Parsing and Generation for the Abstract Meaning Representation

Jeffrey Flanigan

CMU-LTI-18-018

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies*

For my family

Abstract

A key task in intelligent language processing is obtaining semantic representations that abstract away from surface lexical and syntactic decisions. The Abstract Meaning Representation (AMR) is one such representation, which represents the meaning of a sentence as labeled nodes in a graph (concepts) and labeled, directed edges between them (relations). Two traditional problems of semantic representations are producing them from natural language (parsing) as well as producing natural language from them (generation). In this thesis, I present algorithms for parsing and generation for AMR.

In the first part of the thesis, I present a parsing algorithm for AMR that produces graphs that satisfy semantic well-formedness constraints. The parsing algorithm uses Lagrangian relaxation combined with an exact algorithm for finding the maximum, spanning, connected subgraph of a graph to produce AMR graphs that satisfy these constraints.

In the second part of the thesis, I present a generation algorithm for AMR. The algorithm uses a tree-transducer that operates on a spanning-tree of the input AMR graph to produce output natural language sentences. Data-sparsity of the training data is an issue for AMR generation, which we overcome by including synthetic rules in the tree-transducer.

Acknowledgments

First, I thank my advisors Jaime Carbonell, Chris Dyer and Noah Smith for their support and encouragement while completing my PhD. They were always there to help me strategize and keep moving forward towards my goals, whether it was the next research paper or this completed thesis. They taught me how to pursue challenging research questions and solve difficult problems along the way. And they taught me how to be rigorous in my research, and how to write better and explain it to others. I thank my thesis committee member Dan Gildea for wonderful discussions about this thesis and other topics.

I thank professors who taught me while at CMU: Lori Levin, Stephan Vogel, Alon Lavie, Alan Black, Eric Xing, Bob Frederking, William Cohen, Teruko Mitamura, Graham Neubig, and Eric Nyberg. Lori Levin taught me a tremendous amount about linguistics and the variation in world's languages, and gave me a solid background of linguistic expertise that I continue to use today. I am thankful for her careful teaching and generosity and support during the first years of my PhD.

I thank my colleagues: Waleed Ammar, Miguel Ballesteros, Dallas Card, Victor Chahuneau, Jon Clark, Shay Cohen, Dipanjan Das, Jesse Dodge, Manaal Faruqui, Kevin Gimpel, Greg Hanneman, Kenneth Heafield, Kazuya Kawakami, Lingpeng Kong, Guillaume Lample, Wang Ling, Fei Liu, Austin Matthews, Avneesh Saluja, Naomi Saphra, Nathan Schneider, Eva Schlinger, Yanchuan Sim, Swabha Swayamdipta, Sam Thomson, Tae Yano, and Yulia Tsvetkov, and all the researchers at the First Fred Jelinek Memorial Summer Workshop. I thank Sam Thomson for wonderful collaborations along the years, some of which went into this thesis.

I thank my wife, Yuchen, for being a great friend and companion, and for helping me stay focused when I had to work and have fun when I needed a break. I thank my parents Jim and Jane for their unconditional love and support throughout the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about producing meaning representations (MRs) from natural language and producing natural language from MRs, or in other words, mapping in both directions between natural language and MRs. The major contributions are algorithms for learning the mapping between text and MR in both directions using supervised structured prediction methods.

The representation we use is the Abstract Meaning Representation (AMR) (Banarescu et al., 2013), which has been designed with the idea of using it as an intermediate representation in machine translation. AMR represents the meaning of a sentence as labeled nodes in a graph (concepts), and labeled directed edges between them (relations). It uses an inventory of concepts from English and PropBank (Palmer et al., 2005), and captures "who-is-doing-what-to-whom" in a propositional style logic, abstracting away from variations in surface syntax. AMR is not an interlingua, because it uses an inventory of concepts based on a natural language lexicon (English). A key recent development in AMR is that is has been used in a large annotation effort (Banarescu et al., 2013). So far over 39,000 sentences have been annotated (Knight et al., 2016).

With annotated data, it becomes conceivable to train supervised algorithms to map in both directions between natural language text and AMR. In the first part of this thesis, we demonstrate that it is indeed possible to learn to map English text into AMR using this corpus (**semantic parsing from English to AMR**), and develop algorithms for doing this. In the second part of this thesis, we demonstrate that it is also possible to learn to map from AMR to English text (**generation of English from AMR**).[1]

While we develop algorithms and techniques for AMR, they are expected to be applicable to any model of semantics that represents the meaning of a sentence as a graph. While syntax is usually captured with tree structures, it is often argued that propositional semantics is better represented with graph structures (Melcuk, 1988, Banarescu et al., 2013). The desire to move from syntactic analysis to deeper semantic analysis has stimulated research into new algorithms that can handle these graph structures. This thesis contributes to this line of research.

Although the work in this thesis pre-dates in a way the rise of neural methods in NLP and we do not use them in this thesis, the work is still relevant as we move into the age of deep learning in two ways. First, the algorithms we develop can be used with neural factors (see for example Lyu and Titov (2018)), and are important for parsing if one wants to guarantee well-formedness of the AMR graphs. Second, discrete semantic representations like AMR capture the propositional content of language explicitly and concisely. This makes these MRs good for downstream applications that depend on preserving this content, and also good for applications that demand interpretability.

It is possible that AMR or some other MR will significantly advance the state of the art in one or more NLP tasks. Because AMR captures relations between concepts but abstracts away from surface syntax, it may be used as an intermediate representation in a variety of tasks such as machine translation (MT), summarization, and question answering (QA). In a preliminary study, my co-authors and I have used AMR as an intermediate representation for summarization (Liu et al., 2015), and based on the orig-

[1]With some underspecification in the generated English because AMR does not express all the semantics English does.

inal motivation and origins of AMR, future work on AMR applied to MT is expected. In fact, recently AMR has been used to considerably advance the state of the art in a set of QA tasks (Mitra and Baral, 2016). Further research is necessary to determine if AMR or some other MR will prove to be widely useful in these tasks. The ability to learn mappings from natural language to MRs from annotated corpora is an important step for building systems that use intermediate MRs.

The rest of this section is an overview of the thesis document, followed by the thesis statement and contributions.

1.1 Parsing

Parsing English into AMR is taking an input English sentence and producing an AMR graph as output. In this work a broad coverage parser is learned from example sentences paired with their meaning representations.

The approach we take is a two-stage approach that first identifies the concepts in a sentence with a sequence labeling algorithm, and then adds relations between them to produce a well-formed AMR graph. We include recent progress in concept identification and relation identification.

1.2 Generation

Generation from AMR is the production of a sentence from an input AMR graph. This is important for downstream applications such as summarization and machine translation, and success in this task increases AMR's usefulness as an intermediate representation.

Our approach to generating English sentences from AMR is to first compute one or more spanning trees of the AMR graph, and then use a tree transducer to convert the tree into an English sentence. The tree transducer rules are learned from the AMR corpus, and there are also synthetic rules that are created on the fly. The system is trained discriminatively, with various features which include a language model.

1.3 Thesis statement

> A key task in intelligent language processing is obtaining a representation of natural language expressions that abstracts away from surface lexical and syntactic decisions, in favor of an abstract semantic representation. We show that we can map into and out of the Abstract Meaning Representation (AMR) formalism using supervised structured prediction techniques based on graphs.

1.4 Contributions of this thesis

The main contributions of this thesis are:

- A two-stage approach to semantic parsing for AMR, where concepts are first identified using a sequence labeling algorithm, and relations are then added using a graph algorithm.
- An approximate inference algorithm for finding the maximum weight, spanning connected subgraph of a graph with linear constraints and weights on the edges.
- A two-stage approach to generation from AMR, where the graph is first converted to a tree, and then the tree is transduced into a string.
- A rule-based approach to automatic alignment of AMR graphs to the training data sentences, which is used to train the semantic parsers and generators.
- A new loss function which generalizes SVM loss for structured prediction when the training data contains unreachable training examples.

- A software package for AMR parsing and generation (called JAMR), released open-source at `http://github.com/jflanigan/jamr`.

Chapter 2

Abstract Meaning Representation

The Abstract Meaning Representation (AMR) is a whole-sentence semantic representation. It captures relational semantics, or "who-is-doing-what-to-whom" in a directed graph structure. While semantic representations like this have been around for a while (Alshawi, 1992, Copestake and Flickinger, 2000, Apresian et al., 2003, inter alia), the major innovation of AMR is that it has been designed for rapid large-scale annotation, and has been used to annotate a corpus of sentences with AMRs, creating a semantics bank or **sembank** (Banarescu et al., 2013).[1] This allows supervised machine learning of semantic parsers and generators, as apposed to using hand-crafting or indirect supervision.

In this chapter we give an overview of AMR, its philosophy, and how various linguistic phenomenon are handled in AMR. Complete understanding of this chapter is not necessary to read the rest of this thesis, but this information should be useful as background information for researchers wishing to use AMR in their work.

[1]Large-scale annotation of semantic representations itself is not new, and has been done in work such as Böhmová et al. (2003) and Basile et al. (2012).

2.1 Overview of AMR

AMR represents meaning as a graph structure, where nodes in the graph are called **concepts** and the edges are called **relations**. See Figure 2.1 for an example. In the sentence "The boy wants to go to the store," there are four AMR concepts: `boy`, `want-01`, `go-02`, and `store`. Relations are labeled, directed edges between concepts.

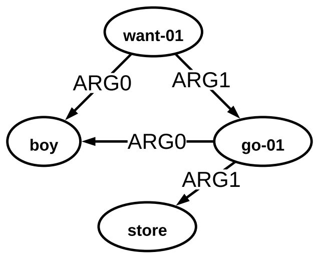


Figure 2.1: AMR for the sentence "The boy wants to go to the store."

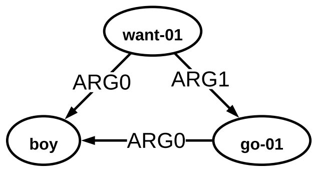
AMR graphs can be represented graphically or textually in various ways (Figure 2.2). The textual format 2.2a, or PENMAN notation, is the format used in the AMR sembank (see §2.2). The graphical format in Figure 2.2b is equivalent to the PENMAN format in Figure 2.2a.

Unlike some meaning representations, AMR does not specify how the graph can be derived from the syntax of the sentence. The annotators write down the meaning of the sentence as specified in AMR, without any explicit connections to the sentence.

Linguistically, AMR abstracts away from syntax while retaining lexical meaning. Language contains **grammatical skewing**, or mismatch between syntactic form and semantic meaning. For example, the grammatical categories nouns and verbs do not correspond one-to-one with the semantic categories of things and events. Many events can be expressed as nouns, and in English, almost any noun can be "verbed," or coerced into a verb using the right context. For example, "destruction" (a noun), represents the event "destroy." An example of "verbing" a noun is the sentence "Beer me," meaning

```
(w \ want-01
   :ARG0 (b / boy)
   :ARG1 (g / go-01
     :ARG0 b))
```

(a) PENMAN textual format.



(b) Graphical format used in this thesis.

Figure 2.2: Ways of representing the AMR graph for the sentence "The boy wants to go."

give me a beer.

One of the goals of AMR is to normalize and remove the effects of grammatical skewing. AMR uses **event concepts** for events, and **open class concepts** for things and properties. More generally, AMR abstract away from idiosyncrasies of syntax and from idioms, the motivation being that by "lifting the veil of syntax," NLP applications can access and express meaning in language more directly.

It is important to note that AMRs are in general not trees. Unlike syntax, which can be represented as trees, semantics is best represented as a graph structure (Melcuk, 1988). This motivates the development of new algorithms for handling semantic graphs, the subject of this thesis, rather than re-using the machinery developed for syntactic trees.

2.2 PENMAN Notation

AMRs are represented textually in the sembank in **PENMAN notation**. This format represents a directed graph in a simple, tree-like form. Each node in the graph is given a **variable**, and is labeled with a concept.[2] The variable is used to refer to the node in other places in the PENMAN notation (called a **re-entrancy**). The edges (red) are relations. The top node in the AMR is called the **focus** or **root**. This node is not a root in the graph theoretic sense, but just a node singled out as being the main focus of the AMR. The PENMAN notation allows for **inverse relations**, which are edges which point in the opposite direction from usual edges. These are indicated by adding `-of` at the end of the relation as in the example: teacher = `(person :ARG0-of teach-01)`.

In PENMAN notation, it does not matter where the concept label goes, or which relations are inverse relations. Although the PENMAN notation does implicitly define a tree over the AMR graph (the tree obtained by removing re-entrancies), AMR eschews using this tree because it does not carry meaningful information according to the annotation guidelines. The information in this tree will be discussed more in the generation chapter (§5).

2.3 Concepts

Concepts in AMR can be categorized into four broad categories: event concepts, open class concepts, special concepts, and constants. These will be discussed in turn. Concepts are token-level as opposed to type-level, so if there are two instances of a concept mentioned in a sentence, then there are two instances in the AMR graph. For example, in the sentence "There is a red truck and a blue truck," there are two trucks, so there are two `truck` concepts in the AMR.

[2]Constants like numbers or strings are not given a variable name.

2.3.1 Event Concepts

Event concepts, such as `run-01`, `walk-01`, are concepts that end in `-XX`, where `XX` is a number less than 91 (numbers greater than 90 denote special concepts, see §2.3.4). The ending number denotes a sense-disambiguated concept or **frame**. The inventory of frames comes from PropBank (Palmer et al., 2005) if there is an appropriate frame in PropBank. For event concepts not in PropBank, the ending `-00` is used as a place-holder frame. The sense disambiguated frame defines the core relations the concept can have, discussed in §2.4.1.

2.3.2 Open Class Concepts

Open class concepts are concepts that do not have a number and come from the lemmatized version of an English word, such as `boy`, `woman`, `red`, `store`. Most "thing" and "property" concepts fall into this class. Nouns that represent events (event nominals) are converted into event concepts, and correspondingly, verbs that represent nouns (verbalized nouns) are converted into open-class concepts.

Open class concepts are not sense disambiguated in the current version of AMR. However, in the future, nouns that take arguments may be disambiguated to nouns in NomBank.

2.3.3 Constants

Constants are concepts which are numbers, strings or symbols, and are not given a variable name. Examples of constants are `2001` in `(d / date :year 2001)`, `"Joe"` in `(p / person :name "Joe")`, and `-` in `(r / responsive :polarity -)`. While it makes sense to think of more than one car in a sentence (more than one instance of the `car` concept) and hence give it a variable name, it does not make sense to think of more than one instance of a constant concept, since it has the same meaning everywhere.

For parsing evaluation, adding a variable name to a constant (for example `(y / 2001)`) is scored as a mistake by Smatch (see §2.5 for discussion of Smatch).

2.3.4 Special Concepts

AMR has a handful of "special concepts" which are concepts that take `ARGN` arguments (core arguments) like event concepts but do not come from PropBank. Special concepts end in a number greater than 90. Examples include `include-91`, which is used to for inclusion, and `have-org-role-91`, which is used for roles in organizations.

2.4 Relations

Relations represent how concepts are related to one another and are labeled, directed edges in the AMR graph. There are two broad categories of relations in AMR: core relations and non-core relations. Additionally, some relations can be expressed either as a relation or as a concept using a process called re-ification. Most non-core relations can be re-ified.

2.4.1 Core relations

Core relations (also called core arguments or core roles) are the relations `ARGN`, where `N` is a number. Core relations are used as numbered arguments for event concepts and special concepts. For event concepts, the `ARGN` relations express core argument roles as specified in the PropBank rolesets (Palmer et al., 2005). `ARG0` and `ARG1` usually correspond to the agent and patient respectively, but the exact meaning and the meaning of higher numbered arguments are dependent on the sense-disambiguated event concept. For special concepts that take arguments, the meaning of each number argument depends on the concept and can be found in the AMR guidelines[3] or in the AMR editor[4] under "AMR dict." The editor's AMR dict is the most detailed and up-to-date information.

2.5 Smatch

Smatch (Cai and Knight, 2013) is used to evaluate the output of AMR parsers, and for scoring the similarity between AMR graphs for inter-annotator agreement. Smatch counts corpus level precision, recall, and F_1 of concepts and relations together between two corpora of AMR graphs.

2.6 Automatic Alignments

AMR does not specify any alignments between the nodes and edges in the graph and the sentence. However, in order to train our parser and generator, we need alignments between nodes in the graph and spans of words. To address this problem, we built an automatic aligner and tested its performance on a small set of alignments we annotated by hand, creating both the first automatic aligner and hand alignment dataset for AMR.

The automatic aligner uses a set of rules to greedily align concepts to spans. The list of rules is given in Table 2.1. The aligner proceeds down the list, first aligning named-entities exactly, then fuzzy matching named-entities, then date-entities, etc. For each rule, an entire pass through the AMR graph is done. The pass considers every concept in the graph and attempts to align a concept fragment rooted at that concept if the rule can apply. Some rules only apply to a particular type of concept fragment, while others can apply to any concept. For example, rule 1 can apply to any NAME concept and its

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

[4] `http://www.isi.edu/~ulf/amr/AMR-editor.html`

OP children. It searches the sentence for a sequence of words that exactly matches its OP children and aligns them to the NAME and OP children fragment.

Concepts are considered for alignment in the order they are listed in the AMR annotation (left to right, top to bottom). Concepts that are not aligned in a particular pass may be aligned in subsequent passes. Concepts are aligned to the first matching span, and alignments are mutually exclusive. Once aligned, a concept in a fragment is never re-aligned.[5] However, more concepts can be attached to the fragment by rules 8–16.

We use WordNet to generate candidate lemmas, and we also use a fuzzy match of a concept, defined to be a word in the sentence that has the longest string prefix match with that concept's label, if the match length is ≥ 4. If the match length is < 4, then the concept has no fuzzy match. For example the fuzzy match for ACCUSE-01 could be "accusations" if it is the best match in the sentence. WordNet lemmas and fuzzy matches are only used if the rule explicitly uses them. All tokens and concepts are lowercased before matches or fuzzy matches are done.

On the 200 sentences of training data we aligned by hand, the aligner achieves 92% precision, 89% recall, and 90% F_1 for the alignments.

[5]As an example, if "North Korea" shows up twice in the AMR graph and twice in the input sentence, then the first "North Korea" concept fragment listed in the AMR gets aligned to the first "North Korea" mention in the sentence, and the second fragment to the second mention (because the first span is already aligned when the second "North Korea" concept fragment is considered, so it is aligned to the second matching span).

