Universal Semantic Parsing with Neural Networks

Thesis submitted for the degree of

“Doctor of Philosophy”

By
Daniel Hershcovich

Submitted to the Senate of the Hebrew University
February, 2019
This work was carried out under the supervision of:

Prof. Ari Rappoport and Dr. Omri Abend
Acknowledgments

Even before I started my studies at the PhD program of the Edmond and Lily Safra Center for Brain Sciences (then the Interdisciplinary Center for Neural Computation), I was interested in human language in general and learning languages in particular, an interest I found early on that I shared with who became my (first) PhD advisor, Ari Rappoport, to whom I am grateful for the insightful conversations all along this process. I am of course also indebted to my second advisor, Omri Abend, whose shared interests with me and Ari have led to a fruitful research project, which is still ongoing and I hope will continue for many years. They have set a very high standard, to which I continuously strive and to which I surely owe much of my success so far. I also thank my PhD committee members, Roi Reichart and Yoav Goldberg, whose invaluable advice has shaped my research direction in the best way possible.

I sincerely appreciate the support of the ELSC staff and students, and particularly the members of my year, Adar Adamsky, Jonathan Bain, Haran Shani, Matan Holtzer, Henrike Horn, Adi Kol, Ori Lavi Rotbain, Nimrod Levin, Oren Peles, Pnina Rappel, Benjamin Weiner, and Noga Zaslavsky, whose company since the early stages of my graduate studies has given me a sense of belonging and comfort. I also thank the many other friends I made during my studies, including Nora Vrieler, Ana Polterovich, Johannes Niediek, Dan Valsky, Nova Fandina and Stav Yardeni, for their support.

My gratitude also goes to Roy Schwartz, who shared a great deal of his experience with me and helped me become confident in my ideas, and to Elior Sulem, who has accompanied me as a student in the lab and with whom I shared a considerable part of my time there. I would also like to thank past and current lab members, including Oren Tsur, Eva Kimel, Dana Rubinstein, Saggy Herman, Effi Levi, Leshem Choshen, Zohar Aizenbud, Aviram Stern, Adi Bitan and Michal Kessler, who have made these few years both pleasant and interesting. I must also express my gratitude towards Dotan Dvir and the UCCA annotation team, without which this work would simply have not been possible.

Finally, I owe my deepest gratitude to my family for their support along the way, and to my partner Michal for her endless love, patience and understanding, which I value greatly.
Abstract

A major scientific effort is dedicated to natural language understanding, which aims to be able to comprehend text, reason about it, and act upon it in an intelligent way. While specific use-cases or benchmarks can be solved with relatively simple systems, which either ignore word order (“bag-of-words” models) or treat it as a simple linear structure (such as the popular sequence-to-sequence framework allowing neural networks to learn tasks in an end-to-end fashion), understanding human language in general requires a hierarchical representation of meaning. Constructing this representation from text has been the goal of an extensive line of work in semantic parsing. While many semantic representation schemes have been proposed, they share many of their basic distinctions, such as between predicates (relations, states and events) and arguments (participants).

This thesis focuses on a particular semantic representation scheme called Universal Conceptual Cognitive Annotation (UCCA), whose main design principles are support for all major linguistic semantic phenomena, cross-linguistic applicability, stability across translations, ease of annotation (even by those who are not experts in linguistics), and a modular architecture supporting multiple layers of semantic annotation. A fully automatic parser is presented, and evaluated on multiple languages (English, French and German). The parser, titled “TUPA” (transition-based UCCA parser), is able to learn very general graph structures: directed acyclic graphs over token sequences with non-terminal nodes for complex units, where these may cover discontinuous terminal yields. This general class of graphs covers the structures annotated in UCCA, as well as other representation schemes. TUPA is implemented as a transition-based parser, whose transition system supports these structural properties. Its transition classifier is a neural network equipped with a bidirectional long short-term memory (BiLSTM) module for calculating feature representations for the input. In an extensive comparison to conversion-based methods, as well as other classifier implementations, TUPA is shown to outperform all baselines in the task of UCCA parsing in both in-domain and out-of-domain settings in three languages.

The parser is subsequently applied to two other semantic representation schemes, DM and AMR, and to syntactic dependencies in the Universal Dependencies (UD) scheme. This demonstrates that the flexible parser is usable not just for UCCA parsing. Furthermore, training TUPA in a multitask setting on all of these schemes improves its UCCA parsing accuracy, by effectively learning generalizations across the different representations: a shared model is thus able to apply semantic distinctions in one task, which have been learned for another.

Finally, in an empirical comparison of the content of semantic and syntactic represen-
tations, we discover several aspects of divergence, i.e., differences in the content captured by these schemes. These have profound impact on the potential contribution of syntax to semantic parsing, and on the usefulness of each of the approaches for semantic tasks in natural language processing.

I see semantic parsing as a means for computers to learn language. While different representations focus on different distinctions and do so with formally different structures, they share an overall goal, which is to support natural language processing applications, such as classifying text into categories, tagging it for linguistic properties, performing inference and reasoning, and generating new text according to some constraints (e.g., machine translation). The combined datasets annotated in every representation are an invaluable resource, which, used effectively, can greatly boost our achievements in language understanding and processing.
# Contents

1 Introduction .......................................................... 7
  1.1 Semantic Representations ........................................ 9
  1.2 Universal Conceptual Cognitive Annotation .................... 11
  1.3 UCCA Parsing ...................................................... 15

2 Methodology .......................................................... 17
  2.1 Transition-Based Parsing ........................................ 17
  2.2 Neural Networks .................................................. 20
  2.3 Multitask Learning ............................................... 23
  2.4 Evaluation ........................................................ 24

3 A Transition-Based Directed Acyclic Graph Parser for UCCA (Published in ACL 2017) ............................................. 26
  3.1 Introduction ....................................................... 27
  3.2 The UCCA Scheme ................................................ 28
  3.3 Transition-based UCCA Parsing .................................. 29
  3.4 Experimental Setup .............................................. 33
  3.5 Results ........................................................... 37
  3.6 Related Work ..................................................... 38
  3.7 Conclusion ......................................................... 39

4 Multitask Parsing Across Semantic Representations (Published in ACL 2018) .................................................. 41
  4.1 Introduction ....................................................... 42
  4.2 Related Work ..................................................... 42
  4.3 Tackled Parsing Tasks ............................................ 43
  4.4 General Transition-based DAG Parser ......................... 45
     4.4.1 TUPA’s Transition Set ..................................... 46
     4.4.2 Transition Classifier ....................................... 46
## 5 Universal Dependency Parsing with a General Transition-Based DAG

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>60</td>
</tr>
<tr>
<td>5.2 Unified DAG Format</td>
<td>61</td>
</tr>
<tr>
<td>5.3 General Transition-based DAG Parser</td>
<td>64</td>
</tr>
<tr>
<td>5.3.1 Transition Set</td>
<td>65</td>
</tr>
<tr>
<td>5.3.2 Transition Classifier</td>
<td>65</td>
</tr>
<tr>
<td>5.3.3 Constraints</td>
<td>67</td>
</tr>
<tr>
<td>5.4 Training details</td>
<td>68</td>
</tr>
<tr>
<td>5.4.1 Hyperparameters</td>
<td>68</td>
</tr>
<tr>
<td>5.4.2 Small Treebanks</td>
<td>68</td>
</tr>
<tr>
<td>5.4.3 Multilingual Model</td>
<td>69</td>
</tr>
<tr>
<td>5.4.4 Out-of-domain Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>5.5 Results</td>
<td>69</td>
</tr>
<tr>
<td>5.5.1 Evaluation on Enhanced Dependencies</td>
<td>70</td>
</tr>
<tr>
<td>5.5.2 Ablation Experiments</td>
<td>70</td>
</tr>
<tr>
<td>5.6 Conclusion</td>
<td>71</td>
</tr>
</tbody>
</table>

## 6 Content Differences in Syntactic and Semantic Representations (Published in NAACL-HLT 2019)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Introduction</td>
<td>75</td>
</tr>
<tr>
<td>6.2 Representations</td>
<td>76</td>
</tr>
<tr>
<td>6.3 Shared Gold-standard Corpus</td>
<td>78</td>
</tr>
<tr>
<td>6.4 Comparison Methodology</td>
<td>79</td>
</tr>
<tr>
<td>6.4.1 Basic Conversion</td>
<td>79</td>
</tr>
<tr>
<td>6.4.2 Extensions to the Converter</td>
<td>80</td>
</tr>
<tr>
<td>6.5 Analysis of Divergences</td>
<td>81</td>
</tr>
<tr>
<td>6.5.1 Confusion Matrix</td>
<td>81</td>
</tr>
<tr>
<td>6.5.2 Scenes vs. Non-Scenes</td>
<td>82</td>
</tr>
<tr>
<td>6.5.3 Primary and Secondary Relations</td>
<td>83</td>
</tr>
<tr>
<td>6.5.4 Multi-Word Expressions</td>
<td>84</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Natural language processing (NLP) or computational linguistics (CL) is a sub-field of computer science with two main goals: (1) developing methods for performing linguistic tasks automatically so that solutions can be engineered for better human-computer interface or analysis of large amounts of natural language data; (2) computational modeling and study of natural language to answer linguistic questions.

Applications in NLP include classifying text into categories (such as spam/non-spam, by topic, by author properties, or by the sentiment expressed in it), tagging it for linguistic properties (such as part-of-speech, i.e., noun/verb/etc.), building abstract representations for it (for linguistic research or for more readily integrating them into downstream applications) and generating new text according to some constraints (for example, machine translation is the task of generating text in a target language given source language text; text simplification is the task of generating simple text with approximately the same meaning as given complex text).

Semantic tasks in natural language processing, such as machine translation and sentiment analysis, require understanding the meaning of text. Since text is merely a sequence of words, it has to be represented in a way that will convey its meaning. One simple approach, known as the bag-of-words model, looks only at which words occur in the text. This can already provide substantial information about the meaning, but it ignores the order of words, which clearly conveys important information as well. The n-gram model counts sequences of words with regard to their order, incorporating at least some of the meaning encoded in the text structure. Many models treat text as a simple linear structure, such as the popular sequence-to-sequence framework allowing neural networks to learn tasks in an end-to-end fashion.

Specific use-cases or benchmarks can be solved with such relatively simple systems, which either ignore word order or make simplifying assumptions about structure. However, understanding human language in general requires a hierarchical representation of
meaning, as an undeniable part in the meaning of language resides in its hierarchical structure.

Indeed, a major effort in NLP is dedicated to natural language understanding, which aims to be able to comprehend text, reason about it, and act upon it in an intelligent way. For example, executable semantic parsing\footnote{Two conceptually distinct tasks are termed semantic parsing: parsing into executable representations, such as SQL; and parsing into descriptive meaning representation, such as AMR or UCCA. We disambiguate by referring to the former as executable semantic parsing.} is the task of generating logical meaning representations in the form of code, such as SQL or Python, which can then be executed against a knowledge base to perform tasks or answer questions. In general, constructing a hierarchical representation from text has been the goal of an extensive line of work in semantic parsing. While many semantic representation schemes have been proposed, they share many of their basic distinctions (Abend and Rappoport, 2017), such as between predicates (relations, states and events) and arguments (participants).

Syntax is a way to model this structure formally. Using syntactic features can improve the performance in semantic tasks. However, syntactic annotations suffer from limitations, since they do not represent the semantic structure of text directly. Simple manipulations such as switching from an active construction to a passive one, which nearly do not alter the meaning of text, can yield a significantly different syntactic structure. Moreover, the same syntactic construct can express conceptually distinct semantic structures: while “chairmen of parliaments” and “dozens of parliaments” have the same syntactic structure, semantically they are quite distinct.

Universal Dependencies (Nivre et al., 2016) is a syntactic dependency scheme aiming for cross-lingual consistency, whose annotation is often similar to the common practice in semantic treebanks (Figure 1.1). See Table C.2 for a concise description of UD relations.
1.1 Semantic Representations

As opposed to syntactic annotation, which reflects language-specific formal patterns, semantic annotation corresponds to a higher level of cognitive processing, and the same framework can potentially apply to any language. Moreover, a rich semantic annotation scheme may be better than syntactic annotation as an input for applications that attempt to solve a semantic task, due to their tighter relation to the meaning of the text. Ideally, a semantic representation abstracts away from detail that does not affect meaning:

\[
\begin{align*}
\text{rest} & \approx \text{take a break} \\
\text{graduation} & \approx \text{סימ חילוץ} \\
\end{align*}
\]

While syntactic description does not suffice to discern meaning from natural language, some syntactic formalisms provide a theory of the syntax-semantics interface. Examples include analyses using Head-driven Phrase Structure Grammar (HPHG; Pollard and Sag, 1994a), Combinatory Categorial Grammar (CCG; Steedman, 2000) and Tree-Adjoining Grammars (TAG; Joshi and Schabes, 1997). They have been used as a basis for meaning representation by defining compositional semantics on top of them: Flickinger (2000) introduced the LinGO English Resource Grammar (ERG), a broad-coverage, linguistically precise HPSG-based grammar of English, which is semantically grounded in Minimal Recursion Semantics (MRS; Copestake et al., 2005), a form of flat semantic representation capable of supporting underspecification.

