university of groningen

faculty of arts

Language-neutral Semantic Parsing using Graph Transformations on Universal Dependencies

Wessel Poelman

Primary supervisor: prof. dr. Johan Bos
Secondary supervisor: dr. Gosse Bouma

Master thesis
Information Science
Wessel Poelman
s2976129
July 22, 2022

ABSTRACT

Current trends in semantic parsing primarily use large, pre-trained neural language models. These models achieve impressive scores but also present some drawbacks. They require vast amounts of training data, cross-lingual performance is often sub-optimal and their performance usually decreases for longer input sequences. Additionally, the output of these models lacks explainability: why are resulting meaning representations composed the way they are? This becomes particularly relevant when a model produces strange output or when it makes mistakes.

This thesis outlines an approach to semantic parsing that uses transparent and explainable graph transformations. We apply these transformations to a Universal Dependencies (UD) parse tree and primarily target language-neutral features. After these structural transformations, we substitute syntactic labels with semantic concepts to compose our final meaning representation. We target Discourse Representation Structures as our semantic formalism. Our system[1] is developed and evaluated using English, Dutch, Italian and German data from the Parallel Meaning Bank project.

Recent developments regarding Discourse Representation Structure notations have shown that, while not originally intended to be graphs, they can be represented as simple, directed acyclic graphs. These simple graphs already somewhat resemble UD parse trees. This is our main motivation for using graph transformations. We test our method on output from two 'off-the-shelf' state-of-the-art UD parsers: Stanza and Trankit.

The main question we want to answer is how our approach compares to a fully neural sequence-to-sequence semantic parser. In addition, we formulate three sub-questions: to what extent is our approach language-neutral, how does our approach deal with input sequences of various lengths and how does our approach handle negation and quantifier constructions.

We compare our system to an English neural system. Our approach performs similarly to the neural method when there is not an abundance of data available to train the neural model. On the evaluation dataset, our system manages a macro F1-score and error rate of 81.5 (0.5% error). The neural model achieves 81.7 (3.9%). When using a more strict evaluation method regarding ill-formed graphs, the neural model drops to 78.4 (8.3%) and our results stay the same. Additionally, our approach outperforms this neural model on longer input sequences.

When there is a lot of training data available for the neural method, our approach lacks behind in scoring. This model achieves a score and error rate of 91.8 (2.8%). Using the strict evaluation, it manages 90.5 (3.7%). Our method does keep up with this model on longer input sequences and produces less ill-formed output.

Our approach is almost fully language-neutral. Some constructions (e.g., negation) are not covered by language-neutral UD information and need language-specific features. These are, however, simple and only consist of a list of ten to twenty words per language. Our negation detection works well. The same is true for quantifier detection, although recall could be improved there.

Our approach is a transparent, almost language-neutral semantic parser that shows good performance on four languages, all achieving F1-scores of >75.0, even with little training data. Our approach currently does not assign proper scope to negation and quantifiers. This is an area of improvement, in addition to better handling of named entities, correctly parsing date and numerical expressions and supporting more complex constructions. In the future, these can be improved by leveraging external resources and systems that go beyond just UD.

[1] System source is available at: `https://github.com/WPoelman/ud-boxer`.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AMR	Abstract Meaning Representation
CCG	Combinatorial Categorical Grammar
DAG	Directed Acyclic Graph
DRG	Discourse Representation Graph
DRS	Discourse Representation Structure
DRT	Discourse Representation Theory
GMB	Groningen Meaning Bank
MRP	Meaning Representation Parsing
NAP	Negative Application Pattern
NER	Named Entity Recognition
NLP	Natural Language Processing
NLU	Natural Language Understanding
PMB	Parallel Meaning Bank
POS	Part of Speech
SBN	Simplified Box Notation
UCCA	Universal Conceptual Cognitive Annotation
UD	Universal Dependencies
UPOS	Universal Part of Speech

PREFACE

About half a year ago, at the time of writing, the faculty from Information Science presented their thesis topics for students to choose from. Professor Johan Bos presented a single slide with a short talk outlining his idea. One sentence from this talk stood out to me:

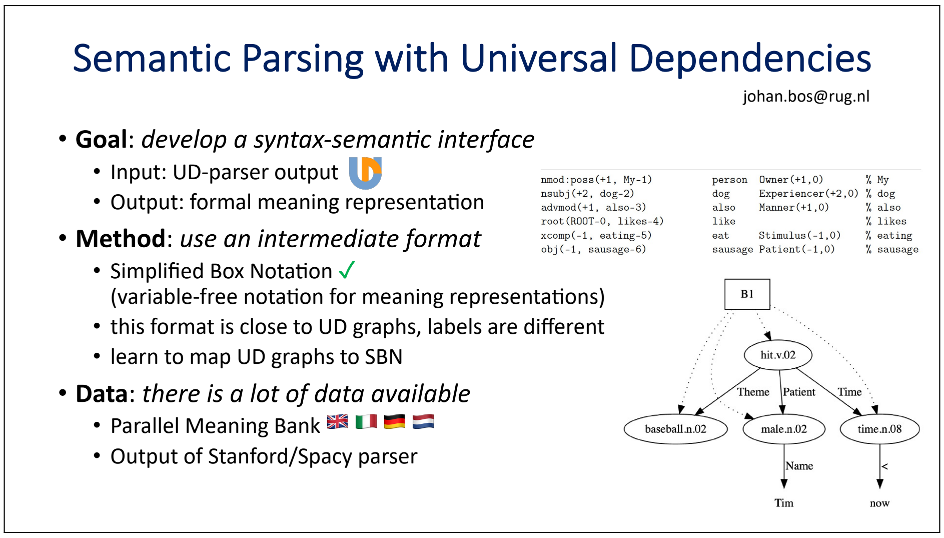
"You have to enjoy creating a complex system and hacking stuff together."

This is something I have enjoyed for a couple of years, ever since I chose to pursue Information Science. To this day, I still enjoy this a lot. So naturally, I chose this thesis topic. In addition, it also seemed like an interesting challenge.

I want to thank my supervisor prof. dr. Johan Bos for his great help, dr. Rik van Noord for helping with the neural comparison system, Bruno Guillaume for answering questions I had regarding GREW and my mother for proofreading.

Lastly, I am grateful for the opportunity to have written a paper about my system for COLING 2022 (currently awaiting a review).

For completeness' sake, the single slide that started this project:



Semantic Parsing with Universal Dependencies

johan.bos@rug.nl

- **Goal**: *develop a syntax-semantic interface*
  - Input: UD-parser output
  - Output: formal meaning representation
- **Method**: *use an intermediate format*
  - Simplified Box Notation ✓
    (variable-free notation for meaning representations)
  - this format is close to UD graphs, labels are different
  - learn to map UD graphs to SBN
- **Data**: *there is a lot of data available*
  - Parallel Meaning Bank
  - Output of Stanford/Spacy parser

```
nmod:poss(+1, My-1)          person  Owner(+1,0)          % My
nsubj(+2, dog-2)             dog     Experiencer(+2,0)    % dog
advmod(+1, also-3)           also    Manner(+1,0)         % also
root(ROOT-0, likes-4)        like                         % likes
xcomp(-1, eating-5)          eat     Stimulus(-1,0)       % eating
obj(-1, sausage-6)           sausage Patient(-1,0)        % sausage
```

1 INTRODUCTION

When diving into the literature surrounding semantic parsing, one cannot avoid a sentence like the following:

> *"Semantic parsing is the task of mapping a natural language expression to a machine-interpretable meaning representation."*

These sentences are often the first or second in a given paper or book. Numerous variations exist, but they all convey the same idea. This idea also encompasses the goal and task of the current project. Of course, for experts in the field, this alone is enough to convey what task is being tackled, but there is a lot to unpack in this relatively short sentence. So, to start, let us take a step back and outline what *semantic parsing* is in the context of the current project.

We start with a *natural language expression*, this can be a word, a sentence, multiple sentences, even a book and so on. In *semantic parsing* this expression is usually a single sentence or a small number (say, < 10) of related sentences. We can analyze and decompose these expressions from several linguistic perspectives that build on top of each other (Jurafsky and Martin, 2009):

- *Morphology* - Knowledge of the meaningful components of words.
- *Syntax* - Knowledge of the structural relationships between words.
- *Semantics* - Knowledge of meaning.
- *Pragmatics* - Knowledge of the relationship of meaning to the goals and intentions of the speaker.
- *Discourse* - Knowledge about linguistic units larger than a single utterance.

Semantic parsing aims to create a representation of the meaning of an expression. To do this, we can (often need to) use morphological, syntactic and possibly 'external' information (such as a knowledge base). We can construct this information in such a way that it is *unambiguous*, has a *canonical form* and can be *reasoned* with.

A famous example to illustrate ambiguity is the sentence: *"I made her duck"* (Jurafsky and Martin, 2009). There are numerous interpretations possible, *I cooked duck for her*, *I made her quickly bow her head*, *I crafted her toy duck* and so on.

Let's assign some simple syntactic information to differentiate these interpretations:

I	*made*	*her*	*duck*
PRON	VERB	PRON	NOUN
PRON	VERB	PRON	VERB

This helps a bit, but we are still left with an ambiguous sentence. *Made as in cook* and *made as in crafted* both refer to verbs. *Duck as in food* and *duck as in (toy) animal* both refer to nouns. To make these words unambiguous, we can use an external lexical resource like WordNet[1] (Miller, 1994; Fellbaum, 1998) to accurately define which interpretation (sense) of a given word we mean:

[1] Other lexical resources that serve similar purposes exist as well. WordNet is chosen since it is also used in the rest of the project.

- `make.v.06` - Create or manufacture a man-made product.
- `make.v.39` - Prepare for eating by applying heat.
- `duck.n.01` - Small wild or domesticated web-footed broad-billed swimming bird usually having a depressed body and short legs.
- `duck.n.02` - Flesh of a duck (domestic or wild).

A *synset* (sometimes referred to as a *concept*) of a word is made up of its lemma, a simple part of speech tag and a sense number that indicates a particular interpretation. The use of a lemma, instead of an inflected form, also relates to the need for a canonical form. In semantic parsing, tense is often modeled separately, not by using verb inflections.

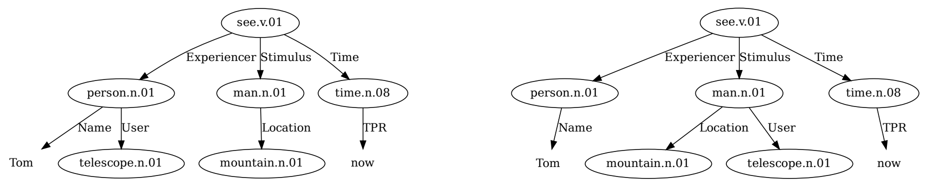
We can now express our interpretation of the sentence in a much more unambiguous manner, but we are currently just dealing with individual words. For a simple sentence like this, that might be sufficient, but for more complex sentences, we also need to disambiguate the relations between words. Consider the following sentence: "*Tom saw the man on the mountain with a telescope*".

We can disambiguate the individual words, but we are still left with an ambiguous sentence. Who has the telescope? Did Tom look through the telescope? Who is standing on the mountain?

To solve this problem we can use *thematic roles*. These roles indicate relations between words or multiple words. There are various lexical resources that provide such roles, such as PropBank (Kingsbury and Palmer, 2002), FrameNet (Ruppenhofer et al., 2006) and VerbNet (Kipper et al., 2008). The meaning representation we use in this project uses the latter, so let's do that here as well. To disambiguate who used the telescope, we connect the telescope to its user with the *User* role. We can also connect the person standing on the mountain with a *Location* role. It is important to note that these connections should be *directed*, otherwise we are still dealing with a possibly ambiguous representation.

Let's introduce a time node, also with an unambiguous synset, that indicates when the event occurred. For this, we need to introduce additional (logical) operators, such as (in)equality and temporal relations. We can compare the time of the event with *now* and indicate how it relates to it. Our example sentence describes an event in the past, so we can use *TPR* (preceding).

If we combine all of this, we have a simple, yet rather expressive meaning representation. Figure 1 shows what we can convey with this representation:



(a) *Tom, using a telescope, saw the man on the mountain.*

(b) *Tom saw the man, who was on the mountain, using a telescope.*

Figure 1: Basic meaning representation examples.

We are lacking some features to express more complex constructions (Abend and Rappoport (2017) give a good overview of what components most meaning representations provide), but this is a good start.

We now have a global overview of what we want to accomplish with semantic parsing and what general components are involved. This brings us to the final part of our requirements for semantic parsing, being able to *reason* with the generated meaning representations. These representations can be used for many Natural Language Processing (NLP) tasks, in particular *Natural Language Understanding* (NLU). One downstream task could be *event extraction* (Schuster et al., 2017), where a meaning representation can be used to show what events occur in a given sentence, the actors involved and how they relate to each other. Another use might be to target a query language as the required meaning representation (Li et al., 2020). This representation can then be used to query a knowledge base to answer questions for instance.

Application to a particular downstream task is outside of the scope of the current project. We purely focus on creating the meaning representations. Of course, we would like to have a system that can do this automatically, where it produces a likely representation of a given expression.

Semantic parsing has a long and varied history with many different methods and formalisms. A quote from Evang (2019) summarizes this well:

> *"A spectrum is haunting semantic parsing, the spectrum ranging from traditional semantic grammars on one end to recent sequence-to-sequence methods on the other."*

Current state-of-the-art results in semantic parsing all come from neural models or hybrid neural models (van Noord et al., 2020; Zhou et al., 2021; Bevilacqua et al., 2021; Bai et al., 2022). These models achieve very impressive scores, but this performance comes at a cost. Most of these models are based on pre-trained language models with billions of parameters. These are practically impossible to explain and often require vast amounts of training data. In recent years, explainability of machine learning and artificial intelligence has gained a lot of traction since these techniques are being used in more and more areas. In semantic parsing, we are interested in *meaning*. The methods and processes by which we get and represent this meaning are crucial components. Especially when it comes to errors in a meaning representation. It is very difficult to determine why such a huge neural model made a mistake. Was it due to errors or bias in the training data (either in pre-training or finetuning)? Does the model make wrong assumptions? Is the model not understanding something? High scores are just one aspect of a larger picture, explainability and performance in situations with little training data are arguably just as important.

The current project proposes a system that uses (symbolic) transparent graph transformations applied to output generated by neural models to produce its meaning representations. While it is not a fully integrated model, the approach could be characterized as *neuro-symbolic* (Li et al., 2020; Susskind et al., 2021). These graph transformations are applied to a *Universal Dependencies* (de Marneffe et al., 2021) dependency parse tree and primarily target language-neutral features. We test our system on English, Dutch, Italian and German, respectively. We go into more detail regarding all these points in the next sections. The project aims to answer the following main research question:

> *How do Discourse Representation Structures derived from Universal Dependencies using a graph transformation approach compare to those created by a fully neural sequence-to-sequence model?*

We formulate three sub-questions regarding the evaluation of the systems. Our first sub-question:

> RQ1: *To what extent would such an approach be language-neutral or easily transferrable to multiple languages?*

One of the key characteristics of Universal Dependencies is the large number of language-neutral features it can express. These include morphological, syntactic and dependency-related features. The prospect of leveraging this information for semantic parsing is promising. Language-neutral semantic parsing with such a well-established framework could be very helpful. Little or no special treatment would be needed for adding new languages on the semantic parser side, only on the UD side. This is what we want to examine with RQ1.

Our second sub-question:

> RQ2: *How well does such an approach deal with input sequences of various lengths?*

State-of-the-art neural language models are somewhat notorious for their drop-off in performance for long input sequences (Press et al., 2021). Recent research in that field is focused on improving this, but it is still a shortcoming of numerous popular large pre-trained language models. As we have mentioned, semantic parsing in the current project is focused on rather short sentences. Still, it is beneficial to take a look at how our approach and a neural approach perform on input sequences of various lengths. This can help us find possible areas of improvement for a particular system. We are also interested in seeing if our graph transformation approach suffers from the same performance drop-off as the neural models. We want to explore this aspect in answering RQ2.

Finally, our third sub-question:

> RQ3: *How well does such an approach handle negation and quantifiers constructions?*

Negation is a crucial phenomenon in semantic parsing. If one wants to reason with a semantic meaning representation, truth values are essential. Within semantic parsing, negation detection and assigning scope to negation are two entire research areas on their own (Morante and Blanco, 2012). In addition to a single negation clause, such as in *Tom is not doing that right now*, multiple negation clauses are commonly used to model *quantifiers*. If we want to create a meaning representation for the sentence *Everybody is doing that*, we can encode it as *There is not somebody who is not doing that*. Semantically, this conveys the same meaning and we need two negation clauses to encode this. Since these are such essential phenomena, we want to examine how our approach and the neural systems handle these.

The next chapter covers the meaning representation we target in this project. We give a brief overview of the Universal Dependencies framework. We provide background information on *graphs* and how this format plays a crucial role in our approach. We cover related work regarding semantic parsers and introduce the dataset and evaluation method used in the project. We outline our method, evaluate it and compare it with a fully neural parser. We discuss our results and provide an error analysis. Lastly, we answer our research questions, list our final conclusions and suggest directions for future work.

2 BACKGROUND AND RELATED WORK

Semantic parsing is a broad field with many formalisms, methods and approaches. In this chapter, we cover our target meaning representation (Discourse Representation Structures) as well as our input (Universal Dependencies). We give an overview of how Universal Dependencies have been used in semantic parsing. We discuss several implementations of semantic parsers. We also outline some of the differences between the methods these parsers implement. With this, we show how our method fits into the existing landscape of semantic parsers and in what ways it provides novel approaches.

2.1 Discourse Representation Structures

The basic meaning representation from Chapter 1 is a stripped-down and graph-like version of Discourse Representation Structures (DRSs)[1]. This formalism stems from Discourse Representation Theory (DRT), originally developed by Kamp (1981). DRT has seen many iterations and improvements over the years. DRSs are complex structures that can contain discourse referents, logical operators, DRSs embedded in DRSs and more. Traditionally, DRSs are represented using *boxes* that indicate what referents and events belong to the same *scope* (box). Figure 2 shows what our examples from Figure 1 look like in box notation:

```
x e y z l t
--------------------
person.n.01(x)
  Name(x,"Tom")
  User(x,z)
see.v.01(e)
  Experiencer(e,x)
  Stimulus(e,y)
telescope.n.01(z)
mountain.n.01(l)
man.n.01(y)
  Location(y,l)
time.n.08(t)
  t ≺ now
```

(a) *Tom, using a telescope, saw the man on the mountain.*

```
x e y z l t
--------------------
person.n.01(x)
  Name(x,"Tom")
see.v.01(e)
  Experiencer(e,x)
  Stimulus(e,y)
telescope.n.01(z)
mountain.n.01(l)
man.n.01(y)
  User(y,z)
  Location(y,l)
time.n.08(t)
  t ≺ now
```

(b) *Tom saw the man, who was on the mountain, using a telescope.*

Figure 2: Box notation of our example representations from Figure 1. Note that the notation style is not classic DRT. Instead, it is in the style that is used in the Parallel Meaning Bank project.

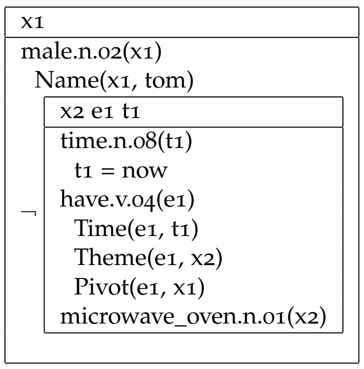
Note that in DRT, WordNet synsets are not necessarily used, we incorporate these since the current project uses the Parallel Meaning Bank (PMB, Abzianidze et al. 2017) as its dataset. In there, non-logical symbols in DRSs are expressed using WordNet synsets. We discuss the dataset in more detail in Chapter 3.

The box notation in Figure 2 is quite different from the graph-like notation from Figure 1. DRSs are in fact not strictly graphs, however, recent work on DRS notations has shown that they can be represented as simple graphs without losing expressive power (Abzianidze et al., 2020; Oepen et al., 2020; Bos, 2021). This graph

[1] When referring to DRSs, we specifically mean DRSs with neo-Davidsonian event semantics.

representation of DRSs lies at the core of the current project. We discuss this more in Section 2.3.

Several notations for DRSs exist. Three are available in the PMB: the *box notation*, the *clause notation* and the *Simplified Box Notation* (SBN). Figure 3 shows all three notations for the sentence *Tom doesn't have a microwave oven.*



(a) Box Notation.

```
male.n.02 Name "Tom"
NEGATION -1
time.n.08 EQU now
have.v.04 Pivot -2 Time -1 Theme +1
microwave_oven.n.01
```

(b) Simplified Box Notation.

b1 REF **x1**	**b2** NEGATION **b3**
b1 Name **x1** "tom"	**b3** REF **e1**
b1 PRESUPPOSITION **b2**	**b3** Pivot **e1** **x1**
b1 male "n.02" **x1**	**b3** Theme **e1** **x2**
b2 REF **t1**	**b3** have "v.04" **e1**
b2 EQU **t1** "now"	**b3** REF **x2**
b2 time "n.08" **t1**	**b3** microwave_oven "n.01" **x2**
b3 Time **e1** **t1**	

(c) Clause Notation.

Figure 3: DRS notations in the PMB for the sentence *Tom doesn't have a microwave oven.*

These notations are all logically equivalent and useful for different applications. They can be transformed from one to the other without losing information (Bos, 2021). We focus on SBN for the current project.

This notation, developed by Bos (2021), shows that DRSs can represented as simple, *Directed Acyclic Graphs* (DAGs). SBN models triples by introducing a node, the label of the outgoing edge and another node *or* an index to a node defined previously or that will be defined later on. The characteristics of the nodes and edges in SBN are shown in Table 1. A central ingredient of this notation is its lack of variables, which makes processing it much simpler compared to notations with variables. The graph that SBN models is a type of *Discourse Representation Graph* (DRG). Previous iterations of DRGs were a lot more complicated. Examples include formats from Abzianidze et al. (2020) or Oepen et al. (2020). They used rather verbose notations where either variables were explicitly modeled in the graph or edges were represented as nodes. SBN is quite a bit simpler in that regard and, as a result, already resembles a Universal Dependencies tree much more than the other notations.

