Ed did not lock the door. Implications depend on the polarity of the implicative construction: in a negative context "Ed did not forget to lock the door", our example yields the opposite implication, Ed did lock the door. For some implicatives, implications are only available in a positive or negative context: "Ed was forced to lock the door" implies he did, but "Ed was not forced to lock the door" does not imply whether he did or did not. Nairn et al [NCK06] recognise 6 implication signatures, based on the availability and direction of entailments in positive and negative polarity contexts. Table 2 shows the signatures, with the monotonicies specified by MacCartney&Manning [MM14].

class	mono	example (en)	example (nl)	interpretation
existential	↑↑	some A B	sommige A B	$\lvert A \cap B \rvert \geq 1$
	↑↑	at least n A B	ten minste n A B	$\lvert A \cap B \rvert \geq n$
universal	↓↑	all A B	alle A B	$\lvert A - B \rvert \leq 0$
non-existential	↓↓	no A B	geen [enkele] A B	$\lvert A \cap B \rvert \leq 0$
	↓↓	at most n A B	hooguit n A B	$\lvert A \cap B \rvert \leq n$
non-universal	↑↓	not all A B	niet alle A B	$\lvert A - B \rvert \geq 0$
exact	==	exactly n A B	exact n A B	$\lvert A \cap B \rvert = n$
relative	=↑	most A B	[de] meeste A B	$\lvert A \cap B \rvert > \lvert A - B \rvert$
relative	=↓	few A B	weinig A B	$\lvert A \cap B \rvert < p\lvert A - B \rvert$
definite	=↑	the A B	de A B	$\lvert A \cap B \rvert = \lvert A \rvert$

Table 1: Monotonicities of various kinds of determiners [Pet08a]. ↑ indicates monotone increasing, ↓ decreasing and = non-monotone. First monotonicity applies to subject, second to predicate.

example	Entailment in ↑ context	Entailment in ↓ context	monotonicity
manage to	↑	↓	↑
forget to	↓	↑	↓
force to	↑	=	↑
refuse to	↓	=	↓
attempt to	=	↓	↑
hesitate to	=	↑	↓

Table 2: Implicative signatures reproduced from Nairn et al [NCK06, Fig. 1] with monotonicities from MacCartney&Manning [MM14, §6]. For the monotonicity column, ↑ indicates the verb is monotone increasing in its clausal argument, ↓ monotone decreasing, and = non-monotone.

Another place where we need to be mindful of our model is when dealing with comparatives. Comparatives like "more/less than", "heavier/lighter than", "taller/shorter than", etc. are upward monotone in one argument and downward monotone in the other. However, they are not monotone with respect to the ordering on phrases we defined previously, but their own orderings.

> "(Some football)$\downarrow$ is ligher than (any standard basketball)$\uparrow$"

Under our phrase ordering, ⟦"Some football"⟧ $\sqsupseteq$ ⟦"Some lead-filled football"⟧. However, under the 'weight' ordering ⟦"Some football"⟧ $\not\sqsupseteq$ ⟦"Some lead-filled football"⟧. Treating comparative polarity marks the same as other polarity marks would thus lead to invalid inferences:

> "Some lead-filled football is ligher than any basketball."

Since we only have one set of polarity marks, the easiest solution would be to replace all comparative marks with =:

> "Some= football= is ligher than any= standard= basketball="

However, we'd be missing out on some useful inferences.

> "Any$\downarrow$ football is ligher than a$\uparrow$ basketball" $\sqsubseteq$ "Some football is ligher than a basketball"
> "5$\downarrow$ footballs are ligher than 5$\uparrow$ basketballs" $\sqsubseteq$ "4 footballs are ligher than 6 basketballs"

For scalar comparatives, marks on numeral quantifiers remain valid.

The comparatives "more/less" compare the size of their arguments. Since $A \subseteq B \implies |A| \leq |B|$, we can treat the "more/less" polarity marks as phrase polarity marks without allowing invalid inferences.

I have omitted many details, and there are other ways to formalise this, but the common conclusion is that upward polarity $\uparrow$ allows replacing with hypernyms (more generic words) or removing adjectives, downward polarity $\downarrow$ allows replacing with hyponyms (more specific words) or adding adjectives.

2.2 Parsing

Most of the polarity marking algorithms mentioned in Section 2 are based on a form of Categorial Grammar. In Categorial Grammars, words form sentences by means of function application. Such grammars are easily mapped to a function application based semantic model, which is ideal for monotonicity and polarity computation. Figure 1 shows an example of CCG, the CG variant used by Hu&Moss [HM18].

Categorial Grammars are closely related to Constituency Grammars. These similarly create a hierarchy of constituents, but are not typed and may allow more or less child nodes. Figure 2 shows an example of the kind of phrase structures used by MacCartney & Manning's NatLog [MM08]. Mapping this to a semantic model is more difficult, requiring complex pattern matching to recognise important constructions. We know in the example "bak" is related to "vandaag" and "een appeltaart", but we know very little about the nature of these relations.

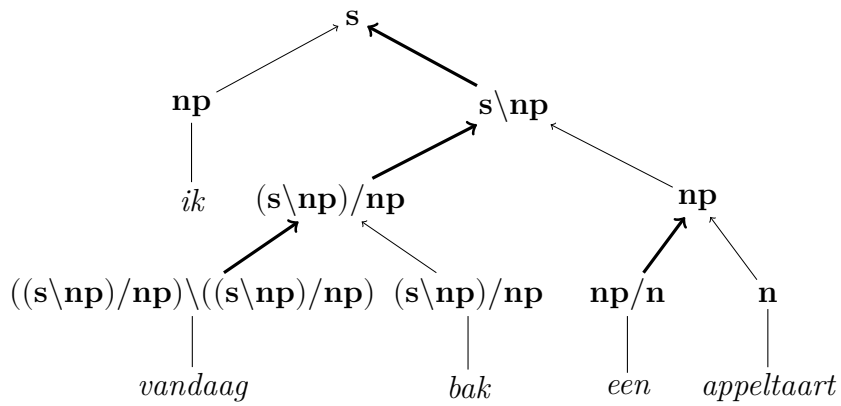


Figure 1: CCG tree for the sentence "Ik bak vandaag een appeltaart" ("I bake today an apple-pie"). Inner nodes show constituent types. **s** and **np** are primitive types for clauses (sentences) and noun phrases, respectively. CCG is directional: (Y/X) is the type of a function which takes an argument of type X on the right to produce a constituent of type Y, $(Y\backslash X)$ instead takes a X on the left. (Note some grammars write $X\backslash Y$ instead of $Y\backslash X$)

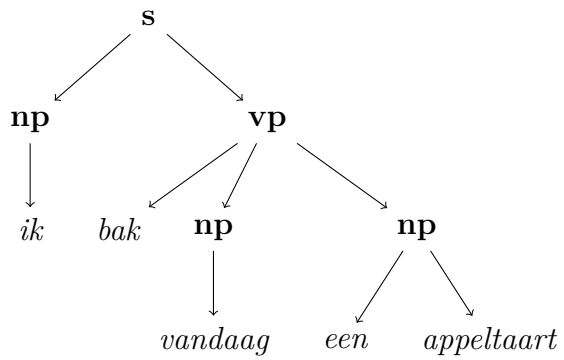


Figure 2: Constituency (Phrase-Structure) tree for the sentence "Ik bak vandaag een appeltaart" ("I bake today an apple-pie"), using Penn Treebank tags: sentence **s**, noun phrase **np**, verb phrase **vp**.

Dependency Grammars usually forego the phrasal constituents, focusing only on directed links between head and dependent words. Figure 3 shows an example of Universal Dependencies (UD), a multi-lingual dependency representation aimed at Natural Language Understanding tasks [NdG+16].