Bos (2005, 2008, 2015) introduced the Boxer parser, which uses syntactic analysis in the form of CCG derivations to compositionally derive Discourse Representation Structures. Discourse Representation Theory (DRT; Kamp and Reyle, 2013) is a general framework for representing the meaning of sentences and discourse, which can handle multiple linguistic phenomena including anaphora, presuppositions, and temporal expressions. The basic meaning-carrying units in DRT are Discourse Representation Structures (DRSs), which are recursive formal meaning structures that have a model-theoretic interpretation and can be translated into first-order logic. Basic DRSs consist of discourse referents representing entities in the discourse, and discourse conditions representing information about discourse referents.

In semantic role labeling (Baker et al., 1998; Palmer et al., 2005), predicates and their arguments are annotated and classified into specific roles. While earlier work on semantic parsing has mostly concentrated on shallow semantic analysis, focusing on semantic role labeling of verbal argument structures, the focus has shifted to parsing of more elaborate representations that account for a wider range of phenomena. Most closely related to my work are Broad-Coverage Semantic Dependency Parsing (SDP) frameworks, such DM
While driving, I listen to podcasts.

Figure 1.2: Top: syntactic dependencies in the Universal Dependencies framework. Bottom: semantic dependencies in the DM (DELPH-IN MRS) framework.

(DELPH-IN MRS), converted from DeepBank \cite{Flickinger2012}, a corpus of hand-corrected parses from LinGO ERG. It addresses a wide range of semantic phenomena, and supports discontinuous units and multiple parents \cite{Oepen2016}. However, SDP uses bi-lexical dependencies (meaning every relation or edge is between a pair of words), disallowing non-terminal nodes (see below), and thus faces difficulties in supporting structures that have no clear head, such as coordination \cite{Ivanova2012}. Figure 1.2 example shows syntactic/semantic bi-lexical dependencies for the sentence “While driving, I listen to podcasts”.

Another line of work addresses parsing into Abstract Meaning Representations (AMRs; \cite{Banarescu2013,Flanigan2014,Vanderwende2015,Pust2015,Artzi2015,Wang2015,Wang2016,Wang2015a,Zhou2016,Damonte2017,Damonte2018}). AMR represents a rich set of semantic distinctions in a single homogeneous formalism, including named entity recognition and linking, semantic role labeling, and (indirectly) coreference resolution. While sharing much of my work’s motivation, AMR does not ground its units in the words and constituents of the text. This complicates the parsing task, as it requires that the alignment between words and logical symbols be automatically (and imprecisely) detected. Furthermore, it complicates applications using the meaning representation, as it is not immediately clear what portion of the text corresponds to what portion of the meaning representation, which may be necessary for tasks such as rephrasing, translation or evaluation of parts of the text.

While many of the parsing approaches mentioned above rely on a theory of syntax-semantics interface, on syntactic features or on syntactic pre-processing for semantic parsing, some semantic annotation schemes and parsing approaches attempt to represent the meaning of natural language utterances directly. While the parser presented in this thesis does make use of syntactic features, it largely belongs to this category, as it makes
Universal Conceptual Cognitive Annotation

Universal Cognitive Conceptual Annotation (UCCA; Abend and Rappoport, 2013) is a cross-linguistically applicable semantic representation scheme. It covers the predicate-argument structures evoked by predicates of all grammatical categories, the inter-relations between them, as well as other major linguistic phenomena. UCCA has demonstrated applicability to multiple languages, rapid annotation (Abend et al., 2017), and benefit for various applications: evaluation of machine translation (Birch et al., 2016), evaluation of grammatical error correction (Choshen and Abend, 2018), as demonstrated in Figure 1.4, evaluation of structural text simplification (Sulem et al., 2018a) and text simplification (Sulem et al., 2018b).

UCCA is a typologically-motivated scheme for analyzing abstract semantic structures
Table 1.1: UCCA data statistics.

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>20K</th>
<th>EWT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>en</td>
<td>fr</td>
<td>de</td>
</tr>
<tr>
<td># sentences</td>
<td>5,141</td>
<td>492</td>
<td>492</td>
</tr>
<tr>
<td># tokens</td>
<td>158,739</td>
<td>12,638</td>
<td>13,021</td>
</tr>
<tr>
<td># non-terminal nodes</td>
<td>62,002</td>
<td>4,699</td>
<td>5,110</td>
</tr>
<tr>
<td>% discontinuous</td>
<td>1.71</td>
<td>3.19</td>
<td>4.64</td>
</tr>
<tr>
<td>% reentrant</td>
<td>1.84</td>
<td>0.89</td>
<td>0.65</td>
</tr>
<tr>
<td># edges</td>
<td>208,937</td>
<td>16,803</td>
<td>17,520</td>
</tr>
<tr>
<td>% primary</td>
<td>97.40</td>
<td>96.79</td>
<td>97.02</td>
</tr>
<tr>
<td>% remote</td>
<td>2.60</td>
<td>3.21</td>
<td>2.98</td>
</tr>
</tbody>
</table>

in text. It aims to abstract away from grammatical particularities of a passage such that paraphrases, and translations tend to have similar UCCA structures. Accordingly, UCCA has been studied with respect to meaning preservation in translation (Sulem et al., 2015), and found to be more stable than phrase-structure syntactic annotation. UCCA corpora are available for English, French and German, and pilot studies were conducted on a few languages more. In English, data was annotated from Wikipedia, the book *Twenty Thousand Leagues under the Sea*, and the English Web Treebank Web Reviews corpus (see Table 1.1).

Figure 1.5: UCCA example including an implicit node, where the subject of “Glad” is omitted.

Formally, UCCA structures are directed acyclic graphs (DAGs) whose nodes (or units) correspond to (are anchored by) words, or elements viewed as a single entity according to some semantic or cognitive consideration. Edges are labeled with categories, indicating the role of a child in the relation the parent represents. See Table C.1 for a concise description of UCCA categories. A Scene is UCCA’s notion of an event or a frame, and is a description of a movement, an action or a state which persists in time. Every Scene contains one primary relation, which can be either a Process or a State. Scenes may

---

https://github.com/UniversalConceptualCognitiveAnnotation
contain any number of Participants, a category which also includes abstract participants and locations. They may also contain temporal relations (Time), and secondary relations (Adverbials), which cover semantic distinctions such as manner, modality and aspect. UCCA also supports implicit units which do not correspond to any tokens, such as the implicit semantic subject of “Glad” in Figure 1.5. The principal kind of unit is a scene denoting a situation mentioned in the sentence, typically involving a scene-evoking predicate, participants, and (perhaps) modifiers.

Scenes may be linked to one another in several ways. First, a Scene can provide information about some entity, in which case it is marked as an Elaborator. This often occurs in the case of participles or relative clauses. For example, “(child) who went to school” is an Elaborator Scene in “The child who went to school is John”. A Scene may also be a Participant in another Scene. For example, “John went to school” in the sentence: “He said John went to school”. In other cases, Scenes are annotated as Parallel Scenes (H), which are flat structures and may include a Linker (L), as in: “When L [he arrives] H, [he will call them] H”.

Non-Scene units are headed by units of the category Center, denoting the type of entity or thing described by the whole unit. Elements in non-Scene units include Quantifiers (such as “dozens of people”) and Connectors (mostly coordinating conjunctions). Other modifiers to the Center are marked as Elaborators.

Figure 1.5 contains five scenes: one anchored by the State Glad; one anchored by the Process called; one anchored by the Process arrived; one anchored by the Process ship; and one anchored by the possessive pronoun my, which indicates a stative possession relation. A Participant (A) of a scene is typically an entity or location involved. Adverbials (D) modify scenes with respect to properties like negation, modality, causativity, direction, manner, etc., which do not constitute an independent situation or entity. Temporal modifiers are labeled Time (T).

A scene can serve as a Participant within a larger scene: Figure 1.5 embeds the “called” scene within the “Glad” scene. A scene can also serve as an Elaborator of a non-scene unit: “my” and “to ship” in Figure 1.5 are both Elaborators of “box”. Other relations between scenes are called parallel linkage: a unit consists of Parallel Scenes (H) and possibly Linkers (L) describing how they are related. This is seen at the top level of Figure 1.5 where the “Glad” and “arrived” scenes are parallel and the word “before” is a linker.

Other categories only apply under non-scene units, i.e., units with no predicate: a semantic head—the Center (C); modifiers of Quantity (Q); and other modifiers, called Elaborators (E). A Connector (N) groups together multiple non-scene units into a larger unit, e.g., “You and I” in Figure 1.6a.
Apart from the main semantic content of scenes, participants, and connectives, UCCA provides the categories: Relator (R) for grammatical markers expressing how a unit relates to its parent unit—in English, these are mainly prepositions and the possessive ’s; Function (F) for other grammatical markers with minimal semantic content, such as tense auxiliaries, light verbs, and articles; and Ground (G) for expressions expressing speaker perspective outside the propositional structure of the sentence. The least contentful elements—Fs, and to a lesser extent Rs—are subject to considerable variation when paraphrasing or translating a sentence. Punctuation tokens are attached to units as U, but are generally also regarded as non-content-bearing.

Note that in Figure 1.6b, the tokens of the unit called off do not receive individual labels: this is called an unanalyzable unit, and is used for semantically opaque multi-word expressions.

The solid edges in the UCCA graphs are called primary edges: the primary edges in UCCA always form a tree. Remote edges, such as the dotted edge from the possession scene unit to “box” in Figure 1.5 and the dotted A edge to “Everything” in Figure 1.7, allow for additional relations, forming a DAG: multiple parents are enabled exclusively by remote edges. Remote edges are commonly used with attributive possessives and
adjectives, and relative clauses, where the head of the noun phrase doubles as a semantic participant of the modifier scene. They also figure in multi-scene control constructions (John told Mary to leave: Mary is a primary participant of the telling scene and a remote participant of the leaving scene).

1.3 UCCA Parsing

In order to represent the full range of semantic structures exhibited by natural language, three properties should be supported: reentrancy, representing arguments shared between predicates; non-terminal nodes for multi-word units; and discontinuity of semantic units in the text.

The first property, reentrancy (or multiple parents), is required for representing arguments and relations (semantic units) that are shared between predicates. For instance, in the sentence “Everything is delicious and cooked perfectly” (Figure 1.7), “Everything” is an argument of both “delicious” and “cooked perfectly”, yielding a DAG structure rather than a tree.

The second is non-terminal nodes for representing units comprising more than one word (Figure 1.6a). While bi-lexical dependencies partially circumvent this requirement, by representing complex units in terms of their headwords, they fall short when representing units that have no clear head. Frequent examples of such constructions include coordination structures (e.g., “You and I will change the world”), some multi-word expressions (e.g., “The Haves and the Have Nots”), and prepositional phrases. In these cases, dependency schemes often apply some annotation convention that selects one of the sub-units as the head, but as different head selections are needed for different purposes, standardization problems arise (Ivanova et al., 2012). For example, selecting the preposition to head prepositional phrases yields better parsing results (Schwartz et al., 2012), while the head noun can be more useful for information extraction purposes.

Third, semantic units may be discontinuous in the text. For instance, in “George called the meeting off” (Figure 1.6b), the phrasal verb “called ... off” forms a single semantic unit. Discontinuities are also pervasive with other multi-word expressions (Schneider et al., 2014).

The only semantic annotation scheme that has units anchored in the text and supports the combination of these criteria is UCCA, for which no parser has existed prior to the work presented here. This thesis is concerned with learning to parse UCCA graphs from text, representing its semantics. A general transition-based graph parser is introduced (see Section 2.1), and the relationship with other representations is investigated to find beneficial commonalities and differences highlighting the potential utility of semantic
parsers for text understanding applications. The analysis also exposes challenges semantic parsers must address, and potential sources for improvement.

This thesis has the following goals: developing techniques for general graph parsing, and specifically, devising a method for automatic prediction of UCCA structure given plain text; investigating and quantifying the relationship between the content captured in UCCA and other semantic or syntactic representations; and taking advantage of the similarities to other schemes, to improve UCCA parsing by learning common distinctions.

An automatic UCCA parser will allow large-scale annotation of text for meaning representation, in turn allowing addressing various scientific questions in semantics, such as regarding compositionality of meaning and cross-lingual divergences. It is also useful from an engineering perspective, allowing fully automatic evaluation of machine translation, grammatical error correction and text simplification, for example, as demonstrated in works cited in Section 1.2 as well as serving as an infrastructure for feature extraction and calculation of neural representation for natural language processing and understanding (see Section 2.2).