1. **(Named Entity)** Applies to `name` concepts and their `opn` children. Matches a span that exactly matches its `opn` children in numerical order.
2. **(Named Entity Acronym)** Applies to `name` concepts and their `opn` children. Matches a span of words whose letters match the first letters of the `opn` children in numerical order, ignoring case, intervening spaces and punctuation.
3. **(Fuzzy Named Entity)** Applies to `name` concepts and their `opn` children. Matches a span that matches the fuzzy match of each child in numerical order.
4. **(Date Entity)** Applies to `date-entity` concepts and their `day`, `month`, `year` children (if exist). Matches any permutation of day, month, year, (two digit or four digit years), with or without spaces.
5. **(Minus Polarity Tokens)** Applies to `-` concepts, and matches the tokens "no", "not", "non", "nt", "n't."
6. **(Single Concept)** Applies to any concept. Strips off trailing '-[0-9]+' from the concept (for example `run-01` $\rightarrow$ `run`), and matches any exact matching word or WordNet lemma.
7. **(Fuzzy Single Concept)** Applies to any concept except `have-org-role-91` and `have-rel-role-91`. Strips off trailing '-[0-9]+', and matches the fuzzy match of the concept.
8. **(U.S.)** Applies to `name` if its `op1` child is `united` and its `op2` child is `states`. Matches a word that matches "us", "u.s." (no space), or "u. s." (with space).
9. **(Entity Type)** Applies to concepts with an outgoing `name` edge whose head is an aligned fragment. Updates the fragment to include the unaligned concept. Ex: `continent` in `(continent :name (name :op1 "Asia"))` aligned to "asia."
10. **(Quantity)** Applies to `.*-quantity` concepts with an outgoing `unit` edge whose head is aligned. Updates the fragment to include the unaligned concept. Ex: `distance-quantity` in `(distance-quantity :unit kilometer)` aligned to "kilometres."
11. **(Person-Of, Thing-Of)** Applies to `person` and `thing` concepts with an outgoing `.*-of` edge whose head is aligned. Updates the fragment to include the unaligned concept. Ex: `person` in `(person :ARG0-of strike-02)` aligned to "strikers."
12. **(Person)** Applies to `person` concepts with a single outgoing edge whose head is aligned. Updates the fragment to include the unaligned concept. Ex: `person` in `(person :poss (country :name (name :op1 "Korea")))` 12. **(Goverment Organization)** Applies to concepts with an incoming `ARG.*-of` edge whose tail is an aligned `government-organization` concept. Updates the fragment to include the unaligned concept. Ex: `govern-01` in `(government-organization :ARG0-of govern-01)` aligned to "government."
13. **(Minus Polarity Prefixes)** Applies to `-` concepts with an incoming `polarity` edge whose tail is aligned to a word beginning with "un", "in", or "il." Updates the fragment to include the unaligned concept. Ex: `-` in `(employ-01 :polarity -)` aligned to "unemployment."
14. **(Degree)** Applies to concepts with an incoming `degree` edge whose tail is aligned to a word ending is "est." Updates the fragment to include the unaligned concept. Ex: `most` in `(large :degree most)` aligned to "largest."
14. **(Have-Role-91 ARG2)** Applies to the concepts `have-org-role-91` and `have-rel-role-91` which are unaligned and have an incoming `ARG2` edge whose tail is aligned. Updates the fragment to include the `have-org-role-91` or `have-rel-role-91` concept.
15. **(Have-Role-91 ARG1)** Same as above, but replace `ARG2` with `ARG1`.
16. **(Wiki)** Applies to any concepts with an incoming `wiki` edge whose tail is aligned. Updates the fragment to include the unaligned concept.

Table 2.1: Rules used in the automatic aligner. As the annotation specification evolves, these rules need to be updated. These rules have been updated since Flanigan et al. (2014) to handle AMR annotation releases up to November 2017.

Chapter 3

Structured Prediction and Infinite Ramp Loss

The two problems tackled in this thesis, as well as their subproblems, are instances of structured prediction problems. **Structured prediction** (BakIr et al., 2007, Smith, 2011) is a paradigm of machine learning, like binary or multi-class classification, but distinguished from these in that the output has structure and the set of possible outputs can be infinite. Examples of structured prediction tasks include predicting a linear sequence of words, predicting a parse tree, or predicting a graph. These tasks can sometime be formulated as a sequence of multi-class classification decisions, but the view from structured prediction is more general and reasons about the entire output as a structured object (although transition-based methods, which rely on a sequence of multi-class classification decisions, are one of the structured prediction techniques). Structured prediction has been successfully applied to a wide variety of NLP and non-NLP problems.

In this chapter, we give as background an overview of the structured prediction and the techniques used in this thesis. Structured prediction models have six parts: the input space $\mathcal{X}$, the output space function $\mathcal{Y}(x)$, the scoring function $score(x, y)$, a decoding algorithm, a loss function $L(\mathcal{D}, \theta)$, and an optimization method for minimizing the loss function. In the following, we give a brief overview of these six parts.

We also present a new loss function for structured prediction as a contribution of this thesis – work that was presented in Flanigan et al. (2016a). This loss function is useful when some (or all) of the gold annotations are not in the output of the model space during training. This situation occurs while training the concept and relation identification models, and using infinite ramp loss improves the results substantially.

3.1 Structured Prediction Models

In many structured prediction models,[1] the predicted output is the highest scoring output under a global scoring function. This is true for the four structured prediction models used in this thesis: the concept (§4.3) and relation (§4.4) identification stages of the parser, and the decoder (§5.3.2) and synthetic rule model (§5.4.2) of the generator. Let x be the input (from the space of possible inputs $\mathcal{X}$), $\hat{y}(x)$ be the output, $\mathcal{Y}(x)$ be the output space (which can depend on the input x), and the parameters of the scoring function be the vector θ. The output is of all these models can be expressed as:

$$\hat{y}_\theta(x) = \arg\max_{y \in \mathcal{Y}(x)} score_\theta(x, y) \tag{3.1}$$

In our models the scoring function is a linear model with parameter vector θ and feature vector $\mathbf{f}(x, y)$:

$$score_\theta(x, y) = \theta \cdot \mathbf{f}(x, y)$$

The feature vector is a sum of local features of the model. A local feature vector $\mathbf{f}_i$ is computed for each of the **parts** i of the output.[2] The feature vector $\mathbf{f}$ is a sum over the

[1]There are transition-based or greedy methods for producing structured objects whose output is based on a series of local decisions, but we do not discuss them here.

[2]Parts are just pieces of the output for which a feature vector and score are computed, and these parts can be overlapping.

local features for each part:

$$\mathbf{f}(x, y) = \sum_{i \in parts(y)} \mathbf{f}_i(x, y)$$

Depending on the parts chosen and the output space, a search algorithm (a **decoding algorithm**) is used to find the exact or approximate argmax in Eq. 3.1. Sometimes the parts chosen and the output space will make finding this argmax NP-hard, so an approximate decoding method must be used. The decoding algorithms used in this thesis are: dynamic programming (§4.3), beam search (§5.3.2), a specialized graph algorithm combined with Lagrangian Relaxation (§4.4), and brute-force search combined with dynamic programming (§5.4.2).

3.2 Loss Functions

Once the parts, the local features, and the decoding algorithm have been decided upon for the model we are using, a method for learning the parameters must selected. Learning the parameters is usually accomplished by minimizing a loss function, so a learning algorithm is usually a loss function and with a particular minimization algorithm. A **loss function**, $L(\mathcal{D}, \theta)$, is a function of the training data and the parameters that is minimized with an optional regularizer to learn the parameters. Let $\hat{\theta}$ be the learned parameters. With an L_2 regularizer, the learned parameters are:[3]

$$\hat{\theta} = \arg\min_{\theta} L(\mathcal{D}, \theta) + \lambda ||\theta||^2 \tag{3.2}$$

3.2.1 Minimizing the Task Loss

A straightforward approach to learning would be to directly minimize 0/1 prediction error or maximize some other metric of performance on the training set. Let $cost(x, y, \hat{y})$

[3]If the loss function is invariant to scaling of θ, then the L_2 regularizer strength λ should be set to zero or a regularizer not invariant to scaling of θ should be used.

be the task-specific error (task specific loss or **cost**) for predicting $\hat{y}$ when the input is x and the gold-standard output is y (lower cost is better). Then minimizing the cost on the training data amounts to using following loss function without a regularizer:

$$L_{\text{cost}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} cost\Big(x_i, y_i, \arg\max_{y \in \mathcal{Y}(x_i)} \theta \cdot \mathbf{f}(x_i, y)\Big) \tag{3.3}$$

3.2.2 Minimum Error Rate Training

Unfortunately, minimizing Eq. 3.3 can be NP-hard, and is NP-hard in the simple case of the task-specific cost being 0/1 prediction error. However, an approximate minimization algorithm to minimize Eq. 3.3 can be used, such as Minimum Error Rate Training (MERT, Och, 2003). MERT was developed for tuning the weights in statistical machine translation systems and can be used to approximately minimize Eq. 3.3 with an arbitrary cost function and a small (less than 20) set of features in a linear model. We use MERT to maximize a task specific metric (BLEU score, Papineni et al., 2002) when training the generator (§5.3.3).

3.2.3 SVM Loss

Alternatively, one can minimize a loss function that approximates the task loss Eq. 3.3 but is easier to minimize. Perhaps the easiest loss functions to minimize are convex approximations to Eq. 3.3. The tightest convex upper bound to Eq. 3.3 is the **SVM loss** function (Taskar et al., 2003, Tsochantaridis et al., 2004):

$$L_{\text{SVM}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} \Big(-\theta \cdot \mathbf{f}(x_i, y_i) + \max_{y \in \mathcal{Y}(x_i)} \big(\theta \cdot \mathbf{f}(x_i, y) + cost(x_i, y_i, y)\big)\Big) \tag{3.4}$$

3.2.4 Perceptron Loss

Another convex approximation to Eq. 3.3 is the **Perceptron loss** function (Rosenblatt, 1957, Collins, 2002), which is not an upper bound to Eq. 3.3. Instead, the Perceptron loss function is motivated by the fact that if the training data can be perfectly classified, that is there exists a θ such that

$$\arg\max_{y \in \mathcal{Y}(x_i)} \theta \cdot \mathbf{f}(x_i, y) = y_i \qquad \forall \quad (x_i, y_i) \in \mathcal{D},$$

then minimizing Eq. 3.3 is equivalent to minimizing the Perceptron loss. The Perceptron loss is:

$$L_{\text{Perceptron}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} \Big(-\theta \cdot \mathbf{f}(x_i, y_i) + \max_{y \in \mathcal{Y}(x_i)} \theta \cdot \mathbf{f}(x_i, y) \Big) \tag{3.5}$$

More precisely, if the training data can be perfectly classified, the minimum (or minimums) of Eq. 3.3 coincide with the minimum (or minimums) of Eq. 3.5 . We use this loss function in the synthetic rule model of the generator (§5.4.2), and in previous versions of the parser (Flanigan et al., 2014).

3.2.5 Conditional Negative Log-Likelihood

A third convex approximation to Eq. 3.3 is **conditional negative log-likelihood** (**CNLL**). If the cost function is $0/1$, that is $cost(x, y, y') = I[y = y']$, then CNLL is a convex upper bound to Eq. 3.3:

$$L_{\text{CNLL}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} \Big(-\theta \cdot \mathbf{f}(x_i, y_i) + \sum_{y \in \mathcal{Y}(x_i)} \exp\big(\theta \cdot \mathbf{f}(x_i, y)\big) \Big) \tag{3.6}$$

This loss function is the loss function underlying binary and multi-class logistic regression and conditional random fields (CRFs Lafferty et al., 2001), but is not used in this thesis. One advantage of CNLL is that the model score can be used to obtain probabilities.

3.2.6 Risk

There are also non-convex approximations to Eq. 3.3. One of the more common is risk (Smith and Eisner, 2006, Gimpel and Smith, 2012, inter alia). **Risk** is the expected value of the cost of the training data under the model, with the model viewed as a probability distribution:

$$L_{\text{risk}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} \frac{\sum_{y \in \mathcal{Y}(x_i)} cost(x_i, y_i, y) e^{\theta \cdot \mathbf{f}(x_i, y)}}{\sum_{y \in \mathcal{Y}(x_i)} e^{\theta \cdot \mathbf{f}(x_i, y)}} \tag{3.7}$$

Although it is non-convex, risk is differentiable and can be optimized to a local optimium using a gradient-based optimizer. Risk has the nice property that $L_{\text{risk}}(\mathcal{D}, \theta) \to L_{\text{cost}}(\mathcal{D}, \theta)$ as $\|\theta\| \to \infty$. However, risk is often unattractive for structured prediction because the numerator in Eq. 3.7 cannot be computed efficiently, and n-best lists are usually used as an approximation to the full sum.

3.2.7 Ramp Loss

A non-convex approximation to Eq. 3.3 that can often be computed exactly is the family of **ramp losses** (Do et al., 2009, Keshet and McAllester, 2011, Gimpel and Smith, 2012):

$$\begin{aligned} L_{\text{ramp}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} \Big(&- \max_{y \in \mathcal{Y}(x_i)} \big(\theta \cdot \mathbf{f}(x_i, y) - \alpha \cdot cost(x_i, y_i, y)\big) \\ &+ \max_{y \in \mathcal{Y}(x_i)} \big(\theta \cdot \mathbf{f}(x_i, y) + \beta \cdot cost(x_i, y_i, y)\big)\Big) \end{aligned} \tag{3.8}$$

α and β are two parameters that control the position and height of the ramp. The ramp height is $\alpha + \beta$, and should be greater than 0. It is typical to set $\alpha = 0$ and $\beta = 1$ (Do et al., 2009, Keshet and McAllester, 2011), but the three combinations $(\alpha, \beta) = (0, 1), (1, 0),$ and $(1, 1)$ have also been advocated (Gimpel and Smith, 2012). $L_{\text{ramp}}(\mathcal{D}, \theta)$ approaches $(\alpha + \beta) L_{\text{cost}}(\mathcal{D}, \theta)$ as $\|\theta\| \to \infty$, so it has a similar appeal that L_{risk} does in that it closely approximates L_{cost}. L_{ramp} is continuous and piecewise differentiable, and can be optimized to a local optimum using a gradient-based solver.

3.2.8 Infinite Ramp Loss

Sometimes in structured prediction problems, features for some training examples cannot be computed (**uncomputable features**), or y_i is not contained in $\mathcal{Y}(x_i)$ for some training examples (**unreachable examples**). Both of these occur, for example, in parsing or machine translation, if a grammar is used and the grammar cannot produce a training example.

In AMR parsing, uncomputable features occur during training for both concept identification and relation identification because the automatic aligner (§2.6) is not able to align all concepts, so some nodes are left unaligned. Both concept and relation identification use the alignment to compute features, so features for some nodes and edges cannot be computed. In the past, we just removed these nodes and edges from the training graphs, but this leads to unreachable examples for relation identification and suboptimal results for both concept and relation identification. This motivated us to use loss functions that could handle uncomputable features and unreachable examples.

Ramp losses and risk have the important property that they can be used even if there are uncomputable features and unreachable examples. This is because the training examples are only used in the cost function, and are not plugged directly into the feature vector $\mathbf{f}(x_i, y_i)$ like in loss functions Eqs. 3.4 - 3.6. And unlike Eqs. 3.4 - 3.6, which become unbounded from below and ill-defined as loss functions because they have no minimum if there are unreachable examples, ramp loss and risk are always bounded from below.

However, one drawback of ramp loss and risk are they can be difficult to optimize due to flat spots in the loss function – places where the derivative of the loss with respect to θ becomes zero. This can occur in ramp loss because terms in the sum become a constant when the margin for an example becomes too negative. In this case, the model score overpowers the cost function in the maxes, and both maxes have the same $\arg\max$

and derivative, which cancel. In risk, the softmax in Eq. 3.7 becomes flat at large model weights.

We wondered, *is there a loss function that does not suffer from vanishing derivatives at large model weights, and still allows for uncomputable features and unreachable examples?* It turns out, there is such a generalization of SVM loss to this case. We call it the infinite ramp loss.

Infinite ramp loss (Flanigan et al., 2016a) is obtained roughly by taking α to infinity in L_{ramp}. In practice however, we just set α to a large number (10^{12} in our experiments). To make the limit $\alpha \to \infty$ well-defined, we re-define the cost function before taking the limit by shifting it by a constant so minimum of the cost function is zero:

$$\widehat{cost}(x_i, y_i, y) = cost(x_i, y_i, y) - \min_{y' \in \mathcal{Y}(x_i)} cost(x_i, y_i, y') \tag{3.9}$$

This shift by a constant is not necessary if one is just setting α to large number. The infinite ramp loss is thus defined as:

$$\begin{aligned} L_{\infty-\text{ramp}}(\mathcal{D}, \theta) = \sum_{(x_i, y_i) \in \mathcal{D}} \Big(& - \lim_{\alpha \to \infty} \max_{y \in \mathcal{Y}(x_i)} \big(\theta \cdot \mathbf{f}(x_i, y) - \alpha \cdot \widehat{cost}(x_i, y_i, y)\big) \\ & + \max_{y \in \mathcal{Y}(x_i)} \big(\theta \cdot \mathbf{f}(x_i, y) + cost(x_i, y_i, y)\big)\Big) \end{aligned} \tag{3.10}$$

An intuitive interpretation of Eq. 3.10 is as follows: if minimizing the cost function is unique (that is the argmin over $y \in \mathcal{Y}(x_i)$ of $cost(x_i, y_i, y)$ is unique), then Eq. 3.10 is equivalent to:

$$\begin{aligned} \sum_{(x_i, y_i) \in \mathcal{D}} \Bigg(& - \theta \cdot \mathbf{f}\Big(x_i, \arg\min_{y' \in \mathcal{Y}(x_i)} cost(x_i, y_i, y')\Big) \\ & + \max_{y \in \mathcal{Y}(x_i)} \Big(\theta \cdot \mathbf{f}(x_i, y) + cost(x_i, y_i, y)\Big)\Bigg) \end{aligned} \tag{3.11}$$

Note that Eq. 3.11 is similar to the SVM loss (Eq. 3.4), but with y_i replaced with $\arg\min_{y' \in \mathcal{Y}(x_i)} cost(x_i, y_i, y')$. If the argmin of the cost function is not unique, then Eq. 3.10 breaks ties in $\arg\min_{y' \in \mathcal{Y}(x_i)} cost(x_i, y_i, y')$ using the model score. Infinite ramp loss turns out to be a generalization of SVM loss and the latent SVM (Yu and Joachims, 2009).

Infinite ramp loss generalizes the structured SVM loss. If y_i is reachable and the minimum over $y \in \mathcal{Y}(x_i)$ of $cost(x_i, y_i, y)$ is unique and occurs when $y = y_i$, then the first max in Eq. 3.10 picks out $y = y_i$ and Eq. 3.10 reduces to the structured SVM loss.

The infinite ramp is also a generalization of the Latent Structured SVM (LSVM) (Yu and Joachims, 2009), which is a generalization of the structured SVM for hidden variables. LSVM loss can be used when the output can be written $y_i = (\tilde{y}_i, h_i)$, where $\tilde{y}_i$ is observed output and h_i is latent (even at training time). Let $\tilde{\mathcal{Y}}(x_i)$ be the space of all possible observed outputs and $\mathcal{H}(x_i)$ be the hidden space for the example x_i. Let $\tilde{c}$ be the cost function for the observed output. The **Latent Structured SVM loss** is:

$$\begin{aligned} L_{\text{LSVM}}(x_i, y_i; \mathbf{w}) = &- \max_{h \in \mathcal{H}(x_i)} \big(\mathbf{w} \cdot \mathbf{f}(x_i, \tilde{y}_i, h)\big) \\ &+ \max_{\tilde{y} \in \tilde{\mathcal{Y}}(x_i)} \max_{h' \in \mathcal{H}(x_i)} \big(\mathbf{w} \cdot \mathbf{f}(x_i, \tilde{y}, h') + \tilde{c}(x_i, \tilde{y}_i, \tilde{y})\big) \end{aligned} \tag{3.12}$$

If we set $cost(x_i, y_i, y) = \tilde{c}(x_i, \tilde{y}_i, \tilde{y})$ in Eq. 3.10, and the minimum of $\tilde{c}(x_i, \tilde{y}_i, \tilde{y})$ occurs when $\tilde{y} = \tilde{y}_i$, then minimizing Eq. 3.10 is equivalent to minimizing Eq. 3.12.

