THE UNIVERSITY
of EDINBURGH

Understanding and Generating Language with Abstract Meaning Representation

Marco Damonte

Doctor of Philosophy

Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh

2019

Abstract

Abstract Meaning Representation (AMR) is a semantic representation for natural language that encompasses annotations related to traditional tasks such as Named Entity Recognition (NER), Semantic Role Labeling (SRL), word sense disambiguation (WSD), and Coreference Resolution. AMR represents sentences as graphs, where nodes represent concepts and edges represent semantic relations between them.

Sentences are represented as graphs and not trees because nodes can have multiple incoming edges, called *reentrancies*. This thesis investigates the impact of reentrancies for parsing (from text to AMR) and generation (from AMR to text). For the parsing task, we showed that it is possible to use techniques from tree parsing and adapt them to deal with reentrancies. To better analyze the quality of AMR parsers, we developed a set of fine-grained metrics and found that state-of-the-art parsers predict reentrancies poorly. Hence we provided a classification of linguistic phenomena causing reentrancies, categorized the type of errors parsers do with respect to reentrancies, and proved that correcting these errors can lead to significant improvements. For the generation task, we showed that neural encoders that have access to reentrancies outperform those who do not, demonstrating the importance of reentrancies also for generation.

This thesis also discusses the problem of using AMR for languages other than English. Annotating new AMR datasets for other languages is an expensive process and requires defining annotation guidelines for each new language. It is therefore reasonable to ask whether we can share AMR annotations across languages. We provided evidence that AMR datasets for English can be successfully transferred to other languages: we trained parsers for Italian, Spanish, German, and Chinese to investigate the cross-linguality of AMR. We showed cases where translational divergences between languages pose a problem and cases where they do not. In summary, this thesis demonstrates the impact of reentrancies in AMR as well as providing insights on AMR for languages that do not yet have AMR datasets.

Lay summary

Smartphones, tablets, and personal computers can predict the words we are about to type, correct spelling mistakes and show us relevant advertisements, among other applications. Personal assistants are becoming increasingly popular with products such as Google Assistant, Microsoft Cortana, Amazon Alexa, and Apple Siri. Automatic translation services such as Google Translate are becoming increasingly reliable. The area of research that enabled these applications is known as Natural Language Processing (NLP).

NLP deals with human-computer interactions based on a natural language, such as English. Its goal is to enable machines to *understand* the meaning of what we say and to *generate* responses and perform actions based on these conversations. Understanding refers to the process of converting natural language into a language interpretable by machines. Generation is the process of allowing machines to generate new text, for instance, in response to a question.

A crucial issue faced by NLP researchers is how to devise a language that is interpretable by machines. It needs to express the meaning of natural language, yet allow machines to easily process it. For instance, when reading the sequence of words *John's red car*, we know that the writer is talking about a *car*, which is of color *red* and is owned by *John*. To facilitate machines to understand such a phrase, we instead use a more explicit language which specifies the relationships between the person *John*, the color *red*, and the object *car*.

To represent the meaning of natural language in machines, this thesis uses a language called Abstract Meaning Representation (AMR). We study both the problem of understanding natural language and the problem of generating natural language. We implement and analyze algorithms that can automatically convert natural language into AMR (understanding) and vice-versa (generation).

Acknowledgements

Inizio i ringraziamenti con la parte sentimentale. La persona che ringrazio di più, per l'amore incondizionato e la fiducia che non ho sempre meritato, è mia madre. Grazie anche a mio padre e mia sorella per avermi dato la possibilità di studiare e perchè, nel bene e nel male, ci sono sempre stati. Nonostante non ce lo diciamo spesso, ci vogliamo bene. Grazie zia, per essere stata un salvagente in momenti difficili. Un grazie anche a tutti i miei amici e le persone che mi sono state vicine e mi hanno convinto di continuare a studiare.

Gracias a mi novia y cómplice, amor de mi vida, por el apoyo moral. Gracias por ser tan juguetona y bromista como yo. Somos un equipo fantastico.

The beginning of my Ph.D. coincided with the start of a sport, Taekwondo, which has been incredibly strategic for not losing my mind. When things do not go your way, kicking people in a safe setting is better than throwing your laptop out the window. I want to thank my instructor Jonathan and all the fantastic people I have met at the UETKD club in these years. Academically, I express immense gratitude to those teachers that, together with my mom, convinced me that it was worth investing in my education. Thank you Davide Testuggine for telling me about Natural Language Processing. I am especially thankful to Shay Cohen and Adam Lopez, who allowed me to work on this thesis and taught me all I know about research. I also want to thank Giorgio Satta, who worked with me during my first year, as well as my MSc supervisor Nathan Schneider and Lexi Birch, who also convinced me to join the Ph.D. program. There is a large group of Ph.D. students that I need to acknowledge for lunch conversations, social gatherings, and for being great friends: Joachim, Joana, Sameer, Sorcha, Ida, Clara, Federico, Esma, Nicola, Kristina, and many others. Thanks to EdinburghNLP, formerly known as ProbModels, for hosting great talks and inviting amazing speakers. I would also like to thank all of those who gave me feedback on drafts, papers, and presentations, even the harshest anonymous reviewers. Finally, I thank the examiners Johan Bos and Ivan Titov for their useful feedback on this thesis.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Marco Damonte*)

A mia mamma

Contents

1

Introduction

"Open the pod bay doors, Hal."
"I'm sorry, Dave. I'm afraid I can't do that."
– 2001: A Space Odyssey

Smartphones, tablets, and personal computers use Natural Language Processing (NLP) to predict the words we type, correct spelling mistakes and show us relevant advertisements, among other applications. Smartphones, tablets, and personal computers can predict the words we are about to type, correct spelling mistakes and show us relevant advertisements, among other applications. Personal assistants are becoming increasingly popular with products such as Google Assistant, Microsoft Cortana, Amazon Alexa, and Apple Siri. Automatic translation services such as Google Translate are becoming increasingly reliable, especially for high-resource language pairs (Wu et al., 2016). Healthcare has also started exploiting NLP to facilitate or speed up information retrieval and improve diagnostics (Hodgson and Coiera, 2015; Demner-Fushman et al., 2009). We now expect machines to *understand* the meaning of what we say and to *generate* responses and perform actions based on these conversations. Ultimately, NLP promises to enable human-computer interfaces, or even computer-computer interfaces, entirely based on natural language, as greatly anticipated by the movie industry. [1]

A pipeline for human-computer interaction A traditional human-computer interaction pipeline includes several components, as shown in Figure 1.1. The

[1] In 1968, Stanley Kubrick released "2001: A Space Odyssey". In 2019, NLP technologies are far behind the psychotic computer from Kubrick's masterpiece, ethics considerations aside.

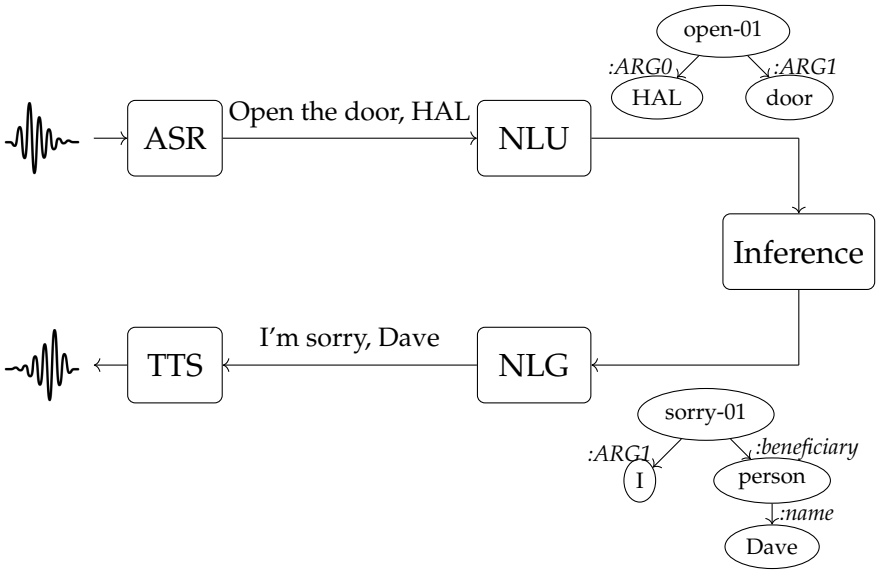


Figure 1.1: Diagram of a pipeline that receives speech, processes it and produces speech in output.

entry point is an Automatic Speech Recognition (ASR) component that transforms input speech into written language. A Natural Language Understanding (NLU) component then converts written language into a Meaning Representation Language (MRL), a process known as semantic parsing. In the example of Figure 1.1, the input sentence *Open the door, HAL* is converted into a meaning representation, which is a tree where the node *open* has two children: *HAL* and *door*. The output of the NLU component is then passed into an inference component (for example, a dialogue manager in dialogue systems and chatbots), which can reason over the meaning representation. The output of this component is the meaning representation of a response (such as an answer or a translation). A Natural Language Generation (NLG) component can then generate text from it. Finally, a Text-To-Speech (TTS) step generates the output speech. Figure 1.1 describes an application of such a pipeline: the modeling of dialogues. Other typical applications are Machine Translation (MT), where a sentence is automatically translated into a different language, and text summarization, where the goal is to shorten a text document (or documents).

Recent advancements in end-to-end approaches attempt to implicitly capture the meaning representations through distributed representations (Bahdanau et al., 2015; Luong et al., 2015; Vinyals and Le, 2015). The pipeline above however remains the de facto standard, and an arguably more elegant solu-

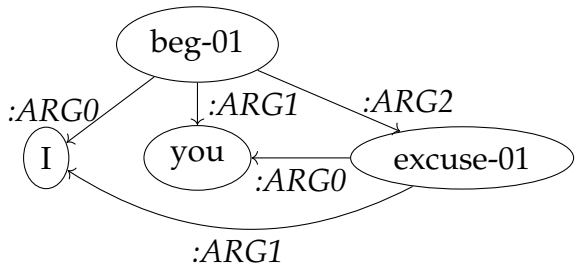


Figure 1.2: AMR graph for the sentence *I beg you to excuse me*.

tion, to commercial human-computer interfaces. In this thesis, we focus on two components of the pipeline: NLU and NLG.

Abstract Meaning Representation The MRL we adopt is Abstract Meaning Representation (AMR; Banarescu et al. 2013). AMR has gained popularity due to its easy-to-read annotation scheme at the sentence level and the success of its shared tasks (May, 2016; May and Priyadarshi, 2017). In the AMR literature, the NLU task is called AMR parsing, where a sentence has to be converted into its corresponding AMR. The NLG task is called AMR-to-text generation, where an AMR has to be converted into the sentence it represents. While AMR is biased towards English, AMR datasets have been created also for other languages (Li et al., 2016; Xue et al., 2014; Anchiêta and Pardo, 2018).

Reentrancies AMR annotations are rooted and directed acyclic graphs where nodes represent concepts and edges represent semantic relations between them. They are graphs and not trees because nodes can have multiple incoming edges, called *reentrancies*. Reentrancies can be caused by various linguistic phenomena. For instance, the AMR graph in Figure 1.2 contains two reentrancies: one is caused by a coreference and one is caused by a control verb. Coreference occurs when multiple words in the sentence refer to the same entity. The two words *I* and *me* refer to the same entity, causing the node *I* in the AMR to have two incoming edges (parents). Control structures such as *beg you to excuse* cause reentrancies because two predicates share an argument. In this case, *you* is an object of *begging* and a subject of *excusing*.

Graph algorithms have higher computational complexity and are less understood than tree algorithms (Gilroy, 2019). As a consequence, reentrancies make parsing and generation more challenging. Previous work removed reen-

trancies to reduce AMR graphs to sequences (Konstas et al., 2017) or trees (Liu et al., 2015; Takase et al., 2016). Others maintained them but did not analyze their impact on performance (e.g., Song et al., 2018; Beck et al., 2018).

Thesis Statement This thesis studies machine learning models to perform AMR parsing and AMR-to-text generation, with a focus on reentrancies. We claim that our ability to parse reentrancies and to generate from them is of paramount importance to improve performance and implement high-quality systems. Furthermore, we propose AMR parsers for languages other than English and discuss the extent to which is it possible to transfer AMR datasets across languages.

1.1 Thesis Contributions

This thesis contributes to several aspects of AMR: parsing, evaluation, analysis of reentrancies, cross-linguality, and generation.

1.1.1 AMR Parsing

We approached AMR parsing by noting its similarities with dependency parsing. Dependency parsing is a well-studied form of syntactic parsing where dependency edges are created between words in the input sentence. Greedy transition-based methods (Nivre, 2008) are one of the most popular choices for dependency parsing, because of their balance between efficiency and accuracy.

We introduced AMREAGER, a transition-based parser for AMR inspired by the ARCEAGER dependency transition system (Nivre, 2004). AMREAGER is a linear-time AMR parser that can recover non-projective and reentrant nodes caused by control structures. We observed that the overall parsing score was not affected by a transition specifically designed for capturing reentrancies. This unexpected discovery motivated a more careful analysis of how AMR parsers are evaluated.

1.1.2 Evaluation of AMR Parsers

The traditional way of evaluating AMR parsers is through a metric called Smatch (Cai and Knight, 2013). AMR parsing involves a large number of sub-

tasks and linguistic phenomena, but Smatch provides only a single score summarizing the overall quality of the parse. In order to allow for a more detailed analysis of AMR parsers, we introduced a suite of fine-grained evaluation metrics. The metrics assess the performance of AMR parsers with respect to several subtasks, one of which is reentrancy prediction. We found that current parsers cannot accurately parse reentrancies, warranting a closer inspection of the role of reentrancies and their impact on parsing performance.

1.1.3 Analysis of Reentrant Structures

While reentrancies are central to AMR, a detailed analysis of their role is yet not available. To address this, we provided a classification of linguistic causes of reentrancy and quantified their prevalence in the corpus. We then took a closer look at how well state-of-the-art AMR parsers deal with reentrancies by analyzing their errors. Finally, we demonstrated that correcting reentrancy-related errors leads to significant improvements in parsing performance.

1.1.4 Cross-linguality

One of the potential applications for the NLP pipeline of Figure 1.1 is MT. In order to translate between two languages, we need to parse text in one language and generate text in the other language. However, AMR is heavily based on English and AMR datasets exist only for a handful of languages. Moreover, the only available AMR dataset large enough to train state-of-the-art machine learning models is for English.[2] Annotating new AMR datasets for other languages is an expensive process and requires defining guidelines for each new language.

We address the lack of training data for other languages by asking whether it is possible to share the same AMR annotation across languages. To answer this question, we trained AMR parsers that take input sentences in other languages and produce English AMR graphs. We analyzed the parsers and showed that structural differences between languages can be often overcome. Our results suggest that this approach can be a viable way to implement AMR tools for other languages when it is not possible to build dedicated datasets.

[2]The largest AMR dataset for non-English is the Chinese dataset, which only contains 1562 sentences. The AMR dataset for English contains 39260 sentences.

1.1.5 AMR-to-text Generation

The NLG task, called AMR-to-text generation or AMR generation, is the opposite problem of AMR parsing: given an AMR graph, we wish to generate a possible realization of the sentence. An important challenge for this task is that there are multiple ways to express the meaning of a given AMR graph. Moreover, because AMR abstracts away from syntax, it also lacks the information required to reproduce the reference realization precisely. One such example is tense information. For instance, the annotation of Figure 1.2 for the sentence *I beg you to excuse me* would not change for the sentence *I begged you to excuse me*.

In previous work, Konstas et al. (2017) linearized AMR graphs to sequences in order to use sequence-to-sequence architectures (Bahdanau et al., 2015). The linearization process loses reentrancy information. Graph encoders, which do not discard reentrancies, were later shown to yield better results (Song et al., 2018; Beck et al., 2018).

When the AMR annotations do not contain reentrancies, they can be encoded as trees rather than graphs. A comparison between tree and graph encoders can therefore shed lights on the impact of reentrancies on AMR-to-text generation. We showed that graph encoders outperform tree encoders, highlighting the importance of reentrancies for the task. Our tree and graph models are based on a novel combination of sequential and structural encoding, outperforming previous work.

1.2 Thesis Structure

The thesis is structured as follows:

- **Chapter 2.** We review previous work on annotation scheme, parsing, generation, and downstream applications.

- **Chapter 3.** We investigate the similarities between AMR parsing and dependency parsing by developing a transition system, inspired by dependency tree parsing, with transitions aimed at recovering reentrant structures caused by control verbs. This chapter is based on Damonte et al. (2017).

- **Chapter 4.** We introduce a fine-grained evaluation suite for AMR parsing. Inspired by the poor performance of state-of-the-art parsers at recovering reentrancies, we discuss what phenomena that cause them and quantify their prevalence in the AMR corpus. We then analyze the impact of reentrancy-related errors on parsing performance. This chapter is based on Damonte et al. (2017) and Damonte et al. (2019).

- **Chapter 5.** We extend the parser of Chapter 3 to Italian, Spanish, German and Chinese via cross-lingual techniques. We provide evidence that AMR annotations, up to a certain extent, can be successfully shared across languages. We also present a novel evaluation procedure for cross-lingual settings. This chapter is based on Damonte and Cohen (2018).

- **Chapter 6.** We finally turn to the AMR-to-text generation problem and compare neural architectures based on how they deal with reentrancies. We show that graph encoders, which account for reentrancies, outperform tree and sequential encoders, which do not. This chapter is based on Damonte and Cohen (2019).

- **Chapter 7.** We summarize and discuss future work. We highlight the findings and contribution with respect to both parsing and generation, with particular attention to reentrancies.

2

Abstract Meaning Representation

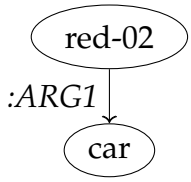
Advancements in syntactic parsing have been greatly favored by the development of a public corpus of sentences annotated with syntactic trees: the Penn Treebank (Marcus et al., 1993). The motivation behind the creation of an AMR dataset (Banarescu et al., 2013) is to replicate this success story for semantic parsing by creating a single dataset covering a wide range of semantic problems. AMR includes semantic tasks that were previously studied individually such as Coreference Resolution (Hobbs, 1979), Named Entity Recognition (NER; Nadeau and Sekine 2007), Word Sense Disambiguation (WSD; Navigli 2009) and Semantic Role Labeling (SRL; Palmer et al. 2010). AMR is biased towards English as the annotation guidelines only consider the English language and many AMR node labels are English words, as discussed in Section 2.1.[1] AMR does not include alignments between the semantics and the words in the sentence. The AMR dataset consists of a corpus of sentences annotated with AMR representations. The publication of this dataset gave rise to the introduction of NLU and NLG tasks for AMR, known as AMR parsing and AMR-to-text generation, respectively. The goal of AMR parsing is to automatically convert a sentence into its AMR representation. AMR-to-text generation is the opposite task: to generate a sentence from its AMR.

In the remainder of this chapter, we first review the most salient properties of AMR and the semantic tasks included in the AMR annotation scheme. We then discuss the relevant literature.

[1] https://github.com/amrisi/amr-guidelines/blob/master/amr.md

2.1 AMR Annotations

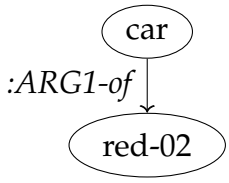
Sentences are annotated into AMR using the PENMAN notation (Mann, 1983), following annotation guidelines mentioned above. AMR annotations can be represented as rooted Directed Acyclic Graphs (DAGs). Nodes in the graphs represent core concepts in the sentence. They can either be words (typically adjectives or stemmed nouns and adverbs) or frames extracted from Propbank (Kingsbury and Palmer, 2002).[2] For example, the AMR for the sentence *The car is red* contains the word *car* and the frame *red-02*:



(**w** / *red-02*
  :ARG1 (**m** / *car*))

We represent the AMR annotations both graphically (on the left) and in the PENMAN format used by the human annotators (on the right). Each concept is identified by a variable in the PENMAN annotations, highlighted in bold.

Labeled edges between a parent node and a child node indicate a semantic relationship between them. Edges can be inverted, through the use of the *-of* suffix. For instance, the edge in the previous AMR can be inverted, resulting in the following AMR:



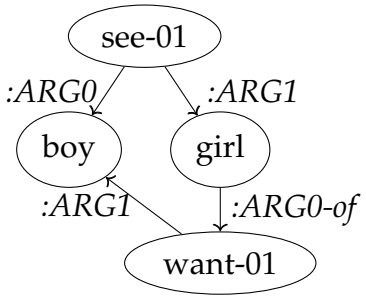
(**m** / *car*
  :ARG1-of (**w** / *red-02*))

While the direction of the edge does not impact the meaning expressed by the AMR, the two previous AMR graphs are not equivalent because their roots are different. The root of an AMR, i.e., the only node with no incoming edges,

[2] https://amr.isi.edu/doc/propbank-amr-frames-arg-descr.txt

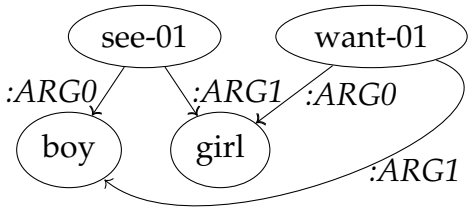
identifies the focus of the sentence, and it is therefore chosen accordingly by the annotators. In the former example, the focus of the sentence is *red*, while in the latter the focus becomes the *car*, for instance for the phrase *The red car*.

Inverted edges are sometimes used to maintain a single root, as in the AMR for the phrase *The boy saw the girl who wanted him*:

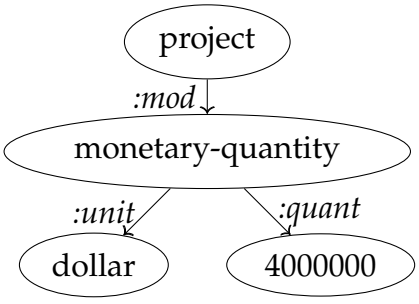


(**s** / *see-01*
  :ARG0 (**b** / *boy*)
  :ARG1 (**g** / *girl*
    :ARG0-of (**w** / *want-01*
      :ARG1 **b**)))

In fact, if we inverted the *ARG0-of* edge, the resulting graph would have two roots:



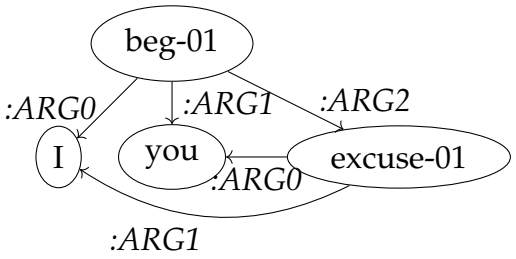
Edges are classified in *core* and *non-core* roles. Core roles have a *ARG-x* prefix. They specify semantic roles between AMR concepts, as further discussed in Section 2.1.3. The following AMR for the sentence *The 4 million-dollar project* contains only non-core roles:



(**p** / *project*
  :mod (**m** / *monetary-quantity*
    :unit (**d** / *dollar*)
    :quant *4000000*))

In the previous AMR graph, *4000000* is not a variable but a constant literal. Constant literals are used in AMR to define names and numbers.

An important property of AMR is the presence of nodes with multiple parents, known as *reentrancies*. Reentrancies are specified in the PENMAN format by the use of co-indexed variables. For instance, the AMR graph for the sentence *I beg you to excuse me* contains two reentrancies:



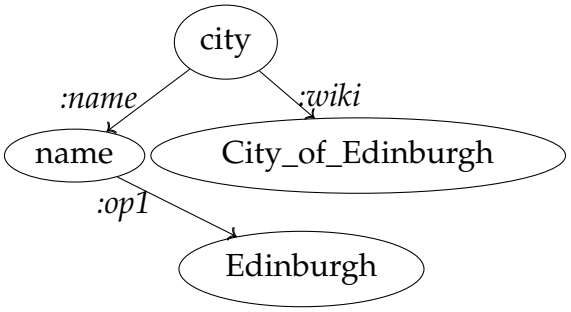

```
(b / beg-01
   :ARG0 (i / I)
   :ARG1 (y / you)
   :ARG2 (e / excuse-01)
         :ARG0 y
         :ARG1 i)
```

The two words *I* and *me* refer to the same entity, causing the reentrancy for the node *I*. Because of the control verb *beg*, the word *you* is argument of both *beg* and *excuse*, causing another reentrancy.

We now review the semantic tasks enclosed in the AMR annotations.

2.1.1 Named Entity Recognition

NER is the task of classifying named entity mentions into coarse categories such as location, person, and organization. See Nadeau and Sekine (2007) for a survey of NER. In AMR, named entities are annotated through specific concepts and roles. For instance, *Edinburgh* is annotated as follows, where *Edinburgh* and *City_of_Edinburgh* are constant literals:




```
(c / city
   :wiki "City_of_Edinburgh"
   :name (n / name
      :op1 Edinburgh))
```

The annotation also includes a *:wiki* role, which identifies a canonical name for the named entity, corresponding to its Wikipedia page (or -, if the named entity has no Wikipedia page).

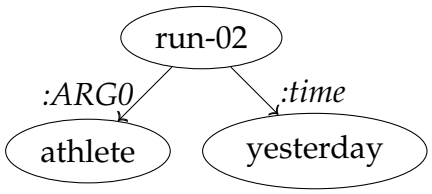
2.1.2 Word Sense Disambiguation

The goal of WSD is to automatically disambiguate between the meaning of words in context (Navigli, 2009). Consider for example the two following sentences, where the word *ran* has different meanings:

(1) The athlete ran yesterday.

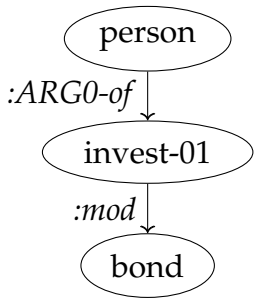
(2) They ran the company.

AMR uses Propbank frames to determine the specific sense of predicates. For instance, the AMR for the sentence *The athlete ran yesterday* uses the frame *run-02*, which means *to walk quickly* and not *run-01*, which means *to operate*:




```
(r / run-02
   :ARG0 (a / athlete)
   :time (y / yesterday))
```

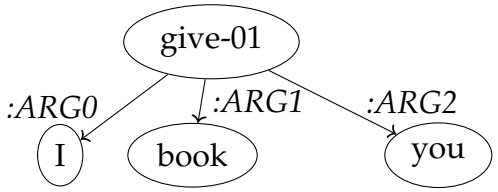
Frames are mainly used for verbs but can be also used for other part of speech categories. For instance, the AMR for the noun phrase *bond investor* uses the *invest-01* frame (an investor is a person who invests):




```
(p / person
   :ARG0-of (i / invest-01
      :ARG2 (b / bond)))
```

2.1.3 Semantic Role Labeling

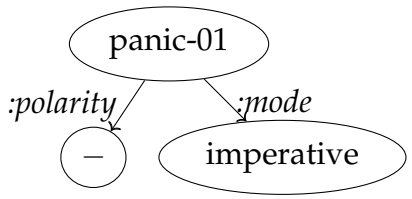
In SRL we look for relations and roles between words in a sentence. See Palmer et al. (2010) for a survey. In AMR, Propbank frames are also used to extract SRL information. For instance, the frame *give-01* (*to transfer*) specifies three roles: the *giver*, the *thing given*, and the *entity given to*. These are represented in the following AMR for the sentence *I gave you a book* by the *:ARG0*, *:ARG1*, and *:ARG2* arguments:



(**g** / *give-01*
:ARG0 (**i** / *I*)
:ARG1 (**b** / *book*)
:ARG2 (**y** / *you*))

2.1.4 Negation Detection

In negation detection, the aim is to identify negation cues and scope (Morante and Blanco, 2012). AMR does not mark scope or cues information but it annotates negation with the *:polarity* role, as shown in the following graph for the sentence *Don't panic*:



(**p** / *panic-01*
:polarity −
:mode imperative)

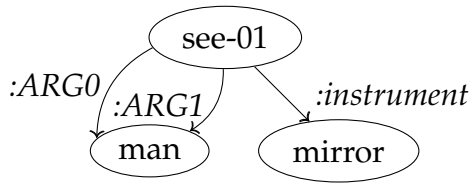
2.1.5 Coreference Resolution

Coreference is a source of reentrancies in AMR. To predict these structures accurately, parsers need to perform Coreference Resolution. Coreference has been defined as a relation holding between noun phrases that refer to the same entity (Hirschman et al., 1997). While by this definition an anaphora is not a coreference (Van Deemter and Kibble, 1999), in the remainder of the thesis we also refer to anaphoric relations as coreference relations. While coreference is a discourse phenomenon (Hobbs, 1979), pronomial anaphora is often sentence-level, such as for *The man saw himself in the mirror*:

2.2 AMR Datasets

Datasets of English sentences annotated with AMR graphs are periodically released through the Linguistic Data Consortium (LDC).[3] The two most commonly adopted datasets are the LDC2015E86, containing 19,572 sentences, and

[3]https://www.ldc.upenn.edu



(**s** / *see-01*
:ARG0 (**m** / *man*)
:ARG1 m
:ARG1 (**m2** / *mirror*))

LDC2017T10, containing 39,260 sentences. The datasets include sentences from newswire and web data (Banarescu et al., 2013).