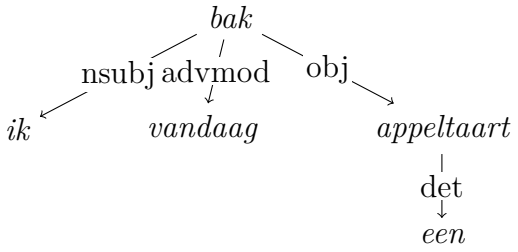


Figure 3: Universal Dependency tree for the sentence "Ik bak vandaag een appeltaart" ("I bake today an apple-pie"). Arrows point from head to dependent. The UD relation labels include some information about the type of the dependent: nsubj stands for 'Nominal Subject', distinguishing it from a clausal subject. See `https://universaldependencies.org/u/dep/` for other relation labels.

The labelled dependencies give us enough information to derive semantic logical forms, as shown by Reddy et al [RTC+16]. Half-way between UD and Reddy's logical forms are binarized dependency trees. In Reddy et al.'s approach [RTC+16], dependency trees are first transformed into binary trees by turning the edge labels into nodes and rotating them up such that all original lexical nodes become leaves. When a lexical node has multiple children, we choose which to rotate first based on the hierarchy shown in Table 4: relation labels higher in the list are placed higher in the binarized tree. Figure 7 shows an example. The process of binarizing dependency trees is described in more detail in Section 3.1. Each node (both lexical and relation labels) is associated with a semantic value. In Reddy et al.'s work, these are tailored to the structure of Freebase. Because we are only interested in polarity marking, we can use a slightly simpler representation, as in section 2.1. To get the value of a tree, the root node is applied to the value of its two child trees:

$$
\begin{aligned}
&det[\![\text{“een”}]\!][\![\text{“appeltaart”}]\!] \\
=&(\lambda d \lambda h\colon d(h))\ \{(A,B) \in 2^E \times 2^E \mid |A \cap B| \geq 1\}\ \{e \in isPie \mid filling(e, [\![\text{“appel”}]\!])\} \\
=&\{B \subseteq E \mid |\{e \in isPie \mid filling(e, [\![\text{“appel”}]\!])\} \cap B| \geq 1\}
\end{aligned}
$$

Udep2Mono and our system do not explicitly perform this last step, but operate directly on the binarized trees. We also don't need to fully specify the value of each node, just its monotonicity signature.

We will also consider Alpino dependencies [vdBBMvN02] and their binarizations, which provide a slightly more sophisticated[6] representation for Dutch. See Figure 4. Not shown in Figure 4 is that unlike UD Alpino dependency structures are not always trees. This allows resolving some elided elements and referring back to arguments of the parent clause in controlled subclauses, as in Figure 10.

[6] Can be automatically converted to UD, see Bouma & Van Noord [Bv17]

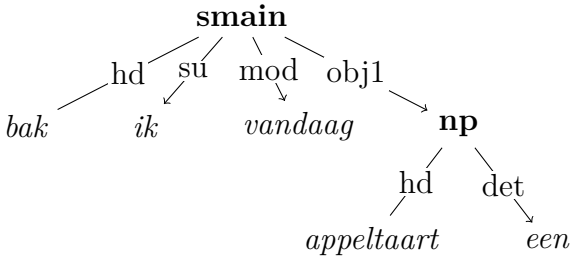


Figure 4: Alpino parse tree for the sentence "Ik bak vandaag een appeltaart" ("I bake today an apple-pie"). **smain** is the category label for the main clause. Despite the presence of phrasal nodes this is still a dependency structure: Every phrasal node has one head child, the other child nodes are dependents of this head. This is shown more clearly in Fig. 5. The use of phrasal categories (along with POS-tags, not shown here) allows the same expressivity as UD with a smaller tagset: there's only one subject relation su, for example. See `nederbooms.ccl.kuleuven.be/php/common/TreebankFreqsLASSY.html` for other relation labels and phrasal categories. And see Bouma & Van Noord [Bv17] for correspondence to UD tags.

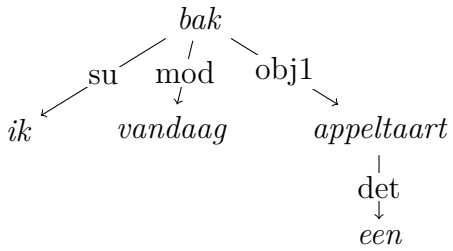


Figure 5: Alpino parse tree as in Fig. 4, but with phrasal nodes replaced by their head nodes. This example has the same structure as the UD parse in Fig. 3, just with different labels. This won't always be the case: UD prefers 'content words' as heads, while Alpino takes markers like "te" ("to") or auxiliaries like "is" as heads of their domain.

2.3 Polarity Marking

At this point, we have a structured representation of the composition of a statement, and each word/token is (implicitly or explicitly) associated with a monotonicity signature.

Now there are a few ways to proceed. If you are Hai Hu or Larry Moss, you first compute monotonicities of all inner nodes. Then you mark the root of your CCG parse tree ↑, look up the appropriate polarity computation based on the CCG rule used to derive the current node, compute polarity of child nodes and repeat for the child nodes until all nodes are marked [HM18].

$$\frac{(x \overset{m}{\to} y)^d \quad x^{m \circ d}}{y^d} >$$

Figure 6: Polarization rule for CCG application rule used by Hu&Moss [HM18]. $x \overset{m}{\to} y$ is the type of a function from x to y with monotonicity m. x^d is an expression with polarity d. The composition operator $\circ$ can be applied to two monotonicities or a monotonicity and polarity. It's isomorphic to multiplication under $\{\downarrow \mapsto -1, \uparrow \mapsto +1, = \mapsto 0\}$.

If however you are Zeming Chen or Qiyue Gao, you will go about it slightly differently. Again you start at the root, you check how the current node was derived, but instead of a CCG rule you find a dependency relation. Again you will look up the associated polarity computation rule. But instead of just setting initial polarities and letting recursion do the rest, you pass the parent's polarity mark on to the function child, and initialize the argument child with ↑, as if it were an independent tree. You recursively polarize the child trees, and only then do you determine the monotonicity of the function child. If it turns out it is downward monotone, you flip the polarity markings of the argument child tree; if it was non-monotone, you set them all to =. One problem you might have noticed is that monotonicity information is not propagated up the tree. Instead some rules, such as for determiners, will directly change their parent tree's polarity. [CG21]

The next section provides some more details on Chen & Gao's algorithm and the problems we encountered when translating it to Dutch, as well as where their approach differs from that of our alternative Alpino-based system.

3 Implementation

We first translated Chen&Gao's Udep2Mono implementation to Dutch. We used the IWCS2021 version, available on github at `github.com/eric11eca/Udep2Mono/blob/06cc712/udep2mono/Udep2Mono.ipynb`[7]. Some rules needed adjustments because of assumptions specific to the English UD parse produced by Stanza, but for the most part rules could be transferred without functional changes. The main effort was in translating the various word lists [8].

[7] Note the implementation in the notebook is not quite the same as the Python modules, which score a good 10 percentage points lower token accuracy on the test described in section 4. We wasted a lot of time initially working from the IWCS2021 version, then trying the latest version, then going back to v0.2, before realising we needed the notebook.

[8] See `git.liacs.nl/monotonicity-marking-from-parse-trees/Udep2Mono` for source code.

Next we reconstructed the algorithm for Alpino dependencies [9]. We ended up following Udep2Mono's design quite closely, precluding some potential improvements. For example, Udep2Mono occasionally modifies the current node's parent tree, which makes properly implementing noun phrase conjunctions difficult.

3.1 Parsing and Binarization

For Universal Dependencies parsing we use Stanza [QZZ+20]. We have to use the older Stanza version 1.2 because Chen & Gao's pretrained models are not compatible with newer versions. For Dutch we used Stanza 1.2's default models, which are trained on the automatically converted Alpino treebank.

We wrote a simple Python client to connect to an Alpino server [vdBBMvN02].