Note that in the DRG that SBN models, nodes do not directly correspond to words or tokens in a sentence. Compound words that have a WordNet entry should be represented as their 'combined' entry. An example is the `microwave_oven.n.01` synset that we have seen. Similarly, multi-word expressions, such as names, can also be represented by a single *constant* node. Another important thing to note is that box nodes by themselves do not represent anything meaningful. Outgoing edges to synset nodes provide the meaning they should convey, namely to assign *scope*. The meaning of multiple boxes is represented by box-to-box edges. **b1** `NEGATION` **b2**, for instance, indicates that the content (nodes in its scope) of **b2** is negated and that this relates to the content of **b1**. Negation is an important aspect in this type

Table 1: SBN DRG characteristics, note that boxes and regular box connections are unlabeled.

	Description	**Examples**	**Can have child nodes**
Node types	Synset (S)	person.n.01, have.v.04	yes
	Constant (C)	now, "John Doe", '2022'	no
	Box (B)	-	yes
	Description	**Examples**	**Node connections**
Edge types	Thematic role	Agent, Time, AttributeOf	S → S, S → C
	DRS operator	EQU, NEQ, TPR	S → S, S → C
	Box connection	-	B → S
	Box-to-box connection	NEGATION, RESULT	B → B

of graph since it is also used to model certain *quantifiers*. If we want to express the sentence *Everyone's eating*, we have to encode this with two `NEGATION` edges and three boxes. Essentially this is stating that *There is not someone who is not eating*, which, semantically speaking, is the same as the original sentence. We discuss negation in more detail in Section 2.7. Figure 4 shows how we visualize DRGs in this thesis and how this particular example is encoded. This style of visualizing DRGs is very similar to the one used in Bos (2021). The dotted edges are the box connections that assign scope.

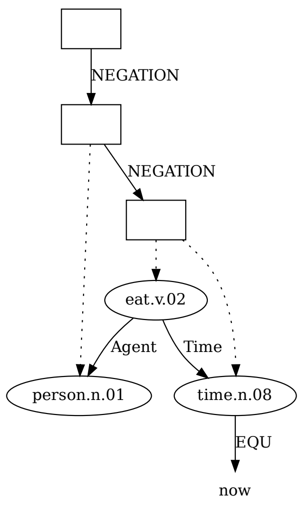


Figure 4: Example of a quantifier in a DRG for the sentence *Everyone's eating*.

There are more intricacies to discuss regarding SBN and DRGs. A lot of these are covered in Chapter 4 when we describe how our approach works. The last important aspect we cover here is encoding multiple sentences in the same DRG. DRSs are not restricted to a single sentence, unlike a lot of other meaning representation formats. We can simply introduce another box *next to* a previous box and indicate how they relate to each other. In a DRG, this works exactly the same and since the graph is *directed*, we know the ordering of the sentences as well. Figure 5 shows what this looks like. There are several box-to-box edge labels that can be used to connect multiple DRGs. We go into more detail regarding these in Chapter 4.

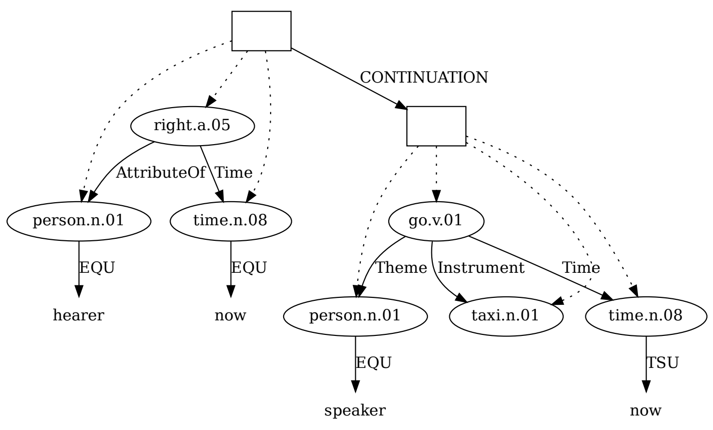


Figure 5: Example of multiple sentences in a DRG for *You're right. I'll go by taxi.*

2.2 Universal Dependencies

To create a semantic meaning representation, there are several options to choose from when representing the input natural language expression. There are systems that work directly with the *surface form* (i.e., the raw text itself), others use a *syntactic parse* or additional linguistic representations and there are systems that combine multiple formats. We go into more detail regarding specific semantic parsers in Section 2.4.

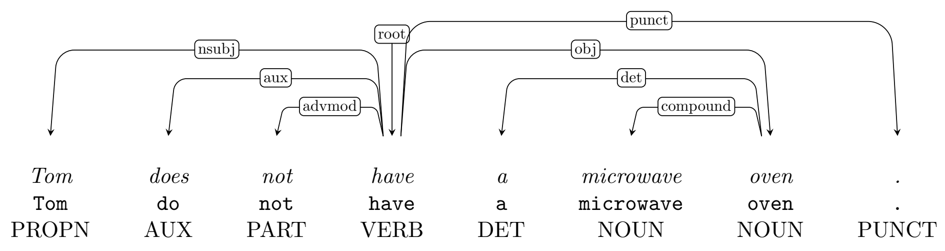
The current project uses a rich syntactic parse from the *Universal Dependencies* (UD, de Marneffe et al. 2021) project[2]. This framework provides tools to consistently annotate parts of speech, morphological features and syntactic dependencies, across different human languages. At the time of writing, it has treebanks available for more than 100 languages.

A UD parse can provide a lot of information. We discuss what our approach uses in detail in Chapter 4. Figure 6a shows how UD parses are commonly visualized and Figure 6b shows how we can format this differently to more resemble our DRG visualizations. This is purely a stylistic choice and we use the latter from now on. Let us take a look at three core components of a UD parse: the *lemma* and *Universal Part of Speech (UPOS) tag* per token, as well as the *dependency relations* between tokens.

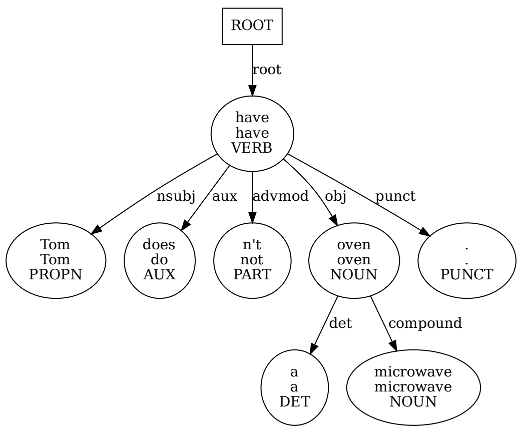
With these components, we can already derive some basic semantic information for the example in Figure 6. We know which token is the subject (`nsubj`) of the sentence and we know that that token is a proper noun (`PROPN`). This means we can indicate that there is an entity with a `Name` role, with the token as its name. We know `NOUN`s connected by a `compound` can probably be combined and represented by a single synset. From the `Number=Sing` morphological feature (not visible in Figure 6), we can derive that this compound word is singular. We can map the `NOUN` part of speech tag to its WordNet equivalent: `n`. In this case, we can already format the compound noun as a synset using the lemmas of both nodes: `microwave_oven.n.01`. Note that no proper word-sense disambiguation is done and that this is not entirely language-neutral if we want to target WordNet for our synsets, which are predominantly in English. However, these cases illustrate how we can combine several features to reach our target representation.

Some more complex constructions *require* looking at language-specific features, such as tokens or lemmas. Here, we would have to look at lemmas to detect the negation for instance. We discuss this in more detail in Section 2.7.

[2] https://universaldependencies.org/



(a) Common visualization style for a UD parse.



(b) UD visualization style used in this thesis.

Figure 6: UD parse visualizations for the sentence *Tom does not have a microwave oven.*

What we have examined so far is all based on *Basic* Universal Dependencies. These parses already contain a lot of information, but some more complicated constructions cannot be fully expressed with Basic UD. For these constructions *Enhanced Universal Dependencies*[3] exist (Schuster and Manning, 2016). A lot more can be represented with these, also features that can be helpful in semantic parsing (Findlay and Haug, 2021). A simple illustration of how this could be useful is with the addition of *Conjoined Modifiers*. Take the sentence *A long and old book* for instance. In basic UD, the only `amod` (adjectival modifier) edge we get is from *book* to *long*. In Enhanced UD, we also get an `amod` edge from *book* to *old*. This could be very useful, particularly in the context of our graph transformation approach, where more information generally results in simpler rules. This is because less matching across multiple nodes or edges is (likely) needed to achieve the same match.

There are more Enhanced UD features that can be useful in semantic parsing. However, Enhanced UD is not as readily available as Basic UD. The performance of Enhanced UD parsers is also not as high as their Basic UD counterparts. For these reasons, we opted to use Basic UD for the current project at the moment. When we mention 'UD' throughout this report, we are thus referring to Basic UD.

2.3 GRAPHS IN SEMANTIC PARSING

A UD parse is a *tree*[4], however, this terminology can be confusing since it can refer to different concepts in the context of linguistics, data structures or graph theory. All trees are a type of graph in the context of the current project and this particular

[3] https://universaldependencies.org/u/overview/enhanced-syntax.html

[4] Note that an Enhanced UD parse is *not* necessarily a tree, it is instead always referred to as a graph as it can have multiple incoming and outgoing edges per node and even cycles.

UD tree is *directed*. Therefore, we refer to a UD parse as a *graph* from now on, in particular a *Directed Acyclic Graph* (DAG). Keep in mind that the documentation and papers surrounding UD parsers likely refer to the parses as *trees*.

Using graph structures for meaning representations is no new concept. We have already seen DRGs for DRSs (Abzianidze et al., 2020; Oepen et al., 2020; Bos, 2021). Other prominent representations include the Abstract Meaning Representation (AMR, Banarescu et al. 2013), Universal Conceptual Cognitive Annotation (UCCA, Abend and Rappoport 2013) and the Alexa Meaning Representation Language (Kollar et al., 2018).

Using graphs in meaning representations is appealing for several reasons. For the Alexa MR Language, a graph directly models the downstream application. Namely, a *control flow graph* of sorts, with properties, roles and actions that can be queried. The motivation for AMR to use graphs is that they are *"easy for people to read and easy for programs to traverse"*. UCCA lists the ability to compactly represent a graph as a positive quality, which makes manually annotating sentences a lot easier.

Another reason for using graphs is the abundance of high-quality tooling that is available to work with them. We discuss the tooling used in our approach in detail in Chapter 4. Here, we give a brief overview of one of the more notable ones: GREW[5]. This toolkit, developed by Guillaume (2021), can match and transform any graph-like structure, but is specifically designed to deal with graphs and trees used in NLP. This includes UD dependency parses, other syntactic representations[6], as well as semantic meaning representations, such as AMR.

The GREW graph transformation framework (referred to as *graph rewriting* by GREW) is quite powerful and expressive. It has been used to achieve state-of-the-art results in transforming Basic UD to Enhanced UD for certain languages (Guillaume and Perrier, 2021). It has also seen usage in translating syntactic annotation formats from one to the other (Gerdes et al., 2019). An early version of the framework was also used in an experimental study to create a syntax-semantics interface using dependency trees. Specifically to convert *Paris7 TreeBank* dependencies to *Dependency Minimal Recursion Semantics* (Bonfante et al., 2011).

As mentioned, the current project targets DRGs. We use GREW to perform the majority of structural changes to a UD graph, as well as for some node and edge labeling. Chapter 4 covers the details regarding the entire graph transformation process.

2.4 UD IN SEMANTIC PARSING

There have been attempts at deriving semantic meaning representations from UD directly. In addition, there are semantic parsers that leverage UD to add additional features to a given input sequence. We discuss some prominent projects in this section.

2.4.1 UD to logical forms

The goals of this thesis are similar to those of `DepLambda` (Reddy et al., 2016) and `UDepLambda` (Reddy et al., 2017). These systems map UD to logical forms. We focus on the latter since it is a language-neutral continuation of the former. The mapping from UD to logical forms is done in four steps:

5 `https://grew.fr/`
6 GREW specifically mentions Surface Syntactic Universal Dependencies and Deep-sequoia as examples.

1. Enhancement: First, a subset of features from Enhanced UD are added to a given UD dependency tree. In particular *long distance dependencies*, *types of coordination* and *refined question words*. Reddy et al. note that these enhancements are particularly useful for the downstream task of question answering.

2. Binarization: The UD parse gets transformed into a binarized tree that encodes the order of the semantic composition. This step also adds a composition hierarchy to encode the modifiers to each head. In other words, this enforces the composition and direction of the edges in the binarized tree.

3. Substitution: Words get substituted by typed λ-expressions that encode their lexical semantics and dependency labels. These expressions also indicate whether to copy, invert or merge given λ-expression to compose predicate-argument structures.

4. Composition: Lastly, β-reduction is applied to all λ-expression to get the final, reduced and normalized logical form.

While similar in spirit, the main difference between `UDepLambda` and our approach is that we do not have to deal with complicated operations involving logical variables. Our target representation (SBN) is free of variables and thus allows us to work with the UD tree directly, instead of needing several intermediate representations. Interestingly, Reddy et al. mention the dataset we use for this project (the Parallel Meaning Bank) explicitly for possible future work:

> *"(...) we view UDepLambda as a first step towards learning rules for converting UD to richer semantic representations such as PropBank, AMR, or the Parallel Meaning Bank (...) "*

An important issue to consider is that we cannot compare `UDepLambda` and our approach directly. First, because it targets a different meaning representation, and second because Reddy et al. chose to evaluate their system on a downstream task directly. They map their final logical forms to Freebase graphs and test those on two question answering benchmark datasets.

Our evaluation is considerably different since we have a gold standard meaning representation available to compare system output with directly. We discuss how we do our evaluation in Section 3.3, but we do want to stress again that this is very different from the evaluation of `UDepLambda`.

2.4.2 UD for additional features

Another more common approach to incorporate UD in semantic parsing, is to add parts from a UD parse as additional input features for a model. This is often used for sequence-to-sequence models. Dozat and Manning (2018), for instance, use UD in semantic dependency parsing. This is not the same as semantic parsing in the current project. Instead of creating a full meaning representation, semantic dependency parsing tries to predict edges between all words of a given sentence and assign semantic labels to those. The input sentence stays intact for this task. Instead of labeling an edge with a thematic role, it would instead be labeled with what arguments belong to a given verb for instance. Dozat and Manning take a word and POS embedding, concatenate these and feed them to two modules: one to predict if an edge should be added between two words (nodes) and one to predict the label that best describes that edge. They found that adding the UD features improves performance considerably, compared to not including it.

Xu et al. (2018) try something similar as Reddy et al. (2017), in that they try to convert a dependency parse to logical forms. They implement this with a graph-to-sequence model, instead of the rule-based approach from Reddy et al. (2017). In addition to dependency features, they also encode word order features, as well as constituency features into a single graph that is fed to the network. No details are given on how these syntactic features are obtained; not in their paper and not in the repository for their system. We assume that at least the dependency features are obtained from UD since they explicitly mention Reddy et al. (2016, 2017). They evaluate their system on a dataset with gold standard logical forms. They show that adding these syntactic features improves the performance of their model in creating these representations.

Finally, we outline work done by Yang et al. from 2021 that comes quite close to how we intend to use UD. They train a variety of neural sequence-to-sequence models that take a natural language expression as input and produce a DRS (as used in the PMB) as output. Their main goal is to find out if a model trained on English data can be used for other languages. Or, in more specific terms, they want to see how a pre-trained multi-lingual language model performs on zero-shot cross-lingual transfer learning for DRS parsing. They experimented with adding additional syntactic input features, specifically UPOS tags and dependency relations, both from a UD parse. They used the PMB as their dataset[7] and found that adding this information was surprisingly effective. Or in their words:

> *"(adding UD features), despite its frustrating simplicity, leads to surprisingly strong zero-shot cross-lingual semantic parsers* (...)*"*

They used a different scoring method compared to ours, namely Counter (van Noord et al., 2018). We go into more detail regarding our evaluation in Section 3.3. Their system also targets the clause DRS notation instead of SBN. The indication that language-neutral syntactic information from UD helps in semantic parsing is a promising sign though. Despite that our graph transformation approach is quite different from their approach.

2.5 DRS PARSING

Within semantic parsing, we can distinguish different parsers by the methods they employ, but also by the meaning representation they target. DRSs are the target in the current project, as mentioned. One of the first semantic parsers that focused on DRSs is *Boxer* (Bos, 2008, 2015). This parser has seen many iterations. To build a meaning representation, Boxer starts with a Combinatorial Categorical Grammar (CCG, Steedman 2001) syntactic parse. Using λ-calculus, it constructs DRSs in a compositional manner. It also applies various tools and methods to construct and derive lexical and logical features, such as thematic role labeling and adding tense. Boxer is used in the PMB to generate and bootstrap meaning representations (Abzianidze et al., 2017). In recent work regarding DRS parsing, Boxer has still shown competitive performance (van Noord et al., 2020).

Recent efforts in DRS parsing have mainly focused on using neural methods, including the previously mentioned work by Yang et al. (2021). The methods these neural approaches use range from structural (Fancellu et al., 2019; Evang, 2019) to sequence-to-sequence (Liu et al., 2019; van Noord et al., 2018, 2020). Current state-of-the-art performance for DRS parsing is achieved by the latter. We discuss two neural DRS systems, a structural graph prediction method from Fancellu et al. (2019) and the state-of-the-art sequence-to-sequence method from van Noord et al. (2020).

Fancellu et al. (2019) target DRSs expressed in a graph format (DRGs). The semantic parser they built is based on a sequence-to-graph encoder-decoder model. Specifically, they define a *restricted DAG grammar* and use it in a similar way as Recurrent Neural Network Grammars have been used. Their model learns actions from training data encoded in this grammar. Interestingly, they illustrate the inner

[7] Note that they used version `2.1.0` and `3.0.0`, whereas the current project uses version `4.0.0`.

workings of this grammar using the Penman notation (Kasper, 1989), which happens to play an important role in the current project, as we will outline in Section 3.3. One of the primary reasons to use a grammar, is to greatly reduce ill-formed output when compared to a full sequence-to-sequence model. They compared their system to two sequence-to-sequence models from van Noord et al. (2018), one trained on just gold data and one on both gold and silver data. The current project uses similar comparison systems, which we discuss in Section 4.7. Chapter 3 explains what these gold and silver datasets refer to. This paper uses an older version of the PMB compared to the current project: `2.1.0` versus `4.0.0`.

Fancellu et al. show that, indeed, the percentage of ill-formed output is much less compared to the sequence-to-sequence models. They also outperformed the comparison model trained on just gold data. However, this was not the case for the model trained on both gold and silver data. This was partially because the silver data often was not well-formed according to their grammar. Curiously, Fancellu et al. also experimented with embeddings of UD information to use as additional input features. They used lemmas, UPOS tags and dependency labels, obtained from the, then state-of-the-art, UD parser `UDPipe` (Straka and Straková, 2017). Their ablation results show that adding all these features improves performance slightly. They also tried their model on the other languages in the PMB, namely, Italian, Dutch and German, and found that they all performed worse than English. This was mainly due to the absence of gold data for these languages in the PMB release they used. Similar to Yang et al. (2021), we cannot compare their results directly to the current project since Fancellu et al. used another version of the PMB. Additionally, they also used Counter, which is different from our evaluation method, as we will discuss in Section 3.3.

Fancellu et al. compared their approach to a system from a paper by van Noord et al. from 2018. In 2020, van Noord et al. developed another sequence-to-sequence semantic parser. In developing this parser, the authors experimented with a range of pre-trained language models. They found that a `BERT` model (Devlin et al., 2019), combined with character-level features, surprisingly outperformed much larger language models. They also experimented with adding additional linguistic features and found that these did not contribute much if anything. These features did not come from UD, but instead from parses already present in the PMB. Their final model outperformed the model we discussed from Fancellu et al. (2019), as well as the, at the time, state-of-the-art model from Liu et al. (2019). Note again that all these models targeted the clause notation for DRSs.

2.6 FROM UD TO DRSs

The final area of related work regarding semantic parsers we discuss lies closest to the current project. Specifically, generating DRSs from UD directly. Little work has been done in this area. The previously mentioned work from Yang et al. (2021) resulted in a fully-fledged semantic parser that employs UD as additional features, but it is not done *purely* from UD.

The main work in this area is from Gotham and Haug (2018). They provide a thorough analysis of deriving DRSs from UD using techniques from Lexical Functional Grammars combined with GLUE semantics. Their approach is somewhat similar to the approach from Reddy et al. (2017). They substitute parts of the UD tree with various logical expressions until they can compose and finally reduce these expressions into the target meaning representation. The primary difference in their approaches is that Gotham and Haug do not have to binarize the UD tree first to start substituting logical expressions. While their work is thorough and provides extensive proofs, it is not a fully-fledged semantic parser. In their words:

"(...) the work described in this paper constitutes a proof of concept tested on carefully crafted examples (...). We have achieved some encouraging results, however we are very far from something practically useful (...)"

The approach and method of the current project are quite different. For one, our graph transformation method is a lot less formally established and is more focused on actual data. This, of course, presents several advantages and disadvantages. The main advantage of our approach is that we can use some *"linguistically intuitive"* and data-driven techniques to formulate and improve our graph transformation and graph labeling methods. This ties into another goal of this thesis, to get a basic system working first and foremost and improve it later. This is almost the opposite of the formal and rigorous method Gotham and Haug apply. Exactly this is also the disadvantage of our approach, there is no formal proof underpinning its development or choices.

If we take into account the broad range of semantic parsers we have discussed so far, we can see a spectrum emerge in terms of explainability, transparency and accountability. The state-of-the-art neural parsers do not employ any formal proof or transparency whatsoever. Our approach lies somewhere in the middle of this spectrum, it is fully transparent and explainable but it does not have any formal proof for its choices. At the other side of this spectrum lies the formal rigor of `UDepLambda` by Reddy et al. and the work done by Gotham and Haug.

2.7 NEGATION IN SEMANTIC PARSING

A final important aspect of semantic parsing we want to discuss is *negation*. Negation is crucial in semantics as it models truth values, which are fundamental to any sort of reasoning we want to do with a given meaning representation. Within computational semantics, an entire research area is dedicated to detecting negation, as well as assigning scope to negation (Morante and Blanco, 2012). The previously discussed `UDepLambda` even has an entire extension dedicated to this, called ¬`UDepLambda` (Fancellu et al., 2017).

Naturally, in DRSs, negation is also an important and complex phenomenon. Work done by Basile et al. from 2012 specifically relates to negation detection and assigning negation scope in DRSs. To detect negation, they look at specific tokens, called *negation cues*. Some easy examples for English are *not* or *none*. In addition, as we have seen, quantifiers are also modeled using negation in this context. These cues are detected in the same way, so *every* or *all* for instance for English.