2.3 AMR Parsing

AMR parsing is the task of converting natural language into AMR graphs. The first parser, called JAMR, was introduced by Flanigan et al. (2014). JAMR first identifies the nodes of the graph (concept identification), framing the problem as sequence labeling. It then approaches the prediction of edges between the nodes as a constrained combinatorial optimization problem. Werling et al. (2015) noticed that concept identification is the most challenging part of the process. They proposed an action classifier to generate concepts by applying predetermined actions.

Various other strategies have been used for AMR parsing. Peng et al. (2015) used a Synchronous Hyperedge Replacement Grammar (Habel, 1992). Pust et al. (2015) presented a syntax-based Machine Translation (MT) parser where a rule extraction step creates a grammar of string-to-tree rules. In order to use tree-based grammars, graphs are converted into trees by removing all reentrancies. The parser by Vanderwende et al. (2015) used a pre-existing logical form parser and a set of rules to transform the output of the parser to AMR graphs. The logical formalism is similar to AMR so that for most relations it is enough to perform simple label renaming steps. Because they do not rely on the AMR annotated data, they can generate AMR for languages for which an AMR dataset is not available yet. Artzi et al. (2015) proposed to parse AMR graphs by first parsing a lambda-calculus representation with a Combinatory Categorial Grammar (CCG; Steedman 1996, 2000) using placeholders to mark non-compositional aspects that are then resolved by a factor graph model.

We now focus on two of the most prevalent strategies for AMR parsing: transition-based parsing and neural parsing.

2.3.1 Transition-based Parsing

Transition-based parsing is a popular approach to dependency parsing. In this section, we first introduce transition-based parsing in the context of dependency parsing. We then discuss attempts to apply it to AMR parsing.

Transition Systems for Dependency Parsing

Dependency parsing is based on the idea that the syntactic structure of a sentence is given by binary relationships (dependencies) between the words in the sentence. Dependencies are labeled and directed edges from one word in the sentence (the *head*) to another word in the sentence (the *dependent*). A dependency tree for a sentence $w_0, \ldots, w_n$ is defined by its vertices (or nodes) V and labeled directed edges E:

$$T = (V, E, L),$$
$$V = \{0, 1, \ldots, n\},$$
$$E \subseteq V \times L \times V,$$

where V is the set of indexes corresponding to the position of a word in the sentence. Each edge in E is a triple (i, l, j), where $i \in V$ is the head, $\ell \in L$ is the label, and $j \in V$ is the dependent. The dependency tree for the sentence *I beg you to excuse me* is shown in Figure 2.1. By comparing the dependency tree with the AMR of Figure 2.2, it is possible to observe similarities between dependency trees and AMR graphs. For instance, in both structures there are edges connecting *beg* (*beg-01*) with *I*, *you*, and *excuse* (*excuse-01*). AMR graphs follow dependency trees in defining binary relationships between items in the sentence. The similarities between the two structures motivate the use of dependency parsing techniques also for AMR parsing.

Transition-based parsing is a general approach to parsing where an input sentence is fed into a transition system, which then outputs a parse tree or graph. A transition system is an abstract machine characterized by a set of states and actions between them. Starting from an initial state, the system applies actions until a terminal state, containing the final parse, is reached.

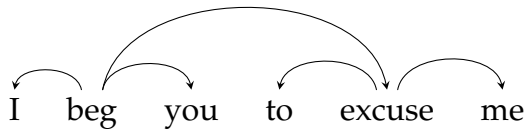


Figure 2.1: Dependency tree for the sentence *I beg you to excuse me*.

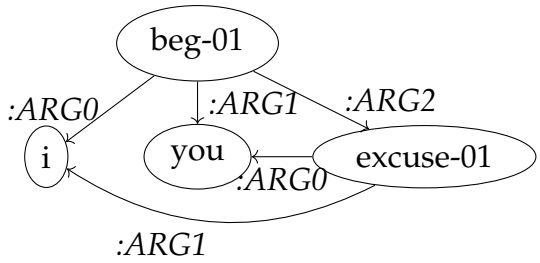


Figure 2.2: AMR graph for the sentence *I beg you to excuse me*.

In dependency parsing, a transition system is usually defined as a quadruple: $T = (S, A, I, E)$, where S is a set of states, A is a set of actions, I is the initial state, and E is a set of end states. A state is composed of a buffer, a stack, and a set of arcs: $S = (\beta, \sigma, A)$. In the initial state, the buffer contains all the words in the input sentence, the stack contains a special root node ($\circ$) and the set of subtrees are empty: $S_0 = (w_0|\ldots|w_N, [\circ], \emptyset)$. Terminal states have empty buffer and only the root symbol in the stack: $S_T = (\emptyset, [\circ], A)$. The buffer is used to store the input sentence, which is usually consumed left-to-right. The stack, initially empty, is used to store words that have been consumed from the buffer but have not been fully processed yet.

A key advantage of transition-based parsing is that, when greedy decoding is used, it allows linear-time parsing. The two most common transition systems for greedy dependency parsing are ARCSTANDARD and ARCEAGER (Nivre, 2004). In ARCSTANDARD, arcs are created among the two top-most elements in the stack, and the dependent is always removed from the stack. It parses sentences in a bottom-up fashion, limiting the parsers' incrementality (left-to-right). ARCEAGER, on the other hand, was designed to support incrementality by mixing bottom-up and top-down approach.

Classifiers, learned from data in a supervised setting, are used to determine which action to apply given the current state of the transition system. Titov and Henderson (2007) used a latent variable model based on Incremental Sigmoid Belief Networks (ISBN), a form of Sigmoid Belief Networks (SBN;

Neal 1992). SBNs are latent variable networks related to feed-forward neural networks. Early attempts also used the perceptron algorithm to train the action classifiers (Zhang and Clark, 2008). Later, Chen and Manning (2014) used feed-forward neural networks. Recurrent neural networks (RNNs) such as Long Short-Term Memory Networks (LSTMs; Hochreiter and Schmidhuber 1997) have gained popularity in NLP, due to their ability to encode sequences of variable lengths. To use RNNs to encode the state of a transition system's stack, Dyer et al. (2015) introduced a variant of LSTM that allows for push and pop operations, called Stack-LSTM.

Transition Systems for AMR Parsing

Due to the similarities between AMR parsing and dependency parsing, transition systems have also become popular for AMR parsing. CAMR (Wang et al., 2015b) used a transition system to convert a dependency tree, predicted by a dependency parser, into the desired AMR graph. The main advantage of CAMR is that the dependency parser can be trained on a much larger training set than the one available for AMR. Wang et al. (2015a) later showed that specific handling of *abstract concepts* (i.e., AMR nodes not syntactically related to any word in the sentence) results in further improvements.

Sawai et al. (2015) used a variant of the ARCSTANDARD transition system to address a simpler task, where single noun phrases (NPs), instead of full sentences, are parsed.

Rao et al. (2016) adopted SEARN (Daumé III et al., 2009), a learning-to-search algorithm akin to transition-based parsing. SEARN solves structured prediction problems by decomposing it into classification problems: the AMR parsing problem is decomposed in concept identification, root identification, and relation predictions. For each subproblem, SEARN learns a policy to find what is the right action in a given state.

Traditional transition-based parsers were devised for tree-structured output. To parse graphs, transition systems with ad-hoc actions for reentrancies and non-projective structures have also been proposed. We discuss our solution to this problem in Chapter 3, where additional edges are created between siblings, hence allowing reentrancies. Other transition systems that include mechanisms to include reentrancies have been proposed (Wang et al., 2015b; Ballesteros and Al-Onaizan, 2017; Peng et al., 2018).

2.3.2 Neural-based Parsing

Parsing sentences into AMR graphs can be seen as a translation task, where English is the source language, and AMR is the target language. Neural Machine Translation (NMT; Bahdanau et al. 2015) is an approach to translation that is proving very successful (Wu et al., 2016; Vaswani et al., 2017; Barone et al., 2017). NMT-based parsing has been explored for AMR parsing. We first review the basics of NMT and then explore its application to AMR parsing.

Neural Machine Translation

In the NMT approach, a sentence in the source language is usually fed, one word at the time, into an LSTM network, called an *encoder*. For each input word, the LSTM updates its state, representing the sentence up to that word. Unlike standard RNNs, LSTM networks use gates to maintain only the important information and handle long-range dependencies, while avoiding the vanishing and exploding gradient problems (Bengio et al., 1994). In LSTMs, words in a sentence are fed into the network from left to right, so that the context of each word is given only by the words on its left. Bidirectional LSTM (BiLSTM; Graves et al. 2013) networks can be used to take into account also the words on their right, by combining the left-to-right reading with a right-to-left reading. Recent work showed that non-recurrent layers based on self-attention can also be used as encoders (Vaswani et al., 2017).

The encoder's output is then used to initialize another LSTM, called a *decoder*. The decoder network is used to predict an output word to generate and subsequently update its state. Each step of the decoder generates one word from the output sentence, left-to-right, until an end-of-sentence token is produced, signaling the end of parsing.

The production of a target word often does not depend on the entire input sentence but only on a small portion of it. To account for this, an attention mechanism is often used (Bahdanau et al., 2015). Its aim is to learn which words carry more information for each word prediction. This is achieved by learning a context vector that specifies how much each encoder step affects the current decoder step.

Instead of processing a word at the time, it is possible to use subword items, such as characters (Chung et al., 2016; Lee et al., 2017). This alleviates the

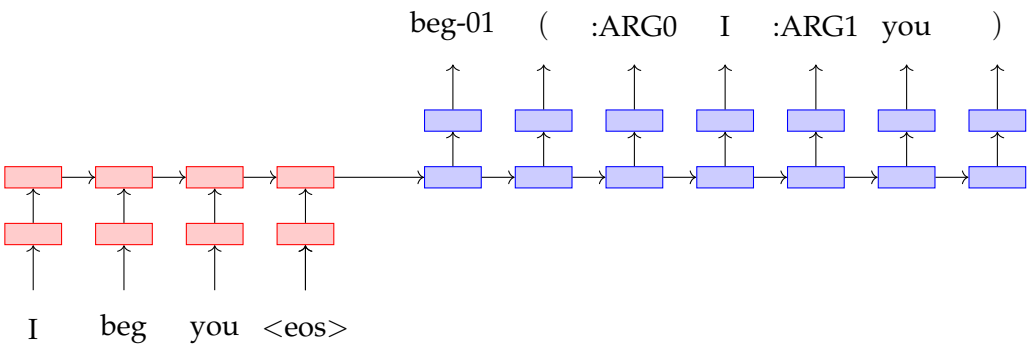


Figure 2.3: Sketch of a sequence-to-sequence model to parse the English sentence *I beg you* into the linearized AMR. The sentence is encoded one word at the time and the AMR is decoded one token at the time, where each token can be either a node label, an edge label or a bracket.

problem of out-of-vocabulary words, as all characters are seen in training.

Neural Machine Translation for AMR Parsing

In order to use vanilla NMT models, like the one discussed above, the AMR graphs must be converted into sequences, as shown in Figure 2.3. The process, called linearization, loses structural information such as all reentrancies. The alleviate the problem (van Noord and Bos, 2017a) discusses pre- and post-processing steps to better deal with reentrancies.

Barzdins and Gosko (2016) carried out experiments with a character-level NMT architecture for AMR parsing. Konstas et al. (2017) achieved competitive results with a word-level architecture. To deal with data sparsity, sentences and AMR were preprocessed by replacing names and rare words with coarse categories — a process called anonymization. Van Noord and Bos (2017b) later introduced a character-level model that outperforms the word-level models of Konstas et al. (2017). To outperform non-neural parsers, Konstas et al. (2017) and van Noord and Bos (2017b) use additional data, obtained by automatically parsing extra unlabeled sentences.

Significantly better results were later obtained by a neural model based on a joint model of concepts, relations, and alignments (Lyu and Titov, 2018). The current state of the art was achieved by an NMT-based parser who implements a target-side copy mechanism (See et al., 2017) aimed at recovering reentrancies (Zhang et al., 2019).

2.3.3 Evaluation

AMR parsers are often evaluated using a semantic graph matching algorithm, called Smatch (Cai and Knight, 2013). It determines how close a predicted AMR is to a reference AMR. AMR annotations are viewed as a conjunction of triples, as shown in Figure 2.4. Each triple represents either an edge between two variables, or the mapping between a variable and its AMR concept. Annotators use arbitrary variable names to identify concepts. The predicted and reference graphs will therefore have different variable names. Therefore, Smatch needs to predict the correct alignments between the variables in the two graphs. For instance, in the example of Figure 2.4, **w** is to be aligned to **v1**, **i** to **v2**, **b** to **v3**, and **y** to **v4**. Different methods can be used to predict the alignments, such as Integer Linear Programming and hill-climbing (Cai and Knight, 2013). Once the alignments are predicted, Smatch computes precision, recall, and F1 of the triples. In the example of Figure 2.4, the precision is 1.0, the recall is 0.89, and the F1 is 0.94.

The Smatch score consists of a single number that does not assess the quality of each semantic subtask separately. To address the issue, in Chapter 4 we propose a suite of fine-grained evaluation metrics to assess the performance on subtasks such as unlabeled parsing, NER, SRL, and reentrancy prediction. Recently, an alternative evaluation method based on the popular BLEU score (Papineni et al., 2002) has been proposed by Song and Gildea (2019), where it is found to be faster and correlate better to human judgment than Smatch.

2.4 AMR-to-text Generation

AMR-to-text generation is the task of generating natural language from AMR graphs. The same AMR graph can be used to represent the meaning of several sentences. Consider the following sentences:

1. I beg you to excuse me.
2. I am begging you to excuse me.
3. Excuse me, I beg you.

The first and second sentences only differ in the tense of the verb *to beg*. Tense information is considered as a morphological category which can be retrieved

```
(w / want-01
    :ARG0 (i / I)
    :ARG1 (b / believe-01
        :ARG1 i
        :ARG0 (y / you)))
```

```
(v1 / want-01
    :ARG0 (v2 / I)
    :ARG1 (v3 / believe-01
        :ARG0 (v4 / you)))
```

```
root(w) ∧
instance(w, want−01) ∧
instance(i, I) ∧
instance(b, believe−01) ∧
instance(y, you) ∧
ARG0(w, i) ∧
ARG1(w, b) ∧
ARG1(b, i) ∧
ARG0(b, y)
```

```
root(v1) ∧
instance(v1, want−01) ∧
instance(v2, I) ∧
instance(v3, believe−01) ∧
instance(v4, you) ∧
ARG0(v1, v2) ∧
ARG1(v1, v3) ∧
ARG0(v3, v4)
```

Figure 2.4: On the top left, the gold standard annotation for the sentence *I want to believe you*. On the top right, the predicted annotation. On the bottom, the respective triples.

from the sentence itself. Hence, annotators are not required to annotate it. There can be other syntactic differences that are not annotated in the AMR such as the use of synonyms or specific function words. The task is therefore to generate one of the possible syntactic realizations of the sentence.

The first AMR-to-text generation system was introduced by Flanigan et al. (2016b), where graphs are converted to trees and fed into a tree-to-string transducer to produce the output sentence. The conversion from graphs to trees is necessary as DAG-to-string transducers are not currently available. However, this process removes all reentrancies, which are an essential characteristic of AMR graphs. The system by Song et al. (2016) converts single AMR fragments and decides their order by solving a traveling salesman problem. Lampouras and Vlachos (2017) introduced a transition-based approach. The system of

Gruzitis et al. (2017) first converts AMR graphs to Grammatical Framework syntax trees (Ranta, 2004). It then uses already available surface realization English grammars to produce the output text.

A popular approach to AMR-to-text generation is to frame it as a translation task. For a survey on MT-based solutions, see Ferreira et al. (2017). Pourdamghani and Knight (2016) developed a phrase-based MT system to convert AMR into sentences. AMR graphs are linearized into a sequence with English-like order of the AMR nodes, losing structural information such as reentrancies. NMT-based approaches, discussed in Section 2.3.2 for parsing, have also been used for AMR-to-text generation. As for parsing, Konstas et al. (2017) reduced the AMR graphs to sequences in order to use sequence-to-sequence models. Song et al. (2018) and Beck et al. (2018) proposed ways to encode AMR as graphs, instead of sequences, resulting in better performance. This allows to explicitly encode structural information such as the reentrancies. In Chapter 6 we investigate their impact on performance by directly comparing sequential, tree and graph encoders.

More recently, Guo et al. (2019) achieved state-of-the-art results with a deeper graph encoder. Cao and Clark (2019) recently proposed a different approach to the task by first predicting the syntactic structure, from which the surface form is then generated.

We mentioned that AMR can be realized in several ways. As a result, evaluation of this task is problematic, as only one reference sentence is given for each AMR graph. AMR-to-text generation systems are usually evaluated with BLEU (Papineni et al., 2002), traditionally used for machine translation tasks. By using BLEU with a single reference, we may penalize generation systems for using a different way to phrase the same meaning.

Meteor (Banerjee and Lavie, 2005) can be used to address this limitation. Meteor is based on matching stems, synonyms, and paraphrasing, hence allowing for different surface realizations. It is a sentence-level metric. On the contrary, BLEU is a corpus-level metric, and it cannot be reliably used to compute the score of a single example. CHRF++ (Popović, 2017) has also been used for evaluating AMR-to-text systems (Beck et al., 2018; Guo et al., 2019). CHRF++ looks for exact matches, similarly to BLEU, but it is a sentence-level metric.

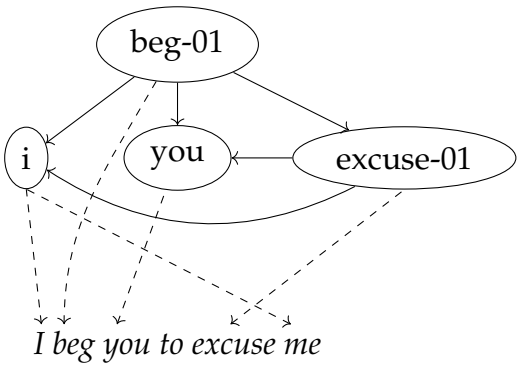


Figure 2.5: AMR alignments for the sentence *I beg you to excuse me*. We omit the edge labels for clarity.

2.5 AMR Alignments

AMR annotations do not include alignments between words in the sentence and nodes in the graph (Figure 2.5). However, parsing and generation algorithms often require them. Flanigan et al. (2014) introduced a rule-based aligner. Pourdamghani et al. (2014) presented a data-driven approach based on unsupervised IBM models (Och and Ney, 2000), traditionally used for word alignments in Machine Transition. Chu and Kurohashi (2016) tackled the problem in a supervised setting as a constituency-based alignment task. The AMR graphs were converted into constituency trees following the method by Pust et al. (2015). The similarities between dependency trees and AMR graphs motivate producing alignments between these structures (Chen, 2015; Chen and Palmer, 2017; Szubert et al., 2018).

A different approach to AMR alignments is to treat them as latent variables during parsing (Lyu and Titov, 2018; Zhang et al., 2019).

2.6 Applications of AMR

There have been early attempts to use AMR graphs for downstream NLP problems. One such example is text summarization, which is the task of generating summaries from one or more documents. Liu et al. (2015) introduced an AMR-based summarizer where the sentences of a document are first parsed into AMR graphs. The graphs are then collapsed in a single summary graph, from which the summary is finally generated. A related task is that of headline

generation, for which an NMT-based model has been proposed (Takase et al., 2016). AMR has also been shown to be beneficial in English-German NMT to encode extra information on the source language (Song et al., 2019).

In the biomedical domain, AMR has been used to improve the performance of bio-molecular interaction extraction (Garg et al., 2016, 2018; Rao et al., 2017). In this task, the goal is to identify biological entities and interactions between them. AMR-based embeddings were also used as features in a classifier for the task of Drug-Drug Interaction Extraction (Wang et al., 2017).

Entity Linking, the task of binding named entities to their knowledge base record, has also been tackled with AMR. Pan et al. (2015) used AMR to disambiguate entity mentions and cluster them into coherent sets.

As previously discussed, semantically related sentences may be represented by the same AMR graph. Thus, when two sentences are paraphrases of each other, they should have equivalent or close AMR representations. Issa et al. (2018) used this intuition to build a paraphrase identification model based on AMR features. It was shown that the features extracted by AMR parsers outperform those extracted by a syntactic parser.

2.7 AMR for Other Languages

Sentences in other languages have been annotated with AMR using language-dependent labels: Chinese (Li et al., 2016; Xue et al., 2014), Czech (Xue et al., 2014) and Brazilian Portuguese (Anchiêta and Pardo, 2018). Bojar (2014) categorized different kinds of divergences in the annotation between English and Czech AMR graphs. Moreover, Xue et al. (2014) showed that structurally aligning English AMR graphs with Czech and Chinese AMR graphs is not always possible but that refined annotation guidelines suffice to resolve some of these cases. The presence of structural differences between AMR graphs for different languages supports the claim of bias towards English. However, structural differences do not always occur and it is worth investigating whether it is possible to deal with them when they do. We attempt to answer this question in Chapter 5, where we train parsers for Italian, Spanish, German and Chinese via cross-lingual techniques. We provide evidence that AMR annotations, up to a certain extent, can be successfully shared across languages.

2.8 Alternatives to AMR

Broad-coverage semantic representation schemes for natural languages similar to AMR have been proposed, such as Universal Conceptual Cognitive Annotation (UCCA; Abend and Rappoport 2013), Discourse Representation Structure (DRS; Bos 2004), and Minimal Recursion Semantics (MRS; Copestake et al. 2005). Each scheme has different underlying formalism: AMR does not have an underlying theoretical formalism but follows a neo-Davidsonian event specification (Davidson, 1969), UCCA follows Basic Linguistic Theory (Dixon, 2010), and DRS is also based on neo-Davidsonian events and follows Discourse Representation Theory (Kamp and Reyle, 1993). AMR, like UCCA, cannot handle scope or tense information. Moreover, similarly to DRS, it abstracts away from syntax. One of the advantages of AMR is that the annotations are easy to read by humans, even though annotators require training. In the Groningen Meaning Bank (Bos et al., 2017), which uses DRS, annotations are generated semi-automatically. Related to our cross-lingual approach to AMR parsing of Chapter 5, the Parallel Meaning Bank introduces shared DRS representations for sentences in English, German, Dutch, and Italian (Abzianidze et al., 2017).

3

Transition-based AMR Parsing

In Chapter 2 we reviewed an approach to parsing called transition-based parsing. The approach allows for left-to-right, linear-time, incremental processing and has been successfully applied to dependency parsing (Nivre, 2004, 2008; Chen and Manning, 2014).[1]

Similarly to dependency parsing, AMR parsing is based on the identification of predicate-argument structures. The similarity of AMR structures to dependency structures suggests that transition systems can be helpful for AMR parsing. AMR parsing differs from dependency parsing in three main aspects. First, in AMR parsing there are no direct alignments between words in the sentence and nodes in the graph. Second, AMR graphs for English are not projective structures. Finally, AMR graphs allow for reentrancies.

In this chapter, we ask whether transition-based parsing can be successfully applied also to AMR. We introduce AMREAGER, a parser for AMR inspired by the ARCEAGER dependency parser (Nivre, 2004). It accounts for the main differences between dependency trees and AMR graphs. AMREAGER brings dependency parsing and AMR parsing closer by showing that dependency parsing algorithms can be adapted to AMR parsing. Key properties such as left-to-right processing, incrementality, and linear complexity further strengthen the relevance of our parser.

Our contributions in this chapter are as follows:

- We develop a left-to-right, linear-time transition system for AMR parsing, inspired by transition systems for dependency tree parsing;

[1]Strictly speaking, transition-based parsing does not always achieve full incrementality, which requires to have a single connected component at all times (Nivre, 2004).

- We evaluate our parser with Smatch and compare it with previous work. We show that AMREAGER achieves competitive parsing scores;
- We run ablation results on a transition aimed at recovering reentrancies and discover that the Smatch score remains surprisingly unaffected.

3.1 Notation

We define an AMR structure as a tuple (G, x, π), where $x = x_1 \cdots x_n$ is a sentence, with each x_i, $i \in \{1, \ldots, n\}$, a word token, and G is a directed graph $G = (V, E)$ with V and E the set of nodes and edges, respectively. We assume G comes along with a node labeling function and an edge labeling function. Finally, π: $V \rightarrow \{1, \ldots, n\}$ is a total alignment function that maps every node of the graph to an index i for the sentence x, with the meaning that node v represents (part of) the concept expressed by the word $x_{\pi(v)}$.[2]

We note that the function π is not invertible, since it is neither injective nor surjective. For each $i \in \{1, \ldots, n\}$, we let

$$\pi^{-1}(i) = \{v \mid v \in V,\ \pi(v) = i\}$$

be the pre-image of i under π (this set can be empty for some i), which means that we map a token in the sentence to a set of nodes in the AMR. In this way we can align each index i for x to the induced subgraph of G. More formally, we define

$$\overleftarrow{\pi}(i) = (\pi^{-1}(i), E \cap (\pi^{-1}(i) \times \pi^{-1}(i))), \tag{3.1}$$

with the node and edge labeling functions of $\overleftarrow{\pi}(i)$ inherited from G. Hence, $\overleftarrow{\pi}(i)$ returns the AMR subgraph aligned with a particular token in the sentence.

3.2 Alignments

In AMR there is no direct mapping between a word in the sentence and a node in the graph: words may generate no nodes, one node or multiple nodes. In addition, the node labels are often not easily determined by the words in the

[2] π is a function because we do not consider coreference, which would otherwise cause a node to map to multiple indices.

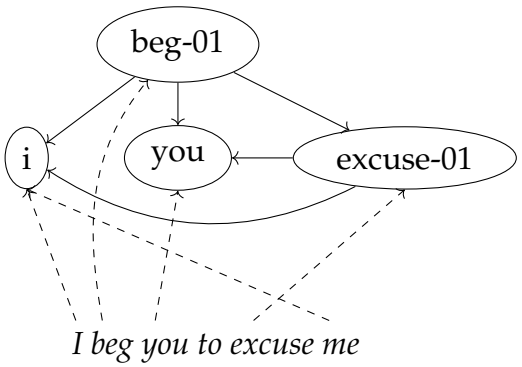


Figure 3.1: Alignments between the AMR graph and the sentence *I beg you to excuse me*. The edge labels were omitted for the sake of clarity.

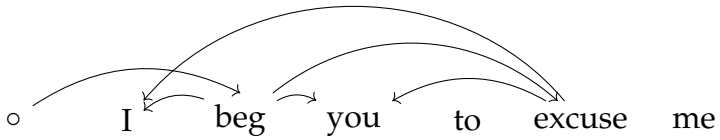


Figure 3.2: Edges of the AMR in Figure 3.1 mapped back to the sentence, according to the alignment. ∘ is a special token representing the root.

sentence. For instance, the word *teacher* translates to the two nodes *teach-01* and *person*, connected through an *:ARG0* edge, expressing that a teacher is a person who teaches. Figure 3.1 shows the alignments between the sentence *I beg you to excuse me* and its AMR.

We define AMR alignments between a token x_i and a subgraph in AMR as $\overleftarrow{\pi}(i)$, defined in Equation (3.1).

3.3 Non-Projectivity

Dependency trees in English are usually projective, roughly meaning that when drawing the edges in the semi-plane above the words, none are crossing. More formally, we first define the *reflexive transitive closure of the dependency*, which we denote as $w_i \rightarrow^* w_j$: $w_i \rightarrow^* w_j$ if and only if $i = j$ or both the following hold for some $w_{i'} \in V$: a) $w_i \rightarrow^* w_{i'}$ and b) $(w_{i'}, l, w_j) \in E$ for some label l. A dependency edge $(w_i, l, w_j) \in E$ can be then said to be projective when:

$$w_i \rightarrow^* w_k, \text{ for all } \begin{cases} i < k < j, & \text{if } i < j \\ j < k < i, & \text{if } j < i. \end{cases}$$

Non-projective edges	6%
AMR graphs with at least one non-projective edge	51%
Reentrant edges	41%
AMR graphs with at least one reentrancy	93%

Table 3.1: Statistics for non-projectivity and reentrancies in 200 AMR manually aligned with the associated sentences.[3] A reentrant edge is an edge pointing to a node that has other incoming edges.

We now generalize the notation of projectivity to AMR graphs. The intuition is that we can use the alignment π to map AMR edges back to the sentence x, and test whether there exist pairs of crossing edges. Figure 3.2 shows this mapping for the AMR of *I beg you to excuse me*, where the edge connecting *excuse* to *I* crosses another edge.