We use the infinite ramp loss in training both the concept and relation identification stages of the parser. These stages suffer from unreachable examples (features of the gold standard that cannot be computed) because of unaligned nodes in the gold standard due to an imperfect aligner.

3.3 Minimizing the Loss

There are many procedures for minimizing the loss function of the training data, each with its own advantages and drawbacks. In principle, any minimization procedure can be used. Depending on the properties of the loss function minimized (such as convexity, non-convexity, strong-convexity, or lipchitz continuity, to name a few) different minimization procedures can have different theoretical guarantees and performance in practice, of both their ability to minimize the loss function and on the generalization performance of the learned parameters. It is beyond the scope of this thesis to discuss all the various procedures, but the reader is encouraged to consult the references . Here we will simply present the optimizer (AdaGrad) we have used throughout this thesis.

AdaGrad (Duchi et al., 2011) is an online algorithm for minimizing a loss function which is a sum over training examples. Similar to stochastic gradient descent, the parameters are updated using noisy gradients obtained from single training examples. The time step parameter t starts at one and increments by one after processing an example, and the algorithm makes many passes through the training data without re-setting this parameter. For each pass through the training data, training examples are processed one at a time in a random order. At time step t, the gradient s^t of the loss function for example being processed is computed and the parameters are then updated before going on to the next example. Each component i of the parameter vector is updated like so:

$$\theta_i^{t+1} = \theta_i^t - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} s_i^{t'}}} s_i^t$$

η is the learning rate, which we set to 1 in all our experiments.

To prevent overfitting we use either early stopping, a regularizer, or a combination of both. If we are using early stopping, we run AdaGrad for a set number of iterations and use the weights from the iteration that gives the highest F_1 on a development set.

Chapter 4

Parsing

Parsing is the annotation of natural language with a linguistic structure, usually a tree or graph structure. In **semantic parsing** the annotated structure is a semantic structure. In this chapter we consider **AMR parsing**, which is semantic parsing where the semantic structure is an AMR graph.

Natural language understanding (NLU) is a major goal of NLP. Whether or not an NLP program has an explicit human and machine-readable representation of meaning, an NLU application should understand semantics of natural language at the level required for performing its task. Semantic parsing makes explicit the meaning of natural language in a semantic representation that is unambiguous in the semantics it represents, and can be used in later processing steps in an NLU application.

Most early NLP systems analysed natural language into a semantic representation using a set of hand-developed rules with no machine learning involved (Darlington and Charney, 1963, Wilks, 1973, Woods, 1978, Alshawi, 1992, Copestake and Flickinger, 2000). Learning semantic parsers with supervised machine learning started on limited domains (Miller et al., 1994, Zelle and Mooney, 1996). Supervised learning of broad-coverage semantic parsing was initiated by Gildea and Jurafsky (2000), advocating construction of shallow semantic parsers (parsers that do not produce a full logical form), and much work was done in the task of semantic role labeling (SRL) and the related task of semantic dependency parsing (SDP).

The introduction of a semantic representation such as AMR with handling of many semantic phenomenon and a sizable annotated corpus in Baneresceu et al (2013) opened the doors to learning a broad coverage deep semantic parser from hand-annotated data. Although AMR does not have quantifier scoping as in a fully specified logical form, it is significantly deeper than previous broad-coverage approaches.

AMR parsing is uniquely challenging compared to other kinds of parsing, such as syntactic parsing and semantic role labeling (SRL). First, the produced structures are graphs, rather than trees as in syntactic parsing or labeled spans of text as in SRL. Second, the nodes in the AMR graphs are concepts that need to be predicted, whereas syntactic parsing produces a tree over the words in the sentence. This motivates the development of new parsing machinery for AMR parsing, rather than adapting existing parsers to the task.

We present the first parser developed for AMR, as well as some improvements to make the approach near state-of-the-art. This work is based on previously published papers (Flanigan et al., 2014, Flanigan et al., 2016a).

Nowadays NLP systems are sometimes designed to perform a NLU or NLG task end-to-end with deep learning, where there is no human-readable semantic representation. In this case, the intermediate vector representations are essentially machine-learned semantic representations that are not designed by humans. This makes one question whether we need human-readable semantic representations at all, and whether researchers should continue to work on semantic parsing.

There are various reasons to continue to work on semantic parsing alongside end-to-end deep learning methods: 1) The semantic parsing task challenges researchers to produce systems that can understand the semantics that is represented in our semantic representations, serving as a testbed and metric of progress in NLU. 2) Although end-to-end deep learning can be used for many tasks, they sometimes require a lot of training data. An approach that leverages a semantic representation in addition to deep learning may perform better overall or perform better when there is less task data. 3) A variety of approaches is always best for research so the field as a whole doesn't get stuck in a local minimum.

4.1 Method Overview and Outline

We solve the AMR parsing problem with a pipeline that first predicts concepts (§4.3) and then relations (§4.4). The pipeline for an example sentence is shown in Figure 4.1. This approach largely follows Flanigan et al. (2014), with improvements to parameter learning (§4.5) from Flanigan et al. (2016a) where we apply the infinite ramp loss introduced in §3.2.8. This loss function is used for boosting concept fragment recall and obtaining more re-entrancies in the AMR graphs. The parser, called JAMR, is released online.[1]

4.2 Notation and Overview

Our approach to AMR parsing represents an AMR parse as a graph $G = \langle N, E \rangle$; nodes and edges are given labels from sets L_N and L_E, respectively. G is constructed in two stages. The first stage identifies the **concepts** evoked by words and phrases in an input sentence $\boldsymbol{w} = \langle w_1, \ldots, w_n \rangle$, each w_i a member of vocabulary W. The second stage connects the concepts by adding L_E-labeled edges capturing the **relations** between concepts, and selects a root in G corresponding to the **focus** of the sentence $\boldsymbol{w}$.

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

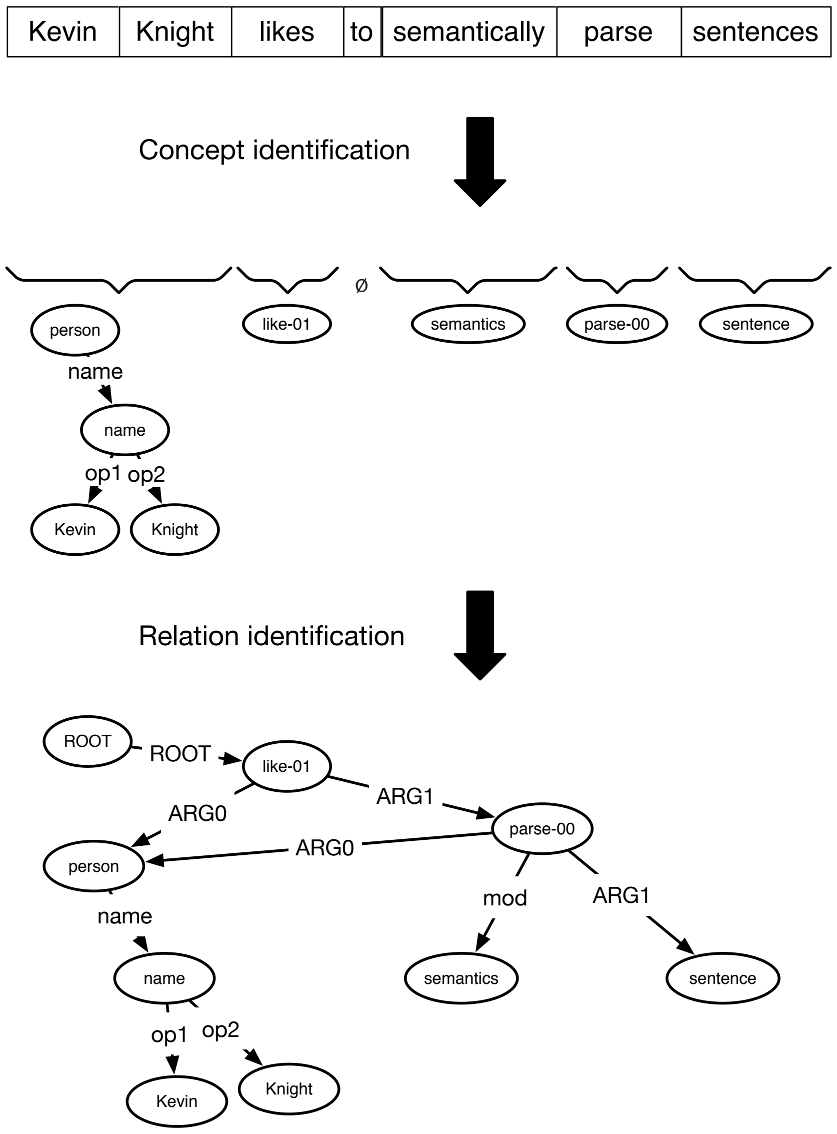
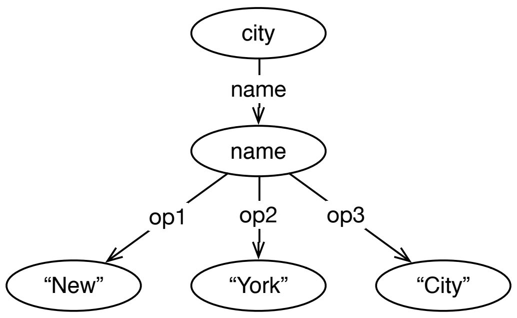


Figure 4.1: Stages in the parsing pipeline: concept identification followed by relation identification.

Concept identification (§4.3) involves segmenting $\boldsymbol{w}$ into contiguous spans and assigning to each span a graph fragment corresponding to a concept from a concept set denoted F (or to $\emptyset$ for words that evoke no concept). In §2.6 we describe how F is constructed. In our formulation, spans are contiguous subsequences of $\boldsymbol{w}$. For example, the words "New York City" can evoke the graph fragment (see next page):



We use a sequence labeling algorithm to identify concepts.

The relation identification stage (§4.4) is similar to a graph-based dependency parser. Instead of finding the maximum-scoring tree over words, it finds the maximum-scoring connected subgraph that preserves concept fragments from the first stage, links each pair of nodes by at most one edge, and is deterministic[2] with respect to a special set of edge labels $L_E^* \subset L_E$. The set L_E^* consists of the labels ARG0–ARG5, and does not include labels such as MOD or MANNER, for example. Linguistically, the determinism constraint enforces that predicates have at most one semantic argument of each type; this is discussed in more detail in §4.4.

To train the parser, spans of words must be labeled with the concept fragments they evoke. Although AMR Bank does not label concepts with the words that evoke them, it is possible to build an automatic aligner (§2.6). The alignments are used to construct the concept lexicon and to train the concept identification and relation identification stages of the parser (§4.5). Each stage is a discriminatively-trained linear structured predictor with rich features that make use of part-of-speech tagging, named entity tagging, and dependency parsing.

In §4.6, we evaluate the parser against gold-standard annotated sentences from the AMR Bank corpus (Banarescu et al., 2013) using the Smatch score (Cai and Knight, 2013), presenting the first published results on automatic AMR parsing.

[2]By this we mean that, at each node, there is at most one outgoing edge with that label type.

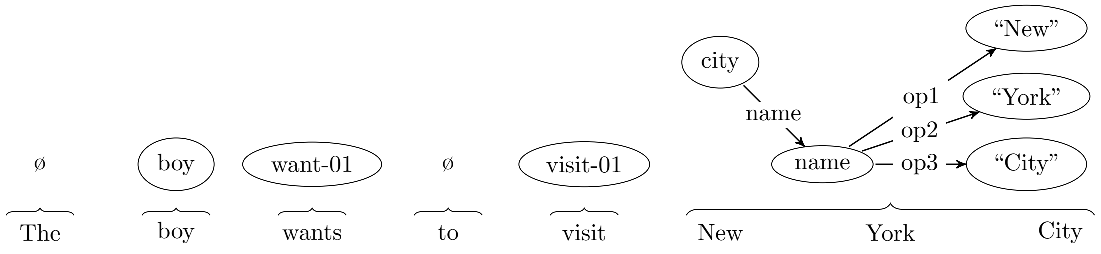


Figure 4.2: A concept labeling for the sentence "The boy wants to visit New York City."

4.3 Concept Identification

The concept identification stage maps spans of words in the input sentence $\boldsymbol{w}$ to concept graph fragments from F, or to the empty graph fragment $\emptyset$. These graph fragments often consist of just one labeled concept node, but in some cases they are larger graphs with multiple nodes and edges.[3] Concept identification is illustrated in Figure 4.2 using our running example, "The boy wants to visit New York City."

Let the concept lexicon be a mapping $clex : W^* \to 2^F$ that provides candidate graph fragments for sequences of words. (The construction of F and $clex$ is discussed in §4.3.1.) Formally, a concept labeling is:

1. a segmentation of $\boldsymbol{w}$ into contiguous spans represented by boundaries $\boldsymbol{b}$, giving spans $\langle \boldsymbol{w}_{b_0:b_1}, \boldsymbol{w}_{b_1:b_2}, \ldots \boldsymbol{w}_{b_{k-1}:b_k} \rangle$, with $b_0 = 0$ and $b_k = n$, and
2. an assignment of each phrase $\boldsymbol{w}_{b_{i-1}:b_i}$ to a concept graph fragment $c_i \in clex(\boldsymbol{w}_{b_{i-1}:b_i}) \cup \{\emptyset\}$. The sequence of c_i's is denoted $\boldsymbol{c}$.

The sequence of spans $\boldsymbol{b}$ and the sequence of concept graph fragments $\boldsymbol{c}$, both of arbitrary length k, are scored using the following locally decomposed, linearly parameterized function:

$$score(\boldsymbol{b}, \boldsymbol{c}; \boldsymbol{\theta}) = \sum_{i=1}^{k} \boldsymbol{\theta}^\top \mathbf{f}(\boldsymbol{w}, b_{i-1}, b_i, c_i) \tag{4.1}$$

The vector $\mathbf{f}$ is a feature vector representation of a span and one of its concept graph fragments in context. These features are discussed below.

[3]About 20% of invoked concept fragments are multi-concept fragments.

We find the highest-scoring $\boldsymbol{b}$ and $\boldsymbol{c}$ using a dynamic programming algorithm: the zeroth-order case of inference under a semi-Markov model (Janssen and Limnios, 1999). Let $S(i)$ denote the score of the best labeling of the first i words of the sentence, $\boldsymbol{w}_{0:i}$; it can be calculated using the recurrence:

$$
\begin{aligned}
S(0) &= 0 \\
S(i) &= \max_{\substack{j:0\leq j<i, \\ c\in clex(\boldsymbol{w}_{j:i})\cup\emptyset}} \left\{ S(j) + \boldsymbol{\theta}^\top \mathbf{f}(\boldsymbol{w}, j, i, c) \right\}
\end{aligned}
$$

The best score will be $S(n)$, and the best scoring concept labeling can be recovered using back-pointers, as in typical implementations of the Viterbi algorithm. Runtime is $O(n^2)$.

The features $\mathbf{f}(\boldsymbol{w}, j, i, c)$ are:

- **Fragment given words and words given fragment**: Relative frequency estimates of the probability of a concept fragment given the sequence of words in the span, and the sequence of words given the concept fragment. These are calculated from the concept-word alignments in the training corpus (§2.6).
- **Length** of the matching span (number of tokens).
- **Bias**: 1 for any concept graph fragment from F and 0 for $\emptyset$.
- **First match**: 1 if this is the first place in the sentence that matches the span.
- **Number**: 1 if the span is length 1 and matches the regular expression "[0-9]+".
- **Short concept**: 1 if the length of the concept fragment string is less than 3 and contains only upper or lowercase letters.
- **Sentence match**: 1 if the span matches the entire input sentence.
- **; list**: 1 if the span consists of the single word ";" and the input sentence is a ";" separated list.
- **POS**: the sequence of POS tags in the span.
- **POS and event**: same as above but with an indicator if the concept fragment is an

event concept (matches the regex ".*-[0-9][0-9]").

- **Span**: the sequence of words in the span if the words have occurred more than 10 times in the training data as a phrase with no gaps.
- **Span and concept**: same as above concatenated with the concept fragment in PENMAN notation.
- **Span and concept with POS**: same as above concatenated with the sequence of POS tags in the span.
- **Concept fragment source**: indicator for the source of the concept fragment (corpus, NER tagger, date expression, frame files, lemma, verb-pass through, or NE pass-through).
- **No match from corpus**: 1 if there is no matching concept fragment for this span in the rules extracted from the corpus.

4.3.1 Candidate Concepts

The functon $clex$ provides candidate concept fragments for spans of words in the input sentence. It returns the union of candidate concepts from these seven sources:

1. **Training data lexicon**: if the span matches a word sequence in the training data that was labeled one or more times with a concept fragment (using automatic alignments), return the set of concept fragments it was labeled with (a set of phrase-concept fragment pairs).
2. **Named entity**: if the span is recognized as a named entity by the named entity tagger, return a candidate concept fragment for the named entity.
3. **Time expression**: if the span matches a regular expression for common time expressions, return a candidate concept fragment for the time expression.
4. **Frame file lookup**: if the span is a single word, and the lemma of the word matches

the name of a frame in the AMR frame files (with sense tag removed), return the lemma concatenated with "-01" as a candidate concept fragment consisting of one node.

5. **Lemma**: if the span is a single word, return the lemma of the word as a candidate concept fragment consisting of one node.

6. **Verb pass-through**: if the span is a single word, and the word is a tagged as a verb by the POS tagger, return the lemma concatenated with "-00" as a candidate concept fragment consisting of one node.

7. **Named entity pass-through**: if the span length is between 1 and 7, return the concept fragment "(thing :name (name :op1 word1 . . . :opn word$n$)" as a candidate concept fragment, where n is the length of the span, and "word1" and "word$n$" are the first and last words in the fragment.

The sources 2-7 complicate concept identification training. These sources improve concept coverage on held-out data but they do not improve coverage on the training data, since one of the concept sources is a lexicon extracted from the training data. Thus correctly balancing use of the training data lexicon versus the additional sources to prevent overfitting is a challenge.

To balance the training data lexicon with the other sources, we use a variant of cross-validation. During training, when processing a training example in the training data, we exclude concept fragments extracted from the same section of the training data. This is accomplished by keeping track of the training instances each phrase-concept fragment pair was extracted from, and excluding all phrase-concept fragment pairs within a window of the current training instance. In our experiments the window is set to 20.

While excluding phrase-concept fragment pairs allows the learning algorithm to balance the use of the training data lexicon versus the other concept sources, it creates another problem: some of the gold standard training instances may be unreachable (cannot be produced), because of the phrase-concept pair need to produce the example has been excluded. This causes problems during learning. To handle this, we use infinite ramp loss, as described in the training section §4.5.

4.4 Relation Identification

The relation identification stage adds edges among the concept subgraph fragments identified in the first stage (§4.3), creating a graph. We frame the task as a constrained combinatorial optimization problem.