An extensive comparison and integration of meaning representation schemes is important for several reasons. Learning architectures that utilize complementary knowledge sources (e.g., via parameter sharing) are effective in that they take advantage of more annotated data and save annotation efforts. Learning from multiple flavors of meaning representation in tandem has hardly been explored, but it provides complementary views of the semantics of natural language, as well as regularization preventing overfitting to particular annotation choices or domains. Cross-framework parsing reduces framework-specific segregation in the field of meaning representation parsing, and is a step towards a unifying formal model over different semantic graph banks, uniform representations and scoring, systematic contrastive evaluation across frameworks, and increased cross-fertilization via transfer and multi-task learning (see Section 4.6).
Chapter 2

Methodology

2.1 Transition-Based Parsing

Various parsing methods have been used for various frameworks, depending on their formal properties. For bi-lexical dependency trees, which have become the most commonly used formalism for syntactic structure representation (Universal Dependencies, for example, is a multilingual bi-lexical syntactic dependency tree framework), the main parsing approaches can be characterized as graph-based and transition-based.

Transition-based parsers (Nivre, 2003, 2008) build trees or graphs as they scan the text incrementally. The parse is created by applying a transition at each step to the parser state, defined using a buffer of tokens and nodes to be processed, a stack of nodes currently being processed, and a graph of constructed nodes and labeled edges. A classifier is used at each step to select the next transition based on features that encode the parser’s current state. During training, an oracle creates training instances for the classifier, based on the gold-standard annotation. Transition-based parsers are also called shift-reduce parsers, as the most common transitions include Shift, moving a node from the buffer to the stack, and Reduce (-left/-right), possibly attaching nodes and removing them from the stack.

Despite being based on local decisions, transition-based methods have yielded excellent results in a variety of parsing tasks. In fact, the transition-based approach has produced some of the best results in syntactic dependency parsing (Dyer et al., 2015; Ballesteros et al., 2015; Kiperwasser and Goldberg, 2016; Andor et al., 2016), including non-projective dependency parsing (Nivre, 2009; Kuhlmann and Nivre, 2010; Bohnet and Nivre, 2012). Transition-based parsers have also demonstrated strong performance in a variety of other semantic and syntactic settings. Within syntactic dependency parsing, transition-based methods have been successfully applied to corpora in many languages and domains, including Universal Dependencies (Straka et al., 2016; Straka and Straková, 2017).
They thought about taking a short break.

Figure 2.1: UCCA graph for the sentences “They thought about taking a short break”.

The approach has also yielded results comparable with the state-of-the-art when applied to constituency parsing (Sagae and Lavie, 2005; Zhang and Clark, 2009; Zhu et al., 2013), and has been extended to discontinuous constituency parsing (Maier, 2015; Maier and Lichte, 2016) yielding improvements in the parsing of discontinuous constituents in German. Transition-based parsers have also been developed for dependency DAG structures (Sagae and Tsujii, 2008; Tokgöz and Eryiğit, 2015) and CCG parsing (Ambati et al., 2015).

The parser presented in this thesis, TUPA, combines techniques from transition-based discontinuous constituency parsing (Maier, 2015) and transition-based dependency DAG parsing (Tokgöz and Eryiğit, 2015). Transition-based methods are a natural starting point for UCCA parsing, as the set of distinctions it represents is similar in spirit to the distinctions conveyed by dependency schemes.

Wang et al. (2015b,a, 2016); Goodman et al. (2016); Wang and Xue (2017) presented a transition-based AMR parser, which requires a syntactic dependency tree as input. It operates on the input tree, transforming it into an AMR graph by a sequence of transitions. The accuracy of the underlying syntactic dependency parser is important, as shown by Wang et al. (2015a), who achieved the best results using the Charniak parser trained on a much larger and more diverse dataset—the full OntoNotes corpus, rather than the Stanford parser trained on the Penn TreeBank. Some transition-based AMR parsers parse to AMR directly, without going through a dependency tree (Zhou et al., 2016; Ballesteros and Al-Onaizan, 2017; Damonte et al., 2017; Damonte and Cohen, 2018). Their transition systems are in principle similar to TUPA and support reentrancies, but are tailored to AMR parsing and rely on alignments between concepts and text tokens.

The set of transitions implemented by TUPA is:

\{\text{Shift}, \text{Reduce}, \text{Node}_X, \text{Left-Edge}_X, \text{Right-Edge}_X, \text{Left-Remote}_X, \text{Right-Remote}_X, \text{Swap}, \text{Finish}\}

These transitions enable non-terminal nodes, reentrancy and discontinuity. To parse
the sentence “They thought about taking a short break” (Figure 2.1), an oracle parser would start from the initial state depicted in Figure 2.2 and subsequently apply the following transition sequence:

**Shift, Right-Edge \(A \), Shift, Swap, Right-Edge \(P \), Reduce, Shift, Shift, Node \(R \), Reduce, Left-Remote \(A \), Shift, Shift, Node \(C \), Reduce, Shift, Right-Edge \(P \), Shift, Right-Edge \(F \), Reduce, Shift, Swap, Right-Edge \(D \), Reduce, Swap, Right-Edge \(A \), Reduce, Reduce, Shift, Reduce, Shift, Right-Edge \(C \), Finish**

A few intermediate steps from this sequence are shown in Figure 2.3. Using supervised learning, TUPA would learn to mimic this oracle transition sequence by training a neural network classifier with a cross-entropy loss (see Section 2.2) to produce the correct transition at each point in the transition sequence, given the current state of the stack, the buffer and the graph.
2.2 Neural Networks

Neural networks are powerful machine learning models. They have yielded state-of-the-art results in many fields, including natural language processing (Goldberg, 2016). Inspired by the brain’s computation mechanism, artificial neural networks operate on dense input representations by a combination of linear and non-linear transformations, organized in several layers and trained end-to-end (deep learning). Deep learning and artificial neural networks have received much attention in several fields of computer science, including computer vision, speech recognition and natural language processing, due to their best performance in various tasks (Collobert et al., 2011). In natural language processing, many of the deep learning methods rely strongly on distributed representation.

A common approach to meaning representation is the representation of words as vectors in a continuous vector space with tens or hundreds of dimensions (Turian et al., 2010), such that linguistic and semantic regularities between the words are captured in the vectors (Mikolov et al., 2013b). This kind of representation, called word embedding, can be learned in an unsupervised manner, from large unlabeled corpora. Several methods have been developed for generalizing these models to create embeddings for multi-word phrases, sentences, and even complete documents. These provides complementary semantic information about the text, readily used in many machine learning techniques that operate on vectors of numbers. The more accurately these vectors represent the meaning of the text, the better semantic tasks can be solved using them. However, composing these representation so as to maintain the semantic structure remains a challenge.

Language data is commonly manifested as sequences (e.g., sentences are sequences of words). Recurrent neural networks (Elman, 1990) allow representing arbitrarily sized sequences in a fixed-size vector, without ignoring the structured properties of the input. A specific flavor called Long Short Term Memory (LSTM) is very common as it learns relatively long-term dependencies. A bidirectional recurrent neural network, such as a BiLSTM, takes into account both the past and future (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997; Graves, 2008; Irsoy and Cardie, 2014).

Recurrent neural networks can learn semantically meaningful continuous vector representations of multi-word phrases and sentences, typically in the same space as the word embedding, and they have a relatively simple model that does not depend on pre-annotation of structure: the network state at each time step is a function of the input and the state at the previous time step:

\[ h_t = f(x_t, h_{t-1}) \]

Where \( f \) could be as simple as matrix multiplication followed by a non-linearity, or a more
A complicated function such as an LSTM (Long-Short Term Memory) cell, which supports additive gated updates for alleviating vanishing gradients:

\[
\begin{align*}
c_j &= c_{j-1} \odot f + g \odot i \\
h_j &= \tanh(c_j) \odot o \\
    i &= \sigma(x_j W^x i + h_{j-1} W^h i) \\
f &= \sigma(x_j W^x f + h_{j-1} W^h f) \\
o &= \sigma(x_j W^x o + h_{j-1} W^h o) \\
g &= \tanh(x_j W^x g + h_{j-1} W^h g)
\end{align*}
\]

A bidirectional LSTM (BiLSTM), or in general, a bidirectional recurrent neural network (BiRNN), applies this function to the input sequence both in the original direction and in reverse, then concatenating the resulting hidden states for each position in the sequence, and potentially applying the same operation again in multiple layers (see Figure 2.4).

Typically, a feedforward neural network (MLP; multilayer perceptron) is applied on top of these representations, and a softmax classifier defines a probability distribution based on the scores for each label:

\[
[\text{softmax}(x)]_i = \frac{e^{x_i}}{\sum_j e^{x_j}}
\]

Recursive neural networks can also learn phrase representations, taking advantage of a pre-annotated hierarchical representation (typically syntactic trees) for compositionality (Socher et al., 2010). These models can learn a representation that facilitates reasoning and inference (Bowman et al., 2014). A recursive neural network has a state at each node.
of the structure: if $C_i$ denotes the set of children of node $i$,

$$h_i = g(\{ f(c) \mid c \in C_i \})$$

where $g$ is an aggregation function, such as concatenation, addition or max. It can also be a variant of the LSTM cell (Tai et al. 2015).

A recursive neural network can also be used for structure prediction, or parsing (Socher et al. 2013; Dyer et al. 2015): as the parse is being created, the model can be used to calculate the intermediate representation at each of the nodes already constructed, and to make subsequent parsing decisions based upon it. However, doing so efficiently requires specialized data structures (Bowman et al. 2016), and is not suitable for predicting complex structures such as UCCA graphs.

![Recursive neural network](image)

Figure 2.5: A recurrent neural network (a) combines distributed representations along a linear structure, whereas a recursive neural network (b) does so along a pre-determined hierarchical structure.

Neural networks are typically trained by gradient descent, where the gradient for the network parameters is calculated using backpropagation. The gradients for training a recursive neural network are calculated by backpropagation through structure, another variant of backpropagation, using a recursive “folding” architecture that can represent a general tree or a directed acyclic graph (DAG) structure (Goller and Kuchler 1996).

Training neural predictors for independent or sequential predictions is often done with the categorical cross-entropy loss (also referred to as negative log likelihood):

$$L(\hat{y}, y) = -\sum_i y_i \log(\hat{y}_i)$$

It measures the dissimilarity between the true label distribution $y$ and the predicted label distribution $\hat{y}$. This is the loss function we use for training the transition classifier in TUPA, given oracle transitions from an input gold graph.
Neural networks-based *machine translation* provides some sort of an intermediate encoding in the form of distributed representation that can be encoded from the source language and then decoded into the target language \cite{Zou2013}, or using memory and treating the text as a sequence to be converted to another sequence \cite{Sutskever2014}, using the popular sequence-to-sequence approach. However, the encoding is based just on averaging across words in the source sentence, on a flat sequence representation, or at best on a syntactic representation. A semantic representation like a UCCA graph would perhaps be a better candidate for the structure by which the encoding and decoding is performed.

While for simple classification tasks such as Textual Entailment or Natural Language Inference \cite{Dagan2005, Bowman2015a}, and to some degree for tasks with a complex output, such as machine translation, too, simple end-to-end neural networks can go a long way \cite{Bowman2015b} among others, combinatorial generalization in learning inevitably requires an inductive bias on the hypothesis class, realized as the network architecture \cite{Mitchell1980, Battaglia2018}. Specifically, scalable learning of meaning and inference requires an inductive bias on the compositional structure of language.

![Diagram](image)

Perhaps just as critical as the successful prediction of UCCA structure, if not even more critical for the NLP community, is the distributed representation created when forming phrases using this structure. Current methods in NLP form multi-word representation based on averaging across words or on syntax, which may be sub-optimal in representing the true meaning of text. Using a more semantically faithful way to compose words, such as UCCA, may be a key factor in enabling computers to understand natural language.

### 2.3 Multitask Learning

Multitask learning \cite{Caruana1997} allows exploiting the overlap between tasks to effectively extend the training data, and has greatly advanced with neural networks and representation learning. It has been used over the years for NLP tasks with varying degrees of similarity. Joint and multitask learning are often used when two or more
tasks share common information. These techniques have been used in NLP with tasks of varying degrees of similarity—as regularization, or to enlarge the effective training data.

In semantic role labeling, Toutanova et al. (2005) learned a joint re-reranking log-linear model across arguments of the same predicate. Ammar et al. (2016); Guo et al. (2016) applied multitask learning to transition-based syntactic parsing in a multilingual setting. Joint learning is also often used as an alternative to the pipeline approach, alleviating error propagation: In transition-based parsing, multitask learning has also been applied to tagging and parsing (Bohnet and Nivre, 2012; Zhang and Weiss, 2016), lexical and syntactic analysis (Constant and Nivre, 2016; More, 2016), and semantic-syntactic analysis (Swayamdipta et al., 2016a; Henderson et al., 2013).