More formally, consider an AMR edge $e = (u, \ell, v)$. Let $\pi(u) = i$ and $\pi(v) = j$, so that u is aligned with x_i and v is aligned with x_j. The spanning set for e, written $\mathcal{S}(e)$, is the set of all nodes w such that:

$$\pi(w) = k, \begin{cases} i < k < j, & \text{if } i < j \\ j < k < i, & \text{if } j < i. \end{cases}$$

We say that e is projective if, for every node $w \in \mathcal{S}(e)$, all of its parent and child nodes are in $\mathcal{S}(e) \cup \{u, v\}$; otherwise, we say that e is non-projective. An AMR is projective if all of its edges are projective, and is non-projective otherwise. This corresponds to the intuitive definition of projectivity for DAGs introduced by Sagae and Tsujii (2008) and is closely related to the definition of non-crossing graphs by Kuhlmann and Jonsson (2015).

While non-projectivity is not frequent in syntactic theories for English Kubler et al. (2009), it is for AMR structures. Table 3.1 demonstrates that a relatively small percentage of all AMR edges are non-projective. Yet, a large fraction of the sentences contain at least one non-projective edge. Our parser can construct non-projective edges, as described in Section 3.5.

[3] `https://github.com/jflanigan/jamr/blob/master/docs/Hand_Alignments.md`

Shift	$(\sigma, \beta_0 \vert \beta, A) \rightarrow (\sigma \vert \text{root}(a(\beta_0)), \beta, A \cup E_a)$
	where $a(\beta_0) = (V_a, E_a)$
LArc(ℓ)	$(\sigma \vert \sigma_1 \vert \sigma_0, \beta, A) \rightarrow (\sigma \vert \sigma_0, \beta, A \cup \{\langle \sigma_0, \ell, \sigma_1 \rangle\})$
RArc(ℓ)	$(\sigma \vert \sigma_1 \vert \sigma_0, \beta, A) \rightarrow (\sigma \vert \sigma_1 \vert \sigma_0, \beta, A \cup \{\langle \sigma_1, \ell, \sigma_0 \rangle\})$
Reduce	$(\sigma \vert \sigma_0, \beta, A) \rightarrow (\sigma, \beta, A)$
	or $(\sigma, \beta, A \cup (\sigma_0, sib(\sigma_0)))$ for reentrancies, see text for details.

Table 3.2: Transitions for AMREAGER. $sib(\sigma_0)$ refers to the latest created node with the same parent as σ_0.

3.4 Reentrancies

As discussed in Chapter 2, AMR annotations are represented as graphs and not trees because nodes can have multiple incoming edges, known as reentrancies. A node u is said to be reentrant if and only if $\exists v, v'$ such that $(v, l, u) \in E$ and $(v', l', u) \in E$, for some edge labels l, l'. Reentrancies are common in AMR, as shown in Table 3.1. Dependency parsers do not allow to create reentrancies so they need to be modified accordingly. In Chapter 2, we noted that control verbs result in edges between siblings, which lead to reentrancies. AMREAGER is able to recover such reentrancies, as discussed in Section 3.5.

3.5 Transition System for AMR Parsing

A stack $\sigma = \sigma_n \vert \cdots \vert \sigma_1 \vert \sigma_0$ is a list of nodes of the partially constructed AMR graph, with the top element σ_0 at the right. We use the symbol '|' as the concatenation operator. A buffer $\beta = \beta_0 \vert \beta_1 \vert \cdots \vert \beta_n$ is a list of indices from x, with the first element β_0 at the left, representing the word tokens from the input sentence still to be processed. A configuration of our parser is a triple (σ, β, A), where A is the set of AMR edges that have been constructed up to this point.

In order to introduce the transitions of AMREAGER, we need some additional notation. We use a function a that maps indices from the sentence x to AMR graph fragments, implementing the function $\overleftarrow{\pi}(\cdot)$. For each $i \in \{1, \ldots, n\}$, $a(i)$ is a graph $G_a = (V_a, E_a)$, with single root $\text{root}(G_a)$, representing the semantic contribution of word x_i to the AMR for the sentence x. As already mentioned, G_a can either have a single node, several nodes, or be empty.

The initial configuration of the system has a ∘ node (representing the root) in the stack and the entire sentence in the buffer. The terminal configuration consists of an empty buffer and a stack with only the ∘ node.

We define four transitions, specified by the rewriting rules shown in Table 3.2:

1. **Shift.** The transition Shift is used to decide if and which AMR node to push onto the stack after consuming a token from the buffer. Intuitively, the graph fragment $a(\beta_0)$ obtained from the token β_0, if not empty, is "merged" with the graph we have constructed so far. We then push onto the stack the node $\text{root}(a(\beta_0))$ for further processing.

2. **RArc.** $\text{RArc}(\ell)$ creates an edge with label ℓ between the second top-most node and the top-most node in the stack.

3. **LArc.** $\text{LArc}(\ell)$ is the symmetric operation: it creates an edge with label ℓ between the top-most node and the second top-most node in the stack. Moreover, LArc pops the top-most node in the stack (the dependent of the newly created edge). The choice of popping the dependent in the LArc transition is inspired by ARCEAGER, where left-arcs are constructed bottom-up to increase the incrementality of the transition system (Nivre, 2004). This affects our ability to recover some reentrant edges (edges that participate in a reentrancy): consider a node u with two parents v and v', where the arc $v \to u$ is a left-arc and $v' \to u$ is any arc. If the first arc to be processed is $v \to u$, we use LArc that pops u, hence making it impossible to create the second arc $v' \to u$. Nevertheless, we discovered that this approach works better than a completely unrestricted allowance of reentrancy. The reason is that if we do not remove dependents at all when first attached to a node, the stack becomes larger, and nodes which should be connected end up being distant from each other, and as such, are never connected.

4. **Reduce.** Finally, Reduce pops the top-most node from the stack, and it also determines whether to create an additional edge between the node being removed and the previously created sibling in the partial graph. These edges lead to reentrancies between siblings and are often caused by control structures, where two predicates share an argument, as dis-

cussed in Chapter 2. The transition system can therefore capture non-projective patterns, according to the definition given in Section 3.3. This way of handling control structures is related to the *REENTRANCE* transition by Wang et al. (2015a).

The transitions required to parse the sentence *The boy and the girl* are shown in Table 3.3, where the first line shows the initial configuration and the last line shows the terminal configuration.

We now show that our transition-based AMR parser takes linear time in n, the length of the input sentence x. We first show that the output graph has size $O(n)$, then bound the maximum number of transitions:

1. **Graph size**. Each token in x is mapped to a subgraph by Shift, but only its root is stored in the stack. Thus the number of nodes that can go in the stack is $O(n)$. Furthermore, each node can have at most three parent nodes, created by transitions RArc, LArc and Reduce, respectively. Thus the number of edges is also $O(n)$.

2. **Number of transitions**. It is possible to bound the maximum number of transitions required to parse x: the number of Shift is bounded by n, and the number of Reduce, LArc and RArc is bounded by the size of the graph, which is $O(n)$. Since each transition can be carried out in constant time, we conclude that our parser runs in linear time.

3.6 Preprocessing Pipeline

We preprocess the input sentences to the transition system by first running a tokenizer. We then collapse consecutive tokens representing multi-word named entities into a single token (e.g., *United Kingdom* becomes *United_Kingdom*). To accomplish this, we also run a Named Entity Recognizer (NER), in order to identify multi-word named entities. Moreover, we extract Part-Of-Speech (POS) tags and run a dependency parser, to extract additional features to train the parser, as discussed in Section 3.7.

Action	Stack	Buffer	Edges
-	[○]	[the,boy,and,the,girl]	{}
Shift	[○]	[boy,and,the,girl]	{}
Shift	[○, *boy*]	[and,the,girl]	{}
Shift	[○, *boy*, *and*]	[the,girl]	{}
LArc	[○, *and*]	[the,girl]	$\{\langle and,:op1,boy\rangle\} = A_1$
RArc	[○, *and*]	[the,girl]	$A_1 \cup \{\langle ○,:top,and\rangle\} = A_2$
Shift	[○, *and*]	[girl]	A_2
Shift	[○, *and*, *girl*]	[]	A_2
RArc	[○, *and*, *girl*]	[]	$A_2 \cup \{\langle and,:op2,girl\rangle\} = A_3$
Reduce	[○, *and*]	[]	A_3
Reduce	[○]	[]	A_3

Table 3.3: Parsing steps for the sentence *The boy and the girl*.

3.7 Training the System

Training a transition-based parser from data requires an oracle—an algorithm that given a gold-standard AMR graph and a sentence returns transition sequences that maximize the overlap between the gold-standard graph and the graph dictated by the sequence of transitions. Several components of our parser have to be learned from the oracle:

- A transition classifier that predicts the next transition given the current configuration;
- A concept identification routine to be called after each Shift to compute $a(\beta_0)$;
- A reentrancy classifier that decides whether or not to create a reentrancy between siblings after each Reduce;
- An edge classifier to predict the edge label after each LArc or RArc.

3.7.1 Oracle

We adopt a *shortest-stack static* oracle similar to Chen and Manning (2014). *Static* means that if the actual configuration of the parser has no mistakes, the

oracle provides a transition that does not introduce any mistake. *Shortest-stack* means that the oracle prefers transitions where the number of items in the stack is minimized. Given the current configuration (σ, β, A) and the gold-standard graph $G = (V_g, A_g)$, the oracle is defined as follows, where we test the conditions in the given order and apply the action associated with the first match:

1. if $\exists \ell[(\sigma_0, \ell, \sigma_1) \in A_g]$ then LArc(ℓ);
2. if $\exists \ell[(\sigma_1, \ell, \sigma_0) \in A_g]$ then RArc(ℓ);
3. if $\neg\exists i, \ell[(\sigma_0, \ell, \text{root}(a(\beta_i)) \in A_g \vee (\text{root}(a(\beta_i)), \ell, \sigma_0) \in A_g]$ then Reduce;
4. Shift otherwise.

The oracle first checks whether an edge should be constructed from the two elements at the top of the stack (conditions 1 and 2), in which case it also determines the label of the edge. If neither LArc nor RArc are possible, the oracle checks whether all possible edges in the gold graph involving σ_0 have already been created, in which case it chooses Reduce (condition 3). To this end, it suffices to check the buffer, since LArc and RArc have already been excluded and the parser can no longer access elements in the stack deeper than the second position. If this transition is chosen, the oracle can also decide whether or not to create an additional edge between σ_0 and its previous sibling. If Reduce is not possible, Shift is finally chosen (condition 4), removing β_0 from the buffer. The oracle needs the AMR alignments for the next token in the sentence. If the next token is not aligned with any node in the AMR graph, the stack does not change. Otherwise, if the next token is aligned to a subgraph, the root of the subgraph is copied to the top of the stack.

3.7.2 Transition Classifier

The transition classifier predicts which transition to apply given the current parser configuration. The examples from which we learn the classifier are extracted by applying the oracle of Section 3.7.1 to the training data. Each example consists of a parser configuration and the transition chosen by the oracle.

To learn this and the other classifiers, we use feed-forward neural networks. The input to the network consists of the concatenation of embeddings for words, POS tags, and edges of the dependency tree. In addition, we use

Name	Feature template
depth	$d(\sigma_0), d(\sigma_1)$
children	$\#c(\sigma_0), \#c(\sigma_1)$
parents	$\#p(\sigma_0), \#p(\sigma_1)$
lexical	$w(\sigma_0), w(\sigma_1), w(\beta_0), w(\beta_1)$,
	$w(p(\sigma_0)), w(c(\sigma_0)), w(cc(\sigma_0))$,
	$w(p(\sigma_1)), w(c(\sigma_1)), w(cc(\sigma_1))$
POS	$s(\sigma_0), s(\sigma_1), s(\beta_0), s(\beta_1)$
entities	$e(\sigma_0), e(\sigma_1), e(\beta_0), e(\beta_1)$
dependency	$\ell(\sigma_0, \sigma_1), \ell(\sigma_1, \sigma_0)$,
	$\forall i \in \{0,1\}$: $\ell(\sigma_i, \beta_0), \ell(\beta_0, \sigma_i)$
	$\forall i \in \{1,2,3\}$: $\ell(\beta_0, \beta_i), \ell(\beta_i, \beta_0)$
	$\forall i \in \{1,2,3\}$: $\ell(\sigma_0, \beta_i), \ell(\beta_i, \sigma_0)$

Table 3.4: Features used in the transition classifier. The function d maps a stack element to the depth of the associated graph fragment. The functions $\#c$ and $\#p$ count the number of children and parents, respectively, of a stack element. The function w maps a stack/buffer element to the word embedding for the associated word in the sentence. The function p gives the leftmost (according to the alignment) parent of a stack element, the function c the leftmost child and the function cc the leftmost grandchild. The function s maps a stack/buffer element to the POS embedding for the associated word. The function e maps a stack/buffer element to its entity. Finally, the function ℓ maps a pair of symbols to the dependency label embedding, according to the edge (or lack of) in the dependency tree for the two words these symbols are mapped to.

one-hot vectors for named entities and additional sparse features, extracted from the current configuration of the transition system. The features used are reported in more detail in Table 3.4. For lexical information, we also extract the leftmost (in the order of the aligned words) child (c), leftmost parent (p) and leftmost grandchild (cc). Leftmost and rightmost items are common features for transition-based parsers (Zhang and Nivre, 2011; Chen and Manning, 2014) but we found only leftmost items to be helpful in our case.

3.7.3 Concept Identification

This routine is called every time the transition classifier decides to do a Shift; it is denoted by $a(\cdot)$ in Section 3.5, approximating the function $\overleftarrow{\pi}(\cdot)$. Even this component could be learned in a supervised manner, but we were not able to improve on a simple heuristic, where we pick the most frequent subgraph for the given input word. During training, for each Shift decided by the oracle, we store the pair $(\beta_0, \overleftarrow{\pi}(i))$ in a phrase-table. During parsing, the most frequent subgraph H for the given token is chosen. In other words, $a(i)$ approximates $\overleftarrow{\pi}(i)$ with the graph most frequently seen among all occurrences of token x_i in the training set.

An obvious problem with the most-frequent heuristic is that it does not generalize to unseen words. In addition, our heuristic relies on the automatically generated alignments, which contain mistakes. In order to alleviate this problem, we observe that there are classes of words such as named entities and numeric quantities that can be treated in a deterministic manner. We define a set of *hooks* that are triggered by the named entity tag of the next token in the sentence, computed during the preprocessing step (Section 3.6). The hooks override the normal concept identification mechanism and apply a fixed rule instead. Table 3.5 reports examples for all hooks we implemented. We employ the same rule for states, cities, countries, and people. To generate the correct root node, which depends on the specific type of entity, we extracted lists of states, cities, and countries. We also have hooks for ordinal numbers (generating the AMR concept *ordinal-entity*), percentages (*percentage-entity*), money (*monetary-quantity*) and dates (*date-entities*). For these hooks, we also need to normalize the tokens. For instance, dates have to be converted in the *dd/mm/yyyy* format. We normalize dates and other entities during the preprocessing step.

3.7.4 Reentrancy Classifier

We train a binary classifier to decide whether or not to create a reentrant edge during a Reduce transition. The features used for this classifier are shown in Table 3.6. We use word and POS embeddings for the two nodes of the candidate reentrancy and their shared parent. If the dependency tree of the sentence contains edges between the two nodes of the candidate edge, dependency em-

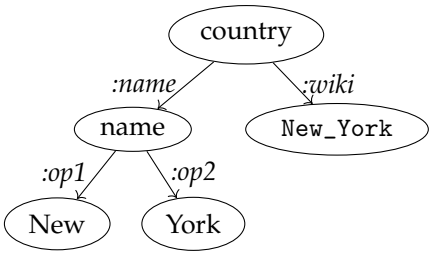
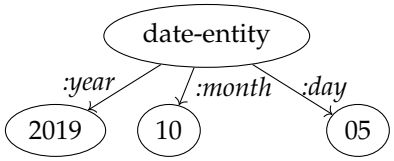
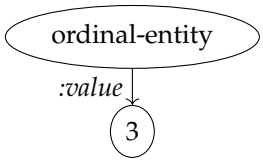
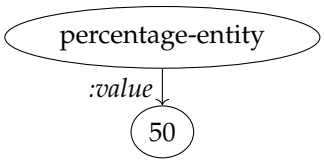
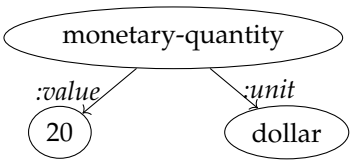
Type	Token	AMR
Name	*New_York*	
Date	*05/10/2019*	
Ordinal	*Third*	
Percentage	*50%*	
Money	*20$*	

Table 3.5: Example of hooks for names, dates, and numbers. The type "Name" include states, countries, cities, people, and organizations.

beddings are also used as features.

Name	Feature template
lexical	$w(\sigma_0)$, $w(sib(\sigma_0))$
POS	$s(\sigma_0)$, $s(sib(\sigma_0))$
dependency	$\ell(\sigma_0, sib(\sigma_0))$, $\ell(sib(\sigma_0), \sigma_0)$

Table 3.6: Features used in the reentrancy classifier. See Table 3.4 for a legend of symbols. $sib(\sigma_0)$ refers to the latest created sibling of σ_0.

3.7.5 Edge Classifier

Every time the transition classifier decides to take an LArc or RArc operation, the edge labeler needs to decide on a label for it. There are more than 100 possible labels such as *:ARG0*, *:ARG0-of*, *:ARG1*, *:location*, *:time* and *:polarity*. The features for the edge classifier are shown in Table 3.7.

Name	Feature template
depth	$d(\sigma_0)$, $d(\sigma_1)$
children	$\#c(\sigma_0)$, $\#c(\sigma_1)$
parents	$\#p(\sigma_0)$, $\#p(\sigma_1)$
lexical	$w(\sigma_0)$, $w(\sigma_1)$,
	$w(p(\sigma_0))$, $w(c(\sigma_0))$, $w(cc(\sigma_0))$,
	$w(p(\sigma_1))$, $w(c(\sigma_1))$, $w(cc(\sigma_1))$
POS	$s(\sigma_0)$, $s(\sigma_1)$
entities	$e(\sigma_0)$, $e(\sigma_1)$
dependency	$\ell(\sigma_0, \beta_0), \ell(\beta_0, \sigma_0)$

Table 3.7: Features used in the edge classifier. See Table 3.4 for a legend of symbols.

If unconstrained, the classifier could predict a label that does not satisfy the requirements of AMR. For instance, the label *:top* can only be applied when the node from which the edge starts is the special $\circ$ node. In order to avoid generating such erroneous labels, we use a set of rules, shown in Table 3.8. These rules determine which labels are allowed for the newly created edge so that we only consider those during prediction. The possible ARG-x roles for each predicate are extracted from Propbank. For example, while *add-01* and *add-02* allow for *:ARG1* and *:ARG2*, *add-03* and *add-04* only allow *:ARG2*.

3.8 Experimental Setup

The AMR dataset used in these experiments is the LDC2015E86 release. We use the JAMR aligner (Flanigan et al., 2014) to obtain alignments between the tokens in the sentence and the nodes in the respective AMR graph.[4] All classifiers are feed-forward neural networks with two hidden layers of 200 tanh

[4] `https://github.com/jflanigan/jamr`

Label	Ex.	Start	End
:top	Yes	◦	
:polarity	Yes		-
:mode	Yes		inter.| expr.|imp.
:value	No		\w+ |[0-9]$^+$
:day	No	d-ent	[1|2|···|31]
:month	No	d-ent	[1|2|···|12]$^+$
:year	No	d-ent	[0-9]$^+$
:decade	No	d-ent	[0-9]$^+$
:century	No	d-ent	[0-9]$^+$
:weekday	Yes	d-ent	[monday|···| sunday]
:quarter	No	d-ent	[1|2|3|4]$^+$
:season	Yes	d-ent	[winter|fall| spring|summer]$^+$
:timezone	Yes	d-ent	[A−Z]^3

Table 3.8: Labeling rules: For each edge label, we provide regular expressions that must hold on the labels at the start node (Start) and the end node (End) of the edge. Ex. indicates when the rule is exclusive, d-ent is the AMR concept *date-entity*, inter. is the AMR constant *interrogative*, expr. is the AMR constant *expressive*, imp. is the AMR constant *imperative*.

units each. We train using SGD with an initial learning rate set to 0.1 and linear decaying. Batch size is set to 32. The embeddings for words and POS tags were pre-trained on a large unannotated corpus consisting of the first 1 billion characters from Wikipedia.[5] All POS tags, dependencies and named entities are generated using Stanford CoreNLP (Manning et al., 2014).

We first evaluate the performance of each classifier on the development split of LDC2015E86. We also run ablation experiments to inspect the contribution of the hooks used to improve concept identification (Section 3.7.3) as well as the additional sibling edges in the Reduce transition for reentrancy prediction (Section 3.7.4). We then compare AMREAGER against previous work

[5] http://mattmahoney.net/dc/enwik9.zip

System	Accuracy	Frequency
Shift	91.45%	28187
LArc	84.86%	10189
RArc	79.43%	5616
Reduce	65.93%	6090

Table 3.9: Accuracy and frequency of each transition for the transition classifier on the development set of LDC2015E86.

on the test split. JAMR (Flanigan et al., 2014) and CAMR (Wang et al., 2015b,a) were popular previous parsers. JAMR works by first predicting the concepts and then identifying the relations between them. CAMR converts the dependency tree of a sentence into an AMR graph through a transition system. Both parsers were also updated for SemEval-2016 Task 8 (Flanigan et al., 2016a; Wang et al., 2016). We further compare against two parsers published after we released AMREAGER: the parser discussed in Peng et al. (2018) also uses a transition system where edges can be created among nodes stored in a cache, while Lyu and Titov (2018) was the state-of-the-art parser at the time of our experiments. It relies on a joint model of concepts, relations, and alignments.

To evaluate the parsers we use Smatch (Cai and Knight, 2013), which finds the optimal alignments between a pair of graphs and then computes precision, recall, and F1 of their edges. Since Smatch is an approximate randomized algorithm, decimal points in the results vary between different runs and are not reported. This approach was also taken by Wang et al. (2015b), inter alia.

3.9 Results

The accuracy of the transition classifier is reported in Table 3.9. The reentrancy classifier, which makes a binary decision, has an accuracy of 97.18%. Finally, the edge classifier achieves an accuracy of 77.51%.

Table 3.10 shows that without hooks the parser achieves lower scores. This demonstrates the effectiveness of the rule-based approach to concept identification for named entities. To investigate the contribution of the additional edges between siblings in Reduce, we test a variant of the transition which does not add them. This change makes the parser projective and limits its ability to

System	Precision	Recall	F1
AMREAGER	68	63	65
AMREAGER - hooks	67	59	63
AMREAGER - siblings	69	62	65

Table 3.10: Ablation experiments on the development set of LDC2015E86.

System	Precision	Recall	F1
JAMR (2014)	62	54	58
CAMR (2015)	69	59	63
JAMR (2016)	70	64	67
CAMR (2016)	70	63	67
AMREAGER (2017)	67	62	64
Peng et al. (2018)	69	59	64
Lyu and Titov (2018)	75	71	73

Table 3.11: Smatch scores on the test set of LDC2015E86. All models were trained on the LDC2015E86 dataset.

recover reentrancies. Table 3.10 shows that, in this case, the recall is lower, due to the fact that some reentrancies cannot be parsed. However, the precision is higher and the F1 does not change. Reentrancies are common in the AMR data (Table 3.1) and as such the parsers' ability to recover them should be reflected in the parsing score. This observation motivates the development of fine-grained evaluation metrics for AMR parsing, which we address in Chapter 4.

The scores of all parsers on the test set of LDC2015E86 are shown in Table 3.11, where we note that the proposed parser is competitive with previous parsers. The most closely related parser is that of Peng et al. (2018), which achieves the same F1 score, with higher precision but lower recall than AMREAGER. It is however difficult to closely compare parsers on the sole basis of the Smatch score. We address this in Chapter 4, where we define a suite of evaluation metrics to make this comparison easier.

3.10 Related Work

As AMR graphs are non-projective strcutures, as discussed in Section 3.3, non-projective transition systems are related to AMREAGER. For non-projective dependency parsing, pseudo-projective parsing has been proposed, where trees are projectivized via a pre-processing step (Nivre and Nilsson, 2005). Attardi (2006) and Cohen et al. (2011b) instead included transitions to create edges between items at non-adjacent positions in the stack, hence allowing crossing edges. Yet another alternative solution to parse non-projective dependency trees is to include a transition that reverses the order of the two topmost items in the stack. This idea has been applied to both dependency parsing (Nivre, 2009; Bohnet and Nivre, 2012) and SRL (Titov et al., 2009). For SRL, the pseudo-projective approach has also been used (Henderson et al., 2008). Unlike these approaches to non-projectivity, AMREAGER can only parse non-projective structures caused by reentrant edges between siblings.

As discussed in Chapter 2, several approaches to AMR parsing have been proposed, one of which is transition-based parsing. AMREAGER draws from the rich literature on transition systems for dependency parsing (Nivre, 2004, 2008; Sagae and Tsujii, 2008; Covington, 2011; Chen and Manning, 2014). Zhou et al. (2016) presented a transition system for AMR parsing, based on ARC-STANDARD (Nivre, 2004). The CAMR parser (Wang et al., 2015a), also defines a transition system. Instead of processing the sentence left-to-right, they process its dependency tree in a bottom-up traversal. In order to recover reentrancies, they also include a transition specifically design to recover reentrancies between siblings, as AMREAGER does. Ballesteros and Al-Onaizan (2017); Peng et al. (2018) proposed transition-based parsers with different ways to deal with reentrancies. The parser by Ballesteros and Al-Onaizan (2017) uses a swap transition, which allows for a restricted subset of reentrancies but is not limited to those between siblings. The parser by Peng et al. (2018) introduces a cache system that allows recovering (with an appropriate cache size) all reentrancies.

Our transition system is also related to an adaptation of ARCEAGER for DAGs, introduced by Sagae and Tsujii (2008). The latter is also the basis for Ribeyre et al. (2015), a transition system used to parse dependency graphs. Similarly, Du et al. (2014) also addressed dependency graph parsing with tran-

sition systems. Analogously to dependency trees, dependency graphs have the property that their nodes consist of the word tokens, which is not true for AMR. As such, these transition systems are more closely related to those used for dependency parsing.

3.11 Summary

In this chapter, we presented a transition system that builds AMR graphs in linear time by processing the sentences left-to-right, trained with feed-forward neural networks. AMREAGER provides linear worst-case complexity and allows for incremental AMR parsing. The parser demonstrates that it is possible to perform AMR parsing using techniques inspired by dependency parsing with few adjustments. Our parser is available at `https://github.com/mdtux89/amr-eager` and a demo is available at `http://cohort.inf.ed.ac.uk/amreager.html`.

We analyzed the contributions of single components of our transition system and showed that it is competitive with previous work. The Smatch score is not affected by the creation of siblings, which however increases the recall of reentrancy structures. A more in-depth comparison of the parsers is discussed in the next chapter, where we introduce a set of fine-grained evaluation metrics, including one for reentrancy prediction, which facilitates comparisons between parsers.

4

Evaluation and Analysis of Reentrant Structures in AMR Parsing

As discussed in Chapter 2, several semantic subtasks are involved in AMR parsing, such as coreference resolution, NER, and SRL. However, Smatch provides only a single score summarizing the overall quality of the parse. In Chapter 3, we discussed the difficulty in analyzing the differences between parsers when Smatch is the only available evaluation metric. In this chapter, we introduce a set of metrics to alleviate these problems and better compare parsers against each other.

One of the main properties of AMR, and the reason why sentences are represented as graphs rather than trees, is the presence of reentrancies, as discussed in Chapter 2. Reentrancies complicate AMR parsing and require the addition of specific transitions in transition-based parsing (Wang et al., 2015a; Damonte et al., 2017) or of pre- and post-processing steps in sequence-to-sequence parsing (van Noord and Bos, 2017a). Enabling AMR parsers to predict reentrancy structures correctly is particularly important because it separates AMR parsing from semantic parsing based on tree structures (Steedman, 2000; Liang, 2013; Cheng et al., 2017). Reentrancy is however not an AMR-specific problem (Kuhlmann and Jonsson, 2015), and other formalisms can benefit from a better understanding of how to parse such structures. For these reasons, one of the metrics that we propose evaluates parsers with respect to reentrancies. We found that the performance of parsers at recovering

reentrancy structures is generally poor, ranging between 40% and 54% F1 on LDC2015E86.

We argue that a better understanding of the role of reentrancies in AMR parsing can improve parser performance. To our knowledge, the AMR literature lacks a detailed discussion of reentrancies. Hence, we provide a classification of linguistic causes of reentrancy and quantify their prevalence in the corpus. We also take a closer look at how well AMR parsers deal with reentrancies and how to improve their performance. For this purpose, we analyze errors made by the parsers and use an oracle to demonstrate that correcting reentrancy-related errors leads to parsing score improvements.

Our contributions in this chapter are as follows:

- We propose a set of metrics to overcome the problems of using a single score for AMR parsing and better compare parsers;
- We use these metrics to more closely evaluate AMREAGER (Chapter 3);
- We classify the phenomena causing reentrancies, some which have not been discussed yet;
- We quantify their prevalence in the AMR corpus and discover additional sources of reentrancies;
- We categorize types of reentrancy errors made by the parsers and perform oracle experiments showing that correcting these errors can lead to improvements up to 20% in reentrancy prediction and 5% Smatch over state-of-the-art results;
- We establish baselines to correct the errors automatically as a post-processing step.

