Abstract Meaning Representation Parsing

A Dissertation

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Computer Science

Nianwen Xue, Advisor

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Chuan Wang

February, 2018

The signed version of this form is on file in the Graduate School of Arts and Sciences.

This dissertation, directed and approved by Chuan Wang's committee, has been accepted and approved by the Graduate Faculty of Brandeis University in partial fulfillment of the requirements for the degree of:

DOCTOR OF PHILOSOPHY

Eric Chasalow, Dean

Graduate School of Arts and Sciences

Dissertation Committee:

Nianwen Xue, Chair

James Pustejovsky, Dept. of Computer Science

Pengyu Hong, Dept. of Computer Science

Sameer Pradhan, cemantix.org and Boulder Learning

Acknowledgments

I would like to thank my dissertation committee as a whole, Nianwen Xue, James Pustejovsky, Pengyu Hong, and Sameer Pradhan. Thank you for your time and advice. I am grateful to many people who have been supportive along the way and I couldn't make it without your help.

Nianwen (Bert) Xue, my advisor: Bert admitted me to Brandeis and taught me everything about being a researcher. During the course of my study, from proposing research ideas to writing papers, I've gone through every step with the advice and help of Bert. He was always patient and gave me a lot of freedom. His kindness, motivation, enthusiasm, and immense knowledge not only helped me a lot in the scientific arena, but also on a personal level. Every meeting with Bert was both enjoyable and meaningful and I could not have imagined having a better advisor for my PhD study.

James Pustejovsky, Sameer Pradhan and Pengyu Hong: thank you for being on my committee and gave me so many insightful advice. Yaqin Yang, Te Rutherford and Yuchen Zhang: thank you for being my awesome lab mates. I am grateful for having met all of you and it is a precious memory working with you at Brandeis.

At last, thank my wife Chuchu Cheng: thank you for always being on my side and supporting me the whole time. The life of a PhD student could be grinding sometimes but you make it a whole lot easier.

Abstract

Abstract Meaning Representation Parsing

A dissertation presented to the Faculty of
the Graduate School of Arts and Sciences of
Brandeis University, Waltham, Massachusetts

by Chuan Wang

Understanding the true meaning of natural language sentence by the machine has been a long-time goal in the field of Natural Language Processing (NLP) and Artificial Intelligence (AI). A meaning representation (or semantic representation) is often required to represent the meaning of a natural language sentence in a machine-understandable way. The task of mapping natural language sentences into their meaning representations is called Meaning Representation Parsing (or Semantic Parsing). In this dissertation, we focus on parsing one particular form of meaning representation, Abstract Meaning Representation (AMR), which is a meaning representation formalism that is used to annotate a large semantic bank and has been shown to be promising in various downstream NLP tasks such as information extraction, text summarization, and machine comprehension.

In this dissertation, we address three challenges in AMR parsing: *Graph Representation*, *Abstraction*, and *Data Sparsity*. *Graph Representation* refers to the fact that AMR is formally a non-tree graph and new algorithms are needed to parse natural language sentences into graphs. We address this with a transition-based algorithm that builds an AMR graph incrementally from a dependency tree. *Abstraction* represents the fact that AMR does not provide alignments between concepts and relations in an AMR graph and word tokens in a natural language, but this connection is needed to model the transformation from the dependency tree of a sentence to an AMR graph. We address this challenge through two efforts, and these are: 1) designing a transition system that is capable of inferring concepts

that are not directly aligned to any particular concepts in the sentence, and 2) building a graph-based string-to-AMR aligner that takes advantages of the structural information in AMR graph. The *Data Sparsity* issue is caused by the large label space of AMR and the relatively small size of the AMR Bank at current stage. We tackle this problem by taking advantage of deep learning technique to build a Bidirectional LSTM based concept identifier upon a redesigned concept label set. We also explore the possibility of building an end-to-end Neural AMR parser through a sequence-to-sequence model. To complete this dissertation, we further apply our approaches to the Chinese AMR bank, where we extend our work to Chinese and discuss unique problems in Chinese AMR parsing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Natural Language Understanding (NLU) has been a long-standing goal within Natural Language Processing (NLP) and Artificial Intelligence (AI). To enable machine truly comprehend the meaning of a sentence, semantic parsing techniques are often employed to map the natural language sentence to some semantic representation. Abstract Meaning Representation (AMR) is one such semantic representation and it is represented as a rooted, directed, acyclic graph with labels on edges (relations) and leaves (concepts). A corpus of over 30 thousand sentences annotated with the AMR formalism (Banarescu et al., 2013) has been released and is still undergoing expansion. The building blocks for the AMR representation are concepts and relations between them. Understanding these concepts and their relations is crucial to understanding the meaning of a sentence and could potentially benefit a number of natural language applications such as Information Extraction, Question Answering and Machine Translation. Figure 1.1 shows one example of AMR graph.

The task of AMR parsing is to parse natural language sentences to AMR semantic graphs. There are several unique properties of AMR formalism which bring up new challenges to the parsing community and require development of novel algorithms:

- REENTRANCY. The property that makes AMR a graph instead of a tree is that AMR allows reentrancy, meaning that same concept can participate in multiple relations. Parsing a sentence into a graph would require more complicated algorithms and grammars and this introduces challenges for both learning and decoding.

- ABSTRACTION. Unlike a syntactic parse tree, AMR is abstract. It may represent any number of natural language sentences. As a result, there is no inherent alignment between the word tokens in a sentence and the concepts in an AMR graph. Intuitively, an English-to-AMR aligner is needed in order to establish the mapping between tokens and concepts. On the other hand, an AMR parser should be able to infer the concepts which carry deeper meaning of the sentence and do not necessarily align to any English tokens.

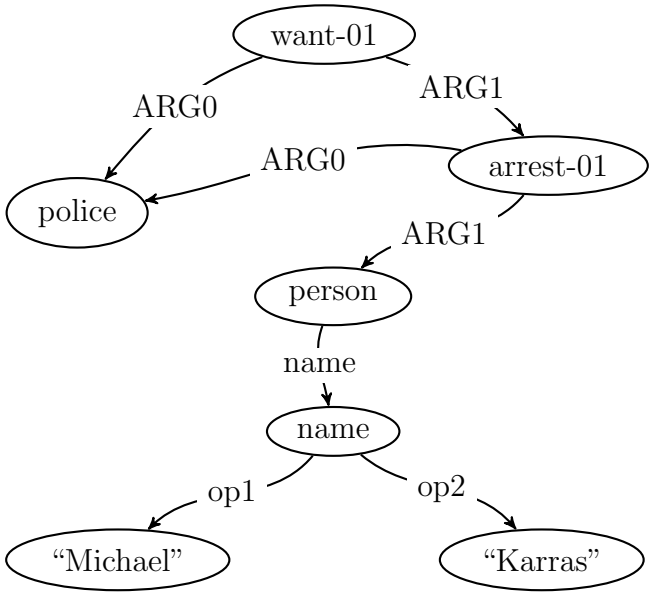


Figure 1.1: AMR graph for the sentence, “The police want to arrest Micheal Karras.”

- SPARSITY. A large portion of AMR concepts are either word lemmas or sense-disambiguated lemmas drawn from Propbank (Palmer et al., 2005). Since the AMR Bank is relatively

small at this stage, many of the concept labels in development set or test set only occur a few times or never appear in the training set. Addressing the sparsity of the dataset requires that the learning algorithm explores the sharing properties among similar labels/features and this means that the model needs to go beyond the one-hot representation and proper neural network techniques should be applied.

In this dissertation, we devise various algorithms focusing on different facets of AMR parsing. To tackle the graph parsing problem, we design a transition-based algorithm which formalizes AMR parsing as tree-to-graph transformation. We then extend the parser with the ability to infer concepts and explore the possible feature space that is beneficial to AMR parsing. To further address the SPARSITY and ABSTRACTION properties of AMR, we employ neural sequence labeling techniques for identifying concepts and design an automatic aligner which is more appropriate for the sentence-to-graph alignment scenario. In addition, we propose an end-to-end Neural AMR parser which explores the possibility of handling all the AMR phenomena using an integrated model. Finally, we extend our work to Chinese AMR parsing and give a preliminary study on our multilingual parser. In this chapter we discuss the motivation of our work and provide an overview of our contributions.

1.1 Contributions

In this section, we summarize the contributions of this dissertation.

1.1.1 Transition-based AMR Parser

This dissertation describes `CAMR`, an open-source transition-based AMR parser, which has achieved state-of-the-art results on various datasets.

The parser works by formalizing the AMR parsing task as a transformation process from a dependency tree to an AMR graph. A linear model is learned using the structures perceptron algorithm. Although we only investigate a simple greedy algorithm for decoding, our parser achieves state-of-the-art result on the initial release of AMR bank. This work is the first effort that tries to model similarities between the dependency tree of a sentence and its AMR graph through a customized set of actions. Empirical results show that our designed action set is able to cover most of the dependency-to-AMR transformation patterns. And the parser also runs in nearly linear time in practice in spite of a worst-case complexity of $O(n^2)$. `CAMR` also takes full advantage of dependency parsers that could be trained on data sets much larger than the AMR Annotation Corpus.

As an extension to `CAMR`, we explore the effectiveness of various external resources and feature sets. In addition to the original features covering lexicon and syntax information, we investigate features generated from existing semantic analyzers like semantic role labeling and coreference. A thorough feature extraction study gives us a systematic overview of what resources could be used to improve AMR parsing performance and gives us insight for future research directions.

`CAMR` also first addresses the **abstract concept** problem explicitly with an additional *infer* action. Existing AMR parsers either leave this problem unattended (Flanigan et al., 2014; Zhou et al., 2016) or solve it along with structure prediction (Pust et al., 2015; Artzi et al., 2015; Peng et al., 2015). We introduce an additional *infer* action to the original transition system which brings significant improvement to the parsing performance.

1.1.2 Neural Concept Identification

In this work, we address the concept identification problem in AMR parsing with the help of deep learning techniques. We formalize the concept identification as a sequence tagging task following Foland and Martin (2016, 2017). However, different from previous approaches that summarize the concept labels with a predefined set of types (Foland and Martin, 2016; Werling et al., 2015), we propose **Factored Concept Label (FCL)**, which is a generic and extensible framework to handle the concept label set based on their shared graph structure. This makes it possible for different concepts to be represented by one common label that captures the shared semantics of these concepts, thus greatly reducing the possible label space of AMR parsing. In addition, our proposed FCL label set is extracted from actual data and can generalize across different datasets or distributions, and are thus more flexible and robust.

Given the FCL label set, a Bidirectional LSTM concept identifier is learned to take advantage of contextual information from both directions. Notably, we are the first to introduce Convolutional Neural Networks (CNN) in the AMR parsing task. We integrate the CNN component into a Bidirectional LSTM network and show that character-level information is critical for capturing morphological and word shape information.

1.1.3 Graph-based Aligner

In this dissertation, we also describe `CAMR-align`, a novel string-to-graph aligner that first takes the graph information into account when building an unsupervised alignment model in AMR parsing. Existing methods used in AMR alignment is either rule-based (Flanigan et al., 2014) or unsupervised word aligner (Pourdamghani et al., 2014). For the unsupervised aligner, the AMR graph is often linearized and the structure information is lost along

the way. Our aligner investigates the effects of structure information in a Hidden Markov Model (HMM)-based word aligner through graph-based distortion and posterior rescoring. Analysis upon the alignment performance and further AMR parsing performance indicates that graph information is necessary when building an aligner between sentences and AMR graphs.

1.1.4 An End-to-End Neural AMR Parser

In this work, we explore the possibility of integrating all pipelines of AMR parsing (alignment, concept identification and relation extraction) into one end-to-end framework, sequence-to-sequence model. We propose a categorization method to reduce the sparsity issue without using self-training on a large external corpus. The categorization approach turns out to be every effective and significantly outperforms its plain sequence-to-sequence counterpart.

In neural translation model, the sequence-to-sequence model replies on an important component called attention mechanism to perform well. The attention under neural translation scenario can be treated as a soft alignment between source and target language. We investigate the effectiveness of attention in neural AMR parser by integrating an external unsupervised aligner into the end-to-end system. The relatively large gain brought by the external aligner shows that attention is not well learned under AMR parsing scenario. This indicates that given a small training set, external alignment information is useful for sequence-to-sequence model to obtain further improvement.

1.2 Chinese AMR Parsing

The development of AMR parser has mainly been focused on English language. Building a multilingual AMR parser is appealing as it will provide whole-sentence semantic represen-

tation to various other languages. We extend our work to Chinese AMR bank (Li et al., 2016), the first large annotated AMR Sembank other than English. We are able to verify that the main techniques used in this dissertation for English AMR parsing are language independent.

Chapter 2

Meaning Representation Parsing

In this chapter, we describe Abstract Meaning Representation in detail and compare it with other popular semantic representation. We then give a thorough review about the existing approaches of developing AMR parser and briefly discuss how AMR parsing has fueled other NLP tasks.

2.1 Abstract Meaning Representation

Understanding the meaning of natural language sentences by machine has long been one of the central goals in the field of natural language processing (NLP) and Artificial Intelligence (AI). With machine learning technique being the dominant approach in the field, the challenge has been to devise a meaning representation that is both computationally-friendly and can be used to consistently annotate a large amount of natural language data in multiple languages, which can be used to train machine learning algorithms.

Abstract Meaning Representation (AMR) (Banarescu et al., 2013), developed collaboratively by the Information Science Institute of University of Southern California, SDL, the

University of Colorado, and the Linguistic Data Consortium, has put an effort into building a robust meaning representation for English that has been used to annotate English language data at scale.

Following the success of syntactic treebanks, AMR builds the logical meanings on the whole sentence, which aims to address the the current state of fragmentation in the field, where semantic annotation has been largely "Balkanized". That is, different resources have been constructed focusing on separate aspects of semantic annotation. Named entities corpus only annotated entities from sentences, semantic relation corpus incrementally builds relations upon these entities and PropBank (Palmer et al., 2005) focuses on the predicate-argument structure of verbs while NomBank (Meyers et al., 2004) focuses on the argument structure of nominalized verbs and relational nouns. The advantage of whole-sentence representation is that it enables joint model to be learned so that various semantic information can interact during parsing, which results in better performance than tackled in isolation.

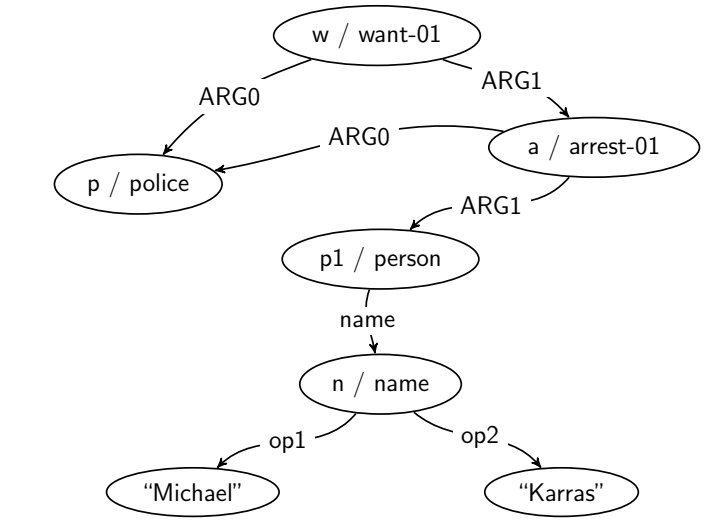
In general, AMR is built based on the following principles:

- **Graph Representation.** AMRs are rooted, directed, edge-labeled, leaf-labeled graphs, which enables coreference to be modeled by reentrancy. AMR format adapts the PENMAN notation (Matthiessen and Bateman, 1991) to represent the human reading and writing form.

- **Abstraction.** AMRs abstract away from syntactic and morphological variation. Different sentences may have exactly same AMR if they all express same semantic meaning. This also leads to the fact that no specific alignment between string and graph component has been provided in the annotation.

- **Framesets.** AMRs' predicates are annotated based on framesets defined in Propbank (Palmer et al., 2005). Also, predicate-argument structure is applied extensively,

for example, "teacher" is represented as "(person :ARG0-of (t / teach-01))".




```
(w / want-01
      :ARG0 (p / police)
      :ARG1 (a / arrest-01
            :ARG0 p
            :ARG1 (p1 / person
                  :name (n / name
                        :op1 "Michael"
                        :op2 "Karras" ))))
```

Figure 2.1: AMR graph and its PENMAN notation for the sentence, "The police want to arrest Micheal Karras."

Figure 2.1 shows one example AMR annotation from real data. Most of the nodes are identified by their **variable**, for example w, which is labeled with **concept**, *want-01*. The labeled edges connecting the nodes are **relations**, *ARG0*. Nodes that don't have variables are referred as **constants**, for example "*Michael*" in the graph, which are usually used to represent name, number or negation.

Intuitively, we can see from the figure that most of the AMR concepts can be associated with a single word in the sentence, which forms an one-to-one mapping and we will later show that how we use an aligner to build up the mapping. However, there are also some

concepts we cannot easily bind it with any single word in the sentence. These concepts normally represent inferred knowledge that are invoked by some special phrases or implicit relations between different clauses. We call this type of concepts **Abstract Concepts**. For example, "*person*" concept in Figure 2.1 is an inferred named entity type for the span "*Michael Karras*".

Although in Banarescu et al. (2013) the author claims that AMR is not an Interlingua, the property that it abstracts away from surface morphosyntactic differences makes it very appealing to develop cross-lingual AMR banks based on similar principles. Xue et al. (2014) shows that it is actually feasible to align English-Chinese AMRs, based on study of 100 English-Chinese sentences manually annotated with AMR pairs, which indicates that AMR formalism could be possibly applied for languages other than English.

Other Meaning Representations

There has been a long-standing study on semantic representation and we will briefly describe several meaning representation banks with sizable annotation and compare them with AMR.

Groningen Meaning Bank The Groningen Meaning Bank (GMB) (Basile et al., 2012) consists thousands of public domain English texts annotated with multi-layer syntactic and semantic representations. GMB builds formal deep semantics based on Discourse Representation Theory (DRT) (Kamp and Reyle, 1993). Specifically, when constructing different layers in GMB, VerbNet (Kipper et al., 2006) is used for thematic roles, a variation of ACE named entity type for entities, WordNet (Fellbaum, 1998) for word senses and Segmented DRT for rhetorical relations. One major difference between GMB and AMR as a semantic representation is that because of its DRT backbone, GMB can be expressed in first-order logic, which naturally covers quantification and scope. However, AMR doesn't handle quan-

tification at current stage.

Propbank Propbank (Palmer et al., 2005) is one of the popular recourses that focus on the thematic role representations, where most of the predicate-argument structures are annotated on verbs and nouns. Using the constituent parse tree as the base layer, for each identified predicate, various of semantic roles such as agent, theme, recipient, location, etc. are then detected and marked on the parse tree. This simple formalism makes it easy to be formulated as a machine learning task. And indeed it boosts the research of semantic role labeling task that has shown to be beneficial to many downstream NLP tasks. However, as a meaning representation, it only covers part of the whole-sentence meaning and is tightly coupled with parse tree, which in turn limits its availability to express the meaning of modification, quantification, or reification.

2.2 AMR Parsing

The task of AMR parsing is to map natural language strings to AMR semantic graphs. The recent releases of AMR bank has drawn a great interest in AMR parsing task and substantial amount of work has been done in this field. We first describe the evaluation metrics and then summarize significant approaches in AMR parsing.

2.2.1 Evaluation

From our discussion in Chapter 2.1, we can see that AMR graph can also be encoded as a conjunction of triples in the form *relation(variable1, variable2)*. Intuitively, given two AMRs, precision, recall and f-score can be calculated by counting the matched triples. However, in AMR there is no inherent alignment between variables, which results in large numbers of

possible matches. Smatch (Cai and Knight, 2013) is proposed to address this issue by finding the maximum f-score obtainable via a one-to-one matching of variables between the two AMRs. This problem turns out to be **NP-hard**, and Smatch utilizes hill-climbing method to get the approximate inference.

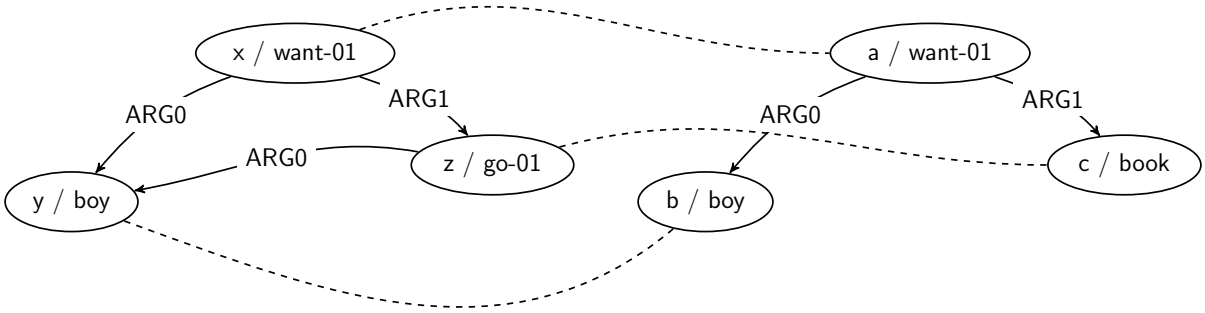


Figure 2.2: Two AMR graphs and the alignment leading to maximum f-score for sentences "The boy wants to go" and "The boy wants the book." [1].

Figure 2.2 shows the highest scoring alignment and the number of matched identical edges and concepts M is 4, total number of edges and concepts in two AMRs are $t_1 = 6$ and $t_2 = 5$ respectively, then the f-score is computed as follows:

$$\textit{f-score} = \frac{2 * M}{t_1 + t_2} = \frac{8}{6 + 5} = 0.73$$

2.2.2 Significant Approaches

In this section we introduce significant approaches in AMR parsing, which can be divided into the following categories:

- Graph-based Parsing;

[1] the example is from `https://github.com/nschneid/amr-tutorial/tree/master/slides`

- Transition-based Parsing;
- Parsing with Grammar Formalisms;
- Neural AMR Parsing;
- Others;

Graph-based Parsing

This line of work focuses on finding the effective inference algorithm for graph parsing. The first AMR parser JAMR (Flanigan et al., 2014) introduces a two-stage procedure: **concept identification** and **relation extraction**. In the first stage, a sequential labeler is utilized to predict the highest scoring spans and their concept graph fragments with dynamic programming algorithm. In the second stage, a novel **maximum spanning connected graph (MSCG)** search algorithm is proposed to find the subgraph G satisfying the following 3 constraints: preserving, simple and connected. The predicted graph is preserving if it includes all graph fragments from stage 1. It is simple if there is at most one edge between two vertices in G. Connected means that the resulting graphs should be weakly connected. Similar to the maximum spanning tree (MST) algorithm, at each iteration MSCG examines the highest non-positive scoring edge and adds the edge if it connects two unconnected components of graph G, until G is spanning and connected. Assuming the vertices set of graph is V, the algorithm runs in $O(|V|^2 \log |V|)$. In addition to satisfy the linguistically motivated constraint that one vertice should not have duplicate outgoing edge labels, also referred to as deterministic, Lagrangian Relaxation (LR) technique is applied to encode the constraint into a set of inequalities.

A competitive alternative to MSCG is Component-Wise Beam Search (CWBS) (Zhou et al., 2016), an effective decoding algorithm that builds graph incrementally. Given a

sequence of concept graph fragments $C = \{c_1, c_2, \ldots, c_n\}$, the decoder processes each graph fragment c_i from left to right and builds up partial graph using the following four actions:

1. SHIFT: Add only current graph fragment c_i to the partial graph.
2. LEFT-ARC: Add c_i and a highest-scoring edge from a node in c_i to a node in some preceding connected component to the partial graph.
3. RIGHT-ARC: Add c_i and a highest-scoring edge from a node in some preceding connected component to a node in c_i to the partial graph.
4. LEFT & RIGHT-ARCS: Add c_i and highest-scoring left arc and right arc to the partial graph.

Note that each search state represents a partial graph constructed incrementally by applying a series of historical actions. In general, at one search state, all four actions are permissible and it will result in exponential search space by considering all possible action sequences. Beam-search has proved to be quite efficient in such a scenario to find the approximate solutions without examining all possible search paths. The proposed CWBS algorithm applies beam-search extensively to address the huge search space in incremental AMR parsing. At each position, the algorithm only keeps the N-best scoring partial graph and more importantly when considering the possible actions applied between current graph fragment c_i and its preceding connected components, an *inner beam search* procedure is further applied to reduce the candidate actions. This *inner beam search* strategy can be treated as an alternative pruning technique applied in incremental parsing. A similar technique, also known as histogram thresholding, has been utilized in Machine Translation, where a maximum number n of partial translations are preserved in the hypotheses stack.