Consider the fully dense labeled multigraph $D = \langle N_D, E_D \rangle$ that includes the union of all labeled nodes and labeled edges in the concept graph fragments, as well as every possible labeled edge $n_1 \xrightarrow{\ell} n_2$, for all $n_1, n_2 \in N_D$ and every $\ell \in L_E$.[4]

We require a subgraph $G = \langle N_G, E_G \rangle$ that respects the following constraints:

1. **Preserving**: all graph fragments (including labels) from the concept identification phase are subgraphs of G.
2. **Simple**: for any two nodes n_1 and $n_2 \in N_G$, E_G includes at most one edge between n_1 and n_2. This constraint forbids a small number of perfectly valid graphs, for example for sentences such as "John hurt himself"; however, we see that $< 1\%$ of training instances violate the constraint. We found in preliminary experiments that including the constraint increases overall performance.[5]
3. **Connected**: G must be weakly connected (every node reachable from every other node, ignoring the direction of edges). This constraint follows from the formal definition of AMR and is never violated in the training data.

[4]To handle numbered OP labels, we pre-process the training data to convert OPN to OP, and post-process the output by numbering the OP labels sequentially.

[5]In future work it might be treated as a soft constraint, or the constraint might be refined to specific cases.

4. **Deterministic**: For each node $n \in N_G$, and for each label $\ell \in L_E^*$, there is at most one outgoing edge in E_G from n with label ℓ. As discussed in §4.2, this constraint is linguistically motivated.

One constraint we do not include is **acyclicity**, which follows from the definition of AMR. In practice, graphs with cycles are rarely produced by the parser. In fact, none of the graphs produced on the test set violate acyclicity.

Given the constraints, we seek the maximum-scoring subgraph. We define the score to decompose by edges, and with a linear parameterization:

$$score(E_G; \boldsymbol{\psi}) = \textstyle\sum_{e \in E_G} \boldsymbol{\psi}^\top \mathbf{g}(e) \tag{4.2}$$

The features are shown in Table 4.1.

Our solution to maximizing the score in Eq. 4.2, subject to the constraints, makes use of (i) an algorithm that ignores constraint 4 but respects the others (§4.4.1); and (ii) a Lagrangian relaxation that iteratively adjusts the edge scores supplied to (i) so as to enforce constraint 4 (§4.4.2).

4.4.1 Maximum Preserving, Simple, Spanning, Connected Subgraph Algorithm

The steps for constructing a maximum preserving, simple, spanning, connected (but not necessarily deterministic) subgraph are as follows. These steps ensure the resulting graph G satisfies the constraints: the initialization step ensures the preserving constraint is satisfied, the pre-processing step ensures the graph is simple, and the core algorithm ensures the graph is connected.

1. (Initialization) Let $E^{(0)}$ be the union of the concept graph fragments' weighted, labeled, directed edges. Let N denote its set of nodes. Note that $\langle N, E^{(0)} \rangle$ is preserving (constraint 1), as is any graph that contains it. It is also simple (constraint 2),

Name	**Description**
Label	For each $\ell \in L_E$, 1 if the edge has that label
Self edge	1 if the edge is between two nodes in the same fragment
Tail fragment root	1 if the edge's tail is the root of its graph fragment
Head fragment root	1 if the edge's head is the root of its graph fragment
Path	Dependency edge labels and parts of speech on the shortest syntactic path between any two words in the two spans
Distance	Number of tokens (plus one) between the two concepts' spans (zero if the same)
Distance indicators	A feature for each distance value, that is 1 if the spans are of that distance
Log distance	Logarithm of the distance feature plus one.
Bias	1 for any edge.

Table 4.1: Features used in relation identification. In addition to the features above, the following conjunctions are used (Tail and Head concepts are elements of L_N): Tail concept $\wedge$ Label, Head concept $\wedge$ Label, Path $\wedge$ Label, Path $\wedge$ Head concept, Path $\wedge$ Tail concept, Path $\wedge$ Head concept $\wedge$ Label, Path $\wedge$ Tail concept $\wedge$ Label, Path $\wedge$ Head word, Path $\wedge$ Tail word, Path $\wedge$ Head word $\wedge$ Label, Path $\wedge$ Tail word $\wedge$ Label, Distance $\wedge$ Label, Distance $\wedge$ Path, and Distance $\wedge$ Path $\wedge$ Label. To conjoin the distance feature with anything else, we multiply by the distance.

assuming each concept graph fragment is simple.

2. (Pre-processing) We form the edge set E by including just one edge from E_D between each pair of nodes:

   - For any edge $e = n_1 \xrightarrow{\ell} n_2$ in $E^{(0)}$, include e in E, omitting all other edges between n_1 and n_2.
   - For any two nodes n_1 and n_2, include only the highest scoring edge between n_1 and n_2.

   Note that without the deterministic constraint, we have no constraints that depend on the label of an edge, nor its direction. So it is clear that the edges omitted in this step could not be part of the maximum-scoring solution, as they could be replaced by a higher scoring edge without violating any constraints.

   Note also that because we have kept exactly one edge between every pair of nodes, $\langle N, E\rangle$ is simple and connected.

3. (Core algorithm) Run Algorithm 1, MSCG, on $\langle N, E\rangle$ and $E^{(0)}$. This algorithm is a (to our knowledge novel) modification of the minimum spanning tree algorithm of Kruskal (1956). Note that the directions of edges do not matter for MSCG.

Steps 1–2 can be accomplished in one pass through the edges, with runtime $O(|N|^2)$. MSCG can be implemented efficiently in $O(|N|^2 \log |N|)$ time, similarly to Kruskal's algorithm, using a disjoint-set data structure to keep track of connected components.[6] The total asymptotic runtime complexity is $O(|N|^2 \log |N|)$.

The details of MSCG are given in Algorithm 1. In a nutshell, MSCG first adds all positive edges to the graph, and then connects the graph by greedily adding the least negative edge that connects two previously unconnected components.

Theorem 1. MSCG *finds a maximum spanning, connected subgraph of* $\langle N, E\rangle$

[6]For dense graphs, Prim's algorithm (Prim, 1957) is asymptotically faster ($O(|N|^2)$). We conjecture that using Prim's algorithm instead of Kruskall's to connect the graph could improve the runtime of MSCG.

```
**input :** weighted, connected graph $\langle N, E\rangle$ and set of edges $E^{(0)} \subseteq E$ to be preserved
**output:** maximum spanning, connected subgraph of $\langle N, E\rangle$ that preserves $E^{(0)}$
let $E^{(1)} = E^{(0)} \cup \{e \in E \mid \boldsymbol{\psi}^\top \mathbf{g}(e) > 0\}$;
create a priority queue $Q$ containing $\{e \in E \mid \boldsymbol{\psi}^\top \mathbf{g}(e) \leq 0\}$ prioritized by scores;
$i = 1$;
**while** *$Q$ nonempty and $\langle N, E^{(i)}\rangle$ is not yet spanning and connected* **do**
    $i = i + 1$;
    $E^{(i)} = E^{(i-1)}$;
    $e = \arg\max_{e' \in Q} \boldsymbol{\psi}^\top \mathbf{g}(e')$;
    remove $e$ from $Q$;
    **if** *$e$ connects two previously unconnected components of $\langle N, E^{(i)}\rangle$* **then**
        add $e$ to $E^{(i)}$
    **end**
**end**
return $G = \langle N, E^{(i)}\rangle$;
```

Algorithm 1: MSCG algorithm.

Proof. We closely follow the original proof of correctness of Kruskal's algorithm. We first show by induction that, at every iteration of MSCG, there exists some maximum spanning, connected subgraph that contains $G^{(i)} = \langle N, E^{(i)}\rangle$:

Base case: Consider $G^{(1)}$, the subgraph containing $E^{(0)}$ and every positive edge. Take any maximum preserving spanning connected subgraph M of $\langle N, E\rangle$. We know that such an M exists because $\langle N, E\rangle$ itself is a preserving spanning connected subgraph. Adding a positive edge to M would strictly increase M's score without disconnecting M, which would contradict the fact that M is maximal. Thus M must contain $G^{(1)}$.

Induction step: By the inductive hypothesis, there exists some maximum spanning connected subgraph $M = \langle N, E_M \rangle$ that contains $G^{(i)}$.

Let e be the next edge added to $E^{(i)}$ by MSCG.

If e is in E_M, then $E^{(i+1)} = E^{(i)} \cup \{e\} \subseteq E_M$, and the hypothesis still holds.

Otherwise, since M is connected and does not contain e, $E_M \cup \{e\}$ must have a cycle containing e. In addition, that cycle must have some edge e' that is not in $E^{(i)}$. Otherwise, $E^{(i)} \cup \{e\}$ would contain a cycle, and e would not connect two unconnected components of $G^{(i)}$, contradicting the fact that e was chosen by MSCG.

Since e' is in a cycle in $E_M \cup \{e\}$, removing it will not disconnect the subgraph, i.e. $(E_M \cup \{e\}) \setminus \{e'\}$ is still connected and spanning. The score of e is greater than or equal to the score of e', otherwise MSCG would have chosen e' instead of e. Thus, $\langle N, (E_M \cup \{e\}) \setminus \{e'\} \rangle$ is a maximum spanning connected subgraph that contains $E^{(i+1)}$, and the hypothesis still holds.

When the algorithm completes, $G = \langle N, E^{(i)} \rangle$ is a spanning connected subgraph. The maximum spanning connected subgraph M that contains it cannot have a higher score, because G contains every positive edge. Hence G is maximal. $\square$

4.4.2 Lagrangian Relaxation

If the subgraph resulting from MSCG satisfies constraint 4 (deterministic) then we are done. Otherwise we resort to Lagrangian relaxation (LR). Here we describe the technique as it applies to our task, referring the interested reader to Rush and Collins (2012) for a more general introduction to Lagrangian relaxation in the context of structured prediction problems.

We begin by encoding the edge set E_G of a graph $G = \langle N_G, E_G \rangle$, which is a subgraph of a fully dense multigraph $D = \langle N_D, E_D \rangle$, as a binary vector. G contains all the nodes of D due to the preserving contraint (§4.4), so $N_G = N_D$. For each edge e in E_D, we associate a binary variable $z_e = \mathbf{1}\{e \in E_G\}$. Here $\mathbf{1}\{P\}$ is the indicator function, taking value 1 if the proposition P is true, 0 otherwise. The collection of z_e form a vector $\mathbf{z} \in \{0,1\}^{|E_D|}$.

Determinism constraints can be encoded as a set of linear inequalities. For example, the constraint that node n has no more than one outgoing ARG0 can be encoded with the inequality:

$$\sum_{n' \in N} \mathbf{1}\{n \xrightarrow{\text{ARG0}} n' \in E_G\} = \sum_{n' \in N} z_{n \xrightarrow{\text{ARG0}} n'} \leq 1. \tag{4.3}$$

All of the determinism constraints can collectively be encoded as one system of inequalities:

$$\mathbf{A}\mathbf{z} \leq \mathbf{b},$$

with each row $\mathbf{A}_i$ in A and its corresponding entry b_i in $\mathbf{b}$ together encoding one constraint. For example, for the inequality 4.3, we have a row $\mathbf{A}_i$ that has 1s in the columns corresponding to edges outgoing from n with label ARG0 and 0's elsewhere, and a corresponding element $b_i = 1$ in $\mathbf{b}$.

The score of graph G (encoded as $\mathbf{z}$) can be written as the objective function $\boldsymbol{\phi}^\top \mathbf{z}$, where $\phi_e = \boldsymbol{\psi}^\top \mathbf{g}(e)$. To handle the constraint $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, we introduce multipliers $\boldsymbol{\mu} \geq 0$ to get the Lagrangian relaxation of the objective function:

$$L_{\boldsymbol{\mu}}(\mathbf{z}) = \boldsymbol{\phi}^\top \mathbf{z} + \boldsymbol{\mu}^\top (\mathbf{b} - \mathbf{A}\mathbf{z}),$$

$$\mathbf{z}^*_{\boldsymbol{\mu}} = \arg\max_{\mathbf{z}} L_{\boldsymbol{\mu}}(\mathbf{z}).$$

And the solution to the dual problem:

$$\boldsymbol{\mu}^* = \arg\min_{\boldsymbol{\mu} \geq \mathbf{0}} L_{\boldsymbol{\mu}}(\mathbf{z}^*_{\boldsymbol{\mu}})$$

Conveniently, $L_{\boldsymbol{\mu}}(\mathbf{z})$ decomposes over edges, so

$$\begin{aligned}\mathbf{z}^*_{\boldsymbol{\mu}} &= \arg\max\nolimits_{\mathbf{z}} \left(\boldsymbol{\phi}^\top \mathbf{z} + \boldsymbol{\mu}^\top(\mathbf{b} - \mathbf{A}\mathbf{z})\right) \\ &= \arg\max\nolimits_{\mathbf{z}} \left(\boldsymbol{\phi}^\top \mathbf{z} - \boldsymbol{\mu}^\top \mathbf{A}\mathbf{z}\right) \\ &= \arg\max\nolimits_{\mathbf{z}} \left((\boldsymbol{\phi} - \mathbf{A}^\top \boldsymbol{\mu})^\top \mathbf{z}\right).\end{aligned}$$

Thus for any $\boldsymbol{\mu}$, we can find $\mathbf{z}^*_{\mu}$ by assigning edges the new Lagrangian adjusted weights $\boldsymbol{\phi} - \mathbf{A}^\top \boldsymbol{\mu}$ and reapplying the algorithm described in §4.4.1. We can find $\mathbf{z}^* = \mathbf{z}^*_{\boldsymbol{\mu}^*}$ by projected subgradient descent, by starting with $\boldsymbol{\mu} = \mathbf{0}$, and taking steps in the direction:

$$-\frac{\partial L_{\boldsymbol{\mu}}}{\partial \boldsymbol{\mu}}(\mathbf{z}^*_{\boldsymbol{\mu}}) = \mathbf{A}\mathbf{z}^*_{\boldsymbol{\mu}} - \mathbf{b}.$$

If any components of $\boldsymbol{\mu}$ are negative after taking a step, they are set to zero.

$L_{\boldsymbol{\mu}^*}(\mathbf{z}^*)$ is an upper bound on the optimal solution to the primal constrained problem, and is equal to it if and only if the constraints $A\mathbf{z}^* \leq \mathbf{b}$ are satisfied. If $L_{\boldsymbol{\mu}^*}(\mathbf{z}^*) = \boldsymbol{\phi}^\top \mathbf{z}^*$, then $\mathbf{z}^*$ is also the optimal solution to the original constrained problem. Otherwise, there exists a duality gap, and Lagrangian relaxation has failed. In that case we still return the subgraph encoded by $\mathbf{z}^*$, even though it might violate one or more constraints. Techniques from integer programming such as branch-and-bound could be used to find an optimal solution when LR fails (Das et al., 2012), but we do not use these techniques here. In our experiments and data, with a stepsize of 1 and max number of steps as 500, Lagrangian relaxation succeeds 100% of the time at finding optimal solution.

4.4.3 Focus Identification

In AMR, one node must be marked as the focus of the sentence. To do this, we augment the relation identification step: we add a special concept node `root` to the dense graph D, and add an edge from `root` to every other node, giving each of these edges the label

FOCUS. We require that `root` have at most one outgoing FOCUS edge. Our system has two feature types for this edge: the concept it points to, and the shortest dependency path from a word in the span to the root of the dependency tree.

4.5 Training

We now describe how to train the two stages of the parser. The training data for the concept identification stage consists of (X, Y) pairs:

- **Input:** X, a sentence annotated with named entities (person, organization, location, misciscellaneous) from the Illinois Named Entity Tagger (Ratinov and Roth, 2009), and part-of-speech tags and basic dependencies from the Stanford Parser (Klein and Manning, 2003, de Marneffe et al., 2006).
- **Output:** Y, the sentence labeled with concept subgraph fragments.

The training data for the relation identification stage consists of (X, Y) pairs:

- **Input:** X, the sentence labeled with graph fragments, as well as named enties, POS tags, and basic dependencies as in concept identification.
- **Output:** Y, the sentence with a full AMR parse.[7]

Alignments (§2.6) are used to induce the concept labeling for the sentences, so no annotation beyond the automatic alignments is necessary.

We train the parameters of the stages separately using AdaGrad (§3.3, Duchi et al., 2011) with the infinite ramp loss presented in §3.2.8. We give equations for concept identification parameters $\boldsymbol{\theta}$ and features $\mathbf{f}(X, Y)$. For a sentence of length k, and spans

[7]Because the alignments are automatic, some concepts may not be aligned, so we cannot compute their features. We remove the unaligned concepts and their edges from the full AMR graph for training. Thus some graphs used for training may in fact be disconnected.

$\boldsymbol{b}$ labeled with a sequence of concept fragments $\boldsymbol{c}$, the features are:

$$\mathbf{f}(X, Y) = \textstyle\sum_{i=1}^{k} \mathbf{f}(\boldsymbol{w}_{b_{i-1}:b_i}, b_{i-1}, b_i, c_i)$$

To train with AdaGrad, we process examples in the training data $((X^1, Y^1), \ldots, (X^N, Y^N))$ one at a time. At time t, we decode with the current parameters and the cost function as an additional local factor to get the two outputs (with α set to a large number, 10^{12} in our experiments):

$$h^t = \underset{y' \in \mathcal{Y}(x_t)}{\arg\max} \left(\mathbf{w}^t \cdot \mathbf{f}(x_t, y') - \alpha \cdot cost(y_i, y)\right) \tag{4.4}$$

$$f^t = \underset{y' \in \mathcal{Y}(x_t)}{\arg\max} \left(\mathbf{w}^t \cdot \mathbf{f}(x_t, y') + cost(y_i, y)\right) \tag{4.5}$$

and compute the stochastic gradient:

$$\mathbf{s}^t = \mathbf{f}(x_t, h^t) - \mathbf{f}(x_t, f^t) - 2\lambda \mathbf{w}^t$$

We then update the parameters and go to the next example. Each component i of the parameter vector gets updated like so:

$$\theta_i^{t+1} = \theta_i^t - \frac{\eta}{\sqrt{\sum_{t'=1}^{t} s_i^{t'}}} s_i^t$$

η is the learning rate which we set to 1. For relation identification training, we replace $\boldsymbol{\theta}$ and $\mathbf{f}(X, Y)$ in the above equations with $\boldsymbol{\psi}$ and

$$\mathbf{g}(X, Y) = \textstyle\sum_{e \in E_G} \mathbf{g}(e).$$

We ran AdaGrad for ten iterations for concept identification, and five iterations for relation identification. The number of iterations was chosen by early stopping on the development set.

4.6 Experiments

We evaluate our parser on LDC2015E86. Statistics about this corpus are given in Table 4.2.

Split	Sentences	Tokens
Train	17k	340k
Dev.	1.4k	29k
Test	1.4k	30k

Table 4.2: Train/dev./test split.

P	R	F_1
.75	.79	.77

Table 4.3: Concept identification performance on test.

For the performance of concept identification, we report precision, recall, and F_1 of labeled spans using the induced labels on the training and test data as a gold standard (Table 4.3). Our concept identifier achieves 77% F_1 on the test data. Precision drops 5% between train and test, but recall dops 13% on test, implicating unseen concepts as a significant source of errors on test data.