4.1 Fine-grained Evaluation

AMR parsers were traditionally evaluated using the Smatch score (Cai and Knight, 2013), as discussed in Chapter 2. We note that the Smatch score has two flaws: (1) while AMR parsing involves a large number of subtasks, the Smatch score consists of a single number that does not assess the quality of each subtask separately; (2) the Smatch score weighs different types of errors

in a way which is not necessarily useful for solving a specific NLP problem. For example, for a given application, concept detection might be deemed more important than edge detection, or guessing the wrong sense for a concept might be considered less severe than guessing the wrong verb altogether.

Consider two alternative parses for the sentence *Silvio Berlusconi gave Lucio Stanca his current role of modernizing Italy's bureaucracy* in Figure 4.1: *Parse 1* is not able to deal with named entities. *Parse 2* overpredicts the edge label *:ARG0*. The Smatch scores for the two parses are 56% and 78% respectively. Both parses contain obvious mistakes, but Smatch penalizes more the three named entity errors in *Parse 1* than the six wrong edge labels in *Parse 2*. This behavior, depending on the downstream application, may not be desirable.

In order to better understand the limitations of AMR parsers, find their strengths and gain insight in which downstream tasks they may be helpful, we instead define a set of fine-grained evaluation metrics.

The first set of metrics rely on Smatch. We preprocess the input AMR or select a subset of the AMR's triples, before running the Smatch algorithm. See Chapter 2 for an explanation of how Smatch extracts triples from the input AMR.

- UNLABELED. Before running Smatch, we preprocess the input graphs by replacing all the edge labels with the same dummy label. We do not normalize the inverse roles, unlike Smatch. In this way, we only assess the graph topology and the node labels. A good UNLABELED score may be enough to perform well at certain downstream applications, as it identifies the basic predicate-argument structure. For instance, we may be interested in knowing whether two events or entities are related to each other, while not being concerned with the precise type of relation holding between them. In the case of *Parse 2*, the UNLABELED score is 100% as the wrong edge labels are not taken into consideration for this metric.[1]

- NO WSD. AMR uses Propbank frames to disambiguate between senses. For example, *run-01* means *to operate* while *run-02* means to *walk quickly*. Before running Smatch, we preprocess the input graphs by removing the suffix from all concepts labeled with a Propbank frame, so that we can

[1]Similarly to how we report Smatch results, we report these scores as percentages.

Gold

```
(g / give-01
      :ARG0 (p3 / person :wiki "Silvio_Berlusconi"
            :name (n4 / name :op1 "Silvio" :op2 "Berlusconi"))
      :ARG1 (r / role :poss p4 :time (c2 / current)
            :mod (m / modernize-01 :ARG0 p4
                  :ARG1 (b / bureaucracy
                        :part-of (c3 / country :wiki "Italy"
                              :name (n6 / name :op1 "Italy")))))
      :ARG2 (p4 / person :wiki –
            :name (n5 / name :op1 "Lucio" :op2 "Stanca")))
```

Parse1

```
(g / give-01
      :ARG0 (p3 / silvio :mod (n4 / berlusconi))
      :ARG1 (r / role :poss p4 :time (c2 / current)
            :mod (m / modernize-01 :ARG0 p4
                  :ARG1 (b / bureaucracy :part-of (c3 / italy))))
      :ARG2 (p4 / lucio :mod (n5 / stanca)))
```

Parse2

```
(g / give-01
      :ARG0 (p3 / person :wiki "Silvio_Berlusconi"
            :name (n4 / name :op1 "Silvio" :op2 "Berlusconi"))
      :ARG0 (r / role :ARG0 p4 :ARG0 (c2 / current)
            :ARG0 (m / modernize-01 :ARG0 p4
                  :ARG0 (b / bureaucracy
                        :ARG0 (c3 / country :wiki "Italy"
                              :name (n6 / name :op1 "Italy")))))
      :ARG0 (p4 / person :wiki –
            :name (n5 / name :op1 "Lucio" :op2 "Stanca")))
```

Figure 4.1: At the top, the gold AMR graph for the sentence *Silvio Berlusconi gave Lucio Stanca his current role of modernizing Italy's bureaucracy*. In the middle, *Parse 1*, that scores 56% Smatch. At the bottom, *Parse 2*, that scores 78% Smatch. The mistakes of each parse are highlighted in red.

evaluate the parsers without taking WSD errors into account. For example, we preprocess the AMR representations in Figure 4.1 by replacing *give-01* with *give* and *modernize-01* with *modernize*. When the parsers do not make any WSD errors, this score is equivalent to Smatch, as seen for *Parse 1* and *Parse 2*.

- NP-ONLY. Similarly to Sawai et al. (2015), we evaluate the parsers on noun phrase parsing. We extract all noun phrases in the AMR dataset that are not included in another noun phrase and contain more than one noun. For the sentence in Figure 4.1, these are *Silvio Berlusconi*, *Lucio Stanca*, and *his current role of modernizing Italy's bureaucracy*. We then evaluate parsing on these phrases using Smatch. In Sawai et al. (2015), sentences with named entities, pronouns and conjunctions, which are difficult to either align or parse, are also filtered out. Our metric is agnostic to the specific alignment and parsing algorithms employed and therefore does not apply such constraints.

- REENTRANCIES. As we previously discussed, the presence of reentrancy is a very important characteristic of AMR graphs and is often difficult to handle. We therefore implement a test for reentrancy prediction. We run Smatch on the subset of the triples that involve variables with more than one parent, together with the instance triples of all the variables that appear in the selected triples. The triples extracted for the parses in Figure 4.1 are shown in Table 4.1. Before extracting triples, we follow Smatch in normalizing the inverse roles. As a result of this normalization, there can appear additional reentrancies. For instance, the triples for the *Gold* parse and *Parse 1* include those for the node *bureaucracy* because the edge *part-of* is inverted, hence creating a reentrancy. According to the REENTRANCIES metric, *Parse 1* produces two wrong triples, while *Parse 2* misses four triples and produces two wrong triples.

- SRL. SRL is an important subtask of AMR concerned with the identification of predicate-argument structures. We compute this metric similarly to REENTRANCIES, by running the Smatch score on the subset of normalized triples with a core (*:ARG*) role and the relative instance triples.

Furthermore, we define a second set of metrics which focuses on concepts

Gold	*Parse 1*	*Parse2*
root(g)	root(g)	root(g)
inst(m, modernize−01)	inst(m, modernize−01)	inst(m, modernize−01)
inst(p4, person)	**inst(p4,lucio)**	inst(p4, person)
inst(r, role)	inst(r, role)	inst(r, role)
inst(b, bureaucracy)	inst(b, bureaucracy)	
inst(c3, country)	**inst(c3,italy)**	
inst(g, give−01)	inst(g, give−01)	inst(g, give−01)
ARG0(m, p4)	ARG0(m, p4)	ARG0(m, p4)
ARG2(g, p4)	ARG2(g, p4)	**ARG0(g,p4)**
poss(r, p4)	poss(r, p4)	**ARG0(r,p4)**
ARG1(m, b)	ARG1(m, b)	
part(c3, b)	part(c3, b)	

Table 4.1: Triples extracted by the REENTRANCIES score for *Gold* (left), *Parse1* (middle), and *Parse 2* (right) of Figure 4.1. Bold and missing triples higlight the differences between the parses.

and not triples. As a consequence, Smatch is not needed and we use the F1 metric instead:

- CONCEPTS. Concept identification is another critical component of the parsing process. Identifying the correct concepts is fundamental: if a concept is not identified, it will not be possible to retrieve any edge involving that concept. To evaluate concept identification, we extract the list of concepts appearing in the predicted graph and the list of concepts appearing in the reference graph. We then compute the overlap between the two lists with the F1 metric. For instance, from both the *Gold* parse and *Parse 2* in Figure 4.1 we extract the following list of concepts: *person*, *bureaucracy*, *person*, *give-01*, *modernize-01*, *role*, *country*, *current*, *name*, *name*, and *name*. The set extracted from *Parse 1* does not have any *person*, *country*, or *name* concepts and instead have the concepts *italy*, *lucio*, *stanca*, *silvio*, and *berlusconi*.

- NAMED ENTITIES. We further compute the F1 score on the of concepts that participate in an outgoing edge with role *:name*. For *Gold* and *Parse 2*, these are *person*, *person*, and *country*, while there are no such concepts

Metric	Parse 1	Parse 2
SMATCH	56	78
UNLABELED	65	100
NO WSD	56	78
NP-ONLY	39	86
REENTRANCIES	83	50
CONCEPTS	48	100
NAMED ENTITIES	0	100
WIKIFICATION	0	100
NEGATIONS	0	0
SRL	83	60

Table 4.2: Evaluation of the two parses in Figure 4.1 with the proposed evaluation suite.

for *Parse 1*. Consequently, the score is 0% for *Parse 1* and 100% for *Parse 2*.

- WIKIFICATION. We also compute the F1 score on the concepts that participate in an ingoing edge with role *:wiki*, hence extracting the wikipedia identifiers: *Silvio_Berlusconi*, -, and *Italy* for *Gold* and *Parse 2*, none for *Parse 1*. Therefore, as for NAMED ENTITIES, the WIKIFICATION score is 0% for *Parse 1* and 100% for *Parse 2*.

- NEGATION. Finally, we define a metric for negation detection by computing the F1 score on the concepts that participate in an outgoing edge with role *:polarity*, hence extracting all negated concepts.

Using this evaluation suite we can evaluate AMR parsers on a wide range of metrics that can help us find the strengths and weaknesses of each parser, hence speeding up research in this area. Table 4.2 reports the scores for the two parses in Figure 4.1, where we see that *Parse 1* gets a high score for SRL while *Parse 2* is optimal for NER. We can also observe that *Parse 2* is optimal with respect to unlabeled score and that *Parse 1* is better at recovering reentrancies.

Metric	J (2014)	C (2015)	J (2016)	A (2017)	L&T (2018)
SMATCH	58	63	67	64	73
UNLABELED	61	69	69	69	73
NO WSD	58	64	68	65	74
NP-ONLY	54	57	64	61	63
REENTRANCIES	40	43	45	44	54
CONCEPTS	77	78	81	81	84
NAMED ENTITIES	73	72	76	81	86
WIKIFICATION	0	0	71	60	73
NEGATIONS	17	17	45	50	56
SRL	58	65	63	61	71

Table 4.3: Results on test split of LDC2015E86. A stands for AMREAGER, J for JAMR, C for CAMR, L&T is the parser by Lyu and Titov (2018).

4.1.1 Evaluation Results

We use the proposed evaluation metrics to compare AMREAGER with publicly available parsers: JAMR (Flanigan et al., 2014, 2016a), CAMR (Wang et al., 2015a), and a more recent parser by Lyu and Titov (2018).

Table 4.3 shows the results.[2] The parser by Lyu and Titov (2018) outperforms the others for all metrics, often by a large margin. The evaluation suite is most useful to compare parsers with similar performance, where we want to discern what are the advantages and disadvantages of using a parser rather than another one. Compared to the JAMR and CAMR parsers, who performs similarly to AMREAGER in terms of Smatch, our parser obtains the best results for UNLABELED, CONCEPT, NAMED ENTITIES, and NEGATIONS.

The good results we obtain for UNLABELED suggests that our parser has more difficulty in labeling the arcs than creating them. We also perform well at CONCEPTS. We predict concepts by choosing the most frequent subgraph for a given token based on a phrase-table (Chapter 3). Achieving good results with such a simple approach suggests that there is a relatively low level of token ambiguity in the dataset. We achieve good performance for NAMED ENTITIES and WIKIFICATION thanks to a rule-based approach (Chapter 3). With-

[2]The version of the evaluation suite used in this thesis differs from the one previously released, see Appendix A.

out these rules, NAMED ENTITIES drops from 83% to 77% and WIKIFICATION drops from 76% to 75%.

Most parsers do not perform well at NEGATIONS, possibly due to wrong automatic alignments with respect to polarity: words bearing negative polarity like *not*, *illegitimate* and *asymmetry* are sometimes not aligned to the - (minus) node in the AMR graph. To alleviate this problem, we perform a simple post-processing step on the aligner output: we collect a list of words bearing negative polarity. Every time that a - (minus) node is unaligned, we align it with one of these words, if they appear in the sentence. This resulted in an increase of the NEGATIONS score from 47% to 50%.

In Chapter 3, we discussed the use of the Reduce transition, which targets reentrancies between siblings, often caused by control verbs. We showed that the transition does not have an impact on the Smatch score but argued that it is useful to recover more reentrancies. The REENTRANCIES score, which drops from 44% to 39% when Reduce is removed, confirms this hypothesis. We note that all parsers do not perform well at recovering reentrancies, which motivates a more careful study of reentrancies in AMR.

4.2 Reentrancies

In Chapter 3, we reported that more than 40% of 200 manually annotated sentences contain at least one reentrancy. Kuhlmann and Oepen (2016) showed detailed statistics of the presence of reentrancies in AMR and other graph formalisms. Van Noord and Bos (2017a) proved that strategies to pre- and post-process reentrancies can improve the performance of sequence-to-sequence AMR parsers. Pop et al. (2018) reported a similar analysis for a transition-based parser. The importance of reentrancies for AMR therefore warrants an analysis of the phenomena that cause them and the errors that AMR parsers typically make when predicting them.

4.2.1 Phenomena Causing Reentrancies

Before diving into the errors caused by reentrancies, we first discuss what phenomena cause them (Table 4.4), and quantify the prevalence of those causes in the AMR corpus. We introduce three broad types of reentrancy triggers: syn-

tactic, pragmatic, and AMR-specific.

We consider a reentrancy as syntactically triggered if the syntactic structure of a sentence forces an interpretation in which one entity performs more than one semantic role. In the examples above, we denote this by co-indexing:

(3) *The man$_i$ saw himself$_i$ in the mirror.*

(4) *They$_i$ want ε_i to believe.*

(5) *I asked you$_i$ ε_i to sing.*

(6) *She$_i$ ate and ε_i drank.*

Some of the syntactic triggers are commonly discussed in the AMR literature: pronominal anaphora resolution (3), prototypical subject and object control (4 and 5), and coordination (6) (Groschwitz et al., 2017; van Noord and Bos, 2017a). In addition to those, other kinds of control structures, primarily adjunct control, are frequent reentrancy triggers. In adjunct control the clause which lacks a subject is an adjunct of the main clause, as in the following examples:

(7) *I$_i$ went home before ε_i eating.*

(8) *She$_i$ left the room ε_i crying.*

Such adjuncts express various additional information regarding the main clause, for example the goal, reason, or timing of an event. Unlike the prototypical cases of control, there is by definition no finite list of verbs associated with adjunct control.

It is worth noting that one would expect relative clauses to be one of the syntactic reentrancy triggers, because the noun involved has a semantic role in both the main and relative clause:

(9) *I saw the woman$_i$ who ε_i won.*

In the example above, the woman is the object of seeing and the subject of winning. However, according to the AMR guidelines (Banarescu et al., 2013) relative clauses should be annotated as attaching to the noun with an inverse role, thereby avoiding a reentrancy (see Table 4.4). Relative clauses therefore cause reentrancies only when normalizing all inverse edges, as done in Smatch and the evaluation metrics of Section 4.1. Because of this, the REENTRANCIES metric also considers reentrancies caused by relative clauses.

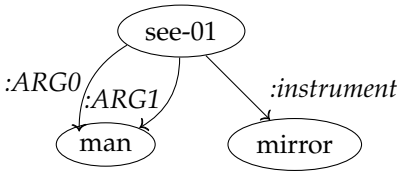
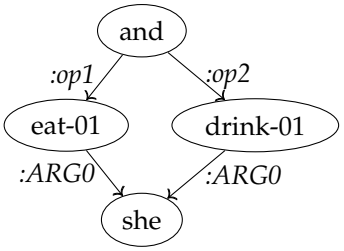
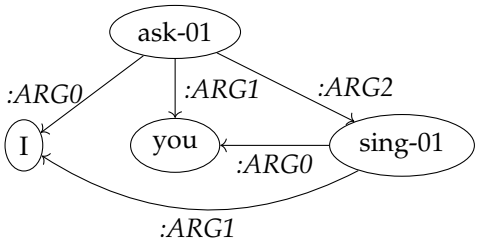
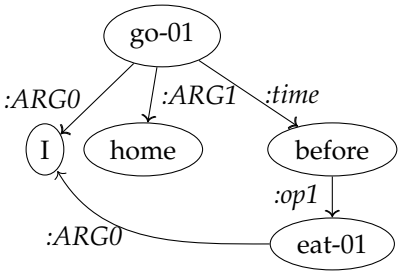
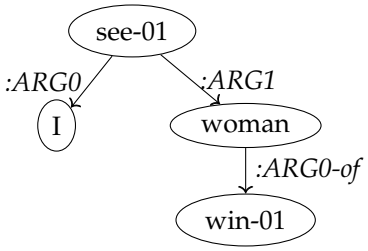
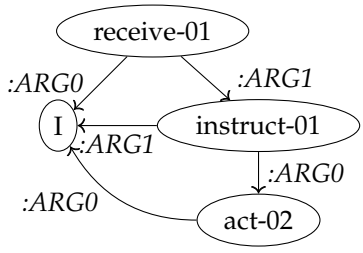
Phenomenon	Sentence	AMR
Coreference	*The man saw himself in the mirror*	
Coordination	*She ate and drank*	
Control	*I asked you to sing*	
Adjunct control	*I went home before eating*	
Relative clause	*I saw the woman who won*	
Verbalization	*I received instructions to act*	

Table 4.4: Several linguistic phenomena causing reentrancies in AMR. In the relative clause example, a reentrancy appears when the *:ARG0-of* role is inverted.

The human annotators resolve instances of coreference even in the absence of definite syntactic clues, giving rise to pragmatically triggered reentrancies. To this class belongs general coreference resolution. While coreference is, in general, a discourse phenomenon (Hobbs, 1979), it is also applicable to individual sentences such as those in the AMR corpora:

(10) *The coach of FC Barcelona said the team had a good season.*

In the example above, it is pragmatically understood that *FC Barcelona* and *the team* refer to the same entity, even though the coach could have been talking about another team. Another example is provided by control-like structures within nominal and adjectival phrases:

(11) *They$_i$ have a right* ε_i *to speak freely.*

(12) *He$_i$ was crazy* ε_i *to trust them.*

An AMR annotation will state that that the possessor of the *right* and the subject of *speak* are the same. The recovery of the subject of the infinitival clause in such constructions is driven by semantics or pragmatics rather than syntax (Huddleston and Pullum, 2002).

Finally, the last source of reentrancies is AMR conventions:

(13) *I received instructions to act.*

The guidelines encourage annotators to use OntoNotes predicates whenever possible (verbalization), regardless of the part of speech of the word. In the sentence above, the plural noun *instructions* appears in the AMR graph as a predicate node *instruct-01*. This encourages explicitly annotating inferred semantic roles and so *I* becomes an object of *instruct-01*, causing a reentrancy. Additionally, because of the control-like structure, *I* is also annotated as an object of acting.

4.2.2 Quantitative analysis

In order to assess the prevalence of the various reentrancy triggers, we designed heuristics to assign each reentrancy in the AMR corpus to one of the above phenomena. [3] We automatically align AMR graphs to their source sen-

[3] Ida Szubert, a co-author of Damonte et al. (2019), contributed to the classification of the causes of reentrancies and designing of the heuristics.

tences using JAMR (Flanigan et al., 2014) and identify the spans of words associated with reentrant nodes.[4] Heuristics based on Universal Dependency (UD) parses (Manning et al., 2014) and automatic coreference resolution are applied to the spans and the AMR subgraphs containing the reentrancy to classify the cause.[5] We use the NeuralCoref project for coreference resolution.[6]

We recognize syntactic reentrancy triggers primarily with UD-based heuristics. For prototypical cases of control we look for common control verbs such as *want*, *try*, and *persuade*,[7] with an outgoing *xcomp* dependency. To identify other types of control, such as adjunct control, we look for *xcomp*, *ccomp* or *advcl* dependency between words aligned to parents of a reentrant node. For coordination we only check the AMR itself, looking for coordination nodes (i.e., nodes labeled with *and*, *contrast-01*, or *or*). For coreference, we look for reentrant nodes associated with more than one span and check if those spans corefer. Finally, for verbalization, we look for nouns or adjectives aligned with OntoNotes predicates in the AMR graph. We tried to identify nominal control-like structures by looking for nominals with an *acl* dependent infinitive or gerund subject-less verb. However, as the precision of the rule is low, and most examples uncovered by this heuristic also fall into the verbalization category, we do not include it in our statistics.

The results of this analysis are in Table 4.5. The most common cause of reentrancy is coreference. Control is almost as frequent but control verbs only account for 15% of all control reentrancies, the rest being mostly adjunct control.

We note that our heuristics cannot find the cause for 46% of all reentrancies. This can happen for several reasons. First, the coreference resolution system is noisy, which can impact the coreference heuristic. Consider the following sentence:

(14) *The countries signed an agreement that binds the signatories.*

The coreference resolution system can fail to detect that *The countries* and *the signatories* corefer, which causes a reentrancy. Similarly, the alignments between words in the sentence and AMR nodes can contain mistakes, which af-

[4]https://github.com/jflanigan/jamr
[5]https://stanfordnlp.github.io/CoreNLP
[6]https://github.com/huggingface/neuralcoref
[7]https://en.wiktionary.org/wiki/Category:English_control_verbs

Phenomenon	Frequency
Coreference	18%
Control	16%
Coordination	11%
Verbalization	9%
Rest	46%

Table 4.5: Percentage of reentrancies in the LDC2015E86 training set found by our heuristics. "Rest" are all reentrancies for which our heuristics fail to detect the cause.

fects the heuristics that rely on them. The dependency parser and the POS tagger introduce additional noise.

Unaccounted reentrancies may also be caused by other phenomena that we did not anticipate. We therefore selected a random sample of 50 sentences and annotated the causes of their 79 unaccounted reentrancies. We found that 8% of these were due to the annotators overreaching in their pragmatic interpretation of the sentence. Consider the sentence:

(15) *The group said the foreign broadcasters are battering their culture and that it is insulting behavior.*

In its AMR, the node *insult-01* takes *group* as its *:ARG1*, making an arguably unwarranted assumption that the behavior is insulting to the group. We note that the inclusion of this type of reentrancies in AMR is controversial as it annotates beyond what semantics should represent. Ellipsis cause 5% of the reentrancies, as in the sentence:

(16) *Who can afford it and who can't.*

In this case, the AMR contains a reentrancy for *it*. Nominal control-like structures are responsible for 5% of the unaccounted reentrancies. We discussed this case in Section 4.2.1 but could not devise a reliable heuristic for it. Finally, 11% of the unaccounted reentrancies were due to mistakes in the AMR annotations. For example, in the following sentence, the annotator created an erroneous edge between *remove-01* (*removed*) and the *make-19* (*make*).

(17) *People were removed from their homeland to make way for the base.*

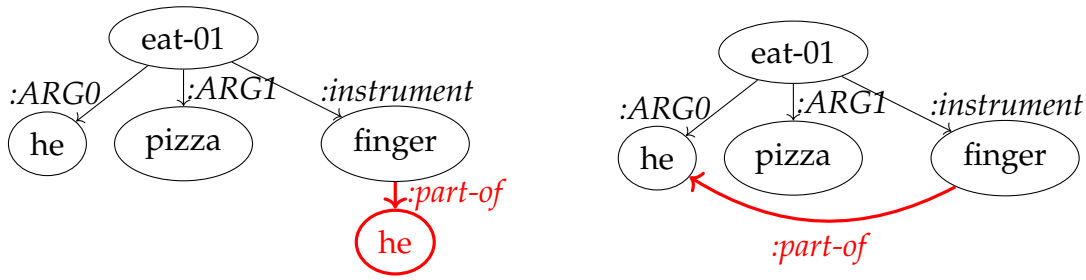


Figure 4.2: Left: a coreference-related reentrancy error for the sentence *He ate the pizza with his fingers*. Right: the correct reentrancy. The difference is highlighted in red.

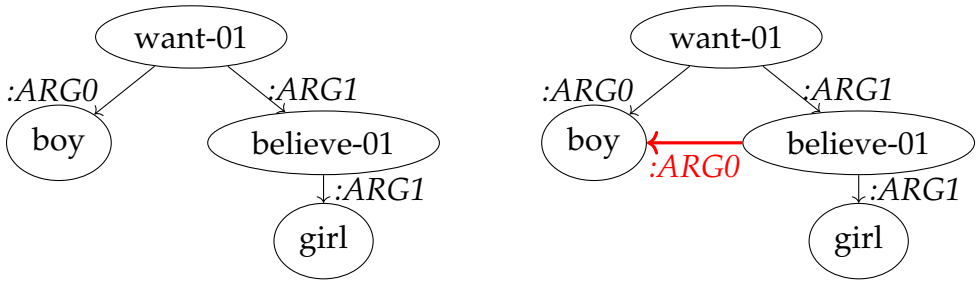


Figure 4.3: Left: a control-related reentrancy error for the sentence *The boy wants to believe the girl*. Right: the correct reentrancy.

4.2.3 Reentrancy-related Parsing Errors

In order to identify the reentrancy errors made by an AMR parser, we compare the predicted AMR graphs with the gold standard. We use Smatch to find the best alignments between variables of the predicted and gold graph. We can then find cases where the predicted graph is either missing a reentrancy or contains an unnecessary one. A typical reentrancy error involves the parser generating two nodes in place of one in the gold standard, as shown in Figure 4.2. The opposite is also possible, where two nodes are erroneously collapsed. Reentrant edges often occur between siblings. This happens in some cases of coreference (Figure 4.2) as well as control (Figure 4.3).[8]

The process of extracting the error patterns is prone to error. The alignments between the predicted and the gold graph are computed by Smatch, introducing noise. When the predicted graph contains errors, the correct alignment may not be found, which can affect our ability to find the error patterns.

[8]It is possible to classify errors by phenomena following Section 4.2.2. However, we found that this approach is too noisy and greatly diminishes the number of errors that can be detected.

4.2.4 Oracle

Our oracle introduces corrections for the errors discussed, implemented as actions that modify the edges and nodes of the predicted AMR.

Let the predicted graph, containing n nodes, be defined as:[9]

$$S = (V_s, E_s),$$
$$V_s = \{s_1, s_2, \ldots, s_n\},$$
$$E_s = \subseteq V_s \times V_s.$$

and the target graph, containing m nodes, be defined as:

$$T = (V_t, E_t),$$
$$V_t = \{t_1, t_2, \ldots, t_m\},$$
$$E_t = \subseteq V_t \times V_t.$$

Let $A(\cdot)$ be an alignment (computed using Smatch) that maps a node in V_s to a node in V_t, or *nil* if the node is not in present in V_t. Let $A^{-1}(\cdot)$ be an alignment that maps a node in V_t to a node in V_s, or *nil* if the node is not in present in V_s. Then, given a source node s_i, we define $t_i = A(s_i)$ and $s_i = A^{-1}(\cdot)$.

The oracle introduces the following actions:

- ADD: An edge is added (Figure 4.4a).
- ADD-ADDN: An edge and a node are added (Figure 4.4b).
- RM: An edge is removed (Figure 4.4c).
- RM-RMN: An edge and a node are removed (Figure 4.4d).
- MERGE: Two nodes are merged (Figure 4.5a).
- MERGE-RMN: Two nodes are merged and a node removed (Figure 4.5b).
- SPLIT: A node is split in two already existing nodes (Figure 4.5c).
- SPLIT-ADDN: A node is split in one existing node and a new node (Figure 4.5d).
- ADD-SIB: An edge between siblings is added (Figure 4.6a).

[9]Note that, for the purpose of our oracle, we ignore the edge labels of the AMR graphs.

- ADD-SIB-ADDN: A node is added and an edge with one of its sibling nodes is added (Figure 4.6b).
- RM-SIB: An edge between siblings is removed (Figure 4.6c).
- RM-SIB-RMN: An edge between siblings and one of the sibling nodes are removed (Figure 4.6d).