Neural multitask learning has mostly been effective in tackling formally similar tasks (Søgaard and Goldberg, 2016), including multilingual semantic parsing (Duong et al., 2017), and cross-domain semantic parsing (Herzig and Berant, 2017; Fan et al., 2017). Sharing parameters with a low-level task has shown great benefit for transition-based syntactic parsing (Bohnet and Nivre, 2012; Zhang and Weiss, 2016; Constant and Nivre, 2016; More, 2016). State-of-the-art results in multiple NLP tasks have been achieved by jointly learning the tasks forming the NLP standard pipeline using a single neural model (Collobert et al., 2011; Hashimoto et al., 2017), thereby avoiding cascading errors, common in pipelines.

Much effort has been devoted to joint learning of syntactic and semantic parsing, including two CoNLL shared tasks (Surdeanu et al., 2008; Hajič et al., 2009) on joint syntactic parsing and semantic role labeling. Despite their conceptual and practical appeal, such joint models rarely outperform the pipeline approach of basing semantic parsing on the output of syntactic parsers (Lluís and Màrquez, 2008; Henderson et al., 2013; Lewis et al., 2015; Swayamdipta et al., 2016a, 2017).

This thesis revisits the idea of multitask learning for transition-based parsing with neural networks, combining an unprecedented diverse set of semantic parsing tasks and learning them with a uniform model. While many multitask architectures exist and have different advantages, the approach taken here is hard parameter sharing in the encoder layers of a neural network used for classification.

2.4 Evaluation

Comparing UCCA structures \( G_p = (V_p, E_p, \ell_p) \) and \( G_g = (V_g, E_g, \ell_g) \), over the same sequence of terminals \( W = \{w_1, \ldots, w_n\} \) is done as follows. For an edge \( e = (u, v) \) in either graph, its yield \( y(e) \subseteq W \) is the set of terminals in \( W \) that are descendants of \( v \).
Define the set of *mutual edges* between $G_p$ and $G_g$:

\[
M(G_p, G_g) = \{ (e_1, e_2) \in E_p \times E_g \mid y(e_1) = y(e_2) \land \ell_p(e_1) = \ell_g(e_2) \}
\]

Labeled precision and recall are defined by dividing $|M(G_p, G_g)|$ by $|E_p|$ and $|E_g|$, respectively, and F-score by taking their harmonic mean. Two variants are reported: one where we consider only primary edges, and another for remote edges. In all cases, punctuation units (U) are excluded from the evaluation.
Chapter 3

A Transition-Based Directed Acyclic
Graph Parser for UCCA (Published
in ACL 2017)

Daniel Hershcovich\textsuperscript{1,2}, Omri Abend\textsuperscript{2} and Ari Rappoport\textsuperscript{2}
\textsuperscript{1}The Edmond and Lily Safra Center for Brain Sciences
\textsuperscript{2}School of Computer Science and Engineering
Hebrew University of Jerusalem
\{danielh, oabend, arir\}@cs.huji.ac.il

Abstract

We present the first parser for UCCA, a cross-linguistically applicable framework for semantic representation, which builds on extensive typological work and supports rapid annotation. UCCA poses a challenge for existing parsing techniques, as it exhibits reentrancy (resulting in DAG structures), discontinuous structures and non-terminal nodes corresponding to complex semantic units. To our knowledge, the conjunction of these formal properties is not supported by any existing parser. Our transition-based parser, which uses a novel transition set and features based on bidirectional LSTMs, has value not just for UCCA parsing: its ability to handle more general graph structures can inform the development of parsers for other semantic DAG structures, and in languages that frequently use discontinuous structures.
3.1 Introduction

Universal Conceptual Cognitive Annotation (UCCA, Abend and Rappoport, 2013) is a cross-linguistically applicable semantic representation scheme, building on the established Basic Linguistic Theory typological framework (Dixon, 2010a,b, 2012), and Cognitive Linguistics literature (Croft and Cruse, 2004). It has demonstrated applicability to multiple languages, including English, French, German and Czech, support for rapid annotation by non-experts (assisted by an accessible annotation interface (Abend et al., 2017)), and stability under translation (Sulem et al., 2015). It has also proven useful for machine translation evaluation (Birch et al., 2016). UCCA differs from syntactic schemes in terms of content and formal structure. It exhibits reentrancy, discontinuous nodes and non-terminals, which no single existing parser supports. Lacking a parser, UCCA’s applicability has been so far limited, a gap this work addresses.

We present the first UCCA parser, TUPA (Transition-based UCCA Parser), building on recent advances in discontinuous constituency and dependency graph parsing, and further introducing novel transitions and features for UCCA. Transition-based techniques are a natural starting point for UCCA parsing, given the conceptual similarity of UCCA’s distinctions, centered around predicate-argument structures, to distinctions expressed by dependency schemes, and the achievements of transition-based methods in dependency parsing (Dyer et al., 2015; Andor et al., 2016; Kiperwasser and Goldberg, 2016). We are further motivated by the strength of transition-based methods in related tasks, including dependency graph parsing (Sagae and Tsujii, 2008; Ribeyre et al., 2014; Tokgoz and Eryigit, 2015), constituency parsing (Sagae and Lavie, 2005; Zhang and Clark, 2009; Zhu et al., 2013; Maier, 2015; Maier and Lichte, 2016), AMR parsing (Wang et al., 2015a,b, 2016; Misra and Artzi, 2016; Goodman et al., 2016; Zhou et al., 2016; Damonte et al., 2017) and CCG parsing (Zhang and Clark, 2011; Ambati et al., 2015, 2016).

We evaluate TUPA on the English UCCA corpora, including in-domain and out-of-domain settings. To assess the ability of existing parsers to tackle the task, we develop a conversion procedure from UCCA to bilexical graphs and trees. Results show superior performance for TUPA, demonstrating the effectiveness of the presented approach.

The rest of the paper is structured as follows: Section 3.2 describes UCCA in more detail. Section 3.3 introduces TUPA. Section 3.4 discusses the data and experimental setup. Section 3.5 presents the experimental results. Section 3.6 summarizes related work, and Section 3.7 concludes the paper.

\[\text{All parsing and conversion code, as well as trained parser models, are available at https://github.com/danielhers/tupa}\]
After graduation, John moved to Paris.

John gave everything up.

John and Mary’s trip home.

Figure 3.1: UCCA structures demonstrating three structural properties exhibited by the scheme. (a) includes a remote edge (dashed), resulting in “John” having two parents. (b) includes a discontinuous unit (“gave ... up”). (c) includes a coordination construction (“John and Mary”). Pre-terminal nodes are omitted for brevity. Right: legend of edge labels.

3.2 The UCCA Scheme

UCCA graphs are labeled, directed acyclic graphs (DAGs), whose leaves correspond to the tokens of the text. A node (or unit) corresponds to a terminal or to several terminals (not necessarily contiguous) viewed as a single entity according to semantic or cognitive considerations. Edges bear a category, indicating the role of the sub-unit in the parent relation. Figure 3.1 presents a few examples.

UCCA is a multi-layered representation, where each layer corresponds to a “module” of semantic distinctions. UCCA’s foundational layer, targeted in this paper, covers the predicate-argument structure evoked by predicates of all grammatical categories (verbal, nominal, adjectival and others), the inter-relations between them, and other major linguistic phenomena such as coordination and multi-word expressions. The layer’s basic notion is the scene, describing a state, action, movement or some other relation that evolves in time. Each scene contains one main relation (marked as either a Process or a State), as well as one or more Participants. For example, the sentence “After graduation, John moved to Paris” (Figure 3.1a) contains two scenes, whose main relations are “graduation” and “moved”. “John” is a Participant in both scenes, while “Paris” only in the latter. Further categories account for inter-scene relations and the internal structure of complex arguments and relations (e.g. coordination, multi-word expressions and modification).

One incoming edge for each non-root node is marked as primary, and the rest (mostly used for implicit relations and arguments) as remote edges, a distinction made by the annotator. The primary edges thus form a tree structure, whereas the remote edges enable reentrancy, forming a DAG.

While parsing technology in general, and transition-based parsing in particular, is well-established for syntactic parsing, UCCA has several distinct properties that distinguish it from syntactic representations, mostly UCCA’s tendency to abstract away from
syntactic detail that do not affect argument structure. For instance, consider the following examples where the concept of a scene has a different rationale from the syntactic concept of a clause. First, non-verbal predicates in UCCA are represented like verbal ones, such as when they appear in copula clauses or noun phrases. Indeed, in Figure 3.1a, “graduation” and “moved” are considered separate events, despite appearing in the same clause. Second, in the same example, “John” is marked as a (remote) Participant in the graduation scene, despite not being overtly marked. Third, consider the possessive construction in Figure 3.1c. While in UCCA “trip” evokes a scene in which “John and Mary” is a Participant, a syntactic scheme would analyze this phrase similarly to “John and Mary’s shoes”.

These examples demonstrate that a UCCA parser, and more generally semantic parsers, face an additional level of ambiguity compared to their syntactic counterparts (e.g., “after graduation” is formally very similar to “after 2pm”, which does not evoke a scene). Section 3.6 discusses UCCA in the context of other semantic schemes, such as AMR (Banarescu et al., 2013).

Alongside recent progress in dependency parsing into projective trees, there is increasing interest in parsing into representations with more general structural properties (see Section 3.6). One such property is reentrancy, namely the sharing of semantic units between predicates. For instance, in Figure 3.1a, “John” is an argument of both “graduation” and “moved”, yielding a DAG rather than a tree. A second property is discontinuity, as in Figure 3.1b, where “gave up” forms a discontinuous semantic unit. Discontinuities are pervasive, e.g., with multi-word expressions (Schneider et al., 2014). Finally, unlike most dependency schemes, UCCA uses non-terminal nodes to represent units comprising more than one word. The use of non-terminal nodes is motivated by constructions with no clear head, including coordination structures (e.g., “John and Mary” in Figure 3.1c), some multi-word expressions (e.g., “The Haves and the Have Nots”), and prepositional phrases (either the preposition or the head noun can serve as the constituent’s head). To our knowledge, no existing parser supports all structural properties required for UCCA parsing.

### 3.3 Transition-based UCCA Parsing

We now turn to presenting TUPA. Building on previous work on parsing reentrancies, discontinuities and non-terminal nodes, we define an extended set of transitions and features that supports the conjunction of these properties.

Transition-based parsers (Nivre, 2003) scan the text from start to end, and create the parse incrementally by applying a transition at each step to the parser’s state, defined...
<table>
<thead>
<tr>
<th>Before Transition</th>
<th>Transition</th>
<th>After Transition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack</td>
<td>Buffer</td>
<td>Nodes</td>
<td>Edges</td>
</tr>
<tr>
<td>$S$</td>
<td>$x$</td>
<td>$B$</td>
<td>$V$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$B$</td>
<td>$V$</td>
<td>$E$</td>
</tr>
<tr>
<td>$S \mid y \mid x$</td>
<td>$B$</td>
<td>$V$</td>
<td>$E$</td>
</tr>
<tr>
<td>$S \mid x \mid y$</td>
<td>$B$</td>
<td>$V$</td>
<td>$E$</td>
</tr>
<tr>
<td>$S \mid y \mid x$</td>
<td>$B$</td>
<td>$V$</td>
<td>$E$</td>
</tr>
<tr>
<td>$S \mid [\text{root}] \mid B$</td>
<td>$B$</td>
<td>$V$</td>
<td>$E$</td>
</tr>
</tbody>
</table>

Figure 3.2: The transition set of TUPA. We write the stack with its top to the right and the buffer with its head to the left. $\cdot \cdot \cdot_x$ denotes a primary $X$-labeled edge, and $\cdot \cdot \cdot_x^*$ a remote $X$-labeled edge. $i(x)$ is a running index for the created nodes. In addition to the specified conditions, the prospective child in an EDGE transition must not already have a primary parent.

using three data structures: a buffer $B$ of tokens and nodes to be processed, a stack $S$ of nodes currently being processed, and a graph $G = (V, E, \ell)$ of constructed nodes and edges, where $V$ is the set of nodes, $E$ is the set of edges, and $\ell : E \rightarrow L$ is the label function, $L$ being the set of possible labels. Some states are marked as terminal, meaning that $G$ is the final output. A classifier is used at each step to select the next transition based on features encoding the parser’s current state. During training, an oracle creates training instances for the classifier, based on gold-standard annotations.

**Transition Set.** Given a sequence of tokens $w_1, \ldots, w_n$, we predict a UCCA graph $G$ over the sequence. Parsing starts with a single node on the stack (an artificial root node), and the input tokens in the buffer. Figure 3.2 shows the transition set.

