



SAARLAND UNIVERSITY  
DEPARTMENT OF COMPUTATIONAL LINGUISTICS

MASTER THESIS

---

# Transformations for tractable SFG parsing

---

*submitted in fulfillment of the degree requirements of the*

**MSc in Language Science and Technology at Saarland University**

*Author:*

Guadalupe ROMERO

Matriculation: 2576689

*Supervisors:*

Prof. Dr. Alexander KOLLER

Dr. Jonas GROSCHWITZ

Date of Submission: November 25, 2020

### **Eidesstattliche Erklärung**

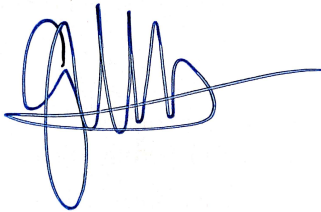
Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ich versichere, dass die gedruckte und die elektronische Version der Masterarbeit inhaltlich übereinstimmen.

### **Declaration**

I hereby confirm that the thesis presented here is my own work, with all assistance acknowledged.

I assure that the electronic version is identical in content to the printed version of the Master's thesis.

A handwritten signature in blue ink, consisting of a series of loops and a long horizontal stroke extending to the right.

Guadalupe Romero

Saarbrücken, November 25, 2020

## **Abstract**

The need to build syntactic parsers with improved usability have fueled a renewed interest in Systemic Functional Grammar (SFG). Among its key features, SFG presents a flatter constituency structure, makes ellipsis explicit and appends functional tags to its constituents. We concentrate our efforts on the first two aspects as they are the most challenging to predict, and present the task of SFG constituency parsing and ellipsis resolution, including a dataset and an evaluation method. We represent ellipsis with secondary edges that form directed acyclic graphs (DAGs) and propose a method for predicting these graphs via tractable DAG to tree transformations. The suggested strategies remove the secondary edges and encode their information in the constituency labels, so that standard tree parsers can be used and the original DAG representations can be later recovered. Experiments with different transformations show that this is a viable approach and that some representations are more learnable than others. Our system outperforms the previous work on SFG parsing and handles a wider arrange of ellipsis types, a feature with potential applications inside and outside the SFG domain.

# Acknowledgements

First of all, I would like to thank my supervisors Jonas Groschwitz and Alexander Koller for your guidance and valuable feedback.

A special thanks to Matthew Honnibal for sharing the idea behind this thesis, and for your help and encouragement during the whole process. I want to thank Ines Montani as well. You both believed in me from the very beginning and always made me feel appreciated and listened to. You have taught me tons about NLP but, more importantly, that it is possible to build a fun and caring working environment. I feel very proud and happy to be part of your team.

I want to thank my family for their unconditional support, especially during the very crazy times I had to live this year. Thank you to César and Ale for making me part of your home and to Martín for being the best quarantine buddy ever! Thank you to my big brother Santi because I know I can always count on you, and to my little brother Emi for making me laugh like nobody else.

Finally, I would like to thank the amazing friends I made during these two years. To my compatriot Lea, who was there for me since I first set a foot in Germany. To my very first friends here, Tayfun and Eda, teşekkürler. To Azin and Mario for your essential first year CoLi survival guides. Danke Jakob for the Philo Cafe lunches and the fifty-cent-machine-coffee breaks in between our library sessions. To my dear Franzi for receiving me always with a hug and a smile. Thank you, Katja, for getting through all the grey Saarländisch winter days with me, and for what I know will be a life-long beautiful friendship. And last but not least, thanks to Kathryn, Vanessa, Leonie, Miri, Anja and Katja, for the million of funny stories I can tell now. Those early morning lectures were so much better with you in my mate round.

The work in this thesis has been supported by an ALEARG scholarship, granted by the Deutscher Akademischer Austauschdienst (DAAD) and the Argentine Ministry of Education.

# Contents

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                     | <b>1</b>  |
| <b>2</b> | <b>Systemic Functional Grammar</b>                      | <b>3</b>  |
| 2.1      | A brief account on SFG . . . . .                        | 3         |
| 2.2      | Why SFG constituency is hard to parse . . . . .         | 7         |
| <b>3</b> | <b>SFG constituency parsing and ellipsis resolution</b> | <b>11</b> |
| 3.1      | Task . . . . .                                          | 11        |
| 3.2      | Related work . . . . .                                  | 11        |
| 3.3      | Dataset . . . . .                                       | 14        |
| 3.4      | Evaluation . . . . .                                    | 15        |
| <b>4</b> | <b>Transformations for tractable SFG parsing</b>        | <b>18</b> |
| 4.1      | Related work . . . . .                                  | 18        |
| 4.2      | Suggested transformations . . . . .                     | 20        |
| 4.3      | Methodology . . . . .                                   | 30        |
| 4.4      | Results and discussion . . . . .                        | 32        |
| <b>5</b> | <b>Comparison to previous SFG parsing work</b>          | <b>42</b> |

|          |                            |           |
|----------|----------------------------|-----------|
| 5.1      | State of the art . . . . . | 42        |
| 5.2      | Evaluation . . . . .       | 42        |
| 5.3      | Discussion . . . . .       | 43        |
| <b>6</b> | <b>Conclusions</b>         | <b>45</b> |

# List of Figures

|     |                                                                                                                                       |    |
|-----|---------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Part of the MOOD system (interpersonal metafunction), as defined in Honnibal (2004) . . . . .                                         | 6  |
| 2.2 | Choices in the MOOD system (interpersonal metafunction) for the clause <i>When the weather was bad</i> . . . . .                      | 6  |
| 2.3 | Choices in the MOOD system (interpersonal metafunction) for the clause <i>she leafed through incomprehensible magazines</i> . . . . . | 7  |
| 2.4 | Part of the TRANSITIVITY system (experiential metafunction), as defined in Eggins (2004) . . . . .                                    | 8  |
| 2.5 | Example of a syntactic tree following the Penn Treebank style . . . . .                                                               | 9  |
| 2.6 | SFG constituency as a tree, where elided elements are duplicated . . . . .                                                            | 9  |
| 2.7 | SFG constituency as a DAG, where ellipsis is represented with reentrant nodes . . . . .                                               | 10 |
| 3.1 | An example DAG for a sentence with noun phrase ellipsis (node 4) and gapping (node 5) . . . . .                                       | 16 |
| 3.2 | Example of a dependency graph, where ellipsis edges are represented with dashed lines . . . . .                                       | 17 |
| 4.1 | Example of a graph to tree transformation using the <b>start</b> strategy . . . . .                                                   | 23 |
| 4.2 | Example of a graph to tree transformation using the <b>start-without-pos</b> strategy . . . . .                                       | 24 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |    |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.3  | Example of a graph to tree transformation using the <b>start-without-pos</b> strategy where the exact location of the ellipsis is ambiguous . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 24 |
| 4.4  | Example of a graph to tree transformation using the <b>end</b> strategy . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 25 |
| 4.5  | Example of a graph to tree transformation using the <b>end-extra-node</b> strategy . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 25 |
| 4.6  | Example of a graph to tree transformation using the <b>start-end-extra-node</b> strategy . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 26 |
| 4.7  | Example of a graph to tree transformation using the <b>start-end-extra-node-heuristic</b> strategy . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 27 |
| 4.8  | Example of a graph to tree transformation using the <b>start-end-extra-node-heuristic</b> strategy, after enumerating the ellipsis labels at post-processing . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 28 |
| 4.9  | The same example from Figures 4.7 and 4.8, but using the Penn Treebank representation, which does not expand coordinated structures in different clauses . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 29 |
| 4.10 | The Berkeley parser combines a chart decoder with a sentence encoder based on self-attention. Illustration taken from Kitaev and Klein (2018) . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 31 |
| 4.11 | Example of a graph converted to the SemEval format. The columns correspond to ID, form, lemma, PoS, top, predicate, frame, and an argument column for as many predicates as there are in the sentence. We use a dummy label <b>_</b> for the lemma, PoS and frame columns, a <b>+</b> for the top column if the token is the root of the sentence, and a <b>+</b> for the predicate column if the token has any outgoing edges. The argument columns $(arg_1, arg_2, \dots, arg_i)$ contain the edge labels relative to the $i$ -th predicate. Additionally, we use an <b>ellipsis</b> marker to differentiate elided edges. For more details on the format, refer to the official SemEval documentation. . . . . | 32 |
| 4.12 | Example of a wrong prediction with the <b>start-end-extra-node</b> strategy. The elided node <b>CC1</b> exists, but generates an ungrammatical/meaningless sentence. The correct tag should be <b>NGendCC2</b> , instead of <b>NGendCC1</b> . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 35 |
| 4.13 | Example of a gold parse for the <b>start-end-extra-node</b> strategy. The label <b>ADVGenCL3</b> indicates that the adverb <i>thus</i> is elided in the last clause, which is actually incorrect. The predicted parse misses this label, generating a more appropriate analysis . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                         | 36 |



|      |                                                                                                                                                                                                                                                                                                                                                                                                           |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.14 | Example of a prediction with the <b>start-end-extra-node</b> strategy, where the extra node carrying the ellipsis label wraps the wrong elided node. The gold standard wraps the <b>NG</b> node from the first clause instead of the second one. However, both options are correct in this case. The last clause can either recover the subject <i>they</i> from the first or the second clause . . . . . | 37 |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|

# List of Tables

|     |                                                                                                                                                 |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Different conflations of Actor, Subject and Theme (1) . . . . .                                                                                 | 5  |
| 2.2 | Different conflations of Actor, Subject and Theme (2) . . . . .                                                                                 | 5  |
| 2.3 | Different conflations of Actor, Subject and Theme (3) . . . . .                                                                                 | 5  |
| 2.4 | Different conflations of Actor, Subject and Theme (4) . . . . .                                                                                 | 6  |
| 2.5 | Different conflations of Actor, Subject and Theme (5) . . . . .                                                                                 | 6  |
| 2.6 | Example of constituents tagged with TRANSITIVITY functions, where the<br>Process is material . . . . .                                          | 6  |
| 2.7 | Example of constituents tagged with TRANSITIVITY functions, where the<br>Process is mental . . . . .                                            | 7  |
| 2.8 | Description of SFG non-terminal tags . . . . .                                                                                                  | 10 |
| 3.1 | Ranks for constituent tags, which are used to find the head of each constituent<br>when converting the parse tree to dependencies . . . . .     | 17 |
| 4.1 | Scores for the ellipsis and coordination expansion task (dependency evaluation)                                                                 | 39 |
| 4.2 | Results from reversing the graph to tree conversion directly, without any train-<br>ing or parsing in between (dependency evaluation) . . . . . | 40 |
| 4.3 | Scores for the ellipsis and coordination expansion subtask (constituency eval-<br>uation) . . . . .                                             | 41 |

|     |                                                                                                                                                         |    |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 5.1 | Comparison with Costetchi (2020)’s parser, using their evaluation method.<br>Only the constituency structure (without ellipsis) is considered . . . . . | 43 |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------|----|

# Chapter 1

## Introduction

Progress in neural syntactic parsing has made it possible to analyse large volumes of data efficiently and accurately. However, these analyses lack semantic and functional information that can potentially be extracted from text and be beneficial for downstream applications. Semantic representations are, on the other hand, harder to predict and difficult to interpret for developers without a linguistic background. This motivates an intermediate approach that extends syntactic representations with functional and semantic information. Several attempts have been made to enrich dependency parsers by extracting semantic relations that are present in syntax but often left implicit, such as the collapsed and propagated dependencies introduced by [De Marneffe and Manning \(2008\)](#), the enhanced dependencies within the Universal Dependencies project ([Nivre et al., 2018](#)) ([Schuster and Manning, 2016](#)) and the pyBART library of [Tiktinsky et al. \(2020\)](#), as well as the work on diathesis alternation by [Candito et al. \(2017\)](#). This is also the motivation behind the renewed interest in Systemic Function Grammar (SFG) for Natural Language Processing (NLP) ([Honnibal, 2004](#)) ([Costetchi, 2020](#)). SFG considers not only syntax but also semantics and pragmatics, and is geared towards usability rather than generative considerations, making it particularly suitable to building a user-friendly parser. Its most salient features are a flatter constituency structure, the recovery of elided elements, and the labeling of constituents with functional tags. Previous work on SFG has shown that the functional tags can be efficiently and accurately inferred from the syntax using deterministic rules. For this reason, we concentrate our efforts on the more challenging task of predicting SFG constituency structure and, particularly, on recovering elided constituents.

Typically, work on SFG makes ellipsis explicit by copying the elided elements in the original string and performing syntactic analysis on top of it. We propose an alternative representa-

tion of ellipsis that is more linguistically motivated and can be used to jointly predict SFG constituency and ellipsis resolution, with potential applications also outside the SFG domain. Instead of transforming the original string, we use secondary edges that originate where the ellipsis occurs and point to the elided constituents, forming a direct acyclic graph (DAG). We perform this conversion on the SFGbank by [Honnibal \(2004\)](#) to develop a corpus for the prediction of SFG constituency and ellipsis resolution. The task, dataset and evaluation method is explained in more detail in Chapter 3.