Before we can start polarizing, we need to decide in which order to process the dependency relations. Given a sentence like "Ik bak vandaag een appeltaart", Alpino will produce a tree as in figure 4. Following Chen&Gao we convert this into a binary tree (Fig. 7): parent-child nodes become siblings with their relation label on the parent node, in case of multiple children they are processed based on the position of their relations in the binarization hierarchy shown in Table 4. The algorithm for binarizing an Alpino tree is almost the same as Chen&Gao's algorithm for UD: compare Algorithm 2 and 1. Instead of using the parent node as rightmost child we take the child with lowest priority, which will be hd if present.

Algorithm 1 Binarization algorithm for Universal Dependencies

function BINARIZE(Dependency Tree τ)
    $C \leftarrow \text{SORT}_{Priority}(\text{CHILDREN}(\tau))$
    if $|C| = 0$ **then return** LEAF(TOKEN(τ))
    $b \leftarrow \text{LEAF}(\text{TOKEN}(\tau))$
    for $i \leftarrow 1, |C|$ **do**
        $b \leftarrow \text{NODE}(\text{RELATION}(C_i, \tau), \text{BINARIZE}(C_i), b)$
    return b

Algorithm 2 Binarization algorithm for Alpino dependencies

function BINARIZE(Dependency Tree τ)
    $C \leftarrow \text{SORT}_{Priority}(\text{CHILDREN}(\tau))$
    if $|C| = 0$ **then return** LEAF(TOKEN(τ))
    $b \leftarrow \text{BINARIZE}(C_1)$ ▷ *head node*
    for $i \leftarrow 2, |C|$ **do**
        $b \leftarrow \text{NODE}(\text{RELATION}(C_i, \tau), \text{BINARIZE}(C_i), b)$
    return b

[9] See `git.liacs.nl/monotonicity-marking-from-parse-trees/alp2mono` for source code.

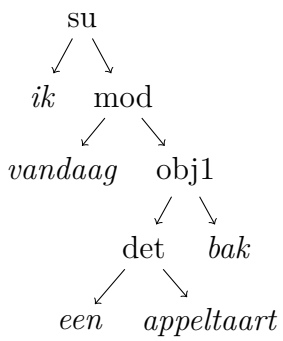
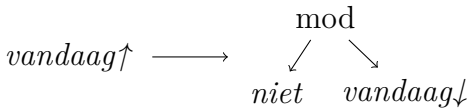


Figure 7: Binarized parse tree.

3.2 Polarization

Once we have our binary tree, we polarize it from the leaves up. Nodes are initially marked ↑, but at each internal node we apply some rules to determine whether its children should have the polarities of their nodes flipped, set to = or otherwise modified.

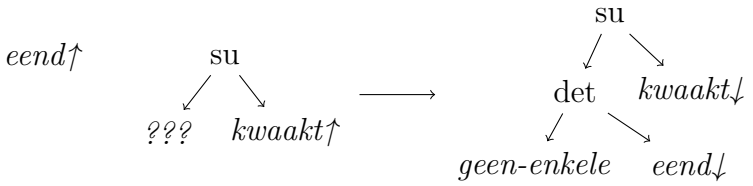
Modifiers Negation modifiers like "niet" or "nooit" flip polarity marks of their sibling trees:



"niet vandaag" ("not today")

UD includes a feature `Polarity=Neg` to identify negation modifiers, but Chen&Gao's models rely on an old version of Stanza which does not use this, so we have to maintain our own word list.

Determiners Determiners take two arguments: a noun and a predicate. Since Chen&Gao's algorithm provides no mechanism to pass the partially-applied function up the tree, we have to deviate from the usual top-down tree traversal and modify the parent node.

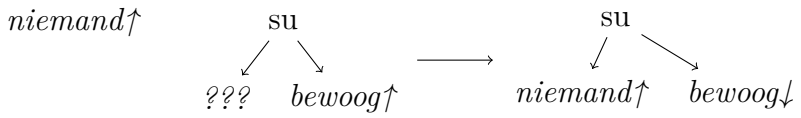


"Geen-enkele eend kwaakt" ("No duck quacks")

Monotonicities of various kinds of determiners are shown in table 1.

The UD features `PronType` and `Definite` can be used to differentiate between the determiner classes, but Chen&Gao's models rely on an old version of Stanza which does not use this for most determiners, so we have to maintain our own word lists.

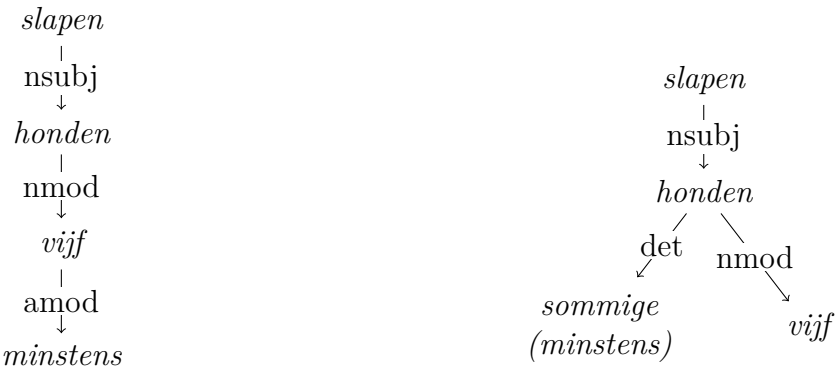
Since rules are only applied at the internal nodes, negative pronouns and other single-token noun phrases can't use the parent modification strategy. The subj rules need to check for such phrases.



"niemand bewoog" ("Nobody moved")

Udep2Mono deals with common multi-word determiners by replacing them with single words before parsing. For example, "geen enkele" ("not any") is replaced by "enkele". The det rule then checks the replaced text to determine the determiner's monotonicities. Numeral modifiers like "at least", "at most", "exactly" are also replaced with determiners, as in Fig. 8.

In Alpino2Mono, we leverage Alpino's determiner phrases. Some additional constructions are converted to determiner phrases for a more uniform analysis, as in Fig. 9. The hd of the **detp** determines the base monotonicity, which is then modified by any number of modifiers or additional determiners. This allows it to handle more complex determiners than Udep2Mono's approach, like "niet minder dan 5" ("not less than 5").

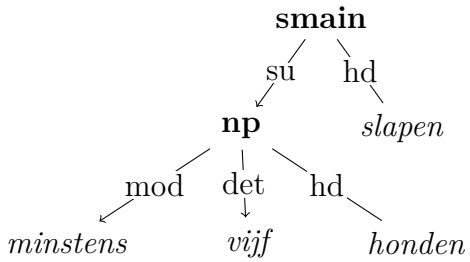


(a) Stanza parse

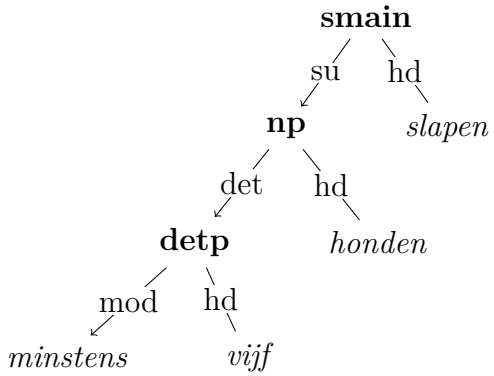
(b) Udep2Mono augmented parse

Figure 8: UD parse tree and Udep2Mono augmented tree for "Minstens vijf honden slapen" ("At-least five dogs sleep")

Superlatives Superlatives ("tallest", "smallest", etc.) are non-monotone.

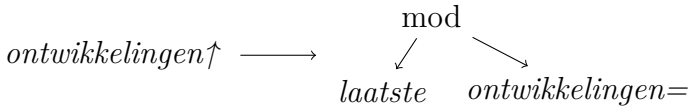


(a) Alpino parse tree



(b) Augmented Alpino parse tree

Figure 9: (Augmented) Alpino parse for "Minstens vijf honden slapen" ("At-least five dogs sleep"). The similar phrase "Meer dan vier honden slapen" would get parsed with a **detp**, so we convert cases like this to that form.