To assign scope once a cue has been found, Basile et al. convert DRSs to a type of DRG. Note that this type of DRG is not the same as we use in the current project. They traverse the DRG and introduce new boxes once a cue has been detected. All 'remaining' tokens after a cue get assigned to a new box. Some other methods and post-processing steps are applied as well, but this is the main idea.

For our approach, we aim to detect negation and quantifier cues and introduce new boxes. We want to do this for multiple languages, in a similar way as Basile et al., by looking at tokens (lemmas). However, we chose to leave out assigning scope to these cues for the time being. The modular structure of our approach does make it feasible to easily add this in the future.

3 DATA AND MATERIAL

In this chapter, we introduce the dataset we use in the project and some imperfections regarding this dataset. We show what the input data for our approach looks like. Lastly, we outline the method we use for evaluating Discourse Representation Graphs.

3.1 THE PARALLEL MEANING BANK

The dataset we use for our approach is part of the Parallel Meaning Bank project[1] (PMB, Abzianidze et al. 2017). This project is a continuation of the Groningen Meaning Bank project (GMB, Bos et al. 2017). The aim of the PMB is to provide a large and high-quality corpus of annotated sentences. It is comprised of a multi-layered, semi-automatic annotation system as well as several datasets that are built using this system. It is ultimately used to create DRSs, but some intermediate annotation and parsing results can be used on their own as well. The seven annotation layers, adapted from Abzianidze et al. (2020), are:

- Tokenisation - Indicate sentence boundaries and word tokens;
- Symbolisation - Assign a non-logical symbol to a word or multi-word token;
- Word-sense disambiguation - Assign concepts to symbols using WordNet senses;
- Co-reference resolution - Mark antecedents for anaphoric expressions;
- Thematic role labeling - Annotate relations between entities using VerbNet roles and comparison operators;
- Syntactic analysis - Provide lexical categories and build a syntactic structure for the sentence, based on Combinatorial Categorical Grammar;
- Semantic tagging - Assign a semantic type to a word token.

The PMB gets the *Parallel* part of its name because a large set of its documents are parallel translations of each other in multiple languages. Once one of those languages gets annotated, it is possible to (partially) project the annotations to other languages as well. The final DRSs aim to be as language-neutral as possible. The current project specifically uses version `4.0.0` of the PMB[2].

3.1.1 Dataset characteristics

Table 2 shows detailed information regarding the PMB release we use for the current project. *Gold* data refers to data that is fully checked and corrected by humans. *Silver* data is partially checked, some of the annotation layers are unchecked for example. Lastly, *bronze* data refers to unchecked and automatically generated data. Note that our system only uses gold data. Some experiments were done with silver data as well. The bronze set was only used alongside the other sets to develop and test the SBN parser for our system. Bronze is included here for completeness' sake. Later on, we outline some interesting phenomena where some examples from the bronze set are used as well.

[1] `https://pmb.let.rug.nl/`
[2] Which can be downloaded from here: `https://pmb.let.rug.nl/data.php`.

Table 2: Counts per language and data layer for the PMB `4.0.0` release. *Multi-sent* refers to documents with > 1 sentence in them (based on `*.iob.tok` files). *Tokens / doc* shows the average tokens per document (based on `*.off.tok` files).

		# Docs	# Multi-sent	# Tokens	Tokens / doc
English	Gold	10,715	94	70,307	6.6
	Silver	127,303	6,282	1,441,919	11.3
	Bronze	156,286	5,391	1,463,721	9.4
Dutch	Gold	1,467	1	9,025	6.2
	Silver	1,440	88	13,843	9.6
	Bronze	28,265	2,139	288,975	10.2
Italian	Gold	1,686	0	9,205	5.5
	Silver	4,088	145	33,314	8.2
	Bronze	100,963	3,197	817,436	8.1
German	Gold	2,844	3	16,571	5.8
	Silver	6,355	260	60,404	9.5
	Bronze	151,493	5,164	1,447,415	9.6

As we can see in Table 2, the PMB primarily consists of short texts, the majority of which are just single sentences. Each document in the PMB has several files associated with it: DRSs in various formats, results from annotation layers and metadata. Not all of these are used in the current project and our system also adds some additional items to this file structure. Two files are essential: the *raw* sentence and the DRS in SBN format. We discuss the relevance of these in more detail in Section 3.2 and 3.3.

The gold data per language is split up into three splits: train, dev(elopment) and test. For historical reasons, English also includes a fourth split: eval(ulation). Silver and bronze data are assumed to be used as possible additional training data. Table 3 shows the counts for the gold data splits per language.

Table 3: Counts per language and data split for the gold data in the PMB `4.0.0` release.

	Train	Dev	Test	Eval
English	7,668	1,169	1,048	830
Dutch	539	437	491	-
Italian	685	540	461	-
German	1,738	559	547	-

We can see that there is much less data available for Dutch, Italian and German, compared to English. This is something we will come back to multiple times in this thesis.

3.1.2 Dataset imperfections

While developing the system, we encountered some imperfections in the data, which we will discuss briefly. The main imperfections are: cyclic SBN graphs, empty SBN documents, whitespace in WordNet ids (synsets) and possibly ambiguous indices.

An important assumption for the graph transformation process, is that both the UD and SBN graphs are *Directed Acyclic Graphs*. The primary reason for this has to do with how the graphs are evaluated. We go into more detail about this in Section 3.3. There are a number of cyclic graphs in the SBN format in the PMB. Figure 7 shows an example of such a graph.

From the manually inspected sample of cyclic graphs, all of them are similar in nature to Figure 7. This might be due to a conversion error with possessive constructions.

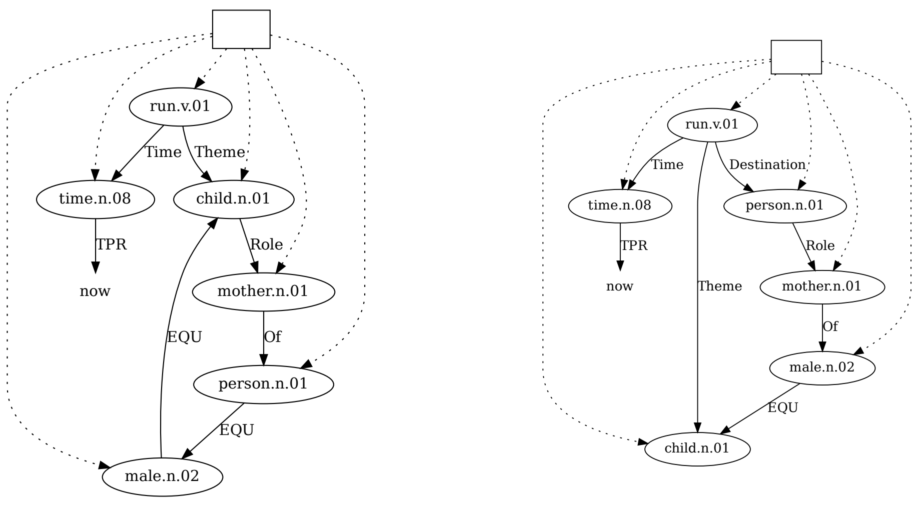


Figure 7: Graphs for the sentence *The child ran to his mother*, with the cyclic SBN graph on the left as in the PMB `4.0.0`. Right is the fixed and updated version in the PMB at the time of writing. PMB id: `en/gold/p67/d1486`.

Another imperfection we encountered while parsing SBN files was whitespace in synset ids. All cases of this had the utf-8 whitespace character `U+00a0` in them.

Table 4 shows an overview of all imperfections. Most of these have already been fixed in the PMB and these fixes will be included in a future release. Note however that, at the time of writing, this has not been released. We are using the dataset with these imperfections.

Table 4: Imperfections in PMB release `4.0.0`.

		Cyclic SBN	Empty SBN	Whitespace in synsets
English	Gold	35 *(0.3%)*	4 *(0.0%)*	0 *(0.0%)*
	Silver	1,527 *(1.2%)*	1 *(0.0%)*	2 *(0.0%)*
	Bronze	1,044 *(0.7%)*	0 *(0.0%)*	1 *(0.0%)*
Dutch	Gold	0 *(0.0%)*	0 *(0.0%)*	0 *(0.0%)*
	Silver	5 *(0.3%)*	0 *(0.0%)*	0 *(0.0%)*
	Bronze	173 *(0.6%)*	0 *(0.0%)*	23 *(0.1%)*
Italian	Gold	0 *(0.0%)*	0 *(0.0%)*	0 *(0.0%)*
	Silver	17 *(0.4%)*	0 *(0.0%)*	0 *(0.0%)*
	Bronze	418 *(0.4%)*	0 *(0.0%)*	3 *(0.0%)*
German	Gold	2 *(0.1%)*	0 *(0.0%)*	0 *(0.0%)*
	Silver	80 *(1.3%)*	0 *(0.0%)*	3 *(0.0%)*
	Bronze	1,129 *(0.7%)*	0 *(0.0%)*	19 *(0.0%)*

We can see that there are very few cases of these imperfections across languages and data splits. Since this number is so low and since these imperfections will be fixed, we did not implement special procedures to deal with them. We ignore these cases in our evaluation and in the mapping extraction, more on this in Section 4.5. It is important to mention that the counts in Table 3 of *usable* documents is ever so slightly lower due to these imperfections.

Note that the training data for the neural comparison system did *not* filter out these cases. It used all English gold data for one system and all English gold and silver data for the other. This was partially due to miscommunication and partially due to some errors that had not been found yet at the time of training these systems.

Again, there are very few cases, so we assume this will not have any noticeable effect, but for the sake of transparency, it is important to mention it here.

The imperfections listed in Table 4 are rather harmless and can easily be detected and ignored. This is not true for a more subtle inconsistency regarding the SBN specification itself. Recall that SBN is modeled around indices, which remove the need for explicit variables. These indices are an integer prefixed with either a + or −, surrounded by whitespace (note that 0 is a valid index, which is written as +0 in the PMB). SBN also has *constants*, which are essentially the leaf nodes of the graph. These can be dates, names and several special tokens (*now* for example). Numerical values are also constants, with no special enclosing symbols (names have ′ " ′ surrounding them for instance). Now let's consider the example in Figure 8.

The mercury plunged to minus 7 overnight.

```
mercury.n.04
plunge.v.01    Theme -1 Time +1
               Destination +2
               Manner +3
time.n.08      TPR now
entity.n.01    EQU -7
overnight.a.01
```

Figure 8: Example of possibly ambiguous index, PMB id: `en/silver/p15/d3131`.

In the PMB, "minus 7" gets normalized to "−7". This is a constant that is a signed integer, which is exactly how we would represent an index in SBN. In this particular case, we can detect that this is indeed a constant since the index points at an invalid synset node; there is no node −7 synset nodes back. However, if it had been "−1" instead, we would have no way of knowing if it is a constant or an index. This makes it quite difficult to find these examples from just the data alone. A simple solution to this problem would be to always quote constants similarly to how multi-word expressions are encoded.

In the entire PMB `4.0.0` release only two such cases can be found: the previous example and a bronze document for German (`de/bronze/p41/d2482`), which is an erroneous parse anyway. We did not find possibly ambiguous constants that point at a correct synset. These are very hard to find and might not even be present in the dataset.

3.2 UD representation

The previously mentioned *raw* file for a document in the PMB simply contains the entire sentence (or multiple sentences) without any processing done to it. In other words, the *raw* string in plain text format. We need these to create the UD parses from where the rest of the process starts. There are several pre-tokenized formats per document available in the PMB. While UD parsers generally accept pre-tokenized input, we chose to use the raw text instead. This allows us to better evaluate different UD parsers since they are entirely self-contained this way. It also makes our system more modular. Different UD parsers might require other pre-processing methods for pre-tokenized input, which introduces unneeded complexity. The UD parses are stored in the *CONLL* file format[3] per document. Table 5 shows what information might be present in a UD parse in CONLL format. We store a parse per document, per UD parser. Which parsers we use and how the CONLL files are processed is discussed in Chapter 4.

[3] Or variations such as CONLL-X or CONLL-U.

Table 5: Example of information present in a CONLL file from a UD parse for the sentence *Tracy lost her glasses*. Underlined headers are required and always available. PMB id: `en/gold/p04/d1646`.

FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
Tracy	Tracy	PROPN	NNP	Number=Sing	2	nsubj	_	_
lost	lose	VERB	VBD	Mood=Ind Tense=Past VerbForm=Fin	0	root	_	_
her	she	PRON	PRP$	Gender=Fem Number=Sing Person=3 Poss=Yes Pron-Type=Prs	4	nmod:poss	_	_
glasses	glasses	NOUN	NNS	Number=Plur	2	obj	_	_
.	.	PUNCT	.	_	2	punct	_	_

3.3 evaluating Discourse Representation Graphs

Apart from the *raw* file, another essential file per document is the DRS in SBN format. Each document in the PMB has one SBN file associated with it. We use this file as the gold standard[4]. We compare this file with output from both our graph transformation system and our neural comparison system.

Since it is a relatively new format, there is not an official evaluation method yet that operates on SBN directly. To evaluate semantic parses, two main options exist: SMATCH (Cai and Knight, 2013) and Counter (van Noord et al., 2018). The latter is designed with DRSs in mind, but specifically for the *clausal* notation of DRSs. This notation does not use a graph-like structure, it instead works with clauses that can have three or four components. Counter is based on SMATCH and directly addresses this issue of four component clauses. SMATCH is originally created to evaluate AMR. A valid AMR parse creates a DAG. SMATCH flattens a reference (gold) graph and the target graph into triples and calculates the overlap between these, resulting in precision, recall and an F1-score as output metrics. A major motivation for the SBN format is that it creates DAGs, which opens up a lot of options and flexibility in terms of processing this format. This is also mentioned in the paper that introduces SBN (Bos, 2021):

> *"SBN shares characteristics with the Penman notation (Kasper, 1989), and formalisms based on that such as Abstract Meaning Representation (Banarescu et al., 2013) (...)"*

Penman (Kasper, 1989) is a notation format to represent DAGs. Since SBN also creates DAGs, we can convert it to the Penman notation. By doing this, we can leverage SMATCH for evaluation. Figure 9 shows the same DRG in the two formats for the same sentence. Note that a single slash "`/`" in Penman is shorthand for the `:instance` relation.

In this example, we can clearly see the absence of explicit variables in SBN. In the Penman notation, variables can have any non-whitespace symbol, as long as these are unique when declared. The relation `:member` is not part of SBN, connections from nodes to boxes are implicit there. In other words, all synset nodes are always connected to a box and the initial box (`b0` in Penman) is implicit since it is always present. The `box` label is given to all boxes since a particular box can be seen as an instance of the *type* or *concept* 'box'. As we outlined before, the meaning these boxes convey is expressed by the edge they might have with another box. Their connection

[4] *Gold* as in ground truth, not as in the data split itself.

```
plastic.n.01
chair.n.01   MadeOf -1
time.n.08    EQU now
cheap.a.01   AttributeOf -2 Time -1
```

```
(b0 / box
  :member (s0 / plastic.n.01)
  :member (s1 / chair.n.01
    :MadeOf s0)
  :member (s2 / time.n.08
    :EQU now)
  :member (s3 / cheap.a.01
    :AttributeOf s1
    :Time s2))
```

Figure 9: SBN notation (left) and Penman notation (right) for the DRG of the sentence *That plastic chair is cheap*.

to synset nodes also conveys meaning, specifically to model scope. Figure 10 shows how scope with multiple boxes is represented in Penman.

```
             NEGATION -1
person.n.01
             NEGATION -1
time.n.08    EQU now
eat.v.02     Agent -2 Time -1
```

```
(b0 / box
  :NEGATION (b1 / box
    :member (s0 / person.n.01)
    :NEGATION (b2 / box
      :member (s1 / time.n.08
        :EQU now)
      :member (s2 / eat.v.02
        :Agent s0
        :Time s1))))
```

Figure 10: SBN notation (left) and Penman notation (right) for a DRG with multiple boxes (scope) of the sentence *Everyone's eating*.

This Penman notation from Figure 10 can already be parsed and scored by SMATCH, even though it is not AMR. However, there are several improvements to be made. The first improvement to our notation is writing it in such a way that SMATCH can invert roles that support it. Consider the following triple from the example in Figure 9:

$$\text{(s3 AttributeOf s1)} \quad \equiv \quad \text{(s1 Attribute s3)}$$

$$\textit{Cheap is an attribute of the chair.} \quad \equiv \quad \textit{The chair has the attribute cheap.}$$

SMATCH can invert these logically equivalent triples when the relation (role) ends with "∗`-of`". This is the AMR way of writing these types of relations. This process is called *role inversion*.

Another improvement we can make to our notation is splitting up the WordNet synsets. If the synsets are fully intact, as in the example, we evaluate three components simultaneously: the lemma, the WordNet part of speech tag and the sense number. The sense number in particular targets the task of word-sense disambiguation, which might not be desirable. For this reason, we include options to filter out certain synset id components when exporting a graph to the Penman notation. Evaluating exports with and without a sense number was particularly useful in developing the mappings for the sense numbers. We discuss this in Section 4.5.

Figure 11 shows part of our previous example in the improved format. The new `synset` label serves the same purpose as the `box` label we discussed.

A final point of improvement is consistent quoting of values. In SBN, several quoting methods are used to indicate different types of constants and tokens. This is, arguably, not that important for the final DRG we want to evaluate. As long as the SBN gets parsed properly to build this DRG. The final evaluation should not target notation-specific conventions, it should evaluate the content of a DRG. For this reason, we quote all constants we insert into the Penman notation in a consistent manner. Note that a constant in the Penman notation is not the same as

```
(b0 / box
  (...)
  :member (s3 / cheap.a.01
    :AttributeOf s1
    :Time s2))
```

```
(b0 / box
  (...)
  :member (s3 / synset
    :lemma cheap
    :pos a
    :sense 01
    :Attribute-of s1
    :Time s2))
```

Figure 11: Revised notation with *-of* and split up synset components. The example notation from before is on the left and the improved notation on the right.

a constant in SBN. In Penman, a constant refers to a non-logical symbol in the last position of a triple. Figure 12 shows the final triples SMATCH works with. These triples are compared to those of another graph. Figure 12 also demonstrates the consistent quoting of constants.

```
(b0, :instance, "box")           (s2, :instance, "synset")
(b0, :member, s0)                (s2, :lemma, "time")
(s0, :instance, "synset")        (s2, :pos, "n")
(s0, :lemma, "plastic")          (s2, :sense, "08")
(s0, :pos, "n")                  (s2, :EQU, c0)
(s0, :sense, "01")               (c0, :instance, "now")
(b0, :member, s1)                (b0, :member, s3)
(s1, :instance, "synset")        (s3, :instance, "synset")
(s1, :lemma, "chair")            (s3, :lemma, "cheap")
(s1, :pos, "n")                  (s3, :pos, "a")
(s1, :sense, "01")               (s3, :sense, "01")
(s1, :MadeOf, s0)                (s1, :Attribute, s3)
(b0, :member, s2)                (s3, :Time, s2)
```

Figure 12: The final triples that SMATCH uses for the DRG of *That plastic chair is cheap*. Note that the `AttributeOf` triple has been inverted and all constants are quoted.

Lastly, we developed two methods to evaluate a given SBN graph: a strict and a lenient method. This particular option indicates whether an SBN graph should be considered to be ill-formed based on possibly wrong indices. We discussed this in more detail in Section 3.1, specifically Figure 8. This does not only pertain to evaluation, but also the SBN specification itself. We can give a parser the benefit of the doubt when it outputs SBN with possibly wrong indices and consider these indices to be constants. Or we can punish it since this is very rare in our dataset. It does not occur in the gold data of any of the languages. It is arguably more plausible that the parser made an error and created an ill-formed DRG. An ill-formed graph, regardless of the reason for it being ill-formed, always gets a score of 0 for all metrics.

The specific SMATCH implementation we use is provided by `mtool`[5], which is the official scoring method in both the 2019 and 2020 editions of the Meaning Representation Parsing (MRP) shared tasks (Oepen et al., 2020).

[5] https://github.com/cfmrp/mtool

4 METHOD

In this chapter, we introduce our system: *UD-Boxer*. First, we give a brief overview of the internal structure of UD-Boxer. Next, we introduce the UD parsers we use for our experiments and how UD-Boxer transforms a UD parse into a DRG. We go through the entire system with a single example sentence. After this example, we go into more detail regarding the rules and mappings used in UD-Boxer. We briefly describe the development process and finally introduce our neural comparison system.

4.1 SYSTEM OVERVIEW

Before going through an example sentence, we briefly want to outline the structure of UD-Boxer. The system works in three basic steps:

1. Apply structural graph transformations.
2. Substitute syntactic labels with semantic concepts.
3. Connect box nodes.

In addition to the core transformation process, other components deal with parsing, validating and exporting various formats. A number of different graph formats are used in the system that serve different purposes. At the core of all these formats lies a `networkx`[1] (Hagberg et al., 2008) graph that is translated into the different formats when needed. Figure 13 gives an overview of the graph formats used in UD-Boxer, specifically for the `SBNGraph`.

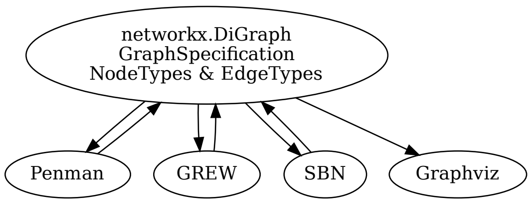


Figure 13: Basic UD-Boxer graph structure overview for the `SBNGraph`.

To support these translations and to comprehensibly model all functionality surrounding these graphs, a type system describing the nodes and edges is used all throughout UD-Boxer. These types can be combined with a particular specification for a given graph type for easy parsing and validating. The node types of the `SBNGraph` are *synsets*, *constants* and *boxes*. The edge types are *roles*, *DRS operators*, *box connections* and *box-to-box connections*. These types are needed to make sure a graph is well-formed. For example, a constant cannot be connected to a box and a box-to-box edge can only exist between two box nodes.

The specification of a graph describes *what* these node and edge types look like and how they should be parsed. These are some examples of the node and edge characteristics that are included in the `SBNGraph` specification:

[1] https://networkx.org/

- All known roles, operators, box indicators and more are present in the specification. These are used in parsing and validating nodes and edges;
- Various regular expressions describe what a constant might look like. Name constants have double quotes surrounding them, years have single quotes surrounding them and so on;
- Roles that can be inverted are listed in the specification. These are used when exporting a graph to the Penman notation, but could also be used in parsing for instance.