We evaluate the performance of the full parser using Smatch v1.0 (Cai and Knight, 2013), which counts the precision, recall and F_1 of the concepts and relations together. Using the full pipeline (concept identification and relation identification stages), our parser achieves 67% F_1 on the test data (Table 4.4). Using gold concepts with the relation identification stage yields a much higher Smatch score of 78% F_1. As a comparison, AMR Bank annotators have a consensus inter-annotator agreement Smatch score of 83% F_1. The runtime of our system is given in Figure 4.3.

The drop in performance of 11% F_1 when moving from gold concepts to system concepts suggests that further improvements to the concept identification stage would be beneficial.

Concepts	P	R	F_1
gold	.83	.74	.78
automatic	.70	.65	.67

Table 4.4: Parser performance on test.

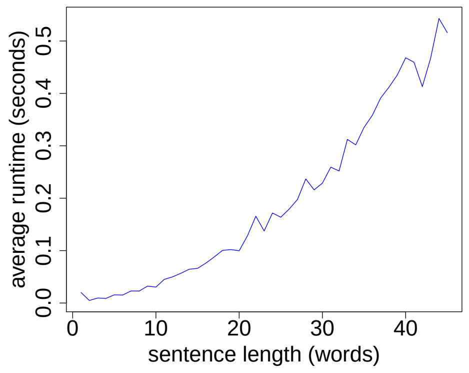


Figure 4.3: Runtime of JAMR (all stages).

4.7 Related Work

AMR parsing methods attempted so far can be categorized into five broad categories: graph, transition, grammar, MT, seq2seq, and conversion based methods. Graph-based algorithms produce a graph by maximizing (approximately or exactly) a scoring function for graphs, e.g. the work in this thesis. Transition-based algorithms construct a graph through a series of actions, and once an action is selected the alternatives for that action are never reconsidered. Grammar-based methods use a grammar to constrain the set of graphs that are considered while producing a parse, and the highest scoring graph according to a scoring function is returned. MT and seq2seq based methods produce the textual form of an AMR graph directly from the input using MT or sequence to sequence neural methods, usually with some pre-processing and/or post-processing to make the task easier. Conversion-based methods convert the output of some other semantic parser into AMR graphs.

4.7.1 Graph-based Methods

Graph-based AMR parsing methods were introduced with the first AMR parser JAMR (Flanigan et al., 2014), which was described in this chapter. JAMR's approach to relation identification is inspired by graph-based techniques for non-projective syntactic dependency parsing. Minimum spanning tree algorithms—specifically, the optimum branching algorithm of Chu and Liu (1965) and Edmonds (1967)—were first used for dependency parsing by McDonald et al. (2005). Later extensions allowed for *higher-order* (non–edge-local) features, often making use of relaxations to solve the NP-hard optimization problem. Mcdonald and Pereira (2006) incorporated second-order features, but resorted to an approximate algorithm. Others have formulated the problem as an integer linear program (Riedel and Clarke, 2006, Martins et al., 2009). TurboParser (Martins et al., 2013) uses AD^3 (Martins et al., 2011), a type of augmented Lagrangian relaxation, to integrate third-order features into a CLE backbone. Future work might extend JAMR to incorporate additional linguistically motivated constraints and higher-order features.

The JAMR parser framework has so far been improved upon in two papers (Werling et al., 2015, Flanigan et al., 2016a). The second is presented in this thesis, and a comparison to the first will be discussed here. Both improvements to (Flanigan et al., 2014) recognized that concept identification was a major bottleneck, and tried to improve this stage of the parser. Werling et al. (2015) introduced a set of actions that generate concepts, training a classifier to generate additional concepts during concept identification. This boosted the Smatch score by $3\ F_1$. In comparison to the work presented here, the actions of Werling et al. (2015) are similar to the new sources of concepts in Flanigan et al. (2016a), and the latter also introduces the infinite ramp loss to boost performance further.

Zhou et al. (2016) present another graph-based parser, which uses beam search to jointly identify concepts and relations. Concepts and relations are given weights, and the predicted graph is the highest scoring graph satisfying the same constrains as JAMR. To compare to JAMR's pipelined model, they use the same feature set as Flanigan et al. (2014), and observe a 5 point increase in Smatch F-score. They also engineer more features for concept ID, and observe an additional 3 point improvement.

Another AMR parser that can be considered graph-based is the parser introduced in Foland and Martin (2016) and extended to a state-of-the-art parser (at the time of introduction) in Foland and Martin (2017). The parser uses five Bi-LSTM networks to predict probabilities for concepts, core argument relations, non-core relations, attributes, and named entities. Similar to the approach of JAMR, the parser first identifies concept fragments and then connnects them together by adding edges which satisfy similar constraints as JAMR's. A notable difference to JAMR is the algorithm used to add edges is greedy, and recalculates probabilities after each edge is added.

Rao et al. (2016) use learning to search to predict AMR graphs. Like JAMR, concepts are first predicted and then relations are predicted. As concepts and relations are predicted, the current prediction can use the results of the previous predictions. This work uses the same aligner and decomposition of the graph into fragments as JAMR, but does not have a connectivity constraint. Instead the graph is connect by selecting a top node and connecting all components to this node.

4.7.2 Transition-based Methods

Transition-based algorithms, inspired from transition-based dependency parsing algorithms , have been used for AMR parsing starting with Wang et al. (2015b). In a transition-based algorithm, the AMR graph is constructed incrementally by actions which are chosen by a classifier. This approach has been used to convert dependency trees to AMR graphs (Wang et al., 2015b, Wang et al., 2015a, Brandt et al., 2016, Barzdins and Gosko, 2016, Puzikov et al., 2016, Goodman et al., 2016b, Goodman et al., 2016a, Wang and Xue, 2017, Nguyen and Nguyen, 2017, Gruzitis et al., 2017), and to convert sentences (with possibly additional annotations) to AMR graphs directly (Damonte et al., 2017, Buys and Blunsom, 2017, Ballesteros and Al-Onaizan, 2017).

4.7.3 Grammar-based Methods

Grammar-based approaches to AMR parsing have also been developed. These approaches include parsers based on Synchronous Hyperedge Replacement Grammars (SHRGs) (Peng et al., 2015, Peng and Gildea, 2016, Jones et al., 2012, Braune et al., 2014) and Combinatory Categorial Grammars (Artzi et al., 2015, Misra and Artzi, 2016). Directed acyclic graph (DAG) automata (Quernheim and Knight, 2012a, Quernheim and Knight, 2012b) have also been investigated (Braune et al., 2014).

4.7.4 Neural Methods

With the rise of deep learning in NLP , researchers have also applied deep learning to AMR parsing. Neural methods have been applied to graph-based methods (Foland and Martin, 2016, Foland and Martin, 2017), transition-based methods (Puzikov et al., 2016, Buys and Blunsom, 2017), CCG-based methods (Misra and Artzi, 2016), sequence to sequence (seq2seq) AMR parsing (Barzdins and Gosko, 2016, Konstas et al., 2017, Peng et al., 2017, Viet et al., 2017), and character-level (seq2seq) AMR parsing (Barzdins and Gosko, 2016, van Noord and Bos, 2017b, van Noord and Bos, 2017a). Structured neural methods tailored to the AMR parsing problem, such as neural graph or transition-based methods, or tailored seq2seq methods (van Noord and Bos, 2017a), have so far shown better performance than out-of-the-box seq2seq methods.

4.7.5 Conversion-based and Other Methods

Conversion-based methods include converting dependency trees to AMR (Wang et al., 2015b, inter alia) and converting the output of existing parsers for other semantic representations to AMR. Conversion-based parsers have been built for Microsoft's Logical Form (Vanderwende et al., 2015), Boxer's Discourse Representation Structures (Bjerva et al., 2016), and the logical forms of the Treebank Semantics Corpus (Butler, 2016). Other methods include using a statistical machine translation system to generate AMRs (Pust et al., 2015).

4.7.6 Previous Semantic Parsing Approaches

Semantic parsing has a long history in natural language processing. Early natural language understanding (NLU) applications parsed to semantic representations ranging from shallow rearrangements of text (Weizenbaum, 1966) to deep semantic representations based on first order or other types of logic (Darlington and Charney, 1963, Raphael, 1964, Coles, 1968, Green, 1969). These systems used semantic analysers that were hand-crafted for limited domains. Limited domain, deep, hand-crafted semantic analysers were constructed with increasing levels of sophistication (Wilks, 1973, Woods, 1978, Hirschman et al., 1989, Goodman and Nirenburg, 1991, Mitamura et al., 1991). Broad-coverage, deep, hand-crafted semantic analysers have also been constructed (Alshawi, 1992, Copestake and Flickinger, 2000, Žabokrtskỳ et al., 2008), sometimes using a statistical parser to build their analysis (Allen et al., 2007, Allen et al., 2008, Bos, 2008, Apresian et al., 2003).

Learning semantic parsers with supervised machine learning started on limited domains with shallow (Miller et al., 1994, Miller et al., 1996) and then deep (Zelle and Mooney, 1996, Zettlemoyer and Collins, 2005, Wong and Mooney, 2006) semantic representations, although there is earlier work on learning semantic parsers in the language acquisition literature (Quillian, 1969, Anderson, 1977). Supervised learning of broad-coverage semantic parsing was initiated by Gildea and Jurafsky (2000), advocating construction of shallow semantic parsers, and much work was done in the shallow semantic parsing tasks of semantic role labeling (Gildea and Jurafsky, 2000, Carreras and Màrquez, 2005, Baker et al., 2007, inter alia) and semantic dependency parsing (Surdeanu et al., 2008, Oepen et al., 2014, Oepen et al., 2015, inter alia). Work continued towards the goal of learning deep semantic parsers, and researchers learned larger, but still limited-domain deep semantic parsers using indirect supervision (Liang et al., 2009, Artzi and Zettlemoyer, 2011). Striving towards the goal of constructing robust broad-coverage deep semantic parsers, work was done learning broad-coverage deep semantic parsers with partial annotation (Riezler et al., 2002), unsupervised learning (Poon and Domingos, 2009), and indirect supervision (Berant et al., 2013).

A close relative to the work in this thesis is the work of Klimeš (Klimeš, 2006b, Klimeš, 2006a, Klimeš, 2007). This work learned a broad-coverage semantic parser using supervised machine learning for the tectogrammatical layer of the Prague Dependency Treebank (PDT, (Böhmová et al., 2003)). Sometimes called deep syntactic analysis (Žabokrtský et al., 2008), the tectogrammatical layer of the PDT can be considered a deep representation of meaning. This representation is similar to AMR, but with different handling of various semantic phenomenon.

Chapter 5

Generation

Many natural language tasks involve the production of natural language. Typical examples include summarization, machine translation, and question answering with natural language responses. An end-to-end system that learns to perform these tasks also learns to produce natural language. However, producing natural language fluently is challenging, and learning to produce natural language at the same time as learning a task usually requires large amounts of supervised, task-specific training data.

One solution to this problem is to introduce a separate component called a **generator**, shared across tasks, that generates natural language from a specification of the desired output. The task of the generator, producing natural language, is called **generation**. We call the representation input into the generator the **input representation**.

To be useful in downstream applications, desired properties of the input representation are that it be:

- Expressive enough to convey the desired meaning of natural language
- Possible for applications to produce
- Possible to generate from

AMR, being a whole-sentence, general-purpose semantic representation that abstracts across close paraphrases, is a candidate for the input representation of a general-purpose generator. Already, even before general purpose generators were developed, there was preliminary work on using AMR as an intermediate representation in applications such as such as machine translation (Jones et al., 2012) and summarization (Liu et al., 2015). And, as this chapter demonstrates, it is possible to generate reasonably high quality English sentences from AMR, with further improvements possible in the future.

In this chapter, to facilitate the use case of AMR as an intermediate representation, we consider generation of English from AMR. We develop the first published method for this task, demonstrating that it is possible to generate English sentences from AMR.

Generation is essentially the inverse problem of parsing, but has its own set of unique challenges. In a way, the generation problem is easier than parsing. The generator only needs to know one way of expressing some given semantics, whereas the parser must be ready to recognize all ways of representing the semantics it is expected to understand. However, the problem is in other ways more difficult: rather than accept any (possibly ungrammatical) input, the generator is expected to produce grammatical output. Also, because the semantic representation may leave important details underspecified, the generator must be able to fill in these details. In our case, AMR for example does not specify tense, number, definiteness, and whether a concept should be referred to nominally or verbally.

The method we develop to generate from AMR first converts the input AMR graph into a tree, and then generates a string using a tree-to-string transducer. Although this approach at first seems unsatisfactory because it throws away the graph-structure aspect of the problem, there is a deeper motivation for considering this approach.

The motivation for our approach is the following: graphs in the AMR corpus have a surprising property – for almost all of the graphs, there is a spanning tree of the graph for which the annotated sentence becomes projective in the tree. This is surprising because one would not expect projectiveness to occur for arbitrary graphs over the concepts in a sentence, or for an arbitrary linearization process. But it seems that humans have an interesting approach to linearizing the semantic graphs in English: they seem to follow a tree structure. This seems to indicate that a tree-to-string transducer can model the generation process from AMR to English.

5.1 Method Overview

Our approach to generation from AMR is the following: the input AMR graph is converted to a tree (§5.3.1), which is input into the weighted intersection of a tree-to-string transducer (§5.2.2) with a language model. The output English sentence is the (approximately) highest-scoring sentence according to a feature-rich discriminatively trained linear model (§5.3.2). The model weights for the tree transducer are tuned on a development set (§5.3.3). The rules for the transducer are extracted from the AMR corpus and learned generalizations; they are of four types: **basic rules** (§5.4.1), **synthetic rules** created using a specialized model (§5.4.2), **abstract rules** (§5.4.3), and a small number of **handwritten rules** (§5.4.4).

After discussing notation and background on tree-transducers (§5.2), we describe our approach (§5.3) and the learning of the rules from the AMR corpus (§5.4). We discuss experiments (§5.5), perform an error analysis (§5.5), and give related work (§5.6).

5.2 Notation and Background

This section gives our notation for the tree-transducers used in the generator. We start with a description of the input to the tree-transducer (§5.2.1), give some notation for the tree-transducers (§5.2.2), and finally give a short-hand notation for the rules in the tree-transducer (§5.2.3).

5.2.1 Transducer Input Representation

In order to be input into the tree transducer, AMR graphs are transformed into trees by removing cycles (discussed in §5.3.1) and applying a straightforward transformation to a representation described here.

The tree input into the tree-transducer is represented as a phrase-structure-like tree which we call the **transducer input representation (transducer input)**, and is as follows. Let the node and edge labels for the AMR graph be from the set of **concepts** L_N and **relations** L_E, respectively. For a node n with label $\mathcal{C}$ and outgoing edges $n \xrightarrow{L_1} n_1, \ldots, n \xrightarrow{L_m} n_m$ sorted lexicographically by L_i (each an element of L_E), the transducer input of the tree rooted at n is:[1]

$$(X\ \mathcal{C}\ (L_1\ T_1) \ldots (L_m\ T_m)) \tag{5.1}$$

where each T_i is the transducer input of the tree rooted at n_i. See Fig. 5.1 for an example. A LISP-like textual formatting of the transducer input in Fig. 5.1 is:

$$(X\ \textit{want-01}\ (\textit{ARG0}\ (X\ \textit{boy}))\ (\textit{ARG1}\ (X\ \textit{ride-01}\ (\textit{ARG0}\ (X\ \textit{bicycle}\ (\textit{mod}\ (X\ \textit{red})))))))$$

To ease notation, we use the function $sort[]$ to lexicographically sort edge labels in a transducer input. Using this function, an equivalent way of representing the transducer input in Eq. 5.1, if the L_i are unsorted, is:

$$(\mathrm{X}\ \mathcal{C}\ sort[(L_1\ T_1) \ldots (L_m\ T_m)])$$

5.2.2 Tree Transducers

The transducer input is converted into a word sequence using a weighted tree-to-string transducer. The tree transducer formalism we use is one-state extended linear, non-deleting tree-to-string (**1-xRLNs**) transducers (Huang et al., 2006, Graehl and Knight,

[1]If there are duplicate child edge labels, then the conversion process is ambiguous and any of the conversions can be used. The ordering ambiguity will be handled later in the tree-transducer rules.

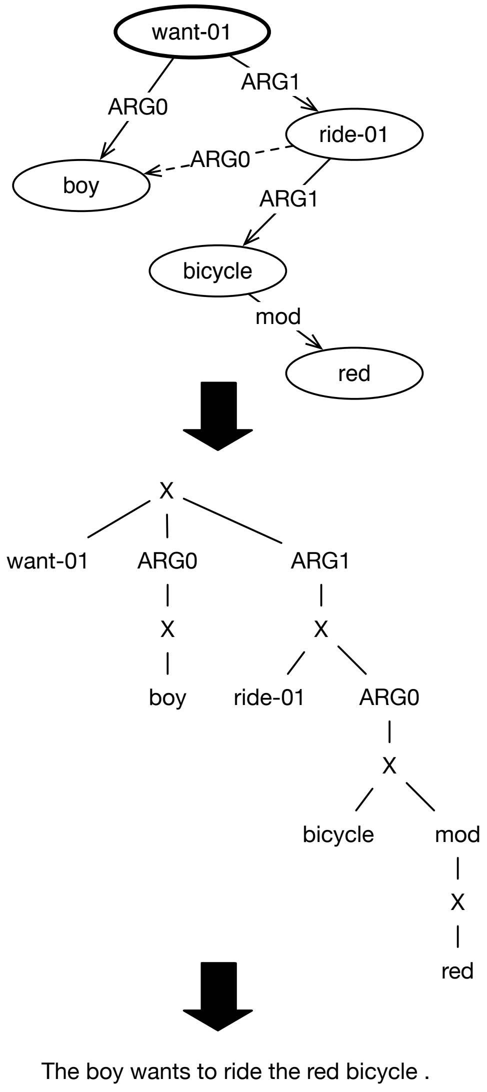


Figure 5.1: The generation pipeline. An AMR graph (top), with a deleted re-entrancy (dashed), is converted into a transducer input representation (transducer input, middle), which is transduced to a string using a tree-to-string transducer (bottom).

2004).[2] What follows is a short introduction to tree-to-string transducers, followed by a shorthand notation we use for the rules in the transducer.

Definition 1. *(From Huang et al., 2006.) A* ***1-xRLNs transducer*** *is a tuple* $(N, \Sigma, W, \mathcal{R})$

[2]Multiple states would be useful for modeling dependencies in the output, but we do not use them here.

where N is the set of nonterminals (relation labels and X), Σ is the input alphabet (concept labels), W is the output alphabet (words), and $\mathcal{R}$ is the set of rules. A rule in $\mathcal{R}$ is a tuple (t, s, ϕ) where:

1. *t is the LHS tree, whose internal nodes are labeled by nonterminal symbols, and whose frontier nodes are labeled terminals from Σ or variables from a set $\mathcal{X} = \{X_1, X_2, \ldots\}$;*
2. *$s \in (\mathcal{X} \cup W)^*$ is the RHS string;*
3. *ϕ is a mapping from $\mathcal{X}$ to nonterminals N.*

A rule is a **purely lexical rule** if it has no variables.