Transition-based Parsing

Transition-based parser formalizes the parsing procedure as application of a series of actions. Note that the incremental parser mentioned in last section is also a transition system. However, it is only applied to address the relation extraction problem while the concept identification part is handled similar to the technique used in JAMR. Specifically, in this section we discuss AMR parsers which handle concept identification and relation extraction jointly with a transition-based system.

Our work CAMR (Wang et al., 2015b,a, 2016) is a transition-based parser that addresses both concept identification and relation extraction from a tree-to-graph transformation perspective. Inspired by the similarity between dependency tree and AMR graph, a transition system with a set of linguistically motivated actions is designed to transform the syntactic tree to a semantic graph. The detail of our approaches will be discussed in the following chapters. Goodman et al. (2016b,a) adopts the transition system of CAMR and explores better learning algorithm in transition-based AMR parsing. Different from our effort to improve the existing AMR parser with rich semantic resources or analyzers as discussed in Chapter 4, this line of work addresses the problem of error propagation in the original greedy-style parser with various imitation learning algorithms. In addition, a noise reduction technique, α-bound (Khardon and Wachman, 2007), is introduced into the original perceptron classifier, where they simply excludes a training example from future training once it has been misclassified α times. Both techniques have been proved to be very effective and improve the standard transition-based parsing by a large margin.

Another appealing approach is that rather than taking the dependency tree of a sentence as the input, the transition system directly models the transformation from string to its semantic graph, as one would imagine that the tree-to-graph approach will become

impractical for languages without a well-trained dependency parser. Damonte et al. (2017) proposed a transition-based AMR parser by modifying the ArcEager transition system for dependency parsing. To handle the non-projectivity property of AMR, the original Shift action is enhanced with the ability to recover reentrant edges between the top-most node in the stack and its sibling nodes.

Parsing with Grammar Formalisms

The release of AMR corpus also draws a significant amount of attention from researchers studying graph parsing formalisms. One of the prominent context-free rewriting grammars, hyperedge replacement grammar (HRG), has been shown in Chiang et al. (2013) that it can be used as an efficient graph recognition formalism for large scale graph-related NLP applications. Chiang et al. (2013) also extends the HRG formalism to its synchronous counterpart *synchronous hyperedge replacement grammar* (SHRG), which lays out the foundation for potential tasks such graph parsing and generation. Peng et al. (2015); Peng and Gildea (2016) later proposes the first real system for AMR parsing with SHRG. The SHRG parser uses CFG for modeling natural language strings and HRG for modeling AMR graphs. A synchronous grammar is then formalized with CFG as the source side and HRG as the target side. Given such a synchronous grammar, the parser is able to parse the input sentence with CFG rules and meanwhile derive the counterpart graph by deduction from the derivation. In order to extract fine set of SHRG rules from training data, a Markov Chain Monte Carlo (MCMC) sampling algorithm is then applied to learn the most likely derivation consistent with the sentence to AMR alignments.

Combinatory Categorical Grammar (CCG) is a grammar formalism which provides a transparent interface between surface syntax and underlying semantic representation and has been extensively used in semantic parsing. Artzi et al. (2015) first propose to use CCG

formalism to parse AMR. In order to fully take the advantage of CCG formalism and maintain a relatively compact grammar, a two-stage process is introduced. First, CCG is applied to parse the input sentence into an underspecified logical form, which is a lambda-calculus equivalent of AMR representation that does not involve non-compositional reasoning. Then a factor graph model resolves the underspecified part that requires global inference. One of the advantages of CCG-based AMR parser is that it doesn't require a sentence-to-AMR alignment which avoids the error propagation from the aligner and also makes it more generalizable to other languages.

Neural AMR Parsing

The breakthrough of neural network in the field of computer vision and speech recognition has inspired significant amount of research that attempt to replicate the success in NLP community. And indeed the recent advances have demonstrated that models based on neural network are quite effective with sizable training data and have achieved state-of-the-art results on many traditional NLP tasks, ranging from syntactic parsing (Weiss et al., 2015; Dyer et al., 2015; Andor et al., 2016; Kiperwasser and Goldberg, 2016; Cross and Huang, 2016), to semantic role labeling (Roth and Lapata, 2016; Zhou and Xu, 2015), to machine translation (Luong and Manning, 2016; Wu et al., 2016).

There has been increasing amount of interests in applying neural network technique on AMR parsing. Barzdins and Gosko (2016) first applies the sequence-to-sequence model (Sutskever et al., 2014a) in neural machine translation to AMR parsing by simply treating pre-order traversal of AMR (PENMAN notation) as foreign language strings. Our work also adopts the sequence-to-sequence model for neural AMR parsing and focuses on reducing the sparsity issue in neural AMR parsing with categorization, which we will discuss in detail in Chapter 7. In contrast, Konstas et al. (2017) takes on a different path and tackles the sparsity

problem with a self-training procedure which utilizes a large set of unannotated external corpus. Foland and Martin (2016, 2017) decompose the AMR parsing task into several separate sub-tasks and handle each sub-task with Bidirectional Long Short-Term Memory (LSTM).

The simplicity and efficiency of transition-based parsing framework also makes it very appealing to combine it with neural technique. Buys and Blunsom (2017) design a generic transition-based system for semantic graph parsing and apply sequence-to-sequence framework to learn the transformation from natural language sequences to action sequences. Similarly, Misra and Artzi (2016) formalizes CCG-style AMR parsing as a shift-reduce transition system and learn it a feedforward neural network.

Other

There are also many other AMR parsing techniques which do not fall into the above paradigms. Werling et al. (2015) addresses the sparsity issue in the concept identification stage by defining a set of generative actions that map words in the sentence to their AMR concepts and use a local classifier to learn these actions. Sawai et al. (2015) adopts a similar technique and focuses on accurately resolving the Noun Phrase (NP) construction in AMR parsing.

The work from Pust et al. (2015) treats AMR parsing as a translation task and directly applies existing string-to-tree, syntax-based machine translation technique to AMR parsing.

2.3 Applications

The availability of AMR parsers has spurred a significant amount of research that aims to integrate whole-sentence semantic information into downstream NLP tasks. In this section, we introduce several prominent research work involving AMR parsing.

Summarization Liu et al. (2015) first applies AMR to text summarization and proposes an *abstractive summarization* framework that goes beyond extractive and compressive summarization. The first step is to parse each sentence to get separate AMR graphs. Then these AMR graphs are combined to form a fully dense graph, from which a *summary graph* is constructed by selecting a subset of the nodes and arcs. Finally, a heuristic generator that uses the *summary graph* as input to construct summarization results has shown promising result on a newswire dataset.

Entity Linking Pan et al. (2015) utilizes AMR in the Entity Linking (EL) task. The proposed EL system uses AMR graph as a rich semantic context for a given entity mention. Combined with a novel unsupervised graph inference algorithm, it outperforms entity linking system relied on Semantic Role Labeling (SRL) information.

Information Extraction Garg et al. (2016) exploits the effectiveness of utilizing semantic graphs in biomolecular interaction extraction. Given parsed AMR graphs of the biomedical text, they propose a graph-kernel based algorithm to score candidate interactions, which significantly outperforms a baseline system that replies on surface- and syntax-based features.

Machine Comprehension One of the motivations behind the design and production of semantic bank is to resolve the long-standing goal in NLP — Natural language Understanding (NLU). The recent Machine Comprehension task focuses on one of the facets in NLU to test the machine's ability to understand and reason with natural language. Sachan and Xing (2016) first propose a machine comprehension system integrated with AMR. Similar to the approach in (Liu et al., 2015), a graph representation for the passage and question is constructed by building up the interactions among parsed AMR graphs. And by modeling both subgraph selection and question mapping with latent variables, a unified max-margin

approach is then applied to jointly learn these latent structures.

In addition to the applications mentioned above, there is also work utilizing AMR parsing for question answering (Mitra and Baral, 2016), headline generation (Takase et al., 2016), etc.

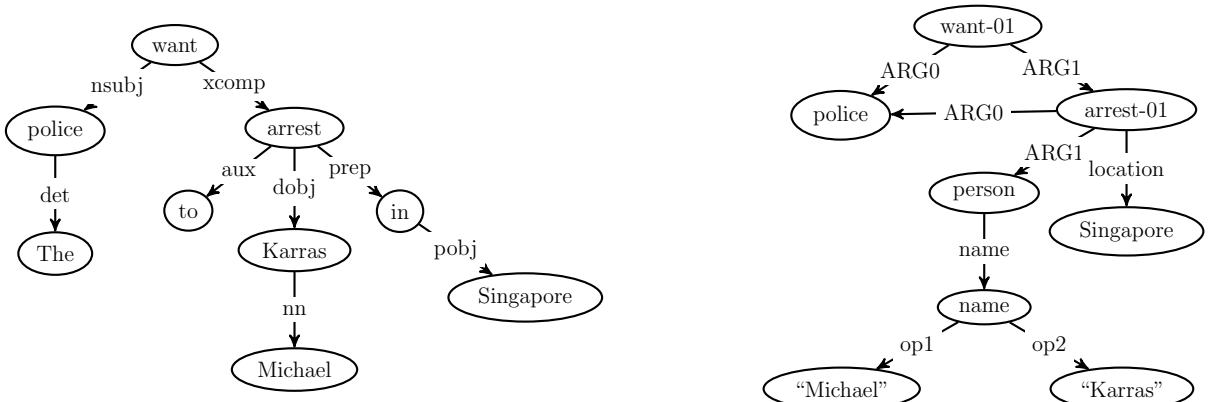
Chapter 3

Transition-based AMR Parsing

3.1 Introduction

In this chapter, we present our transition-based AMR parser, `CAMR`, which is the fundamental work of this thesis and will be discussed extensively through the following chapters. The development of our AMR parser in this chapter focuses on the graph parsing problem.

Parsing a sentence into an AMR would seem to require graph-based algorithms, but moving to graph-based algorithms from the typical tree-based algorithms that we are familiar with is a big step in terms of computational complexity. Linguistically, however, there are many similarities between an AMR and the dependency structure of a sentence. Both describe relations as holding between a head and its dependent, or between a parent and its child. AMR concepts and relations abstract away from actual word tokens, but there are regularities in their mappings. Content words generally become concepts while function words either become relations or get omitted if they do not contribute to the meaning of a sentence. This is illustrated in Figure 3.1, where 'the' and 'to' in the dependency tree are omitted from the AMR and the preposition 'in' becomes a relation of type *location*.



(a) Dependency tree

(b) AMR graph

Figure 3.1: Dependency tree and AMR graph for the sentence, “The police want to arrest Micheal Karras in Singapore.”

In AMR, reentrancy is also used to represent co-reference, but this only happens in some limited contexts. In Figure 3.1, ‘police’ is both an argument of ‘arrest’ and ‘want’ as the result of a control structure. This suggests that it is possible to transform a dependency tree into an AMR with a limited number of actions and learn a model to determine which action to take given pairs of aligned dependency trees and AMRs as training data.

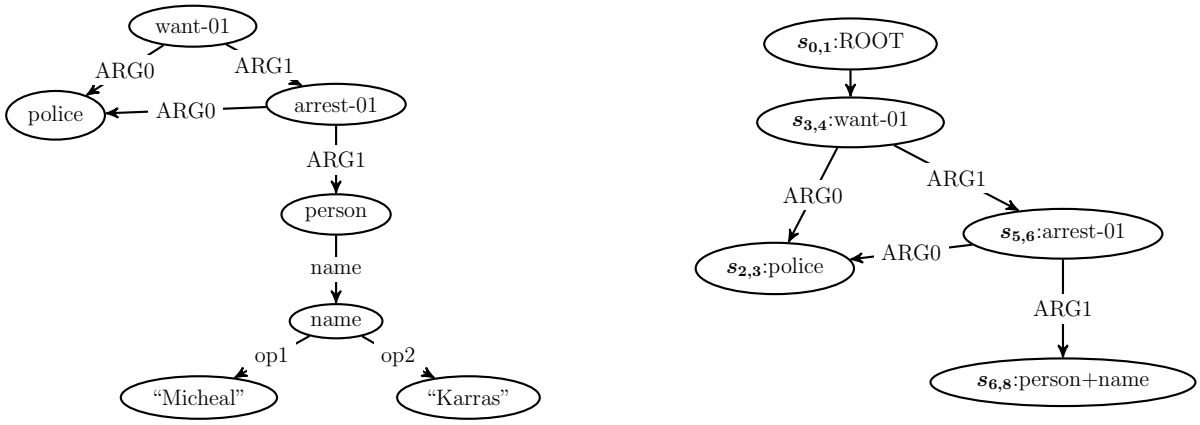
This is the approach we adopt in this chapter. We present a transition-based framework in which we parse a sentence into an AMR by taking the dependency tree of that sentence as input and transforming it to an AMR representation via a series of actions. This means that a sentence is parsed into an AMR in two steps. In the first step the sentence is parsed into a dependency tree with a dependency parser, and in the second step the dependency tree is transformed into an AMR graph. One advantage of this approach is that the dependency parser does not have to be trained on the same data set as the dependency to AMR transducer. This allows us to use more accurate dependency parsers trained on data sets much larger than the AMR Annotation Corpus and have a more advantageous starting point. Our experiments show that this approach is very effective and yields an improvement

of 5% absolute over the previously reported best result (Flanigan et al., 2014) in F-score, as measure by the Smatch metric (Cai and Knight, 2013).

The rest of the chapter is as follows. In §3.2, we describe how we align the word tokens in a sentence with its AMR to create a span graph based on which we extract contextual information as features and perform actions. In §3.3, we present our transition-based parsing algorithm and describe the actions used to transform the dependency tree of a sentence into an AMR. In §3.4, we present the learning algorithm and the features we extract to train the transition model. In §3.5, we present experimental results and we conclude in §3.6.

3.2 Graph Representation

Unlike the dependency structure of a sentence where each word token is a node in the dependency tree and there is an inherent alignment between the word tokens in the sentence and the nodes in the dependency tree, AMR is an abstract representation where the word order of the corresponding sentence is not maintained. In addition, some words become abstract concepts or relations while other words are simply deleted because they do not contribute to meaning. The alignment between the word tokens and the concepts is non-trivial, but in order to learn the transition from a dependency tree to an AMR graph, we have to first establish the alignment between the word tokens in the sentence and the concepts in the AMR. We use the aligner that comes with JAMR (Flanigan et al., 2014) to produce this alignment. The JAMR aligner attempts to greedily align every concept or graph fragment in the AMR graph with a contiguous word token sequence in the sentence.



(a) AMR graph (b) Span graph

Figure 3.2: AMR graph and its span graph for the sentence, "The police want to arrest Micheal Karras."

We use a data structure called **span graph** to represent an AMR graph that is aligned with the word tokens in a sentence. For each sentence $w = w_0, w_1, \ldots, w_n$, where token w_0 is a special root symbol, a **span graph** is a directed, labeled graph $G = (V, A)$, where $V = \{s_{i,j} | i, j \in (0, n) \text{ and } j > i\}$ is a set of nodes, and $A \subseteq V \times V$ is a set of arcs. Each node $s_{i,j}$ of G corresponds to a continuous span $(w_i, \ldots, w_{j-1})$ in sentence w and is indexed by the starting position i. Each node is assigned a **concept** label from a set L_V of concept labels and each arc is assigned a **relation** label from a set L_A of relation labels, respectively.

For example, given an AMR graph G_{AMR} in Figure 3.2a, its span graph G can be represented as Figure 3.2b. In span graph G, node $s_{3,4}$'s sentence span is *(want)* and its concept label is *want-01*, which represents a single node *want-01* in AMR. To simplify the alignment, when creating a span graph out of an AMR, we also collapse some AMR subgraphs in such a way that they can be deterministically restored to their original state for evaluation. For example, the four nodes in the AMR subgraph that correspond to span *(Micheal, Karras)* is collapsed into a single node $s_{6,8}$ in the span graph and assigned the concept label *person+name*, as shown in Figure 3.3. So the concept label set that our model predicts consists

of both those from the concepts in the original AMR graph and those as a result of collapsing the AMR subgraphs.

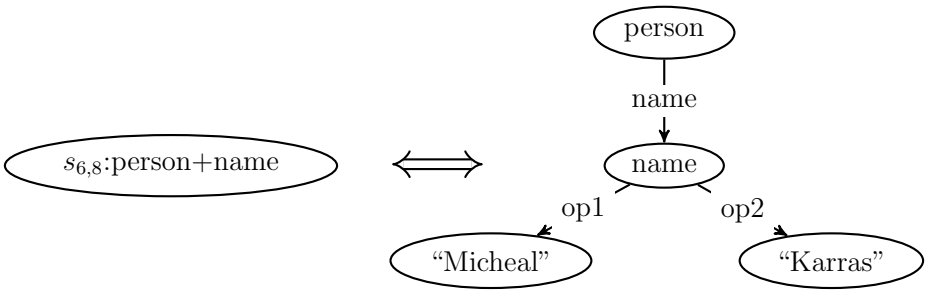


Figure 3.3: Collapsed nodes

Representing AMR graph this way allows us to formulate the AMR parsing problem as a joint learning problem where we can design a set of actions to simultaneously predict the concepts (nodes) and relations (arcs) in the AMR graph as well as the labels on them.

3.3 Transition-based Parsing Algorithm

3.3.1 Transition System

Similar to transition-based dependency parsing (Nivre, 2008), we define a *transition system* for AMR parsing as a quadruple $S = (S, T, s_0, S_t)$, where

- S is a set of parsing *states* (configurations).
- T is a set of parsing *actions* (transitions), each of which is a function $t : S \rightarrow S$.
- s_0 is an *initialization function*, mapping each input sentence w and its dependency tree D to an *initial state*.
- $S_t \subseteq S$ is a set of *terminal states*.

Each state (configuration) of our transition-based parser is a triple (σ, β, G). σ is a buffer that stores indices of the nodes which have not been processed and we write $\sigma = \sigma_0|\sigma'$ to

indicate that σ_0 is the topmost element of σ. β is also a buffer $[\beta_0, \beta_1, \ldots, \beta_j]$ and each element β_i of β indicates the edge (σ_0, β_i) which has not been processed in the partial graph. We also write $\beta = \beta_0|\beta'$ to indicate the topmost element of β is β_0. We use span graph G to store the partial parses for the input sentence w. Note that unlike traditional transition-based syntactic parsers which store partial parses in the stack structure and build a tree or graph incrementally, here we use σ and β buffers only to guide the parsing process (which node or edge to be processed next) and the actual tree-to-graph transformations are applied to G.

When the parsing procedure starts, σ is initialized with a post-order traversal of the input dependency tree D with topmost element σ_0, β is initialized with node σ_0's children or set to null if σ_0 is a leaf node. G is initialized with all the nodes and edges of D. Initially, all the nodes of G have a span length of one and all the labels for nodes and edges are set to null. As the parsing procedure goes on, the parser will process all the nodes and their outgoing edges in dependency tree D in a bottom-up left-right manner, and at each state certain action will be applied to the current node or edge. The parsing process will terminate when both σ and β are empty.

The most important part of the transition-based parser is the set of actions (transitions). As stated in (Sartorio et al., 2013), the design space of possible actions is actually infinite since the set of parsing states is infinite. However, if the problem is amenable to transition-based parsing, we can design a finite set of actions by categorizing all the possible situations we run into in the parsing process. In §3.5.2 we show this is the case here and our action set can account for almost all the transformations from dependency trees to AMR graphs.

We define 8 types of actions for the actions set T, which is summarized in Table 3.1. The action set could be divided into two categories based on conditions of buffer β. When β is not empty, parsing decisions are made based on the edge (σ_0, β_0); otherwise, only the

Action	Current state ⇒ Result state	Assign labels	Precondition
NEXT EDGE-l_r	$(\sigma_0\|\sigma', \beta_0\|\beta', G) \Rightarrow (\sigma_0\|\sigma', \beta', G')$	$\delta[(\sigma_0, \beta_0) \rightarrow l_r]$	β is not empty
SWAP-l_r	$(\sigma_0\|\sigma', \beta_0\|\beta', G) \Rightarrow (\sigma_0\|\beta_0\|\sigma', \beta', G')$	$\delta[(\beta_0, \sigma_0) \rightarrow l_r]$	
REATTACH$_k$-l_r	$(\sigma_0\|\sigma', \beta_0\|\beta', G) \Rightarrow (\sigma_0\|\sigma', \beta', G')$	$\delta[(k, \beta_0) \rightarrow l_r]$	
REPLACE HEAD	$(\sigma_0\|\sigma', \beta_0\|\beta', G) \Rightarrow (\beta_0\|\sigma', \beta = CH(\beta_0, G'), G')$	NONE	
REENTRANCE$_k$-l_r	$(\sigma_0\|\sigma', \beta_0\|\beta', G) \Rightarrow (\sigma_0\|\sigma', \beta_0\|\beta', G')$	$\delta[(k, \beta_0) \rightarrow l_r]$	
MERGE	$(\sigma_0\|\sigma', \beta_0\|\beta', G) \Rightarrow (\tilde{\sigma}\|\sigma', \beta', G')$	NONE	
NEXT NODE-l_c	$(\sigma_0\|\sigma_1\|\sigma', [], G) \Rightarrow (\sigma_1\|\sigma', \beta = CH(\sigma_1, G'), G')$	$\gamma[\sigma_0 \rightarrow l_c]$	β is empty
DELETE NODE	$(\sigma_0\|\sigma_1\|\sigma', [], G) \Rightarrow (\sigma_1\|\sigma', \beta = CH(\sigma_1, G'), G')$	NONE	

Table 3.1: Transitions designed in our parser. $CH(x, y)$ means getting all node x's children in graph y.

current node σ_0 is examined. Also, to simultaneously make decisions on the assignment of concept/relation label, we augment some of the actions with an extra parameter l_r or l_c. We define $\gamma : V \rightarrow L_V$ as the concept labeling function for nodes and $\delta : A \rightarrow L_A$ as the relation labeling function for arcs. So $\delta[(\sigma_0, \beta_0) \rightarrow l_r]$ means assigning relation label l_r to arc (σ_0, β_0). All the actions update buffer σ, β and apply some transformation $G \Rightarrow G'$ to the partial graph. The 8 actions are described below.

- NEXT-EDGE-l_r (ned). This action assigns a relation label l_r to the current edge (σ_0, β_0) and makes no further modification to the partial graph. Then it pops out the top element of buffer β so that the parser moves one step forward to examine the next edge if it exists.

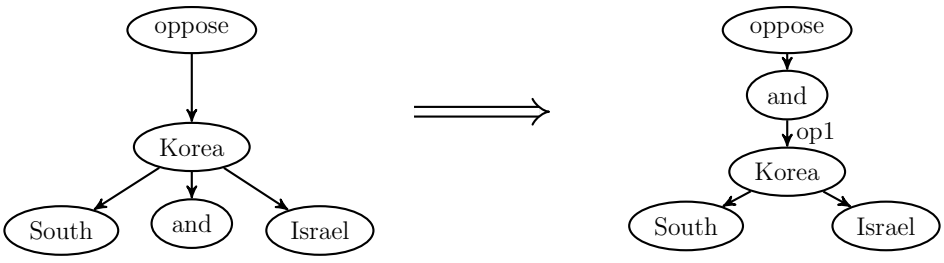


Figure 3.4: SWAP action

- SWAP-l_r (sw). This action reverses the dependency relation between node σ_0 and β_0

and then makes node β_0 as new head of the sub-graph. Also it assigns relation label l_r to the arc (β_0, σ_0). Then it pops out β_0 and inserts it into σ right after σ_0 for future revisiting. This action is to resolve the difference in the choice of head between the dependency tree and the AMR graph. Figure 3.4 gives an example of applying SWAP-op1 action for arc (*Korea*, *and*) in the dependency tree of sentence "South Korea and Israel oppose ...".

- REATTACH$_k$-l_r (reat). This action removes the current arc (σ_0, β_0) and reattaches node β_0 to some node k in the partial graph. It also assigns a relation label l_r to the newly created arc (k, β_0) and advances one step by popping out β_0. Theoretically, the choice of node k could be any node in the partial graph under the constraint that arc (k, β_0) doesn't produce a self-looping cycle. The intuition behind this action is that after swapping a head and its dependent, some of the dependents of the old head should be reattached to the new head. Figure 3.5 shows an example where node *Israel* needs to be reattached to node *and* after a head-dependent swap.