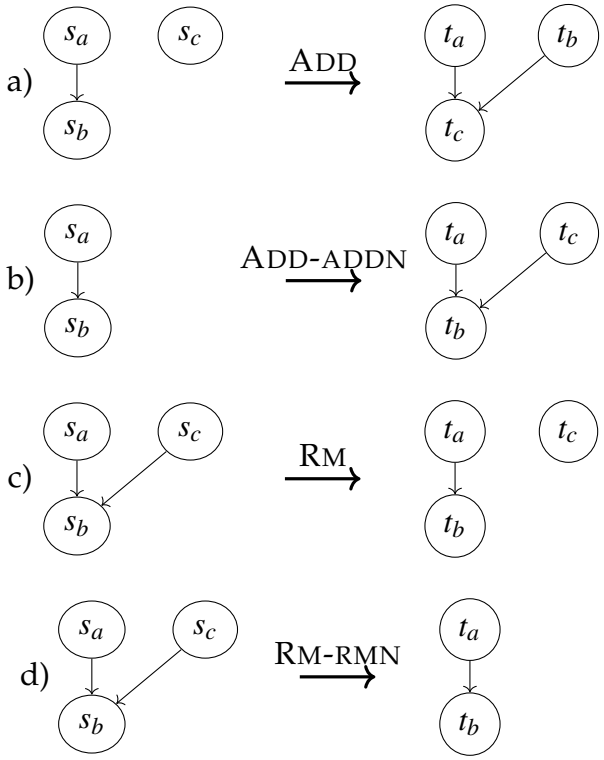


Figure 4.4: Actions to solve errors caused by missing or extra reentrancies.

For instance, for ADD (Figure 4.4a), we identify three variables s_a, s_b, s_c and the aligned variable in the target graph $t_a = A(s_a), t_b = A(s_b), t_c = A(s_c)$ such that:

$$(s_a, s_b) \in E_s, (s_c, s_b) \notin E_s,$$
$$(t_a, t_b) \in E_t, (t_c, t_b) \in E_t.$$

The oracle then creates an edge between the two siblings:

$$E_s = E_s \cup (s_c, s_b).$$

The definition of all actions is reported in Table 4.6. We also consider the combination of all actions (ALL). We do so by correcting one error type at the time in a pre-determined order:[10] for each error type, we re-run the oracle to find all errors after the actions for the previous type were applied.

[10]We sort the actions by the REENTRANCY score on LDC2017T10 in decreasing order.

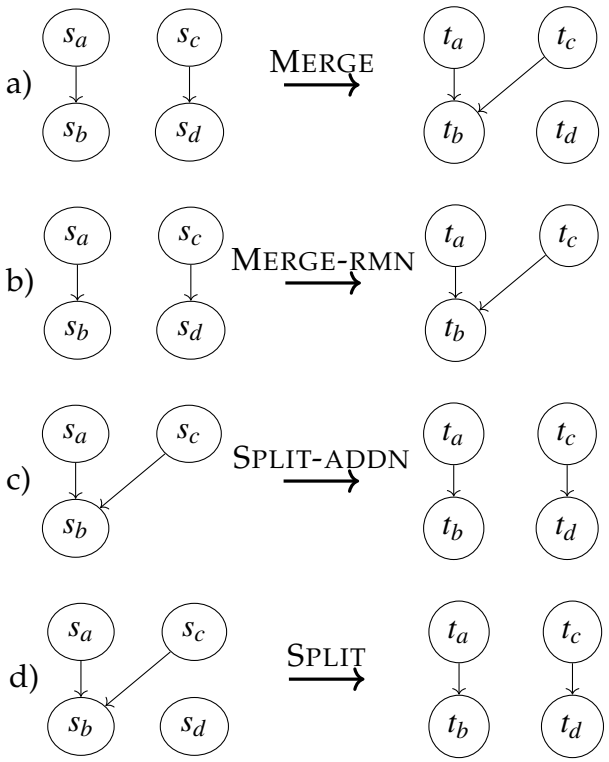


Figure 4.5: Actions to solve errors due to duplicated or collpased nodes.

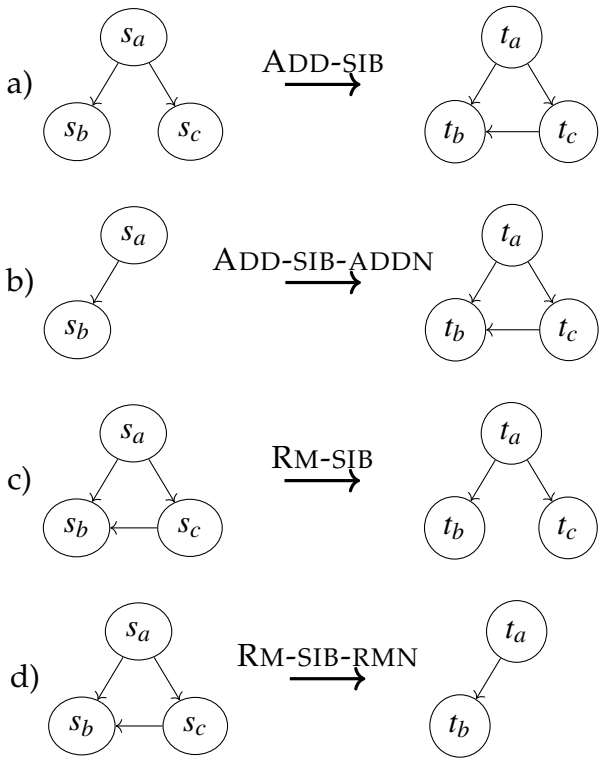


Figure 4.6: Actions to solve errors due to reentrancies between siblings.

Action	Condition	Effect
ADD	$(s_a,s_b) \in E_s, (s_c,s_b) \notin E_s, (t_a,t_b) \in E_t$ $(t_c,t_b) \in E_t$	$E_s = E_s \cup (s_c,s_b)$
ADD-ADDN	$s_a,s_b) \in E_s, A^{-1}(t_c) = nil, (t_a,t_b) \in E_t$ $(t_c,t_b) \in E_t$	$V_s = V_s \cup t_c,$ $E_s = E_s \cup (s_c,s_b)$
RM	$(s_a,s_b) \in E_s, (s_c,s_b) \in E_s, (t_a,t_b) \in E_t$ $(t_c,t_b) \notin E_t$	$E_s = E_s - (s_c,s_b)$
RM-RMN	$(s_a,s_b) \in E_s, (s_c,s_b) \in E_s, (t_a,t_b) \in E_t$ $A(s_c) = nil$	$V_s = V_s - s_c,$ $E_s = E_s - (s_c,s_b)$
MERGE	$(s_a,s_b) \in E_s, (s_c,s_d) \in E_s, (s_c,s_b) \notin E_s$ $(t_a,t_b) \in E_t, (t_c,t_d) \notin E_t, (t_c,t_b) \in E_t$	$E_s = E_s \cup (s_c,s_b) - (s_c,s_d)$
MERGE-RMN	$(s_a,s_b) \in E_s, A(s_d) = nil, (s_c,s_b) \notin E_s$ $(t_a,t_b) \in E_t, (t_c,t_d) \notin E_t, (t_c,t_b) \in E_t$	$V_s = V_s - s_d,$ $E_s = E_s \cup (s_c,s_b) - (s_c,s_d)$
SPLIT	$(s_a,s_b) \in E_s, (s_c,s_b) \in E_s, (s_c,s_d) \in E_s$ $(t_a,t_b) \in E_t, (t_c,t_d) \in E_t$	$E_s = E_s \cup (s_c,s_d) - (s_c,s_b)$
SPLIT-ADDN	$(s_a,s_b) \in E_s, (s_c,s_b) \in E_s, A^{-1}(t_d) = nil$ $(t_a,t_b) \in E_t, (t_c,t_d) \in E_t$	$V_s = V_s \cup t_d,$ $E_s = E_s \cup (s_c,t_d) - (s_c,s_b)$
ADD-SIB	$(s_a,s_b) \in E_s, (s_a,s_c) \in E_s, (s_c,s_b) \notin E_s$ $(t_a,t_b) \in E_t, (t_a,t_c) \in E_t, (t_c,t_b) \in E_t$	$E_s = E_s \cup (s_c,s_b)$
ADD-SIB-ADDN	$(s_a,s_b) \in E_s, A^{-1}(t_c) = nil, (t_a,t_b) \in E_t$ $(t_a,t_c) \in E_t, (t_c,t_b) \in E_t$	$V_s = V_s \cup t_c,$ $E_s = E_s \cup (t_c,s_b)$
RM-SIB	$(s_a,s_b) \in E_s, (s_a,s_c) \in E_s, (s_c,s_b) \in E_s$ $(t_a,t_b) \in E_t, (t_a,t_c) \in E_t, (t_c,t_b) \notin E_t$	$E_s = E_s - (s_c,s_b)$
RM-SIB-RMN	$(s_a,s_b) \in E_s, (s_a,s_c) \in E_s, (s_c,s_b) \in E_s$ $(t_a,t_b) \in E_t, A(s_c) = nil$	$V_s = V_s - t_c,$ $E_s = E_s - (s_c,s_b)$

Table 4.6: Definition of all oracle actions.

4.2.5 Oracle Results

We run oracle experiments to explore the impact of reentrancy-related errors, on both Smatch score and REENTRANCIES score (Section 4.1). We experiment

with the parser by Lyu and Titov (2018) and follow their experimental setup, evaluating on both LDC2015E86 and LDC2017T10 datasets.

Because Smatch is randomized, different runs of the oracle can identify different errors to correct: we therefore compute the mean and standard deviation of 3 runs.

Results are shown in Table 4.7.[11]

While the largest improvements are observed when correcting all error types, the most relevant single oracle action is ADD. For this action, we obtain considerable improvements for both corpora, especially for reentrancy prediction (increase by 10.4 and 10.3 points), but also for Smatch (increase by 1.7 points for both corpora). The ADD corrections provide more than half of the reentrancy score improvement provided by ALL corrections, and slightly less than half of the Smatch improvement. Actions ADD-SIB and RM-SIB do not account for large improvements. This explains the ablation results of Chapter 3, where we observed that a transition that recovers reentrancies between siblings did not affect the overall Smatch score.

Because of the use of noisy alignment in oracle action prediction, the oracle provides a lower band estimate of the possible gains. Overall, we argue that the room for improvement is large enough to warrant more careful treatment of reentrancies, either during training or as a post-processing step.

[11]To find and correct errors, we act directly on the triples, not on the PENMAN notation used by Smatch. We therefore implemented a variant of Smatch that directly read triples.

	LDC2015E86			LDC2017T10		
Action	**Freq.**	**Smatch**	**Reent.**	**Freq.**	**Smatch**	**Reent.**
VANILLA	-	73.9	54.3	-	75.2	56.9
ALL	3108.3 (11.59)	+4.6	+18.8	3093.7 (10.12)	+4.4	+18.0
ADD	1292.0 (7.94)	+1.7	+10.4	1305.7 (3.21)	+1.7	+10.3
ADD-ADDN	330.0 (4.36)	+0.8	+4.2	281.3 (5.51)	+0.7	+3.1
RM	545.7 (3.06)	+0.4	-0.1	572.3 (4.04)	+0.4	-0.1
RM-RMN	217.0 (2.00)	+0.3	+0.6	224.7 (3.06)	+0.2	+0.8
MERGE	187.3 (1.53)	+0.4	+1.6	193.3 (3.06)	+0.4	+1.7
MERGE-RMN	94.3 (1.15)	+0.3	+1.0	84.0 (2.00)	+0.2	+0.9
SPLIT	574.7 (3.21)	+1.2	+1.8	541.3 (4.16)	+1.1	+1.7
SPLIT-ADDN	333.0 (1.00)	+0.9	-0.2	347.3 (3.79)	+0.9	-0.0
ADD-SIB	128.0 (1.00)	+0.2	+1.3	119.7 (1.15)	+0.1	+1.2
ADD-SIB-ADDN	99.7 (3.06)	+0.1	-0.1	104.3 (1.53)	+0.1	-0.0
RM-SIB	69.3 (0.58)	+0.1	+0.2	89.3 (0.58)	+0.0	+0.2
RM-SIB-RMN	0.0 (0.00)	+0.0	-0.1	0.0 (0.00)	+0.0	+0.0

Table 4.7: Relative Smatch improvements with respect to Lyu and Titov (2018) of all actions on the test split of LDC2015E86 and LDC2017T10. Freq. is the number of times the action could be applied, Smatch is the parsing score and Reent. is the REENTRANCIES score. ALL is the combination of all actions. VANILLA are the scores obtained by the original parsers. In parentheses, we report the standard deviation of the actions' frequency. The standard deviation for the Smatch and REENTRANCIES scores is less or equal than 0.12.

4.2.6 Automatic Error Correction

We provide baseline systems that learn when to apply the ADD action:

- RANDOM. We randomly select two nodes in the predicted graph that are not connected by an edge and add one with the most likely label (*ARG0*).
- PATTERNS. We store frequent patterns in the training set that cause the application of the action ADD. Patterns consist of the labels of the nodes s_a, s_b, s_c on the left-hand side of Figure 4.4a. During testing, when one of the stored patterns is found, we apply the action ADD.
- SEQ2SEQ. We train a OpenNMT-py (Klein et al., 2017) sequence-to-sequence model (Bahdanau et al., 2015) with a copy mechanism (Gulcehre et al., 2016). The input sequence is the predicted graph and the output sequence is the sequence of edges to add. For each edge, the output contains three tokens: the parent node, the child node, and the edge label.

Table 4.8 shows the reentrancy prediction results. To study the impact of reentrancies on a different parser, we also report results on the character-level neural parser by van Noord and Bos (2017b). None of the baselines can improve over the predictions of the original parsers (VANILLA), with SEQ2SEQ being the baseline that gets closer to improve results. While sequence modeling of the output is convenient, other options can be attempted. We are also only exploiting the input AMR parse but not the input sentence. We leave it to future work to address these issues and achieve better results.

4.3 Related Work

Traditional evaluation of AMR parsers with Smatch was discussed in Chapter 2. The proposed evaluation suite has been widely used (May and Priyadarshi, 2017; van Noord and Bos, 2017b; Anchiêta and Pardo, 2018; Lyu and Titov, 2018, inter alia). The REENTRANCIES metric demonstrated the poor performance of parsers at predicting reentrancy structures, and motivated ad-hoc processing of reentrancies (van Noord and Bos, 2017a; Pop et al., 2018), not exclusively in AMR (van Noord et al., 2018). Opitz and Frank (2019) introduced the task of automatically predicting the scores of our evaluation metrics.

System	L&T	V&B
VANILLA	56.9 (0.00)	53.3 (0.00)
ORACLE	+10.3 (0.00)	+12.3 (0.06)
RANDOM	-4.2 (0.06)	-3.8 (0.06)
PATTERNS	-0.5 (0.06)	-0.6 (0.06)
SEQ2SEQ	-0.1 (0.25)	-0.1 (0.00)

Table 4.8: Relative improvements in reentrancy prediction scores on the test set of LDC2017T10, obtained by the oracle and the proposed baselines. L&T is Lyu and Titov (2018) and V&B is van Noord and Bos (2017b). VANILLA are the scores obtained by the original parsers. Results are the mean of three runs, with standard deviation in parentheses.

Our classification of the phenomena causing reentrancies extends previous work in this direction (Groschwitz et al., 2017). van Noord and Bos (2017a) previously attempted to improve the prediction of reentrancies in a neural parser. They experiment with several pre- and post-processing techniques and showed that co-indexing reentrancies nodes in the AMR annotations yields the best results. Several transition-based parsers have been specifically designed to handle reentrancies, as discussed in Chapter 3.

Transformation-based learning (Brill, 1993) inspired the idea of correcting existing parses. This approach has been mostly used for tagging (Ramshaw and Marcus, 1999; Brill, 1995; Nguyen et al., 2016) but it has also shown promises for semantic parsing (Jurčíček et al., 2009). A similar approach has been also used to add empty nodes in constituent parses (Johnson, 2002), with considerable success. The PATTERN baseline we presented is related to the approach by Johnson (2002) in generating transformation rules based on gold standard data, but the rules themselves are considerably different. Our rules are lexicalized and do not contain the relation labels, while the rules by Johnson (2002) focus only on unlexicalized syntactic structures. Moreover, our pattern extraction procedure relies on a noisy matching between gold standard and parser output graphs. The SEQ2SEQ baseline is a simple adaptation of the popular sequence-to-sequence modeling (Bahdanau et al., 2015).

4.4 Summary

In this chapter, we discussed the evaluation of AMR parsers. We noted that it is less informative to evaluate the entire parsing process with Smatch than to use a collection of metrics aimed at evaluating the various subproblems in the parsing process. We proposed a suite of evaluation metrics to better assess the quality of AMR parsers, which is available at `https://github.com/mdtux89/amr-evaluation`. We evaluated the parser of Chapter 3 with the proposed metrics, shedding lights on its strengths and limitations. Using the REENTRANCIES score, we could demonstrate that the Reduce transition we proposed in Chapter 3 improves reentrancy prediction.

Building upon previous observations that AMR parsers do not perform well at recovering reentrancies, we carried out an in-depth analysis of the linguistic phenomena responsible for reentrancies in AMR. We found sources of reentrancies which have not been acknowledged in the AMR literature such as adjunct control, verbalization, ellipsis, and pragmatics. We then quantified their prevalence in an AMR corpus. The inclusion of reentrancies due to pragmatics is controversial; we hope that this work can spur new discussions on the role of reentrancies. Our heuristics fail to detect the causes of many reentrancies. For a more precise estimate of the most common causes of reentrancies, it is necessary to manually annotate the reentrancies in the AMR corpora, which we leave for future work.

Our oracle experiments show that there is room for improvement in predicting reentrancies, which in turn can translate to better parsing results. Future work is necessary to outperform the proposed baselines and more effectively learn how to correct reentrancy errors. An alternative approach is to reduce reentrancy errors by better informing training so that the errors are avoided in the first place. We note that a recent AMR parser (Zhang et al., 2019) outperforms the previous state of the art (Lyu and Titov, 2018) by implementing a copy mechanism aimed at recovering reentrancies, confirming that reentrancies are critical for achieving good AMR parsing performance.

In the next chapter, we temporarily put aside the issue of reentrancies to discuss another aspect that has not received enough attention in the AMR literature: AMR parsing for other languages and the cross-linguality of the AMR annotation scheme.

5

Cross-lingual AMR Parsing

So far, we have focused only on AMR parsing for English. Annotating new AMR datasets for other languages is an expensive process and requires defining guidelines for each new language. It is therefore reasonable to explore whether we can transfer AMR annotations across languages. The cross-lingual properties of AMR have been the subject of preliminary discussions: the AMR guidelines state that AMR is not an interlingua (Banarescu et al., 2013). Bojar (2014) categorized different kinds of divergences in the annotation between English AMRs and Czech AMRs. Xue et al. (2014) showed that structurally aligning English AMRs with Czech and Chinese AMRs is not always possible but argue that refined annotation guidelines would suffice to resolve some of these cases.

In this chapter, we ask whether it is possible to use the AMR annotated for English sentences as semantic representations for their translations in other languages, as in Figure 5.1, while maintaining good parsing accuracy. We therefore introduce cross-lingual AMR parsing, the task of parsing natural language sentences for languages other than English to AMR graphs annotated for English. The task has two distint purposes: to allow parsing for other languages, and to explore the cross-linguality aspects of AMR.

A trivial way to perform cross-lingual AMR parsing is to use MT to translate the input sentences into English so that an available English AMR parser can be employed. This method only requires translation models between the target languages and English. While we show that this method provides a compelling engineering solution for the problem of parsing AMR for other languages, its performance uniquely depends on translation quality.

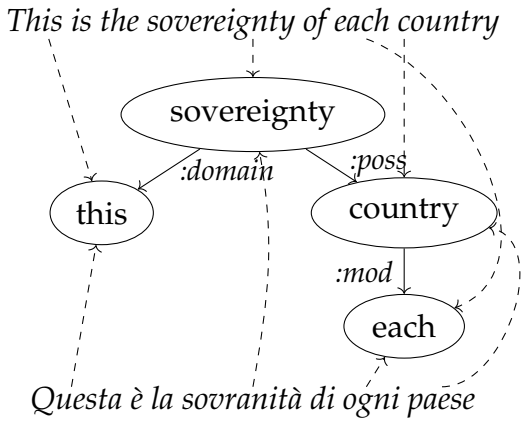


Figure 5.1: AMR alignments for a English sentence and its Italian translation.

To investigate the cross-linguality aspects of AMR, we need to train AMR parsing models for the target languages. Hence, we adapt AMREAGER (Chapter 3) to Italian, Spanish, German and Chinese. To achieve this we use annotation projection, where existing annotations are projected from a source language (English) to a target language through a parallel corpus (e.g., Yarowsky et al., 2001; Hwa et al., 2005; Padó and Lapata, 2009; Evang and Bos, 2016). We refer to parsers for the target languages as *target parsers*. We show that the cross-lingual parsers can be successful even in the presence of translational divergences (Dorr, 1994).

To evaluate the target parsers, similarly to Evang and Bos (2016), we perform SILVER evaluation: we evaluate them on data obtained by parsing the English side of a parallel corpus and projecting the AMR graphs to the target languages. We also propose a novel method that we call FULL-CYCLE evaluation: using the same method used to go from English to the target language, we then go from the target language to English, which we know how to evaluate. To assess the reliability of these evaluation methods, we collect data to perform GOLD evaluation.

Our contributions in this chapter are as follows:

- We introduce the task of cross-lingual AMR parsing and propose two methods that do not require annotated datasets;
- We provide evidence that AMR annotations can be successfully shared across languages, though some translational divergences between languages can be challenging to overcome;

- We propose FULL-CYCLE, a novel method to evaluate non-English AMR parsers when gold annotations in the target languages are missing;
- We release human translations of the test set of LDC2015E86 to Italian, Spanish, German and Chinese. We use the translations to show that FULL-CYCLE approximates GOLD better than SILVER does.

5.1 Task definition

The goal of AMR is to abstract away from the syntactic realization of the original sentences while maintaining its underlying meaning. As a consequence, different phrasings of one sentence are expected to provide identical AMR representations. This canonicalization does not hold across languages: two sentences that express the same meaning in two different languages are not guaranteed to produce identical AMR structures due to translational divergence and language-specific guidelines (Bojar, 2014; Xue et al., 2014). However, Xue et al. (2014) showed that in many cases AMR graphs align well structurally across languages (i.e., their nodes and edges can be aligned). We are encouraged by this finding and argue that it should be possible to develop algorithms that account for some of these differences when they arise. We introduce a new task, which we call *cross-lingual AMR parsing*: given a sentence in any language, the goal is to recover the AMR graph that would have been generated for its English translation. This task is harder than traditional AMR parsing as it requires to recover English labels as well as to deal with structural differences between languages, usually referred to as translation divergence. We use this task as a way to explore the cross-linguality aspects of AMR and see if we can successfully learn models that can overcome the differences between the languages and recover the AMR from a sentence in another language.

5.2 Machine Translation

By definition of the task, the reference AMR used for evaluation is the AMR of the reference English translation. A simple way to address the task is therefore to use MT to translate the sentence in English and then use an available English parser to predict its AMR graph. Naturally, the quality of the output

graph depends on the quality of the translations. If the automatic translation is close to the reference translation, then the predicted AMR graph will be close to the reference AMR graph. The quality of these parses is therefore not an indication of the cross-lingual properties of AMR. However, its simplicity makes it a compelling engineering solution for parsing other languages.

5.3 Annotation Projection

We propose an alternative method where we use annotation projection to train cross-lingual parsers. By means of parallel corpora, we project the AMR annotations from English to other languages. Unfortunately, there are no available parallel corpora which are also annotated with AMR. Hence, we obtain the AMR annotations using an available AMR parser for English, which introduces noise.

In order to train most AMR parsers, we also need to project the AMR alignments between AMR nodes and words in the sentence. Similarly to other annotation projection work (Yarowsky et al., 2001), we use unsupervised word aligners (Dyer et al., 2013) to compute alignments between words in English and words in the target languages.

Our approach depends on the underlying assumption that we make. Let $S = s_1 \dots s_{|s|}$ be the source language sentence and $T = t_1 \dots t_{|t|}$ be the target language sentence; $A_s(\cdot)$ be the AMR alignment mapping word tokens in S to the set of AMR nodes that are triggered by it; $A_t(\cdot)$ be the same function for T; v be a node in the AMR graph; and finally, $W(\cdot)$ be an alignment that maps a word in S to a subset of words in T. Then, if a source word s_i is word-aligned to a target word t_j and it is AMR aligned with an AMR node v, then the target word t_j is also aligned to the AMR node v:

$$\forall i, j, v \;\; t_j \in W(s_i) \wedge v \in A_s(s_i) \Rightarrow v \in A_t(t_j)$$

In the example of Figure 5.1, *Questa* is word-aligned with *This* and therefore AMR-aligned with the node *this*, and the same logic applies to the other aligned words. The words *is*, *the* and *of* do not generate any AMR nodes, so we ignore their word alignments.

5.4 Evaluation

We now turn to the problem of evaluation. There are no available parallel corpora with AMR annotations that we can exploit for evaluation. Hence, to obtain gold evaluation data, we collected professional translations for the 1371 English sentences in the AMR test set of LDC2015E86.[1] We acquired translations to Italian, Spanish, German, and Chinese, which are currently available upon request. We then paired the translated sentences to the original AMR graphs.

We also consider the case where we have no access to gold evaluation data. We explore two different ways to evaluate parsers in such conditions:

- SILVER. We can generate a silver test set by running an (English) AMR parser on the English side of a parallel corpus and use the output AMR graphs as references. However, the silver test set is affected by mistakes made by the English AMR parser.

- FULL-CYCLE. In order to evaluate on a gold test set, we propose an alternative method: after learning the target parser from the English parser, we invert this process to learn a new English parser from the target parser. The resulting English parser is then evaluated against the (English) AMR gold standard. We hypothesize that the score of the new English parser can be used as a proxy for the score of the target parser.

A diagram summarizing the different evaluation stages is shown in Figure 5.2. In the case of MT-based systems, FULL-CYCLE scores are obtained by first translating from English to the target language and then back to English (back-translation), and then parsing the resulting sentences with the English AMR parser.

5.5 Experimental Setup

We run experiments on four languages: Italian, Spanish, German and Chinese. We use Europarl (Koehn, 2005) to obtain parallel corpora for English-Italian, English-Spanish, and English-German. These datasets contain around 1.9M

[1] `https://translated.com`

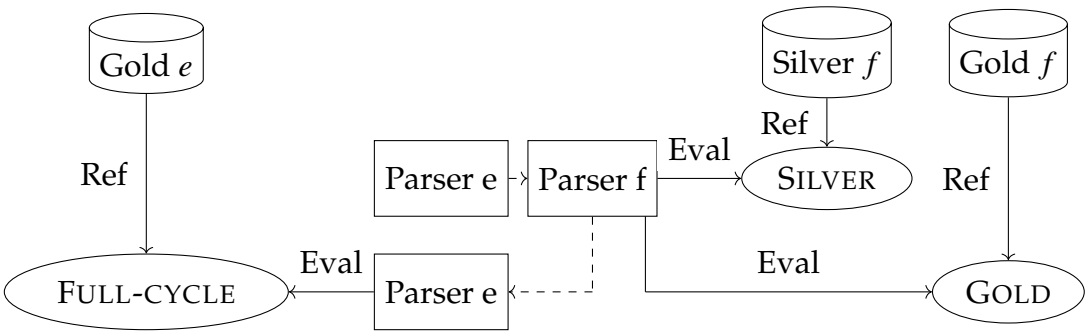


Figure 5.2: Description of SILVER, FULL-CYCLE and GOLD evaluations. e stands for English and f stands for the target (foreign) language. Dashed lines represent the process of transferring learning across languages (e.g. with annotation projection). SILVER uses a parsed parallel corpus as reference ("Ref"), FULL-CYCLE uses the English gold standard (Gold e) and GOLD uses the target language gold standard we collected (Silver f).

sentences for each language pair. For English-Chinese, we use the first 2M sentences from the United Nations Parallel Corpus (Ziemski et al., 2016).

In order to train and evaluate the AMR parsers, for each target language, we extract two parallel datasets of 20,000/2,000/2,000 (train/dev/test) sentences for the two steps of the annotation projection. The first step is English $\rightarrow$ target, for the target parser, and the second step is target $\rightarrow$ English, for FULL-CYCLE evaluation. The projection approach also requires training the word aligner, for which we use all the remaining sentences from the parallel corpora (Europarl for Spanish/German/Italian and UN Parallel Corpus for Chinese). We use the same data to train the MT models. The gold AMR dataset is LDC2015E86, containing 16,833 training sentences, 1,368 development sentences, and 1,371 testing sentences.

Word alignments (i.e., the function $W(\cdot)$) were generated using fast_align (Dyer et al., 2013), while AMR alignments (i.e., the function $A(\cdot)$) were generated with JAMR (Flanigan et al., 2014). AMREAGER (Chapter 3) was chosen as the pre-existing English AMR parser. It requires tokenization, POS tagging, NER tagging and dependency parsing, which for English, German and Chinese are provided by CoreNLP (Manning et al., 2014). We use Freeling (Carreras et al., 2004) for Spanish, as CoreNLP does not provide dependency parsing for this language. Italian is not supported in CoreNLP: we use Tint (Aprosio and Moretti, 2016), a CoreNLP-compatible NLP pipeline for Italian.

For the translation approach, we experimented with different translation systems. Google Translate[2], which has access to a much larger training corpus, Moses (Koehn et al., 2007), a Statistical MT (SMT) toolkit, and Nematus (Sennrich et al., 2017), an NMT toolkit. We train Moses and Nematus with the same training data we use for the projection method and default hyper-parameters.