"[de] laatste ontwikkelingen" ("[the] latest developments")]

They are identified by the UD feature `Degree=Sup` or the Alpino feature `graad=sup`.

Conjunctions UD prefers content words as heads, so one of the conjuncts is the head of the conjunction. The coordinator is attached as a dependent of the other conjunct.

Alpino instead takes the coordinator as head. This gives equal treatment of the conjuncts, but does mean that in some edge cases the conjunction has no head: "zijn er verder nog vragen, opmerkingen?" [vSB24, 596].

Since our system does not implement unification of conjunct monotonicities yet, this difference does not have much effect.

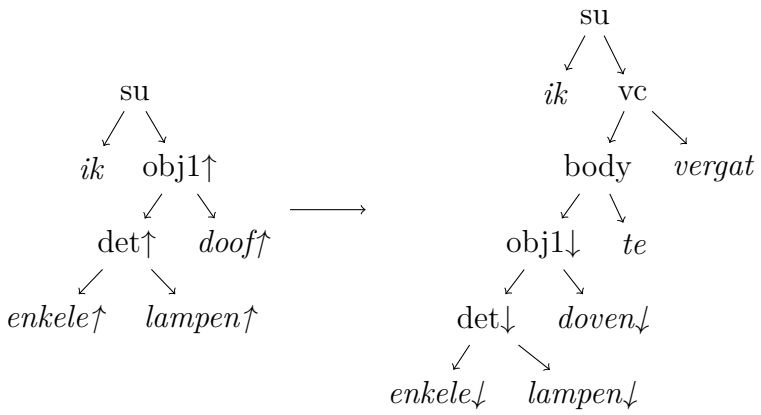
Conditionals In English conditional expressions can be easily identified by the presence of words like "if" and "whenever". In Dutch, however, "als" ("if") has multiple functions, and we need some heuristic to tell them apart. For example, compare

> "Als het regent komt hij altijd op het verkeerde moment" ("Whenever it rains, he arrives at the wrong time")
> "Als regen komt hij altijd op het verkeerde moment" ("Like rain, he always arrives at the wrong time)

In both cases Stanza parses "als" as mark of "regen(t)" which is advcl of the root. I have chosen quite an unfortunate example, because the heuristic we use assumes in the second example "regen"

would get dependency relation advmod, as it is not a clause, allowing us to disambiguate the uses of "als". With alpino, we look at the syntactic category of "als (het) regen(t)": in the first case it is a verbal category **smain**, indicating a conditional expression, whereas in the second case it is the nominal category **np**, indicating no conditional expression.

Implicatives For Alpino2Mono, we follow MacCartney's interpretation [Mac09, §6.3.4] of Nairn et al.'s implicatives inventory [NCK06]. It is likely we missed some common implicatives, as we translated `https://web.stanford.edu/group/csli_faust/Lexical_Resources/Polarity-Lexicon-of-Verbs/simple-implicatives/` to Dutch rather than composing our own list. Phrasal implicatives [Kar12] are not implemented.



"ik vergat enkele lampen te doven" ("I forgot to turn-off some lights")

Udep2Mono uses a different list of implicatives, which we translated for the Dutch version. Udep2Mono only checks the head verb (for example, it does not differentiate between "forget to" and "forget that", while Nairn et al don't consider the latter an implicative.). The word list also includes some questionable choices, leading to polarizations like the following:

"I↑ desinfect↑ a↓ wound↓" $\sqsubseteq$ "I desinfect a burn wound"

Ellipsis Though Alpino dependencies are able to represent elided elements as in Figure 10, the Alpino parser is unable to recognise this in most cases, leading to poor quality parse trees as in Figure 11. Here the parser was unable to establish the relation between "jij" and "je paraplu" because of the elision of "vergat te pakken", and has fallen back on the uninformative dp (Discourse Part) relation.

Basic UD does not represent elided elements. In case of predicate elision, (if there are no cop or aux) the most core argument is moved into the predicate's position, and all other arguments are attached to it as orphans. Figure 12b shows an example. While it is possible to reconstruct the elided relations in some cases, Udep2Mono does not attempt to do this, because the old Stanza models it works with often don't correctly parse sentences with ellipsis anyway. See for example Fig. 12a, where contrary to UD guidelines a predicate has two objs, and a nsubj is not attached to a predicate.

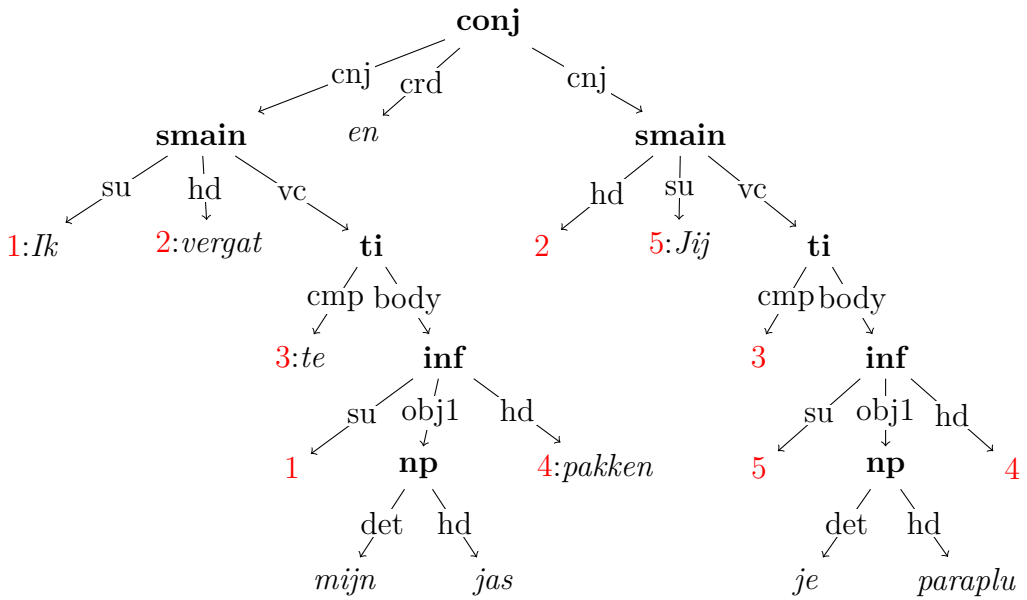


Figure 10: Ideal Alpino parse tree for "Ik vergat mijn jas te pakken en jij je paraplu" ("I forgot to get my jacket and you your umbrella"). Numbers without nodes represent connections to a node with that number.