In addition to the standard **SHIFT** and **REDUCE** operations, we follow previous work in transition-based constituency parsing (Sagae and Lavie, 2005), adding the **NODE** transition for creating new non-terminal nodes. For every $X \in L$, **NODE**$_X$ creates a new node on the buffer as a parent of the first element on the stack, with an $X$-labeled edge. **LEFT-EDGE**$_X$ and **RIGHT-EDGE**$_X$ create a new primary $X$-labeled edge between the first two elements on the stack, where the parent is the left or the right node, respectively. As a UCCA node may only have one incoming primary edge, EDGE transitions are disallowed if the child node already has an incoming primary edge. **LEFT-REMOTE**$_X$ and **RIGHT-REMOTE**$_X$ do not have this restriction, and the created edge is additionally marked as remote. We distinguish between these two pairs of transitions to allow the parser to create remote edges without the possibility of producing invalid graphs. To support the prediction of multiple parents, node and edge transitions leave the stack unchanged, as in other work on transition-based dependency graph parsing (Sagae and Tsujii, 2008; Ribeyre et al., 2014; Tokgöz and Eryiğit, 2015). **REDUCE** pops the stack,
to allow removing a node once all its edges have been created. To handle discontinuous
nodes, Swap pops the second node on the stack and adds it to the top of the buffer, as
with the similarly named transition in previous work (Nivre 2009; Maier 2015). Finally,
Finish pops the root node and marks the state as terminal.

Classifier. The choice of classifier and feature representation has been shown to play an
important role in transition-based parsing (Chen and Manning 2014; Andor et al. 2016;
Kiperwasser and Goldberg 2016). To investigate the impact of the type of transition
classifier in UCCA parsing, we experiment with three different models.

1. Starting with a simple and common choice (e.g., Maier and Lichte 2016), TUPASparse
   uses a linear classifier with sparse features, trained with the averaged structured
   perceptron algorithm (Collins and Roark 2004) and MinUpdate (Goldberg and
   Elhadad, 2011): each feature requires a minimum number of updates in training to
   be included in the model.\footnote{We also experimented with a linear model using dense embedding features, trained with the averaged structured perceptron algorithm. It performed worse than the sparse perceptron model and was hence discarded.}

2. Changing the model to a feedforward neural network with dense embedding fea-
tures, TUPAMLPl (“multi-layer perceptron”), uses an architecture similar to that
of Chen and Manning (2014), but with two rectified linear layers instead of one
layer with cube activation. The embeddings and classifier are trained jointly.

3. Finally, TUPAbiLSTM uses a bidirectional LSTM for feature representation, on
top of the dense embedding features, an architecture similar to Kiperwasser and
Goldberg (2016). The BiLSTM runs on the input tokens in forward and backward
directions, yielding a vector representation that is then concatenated with dense
features representing the parser state (e.g., existing edge labels and previous parser
actions; see below). This representation is then fed into a feedforward network
similar to TUPAMLPl. The feedforward layers, BiLSTM and embeddings are all
trained jointly.

For all classifiers, inference is performed greedily, i.e., without beam search. Hyper-
parameters are tuned on the development set (see Section 3.4).

Features. TUPASparse uses binary indicator features representing the words, POS tags,
syntactic dependency labels and existing edge labels related to the top four stack elements
and the next three buffer elements, in addition to their children and grandchildren in
the graph. We also use bi- and trigram features based on these values (Zhang and
Figure 3.3: Illustration of the TUPA model. Top: parser state (stack, buffer and intermediate graph). Bottom: TUPA\textsubscript{BILSTM} architecture. Vector representation for the input tokens is computed by two layers of bidirectional LSTMs. The vectors for specific tokens are concatenated with embedding and numeric features from the parser state (for existing edge labels, number of children, etc.), and fed into the MLP for selecting the next transition.

Clark, 2009; Zhu et al., 2013), features related to discontinuous nodes (Maier, 2015, including separating punctuation and gap type), features representing existing edges and the number of parents and children, as well as the past actions taken by the parser. In addition, we use use a novel, UCCA-specific feature: number of remote children\footnote{See Appendix A.1 for a full list of used feature templates.}

For TUPA\textsubscript{MLP} and TUPA\textsubscript{BILSTM}, we replace all indicator features by a concatenation of the vector embeddings of all represented elements: words, POS tags, syntactic dependency labels, edge labels, punctuation, gap type and parser actions. These embeddings are initialized randomly. We additionally use external word embeddings initialized with pre-trained word2vec vectors (Mikolov et al., 2013a)\footnote{https://goo.gl/6ovEhC} updated during training. In addition to dropout between NN layers, we apply word dropout (Kiperwasser and Goldberg, 2016): with a certain probability, the embedding for a word is replaced with a zero vector.
We do not apply word dropout to the external word embeddings.

Finally, for all classifiers we add a novel real-valued feature to the input vector, \textit{ratio}, corresponding to the ratio between the number of terminals to number of nodes in the graph $G$. This feature serves as a regularizer for the creation of new nodes, and should be beneficial for other transition-based constituency parsers too.

\textbf{Training.} For training the transition classifiers, we use a dynamic oracle \cite{GoldbergNivre2012}, i.e., an oracle that outputs a set of optimal transitions: when applied to the current parser state, the gold standard graph is reachable from the resulting state. For example, the oracle would predict a \texttt{NODE} transition if the stack has on its top a parent in the gold graph that has not been created, but would predict a \texttt{RIGHT-EDGE} transition if the second stack element is a parent of the first element according to the gold graph and the edge between them has not been created. The transition predicted by the classifier is deemed correct and is applied to the parser state to reach the subsequent state, if the transition is included in the set of optimal transitions. Otherwise, a random optimal transition is applied, and for the perceptron-based parser, the classifier’s weights are updated according to the perceptron update rule.

POS tags and syntactic dependency labels are extracted using spaCy \cite{HonnibalJohnson2015}. We use the categorical cross-entropy objective function and optimize the NN classifiers with the Adam optimizer \cite{KingmaBa2014}.

\section{Experimental Setup}

\textbf{Data.} We conduct our experiments on the UCCA Wikipedia corpus (henceforth, \textit{Wiki}), and use the English part of the UCCA \textit{Twenty Thousand Leagues Under the Sea} English-French parallel corpus (henceforth, \textit{20K Leagues}) as out-of-domain data. Table 3.1 presents some statistics for the two corpora. We use passages of indices up to 676 of the \textit{Wiki} corpus as our training set, passages 688–808 as development set, and passages 942–1028 as in-domain test set. While UCCA edges can cross sentence boundaries, we adhere to the common practice in semantic parsing and train our parsers on individual sentences, discarding inter-relations between them (0.18\% of the edges). We also discard linkage nodes and edges (as they often express inter-sentence relations and are thus mostly redundant when applied at the sentence level) as well as implicit nodes.\footnote{Appendix A.2 further discusses linkage and implicit units.} In the out-of-domain experiments, we apply the same parsers (trained on the \textit{Wiki} training set) to the \textit{20K Leagues} corpus without parameter re-tuning.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Dataset & \textit{Wiki} & \textit{20K Leagues} & \textit{Wiki} \textit{20K Leagues} \textit{Wiki} \textit{20K Leagues} \textit{Wiki} \textit{20K Leagues} \textit{Wiki} \textit{20K Leagues} \\
\hline
\textit{Graphs} & 1028 & 1028 & 0.18 & 0.18 \\
\textit{Edges} & 942 & 942 & 19 & 19 \\
\textit{Nodes} & 2042 & 2042 & 4 & 4 \\
\hline
\end{tabular}
\caption{Statistics for the two corpora.}
\end{table}
Table 3.1: Statistics of the Wiki and 20K Leagues UCCA corpora. All counts exclude the root node, implicit nodes, and linkage nodes and edges.

**Implementation.** We use the DyNet package [Neubig et al. 2017] for implementing the NN classifiers. Unless otherwise noted, we use the default values provided by the package. See Appendix A.3 for the hyperparameter values we found by tuning on the development set.

**Evaluation.** We define a simple measure for comparing UCCA structures $G_p = (V_p, E_p, \ell_p)$ and $G_g = (V_g, E_g, \ell_g)$, the predicted and gold-standard graphs, respectively, over the same sequence of terminals $W = \{w_1, \ldots, w_n\}$. For an edge $e = (u, v)$ in either graph, $u$ being the parent and $v$ the child, its yield $y(e) \subseteq W$ is the set of terminals in $W$ that are descendants of $v$. Define the set of *mutual edges* between $G_p$ and $G_g$:

$$M(G_p, G_g) = \{(e_1, e_2) \in E_p \times E_g \mid y(e_1) = y(e_2) \land \ell_p(e_1) = \ell_g(e_2)\}$$

Labeled precision and recall are defined by dividing $|M(G_p, G_g)|$ by $|E_p|$ and $|E_g|$, respectively, and F-score by taking their harmonic mean. We report two variants of this measure: one where we consider only primary edges, and another for remote edges (see Section 3.2). Performance on remote edges is of pivotal importance in this investigation, which focuses on extending the class of graphs supported by statistical parsers.

We note that the measure collapses to the standard PARSEVAL constituency evaluation measure if $G_p$ and $G_g$ are trees. Punctuation is excluded from the evaluation, but not from the datasets.

**Comparison to bilexical graph parsers.** As no direct comparison with existing parsers is possible, we compare TUPA to bilexical dependency graph parsers, which
After graduation, John moved to Paris. John gave everything up. John and Mary went home.

Figure 3.4: Bilexical graph approximation (dependency graph) for the sentences in Figure 3.1.

<table>
<thead>
<tr>
<th></th>
<th>Wiki (in-domain)</th>
<th>20K Leagues (out-of-domain)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Primary</td>
<td>Remote</td>
</tr>
<tr>
<td></td>
<td>LP   LR LF</td>
<td>LP   LR LF</td>
</tr>
<tr>
<td>TUPASparse</td>
<td>64.5 63.7 64.1</td>
<td>19.8 13.4 16</td>
</tr>
<tr>
<td>TUPAMLPe</td>
<td>65.2 64.6 64.9</td>
<td>23.7 13.2 16.9</td>
</tr>
<tr>
<td>TUPABiLSTM</td>
<td>74.4 72.7 73.5</td>
<td>47.4 51.6 49.4</td>
</tr>
</tbody>
</table>

|                | Upper Bound | 58.3 | 91.3 | 43.4 |
|                | DAGParser   | 61.8 55.8 58.6 9.5 0.5 1 | 56.4 50.6 53.4 – 0 0 |
|                | TurboParser | 57.7 46 51.2 77.8 1.8 3.7 | 50.3 37.7 43.1 100 0.4 0.8 |

|                | Upper Bound | – | 100 | – |
|                | uparse      | 100 | 60.9 61.2 61.1 | 52.7 52.8 52.8 – – – |

|                | Upper Bound | 91 | – | 91.3 |
|                | MaltParser  | 62.8 57.7 60.2 | – – – | 57.8 53 55.3 – – – |
|                | LSTM Parser | 73.2 66.9 69.9 | – – – | 66.1 61.1 63.5 – – – |

Table 3.2: Experimental results, in percents, on the Wiki test set (left) and the 20K Leagues set (right). Columns correspond to labeled precision, recall and F-score, for both primary and remote edges. F-score upper bounds are reported for the conversions. For the tree approximation experiments, only primary edges scores are reported, as they are unable to predict remote edges. TUPABiLSTM obtains the highest F-scores in all metrics, surpassing the bilexical parsers, tree parsers and other classifiers.
After graduation, John moved to Paris.

Figure 3.5: Tree approximation (constituency) for the sentence in Figure 3.1a (top), and bilexical tree approximation (dependency) for the same sentence (bottom). These are identical to the original graphs, apart from the removal of remote edges.

support reentrancy and discontinuity but not non-terminal nodes.

To facilitate the comparison, we convert our training set into bilexical graphs (see examples in Figure 3.4), train each of the parsers, and evaluate them by applying them to the test set and then reconstructing UCCA graphs, which are compared with the gold standard. The conversion to bilexical graphs is done by heuristically selecting a head terminal for each non-terminal node, and attaching all terminal descendents to the head terminal. In the inverse conversion, we traverse the bilexical graph in topological order, creating non-terminal parents for all terminals, and attaching them to the previously-created non-terminals corresponding to the bilexical heads.

In Section 3.5 we report the upper bounds on the achievable scores due to the error resulting from the removal of non-terminal nodes.

Comparison to tree parsers. For completeness, and as parsing technology is considerably more mature for tree (rather than graph) parsing, we also perform a tree approximation experiment, converting UCCA to (bilexical) trees and evaluating constituency and dependency tree parsers on them (see examples in Figure 3.5). Our approach is similar to the tree approximation approach used for dependency graph parsing (Agić et al., 2015; Fernández-González and Martins, 2015), where dependency graphs were converted into dependency trees and then parsed by dependency tree parsers. In our setting, the conversion to trees consists simply of removing remote edges from the graph, and then to bilexical trees by applying the same procedure as for bilexical graphs.