The original English parser achieves 65% Smatch score on the test split of LDC2015E86. The FULL-CYCLE and GOLD evaluation methods use the same dataset. SILVER is performed on the 2,000 sentences reserved for testing, as discussed above.

5.6 Results

The parsing results are shown in Table 5.1. The Google Translate (MT-GOOGLE) system outperforms all other systems, but is not directly comparable to them, as it has the advantage of being trained on a much larger dataset. The BLEU scores of all translation systems are shown in Table 5.2.

There are several sources of noise in the PROJECTION method: 1) the parsers are trained on noisy AMR graphs, obtained by an automatic parser for English; 2) the projection uses noisy word alignments; 3) the AMR alignments on the source side are also noisy; 4) translation divergences exist between the languages, making it sometimes difficult to project the annotation without loss of information. Nevertheless, the PROJECTION parsers allow us to investigate the cross-linguality aspects of AMR, as we will discuss in the next section.

5.7 Qualitative Analysis

We first look at the graphs produced by the PROJECTION parsers to observe the overall quality of producing cross-lingual AMR representations. We then focus our attention on known translational divergences that may pose a problem for cross-lingual AMR parsing.

[2] `https://translate.google.com/toolkit`.

Language	Method	GOLD	SILVER	FULL-CYCLE
Italian	PROJECTION	43	45	45
	MT-SMT	52	51	51
	MT-NMT	43	49	41
	MT-GOOGLE	58	52	59
Spanish	PROJECTION	42	44	44
	MT-SMT	53	53	51
	MT-NMT	43	51	42
	MT-GOOGLE	60	56	60
German	PROJECTION	39	45	43
	MT-SMT	49	50	49
	MT-NMT	38	47	39
	MT-GOOGLE	57	54	59
Chinese	PROJECTION	35	45	32
	MT-SMT	42	57	48
	MT-NMT	39	57	40
	MT-GOOGLE	50	64	55

Table 5.1: SILVER, GOLD and FULL-CYCLE Smatch scores for projection-based (PROJECTION), MT with Moses (MT-SMT), MT with Nematus (MT-NMT), and MT with Google Translate (MT-GOOGLE).

Model	SMT	NMT	Google
EN-IT	23.83	21.27	61.31
IT-EN	23.74	19.77	42.20
EN-ES	29.00	26.14	78.14
ES-EN	27.66	21.63	50.78
EN-DE	15.47	15.74	63.48
DE-EN	21.50	14.96	41.78
EN-ZH	9.19	8.67	26.75
ZH-EN	10.81	10.37	22.21

Table 5.2: BLEU scores for Moses (SMT), Nematus (NMT), and Google Translate (Google) on the (out-of-domain) LDC2015E86 test set

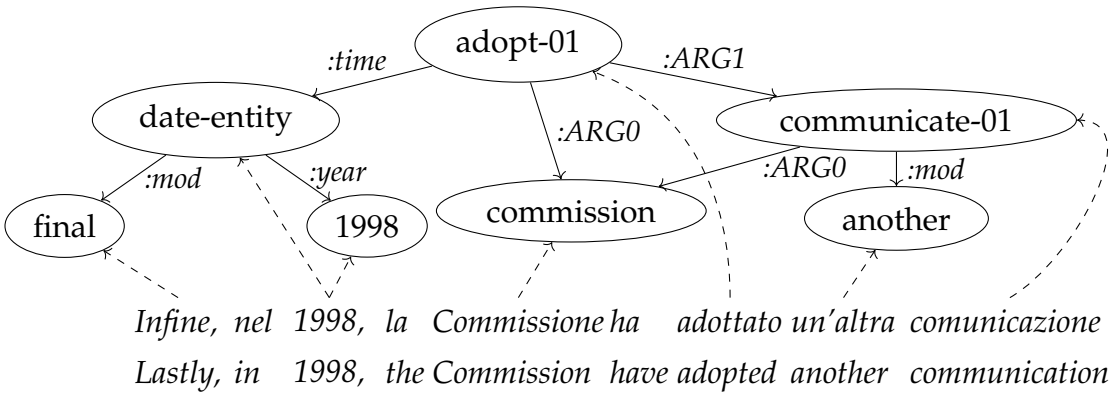


Figure 5.3: Parsed AMR graph and alignments (dashed lines) for the Italian translation of *Lastly, in 1998, the Commission adopted a further communication*.

5.7.1 Manual Inspection

We note that most errors involve concept identification, that is the task of predicting the nodes in the AMR graphs. In the Italian example of Figure 5.3, the only evident error is that *Infine* (*Lastly*) should not trigger the node *final* in the graph. In the Spanish example of Figure 5.4, the word *medida* (*measure*) is incorrectly ignored: it should be used to generate a child of the node *impact-01*. Some of the *:ARG* roles are also not correct. In the German example of Figure 5.5, *meines* (*my*) should reflect the fact that the speaker is talking about his own country. Finally, in the Chinese example of Figure 5.6, there are several mistakes including yet another concept identification mistake: the *intend-01* node is erroneously added to the graph.

We argue that concept identification mistakes are often due to the problem of noisy alignments, discussed in Section 5.6. The parsers learn what words are likely to trigger a node in the AMR — we refer to those as content-bearing words — by looking at their AMR alignments. These are induced by the word alignments and the original English-AMR alignments. Accurate alignments are therefore crucial in order to achieve good parsing results.

To study the extent of the problem, we computed the percentage of words in the training data that are learned to be non-content-bearing (Table 5.3). We found that the Chinese parser, which is our least accurate parser, is the one that most suffers from this, with almost 42% non-content-bearing words. On the other hand, in the German parser, which is the highest scoring, less than 26% of the words are non-content-bearing, which is the lowest percentage amongst

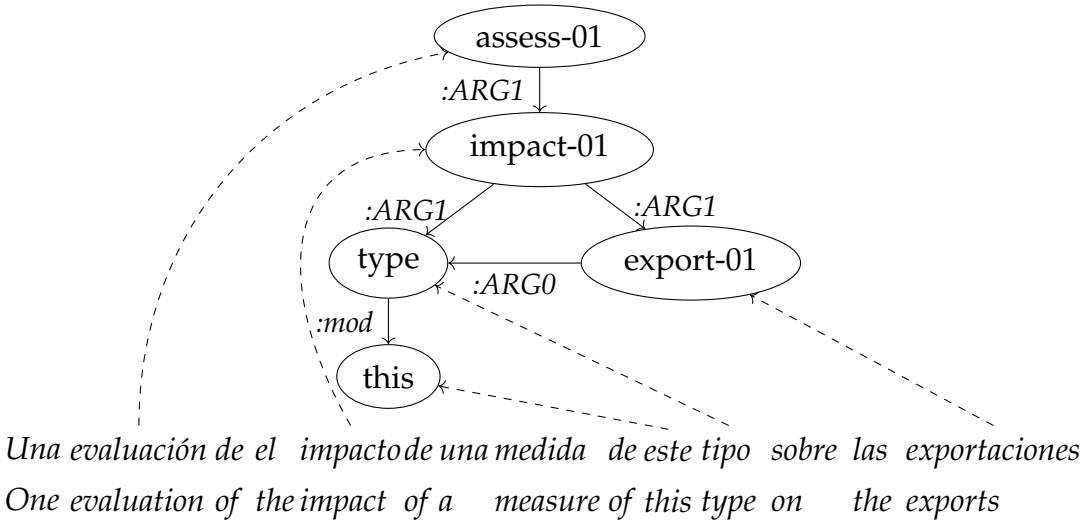


Figure 5.4: Parsed AMR graph and alignments (dashed lines) for the Spanish translation of *A study into the impact of such a measure on the export*.

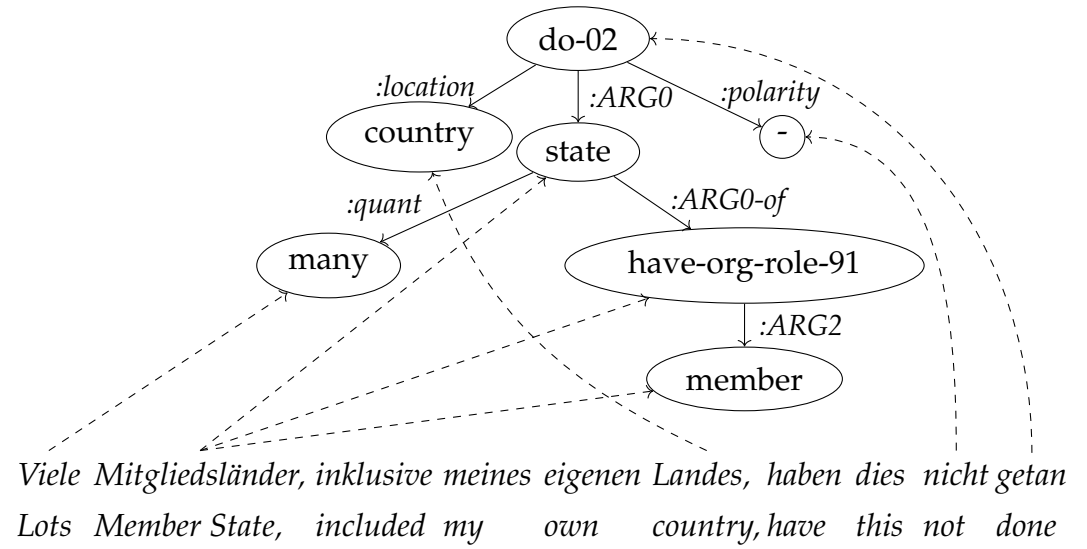


Figure 5.5: Parsed AMR graph and alignments (dashed lines) for the German translation of *Many Member States, including my own, did not do as required*.

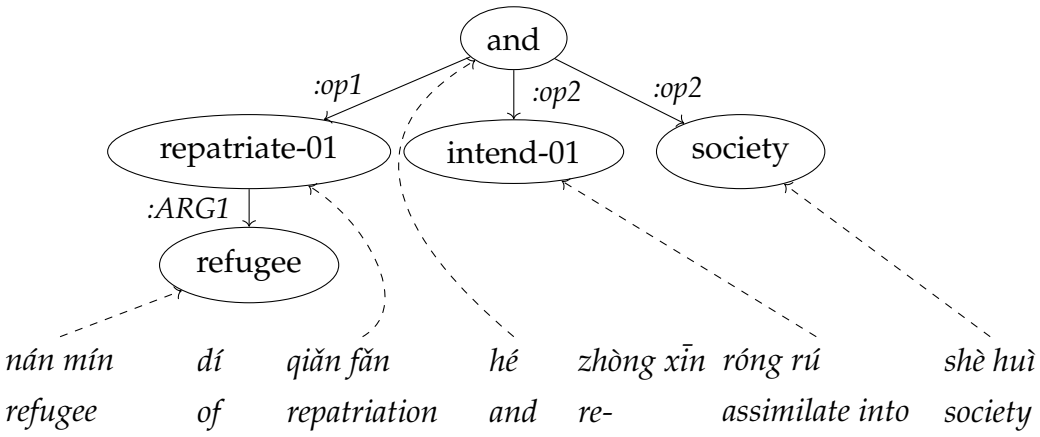


Figure 5.6: Parsed AMR graph and alignments (dashed lines) for the Chinese translation of *Repatriation and reintegration of Malian refugees*.

Language	**%**
EN	9.85
IT	28.93
ES	28.32
DE	25.53
ZH	41.85

Table 5.3: Percentage of words predicted as non-content-bearing in each parser.

all non-English parsers. It appears that the percentage of tokens that trigger nodes has an impact on parser performance. To achieve better performance, parsers for other languages should aim to achieve a percentage of non-content-bearing words closer to the one for English, which is less than 10%.

5.7.2 Translational Divergence

We now turn to the hypothesis that AMR can, to some extent, be shared across languages. We look at translational divergence and discuss how it affects parsing, following the classification used in previous work (Dorr, 1994; Dorr et al., 2002; Sulem et al., 2015).

Categorical This divergence happens when two languages use different POS tags to express the same meaning. For example, the English sentence *I am*

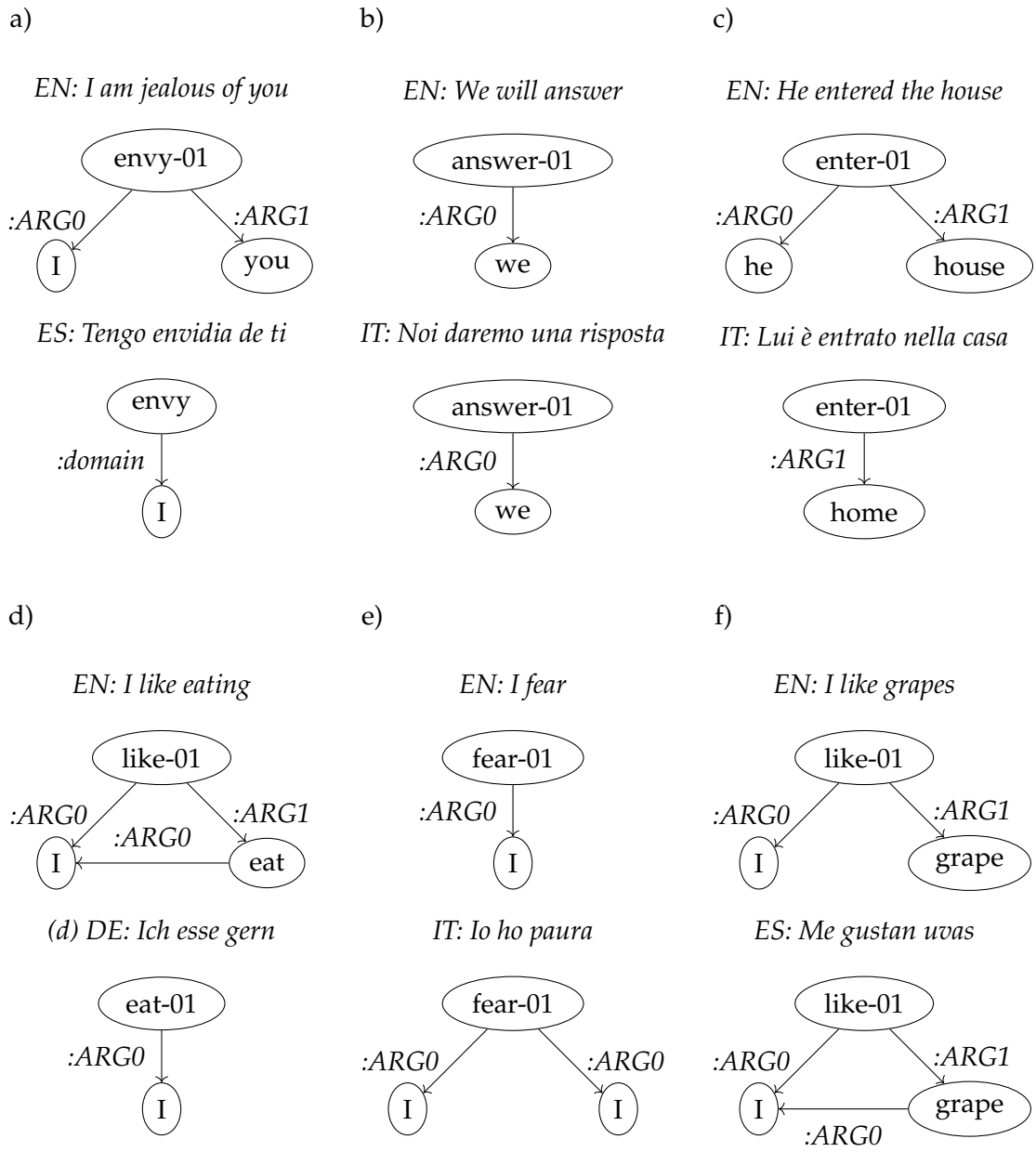


Figure 5.7: Parsing examples in several languages involving common translational divergence phenomena: (a) contains a categorical divergence, (b) and (e) conflational divergences, (c) a structural divergence, (d) an head swapping and (f) a thematic divergence. For each example, we report the gold AMR for the English sentence (top) as well as the parsed AMR for the target language (bottom).

jealous of you is translated into Spanish as *Tengo envidia de ti* (*I have jealousy of you*). The English adjective *jealous* is translated in the Spanish noun *envidia*. In Figure 5.7a we note that the parser, while making other mistakes, correctly recognized that *envidia* (*jealousy/envy*) should be used as the predicate, regardless of its POS tag.

Conflational This divergence happens when verbs expressed with a single word in a language can be expressed with more words in another language. Two subtypes are distinguished: *manner* and *light verb*.

Manner refers to a manner verb that is mapped to a motion verb plus a manner-bearing word. For example, *We will answer* is translated in the Italian sentence *Noi daremo una riposta* (*We will give an answer*), where *to answer* is translated as *daremo una risposta* (*will give an answer*). Figure 5.7b shows that the Italian parser generates the correct AMR for this sentence by creating a single node labeled *answer-01* for the expression *dare una riposta*.

In a light verb conflational divergence, a verb is mapped to a light verb plus an additional meaning unit, such as when *I fear* is translated as *Io ho paura* (*I have fear*) in Italian: *to fear* is mapped to the light verb *ho* (*have*) plus the noun *paura* (*fear*). Figure 5.7e shows that also this divergence is dealt properly by the Italian parser: *ho paura* correctly triggers the root *fear-01*.

Structural This divergence happens when verb arguments result in different syntactic configurations, for example, due to an additional PP attachment. When translating *He entered the house* with *Lui è entrato nella casa* (*He entered in the house*), the Italian translation has an additional *in* preposition. Figure 5.7c shows that, regardless of the missing preposition, the parse contains a *ARG1* role between the node for *entrato* (*entered*) and the node for *casa* (*house/home*). The missing node *he* is due to pronoun-dropping (the pronoun can be omitted). The parser could not learn to align *Lui* with the node *He* because, in the training data, the pronoun *Lui* is often omitted in sentences corresponding to AMR graphs containing the node *He*.

Head Swapping This divergence occurs when the direction of the dependency between two words is inverted. For example, *I like eating*, where *like* is head of *eating*, becomes *Ich esse gern* (*I eat likingly*) in German, where the

dependency is inverted. In this case, the German parser does not cope well with this divergence: it is unable to recognize *like-01* as the main concept in the sentence, as shown in Figure 5.7d.

Thematic Finally, the parse in Figure 5.7f has to deal with a thematic divergence, which happens when the semantic roles of a predicate are inverted. In the sentence *I like grapes*, translated to Spanish as *Me gustan uvas*, *I* is the subject in English while *Me* is the object in Spanish. Even though we note an erroneous reentrant edge between *grape* and *I*, the thematic divergence does not create problems: the parser correctly recognizes the *:ARG0* relationship between *like-01* and *I* and the *:ARG1* relationship between *like-01* and *grape*.

5.7.3 Discussion

As mentioned in Section 5.2, the MT-based systems do not help answer the question of cross-linguality of AMR, and we instead focus on the projection-based parsers. Qualitative analysis showed that the parsers can overcome at least some of the translational divergences. However, we also showed an example of head swapping where the parsers did not handle the divergence successfully. We speculate that concept identification must be more accurate to provide good cross-lingual parsing results. We further argue that the suboptimal performance of the parsers in terms of Smatch scores is due to the many sources of noise in the annotation projection approach rather than the instability of AMR across languages. We found that the Chinese parser is the one that most suffer from noisy alignments, which explains its lower performance compared to the other languages. We provide evidence that cross-lingual AMR parsing is feasible. We hope that the release of the gold standard test sets will motivate further work in this direction.

5.7.4 Analysis of Evaluation Methods

We computed the Pearson correlation coefficients for the Smatch scores of Table 5.1 to determine how well SILVER and FULL-CYCLE correlate with GOLD. FULL-CYCLE correlates better than SILVER: the Pearson coefficient is 0.95 for FULL-CYCLE and 0.47 for SILVER. Figure 5.8 shows linear regression lines, where it is easy to note the closer relationship between FULL-CYCLE and GOLD.

Unlike SILVER, FULL-CYCLE uses the same dataset as GOLD and gold AMR graphs as references, which makes it more reliable than SILVER. Interestingly, if we ignore the scores obtained for Chinese, the correlation between SILVER and GOLD dramatically increases, indicative of the lower performance of the Chinese pipeline compared to the other languages: the Pearson coefficient becomes 0.97 for FULL-CYCLE and 0.87 for SILVER.

A good proxy for GOLD should rank different systems similarly. To test the ranking ability of the evaluation methods, we use the Kendall-tau score (Kendall, 1945), a measure for the similarity between two permutations. We extracted the rankings from Table 1 and computed its Kendall-tau scores. The results further confirm that FULL-CYCLE approximate GOLD better than SILVER does: the score is 0.40 for SILVER and 0.82 for FULL-CYCLE.

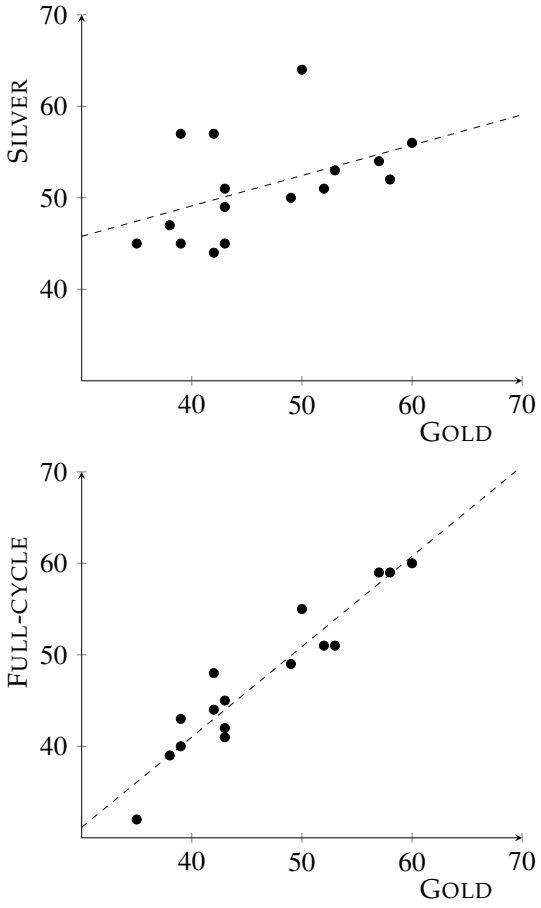


Figure 5.8: Linear regression lines for SILVER and FULL-CYCLE.

5.8 Related Work

AMR parsing for languages other than English has made only a few steps forward. In previous work (Li et al., 2016; Xue et al., 2014; Bojar, 2014), nodes of the target graph were labeled with either English words or with words in the target language. We instead use the AMR annotation used for English for the target language as well, without translating any word. To the best of our knowledge, the only previous work that attempts to parse AMR graphs for non-English sentences automatically is by Vanderwende et al. (2015). Sentences in several languages (French, German, Spanish and Japanese) are parsed into a logical representation, which is then converted to AMR using a small set of rules. A comparison with this work is difficult, as the authors do not report results for the parsers (due to the lack of an annotated corpus) or release their code.

Besides AMR, other semantic parsing frameworks for non-English languages have been investigated (Hoffman, 1992; Cinková et al., 2009; Gesmundo et al., 2009; Evang and Bos, 2016). The system by Evang and Bos (2016) is the most closely related to our work. They used a projection mechanism similar to ours for CCG. The main difference is that, in order to project CCG parse trees to the target languages, only literal translations were used. Previous work has also focused on assessing the stability across languages of semantic frameworks such as AMR (Xue et al., 2014; Bojar, 2014), UCCA (Sulem et al., 2015) and Propbank (van der Plas et al., 2010).

Cross-lingual techniques can cope with the lack of labeled data on languages when this data is available in at least one language, usually English. The annotation projection method, which we follow in this work, is one way to address this problem. It was introduced for POS tagging, base noun phrase bracketing, NER tagging, and inflectional morphological analysis (Yarowsky et al., 2001) but it has also been used for dependency parsing (Hwa et al., 2005), role labeling (Padó and Lapata, 2009; Akbik et al., 2015) and semantic parsing (Evang and Bos, 2016). Another common thread of cross-lingual work is model transfer, where parameters are shared across languages (Zeman and Resnik, 2008; Cohen and Smith, 2009; Cohen et al., 2011a; McDonald et al., 2011; Søgaard, 2011).

Our PROJECTION parser has since been used as a baseline for a rule-based

Brazilian Portuguese AMR parser (Anchiêta and Pardo, 2018).

5.9 Summary

In this chapter, we introduced the task of parsing AMR structures, annotated for English, from sentences written in other languages. We devised the task as a way to test the cross-lingual properties of AMR. We provided evidence that AMR can be shared across the languages tested but that it may be challenging to overcome some translational divergences. The multilingual parser is available at `http://www.github.com/mdtux89/amr-eager-multilingual` and a demo is available at `http://cohort.inf.ed.ac.uk/amreager.html`. The results indicate that there is room for improvements, especially in terms of generating better alignments. We encourage further work in this direction by releasing professional translations of the AMR test set into four languages. Notably, recent state-of-the-art AMR parsers (Lyu and Titov, 2018; Zhang et al., 2019) do not rely on automatically generated alignments, therefore reducing the noise that our annotation projection method would introduce.

We further proposed a novel way to evaluate the target parsers that does not require manual annotations of the target language. The FULL-CYCLE procedure, which we show to correlate well with GOLD evaluation, is not limited to AMR parsing and could be used for other cross-lingual problems in NLP.

So far, we focussed on the task of AMR parsing. In the next chapter, we turn to AMR-to-text generation, where natural language sentences are generated from AMR graphs, and investigate the importance of explicitly encoding reentrancies and other structural information.

6

AMR Generation with Structured Neural Encoders

In the previous chapters, we looked at AMR parsers that convert sentences into AMR graphs. Downstream NLP applications such as summarization and MT also require the ability to generate language. Hence, in this chapter, we look at AMR-to-text generation, the task of converting AMR graphs into text.

As discussed in Chapter 2, attentive encoder/decoder architectures commonly used for NMT have been explored for this task. Konstas et al. (2017) linearized AMR graphs to sequences in order to use sequence-to-sequence architectures (Bahdanau et al., 2015). The linearization process loses reentrancy information. Graph encoders, which do not discard reentrancies, were later shown to yield better results (Song et al., 2018; Beck et al., 2018). When the AMR annotations do not contain reentrancies, they can be encoded as trees rather than graphs. A comparison between tree and graph encoders can therefore shed lights on the impact of reentrancies on AMR-to-text generation. Figure 6.1 shows an example of an AMR and its sequential, tree, and graph representations.

In this chapter, we compare three types of encoders for AMR: 1) sequential encoders, which reduce AMR graphs to sequences and ignore reentrancies; 2) tree encoders, which consider structural information but still ignores reentrancies; and 3) graph encoders, which include reentrancies. As in the rest of this thesis, we pay particular attention to reentrancies: we investigate whether explicitly encoding them results in better generation results. We further investigate the impact of long-range dependencies in the AMR graph, which are

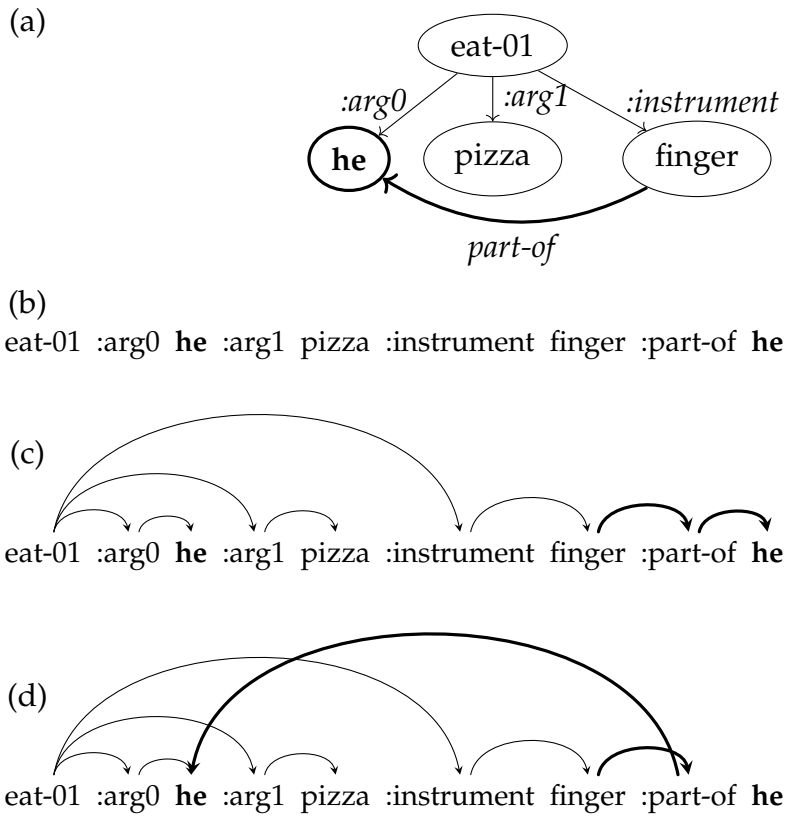


Figure 6.1: (a) AMR for the sentence *He ate the pizza with his fingers* and different input representations: (b) sequential; (c) tree-structured; (d) graph-structured. The nodes and edges in bold highlight a reentrancy.

also expected to benefit from structural encoding.