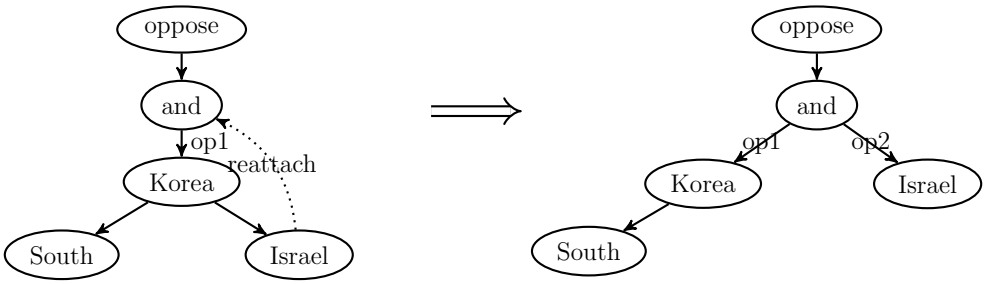


Figure 3.5: REATTACH action

- REPLACE-HEAD (rph). This action removes node σ_0, replaces it with node β_0. Node β_0 also inherits all the incoming and outgoing arcs of σ_0. Then it pops out β_0 and inserts it into the top position of buffer σ. β is re-initialized with all the children of β_0 in the transformed graph G'. This action targets nodes in the dependency tree that do

not correspond to concepts in AMR graph and become a relation instead. An example is provided in Figure 3.6, where node *in*, a preposition, is replaced with node *Singapore*, and in a subsequent NEXT-EDGE action that examines arc (*live*, *Singapore*), the arc is labeled *location*.

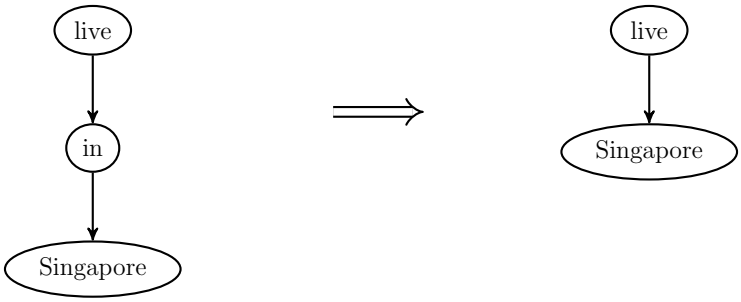


Figure 3.6: REPLACE-HEAD action

- REENTRANCE$_k$-l_r (reen). This is the action that transforms a tree into a graph. It keeps the current arc unchanged, and links node β_0 to every possible node k in the partial graph that can also be its parent. Similar to the REATTACH action, the newly created arc (k, β_0) should not produce a self-looping cycle and parameter k is bounded by the sentence length. In practice, we seek to constrain this action as we will explain in §3.3.2. Intuitively, this action can be used to model co-reference and an example is given in Figure 3.7.

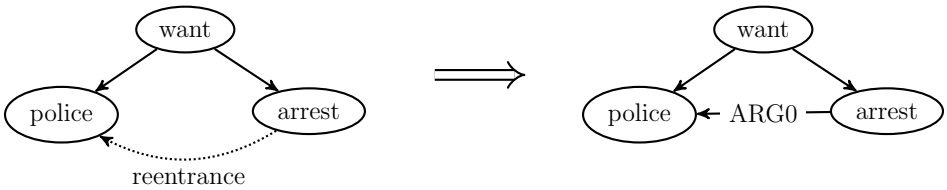


Figure 3.7: REENTRANCE action

- MERGE (mrg). This action merges nodes σ_0 and β_0 into one node $\tilde{\sigma}$ which covers multiple words in the sentence. The new node inherits all the incoming and outgoing

arcs of both nodes σ_0 and β_0. The MERGE action is intended to produce nodes that cover a continuous span in the sentence that corresponds to a single name entity in AMR graph. see Figure 3.8 for an example.

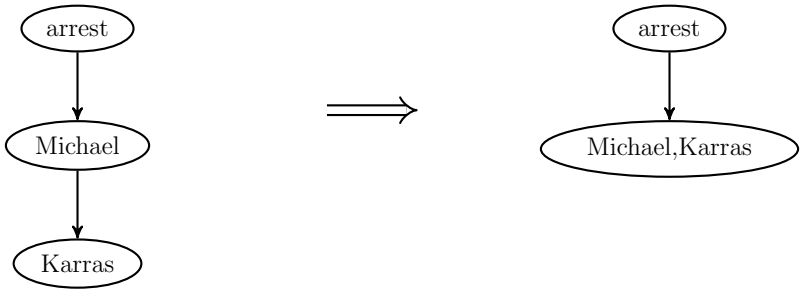


Figure 3.8: MERGE action

When β is empty, which means all the outgoing arcs of node σ_0 have been processed or σ_0 has no outgoing arcs, the following two actions can be applied:

- NEXT-NODE-l_c (nnd). This action first assigns a concept label l_c to node σ_0. Then it advances the parsing procedure by popping out the top element σ_0 of buffer σ and re-initializes buffer β with all the children of node σ_1 which is the current top element of σ. Since this action will be applied to every node which is kept in the final parsed graph, concept labeling could be done simultaneously through this action.
- DELETE-NODE (dnd). This action simply deletes the node σ_0 and removes all the arcs associated with it. This action models the fact that most function words are stripped off in the AMR of a sentence. Note that this action only targets function words that are leaves in the dependency tree, and we constrain this action by only deleting nodes which do not have outgoing arcs.

When parsing a sentence of length n (excluding the special root symbol w_0), its corresponding dependency tree will have n nodes and $n-1$ arcs. For projective transition-based

dependency parsing, the parser needs to take exactly $2n - 1$ steps or actions. So the complexity is $O(n)$. However, for our tree-to-graph parser defined above, the actions needed are no longer linearly bounded by the sentence length. Suppose there are no REATTACH, REENTRANCE and SWAP actions during the parsing process, the algorithm will traverse every node and edge in the dependency tree, which results in $2n$ actions. However, REATTACH and REENTRANCE actions would add extra edges that need to be re-processed and the SWAP action adds both nodes and edges that need to be re-visited. Since the space of all possible extra edges is $(n-2)^2$ and re-visiting them only adds more actions linearly, the total asymptotic runtime complexity of our algorithm is $O(n^2)$.

In practice, however, the number of applications of the REATTACH action is much less than the worst case scenario due to the similarities between the dependency tree and the AMR graph of a sentence. Also, nodes with reentrancies in AMR only account for a small fraction of all the nodes, thus making the REENTRANCE action occur at constant times. These allow the tree-to-graph parser to parse a sentence in nearly linear time in practice.

3.3.2 Greedy Parsing Algorithm

Algorithm 1 Parsing algorithm

Input: sentence $w = w_0 \dots w_n$ and its dependency tree D_w
Output: parsed graph G_p

$s \leftarrow s_0(D_w, w)$
while $s \notin S_t$ **do**
    $\mathcal{T} \leftarrow$ all possible actions according to s
    $bestT \leftarrow \arg\max_{t \in \mathcal{T}} score(t, c)$
    $s \leftarrow$ apply $bestT$ to s
end while
return G_p

Our parsing algorithm is similar to the parser in (Sartorio et al., 2013). At each parsing

state $s \in S$, the algorithm greedily chooses the parsing action $t \in T$ that maximizes the score function $score()$. The score function is a linear model defined over parsing action t and parsing state s.

$$score(t, s) = \vec{\omega} \cdot \phi(t, s) \tag{3.1}$$

where $\vec{\omega}$ is the weight vector and ϕ is a function that extracts the feature vector representation for one possible state-action pair $\langle t, s \rangle$.

First, the algorithm initializes the state s with the sentence w and its dependency tree D_w. At each iteration, it gets all the possible actions for current state s (line 3). Then, it chooses the action with the highest score given by function $score()$ and applies it to s (line 4-5). When the current state reaches a terminal state, the parser stops and returns the parsed graph.

As pointed out in (Bohnet and Nivre, 2012), constraints can be added to limit the number of possible actions to be evaluated at line 3. There could be formal constraints on states such as the constraint that the SWAP action should not be applied twice to the same pair of nodes. We could also apply soft constraints to filter out unlikely concept labels, relation labels and candidate nodes k for REATTACH and REENTRANCE. In our parser, we enforce the constraint that NEXT-NODE-l_c can only choose from concept labels that co-occur with the current node's lemma in the training data. We also empirically set the constraint that REATTACH$_k$ could only choose k among σ_0's grandparents and great grandparents. Additionally, REENTRANCE$_k$ could only choose k among its siblings. These constraints greatly reduce the search space, thus speeding up the parser.

3.4 Learning

3.4.1 Learning Algorithm

As stated in section 3.3.2, the parameter of our model is weight vector $\vec{\omega}$ in the score function. To train the weight vector, we employ the averaged perceptron learning algorithm (Collins, 2002).

Algorithm 2 Learning algorithm

Input: sentence $w = w_0 \dots w_n, D_w, G_w$
Output: $\vec{\omega}$

1: $s \leftarrow s_0(D_w, w)$
2: **while** $s \notin S_t$ **do**
3: $\quad \mathcal{T} \leftarrow$ all possible actions according to s
4: $\quad bestT \leftarrow \arg\max_{t \in \mathcal{T}} score(t, s)$
5: $\quad goldT \leftarrow oracle(s, G_w)$
6: $\quad$ **if** $bestT \neq goldT$ **then**
7: $\quad\quad \vec{\omega} \leftarrow \vec{\omega} - \phi(bestT, s) + \phi(goldT, s)$
8: $\quad$ **end if**
9: $\quad s \leftarrow$ apply $goldT$ to s
10: **end while**

The pseudo code for learning algorithm is illustrated in Algorithm 2: For each sentence w and its corresponding AMR annotation G_{AMR} in the training corpus, we could get the dependency tree D_w of w with a dependency parser. Then we represent G_{AMR} as span graph G_w, which serves as our learning target. The learning algorithm takes the training instances (w, D_w, G_w), parses D_w according to Algorithm 1, and get the best action using current weight vector $\vec{\omega}$. The gold action for current state s is given by consulting span graph G_w, which we formulate as a function *oracle*() (line 5). If the gold action is equal to the best action we get from the parser, then the best action is applied to current state; otherwise, we update the weight vector (line 6-7) and continue the parsing procedure by applying the gold action.

3.4.2 Feature Extraction

For transition-based dependency parsers, the feature context for a parsing state is represented by the neighboring elements of a word token in the stack containing the partial parse or the buffer containing unprocessed word tokens. In contrast, in our tree-to graph parser, as already stated, buffers σ and β only specify which arc or node is to be examined next. The feature context associated with current arc or node is mainly extracted from the partial graph G. As a result, the feature context is different for the different types of actions, a property that makes our parser very different from a standard transition-based dependency parser. For example, when evaluating action SWAP we may be interested in features about individual nodes σ_0 and β_0 as well as features involving the arc (σ_0, β_0). In contrast, when evaluating action REATTACH$_k$, we want to extract not only features involving σ_0 and β_0, but also information about the reattached node k. To address this problem, we define the feature context as $\langle \bar{\sigma}_0, \bar{\beta}_0, \bar{k}, \bar{\sigma}_{0p} \rangle$, where each element $\bar{x}$ consists of its atomic features of node x and σ_{0p} denotes the immediate parent of node σ_0. For elements in feature context that are not applicable to the candidate action, we just set the element to NONE and only extract features which are valid for the candidate action. The list of features we use is shown in Table 3.2.

Single node features are atomic features concerning all the possible nodes involved in each candidate state-action pair. We also include path features and distance features as described in (Flanigan et al., 2014). A path feature $path_{x,y}$ is represented as the dependency labels and parts of speech on the path between nodes x and y in the partial graph. Here we combine it with the lemma of the starting and ending nodes. Distance feature $dist_{x,y}$ is the number of tokens between two node x, y's spans in the sentence. Action-specific features record the history of actions applied to a given node. For example, $\bar{\beta}_0.nswp$ records how

many times node β_0 has been swapped up. We combine this feature with the lemma of node β_0 to prevent the parser from swapping a node too many times. $\bar{\beta}_0.reph$ records the word feature of nodes that have been replaced with node β_0. This feature is helpful in predicting relation labels. As we have discussed above, in an AMR graph, some function words are deleted as nodes but they are crucial in determining the relation label between its child and parent.

Single node features
$\bar{\sigma}_0.w, \bar{\sigma}_0.lem, \bar{\sigma}_0.ne, \bar{\sigma}_0.t, \bar{\sigma}_0.dl, \bar{\sigma}_0.len$
$\bar{\beta}_0.w, \bar{\beta}_0.lem, \bar{\beta}_0.ne, \bar{\beta}_0.t, \bar{\beta}_0.dl, \bar{\beta}_0.len$
$\bar{k}.w, \bar{k}.lem, \bar{k}.ne, \bar{k}.t, \bar{k}.dl, \bar{k}.len$
$\bar{\sigma}_{0p}.w, \bar{\sigma}_{0p}.lem, \bar{\sigma}_{0p}.ne, \bar{\sigma}_{0p}.t, \bar{\sigma}_{0p}.dl$

Node pair features
$\bar{\sigma}_0.lem + \bar{\beta}_0.t, \bar{\sigma}_0.lem + \bar{\beta}_0.dl$
$\bar{\sigma}_0.t + \bar{\beta}_0.lem, \bar{\sigma}_0.dl + \bar{\beta}_0.lem$
$\bar{\sigma}_0.ne + \bar{\beta}_0.ne, \bar{k}.ne + \bar{\beta}_0.ne$
$\bar{k}.t + \bar{\beta}_0.lem, \bar{k}.dl + \bar{\beta}_0.lem$

Path features
$\bar{\sigma}_0.lem + \bar{\beta}_0.lem + path_{\sigma_0,\beta_0}$
$\bar{k}.lem + \bar{\beta}_0.lem + path_{k,\beta_0}$

Distance features
$dist_{\sigma_0,\beta_0}$, $dist_{k,\beta_0}$
$dist_{\sigma_0,\beta_0} + path_{\sigma_0,\beta_0}$
$dist_{\sigma_0,\beta_0} + path_{k,\beta_0}$

Action specific features
$\bar{\beta}_0.lem + \bar{\beta}_0.nswp$
$\bar{\beta}_0.reph$

Table 3.2: Features used in our parser. $\bar{\sigma}_0, \bar{\beta}_0, \bar{k}, \bar{\sigma}_{0p}$ represents elements in feature context of nodes $\sigma_0, \beta_0, k, \sigma_{0p}$, separately. Each atomic feature is represented as follows: w - word; lem - lemma; ne - name entity; t - POS-tag; dl - dependency label; len - length of the node's span.

3.5 Experiments

3.5.1 Experiment Setting

Our experiments are conducted on the newswire section of corpus LDC2013E117 (Banarescu et al., 2013). We follow Flanigan et al. (2014) in setting up the train/development/test splits for easy comparison: 4.0k sentences with document years 1995-2006 as the training set; 2.1k sentences with document year 2007 as the development set; 2.1k sentences with document

year 2008 as the test set. Each sentence w is preprocessed with the Stanford CoreNLP toolkit (Manning et al., 2014) to get part-of-speech tags, name entity information, and basic dependencies. We have verified that there is no overlap between the training data for the Stanford CoreNLP toolkit[1] and the AMR Annotation Corpus. We evaluate our parser with the Smatch tool (Cai and Knight, 2013), which seeks to maximize the semantic overlap between two AMR annotations.

3.5.2 Action Set Validation

One question about the transition system we presented above is whether the action set defined here can cover all the situations involving a dependency-to-AMR transformation. Although a formal theoretical proof is beyond the scope of this dissertation, we can empirically verify that the action set works well in practice. To validate the actions, we first run the *oracle*() function for each sentence w and its dependency tree D_w to get the "pseudo-gold" G'_w. Then we compare G'_w with the gold-standard AMR graph represented as span graph G_w to see how similar they are. On the training data we got an overall 99% F-score for all $\langle G'_w, G_w \rangle$ pairs, which indicates that our action set is capable of transforming each sentence w and its dependency tree D_w into its gold-standard AMR graph through a sequence of actions.

3.5.3 Results

Table 3.3 gives the precision, recall and F-score of our parser given by Smatch on the test set. Our parser achieves an F-score of 63% (Row 3) and the result is 5% better than the first published result reported in (Flanigan et al., 2014) with the same training and test set

[1]Specifically we used CoreNLP toolkit v3.3.1 and parser model wsjPCFG.ser.gz trained on the WSJ treebank sections 02-21.

(Row 2). We also conducted experiments on the test set by replacing the parsed graph with gold relation labels or/and gold concept labels. We can see in Table 3.3 that when provided with gold concept and relation labels as input, the parsing accuracy improves around 8% F-score (Row 6). Rows 4 and 5 present results when the parser is provided with just the gold relation labels (Row 4) or gold concept labels (Row 5), and the results are expectedly lower than if both gold concept and relation labels are provided as input.

	Precision	Recall	F-score
JAMR (2014)	.52	.66	.58
Our parser	**.64**	.62	**.63**
Our parser $+l_{gr}$	.68	.65	.67
Our parser $+l_{gc}$	.69	.67	.68
Our parser $+l_{grc}$	.72	.70	.71

Table 3.3: Results on the test set. Here, l_{gc} - gold concept label; l_{gr} - gold relation label; l_{grc} - gold concept label and gold relation label.

3.5.4 Error Analysis

Wrong alignments between the word tokens in the sentence and the concepts in the AMR graph account for a significant proportion of our AMR parsing errors, but here we focus on errors in the transition from the dependency tree to the AMR graph. Since in our parsing model, the parsing process has been decomposed into a sequence of actions applied to the input dependency tree, we can use the $oracle()$ function during parsing to give us the correct action t_g to take for a given state s. A comparison between t_g and the best action t actually taken by our parser will give us a sense about how accurately each type of action is applied. When we compare the actions, we focus on the structural aspect of AMR parsing and only take into account the eight action types, ignoring the concept and edge labels attached to them. For example, NEXT-EDGE-$ARG0$ and NEXT-EDGE-$ARG1$ would be considered to

be the same action and counted as a match when we compute the errors even though the labels attached to them are different.

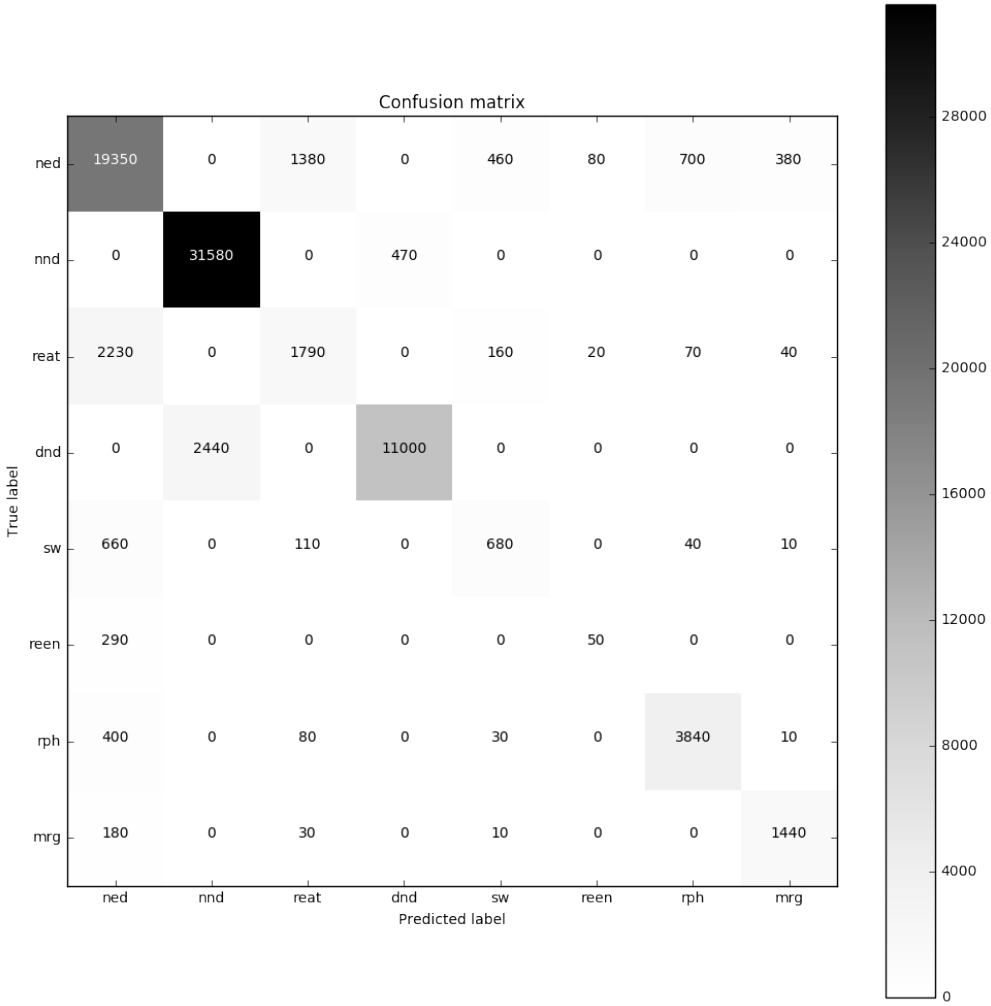


Figure 3.9: Confusion Matrix for actions $\langle t_g, t \rangle$. Vertical direction goes over the correct action type, and horizontal direction goes over the parsed action type.

Figure 3.9 shows the confusion matrix that presents a comparison between the parser-predicted actions and the correct actions given by *oracle*() function. It shows that the NEXT-EDGE (ned), NEXT-NODE (nnd), and DELETENODE (dnd) actions account for a large proportion of the actions. These actions are also more accurately applied. As expected,

the parser makes more mistakes involving the REATTACH (reat), REENTRANCE (reen) and SWAP (sw) actions. The REATTACH action is often used to correct PP-attachment errors made by the dependency parser or readjust the structure resulting from the SWAP action, and it is hard to learn given the relatively small AMR training set. The SWAP action is often tied to coordination structures in which the head in the dependency structure and the AMR graph diverges. In the Stanford dependency representation which is the input to our parser, the head of a coordination structure is one of the conjuncts. For AMR, the head is an abstract concept signaled by one of the coordinating conjunctions. This also turns out to be one of the more difficult actions to learn. We expect, however, as the AMR Annotation Corpus grows bigger, the parsing model trained on a larger training set will learn these actions better.

3.6 Conclusion

We presented a novel transition-based parsing algorithm that takes the dependency tree of a sentence as input and transforms it into an Abstract Meaning Representation graph through a sequence of actions. We show that our approach is linguistically intuitive and our experimental results also show that our parser outperformed the previous best reported results by a significant margin. In next chapter, we continue to perfect our parser via improved feature engineering and enhanced transition system.

Chapter 4

Enhanced Transition-based AMR Parsing

4.1 Introduction

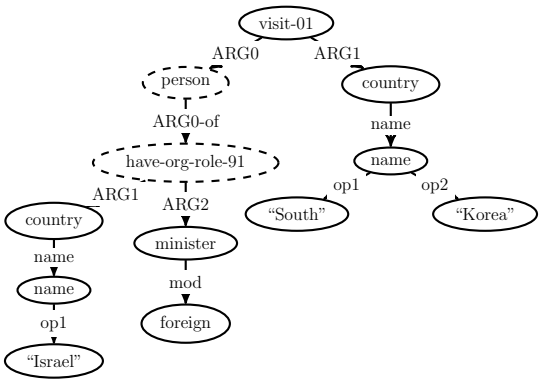


Figure 4.1: An example showing abstract concept `have-org-role-91` for the sentence "Israel foreign minister visits South Korea."

As we have discussed in Chapter 2, unlike a dependency parse where each leaf node corresponds to a word in a sentence and there is an inherent alignment between the words in a

sentence and the leaf nodes in the parse tree, the alignment between the word tokens in a sentence and the concepts in an AMR graph is non-trivial. Both JAMR and our transition-based parser rely on a heuristics based aligner that can align the words in a sentence and concepts in its AMR with a 90% F_1 score, but there are some concepts in the AMR that cannot be aligned to any word in a sentence. This is illustrated in Figure 4.1 where the concept `have-org-role-91` is not aligned to any word or word sequence. We refer to these concepts as **abstract** concepts, and our baseline AMR parser do not have a systematic way of inferring such abstract concepts.