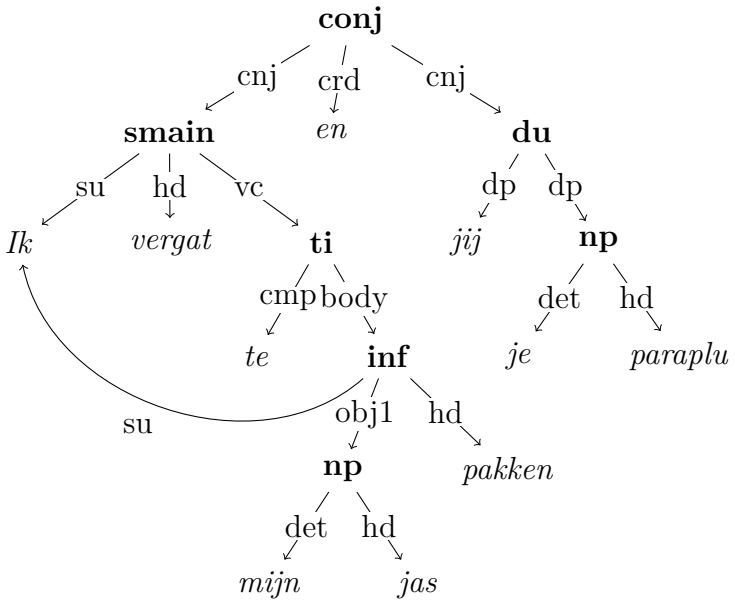
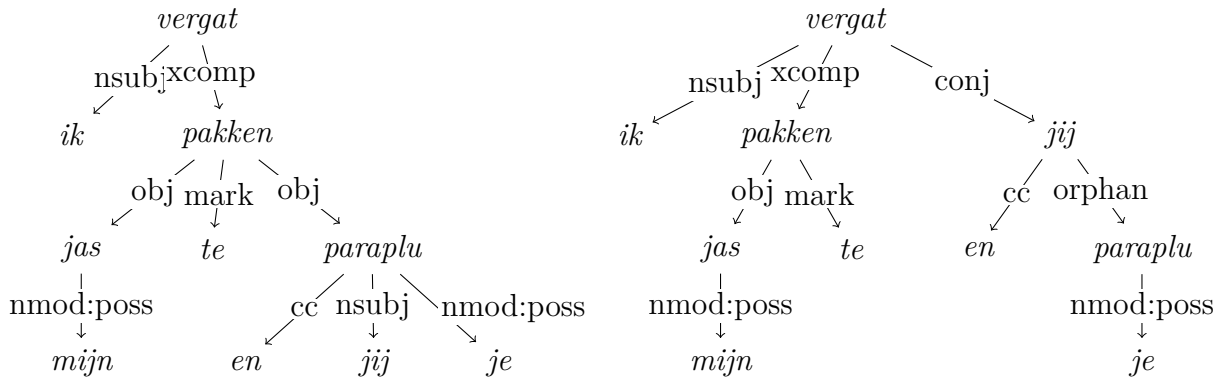


Figure 11: Actual Alpino parse tree for "Ik vergat mijn jas te pakken en jij je paraplu" ("I forgot to get my jacket and you your umbrella")



(a) Stanza 1.2 (incorrect)

(b) Stanza 1.6

Figure 12: UD parse trees for "Ik vergat mijn jas te pakken en jij je paraplu" ("I forgot to get my jacket and you your umbrella")

Bare numerals In the absence of numeral operators, the safest option is to mark numerals =:

"I↑ want↑ 4= apples↑"

But sometimes the intended reading is "at least" or "at most" rather than exact:

"I↑ bought↑ 4↓ apples↑" ⊑ "I bought 3 apples"

The current implementation recognises this for objects of a handful of verbs.

Comparatives Udep2Mono recognises a number of scalar comparatives, but only correctly polarizes the standard of comparison ("12 tennis balls"), not the entity being compared ("two standard footballs"):

Expected: "two↑ standard footballs are↑ heavier↑ than↑ 12↓ tennis balls"
Udep2Mono: "two= standard= footballs= are↑ heavier↑ than↑ 12↓ tennis= balls="

Alpino2Mono is currently limited to the special case "more/less than":

"Minder↑ honden↓ dan↑ katten↑ zitten="

4 Results

Table 3 shows accuracy on the small evaluation dataset used by Chen & Gao and originally from Hu & Moss [HM20]. This dataset was hand-crafted to cover various quantifiers in combination with conditionals and conjunctions. It does not specifically cover other interesting polarity-affecting phenomena like implicatives.

We translated the 56 test sentences included in the Udep2Mono repository to Dutch, preferring dependency structure preserving translations but choosing more natural translations where appropriate [10]. It turns out these are slightly different from the ones used for the results reported in

[10]The translations can be found at `git.liacs.nl/monotonicity-marking-from-parse-trees/Udep2Mono/-/`

lang	system	token-level all-tokens	sentence-level all-tokens	token-level key-tokens	sentence-level key-tokens
NL	Alpino2Mono	85.6	**58.9**	83.8	**62.5**
NL	Udep2Mono	**87.9**	55.4	**86.9**	**62.5**
EN	Udep2Mono	**96.5**	**87.5**	**96.5**	**89.2**
EN	CCG2Mono	76.0	44.6	78.2	50.0

Table 3: Accuracy on 56 (translated) gold-label test sentences used in Chen&Gao's paper. Key-token accuracy considers only marks on nouns, numbers, verbs, adjectives, and determiners. For English, this corresponds to the Penn POS tags CD, DT, JJ, NN, NNS, RB, RBR, RBS, VB, VBD, VBG, VBN, VBP, and VBZ. For Dutch the UD POS tags ADJ, ADV, DET, NOUN, NUM, and VERB. Results of Udep2Mono are taken from Chen&Gao's paper [CG21] and verified using the provided source code at `https://github.com/eric11eca/Udep2Mono`. Results of CCG2Mono are taken from Hu&Moss' second CCG2Mono paper [HM20].

Chen & Gao's paper [CG21], but the differences are small enough that the translations could be the same.

The direct translation of Udep2Mono from English to Dutch UD results in a substantial decrease in accuracy (Though still outperforming the pre-Udep2Mono state of the art). Unfortunately, our alternative Alpino-based polarity marker does not do much better. In the next sections we will look at the sources of the various polarity marking errors in detail.

4.1 Parsing Errors

The polarization system relies on correct dependency parses from Stanza or Alpino.

Because the English models are not compatible with newer versions, we used Stanza version 1.2 for both languages. In retrospect it would have been better to use a different version for Dutch, as many of the parsing errors we observed do not occur with newer Stanza versions; see Fig. 15 for an example.

The Alpino parser is of course also not perfect. While a bad Stanza parse usually means a single node gets the wrong dependency label, a bad Alpino parse often results in many nodes with uninformative dp (discourse part) labels or not attached to the tree at all. Figure 16 shows an example.

For Dutch UD, 5 sentences were polarized incorrectly due to parsing errors, accounting for up to[11] 7.1 percentage points of sentence level accuracy and 2 points of token level accuracy.

With the Alpino parser, only 3 sentences were parsed incorrectly, but the parsing errors were more severe, accounting for up to 2.6 points of token accuracy but only 5.4 points sentence accuracy.

`blob/nb/data/gold-nl/gold-nl.label.txt`. These are based on `data/gold/gold.label.txt` in the Udep2Mono repository. The labels used in Chen&Gao's paper are in the attachment to `https://aclanthology.org/2021.iwcs-1.12/`.

[11] 'up to', because this assumes the sentences would be polarized correctly had they been parsed correctly.

While the English UD parse sometimes deviates from UD guidelines, Chen & Gao's implementation works around these various deviations. While effective, it undermines the multilingual ability of UD and prevents upgrading to better parsers.

4.2 Labelling Disagreement

We disagree with a few of the labels in the English test set, but have nonetheless preserved these in the Dutch translation.

Intended reading of bare numerals

> "Five↑ dogs= see= at= least= six= cats="
> "Vijf↑ honden= zien= minstens= zes= katten="

It is not obvious to me why anything beyond "dogs" should be marked =, as it seems upward inference is possible: "Five dogs= see↑ at-least-six↑ cats↑" $\implies$ "Five dogs observe some mammals" Hu&Moss chose the "Exactly five" reading, whereas to me the "At least five" reading seems more natural.

"At least" operator in negative context

> "No↑ dentist↓ who↓ recommends↓ that↓ their↓ patients↓ brush↓ their↓ teeth↓ at↓ least↓ four↓ times↓ a↓ day↓ gave↓ five↓ patients↓ a↓ toothbrush↓"
> "Geen↑ enkele↓ tandarts↓ die↓ aanbeveelt↓ dat↓ haar↓ patiënten↓ hun↓ tanden↓ ten↓ minste↓ vier↓ keer↓ per↓ dag↓ poetsen↓ gaf↓ vijf↓ patiënten↓ een↓ tandenborstel↓"

It seems reasonable to me that "four" in "at least four times a day" should be marked ↑ rather than ↓. This is how it is labelled in a similar sentence: "Three= out↓ of↓ five= dentists↑ recommend↑ that↑ their↑ patients↑ brush↑ their↑ teeth↑ at↑ least↑ four↓ times↑ a↑ day↑" The current label would allow inferring "No dentist who recommends that their patients brush their teeth at least three times a day gave five patients a toothbrush" which does not seem to follow from the premise.