Baseline parsers. We evaluate two bilexical graph semantic dependency parsers: DAGParser (Ribeyre et al., 2014), the leading transition-based parser in SemEval 2014 (Oepen et al.,

8See Appendix A.4 for a detailed description of the conversion procedures.
and TurboParser (Almeida and Martins, 2015), a graph-based parser from SemEval 2015 (Oepen et al., 2015); uparse (Maier and Lichte, 2016), a transition-based constituency parser supporting discontinuous constituents; and two bilexical tree parsers: MaltParser (Nivre et al., 2007), and the stack LSTM-based parser of Dyer et al. (2015, henceforce “LSTM Parser”). Default settings are used in all cases.

DAGParser and uparse use beam search by default, with a beam size of 5 and 4 respectively. The other parsers are greedy.

3.5 Results

Table 3.2 presents our main experimental results, as well as upper bounds for the baseline parsers, reflecting the error resulting from the conversion.

DAGParser and uparse are most directly comparable to TUPA Sparse, as they also use a perceptron classifier with sparse features. TUPA Sparse considerably outperforms both, where DAGParser does not predict any remote edges in the out-of-domain setting. TurboParser fares worse in this comparison, despite somewhat better results on remote edges. The LSTM parser of Dyer et al. (2015) obtains the highest primary F-score among the baseline parsers, with a considerable margin.

Using a feedforward NN and embedding features, TUPA MLP obtains higher scores than TUPA Sparse, but is outperformed by the LSTM parser on primary edges. However, using better input encoding allowing virtual look-ahead and look-behind in the token representation, TUPA BiLSTM obtains substantially higher scores than TUPA MLP and all other parsers, on both primary and remote edges, both in the in-domain and out-of-domain settings. Its performance in absolute terms, of 73.5% F-score on primary edges, is encouraging in light of UCCA’s inter-annotator agreement of 80–85% F-score on them (Abend and Rappoport, 2013).

The parsers resulting from tree approximation are unable to recover any remote edges, as these are removed in the conversion. The bilexical DAG parsers are quite limited in this respect as well. While some of the DAG parsers’ difficulty can be attributed to the conversion upper bound of 58.3%, this in itself cannot account for their poor performance on remote edges, which is an order of magnitude lower than that of TUPA BiLSTM.

9For MaltParser we use the ArcEager transition set and SVM classifier. Other configurations yielded lower scores.

10The low upper bound for remote edges is partly due to the removal of implicit nodes (not supported in bilexical representations), where the whole sub-graph headed by such nodes, often containing remote edges, must be discarded.

11We also experimented with a simpler version of TUPA lacking Remote transitions, obtaining an increase of up to 2 labeled F-score points on primary edges, at the cost of not being able to predict remote edges.
3.6 Related Work

While earlier work on anchored semantic parsing has mostly concentrated on shallow semantic analysis, focusing on semantic role labeling of verbal argument structures, the focus has recently shifted to parsing of more elaborate representations that account for a wider range of phenomena (Abend and Rappoport 2017).

Grammar-Based Parsing. Linguistically expressive grammars such as HPSG (Pollard and Sag 1994a), CCG (Steedman 2000) and TAG (Joshi and Schabes 1997) provide a theory of the syntax-semantics interface, and have been used as a basis for semantic parsers by defining compositional semantics on top of them (Flickinger 2000; Bos 2005 among others). Depending on the grammar and the implementation, such semantic parsers can support some or all of the structural properties UCCA exhibits. Nevertheless, this line of work differs from our approach in two important ways. First, the representations are different. UCCA does not attempt to model the syntax-semantics interface and is thus less coupled with syntax. Second, while grammar-based parsers explicitly model syntax, our approach directly models the relation between tokens and semantic structures, without explicit composition rules.

Broad-Coverage Semantic Parsing. Most closely related to this work is Broad-Coverage Semantic Dependency Parsing (SDP), addressed in two SemEval tasks (Oepen et al. 2014, 2015). Like UCCA parsing, SDP addresses a wide range of semantic phenomena, and supports discontinuous units and reentrancy. In SDP, however, bilexical dependencies are used, and a head must be selected for every relation—even in constructions that have no clear head, such as coordination (Ivanova et al. 2012). The use of non-terminal nodes is a simple way to avoid this liability. SDP also differs from UCCA in the type of distinctions it makes, which are more tightly coupled with syntactic considerations, where UCCA aims to capture purely semantic cross-linguistically applicable notions. For instance, the “poss” label in the DM target representation is used to annotate syntactic possessive constructions, regardless of whether they correspond to semantic ownership (e.g., “John’s dog”) or other semantic relations, such as marking an argument of a nominal predicate (e.g., “John’s kick”). UCCA reflects the difference between these constructions.

Recent interest in SDP has yielded numerous works on graph parsing (Ribeyre et al. 2014; Thomson et al. 2014; Almeida and Martins 2015; Du et al. 2015), including tree

---

12By anchored we mean that the semantic representation directly corresponds to the words and phrases of the text.
approximation (Agić and Koller, 2014; Schluter et al., 2014) and joint syntactic/semantic parsing (Henderson et al., 2013; Swayamdipta et al., 2016a).

Abstract Meaning Representation. Another line of work addresses parsing into AMRs (Flanigan et al., 2014; Vanderwende et al., 2015; Pust et al., 2015; Artzi et al., 2015), which, like UCCA, abstract away from syntactic distinctions and represent meaning directly, using OntoNotes predicates (Weischedel et al., 2013). Events in AMR may also be evoked by non-verbal predicates, including possessive constructions.

Unlike in UCCA, the alignment between AMR concepts and the text is not explicitly marked. While sharing much of this work’s motivation, not anchoring the representation in the text complicates the parsing task, as it requires the alignment to be automatically (and imprecisely) detected. Indeed, despite considerable technical effort (Flanigan et al., 2014; Pourdamghani et al., 2014; Werling et al., 2015), concept identification is only about 80%–90% accurate. Furthermore, anchoring allows breaking down sentences into semantically meaningful sub-spans, which is useful for many applications (Fernández-González and Martins, 2015; Birch et al., 2016).

Several transition-based AMR parsers have been proposed: CAMR assumes syntactically parsed input, processing dependency trees into AMR (Wang et al., 2015a,b, 2016; Goodman et al., 2016). In contrast, the parsers of Damonte et al. (2017) and Zhou et al. (2016) do not require syntactic pre-processing. Damonte et al. (2017) perform concept identification using a simple heuristic selecting the most frequent graph for each token, and Zhou et al. (2016) perform concept identification and parsing jointly. UCCA parsing does not require separately aligning the input tokens to the graph. TUPA creates non-terminal units as part of the parsing process.

Furthermore, existing transition-based AMR parsers are not general DAG parsers. They are only able to predict a subset of reentrancies and discontinuities, as they may remove nodes before their parents have been predicted (Damonte et al., 2017). They are thus limited to a sub-class of AMRs in particular, and specifically cannot produce arbitrary DAG parses. TUPA’s transition set, on the other hand, allows general DAG parsing.

3.7 Conclusion

We present TUPA, the first parser for UCCA. Evaluated in in-domain and out-of-domain settings, we show that coupled with a NN classifier and BiLSTM feature extractor, it

\[ \text{See Appendix A.5 for a proof sketch for the completeness of TUPA’s transition set.} \]
accurately predicts UCCA graphs from text, outperforming a variety of strong baselines by a margin.

Despite the recent diversity of semantic parsing work, the effectiveness of different approaches for structurally and semantically different schemes is not well-understood (Kuhlmann and Oepen, 2016). Our contribution to this literature is a general parser that supports multiple parents, discontinuous units and non-terminal nodes.

Future work will evaluate TUPA in a multilingual setting, assessing UCCA’s cross-linguistic applicability. We will also apply the TUPA transition scheme to different target representations, including AMR and SDP, exploring the limits of its generality. In addition, we will explore different conversion procedures (Kong et al., 2015) to compare different representations, suggesting ways for a data-driven design of semantic annotation.

A parser for UCCA will enable using the framework for new tasks, in addition to existing applications such as machine translation evaluation (Birch et al., 2016). We believe UCCA’s merits in providing a cross-linguistically applicable, broad-coverage annotation will support ongoing efforts to incorporate deeper semantic structures into various applications, such as sentence simplification (Narayan and Gardent, 2014) and summarization (Liu et al., 2015).

Acknowledgments

This work was supported by the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister’s Office, and by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI). The first author was supported by a fellowship from the Edmond and Lily Safra Center for Brain Sciences. We thank Wolfgang Maier, Nathan Schneider, Elior Sulem and the anonymous reviewers for their helpful comments.
Chapter 4

Multitask Parsing Across Semantic Representations (Published in ACL 2018)

Daniel Hershcovich\textsuperscript{1,2}, Omri Abend\textsuperscript{2} and Ari Rappoport\textsuperscript{2}
\textsuperscript{1}The Edmond and Lily Safra Center for Brain Sciences
\textsuperscript{2}School of Computer Science and Engineering
Hebrew University of Jerusalem
{danielh,oabend,arir}@cs.huji.ac.il

Abstract

The ability to consolidate information of different types is at the core of intelligence, and has tremendous practical value in allowing learning for one task to benefit from generalizations learned for others. In this paper we tackle the challenging task of improving semantic parsing performance, taking UCCA parsing as a test case, and AMR, SDP and Universal Dependencies (UD) parsing as auxiliary tasks. We experiment on three languages, using a uniform transition-based system and learning architecture for all parsing tasks. Despite notable conceptual, formal and domain differences, we show that multitask learning significantly improves UCCA parsing in both in-domain and out-of-domain settings. Our code is publicly available\textsuperscript{3}.

\textsuperscript{3}http://github.com/danielhers/tupa
4.1 Introduction

Semantic parsing has arguably yet to reach its full potential in terms of its contribution to downstream linguistic tasks, partially due to the limited amount of semantically annotated training data. This shortage is more pronounced in languages other than English, and less researched domains.

Indeed, recent work in semantic parsing has targeted, among others, Abstract Meaning Representation (AMR; Banarescu et al., 2013), bilexical Semantic Dependencies (SDP; Oepen et al., 2016) and Universal Conceptual Cognitive Annotation (UCCA; Abend and Rappoport, 2013). While these schemes are formally different and focus on different distinctions, much of their semantic content is shared (Abend and Rappoport, 2017).

Multitask learning (MTL; Caruana, 1997) allows exploiting the overlap between tasks to effectively extend the training data, and has greatly advanced with neural networks and representation learning (see §4.2). We build on these ideas and propose a general transition-based DAG parser, able to parse UCCA, AMR, SDP and UD (Nivre et al., 2016). We train the parser using MTL to obtain significant improvements on UCCA parsing over single-task training in (1) in-domain and (2) out-of-domain settings in English; (3) an in-domain setting in German; and (4) an in-domain setting in French, where training data is scarce.

The novelty of this work is in proposing a general parsing and learning architecture, able to accommodate such widely different parsing tasks, and in leveraging it to show benefits from learning them jointly.

4.2 Related Work

MTL has been used over the years for NLP tasks with varying degrees of similarity, examples including joint classification of different arguments in semantic role labeling (Toutanova et al., 2005), and joint parsing and named entity recognition (Finkel and Manning, 2009). Similar ideas, of parameter sharing across models trained with different datasets, can be found in studies of domain adaptation (Blitzer et al., 2006; Daume III, 2007; Ziser and Reichart, 2017). For parsing, domain adaptation has been applied successfully in parser combination and co-training (McClosky et al., 2010; Baucom et al., 2013).

Neural MTL has mostly been effective in tackling formally similar tasks (Sogaard and Goldberg, 2016), including multilingual syntactic dependency parsing (Ammar et al., 2016; Guo et al., 2016), as well as multilingual (Duong et al., 2017), and cross-domain semantic parsing (Herzig and Berant, 2017; Fan et al., 2017).
Sharing parameters with a low-level task has shown great benefit for transition-based syntactic parsing, when jointly training with POS tagging (Bohnet and Nivre, 2012; Zhang and Weiss, 2016), and with lexical analysis (Constant and Nivre, 2016; More, 2016). Recent work has achieved state-of-the-art results in multiple NLP tasks by jointly learning the tasks forming the NLP standard pipeline using a single neural model (Collobert et al., 2011; Hashimoto et al., 2017), thereby avoiding cascading errors, common in pipelines.

Much effort has been devoted to joint learning of syntactic and semantic parsing, including two CoNLL shared tasks (Surdeanu et al., 2008; Hajič et al., 2009). Despite their conceptual and practical appeal, such joint models rarely outperform the pipeline approach (Lluís and Màrquez, 2008; Henderson et al., 2013; Lewis et al., 2015; Swayamdipta et al., 2016a, 2017).

Peng et al. (2017a) performed MTL for SDP in a closely related setting to ours. They tackled three tasks, annotated over the same text and sharing the same formal structures (bilexical DAGs), with considerable edge overlap, but differing in target representations (see §4.3). For all tasks, they reported an increase of 0.5-1 labeled $F_1$ points. Recently, Peng et al. (2018) applied a similar approach to joint frame-semantic parsing and semantic dependency parsing, using disjoint datasets, and reported further improvements.