Representing SFG constituency as a DAG prevents the use of standard tree parsers to predict such structure. The most direct approach would be to use graph parsers instead. However, these are harder to implement and have not yet achieved the accuracy and robustness of tree parsers. For this reason, we propose a method that transforms DAGs to tree structures which are isomorphic to the original SFG representations and can be reversed to their original form, allowing the use of standard tree parsers. We experiment with different transformation strategies to find a learnable representation with minimum information loss. Additionally, we use the graph dependency parser of [Dozat and Manning \(2018\)](#) to have an external comparison to our approach and identify its strengths and limitations. We present a thorough analysis of this comparison and all our suggested transformations in Chapter 4.

Our model also represents an improvement from earlier attempts at building an SFG parser. [Costetchi \(2020\)](#) implements a system with Stanford’s Dependency parser 3.5 ([Chen and Manning, 2014](#)) as a base. The output is converted to SFG constituency structure and enriched with functional tags. Part of the conversion rules in this model are designed to correct errors made by the Stanford Dependency parser. Substituting a different parser into the system would introduce different error profiles, which in turn would require updates to these correction rules. We address this issue by moving the transformations prior to training, which facilitates the substitution of different parsers into the system, requires less rule-based work to be done at run-time and improves accuracy. The parser by [Costetchi \(2020\)](#) recovers elided subjects but, unlike our system, is unable to handle any other types of ellipsis, such as gapping and noun-phrase ellipsis. We compare both approaches in Chapter 5.

The code we used to train and evaluate our models is publicly available at <https://github.com/guadiromero/sfg-parser>.

# Chapter 2

## Systemic Functional Grammar

### 2.1 A brief account on SFG

Systemic Functional Grammar (SFG) was developed by Michael Halliday in the 1960s, following Firth's notion of system. It approaches language from a social semiotic perspective and as such, it aims to describe the set of meaningful choices a speaker makes when putting thought into words. This focus on the paradigmatic axis makes SFG fundamentally different from other approaches which consider structure and the syntagmatic axis as the most important aspects of linguistic description.

There are two parts in SFG analysis. In the first place, it describes syntactic structure, which is constituency-based. This originates from Halliday's concept of rank: morphemes compose words, which compose groups and phrases, which compose clauses. The second part of SFG analysis focuses on function structure. Functions can refer to any rank, but the clause level is generally considered the most important.

Halliday recognized three basic functions of language, named metafunctions: ideational, interpersonal and textual (Halliday and Matthiessen, 2004). Language names things and processes, and while doing so it also conceptualizes them and classifies them in categories. The linguistic resources that construe this theory of human experience belong to the ideational metafunction, further divided into two components: the experiential and the logical. At the same time, language enacts our personal and social relationships. We inform and ask questions, make demands and offers, agree and disagree. This metafunction of language is called interpersonal. A third component of language organizes discourse in a cohesive se-

quence, the textual metafunction. The three metafunctions are enacted simultaneously and do not constrict each other. This multidimensional architecture of language is one of the key concepts behind SFG.

Halliday and Matthiessen (2004) take the notion of Subject as parting point to illustrate multifunctionality. There is no agreed definition for it, with various interpretations that could be summarized in three broad functions:

1. The doer of the action.
2. That of which something is being predicated and that concords in person and number with the main verb.
3. The concern of the message, what the speaker had in their mind when starting the production of the sentence.

Consider the clause:

*Curious dishes made her ill sometimes*<sup>1</sup> (see Table 2.1)

We can identify *curious dishes* as the Subject in its three senses. It represents the doer of the action, it is in concordance with the main verb and it serves as the point of departure of the message. However, not all clauses have an element that encloses these three functions, and this is why the concept of Subject remained so debated throughout the history of grammar. For example:

*Sometimes she was made ill by curious dishes* (see Table 2.2)

In this sentence, the three functions are no longer conflated in the same element: *curious dishes* is the doer of the action, *she* is that of which something is being predicated, and *Sometimes* is the parting point of the sentence. Therefore, it seems more suitable to consider them not as different aspects of the same notion, but as three separate functions, each with its own label and corresponding to a certain metafunction:

1. Actor (experiential metafunction, which is part of the ideational metafunction)
2. Subject (interpersonal metafunction)
3. Theme (textual metafunction)

---

<sup>1</sup>Most of the example text in this chapter is taken from Gorey (1980)

|                       |                               |
|-----------------------|-------------------------------|
| <i>Curious dishes</i> | <i>made her ill sometimes</i> |
| Actor                 |                               |
| Subject               |                               |
| Theme                 |                               |

Table 2.1: Different conflations of Actor, Subject and Theme (1)

|                  |            |                        |                       |
|------------------|------------|------------------------|-----------------------|
| <i>Sometimes</i> | <i>she</i> | <i>was made ill by</i> | <i>curious dishes</i> |
|                  |            |                        | Actor                 |
|                  | Subject    |                        |                       |
| Theme            |            |                        |                       |

Table 2.2: Different conflations of Actor, Subject and Theme (2)

In tables 2.1-2.5, we tag each function in the previous two sentences, as well as in other example sentences presenting different conflations of Actor, Subject and Theme.

SFG organizes the different ideational, interpersonal and textual functions in system networks, representing the choices that a speaker has to make along each dimension when producing language. As an example, Figure 2.1 represents a part of the MOOD system, which belongs to the interpersonal metafunction. In Figures 2.2 and 2.3, we show the selected options from the MOOD system that correspond to the clauses in the sentence *When the weather was bad, she leafed through incomprehensible magazines*.

Functions such as major/minor, bound/free are realized in the clause as a whole, whereas some other functions are realized in specific clause constituents. For example, in the TRANSITIVITY system in Figure 2.4, we can see that if the process type is material, it is realized in at least two constituents: Process:material and Actor, and optionally also in the constituents Goal, Range and Beneficiary. If the type of process is mental, it is realized in the constituents Process:mental, Sensor and Phenomenon. The sentences *There she climbed endless flights of stairs* and *She tried to make out the subjects of vast dark paintings* are tagged in Tables 2.6 and 2.7 as illustration.

|                  |                       |                     |
|------------------|-----------------------|---------------------|
| <i>Sometimes</i> | <i>curious dishes</i> | <i>made her ill</i> |
|                  | Actor                 |                     |
|                  | Subject               |                     |
| Theme            |                       |                     |

Table 2.3: Different conflations of Actor, Subject and Theme (3)



|            |                        |                       |                  |
|------------|------------------------|-----------------------|------------------|
| <i>She</i> | <i>was made ill by</i> | <i>curious dishes</i> | <i>sometimes</i> |
|            |                        | Actor                 |                  |
| Subject    |                        |                       |                  |
| Theme      |                        |                       |                  |

Table 2.4: Different conflations of Actor, Subject and Theme (4)

|           |                       |            |                               |
|-----------|-----------------------|------------|-------------------------------|
| <i>By</i> | <i>curious dishes</i> | <i>she</i> | <i>was made ill sometimes</i> |
|           | Actor                 |            |                               |
|           |                       | Subject    |                               |
|           | Theme                 |            |                               |

Table 2.5: Different conflations of Actor, Subject and Theme (5)

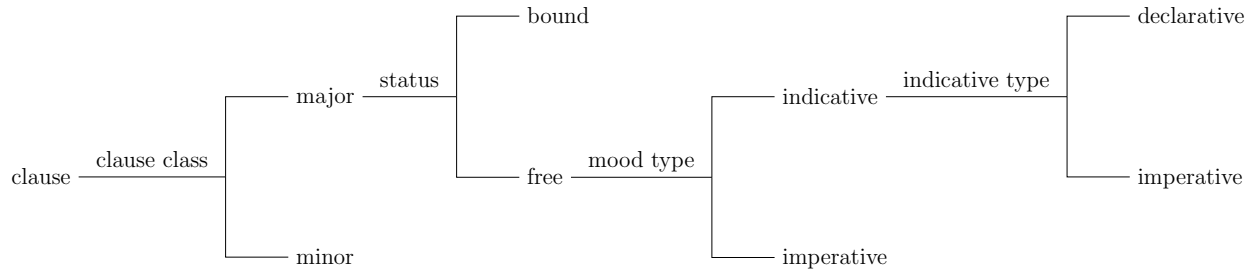


Figure 2.1: Part of the MOOD system (interpersonal metafunction), as defined in [Honnibal \(2004\)](#)

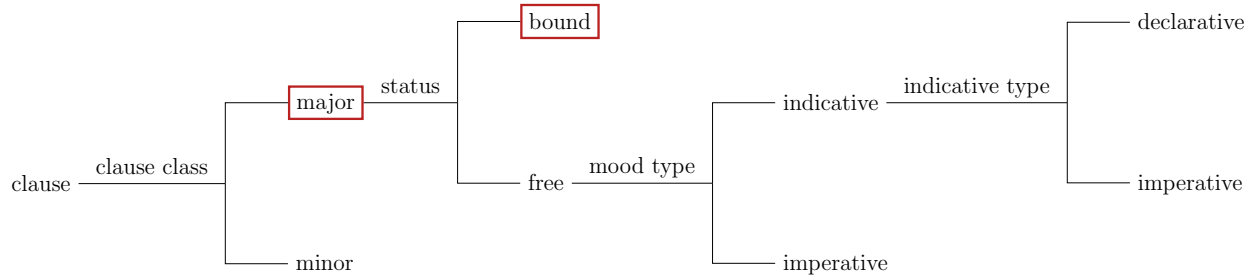


Figure 2.2: Choices in the MOOD system (interpersonal metafunction) for the clause *When the weather was bad*

|              |            |                   |                                  |
|--------------|------------|-------------------|----------------------------------|
| <i>There</i> | <i>she</i> | <i>climbed</i>    | <i>endless flights of stairs</i> |
| Circumstance | Actor      | Process: material | Goal                             |

Table 2.6: Example of constituents tagged with TRANSITIVITY functions, where the Process is material

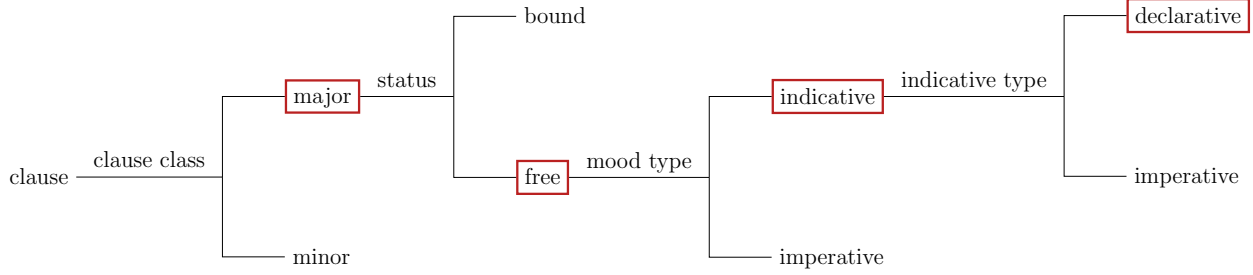


Figure 2.3: Choices in the MOOD system (interpersonal metafunction) for the clause *she leafed through incomprehensible magazines*

|            |                          |                                            |
|------------|--------------------------|--------------------------------------------|
| <i>She</i> | <i>tried to make out</i> | <i>the subjects of vast dark paintings</i> |
| Senser     | Process: mental          | Phenomenon                                 |

Table 2.7: Example of constituents tagged with TRANSITIVITY functions, where the Process is mental

To tag the constituents with the appropriate functional labels, it is crucial to perform a correct constituency analysis first, including ellipsis resolution. A constituent may have a certain tag in the clause where it occurs explicitly, and a different tag in the clause where it is elided. For example, consider the sentence *There she climbed endless flights of stairs and tried to make out the subjects of vast dark paintings*. In the first clause, the constituent *she* is explicit and serves the role of Actor (see Table 2.6), while in the second clause it is elided and its function is Senser instead (see Table 2.7).

## 2.2 Why SFG constituency is hard to parse

Systemic Functional Grammar does not provide any account of the mechanism by which surface strings should be mapped to or from syntactic or semantic structures. This results in much flatter analyses, as the description does not have any of the internal brackets that would need to be introduced to account for coordination. Another particularity of SFG is its representation of ellipsis. Parses are typically presented on top of transformations of the original source string that expand out coordination structures so that elided elements are made explicit. As an illustration, we analyse the same sentence following the style of the Penn Treebank (see Figure 2.5) and the typical SFG representation (see Figure 2.6). A description of SFG non-terminal tags is provided in Table 2.8.