UD-Boxer also has a `UDGraph`, which has its own node and edge types. It also has a graph specification describing those, specifically for Basic Universal Dependencies. This specification includes dependency relations, morphological features, POS tags and more. Additionally, this specification also lists several mappings from UD features to semantic concepts. We go into more detail regarding these mappings in Section 4.5. We use this UD specification when parsing CONLL files, resolving syntactic labels to semantic concepts and when extracting label mappings.

4.2 OFF-THE-SHELF UD PARSERS

To start the process, we need a UD parse of an input sentence. We can get this by putting our sentence through an 'off-the-shelf' UD parser. These types of parsers all support exporting their parse to the CONLL format, which we have seen. We have also already seen what information is present in a UD parse in Table 5. This table actually shows the parse of the example sentence we are using in this chapter: *Tracy lost her glasses*[2]. Figure 14 shows the UD graph visualization of this sentence. Note that not all information from the UD parse is present in these visualizations. Optional information, such as morphological features, are left out, but *are* used in creating the final DRG — this is discussed later. Most UD features are language-neutral and one of the main goals of UD-Boxer is to target these features before targeting language-specific features.

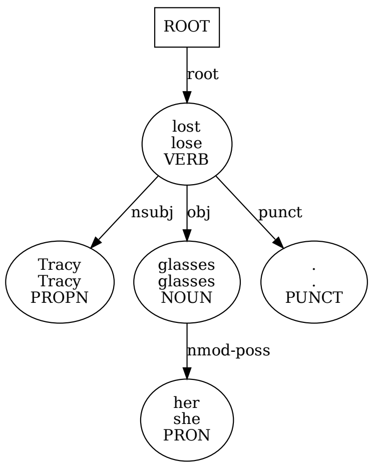


Figure 14: UD graph of the sentence *Tracy lost her glasses*, created using Stanza.

[2] PMB id: `en/gold/p04/d1646`.

The UD parser component is modular, so any parser can be inserted, as long as it is able to produce a CONLL file as output. UD-Boxer integrates two state-of-the-art neural UD parsers: Stanza[3] (Qi et al., 2020) and Trankit[4] (Nguyen et al., 2021).

4.3 FULL TRANSFORMATION EXAMPLE

We have seen how we obtain a UD parse for our example sentence. As discussed in Section 2.3, we are using the GREW framework to perform structural changes to the UD graph. In GREW, rules are defined as a *pattern* combined with one or more *commands* and an optional *Negative Application Pattern* (NAP), called a *without* block.

Both the pattern and NAP are themselves graphs, where the pattern can match a subgraph of the input graph, based on certain features. The NAP can filter out particular occurrences of the first pattern. This is especially useful when transforming a graph and a given rule should be applied only once. The NAP prevents an infinite loop of applying the pattern.

The *commands* sequentially describe steps that modify the structure of the graph. They can also describe changes to attributes of nodes and edges. UD-Boxer starts by adding a new attribute to all nodes and edges to 'store' semantic concepts. We use a new attribute, instead of overwriting the existing UD features, so we can keep the UD information intact during the entire process. This attribute is called *token*[5] (*tok* in visualizations for brevity). This attribute can hold a DRG label, a special keyword or a constant. For nodes, the initial value for this attribute is the `lemma` UD feature. This is done since most nodes will probably be converted to WordNet synsets in the substitution step. Synsets, as we have seen, are partially made up of one or more lemmas.

Edges get the special keyword `NONE` as their initial *token* value. Ideally, this should be overwritten with a proper edge label at some point during the transformation process. If not, the keyword is picked up in the substitution step. There, UD-Boxer will try to properly assign an edge label. Figure 15 shows the rules for adding this new node and edge attribute.

The next step is to connect the `User` role in a possessive construction. This is done by connecting the target of an `nmod:poss` edge to the subject node (`nsubj`). After we have connected this edge, we can remove the indicator (IND in Figure 16). Here, the indicator is the pronoun *her*. In our meaning representation, this relation is encoded directly into the `User` edge. Figure 16 shows the result of applying this rule. Note that the NAP makes sure this only happens once. In all following examples in this section, we use colors to clarify the process. Green indicates that components of the graph are newly added and red indicates they are deleted, both as a result of applying the rule(s).

Note that labeling this edge as *User* is not ideal. UD is too ambiguous to label this properly in all cases. If, for instance, the target was *her dog* instead of *her glasses*, the UD parse would be identical in terms of part of speech tags and dependency relations. However, according to the VerbNet role specifications, this would be an *Owner*, not a *User* role. In developing UD-Boxer, we chose to use a *most frequent* approach to these cases. We took the most frequent role or label from the training data when applying them to certain patterns. We go into more detail regarding this in Section 4.5.

Our next step is separating the 'entity' *Tracy* and her name. In other words, if we know we are dealing with a proper noun (`PROPN`), we need separate nodes to

[3] `https://stanfordnlp.github.io/stanza/`
[4] `https://github.com/nlp-uoregon/trankit`
[5] We are aware that 'token' is quite a loaded term in NLP. We considered using 'label', but this is a reserved keyword in GREW (`https://grew.fr/doc/graph/`). 'Semantic label' would be a good attribute name, but that is quite verbose, especially when reading and writing rules. This is why we chose 'token'. Additionally, in the context of parsing SBN, nodes and edges (and indices) *are* individual tokens.

```
rule add_token_nodes {
  pattern {
    N [lemma, !token];
  }
  commands {
    N.token = N.lemma;
  }
}

rule add_token_edges {
  pattern {
    E: N -[!token]-> M;
  }
  commands {
    E.token = "NONE";
  }
}
```

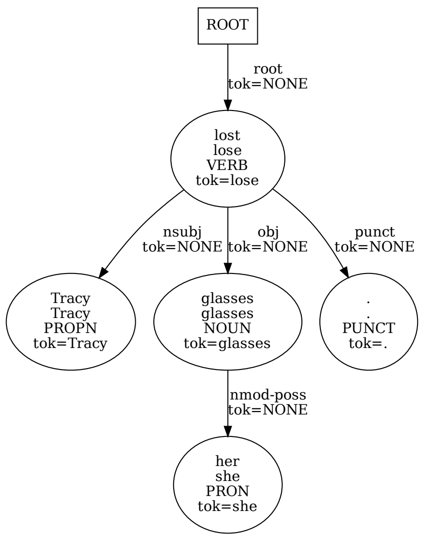


Figure 15: Rules that add a *token* attribute to nodes and edges.

indicate the entity and the name of this entity. Figure 17 shows what this looks like. Assigning the proper synset to the entity node is done in the substitution step.

Next, we are going to add a node and edge to indicate temporal information. The actual resolving of this information is done in the substitution step, which picks up the special keyword `TIMERELATION`. One label we can already resolve is the time node synset since this is overwhelmingly used in the PMB to indicate time relations. Adding the time node and edge to our graph is shown in Figure 18.

Lastly, we need to discard all nodes and edges that are not semantically relevant. In this particular sentence, we only need to remove the explicit root and the punctuation mark. Figure 19 shows this transformation. In other sentences, there might be a lot more nodes and edges that need to be discarded. We discuss such rules, and more, in Section 4.4.

Once these steps are complete, no more rules can be applied in GREW.

```
rule connect_user {
  pattern {
    USER [upos=PROPN|NOUN];
    * -[1=nsubj]-> USER;
    REL: ITEM -[1=nmod, 2=poss]-> IND;
  }
  without {
    ITEM -[token="User"]-> USER;
  }
  commands {
    add_edge ITEM -[token="User"]-> USER;
    del_edge REL;
    del_node IND;
  }
}
```

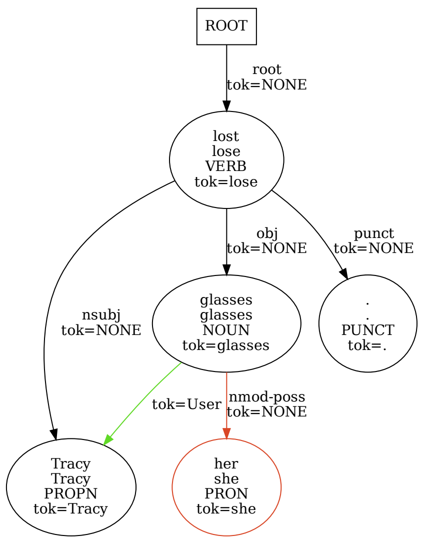


Figure 16: Rule that connects the *User* in a possessive construction in the graph.

```
rule expand_name {
  pattern {
    NAME [upos=PROPN];
  }
  without {
    NAME -> *;
  }
  commands {
    add_node CONST;
    CONST.token = NAME.textform;
    add_edge NAME -[token="Name"]-> CONST;
  }
}
```

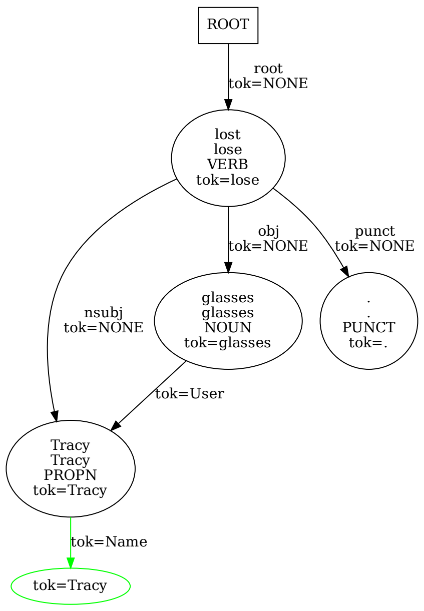


Figure 17: Rule that adds a constant node to indicate the name of a *PROPN* node. It also adds the corresponding *Name* edge.

```
rule add_time {
  pattern {
    N [];
    * -[1=root]-> N;
  }
  without {
    N -[token="Time"]-> *;
  }
  commands {
    add_node TS;
    add_node TC;
    TS.token = "time.n.08";
    TC.token = "now";
    add_edge N -[token="Time"]-> TS;
    add_edge TS -[token="TIMERELATION"]-> TC;
  }
}
```

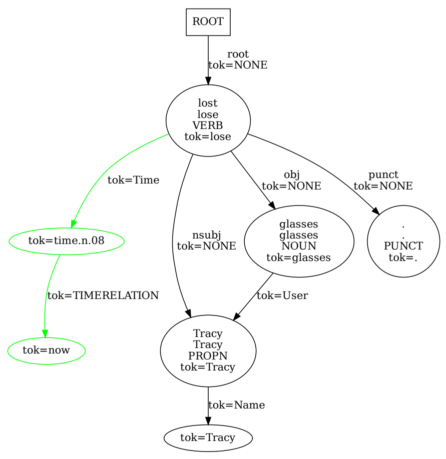


Figure 18: Rule that adds a time synset node (TS), a time constant node (TC) and their corresponding edges.

```
rule remove_root_punct {
  pattern {
    * -[1=root]-> ROOT;
    E: ROOT -> N;
    N [upos=PUNCT];
  }
  commands {
    del_edge E;
    del_node N;
  }
}

rule remove_explicit_root {
  pattern {
    N [];
    E: N -[1=root]-> T;
  }
  commands {
    del_edge E;
    del_node N;
  }
}
```

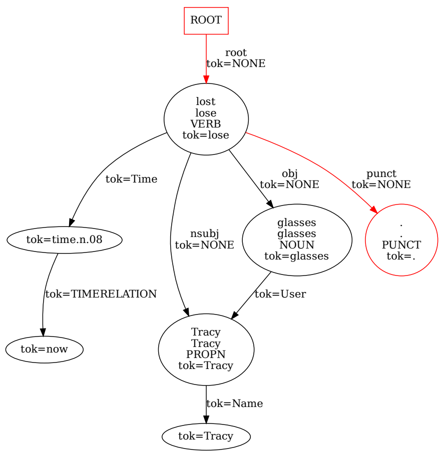


Figure 19: Rules that discard the explicit root and main punctuation mark nodes and edges.

Once the transformations in the GREW step are complete, UD-Boxer converts the output graph to an `SBNGraph` using its graph specification. This makes sure node and edge types are correct and that the output graph from GREW is well-formed. During this step, syntactic labels and special keywords get resolved to semantic concepts through various strategies. One key approach of UD-Boxer is that it always tries to generate a valid meaning representation. This means that, at various points, if a label cannot be resolved properly with the information given, UD-Boxer chooses from a set of default or fallback labels. These are also based on frequencies in the training data.

Let us take a look at our running example again. First, all nodes get resolved. This is done by checking if the *token* attribute of a given node matches anything in the `SBNGraph` specification. This could be a synset or a new box indicator for instance. If this yields no results, the graph resolver looks for special keywords that were introduced in the GREW step. Each keyword is associated with a particular strategy. At the time of writing there is just one keyword for nodes: `GENDER`. This particular strategy tries to look for the `Gender`[6] morphological feature that could be present in a UD parse for a given node. If present, this feature gets mapped to the correct synset. If not, a default value will be chosen: *person.n.01*[7]. Note that some UD parsers assign multiple values to this feature as in some languages multiple values can apply. In that case, the first is chosen.

If the previous step does not apply, the next strategy is tried. This is to format the token as a synset. This only happens if the `upos` node attribute is present. If this happens to be a proper noun (`PROPN`), the previous gender resolving strategy is used here as well. Otherwise, UD-Boxer maps the UD UPOS value to its approximate WordNet POS equivalent. This is not ideal since UPOS has eighteen POS tags and WordNet only four. Nonetheless, this mapping performed very well. As we have seen, the *token* attribute initially gets assigned the `lemma` UD feature. Compound words get combined: *token* attributes are concatenated with an underscore. This is the same as how WordNet synsets are formatted. With this combination of a (combined) lemma and WordNet POS tag, UD-Boxer looks for a sense number mapping. If this does not exist, it tries it again with just the lemma and if that fails it assigns *01* as the default sense number. Lastly, if all this does not apply, UD-Boxer assumes the node is a constant. Tabel 6 shows what node resolving steps were applied to our example graph.

[6] https://universaldependencies.org/u/feat/Gender.html
[7] Note that all results were obtained by using *female.n.02* since this was the most common in the training data. For normal use *person.n.01* or the more general *entity.n.01* make more sense. The former is the current default.

Table 6: Node resolving for our example sentence. Note that some details are left out.

Input	Strategy	Type	Resulting token
Tracy [upos=PROPN]	PROPN gender resolving	Synset	female.n.02 (default gender)
lose [upos=VERB]	Lemma + POS lookup	Synset	lose.v.01
glasses [upos=NOUN]	Lemma + POS lookup	Synset	glasses.n.01
time.n.08	Specification match	Synset	time.n.08
now	No match	Constant	now
Tracy	No match	Constant	Tracy

Next, the edge types and tokens get resolved. This process is very similar to the node resolving. We start again by checking if a given edge *token* attribute matches the `SBNGraph` specification. This can result in a *Role* or *DRS Operator* for example. Next, we look for the keywords `TIMERELATION` or `NONE`. To resolve the first, UD-Boxer tries to collect all `Tense`[8] attributes from all nodes in the graph. This is an optional morphological feature in a UD parse. If multiple exist, the most common is picked and mapped to the corresponding temporal DRS relation. If UD-Boxer encounters `NONE`, it tries to get the `upos` attribute from the *from* and *to* nodes as well as the `deprel` (dependency relation) attribute from the edge. These get combined and looked up in pre-extracted mappings to find the corresponding type and token. If this combination is not present in the mappings, the (configurable) default type and token will be used. Based on training data, this is the *Agent* role. Tabel 7 shows what edge resolving steps were applied to our example graph.

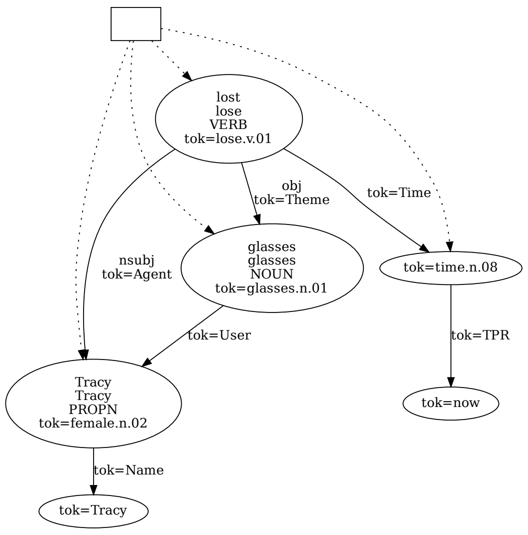
Table 7: Edge resolving for our example sentence. Note that some details are left out.

Input	Strategy	Type	Resulting token
TIMERELATION	Tense resolving	DRS operator	TPR
Time	Specification match	Role	Time
NONE [deprel=nsubj]	VERB-nsubj-PROPN mapping	Role	Agent
NONE [deprel=obj]	VERB-obj-NOUN mapping	Role	Theme
User	Specification match	Role	User
Name	Specification match	Role	Name

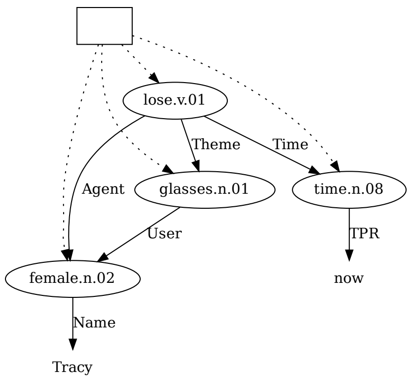
One task left to do is to introduce the 'starting box' in the graph and connect all synset nodes to it. Newly introduced boxes, that were encountered during the resolving process, get connected here as well. This is currently done simply by connecting them to the starting box. There is no support yet for assigning scope to individual nodes using boxes. This also ties into the box constructions UD-Boxer currently supports, which are multiple sentences, negation and quantifiers. This is not ideal and clearly an area of improvement, which we also discuss in Chapter 5. Figure 20a shows the final resulting graph.

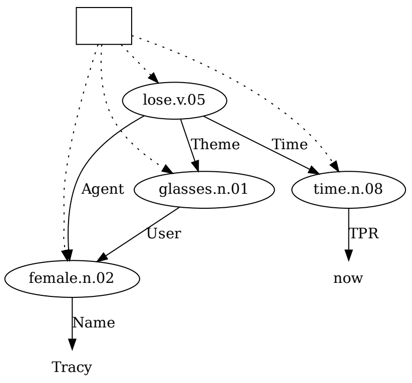
With this resulting graph we can do several things, for instance: extract information from it, traverse it or convert it to various formats. We can export it to the Penman format in order to evaluate it. Let us compare it to the gold DRG for this sentence. Figure 20b shows a visually cleaned version of our graph and Figure 20c the gold standard graph for the example sentence.

[8] https://universaldependencies.org/u/feat/Tense.html



(a) Resulting graph from our running example.





(b) Cleaned resulting graph.

(c) Gold graph.

Figure 20: Resulting DRG after full process and gold DRG for our example sentence *Tracy lost her glasses*.

We can see that this particular example went well, the only thing wrong is the sense number mapping for *lose*. If we export both graphs to the Penman format and use SMATCH to compare them, we get the following scores: precision 96.7, recall 96.7 and F1 96.7. This is, of course, an example that shows how the system performs with a graph it can deal with rather well. Chapter 5 provides an extensive error analysis with examples where UD-Boxer falls short.

4.4 GRAPH TRANSFORMATIONS

Let us take a closer look at the graph transformations UD-Boxer can perform. We can divide these into three categories: *structural*, *attribute* and *language-specific*. There is overlap between these, however, for the sake of explaining, we keep this distinction.

Structural rules change the composition of the graph by adding or removing nodes or edges. Rules that discard nodes and edges that are not semantically relevant, such as in Figure 19, are an example of this type of rule. A considerable number of part of speech tags are generally not semantically relevant in DRSs. The rule in Figure 21 discards nodes with these particular tags. This rule happened to not apply to our example sentence. Note that some rules explicitly remove or overwrite the `upos` feature after they transformed a given node. This is done to not accidentally remove them later on with the cleaning rule. This rule is applied as one of the last, again in order to not remove information another rule might need.

```
rule remove_unwanted_pos {
  pattern {
    N [upos=PUNCT|DET|AUX|ADP|PART|CCONJ|SCONJ];
  }
  commands {
    del_node N;
  }
}
```

Figure 21: Cleaning rule to remove unwanted nodes with particular part of speech tags.

The *attribute* rules add, change, combine or remove node or edge attributes. An example is the rule that adds the *token* attribute, as we have seen. Figure 22 shows two rules that combine features of two nodes into one. Note that these rules use an underscore to concatenate tokens in order to be compatible with how WordNet formats synsets.

```
rule combine_compound_prt {
  pattern {
    A [];
    B [];
    R: A -[1=compound, 2=prt]-> B;
  }
  commands {
    A.token = A.token + "_" + B.token;
    % Ensure it does not get cleaned.
    A.upos = "VERB";
    del_edge R;
    del_node B;
  }
}
```

(a) Rule to combine phrasal verb particle components, such as *burn down* or *hang up*.

```
rule combine_nouns {
  pattern {
    A [upos=NOUN];
    B [upos=NOUN];
    % Restriction on ordering.
    B < A;
    R: A -[1=compound]-> B;
  }
  commands {
    A.token = B.token + "_" + A.token;
    del_edge R;
    del_node B;
  }
}
```

(b) Rule to combine compound nouns, such as *credit card* or *microwave oven*.

Figure 22: Examples of attribute rules.

Rules that introduce DRS constants based on certain patterns are another example of this. The *time* rule we have seen for instance. UD-Boxer can similarly add a *speaker* or *hearer* constant based on the `Person`[9] UD feature. This morphological feature indicates the type of a pronoun. Figure 23 shows these rules.

```
rule expand_first_person {
  pattern {
    S [upos=PRON, Person=1];
  }
  without {
    S [token="GENDER"];
  }
  commands {
    S.token = "GENDER";
    add_node SC;
    SC.token = "speaker";
    add_edge S -[token="EQU"]-> SC;
  }
}
```

(a) Rule to add a *speaker* constant.

```
rule expand_second_person {
  pattern {
    H [upos=PRON, Person=2];
  }
  without {
    H [token="GENDER"];
  }
  commands {
    H.token = "GENDER";
    add_node HC;
    HC.token = "hearer";
    add_edge H -[token="EQU"]-> HC;
  }
}
```

(b) Rule to add a *hearer* constant.

Figure 23: Examples of attribute rules that add constants.