As an example, the tree-to-string transducer rules which produce the output sentence from the transducer input in Fig. 5.1 are:

$$
\begin{gathered}
(X\ \textit{want-01}\ (\textit{ARG0}\ X_1)\ (\textit{ARG1}\ X_2)) \rightarrow \text{The } X_1 \text{ wants to } X_2\ . \\
(X\ \textit{ride-01}\ (\textit{ARG1}\ X_1)) \rightarrow \text{ride the } X_1 \\
(X\ \textit{bicycle}\ (\textit{mod}\ X_1)) \rightarrow X_1 \text{ bicycle} \\
(X\ \textit{red}) \rightarrow \text{red} \\
(X\ \textit{boy}) \rightarrow \text{boy}
\end{gathered}
\tag{5.2}
$$

Here, all X_i are mapped by a trivial ϕ to the nonterminal X.

The output string of the transducer is the target projection of the derivation, defined as follows:

Definition 2. *(From Huang et al., 2006.) A **derivation** d, its **source and target projections**, denoted $\mathcal{S}(d)$ and $\mathcal{E}(d)$ respectively, are recursively defined as follows:*

1. *If $r = (t, s, \phi)$ is a purely lexical rule, then $d = r$ is a derivation, where $\mathcal{S}(d) = t$ and $\mathcal{E}(d) = s$;*
2. *If $r = (t, s, \phi)$ is a rule, and d_i is a (sub)-derivation with the root symbol of its source projection maching the corresponding substition node in r, i.e., root$(\mathcal{S}(d_i)) = \phi(x_i)$, then $d = r(d_1, \ldots, d_m)$ is also a derivation, where $\mathcal{S}(d) = [x_i \mapsto \mathcal{S}(d_i)]t$ and $\mathcal{E}(d) = [x_i \mapsto$*

$\mathcal{E}(d_i)]s$.

The notation $[x_i \mapsto y_i]t$ is shorthand for the result of substituting y_i for each x_i in t, where x_i ranges over all variables in t.

The set of all derivations of a target string e with a transducer T is denoted

$$\mathcal{D}(e, T) = \{d \mid \mathcal{E}(d) = e\}$$

where d is a derivation in T.

5.2.3 Shorthand Notation for Transducer Rules

We use a shorthand notation for the transducer rules that is useful when discussing rule extraction and synthetic rules. Let f_i be a transducer input. The transducer input has the form

$$f_i = (X\ \mathcal{C}\ (L_1\ T_1) \ldots (L_m\ T_m))$$

where $L_i \in L_E$ and $T_1, \ldots, T_m$ are transducer inputs.[3] Let $A_1, \ldots A_n \in L_E$. We use

$$(f_i, A_1, \ldots, A_n) \to r \tag{5.3}$$

as shorthand for the rule:

$$(X\ \mathcal{C}\ sort[(L_1\ T_1)\ \ldots\ (L_m\ T_m)(A_1\ X_1)\ \ldots\ (A_n\ X_n)]) \to r \tag{5.4}$$

Note r must contain the variables $X_1\ \ldots\ X_n$. In (5.3) and (5.4), argument slots with relation labels A_i have been added as children to the root node of the transducer input f_i.

For example, the shorthand for the transducer rules in (5.2) is:

$$((X\ \textit{want-01}), \textit{ARG0}, \textit{ARG1}) \to \text{The } X_1 \text{ wants to } X_2 \text{ .}$$

$$((X\ \textit{ride-01}), \textit{ARG1}) \to \text{ride the } X_1$$

$$((X\ \textit{bicycle}), \textit{mod}) \to X_1 \text{ bicycle}$$

$$((X\ \textit{red})) \to \text{red} \tag{5.5}$$

[3]If f_i is just a single concept with no children, then $m = 0$ and $f_i = (X\ \mathcal{C})$.

5.3 Generation

To generate a sentence e from an input AMR graph G, a spanning tree G' of G is computed, and then transformed into a string using a tree-to-string transducer. Here we discuss the algorithm for choosing the spanning tree, and the decoding and learning algorithms for the tree-to-string transducer.

5.3.1 Spanning Tree

The choice of spanning tree may have a large effect on the output, since the transducer output will always be a projective reordering of the transducer input tree's leaves. Our spanning tree results from a breadth-first-search traversal, visiting child nodes in lexicographic order of the relation label (inverse relations are visited last). The edges traversed are included in the tree. This simple heuristic, which works well for control structures, is a baseline which can potentially be improved in future work.

5.3.2 Decoding

Let $T = (N, \Sigma, W, \mathcal{R})$ be the tree-to-string transducer used for generation. The output sentence is the highest scoring transduction of G':

$$e = \mathcal{E}\left(\arg\max_{d \in \mathcal{D}(G',T)} score(d; \boldsymbol{\theta})\right) \tag{5.6}$$

Eq. 5.6 is solved approximately using the cdec decoder for machine translation (Dyer et al., 2010). The score of the transduction is a linear function (with coefficients $\boldsymbol{\theta}$) of a vector of features. The features are the output sequence's language model log-probability and features associated with each rule r in the derivation (denoted $\mathbf{f}(r)$ and listed in Table 5.1). Thus the scoring function for derivation d of a transduction is:

$$score(d; \boldsymbol{\theta}) = \theta_{LM} \log(p_{LM}(\mathcal{E}(d))) + \sum_{r \in d} \boldsymbol{\theta}^\top \mathbf{f}(r)$$

Name	**Description**
Rule	1 for every rule
Basic	1 for basic rules, else 0
Synthetic	1 for synthetic rules, else 0
Abstract	1 for abstract rules, else 0
Handwritten	1 for handwritten rules, else 0
Rule given concept	log(number of times rule extracted / number of times concept observed in training data) (only for basic rules, 0 otherwise)
… without sense	same as above, but with sense tags for concepts removed
Synthetic score	model score for the synthetic rule (only for synthetic rules, 0 otherwise)
Word count	number of words in the rule
Non-stop word count	number of words not in a stop word list
Bad stop word	number of words in a list of meaning-changing stop words, such as "all, can, could, only, so, too, until, very"
Negation word	number of words in "no, not, n't"

Table 5.1: Rule features. There is also an indicator feature for every handwritten rule.

5.3.3 Discriminative Training

The feature weights $\boldsymbol{\theta}$ of the transducer are learned by maximizing the BLEU score (Papineni et al., 2002) of the generator on a development dataset (the standard development set for LDC2014T12) using MERT (Och, 2003). The features are the language model log-probability and the rule features listed in Table 5.1.

5.4 Rule Learning

In the next four sections, we describe how the rules $\mathcal{R}$ of the trandsucer are extracted and generalized from the training corpus. The rules are a union of four types of rules:

basic rules (§5.4.1), **synthetic rules** (§5.4.2), **abstract rules** (§5.4.3), and a small number of **handwritten rules** (§5.4.4).

5.4.1 Basic Rules

The basic rules, denoted $\mathcal{R}_B$, are extracted from the training AMR data using an algorithm similar to extracting tree transucers from tree-string aligned parallel corpora (Galley et al., 2004). Informally, the rules are extracted from a sentence $\boldsymbol{w} = \langle w_1, \ldots, w_n \rangle$ with AMR graph G as follows:

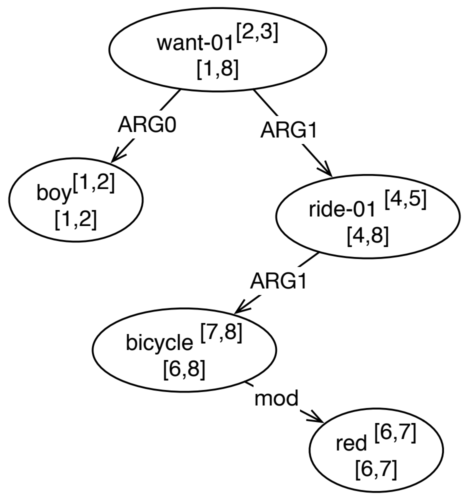
1. The AMR graph and the sentence are aligned; we use the aligner from §2.6, which aligns non-overlapping subgraphs of the graph to spans of words. The subgraphs that are aligned are called **fragments**. In our aligner, all fragments are trees.
2. G is replaced by its spanning tree by deleting relations that use a variable in the AMR annotation.
3. In the spanning tree, for each node i, we keep track of the word indices $b(i)$ and $e(i)$ in the original sentence that trap all of i's descendants. (This is calculated using a simple bottom-up propagation from the leaves to the root.)
4. For each aligned fragment i, a rule is extracted by taking the subsequence $\langle w_{b(i)} \ldots w_{e(i)} \rangle$ and "punching out" the spans of the child nodes (and their descendants) and replacing them with argument slots.

See Fig. 5.2 for examples.

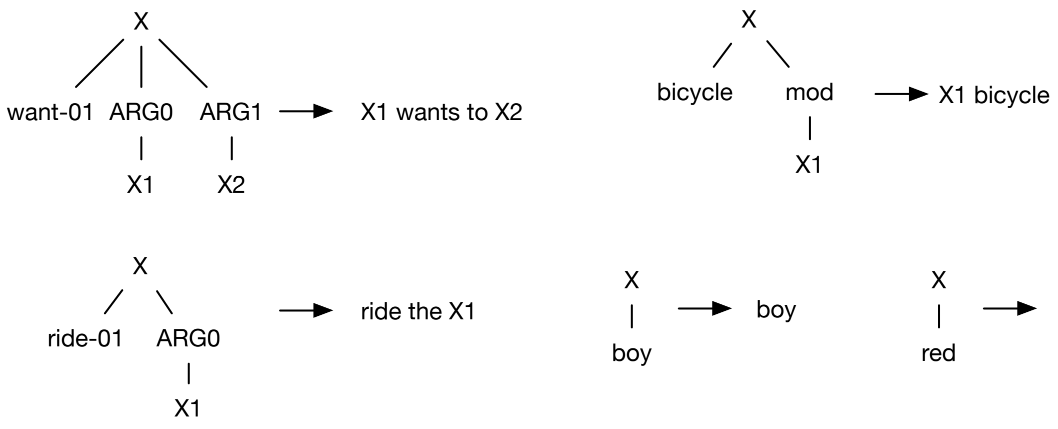
More formally, assume the nodes in G are numbered $1, \ldots, N$ and the fragments are numbered $1, \ldots, F$. Let $\mathrm{nodes} : \{1, \ldots, F\} \rightarrow 2^{\{1,\ldots,N\}}$ and $\mathrm{root} : \{1, \ldots, F\} \rightarrow \{1, \ldots, N\}$ be functions that return the nodes in a fragment and the root of a fragment, respectively, and let $\mathrm{children} : \{1, \ldots, N\} \rightarrow 2^{\{1,\ldots,N\}}$ return the child nodes of a node. We consider a node aligned if it belongs to an aligned fragment. Let the span of an aligned node i be denoted by endpoints a_i and a'_i; for unaligned nodes, $a_i = \infty$ and $a'_i = -\infty$ (depicted

$_0$ The $_1$ ((boy) $_2$ wants $_3$ to $_4$ (ride $_5$ the $_6$ ((red) $_7$ bicycle))) $_8$

(a) Example sentence annotated with word indices, and bracketed according to $b(i)$ and $e(i)$ from the graph in (b).



(b) AMR tree for the example sentence in (a) annotated with a_i, a'_i (superscripts) and $b(i)$, $e(i)$ (subscripts).



(c) Extracted rules from this example pair.

Figure 5.2: Example rule extraction from an AMR-annotated sentence. The AMR graph has already been converted to a tree by deleting relations that use a variable in the AMR annotation (step 2 in §5.4.1).

with superscripts in Fig. 5.2). The node alignments are propagated by defining $b(\cdot)$ and $e(\cdot)$ recursively, bottom up:

$$b(i) = \min(a_j, \min_{j \in \text{children}(i)} b(j))$$

$$e(i) = \max(a'_j, \max_{j \in \text{children}(i)} e(j))$$

Also define functions $\tilde{b}$ and $\tilde{e}$, from fragment indices to integers, as:

$$\tilde{b}(i) = b(\text{root}(i))$$

$$\tilde{e}(i) = e(\text{root}(i))$$

For fragment i, let $C_i = \text{children}(\text{root}(i)) - \text{nodes}(i)$, which is the children of the fragment's root concept that are not included in the fragment. Let f_i be the transducer input for fragment i.[4] If C_i is empty, then the rule extracted for fragment i is:

$$r_i : (f_i) \rightarrow w_{\tilde{b}(i):\tilde{e}(i)} \tag{5.7}$$

Otherwise, let $m = |C_i|$, and denote the edge labels from $\text{root}(i)$ to elements of C_i as $A_1(i) \dots A_m(i)$. For $j \in \{1, \dots, m\}$, let k_j select the elements c_{k_j} of C_i in ascending order of $b(k_j)$. Then the rule extracted for fragment i is:

$$\begin{gathered} r_i : (f_i, A_{k_1}(i), \dots A_{k_m}(i)) \rightarrow \\ w_{\tilde{b}(i):\tilde{b}(k_1)}\ X_1\ w_{\tilde{e}(k_1):\tilde{b}(k_2)}\ X_2 \dots\dots w_{\tilde{e}(k_{m-1}):\tilde{b}(k_m)} X_m\ w_{\tilde{e}(k_m):\tilde{e}(i)} \end{gathered} \tag{5.8}$$

A rule is only extracted if the fragment i is aligned and the child spans do not overlap. Fig. 5.2 gives an example of a tree annotated with alignments, b and e, and the extracted rules.

[4]I.e., the nodes in fragment i, with the edges between them, represented as a transducer input.

5.4.2 Synthetic Rules

The synthetic rules, denoted $\mathcal{R}_S(G)$, are created to generalize the basic rules and overcome data sparseness resulting from our relatively small training dataset. Our synthetic rule model considers an AMR graph G and generates a set of rules for each node in G. A synthetic rule's LHS is a transducer input f with argument slots $A_1 \dots A_m$ (this is the same form as the LHS for basic rules). For each node in G, one or more LHS are created (we will discuss this further below), and for each LHS, a set of k-best synthetic rules are produced. The simplest case of a LHS is just a concept and argument slots corresponding to each of its children.

For a given LHS, the synthetic rule model creates a RHS by concatenating together a string in W^* (called a **concept realization** and corresponding to the concept fragment) with strings in $W^*\mathcal{X}W^*$ (called an **argument realization** and corresponding to the argument slots). See the right of Fig. 5.3 for a synthetic rule with concept and argument realizations highlighted.

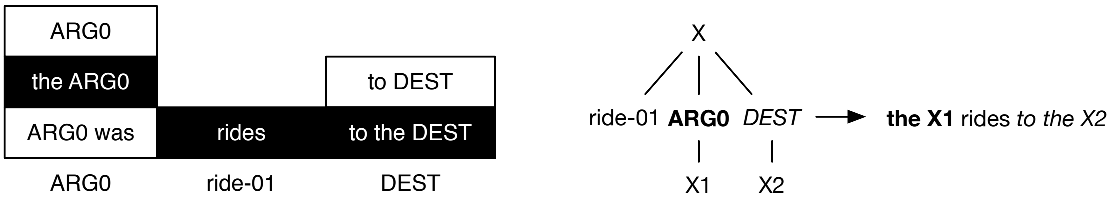


Figure 5.3: Synthetic rule generation for the rule shown at right. For a fixed permutation of the concept and arguments, choosing the argument realizations can be seen as a sequence labeling problem (left, the highlighted sequence corresponds to the rule at right). In the rule RHS on the right, the realization for ARG0 is bold, the realization for DEST is italic, and the realization for ride-01 is normal font.

Thus, synthetic rules are a concept fragment with a list of arguments (relation labels) transduced into a linear string, where the linear string is a permutation of the argument realizations and concept realization. More precisely, in our notation synthetic rules have the form:

$$r : (f, A_1, \dots A_m) \rightarrow$$
$$\mathbf{l}_{k_1} X_{k_1} \mathbf{r}_{k_1} \dots \mathbf{l}_{k_c} X_{k_c} \mathbf{r}_{k_c} \; \mathbf{c} \; \mathbf{l}_{k_{c+1}} X_{k_{c+1}} \mathbf{r}_{k_{c+1}} \dots \mathbf{l}_{k_m} X_{k_m} \mathbf{r}_{k_m} \tag{5.9}$$

where:

- f is a transducer input.
- Each $A_i \in L_E$.
- $\langle k_1, \dots, k_m \rangle$ is a permutation of $\langle 1, \dots, m \rangle$
- $\mathbf{c} \in W^*$ is the **realization** of transducer input f.
- Each $\mathbf{l}_i, \mathbf{r}_i \in W^*$ and $X_i \in \mathcal{X}$. Let $R_i = \langle \mathbf{l}_i, \mathbf{r}_i \rangle$ denote the **realization of argument** i.
- $c \in [0, m]$ is the position of $\mathbf{c}$ among the realizations of the arguments.

Let $\mathcal{F}$ be the space of all possible transducer inputs. Synthetic rules make use of three lookup tables (which are partial functions) to provide candidate realizations for concepts and arguments: a table for concept realizations $lex : \mathcal{F} \rightarrow 2^{W^*}$, a table for argument realizations when the argument is on the left $left_{lex} : \mathcal{F} \times L_E \rightarrow 2^{W^*}$, and a table for argument realizations when the argument is on the right $right_{lex} : \mathcal{F} \times L_E \rightarrow 2^{W^*}$. These tables are constructed during basic rule extraction, the details of which are discussed below.

Synthetic rules are selected using a linear model with features g and coefficients $\boldsymbol{\psi}$, which scores each RHS for a given LHS. For LHS = $(f, A_1, \dots A_m)$, the RHS is specified completely by $\mathbf{c}, c, R_1, \dots, R_m$ and a permutation $k_1, \dots, k_m$. For each node in G, and for each transducer input f in the domain of lex that matches the node, a LHS is created, and a set of K synthetic rules is produced for each $\mathbf{c} \in lex(f)$. The rules produced are the K-best solutions to:

$$\begin{aligned}\operatorname*{arg\,max}_{c,k_1\ldots k_m,R_1,\ldots,R_m} &\Big(\sum_{i=1}^{c} \boldsymbol{\psi}^\top \mathbf{g}(R_{k_i}, A_{k_i}, \mathbf{c}, i, c) \\ &+ \boldsymbol{\psi}^\top \mathbf{g}(\langle\epsilon,\epsilon\rangle, *, \mathbf{c}, c+1, c) + \sum_{i=c+1}^{m} \boldsymbol{\psi}^\top \mathbf{g}(R_{k_i}, A_{k_i}, \mathbf{c}, i+1, c)\Big) \end{aligned} \tag{5.10}$$

where the max is over $c \in 0 \ldots m$, $k_1, \ldots, k_m$ is any permutation of $1, \ldots, m$, and $R_i \in \mathit{left}_{\mathit{lex}}(A_i)$ for $i < c$ and $R_i \in \mathit{right}_{\mathit{lex}}(A_i)$ for $i > c$. $*$ is used to denote the concept position. ϵ is the empty string.

The generated synthetic rules are added to the basic rules in the tree-transducer, using the features in listed Table 5.1. These features have binary indicator features that distinguish basic rules from synthetic rules, and include the synthetic rule model score as a feature in the tree transducer.