### 4.3 Tackled Parsing Tasks

In this section, we outline the parsing tasks we address. We focus on representations that produce full-sentence analyses, i.e., produce a graph covering all (content) words in the text, or the lexical concepts they evoke. This contrasts with “shallow” semantic parsing, primarily semantic role labeling (SRL; Gildea and Jurafsky, 2002; Palmer et al., 2005), which targets argument structure phenomena using flat structures. We consider four formalisms: UCCA, AMR, SDP and Universal Dependencies. Figure 4.1 presents one sentence annotated in each scheme.

**Universal Conceptual Cognitive Annotation.** UCCA (Abend and Rappoport, 2013) is a semantic representation whose main design principles are ease of annotation, cross-linguistic applicability, and a modular architecture. UCCA represents the semantics of linguistic utterances as directed acyclic graphs (DAGs), where terminal (childless) nodes correspond to the text tokens, and non-terminal nodes to semantic units that participate in some super-ordinate relation. Edges are labeled, indicating the role of a child in the relation the parent represents. Nodes and edges belong to one of several layers, each corresponding to a “module” of semantic distinctions. UCCA’s foundational layer (the only layer for which annotated data exists) mostly covers predicate-argument structure,
semantic heads and inter-Scene relations.

UCCA distinguishes primary edges, corresponding to explicit relations, from remote edges (appear dashed in Figure 4.1a) that allow for a unit to participate in several superordinate relations. Primary edges form a tree in each layer, whereas remote edges enable reentrancy, forming a DAG.

**Abstract Meaning Representation.** AMR ([Banarescu et al., 2013](#)) is a semantic representation that encodes information about named entities, argument structure, semantic roles, word sense and co-reference. AMRs are rooted directed graphs, in which both nodes and edges are labeled. Most AMRs are DAGs, although cycles are permitted.

AMR differs from the other schemes we consider in that it does not anchor its graphs in the words of the sentence (Figure 4.1b). Instead, AMR graphs connect variables, concepts (from a pre-defined set) and constants (which may be strings or numbers). Still, most AMR nodes are alignable to text tokens, a tendency used by AMR parsers, which align a subset of the graph nodes to a subset of the text tokens (concept identification). In this work, we use pre-aligned AMR graphs.

Despite the brief period since its inception, AMR has been targeted by a number of works, notably in two SemEval shared tasks ([May, 2016](#) May and Priyadarshi, 2017). To tackle its variety of distinctions and unrestricted graph structure, AMR parsers often use specialized methods. Graph-based parsers construct AMRs by identifying concepts and scoring edges between them, either in a pipeline fashion ([Flanigan et al., 2014](#) Artzi et al., 2015, Pust et al., 2015, Foland and Martin, 2017), or jointly ([Zhou et al., 2016](#)). Another line of work trains machine translation models to convert strings into linearized AMRs ([Barzdins and Gosko, 2016](#) Peng et al., 2017b, Konstas et al., 2017, Buys and Blunsom, 2017b). Transition-based AMR parsers either use dependency trees as pre-processing, then mapping them into AMRs ([Wang et al., 2015a](#) 2015b, 2016, Goodman et al., 2016), or use a transition system tailored to AMR parsing ([Damonte et al., 2017](#) Ballesteros and Al-Onaizan, 2017). We differ from the above approaches in addressing AMR parsing using the same general DAG parser used for other schemes.

**Semantic Dependency Parsing.** SDP uses a set of related representations, targeted in two recent SemEval shared tasks ([Oepen et al., 2014](#) 2015), and extended by [Oepen et al., 2016](#). They correspond to four semantic representation schemes, referred to as DM, PAS, PSD and CCD, representing predicate-argument relations between content words in a sentence. All are based on semantic formalisms converted into bilexical dependencies–directed graphs whose nodes are text tokens. Edges are labeled, encoding semantic relations between the tokens. Non-content tokens, such as punctuation, are left
out of the analysis (see Figure 4.1c). Graphs containing cycles have been removed from
the SDP datasets.

We use one of the representations from the SemEval shared tasks: DM (DELPH-IN
MRS), converted from DeepBank (Flickinger et al. 2012), a corpus of hand-corrected
parses from LinGO ERG (Copestake and Flickinger 2000), an HPSG (Pollard and Sag,
1994b) using Minimal Recursion Semantics (Copestake et al., 2005).

Universal Dependencies. UD (Nivre et al. 2016, 2017) has quickly become the dom-
inant dependency scheme for syntactic annotation in many languages, aiming for cross-
linguistically consistent and coarse-grained treebank annotation. Formally, UD uses bilex-
ical trees, with edge labels representing syntactic relations between words.

We use UD as an auxiliary task, inspired by previous work on joint syntactic and
semantic parsing (see §4.2). In order to reach comparable analyses cross-linguistically,
UD often ends up in annotation that is similar to the common practice in semantic
treebanks, such as linking content words to content words wherever possible. Using UD
further allows conducting experiments on languages other than English, for which AMR
and SDP annotated data is not available (§4.7).

In addition to basic UD trees, we use the enhanced++ UD graphs available for En-
glish, which are generated by the Stanford CoreNLP converters (Schuster and Manning,
2016). These include additional and augmented relations between content words, par-
tially overlapping with the notion of remote edges in UCCA: in the case of control verbs,
for example, a direct relation is added in enhanced++ UD between the subordinated verb
and its controller, which is similar to the semantic schemes’ treatment of this construction.

4.4 General Transition-based DAG Parser

All schemes considered in this work exhibit reentrancy and discontinuity (or non-projectivity),
to varying degrees. In addition, UCCA and AMR contain non-terminal nodes. To parse
these graphs, we extend TUPA (Hershcovich et al., 2017), a transition-based parser origi-
nally developed for UCCA, as it supports all these structural properties. TUPA’s transi-
tion system can yield any labeled DAG whose terminals are anchored in the text tokens.
To support parsing into AMR, which uses graphs that are not anchored in the tokens, we
take advantage of existing alignments of the graphs with the text tokens during training
(§4.5).

First used for projective syntactic dependency tree parsing (Nivre 2003), transition-
based parsers have since been generalized to parse into many other graph families, such

\footnote{http://github.com/stanfordnlp/CoreNLP}
as (discontinuous) constituency trees (e.g., Zhang and Clark, 2009; Maier and Lichte, 2016), and DAGs (e.g., Sagae and Tsujii, 2008; Du et al., 2015). Transition-based parsers apply transitions incrementally to an internal state defined by a buffer $B$ of remaining tokens and nodes, a stack $S$ of unresolved nodes, and a labeled graph $G$ of constructed nodes and edges. When a terminal state is reached, the graph $G$ is the final output. A classifier is used at each step to select the next transition, based on features that encode the current state.

### 4.4.1 TUPA’s Transition Set

Given a sequence of tokens $w_1, \ldots, w_n$, we predict a rooted graph $G$ whose terminals are the tokens. Parsing starts with the root node on the stack, and the input tokens in the buffer.

The TUPA transition set includes the standard Shift and Reduce operations, NODE$_X$ for creating a new non-terminal node and an $X$-labeled edge, LEFT-EDGE$_X$ and RIGHT-EDGE$_X$ to create a new primary $X$-labeled edge, LEFT-REMOTE$_X$ and RIGHT-REMOTE$_X$ to create a new remote $X$-labeled edge, Swap to handle discontinuous nodes, and Finish to mark the state as terminal.

Although UCCA contains nodes without any text tokens as descendants (called implicit units), these nodes are infrequent and only cover 0.5% of non-terminal nodes. For this reason we follow previous work (Hershcovich et al., 2017) and discard implicit units from the training and evaluation, and so do not include transitions for creating them.

In AMR, implicit units are considerably more common, as any unaligned concept with no aligned descendents is implicit (about 6% of the nodes). Implicit AMR nodes usually result from alignment errors, or from abstract concepts which have no explicit realization in the text (Buys and Blunsom, 2017a). We ignore implicit nodes when training on AMR as well. TUPA also does not support node labels, which are ubiquitous in AMR but absent in UCCA structures (only edges are labeled in UCCA). We therefore only produce edge labels and not node labels when training on AMR.

### 4.4.2 Transition Classifier

To predict the next transition at each step, we use a BiLSTM with embeddings as inputs, followed by an MLP and a softmax layer for classification (Kiperwasser and Goldberg, 2016). The model is illustrated in Figure 4.2. Inference is performed greedily, and training is done with an oracle that yields the set of all optimal transitions at a given state (those that lead to a state from which the gold graph is still reachable). Out of this set, the actual transition performed in training is the one with the highest score given by the classifier,
which is trained to maximize the sum of log-likelihoods of all optimal transitions at each step.

Figure 4.2: Illustration of the TUPA model, adapted from Hershcovich et al. (2017). Top: parser state. Bottom: BiLSTM architecture.

**Features.** We use the original TUPA features, representing the words, POS tags, syntactic dependency relations, and previously predicted edge labels for nodes in specific locations in the parser state. In addition, for each token we use embeddings representing the one-character prefix, three-character suffix, shape (capturing orthographic features, e.g., “Xxxx”), and named entity type, all provided by spaCy (Honnibal and Montani, 2018).\(^4\) To the learned word vectors, we concatenate the 250K most frequent word vectors from fastText (Bojanowksi et al., 2017),\(^5\) pre-trained over Wikipedia and updated during training.

**Constraints.** As each annotation scheme has different constraints on the allowed graph structures, we apply these constraints separately for each task. During training and parsing, the relevant constraint set rules out some of the transitions according to the parser state. Some constraints are task-specific, others are generic. For example, in UCCA, a terminal may only have one parent. In AMR, a concept corresponding to a PropBank frame may only have the core arguments defined for the frame as children. An

\(^3\)See Supplementary Material for a full listing of features.

\(^4\)http://spacy.io

\(^5\)http://fasttext.cc
example of a generic constraint is that stack nodes that have been swapped should not be swapped again.\footnote{To implement this constraint, we define a \textit{swap index} for each node, assigned when the node is created. At initialization, only the root node and terminals exist. We assign the root a swap index of 0, and for each terminal, its position in the text (starting at 1). Whenever a node is created as a result of a \texttt{Node} transition, its swap index is the arithmetic mean of the swap indices of the stack top and buffer head.}

### 4.5 Unified DAG Format

To apply our parser to the four target tasks (§4.3), we convert them into a unified DAG format, which is inclusive enough to allow representing any of the schemes with very little loss of information.\footnote{See Supplementary Material for more conversion details.}

The format consists of a rooted DAG, where the tokens are the terminal nodes. As in the UCCA format, edges are labeled (but not nodes), and are divided into \textit{primary} and \textit{remote} edges, where the primary edges form a tree (all nodes have at most one primary parent, and the root has none). Remote edges enable reentrancy, and thus together with primary edges form a DAG. Figure 4.3 shows examples for converted graphs. Converting UCCA into the unified format consists simply of removing linkage nodes and edges (see Figure 4.3a), which were also discarded by Hershcovich et al. (2017).

#### Converting bilexical dependencies.

To convert DM and UD into the unified DAG format, we add a pre-terminal for each token, and attach the pre-terminals according to the original dependency edges: traversing the tree from the root down, for each head token we create a non-terminal parent with the edge label \textit{head}, and add the node’s dependents as children of the created non-terminal node (see Figures 4.3c and 4.3d). Since DM allows multiple roots, we form a single root node, whose children are the original roots. The added edges are labeled \textit{root}, where top nodes are labeled \textit{top} instead. In case of reentrancy, an arbitrary parent is marked as primary, and the rest as remote (denoted as dashed edges in Figure 4.3).

#### Converting AMR.

In the conversion from AMR, node labels are dropped. Since alignments are not part of the AMR graph (see Figure 4.3b), we use automatic alignments (see §4.7), and attach each node with an edge to each of its aligned terminals.

Named entities in AMR are represented as a subgraph, whose \textit{name}-labeled root has a child for each token in the name (see the two \textit{name} nodes in Figure 4.1b). We collapse this subgraph into a single node whose children are the name tokens.
4.6 Multitask Transition-based Parsing

Now that the same model can be applied to different tasks, we can train it in a multitask setting. The fairly small training set available for UCCA (see §4.7) makes MTL particularly appealing, and we focus on it in this paper, treating AMR, DM and UD parsing as auxiliary tasks.

Following previous work, we share only some of the parameters (Klerke et al., 2016; Søgaard and Goldberg, 2016; Bollmann and Søgaard, 2016; Plank, 2016; Braud et al., 2016; Martínez Alonso and Plank, 2017; Peng et al., 2017a, 2018), leaving task-specific sub-networks as well. Concretely, we keep the BiLSTM used by TUPA for the main task (UCCA parsing), add a BiLSTM that is shared across all tasks, and replicate the MLP (feedforward sub-network) for each task. The BiLSTM outputs (concatenated for the main task) are fed into the task-specific MLP (see Figure 4.4). Feature embeddings are shared across tasks.