Multi-word quantifiers

> "Some↑ but↑ not↑ all↑ students↑ cheated↑ in↑ this↑ class="
> "Sommige↑ maar↑ niet↑ alle↑ studenten↑ speelden↑ vals↑ in↑ deze↑ les="

multi-word quantifiers should be internally polarized, as in this other label: "Not↑ every↓ student↑ likes↓ semantics↓". Hu&Moss' CCG2Mono collapses multi-word quantifiers into single tokens "some-but-not-all", so this how the original labels were formatted. Chen&Gao presumably missed this one when expanding the quantifiers again and did not notice because their system happened to make the same mistake.

4.3 Udep2Mono Failing to Generalize to Dutch

Verb sense The verbs "like" and "love" occur quite often in the test sentences. Udep2Mono calls these 'exactly-implicatives', as they suggest an 'Exactly' reading for bare numerals in their scope (both subject and object). The Dutch translation of both verbs would be "houden van".

"Three= dogs= love↑ four= cats="
"Drie= honden= houden↑ van↑ vier= katten="

Udep2Mono does not check case markers when determining verb properties, and 'houden' on its own is not an 'exactly-implicative', so it does not correctly polarize the Dutch translation above.

Alpino provides a convenient attribute `sense` which contains the verb stem with any associated auxiliaries and case markers, but in UD we can still just look at the dependents of any (i)obj and obl dependents to find case markers, it's just not something we implemented (yet).

Duration case marker English uses a duration case marker "for", as in "A↑ dog↑ [..] was↑ sick↑ for↑ three↓ days↓". In Dutch such a marker is usually omitted: "Een↑ hond↑ [..] was↑ drie↓ dagen↓ ziek↑". This makes it difficult to distinguish durations from other nominal modifiers, like "Een hond [..] was drie keer ziek", as both have identical dependency structures.

Parser-specific assumptions about binarized structure Udep2Mono sometimes has too much faith in its binarization process. In the following example,

"Few↑ math↓ students↓ love↓ any↓ subject↓ in↓ linguistics↓"
"Weinig↑ wiskunde↓ studenten↓ houden↓ van↓ enig↓ onderwerp↓ in↓ de↓ taalkunde="

In the Dutch UD parse "wiskunde" is considered a nominal modifier of "studenten", whereas in the English parse "math students" is considered a compound. Both interpretations seem valid, but Udep2Mono fails on the first.

"Weinig↑ wiskunde↓ studenten↓ houden↓ van↓ enig↓ onderwerp↓ in↓ de↓ taalkunde="

This happens because in the binarized tree the nmod gets placed between the det and its predicate, so when the det rule negates its parent, it actually just negates "wiskunde".

Reliance on non-UD behaviour I noted previously that Udep2Mono sometimes expects dependency trees which do not follow UD guidelines. For example, with the following sentences:

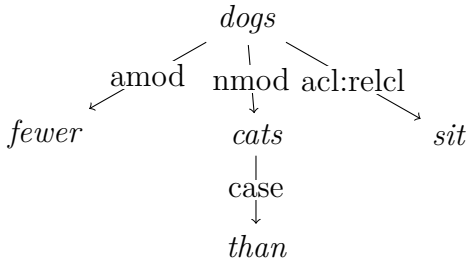
"More↑ dogs↑ than↑ cats↓ sit="
"Meer↑ honden↑ dan↑ katten↓ zitten="

For the English version Chen&Gao's models produce the parse shown in Fig. 13a. Contrary to UD guidelines, It takes "dogs" as the root rather than the verb "sit", which it attaches as a single-token relative clause. Stanza's default Dutch models produce the parse in Fig. 13b, which does conform to the guidelines.

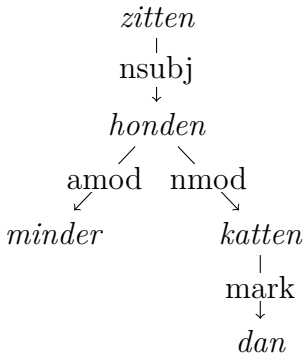
I don't know whether these odd parses are an intentional feature of Chen&Gao's custom models or just a strange idiosyncrasy they have worked around.

Unanticipated parsing variations

"If↑ you↓ are↓ addicted↓ to↓ cigarettes↓, you↑ can↑ smoke↑ two↓ a↑ day↑"
"Als↑ je↓ verslaafd↓ bent↓ aan↓ sigaretten↓, kun↑ je↑ er↑ twee↓ per↑ dag↑ roken↑"



(a) Dependency parse of "Fewer dogs than cats sit" with Chen&Gao's models



(b) Dependency parse of "Minder honden dan katten zitten" with Stanza v1.2 Dutch models

In the Dutch parse of the above, "er twee per dag" is considered an oblique argument rather than an object, evading Udep2Mono's mechanism for changing the interpretation of bare numerals under verbs like "smoke" to at-least.

4.4 Udep2Mono Features Not Implemented in Alpino2Mono

In our effort to re-implement Udep2Mono's features in a more robust way, without relying on brittle heuristics, many features did not get implemented at all.

Bare plurals Udep2Mono correctly polarizes the following, while Alpino2Mono marks it all ↑:

> "Cats↓ who↓ like↓ squirrels↓ hate↑ dogs↑"
> "Katten↓ die↓ houden↓ van↓ eekhoorns↓ haten↑ honden↑"

Udep2Mono succeeds because it negates relative clauses ("katten die houden van eekhoorns") if they are the first tokens of the sentence. This is an interesting heuristic for the "All" reading of bare plurals, but it is insufficient or incorrect in cases such as the following:

> "Cats are mammals"
> "Cats who like squirrels are uncommon"

Of course, it's easy criticizing others' work, but we did not come up with an alternative all-encompassing bare plural strategy either.

Bare numerals in clausal complements Udep2Mono has a list of 'willing verbs', which cause numerals and determiners in their scope to equalize their arguments (only the noun in case of numerals and both noun and predicate in case of other determiners). This allows it to correctly polarize the following, while Alpino2Mono fails:

> Label: "Er↑ is↑ een↑ hond↑ in↑ het↑ huis=, die↑ drie= koekjes= wil↑ eten↑"
> Alpino2Mono: "Er↑ is↑ een↑ hond↑ in↑ het↑ huis= , die↑ drie= koekjes↑ wil↑ eten↑"

We do have a somewhat related 'numeric base assumption' system for bare numerals, but that only looks at the closest verb head, "eten" in this case, not the entire stack ("is", "wil", "eten").

"All but n"

English: "All↑ but↑ five= dogs= died="
Dutch Label: "Alle↑ honden= op↑ vijf= na↑ stierven="
Alpino2Mono: "Alle↑ honden↓ op↓ vijf= na↓ stierven↑"

"All but five" forms a single quantifier, but in both UD and Alpino dependencies, the "but five" is considered a modifier of the noun phrase "All dogs", rather than of the determiner "All". Udep2Mono includes some dedicated logic just to deal with this special case. Alpino2Mono does not, in the expectation that at some point we would overhaul how we deal with higher-order functions so we can elegantly implement this for all kinds of modifiers. (Of course, that just means it will never get done.)

Implemenation oversight

English: "John↑ is↑ dancing↑ without↑ clothes↓"
Dutch Label: "John↑ danst↑ zonder↑ kleding↓"
Alpino2Mono: "John↑ danst↑ zonder↑ kleding↑"

"A without B" is downward monotone in B, but we did not think to implement it as such.