The best solution to Eq. 5.10 is found exactly by brute force search over concept position $c \in [0, m+1]$ and the permutation $k_1, \ldots, k_m$. With fixed concept position and permutation, each R_i for the $\arg\max$ is found independently. To obtain the exact K-best solutions, we use dynamic programming with a K-best semiring (Goodman, 1999) to keep track of the K best sequences for each concept position and permutation, and take the best K sequences over all values of c and $k_\cdot$.

The synthetic rule model's parameters $\boldsymbol{\psi}$ are estimated using basic rules extracted from the training data. These parameters are learned independently from the parameters in the tree-transducer (whose training procedure is described in §5.3.3). However, the model score for synthetic rules is included as a feature in tree-transducer (see Table 5.1). The synthetic rule parameters $\boldsymbol{\psi}$ are learned using AdaGrad (§3.3, Duchi et al., 2011) with the Perceptron loss function (§3.2.4, Rosenblatt, 1957, Collins, 2002) for 10 iterations over all the extracted basic rules. The synthetic rule features g are listed in Table 5.2.

Feature name	**Value**
POS + A_i + "dist"	$\lvert c - i \rvert$
POS + A_i + side	1.0
POS + A_i + side + "dist"	$\lvert c - i \rvert$
POS + A_i + R_i + side	1.0
c + A_i + "dist"	$\lvert c - i \rvert$
c + A_i + side	1.0
c + A_i + side + "dist"	$\lvert c - i \rvert$
c + POS + A_i + side + "dist"	$\lvert c - i \rvert$

Table 5.2: Synthetic rule model features. POS is the most common part-of-speech tag sequence for **c**, "dist" is the string "dist", and side is "L" if $i < c$, "R" otherwise. $+$ denotes string concatenation.

The training examples for the synthetic rule model are individual extracted basic rules (duplicate example rules allowed). Thus, each extracted basic rule (duplicate rules allowed) is an example LHS-RHS pair used to as a training example when training the model parameters $\boldsymbol{\psi}$.

To allow the basic rules to be used as training examples for the synthetic rule model, the basic rules are put into the form of Eq. 5.9 by segmenting the RHS into the form

$$\mathbf{l}_1 X_1 \mathbf{r}_1 \ \ldots \ \mathbf{c} \ \ldots \ \mathbf{l}_m X_m \mathbf{r}_m \tag{5.11}$$

by choosing $\mathbf{c}, \mathbf{l}_i, \mathbf{r}_i \in W^*$ for $i \in \{1, \ldots, m\}$. Segmenting the RHS of the basic rules into the form of Eq. 5.11 is done as follows (An example segmentation is the rule RHS on the right in Fig. 5.3.): **c** is the aligned span for f. For the argument realizations, arguments to the left of **c** are associated with words to their right, and arguments to the right are associated with words to their left. Specifically, for $i < c$ (R_i to the left of **c** but not next to **c**), $\mathbf{l}_i$ is empty and $\mathbf{r}_i$ contains all words between a_i and a_{i+1}. For $i = c$ (R_i directly to the left of $\mathbf{c}$), $\mathbf{l}_i$ is empty and $\mathbf{r}_i$ contains all words between a_c and $\mathbf{c}$. For $i > c+1$, $\mathbf{l}_i$ contains all words between a_{i-1} and a_i, and for $i = c+1$, $\mathbf{l}_i$ contains all words between $\mathbf{c}$ and a_i.

The tables for lex, $left_{lex}$, and $right_{lex}$ are populated using the segmented basic rules. For each basic rule extracted from the training corpus and segmented according to the previous paragraph, $f \rightarrow \mathbf{c}$ is added to lex, and $A_{k_i} \rightarrow \langle \mathbf{l}_i, \mathbf{r}_i \rangle$ is added to $left_{lex}$ for $i \leq c$ and $right_{lex}$ for $i > c$. The permutation k_i is known during extraction in Eq. 5.8.

5.4.3 Abstract Rules

Like the synthetic rules, the abstract rules $\mathcal{R}_A(G)$ generalize the basic rules. However, abstract rules are much simpler generalizations which use part-of-speech (POS) tags to generalize. Abstract rules make use of a **POS abstract rule table**, which is a table listing every combination of the POS of the concept realization, the child arguments' labels, and rule RHS with the concept realization removed and replaced with $*$. This table is populated from the basic rules extracted from the training corpus. An example entry in the table is:

$$(\mathrm{VBD}, \mathrm{ARG0}, \mathrm{DEST}) \rightarrow X_1 \ \langle * \rangle \ \text{to the} \ X_2$$

For the LHS $(f, A_1, \ldots A_m)$, an abstract rule is created for each member of $\mathbf{c} \in lex(f)$ and the most common POS tag p for $\mathbf{c}$ by looking up p, $A_1, \ldots A_m$ in the POS abstract rule table, finding the common RHS, and filling in the concept position with $\mathbf{c}$. The set of all such rules is returned.

5.4.4 Handwritten Rules

We also have a small number of handwritten rules $\mathcal{R}_H$ for things that are regular and hard to learn. We have handwritten rules for dates, conjunctions, multiple sentences,

Split	Sentences	Tokens
Train	10,000	210,000
Dev.	1,400	29,000
Test	1,400	30,000
MT09	204	5,000

Table 5.3: Train/dev./test/MT09 split.

and the concept `have-org-role-91`. For `have-org-role-91`, we use handwritten rules because it is a difficult concept for the aligner to align, and is handled relatively well with a small set of rules. We also create pass-through rules for concepts by removing sense tags and quotes (for string literals).

5.5 Experiments

We evaluate on the AMR Annotation Release version 1.0 (LDC2014T12) dataset. We follow the recommended train/dev./test splits, except that we remove MT09 data (204 sentences) from the training data and use it as another test set. Statistics for this dataset and splits are given in Table 5.3. We use a 5-gram language model trained with KenLM (Heafield et al., 2013) on Gigaword (LDC2011T07), and use 100-best synthetic rules.

We evaluate with the BLEU scoring metric (Papineni et al., 2002) (Table 5.4). We report single reference BLEU for the LCD2014T12 test set, and four-reference BLEU for the MT09 set. We report ablation experiments for different sources of rules. When ablating handwritten rules, we do not ablate pass-through rules.

The full system achieves 22.1 BLEU on the test set, and 21.2 on MT09. Removing the synthetic rules drops the results to 9.1 BLEU on test and 7.8 on MT09. Removing the basic and abstract rules has little impact on the results. This may be because the

Rules	Test	MT09
Full	22.1	21.2
Full − basic	22.1	20.9
Full − synthetic	9.1	7.8
Full − abstract	22.0	21.2
Full − handwritten	21.9	20.5

Table 5.4: Uncased BLEU scores with various types of rules removed from the full system.

synthetic rule model already contains much of the information in the basic and abstract rules. Removing the handwritten rules has a slightly larger effect, demonstrating the value of handwritten rules in this statistical system.

5.6 Related Work

So far the work in AMR generation can be categorized into five major types: tree-transducer-based, graph-based, graph-grammar-based, MT or seq2seq-based, transition-based, and rule-based. We will discuss each one in turn (§5.6.1 - §5.6.6). The AMR generator presented in Flanigan et al. (2016b) is the first AMR generator, and to our knowledge, this is the first broad-coverage generator from a deep semantic representation learned using supervised learning. We review other approaches to generation in §5.6.7.

5.6.1 Tree-Transducer Approaches

There are two systems that have used tree-transducers for AMR generation. The first system is described in this chapter (Flanigan et al., 2016b). The second system by Gruzitis et al. (2017), which placed first in the human evaluation at SemEval 2017 (May and Priyadarshi, 2017), combines the JAMR system of Flanigan et al. (2016b) with a handwritten rule-based generator. The AMR graphs are converted to abstract syntax trees, and the Grammatical Framework (GF) generator (Ranta, 2011) is used to generate English. If the GF generator can not produce an output, the output of JAMR is used, which occurs for 88% of the sentences in the SemEval 2017 test set.

5.6.2 Graph-based Approaches

Song et al. (2016) casts AMR generation as an asymmetric generalized traveling salesman problem (ASGTP), where nodes in the ASGTP are applications of rules which map AMR graph fragments to strings. A traversal of the ASGTP graph covers the AMR graph (generates something for each concept) and orders the generated strings based on the order in which nodes in the ASGTP are visited. This is similar to casting phrase-based machine translation as a traveling salesman problem (Zaslavskiy et al., 2009), but generalized to graph-to-string generation.

5.6.3 Graph-grammar Approaches

Song et al. (2017) uses a synchronous node replacement grammar (SNRG) to generate from AMR. SNRGs are similar to synchronous context free grammars (SCFGs), but one side of the grammar is a grammar over graphs (a *graph grammar*). Song et al. (2017) combines a SNRG with a feature-based statistical model which includes a language model to generate natural language.

5.6.4 MT or seq2seq Approaches

Machine translation and seq2seq approaches have been applied to AMR generation. Pourdamghani et al. (2016) applies phrase-based machine translation to linearized AMR graphs, trying three different linearization methods. Interestingly, although they obtain a much higher BLEU score than JAMR (26.9 vs 22.0), they were judged significantly worse in the human evaluation of SemEval 2017 (May and Priyadarshi, 2017). This indicates that BLEU score is not a good metric of comparison despite its widespread use in AMR generation. It is important to note that BLEU score has been observed to be poorly correlated with BLEU score for many generation tasks (Reiter, 2018). Konstas et al. (2017) applies seq2seq neural machine translation to linearized AMR graphs. They see an improvement over previous approaches when including pseudo-gold standard AMR graphs derived from applying existing AMR parsers to English Gigaword corpus (LDC2011T07).

5.6.5 Transition-based Approaches

Transition-based AMR generation casts the generation problem as a sequence of actions which are chosen by a machine-learned classifier. In Lampouras and Vlachos (2017) and Schick (2017), AMR graphs are transformed using a sequence of actions into syntactic dependency trees (Lampouras and Vlachos, 2017) or tree structures similar to dependency trees (Schick, 2017), and then linearized as a second step in the generation process.

5.6.6 Rule-based Approaches

Two rule-based AMR generators have been developed. In the system of Gruzitis et al. (2017), AMR graphs are converted to abstract syntax trees, and the Grammatical

Framework (GF) generator (Ranta, 2011) is used to generate English. If the GF generator can not produce an output, the output of JAMR (Flanigan et al., 2016b) is used. In the system of Mille et al. (2017), AMR graphs are transformed to surface-syntactic structures using a series of rule-based graph transducers. The surface-syntactic structures are then linearized using an off-the-shelf linearizer (Bohnet et al., 2011) from the first surface realization shared task (Belz et al., 2011).

5.6.7 Previous Generation Approaches

There is a large body of work for statistical and non-statistical NLG from a variety of input representations, and we will give the major milestones and representative papers for each approach. Researchers have determined that generation involves three distinct problems: planning what to say to achieve a communicative goal (**text planning**), splitting this information into concepts and sentences (**sentence planning**), and expressing these concepts in natural language (**surface realization**) (Thompson, 1977, Rambow and Korelsky, 1992, Reiter, 1994). As the generation work in this thesis can be classified as surface realization, we will focus on that here. In summary, broad-coverage deep semantic generators have been constructed with hand-written rules, limited domain deep semantic generators have been learned using supervised learning, broad-coverage deep syntactic generators have been learned using supervised learning, and various hand-written generators have been augmented with statistical techniques, but, to our knowledge, AMR generators are the first broad-coverage deep semantic generators learned using supervised learning.

The first generators were developed using hand-crafted grammars, and high quality systems with hand-crafted grammars have been developed for both limited and broad-coverage domains. The input specification for these systems has ranged from templates (McRoy et al., 2000, Deemter et al., 2005), to abstract syntactic trees, to deep syntax, to deep semantic representations (Mann, 1983, Kasper, 1989, Bateman, 1997, Apresian et al., 2003, Žabokrtský et al., 2008). Modern surface realizers with hand-crafted grammars include PENMAN, FUF/SURGE, RealPro, and KPML. Although these systems are general purpose, they can require significant fine-tuning of grammatical and lexical resources to adapt them to a particular domain (Bohnet et al., 2010), and they are not truly broad-coverage off-the-shelf.[5] Broad-coverage hand-crafted generators with large lexical databases do exist, such as SimpleNLG (Gatt and Reiter, 2009) which maps from a slightly lower level specification than other generators, and the generators in the machine translation systems ETAP-3 (Apresian et al., 2003) and TectoMT (Žabokrtský et al., 2008), which map from UNL or deep syntax to Russian and English (ETAP-3) or deep syntax to Czech and English (TectoMT).

The first work using machine learning for generation combined rule-based generators with statistical techniques. These systems take a rule-based generator that overgenerates and rescore the output using an n-gram language model (Knight and Hatzivassiloglou, 1995, Langkilde and Knight, 1998). They generate from abstract syntax trees (Bangalore and Rambow, 2000, inter alia) or from deep semantic representations (Langkilde and Knight, 1998, inter alia).

The first work to learn surface realization from supervised training examples learned to map from shallow semantics to natural language for limited domains (Ratnaparkhi, 2000, Oh and Rudnicky, 2002, Varges and Mellish, 2001). Each of these systems learned a mapping from attribute tuples to templates. The first work we are aware of to learn to map from a deep semantic representation to natural language is Wong and Mooney (2007), which learned limited domains (GeoQuery and RoboCup) and there has been significant follow up work on this domain.

[5]To our knowledge, these systems contains fairly small lexical databases (such as KPML which contains around 1000 lexical entries). Systems that we classify as broad-coverage, such as SimpleNLG and ETAP-3, contain over 300,000 and 65,000 lexical entries respectively.

There is also work learning broad-coverage generators from shallow and deep syntax using supervised examples (Bohnet et al., 2010, Bohnet et al., 2011, Belz et al., 2011). Ballesteros et al. (2014) and Ballesteros et al. (2015) generate from a deep syntactic representation that does not have the same number of nodes as the surface syntax, an issue that must be addressed when generating from deep semantic representations such as AMR. Similar to the work in this chapter, they use also tree-transducers for generation. However they use a series of SVM classifiers for transduction rather than a weighted tree-transducer as we do here.

Chapter 6

Conclusions and Future Work

We conclude with a summary of our contributions and directions for future research.

6.1 Summary of Contributions

In this thesis, we have made the following contributions to semantic parsing and generation from semantic structures:

- A two-stage approach to semantic parsing for AMR, where concepts are first identified using a sequence labeling algorithm, and relations are then added using a graph algorithm.
- An approximate inference algorithm for finding the maximum weight, spanning connected subgraph of a graph with linear constraints and weights on the edges, which is used to find relations within an AMR parser.
- A two-stage approach to generation from AMR, where the AMR graph is first converted to a tree, and the resulting tree is transduced into a string.
- A rule-based approach to automatic alignment of AMR graphs to the training data sentences, which is used to train the AMR parser and generator.

- A new loss function which generalizes SVM loss for structured prediction when the training data contains unreachable training examples, which we use to train an AMR parser.

6.2 Future Work

Directions for future work for AMR parsing include replacing the linear model in the parsing algorithm presented here with a deep learning model, improved alignment methods, and using decoding methods for higher-order features like AD^3 or global neural scoring functions.

For AMR generation, future directions include improved modeling of interdependencies between the parent's and child's realizations (to account for subject-verb agreement, among other things), and combining the tree-transducer approach we presented with deep learning.

For both AMR parsing and generation, semi-supervised and human-in-the-loop techniques may improve performance. Annotating more data with AMR parses induced from interactions with humans, such as learning semantic representations from question answer pairs, may be a fruitful area of research.

Exploring the utility of AMR in downstream tasks is also likely a fruitful avenue of research. After all, a semantic representation is only as useful as what it can be used for.

I conclude with a quote from Vanilla Ice, whose lyric serves as encouragement to computer scientists and engineers throughout the world, and inspired me while solving problems in this thesis:

> If there was a problem
> Yo I'll solve it
> Check out the hook while my DJ revolves it
> (Ice, 1990)

Bibliography

[Allen et al.2007] James Allen, Mehdi Manshadi, Myroslava Dzikovska, and Mary Swift. 2007. Deep linguistic processing for spoken dialogue systems. In *Proceedings of the Workshop on Deep Linguistic Processing*, pages 49–56. Association for Computational Linguistics.

[Allen et al.2008] James F Allen, Mary Swift, and Will De Beaumont. 2008. Deep semantic analysis of text. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 343–354. Association for Computational Linguistics.

[Alshawi1992] Hiyan Alshawi. 1992. *The core language engine*. MIT press.

[Anderson1977] John R Anderson. 1977. Induction of augmented transition networks. *Cognitive Science*, 1(2):125–157.

[Apresian et al.2003] Juri Apresian, Igor Boguslavsky, Leonid Iomdin, Alexander Lazursky, Vladimir Sannikov, Victor Sizov, and Leonid Tsinman. 2003. ETAP-3 linguistic processor: A full-fledged NLP implementation of the MTT. In *First International Conference on Meaning–Text Theory (MTT'2003)*, pages 279–288.

[Artzi and Zettlemoyer2011] Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the conference on empirical methods in natural language processing*, pages 421–432. Association for Computational Linguistics.

[Artzi et al.2015] Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage

CCG semantic parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710.

[Baker et al.2007] Collin Baker, Michael Ellsworth, and Katrin Erk. 2007. SemEval'07 task 19: frame semantic structure extraction. In *Proceedings of the 4th International Workshop on Semantic Evaluations*.

[BakIr et al.2007] Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, Ben Taskar, and SVN Vishwanathan. 2007. *Predicting structured data*. MIT press.

[Ballesteros and Al-Onaizan2017] Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR Parsing using Stack-LSTMs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1269–1275.

[Ballesteros et al.2014] Miguel Ballesteros, Simon Mille, and Leo Wanner. 2014. Classifiers for data-driven deep sentence generation. In *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, pages 108–112.

[Ballesteros et al.2015] Miguel Ballesteros, Bernd Bohnet, Simon Mille, and Leo Wanner. 2015. Data-driven sentence generation with non-isomorphic trees. In *Mihalcea R, Chai J, Anoop S, editors. Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; 2015 May 31-June 5; Denver, Colorado, United States.[Stroudsburg]: ACL; 2015. p. 387-97.* ACL (Association for Computational Linguistics).

[Banarescu et al.2013] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In *Proc. of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*.

[Bangalore and Rambow2000] Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th conference on Computational linguistics*. Association for Computational Linguistics.

[Barzdins and Gosko2016] Guntis Barzdins and Didzis Gosko. 2016. RIGA at SemEval-2016 Task 8: Impact of Smatch Extensions and Character-Level Neural Translation on AMR Parsing Accuracy. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1143–1147.

[Basile et al.2012] Valerio Basile, Johan Bos, Kilian Evang, and Noortje Venhuizen. 2012. A platform for collaborative semantic annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics.

[Bateman1997] John A Bateman. 1997. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*, 3(1):15–55.

[Belz et al.2011] Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the 13th European workshop on natural language generation*, pages 217–226. Association for Computational Linguistics.

[Berant et al.2013] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

[Bjerva et al.2016] Johannes Bjerva, Johan Bos, and Hessel Haagsma. 2016. The meaning factory at semeval-2016 task 8: Producing amrs with boxer. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1179–1184.

[Böhmová et al.2003] Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2003. The Prague dependency treebank. In *Treebanks*, pages 103–127. Springer.

[Bohnet et al.2010] Bernd Bohnet, Leo Wanner, Simon Mille, and Alicia Burga. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 98–106. Association for Computational Linguistics.