Instead of transforming the original string to account for ellipsis, we introduce secondary

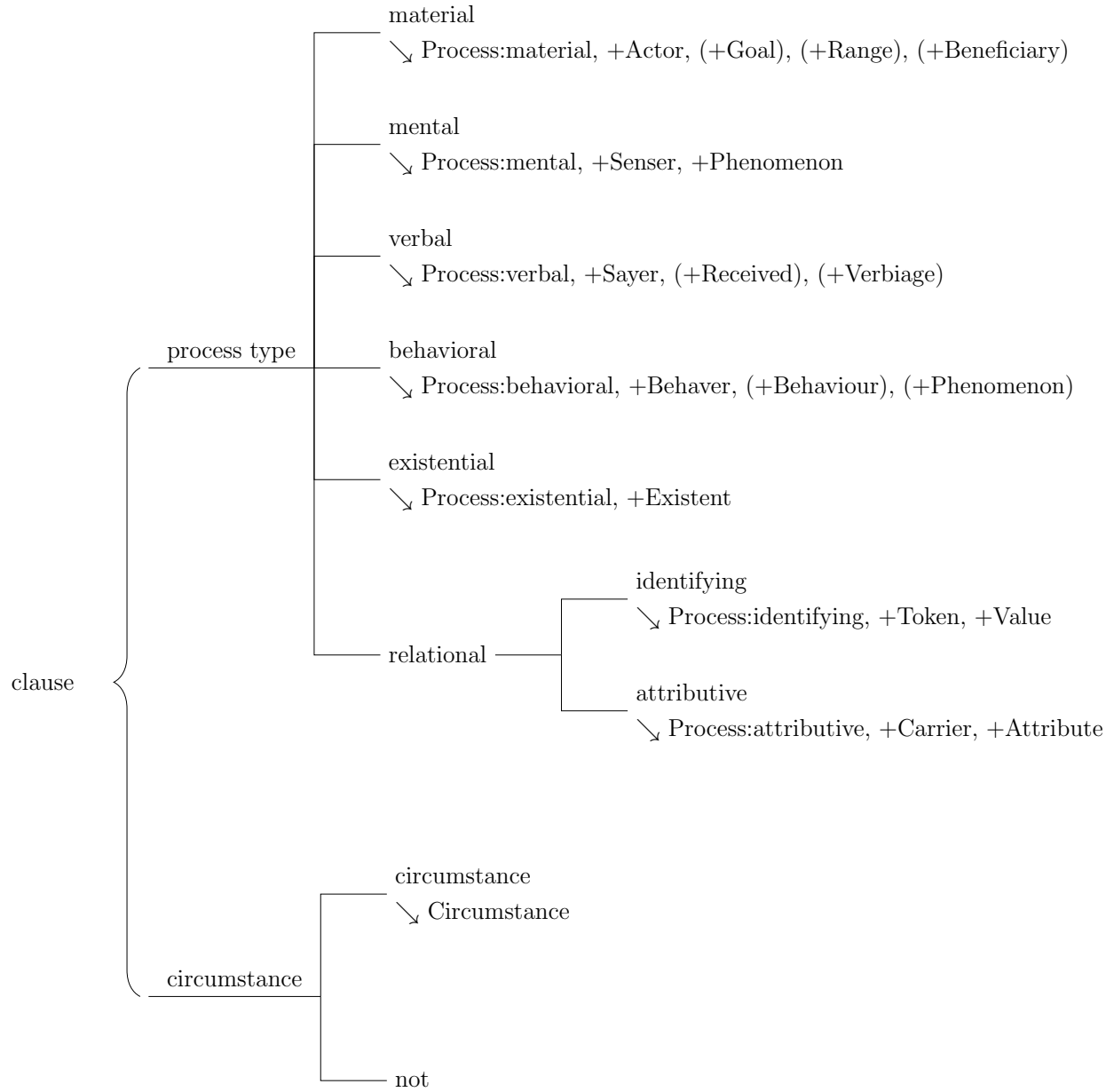


Figure 2.4: Part of the TRANSITIVITY system (experiential metafunction), as defined in Eggins (2004)

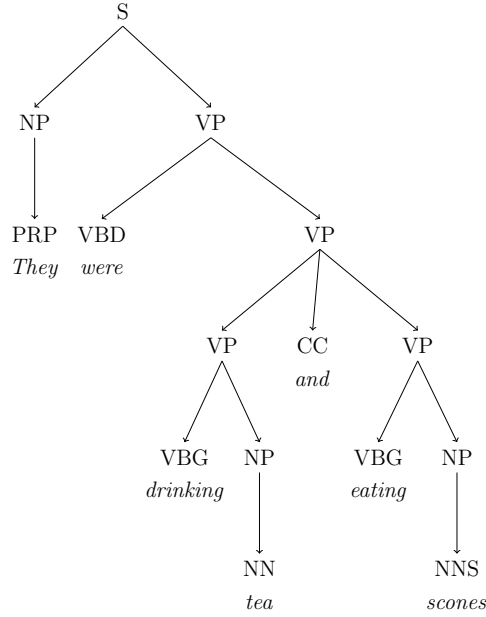


Figure 2.5: Example of a syntactic tree following the Penn Treebank style

edges (see Figure 2.7). This representation is more linguistically motivated than its alternative, conceiving ellipsis as a trace or reference rather than a repetition of explicit parts in the sentence. It also presents a much more interesting task, where ellipsis is to be predicted from the original string. At the same time, the secondary edges make the prediction of SFG constituency more challenging, because they turn the tree representation into a directed acyclic graph. Graph parsing is much more difficult than tree parsing, which is precisely why most linguistic theories describe syntactic structures with trees rather than graphs. In the following chapter, we describe a method for transforming graphs to tractable trees, so that well-tested and accurate tree parsers can be used to predict SFG constituency.

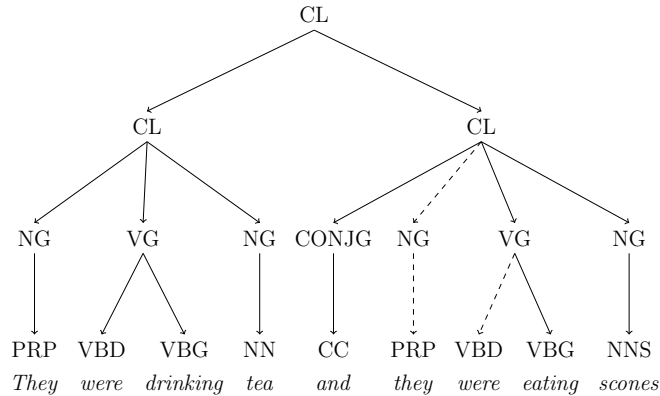


Figure 2.6: SFG constituency as a tree, where elided elements are duplicated

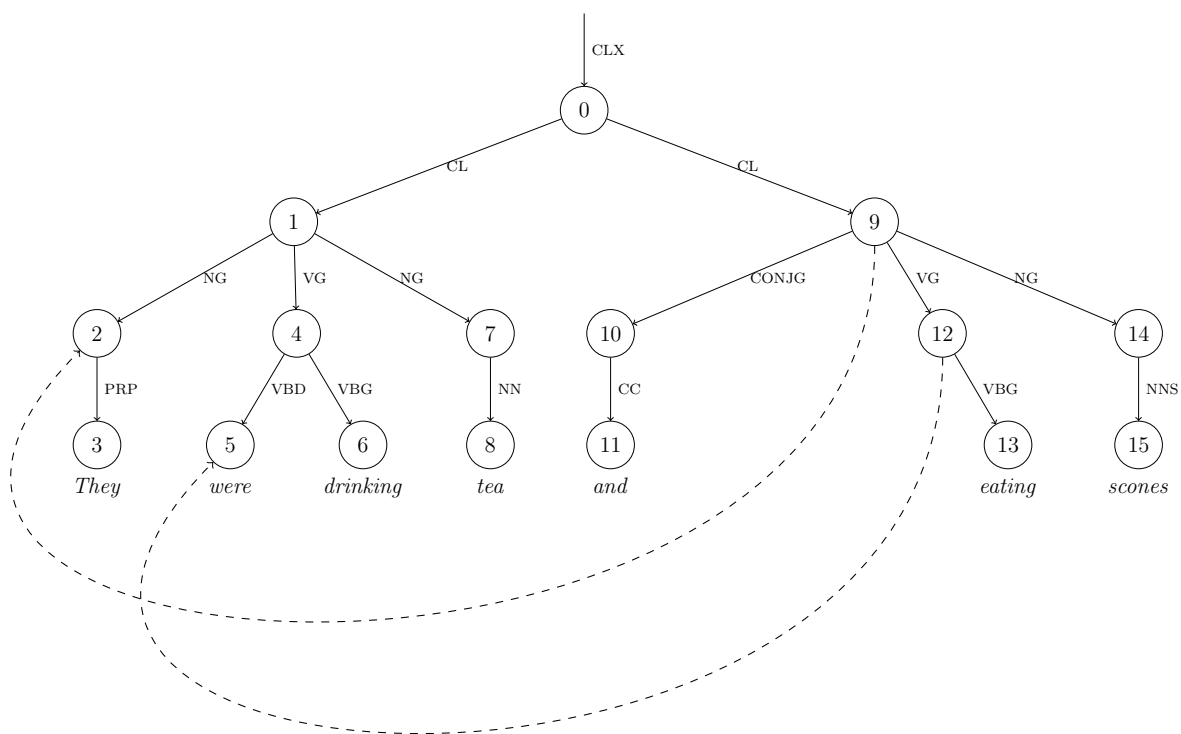


Figure 2.7: SFG constituency as a DAG, where ellipsis is represented with reentrant nodes

| Tag   | Description                  |
|-------|------------------------------|
| CLX   | Clause complex               |
| CL    | Clause                       |
| ADVX  | Adverbial group complex      |
| ADVG  | Adverbial group              |
| CONJG | Conjunction group            |
| INTJX | Interjection group           |
| INTJ  | Interjection                 |
| NGX   | Nominal group complex        |
| NG    | Nominal group                |
| VGX   | Verbal group complex         |
| VG    | Verbal group                 |
| PRT   | Particle                     |
| PPX   | Prepositional phrase complex |
| PP    | Prepositional phrase         |

Table 2.8: Description of SFG non-terminal tags

# Chapter 3

## SFG constituency parsing and ellipsis resolution

### 3.1 Task

In this thesis, we focus on SFG constituency parsing and ellipsis resolution. We leave out the prediction of functional tags, as previous work has shown that this aspect of SFG parsing can be accurately and efficiently derived from the syntax using deterministic rules. For a given sentence  $S$ , the task consists of predicting its graph  $G = (n_1, n_2, \dots, n_m)$ . Each node  $n_i$  consists of the quadruplet  $(L, C, P, EP)$ , where  $L$  is a non-terminal label or PoS tag,  $C$  is an ordered list of children nodes,  $P$  is the parent node, and  $EP$  is a set of parent nodes that elide the current node. We limit the task to resolving ellipsis within sentences.

### 3.2 Related work

As explained in the last section, the most challenging aspect of parsing SFG constituency lies in the identification and resolution of elided nodes, which turn the syntactic structure into a DAG. At the same time, this is the aspect of SFG parsing with most applications outside the SFG domain. Ellipsis resolution is a major source of error in other fields, such as machine translation (Vicedo and Ferrández, 2000) (Chung and Gildea, 2010) (Macketanz et al., 2018), question answering (Hansen and Søgaard, 2019), dialogue understanding (Dzikovska et al., 2009), information extraction (Yuan et al., 2020) and semantic parsing (Groschwitz,

2019). However, it was only recently that ellipsis resolution was recognized as an important NLP task, which explains the scarce literature and annotated corpora dedicated to it. Previous work dealt with verb and noun phrase ellipsis, sluices, coordination ellipsis, gapping and stripping. The approaches range from pattern-based to machine learning and neural network models. State of the art methods experiment with transfer learning and multi-task settings.

Anand and Hardt (2016) present the first system to handle sluice resolution. This type of ellipsis is introduced by a *wh*-expression and, in most of the cases, elides the rest of the question, as in *Harry traveled to southern Denmark to study botany. I want to know why [Harry traveled to southern Denmark to study botany]*. The task can be divided in three parts: ellipsis detection, in which a case of ellipsis is identified, antecedent selection, where the antecedent for a case of ellipsis is found, and ellipsis resolution, in which the ellipsis is filled in with the reference to the antecedent. This paper focuses on the first two parts of the task. The set of candidate antecedents consists of all the sentences within an *n*-sentences radius around the sluice sentence. After extracting structural and content features from them, a maximum entropy classifier is trained to assign probabilities to the sluice candidates.

Liu et al. (2016) decompose the verbal phrase ellipsis (VPE) task in different subtasks and experiment with different learning-based models to solve them both jointly and separately. The three subtasks consist of: 1) target detection: identification of VPE targets, 2) antecedent head resolution: linking each target with the head of its antecedent, and 3) antecedent boundary detection: determining the boundaries of the antecedent. Typically, the first and last subtasks are modeled separately and the second and last are modeled jointly. The authors show that better results in VPE can be achieved if target detection and antecedent boundary detection are solved with the same approach, and a different model is used for antecedent head resolution.

Khullar et al. (2020) present a corpus for noun ellipsis using the Cornell Movie Dialogs Dataset. This type of ellipsis occur when the head inside a noun phrase is elided. For example, consider the following dialog: *Do you have coffee? / In the kitchen / I will make some [coffee] for us*. The authors show that the corpus can be used to train machine learning models by performing both noun ellipsis detection and noun ellipsis resolution experiments. They try different classifiers and get the best results with a Linear Support Vector Machine classifier for ellipsis detection and a Random Forest classifier for ellipsis resolution.