[9] https://universaldependencies.org/u/feat/Person.html

Finally, we want to highlight how *language-specific* rules are used in UD-Boxer. Some constructions are almost impossible to detect from syntactic and morphological features alone. Two prominent examples are *negation* and *quantifiers*. We have seen how these are encoded in DRGs using boxes and NEGATION edges. In the transformation step, UD-Boxer tries to find these constructions by looking for specific lemmas. Figure 24 lists some of these language-specific rules.

```
rule box_negation_det {
  pattern {
    N [lemma=<list of lemmas>];
    * -[1=advmod|det]-> N;
  }
  without {
    P [token="NEGATION"];
  }
  commands {
    del_node N;
    add_node N_BOX;
    N_BOX.token = "NEGATION";
  }
}
```

(a) Rule to introduce negation box.

EN	no, not, never
NL	niet, geen, nooit
IT	non, mai
DE	nicht, kein, keine, keines, nie

(b) Language-specific negation lemmas.

```
rule box_quantifier {
  pattern {
    N [lemma=<list of lemmas>];
  }
  without {
    P [token="NEGATION"];
    Q [token="NEGATION"];
  }
  commands {
    del_node N;
    add_node N_BOX;
    add_node N_BOX_2;
    N_BOX.token = "NEGATION";
    N_BOX_2.token = "NEGATION";
    add_edge N_BOX -[token="NEGATION"]-> N_BOX_2;
  }
}
```

(c) Rule to introduce quantifier boxes.

EN	every, everyone, everybody, everything, always, all, whoever, whomever, both, whatever
NL	iedereen, elk, elke, alle, alles, altijd, iedere
IT	tutto, tutti, entrambe, entrambi, ogni, ciascuno, qualsiasi
DE	jeder, jedes, jederman, jegliche, alle, alles, stets, beide, immer

(d) Language-specific quantifier lemmas.

Figure 24: Examples of language-specific rules.

These are not all language-specific rules, but the examples demonstrate how these rules are structured. All language-specific rules deal with negation and quantifiers and use a short list of language-specific lemmas. We can see that the rules in Figure 24 do not connect the box nodes to any existing nodes in the graph. This is rather difficult to achieve in this step since we have not resolved the node and edge types at this stage. This also ties into the limitation of UD-Boxer we mentioned previously, it assigns all nodes to the same NEGATION scope. Often, the subject of a sentence that contains negation should not be placed in the negation scope. An example is the *Tom does not have a microwave oven* sentence we have seen before.

Another important aspect to consider in the transformation process is *rule ordering*. The rules in UD-Boxer are roughly ordered in the following way:

1. Add the *token* attribute - this node and edge attribute to store semantic concepts is needed in almost all subsequent rules;

2. Combine nodes - we want to apply this rule early since several later rules should work with the combined node of a multi-word token, instead of its individual parts;

3. Label specific edges - some roles can already be derived from graph patterns, it is not that important where this happens, as long as it happens before the discard rules;

4. Connect user and expand names - this needs to happen after the node combining and before the discard rules;
5. Expand speaker and hearer nodes - same rationale as the previous;
6. Add time - adding temporal information should be done before the discard rules;
7. Apply all language-specific rules - these rules also need to be applied before the discard rules;
8. Apply discard rules - discard all semantically irrelevant nodes and edges;
9. Ensure numbers are constants at leaf nodes - leaf nodes that are numbers should be considered as constants in a DRG, this step makes sure that happens and it needs to be done *after* the discard rules since those can change what the leaf nodes in a graph are;
10. Detach cycles where possible - this is a last step to fix possible errors by detaching two node cycles.

A final important factor to mention is how UD-Boxer deals with multiple sentences. This can occur when multiple sentences exist in the same document. It can also happen when a UD parser makes a mistake in parsing a single sentence. It could introduce wrong sentence boundaries, making it look like multiple sentences. These both result in separate UD graphs per sentence in the same CONLL file.

For multiple sentences, their individual UD graphs are given to GREW and are then resolved one by one. After this, the step of *gluing* multiple graphs together is performed. This is done by going through the graphs in sequential order and connecting their *active* (latest) boxes. Algorithm 1 shows pseudocode for this process. This operation keeps all graphs intact, nothing gets overwritten or removed. This is also why we refer to this as 'glueing' graphs together, instead of *combining* or *merging* for instance. The label the edge connecting these boxes gets assigned *should be* dependent on the semantic content of the graphs. However, for now, this is always `CONTINUATION`, the most frequent box-to-box edge in the training data. Improving this likely requires an additional component in UD-Boxer that receives the 'glued' graph and does another round of edge resolving, specifically for the box-to-box edges. We further discuss this point of improvement in Chapter 5.

Algorithm 1 Pseudocode for 'glueing' DRGs together.

```
1:  listOfGraphs = [G1, G2, G3, ..., Gn]
2:  A = listOfGraphs.popLeft()                  ▷ The initial graph to glue the rest onto.
3:  nodeMapping = hashMap()                     ▷ Needed to prevent overwriting content in A.
4:
5:  for B = listOfGraphs do
6:      for node = B.nodes() do
7:          if node.type == BOX_NODE then
8:              activeBox = A.boxNodes.last()
9:              newNode = A.addNode(node)       ▷ The active box is now the new box.
10:             A.addEdge(activeBox, newNode)
11:         else
12:             newNode = A.addNode(node)
13:         end if
14:         nodeMapping[node.id] = newNode.id
15:     end for
16:
17:     for (fromId, toId) = B.edges() do
18:         A.addEdge(nodeMapping[fromId], nodeMapping[toId])
19:     end for
20: end for
```

4.5 SUBSTITUTING SYNTACTIC LABELS WITH SEMANTIC CONCEPTS

UD-Boxer uses various mappings to substitute syntactic labels with semantic concepts. The main mappings are:

- UD UPOS tag to WordNet POS tag - for creating synset ids;
- UD morphological Tense to DRS tense - for resolving temporal relations;
- UD morphological Gender to synset - for resolving persons and entities;
- UPOS tags and dependency to role or operator - for general edge labeling;
- Lemma and WordNet POS tag to sense - for creating synset ids;
- Lemma to sense - for creating synset ids.

The first three mappings are rather straightforward and do not require training data. Figure 25 shows these in full.

FUT	→	TSU
IMP	→	TPR
PAST	→	TPR
PQP	→	TPR
PRES	→	EQU

(a) Morphological *Tense* to DRS tense mapping.

ADJ	→	a	PART	→	r
ADP	→	n	PRON	→	n
ADV	→	r	PROPN	→	n
AUX	→	v	PUNCT	→	n
CCONJ	→	n	SCONJ	→	n
DET	→	n	SYM	→	n
INTJ	→	n	VERB	→	v
NOUN	→	n	X	→	n
NUM	→	n			

(b) UPOS tag to WordNet POS tag mapping.

COM	→	person.n.01
FEM	→	female.n.02
MASC	→	male.n.02
NEUT	→	person.n.01

(c) Morphological *Gender* to synset mapping.

Figure 25: Basic mappings to get semantic concepts or information from syntactic features.

Most are self-explanatory if one is familiar with UD. Some of the POS tag mappings are not ideal. `PART` can refer to a lot of semantically different tokens for instance.

The next three mappings are created automatically using training data. All mappings are extracted using the same method. This method, while rather simple in its approach, is somewhat intricate to describe.

To start the extraction process, we take the UD graph and gold DRG for a particular document from our training data. Next, we apply the transformation and substitution steps. If the resulting graph and gold graph are isomorphic, we iterate through the nodes and edges of both. While doing this, we store *correct* semantic labels and their corresponding syntactic labels from the UD graph for later use. This is what makes this process intricate to describe. To start the process, we need to bootstrap it in an already working system. This was done by first storing *all* mappings from isomorphic graphs, not just the correct ones. The better the graph transformations become, the better the mapping extraction also becomes.

The dataset created by the mapping extraction consists of all available information per triple, so all UD and DRG information belonging to the *from node*, the *edge* and the *to node*. This dataset can be used for many different approaches. For the current project, for example, an experiment was done with training an edge classifier with this data based on UD information. However, this proved to be quite difficult since a lot of the UD triples are ambiguous and can map to multiple semantic concepts. For this reason, and to stay with the transparency and explainability design goals of UD-Boxer, we chose the mapping approach instead. This approach mainly serves as a baseline of sorts and is certainly an area of improvement.

We mentioned the edge mappings before in the examples. These are based on the most frequent counts of triples of attributes in the training data. They are formatted as follows:

Definition:	`UPOS-deprel-UPOS`	→	`Role` or `DRS operator`
Example:	ADJ-obl-NOUN	→	Stimulus
Example:	ADJ-nsubj-NOUN	→	AttributeOf

Again, this is by no means ideal, but most common constructions are covered quite well by this method. This also results in an entirely transparent and easily editable file with mappings. Recall that the edge resolving only happens if no valid label was applied in the transformation step. These mappings are a second-tier fallback of sorts, with the default edge label (role) being the final fallback. An advantage of this approach is that these mappings are language-neutral, which is also how they are used in UD-Boxer. The results in Chapter 5 were obtained with 124 automatically extracted mappings. These all stem from the English gold training data.

Next are the node mappings. These are also based on the most frequent counts in the training data. This mapping *is* language-specific since the lemmas in synsets are almost exclusively in English in WordNet. Some foreign words (compared to English) are present in WordNet, but WordNet is intended as an English resource. All node mappings are based on the lemmas from the UD parse, which includes the combined lemmas from the transformation step. UD-Boxer has two strategies to get a sense number for a given synset node. First, it looks in its mappings with both the lemma and WordNet POS tag. This makes the lookup rather fine-grained. These mappings are formatted as follows:

Definition:	`lemma.wordnet-pos`	→	`Synset`
Example:	badger.n	→	badger.n.02
Example:	badger.v	→	badger.v.01

This is a real example and shows the value of including the POS tag in the lookup. The first example refers to the animal and the second to rudely walking somewhere. If this lookup fails, due to an unknown lemma and POS tag combination, UD-Boxer tries again with just the lemma. These are very similar to the previous mappings:

Definition:	`lemma`	→	`Synset`
Example:	serious	→	serious.a.06
Example:	finish_off	→	finish_off.v.01

Some experiments were done with adding silver data, in addition to the gold training data, to extract the mappings. This did not improve results, using only gold training data performed better on the development and training data splits. This might indicate the bottleneck of this approach, with too much data the ambiguity overpowers any fine-grained distinctions, which get flattened into a single lookup anyway. Table 8 shows the number of mappings per language. We can clearly see the difference in available data per language here as well. Despite this, results are decent for the languages with far less data than English. We discuss this in the next chapter.

Table 8: Number of mappings per language.

	Lemma + POS	**Lemma only**
English	4,054	3,831
Dutch	355	355
Italian	479	479
German	823	819

4.6 DEVELOPMENT PROCESS

UD-Boxer has been through various revisions throughout its development. Initial attempts for the graph transformation process were done without GREW. We started by traversing the UD graph and recursively applying transformations to it until it was isomorphic to the gold SBN graph. This worked decently but was hard to maintain and extend since transformation ordering became cumbersome to work with.

At this point, we came across GREW and the extensive tooling surrounding it. The main development of the GREW rules was done by picking interesting examples from the training data and experimenting with those. These examples were chosen based on certain phenomena in gold DRGs, such as named entities, multi-word tokens, possessive constructions, multiple entities, negation, quantifiers, attribute roles, multiple sentences, certain box constructions and so on. Similarly, some specific UD examples were also chosen based on interesting phenomena on the UD side. These mainly originated from GREW Match[10], an interactive treebank query tool that uses user-defined GREW patterns to search in treebanks. Once the system was working decently, we added and improved rules based on the lowest scoring examples from the train and dev data splits per language.

Rules were developed by having the UD graph and gold SBN graph side-by-side for a given example sentence. We then tried to create the most general and simple rules that (structurally) transformed the UD graph into the SBN graph. The GREW Transform[11] tool was very helpful in this regard, due to its ability to show the results of intermediate rule applications. It also helped considerably in developing the rule ordering. All rules used in UD-Boxer are included in Appendix A.3. These also include more detailed explanations and provide examples from the PMB dataset.

The mapping approach has stayed largely the same during development. The main additional development done for this component was increasing its granularity. Initial attempts mapped just the `deprel` label to a semantic label for instance. This was not accurate at all, so the `UPOS-deprel-UPOS` mapping was developed instead. The same goes for the sense mappings. The WordNet POS tag addition to the sense lookup was added later to increase granularity.

[10] http://match.grew.fr/
[11] http://transform.grew.fr/index.html

4.7 NEURAL COMPARISON SYSTEM

To assess the results from UD-Boxer, and to answer our main research question, we need a neural comparison system. We base our neural system on the current state-of-the-art work done by van Noord et al. (2020)[12]. Our particular sequence-to-sequence parser consists of an English `BERT` (Devlin et al., 2019) model that is trained and finetuned on SBN from the `4.0.0` version of the PMB[13]. Note that the systems described in van Noord et al. (2020) did not use SBN as their target format, they instead targeted the clause notation for DRSs. Apart from this, the method is the same: the model receives a raw sentence as its input sequence and the output sequence it produces is a DRS notation, SBN in our case.

We developed two versions of this neural parser: one trained and finetuned on just gold data and one on both gold and silver data. Note that we only created such a system for English due to time constraints[14]. The neural parser has been named *Neural-Boxer* in the current project. This in keeping with the *Boxer*, *Neural-Boxer* and *UD-Boxer* naming convention. Neural-Boxer and UD-Boxer are the first semantic parsers to target SBN directly.

[12] We want to thank the main author of van Noord et al. (2020), dr. Rik van Noord, for helping to train and finetune this model.

[13] This parser is based on: `https://github.com/RikVN/Neural_DRS`.
How to run the parser with SBN:
`https://github.com/RikVN/Neural_DRS/blob/master/AllenNLP.md#sbn-experiments`.

[14] After this thesis was handed in, results for other languages for the neural approach were obtained. They are shown in Table 13 in Appendix A.2.

5 RESULTS AND DISCUSSION

In this chapter, we evaluate the performance of UD-Boxer on all four languages. We present the results we need in order to answer our research questions. We also compare the results of UD-Boxer and Neural-Boxer, just for English. We carry out a detailed analysis to see the effect of input length on performance, as well as how the systems deal with negation. Next, we perform an in-depth error analysis on output from both UD-Boxer and Neural-Boxer. Lastly, we discuss the implications of these results and outline areas of improvement.

5.1 GENERAL RESULTS

Tabel 9 shows results per UD parser, language and data split. We need these results for RQ1. All scores are on macro level, i.e., the average across a data split. Ill-formed output gets assigned all zeroes for the precision, recall and F1 metrics. Note that only English has a separate *eval* data split.

Table 9: UD-Boxer results per data split, language and UD parser (Stanza and Trankit). Scores are on macro level. 'Err' refers to the percentage of ill-formed graphs a system produced for that data split. No output is also considered to be ill-formed.

		Dev				**Test**				**Eval**			
		P	**R**	**F1**	**Err**	**P**	**R**	**F1**	**Err**	**P**	**R**	**F1**	**Err**
Stanza	English	83.6	81.6	82.1	0.3%	83.8	81.4	82.0	0.0%	83.1	80.5	81.3	0.5%
	Dutch	78.0	74.5	75.5	0.0%	77.3	75.4	75.8	0.0%	-	-	-	-
	Italian	79.5	74.2	76.2	1.9%	81.2	76.6	78.4	0.9%	-	-	-	-
	German	80.8	77.2	78.4	0.0%	80.1	75.7	77.3	0.0%	-	-	-	-
Trankit	English	83.7	81.3	81.9	0.3%	83.6	81.1	81.8	0.0%	83.6	80.7	81.5	0.0%
	Dutch	77.8	74.8	75.8	0.0%	77.1	75.6	75.8	0.0%	-	-	-	-
	Italian	80.8	76.0	77.8	0.0%	81.9	77.3	79.1	0.0%	-	-	-	-
	German	79.9	77.9	78.4	0.0%	79.2	76.8	77.5	0.0%	-	-	-	-

UD-Boxer scores slightly higher with Stanza for English, while the other languages perform better with Trankit or are tied between the two parsers. When looking at the percentage of ill-formed output (*Err*), we see that UD-Boxer almost always generates well-formed output.

It is interesting to see that UD parses from Trankit seem to be more robust that Stanza's for Italian. Trankit has no errors for Italian, but Stanza has rather high error percentages across both data splits. This indicates that UD-Boxer is not the problem in this particular case. For English, the results of both UD parsers have the exact same error count for the dev split, which could indicate a problem with UD-Boxer.

However, at closer inspection, the 0.3% error percentage consists of three errors for Trankit and three for Stanza, of which only one overlaps[1]. Except for the overlapping one, all errors are due to unusual UD parses. In the next section, we go into more detail regarding these errors.

We can see that English scores the highest across the board. This is likely due to the large amount of gold data that is available for English, compared to the other languages. Dutch has seven times less gold data compared to English, Italian six and German four. Still, even with considerably less gold data, the scores are not

[1] Stanza: `en/gold/p30/d1768`, `en/gold/p40/d2942`, `en/gold/p10/d3030`.
Trankit: `en/gold/p10/d3099`, `en/gold/p10/d3084`, `en/gold/p10/d3030`.

that far behind English. Additionally, *all* languages across *all* data splits manage F1-scores of >75.0, which shows UD-Boxer's potential. The (almost) language-neutral approach is already quite decent and there are clearly identifiable areas of improvement, which underlines this potential even more.

5.2 COMPARISON WITH A NEURAL SYSTEM

The scores we have seen so far are not that insightful without a system to compare against. We also need this comparison to answer our main research question. Let us take a look at how UD-Boxer stacks up to the performance of Neural-Boxer[2]. Table 10 shows the results for English for UD-Boxer per UD parser. It also shows results for Neural-Boxer per training dataset and evaluation method. As we have discussed in Section 3.3, the strict evaluation method considers wrong indices to be ill-formed and lenient considers them to be constants. Ill-formed graphs, regardless of the reason why they are ill-formed, always receive all zeroes for all metrics. We do not provide strict results for UD-Boxer since it achieves identical scores between the two evaluation methods. In addition, it is technically impossible for UD-Boxer to make these index errors.

Table 10: UD-Boxer and Neural-Boxer comparison for English. Results from both UD parsers (Stanza and Trankit) are shown for UD-Boxer. Results from Neural-Boxer trained on just gold data and trained on both gold and silver data are shown. Strict scores refer to the strict evaluation mode regarding ill-formed graphs. Scores are on macro level. 'Err' refers to the percentage of ill-formed graphs a system produced for that data split. No output is also considered to be ill-formed.

	Dev				**Test**				**Eval**			
	P	**R**	**F1**	**Err**	**P**	**R**	**F1**	**Err**	**P**	**R**	**F1**	**Err**
UD-Boxer (Stanza)	83.6	81.6	82.1	0.3%	83.8	81.4	82.0	0.0%	83.1	80.5	81.3	0.5%
UD-Boxer (Trankit)	83.7	81.3	81.9	0.3%	83.6	81.1	81.8	0.0%	83.6	80.7	81.5	0.0%
Gold												
Neural-Boxer	84.0	82.1	82.8	4.6%	85.5	83.0	84.0	3.7%	83.4	80.8	81.7	3.9%
Neural-Boxer (strict)	81.1	79.4	80.0	8.3%	83.4	81.1	82.0	6.4%	79.9	77.6	78.4	8.3%
Gold + Silver												
Neural-Boxer	92.8	92.4	92.5	2.0%	93.0	92.3	92.5	2.3%	92.4	91.5	91.8	2.8%
Neural-Boxer (strict)	92.0	91.5	91.6	3.1%	92.4	91.7	91.9	3.0%	91.0	90.3	90.5	3.7%

We can see that Neural-Boxer, trained on gold and silver, outperforms everything else by quite a margin. However, it still produces considerably more ill-formed output than UD-Boxer, even with the lenient evaluation method. These scores are impressive but do not tell the whole story, as we will see in Section 5.5.

Neural-Boxer trained on just gold data and UD-Boxer perform very similarly. This Neural-Boxer system produces considerable amounts of ill-formed output. It is also struggling quite a bit in the strict evaluation mode. Less data is, unsurprisingly, noticeably detrimental to the neural method.

5.3 PERFORMANCE BY INPUT LENGTH

An important factor to consider when evaluating a semantic parser is how well it deals with input sequences of various lengths. We need these results to answer RQ2. As we have mentioned, neural sequence-to-sequence models are somewhat notorious for their drop-off in performance for long inputs (Press et al., 2021). As

[2] After this thesis was handed in, results for other languages for the neural approach were obtained. They are shown in Table 13 in Appendix A.2.

we have seen in Section 3.1, the PMB primarily consists of short sentences. Nevertheless, it is interesting to look at how UD-Boxer and Neural-Boxer perform on different input lengths. Figure 26 shows F1-scores by the number of characters in the input sequence.