Conjunction coordinator polarity

English: "Ursula↑ refused↑ to↑ sing↓ or↑ dance↓"
Dutch Label: "Ursula↑ weigerde↑ te↑ zingen↓ of↑ dansen↓"
Alpino2Mono: "Ursula↑ weigerde↑ te↓ zingen↓ of↓ dansen↓"

Alpino2Mono just negates the entire subclause, while Udep2Mono correctly distributes across "or".

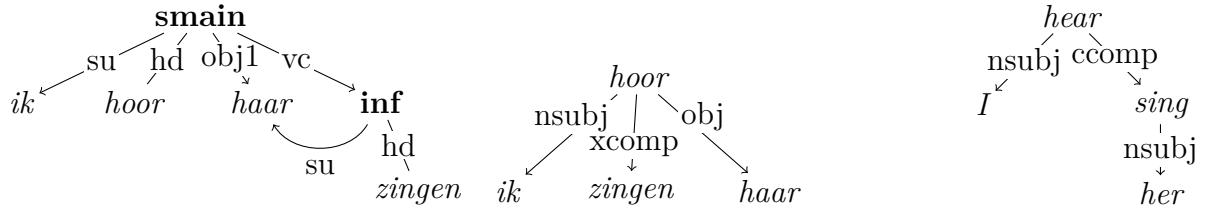
5 Comparison of Alpino and Universal Dependencies for Polarity Marking

We wanted to see whether Alpino dependencies offered an advantage over UD for polarity marking Dutch; No such advantage has been demonstrated.

Phrasal nodes Phrasal nodes are convenient to work with, but don't seem to provide any useful information which could not be extracted from UD annotations.

Co-indexed nodes for relative clauses The ability to refer back to nodes in the parent clause seems useful. However, in actuality cases where it provides an advantage over UD are rare. If the outer node is not the subject of the relative clause, it will not have any effect on polarity (in our semantic model). If it is subject of both clauses, in the current implementation predicate polarity flips due to the outer node will also be applied in the relative clause, so no special treatment is needed here either. If the node is not subject of the outer clause but is subject of the inner clause, the coindexing could be useful (Fig. 14a). However, in most such situations Stanza's Dutch UD parser would annotate the relative clause as xcomp (Fig. 14b), which strongly suggests that

the object of the outer clause is the subject of the inner clause, covering most common cases. I specifically specified the Dutch UD parser here, because in English UD parses the node would simply only be attached as subject of the inner clause (Fig. 14c), which already results in correct polarization (again, in our current model where objects have no effect on polarity).



(a) Alpino (b) Stanza Dutch UD (c) Stanza English UD

Figure 14: Parses for "Ik hoor haar zingen" ("I hear her sing").

Perhaps coindexing has some use in factive constructions, but we did not have time to implement those.

Co-indexed nodes for questions Alpino also uses co-indexing for referring back to question heads (who/what/where/etc.) as arguments of the question body. UD just skips this extra layer and only has body annotations. Question sentences can be easily recognised by the presence of a question mark, in nested clauses I don't think there's any benefit to treating question heads different from pronouns.

6 Discussion

We built a functioning polarity marking system for Dutch (two in fact), but performance is not on par with English yet. Even in this state, it may be of some use in natural language inference or data augmentation. CCG2Mono, for example, was used in creating the MED dataset while having worse accuracy on the dataset described in section 4. [12]

A few things are holding our system back: Parser accuracy tends to be lower for Dutch than English. Our semantic model makes many simplifying assumptions, and various features of the implementation are not backed by theory. Given the limited evaluation data, assessing the robustness of such ad-hoc features is difficult. But there's also more work to be done implementing such features, like support for factives, better rules for bare numerals and bare plurals, non-intersective adjectives, proportional quantifiers, scalar comparatives, etc:

Factives Earlier we saw implicatives allow inference in subclauses. Factives similarly allow inference in subclauses, but unlike implicative constructions they are not affected by the polarity of their context. An example will make this difference more clear.

[12] Though it is important to recognise the difference in sources of error. CCG2Mono is theoretically sound, errors come from bad parses or simplifying assumptions in the model. We also suffer the same problems, but in addition there are a lot of odd heuristic rules which need to be carefully balanced to cover as many relevant situations as possible while minimising false positives.

"Forget to" is an implicative. "I Forget to close the door" implies I did not close the door; "I did not forget to close the door" implies I did. It effectively acts as a monotone decreasing operator on subclauses.

"Forget that" is a factive. "I forgot that the door was closed" implies the door was closed; "I did not forget that the door was closed" *also* implies the door was closed. This polarity-ignoring behaviour does not fit neatly into the single-pass polarity-flipping strategy of Udep2Mono.

Bare Numerals As mentioned in section 3.2, we use some simple heuristics to determine whether numeral quantifiers without modifiers are treated as "at least" or "exactly". While a complete disambiguation of bare numerals might be out of reach, the current heuristics are rather ad-hoc, and it might be beneficial to replace them with a more theoretically grounded system. [Spe13].

Bare Plurals Another point of ambiguity are plurals without quantifiers. As we saw in 4.4, Udep2Mono's heuristic for disambiguating between the universal and existential readings of bare plurals is seriously flawed. Again, we should strive for a more robust system backed by theory. [Car77]

Non-Intersective Adjectives We assume all adjectives are intersective and therefore upward monotone, but this obviously does not hold for adjectives like "fake", "former".

Proportional Quantifiers The fraction operator / is downward monotone in its right argument, but implementing this for all its possible natural language renditions – "one in three", "one out of five", "two thirds", etc. – requires some care to avoid false positives.

Scalar Comparatives As mentioned in section 3.2, scalar comparatives like "two footballs are heavier than 12 tennis balls" are not (properly) implemented. The difficult part is determining the scope of the entity being compared.

7 Conclusions and Further Research

We translated Chen & Gao's polarity marking system Udep2Mono to Dutch and built an alternative system based on Alpino dependencies rather than Universal Dependencies. Unfortunately, we did not manage to leverage Alpino features to improve on the accuracy of the UD system. A number of improvements are yet to be made; if these can lead to a sufficient increase in accuracy, also beyond the small evaluation dataset used here, an obvious next step would be integrating the system in a natural language inference system like MonaLog [HCR$^+$20].

References

[Ber02] R. A. Bernardi. Reasoning with Polarity in Categorial Type Logic, October 2002.

[Bv17] Gosse Bouma and Gertjan van Noord. Increasing Return on Annotation Investment: The Automatic Construction of a Universal Dependency Treebank for Dutch.

In Marie-Catherine de Marneffe, Joakim Nivre, and Sebastian Schuster, editors, *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 19–26, Gothenburg, Sweden, May 2017. Association for Computational Linguistics.

[Car77] Greg N. Carlson. A Unified Analysis of the English Bare Plural. *Linguistics and Philosophy*, 1(3):413–456, 1977.

[CG21] Zeming Chen and Qiyue Gao. Monotonicity Marking from Universal Dependency Trees. In *Proceedings of the 14th International Conference on Computational Semantics (IWCS)*, pages 121–131, Groningen, The Netherlands (online), June 2021. Association for Computational Linguistics.

[CGM21] Zeming Chen, Qiyue Gao, and Lawrence S. Moss. NeuralLog: Natural Language Inference with Joint Neural and Logical Reasoning. In *Proceedings of *SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 78–88, Online, August 2021. Association for Computational Linguistics.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019.

[Dow94] David Dowty. The Role of Negative Polarity and Concord Marking in Natural Language Reasoning. *Semantics and Linguistic Theory*, 4:114, November 1994.

[HCR+20] Hai Hu, Qi Chen, Kyle Richardson, Atreyee Mukherjee, Lawrence Moss, and Sandra Kübler. MonaLog: A Lightweight System for Natural Language Inference Based on Monotonicity. *Proceedings of the Society for Computation in Linguistics*, 3(1):319–329, January 2020.