A number of publications tackle coordination ellipsis, especially in the medical domain, in

which it is important to decompose and fill in ellipsis in coordination structures so that specific terminologies can be correctly mapped to concepts in ontologies. The earliest approaches are either rule-based, such as those of [Nhàn et al. \(1989\)](#) and [Okumura and Muraki \(1994\)](#), or based on statistical models, as in [Klavans et al. \(1997\)](#) and [Goldberg \(1999\)](#). More recently, [Teranishi et al. \(2017\)](#) train a deep neural network to identify boundaries in coordinated structures, while [Yuan et al. \(2020\)](#) adopts a pure unsupervised method that reconstructs concepts from coordinated elliptical expressions using graphs.

[Bakhshandeh et al. \(2016\)](#) investigates ellipsis resolution in comparative structures as a crucial aspect of deep language understanding. This work introduces a structured prediction model that jointly captures comparison and ellipsis constructions in sentences such as *The steak sizzled more appetizingly than the burger [sizzled appetizingly]*. Six different types of ellipsis are included: VP-deletion, stripping, pseudo-gapping, gapping, sluicing and subdeletion.

[Zhang et al. \(2019\)](#) explore neural network models for verbal phrase ellipsis (VPE) resolution in both pipeline and end-to-end processes. They divide the task in two parts: VPE detection, which is to detect the auxiliary verb introducing the ellipsis of a verb phrase, and VPE resolution, which consists of identifying the elided verb phrase. The authors experiment with different neural architectures in a VPE detection and resolution pipeline, as well as in an end-to-end process. The neural models they propose achieve better results than the baselines, which are two rule-based approaches and the machine learning model proposed by [Liu et al. \(2016\)](#).

[Aralikatte et al. \(2020\)](#) reformulate ellipsis resolution as a machine reading comprehension (MRC) problem, which enables the use of state-of-the-art neural architectures originally designed for MRC. To fit the MRC format of *(context, question, answer)*, the datasets are restructured so that the document becomes the context, the sentence where the ellipsis occurs in becomes the question, and the antecedent/entity becomes the answer. The authors train different MRC models and transfer them to two ellipsis tasks, namely, sluice ellipsis and verb phrase ellipsis resolution. In addition, they perform experiments in a multi-task setting training both on sluice and verb phrase ellipsis resolution, as well as coreference resolution as an auxiliary task. This approach compensates for the lack of large annotated corpora dedicated to ellipsis resolution, and significantly outperforms previously reported results on this task.

We follow [Yuan et al. \(2020\)](#) and [Aralikatte et al. \(2020\)](#) in jointly solving different tasks, in this case, SFG constituency structure and ellipsis resolution as well as different types of



ellipsis. We propose a novel approach for ellipsis resolution, namely, representing ellipsis with reentrant nodes in DAG structures, and using tractable transformations to transform those DAGs to trees so that they can be learned by standard parsers. We experiment with different transformation techniques, described in detail in the following chapter. Like [Khullar et al. \(2020\)](#), we obtain the best results when we tackle ellipsis detection and ellipsis resolution separately: the first one with the tree parser of choice, and the second one with deterministic rules at post-processing (see the `start-end-extra-node-heuristic` strategy described in Chapter 4).

### 3.3 Dataset

The SFGbank of [Honnibal \(2004\)](#) was developed by converting the Penn Treebank II ([Marcus et al., 1994](#)) to SFG constituency structure, making ellipsis explicit and adding SFG function tags. We make further transformations to this corpus by converting the structure from trees to graphs and representing ellipsis with secondary edges instead of transforming the original string.

The SFGbank follows a minimal bracketing style that makes its constituency structure much flatter than that of the Penn Treebank. All complements and adjuncts are raised by attaching them to the clause node instead of the verb. Hypotactic clauses are also raised to be siblings of the nearest clause above them. In the Penn Treebank, each auxiliary and main verb is given their own nodes and are dominated by the auxiliary before it. The SFGbank flattens these auxiliaries as well by attaching them to the verb phrase below it. Finally, verbal phrases that belong to the same clause are raised to be siblings of a same verbal group complex. Other changes in the SFGbank include node relabelling, the addition of group nodes for the conjunctions, and pruning and truncating nodes that contain no lexis or only contain punctuation or traces. In addition, ellipsis is reconstructed by using traces in the Penn Treebank. The elided nodes are duplicated both in the string and the syntactic tree, making them explicit.

Regarding the functional analysis, clauses are tagged with the function labels and constituents with participant roles specific to SFG. [Honnibal \(2004\)](#) and [Costetchi \(2020\)](#) demonstrated that most of these labels can be accurately inferred from the syntax with rules. For this reason, we limit this thesis to the more challenging task of predicting constituency structure, including ellipsis, and leave out the function tags.

We propose a different representation of ellipsis that does not modify the original string

and is also more linguistically motivated. Instead of copying the elided elements, we use secondary edges that point to them. These edges generate reentrant nodes, i.e., nodes with more than one parent. This requires converting the tree structures of the SFGbank to DAGs. We create a node for each constituent and include in it the information of its explicit parent and any other parents that elide the given node, separately. We also include a list of its children, both elided and non-elided, in the order that they appear in the string. Other metadata that we provide with each node includes: SFG constituent label (e.g., **CLX**, **CL**, **VG**, **NG**, etc), function label (e.g., **subject**, **object**), index of the node introducing the parent clause, whether it is a terminal or not, and the text of the node if it is a terminal.

Our dataset includes subject ellipsis, noun phrase ellipsis and gapping, which occur mostly within coordinated structures. Subject ellipsis sometimes includes an elided auxiliary, as in Figure 2.7, and/or an adjunct. Noun phrase ellipsis refers to the omission of a noun and potentially its modifiers from a noun phrase, like *the other [teapot]* in Figure 3.1. Gapping occurs in coordinate structures, when a finite verb and potentially any other non-finite verbs in the conjunct are elided. We can find an example of gapping in the elided verb *had* in Figure 3.1.

We use the conventional split of the Penn Treebank for our dataset. The train set includes sections 2-21 of the Wall Street Journal, the development set consists of section 22 and the testing set of section 23.

## 3.4 Evaluation

Problems with constituency evaluation have been known for a long time (Carroll et al., 2002). These issues are particularly acute for our data. If we compare each predicted node  $n_{p_i}$  with its corresponding gold node  $n_{g_i}$ , the accuracy scores can be severely affected by misalignment issues. When our system fails to predict a non-terminal node or predicts an extra one, the indices for all the posterior nodes in the tree are shifted and consequently tagged as wrong, even if the prediction error was actually minimal. For this reason, we decided to convert the constituency graphs to dependencies before evaluating them. This fixes the number of nodes to evaluate to the number of terminals, solving the misalignment problem. Even though some information is lost through this transformation, it preserves the key structure we are interested in evaluating and proves to reflect the performance of the models much more accurately.

The first step of the conversion consists of assigning a terminal as a head for each constituent.

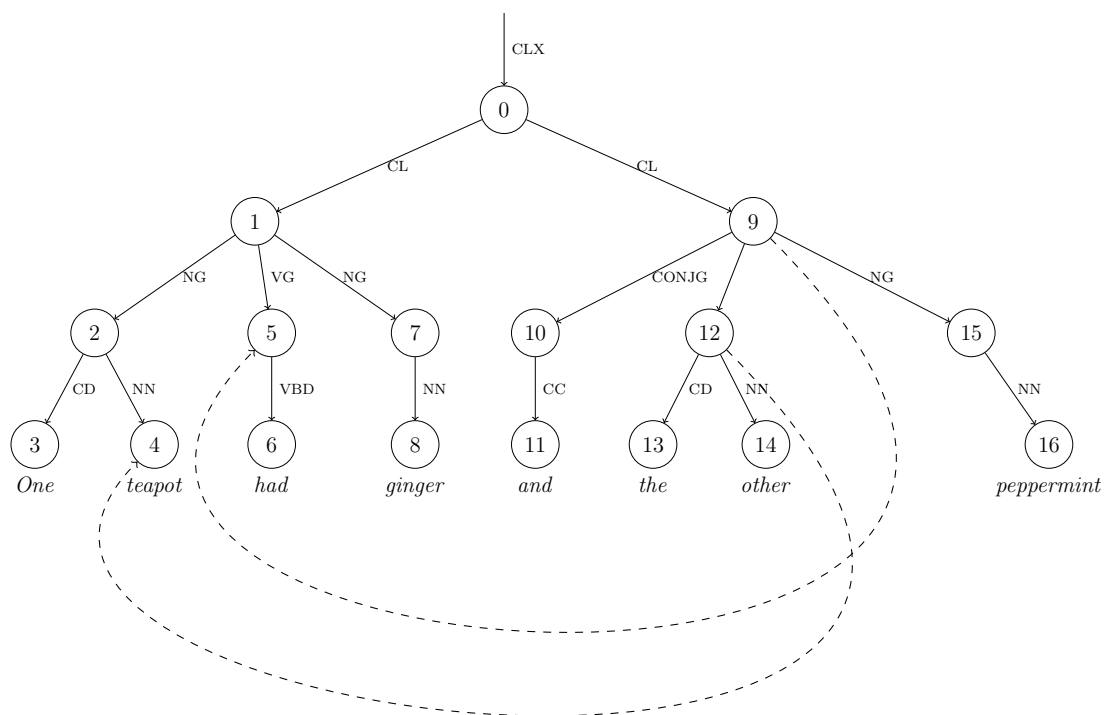


Figure 3.1: An example DAG for a sentence with noun phrase ellipsis (node 4) and gapping (node 5)

| Rank | Non-terminal labels     | Terminal labels                                                   |
|------|-------------------------|-------------------------------------------------------------------|
| 0    | CLX                     | -                                                                 |
| 1    | CL                      | -                                                                 |
| 2    | VGX, VG                 | MD, VB, VBD, VBG, VBN, VBP, VBZ                                   |
| 3    | NGX, NG                 | NN, NNS, NNP, NNPS, PRP, PRP\$                                    |
| 4    | ADVX, ADVG, PPX, PP     | DT, EX, FW, IN, J, JJR, JJS, RB, RBR, RBS, TO, WDT, WP, WP\$, WRB |
| 5    | CONJG, INTJX, INTJ, PRT | CC, CD, LS, PDT, RP, SYM, UH                                      |

Table 3.1: Ranks for constituent tags, which are used to find the head of each constituent when converting the parse tree to dependencies

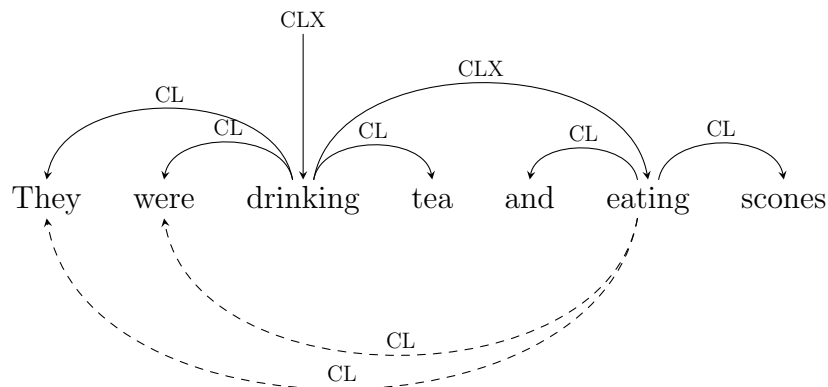


Figure 3.2: Example of a dependency graph, where ellipsis edges are represented with dashed lines

We use a set of rules to rank the terminals in the constituent according to their PoS tags and choose the one with the highest rank (see Table 3.1). For example, the head of the constituent *They were drinking tea* in Figure 2.7 is the terminal *drinking*, with PoS tag *VBG*. Instead of using the constituent IDs as parents for each node, we now use their assigned head. Finally, we delete all the non-terminal nodes. The conversion produces a dependency graph like the one in Figure 3.2, which corresponds to the constituency graph in Figure 2.7.

We consider a node correct when it has the right parents for both elided and non-elided edges. When a parent elides the node, we also check that it positions it in the right order among its children. In addition, we check that the edges have the correct labels for labeled scores. We calculate precision, recall and F scores.

# Chapter 4

## Transformations for tractable SFG parsing

### 4.1 Related work

Graph parsing can be approached by developing dedicated graph parsers or, alternatively, by using tree approximations. The latter involves converting graphs to trees, training a standard tree parser and reversing the output trees to graphs. The main advantage of this approach is that it makes it possible to use very well-tested and accurate tree parsers. Its main drawback is that some information is lost when converting the graphs to trees and reversing these transformations. Previous work that deals with graph parsing via tree approximations includes [Nivre and Nilsson \(2005\)](#)’s pseudo-projective dependency parsing, [Koller and Kuhlmann \(2009\)](#)’s algorithm for extracting dependencies from combinatory categorial grammar (CCG) derivations, [Kummerfeld et al. \(2012\)](#)’s CCG to probabilistic context-free grammar (PCFG) conversion method and [Agić et al. \(2015\)](#), who present tree approximations of semantic dependency graphs. In all of these works, structures that would not be recognisable with some algorithmic approach are transformed prior to parsing, with information being packed into some dimension that the algorithm is less sensitive to, such as the label set or bracketing depth.