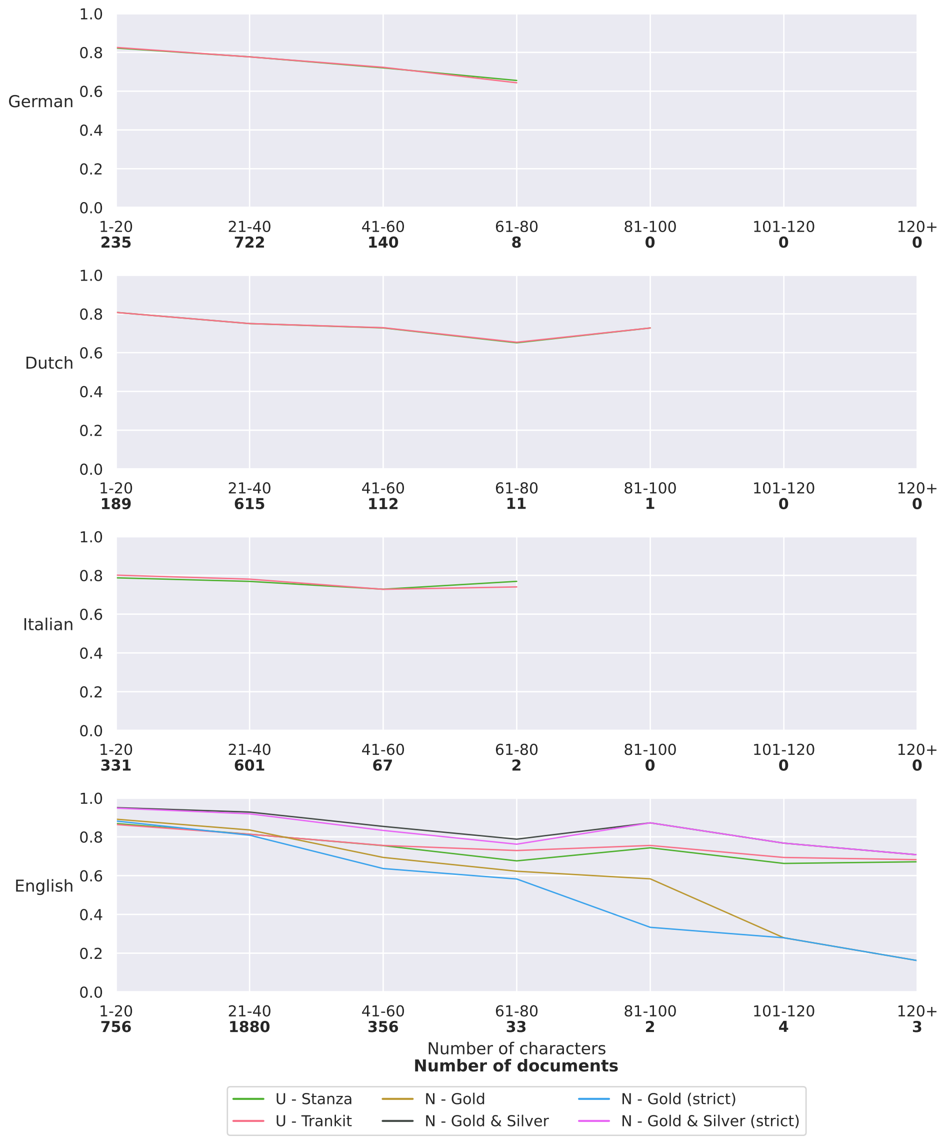


Figure 26: Macro F1-score by input length, per system, for all languages. Results from both UD parsers (Stanza and Trankit) are shown for UD-Boxer. Results from Neural-Boxer trained on just gold data and trained on both gold and silver data are shown. Strict scores refer to the strict evaluation mode regarding ill-formed graphs. The data is an aggregate of dev and test splits (eval as well for English). The number of characters is based on the `*.raw` files from the PMB, stripped from any leading or trailing whitespace. In bold are the number of documents per grouping.

We can see that performance for all languages drops at about the same rate. Additionally, except for English, the maximum sequence length in the dataset is about 80 characters (~11 tokens). This makes it difficult to make substantial claims about the performance of the systems on those languages at those lengths. The low document count at those lengths also coincides with this. The trend for all languages and the drop-off is about the same. We can likely assume similar patterns to English at longer lengths for those languages, only with slightly lower scores across the board.

The results for English show the most interesting patterns. All systems start at roughly the same performance level. At around the 45-character mark we can see a drop-off in performance for Neural-Boxer trained on only gold data. UD-Boxer is quite stable in its performance, regardless of UD parser. There are no big spikes or dips, just a general downwards trend. Neural-Boxer trained on gold and silver data generally outperforms everything else. UD-Boxer stays somewhat competitive though, while Neural-Boxer trained on just gold data lacks behind. This is quite clear up until the 61-80 character mark. This trend continues afterwards, but due to the low document count at those lengths, we cannot reliably draw conclusions from it.

Figure 27 shows a similar plot, with scores by number of *tokens* instead of number of *characters*.

This figure shows similar patterns as Figure 26. The slight downwards trend for all systems is also visible here. The trend across languages for UD-Boxer is similar once again. For English, we see that Neural-Boxer trained on gold and silver data outperforms the other systems, although not by a huge margin. Neural-Boxer trained on just gold data and UD-Boxer perform very similarly. However, this Neural-Boxer drops off after the 9+ token mark, while UD-Boxer still shows decent performance. Although, again, it is hard to make substantial claims about this due to the low document counts.

The main findings from Figure 26 and 27 are: UD-Boxer shows similar performance patterns across languages, Neural-Boxer trained on gold and silver data outperforms everything else and Neural-Boxer trained on just gold data performs very similarly to UD-Boxer up to a certain point, after that it tapers off.

Sadly, we do not have Neural-Boxers trained on languages other than English[3]. Still, one can imagine the performance considering the large differences in gold data that is available for the other languages when compared to English. As we have mentioned in Section 3.1, Dutch has seven, Italian six and German four times less gold data available, compared to English.

5.4 NEGATION PERFORMANCE

Since negation is a crucial phenomenon in semantics, it is good to take a look at how UD-Boxer and Neural-Boxer deal with it. Especially since negation is also used to model quantifiers in DRGs. We need these results to answer our third research question. As we have seen before, UD-Boxer's implementation of negation is rather basic. It simply looks for some particular lemmas combined with certain dependency patterns and, based on that, inserts a `NEGATION` box. This box is just connected to the previously inserted box, so no proper negation scope is assigned. Nonetheless, we can still evaluate how well it performs in *detecting* negation. We do this in two ways. One is to frame the negation detection as its own information retrieval task. With this, we can list precision, recall and F1-scores for just that component. The other is to look at how well correctly detected DRGs containing negation perform using SMATCH. For both evaluation methods, we split the scores per number of negation clauses. Table 11 shows the counts of negation clauses in our data.

We can see that there are not many negation clauses in the data. Still, we can gain some insight into how the systems handle these. Since there are very few documents with three or four clauses, we will not include those in the results. Table 12 shows the results of the negation evaluation.

[3] General performance results for the neural approach on other languages were obtained after this thesis was handed in. They are shown in Table 13 in Appendix A.2. An analysis of the effect of input length was not done for these results.

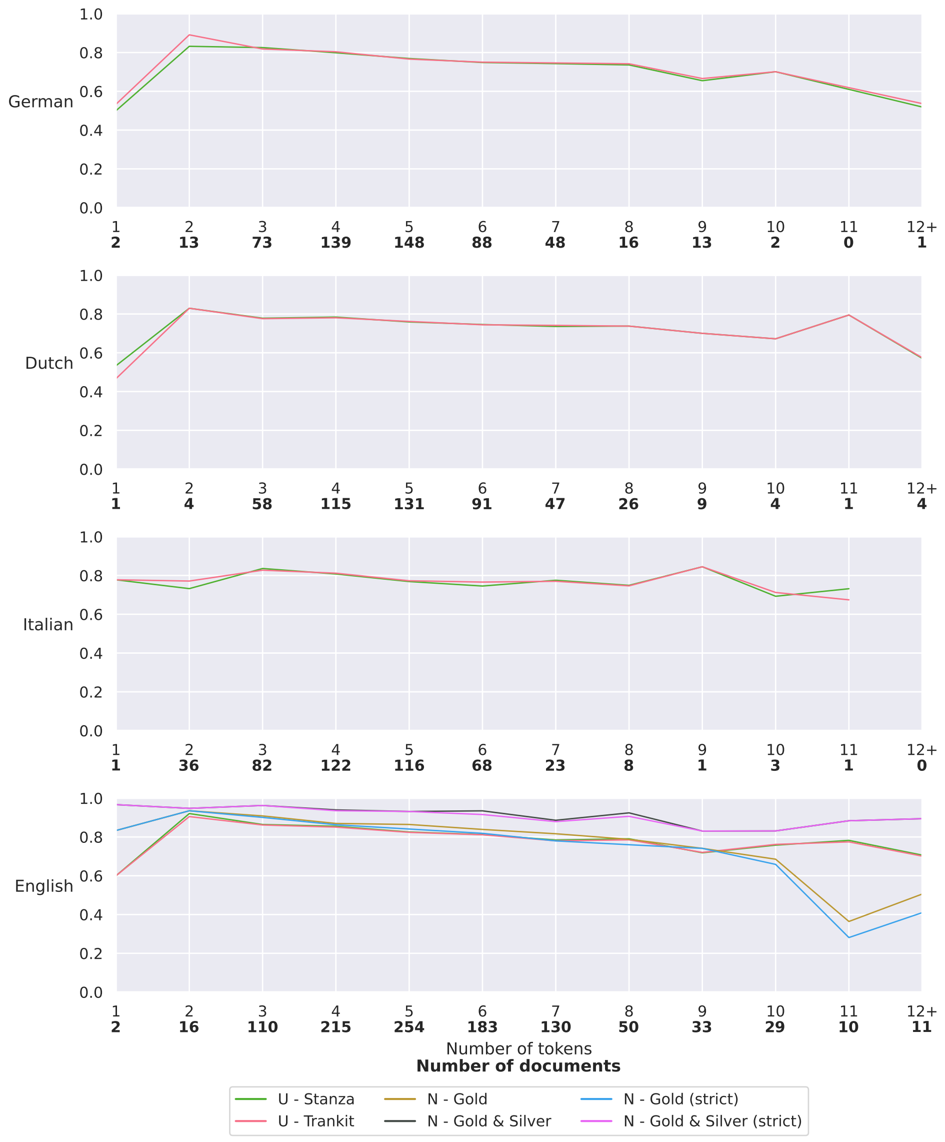


Figure 27: Macro F1-score by input length, per system, for all languages. Results from both UD parsers (Stanza and Trankit) are shown for UD-Boxer. Results from Neural-Boxer trained on just gold data and trained on both gold and silver data are shown. Strict scores refer to the strict evaluation mode regarding ill-formed graphs. The data is an aggregate of dev and test splits (eval as well for English). Tokens are based on `*.tok.off` files from the PMB, note that punctuation marks are counted as single tokens in those files. In bold are the number of documents per grouping.

The results are quite varied and this is partially due to the limited amount of data. However, between systems, there is also a lot of variation. We want to highlight one surprising result from the neural parsers that is not shown in the table. This is a single DRG with six negation clauses, identically produced by both Neural-Boxer systems for the sentence *Everyone knows everyone*[4], the gold standard for that sentence has four negation clauses.

First let's discuss the negation detection, as seen in the left main column of Table 12. Neural-Boxer trained on gold and silver data outperforms everything yet again in terms of F1-scores. UD-Boxer fares rather well overall. Neural-Boxer trained on just gold data lacks behind in almost everything. We can see that preci-

[4] PMB id: `en/gold/p01/d2144`.

Table 11: Number of negation clauses per language. The data is an aggregate of dev and test splits (eval as well for English).

	# Negation clauses				
	Zero	**One**	**Two**	**Three**	**Four**
English	2,676	253	93	7	5
Dutch	841	65	20	2	-
Italian	886	80	34	1	-
German	1,005	80	19	-	1

Table 12: Negation performance of UD-Boxer and Neural-Boxer, per language, per number of negation clauses. Results from both UD parsers (Stanza and Trankit) are shown for UD-Boxer. Results from Neural-Boxer trained on just gold data and trained on both gold and silver data are shown. Strict scores are left out since they were identical to the lenient scores. The data is an aggregate of dev and test splits (eval as well for English). Scores are on macro level.

		Negation detection performance per # negation clauses.						**True positive DRG scores per # negation clauses.**					
		One			**Two**			**One**			**Two**		
		P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
U - Stanza	English	94.3	85.8	89.9	78.0	83.9	80.8	82.7	80.0	80.7	85.3	72.0	77.5
	Dutch	98.4	95.4	96.9	91.7	55.0	68.7	75.8	74.1	74.3	85.3	70.6	77.0
	Italian	98.7	96.3	97.5	100.0	61.8	76.4	82.1	77.9	79.4	81.9	67.4	73.6
	German	100.0	68.8	81.5	83.3	52.6	64.5	80.4	77.3	78.2	86.6	73.8	79.6
U - Trankit	English	98.3	47.0	63.6	78.8	83.9	81.2	84.1	80.2	81.4	86.3	71.7	77.8
	Dutch	98.4	95.4	96.9	91.7	55.0	68.7	75.5	74.4	74.4	85.4	71.0	77.3
	Italian	98.7	97.5	98.1	100.0	61.8	76.4	81.6	77.7	79.1	82.3	67.7	73.9
	German	100.0	68.8	81.5	83.3	52.6	64.5	79.9	78.0	78.4	86.6	73.8	79.6
N (gold)	English	76.5	81.0	78.7	78.0	41.9	54.5	91.4	88.3	89.6	87.6	87.5	87.2
N (gold + silver)	English	96.4	95.3	95.8	94.1	86.0	89.9	96.1	96.4	96.2	94.8	94.4	94.4

sion is generally quite high for UD-Boxer, which indicates that the current language-specific lemmas are quite good. Recall is not always as high, indicating that UD-Boxer is missing some lemmas or constructions, especially for German. This lower recall is also noticeable between Stanza and Trankit for English. This is quite surprising since the overall scores between these parsers are quite similar. When taking a closer look, we see that Trankit treats lemmas of contractions differently compared to Stanza. Consider the token *didn't* for instance. Stanza splits this into two tokens and lemmatizes both, resulting in *do* and *not*. Trankit does not do this and treats it as a single token, which then stays the same during lemmatization. UD-Boxer was designed with the Stanza way of parsing in mind and this difference slipped through the cracks in development. In fairness, UD states in their guidelines that these *multi-word tokens* should be split up *or* it should be well documented in case they are not[5]. At the time of writing, this is not the case for Trankit[6]. This effect is only visible for English since it has plenty of negation-related contractions (*doesn't, didn't, don't, hasn't, won't etc.*), whereas the other languages do not.

Lastly, let us examine the performance of the parsers in case they *did* insert negation. These are the SMATCH scores of the graphs where the number of clauses was detected correctly. In other words, the true positives of the negation detection task. The second main column in Table 12 shows these results. This is, however, not a full analysis of evaluating scope, it is instead indirectly evaluated when comparing the graphs with SMATCH. For this reason, we cannot draw full conclusions from these scores regarding scope. The comparison between the systems is valuable nonetheless. We can see that UD-Boxer lacks behind quite a bit in this area. This is largely due to the complete absence of assigning negation scope. The Neural-Boxers

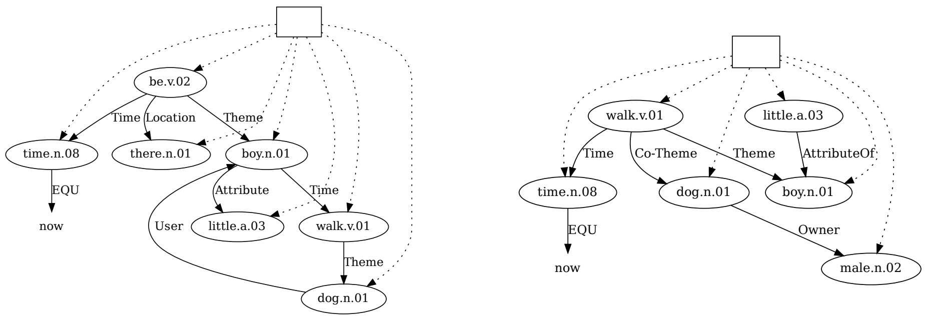
5 https://universaldependencies.org/u/overview/tokenization.html
6 https://trankit.readthedocs.io/en/stable/lemmatize.html

seem to fare well in this regard, especially with a single negation clause. This is a clear area of improvement for UD-Boxer.

5.5 ERROR ANALYSIS

Recall the UD-Boxer error from page 37 that went wrong for both Stanza and Trankit for English. Let us take a closer look at it since it contains several interesting mistakes.

The sentence in question is *There is a little boy walking with his dog*. The Stanza and Trankit parses are identical and not unusual. The problem is that UD-Boxer generates a cyclic graph. As we have discussed in Section 3.3, this is not allowed by the SBN specification, as well as the Penman format, which we need for evaluation. Figure 28 shows the ill-formed output as well as the gold graph.



(a) Ill-formed output.

(b) Gold graph.

Figure 28: Ill-formed (cyclic) UD-Boxer output and gold graph for the sentence *There is a little boy walking with his dog*. PMB id: `en/gold/p10/d3030`.

We can see that UD-Boxer tries to connect the *User* of the dog. This is done by connecting the target of an outgoing `nmod:poss` edge to the `nsubj` node of the graph. In a lot of cases, this goes well, such as in our running example from Chapter 4. However, the further away the target of the `nmod:poss` edge is from the subject, and the more nodes are present *after* the target, the higher the risk of creating a cyclic graph becomes. There is a rule that detaches possible cycles, but it only looks at cycles between two direct nodes, here we are dealing with a cycle spanning three nodes.

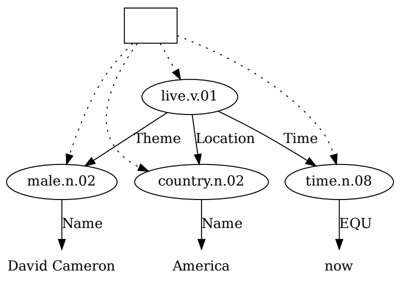
We can see some other interesting mistakes in this example. Note that in the gold graph *walking* is considered to be the root (or primary verb), while in both UD parses *is* is considered to be the root. This leads to a different structure. The *Time* relation from *boy* to *walk* shows the weak spots of the mapping approach. This particular node-edge-node mapping (`NOUN-obl-VERB`) happens to be highly ambiguous. Lastly, we can see that UD-Boxer adds a location *there* to the graph. Although it is slightly awkward English[7], this interpretation of the sentence can be read as *A little boy is walking with his dog* ***there***, while the gold graph views the sentence as a general event without a particular location. This likely also coincides, accidentally or not, with the `be.v.02` node being there since it is, arguably, required for this interpretation of the sentence.

As mentioned, the high scores of Neural-Boxer do not tell the whole story. While UD-Boxer can certainly output meaning representations of low quality, it always stays close to the input sentence. This is of course no surprise since the only information it uses stems from a UD parse. This means that a bad meaning representation from UD-Boxer is mainly due to constructions that are too complex, poor

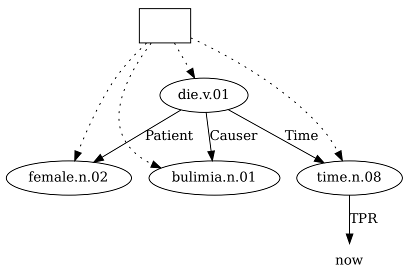
7 A native Dutch speaker might produce such a sentence.

node and edge labels or incorrect word-sense disambiguation. These all result in a meaning representation that is either lacking in its expression or wrongly describes the input sentence. The previously mentioned default values used by UD-Boxer are a good example of this. Specifically, where it assigns *female.n.02* to all PROPNs it cannot resolve properly. This leads to a bad representation, but we can identify why it happened and what can improve it (a *Named Entity Recognition (NER)* component). In addition, it does not suddenly add new constants or concepts.

When we look at Neural-Boxer, this becomes a different matter. While skimming through the output from system experiments, we noticed some strange *hallucinations* produced by Neural-Boxer. Figure 29 shows two interesting examples that give some idea of this problem. We want to give the system the benefit of the doubt, so we only consider output from the best Neural-Boxer model, trained on both gold and silver data.



(a) DRG for *David Beckham now lives in America.* PMB id: `en/gold/p51/d3003`. *(F1 96.7)*



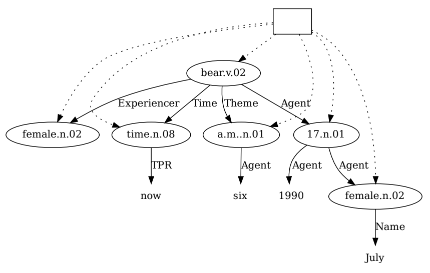
(b) DRG for *She died from tuberculosis.* PMB id: `en/gold/p80/d2937`. *(F1 96.3)*

Figure 29: Hallucinations from Neural-Boxer trained on gold and silver data.

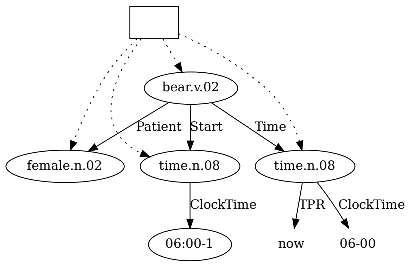
The output is well-formed and scores high. However, the meaning of these DRGs is *drastically* different from the input sentence and we do not know why. In Figure 29a, Neural-Boxer decides the sentence is suddenly about an entirely different person. Figure 29b lists a completely different cause of death. We cannot go through all examples here, but there are a remarkable number of hallucinations in the neural output. Again, both examples are produced by the best model trained on both gold and silver data. This approach is almost identical to the current state-of-the-art method in DRS parsing. This begs the question of what other unexplainable errors are hidden behind high scores.

As for the cause of these particular hallucinations, it is likely due to frozen BERT layers or bias in the training data. The model might be recognizing certain sequences of tokens it has seen often during training. In our first example, it might see *David* and continue with *Cameron*, since it has seen that particular combination more often than *David* and *Beckham*. This is likely similar to the second example, where it saw *died from*, which happened to result in *bulimia* being the most likely token to follow it, instead of *tuberculosis*. However, this is all guesswork since we cannot ask the model or look inside it to figure out why it chose these particular tokens. This is something we can easily do with UD-Boxer, as we have seen numerous times.

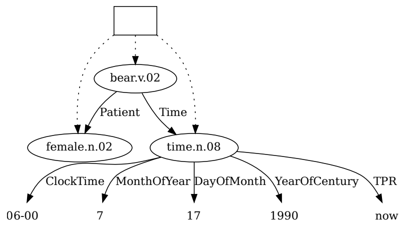
Next, let us take a look at a sentence both UD-Boxer and Neural-Boxer struggle with. Figure 30 shows an example of this.



(a) UD-Boxer produced DRG. *(F1 56.7)*



(b) Neural-Boxer produced DRG. *(F1 76.6)*



(c) Gold DRG.

Figure 30: UD-Boxer, Neural-Boxer (gold + silver) and gold DRG for the sentence *She was born at six a.m. on July 17, 1990*. PMB id: `en/gold/p90/d2854`.

We can see that both UD-Boxer and Neural-Boxer have a hard time parsing this sentence. UD-Boxer tries to interpret parts of the date expressions as synsets, which causes problems. Neural-Boxer loses the year, month and day. It also adds two time nodes and two `06:00` times, which is rather strange. Lastly, UD-Boxer indicates that the person being born is the *Experiencer* of this, while Neural-Boxer produces the correct label for this edge, namely *Patient*.

In Appendix A.1 we provide some additional error analyses. These include: strict versus lenient results, more interesting poor results and good results from both systems.

6 CONCLUSION

In this chapter, we answer our research questions, discuss limitations of our approach and indicate directions for future work.