Our contributions in this chapter are as follows:

- We present structural encoders for the encoder/decoder framework and show the benefits of graph encoders not only compared to sequential encoders but also compared to tree encoders;

- We show that better treatment of reentrancies and long-range dependencies contributes to improvements in the graph encoders.

Our best model, based on a graph encoder, improves on previous results for both the LDC2015E86 dataset (24.40 on BLEU and 23.79 on Meteor) and the LDC2017T10 dataset (24.54 on BLEU and 24.07 on Meteor).

6.1 Input Representations

In this section, we describe in detail the difference between encoding AMR as a graph, tree, and sequence.

6.1.1 Graph-structured AMRs

AMRs are normally represented as labeled and directed graphs:

$$
\begin{aligned}
&G_0 = (V_0, E_0, L),\\
&V_0 = \{v_1, v_2, \ldots, v_n\},\\
&E_0 \subseteq V_0 \times L \times V_0,\\
&L = \{\ell_1, \ell_2, \ldots, \ell_l\},
\end{aligned}
$$

where V_0 are the graph vertices (or nodes), E_0 are the graph edges, and L is the set of edge labels. Each edge $e \in E_0$ is a triple: $e = (i, \ell, j)$, where $i \in V_0$ is the parent node, $\ell \in L$ is the edge label and $j \in V_0$ is the child node.

In order to obtain unlabeled edges, thus decreasing the total number of parameters required by the models, we replace each labeled edge $e = (i, \ell, j)$ with two unlabeled edges: $e'_1 = (i, \ell), e'_2 = (\ell, j)$:

$$
\begin{aligned}
&G = (V, E),\\
&V = V_0 \cup L = \{v_1, \ldots, v_n, \ell_1, \ldots, \ell_l\},\\
&E \subseteq (V_0 \times L) \cup (L \times V_0).
\end{aligned}
$$

Each unlabeled edge $e' \in E$ is a pair: $e' = (i, j)$, where one of the following holds:

1. $i \in V_0$ and $j \in L$;

2. $i \in L$ and $j \in V_0$.

For instance, the edge between *eat-01* and *he* with label *:arg0* in Figure 6.1(a) is replaced by two edges in Figure 6.1(d): an edge between *eat-01* and *:arg0* and another one between *:arg0* and *he*. The process, also used in Beck et al. (2018), tranforms the input graph into its equivalent Levi graph (Levi, 1942).

6.1.2 Tree-structured AMRs

In order to obtain tree structures, it is necessary to discard the reentrancies from the AMR graphs. Similarly to Takase et al. (2016), we replace nodes with $k > 1$ incoming edges with k identically labeled nodes, each with a single incoming edge.

6.1.3 Sequential AMRs

Following Konstas et al. (2017), the input AMR graphs is a linearized into a sequence:

$$x = x_1, \ldots, x_N,$$
$$x_i \in V.$$

The depth-first traversal of the graph defines the indexing between nodes and tokens in the sequence. For instance, the root of the graph is x_1, its leftmost child is x_2 and so on. Nodes with multiple parents are visited more than once. At each visit, their labels are repeated in the sequence, effectively losing reentrancy information, as shown in Figure 6.1(b).

6.2 Encoders

In this section, we review the encoders adopted as building blocks for our encoders.

6.2.1 Recurrent Neural Network Encoders

We reimplement the sequential encoder by Konstas et al. (2017), where the sequential linearization is the input to a bidirectional LSTM (BiLSTM; Graves et al. 2013) network. The hidden state of the BiLSTM at step i is used as a context-aware word representation of the i-th token in the sequence:

$$o_{1:N} = \text{BiLSTM}(x_{1:N}),$$

where $x_{1:N}$ denotes the sequence $x_1, \ldots, x_N$, $o_i \in \mathbb{R}^d$, $o_{1:N}$ denotes the sequence $o_1, \ldots, o_N$, and d is the size of the output embeddings.

6.2.2 TreeLSTM Encoders

Tree-Structured Long Short-Term Memory Networks (TreeLSTM; Tai et al. 2015) have been introduced primarily as a way to encode the hierarchical structure of syntactic trees (Tai et al., 2015). However, they have also been applied to AMR for the task of headline generation (Takase et al., 2016). TreeLSTMs assume tree-structured input, hence reentrancies must be removed.

We use the Child-Sum variant introduced by Tai et al. (2015), which processes the tree in a bottom-up pass. When visiting a node, the hidden states of its children are summed up in a single vector which is then passed into recurrent gates.

In order to use information from both incoming and outgoing edges, we employ bidirectional TreeLSTMs (Eriguchi et al., 2016), where the bottom-up pass is followed by a top-down pass. The top-down state of the root node is obtained by feeding the bottom-up state of the root node through a feed-forward layer:

$$h^{\downarrow}_{\text{root}} = \tanh(W_r h^{\uparrow}_{\text{root}} + b),$$

where $h^{\uparrow}_i$ is the hidden state of node $x_i \in V$ for the bottom-up pass and $h^{\downarrow}_i$ is the hidden state of node x_i for the top-down pass.

The top-down states for the other nodes are computed by feeding the bottom-up state of each node $h^{\uparrow}_i$ into an LSTM, with the cell state given by the top-down state of its parent node $h^{\downarrow}_{p(i)}$:

$$h^{\downarrow}_i = \text{LSTM}(h^{\downarrow}_{p(i)}, h^{\uparrow}_i),$$

where $p(i)$ is the parent of node x_i in the tree. The final hidden states are obtained by concatenating the states from the bottom-up pass and the top-down pass:

$$h_i = \left[h^{\downarrow}_i; h^{\uparrow}_i\right].$$

The hidden state of the root node is usually used as a representation of the entire tree. In order to use attention over all nodes, as in traditional NMT (Bahdanau et al., 2015), we need to build embeddings for each node in the AMR. We do so by extracting the hidden states of each node in the tree:

$$o_{1:N} = h_{1:N},$$

where $o_i \in \mathbb{R}^d$, d is the size of the output embeddings.

The encoder is related to the TreeLSTM encoder by Takase et al. (2016), which however encodes labeled trees and does not use a top-down pass.

6.2.3 Graph Convolutional Network Encoders

Graph Convolutional Network (GCN; Duvenaud et al. 2015; Kipf and Welling 2017) is a neural network architecture that learns embeddings of nodes in a graph by looking at its nearby nodes. In NLP, GCNs have been used for Semantic Role Labeling (Marcheggiani and Titov, 2017), NMT (Bastings et al., 2017), Named Entity Recognition (Cetoli et al., 2017) and text generation (Marcheggiani and Perez-Beltrachini, 2018).

A graph-to-sequence neural network was first introduced by Xu et al. (2018). The authors review the similarities between their approach, GCN and another approach, based on GRUs (Li et al., 2015). The latter recently inspired a graph-to-sequence architecture for AMR-to-text generation (Beck et al., 2018). Song et al. (2018) also proposed a graph encoder based on LSTMs.

The architectures of Song et al. (2018) and Beck et al. (2018) are both based on the same core computation of a GCN, which sums over the embeddings of the immediate neighborhood of each node:

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W_{(j,i)}^{(k)} h_j^{(k)} + b^{(k)}\right),$$

where $h_i^{(k)}$ is the embeddings of node $x_i \in V$ at layer k, σ is a non-linear activation function, $\mathcal{N}(i)$ is the set of the immediate neighbors of x_i, $W_{(j,i)}^{(k)} \in \mathbb{R}^{m \times m}$ and $b^{(k)} \in \mathbb{R}^m$, with m being the size of the embeddings.

It is possible to use recurrent networks to model the update of the node embeddings. Specifically, Beck et al. (2018) used a GRU layer where the gates are modeled as GCN layers. Song et al. (2018) did not use the activation function σ and perform an LSTM update instead.

The systems of Song et al. (2018) and Beck et al. (2018) further differ in design and implementation decisions such as in the use of edge label and edge directionality. Throughout the rest of the chapter, we follow the traditional, non-recurrent, implementation of GCN also adopted in other NLP tasks (Marcheggiani and Titov, 2017; Bastings et al., 2017; Cetoli et al., 2017). In our

experiments, the node embeddings are computed as follows:

$$h_i^{(k+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} W_{\mathrm{dir}(j,i)}^{(k)} h_j^{(k)} + b^{(k)}\right), \tag{6.1}$$

where $\mathrm{dir}(j,i)$ indicates the direction of the edge between x_j and x_i (i.e., outgoing or incoming edge). The hidden vectors from the last layer of the GCN network are finally used to represent each node in the graph:

$$o_{1:N} = h_{1:N}^{(K)},$$

where K is the number of GCN layers used, $o_i \in \mathbb{R}^d$, d is the size of the output embeddings.

To regularize the models we apply dropout (Srivastava et al., 2014) as well as edge dropout (Marcheggiani and Titov, 2017). We also include highway connections (Srivastava et al., 2015) between GCN layers.

While GCNs are intended to encode graphs, they can also be applied to the trees obtained by removing reentrancies from the input graphs. In the experiments of Section 6.4, we explore GCN-based models both as graph encoders (reentrancies are maintained) as well as tree encoders (reentrancies are ignored).

6.3 Stacking Encoders

We aimed at stacking the explicit source of structural information provided by TreeLSTM layers and GCN layers with the sequential information which BiLSTM layers extract well. This was shown to be effective for other tasks with both TreeLSTMs (Eriguchi et al., 2016; Chen et al., 2017) and GCNs (Marcheggiani and Titov, 2017; Cetoli et al., 2017; Bastings et al., 2017).

In previous work, the structural encoders (tree or graph) were used on top of the BiLSTM network: first, the input is passed through the sequential encoder, the output of which is then fed into the structural encoder. While we experiment with this approach, we also propose an alternative solution where the BiLSTM network is used on top of the structural encoder: the input embeddings are refined by exploiting the explicit structural information given by the graph. The refined embeddings are then fed into the BiLSTM networks. See Figure 6.2 for a graphical representation of the two approaches. In our experiments, we found the latter approach to be more effective.

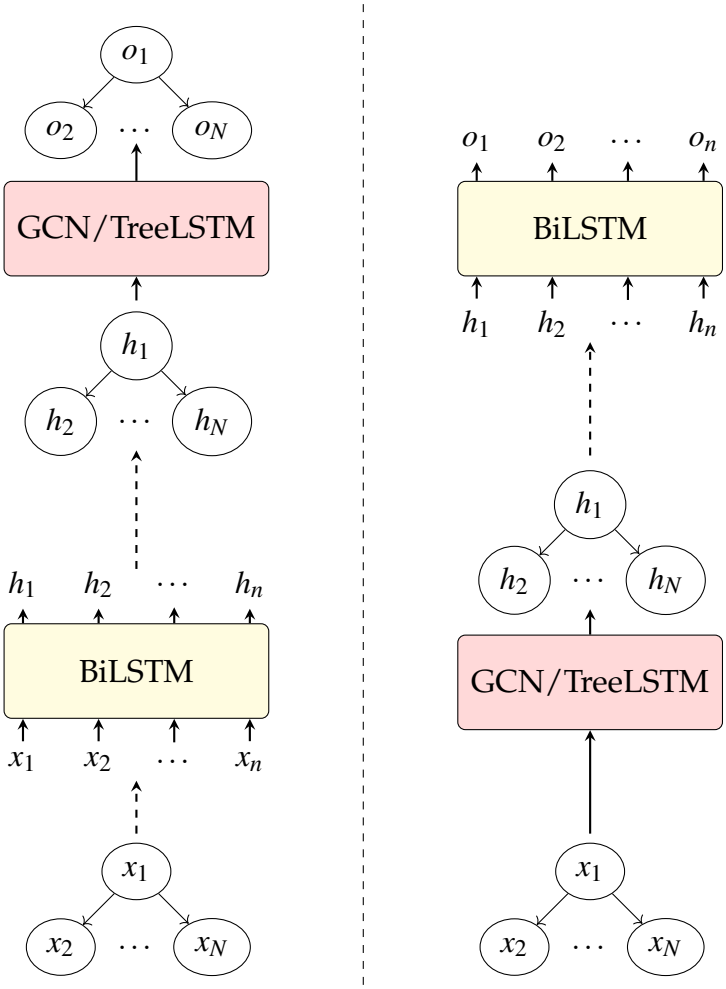


Figure 6.2: Two ways of stacking recurrent and structural models. Left side: structure on top of sequence, where the structural encoders are applied to the hidden vectors computed by the BiLSTM. Right side: sequence on top of structure, where the structural encoder is used to create better embeddings which are then fed to the BiLSTM. The dotted lines refer to the process of converting the graph into a sequence or vice-versa.

Compared to models that interleave structural and recurrent components such as the systems of Song et al. (2018) and Beck et al. (2018), stacking the components allows us to test for their contributions more easily.

6.3.1 Structure on Top of Sequence

In this setup, BiLSTMs are used as in Section 6.2.1 to encode the linearized AMR. The context provided by the BiLSTM is a sequential one. We then apply either GCN or TreeLSTM on the output of the BiLSTM, by initializing the

GCN or TreeLSTM embeddings with the BiLSTM hidden states. We call these models SEQGCN and SEQTREELSTM.

6.3.2 Sequence on Top of Structure

We also propose a different approach, by swapping the order of the BiLSTM and the structural encoder. We use the structured information provided by the AMR graph as a way to refine the original word embeddings. To achieve this, we first apply the structural encoder to the input graphs. The GCN or TreeLSTM representations are then fed into the BiLSTM network, which provides the final encoding. We call these models GCNSEQ and TREELSTMSEQ.

The motivation behind this approach is that we know that BiLSTM networks, given appropriate input embeddings, are very effective at encoding the input sequences. In order to exploit their strength, we do not amend their output but rather provide them with better input embeddings.

6.4 Experiments

We use both BLEU (Papineni et al., 2002) and Meteor (Banerjee and Lavie, 2005) as evaluation metrics.[1] We report results on LDC2015E86 and LDC2017T10, as relevant previous work report results on either dataset. Following Konstas et al. (2017), we anonymize the input AMR graphs. Anonymization removes names and rare words with coarse categories to reduce data sparsity.[2] All systems are implemented in PyTorch (Paszke et al., 2017) using the OpenNMT-py framework (Klein et al., 2017). Hyperparameters of each model were tuned on the development set of LDC2015E86. For the GCN components, we use two layers, ReLU activations, and tanh highway layers. We use single-layer LSTM networks. We train with SGD with the initial learning rate set to 1 and decay set to 0.8. Batch size is set to 100.

We first evaluate the overall performance of the models, after which we focus on two phenomena that we expect to benefit most from structural encoders: reentrancies and long-range dependencies. Table 6.1 shows the comparison on the development split of the LDC2015E86 dataset between sequen-

[1] We used the evaluation script available at `https://github.com/sinantie/NeuralAmr`.

[2] An alternative to anonymization which we did not explore is to employ a copy mechanism (Gulcehre et al., 2016), where the models learn to copy rare words from the input itself.

Input	Model	BLEU	Meteor
Seq	SEQ	21.40	22.00
Tree	SEQTREELSTM	21.84	22.34
	TREELSTMSEQ	22.26	22.87
	TREELSTM	22.07	22.57
	SEQGCN	21.84	22.21
	GCNSEQ	**23.62**	**23.77**
	GCN	15.83	17.76
Graph	SEQGCN	22.06	22.18
	GCNSEQ	**23.95**	**24.00**
	GCN	15.94	17.76

Table 6.1: BLEU and Meteor (%) scores on the development split of LDC2015E86.

tial, tree and graph encoders. The sequential encoder (SEQ) is a re-implementation of Konstas et al. (2017). We test both approaches of stacking structural and sequential components: structure on top of sequence (SEQTREELSTM and SEQGCN), and sequence on top of structure (TREELSTMSEQ and GCNSEQ). To inspect the effect of the sequential component, we run ablation tests by removing the RNNs altogether (TREELSTM and GCN). GCN-based models are used both as tree encoders (reentrancies are removed) and graph encoders (reentrancies are maintained). TreeLSTM-based models are only used as tree encoders.

For both TreeLSTM-based and GCN-based models, we achieve better results with our proposed approach of applying sequential encoding on top of structural encoding. This is more evident for GCN-based models. We also note a drastic drop in performance when the RNN is removed, highlighting the importance of including a sequential component. On the other hand, RNN layers seem to have less impact on TreeLSTM-based models. This outcome is not unexpected, as TreeLSTMs already include LSTM gates.

The results show a clear advantage of tree and graph encoders over the sequential encoder. The best performing model is GCNSEQ, both as a tree and as a graph encoder, with the latter obtaining the highest results.

Table 6.2 shows results on the test set of LDC2015E86 and LDC2017T10.

Model	BLEU	Meteor
LDC2015E86		
SEQ	21.43	21.53
TREE	23.93	23.32
GRAPH	**24.40**	**23.60**
Konstas et al. (2017)	22.00	-
Song et al. (2018)	23.30	-
LDC2017T10		
SEQ	22.19	22.68
TREE	24.06	23.62
GRAPH	**24.54**	**24.07**
Beck et al. (2018)	23.30	-

Table 6.2: Scores on the test split of LDC2015E86 and LDC2017T10. TREE is the tree-based GCNSEQ and GRAPH is the graph-based GCNSEQ.

We report results for our best sequential (SEQ), tree (GCNSEQ without reentrancies, henceforth called TREE) and graph encoders (GCNSEQ with reentrancies, henceforth called GRAPH). We also include previous results reported on these datasets for sequential encoding (Konstas et al., 2017) and graph encoding (Song et al., 2018; Beck et al., 2018).[3] To mitigate the effects of random seeds, we train five models with different random seeds and report the results of the median model, according to their BLEU score on the development set (Beck et al., 2018). We outperform previous work with both tree and graph encoders, demonstrating the efficacy of our GCNSEQ approach. The differences between our graph encoder and that of Song et al. (2018) and Beck et al. (2018) were discussed in Section 6.2.3. The results demonstrate the benefit of structural encoders over purely sequential ones. They also highlight the advantage of explicitly including reentrancies, as graph encoders always outperform tree encoders.

# reentrancies	# dev sents.	# test sents.
0	619	622
1-5	679	679
6-20	70	70

Table 6.3: Counts of reentrancies in the development and test split of LDC2017T10

Model	Number of reentrancies		
	0	1-5	6-20
SEQ	42.94	31.64	23.33
TREE	+0.63	+1.41	+0.76
GRAPH	**+1.67**	**+1.54**	**+3.08**

Table 6.4: Differences, with respect to the sequential baseline, in the Meteor score of the test split of LDC2017T10 as a function of the number of reentrancies.

6.4.1 Reentrancies

We observed an advantage of graph encoders over tree and sequential encoders, but it is not yet clear which factors contribute to the improvements. Since graph encoders are the only type of encoders to model reentrancies explicitly, we expect them to deal better with these structures. However, AMR datasets contain a large number of examples that do not involve any reentrancies, as shown in Table 6.3. Hence, the ability of models to capture reentrancies may not be reflected in the overall BLEU scores. We therefore expected that the benefit of the graph models will be more evident for those examples containing more reentrancies. To test this hypothesis, we evaluate the various scenarios as a function of the number of reentrancies in each example, using the Meteor score as a metric.[4] Table 6.4 shows that the gap between the graph encoder and the other encoders is widest for examples with a large number of reentrancies. The Meteor score of the graph encoder for these cases is 3.1% higher than the score for the sequential encoder and 2.3% higher than the score for the tree encoder. The large gaps demonstrate that explicitly encoding reentrancies

[3]We run comparisons on systems without ensembling nor additional data.

[4]For this analysis we use Meteor instead of BLEU because it is a sentence-level metric, unlike BLEU, which is a corpus-level metric.

is more beneficial than the overall BLEU scores would suggest. Interestingly, it can also be observed that the graph model outperforms the tree model also for examples with no reentrancies, where tree and graph structures are identical. This suggests that preserving reentrancies in the training data has other beneficial effects that lead to better node embeddings.

6.4.1.1 Manual Inspection

In order to further explore how the graph model handles reentrancies differently from the other models, we performed a manual inspection of the models' output. We selected examples containing reentrancies, where the graph model performs better than the other models. These are shown in Table 6.5. In Example (1), we note that the graph model is the only one that correctly predicts the phrase *he finds out*. The wrong verb tense is due to the lack of tense information in AMR graphs. In the sequential model, the pronoun is chosen correctly, but the wrong verb is predicted, while in the tree model the pronoun is missing. In Example (2), only the graph model correctly generates the phrase *you tell them*, while none of the models use *people* as the subject of the predicate *can*. In Example (3), both the graph and the sequential models deal well with the control structure caused by the *recommend* predicate. The sequential model, however, overgenerates a wh-clause. Finally, in Example (4) the tree and graph models deal correctly with the possessive pronoun to generate the phrase *tell your ex*, while the sequential model does not. Overall, we note that the graph model produces a more accurate output than sequential and tree models by generating the correct pronouns and mentions when control and coreference are involved.

6.4.1.2 Contrastive Pairs

For a quantitative analysis of how the different models handle reentrancies, we use a method to inspect NMT output for specific linguistic analysis based on contrastive pairs (Sennrich, 2017). Given a reference output sentence, a contrastive sentence is generated by introducing a mistake related to the phenomenon we are interested in evaluating. The probability that the model assigns to the reference sentence is then compared to that of the contrastive sentence. The accuracy of a model is determined by the percentage of examples

(1)	REF	i dont tell him but **he finds out** ,
	SEQ	i did n't tell him but **he was out** .
	TREE	i do n't tell him but **found out** .
	GRAPH	i do n't tell him but **he found out** .
(2)	REF	if **you tell people** they can help you ,
	SEQ	if **you tell him** , you can help you !
	TREE	if **you tell person_name_0 you** , you can help you .
	GRAPH	if **you tell them** , you can help you .
(3)	REF	**i 'd recommend** you go and see your doctor too .
	SEQ	**i recommend** you go to see your doctor who is going to see your doctor .
	TREE	**you recommend** going to see your doctor too .
	GRAPH	**i recommend** you going to see your doctor too .
(4)	REF	(you) **tell your ex** that all communication needs to go through the lawyer .
	SEQ	(you) **tell** that all the communication go through lawyer .
	TREE	(you) **tell your ex** , tell your ex , the need for all the communication .
	GRAPH	(you) **tell your ex** the need to go through a lawyer .

Table 6.5: Examples of generation from AMR graphs containing reentrancies. REF is the reference sentence.

in which the reference sentence has a higher probability than the contrastive sentence.

We produce contrastive examples by running CoreNLP (Manning et al., 2014) to identify coreference, which is one of the main causes of reentrancies, and introducing a mistake. When an expression has multiple mentions, the antecedent is repeated in the linearized AMR. For instance, the linearization in Figure 6.1(b) contains the token *he* twice, which instead appears only once in the sentence. This repetition may result in generating the token *he* twice, rather than using a pronoun to refer back to it. To investigate this possible mistake, we replace one of the mentions with the antecedent (e.g., *He ate the pizza with his fingers* is replaced with *He ate the pizza with he fingers*, which is

Model	Antec.	Type	Num.	Gender
SEQ	96.02	97.70	94.89	94.74
TREE	96.02	96.38	93.70	92.63
GRAPH	96.02	96.49	95.11	95.79

Table 6.6: Accuracy (%) of models, on the test split of LDC201T10, for different categories of contrastive errors: antecedent (Antec.), pronoun type (Type), number (Num.), and gender (Gender).

ungrammatical and as such should be less likely).

An alternative hypothesis is that even when the generation system correctly decides to predict a pronoun, it selects the wrong one. To test for this, we produce contrastive examples where a pronoun is replaced by either a different type of pronoun (e.g., *He ate the pizza with his fingers* is replaced with *He ate the pizza with him fingers*) or by the same type of pronoun but for a different number (*He ate the pizza with their fingers*) or different gender (*He ate the pizza with her fingers*). Note from Figure 6.1 that the graph-structured AMR is the one that more directly captures the relation between *finger* and *he*, and as such it is expected to deal better with this type of mistakes.

From the test split of LDC2017T10, we generated 251 contrastive examples due to antecedent replacements, 912 due to pronoun type replacements, 1840 due to number replacements and 95 due to gender replacements. The results are shown in Table 6.6. The sequential encoder performs well at this task, with better or on par performance with respect to the tree encoder. The graph encoder outperforms the sequential encoder only for pronoun number and gender replacements. The models achieve similar accuracies and more subtle contrastive examples may be needed to empirically confirm the results of the qualitative analysis of Section 6.4.1.1. Other approaches to inspect phenomena of coreference and control verbs can also be explored, for instance by devising specific training objectives (Linzen et al., 2016).

6.4.2 Long-range Dependencies

When we encode a long sequence, interactions between items that appear distant from each other in the sequence are difficult to capture. The problem of

# max length	# dev sents.	# test sents.
0-10	292	307
11-50	350	297
51-250	21	18

Table 6.7: Counts of longest edges in the development and test split of LDC2017T10

Model	Max dependency length		
	0-10	11-50	51-200
SEQ	50.49	36.28	24.14
TREE	-0.48	+1.66	+2.37
GRAPH	**+1.22**	**+2.05**	**+3.04**

Table 6.8: Differences, with respect to the sequential baseline, in the Meteor score of the test split of LDC2017T10 as a function of the maximum edge length.

long-range dependencies in natural language is well known for RNN architectures (Bengio et al., 1994). Indeed, the need to solve this problem motivated the introduction of LSTM models, which are known to model long-range dependencies better than traditional RNNs.

Because the nodes in the graphs are not aligned with words in the sentence, AMR has no notion of distance between the nodes taking part in an edge. In order to define the length of an AMR edge, we resort to the AMR linearization discussed in Section 6.1. Given the linearization of the AMR $x_1, \ldots, x_N$, as discussed in Section 6.1, and an edge between two nodes x_i and x_j, the length of the edge is defined as $|j - i|$.

In order to verify the hypothesis that long-range dependencies contribute to the improvements of graph models, we compare the models as a function of the maximum edge length in each example. Longer dependencies are sometimes caused by reentrancies, as in the edge between *:part-of* and *he* in Figure 6.1. To verify that the contribution in terms of longer dependencies is complementary to that of reentrancies, we exclude sentences with reentrancies from this analysis. Table 6.7 shows the statistics for this measure. Results are shown in Table 6.8. The graph encoder always outperforms both the sequential and the tree encoder. The gap of both structural encoders (tree and

graph) with the sequential encoder increases for longer dependencies. This indicates that longer dependencies are an important factor in improving results for structural encoders.

6.5 Summary

In this chapter, we shifted our focus from parsing to generation. We introduced models for AMR-to-text generation to investigate the difference between sequential, tree and graph encoders. We showed that tree encoders, which encode most structural information, outperform the sequential encoder and that reentrancies further improve generation results in the graph encoders. These results support the hypothesis that taking into account reentrancies is important for AMR-to-text generation. As expected, we observed larger improvements in the structural encoders when the input AMR graphs have a larger number of reentrant structures and longer dependencies. Our best graph encoder, which consists of a GCN wired to a BiLSTM network, improves over previous work on all tested datasets. AMR-to-text generation systems that report higher BLEU scores have since been published (Cao and Clark, 2019; Guo et al., 2019). The source code of our system is available at `https://github.com/mdtux89/OpenNMT-py-AMR-to-text` and a demo is available at `http://cohort.inf.ed.ac.uk/amrgen.html`.

We inspected the differences between the models in terms of pronominal anaphora and control structures, which are caused by reentrancies. Using contrastive pair analysis, we tested the hypothesis that graph encoders, with access to reentrancies, would result in a better generation of pronouns. The generated contrastive examples are available at `https://github.com/mdtux89/OpenNMT-py-AMR-to-text/tree/master/contrastive_examples`. Our results do not fully confirm our hypothesis and future work is needed to answer this question conclusively. Other approaches to inspect phenomena of co-reference and control verbs can also be explored, for instance by devising specific training objectives (Linzen et al., 2016).

7

Conclusions

In this thesis, we studied AMR as a way to find solutions to NLP problems based on an abstract representation of language. To this end, we studied AMR parsing, where sentences are represented as AMR graphs, and AMR-to-text generation, where sentences are generated from AMR graphs. For both AMR parsing and AMR-to-text generation, we demonstrated the importance of reentrancies, without which AMR could be represented as trees rather than graphs.