[Nivre and Nilsson \(2005\)](#) show that a standard dependency parser restricted to projective structures can be combined with graph transformation techniques to predict non-projective structures. The training data for the parser is transformed by reattaching non-projective arcs higher in the tree until they are projective. The information from the original attachment

place is encoded in the label of the new arc. The parser learns the projective arcs with the decorated labels, which are used later at the post-processing stage to lower the arcs to their proper place in the graph. This inverse transformation re-introduces the non-projective arcs, which significantly improves the parsing accuracy for languages with less restrictive word order than English, such as Czech and German. At the same time, this technique allows the use of a standard dependency parser based on the projective assumption, which simplifies the complexity of the task considerably and results in a more robust, accurate and efficient system.

Koller and Kuhlmann (2009) propose a method for encoding CCG derivations as dependency trees. Each composition rule is associated with an equivalent operation for combining partial dependency trees. Instead of combining semantic representations by functional composition and application, the partial dependency trees for the substrings are combined into partial dependency trees for the larger string using forward and backward concatenation. The conversion is not lossless, given that a same dependency tree can be induced by different CCG derivations. However, it encodes the key semantic information that is needed to reconstruct the original CCG representation. The conversion method is used for comparing the strong generative capacities of CCG and tree-adjoining grammar (TAG), and potentially makes it easier to transfer algorithms between formalisms.

Kummerfeld et al. (2012) present a bottom-up method for converting CCG derivations into Penn Treebank phrase-structure trees. Stress is placed on developing a reversible conversion method, so that the original CCG derivations can be recovered, facilitating downstream parser use and parser comparison. First, a set of bracket instructions is assigned to each word based on its lexical category. After that, each combinatory step in the CCG derivation is corresponded by applying or combining instructions to build up the phrase-structure tree. The method preserves 51.1% sentences after reconvertng the trees to CCG derivations, which signifies an improvement with respect to previous work.

Agić et al. (2015) experiment with tree approximations for the Semantic Dependency Parsing (SDP) task. They lossily convert graphs to dependency trees in pre-processing, use them to train standard tree dependency parsers, and finally convert their output back to graphs in post-processing. They highlight the utility of this approach in comparison to developing dedicated graph parsers. Firstly, graph parsing via tree approximation makes use of accurate, efficient and robust standard tree parsers. At the same time, testing the limits of this approach leads to a better understanding of why graphs are preferred for encoding semantic relations, i.e., whether the use of graphs is linguistically motivated or just based on convention. The paper presents four different tree approximations. Prior to training,

the DAGs are transformed to dependency trees by removing edges from reentrant nodes, so that they only have one head. The edges can be either removed via deletion or trimming. Deleted edges cannot be reconstructed at post-processing, while trimmed edges can be recovered deterministically or using lossy heuristics. One way of trimming edges is through label overloading, which involves recording the deleted edge in the label of another edge that is kept. This information can later be used at post-processing to reconstruct the removed edge. The authors explore different combinations of edge deletion and trimming, investigating the properties of DAGs in SDP to provide meaningful representations, as well as measuring the impact of linguistically motivated tree approximations to graph parsing.

Taking inspiration from these approaches, we propose a method for converting SFG derivations to phrase-structure trees that are isomorphic to their original DAG representation. In this way, the SFG structure can be predicted using standard constituency parsers which are accurate and efficient. Such corpus transformations are hard to develop, as they should allow for the recovery of the original SFG structure with minimum information loss. We explore different ways of converting the SFG representations to find an approach that results in a learnable intermediate representation while still allowing the information to be recovered. Our proposed transformations overload labels with the information necessary to convert the trees back to graphs, which is similar to the approach in [Nivre and Nilsson \(2005\)](#) and [Agić et al. \(2015\)](#). Some of our strategies, namely **start-end-extra-node** and **start-end-extra-node-heuristic**, use co-indices to encode both the start and end vertices of secondary nodes, a method that is also employed by [van Noord and Bos \(2017\)](#) in the context of co-reference resolution with graphs.

## 4.2 Suggested transformations

We implemented six different transformation strategies to convert DAGs to reversible trees. We remove the secondary edges, but encode their information in the constituent labels so that they can be later recovered. Such information includes both the location of the ellipsis and the elided node, i.e. the start and end vertices of the secondary edge. We later use these trees to train our parser of choice. Finally, we use the information in the overloaded labels to reconstruct the ellipsis edges, converting the trees back to graphs. We provide examples of the intermediate trees for all the strategies, corresponding to the graph in [Figure 2.7](#).

Our baseline transformation, named **start** strategy, attaches a label to the constituent preceding the ellipsis (where the secondary edge starts). The label includes a marker indicating

that the secondary edge starts in this location, and a tag and an index pointing to the elided constituent (where the secondary edge ends). For example, in Figure 4.1, the label `VGstartVBD0` consists of four parts: `VG`, `start`, `VBD` and `0`. Here, `VG` is the constituent tag of this node, and the rest of the string corresponds to the extra information related to the secondary edge. The substring `start` works as a marker and indicates that the ellipsis is located right after this node. The substring `VBD` means that the elided constituent has a `VBD` tag, and the index `0` indicates that the elided constituent is the first one in the tree with a `VBD` tag.

For the `start-without-pos` strategy, we attach the ellipsis labels only to non-terminal nodes. The intuition behind this approach is that it is harder for the model to learn the ellipsis labels if they are part of the PoS tags. The parser of Kitaev and Klein (2018) that we use predicts the PoS tags with a separate tagger, after building the syntactic parse. The exact mechanism of the tagger is not discussed in the documentation, so we assume that, like in other neural syntactic taggers, it predicts each PoS label by looking at the input signal and only one or two of the previous predictions, or none at all. For the ellipsis resolution task, however, the surrounding structure is very important. The model needs to know what the rest of the parse looks like to predict something valid. If we encode the ellipsis information in the non-terminal labels instead, the main parsing algorithm will take them into consideration for its predictions. Following this intuition, the `start-without-pos` strategy does not overload a terminal node whenever an ellipsis occurs right after it, but uses the last non-terminal node instead. An example is provided in Figure 4.2. A possible drawback of this strategy is that in some cases it is not able to encode the exact place where the ellipsis occurs. Consider the example in Figure 4.3. The representation is ambiguous and can either produce the sentence *One teapot had ginger and the [teapot] other [had] peppermint* or *One teapot had ginger and the other [teapot] [had] peppermint*.

The `end` strategy is inverse to the two previous methods. Instead of placing the ellipsis label where the secondary edge starts, it uses the tag of the node where the edge ends. For example, in Figure 4.4, the `VBDendVG1` label means that this `VBD` node is elided right after the second `VG` node.

The `end-extra-node` strategy shares the same motivation with the `start-without-pos` strategy. The elided constituents are wrapped with an extra node, which carries the ellipsis tag. This avoids the use of PoS tags whenever the elided constituent is located right after a terminal. An example is provided in Figure 4.5.

A potential problem with the aforementioned transformations is that they produce a large



number of different ellipsis tags. To address this issue, we developed a further strategy, named **start-end-extra-node**. This technique encodes both the start and end of the ellipsis edges by wrapping the appropriate constituents with extra nodes, as in the **end-extra-node** transformation. Since the location of the start and end labels indicates both vertices of the secondary edge, it is not necessary to encode them by using constituent tags and indices like we do for the rest of the techniques, resulting in coarser ellipsis labels. In addition to the **start** and **end** markers, we use an ID to identify which of the ellipsis in the sentences those markers refer to. For example, in the tree approximation provided in Figure 4.6, one of the secondary edges is encoded in the extra nodes **CCstart0** and **NGend0**, which indicate the start and end vertices respectively. The other secondary edge is encoded in the constituents **VGstart1** and **VBDend1**.

Finally, we implemented the **start-end-extra-node-heuristic** strategy, which converts graphs to trees in the same way as the **start-end-extra-node** technique, but without the IDs. The start and end nodes that correspond to a same ellipsis edge are identified by using an heuristic at the post-processing stage. An example of a tree before using the heuristic is provided in Figure 4.7, and the same example after assigning the IDs at post-processing can be seen in Figure 4.8. The ending nodes are enumerated from left to right. If a node has two or more **end** markers, all of them get the same IDs. In the example, there are two nodes with **end** markers. Each of them has two markers, meaning that the constituent is elided in two different clauses. Following our heuristic, the first one gets the ID 0 for both of its markers, and the second one gets the ID 1 also for both of its markers. The starting nodes are enumerated from left to right as well. If a node has two or more **start** markers, they all get different start IDs. In addition, whenever a new clause is encountered, the **start** IDs are restored to zero. Following this logic, the markers in the second clause *and [they] [were] eating scones* get the IDs 0 and 1, forming the respective **CCstart0** and **VGstart1** labels. The same process applies to the third clause, *but [they] [were] taking out the raisins*. Our heuristic is built on the assumption that elided constituents are always recovered from the same clause (usually the first one), and that they appear necessarily in the same order. We can make this assumption because the types of ellipsis that occur in our dataset originate from expanding coordinated structures into different clauses, carrying the subject along with their modifiers (auxiliaries, adjuncts, parts of noun-phrases, etc) as a same block. Consider the previous example: in the Penn Treebank, the parse tree would contain only one clause, as in Figure 4.9. In the SFG parse, the **VP** nodes corresponding to *eating scones* and *taking out the raisins* are assigned their own clauses and raised in the tree so that they are a siblings of the main clause *They were drinking tea*. The new clauses are no longer children or siblings of the **NP** and **VBD** constituents corresponding to the subject *They* and the auxiliary *were*. These constituents are instead elided in our SFG parse, but

```

(CLX
  (CL
    (NG
      (PRP They))
    (VG
      (VBD were)
      (VBG drinking)
    (NG
      (NN tea))))
  (CL
    (CONJG
      (CCstartNG0 and))
    (VGstartVBDO
      (VBG eating))
    (NG
      (NNS scones))))

```

Figure 4.1: Example of a graph to tree transformation using the **start** strategy

they still keep the same structure from the original Penn Treebank representation, i.e., they appear in the same order. The representation of coordinated structures as distinct clauses with the same hierarchy also justifies restoring the IDs for the ellipsis markers for each one of them. The gain in performance with this strategy (see 4.1) supports the assumptions on which we base our heuristic. In addition, we converted the trees to graphs right away, without any training or parsing in between (see Table 4.2) and manually checked the wrong examples to identify problems with this transformation strategy. We could observe that they are originated from inconsistencies in the SFGbank and not with the method itself. For example, the following sentence breaks the assumption that constituents are elided in the order that they appear in the clause where they are explicit, but this happens only because the quotation marks are wrongly tagged as elided: *In an announcement to its staff last week , executives at Time Warner Inc. ’s weekly magazine said Time will “ de-emphasize ” dramatically its use of electronic giveaways such as telephones in television subscription drives ; [“] [Time] [will] cut the circulation it guarantees advertisers by 300,000 , to four million ; and [“] [Time] [will] increase the cost of its annual subscription rate by about \$ 4 to \$ 55 .*

```

(CLX
  (CL
    (NG
      (PRP They))
    (VG
      (VBD were)
      (VBG drinking))
    (NG
      (NN tea))))
  (CL
    (CONJGstartNGO
      (CC and))
    (VGstartVBDO
      (VBG eating))
    (NG
      (NNS scones))))

```

Figure 4.2: Example of a graph to tree transformation using the **start-without-pos** strategy

```

(CLX
  (CL
    (NP
      (CD One)
      (NN teapot))
    (VP
      (VBD had))
    (NP
      (NN ginger)))
  (CL
    (CONJG
      (CC and))
    (NPstartNNO
      (CD the)
      (NN other))
    (NP
      (NN peppermint))))

```

Figure 4.3: Example of a graph to tree transformation using the **start-without-pos** strategy where the exact location of the ellipsis is ambiguous

```

(CLX
  (CL
    (NGendCC0
      (PRP They))
    (VG
      (VBDendVG1 were)
      (VBG drinking)
      (NG
        (NN tea))))
  (CL
    (CONJG
      (CC and))
    (VG
      (VBG eating))
    (NG
      (NNS scones))))

```

Figure 4.4: Example of a graph to tree transformation using the **end** strategy

```

(CLX
  (CL
    (NGendCC0
      (NG
        (PRP They)))
    (VG
      (VBDendVG1
        (VBD were))
      (VBG drinking)
      (NG
        (NN tea))))
  (CL
    (CONJG
      (CC and))
    (VG
      (VBG eating))
    (NG
      (NNS scones))))

```

Figure 4.5: Example of a graph to tree transformation using the **end-extra-node** strategy

```

(CLX
  (CL
    (NGend0
      (NG
        (PRP They)))
    (VG
      (VBDend1
        (VBD were))
      (VBG drinking)
      (NG
        (NN tea))))
    (CL
      (CONJG
        (CCstart0
          (CC and)))
      (VGstart1
        (VG
          (VBG eating)))
      (NG
        (NNS scones))))
  )

```

Figure 4.6: Example of a graph to tree transformation using the `start-end-extra-node` strategy

```

(CLX
  (CL
    (NGendend
      (NG
        (PRP They)))
    (VG
      (VBDendend
        (VBD were))
      (VBG drinking)
      (NG
        (NN tea))))
    (CL
      (CONJG
        (CCstart
          (CC and)))
      (VGstart
        (VG
          (VBG eating)))
      (NG
        (NNS scones)))
    (CL
      (CONJG
        (CCstart
          (CC but)))
      (VGstart
        (VG
          (VBG taking)
          (IN out)))
      (NG
        (DT the)
        (NNS raisins))))
  )

```

Figure 4.7: Example of a graph to tree transformation using the **start-end-extra-node-heuristic** strategy

```

(CLX
  (CL
    (NGend0end0
      (NG
        (PRP They)))
    (VG
      (VBDend1end1
        (VBD were))
      (VBG drinking)
      (NG
        (NN tea))))
    (CL
      (CONJG
        (CCstart0
          (CC and)))
      (VGstart1
        (VG
          (VBG eating)))
      (NG
        (NNS scones)))
    (CL
      (CONJG
        (CCstart0
          (CC but)))
      (VGstart1
        (VG
          (VBG taking)
          (IN out)))
      (NG
        (DT the)
        (NNS raisins))))
  )

```

Figure 4.8: Example of a graph to tree transformation using the **start-end-extra-node-heuristic** strategy, after enumerating the ellipsis labels at post-processing

```

(S
  (NP
    (PRP They))
  (VP
    (VBD were)
    (VP
      (VP
        (VP
          (VBG drinking)
          (NP (NN tea)))
        (CC and)
        (VP
          (VBG eating)
          (NP
            (NNS scones))))))
      (CC but)
      (VP
        (VBG taking)
        (PRT
          (RP out))
        (NP
          (DT the)
          (NNS raisins))))))

```

Figure 4.9: The same example from Figures 4.7 and 4.8, but using the Penn Treebank representation, which does not expand coordinated structures in different clauses



### 4.3 Methodology

Our system consists of three main parts: 1) pre-processing, where the graphs are converted to trees using one of our tractable transformation strategies, 2) training/parsing, in which we train and parse the trees with a standard parser, and 3) post-processing, where we convert the output of the parser from trees to DAGs.