Another apparent issue in our baseline parser mentioned in last chapter is that its feature set is yet to be explored. There are several linguistic analyzers which could be potentially beneficial for AMR parsing. For example, the AMR makes heavy use of the framesets and semantic role labels used in the Proposition Bank (Palmer et al., 2005), and it would seem that information produced by a semantic role labeling system trained on the PropBank can be used as features to improve the AMR parsing accuracy. Similarly, since AMR represents limited within-sentence coreference, coreference information produced by an off-the-shelf coreference system should benefit the AMR parser as well.

In addition, the release of AMR annotation (LDC2015E86) in SemEval 2016 Task 8 (May, 2016) also requires the parser to predict the entity link which associates named entity in AMR graph with its wikipedia entry. We apply an existing wikifier (Pan et al., 2015) to add the wiki links as a post-processing step. A more fine-grained named entity tag information and utilization of external resource list are also explored for boosting the performance of AMR parsing.

In this chapter, we first describe an extension to our baseline AMR parser by adding a new action to infer the abstract concepts in an AMR. Then, a comprehensive experiment is conducted to examine the effectiveness of various external features from linguistic knowledge,

resource list and unsupervised word cluster. Additionally, we experiment with using different syntactic parsers in the first stage.

Our results show that (i) the transition-based AMR parser is very stable across the different parsers used in the first stage, (ii) adding the new action significantly improves the parser performance, (iii) semantic role information is beneficial to AMR parsing when used as features, and (iv) the Brown clusters do not make a difference while coreference information slightly hurts the AMR parsing performance.

The rest of this chapter is organized as follows. In section 4.2 we describe how to infer abstract concepts. Section 4.3 illustrates the detail of various features we have examined. We report experimental results in Section 4.4 and summarize in Section 4.5.

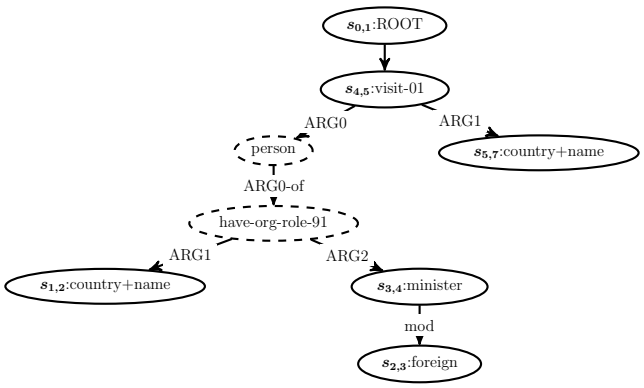


Figure 4.2: Enhanced Span Graph for AMR in Figure 1.1, "Israel foreign minister visits South Korea." $s_{x,y}$ corresponds to sentence span (x, y).

4.2 Inferring Abstract Concepts

We previously create the learning target by representing an AMR graph as a **Span Graph**, where each AMR concept is annotated with the text span of the word or the (contiguous) word sequence it is aligned to. However, abstract concepts that are not aligned to any word or word sequence are simply ignored and are unreachable during training. To address this,

we construct the span graph by keeping the abstract concepts as they are in the AMR graph, as illustrated in Figure 4.2.

In order to predict these abstract concepts, we design an INFER-l_c action that is applied in the following way: when the parser visits an node in dependency tree, it inserts an abstract node with concept label l_c right between the current node and its parent. For example in Figure 4.3, after applying action INFER-`have-org-role-91` on node *minister*, the abstract concept is recovered and subsequent actions can be applied to transform the subgraph to its correct AMR.

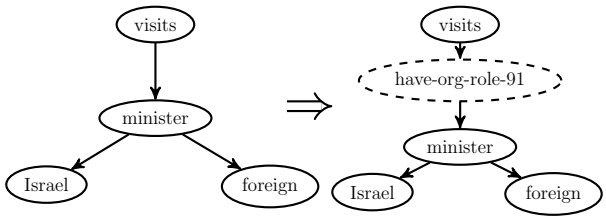


Figure 4.3: INFER-`have-org-role-91` action

4.3 Feature Enrichment

In our previous work, we only use simple lexical features and structural features. We extend the feature set to include (i) features generated by a semantic role labeling system—ASSERT (Pradhan et al., 2004), including a frameset disambiguator trained using a word sense disambiguation system—IMS (Zhong and Ng, 2010) and a coreference system (Lee et al., 2013), (ii) features generated using semi-supervised word clusters (Turian et al., 2010; Koo et al., 2008), and (iii) fine-grained named entity information and utilization of verbalization list.

Coreference features Coreference is typically represented as a chain of mentions realized as noun phrases or pronouns. AMR, on the other hand, represents coreference as re-entrance and uses one concept to represent all co-referring entities. To use the coreference information to inform AMR parsing actions, we design the following two features: 1) SHARE_DEPENDENT. When applying REENTRANCE$_k$-l_r action on edge (a, b), we check whether the corresponding head node k of a candidate concept has any dependent node that co-refers with current dependent b. 2) DEPENDENT_LABEL. If SHARE_DEPENDENT is true for head node k and assuming k's dependent m co-refers with the current dependent, the value of this feature is set to the dependency label between k and m.

For example, for the partial graph shown in Figure 4.4, when examining edge $(wants, boy)$, we may consider REENTRANCE$_{believe}$-ARG1 as one of the candidate actions. The candidate head *believe* has dependent *him* which is co-referred with current dependent *boy*, therefore the value of feature SHARE_DEPENDENT is set to true for this candidate action. Also the value of feature DEPENDENT_LABEL is dobj given the dependency label between $(believe, him)$.

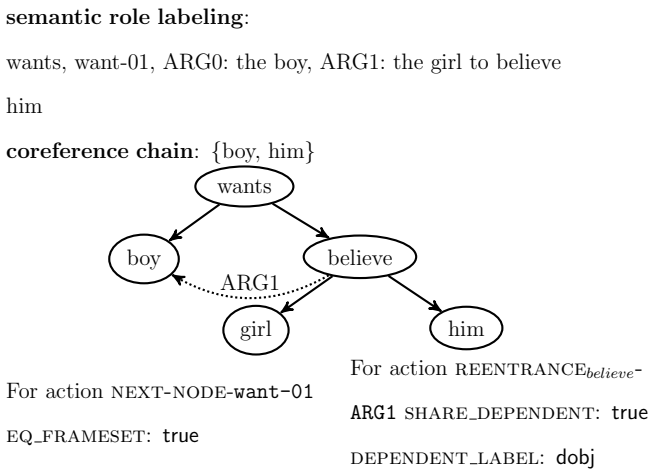


Figure 4.4: An example of coreference feature and semantic role labeling feature in partial parsing graph of sentence,"The boy wants the girl to believe him."

Semantic role labeling features We use the following semantic role labeling features: 1) EQ_FRAMESET. For action that predicts the concept label (NEXT-NODE-l_c), we check whether the candidate concept label l_c matches the frameset predicted by the semantic role labeler. For example, for partial graph in Figure 4.4, when we examining node *wants*, one of the candidate actions would be NEXT-NODE-`want-01`. Since the candidate concept label `want-01` is equal to node *wants*'s frameset `want-01` as predicted by the semantic role labeler, the value of feature EQ_FRAMESET is set to `true`. 2) IS_ARGUMENT. For actions that predicts the edge label, we check whether the semantic role labeler predicts that the current dependent is an argument of the current head.

Word Clusters For the semi-supervised word cluster feature, we use Brown clusters, more specifically, 1000 classes word cluster trained by Turian et al. (2010). We use the prefixes of lengths 4,6,10,20 of the word's bit-string as features.

Rich named entity tags Since named entity types in AMR are much more fine-grained than the named entity types defined in a typical named entity tagging system, we assume that using a richer named entity tagger could improve concept identification in parsing. Here we use the 18 named entity types defined in the OntoNotes v5.0 Corpus Pradhan et al. (2013).

The ISI verbalization list A large proportion of AMR concepts are "normalized" English words. This typically involves cases where the verb form of a noun or an adjective is used as the AMR concept. For example, the AMR concept "attract-01" is used for the adjective "attractive". Similarly, the noun "globalization" would invoke the AMR concept "globalize-01". To help CAMR produce these AMR concepts correctly, we use the verbalization-list

provided by ISI[1] to improve the word-to-AMR-concepts alignment. If any alignment is missed by the JAMR aligner and left un-aligned, we simply add an alignment to map the unaligned concept to its corresponding word token if the word token in the input sentence is in the verbalization list.

4.4 Experiments

We first tune and evaluate our system on the newswire section of LDC2013E117 dataset. Then we show our parser's performance on the LDC2014T12 dataset.

4.4.1 Experiments on LDC2013E117

We first conduct our experiments on the newswire section of AMR annotation corpus (LDC-2013E117). The train/dev/test split of dataset is 4.0K/2.1K/2.1K, which is identical to the settings of JAMR. We evaluate our parser with Smatch v2.0 (Cai and Knight, 2013) on all the experiments.

Impact of different syntactic parsers

We experimented with four different parsers: the Stanford parser (Manning et al., 2014), the Charniak parser (Charniak and Johnson, 2005) (Its phrase structure output is converted to dependency structure using the Stanford CoreNLP converter), the Malt Parser (Nivre et al., 2006), and the Turbo Parser (Martins et al., 2013). All the parsers we used are trained on the 02-22 sections of the Penn Treebank, except for CHARNIAK(ON), which is trained on the OntoNotes corpus (Hovy et al., 2006) on the training and development partitions used by Pradhan et al. (2013) after excluding a few documents that overlapped with the AMR

[1] http://amr.isi.edu/download/lists/verbalization-list-v1.01.txt

corpus[2]. All the different dependency trees are then used as input to our baseline system and we evaluate AMR parsing performance on the development set.

System	P	R	F_1
Charniak (ON)	**.67**	**.64**	**.65**
Charniak	.66	.62	.64
Stanford	.64	.62	.63
Malt	.65	.61	.63
Turbo	.65	.61	.63

Table 4.1: AMR parsing performance on development set using different syntactic parsers.

From Table 4.1, we can see that the performance of the baseline transition-based system remains very stable when different dependency parsers used are trained on same data set. However, the Charniak parser that is trained on a much larger and more diverse dataset (CHARNIAK (ON)) yields the best overall AMR parsing performance. Subsequent experiments are all based on this version of the Charniak parser.

Impact of parser extensions

System	P	R	F_1
Charniak (ON)	.67	.64	.65
+INFER	.71	.67	.69
+INFER+BROWN	.71	.68	.69
+INFER+BROWN+SRL	**.72**	**.69**	**.71**
+INFER+BROWN+SRL+COREF	.72	.69	.70

Table 4.2: AMR parsing performance on the development set.

In Table 4.2 we present results from extending the transition-based AMR parser. All

[2]Documents in the AMR corpus have some overlap with the documents in the OntoNotes corpus. We excluded these documents (which are primarily from Xinhua newswirte) from the training data while retraining the Charniak parser, ASSERT semantic role labeler, and IMS frameset disambiguation tool). The full list of overlapping documents is available at `http://cemantix.org/ontonotes/ontonotes-amr-document-overlap.txt`

experiments are conducted on the development set. From Table 4.2, we can see that the INFER action yields a 4 point improvement in F_1 score over the CHARNIAK(ON) system. Adding Brown clusters improves the recall by 1 point, but the F_1 score remains unchanged. Adding semantic role features on top of the Brown clusters leads to an improvement of another 2 points in F_1 score, and gives us the best result. Adding coreference features actually slightly hurts the performance.

Final Result on Test Set

We evaluate the best model we get from §4.4.1 on the test set, as shown in Table 4.3. For comparison purposes, we also include results of three published parsers on the same dataset: the updated version of JAMR, the old version of JAMR (Flanigan et al., 2014), the Stanford AMR parser (Werling et al., 2015), the SHRG-based AMR parser (Peng et al., 2015) and our baseline parser (Wang et al., 2015b). From Table 7.2 we can see that our parser has

System	P	R	F_1
Extended Parser	**.71**	**.69**	**.70**
JAMR (GitHub)[3]	.69	.58	.63
JAMR (Flanigan et al., 2014)	.66	.52	.58
Stanford	.66	.59	.62
SHRG-based	.59	.58	.58
Wang et al. (2015b)	.64	.62	.63

Table 4.3: AMR parsing performance on the newswire test set of LDC2013E117.

significant improvement over all the other parsers and outperforms the previous best parser by 7% points in Smatch score.

4.4.2 Experiments on LDC2014T12

In this section, we conduct experiments on the AMR annotation release 1.0 (LDC2014T12), which contains 13,051 AMRs from newswire, weblogs and web discussion forums. We use the training/development/test split recommended in the release: 10,312 sentences for training, 1,368 sentences for development and 1,371 sentences for testing. We re-train the parser on the LDC2014T12 training set with the best parser configuration given in §4.4.1, and test the parser on the test set. The result is shown in Table 4.4. For comparison purposes, we also include the results of the updated version of JAMR and our baseline parser in Wang et al. (2015b) which are also trained on the same dataset. There is a significant drop-off in performance compared with the results on the LDC2013E117 test set for all the parsers, but our parser outperforms the other parsers by a similar margin on both test sets.

System	P	R	F
Extended Parser	**.70**	**.62**	**.66**
Wang et al. (2015b)	.63	.56	.59
JAMR (GitHub)	.64	.53	.58

Table 4.4: AMR parsing performance on the full test set of LDC2014T12.

We also evaluate our parser on the newswire section of LDC2014T12 dataset. Table 4.5 compares the performance of JAMR, the Stanford AMR parser and our system on the same dataset.

System	P	R	F
Extended Parser	**.72**	**.67**	**.70**
Stanford	.67	.58	.62
JAMR (GitHub)	.67	.53	.59

Table 4.5: AMR parsing performance on newswire section of LDC2014T12 test set

And our system still outperforms the other parsers by a similar margin.

4.5 Conclusion

We presented extensions to our transition-based AMR parser and try to resolve the low-hanging fruit in AMR parsing. Our extensions lead to an improvement of 7% in absolute F_1 score over the our baseline results, which include designing a new action to infer abstract concepts and training the parser with additional semantic role labeling and coreference based features.

Chapter 5

AMR Parsing with Neural Concept Identifier

5.1 Introduction

Based on our transition-based AMR parsing framework, we first start off to resolve the low-hanging fruit, such as exploring feature enrichment and addressing hallucinated concept. However, as we have a more comprehensive understanding of the task, we find out several sub-tasks are especially crucial to AMR parsing and deserve much more attention. In this Chapter, we break down the AMR parsing process and focus on improving one of the sub-task — Concept Identification. Previous research has shown that concept identification is the bottleneck to further improvement in AMR parsing. For example, JAMR (Flanigan et al., 2014), the first AMR parser, is able to achieve an F-score of 80% (close to the inter-annotator agreement of 83) if gold concepts are provided. Its parsing accuracy drops sharply to 62.3% when the concepts are identified automatically.

One of the challenges in AMR concept identification is data sparsity. A large portion

of AMR's concepts are either word lemmas or sense-disambiguated lemmas drawn from Propbank (Palmer et al., 2005). Since the AMR Bank is relatively small, many of the concept labels in development set or test set only occur a few times or never appear in the training set.

One common approach to concept identification is to construct a look-up table that stores the mapping between spans of tokens in a sentence and its candidate AMR graph fragments or concept labels extracted from the training set (Flanigan et al., 2014; Wang et al., 2015b; Zhou et al., 2016), then use a local classifier to select the best candidates. Werling et al. (2015) has shown that 38% of the words in the LDC2014E113 development set are unseen in the training set, and the gold concepts for these words will be unreachable during decoding time. Werling et al. (2015) addresses this problem by defining a set of *generative* actions that maps words in the sentence to their AMR concepts and use a local classifier to learn these actions.

Given such sparse data, making full use of contextual information is crucial to accurate concept labeling. Bidirectional LSTM has shown its success on many sequence labeling tasks since it is able to combine contextual information from both directions and avoid manual feature engineering. However, it is non-trivial to formalize concept identification as a sequence labeling problem because the large concept label set. Inspired by Foland and Martin (2017), who first apply the Bidirectional LSTM to AMR concept identification by categorizing the large labels into a finite set of predefined types, we propose to address concept identification using Bidirectional LSTM with **Factored Concept Labels (FGL)**, where we re-group the concept label set based on their shared graph structure. This makes it possible for different concepts to be represented by one common label that captures the shared semantics of these concepts.

We present experimental results that show incorporating the Bidirectional LSTM concept

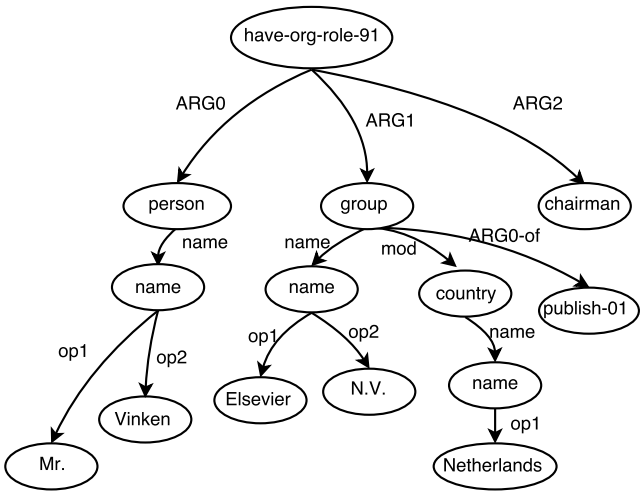


Figure 5.1: AMR graph for sentence: "*Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.*".

identifier to our transition-base AMR parsing framework results in consistently improved Smatch scores on various datasets. The rest of chapter is organized as follows. Section 5.2 describes our improved LSTM based concept identification model. We present experimental results in Section 5.3, and conclude in Section 5.4.

5.2 Concept Identification with Bidirectional LSTM

In this section, we first introduce how we categorize AMR concepts using **Factored Concept Labels**. We then integrate character-level information into Bidirectional LSTM through a Convolutional Neural Network (CNN)-based embedding.

5.2.1 Background and Notation

Given a pair of AMR graph G and English sentence S, a look-up table M is first generated which maps a span of tokens to concepts using an aligner. Although there are differences

among results generated by different aligners, in general, the aligned AMR concepts can be classified into the following types:

- PREDICATE. Concepts with sense tag belong to this case, which are frames borrowed from Propbank. Most of the tokens aligned to this type are verbs and nouns that trigger events.
- NON-PREDICATE. This type of concepts are mostly taken from the original sentences.
- CONST. Most of the quantity related tokens in sentence are aligned to this type, where AMR concepts are normalized numbers.
- MULTICONCEPT. In this type, one or more tokens in sentence are aligned to multiple concepts, where they form a sub-structure in AMR graph. The most frequent case is named entity subgraph. For example, in Figure 5.1, *"Mr. Vinken"* is aligned to subgraph `(p / person :name (m / name :op1 "Mr." :op2 "Vinken")`.

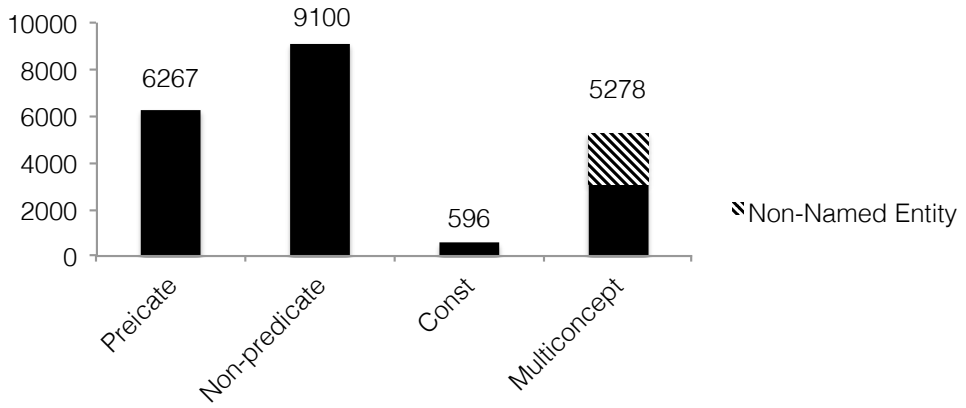


Figure 5.2: AMR concept label distribution for development set of LDC2015E86

5.2.2 Factored Concept Labels

To be able to fit AMR's large concept label space into a sequence labeling framework, redefining the label set is necessary in order to make the learning process efficient. While it is trivial to categorize PREDICATE, NON-PREDICATE, CONST, there is no straightforward way to deal with MULTICONCEPT type. Foland and Martin (2016) only handle named entities, which constitute the majority of the MULTICONCEPT cases, where they adopt BIOES tags to detect the boundary and use an additional Bidirectional LSTM to learn the fine-grained named entity concept type. For other MULTICONCEPT cases, they only use the leaf concept and ignore internal structures. Figure 5.2 shows the concept label distribution on development set of LDC2015E86, where we can see nearly half of the MULTICONCEPT is not named entity type.

Based on the observation that many of the MULTICONCEPT cases are actually similarly structured subgraphs that only differ in the lexical items, we choose to factor the lexical items out of the subgraph fragment and use the skeletal structure as the fine-grained label, which we refer as **Factored Concept Label (FGL)**.

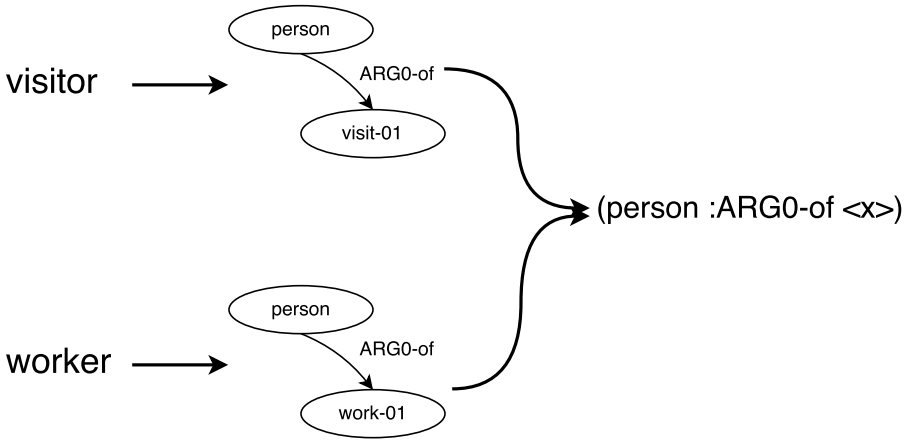


Figure 5.3: One example of generating FGL

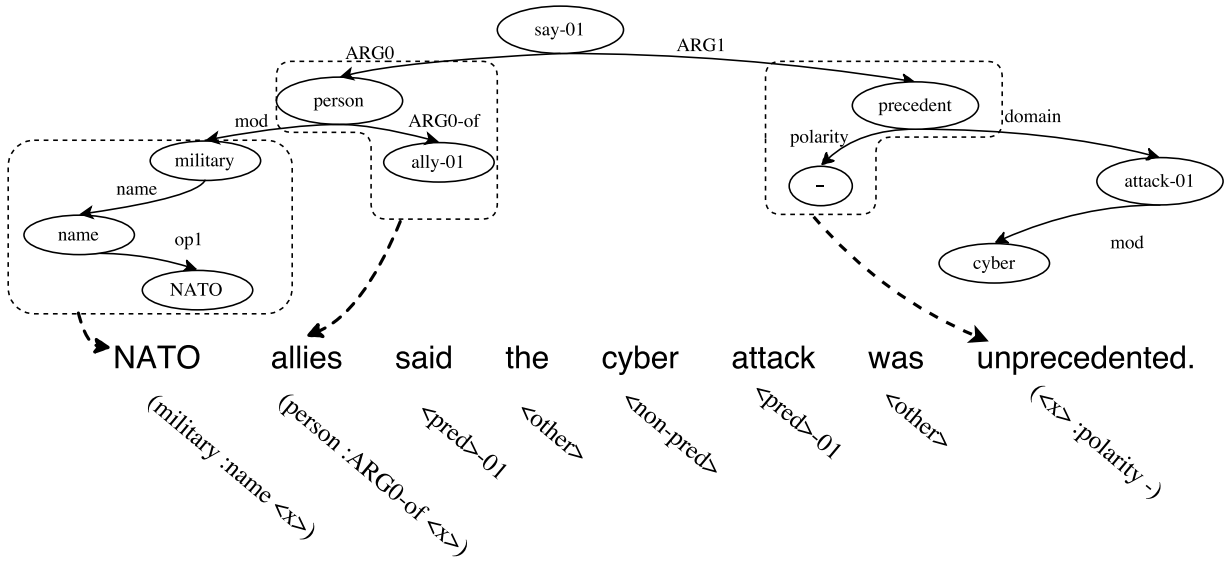


Figure 5.4: One example of generating FGL for sentence "*NATO allies said the cyber attack was unprecedented.*"

Figure 5.3 shows that although English words "visitor" and "worker" have been aligned to different subgraph fragments, after replacing the lexical items, in this case the leaf concepts `visit-01` and `work-01` with a placeholder "x", we are able to arrive at the same FGL. The strategy for determining similar lexical items is simple: for each English word w in a span, we greedily find the concept instance c where longest continuous substring of w and c is longer than 4, then we substitute the concept instance with a placeholder.