6.1 OVERALL CONCLUSIONS

Before we answer our research questions, we want to give a quick summary of our approach and what we have discussed so far. The main strengths of our approach are:

- It supports four languages and is easily extendible to more;
- It shows promising performance, especially considering little training data is available for languages other than English;
- It is fully explainable;
- It produces very little ill-formed output;
- It is decent at handling long input sequences.

The main drawbacks of our approach are:

- Node and edge labeling is basic;
- No actual word-sense disambiguation is done;
- No proper scope is assigned to negation and quantifier constructions;
- Named entities and date and numeric expressions are not handled well.

We go into more detail regarding these points when we discuss directions for future work. With these lists and the previous chapter in mind, let us answer the research questions of this project.

First, our main research question:

> *How do Discourse Representation Structures derived from Universal Dependencies using a graph transformation approach compare to those created by a fully neural sequence-to-sequence model?*

As we have seen, the answer to this question depends on multiple factors. In terms of scoring, if there is a lot of training data available for the neural method, our approach lacks behind. If there is not a lot of training data, our approach performs very similarly to the neural method, even beating it in some aspects.

As we have also seen, scoring alone is not the full story. The neural approach has shortcomings that need to be taken into consideration. For one, both neural systems produce far more ill-formed output compared to ours, even with the lenient evaluation method. Another point to consider is the rather poor performance regarding negation detection of Neural-Boxer trained on just gold data. The overall performance of this model also drops noticeably for longer input sequences. The final point to keep in mind are the *hallucinations* of both neural models. This point is arguably the most devious one since SMATCH scores can be very high for a given graph with one or more hallucinations. In fact, this is often the case. It is hard to detect these with automatic measures. The lack of explainability of the neural systems only furthers this problem. Our system can also produce bad meaning representations, but it does not suddenly introduce new constants or entirely new concepts that were not present in the input sentence. The cause of these bad representations can also be fully traced and explained in UD-Boxer.

So, to answer our main research question:

> *Discourse Representation Structures derived from Universal Dependencies, using a graph transformation approach can come close to or beat a fully neural model, but only when little data is available to train the neural model. The output of our approach is almost always well-formed, which is not true for the neural method. Lastly, the neural method lacks explainability and can produce strange output. These shortcomings are noticeable even when there is a lot of training data available for the neural method.*

This answer does not tell us anything about the different aspects of the system and their performance. So, to provide more background to the previous answer, let us examine the sub-questions. Our first sub-question:

> RQ1: *To what extent would such an approach be language-neutral or easily transferrable to multiple languages?*

We have seen that it is basically impossible to do our approach without any language-specific features. Some crucial constructions require these features, such as negation and quantifiers. These cannot be derived from the language-neutral information in a UD parse alone. Additionally, the node labels, specifically the synsets, also require some language-specific training data. However, both of these are, arguably, very simple to implement. For negation and quantifier detection, we re-use four identical rules with their only language-specific part being a list of five to ten lemmas per rule. The only work required when adding new languages in this regard would be to add some lemmas.

As for the sense number labeling, these are automatically extracted when some training data is available. Still, even this is not strictly necessary as the system can function fine without it. In that case, it would simply not attempt to do any word-sense disambiguation and consider everything to be the `01` sense. Considering WordNet sense numbers are based on frequencies and a substantial number of synsets have just a single entry, this would not result in a drastic loss of performance.

Lastly, we have seen that our approach works well even with very little training data. Sadly, we did not train neural models for the other languages[1]. However, the large difference in gold data availability for the other languages would very likely have a much larger effect on the neural method compared to ours. We can even see this happen for English when comparing the neural system trained on just gold data and the one on gold and silver data.

So, to answer RQ1:

> *Such an approach can almost be fully language-neutral. Detecting negation and quantifier constructions and labeling synset nodes requires language-specific information. These parts are, however, small, sometimes even optional and require minimal work. We can say that the vast majority of such a system is language-neutral.*

Let's examine our next sub-question:

> RQ2: *How well does such an approach deal with input sequences of various lengths?*

[1] These results were obtained after this thesis was handed in. They are shown in Table 13 in Appendix A.2. Our assumption here was correct and our system outperforms all neural systems for all languages except English. Surprisingly, this is even true when there is quite a lot of training data available for the neural systems for those languages.

We have seen that our approach shows similar performance trends across languages. It starts high and gradually tapers off the longer the input sequences get. This pattern is roughly the same when looking at the number of characters, as well as the number of tokens in a given input sequence. When compared to the neural systems, we see that the neural system trained on silver and gold data outperforms ours, but not by a large margin. Our system stays competitive at long input sequences. All systems start at roughly the same performance level. However, the neural system trained on just gold data noticeably drops off at some point. Our system and the other neural system, trained on gold and silver data, remain quite stable at longer input sequences.

So, to answer RQ2:

> *The performance of such an approach gradually tapers off the longer the input sequences get. It shows this trend across languages. Overall, the performance is quite stable and the drop-off is relatively small.*

Lastly, let's examine RQ3:

> RQ3: *How well does such an approach handle negation and quantifiers constructions?*

As mentioned, our approach currently can only *detect* negation and quantifiers, it cannot properly resolve their scope. If we look at just the detection, it holds up well and considerably outperforms the neural approach trained on just gold data. The neural system trained on gold and silver data outperforms ours in this regard by a decent margin. Across languages, we see that detecting a single negation clause performs well for our approach. There is room for improvement in detecting two clauses (quantifiers) however. Precision is generally very high, from this we can conclude that the current language-specific lemmas (cues) are good. Recall can be improved, especially for two clause detection, which means our system is missing some lemmas. When we look at the scores of the graphs where negation detection was correct, we see that our approach lacks behind both neural systems. This is of course no surprise since our system does not assign any scope to negation or quantifier constructions.

In closing, to answer RQ3:

> *Such an approach can detect single negation clauses quite well. It misses a number of two clause cues, however, the ones it does detect are generally correct. This is true across languages. Assigning scope to negation is not done at all, a clear area of improvement.*

As it stands, our approach and system are quite complete in terms of framework and features. Naturally, some of these features can still be improved (and probably will). The goal of creating a fully working system has been met. The performance of our approach is promising and in certain aspects quite good. The system can already be used as a lightweight semantic parser to play around with or even to assist in DRS annotating. We have also shown that the SBN format is a promising new approach in DRS parsing, particularly when considering DRGs as a target format. In the process of using this new notation, we developed some useful tooling for it regarding parsing, validating, translating it to other formats, evaluating it in various ways and more.

6.2 DIRECTIONS FOR FUTURE WORK

Finally, let us consider some directions for future work. In practical terms regarding our system, the following is a non-exhaustive list of possible improvements:

- Add support for Enhanced UD - as discussed in Section 2.2, Enhanced UD could be beneficial for detecting certain constructions and in semantic parsing in general (Findlay and Haug, 2021);
- Add proper word-sense disambiguation - our system currently has two options for choosing a sense, either a single option from the mappings or `01` as a default, this is just a baseline of sorts compared to actual word-sense disambiguation;
- Do more research into creating an edge labeling classifier - the edge mappings mainly suffer from semantically ambiguous UD constructions at the moment, improving this could involve training a sophisticated edge classifier that takes the entire graph structure and more UD attributes into account for example;
- Validate constructed WordNet synsets - WordNet synsets created in our system are not checked if they actually exist in WordNet, we mainly left this out since most synsets are based on the gold training data, which is assumed to be valid;
- Improve handling of named entities, dates and numerical expressions - this can likely be done by incorporating a NER system since it is almost impossible to do this from purely UD information alone;
- Handle more box constructions - there are several complex box constructions, such as `ALTERNATION`, `CONSEQUENCE` or `POSSIBILITY`, that are not supported currently.

In terms of future work regarding DRS parsing, we have shown that UD is certainly a good fit in this context. Previous work has shown it helps as an additional feature in semantic parsing in general (Dozat and Manning, 2018; Xu et al., 2018) and DRS in particular (Fancellu et al., 2019; Yang et al., 2021). Our system is the first to create DRSs from UD directly, specifically to produce DRGs. There is of course a lot more to be explored in this area, something Gotham and Haug (2018) have worked on specifically. Our approach is also almost language-neutral with a relatively low number of rules. These rules are also quite simple. The previously discussed `UDepLambda` (Reddy et al., 2017) uses 123 rules in total for instance. This is considerably more than the 33 our approach uses in total. We are targeting a different meaning representation, but this difference is still notable.

An interesting possible extension or variation of our method could be a fully integrated neuro-symbolic approach. In that case, the graph transformations could be learned instead of entirely pre-defined. This is an area that was explored briefly but proved to be too much for the scope of the project. A decision was made to aim for a fully working and modular system, instead of a learnable graph transformation approach. The work from Fancellu et al. (2019) we have discussed, using graph grammars, is certainly a good starting point for this.

The explainability and transparency aspect of DRS parsing is certainly an interesting and important factor to consider in future work. As we have seen, Neural-Boxer, which is based on the state-of-the-art in DRS parsing (van Noord et al., 2020), can produce dubious output that is not easily detected with scoring metrics. These *hallucinations* are quite detrimental in the context of semantic parsing. The neuro-symbolic approach could help with this by producing a broader representation that captures *how* a model got to its decision for example. Another approach could be using two models side-by-side, a neural model that performs the actual inference and another that watches this model and creates an explanation for it. These are often referred to as *surrogate models* and this technique has been successfully applied in other areas (Angelov et al., 2021).

In general, it would also be interesting to look at the impact of this transparency aspect in a downstream task. This can be a particularly important consideration for semantic parsing in less academic settings for example.

BIBLIOGRAPHY

Abend, O. and A. Rappoport (2013, August). Universal Conceptual Cognitive Annotation (UCCA). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Sofia, Bulgaria, pp. 228–238. Association for Computational Linguistics.

Abend, O. and A. Rappoport (2017, July). The state of the art in semantic representation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada, pp. 77–89. Association for Computational Linguistics.

Abzianidze, L., J. Bjerva, K. Evang, H. Haagsma, R. van Noord, P. Ludmann, D.-D. Nguyen, and J. Bos (2017, April). The Parallel Meaning Bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Valencia, Spain, pp. 242–247. Association for Computational Linguistics.

Abzianidze, L., J. Bos, and S. Oepen (2020, November). DRS at MRP 2020: Dressing up discourse representation structures as graphs. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, Online, pp. 23–32. Association for Computational Linguistics.

Abzianidze, L., R. van Noord, C. Wang, and J. Bos (2020). The parallel meaning bank: A framework for semantically annotating multiple languages. *Applied mathematics and informatics 25*(2), 45–60.

Angelov, P. P., E. A. Soares, R. Jiang, N. I. Arnold, and P. M. Atkinson (2021). Explainable artificial intelligence: an analytical review. *WIREs Data Mining and Knowledge Discovery 11*(5), e1424.

Bai, X., Y. Chen, and Y. Zhang (2022, May). Graph pre-training for AMR parsing and generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland, pp. 6001–6015. Association for Computational Linguistics.

Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, Sofia, Bulgaria, pp. 178–186. Association for Computational Linguistics.

Basile, V., J. Bos, K. Evang, and N. Venhuizen (2012, 7-8 June). UGroningen: Negation detection with discourse representation structures. In **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)*, Montréal, Canada, pp. 301–309. Association for Computational Linguistics.

Bevilacqua, M., R. Blloshmi, and R. Navigli (2021, May). One spring to rule them both: Symmetric amr semantic parsing and generation without a complex pipeline. *Proceedings of the AAAI Conference on Artificial Intelligence 35*(14), 12564–12573.

Bonfante, G., B. Guillaume, M. Morey, and G. Perrier (2011). Modular graph rewriting to compute semantics. In *Proceedings of the Ninth International Conference on Computational Semantics (IWCS 2011)*.

Bos, J. (2008). Wide-coverage semantic analysis with Boxer. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, pp. 277–286. College Publications.

Bos, J. (2015, May). Open-domain semantic parsing with boxer. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, Vilnius, Lithuania, pp. 301–304. Linköping University Electronic Press, Sweden.

Bos, J. (2021). Variable-free discourse representation structures. *Semantics Archive*.

Bos, J., V. Basile, K. Evang, N. Venhuizen, and J. Bjerva (2017). The Groningen Meaning Bank. In N. Ide and J. Pustejovsky (Eds.), *Handbook of Linguistic Annotation*, Volume 2, pp. 463–496. Springer.

Cai, S. and K. Knight (2013, August). Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Sofia, Bulgaria, pp. 748–752. Association for Computational Linguistics.

de Marneffe, M.-C., C. D. Manning, J. Nivre, and D. Zeman (2021, 07). Universal Dependencies. *Computational Linguistics* 47(2), 255–308.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019, June). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, pp. 4171–4186. Association for Computational Linguistics.

Dozat, T. and C. D. Manning (2018, July). Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Melbourne, Australia, pp. 484–490. Association for Computational Linguistics.

Evang, K. (2019, May). Transition-based DRS parsing using stack-LSTMs. In *Proceedings of the IWCS Shared Task on Semantic Parsing*, Gothenburg, Sweden. Association for Computational Linguistics.

Fancellu, F., S. Gilroy, A. Lopez, and M. Lapata (2019, November). Semantic graph parsing with recurrent neural network DAG grammars. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, pp. 2769–2778. Association for Computational Linguistics.

Fancellu, F., S. Reddy, A. Lopez, and B. Webber (2017, April). Universal Dependencies to logical form with negation scope. In *Proceedings of the Workshop Computational Semantics Beyond Events and Roles*, Valencia, Spain, pp. 22–32. Association for Computational Linguistics.

Fellbaum, C. (1998). Wordnet: An electronic lexical database. Cambridge, MA. MIT Press.

Findlay, J. Y. and D. T. T. Haug (2021, December). How useful are enhanced Universal Dependencies for semantic interpretation? In *Proceedings of the Sixth International Conference on Dependency Linguistics (Depling, SyntaxFest 2021)*, Sofia, Bulgaria, pp. 22–34. Association for Computational Linguistics.

Gerdes, K., B. Guillaume, S. Kahane, and G. Perrier (2019, August). Improving surface-syntactic Universal Dependencies (SUD): MWEs and deep syntactic features. In *Proceedings of the 18th International Workshop on Treebanks and Linguistic Theories (TLT, SyntaxFest 2019)*, Paris, France, pp. 126–132. Association for Computational Linguistics.

Gotham, M. and D. Haug (2018). Glue semantics for universal dependencies. In M. Butt and T. King (Eds.), *Proceedings of the LFG'18 Conference, University of Vienna*, pp. 208–226. CSLI Publications.

Guillaume, B. (2021, April). Graph Matching and Graph Rewriting: GREW tools for corpus exploration, maintenance and conversion. In *EACL 2021 - 16th conference of the European Chapter of the Association for Computational Linguistics*, Kiev/Online, Ukraine.

Guillaume, B. and G. Perrier (2021, August). Graph rewriting for enhanced Universal Dependencies. In *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT 2021 Shared Task on Parsing into Enhanced Universal Dependencies (IWPT 2021)*, Online, pp. 175–183. Association for Computational Linguistics.

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman (Eds.), *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA, pp. 11 – 15.

Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kamp, H. (1981). A theory of truth and semantic representation, 277-322, jag groenendijk, tmv janssen and mbj stokhof, eds. In J. Groenendijk (Ed.), *Formal Methods in the Study of Language*. University of Amsterdam.

Kasper, R. T. (1989). A flexible interface for linking applications to Penman's sentence generator. In *Speech and Natural Language: Proceedings of a Workshop Held at Philadelphia, Pennsylvania, February 21-23, 1989*.

Kingsbury, P. and M. Palmer (2002, May). From TreeBank to PropBank. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain. European Language Resources Association (ELRA).

Kipper, K., A. Korhonen, N. Ryant, and M. Palmer (2008, Mar). A large-scale classification of english verbs. *Language Resources and Evaluation 42*(1), 21–40.

Kollar, T., D. Berry, L. Stuart, K. Owczarzak, T. Chung, L. Mathias, M. Kayser, B. Snow, and S. Matsoukas (2018, June). The Alexa meaning representation language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, New Orleans - Louisiana, pp. 177–184. Association for Computational Linguistics.

Li, Z., L. Qu, and G. Haffari (2020, December). Context dependent semantic parsing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain (Online), pp. 2509–2521. International Committee on Computational Linguistics.

Liu, J., S. B. Cohen, and M. Lapata (2019, May). Discourse representation structure parsing with recurrent neural networks and the transformer model. In *Proceedings of the IWCS Shared Task on Semantic Parsing*, Gothenburg, Sweden. Association for Computational Linguistics.

Miller, G. A. (1994). WordNet: A lexical database for English. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Morante, R. and E. Blanco (2012, 7-8 June). *SEM 2012 shared task: Resolving the scope and focus of negation. In **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)*, Montréal, Canada, pp. 265–274. Association for Computational Linguistics.

Nguyen, M. V., V. D. Lai, A. Pouran Ben Veyseh, and T. H. Nguyen (2021, April). Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Online, pp. 80–90. Association for Computational Linguistics.

Oepen, S., O. Abend, L. Abzianidze, J. Bos, J. Hajic, D. Hershcovich, B. Li, T. O'Gorman, N. Xue, and D. Zeman (2020, November). MRP 2020: The second shared task on cross-framework and cross-lingual meaning representation parsing. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing*, Online, pp. 1–22. Association for Computational Linguistics.

Press, O., N. A. Smith, and M. Lewis (2021, August). Shortformer: Better language modeling using shorter inputs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Online, pp. 5493–5505. Association for Computational Linguistics.

Qi, P., Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning (2020). Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Reddy, S., O. Täckström, M. Collins, T. Kwiatkowski, D. Das, M. Steedman, and M. Lapata (2016). Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics 4*, 127–140.

Reddy, S., O. Täckström, S. Petrov, M. Steedman, and M. Lapata (2017, September). Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark, pp. 89–101. Association for Computational Linguistics.

Ruppenhofer, J., M. Ellsworth, M. R. Petruck, C. R. Johnson, and J. Scheffczyk (2006). *FrameNet II: Extended Theory and Practice*. Berkeley, California: International Computer Science Institute.

Schuster, S. and C. D. Manning (2016, May). Enhanced English Universal Dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Portorož, Slovenia, pp. 2371–2378. European Language Resources Association (ELRA).

Schuster, S., É. Villemonte de La Clergerie, M. D. Candito, B. Sagot, C. D. Manning, and D. Seddah (2017, September). Paris and Stanford at EPE 2017: Downstream Evaluation of Graph-based Dependency Representations. In *EPE 2017 - The First Shared Task on Extrinsic Parser Evaluation*, Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation, Pisa, Italy, pp. 47–59.

Steedman, M. (2001). *The Syntactic Process*. Language, Speech, and Communication. MIT Press.

Straka, M. and J. Straková (2017, August). Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Vancouver, Canada, pp. 88–99. Association for Computational Linguistics.

Susskind, Z., B. Arden, L. K. John, P. Stockton, and E. B. John (2021). Neuro-symbolic AI: an emerging class of AI workloads and their characterization. *CoRR abs/2109.06133*.

van Noord, R., L. Abzianidze, H. Haagsma, and J. Bos (2018, May). Evaluating scoped meaning representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, pp. 1685–1693. European Language Resources Association (ELRA).

van Noord, R., L. Abzianidze, A. Toral, and J. Bos (2018). Exploring neural methods for parsing discourse representation structures. *Transactions of the Association for Computational Linguistics 6*, 619–633.

van Noord, R., A. Toral, and J. Bos (2020). Character-level representations improve DRS-based semantic parsing even in the age of BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4587–4603. Association for Computational Linguistics.

Xu, K., L. Wu, Z. Wang, M. Yu, L. Chen, and V. Sheinin (2018, October-November). Exploiting rich syntactic information for semantic parsing with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, pp. 918–924. Association for Computational Linguistics.

Yang, J., F. Fancellu, B. Webber, and D. Yang (2021, November). Frustratingly simple but surprisingly strong: Using language-independent features for zero-shot cross-lingual semantic parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic, pp. 5848–5856. Association for Computational Linguistics.

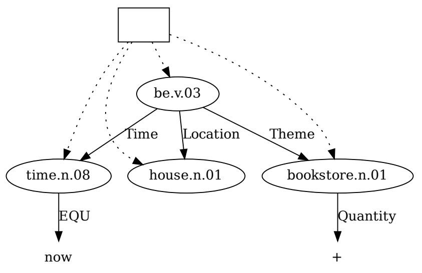
Zhou, J., T. Naseem, R. Fernandez Astudillo, Y.-S. Lee, R. Florian, and S. Roukos (2021, November). Structure-aware fine-tuning of sequence-to-sequence transformers for transition-based AMR parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic, pp. 6279–6290. Association for Computational Linguistics.

A APPENDICES

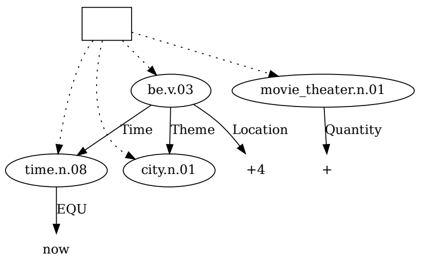
A.1 EXTENSIVE ERROR ANALYSIS

A.1.1 Strict versus lenient output

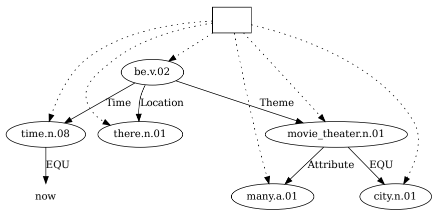
Let us consider some strict versus lenient output from Neural-Boxer and compare it to UD-Boxer and the gold graph. This gives us some insight into what these evaluation methods contributed to the final results. Figure 31 shows some of examples of this.



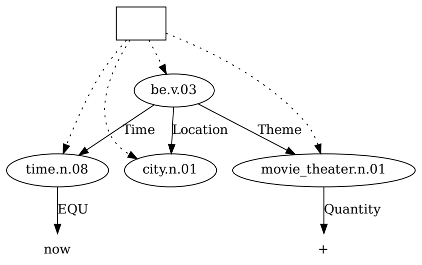
(a) Neural-Boxer (gold) produced DRG. *(F1 93.1)*



(b) Neural-Boxer (gold + silver) produced DRG. *(F1 91.5 lenient, F1 0 strict)*



(c) UD-Boxer produced DRG. *(F1 73.5)*



(d) Gold DRG.

Figure 31: UD-Boxer, Neural-Boxer and gold DRG for the sentence *There are many movie theaters in this city*. PMB id: `en/gold/p91/d2583`.