At the pre-processing step, we convert our graph representations to phrase-structure trees. The graphs are linearized with a non-recursive depth-first search algorithm and the ellipsis information is encoded using the transformations described in Section 4.2.

The output trees are used to train the standard parser of choice, which later predicts the constituency structure. We use the Berkeley neural parser developed by Kitaev and Klein (2018), with further modifications from Kitaev et al. (2019). This model uses a self-attentive encoder and a chart decoder customized for parsing, as can be seen in Figure 4.10.

The first part of the encoder assigns a context-aware representation  $y_t$  to each position  $t$  in the sentence. We use the best performing model of the parser, which fine-tunes pre-trained BERT embeddings (Devlin et al., 2019) to generate these representations.

The second part of the encoder combines the vectors  $y_t$  to calculate span scores  $s(i, j, l)$  for each constituent in the sentence, where  $i$  is the position where the span begins,  $j$  marks where the span ends, and  $l$  corresponds to the label of the span. Unlike previous work on constituency parsing, this portion of the encoder does not rely on a sequence-to-sequence architecture, such as a recurrent neural network (RNN) or a long-short-term memory network (LSTM). Instead, position information is captured with a self-attentive mechanism adapted from Vaswani et al. (2017). With this method, different parts of the sentence can attend to each other based on their positions and contents, and be weighted according to their relevance to the task.

The score of each tree  $T$  is defined as the sum of its constituent scores  $s(i, j, l)$ :

$$s(T) = \sum_{(i,j,l) \in T} s(i, j, l)$$

The model optimal tree

$$\hat{T} = \operatorname{argmax} s(T)$$

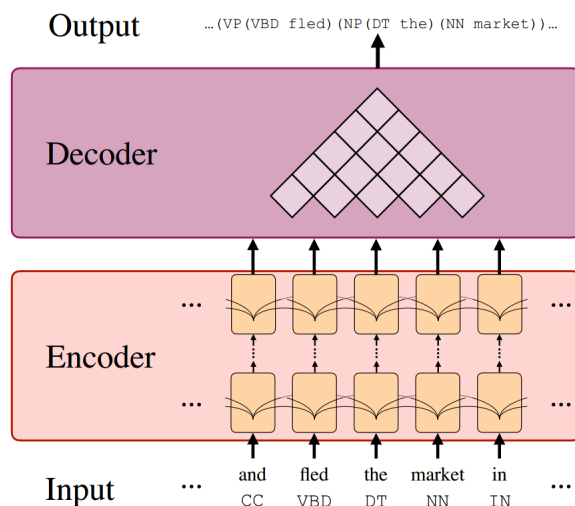


Figure 4.10: The Berkeley parser combines a chart decoder with a sentence encoder based on self-attention. Illustration taken from [Kitaev and Klein \(2018\)](#)

is found using a chart decoder with a CKY-style inference algorithm adapted from [Stern et al. \(2017\)](#) and [Gaddy et al. \(2018\)](#).

A separate tagger is used to predict part-of-speech labels, either from an external library or an implementation that comes with the same parser. We opted for the latter. The architecture of this tagger is not explained in the papers or the code documentation, so we assume that it follows a conventional neural architecture for part-of-speech taggers, which only considers one or two of the surrounding predictions to decide a label, or does not consider them at all.

The final stage of post-processing in our system consists of transforming the predicted trees back to graphs. The phrase-structure trees are traversed from left to right, and a node is created for each constituent, including the information of its children, parent, tag and token text in the case of terminals. The ellipsis information is stripped away from the labels and used to recover the secondary edges. These are saved in the graph by using children and parent information as with the rest of the edges, but a different parent list is used so that parents that explicitly point to the node can be differentiated from parents that elide it.

As an external comparison, we use [Dozat and Manning \(2018\)](#)’s semantic dependency parser (SDP) after converting our constituency graphs to dependencies. We use the [SemEval format](#) and add ellipsis markers to differentiate secondary edges from explicit ones. An example can be seen in Figure 4.11. The SDP parser we use is based on the dependency tree parser of [Dozat and Manning \(2017\)](#), which is extended to process DAGs. The model consists of two

|   |          |   |   |   |   |   |     |            |
|---|----------|---|---|---|---|---|-----|------------|
| 1 | They     | - | - | - | - | - | CL  | CLellipsis |
| 2 | were     | - | - | - | - | - | CL  | CLellipsis |
| 3 | drinking | - | - | + | + | - | -   | -          |
| 4 | tea      | - | - | - | - | - | CL  | -          |
| 5 | and      | - | - | - | - | - | -   | CL         |
| 6 | eating   | - | - | - | + | - | CLX | -          |
| 7 | scones   | - | - | - | - | - | -   | CL         |

Figure 4.11: Example of a graph converted to the SemEval format. The columns correspond to ID, form, lemma, PoS, top, predicate, frame, and an argument column for as many predicates as there are in the sentence. We use a dummy label `-` for the lemma, PoS and frame columns, a `+` for the top column if the token is the root of the sentence, and a `+` for the predicate column if the token has any outgoing edges. The argument columns ( $arg_1, arg_2, \dots, arg_i$ ) contain the edge labels relative to the  $i$ -th predicate. Additionally, we use an `ellipsis` marker to differentiate elided edges. For more details on the format, refer to the official [SemEval documentation](#).

modules: one that predicts whether an edge exists between two words, and another one that predicts the best label for that edge. First, word and PoS tag embeddings are concatenated and fed into a multilayer bidirectional LSTM to generate contextual word representations. Then, the edges and labels are predicted using a linear classifier. We predict the SFG DAGs with this model and evaluate the output directly, given that our evaluation method is also based on dependencies. However, it is important to note that this makes the evaluation biased towards the semantic dependency parser, as they both use the same representations.

## 4.4 Results and discussion

We test the different transformation strategies with our dependency-based evaluation method and calculate scores for all the edges, only the edges that recover elided constituents, and only the edges that represent explicit parts of the sentence. We report unlabeled and labeled precision, recall and F scores in Table 4.1.

To identify how lossy our conversion methods are, we transform the trees in the test set back to graphs directly, without any training or parsing in between. As can be observed in Table 4.2, the loss in unlabeled F score for elided nodes is less than 0.3% for all of our methods, except for the `start-end-extra-node-heuristic` strategy, which has a loss of 3.3%. Considering both the elided and non-elided edges, the loss in unlabeled F score is

0.2% for this method and close to zero for all the others.

As expected, it is much harder for the model to learn the secondary edges, which can be observed in the stark difference in scores for elided edges and non-elided edges.

The **start** and **start-without-pos** strategies achieve similar scores, with **start-without-pos** performing worse at recovering elided edges. As explained before, the **start-without-pos** strategy only uses non-terminals for the ellipsis labels and, for this reason, it fails to encode the exact location of the ellipsis when it occurs between terminal leaves. This could explain the drop in performance.

The **end** and **end-extra-node** strategies get significantly higher scores than the other approaches. This shows that the parser is better at learning transformations that encode ellipsis by appending tags to the end rather than the start of the secondary edges. Between the two methods, **end-extra-node** performs better, with an increase in F score of 1.1 for unlabeled edges and 1.4 for labeled edges. As expected, using wrapper nodes to carry the ellipsis tags constitutes a more learnable representation. Probably, this is because the strategy avoids appending ellipsis tags to the terminals. In this way, ellipsis is predicted fully by the main parser algorithm and not by the PoS tagger.

We examined wrong predictions made by the **end-extra-node** model to identify its main issues and address them in improved transformation strategies. The most common problem is that the system does not predict any ellipsis or it predicts an incomplete tag. For example, instead of predicting (VBPendVG1endVG2 (VBP do)), it predicts (VBP do) or (VBPendVG1 (VBP do)). This explains why recall is lower than precision.

Another recurrent error is that the model correctly identifies a node as elided, wrapping it with an extra node, but it assigns the wrong label to it. The ellipsis label can have the wrong constituent tag, the wrong index for that tag, or both. Sometimes, the label points to a node that does not exist at all. For example, the label VBPendVG2 for a tree that only contains VG constituents up to index 1 or does not contain any VG constituents at all.

In other cases, the ellipsis label contains the tag of a node that is present in the tree, but does not coincide with the gold label and generates an ungrammatical or meaningless sentence. Take the predicted tree in Figure 4.12 as an example. The gold secondary edge created by the label NGendCC2 produces the sentence *Mega-hits in Germany or Italy rarely make it even to France or Great Britain, and [Mega-hits in Germany or Italy] almost never show up on U.S. screens.*, while the predicted secondary edge generated by the label NGendCC1 produces the wrong sentence *Mega-hits in Germany or Italy rarely make it even to France or*

*[Mega-hits in Germany or Italy] Great Britain, and almost never show up on U.S. screens.*

Sometimes, missing, incomplete or seemingly wrong ellipsis labels generate more appropriate sentences than the gold standard. We provide an example in Figure 4.13. The predicted sentence *Renaissance, which manages about \$1.8 billion, drew stiff criticism from many clients earlier this year because it pulled entirely out of stocks at the beginning of the year and thus [it] missed a strong rally.* is more appropriate than the gold *Renaissance, which manages about \$1.8 billion, drew stiff criticism from many clients earlier this year because it pulled entirely out of stocks at the beginning of the year and thus [it] [thus] missed a strong rally.* This originates from problems in the SFGbank corpus and usually involves the incorrect recovery of adverbial phrases and other adjuncts as ellipsis.

In a few number of cases, the extra node wraps the wrong constituent, resulting in a graph that recovers the incorrect elided elements. However, this usually happens with ambiguous examples, where an elided node can be more than one constituent in the sentence. For example, consider the predicted parse in Figure 4.14, corresponding to the sentence *Analysts say they're faster and they're priced aggressively at \$2400 to \$5000 and carry more memory than anything else of their size in the market.* The last clause can either recover the subject *they* from the first or the second clause.

Finally, there are some examples where the ellipsis tag is correctly predicted, but mistakes in the rest of the constituency parse affect the recovery of the right elided elements.

We decided to address the most common issue, which are false negatives, i.e., predicting that a node is not elided when it actually is. One possible explanation for this problem is that the vast majority of nodes in the data are not elided, resulting in a very unbalanced dataset. The issue could also be caused by the use of very fine-grained ellipsis tags. All of the mentioned strategies identify one vertex of a secondary edge with the location of the ellipsis label, and the other vertex with the label description. This label includes the constituent tag of the node itself, the **start** or **end** marker, the constituent of the node to recover and its index. The number of available tags to predict is equal to the number of combinations that all those variables can produce. Too many tags means there will be too little probability mass for ellipsis tags, and too much for the non-ellipsis tags, which are fewer in number. The model will be then more predisposed to predict that there is no ellipsis at all. One way to address this issue is by using coarser, and thus fewer, ellipsis tags. We developed the **start-end-extra-node** transformation with this in mind and achieved better results, suggesting that the fine-grained tags are indeed a problem.

Our ellipsis resolution task can be seen as two-sided. It involves predicting start and end tags

```

(CLX
  (CL
    (NGendCC1
      (NG
        (NG
          (NNS Mega-hits))
        (PP
          (IN in)
          (NG
            (NNP Germany)
            (CC or)
            (NNP Italy))))))
      (ADVG
        (RB rarely))
      (VG
        (VBP make))
      (NG
        (PRP it))
      (PP
        (ADVG
          (RB even))
        (PP
          (TO to)
          (NGX
            (NG
              (NNP France))
            (CC or)
            (NG
              (NNP Great)
              (NNP Britain))))))
    (, ,))
  (CL
    (CONJG
      (CC and))
    (ADVG
      (RB almost)
      (RB never))
    (VG
      (VBP show)
      (PRT
        (RP up)))
    (PP
      (IN on)
      (NG
        (NNP U.S.)
        (NNS screens)))
    (. .)))