Despite this simple strategy, our results show that it can capture a wide range of MULTI-CONCEPT cases while keeping the new label space manageable. While the named entity can be easily categorized using FGL, it can also cover some other common cases like negation, comparatives, and other morphological variations. Setting a frequency threshold to prune out the noisy labels, we are able to extract 91 canonical FGL labels on the training set. Our empirical results show that this canonical FGL label set can cover 96% of the MULTI-CONCEPT cases on development set. Figure 5.4 gives one full example of generating FGLs

for one sentence. For the PREDICATE cases, following Foland and Martin (2016), we only preserve the sense tag. We use label $\langle other \rangle$ to label stopwords that do not map to AMR concepts. The MULTICONCEPT cases are handled by FGL. The FGL label set generated by this procedure can be treated as an abstraction of the original AMR concept label space, where it groups the concepts sharing similar semantic structure into the same category.

5.2.3 CNN-based Character-level Embedding

After constructing the new label set with FGL, we set up a baseline Bidirectional LSTM using the concatenation of word and NER embeddings as the input. Although this architecture is able to capture long-range contextual information, it fails to extract information originating from the word itself. As we have discussed above, some of the MULTICONCEPT cases are associated with the word form and won't benefit from the contextual information. For example, the negation case "*unprecedented*", where the prefix "*un*" already gives enough information to predict the FGL label $\langle x \rangle : polarity-$. In order to incorporate such morphological and shape information, we choose to add a convolutional layer to extract character-level representations. A similar technique has been applied to Named Entity Recognition (Santos and Guimaraes, 2015) and we only provide a brief description of the architecture here.

For input word w and NER tag t, we have a word embedding matrix $W_{wd} \in \mathbb{R}^{d_{wd} \times |V_{wd}|}$ and NER tag embedding $W_t \in \mathbb{R}^{d_t \times |V_t|}$ separately, where d_{wd} and d_t are the dimension of the word and NER tag embedding, $|V_{wd}|$ and $|V_t|$ are the size of word and NER tag vocabulary. And we can retrieve specific word embedding e_w or NER tag embedding e_t through its index defined by corresponding vocabulary. Similarly, assuming a word w composed of characters $\{c_1, c_2, \ldots, c_l\}$, where l is the length of word w, we also have a character embedding matrix $W_c \in \mathbb{R}^{d_c \times |V_c|}$, where d_c is the character embedding dimension defined by user and V_c is

character vocabulary size. After retrieving the character embedding ch_i for each character c_i in word w, we obtain a sequence of vectors $\{ch_1, ch_2, \ldots, ch_l\}$. This serves as the input to convolutional layer.

The convolutional layer applies a linear transformation on local context of the input sequence, where local context is parameterized by window size k. Here we define the local context of character embedding ch_i to be:

$$f_i = (ch_{i-(k-1)/2}, \ldots, ch_{i+(k-1)/2})^\top$$

The j-th element of the convolutional layer output vector e_{wch} is computed by:

$$e_{wch}[j] = \max_{1 \leq i \leq l} [W^0 f_i + b^0]$$

W^0 and b^0 are the parameters of the convolutional layer. And the output vector e_{wch} is the character level representation of the word w. The architecture of our CNN module is presented in Figure 5.5.

The final input to the Bidirectional LSTM is the concatenation of three embeddings $[e_w, e_t, e_{wch}]$ for each word position.

5.3 Experiments

We first test the performance of our Bidirectional LSTM concept identifier as standalone task, where we investigate the effectiveness of our proposed methods. Then we report the final results by incorporating the concept identifier to our transition-based AMR parser, `CAMR`. At the model development stage, we mainly use the dataset LDC2015E86 used in the SemEval Shared Task (May, 2016). Note that this dataset also annotate AMR graph

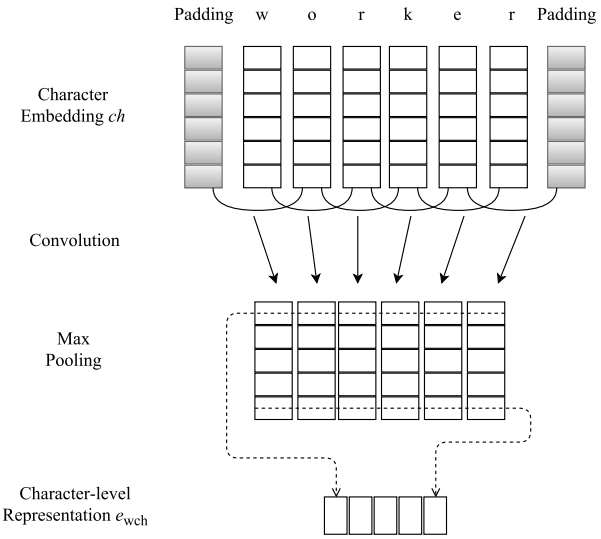


Figure 5.5: The architecture of the CNN-based character-level embedding.

with `:wiki` relations where it links every named entity concept to its wikipedia entry. We remove this information in training data throughout development of our models. At the final testing stage, we add wikification using an off-the-shelf AMR wikifier (Pan et al., 2015) as a post-processing step. All AMR parsing results are evaluated using Smatch (Cai and Knight, 2013).

5.3.1 Bidirectional LSTM Concept Identification Evaluation

In order to isolate the effects of our concept identifier, we first use the official alignments provided by SemEval. The alignment is generated by the unsupervised aligner (Pourdamghani et al., 2014). After getting the alignment table, we generate our FGL label set by filtering out noisy FGL labels that occur fewer than 30 times in training data. These FGL labels account for 96% of the MULTICONCEPT cases in development set. Adding other labels that include PREDICATE, NON-PREDICATE and CONST gives us 116 canonical labels. UNK label is added to handle the unseen concepts.

In the Bidirectional LSTM, the hyperparameter settings are as follows: word embedding dimension $d_{wd} = 128$, NER tag embedding dimension $d_t = 8$, character embedding dimension $d_c = 50$, character level embedding dimension $d_{wch} = 50$, convolutional layer window size $k = 2$.

Input	P	R	F_1	Acc
word,NER	81.2	80.6	80.9	85.4
word,NER,CNN	83.3	82.7	83.0	87.0

Table 5.1: Performance of Bidirectional LSTM with different input.

Table 5.1 shows the performance on development set of LDC2015E86, where the precision, recall and F-score are computed by treating $\langle other \rangle$ as the negative label and accuracy is calculated using all labels. We include accuracy here since correctly predicting words that don't invoke concepts is also important. We can see that utilizing CNN-based character level embedding yields around 2 points absolute improvement for both F-score and accuracy, which indicates that morphological and word shape information is important for concept identification.

Impact on AMR Parsing. In order to test the impact of our concept identification component on AMR parsing, we add the predicted concept labels as features to `CAMR`. Here is the detailed feature set we add to `CAMR`'s feature templates. To clarify the notation, we refer the concept label predicted by our concept identifier as c_{pred} and the candidate concept label in CAMR as c_{cand}:

- `pred_label`. Unary feature of c_{pred}.
- `is_eq_sense`. Whether c_{pred} and c_{cand} has same sense (if applicable).

One reason why we choose to add this information as feature rather than directly use the

predicted concept directly is that it is not straightforward to recover the original concept based on the predicted label. For example, since we generalize all the predicates to a compact form `<pred-xx>`, for irregular verbs like "*became*" ⇒ `become-01`, simply stemming the inflected verb form will not give us the correct concept even if the sense is predicted correctly. However, since `CAMR` uses the alignment table to store all possible concept candidates for a word, adding our predicated label as a feature could potentially help the parser to choose the correct concept. In order to take full advantage of this new feature, we also extend `CAMR` so that it can discover candidate concepts outside of the alignment table. To achieve this, during the FGL label generation process, we first store the string-to-concept mapping as a template. For example, when we generate the FGL label `(person :ARG0-of <x>-01)` from "worker", we also store the template `<x>er -> (person :ARG0-of <x>-01)`. Then during decoding time, even we haven't seen "teacher", we could use the above template and generate the correct answer `(person :ARG0-of teach-01)`. We refer this process as *unknown concept generation*.

Table 5.2 summarizes the impact of our proposed methods on development set of LDC2015E86. We can see that by utilizing the unknown concept generation and extended c_{pred} feature, both precision and recall improve by about 1 percentage point, which indicates that the new feature brings richer information to the concept predicting process to help correctly score candidate concepts from the alignment table.

Parsers	P	R	F_1
CAMR Wang et al. (2016)	72.3	61.4	66.5
CAMR-*gen*	72.1	62.0	66.6
CAMR-*gen*-c_{pred}	**73.6**	**62.6**	**67.6**

Table 5.2: Performance of AMR parsing with c_{pred} as feature without *wikification* on dev set of LDC2015E86. The fist row is the baseline parser. The second row is adding unknown concept generation and the last row additionally extends the baseline parser with c_{pred}.

5.4 Conclusion

In this chapter, we build a Bidirectional LSTM concept identifier based on a novel concept categorization technique, Factored Concept Label (FCL). We argue that the proposed method is able to incorporate richer context and learn sparse concept labels. Empirical results show that integrating the new concept identifier to an existing AMR parser improve the Smatch score by around 1 points.

Chapter 6

AMR Parsing with Graph-based Alignment

6.1 Introduction

We have shown in last chapter that refined concept identification with neural network-based technique has positive impact on overall AMR parsing by resolving the *sparsity* property. However, the process of generating the training instances for concept identification is not error-free and still replies on the alignment between word and AMR concept. In this chapter, we focus on the fundamental issue in AMR parsing — ABSTRACTION, which is closely related to how we extract concepts from AMR and build mappings between word's surface form and its semantic meaning. We propose to tackle it with a novel graph-based aligner designed specifically for word-to-concept scenario and later show that better alignment result could improve AMR parsing result.

Building the alignment between word and AMR concept is often conducted as a preprocessing step. As a result, accurate concept identification crucially depends on the word-to-

AMR-concept alignment. Since there is no manual alignment in AMR annotation, typically either a rule-based or unsupervised aligner is applied to the training data to extract the mapping between words and concepts. This mapping will then be used as reference data to train concept identification models. The JAMR aligner (Flanigan et al., 2014) greedily aligns a span of words to graph fragment using a set of heuristic rules. While it can easily incorporate information from additional linguistic sources such as WordNet, it is not adaptable to other domains. Unsupervised aligners borrow techniques from Machine Translation and treat sentence-to-AMR alignment as a word alignment problem between a source sentence and its linearized AMR graph (Pourdamghani et al., 2014) and solve it with IBM word alignment models (Brown et al., 1993). However, the distortion model in the IBM models is based on the linear distance between source side words while the linear order of the AMR concepts has no linguistic significance, unlike word order in natural language. One example of such inconsistency is shown in Figure 6.1. A more appropriate sentence-to-AMR alignment model should be one that takes the hierarchical structure of the AMR into account. We develop a Hidden Markov Model (HMM)-based sentence-to-AMR alignment method with a novel **Graph Distance** distortion model to take advantage of the structural information in AMR, and apply a structural constraint to re-score the posterior during decoding time

We present experimental results that show incorporating the improved aligner to our transition-based AMR parser results in consistently better Smatch scores on various datasets. The rest of this chapter is organized as follows. Section 6.2 describes our alignment method. We present experimental results in Section 6.3, and conclude in Section 6.4.

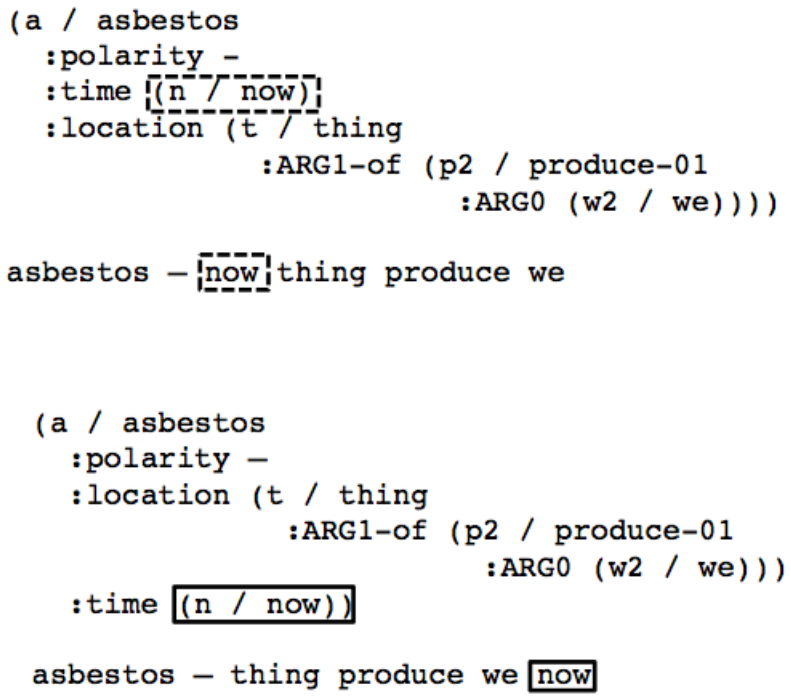


Figure 6.1: An example of inconsistency in AMR linearization for sentence: "*There is no asbestos in our products now .*". While both annotations (above) here are valid, the linearized AMR concepts (below) are inconsistent input to word aligner.

6.2 Aligning English Sentence to AMR graph

Given a AMR graph G and English sentence $\boldsymbol{e} = \{e_1, e_2, \ldots, e_i, \ldots, e_I\}$, in order to fit them into the traditional word alignment framework, the AMR graph G is normally linearized using depth first search by printing each node as soon as it it visited. The re-entrance node is printed but not expanded to preserve the multiple mentions of concept. The relation (also called AMR role token) between concepts are preserved in the unsupervised aligner (Pourdamghani et al., 2014) because they also try to align relations to English words. We ignore the relations here since we focus on aligning concepts. Therefore the linearized concept sequences can be represented as $\boldsymbol{g} = \{g_1, g_2, \ldots, g_j, \ldots, g_J\}$. However, although this configuration makes it easy to adopt the existing word alignment model, it also ignore the structure information coming with AMR graph.

In this section, we first introduce to incorporate this graph structure information through the distortion model inside a HMM-based word aligner. Then we further improve the model with a re-scoring method during decoding time.

6.2.1 HMM-based Aligner with Graph Distance Distortion

Given the sequence pair $(\boldsymbol{e}, \boldsymbol{g})$, the HMM-based word alignment model assumes that each source word is assigned to exactly one target word, and defines an asymmetric alignment for the sentence pair as $\boldsymbol{a} = \{a_1, a_2, \ldots, a_i, \ldots, a_I\}$, where each $a_i \in [0, J]$ is an alignment from source position i to target position a_i, $a_i = 0$ means that e_i is not aligned to any target words. Note that under the AMR to English alignment context, both the alignment and the graph structure is asymmetric, since we only have AMR graph annotation on linearized AMR sequences g. Unlike the traditional word alignment for machine translation, here we will have different formulas for each translation direction. In this section, we only discuss

the translation from English (source) to linearized AMR concepts (target) and we'll further discuss translating the other direction in the following section.

The HMM-based model breaks the generative alignment process into two factors:

$$P(\boldsymbol{e}, \boldsymbol{a} \mid \boldsymbol{g}) = \prod_{i=1}^{I} P_d(a_i \mid a_{i-1}, J) P_t(e_i | g_{a_i})$$

where P_d is the distortion model and P_t is the translation model. Traditionally, the distortion probability $P_d(j \mid j', J)$ is modeled to depend only on the jump width $(j - j')$ Vogel et al. (1996) and is defined as:

$$P_d(j \mid j', J) = \frac{c(j - j')}{\sum_{j''=1}^{J} c(j'' - j')}$$

where $c(j - j')$ is the count of jump width. This formula both satisfies normalization constraint and expresses the *locality* assumption where words are adjacent in the source sentence tend to be closer in target sentence.

As the linear *locality* doesn't hold among linearized AMR concepts, we choose to instead encode the distortion probability through graph distance, which is given by:

$$P_{gd}(j \mid j', G) = \frac{c(d(j, j'))}{\sum_{j''} c(d(j'', j'))}$$

The graph distance $d(j, j')$ is the length of shortest path on AMR graph G from concept j to concept j'. Note that we have to manually normalize $P_{gd}(j \mid j', G)$, because unlike in the case of the linear distance, there are multiple concepts that can have the same distance from the j'-th concept on AMR graph.

During training, just like original HMM-based aligner, EM algorithm can be applied to update the parameters of the model.

6.2.2 Improved Decoding with Posterior Rescoring

So far, we have integrated the graph structure information into forward direction (English to AMR). To also improve the reverse direction model (AMR to English), we choose to use the graph structure to rescore the posterior during decoding time.

Compared to Viterbi decoding, posterior thresholding has shown better results in word alignment task (Liang et al., 2006). Given threshold γ, for all possible alignments, we select the final alignment result according to the following criteria:

$$\boldsymbol{a} = \{(i, j) : p(a_j = i \mid \boldsymbol{g}, \boldsymbol{e}) > \gamma\}$$

where the state probability $p(a_j = i \mid \boldsymbol{g}, \boldsymbol{e})$ is computed using the forward-backward algorithm. The forward algorithm is defined as:

$$\alpha_{j,i} = \sum_{i'} \alpha_{j-1,i'} p(a_j = i \mid a_{j-1} = i') p(g_j \mid e_{a_j})$$

To incorporate the graph structure, we rescale the distortion probability as:

$$\begin{aligned} p_{new}(a_j = i \mid & a_{j-1} = i') \\ & = p(a_j = i \mid a_{j-1} = i')^{e^{\Delta d}} \end{aligned}$$

where the scaling factor $\Delta d = d_j - d_{j-1}$ is the graph depth difference between the adjacent AMR concepts g_j and g_{j-1}. We also apply the same procedure for the backward computation. Note that since the model is in reverse direction, the distortion $p(a_j = i \mid a_{j-1} = i')$ here is still based on English word distance, jump width.

This rescaling procedure is based on the intuition that when we've done processing the last concept g_{j-1} in some subgraph, the next concept g_j's aligned English position i doesn't

necessarily have relation to the last aligned English position i'. Figure 6.2 illustrates this phenomenon: Although `we` and `current` are adjacent concepts in linearized AMR sequence, they are actually far away from each other in the graph (has graph depth difference -2). However, the distortion based on English word distance mostly tends to choose the closer word, which may have a very low probability for our correct answer here (the jump width between "Currently" and "our" is -6). By applying the exponential scaling factor, we are able to reduce the differences between different distortion probabilities. On the contrary, when the distortion probability is reliable (the absolution value of graph depth difference is small), the model choose to trust the distortion and pick the closer English word.

The rescaling factor can be treated as a selection filter for decoding, where it depends on the graph depth difference Δd to control the effect of learned distortion probability. Note that after the rescaling , the resulting distortion probability won't satisfy the normalization constraint. However, we only apply this during decoding time and experiments show that the typical threshold $\gamma = 0.5$ still works well for our case.

6.2.3 Combining Both Directions

Empirical results show that combining alignments from both directions improve the alignment quality DeNero and Klein (2007); Och and Ney (2003); Liang et al. (2006). To combine the alignments, we adopt a slightly modified version of posterior thresholding, *competitive thresholding*, as proposed in DeNero and Klein (2007), which tends to select alignments that form a contiguous span.

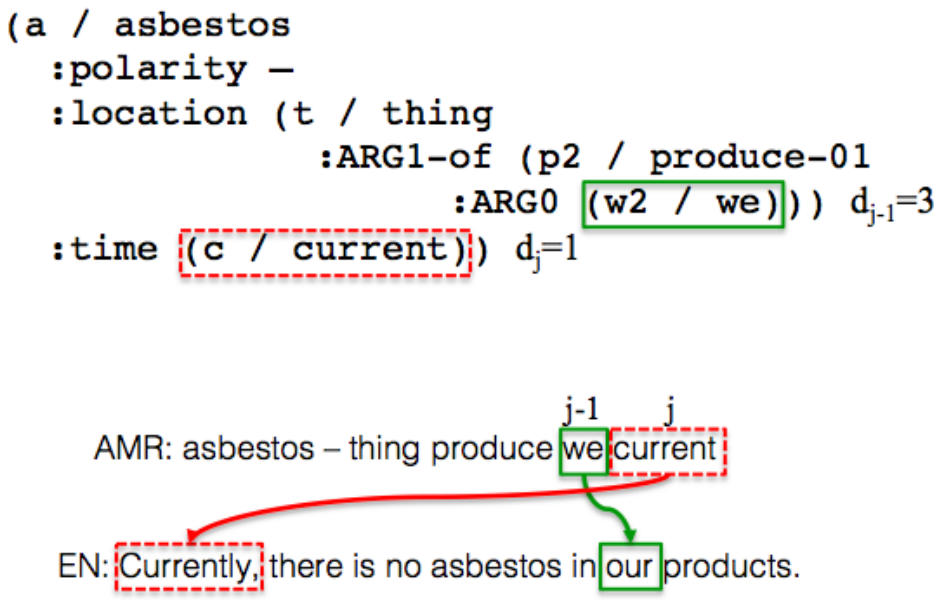


Figure 6.2: AMR graph annotation, linearized concepts for sentence "*Currently, there is no asbestos in our products*". The concept `we` in solid line is the $(j-1)$-th token in linearized AMR. It is aligned to English word "our" and its depth in graph d_{j-1} is 3. While the word distance-based distortion prefers an alignment near "our", the correct alignment needs a longer distortion.

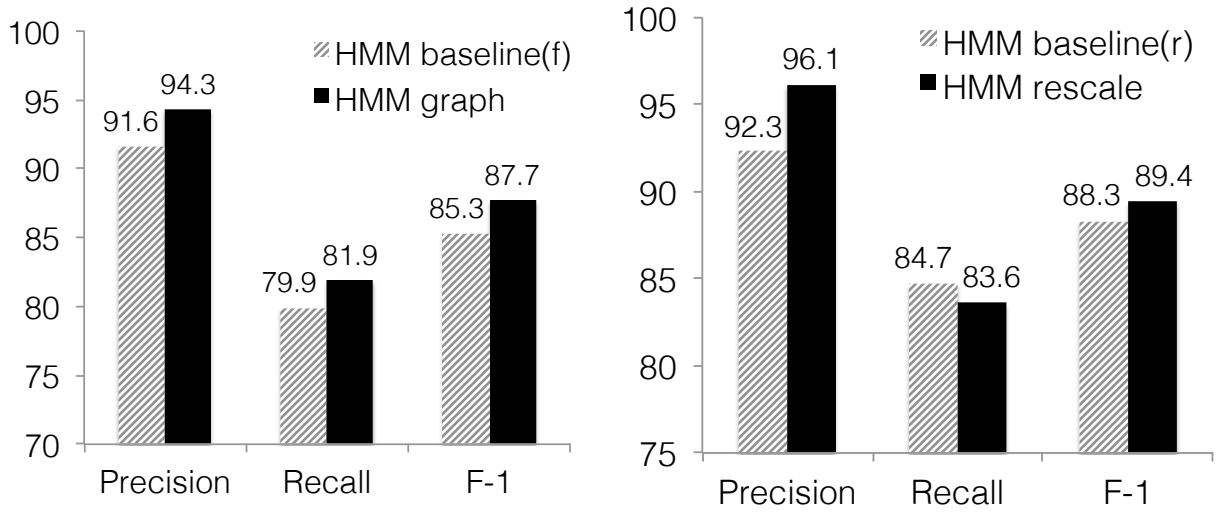
6.3 Experiments

We first test the performance of our graph-based aligner as standalone task, where we investigate the effectiveness of our proposed method based on alignment performance. Then we report the final results by incorporating the improved aligner to `CAMR`. Here we use the setup similar to the experiment discussed in Chapter 5.