For AMR parsing, we proposed a transition system that accounts for reentrancies occurring between sibling nodes. We highlighted the positive impact of predicting such reentrancies by developing a set of fine-grained evaluation metrics, including one for reentrancies prediction. The reentrancy prediction metric also showed that state-of-the-art parsers do not cope well with reentrancies, and motivated us to carry out a deeper analysis of the role of reentrancies. We found sources of reentrancies that have not been acknowledged in the AMR literature such as adjunct control, verbalization, and pragmatics. We quantified their prevalence in an AMR corpus and found that control and coreference are the most frequent causes of reentrancies. We argued that reentrancies due to a pragmatic interpretation of the annotators are controversial as AMR is not intended to represent such aspects of language. Oracle experiments showed that if we could correct errors due to reentrancies, the overall parsing performance would significantly improve, demonstrating their importance for AMR parsing.

For AMR-to-text generation, we showed that neural encoders that have access to reentrancies outperform those who do not, demonstrating their relevance also for this task.

We also explored the use of AMR for languages other than English. Building AMR datasets for other languages is expensive and requires language-specific annotation guidelines. We therefore investigated the possibility of reusing existing English AMR annotations for other languages. We provided supporting evidence to this claim by training and analyzing AMR parsers for Italian, Spanish, German, and Chinese.

7.1 Future Directions

We conclude this thesis with a brief discussion on future directions for research on AMR.

AMR Parsing The fine-grained metrics we proposed have been used to compare AMR parsers (van Noord and Bos, 2017b; Anchiêta and Pardo, 2018; Lyu and Titov, 2018; Zhang et al., 2019, inter alia). However, they also indicate the subtasks that are worth investigating further. While we focused on improving reentrancy prediction, another subtask worth looking at is concept identification. Concept identification involves predicting the correct nodes of the AMR graph. Compared to syntactic parsing and other forms of semantic parsing (e.g., UCCA, Abend and Rappoport 2013), alignments between surface form and semantic representation are latent in AMR parsing, making concept identification a challenging task. Zhang et al. (2019) reported that parsers can achieve 90.9% Smatch, when using gold concepts. The state-of-the-art concept identification score of 86% F1 is achieved by Lyu and Titov (2018), where AMR alignments are treated as latent variables and learned jointly with parsing. Improving concept identification may be challenging, but its impact on overall parsing is worth the effort. Another task that can have a significant impact on parsing performance is SRL. Its current state-of-the-art is 70% F1 (Lyu and Titov, 2018), leaving large room for improvement.

AMR-to-text Generation Recent work proved the effectiveness of the graph-to-sequence approach to the generation task (Song et al., 2018; Beck et al., 2018). Our experiments showed that graph-to-sequence models are not only superior to sequence-to-sequence models but also to tree-to-sequence models. We achieved state-of-the-art results with GCN by a combination of structural and

sequential encoding, while maintaining a shallower, faster to train, architecture. In contrast, a deep GCN architecture was used by Guo et al. (2019), which outperforms our models by allowing the GCN to learn more global node embeddings. It is worth carrying out a direct comparison of the two approaches and analyzing advantages and disadvantages. Another promising approach to the task consists in first predicting the syntactic structure, from which the surface form is generated (Cao and Clark, 2019).

Reentrancies Our analysis revealed some unexpected causes of reentrancies. For example, annotators sometimes add reentrancies based on their pragmatic interpretation of a sentence. This inevitably adds noise to the data. The phenomena that are intended to lead to reentrancies should be clarified, facilitating the work of annotators and parsing algorithms alike.

Our analysis fails to detect the causes of many reentrancies in the data. For a more precise estimate of the most common causes of reentrancies, it is necessary to perform a manual analysis.

While we demonstrated that correcting reentrancy errors with an oracle considerably improves performance, the baselines we proposed were not able to effectively learn how to correct such errors. We leave for future work to improve upon our baselines. An alternative solution is to incorporate reentrancy prediction in training so that the parsers make fewer reentrancy errors to start with. The latter approach was recently followed by Zhang et al. (2019). Their state-of-the-art score for reentrancy is however only 60%, leaving room for improvement. A third approach is that of pre-processing reentrancy information, which has been shown to be a successful way to improve semantic parsing results (van Noord and Bos, 2017a; van Noord et al., 2018).

AMR for other languages We projected English AMR corpora to other languages and argued that the noise in the projected AMR alignments is responsible for poor performance. Hence, we expect that performance could improve when replacing AMREAGER with parsers that do not rely on alignments (Lyu and Titov, 2018; Zhang et al., 2019).

Differences between languages result in loss of information when sharing AMR graphs across languages. Even though we showed that parsers can learn how to cope with some of these differences, it is preferable to acquire language-

specific datasets, if possible. This also requires the definition of annotation guidelines for each target language.

Applications There have been attempts to use AMR graphs for downstream NLP problems (Liu et al., 2015; Issa et al., 2018; Song et al., 2019, inter alia). Our solutions to parsing and generation, and subsequent work by other researchers on these tasks, now offer tools to convert natural language text to AMR and vice-versa with higher accuracy. These tools may be mature enough to allow us to test AMR on downstream NLP applications such as MT and summarization.

7.2 Software and Data

We release the following resources used for the experiments in this thesis:

- AMREAGER parser for English and its adaption to Italian, Spanish, German, and Chinese: `https://github.com/mdtux89/amr-eager-multilingual`. It allows parsing sentences into AMR in linear time, incrementally. We also released a demo: `http://cohort.inf.ed.ac.uk/amreager.html`;
- Fine-grained evaluation suite, now commonly used to compare AMR parsers: `https://github.com/mdtux89/amr-evaluation`;
- Analysis of reentrancies: It includes scripts to quantify the causes of reentrancies in AMR and the oracle used to measure the impact of reentrancies on AMR parsing performance (the code is yet not accessible to facilitate double-blind reviewing);
- Professional translations of the test set of the LDC2015E86 AMR dataset to Italian, Spanish, German, and Chinese (available upon request). These can be used to evaluate AMR parsers for languages that do not have a dedicated AMR dataset yet;
- AMR-to-text systems based on sequential, tree, and graph encoders: `https://github.com/mdtux89/OpenNMT-py-AMR-to-text`. We also released a demo: `http://cohort.inf.ed.ac.uk/amrgen.html`;
- Constrastive examples: `https://github.com/mdtux89/OpenNMT-py-AMR-to-text/tree/master/contrastive_examples`. We generated these examples to

compare how the different AMR-to-text models handle reentrancies caused by pronouns.

Bibliography

Abend, O. and Rappoport, A. (2013). Universal conceptual cognitive annotation (ucca). In *Proceedings of ACL*.

Abzianidze, L., Bjerva, J., Evang, K., Haagsma, H., van Noord, R., Ludmann, P., Nguyen, D.-D., and Bos, J. (2017). The parallel meaning bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. *Proceedings of EACL*.

Akbik, A., Chiticariu, L., Danilevsky, M., Li, Y., Vaithyanathan, S., and Zhu, H. (2015). Generating high quality proposition banks for multilingual semantic role labeling. In *Proceedings of ACL*.

Anchiêta, R. T. and Pardo, T. A. S. (2018). A rule-based amr parser for portuguese. In *Proceedings of IBERAMIA*.

Anchiêta, R. T. and Pardo, T. A. S. (2018). Towards amr-br: A sembank for brazilian portuguese language. In *Proceedings of LREC*.

Aprosio, A. P. and Moretti, G. (2016). Italy goes to stanford: a collection of corenlp modules for italian. *arXiv preprint arXiv:1609.06204*.

Artzi, Y., Lee, K., and Zettlemoyer, L. (2015). Broad-coverage CCG semantic parsing with AMR. *Proceedings of EMNLP*.

Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser. In *Proceedings of CoNLL*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *Proceedings of ICLR*.

Ballesteros, M. and Al-Onaizan, Y. (2017). Amr parsing using stack-lstms. In *Proceedings of EMNLP*.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. *Linguistic Annotation Workshop*.

Banerjee, S. and Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*.

Barone, A. V. M., Helcl, J., Sennrich, R., Haddow, B., and Birch, A. (2017). Deep architectures for neural machine translation. In *Proceedings of the Second Conference on Machine Translation*.

Barzdins, G. and Gosko, D. (2016). Riga: Impact of smatch extensions and character-level neural translation on AMR parsing accuracy. *International Workshop on Semantic Evaluation*.

Bastings, J., Titov, I., Aziz, W., Marcheggiani, D., and Simaan, K. (2017). Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of EMNLP*.

Beck, D., Haffari, G., and Cohn, T. (2018). Graph-to-sequence learning using gated graph neural networks. *Proceedings of ACL*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bohnet, B. and Nivre, J. (2012). A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465. Association for Computational Linguistics.

Bojar, Z. U. J. H. O. (2014). Comparing czech and english amrs. In *Workshop on Lexical and Grammatical Resources for Language Processing*.

Bos, J. (2004). Computational semantics in discourse: Underspecification, resolution, and inference. *Journal of Logic, Language and Information*, 13(2):139–157.

Bos, J., Basile, V., Evang, K., Venhuizen, N. J., and Bjerva, J. (2017). The groningen meaning bank. In *Handbook of linguistic annotation*, pages 463–496. Springer.

Brill, E. (1993). *Transformation-Based Learning*. PhD thesis, PhD thesis, Univ. of Pennsylvania.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational linguistics*, 21(4):543–565.

Cai, S. and Knight, K. (2013). Smatch: an evaluation metric for semantic feature structures. *Proceedings of ACL*.

Cao, K. and Clark, S. (2019). Factorising amr generation through syntax. In *Proceedings of NAACL*.

Carreras, X., Chao, I., PadrÃş, L., and PadrÃş, M. (2004). Freeling: An open-source suite of language analyzers. In *Proceedings of LREC*.

Cetoli, A., Bragaglia, S., O'Harney, A., and Sloan, M. (2017). Graph convolutional networks for named entity recognition. In *International Workshop on Treebanks and Linguistic Theories*.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*.

Chen, H., Huang, S., Chiang, D., and Chen, J. (2017). Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of ACL*.

Chen, W.-T. (2015). Learning to map dependency parses to abstract meaning representations. *ACL-IJCNLP 2015 Student Research Workshop*.

Chen, W.-T. and Palmer, M. (2017). Unsupervised amr-dependency parse alignment. In *Proceedings of EACL*.

Cheng, J., Reddy, S., Saraswat, V., and Lapata, M. (2017). Learning structured natural language representations for semantic parsing. In *Proceedings of ACL*.

Chu, C. and Kurohashi, S. (2016). Supervised syntax-based alignment between english sentences and abstract meaning representation graphs. *arXiv:1606.02126v1*.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of ACL*.

Cinková, S., Toman, J., Hajic, J., Cermáková, K., Klimeš, V., Mladová, L., Šindlerová, J., Tomšu, K., and Zabokrtskỳ, Z. (2009). Tectogrammatical annotation of the wall street. *The Prague Bulletin of Mathematical Linguistics*.

Cohen, S. B., Das, D., and Smith, N. A. (2011a). Unsupervised structure prediction with non-parallel multilingual guidance. In *Proceedings of EMNLP*.

Cohen, S. B., Gómez-Rodríguez, C., and Satta, G. (2011b). Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of EMNLP*.

Cohen, S. B. and Smith, N. A. (2009). Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In *Proceedings of NAACL-HLT*.

Copestake, A., Flickinger, D., Pollard, C., and Sag, I. A. (2005). Minimal recursion semantics: An introduction. *Research on language and computation*, 3(2-3):281–332.

Covington, M. A. (2011). A fundamental algorithm for dependency parsing. *Proceedings of ACM southeast conference*.

Damonte, M. and Cohen, S. B. (2018). Cross-lingual abstract meaning representation parsing. In *Proceedings of NAACL*.

Damonte, M. and Cohen, S. B. (2019). Structural neural encoders for amr-to-text generation. In *Proceedings of NAACL*.

Damonte, M., Cohen, S. B., and Satta, G. (2017). An incremental parser for abstract meaning representation. In *Proceedings of EACL*.

Damonte, M., Szubert, I., Cohen, S., and Steedman, M. (2019). The role of reentrancies in abstract meaning representation parsing. *arXiv preprint*.

Daumé III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325.

Davidson, D. (1969). The individuation of events. In *Essays in honor of Carl G. Hempel*, pages 216–234. Springer.

Demner-Fushman, D., Chapman, W. W., and McDonald, C. J. (2009). What can natural language processing do for clinical decision support? *Journal of biomedical informatics*, 42(5):760–772.

Dixon, R. M. (2010). *Basic linguistic theory volume 2: Grammatical topics*, volume 2. Oxford University Press on Demand.

Dorr, B. J. (1994). Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, 20(4):597–633.

Dorr, B. J., Pearl, L., Hwa, R., and Habash, N. (2002). Improved word-level alignment: Injecting knowledge about mt divergences. Technical report, DTIC Document.

Du, Y., Zhang, F., Sun, W., and Wan, X. (2014). Peking: Profiling syntactic tree parsing techniques for semantic graph parsing. In *International Workshop on Semantic Evaluation*.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of NIPS*.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (2015). Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL*.

Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of NAACL-HLT*.

Eriguchi, A., Hashimoto, K., and Tsuruoka, Y. (2016). Tree-to-sequence attentional neural machine translation. In *Proceedings of ACL*.

Evang, K. and Bos, J. (2016). Cross-lingual learning of an open-domain semantic parser. In *Proceedings of COLING*.

Ferreira, T. C., Calixto, I., Wubben, S., and Krahmer, E. (2017). Linguistic realisation as machine translation: Comparing different mt models for amr-to-text generation. In *Proceedings of INLG*.

Flanigan, J., Dyer, C., Smith, N. A., and Carbonell, J. (2016a). CMU at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss. *International Workshop on Semantic Evaluation*.

Flanigan, J., Dyer, C., Smith, N. A., and Carbonell, J. (2016b). Generation from abstract meaning representation using tree transducers. *Proceedings of NAACL*.

Flanigan, J., Thomson, S., Carbonell, J. G., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation. *Proceedings of ACL*.

Garg, S., Galstyan, A., Hermjakob, U., and Marcu, D. (2016). Extracting biomolecular interactions using semantic parsing of biomedical text. *Proceedings of AAAI*.

Garg, S., Steeg, G. V., and Galstyan, A. (2018). Stochastic learning of nonstationary kernels for natural language modeling. *arXiv preprint arXiv:1801.03911*.

Gesmundo, A., Henderson, J., Merlo, P., and Titov, I. (2009). A latent variable model of synchronous syntactic-semantic parsing for multiple languages. In *Proceedings of CoNLL*.

Gilroy, S. (2019). Probabilistic graph formalisms for meaning representations. *The University of Edinburgh*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Proceedings of ICASSP*.

Groschwitz, J., Fowlie, M., Johnson, M., and Koller, A. (2017). A constrained graph algebra for semantic parsing with amrs. In *IWCS 2017-12th International Conference on Computational Semantics-Long papers*.

Gruzitis, N., Gosko, D., and Barzdins, G. (2017). Rigotrio at semeval-2017 task 9: Combining machine learning and grammar engineering for amr parsing and generation. In *International Workshop on Semantic Evaluation*.

Gulcehre, C., Ahn, S., Nallapati, R., Zhou, B., and Bengio, Y. (2016). Pointing the unknown words. In *Proceedings of ACL*.

Guo, Z., Zhang, Y., Teng, Z., and Lu, W. (2019). Densely connected graph convolutional networks for graph-to-sequence learning. In *Proceedings of TACL*.

Habel, A. (1992). *Hyperedge replacement: grammars and languages*, volume 643. Springer Science & Business Media.

Henderson, J., Merlo, P., Musillo, G., and Titov, I. (2008). A latent variable model of synchronous parsing for syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 178–182. Association for Computational Linguistics.

Hirschman, L., Robinson, P., Burger, J., and Vilain, M. (1997). Automating coreference: The role of annotated training data. In *Proceedings of the AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*.

Hobbs, J. R. (1979). Coherence and coreference. *Cognitive science*, 3(1):67–90.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hodgson, T. and Coiera, E. (2015). Risks and benefits of speech recognition for clinical documentation: a systematic review. *Journal of the American Medical Informatics Association*, 23(e1):e169–e179.

Hoffman, B. (1992). A ccg approach to free word order languages. In *Proceedings of ACL*.

Huddleston, R. and Pullum, G. K. (2002). *Non-finite and verbless clauses*, page 1171âĂŞ1272. Cambridge University Press.

Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., and Kolak, O. (2005). Bootstrapping parsers via syntactic projection across parallel texts. *Natural language engineering*, 11(03):311–325.

Issa, F., Damonte, M., Cohen, S. B., Yan, X., and Chang, Y. (2018). Abstract meaning representation for paraphrase detection. In *Proceedings of NAACL*.

Johnson, M. (2002). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of ACL*, pages 136–143. Association for Computational Linguistics.

Jurčíček, F., Gašić, M., Keizer, S., Mairesse, F., Thomson, B., Yu, K., and Young, S. (2009). Transformation-based learning for semantic parsing. In *Tenth Annual Conference of the International Speech Communication Association*.

Kamp, H. and Reyle, U. (1993). From discourse to logic: an introduction to modeltheoretic semantics, formal logic and discourse representation theory. *Hingham, MA: Kluwer*.

Kendall, M. G. (1945). The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251.

Kingsbury, P. and Palmer, M. (2002). From treebank to propbank. *Proceedings of LREC*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *Proceedings of ICLR*.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL*.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL*.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural amr: Sequence-to-sequence models for parsing and generation. *Proceedings of ACL*.

Kubler, S., McDonald, R., Nivre, J., and Hirst, G. (2009). *Dependency Parsing*. Morgan and Claypool Publishers.

Kuhlmann, M. and Jonsson, P. (2015). Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics*, pages 559–570.

Kuhlmann, M. and Oepen, S. (2016). Towards a catalogue of linguistic graph banks. *Computational Linguistics*, 42(4):819–827.

Lampouras, G. and Vlachos, A. (2017). Sheffield at semeval-2017 task 9: Transition-based language generation from amr. In *International Workshop on Semantic Evaluation*.

Lee, J., Cho, K., and Hofmann, T. (2017). Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378.

Levi, F. W. (1942). *Finite geometrical systems*. University of Calcutta.

Li, B., Wen, Y., Weiguang, Q., Bu, L., and Xue, N. (2016). Annotating the little prince with chinese amrs. In *Linguistic Annotation Workshop*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Liang, P. (2013). Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Linzen, T., Dupoux, E., and Goldberg, Y. (2016). Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations. *Proceedings of NAACL*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *Proceedings of EMNLP*.

Lyu, C. and Titov, I. (2018). Amr parsing as graph prediction with latent alignment. *Proceedings of ACL*.

Mann, W. C. (1983). An overview of the penman text generation system. In *Proceedings of AAAI*, pages 261–265.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of ACL*.

Marcheggiani, D. and Perez-Beltrachini, L. (2018). Deep graph convolutional encoders for structured data to text generation. *Proceedings of INLG*.

Marcheggiani, D. and Titov, I. (2017). Encoding sentences with graph convolutional networks for semantic role labeling. *Proceedings of EMNLP*.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

May, J. (2016). Semeval-2016 task 8: Meaning representation parsing. In *International Workshop on Semantic Evaluation*.

May, J. and Priyadarshi, J. (2017). Semeval-2017 task 9: Abstract meaning representation parsing and generation. In *International Workshop on Semantic Evaluation*.

McDonald, R., Petrov, S., and Hall, K. (2011). Multi-source transfer of delexicalized dependency parsers. In *Proceedings of EMNLP*.

Morante, R. and Blanco, E. (2012). *SEM 2012 shared task: Resolving the scope and focus of negation. In *Proceedings of *SEM*.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Navigli, R. (2009). Word sense disambiguation: A survey. *ACM computing surveys (CSUR)*, 41(2):10.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113.

Nguyen, D. Q., Nguyen, D. Q., Pham, D. D., and Pham, S. B. (2016). A robust transformation-based learning approach using ripple down rules for part-of-speech tagging. *AI Communications*, 29(3):409–422.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. *Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.

Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Nivre, J. (2009). Non-projective dependency parsing in expected linear time. In *Proceedings of ACL*.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2000). Improved statistical alignment models. In *Proceedings of ACL*.

Opitz, J. and Frank, A. (2019). Automatic accuracy prediction for amr parsing. In *Proceedings of *SEM*.

Padó, S. and Lapata, M. (2009). Cross-lingual annotation projection for semantic roles. *Journal of Artificial Intelligence Research*, 36(1):307–340.

Palmer, M., Gildea, D., and Xue, N. (2010). Semantic role labeling. *Synthesis Lectures on Human Language Technologies*, 3(1):1–103.

Pan, X., Cassidy, T., Hermjakob, U., Ji, H., and Knight, K. (2015). Unsupervised entity linking with abstract meaning representation. *Proceedings of NAACL*.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS Workshop*.

Peng, X., Gildea, D., and Satta, G. (2018). Amr parsing with cache transition systems. In *Proceedings of AAAI*.

Peng, X., Song, L., and Gildea, D. (2015). A synchronous hyperedge replacement grammar based approach for AMR parsing. *Proceedings of CoNLL*.

Pop, R., Dregan, A., Măcicăşan, F., Lemnaru, C., and Potolea, R. (2018). Enhancements on a transition-based approach for amr parsing using lstm networks. In *Proceedings of ICCP*.

Popović, M. (2017). chrf++: words helping character n-grams. In *Proceedings of the second conference on machine translation*, pages 612–618.

Pourdamghani, N., Gao, Y., Hermjakob, U., and Knight, K. (2014). Aligning english strings with abstract meaning representation graphs. *Proceedings of EMNLP*.

Pourdamghani, N. and Knight, K. (2016). Generating english from abstract meaning representation. In *Proceedings of INLG*.

Pust, M., Hermjakob, U., Knight, K., Marcu, D., and May, J. (2015). Parsing english into abstract meaning representation using syntax-based machine translation. *Proceedings of EMNLP*.

Ramshaw, L. A. and Marcus, M. P. (1999). Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Ranta, A. (2004). Grammatical framework. *Journal of Functional Programming*, 14(2):145–189.

Rao, S., Marcu, D., Knight, K., and Daumé III, H. (2017). Biomedical event extraction using abstract meaning representation. *BioNLP Workshop*.

Rao, S., Vyas, Y., Daume III, H., and Resnik, P. (2016). Clip@umd at semeval-2016 task 8: Parser for abstract meaning representation using learning to search. *International Workshop on Semantic Evaluation*.

Ribeyre, C., de La Clergerie, É. V., and Seddah, D. (2015). Because syntax does matter: Improving predicate-argument structures parsing using syntactic features. In *Proceedings of NAACL-HLT*.

Sagae, K. and Tsujii, J. (2008). Shift-reduce dependency dag parsing. *Proceedings of COLING*.

Sawai, Y., Shindo, H., and Matsumoto, Y. (2015). Semantic structure analysis of noun phrases using abstract meaning representation. *Proceedings of ACL*.

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of ACL*.

Sennrich, R. (2017). How grammatical is character-level neural machine translation? assessing mt quality with contrastive translation pairs. In *Proceedings of EACL*.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Miceli Barone, A. V., Mokry, J., and Nadejde, M. (2017). Nematus: a toolkit for neural machine translation. In *Proceedings of EACL*.

Søgaard, A. (2011). Data point selection for cross-language adaptation of dependency parsers. In *Proceedings of ACL-HLT*.

Song, L. and Gildea, D. (2019). Sembleu: A robust metric for amr parsing evaluation. *arXiv preprint arXiv:1905.10726*.

Song, L., Gildea, D., Zhang, Y., Wang, Z., and Su, J. (2019). Semantic neural machine translation using AMR. *Transactions of the Association for Computational Linguistics*, 7:19–31.

Song, L., Zhang, Y., Peng, X., Wang, Z., and Gildea, D. (2016). Amr-to-text generation as a traveling salesman problem. *Proceedings of EMNLP*.

Song, L., Zhang, Y., Wang, Z., and Gildea, D. (2018). A graph-to-sequence model for amr-to-text generation. In *Proceedings of ACL*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

Steedman, M. (1996). *Surface Structure and Interpretation*. The MIT Press.

Steedman, M. (2000). *The Syntactic Process*. The MIT Press.

Sulem, E., Abend, O., and Rappoport, A. (2015). Conceptual annotations preserve structure across translations: A french-english case study. In *Workshop on Semantics-Driven Statistical Machine Translation*.

Szubert, I., Lopez, A., and Schneider, N. (2018). A structured syntax-semantics interface for english-amr alignment. In *Proceedings of NAACL*.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *Proceedings of ACL*.

Takase, S., Suzuki, J., Okazaki, N., Hirao, T., and Nagata, M. (2016). Neural headline generation on abstract meaning representation. In *Proceedings of EMNLP*.

Titov, I. and Henderson, J. (2007). A latent variable model for generative dependency parsing. In *Proceedings of IWPT*.

Titov, I., Henderson, J., Merlo, P., and Musillo, G. (2009). Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Proceedings of IJCAI*.

Van Deemter, K. and Kibble, R. (1999). What is coreference, and what should coreference annotation be? In *Workshop on Coreference and its Applications*. Association for Computational Linguistics.

van der Plas, L., Samardžić, T., and Merlo, P. (2010). Cross-lingual validity of propbank in the manual annotation of french. In *Linguistic Annotation Workshop*.

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

van Noord, R. and Bos, J. (2017a). Dealing with co-reference in neural semantic parsing. In *Workshop on Semantic Deep Learning*.

van Noord, R. and Bos, J. (2017b). Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands Journal*, 7:93–108.

Vanderwende, L., Menezes, A., and Quirk, C. (2015). An amr parser for english, french, german, spanish and japanese and a new amr-annotated corpus. In *Proceedings of NAACL-HLT*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of NIPS*.

Vinyals, O. and Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Wang, C., Pradhan, S., Xue, N., Pan, X., and Ji, H. (2016). CAMR at SemEval-2016 task 8: An extended transition-based AMR parser. *International Workshop on Semantic Evaluation*.

Wang, C., Xue, N., and Pradhan, S. (2015a). Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. *Proceedings of ACL*.

Wang, C., Xue, N., and Pradhan, S. (2015b). A transition-based algorithm for AMR parsing. *Proceedings of NAACL*.

Wang, Y., Liu, S., Rastegar-Mojarad, M., Wang, L., Shen, F., Liu, F., and Liu, H. (2017). Dependency and amr embeddings for drug-drug interaction extraction from biomedical literature. In *Proceedings of ACM-BCB*.

Werling, K., Angeli, G., and Manning, C. D. (2015). Robust subgraph generation improves abstract meaning representation parsing. In *Proceedings of ACL*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, K., Wu, L., Wang, Z., and Sheinin, V. (2018). Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*.

Xue, N., Bojar, O., Hajic, J., Palmer, M., Uresova, Z., and Zhang, X. (2014). Not an interlingua, but close: Comparison of english amrs to chinese and czech. In *Proceedings of LREC*.

Yarowsky, D., Ngai, G., and Wicentowski, R. (2001). Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of HLT*.

Zeman, D. and Resnik, P. (2008). Cross-language parser adaptation between related languages. In *Proceedings of IJCNLP*.

Zhang, S., Ma, X., Duh, K., and Van Durme, B. (2019). Amr parsing as sequence-to-graph transduction. In *Proceedings of ACL*.

Zhang, Y. and Clark, S. (2008). A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*.

Zhang, Y. and Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. *Proceedings of ACL*.

Zhou, J., Xu, F., Uszkoreit, H., Weiguang, Q., Li, R., and Gu, Y. (2016). Amr parsing with an incremental joint model. In *Proceedings of EMNLP*.

Ziemski, M., Junczys-Dowmunt, M., and Pouliquen, B. (2016). The united nations parallel corpus v1. 0. In *Proceedings of LREC*.

Appendix A

Implementation mistake in parsing evaluation metrics

At the time of writing the final version of this thesis, we noticed an implementation mistake in the evaluation metrics of Chapter 4 based on the F1 score. We collapsed AMR concepts with the same lexical label, missing some of the errors made by the parser. For instance, consider the parses of Table 4.1. The *Gold* parse contains two *person* concepts, while the *Parse 1* does not contain any. Because the two *person* concepts are collapsed, the CONCEPTS metric only finds one error, instead of two. The error also applies to the metrics NAMED ENTITIES, WIKIFICATION, and NEGATIONS. Table A.1 shows the results before and after fixing the implementation mistake. Correcting the mistake results in generally lower scores, as more errors are found, but the comparison between the parsers is not affected. We recommend AMR researchers to use the latest version of the evaluation script available at `https://github.com/mdtux89/amr-evaluation`, and if possible, directly evaluate outputs of previous systems instead of indirectly comparing them as reported in a paper published prior to the release of the new script.

Metric	J (2014)	C (2015)	J (2016)	A (2017)	L&T (2018)
CONCEPTS*	79	80	83	83	85
CONCEPTS	77	78	81	81	84
NAMED ENTITIES*	75	75	79	83	86
NAMED ENTITIES	73	72	76	81	86
WIKIFICATION*	0	0	75	64	76
WIKIFICATION	0	0	71	60	73
NEGATIONS*	16	18	45	50	55
NEGATIONS	17	17	45	50	56

Table A.1: Results on test split of LDC2015E86 for the F1 metrics of Chapter 4 before (marked with the * symbol) and after correcting the implementation mistake.