```

Figure 4.12: Example of a wrong prediction with the **start-end-extra-node** strategy. The elided node CC1 exists, but generates an ungrammatical/meaningless sentence. The correct tag should be NGendCC2, instead of NGendCC1

```

(CLX
  (CL
    (NG
      (NG
        (NNP Renaissance)
        (, .))
      (CL
        (NG
          (WDT which))
        (VG
          (VBZ manages))
        (NG
          (RB about)
          ($ $)
          (CD 1.8)
          (CD billion)))
        (, .))
      (VG
        (VBD drew))
      (NG
        (JJ stiff)
        (NN criticism))
      (PP
        (IN from)
        (NG
          (JJ many)
          (NNS clients)))
      (NG
        (RBR earlier)
        (DT this)
        (NN year))
      (CL
        (CONJG
          (IN because))
        (NGendCCO
          (NG
            (PRP it)))
        (VG
          (VBD pulled))
        (ADVG
          (RB entirely))
        (PP
          (IN out)
          (PP
            (IN of)
            (NG
              (NNS stocks))))
        (PP
          (IN at)
          (NG
            (DT the)
            (NN beginning))
          (PP
            (IN of)
            (NG
              (DT the)
              (NN year))))
        (CONJG
          (CC and))
        (ADVGenCL3
          (ADVG
            (RB thus))))
      (CL
        (VG
          (VBD missed))
        (NG
          (DT a)
          (JJ strong)
          (NN rally))
        (. .)))
    )
  )

```

Figure 4.13: Example of a gold parse for the **start-end-extra-node** strategy. The label **ADVGenCL3** indicates that the adverb *thus* is elided in the last clause, which is actually incorrect. The predicted parse misses this label, generating a more appropriate analysis

```

(CLX
  (CL
    (NG
      (NNS Analysts))
    (VG
      (VBP say))
    (CL
      (NG
        (PRP they))
      (VG
        (VBP 're))
      (NG
        (JJR faster)))
    (CL
      (CONJG
        (CC and))
      (NGendCC1
        (NG
          (PRP they)))
      (VG
        (VBP 're)
        (VBN priced))
      (ADVG
        (RB aggressively))
      (PP
        (IN at)
        (NG
          (\$ \$)
          (CD 2,400)
          (TO to)
          (\$ \$)
          (CD 5,000))))
    (CL
      (CONJG
        (CC and))
      (VG
        (VBP carry))
      (NG
        (JJR more)
        (NN memory))
      (PP
        (IN than)
        (NG
          (NG
            (NN anything))
          (NG
            (RB else)))
        (PP
          (PP
            (IN of)
            (NG
              (PRP\$ their)
              (NN size)))
          (PP
            (IN on)
            (NG
              (DT the)
              (NN market))))))
    (: --))
    (. .))

```

Figure 4.14: Example of a prediction with the **start-end-extra-node** strategy, where the extra node carrying the ellipsis label wraps the wrong elided node. The gold standard wraps the **NG** node from the first clause instead of the second one. However, both options are correct in this case. The last clause can either recover the subject *they* from the first or the second clause



for the right nodes, and identifying which of them correspond to the same ellipsis edge. We developed a subtask where we only evaluate the first part of the task, i.e., predicting start and end tags for the right nodes, to identify which part of the task our models struggle with the most. We tested all our strategies on this subtask, as well as an additional strategy called **subtask-start-end-extra-node**, which is trained to predict only that a node contains start or end tags, without encoding any information needed for actually resolving the ellipsis. The evaluation is performed on the predicted trees directly, before any post-processing step. We perform constituency evaluation and consider only elided nodes by checking how many elided spans in the gold set are predicted by our models. The results can be observed in Table 4.3. We obtained the best scores, particularly recall, with the **subtask-start-end-extra-node**. This suggests that the parser struggles at learning the ellipsis indices. One possible explanation is that the information needed to decide the ellipsis indices is too global. The parser needs to look at the whole tree to infer how to enumerate the **start** and **end** markers, and that tree (including the ellipsis markers) is not determined yet when the scores are computed. The gain in parser performance from omitting the indices is significant enough as to potentially compensate the increase in information loss.

Following this intuition, we developed the **start-end-extra-node-heuristic** strategy, which omits indices in the ellipsis tags, and instead uses a simple heuristic at the post-processing stage to reconstruct the graphs. This strategy achieves a gain in recall of 4.5 for unlabeled elided edges with respect to the **start-end-extra-node** transformation, confirming our hypothesis. It also gets a significant improvement in unlabeled F1 score, placing it as the best performing strategy. All the scores are the highest for this technique when considering both elided and non-elided edges.

Additionally, we report the scores obtained with Dozat and Manning (2018)’s semantic dependency parser. It is important to recall that our evaluation is dependency-based and therefore it might be biased towards the SDP approach, although interesting conclusions can still be inferred from this experiment. The model performs better than our approach for elided edges only, which is expected given that the parser is developed with the aim to parse graphs and is therefore better suited to handling reentrancy. However, the semantic dependency parser is worse than our approach at predicting the rest of the constituency parse, which can be observed in the lower scores for non-elided edges. This is also an expected result, considering that our task is based on constituencies, not dependencies. Overall, these scores suggest that our approach may be preferable to using a graph parser for solving this specific task, but it also shows that there is significant room for improvement.

|                                | UP          | UR          | UF          | LP          | LR          | LF          |
|--------------------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| <i>all edges</i>               |             |             |             |             |             |             |
| start                          | 86.5        | 86.5        | 86.5        | 85.1        | 85.1        | 85.1        |
| start-without-pos              | 86.6        | 86.6        | 86.6        | 85.2        | 85.2        | 85.2        |
| end                            | 87.7        | 87.7        | 87.7        | 86.1        | 86.1        | 86.1        |
| end-extra-node                 | 88.8        | 88.8        | 88.8        | 87.5        | 87.5        | 87.5        |
| start-end-extra-node           | 88.9        | 88.9        | 88.9        | 87.3        | 87.3        | 87.3        |
| start-end-extra-node-heuristic | <b>89.1</b> | <b>89.1</b> | <b>89.1</b> | <b>87.6</b> | <b>87.6</b> | <b>87.6</b> |
| sdp                            | 88.0        | 88.0        | 88.0        | 86.5        | 86.5        | 86.5        |
| <i>elided edges</i>            |             |             |             |             |             |             |
| start                          | 65.6        | 44.5        | 53.0        | 65.1        | 44.2        | 52.6        |
| start-without-pos              | 65.5        | 40.6        | 50.1        | 64.9        | 40.3        | 49.7        |
| end                            | 75.6        | 61.4        | 67.7        | 75.6        | 61.4        | 67.7        |
| end-extra-node                 | 77.0        | 60.7        | 67.9        | 77.0        | 60.7        | 67.9        |
| start-end-extra-node           | 86.6        | 62.7        | 72.7        | <b>84.3</b> | 61.0        | 70.8        |
| start-end-extra-node-heuristic | <b>87.0</b> | 67.2        | 75.8        | 83.2        | 64.3        | 72.6        |
| sdp                            | 81.2        | <b>77.0</b> | <b>79.0</b> | 80.5        | <b>76.3</b> | <b>78.3</b> |
| <i>non-elided edges</i>        |             |             |             |             |             |             |
| start                          | 90.8        | 90.8        | 90.8        | 89.4        | 89.4        | 89.4        |
| start-without-pos              | 91.0        | 91.0        | 91.0        | 89.5        | 89.5        | 89.5        |
| end                            | 89.8        | 89.8        | 89.8        | 88.2        | 88.2        | 88.2        |
| end-extra-node                 | 91.1        | 91.1        | 91.1        | <b>89.7</b> | <b>89.7</b> | <b>89.7</b> |
| start-end-extra-node           | <b>91.2</b> | <b>91.2</b> | <b>91.2</b> | 89.5        | 89.5        | 89.5        |
| start-end-extra-node-heuristic | 91.0        | 91.0        | 91.0        | <b>89.7</b> | <b>89.7</b> | <b>89.7</b> |
| sdp                            | 90.1        | 90.1        | 90.1        | 88.6        | 88.6        | 88.6        |

Table 4.1: Scores for the ellipsis and coordination expansion task (dependency evaluation)

|                                | UP    | UR    | UF    | LP    | LR    | LF    |
|--------------------------------|-------|-------|-------|-------|-------|-------|
| <i>all edges</i>               |       |       |       |       |       |       |
| start                          | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| start-without-pos              | 100.0 | 100.0 | 100.0 | 99.9  | 99.9  | 99.9  |
| end                            | 100.0 | 100.0 | 100.0 | 99.9  | 99.9  | 99.9  |
| end-extra-node                 | 100.0 | 100.0 | 100.0 | 99.9  | 99.9  | 99.9  |
| start-end-extra-node           | 100.0 | 100.0 | 100.0 | 99.9  | 99.9  | 99.9  |
| start-end-extra-node-heuristic | 99.8  | 99.8  | 99.8  | 99.6  | 99.6  | 99.6  |
| sdp                            | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| <i>elided edges</i>            |       |       |       |       |       |       |
| start                          | 99.7  | 99.7  | 99.7  | 98.7  | 98.7  | 98.7  |
| start-without-pos              | 99.7  | 99.7  | 99.7  | 99.0  | 99.0  | 99.0  |
| end                            | 99.7  | 99.7  | 99.7  | 98.7  | 98.7  | 98.7  |
| end-extra-node                 | 99.7  | 99.7  | 99.7  | 98.7  | 98.7  | 98.7  |
| start-end-extra-node           | 99.7  | 99.7  | 99.7  | 98.7  | 98.7  | 98.7  |
| start-end-extra-node-heuristic | 97.4  | 96.1  | 96.7  | 94.7  | 93.5  | 94.1  |
| sdp                            | 100   | 99.7  | 99.8  | 100.0 | 99.7  | 99.8  |
| <i>non-elided edges</i>        |       |       |       |       |       |       |
| start                          | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| start-without-pos              | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| end                            | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| end-extra-node                 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| start-end-extra-node           | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| start-end-extra-node-heuristic | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| sdp                            | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |

Table 4.2: Results from reversing the graph to tree conversion directly, without any training or parsing in between (dependency evaluation)

|                              | UP          | UR          | UF          | LP          | LR          | LF          |
|------------------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| start                        | 93.1        | 77.1        | 84.4        | 87.9        | 72.7        | 79.7        |
| start-without-pos            | <b>96.9</b> | 73.7        | 83.7        | <b>95.3</b> | 72.5        | 82.4        |
| end                          | 94.6        | 79.2        | 86.2        | 93.5        | 78.3        | 85.2        |
| end-extra-node               | 93.6        | 76.5        | 84.2        | 93.2        | 76.2        | 83.8        |
| start-end-extra-node         | 94.6        | 82.5        | 88.1        | 92.9        | 81.3        | 86.7        |
| subtask-start-end-extra-node | 94.7        | <b>84.8</b> | <b>89.5</b> | 93.7        | <b>84.0</b> | <b>88.6</b> |

Table 4.3: Scores for the ellipsis and coordination expansion subtask (constituency evaluation)

# Chapter 5

## Comparison to previous SFG parsing work

### 5.1 State of the art

The earliest attempts at building a SFG parser date back to the 1970s, 1980s and 1990s and include the rule-based systems of [Winograd \(1972\)](#), [Kasper \(1988\)](#), [O’Donoghue \(1991\)](#) and [O’Donnell \(1994\)](#). To our knowledge, the only work on SFG parsing in recent years is that of [Costetchi \(2020\)](#), which achieves the state of the art results to this date. Their system is based on the Stanford Dependency parser 3.5 ([Chen and Manning, 2014](#)) and trained on the original Penn Treebank. They convert the output of the Stanford parser at post-processing to fit the SFG constituency structure via hand-made rules. The conversion recovers elided subjects but ignores all other types of ellipsis. The constituents are tagged with functional labels, which are inferred deterministically from the syntax. In addition, external lexical sources are used to tag constituents with transitivity labels, as they carry semantic information that cannot be inferred from the syntax alone.

### 5.2 Evaluation

We use the same evaluation method and corpora from [Costetchi \(2020\)](#) to compare the performance of our system with respect to theirs. The test data consists of the OE corpus, annotated by [Schulz \(2015\)](#) and consisting of 1503 clauses and 20864 words, and the OCD

|                   | Precision    | Recall       | F score      |
|-------------------|--------------|--------------|--------------|
| <i>OE corpus</i>  |              |              |              |
| Costetchi (2020)  | 50.16        | <b>97.15</b> | 65.73        |
| Our system        | <b>54.42</b> | 93.75        | <b>68.75</b> |
| <i>OCD corpus</i> |              |              |              |
| Costetchi (2020)  | 61.45        | 90.67        | 73.25        |
| Our system        | <b>76.32</b> | <b>94.61</b> | <b>84.48</b> |
| <i>Average</i>    |              |              |              |
| Costetchi (2020)  | 55.81        | 93.91        | 69.49        |
| Our system        | <b>65.36</b> | <b>94.18</b> | <b>76.61</b> |

Table 5.1: Comparison with Costetchi (2020)’s parser, using their evaluation method. Only the constituency structure (without ellipsis) is considered

corpus, created by Ela Oren and Eugen Costetchi at the psychology faculty of Tel-Aviv University and comprising 529 clauses and 8605 words. The datasets contain syntactic and functional annotations specific to SFG. The constituent spans predicted by the systems are compared to those in the gold standard to calculate precision, recall and F scores. As this thesis is focused only on constituency structure, we ignore the functional annotations for our comparison. We do not consider ellipsis resolution either, given that Costetchi (2020)’s system is only able to predict subject ellipsis. The results can be seen in Table 5.1.