6.3.1 HMM-based AMR-to-English Aligner Evaluation

To validate the effectiveness of our proposed alignment methods, we first evaluate our forward (English-to-AMR) and reverse (AMR-to-English) aligners against the baseline HMM word alignment model, which is the Berkeley aligner toolkit (DeNero and Klein, 2007). Then we combine the forward and reverse alignment results using competitive thresholding (DeNero and Klein, 2007), which tends to select alignments which form a contiguous span. We set the threshold γ to be 0.5 in the following experiments. To evaluate the alignment quality, we use 200 hand-aligned sentences (Pourdamghani et al., 2014) as development and test set. We process the English sentences by removing stopwords, following similar procedure as in (Pourdamghani et al., 2014). When linearizing AMR graphs, we instead remove all the relations and only keep the concepts. For all the models, we run 5 iterations of IBM Model 1 and 2 iterations of HMM on the whole dataset.

From Figure 6.3a, we can see that our graph-distance based model improve both the precision and recall by a large margin, which indicates the graph distance distortion better fits the English-to-AMR alignment task. For the reverse model, although our HMM rescaling model loses accuracy in recall, it is able to improve the precision by around 4 percentage points, which confirms our intuition that the rescoring factor is able to keep reliable alignments and penalize unreliable ones. We then combine our forward and reverse alignment



(a) HMM forward

(b) HMM reverse

Figure 6.3: Our improved forward (graph) and reverse(rescale) model compared with HMM baseline on hand aligned development set.

result using competitive thresholding. Table 6.1 shows the combined result on hand-aligned dev and test sets.

Datasets	P	R	F_1
dev	97.7	84.3	90.5
test	96.9	84.6	90.3

Table 6.1: Combined HMM alignment result evaluation.

Impact on AMR Parsing To investigate our aligner's contribution to AMR parsing, we replace the alignment table generated by the best performing aligner (the forward and reverse combined) in the previous section and re-train `CAMR` with the Bi-LSTM concept label feature mentioned in last chapter included.

From Table 6.2, we can see that the unsupervised aligner (ISI and HMM) generally outperforms the JAMR rule-based aligner, and our improved HMM aligner is more consistent than the IBM Model 4 aligner (Pourdamghani et al., 2014).

Dataset	Aligner	P	R	F_1
Dev	JAMR	73.6	62.6	67.6
	ISI	72.8	**64.6**	**68.4**
	Our HMM	**73.6**	63.6	68.2
Test	JAMR	71.6	61.3	66.0
	ISI	70.6	**62.7**	66.4
	Our HMM	**72.1**	62.5	**67.0**

Table 6.2: AMR parsing result (without *wikification*) with different aligner on development and test of LDC2015E86, where JAMR is the rule-based aligner, ISI is the modified IBM Model 4 aligner

6.3.2 Comparison with other Parsers

We first add the *wikification* information on the parser output using the off-the-shelf AMR wikifier (Pan et al., 2015) and compare results with the state-of-the-art parsers in SemEval16 share task. We also report our result on the previous release (LDC2014T12), AMR annotation release 1.0, which is another popular dataset that most of the existing parsers report result on. Note that the release 1.0 annotation doesn't include *wiki* information.

Dataset	Parsers	P	R	F_1
SemEval Test	CAMR	70.3	63.1	66.5
	RIGA	-	-	67.2
	JAMR(Flanigan et al., 2016)	70	65	67
	Our parser	**71.7**	**64.9**	**68.1**
SemEval Blind Test	CAMR	67.4	57.3	62.0
	RIGA	68.0	57.0	62.0
	JAMR(Flanigan et al., 2016)	-	-	56
	Our parser	**68.2**	**59.5**	**63.6**

Table 6.3: Comparison with the winning system in SemEval (**with** *wikification*) on test and blind test

`CAMR` and `RIGA` (Barzdins and Gosko, 2016) are the two best performing parsers that participated in SemEval 2016 share task. While we use `CAMR` as our baseline system, the parser from `RIGA` is also based on `CAMR` and extended with a error-correction wrapper and

the ensemble with a character-level neural translation model. In table 6.3 we can see that our parser outperforms both systems by around 1.5 percentage points, where the recall improvement is more significant, around 2 percentage points.

Parsers	P	R	F_1
CAMR	71.3	62.2	66.5
Zhou et al. (2016)	70	62	66
Pust et al. (2015)	-	-	65.8
Our parser	**72.7**	**64.0**	**68.07**

Table 6.4: Comparison with the existing parsers on full test set of LDC2014T12

Table 6.4 shows the performance of our parser on the full test set of LDC2014T12. We include the previous best results on this dataset. The parser proposed in (Zhou et al., 2016) jointly learns the concept and relation through a incremental joint model. And the syntax-based MT system treats parsing as a machine translation task and incorporate various external resources. Our parser still achieves best result by only incorporating name entity information.

6.4 Conclusion

In this chapter, we improve sentence-to-AMR alignment from two aspects. We first extend the HMM-based word alignment model with a graph distance distortion in forward direction. Then in reverse direction, A rescoring method is applied during decoding to incorporate the graph structure information. Consistent improvement over our transition-based parser shows that better alignment is crucial for AMR parsing and graph information is essential to build a word-to-AMR-concept aligner.

Chapter 7

Neural AMR Parsing

7.1 Introduction

All of our work we have discussed in the previous chapters involve building separate components to tackle sub-problems in AMR parsing, resulting in a pipeline system. However, the major drawback in a pipeline system is error propagation. For instance, although the automatic aligners used in AMR parsing are pretty accurate, achieving around 90% F-score on held-out test set, the rest 10% unaligned or incorrectly aligned concepts still have a crucial effect on the downstream steps, as they will never be recovered during concept identification phrase, causing additional relations missing from training data. In this chapter, we explore the possibility of unifying all the sub-components into a neural end-to-end model, where we treat the input sentences and output AMR graph both as sequences and address the parsing problem through a neural machine translation paradigm.

Recently, Sutskever et al. (2014b) introduced a neural network model for solving the general sequence-to-sequence problem, and Bahdanau et al. (2014) proposed a related model with an attention mechanism that is capable of handling long sequences. Both models achieve

state-of-the-art results on large scale machine translation tasks.

However, sequence-to-sequence models mostly work well for large scale parallel data, usually involving millions of sentence pairs. Vinyals et al. (2015) present a method which linearizes parse trees into a sequence structure and therefore a sequence-to-sequence model can be applied to the constituent parsing task. Competitive results have been achieved with an attention model on the Penn Treebank dataset, with only 40K annotated sentences.

AMR parsing is a much harder task in that the target vocabulary size is much larger, while the size of the dataset is much smaller. While for constituent parsing we only need to predict non-terminal labels and the output vocabulary is limited to 128 symbols, AMR parsing has both concepts and relation labels, and the target vocabulary size consists of tens of thousands of symbols. Barzdins and Gosko (2016) applied a similar approach where AMR graphs are linearized using depth-first search and both concepts and relations are treated as tokens (see Figure 7.4). Due to the data sparsity issue, their AMR parsing results are significantly lower than state-of-the-art models when using the neural attention model.

In this chapter, we present a method which linearizes AMR graphs in a way that captures the interaction of concepts and relations. To overcome the data sparsity issue for the target vocabulary, we propose a categorization strategy which first maps low frequency concepts and entity subgraphs to a reduced set of category types. In order to map each type to its corresponding target side concepts, we use heuristic alignments to connect source side spans and target side concepts or subgraphs. During decoding, we use the mapping dictionary learned from the training data or heuristic rules for certain types to map the target types to their corresponding translation as a post-processing procedure.

Experiments show that our linearization strategy and categorization method are effective for the AMR parsing task. Our model improves significantly in comparison with the previously reported sequence-to-sequence results and provides a competitive benchmark in

comparison with state-of-the-art results without using dependency parses or other external semantic resources.

7.2 Sequence-to-sequence Parsing Model

Our model is based on an existing sequence-to-sequence parsing model (Vinyals et al., 2015), which is similar to models used in neural machine translation.

7.2.1 Encoder-Decoder

Encoder. The encoder learns a context-aware representation for each position of the input sequence by mapping the inputs $w_1, \ldots, w_m$ into a sequence of hidden layers $h_1, \ldots, h_m$. To model the left and right contexts of each input position, we use one special type of recurrent neural network (RNN), bidirectional Long-Short-Term-Memory (LSTM) (Bahdanau et al., 2014). First, each input's word embedding representation $x_1, \ldots, x_m$ is obtained though a lookup table. Then these embeddings serve as the input to two RNNs: a forward RNN and a backward RNN. The forward RNN can be seen as a recurrent function defined as follows:

$$h_i^{fw} = f(x_i, h_{i-1}^{fw}) \tag{7.1}$$

Here the recurrent function f (or RNN cell) we use is LSTM (Hochreiter and Schmidhuber, 1997). The backward RNN works similarly by repeating the process in reverse order. The outputs of forward RNN and backward RNN are then depth-concatenated to get the final representation of the input sequence.

$$h_i = [h_i^{fw}, h_{m-i+1}^{bw}] \tag{7.2}$$

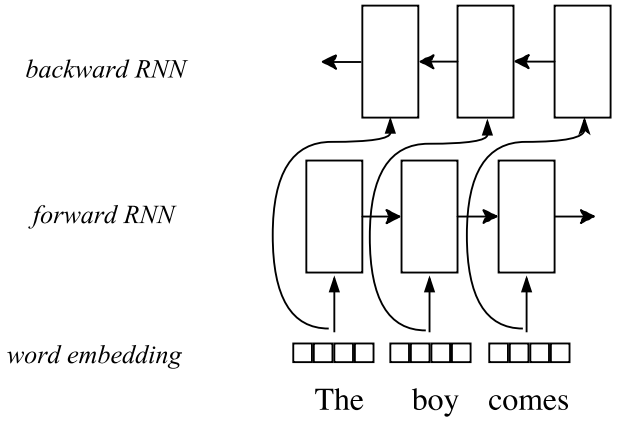


Figure 7.1: The architecture of bidirectional LSTM.

The architecture of the bidirectional LSTM encoder is illustrated in Figure 7.1.

Decoder. The decoder is also an LSTM model which generates the hidden layers recurrently. Additionally, it utilizes an attention mechanism to put a "focus" on the input sequence. At each output time step j, the attention vector $d_j^{'}$ is defined as a weighted sum of the input hidden layers, where the masking weight α_i^j is calculated using a feedforward neural network. Formally, the attention vector is defined as follows:

$$u_i^j = v^T \tanh(W_1 h_i + W_2 d_j) \tag{7.3}$$

$$\alpha_i^j = \text{softmax}(u_i^j) \tag{7.4}$$

$$d_j^{'} = \sum_{i=1}^{m} \alpha_i^j h_i \tag{7.5}$$

where d_j is the output hidden layer at time step j, and v, W_1, and W_2 are parameters for the model. Here the weight vector $\alpha_1^j, \ldots, \alpha_m^j$ is also interpreted as a soft alignment in the neural machine translation model, which similarly could also be treated as a soft alignment between token sequences and AMR relation/concept sequences in the AMR parsing task. Finally, we concatenate the hidden layer d_j and attention vector $d_j^{'}$ to get the new hidden

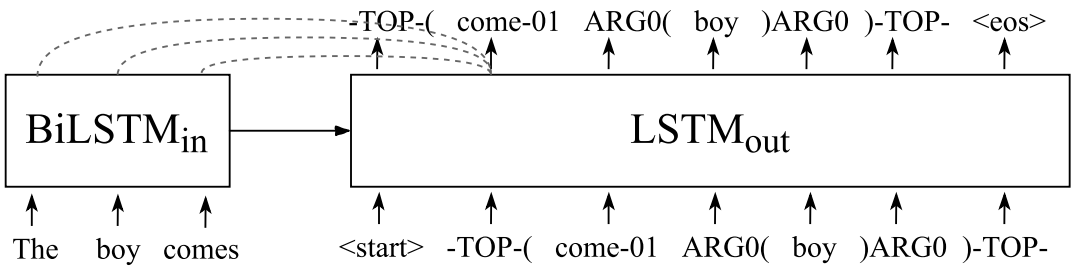


Figure 7.2: The architecture of the encoder-decoder framework for the example input "The boy comes".

layer, which is used to predict the output sequence label.

$$P(y_j|w_{1:m}, y_{1:j-1}) = \text{softmax}(W_3[d_j, d_j^{'}]) \tag{7.6}$$

The overall architecture for the encoder-decoder framework with attention mechanism is illustrated in Figure 7.2.

7.2.2 Parse Tree as Target Sequence

Vinyals et al. (2015) designed a reversible way of converting the parse tree into a sequence, which they call linearization. The linearization is performed in the depth-first traversal order. Figure 7.3 shows an example of the linearization result. The target vocabulary consists of 128 symbols.

In practice, they found that using the attention model is more data efficient and works well on the parsing task. They also reversed the input sentence and normalized the part-of-speech tags. After decoding, the output parse tree is recovered from the output sequence of the decoder in a post-processing procedure. Overall, the sequence-to-sequence model is able to match the performance of the Berkeley Parser (Petrov et al., 2006).

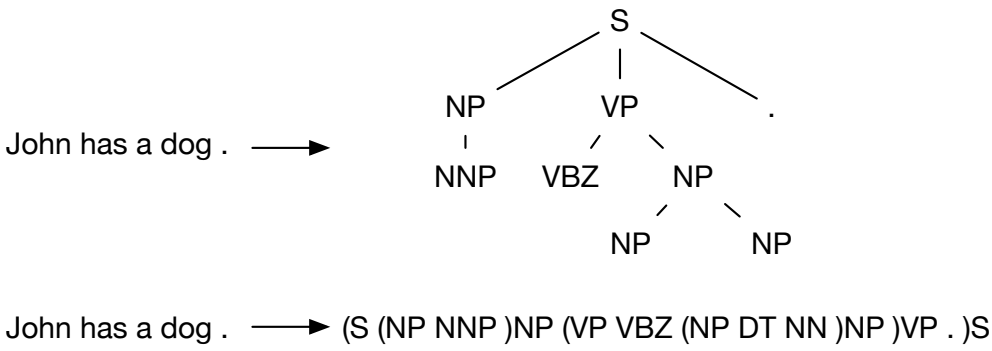
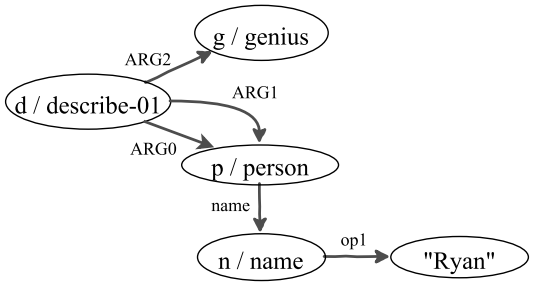


Figure 7.3: Example parsing task and its linearization.

7.3 AMR Linearization

Barzdins and Gosko (2016) present a similar linearization procedure where the depth-first traversal result of an AMR graph is used as the AMR sequence (see Figure 7.4). The bracketing structure of AMR is hard to maintain because the prediction of relation (with left parenthesis) and the prediction of an isolated right parenthesis are not correlated. As a result, the output AMR sequences usually have parentheses that do not match.

We present a linearization strategy which captures the bracketing structure of AMR and the connection between relations and concepts. Figure 7.4b shows the linearization result of the AMR graph shown in Figure 7.4a. Each relation connects the head concept to a subgraph structure rooted at the tail concept, which shows one branch below the head concept. We use the relation label and left parenthesis to show the beginning of the branch (subgraph) and use right parenthesis paired with the relation label to show the end of the branch. We additionally add "-TOP-(" at the beginning to show the start of the traversal of the AMR graph and add ")-TOP-" at the end to show the end of traversal. When a symbol is revisited, we replace the symbol with "-RET-". We additionally add the revisited symbol before "-



(a)

Barzdins and Gosko (2016)

```
(describe-01 :ARG0 (person :name
(name :op1 "Ryan") ) :ARG1 (person ) :ARG2 genius)
```

Our linearization

```
-TOP-( describe-01 ARG0( person name( name op1( "Ryan" )op1)name
)ARG0 ARG1( person -RET- )ARG1 ARG2( genius )ARG2 )-TOP
```

(b)

Figure 7.4: One example AMR graph for sentence "Ryan' s description of himself: a genius." and its different linearization strategies.

RET-" to decide where the reentrancy is introduced to.[1] We also get rid of variables and only keep the full concept label. For example, "g / genius" to "genius".

We can easily recover the original AMR graph from its linearized sequence. The sequence also captures the branching information of each relation explicitly by representing it with a start symbol and an end symbol specific to that relation. During our experiments, most of the output sequences have a matching bracketing structure using this linearization strategy. The idea of linearization is basically a depth-first traversal of the AMR where the original graph structure can be reconstructed with the linearization result. Even though we call it a sequence, its core idea is actually generating a graph structure from top-down.

[1]This is an approximation because one concept can appear multiple times, and we simply attach the reentrancy to the most recent appearance of the concept. An additional index would be needed to identify the accurate place of reentrancy.

	Sentence side (lemmatized, lower cased)	AMR side
Before linearization:	chinese seismology be gallop down thewrong road .	(g / gallop-01 :ARG0 (s / seismology :mod (c / country :wiki "China" :name (n / name :op1 "China"))) :ARG1 (r / road :mod (w / wrong)) :direction (d / down))
After linearization:	NE_country-0 -SURF--0 be -VERB--0down the wrong -SURF—1 .	-TOP-(-VERB--0 ARG0(-SURF--0 mod(NE_country-0)mod)ARG0 ARG1(-SURF—1 mod(wrong)mod)ARG1 direction(down)direction)-TOP

Figure 7.5: An example of categorized sentence-AMR pair.

7.4 Dealing with the Data Sparsity Issue

While sequence-to-sequence models can be successfully applied to constituent parsing, they do not work well on the AMR parsing task as shown by Barzdins and Gosko (2016). The main bottleneck is that the size of target vocabulary for AMR parsing is much larger than constituent parsing, tens of thousands in comparison with 128, and the size of training data is less than half of that available for parsing.

In this section, we present a categorization method which significantly reduces the target vocabulary size, as the alignment from the attention model does not work well on the relatively small dataset. To adjust for the alignment errors made by the attention model, we propose to add supervision from an alignment produced by an external aligner which can use lexical information to overcome the limit of data size.

7.4.1 AMR Categorization

We define several types of categories and map low frequency words into these categories.

1. DATE: we reduce all the date entity subgraphs to this category, ignoring details of the specific date entity.

2. NE_{*ent*}: we reduce all named entity subgraphs to this category, where *ent* is the root label of each subgraph, such as *country* or *person*.

3. -VERB-: we map predicate variables with low frequency ($n < 50$) to this category

4. -SURF-: we map non-predicate variables with low frequency ($n < 50$) to this category

5. -CONST-: we map constants other than numbers, "-", "interrogative", "expressive", "imperative" to this category.

6. -RET-: we map all revisited concepts to this category.

7. -VERBAL-: we additionally use the verbalization list [2] from the AMR website and map matched subgraphs to this category.

After the re-categorization, the vocabulary size is substantially reduced to around 2000, though this vocabulary size is still very large for the relatively small dataset. These categories and the frequent concepts amount to more than 90% of all the target words, and each of these are learned with a larger number of occurrences.

7.4.2 Categorize Source Sequence

The source side tokens also have sparsity issues. For example, even if we have mapped the number *1997* to "DATE", we can not easily generalize it to the token *1993* if it does not appear in the training data. Also, some special 6-digit date formats such as "YYMMDD" are hard to learn using co-occurrence statistics.

[2] http://amr.isi.edu/download/lists/verbalization-list- v1.06.txt

Our basic approach to dealing with this issue is to generalize these sparse tokens or spans to some special categories (currently we use the same set of categories defined in the previous section). On the training data, we can use the heuristic alignment. For example, if we learned from the heuristic alignment that "010911" is aligned to a date-entity of September 11, 2001 on the AMR side, we use the same category "DATE" to replace this token. We distinguish this alignment from other date alignments by assigning a unique indexed category "DATE-X" to both sides of the alignment, where "X" counts from 0 and adds one for each new date entity from left to right on the sentence side. The same index strategy goes for all the other categories. Figure 7.5 shows an example of the linearized parallel sequence. The first infrequent non-predicate variable "seismology" is mapped to "-SURF–0", then "wrong" to "-SURF–1" based on its position on the sentence side. The indexed category labels are then projected onto the target side based on the heuristic alignment. During this re-categorization procedure, we build a map Q from each token or span to its most likely concept or category on the target side based on relative frequency. We also dump a DATE template for recognizing new date entities by abstracting away specific date fields such as "1997" to "YEAR", "September" to "MONTH". For example, we build a template "MONTH DAY, YEAR" from the specific date mention "June 6, 2007".

During decoding, we are only given the sentence. We first use the date templates learned from the training data to recognize dates in each sentence. We also use a named entity tagger to recognize named entity mentions in the sentence. We use the entity name and wiki information from Q if there is a match of the entity mention, otherwise for convenience we simply use "person" as the entity name and use wiki "-". For each of the other tokens, we first look it up in Q and replace it with the most likely mapping. If there is no match, we further look it up in the verbalization list. In case there is still no match, we use the part of speech information to assign its category. We replace verbs with category "-VERB-" and

nouns with category "-SURF-". In accordance with the categorized token sequence, we also index each category in the resulting sequence from left to right.

7.4.3 Recovering AMR graph

During decoding, our output sequences usually have categories and we need to map each category to AMR concepts or subgraphs. When we categorize the tokens in each sentence before decoding, we save the mapping from each category to its original token as table D. As we use the same set of categories on both source and target sides, we heuristically align target side category label to its source side counterpart from left to right. Given table D, we know which source side token it comes from and use the most frequent concept or subgraph of the token to replace the category.

7.4.4 Supervised Attention Model

In this section, we propose to learn the attention vector in a supervised manner. There are two main motivations behind this. First, the neural attention model usually utilizes millions of data points to train the model, which learns a quite reasonable attention vector that at each output time step constrains the decoder to put a focus on the input sequences (Bahdanau et al., 2014; Vinyals et al., 2015). However, we only have 16k sentences in the AMR training data and our output vocabulary size is quite large, which makes it hard for the model to learn a useful alignment between the input sequence and AMR concepts/relations. Second, as argued by Liu et al. (2016), the sequence-to-sequence model tries to calculate the attention vector and predict the current output label simultaneously. This makes it impossible for the learned soft alignment to combine information from the whole output sentence context. However, traditional word alignment can easily use the whole output sequence, which will

produce a more informed alignment.

Similar to the method used by Liu et al. (2016), we add an additional loss to the original objective function to model the disagreement between the reference alignment and the soft alignment produced by the attention mechanism. Formally, for each input/output sequence pair $(\mathbf{w}, \mathbf{y})$ in the training set, the objective function is defined as:

$$-\frac{1}{n}\sum_{j=1}^{n}\log p(y_j|\mathbf{w}, y_{1:j-1}) + \lambda\Theta(\bar{\alpha}^j, \alpha^j) \tag{7.7}$$

where $\bar{\alpha}^j$ is the reference alignment for output position j, which is provided by the existing aligner, α^j is the soft alignment, $\Theta()$ is cross-entropy function, n is the length of output sequence and λ is the hyperparameter which serves as a trade-off between sequence prediction and alignment supervision. Note that the supervised attention part doesn't affect the decoder which will predict the output label given learned weights.

One issue with this method is how we represent $\bar{\alpha}$. As the output of conventional aligner is a hard decision, alignment is either one or zero for each input position. In addition, multiple input words could be aligned to one single concept. Finally, in the AMR sequences, there are many output labels (mostly relations) which don't align to any word in the input sentence. We utilize a heuristic method to process the reference alignment. We assign an equal probability among the words that are aligned to one AMR concept. Then if the output label doesn't align to any input word, we assign an even probability for every input word.

7.5 Experiments

We evaluate our system on the released dataset (LDC2015E86) for SemEval 2016 task 8 on meaning representation parsing (May, 2016). The dataset contains 16,833 training, 1,368

development and 1,371 test sentences which mainly cover domains like newswire, discussion forum, etc. All parsing results are measured by Smatch (version 2.0.2) (Cai and Knight, 2013).