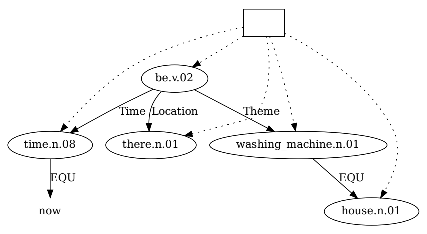
We can see that Neural-Boxer trained on just gold data is hallucinating a *bookstore* instead of a *movie theatre* as well as a *house* instead of a *city*. Neural-Boxer trained on gold and silver produces arguably ill-formed output by creating a wrong index. This results in a 'free floating' node, which is technically valid SBN[1], but not really useful. This also leads to the rather strange *Location* of '+4'.

UD-Boxer has some trouble with this sentence as well. It interprets *there* as a location once again. In addition, it cannot handle the *Quantity* construction, resulting in *many* being interpreted as an attribute instead. The *city* also ends up in the wrong position with a rather strange edge. This is the exact same problem as in Figure 32a.

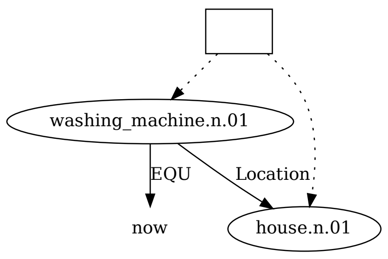
[1] With this we mean *parsable by our SBN parser*, i.e., it is a DAG in the proper format according to our specification. If it is valid according to the official SBN specification is unclear.

A.1.2 Poor Neural-Boxer output

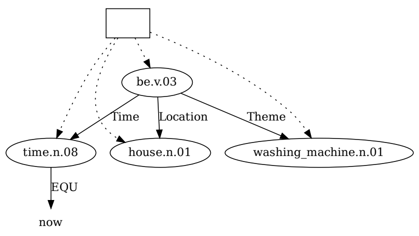
Figure 32 shows an example of a sentence Neural-Boxer struggles with.



(a) UD-Boxer produced DRG. *(F1 83.3)*



(b) Neural-Boxer produced DRG. *(F1 61.9)*



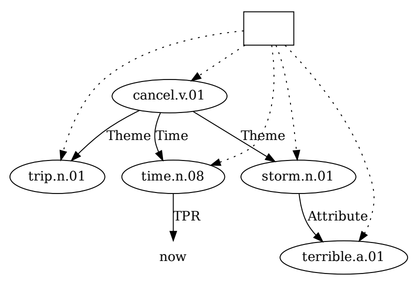
(c) Gold DRG.

Figure 32: UD-Boxer, Neural-Boxer (gold + silver) and gold DRG for the sentence *Is there a washing machine in the house?* PMB id: `en/gold/p80/d2803`.

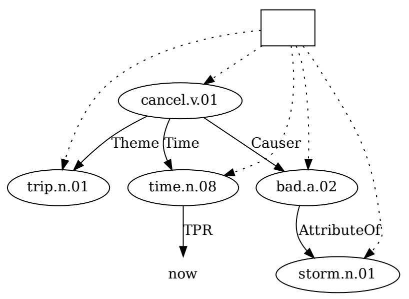
For some reason, Neural-Boxer adds the time relation to the *washing machine* synset. It also does not model the *is* from the original sentence in the required *be* node. The output from UD-Boxer is also not ideal since it interprets *there* as a location. We have seen this before in Figure 28. On top of that, it labels the edge between the house and the washing machine as *EQU*. When looking at the reason for this, we can see that this particular mapping stems from `NOUN-nmod-NOUN`, which is quite ambiguous.

A.1.3 Good output

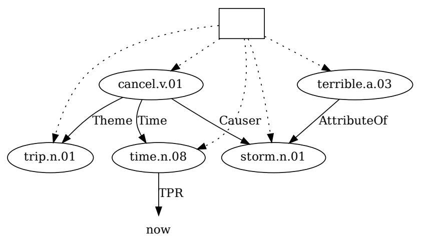
Figure 33 shows a somewhat long input sentence that both systems deal with well.



(a) UD-Boxer produced DRG. *(F1 93.9)*



(b) Neural-Boxer produced DRG. *(F1 90.9)*



(c) Gold DRG.

Figure 33: UD-Boxer, Neural-Boxer (gold + silver) and gold DRG for the sentence *The trip was canceled because of a terrible storm.* PMB id: `en/gold/p50/d1422`.

Neural-Boxer changes *terrible* to *bad*, which is not a big problem since the semantic meaning is still close. The *Causer* edge connecting to *bad* instead of the storm is also something to note. UD-Boxer has some trouble with its edge labeling and did not figure out the *Causer* relation. It also did not get the correct sense for *terrible*. Apart from these issues, both parsers perform similarly and very well.

A.2 FULL RESULTS

Table 13: Full results for all experiments, per data split, language, training data (Neural-Boxer only, gold, silver and bronze), evaluation method (Neural-Boxer only, lenient and strict) and UD parser (UD-Boxer only, Stanza and Trankit). Scores are on macro level. 'Err' refers to the percentage of ill-formed graphs a system produced for that data split. No output is also considered to be ill-formed. Strict scores refer to the strict evaluation mode regarding ill-formed graphs. Note that we do not have a neural system for English trained on gold + silver + bronze data. Also note that the 'Eval' data split only exists for English.

		Dev				Test				Eval			
		P	R	F1	Err	P	R	F1	Err	P	R	F1	Err
	English	83.6	81.6	82.1	0.3%	83.8	81.4	82.0	0.0%	83.1	80.5	81.3	0.5%
UD-Boxer	Dutch	78.0	74.5	75.5	0.0%	77.3	75.4	75.8	0.0%	-	-	-	-
(Stanza)	Italian	79.5	74.2	76.2	1.9%	81.2	76.6	78.4	0.9%	-	-	-	-
	German	80.8	77.2	78.4	0.0%	80.1	75.7	77.3	0.0%	-	-	-	-
	English	83.7	81.3	81.9	0.3%	83.6	81.1	81.8	0.0%	83.6	80.7	81.5	0.0%
UD-Boxer	Dutch	77.8	74.8	75.8	0.0%	77.1	75.6	75.8	0.0%	-	-	-	-
(Trankit)	Italian	80.8	76.0	77.8	0.0%	81.9	77.3	79.1	0.0%	-	-	-	-
	German	79.9	77.9	78.4	0.0%	79.2	76.8	77.5	0.0%	-	-	-	-
	English	84.0	82.1	82.8	4.6%	85.5	83.0	84.0	3.7%	83.4	80.8	81.7	3.9%
Neural-Boxer	Dutch	60.5	45.3	51.2	0.2%	60.2	45.3	51.1	0.4%	-	-	-	-
(g)	Italian	60.7	51.9	55.5	1.5%	61.2	51.6	55.7	1.5%	-	-	-	-
	German	67.8	61.7	64.2	0.4%	67.2	61.4	63.8	0.2%	-	-	-	-
	English	81.1	79.4	80.0	8.3%	83.4	81.1	82.0	6.4%	79.9	77.6	78.4	8.3%
Neural-Boxer	Dutch	33.9	26.4	29.3	43.5%	28.2	22.7	24.9	51.5%	-	-	-	-
(g) (strict)	Italian	47.2	41.1	43.6	23.1%	47.5	40.6	43.5	23.6%	-	-	-	-
	German	64.7	59.0	61.4	5.2%	64.3	58.9	61.1	4.4%	-	-	-	-
	English	92.8	92.4	92.5	2.0%	93.0	92.3	92.5	2.3%	92.4	91.5	91.8	2.8%
Neural-Boxer	Dutch	59.7	53.4	55.7	3.0%	59.2	54.4	56.2	2.6%	-	-	-	-
(g+s)	Italian	69.6	66.4	67.6	0.0%	70.8	67.8	69.1	0.0%	-	-	-	-
	German	71.0	66.7	68.4	0.7%	70.8	66.7	68.3	0.4%	-	-	-	-
	English	92.0	91.5	91.6	3.1%	92.4	91.7	91.9	3.0%	91.0	90.3	90.5	3.7%
Neural-Boxer	Dutch	53.1	47.8	49.7	14.0%	52.2	48.1	49.7	14.3%	-	-	-	-
(g+s) (strict)	Italian	67.9	64.9	66.1	2.6%	69.7	66.8	68.0	1.7%	-	-	-	-
	German	68.3	64.3	65.9	4.5%	67.5	63.9	65.3	5.3%	-	-	-	-
Neural-Boxer	Dutch	73.2	71.0	71.9	0.9%	73.0	70.8	71.6	1.0%	-	-	-	-
(g+s+b)	Italian	76.2	75.4	75.6	0.0%	76.1	74.9	75.4	0.0%	-	-	-	-
	German	75.5	74.1	74.6	0.4%	75.4	74.5	74.7	0.5%	-	-	-	-
Neural-Boxer	Dutch	71.4	69.3	70.1	3.7%	71.2	69.3	70.0	3.5%	-	-	-	-
(g+s+b)	Italian	75.9	75.1	75.3	0.6%	75.3	74.2	74.6	1.1%	-	-	-	-
(strict)	German	75.3	73.9	74.3	0.7%	74.8	74.0	74.2	1.5%	-	-	-	-

These results were obtained after this thesis was handed in. Our assumptions in the conclusion regarding the performance of neural models for the other languages were correct. Due to the lack of data, the neural system struggles a fair bit, especially with only gold data. Recall that UD-Boxer only uses gold training data. A surprising pattern we see in Table 13, is that UD-Boxer outperforms *all* neural systems for Dutch, Italian and German. This happens even when there is quite a lot of training data available for Neural-Boxer, such as with the models trained on gold, silver and bronze data. Bronze data might have an effect on this somewhat lacking performance since the quality of this data is (likely) lower than silver data.

A.3 TRANSFORMATION RULES

A.3.1 Language-neutral transformation rules

These are the literal contents of the main GREW file.

```
%%% Ideas %%%
% Maybe combine NOUN -[amod]-> ADJ into single token, example: good- bye
% Maybe combine A -[xcomp]-> ADJ1 and A -[xcomp]-> ADJ2 (or more) into single token, example: bright
      blue
% nummod deprel kan vervangen worden door synset -[Quantity]-> number?
% SCONJ as EXPLANATION box? See en/gold/p00/d0801

%%% NOTES %%%
% Make sure to read https://grew.fr/doc/pattern/ and especially the part on edge clauses.
% For both the nodes and edges the 'token' feature is used to store already known DRS components
% or defaults that might help to resolve those later.

import "$$LANGUAGE$$.grs"

%%% LABELING RULES %%%
% Add the token feature to all nodes, this will be used to build up the string required for SBN.
% This is done in order to not lose the UD information that rules might use.
% Ideally, the token feature is only used in a write (or concat) only manner in the rules.
rule add_token_nodes {
   pattern {
      N [lemma, !token];
   }
   commands {
      N.token = N.lemma;
   }
}

% Add the token feature to all edges. This makes sure we're not losing the deprel information.
rule add_token_edges {
   pattern {
      E: N -[!token]-> M;
   }
   commands {
      % The initial feature is '1' by default, which is the 'deprel'.
      E.token = "NONE";
   }
}

% This is not really needed since the mappings can also handle this. This is more of an example of a
% direct labeling rule.
% Example: en/gold/p05/d1993
rule label_adj {
   pattern {
      N [upos=NOUN];
      E: N -[1=amod]-> M;
   }
   without {
      E[token="Attribute"];
   }
   commands {
      E.token = "Attribute";
   }
}

% Numbers are constants in sbn, this is not the cleanest option, but without a POS tag we ensure that
       the
% number won't be converted to a synset. We cannot allow non-leaf nodes to be constants, this results
       in
% invalid SBN.
% Example: en/gold/p01/d2141/
rule indicate_number {
   pattern {
      N [upos=NUM];
   }
   without {
      N -> *;
   }
   commands {
      del_feat N.upos;
   }
}

%%% CONNECTING RULES %%%
% Connect the Owner / User of something directly
% Example: en/gold/p04/d1646
rule connect_user {
   pattern {
      USER [upos=PROPN|NOUN];
      * -[1=nsubj]-> USER;
      REL: TARGET -[1=nmod, 2=poss]-> INDICATOR;
   }
   without {
      TARGET -[token="User"]-> USER;
   }
   commands {
      add_edge TARGET -[token="User"]-> USER;
      del_edge REL;
      del_node INDICATOR;
   }
}

%%% EXPANDING RULES %%%
% Expand a name into an entity synset node, a name edge and the name constant.
% Example: en/gold/p04/d1646
rule expand_name {
   pattern {
      NAME [upos=PROPN];
   }
   without {
      NAME -> *;
   }
   commands {
      % NAME.token = "entity.n.01"; see note above

      add_node NAME_CONST;
      NAME_CONST.token = NAME.textform;

      add_edge NAME -[token="Name"]-> NAME_CONST;
   }
}

% Add a speaker nodes split up into 'person-synset' -> 'speaker'.
% Example: en/gold/p04/d1646
rule expand_first_person {
   pattern {
      SPEAKER [upos=PRON, Person=1];
   }
   without {
      SPEAKER [token="GENDER"];
   }
   commands {
      SPEAKER.token = "GENDER";

      add_node SPEAKER_CONST;
      SPEAKER_CONST.token = "speaker";

      add_edge SPEAKER -[token="EQU"]-> SPEAKER_CONST;
   }
}

% Add a speaker nodes split up into 'person-synset' -> 'hearer'.
% Example: en/gold/p05/d2340
rule expand_second_person {
   pattern {
      HEARER [upos=PRON, Person=2];
   }
   without {
      HEARER [token="GENDER"];
   }
   commands {
      HEARER.token = "GENDER";

      add_node HEARER_CONST;
      HEARER_CONST.token = "hearer";

      add_edge HEARER -[token="EQU"]-> HEARER_CONST;
   }
}

% Indicate pronoun to resolve.
% Example: en/gold/p05/d2340
rule expand_third_person {
   pattern {
      PERSON [upos=PRON, Person=3];
   }
   without {
      PERSON [token="GENDER"];
   }
   commands {
      PERSON.token = "GENDER";
   }
}

% Add a time synset node.
% Example: en/gold/p04/d1646
rule add_time {
   pattern {
      N [];
      * -[1=root]-> N;
   }
   without {
      N -[token="Time"]-> *;
   }
   commands {
      add_node TIME_SYNSET;
      TIME_SYNSET.token = "time.n.08";

      add_edge N -[token="Time"]-> TIME_SYNSET;

      add_node TIME_CONST;
      TIME_CONST.token = "now";

      add_edge TIME_SYNSET -[token="TIMERELATION"]-> TIME_CONST;
   }
}

%%% COMBING RULES %%%
% Combine multiple PROPNs that probably belong together. This is quite tricky since this is very
% vunrable to UD parse error or oddities (see example).
% Example: en/gold/p65/d1215
rule combine_propn {
   pattern {
      A [upos=PROPN];
      B [upos=PROPN];
      R: A -[1=flat|compound]-> B;
   }
   commands {
      A.token = A.token + "_" + B.token;
      del_edge R;
      del_node B;
   }
}

% Combine phrasal verb particle components. These are often treated as a single synset.
% Some common cases include: "cut off", "burn down", "hang up" etc.
% Example: en/gold/p00/d1469
rule combine_compound_prt {
   pattern {
      A [];
      B [];
      R: A -[1=compound, 2=prt]-> B;
   }
   commands {
      A.token = A.token + "_" + B.token;
      % We know it's most likely a verb at this point.
      A.upos = "VERB";
      del_edge R;
      del_node B;
   }
}

% Combine multiple NOUNS that are probably compounds.
% Example: en/gold/p50/d2408
rule combine_nouns {
   pattern {
      A [upos=NOUN];
      B [upos=NOUN];
      % B < A;
      R: A -[1=compound]-> B;
   }
   commands {
      A.token = B.token + "_" + A.token;
      del_edge R;
      del_node B;
   }
}

%%% CLEANING RULES %%%
% Remove any punctuation that is connected to the root directly. These are the sentence ending
      punctuation marks.
% Example: en/gold/p04/d1646
rule remove_root_punct {
   pattern {
      * -[1=root]-> ROOT;
      E: ROOT -> N;
      N [upos=PUNCT];
   }
   commands {
      del_edge E;
      del_node N;
   }
}

% Remove nodes that are possibly useless. Ideally this rule is applied *after* all other rules that
      might
% use the nodes (such as combining node tokens etc.) in order to not lose information.
% NOTE: Not sure about PART here since that can also indicate negation or possession, which is
      semantically useful
% Same goes for CCONJ and SCONJ, for now they are removed, since they often also don’t contribute
      anything
% and are basically never used as a node on their own.
% Example: en/gold/p04/d1646
rule remove_unwanted_pos {
   pattern {
      N [upos=PUNCT|DET|AUX|ADP|PART|CCONJ|SCONJ];
   }
   commands {
      del_node N;
   }
}

% Remove the explicit ROOT node.
% Example: en/gold/p04/d1646
rule remove_explicit_root {
   pattern {
      N [];
      E: N -[1=root]-> T;
   }
   commands {
      del_edge E;
      del_node N;
   }
}


%%% SPECIAL CASES %%%
% This can happen if a rule connects nodes together and later removes some.
% If for instance the Owner role gets added, this might introduce a cycle.
% This is a trade-off, there is information loss, but at least the output is
% expected be a valid DAG. Possibly deal with cases more specifically,
% though it is quite rare that this happens.
% NOTE: this does not scale, when there are hops between the connecting nodes,
% this does not fix it. Probably need to deal with this on the networkx side.
% Example: en/gold/p96/d1385 (caused by connect_user)
rule detach_cycles {
   pattern {
      A: N -> M;
      B: M -> N;
   }
   commands {
      del_edge B
   }
}

% Main strat to apply all rules. Ordering is very important here.
strat main {
   Pick(
      Iter (
         Seq (
            Onf(add_token_nodes),
            Onf(add_token_edges),

            Iter(combine_propn),
            Iter(combine_nouns),
            Iter(combine_compound_prt),

            Iter(label_adj),

            Iter(connect_user),
            Iter(expand_name),

            Onf(expand_first_person),
            Onf(expand_second_person),
            Onf(expand_third_person),

            Iter(add_time),

            Iter($$LANGUAGE$$),

            Iter(remove_root_punct),
            Iter(remove_explicit_root),
            Onf(remove_unwanted_pos),

            Iter(indicate_number),

            Iter(detach_cycles),
         )
      )
   )
}
```

A.3.2 Language-specific transformation rules

English

```
%%% English specific rules %%%
%%% BOX RULES %%%
% Example: en/gold/p02/d1681
rule box_negation_det {
   pattern {
      N [lemma=no|not|never];
      * -[1=advmod|det]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: en/gold/p03/d0823
rule box_negation_nmod {
   pattern {
      N [lemma=none|nothing];
      N -[1=nmod|obj]-> *;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: en/gold/p04/d0830
rule box_negation_pron {
   pattern {
      N [lemma=nobody];
      * -[1=nsubj]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: en/gold/p04/d2804
rule box_quantifier {
   pattern {
      N [lemma=every|everyone|everybody|everything|always|all|whoever|whomever|both|whatever];
   }
   without {
      P [token="NEGATION"];
      Q [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";

      add_node NEGATION_BOX_2;
      NEGATION_BOX_2.token = "NEGATION";

      add_edge NEGATION_BOX -[token="NEGATION"]-> NEGATION_BOX_2;
   }
}
```

Dutch

```
%%% Dutch specific rules %%%

%%% BOX RULES %%%
% Example: nl/gold/p06/d0785
rule box_negation_det {
   pattern {
      N [lemma=niet|geen|nooit];
      * -[1=advmod|det]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: nl/gold/p57/d2423
rule box_negation_nmod {
   pattern {
      N [lemma=niets|niks|nada];
      N -[1=nmod|obj]-> *;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: nl/gold/p96/d0945
rule box_negation_pron {
   pattern {
      N [lemma=niemand];
      * -[1=nsubj]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: nl/gold/p36/d2853
rule box_quantifier {
   pattern {
      N [lemma=iedereen|elk|elke|alle|alles|altijd|iedere];
   }
   without {
      P [token="NEGATION"];
      Q [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";

      add_node NEGATION_BOX_2;
      NEGATION_BOX_2.token = "NEGATION";

      add_edge NEGATION_BOX -[token="NEGATION"]-> NEGATION_BOX_2;
   }
}
```

Italian

```
%%% Italian specific rules %%%

%%% BOX RULES %%%
% Example: it/gold/p09/d1743
rule box_negation_det {
   pattern {
      N [lemma=non|mai];
      * -[1=advmod|det]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: it/gold/p19/d2943
rule box_negation_nmod {
   pattern {
      N [lemma=nulla|niente|zero];
      N -[1=nmod|obj]-> *;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: it/gold/p65/d1668
rule box_negation_pron {
   pattern {
      N [lemma=nessuno|niente];
      * -[1=nsubj]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: it/gold/p37/d2571
rule box_quantifier {
   pattern {
      N [lemma=tutto|tutti|entrambe|entrambi|ogni|ciascuno|qualsiasi];
   }
   without {
      P [token="NEGATION"];
      Q [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";

      add_node NEGATION_BOX_2;
      NEGATION_BOX_2.token = "NEGATION";

      add_edge NEGATION_BOX -[token="NEGATION"]-> NEGATION_BOX_2;
   }
}
```

German

```
%%% German specific rules %%%

%%% BOX RULES %%%
% Example: de/gold/p03/d2800
rule box_negation_det {
   pattern {
      N [lemma=nicht|kein|keine|keines|nie];
      * -[1=advmod|det]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: de/gold/p05/d2383
rule box_negation_nmod {
   pattern {
      N [lemma=nichts|niemanden|keine];
      N -[1=nmod|obj]-> *;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: de/gold/p06/d3500
rule box_negation_pron {
   pattern {
      N [lemma=niemand|niemanden|keiner];
      * -[1=nsubj]-> N;
   }
   without {
      P [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";
   }
}

% Example: de/gold/p06/d1718
rule box_quantifier {
   pattern {
      N [lemma=jeder|jedes|jederman|jegliche|alle|alles|stets|beide|immer];
   }
   without {
      P [token="NEGATION"];
      Q [token="NEGATION"];
   }
   commands {
      del_node N;

      add_node NEGATION_BOX;
      NEGATION_BOX.token = "NEGATION";

      add_node NEGATION_BOX_2;
      NEGATION_BOX_2.token = "NEGATION";

      add_edge NEGATION_BOX -[token="NEGATION"]-> NEGATION_BOX_2;
   }
}
```