## 5.3 Discussion

As can be observed in Table 5.1, our approach to predicting SFG constituency structure outperforms Costetchi (2020) system. This shows that training a parser directly on an SFG corpus is not only simpler and consequently more efficient than their pipeline architecture, but also more accurate. Costetchi’s system requires the implementation of hand-written rules to convert the Stanford’s dependencies to SFG constituency structure, which is time-consuming, error prone and computationally more expensive, given that the conversion is performed during run-time. In addition, most of the transformation rules tackle known errors produced by the Stanford parser, which makes it harder to use another parser as a base. Changing it by a newer and improved parser would require hand-writing the rules again, making Costetchi (2020)’s system less scalable than ours. Another advantage of our

approach with respect to this previous system is that it handles other types of ellipsis besides subject ellipsis, namely, noun phrase ellipsis and gapping.

# Chapter 6

## Conclusions

In this thesis, we have presented the task of SFG constituency prediction and ellipsis resolution. We represent SFG syntactic structure as a direct acyclic graph (DAG), where secondary edges are introduced to make ellipsis explicit in a way that is linguistically sound. We provide a dataset and an evaluation method, and demonstrate their utility to train machine learning models by performing our own experiments.

Our approach jointly learns SFG constituency structure and ellipsis resolution via tractable DAG to tree transformations. During pre-processing, we remove the secondary edges that represent ellipsis and overload the constituency labels of the starting and/or ending nodes of those edges so that they can be recovered at a later stage. The resulting trees are used to train a standard tree parser and then converted back to DAGs by reconstructing the secondary edges using the overloaded constituency labels.

We experimented with different transformation strategies to find a learnable representation with minimum information loss and that is able to predict SFG constituency and ellipsis with high accuracy. As expected, the elided edges were harder to predict than the non-elided ones. Some of the representations overload only the label of the starting node of the secondary edge, some only the label of the ending node, and some overload both. We got the best results when we used both vertices to encode this information and employed IDs to indicate which starting and ending vertices correspond to a same secondary edge. We also noticed a gain in accuracy if we wrapped the starting and ending constituents with an extra node containing the overloaded tag.

To identify which part of the ellipsis resolution task our models struggled with the most, we designed a subtask for only ellipsis prediction. This consists of identifying that an ellipsis



occurs, without recovering the constituent that is elided. The difference in scores with the full task was significant enough as to suggest that the hardest part to learn is not recognizing the vertices of the secondary edges, but identifying which vertices correspond to each other, i.e., predicting the IDs that match starting and ending nodes. For this reason, we tried an additional strategy that does not learn any IDs during parsing and instead uses a simple heuristic at the post-processing stage to match the starting and ending vertices. Even though the heuristic is not lossless, the gain from using a simpler and more learnable representation was high enough to boost the accuracy of our system. Through these experiments, we have demonstrated that SFG constituency and ellipsis resolution can be jointly predicted via tractable DAG to tree transformations, and that the way in which we structure these transformations have an impact on performance.

Additionally, we predicted the graphs directly with a graph dependency parser to compare it to our approach. Each system performed better at what was initially built for: the graph parser was better at recovering the secondary edges, i.e., predicting multiple heads per node, while our approach using a constituency tree parser was more suitable to predicting the rest of the analysis, which originates precisely from constituencies. As the non-elided edges are much more numerous than the elided ones, the overall accuracy of our system is higher than that of the graph dependency parser. However, the experiment involving the semantic dependency parser showed that there is still room for improvement and that a better system could be built if we combined the strengths of both approaches.

Finally, we compared our approach to predicting SFG constituency structure to that of previous work, specifically, [Costetchi \(2020\)](#)’s parser, which presented the best results in SFG parsing up to this date. We used their same testing set and evaluation method and obtained higher scores with our system, showing that training a parser directly from an SFG specific dataset is not only a simpler and more scalable technique, but also results in better accuracy. In addition, our approach to parsing using tractable transformations is able to predict subject ellipsis, gapping and noun-phrase ellipsis, unlike the parser of [Costetchi \(2020\)](#), which only focuses on subject ellipsis. Our system therefore outperforms the previous work on SFG parsing and presents a new and viable approach to ellipsis resolution, with potential applications inside and outside the SFG domain.

# Bibliography

- Željko Agić, Alexander Koller, and Stephan Oepen. 2015. [Semantic dependency graph parsing using tree approximations](#). In *The 11th International Conference on Computational Semantics (IWCS 2015)*.
- Pranav Anand and Daniel Hardt. 2016. [Antecedent selection for sluicing: Structure and content](#). In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1243, Austin, Texas. Association for Computational Linguistics.
- Rahul Aralikkatte, Matthew Lamm, Daniel Hardt, and Anders Søgaard. 2020. [A simple transfer learning baseline for ellipsis resolution](#).
- Omid Bakhshandeh, Alexis Cornelia Wellwood, and James Allen. 2016. [Learning to jointly predict ellipsis and comparison structures](#). In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 62–74, Berlin, Germany. Association for Computational Linguistics.
- Marie Candito, Bruno Guillaume, Guy Perrier, and Djamé Seddah. 2017. [Enhanced UD Dependencies with Neutralized Diathesis Alternation](#). In *Depling 2017 - Fourth International Conference on Dependency Linguistics*, Pisa, Italy.
- John Carroll, Anette Frank, Dekang Lin, Detlef Prescher, and Hans Uszkoreit. 2002. Beyond parseval-towards improved evaluation measures for parsing systems. In *Workshop at the 3rd International Conference on Language Resources and Evaluation LREC-02., Las Palmas*.
- Danqi Chen and Christopher Manning. 2014. [A Fast and Accurate Dependency Parser using Neural Networks](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

- Tagyoung Chung and Daniel Gildea. 2010. *Effects of empty categories on machine translation*. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 636–645, Cambridge, MA. Association for Computational Linguistics.
- Eugeniu Costetchi. 2020. *Parsimonious Vole: A Systemic Functional Parser for English*. Ph.D. thesis.
- Marie-Catherine De Marneffe and Christopher D Manning. 2008. *Stanford typed dependencies manual*. Technical report, Stanford University.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. *Bert: Pre-training of deep bidirectional transformers for language understanding*.
- Timothy Dozat and Christopher D. Manning. 2017. *Deep biaffine attention for neural dependency parsing*.
- Timothy Dozat and Christopher D. Manning. 2018. *Simpler but more accurate semantic dependency parsing*.
- Myroslava Dzikovska, Charles Callaway, Elaine Farrow, Johanna Moore, Natalie Steinhauser, and Gwendolyn Campbell. 2009. *Dealing with interpretation errors in tutorial dialogue*. In *Proceedings of the SIGDIAL 2009 Conference*, pages 38–45, London, UK. Association for Computational Linguistics.
- Suzanne Eggins. 2004. *Introduction to Systemic Functional Linguistics*. A&C Black.
- David Gaddy, Mitchell Stern, and Dan Klein. 2018. *What’s going on in neural constituency parsers? an analysis*.
- Miriam Goldberg. 1999. An unsupervised model for statistically determining coordinate phrase attachment. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 610–614.
- Edward Gorey. 1980. *Amphigorey: Fifteen Books*. Penguin.
- Jonas Groschwitz. 2019. *Methods for taking semantic graphs apart and putting them back together again*. Ph.D. thesis.
- M.A.K. Halliday and C.M.I.M. Matthiessen. 2004. *An Introduction to Functional Grammar*. Arnold.
- Victor Petrén Bach Hansen and Anders Søgaard. 2019. *What do you mean ‘why?’: Resolving sluices in conversations*.

- Matthew Honnibal. 2004. [Converting the Penn Treebank to systemic functional grammar](#). In *Proceedings of the Australasian Language Technology Workshop 2004*, pages 147–154.
- Robert T Kasper. 1988. An experimental parser for systemic grammars. In *Coling Budapest 1988 Volume 1: International Conference on Computational Linguistics*.
- Payal Khullar, Kushal Majmundar, and Manish Shrivastava. 2020. [NoEl: An annotated corpus for noun ellipsis in English](#). In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 34–43, Marseille, France. European Language Resources Association.
- Nikita Kitaev, Steven Cao, and Dan Klein. 2019. [Multilingual Constituency Parsing with Self-Attention and Pre-Training](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.
- Nikita Kitaev and Dan Klein. 2018. [Constituency Parsing with a Self-Attentive Encoder](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.
- Judith L Klavans, Christian Jacquemin, and Evelyne Tzoukermann. 1997. A natural language approach to multi-word term conflation. In *Proceedings of the DELOS conference*. Citeseer.
- Alexander Koller and Marco Kuhlmann. 2009. [Dependency trees and the strong generative capacity of ccg](#). In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–468. Association for Computational Linguistics.
- Jonathan K. Kummerfeld, Dan Klein, and James R. Curran. 2012. [Robust Conversion of CCG Derivations to Phrase Structure Trees](#). In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 105–109, Jeju Island, Korea.
- Zhengzhong Liu, Edgar González Pellicer, and Daniel Gillick. 2016. [Exploring the steps of verb phrase ellipsis](#). In *Proceedings of the Workshop on Coreference Resolution Beyond OntoNotes (CORBON 2016)*, pages 32–40, San Diego, California. Association for Computational Linguistics.
- Vivien Macketanz, Eleftherios Avramidis, Aljoscha Burchardt, and Hans Uszkoreit. 2018. [Fine-grained evaluation of German-English machine translation based on a test suite](#). In

- Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 578–587, Belgium, Brussels. Association for Computational Linguistics.
- Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. *The Penn Treebank: Annotating Predicate Argument Structure*. In *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.
- NT Nhàn, N Sager, M Lyman, LJ Tick, F Borst, and Y Su. 1989. A medical language processor for two indo-european languages. In *Proceedings. Symposium on Computer Applications in Medical Care*, pages 554–558. American Medical Informatics Association.
- Joakim Nivre, Paola Marongiu, Filip Ginter, Jenna Kanerva, Simonetta Montemagni, Sebastian Schuster, and Maria Simi. 2018. *Enhancing Universal Dependency Treebanks: A Case Study*. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 102–107, Brussels, Belgium. Association for Computational Linguistics.
- Joakim Nivre and Jens Nilsson. 2005. *Pseudo-Projective Dependency Parsing*. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.
- Rik van Noord and Johan Bos. 2017. *Dealing with co-reference in neural semantic parsing*. In *Proceedings of the 2nd Workshop on Semantic Deep Learning (SemDeep-2)*, pages 41–49, Montpellier, France. Association for Computational Linguistics.
- Michael O’Donnell. 1994. *Sentence Analysis and Generation—a Systemic Perspective*. Ph.D. thesis, University of Sydney.
- Tim O’Donoghue. 1991. *The Vertical Strip Parser: A lazy approach to parsing*. University of Leeds, School of Computer Studies.
- Akitoshi Okumura and Kazunori Muraki. 1994. Symmetric pattern matching analysis for english coordinate structures. In *Fourth Conference on Applied Natural Language Processing*, pages 41–46.
- Anke Schulz. 2015. *Me, myself and I: A corpus-based, contrastive study of English and German computer-mediated communication from a Systemic Functional perspective*. Ph.D. thesis, Technische Universität, Darmstadt.
- Sebastian Schuster and Christopher D. Manning. 2016. *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*), pages 2371–2378, Portorož, Slovenia. European Language Resources Association (ELRA).
- Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. [A Minimal Span-Based Neural Constituency Parser](#).
- Hiroki Teranishi, Hiroyuki Shindo, and Yuji Matsumoto. 2017. Coordination boundary identification with similarity and replaceability. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 264–272.
- Aryeh Tiktinsky, Yoav Goldberg, and Reut Tsarfaty. 2020. [pyBART: Evidence-based Syntactic Transformations for IE](#).
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#).
- José L. Vicedo and Antonio Ferrández. 2000. [Importance of pronominal anaphora resolution in question answering systems](#). In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 555–562, Hong Kong. Association for Computational Linguistics.
- Terry Winograd. 1972. Understanding natural language. *Cognitive psychology*, 3(1):1–191.
- Chi Yuan, Yongli Wang, Ning Shang, Ziran Li, Ruxin Zhao, and Chunhua Weng. 2020. A graph-based method for reconstructing entities from coordination ellipsis in medical text. *Journal of the American Medical Informatics Association*, 27(9):1364–1373.
- Wei-Nan Zhang, Yue Zhang, Yuanxing Liu, Donglin Di, and Ting Liu. 2019. A neural network approach to verb phrase ellipsis resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7468–7475.