7.5.1 Experiment Settings

We first preprocess the input sentences with tokenization and lemmatization. Then we extract the named entities using the Illinois Named Entity Tagger (Ratinov and Roth, 2009).

For training all the neural AMR parsing systems, we use 256 for both hidden layer size and word embedding size. Stochastic gradient descent is used to optimize the cross-entropy loss function and we set the drop out rate to be 0.5. We train our model for 150 epochs with initial learning rate of 0.5 and learning rate decay factor 0.95 if the model doesn't improve for the 3 last epochs.

7.5.2 Baseline Model

Our baseline model is a plain sequence-to-sequence model which has been used in the constituent parsing task (Vinyals et al., 2015). While they use a 3-layer deep LSTM, we only use a single-layer LSTM for both encoder and decoder since our data is relatively small and empirical comparison shows that stacking more layers doesn't help in our case. AMR linearization follows Section 7.3 without categorization. Since we don't restrict the input/output vocabulary in this case, our vocabulary size is quite large: 10,886 for output vocabulary and 2,2892 for input vocabulary. We set them to 10,000 and 20,000 respectively and replace the out of vocabulary words with `_UNK_`.

7.5.3 Impact of Re-Categorization

We first inspect the influence of utilizing categorization on the input and output sequence. Table 7.1 shows the Smatch evaluation score on development set.

System	P	R	F
Baseline	0.42	0.34	0.37
Re-Categorization ($n = 50$)	0.55	0.46	0.50

Table 7.1: Re-Categorization impact on development set

We see from the table that re-categorization improves the F-score by 13 points on the development set. As mentioned in section 7.4.1, by setting the low frequency threshold n to 50 and re-categorizing them into a reduced set of types, we now reduce the input/output vocabulary size to (2,000, 6,000). This greatly reduces the label sparsity and enables the neural attention model to learn a better representation on this small scale data. Another advantage of this method is that although AMR tries to abstract away surface forms and retain the semantic meaning structure of the sentence, a large portion of the concepts are coming from the surface form and have exactly same string form both in input sentence and AMR graph. For example, "`nation`" in sentence is mapped to concept "`(n / nation)`" in the AMR. For the frequent concepts in the output sequence, since the model can observe many training instances, we assume that it can be predicted by the attention model. For the infrequent concepts, because of the categorization step, we only require the model to predict the concept type and its relative position in the graph. By applying the post-processing step mentioned in Section 7.4.3, we can easily recover the categorized concepts to their original form.

We also inspect how the value of re-categorization frequency threshold n affects the AMR parsing result. As shown in Figure 7.6, setting n to 0, which means no output labels will

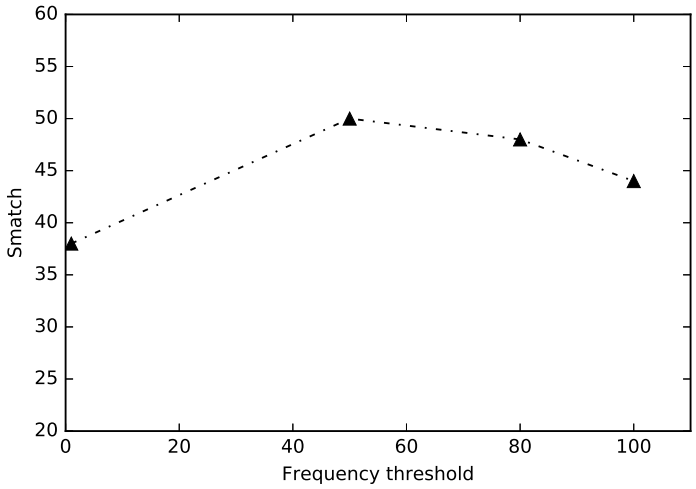


Figure 7.6: AMR parsing performance on development set given different categorization frequency.

be categorized into types -VERB- and -SURF-, doesn't improve the baseline system. The reason is that we still have a large output vocabulary size and training data is still sparse with respect to the low frequency output labels. Also, if we set n too high, although the output vocabulary size becomes smaller, some of the frequent output labels that the model handles well originally will be put into the coarse-grained types, losing information in the recovery process. Thus we can see from the plot that after the optimal point the Smatch score will drop. Therefore, we choose to set $n = 50$ in the subsequent experiments.

7.5.4 Impact of Supervised Alignment

Choice of External Aligner. There are two existing AMR aligners: one is a rule-based aligner coming with JAMR (Flanigan et al., 2014), which defines regular expression patterns to greedily match between AMR graph fragment and input token spans; another one is an unsupervised aligner (Pourdamghani et al., 2014) which adopts the traditional word

alignment method in machine translation. Although evaluated on different set of manual alignment test sentences, both aligners achieved ∼90% F1 score. Here we choose to use the second aligner, as it covers broader domains.

Different alignment configurations To balance between the sequence learning and alignment agreement, We empirically tune the hyperparameter λ and set it to 0.3. For the external alignment we use for reference, we convert it to a vector with equal probability as discussed in Section 7.4.4. We then train a sequence-to-sequence model with re-categorized input/output and report the result on development set.

System	P	R	F
Baseline	0.42	0.34	0.37
Categorization ($n = 50$)	0.55	0.46	0.50
SuperAttn+Cate ($n = 50$)	0.56	0.49	0.52

Table 7.2: Supervised attention impact on development set

As shown in Table 7.2, the supervised attention model is able to further improve the Smatch score by another 2 points, which are mainly contributed by 3 points increase in recall. Since the reference/external alignment is mostly between the input tokens and AMR graph concepts, we believe that the supervised attention model is able to constrain the decoder so that it will output concepts which can be aligned to some tokens in the input sentence.

System	P	R	F
SuperAttn+Cate ($n = 50$)	0.56	0.49	0.52
No-Relation-Align	0.46	0.40	0.43

Table 7.3: Supervised attention impact on development set

Because we have relations in the AMR graph, the alignment problem here is different

from the word alignment in machine translation. To verify the effectiveness of our setup, we also compare our configuration to the condition **No-Relation-Align** where we only ignore the alignment between sentence and AMR relations by putting an all zero vector as the reference attention for each output relation label. From Table 7.3 we see that simply ignoring the reference attention for relations would greatly affect the model performance, and how we effectively represent the reference alignment for relations is crucial for the supervised attention model.

7.5.5 Results

In this section we report our final result on the test set of SemEval 2016 Task 8 and compare our model with other parsers. We train our model utilizing re-categorization and supervised attention with hyperparameters tuned on the development set. Then we apply our trained model on the test set.

Firstly, we compare our model to the existing sequence-to-sequence AMR parsing model of Barzdins and Gosko (2016). As shown in table 7.4, the word-level model in Barzdins and Gosko (2016) is basically our baseline model. The second model they use is a character-based sequence-to-sequence model. Our model can also be regarded as a word-level model; however, by utilizing carefully designed categorization and supervised attention, our system outperforms both their results by a large margin.

Table 5 gives the comparison of our system to some of the teams participating in SemEval16 Task 8. Since a large portion of the teams extend on the our previous transition-based parser, `CAMR`, here we just pick typical teams that represent different approaches. We can see from the table that our system fails to outperform the state-of-the-art system. However, `CAMR` uses a dependency structure as a starting point, where dependency parsing has

System	P	R	F
Our system	**0.55**	**0.50**	**0.52**
Barzdins and Gosko (2016)†	-	-	0.37
Barzdins and Gosko (2016)⋆	-	-	0.43

Table 7.4: Compare to other sequence-to-sequence AMR parser. Barzdins and Gosko (2016)† is the word-level neural AMR parser, Barzdins and Gosko (2016)⋆ is the character-level neural AMR parser.

achieved high accuracy recently and can be trained on larger corpora. Also, it utilizes semantic role labeling and complex features, which makes the training process a long pipeline. Our system only needs minimal preprocessing, and doesn't need the dependency parsing step. Our approach is competitive with the SHRG (Synchronous Hyperedge Replacement Grammar) method of Peng et al. (2015), which does not require a dependency parser and uses SHRG to formalize the string-to-graph problem as a chart parsing task. However, they still need a concept identification stage, while our model can learn the concepts and relations jointly.

System	P	R	F
Our system	0.55	0.50	0.52
Peng and Gildea (2016)	0.56	0.55	0.55
CAMR	0.70	0.63	0.66

Table 7.5: Comparison to other AMR parsers.

7.6 Discussion

In this chapter, we have proposed several methods to make the sequence-to-sequence model work competitively against conventional AMR parsing systems. Although we haven't outperformed state-of-the-art system using the conventional methods, our results show the effec-

tiveness of our approaches to reduce the sparsity problem when training sequence-to-sequence model on a relatively small dataset. Our work could be aligned with the effort to handle low-resource data problems when building the end-to-end neural network model.

In neural machine translation, the attention model is traditionally trained on millions of sentence pairs, while facing low-resource language pairs, the neural MT system performance tends to downgrade (Zoph et al., 2016). There has been growing interest in tackling sparsity/low-resource problem in neural MT. Zoph et al. (2016) use a transfer learning method to first pre-train the neural model on rich-resource language pairs and then import the learned parameters to continue training on low-resource language pairs so that the model can alleviate the sparsity problem through shared parameters. Firat et al. (2016) builds a multilingual neural system where the attention mechanism can be shared between different language pairs. Our work could be seen as parallel efforts to handle the sparsity problem since we focus on the input/output categorization and external alignment, which are both handy for low-resource languages.

In this chapter, we haven't used any syntactic parser. However, as shown in previous works Flanigan et al. (2014); Wang et al. (2015b); Artzi et al. (2015); Pust et al. (2015), using dependency features helps improve the parsing performance significantly because of the linguistic similarity between the dependency tree and AMR structure. An interesting extension would be to use a linearized dependency tree as the source sequence and apply sequence-to-sequence to generate the AMR graph. Our parser could also benefit from the modeling techniques in Wu et al. (2016).

7.7 Conclusion

Neural attention models have achieved great success in different NLP tasks. However, they have not been as successful on AMR parsing due to the data sparsity issue. In this chapter, we described a sequence-to-sequence model for AMR parsing and present different ways to tackle the data sparsity problems. We show that our methods have led to significant improvement over a baseline neural attention model, and our model is also competitive against models that do not use extra linguistic resources.

Chapter 8

A Chinese AMR Parser

8.1 Introduction

Building AMR parsers that operate uniformly for multiple languages is important in a number of ways. In addition to benefiting downstream applications in each individual language, it is also beneficial to language processing tasks in multilingual settings. For NLP tasks involving parallel language processing, a multilingual AMR parser makes parallel semantic representations accessible. For example, one promising application is machine translation. As AMR is a meaning representation that abstracts away from the morpho-syntactic idiosyncracies of each individual language, it would be interesting to see whether it will be easier to conduct translations based on such intermediate layers. For NLP applications that aim to robustly work across multiple languages, a multilingual AMR parser provides an uniform interface to the semantic representation, potentially enabling applications to take advantages of the"deeper" meanings of text.

Previous work on multi-lingual AMR has either focused on the AMR representation or AMR annotated data projected from English. In a preliminary work, Xue et al. (2014) have

shown that it is feasible to align English-Chinese AMRs, based on an examination of 100 English-Chinese sentence pairs manually annotated with AMRs. Damonte and Cohen (2017) propose a transition-based AMR parser that adapts the ArcEager transition system for dependency parsing to AMR parsing, aiming to parse multilingual AMR data using a uniform framework. However, their parser is trained on AMR annotation generated by projecting the English AMR to a target language through word alignment. This process is prone to error propagation and based on strong isomorphism assumptions between language pairs that is often unrealistic, and as a result the performance of the parser is severely affected by the noise in the training data.

The recent availability of the Chinese AMR bank (Li et al., 2016) makes it possible to build an AMR parser for Chinese. In this chapter, we present the first AMR parser built using the Chinese AMR Bank. We adopt our transition-based parsing framework discussed in Chapter 3 and 4. Using the same parsing framework allows us to compare English and Chinese AMR parsing without considering parsing algorithm difference.

The rest of the chapter is organized as follows. In Section 8.2 we briefly describe the Chinese AMR Bank. In particular, we highlight areas where it departs from the English AMR Bank. As the theoretical background of our transition-based framework has been extensively discussed in the previous chapters, we mostly emphasize on the experimental analysis of Chinese AMR parsing in the following sections.

8.2 The Chinese AMR Bank

The Chinese AMR Bank (Li et al., 2016)[1] consists of 10,150 sentences extracted from the Chinese Treebank (CTB) 8.0 (Xue et al., 2005) [2], which mainly consists of Chinese texts of web news and discussion forum. The average length of the sentences is 22.43 words.

Similar to English AMR, the Chinese AMR Bank also represents the meaning of a sentence with a rooted, directed and acyclic graph, and it also uses many of the *abstract* concepts and relations used in the English AMR Bank. The sense-disambiguated predicates are drawn from the frame files developed for the Chinese Propbank (Xue and Palmer, 2009), just as the sense-disambiguated predicates in the AMR Bank are drawn from the Propbank (Palmer et al., 2005). About 47% sentences in the 10,150 sentences have re-entrance arcs, meaning that they have a graph structure that cannot be represented with a tree representation.

To accommodate the linguistic differences between the two languages, the Chinese AMR Bank also made a number of adaptations. First, the word-to-concept alignment is integrated with Chinese AMR annotation. The basic idea is to take the index of a word token as the ID of the concept it aligns to in the AMR representation, thus establishing the alignment between the AMR concepts and the word tokens. Sentence 1 is an example of Chinese AMR annotation with word-to-concept alignment. Using index of a word, the Chinese AMR Bank can handle complex alignment patterns such as one-to-zero, zero-to-one, one-to-many, and many-to-one alignments. When necessary, the index of a character within a word is also used to align the character to an AMR concept. For example, the index of the character 孩 is x1_2 and it can be used to align the character to a concept if needed.

Second, 10 intra-sentential discourse relations are proposed and 4 new kinds of relations are added in the annotation of Chinese AMRs. In English AMR, intra-sentential discourse

[1] `http://www.cs.brandeis.edu/~clp/camr/camr.html`.
[2] Available at `https://catalog.ldc.upenn.edu/LDC2013T21`.

relations are represented with a combination of abstract concepts (e.g., *and*, *or*, *contrast.01*) and relations (*:cause*, *:condition*, *:concession*, *:purpose*). In Chinese AMR, intra-sentential discourse relations are represented in a more uniform manner with 10 concepts defined in the Chinese Discourse TreeBank (CDTB) (Zhou and Xue, 2015). These discourse relations include *and*, *or*, *causation*, *condition*, *contrast*, *expansion*, *purpose*, *temporal*, *progression*, *concession*.

(1) 男孩[1] 想[2] 去[3] 纽约[4]。

boy want go New York

"The boy wants to go to New York."

```
(x2 / 想-01
    :arg0 (x1 / 男孩)
    :arg1 (x3 / 去-01
        :arg0 x1
        :arg1 (x5 / city
            :wiki (x4 / "纽约")
            :name (n / name :op1 (x4 /"纽约")))))
```

The 4 new relations added for Chinese are *:cunit*,*:tense*,*:aspect*,*:perspective*. *:cunit* is used to represent the "individual" classifiers in Chinese. For example, 匹 can be used to modify "wolf" or "horse" (e.g., 一/one 匹/CL 马/horse), but not other types of animals such as cows or pigs (e.g., *一/one 匹/CL 牛/cow). Tense and aspect are not realized as morphological inflections in Chinese. Instead, they are realized as separate lexical items that modify the verb. The new relations *:tense* and *:aspect* are used to represent the semantic

relation between the verb and its tense or aspect markers in Chinese AMR. *:perspective* is used to represent the perspective from which a statement made.

Finally, the Chinese AMR Bank has specification of how to handle linguistic phenomena that is specific to Chinese, and they include the number and classifier construction, the serial-verb construction, the headless relative construction, the verb-complement construction, the split verb construction, reduplications, etc.

8.3 Transition-based AMR Parsing

As we have discussed the transition-based parsing framework extensively in Chapter 3 and 4, we will briefly introduce here how the algorithm is applied on Chinese AMR bank.

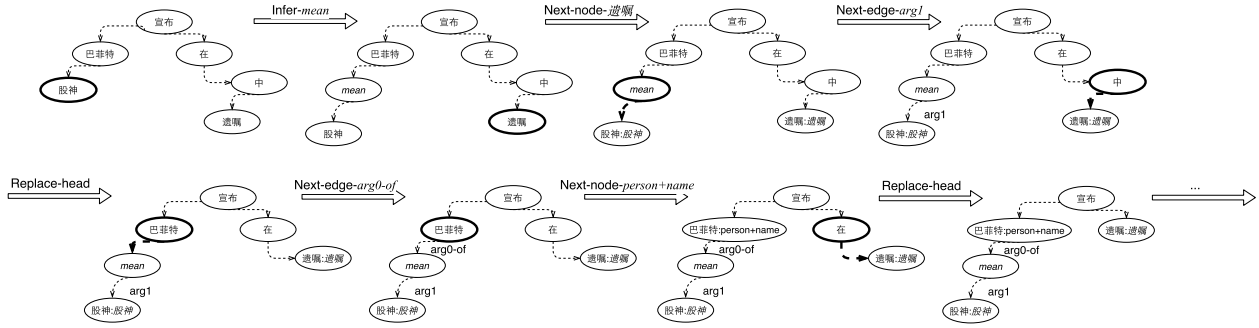


Figure 8.1: A running example for parsing sentence "股神(Stock god) 巴菲特 (Buffet) 在(in) 遗嘱 (testament) 中(inside) 宣布 (announce)."

Figure 8.1 shows a partial parsing process to transform the dependency tree for the Chinese sentence "股神 (Stock god) 巴菲特 (Buffet) 在 (in) 遗嘱 (testament) 中 (inside) 宣布 (announce)", where current node (topmost element in buffer σ) or current edge (σ_0,β_0) is highlighted in bold font. The italic part in each action is the parameter. We also sketch the *oracle* function for Chinese AMR parsing in Algorithm 3. Similar to its English counterpart, we use a set of heuristic rules to determine the "gold" action.

Algorithm 3 Oracle function

Input: partial parsed graph G_p, gold AMR graph G_g, two buffers (σ, β) and current parsing state indicator (σ_0, β_0)
Output: gold action t_g for current parsing state

$t_g \leftarrow None$
if β is empty **then**
    if node σ_0 is not in graph G_g **then**
        $t_g \leftarrow$ DELETE-NODE
    else
        $t_g \leftarrow$ NEXT-NODE-l_c
    end if
else
    if edge (σ_0, β_0) in graph G_g **then**
        $t_g \leftarrow$ NEXT-EDGE-l_r
    else if edge (β_0, σ_0) in graph G_g **then**
        $t_g \leftarrow$ SWAP-l_r
    else if ... **then**
        .....
    end if
end if
return G_p

Note that as stated in previous chapters, the action set is designed based on the intuition that the dependency tree and the AMR graph of a sentence share a lot of common structures. And some of the actions are inspired by some specific linguistic transformations from an English dependency tree to an English AMR. For example, SWAP-l_r (sw) addresses the case that in an English dependency tree, the head of a coordination structure is often the first conjunct but in its AMR, the coordinating conjunction is always the head. One critical question is whether such actions are generalizable to Chinese AMR parsing. We will verify this empirically in section 8.4.2 and show that the transition-based AMR parsing framework can largely work for Chinese AMR with little adaptation.

8.4 Experiments

In this section, we present a series of experiments designed to probe the behavior of our Chinese AMR parser, and where appropriate, compare it to its English counterpart. We also devise several ablation tests to further investigate the errors produced by our Chinese AMR parser to gain insight that can be used to guide future research.

8.4.1 Experiment Settings

We use the 10,150 sentences from the Chinese AMR Bank and split the data according to their original CTB8.0 document IDs, where articles 5061-5558 are used as the training set, articles 5000-5030 are used as the development set and articles 5031-5060 are used as the test set. The train/development/test ratio in this dataset is 7608/1264/1278. As the data are drawn from the Chinese Treebank where words are manually segmented, we will simply use the gold segmentation in our experiments. We then process the whole Chinese dataset using the Stanford CoreNLP (Manning et al., 2014) toolkit to get the POS and Named Entity tags. To get the dependency parse for the Chinese data, we use the transition-based constituent parser in (Wang and Xue, 2014) to first parse the Chinese sentences into constituent trees, which are then transformed into dependency trees using the converter in the Stanford CoreNLP toolkit. Note that this Chinese constituent parser also uses the Chinese Treebank 8.0 to train its model. To avoid training on the parser on AMR test set, we train the constituent parser using a 10-fold cross-validation with each fold parsed using a model trained on the other 9 folds. In order to compare results between Chinese and English, we also train an English AMR parsing model on the LDC2015E86 dataset used in SemEval 2016 Task 8 with the standard split 16833/1368/1371. All the AMR parsing results

are evaluated by the Smatch toolkit (Cai and Knight, 2013)[3].

8.4.2 Action Distribution

Before we train the parser, we first perform a quantitative comparison of the actions that are invoked in English and Chinese AMR parsing. We run the *oracle* function separately on the training data of both languages and record the distribution of actions invoked, as shown in Figure 8.2. Note that without any modification of the action set designed for English, the "pseudo-gold" graphs generated by the *oracle* function have reached 0.99 F1-score when compared with the gold Chinese AMR graphs, and this indicates that the action set is generalizable to Chinese. The numbers in the chart are showing the distribution of action types. We leave out the NEXT-EDGE-l_r and NEXT-NODE-l_c actions in the histogram as their main purpose are assigning labels and they do not trigger structural transformations, as other actions, thus are not our point of interest.

In Figure 8.2 we can see that there is a large difference in action distribution between Chinese and English. First of all, there are a lot fewer DELETE-NODE actions applied in the dependency-to-AMR transformation process, which indicates that in Chinese data there is a smaller percentage of "stop words" that do not encode semantic information. Also, in the Chinese data, more INFER-l_c actions are invoked than in English, implying that Chinese AMRs use more inferred concepts that don't align to any word tokens.

To further investigate the different linguistic patterns associated with each action in different languages, for each action type t, we randomly sample 100 sentences in which action t is invoked for both English and Chinese. We then conduct a detailed linguistic analysis over the sampled data. In the case of MERGE, we find that in English AMR parsing

[3] `http://alt.qcri.org/semeval2016/task8/data/uploads/smatch-v2.0.2.tar.gz`

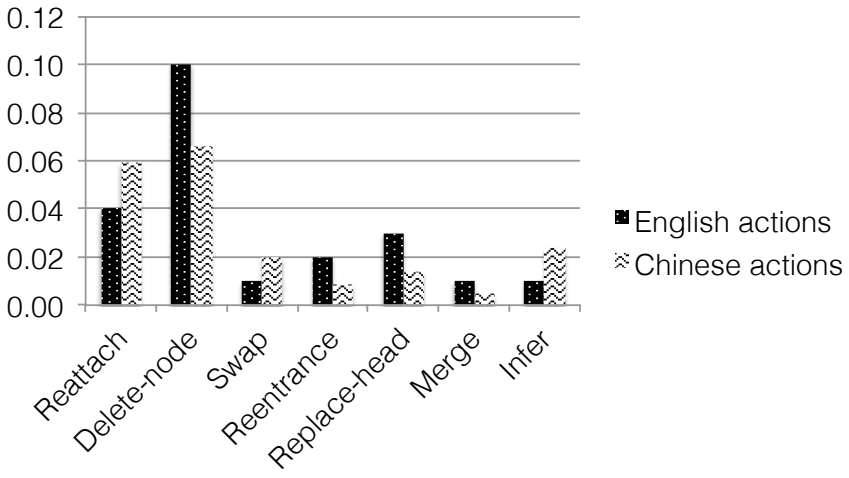


Figure 8.2: Action distribution on English and Chinese

it is mostly responsible for combining spans of words to form a named entity. However, in Chinese AMR parsing, besides being largely invoked to produce named entity concepts, the MERGE action also handles a large portion of split verb constructions. A "split verb" is a linguistic phenomenon in Chinese in which the characters in a multi-character verb can be split into two discontinuous parts by other other lexical items. For example, in the example shown in sentence (2), the MERGE action is invoked to merge the discontinuous parts "帮 (help)" and "忙 (business)" to form the AMR concept "帮忙-01".