[Bohnet et al.2011] Bernd Bohnet, Simon Mille, Benoît Favre, and Leo Wanner. 2011. < StuMaBa>: from deep representation to surface. In *Proceedings of the 13th European workshop on natural language generation*, pages 232–235. Association for Computational Linguistics.

[Bos2008] Johan Bos. 2008. Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 277–286. Association for Computational Linguistics.

[Brandt et al.2016] Lauritz Brandt, David Grimm, Mengfei Zhou, and Yannick Versley. 2016. ICL-HD at SemEval-2016 task 8: Meaning representation parsing-augmenting AMR parsing with a preposition semantic role labeling neural network. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1160–1166.

[Braune et al.2014] Fabienne Braune, Daniel Bauer, and Kevin Knight. 2014. Mapping Between English Strings and Reentrant Semantic Graphs. In *LREC*, pages 4493–4498. Citeseer.

[Butler2016] Alastair Butler. 2016. DynamicPower at SemEval-2016 Task 8: Processing syntactic parse trees with a Dynamic Semantics core. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1148–1153.

[Buys and Blunsom2017] Jan Buys and Phil Blunsom. 2017. Robust Incremental Neural Semantic Graph Parsing. pages 1215–1226.

[Cai and Knight2013] Shu Cai and Kevin Knight. 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)*, page 748–752. Association for Computational Linguistics, August.

[Carreras and Màrquez2005] Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the ninth conference on computational natural language learning*.

[Chu and Liu1965] Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14.

[Coles1968] L Stephen Coles. 1968. An on-line question-answering systems with natural language and pictorial input. In *Proceedings of the 1968 23rd ACM national conference*, pages 157–167. ACM.

[Collins2002] Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proc. of EMNLP*.

[Copestake and Flickinger2000] Ann A Copestake and Dan Flickinger. 2000. An Open Source Grammar Development Environment and Broad-coverage English Grammar Using HPSG. In *LREC*.

[Damonte et al.2017] Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An Incremental Parser for Abstract Meaning Representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 536–546.

[Darlington and Charney1963] J. L. Darlington and Elinor K. Charney. 1963. Mechanical Translation: Translating Ordinary Language into Functional Logic.

[Das et al.2012] Dipanjan Das, André F. T. Martins, and Noah A. Smith. 2012. An Exact Dual Decomposition Algorithm for Shallow Semantic Parsing with Constraints. In

Proceedings of the Joint Conference on Lexical and Computational Semantics.

[de Marneffe et al.2006] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *IN PROC. INT'L CONF. ON LANGUAGE RESOURCES AND EVALUATION (LREC)*.

[Deemter et al.2005] Kees Van Deemter, Mariët Theune, and Emiel Krahmer. 2005. Real versus Template-Based Natural Language Generation: A False Opposition? *Computational Linguistics*.

[Do et al.2009] Chuong B. Do, Quoc V. Le, Choon H Teo, Olivier Chapelle, and Alex J. Smola. 2009. Tighter bounds for structured estimation. In *NIPS*.

[Duchi et al.2011] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *JMLR*, 12.

[Dyer et al.2010] Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. 2010. cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. In *Proc. of ACL*.

[Edmonds1967] Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards, B*, 71:233–240.

[Flanigan et al.2014] Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A Discriminative Graph-Based Parser for the Abstract Meaning Representation. In *Proceedings of ACL*.

[Flanigan et al.2016a] Jeffrey Flanigan, Chris Dyer, Noah A Smith, and Jaime Carbonell. 2016a. 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.

[Flanigan et al.2016b] Jeffrey Flanigan, Chris Dyer, Noah A Smith, and Jaime Carbonell. 2016b. Generation from abstract meaning representation using tree transducers. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 731–739.

[Foland and Martin2016] William Foland and James H Martin. 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.

[Foland and Martin2017] William Foland and James H Martin. 2017. Abstract meaning representation parsing using lstm recurrent neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 463–472.

[Galley et al.2004] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proc. of HLT-NAACL*.

[Gatt and Reiter2009] Albert Gatt and Ehud Reiter. 2009. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 90–93. Association for Computational Linguistics.

[Gildea and Jurafsky2000] Daniel Gildea and Daniel Jurafsky. 2000. Automatic Labeling of Semantic Roles. In *38th Annual Meeting of the Association for Computational Linguistics, Hong Kong, China, October 1-8, 2000.*

[Gimpel and Smith2012] Kevin Gimpel and Noah A. Smith. 2012. Structured Ramp Loss Minimization for Machine Translation. In *Proc. of NAACL*.

[Goodman and Nirenburg1991] Kenneth Goodman and Sergei Nirenburg. 1991. *The KBMT Project: A case study in knowledge-based machine translation*. Elsevier.

[Goodman et al.2016a] James Goodman, Andreas Vlachos, and Jason Naradowsky.

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)*, volume 1, pages 1–11.

[Goodman et al.2016b] James Goodman, Andreas Vlachos, and Jason Naradowsky. 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.

[Goodman1999] Joshua Goodman. 1999. Semiring Parsing. *CL*, 25(4).

[Graehl and Knight2004] Jonathan Graehl and Kevin Knight. 2004. Training Tree Transducers. In *Proc. of HLT-NAACL*.

[Green1969] Cordell Green. 1969. Theorem proving by resolution as a basis for question-answering systems. *Machine intelligence*, 4:183–205.

[Gruzitis et al.2017] Normunds Gruzitis, Didzis Gosko, and Guntis Barzdins. 2017. RIGOTRIO at SemEval-2017 Task 9: Combining Machine Learning and Grammar Engineering for AMR Parsing and Generation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 924–928.

[Heafield et al.2013] Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable Modified Kneser-Ney Language Model Estimation. In *Proc. of ACL*.

[Hirschman et al.1989] Lynette Hirschman, Martha Palmer, John Dowding, Deborah Dahl, Marcia Linebarger, Rebecca Passonneau, F-M Land, Catherine Ball, and Carl Weir. 1989. The PUNDIT natural-language processing system. In *AI Systems in Government Conference, 1989., Proceedings of the Annual*, pages 234–243. IEEE.

[Huang et al.2006] Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical Syntax-Directed Translation with Extended Domain of Locality. In *Proc. of AMTA*.

[Ice1990] Vanilla Ice. 1990. Ice Ice Baby. In *To The Extreme*.

[Janssen and Limnios1999] Jacques Janssen and Nikolaos Limnios. 1999. *Semi-Markov Models and Applications*.

[Jones et al.2012] Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-Based Machine Translation with Hyperedge Replacement Grammars. In *Proc. of COLING*.

[Kasper1989] Robert T Kasper. 1989. A flexible interface for linking applications to Penman's sentence generator. In *Proceedings of the workshop on Speech and Natural Language*, pages 153–158. Association for Computational Linguistics.

[Keshet and McAllester2011] Joseph Keshet and David A McAllester. 2011. Generalization bounds and consistency for latent structural probit and ramp loss. In *Advances in Neural Information Processing Systems*.

[Klein and Manning2003] Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*.

[Klimeš2006a] Václav Klimeš. 2006a. *Analytical and Tectogrammatical Analysis of a Natural Language*. Ph.D. thesis, Charles University in Prague.

[Klimeš2006b] Václav Klimeš. 2006b. Transformation-based tectogrammatical analysis of czech. In *International Conference on Text, Speech and Dialogue*, pages 135–142. Springer.

[Klimeš2007] Václav Klimeš. 2007. Transformation-based tectogrammatical dependency analysis of english. In *International Conference on Text, Speech and Dialogue*, pages 15–22. Springer.

[Knight and Hatzivassiloglou1995] Kevin Knight and Vasileios Hatzivassiloglou. 1995. Two-level, many-paths generation. In *Proceedings of the 33rd annual meeting on As-*

sociation for Computational Linguistics, pages 252–260. Association for Computational Linguistics.

[Knight et al.2016] Kevin Knight, Ulf Hermjakob, Kira Griffitt, Martha Palmer, Claire Bonial, Tim O'Gorman, Nathan Schneider, Bianca Badarau, and Madalina Bardocz. 2016. DEFT Phase 2 AMR Annotation R2 LDC2016E25. Linguistic Data Consortium.

[Konstas et al.2017] Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-Sequence Models for Parsing and Generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 146–157.

[Kruskal1956] Joseph B. Kruskal. 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1).

[Lafferty et al.2001] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. ICML*.

[Lampouras and Vlachos2017] Gerasimos Lampouras and Andreas Vlachos. 2017. Sheffield at SemEval-2017 Task 9: Transition-based language generation from AMR. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 586–591.

[Langkilde and Knight1998] Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 704–710. Association for Computational Linguistics.

[Liang et al.2009] Percy Liang, Michael I Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 91–99. Association for Computational Linguistics.

[Liu et al.2015] Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 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, May–June. Association for Computational Linguistics.

[Lyu and Titov2018] Chunchuan Lyu and Ivan Titov. 2018. AMR Parsing as Graph Prediction with Latent Alignment. *arXiv preprint arXiv:1805.05286*.

[Mann1983] William C Mann. 1983. An overview of the Penman text generation system. In *AAAI*, pages 261–265.

[Martins et al.2009] André F. T. Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*.

[Martins et al.2011] André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2011. Dual Decomposition with Many Overlapping Components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, page 238–249. Association for Computational Linguistics.

[Martins et al.2013] André F. T. Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *ACL (2)*.

[May and Priyadarshi2017] Jonathan May and Jay Priyadarshi. 2017. Semeval-2017 task 9: Abstract meaning representation parsing and generation. In *Proceedings of the*

11th International Workshop on Semantic Evaluation (SemEval-2017), pages 536–545.

[Mcdonald and Pereira2006] Ryan Mcdonald and Fernando Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of EACL*, page 81–88.

[McDonald et al.2005] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*.

[McRoy et al.2000] Susan W McRoy, Songsak Channarukul, and Syed S Ali. 2000. YAG: A template-based generator for real-time systems. In *Proceedings of the first international conference on Natural language generation*. Association for Computational Linguistics.

[Melcuk1988] Igor Melcuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, NY, USA.

[Mille et al.2017] Simon Mille, Roberto Carlini, Alicia Burga, and Leo Wanner. 2017. FORGe at SemEval-2017 Task 9: Deep sentence generation based on a sequence of graph transducers. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 920–923.

[Miller et al.1994] Scott Miller, Robert Bobrow, Robert Ingria, and Richard Schwartz. 1994. Hidden understanding models of natural language. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 25–32. Association for Computational Linguistics.

[Miller et al.1996] Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 55–61. Association for Computational Linguistics.

[Misra and Artzi2016] Dipendra Kumar Misra and Yoav Artzi. 2016. Neural shift-reduce ccg semantic parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1775–1786.

[Mitamura et al.1991] Teruko Mitamura, Eric H Nyberg, and Jaime G Carbonell. 1991. An efficient interlingua translation system for multi-lingual document production. In *Proceedings of the Third Machine Translation Summit*.

[Mitra and Baral2016] Arindam Mitra and Chitta Baral. 2016. Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning. *AAAI*.

[Nguyen and Nguyen2017] Khoa Nguyen and Dang Nguyen. 2017. UIT-DANGNT-CLNLP at SemEval-2017 Task 9: Building Scientific Concept Fixing Patterns for Improving CAMR. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 909–913.

[Och2003] Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL*.

[Oepen et al.2014] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 Task 8: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 63–72.

[Oepen et al.2015] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresová. 2015. SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2015*, pages 915–926.

[Oh and Rudnicky2002] Alice H Oh and Alexander I Rudnicky. 2002. Stochastic natural language generation for spoken dialog systems. *Computer Speech & Language*, 16(3-4):387–407.

[Palmer et al.2005] Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *CL*, 31(1).

[Papineni et al.2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

[Peng and Gildea2016] Xiaochang Peng and Daniel Gildea. 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.

[Peng et al.2015] Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for AMR parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 32–41.

[Peng et al.2017] Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. 2017. Addressing the Data Sparsity Issue in Neural AMR Parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 366–375.

[Poon and Domingos2009] Hoifung Poon and Pedro Domingos. 2009. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 1–10. Association for Computational Linguistics.

[Pourdamghani et al.2016] Nima Pourdamghani, Kevin Knight, and Ulf Hermjakob.

2016. Generating english from abstract meaning representations. In *Proceedings of the 9th International Natural Language Generation conference*, pages 21–25.

[Prim1957] Robert C. Prim. 1957. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36:1389–1401.

[Pust et al.2015] Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 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.

[Puzikov et al.2016] Yevgeniy Puzikov, Daisuke Kawahara, and Sadao Kurohashi. 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.

[Quernheim and Knight2012a] Daniel Quernheim and Kevin Knight. 2012a. Dagger: A toolkit for automata on directed acyclic graphs. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 40–44.

[Quernheim and Knight2012b] Daniel Quernheim and Kevin Knight. 2012b. Towards probabilistic acceptors and transducers for feature structures. In *Proceedings of the Sixth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 76–85. Association for Computational Linguistics.

[Quillian1969] M Ross Quillian. 1969. The teachable language comprehender: A simulation program and theory of language. *Communications of the ACM*, 12(8):459–476.

[Rambow and Korelsky1992] Owen Rambow and Tanya Korelsky. 1992. Applied text generation. In *Proceedings of the third conference on Applied natural language processing*, pages 40–47. Association for Computational Linguistics.

[Ranta2011] Aarne Ranta. 2011. *Grammatical framework: Programming with multilingual grammars*. CSLI Publications, Center for the Study of Language and Information.

[Rao et al.2016] Sudha Rao, Yogarshi Vyas, Hal Daumé III, and Philip Resnik. 2016. CLIP @ UMD at SemEval-2016 Task 8: Parser for Abstract Meaning Representation using Learning to Search. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1190–1196.

[Raphael1964] Bertram Raphael. 1964. *SIR: A Computer Program for Semantic Information Retrieval*. Ph.D. thesis, Massachusetts Institute of Technology.

[Ratinov and Roth2009] Lev Ratinov and Dan Roth. 2009. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*.

[Ratnaparkhi2000] Adwait Ratnaparkhi. 2000. Trainable methods for surface natural language generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics*, pages 194–201. Association for Computational Linguistics.

[Reiter1994] Ehud Reiter. 1994. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation*, pages 163–170. Association for Computational Linguistics.

[Reiter2018] Ehud Reiter. 2018. A Structured Review of the Validity of BLEU. *Computational Linguistics*.

[Riedel and Clarke2006] Sebastian Riedel and James Clarke. 2006. Incremental Integer Linear Programming for Non-projective Dependency Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*.

[Riezler et al.2002] Stefan Riezler, Tracy H King, Ronald M Kaplan, Richard Crouch, John T Maxwell III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Pro-*

ceedings of the 40th Annual Meeting on Association for Computational Linguistics, pages 271–278. Association for Computational Linguistics.

[Rosenblatt1957] Frank Rosenblatt. 1957. The Perceptron—a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory.

[Rush and Collins2012] Alexander M. Rush and Michael Collins. 2012. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *J. Artif. Int. Res.*, 45(1).

[Schick2017] Timo Schick. 2017. Transition-Based Generation from Abstract Meaning Representations. Master's thesis.

[Smith and Eisner2006] David A Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of COLING/ACL*.

[Smith2011] Noah A Smith. 2011. Linguistic Structure Prediction. *Synthesis lectures on human language technologies*.

[Song et al.2016] Linfeng Song, Yue Zhang, Xiaochang Peng, Zhiguo Wang, and Daniel Gildea. 2016. AMR-to-text generation as a Traveling Salesman Problem. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2084–2089.

[Song et al.2017] Linfeng Song, Xiaochang Peng, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2017. AMR-to-text Generation with Synchronous Node Replacement Grammar. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 7–13.

[Surdeanu et al.2008] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*.

[Taskar et al.2003] Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003. Max-margin Markov networks. In *NIPS*.

[Thompson1977] Henry Thompson. 1977. Strategy and tactics: A model for language production. In *Regional Meeting. Chicago Ling. Soc. Chicago, Ill.*, volume 13, pages 651–668.

[Tsochantaridis et al.2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proc. of ICML*.

[van Noord and Bos2017a] Rik van Noord and Johan Bos. 2017a. Dealing with Co-reference in Neural Semantic Parsing. In *Proceedings of the 2nd Workshop on Semantic Deep Learning (SemDeep-2)*, pages 41–49.

[van Noord and Bos2017b] Rik van Noord and Johan Bos. 2017b. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *arXiv preprint arXiv:1705.09980*.

[Vanderwende et al.2015] Lucy Vanderwende, Arul Menezes, and Chris Quirk. 2015. An AMR parser for English, French, German, Spanish and Japanese and a new AMR-annotated corpus. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 26–30.

[Varges and Mellish2001] Sebastian Varges and Chris Mellish. 2001. Instance-based natural language generation. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8. Association for Computational Linguistics.

[Viet et al.2017] Lai Dac Viet, Nguyen Le Minh, and Ken Satoh. 2017. ConvAMR: Abstract meaning representation parsing. *arXiv preprint arXiv:1711.06141*.

[Wang and Xue2017] Chuan Wang and Nianwen Xue. 2017. Getting the Most out of

AMR Parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1257–1268.

[Wang et al.2015a] Chuan Wang, Nianwen Xue, and Sameer Pradhan. 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 (Volume 2: Short Papers)*, volume 2, pages 857–862.

[Wang et al.2015b] Chuan Wang, Nianwen Xue, and Sameer Pradhan. 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.

[Weizenbaum1966] Joseph Weizenbaum. 1966. ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

[Werling et al.2015] Keenon Werling, Gabor Angeli, and Christopher D. Manning. 2015. Robust Subgraph Generation Improves Abstract Meaning Representation Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 982–991.

[Wilks1973] Yorick Wilks. 1973. An Artificial Intelligence Approach to Machine Translation. In J.C. Shank and K.M. Colby, editors, *Computer Models of Thought and Language*, pages 114–151. W.H. Freeman and Company, San Franscisco.

[Wong and Mooney2006] Yuk Wah Wong and Raymond J Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the*

Association of Computational Linguistics, pages 439–446. Association for Computational Linguistics.

[Wong and Mooney2007] Yuk Wah Wong and Raymond Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 172–179.

[Woods1978] William A Woods. 1978. Semantics and quantification in natural language question answering. In *Advances in computers*, volume 17, pages 1–87. Elsevier.

[Yu and Joachims2009] Chun-Nam John Yu and Thorsten Joachims. 2009. Learning Structural SVMs with Latent Variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1169–1176, New York, NY, USA. ACM.

[Žabokrtský et al.2008] Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. 2008. TectoMT: Highly modular MT system with tectogrammatics used as transfer layer. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 167–170. Association for Computational Linguistics.

[Zaslavskiy et al.2009] Mikhail Zaslavskiy, Marc Dymetman, and Nicola Cancedda. 2009. Phrase-based statistical machine translation as a traveling salesman problem. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 333–341. Association for Computational Linguistics.

[Zelle and Mooney1996] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

[Zettlemoyer and Collins2005] Luke S. Zettlemoyer and Michael Collins. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, pages 658–666.

[Zhou et al.2016] Junsheng Zhou, Feiyu Xu, Hans Uszkoreit, QU Weiguang, Ran Li, and Yanhui Gu. 2016. AMR parsing with an incremental joint model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 680–689.