[HM18] Hai Hu and Larry Moss. Polarity Computations in Flexible Categorial Grammar. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 124–129, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[HM20] Hai Hu and Larry Moss. An Automatic Monotonicity Annotation Tool Based on CCG Trees. In *Second Tsinghua Interdisciplinary Workshop on Logic, Language, and Meaning: Monotonicity in Logic and Language*, 2020.

[IIM14] Thomas F. Icard III and Lawrence S. Moss. Recent Progress on Monotonicity. *Linguistic Issues in Language Technology*, 9, 2014.

[Kar12] Lauri Karttunen. Simple and Phrasal Implicatives. In Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret, editors, **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the Main Conference and the Shared Task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 124–131, Montréal, Canada, July 2012. Association for Computational Linguistics.

[LMV+18] José-de-Jesús Lavalle-Martínez, Manuel Montes-y-Gómez, Luis Villaseñor-Pineda, Héctor Jiménez-Salazar, and Ismael-Everardo Bárcenas-Patiño. Equivalences Among Polarity Algorithms. *Studia Logica*, 106(2):371–395, April 2018.

[LNT+23] Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji Zhou, and Yue Zhang. Evaluating the Logical Reasoning Ability of ChatGPT and GPT-4, May 2023.

[Mac09] Bill MacCartney. *Natural Language Inference*. PhD thesis, Stanford, 2009.

[MM08] Bill MacCartney and Christopher D. Manning. Modeling Semantic Containment and Exclusion in Natural Language Inference. In Donia Scott and Hans Uszkoreit, editors, *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 521–528, Manchester, UK, August 2008. Coling 2008 Organizing Committee.

[MM14] Bill MacCartney and Christopher D. Manning. Natural Logic and Natural Language Inference. In Harry Bunt, Johan Bos, and Stephen Pulman, editors, *Computing Meaning*, volume 47, pages 129–147. Springer Netherlands, Dordrecht, 2014.

[Mos12] Lawrence S. Moss. The Soundness of Internalized Polarity Marking. *Studia Logica*, 100(4):683–704, August 2012.

[NCK06] Rowan Nairn, Cleo Condoravdi, and Lauri Karttunen. Computing relative polarity for textual inference. In Johan Bos and Alexander Koller, editors, *Proceedings of the Fifth International Workshop on Inference in Computational Semantics (ICoS-5)*, 2006.

[NdG+16] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[Pet08a] Stanley Peters. Monotone Quantifiers. In Stanley Peters and Dag Westerståhl, editors, *Quantifiers in Language and Logic*, page 0. Oxford University Press, May 2008.

[Pet08b] Stanley Peters. Type ⟨1⟩ Quantifiers of Natural and Logical Languages. In Stanley Peters and Dag Westerståhl, editors, *Quantifiers in Language and Logic*, page 0. Oxford University Press, May 2008.

[QZZ+20] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages, April 2020.

[RTC+16] Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140, April 2016.

[San91] Victor Sanchez. *Studies on Natural Logic and Categorial Grammar | Institute for Logic, Language and Computation*. PhD thesis, Amsterdam, 1991.

[Spe13] Benjamin Spector. Bare Numerals and Scalar Implicatures. *Language and Linguistics Compass*, 7(5):273–294, 2013.

[TC19] Aarne Talman and Stergios Chatzikyriakidis. Testing the Generalization Power of Neural Network Models across NLI Benchmarks. In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors, *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 85–94, Florence, Italy, August 2019. Association for Computational Linguistics.

[TNH02] Mariët Theune, Anton Nijholt, and Hendri Hondorp. Computational Linguistics in the Netherlands 2001: Selected Papers from the Twelfth CLIN Meeting. BRILL, January 2002.

[van07] Jan van Eijck. Natural Logic for Natural Language. In Balder D. ten Cate and Henk W. Zeevat, editors, *Logic, Language, and Computation*, pages 216–230, Berlin, Heidelberg, 2007. Springer.

[VB86] Johan Van Benthem. *Essays in Logical Semantics*, volume 29 of *Studies in Linguistics and Philosophy*. Springer Netherlands, Dordrecht, 1986.

[vdBBMvN02] Leonoor van der Beek, Gosse Bouma, Rob Malouf, and Gertjan van Noord. The Alpino Dependency Treebank. In *Computational Linguistics in the Netherlands 2001*, pages 8–22. Brill, January 2002.

[vSB24] Gertjan van Noord, Ineke Schuurman, and Gosse Bouma. Lassy syntactic annotation manual, April 2024.

[Wij23] Gijs Wijnholds. Assessing Monotonicity Reasoning in Dutch through Natural Language Inference. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1494–1500, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

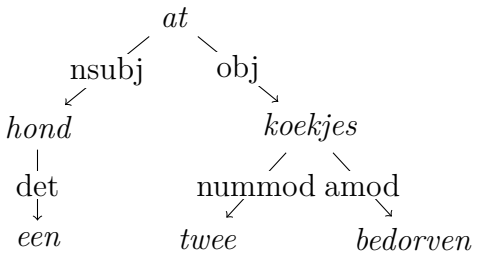
[YMB+19a] Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. Can Neural Networks Understand Monotonicity Reasoning? In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors, *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 31–40, Florence, Italy, August 2019. Association for Computational Linguistics.

[YMB+19b] Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. HELP: A Dataset for Identifying Shortcomings of Neural Models in Monotonicity Reasoning. In Rada Mihalcea, Ekaterina Shutova, Lun-Wei Ku, Kilian Evang, and Soujanya Poria, editors, *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019)*, pages 250–255, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

CP-mod
se
pobj1
su
sup
pc
obcomp
det
mod
predc
predm
me
ld
vc
obj1
obj2
app
cnj
svp
mwp
dp
sat
tag
body
crd
cmp
whd
rhd
hdf
hd
dlink
nucl
top

Table 4: Binarization hierarchy. Children with relation labels higher in the list are placed higher in the binarized tree.

(a) In Stanza 1.2 "at 2" ("ate 2") is erroneously considered a fixed expression, because "at" is not recognised as the past tense of "eten" by the POS-tagger



(b) Stanza 1.6 correctly parses the sentence.

Figure 15: UD parses for "Een hond at twee bedorven koekjes" ("A dog ate two rotten biscuits") with different versions of default Stanza models.

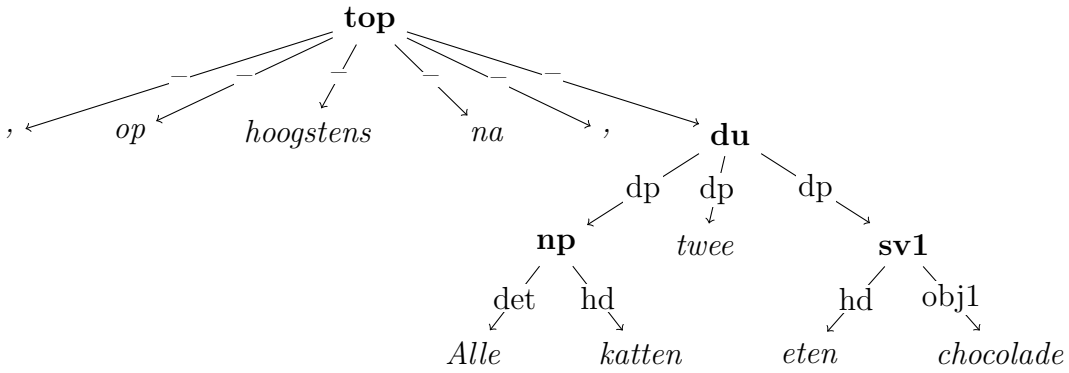


Figure 16: Failed Alpino parse of the sentence "Alle katten, op hoogstens twee na, eten chocolade" (usually **top** is not shown, as it only has one non-punctuation child)