In the cases of SWAP and REPLACE-HEAD, we notice that the linguistic transformations associated with the action are mostly consistent across the two languages. For example, as we already mentioned, the SWAP action is used to handle the divergent dependency treebank and AMR structures for the coordination construction. This observation also holds on Chinese data. Similarly, the REPLACE-HEAD action is designed to resolve the divergence involving prepositional phrases between a dependency tree and its corresponding AMR graph. In the version of the dependency structure we use, the preposition is always the head but

in the AMR graph, the preposition is often removed and instead represented as a relation (edge label). Based on our analysis on sampled data, the REPLACE-HEAD action resolves the same dependency-AMR divergence in Chinese AMR parsing.

(2) 他[1] 帮[2] 了[3] 我[4] 很[5] 大[6] 的[7] 忙[8]。

He helped PAST me very big DE business

"He helped me a lot."

```
(x / 帮忙-01
    :arg0 (x1 / 他)
    :arg1 (x4 / 我)
    :aspect (x3 / 了)
        :degree (x8 / 大
            :degree (x5 / 很)))
```

Being able to identify the linguistic environment for each action helps us understand what the parser actually does when actions are applied. More importantly, making the relation between the linguistic structure and the parser actions transparent is crucial to our ability to devise effective features for the parsing model which directly impact the performance of the parser. For example, knowing that the MERGE action is responsible for producing concepts from the split verb construction helps us understand the need to design character-level features in addition to named entity features.

8.4.3 Main results for Chinese AMR Parsing

Using the configuration in Section 8.4.1, we train our Chinese AMR parser with 5 iterations and report results both on the development and test set.

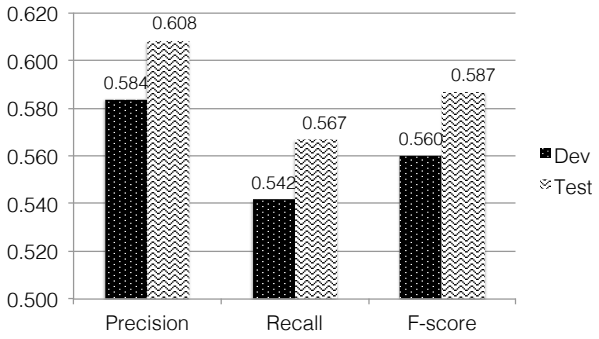


Figure 8.3: AMR parsing result on dev and test data of Chinese AMR bank

Figure 8.3 presents the parsing performance on the development and test set in terms of the Smatch score. Compared with the state of the art in English AMR parsing (high 60%), this initial parsing performance here is very strong, considering the model is trained on a smaller training set. The Chinese AMR parser also does not benefit from the more extensive feature engineering that has done for English AMR parsing. For example, our English AMR parser, `CAMR`, uses semantic role and coreference features that are not available to the Chinese AMR parser. The other important factor is that most of the Chinese linguistic analyzers (dependency parsers, named entity taggers, etc.) have a lower accuracy than their English counterparts, and when used as a preprocessors for the AMR parser, could further disadvantage the Chinese AMR parsing model as they could introduce noise into the Chinese AMR parser. Another disadvantage is that these Chinese analyzers are mostly trained on newswire data and when applied to weblog and discussion forum data they are prone to performance degradation.

8.4.4 Impact of Gold Syntactic Trees

Since Chinese AMR data is drawn from the Chinese Treebank, we could readily get the gold dependency tree by simply applying constituent-to-dependency converter to the Chinese treebank data. This gives us a way to investigate the impact of automatic dependency parsing on AMR parsing.

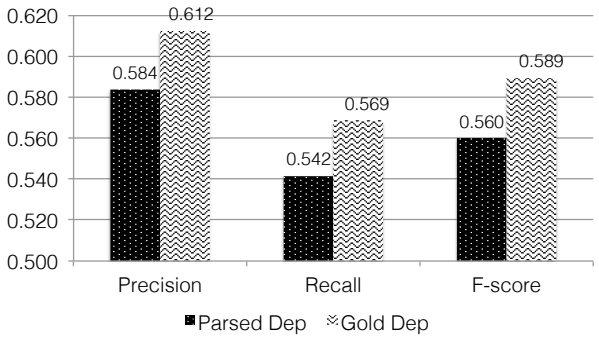


Figure 8.4: AMR parsing result on development set using parsed and gold dependency tree.

From Figure 8.4 we can see that using the gold dependency tree brings about a gain of 2 percentage to the AMR parsing result, boosting precision and recall by a similar margin. However, the improvement is still relatively small. This indicates that the presence of gold dependency structure has a limited impact on inferring semantic structures in AMR. This also implies that although utilizing the dependency structure helps to define a good starting point, a lot more work is happening in the second step of AMR parsing, which is the step that transforms the dependency structure to AMR structure.

8.4.5 Impact of Gold Word-to-AMR Concept Alignment

As the Chinese AMR Bank comes with manual alignment between word tokens and AMR concepts, all the parsing results are reported using the gold standard alignment. Notice

that just like in machine translation, alignment is needed only in the training phase and it is not used in test phase, so using manual alignment does not artificially boost the AMR parsing performance. However, we do expect that manual alignment in the training data will help train a model with less noise and this will translate to improved, though still realistic, performance in the testing phase. However, this is still a hypothesis and no previous study has been conducted to support this since the only alternative AMR resource available is the English AMR Bank and it does not have manual alignment. To investigate this, we use the unsupervised sentence-to-AMR aligner from Pourdamghani et al. (2014), which has been often used in training English AMR parsers. As it works similar to the word aligner used in machine translation and can be trained in an unsupervised manner, it can be easily applied to Chinese AMR parsing.

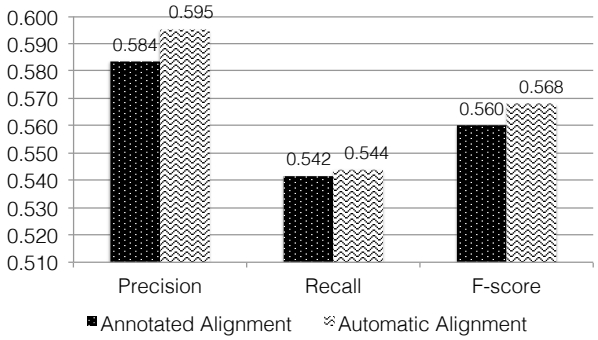


Figure 8.5: AMR parsing with annotated alignment and automatic alignment

The results are somewhat surprising. From Figure 8.5 we can see that the AMR parser trained on the automatic alignment is slightly better than the parser trained on gold (annotated) alignment. One possible reason is that part of the annotated alignment information is hard to be utilized by our transition-based parser. In the gold alignment provided by annotators, sometimes one Chinese word with more than one characters could be aligned to a subgraph (one-to-many). For example, "30余 (more than 30)" as a single word is usually

aligned to the AMR subgraph in character level, as illustrated in Figure 8.6. However, it will be quite difficult for our transition-based parser to utilize this level alignment information as it doesn't define concept label at such granularity. Future work has to extend the parser to take advantage of such alignment information and character-level feature also has to be incorporated in the parser in order to predict concept label like this. We plan to further investigate the problem to better understand this result in our future work.

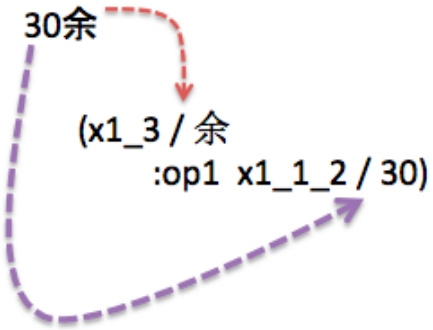


Figure 8.6: Example for one annotated alignment

8.4.6 Fine-grained Error Analysis

So far, all of our experiments are evaluated using Smatch score, where only precision, recall and F-score are reported based on the overall performance of the parsing result. To gain more insights, we further break down the Smatch score and report the performance for each component using the evaluation tool from Damonte et al. (2017). The evaluation tool examines different facets of the AMR parsing result through different ablation tests that we summarize as follows. The detailed description of the ablation test can be found in (Damonte et al., 2017).

- **Unlabeled**. Smatch score obtained by ignoring the edge labels (relation).
- **No WSD**. Smatch score without the word sense disambiguation.
- **NP (Noun Phrase)-only**. Only evaluating the noun phrases.
- **Reentrancy**. Only evaluating reentrancy edges.
- **Concepts**. Evaluating the node labels (concept).
- **Named Ent**. Named entity evaluation.
- **Negation**. Evaluation on negation detection.
- **SRL**. Semantic Role Labeling, which only evaluates triples in AMR that have relations starting with *:ARG*.

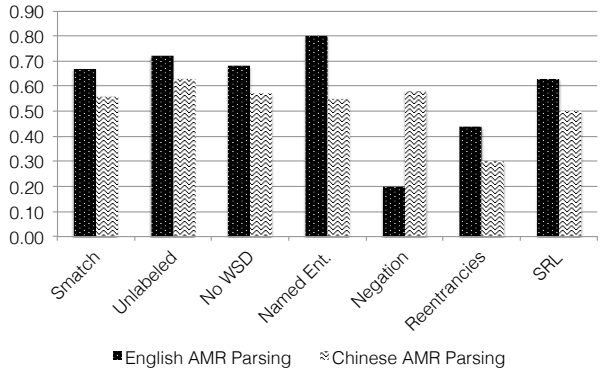


Figure 8.7: Fine-grained AMR parsing evaluation on dev

Note that we simply ignore the wikification evaluation as Chinese AMRs do not have wikification information annotated at the current stage.

Figure 8.7 shows the performance breakdown on the Chinese and English development set, where we can see that the overall gap is around 0.11 Smatch score and there is a similar

gap for **Unlabeled**, **No WSD** and **SRL** evaluations. However, the largest drop comes from **Named Ent.**, where the F-score for Chinese is 0.55 which is 0.25 away from the English named entity performance. This indicates that named entity is one of the bottlenecks in Chinese AMR parsing. To resolve this problem, one way is to improve the named entity recognizer in the preprocessing step. As we mentioned in section 8.4.3, there are two factors affecting the performance of the Chinese named entity recognizer (NER): inherent difficulty in Chinese named entity recognition due to the lack of surface clues such as capitalization and domain shift. We can either experiment with a stronger NER system or train it on same domain data. The other approach is to treat NER in AMR as an standalone task and directly optimize the model towards extracting the actual named entity labels existing in AMR. Similar ideas have been applied on predicting the concept labels (Foland and Martin, 2017; Wang and Xue, 2017) in English AMR parsing, leading to performance gains.

8.5 Conclusion

In this chapter, we present the first Chinese AMR parser trained on the Chinese AMR Bank. We show our transition-based AMR parsing framework has produced strong first results. In addition, we show that the same set of actions designed for English can apply to Chinese AMR parsing with little adaptation. We also investigated the impact of gold standard syntactic parses and word-to-AMR alignment. Our results show that there is a modest improvement when gold syntactic parses are used. When gold standard word-to-AMR alignment are used, however, the results are comparable to that achieved with fully automatic alignment. Further investigation is needed to pinpoint the reason for this somewhat surprising result. Finally, we performed a detailed error analysis that shows improved named entity recognition would be crucial in bridging the gap between English

and Chinese AMR parsing.

Chapter 9

Conclusion and Future Directions

In this dissertation, we have discussed various properties of AMR parsing and devised different algorithms to address them accordingly. To build the fundamental framework for graph parsing, we design a transition-based algorithm which formalizes AMR parsing as tree-to-graph transformation. Several natural extensions to the parser have been explored including exploring the richer feature space and inferring abstract concepts . To further address the Sparsity properties of AMR, we employ a neural sequence labeling technique for identifying concepts. We have observed that the rich context information brought by the neural framework is beneficial to AMR parsing and combining character-level information will further improve the neural concept identifier. We then come to the fundamental issue in AMR parsing — Abstraction and design an automatic aligner which is more appropriate for the sentence-to-graph alignment scenario. We also propose an end-to-end Neural AMR parser which explores the possibility of handling all the AMR phenomenons using an integrated model. Finally, we extend our knowledge learned on English AMR parsing to Chinese AMR corpus, where we show that our transition-based framework can be easily applied to Chinese AMR parsing without much modification. We also make a thorough comparison between

parsing on these two languages.

The task of AMR parsing or semantic parsing in general has gained increasing amount of attention as it is essential to genuine natural language understanding. However, there is still a lot of room for improvement and we still have a long way to go in order to put such technique to practice. The following of this chapter summarizes the promising future directions for AMR parsing. Hopefully these ideas could inspire the researchers in this field and lead to further improvement.

9.1 Transition-based Neural AMR Parsing

The idea of building transition-based parsers with a feed-forward neural network is first introduced in (Chen and Manning, 2014). Such framework is quite appealing as the parser has a superior practical run-time and also achieves comparable results on the standard dependency parsing task. Subsequent research along this line (Dyer et al., 2015; Weiss et al., 2015; Zhou et al., 2015; Andor et al., 2016) further improves this framework and has made great progress in dependency parsing. Similarly, it is also promising to integrate neural learner in transition-based AMR parsing, especially in `CAMR`, where we already have a transition system tailored for AMR. Puzikov et al. (2016) makes the first attempt by substituting the perceptron learner in `CAMR` with a feed-forward neural network. However, their neural parser doesn't outperform its baseline system. One reason is that the feed-forward network only have access to local features and fails to capture the rich context as defined in `CAMR`. This indicates one possible research direction which focuses on how to integrate richer context information into a neural transition-based parser.

9.2 Sequence-to-Sequence Models with Graph Decoding

As we have discussed this topic in Chapter 2 and chapter 7, applying sequence-to-sequence models to AMR parsing has drawn an increasing attention in the graph parsing field. Such efforts can be roughly categorized into two directions: 1) one direction proceeds with more emphasis on the data augmentation technique, where either self-training (Konstas et al., 2017) or co-training (van Noord and Bos, 2017) is applied to obtain more data from external unannotated corpora. 2) The other direction focuses on graph parsing. One intuition is that although sequence-to-sequence models have proved to be capable of solving structural predication problems (Vinyals et al., 2015), it still requires that each output symbol in the decoder depends on its previous output symbol. This might be working fine for cases like machine translation or constituent parsing, where the bi-gram dependency among output symbols is able to reflect the *order* of output sequences. However, this property does not hold in AMR parsing scenario, as the linearized AMR sequences does not have an specific order. Buys and Blunsom (2017) proposes one solution by forcing the decoder to predict actions (as in the transition-based parser) rather than linearized AMR sequences. This enables the parser to use the unchanged sequence-to-sequence model without enforcing ordering among output sequences. Another possible solution is that we could augment the sequence-to-sequence model with a graph decoder, where we can relax the ordering constraint and meanwhile reflect graph properties in output sequences.

Bibliography

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Artzi, Y., Lee, K., and Zettlemoyer, L. (2015). Broad-coverage CCG semantic parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal. Association for Computational Linguistics.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

Barzdins, G. and Gosko, D. (2016). Riga at semeval-2016 task 8: Impact of smatch extensions and character-level neural translation on amr parsing accuracy. *arXiv preprint*

arXiv:1604.01278.

Basile, V., Bos, J., Evang, K., and Venhuizen, N. (2012). Developing a large semantically annotated corpus. In *LREC*, volume 12, pages 3196–3200.

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.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Buys, J. and Blunsom, P. (2017). Robust incremental neural semantic graph parsing. *CoRR*, abs/1704.07092.

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

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-*

guage Processing (EMNLP), pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Chiang, D., Andreas, J., Bauer, D., Hermann, K. M., Jones, B., and Knight, K. (2013). Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Meeting of the Association of Computational Linguistics*, Sofia, Bulgaria.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

Cross, J. and Huang, L. (2016). Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

Damonte, M. and Cohen, S. B. (2017). Cross-lingual abstract meaning representation parsing. *CoRR*, abs/1704.04539.

Damonte, M., Cohen, S. B., and Satta, G. (2017). An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 536–546, Valencia, Spain. Association for Computational Linguistics.

DeNero, J. and Klein, D. (2007). Tailoring Word Alignments to Syntactic Machine Translation. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 17–24.

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 the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China. Association for Computational Linguistics.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Language, speech, and communication. MIT Press.

Firat, O., Cho, K., and Bengio, Y. (2016). Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, San Diego, California. Association for Computational Linguistics.

Flanigan, J., Dyer, C., Smith, A. N., and Carbonell, J. (2016). Cmu at semeval-2016 task 8: Graph-based amr parsing with infinite ramp loss. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1202–1206. Association for Computational Linguistics.

Flanigan, J., Thomson, S., Carbonell, J., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Baltimore, Maryland. Association for Computational Linguistics.

Foland, W. and Martin, H. J. (2016). CU-NLP at SemEval-2016 Task 8: AMR parsing using LSTM-based recurrent neural networks. In *Proceedings of the 10th International Workshop*

on Semantic Evaluation (SemEval-2016), pages 1197–1201. Association for Computational Linguistics.

Foland, W. and Martin, J. (2017). Abstract meaning representation parsing using lstm recurrent neural networks. In *Proceedings of the 55th Annual Meeting of the Association of the Computational Linguistics*, Vancouver, Canada. Association for Computational Linguistics.

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

Goodman, J., Vlachos, A., and Naradowsky, J. (2016a). Noise reduction and targeted exploration in imitation learning for abstract meaning representation parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–11. Association for Computational Linguistics.

Goodman, J., Vlachos, A., and Naradowsky, J. (2016b). Ucl+sheffield at semeval-2016 task 8: Imitation learning for amr parsing with an alpha-bound. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1167–1172. Association for Computational Linguistics.

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

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics.

Kamp, H. and Reyle, U. (1993). *From discourse to logic: Introduction to model theoretic semantics of natural language, formal logic and discourse representation theory.* Kluwer, Dordrecht.

Khardon, R. and Wachman, G. (2007). Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8(Feb):227–248.

Kiperwasser, E. and Goldberg, Y. (2016). Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Kipper, K., Korhonen, A., Ryant, N., and Palmer, M. (2006). Extending verbnet with novel verb classes. In *Proceedings of LREC*, volume 2006, page 1.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural AMR: sequence-to-sequence models for parsing and generation. *CoRR*, abs/1704.08381.

Koo, T., Carreras, X., and Collins, M. (2008). Simple semi-supervised dependency parsing. *ACL-08: HLT*, page 595.

Lee, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M., and Jurafsky, D. (2013). Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916.

Li, B., Wen, Y., Bu, L., Qu, W., and Xue, N. (2016). Annotating the little prince with Chinese AMRs. *LAW X*, page 7.

Liang, P., Taskar, B., and Klein, D. (2006). Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American*

Chapter of the Association of Computational Linguistics, pages 104–111. Association for Computational Linguistics.

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1077–1086, Denver, Colorado. Association for Computational Linguistics.

Liu, L., Utiyama, M., Finch, A., and Sumita, E. (2016). Neural Machine Translation with Supervised Attention. *ArXiv e-prints*.

Luong, M.-T. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. In *Association for Computational Linguistics (ACL)*, Berlin, Germany.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60.

Martins, A., Almeida, M., and Smith, N. A. (2013). Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics.

Matthiessen, C. and Bateman, J. (1991). *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Communication in artificial intelligence. Pinter.

May, J. (2016). Semeval-2016 task 8: Meaning representation parsing. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1063–1073, San Diego, California. Association for Computational Linguistics.

Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). The nombank project: An interim report. In Meyers, A., editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 24–31, Boston, Massachusetts, USA. Association for Computational Linguistics.

Misra, D. K. and Artzi, Y. (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1775–1786, Austin, Texas. Association for Computational Linguistics.

Mitra, A. and Baral, C. (2016). Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning.

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

Nivre, J., Hall, J., and Nilsson, J. (2006). Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

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

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1130–1139.

Peng, X. and Gildea, D. (2016). UofR at SemEval-2016 Task 8: Learning synchronous hyperedge replacement grammar for AMR parsing. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1185–1189. Association for Computational Linguistics.

Peng, X., Song, L., and Gildea, D. (2015). A synchronous hyperedge replacement grammar based approach for AMR parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Pourdamghani, N., Gao, Y., Hermjakob, U., and Knight, K. (2014). Aligning English strings with abstract meaning representation graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 425–429.

Pradhan, S., Moschitti, A., Xue, N., Ng, H. T., Björkelund, A., Uryupina, O., Zhang, Y., and Zhong, Z. (2013). Towards robust linguistic analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152, Sofia, Bulgaria.

Pradhan, S., Ward, W., Hacioglu, K., Martin, J., and Jurafsky, D. (2004). Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technol-*

ogy Conference/North American chapter of the Association of Computational Linguistics (HLT/NAACL), Boston, MA.

Pust, M., Hermjakob, U., Knight, K., Marcu, D., and May, J. (2015). Parsing English into abstract meaning representation using syntax-based machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1143–1154, Lisbon, Portugal. Association for Computational Linguistics.

Puzikov, Y., Kawahara, D., and Kurohashi, S. (2016). M2l at semeval-2016 task 8: Amr parsing with neural networks. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1154–1159, San Diego, California. Association for Computational Linguistics.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado. Association for Computational Linguistics.

Roth, M. and Lapata, M. (2016). Neural semantic role labeling with dependency path embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1192–1202, Berlin, Germany. Association for Computational Linguistics.

Sachan, M. and Xing, E. (2016). Machine comprehension using rich semantic representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 486–492. Association for Computational Linguistics.

Santos, C. N. d. and Guimaraes, V. (2015). Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*.

Sartorio, F., Satta, G., and Nivre, J. (2013). A transition-based dependency parser using a dynamic parsing strategy. In *ACL (1)*, pages 135–144.

Sawai, Y., Shindo, H., and Matsumoto, Y. (2015). Semantic structure analysis of noun phrases using abstract meaning representation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 851–856, Beijing, China. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014a). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014b). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Takase, S., Suzuki, J., Okazaki, N., Hirao, T., and Nagata, M. (2016). Neural headline generation on abstract meaning representation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1054–1059, Austin, Texas. Association for Computational Linguistics.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

van Noord, R. and Bos, J. (2017). Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *CoRR*, abs/1705.09980.

Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Vogel, S., Ney, H., and Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.

Wang, C., Pradhan, S., Pan, X., Ji, H., and Xue, N. (2016). CAMR at semeval-2016 task 8: An extended transition-based AMR parser. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1173–1178, San Diego, California. Association for Computational Linguistics.

Wang, C. and Xue, N. (2017). Getting the most out of amr parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1268–1279, Copenhagen, Denmark. Association for Computational Linguistics.

Wang, C., Xue, N., and Pradhan, S. (2015a). Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers)*, pages 857–862.

Wang, C., Xue, N., and Pradhan, S. (2015b). A transition-based algorithm for AMR parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado. Association for Computational Linguistics.

Wang, Z. and Xue, N. (2014). Joint pos tagging and transition-based constituent parsing in chinese with non-local features. In *Proceedings of the 52nd Annual Meeting of the Associ-*

ation for Computational Linguistics (Volume 1: Long Papers), pages 733–742, Baltimore, Maryland. Association for Computational Linguistics.

Weiss, D., Alberti, C., Collins, M., and Petrov, S. (2015). Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China. Association for Computational Linguistics.

Werling, K., Angeli, G., and Manning, C. (2015). Robust subgraph generation improves abstract meaning representation parsing. *arXiv preprint arXiv:1506.03139*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Xue, N., Bojar, O., Hajič, J., Palmer, M., Urešová, Z., and Zhang, X. (2014). Not an interlingua, but close: Comparison of English AMRs to Chinese and Czech. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation(LREC)*.

Xue, N. and Palmer, M. (2009). Adding semantic roles to the chinese treebank. *Natural Language Engineering*, 15(1):143–172.

Xue, N., Xia, F., Chiou, F., and Palmer, M. (2005). The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Zhong, Z. and Ng, H. T. (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. In *Proceedings of the ACL 2010 System Demonstrations*, pages 78–83, Uppsala, Sweden.

Zhou, H., Zhang, Y., Huang, S., and Chen, J. (2015). A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China. Association for Computational Linguistics.

Zhou, J., Xu, F., Uszkoreit, H., QU, W., Li, R., and Gu, Y. (2016). Amr parsing with an incremental joint model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 680–689, Austin, Texas. Association for Computational Linguistics.

Zhou, J. and Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1127–1137, Beijing, China. Association for Computational Linguistics.

Zhou, Y. and Xue, N. (2015). The chinese discourse treebank: a chinese corpus annotated with discourse relations. *Language Resources & Evaluation*, 49(2):1–35.

Zoph, B., Yuret, D., May, J., and Knight, K. (2016). Transfer Learning for Low-Resource Neural Machine Translation. *ArXiv e-prints*.