Unlabeled parsing for auxiliary tasks. To simplify the auxiliary tasks and facilitate generalization (Bingel and Søgaard, 2017), we perform unlabeled parsing for AMR, DM and UD, while still predicting edge labels in UCCA parsing. To support unlabeled parsing, we simply remove all labels from the Edge, Remote and Node transitions output by the oracle. This results in a much smaller number of transitions the classifier has to select from (no more than 10, as opposed to 45 in labeled UCCA parsing), allowing us to use no BiLSTMs and fewer dimensions and layers for task-specific MLPs of auxiliary tasks (see §4.7). This limited capacity forces the network to use the shared parameters for all tasks, increasing generalization (Martínez Alonso and Plank, 2017).

Figure 4.4: MTL model. Token representations are computed both by a task-specific and a shared BiLSTM. Their outputs are concatenated with the parser state embedding, identical to Figure 4.2, and fed into the task-specific MLP for selecting the next transition. Shared parameters are shown in blue.
Table 4.1: Number of tokens and sentences in the training, development and test sets we use for each corpus and language.

### 4.7 Experimental Setup

We here detail a range of experiments to assess the value of MTL to UCCA parsing, training the parser in single-task and multitask settings, and evaluating its performance on the UCCA test sets in both in-domain and out-of-domain settings.

**Data.** For UCCA, we use v1.2 of the English Wikipedia corpus (Wiki; Abend and Rappoport 2013), with the standard train/dev/test split (see Table 4.1), and the Twenty Thousand Leagues Under the Sea corpora (20K; Sulem et al. 2015), annotated in English, French and German. For English and French we use 20K v1.0, a small parallel corpus comprising the first five chapters of the book. As in previous work (Hershcovich et al. 2017), we use the English part only as an out-of-domain test set. We train and test on the French part using the standard split, as well as the German corpus (v0.9), which is a pre-release and still contains a considerable amount of noisy annotation. Tuning is performed on the respective development sets.

For AMR, we use LDC2017T10, identical to the dataset targeted in SemEval 2017 (May and Priyadarshi 2017). For SDP, we use the DM representation from the SDP 2016 dataset (Oepen et al. 2016). For Universal Dependencies, we use all English, French and German treebanks from UD v2.1 (Nivre et al. 2017). We use the enhanced++ UD representation (Schuster and Manning 2016) in our English experiments, henceforth referred to as UD+++. We use only the AMR, DM and UD training sets from standard splits.

While UCCA is annotated over Wikipedia and over a literary corpus, the domains for AMR, DM and UD are blogs, news, emails, reviews, and Q&A. This domain difference between training and test is particularly challenging (see §4.9). Unfortunately, none of the other schemes have available annotation over Wikipedia text.
Settings. We explore the following settings: (1) in-domain setting in English, training and testing on Wiki; (2) out-of-domain setting in English, training on Wiki and testing on 20K; (3) French in-domain setting, where available training dataset is small, training and testing on 20K; (4) German in-domain setting on 20K, with somewhat noisy annotation. For MTL experiments, we use unlabeled AMR, DM and UD++ parsing as auxiliary tasks in English, and unlabeled UD parsing in French and German.\footnote{We did not use AMR, DM or UD++ in French and German, as these are only available in English.} We also report baseline results training only the UCCA training sets.

Training. We create a unified corpus for each setting, shuffling all sentences from relevant datasets together, but using only the UCCA development set $F_1$ score as the early stopping criterion. In each training epoch, we use the same number of examples from each task—the UCCA training set size. Since training sets differ in size, we sample this many sentences from each one. The model is implemented using DyNet (Neubig et al., 2017).\footnote{http://dynet.io}

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Single</th>
<th>Main</th>
<th>Aux</th>
<th>Shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-trained word dim.</td>
<td>300</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learned word dim.</td>
<td>200</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS tag dim.</td>
<td>20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependency relation dim.</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Named entity dim.</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Punctuation dim.</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action dim.</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge label dim.</td>
<td>20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLP layers</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>MLP dimensions</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>BiLSTM layers</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>BiLSTM dimensions</td>
<td>500</td>
<td>300</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Hyperparameter settings. Middle column shows hyperparameters used for the single-task architecture, described in §4.4.2, and right column for the multitask architecture, described in §4.6. **Main** refers to parameters specific to the main task—UCCA parsing (task-specific MLP and BiLSTM, and edge label embedding), **Aux** to parameters specific to each auxiliary task (task-specific MLP, but no edge label embedding since the tasks are unlabeled), and **Shared** to parameters shared among all tasks (shared BiLSTM and embeddings).

Hyperparameters. We initialize embeddings randomly. We use dropout (Srivastava et al., 2014) between MLP layers, and recurrent dropout (Gal and Ghahramani, 2016).
Table 4.3: Labeled precision, recall and $F_1$ (in %) for primary and remote edges, on the Wiki test set. $\ast$ indicates significantly better than Single. HAR17: Hershcovich et al. (2017).

<table>
<thead>
<tr>
<th></th>
<th>Primary</th>
<th></th>
<th></th>
<th>Remote</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LP</td>
<td>LR</td>
<td>LF</td>
<td>LP</td>
<td>LR</td>
<td>LF</td>
</tr>
<tr>
<td>English (in-domain)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAR17</td>
<td>74.4</td>
<td>72.7</td>
<td>73.5</td>
<td>47.4</td>
<td>51.6</td>
<td>49.4</td>
</tr>
<tr>
<td>Single</td>
<td>74.4</td>
<td>72.9</td>
<td>73.6</td>
<td>53</td>
<td>50</td>
<td>51.5</td>
</tr>
<tr>
<td>AMR</td>
<td>74.7</td>
<td>72.8</td>
<td>73.7</td>
<td>48.7$\ast$</td>
<td>51.1</td>
<td>49.9</td>
</tr>
<tr>
<td>DM</td>
<td>75.7$\ast$</td>
<td>73.9$\ast$</td>
<td>74.8$\ast$</td>
<td>54.9</td>
<td>53</td>
<td>53.9</td>
</tr>
<tr>
<td>UD++</td>
<td>75$\ast$</td>
<td>73.2</td>
<td>74.1$\ast$</td>
<td>49</td>
<td>52.7</td>
<td>50.8</td>
</tr>
<tr>
<td>AMR + DM</td>
<td>75.6$\ast$</td>
<td>73.9$\ast$</td>
<td>74.7$\ast$</td>
<td>49.9</td>
<td>53</td>
<td>51.4</td>
</tr>
<tr>
<td>AMR + UD++</td>
<td>74.9</td>
<td>72.7</td>
<td>73.8</td>
<td>47.1</td>
<td>50</td>
<td>48.5</td>
</tr>
<tr>
<td>DM + UD++</td>
<td>75.9$\ast$</td>
<td>73.9$\ast$</td>
<td>74.9$\ast$</td>
<td>48</td>
<td>54.8</td>
<td>51.2</td>
</tr>
<tr>
<td>All</td>
<td>75.6$\ast$</td>
<td>73.1</td>
<td>74.4$\ast$</td>
<td>50.9</td>
<td>53.2</td>
<td>52</td>
</tr>
</tbody>
</table>

Evaluation. We evaluate on UCCA using labeled precision, recall and $F_1$ on primary and remote edges, following previous work (Hershcovich et al. 2017). Edges in predicted and gold graphs are matched by terminal yield and label. Significance testing of improvements over the single-task model is done by the bootstrap test (Berg-Kirkpatrick et al. 2012), with $p < 0.05$. 

between BiLSTM layers, both with $p = 0.4$. We also use word ($\alpha = 0.2$), tag ($\alpha = 0.2$) and dependency relation ($\alpha = 0.5$) dropout (Kiperwasser and Goldberg 2016). In addition, we use a novel form of dropout, node dropout: with a probability of 0.1 at each step, all features associated with a single node in the parser state are replaced with zero vectors. For optimization we use a minibatch size of 100, decaying all weights by $10^{-5}$ at each update, and train with stochastic gradient descent for $N$ epochs with a learning rate of 0.1, followed by AMSGrad (Sashank J. Reddi 2018) for $N$ epochs with $\alpha = 0.001, \beta_1 = 0.9$ and $\beta_2 = 0.999$. We use $N = 50$ for English and German, and $N = 400$ for French. We found this training strategy better than using only one of the optimization methods, similar to findings by Keskar and Socher (2017). We select the epoch with the best average labeled $F_1$ score on the UCCA development set. Other hyperparameter settings are listed in Table 4.2.
Table 4.4: Labeled precision, recall and $F_1$ (in %) for primary and remote edges, on the 20K test sets. * indicates significantly better than Single. HAR17: [Herscovich et al., 2017].

4.8 Results

Table 4.3 presents our results on the English in-domain Wiki test set. MTL with all auxiliary tasks and their combinations improves the primary $F_1$ score over the single task baseline. In most settings the improvement is statistically significant. Using all auxiliary tasks contributed less than just DM and UD++, the combination of which yielded the best scores yet in in-domain UCCA parsing, with 74.9% $F_1$ on primary edges. Remote $F_1$ is improved in some settings, but due to the relatively small number of remote edges (about 2% of all edges), none of the differences is significant. Note that our baseline single-task model (Single) is slightly better than the current state-of-the-art (HAR17; [Herscovich et al., 2017]), due to the incorporation of additional features (see §4.4.2).

Table 4.4 presents our experimental results on the 20K corpora in the three languages. For English out-of-domain, improvements from using MTL are even more marked. Moreover, the improvement is largely additive: the best model, using all three auxiliary tasks (All), yields an error reduction of 2.9%. Again, the single-task baseline is slightly better than HAR17.

The contribution of MTL is also apparent in French and German in-domain parsing: 3.7% error reduction in French (having less than 10% as much UCCA training data as

---

14 In training, the embedding for a feature value $w$ is replaced with a zero vector with a probability of $\frac{\alpha}{\#(w) + \alpha}$, where $\#(w)$ is the number of occurrences of $w$ observed.
Table 4.5: L1 distance between dataset word distributions, quantifying domain differences in English (low is similar).

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>20K</th>
<th>AMR</th>
<th>DM</th>
<th>UD</th>
</tr>
</thead>
<tbody>
<tr>
<td>20K</td>
<td>1.047</td>
<td>0.895</td>
<td>0.913</td>
<td>0.843</td>
<td></td>
</tr>
<tr>
<td>20K</td>
<td></td>
<td>0.949</td>
<td>0.971</td>
<td>0.904</td>
<td></td>
</tr>
<tr>
<td>AMR</td>
<td></td>
<td></td>
<td>0.757</td>
<td>0.469</td>
<td></td>
</tr>
<tr>
<td>DM</td>
<td></td>
<td></td>
<td></td>
<td>0.754</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: Unlabeled $F_1$ scores between the representations of the same English sentences (from PTB WSJ), converted to the unified DAG format, and annotated UCCA graphs.

<table>
<thead>
<tr>
<th></th>
<th>Primary</th>
<th>Remote</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UP</td>
<td>UR</td>
</tr>
<tr>
<td>AMR</td>
<td>53.8</td>
<td>15.6</td>
</tr>
<tr>
<td>DM</td>
<td>65</td>
<td>49.2</td>
</tr>
<tr>
<td>UD++</td>
<td>82.7</td>
<td>84.6</td>
</tr>
</tbody>
</table>

4.9 Discussion

Quantifying the similarity between tasks. Task similarity is an important factor in MTL success (Bingel and Søgaard, 2017; Martínez Alonso and Plank, 2017). In our case, the main and auxiliary tasks are annotated on different corpora from different domains (§4.7), and the target representations vary both in form and in content.

To quantify the domain differences, we follow Plank and van Noord (2011) and measure the L1 distance between word distributions in the English training sets and 20K test set (Table 4.5). All auxiliary training sets are more similar to 20K than Wiki is, which may contribute to the benefits observed on the English 20K test set.

As a measure of the formal similarity of the different schemes to UCCA, we use unlabeled $F_1$ score evaluation on both primary and remote edges (ignoring edge labels). To this end, we annotated 100 English sentences from Section 02 of the Penn Treebank Wall Street Journal (PTB WSJ). Annotation was carried out by a single expert UCCA annotator, and is publicly available.\(^{15}\) These sentences had already been annotated by the AMR, DM and PTB schemes\(^ {16}\), and we convert their annotation to the unified DAG

---

\(^{15}\) [http://github.com/danielhers/wsj](http://github.com/danielhers/wsj)

\(^{16}\) We convert the PTB format to UD++ v1 using Stanford CoreNLP, and then to UD v2 using Udapi: [http://github.com/udapi/udapi-python](http://github.com/udapi/udapi-python)
Unlabeled $F_1$ scores between the UCCA graphs and those converted from AMR, DM and UD++ are presented in Table 4.6. UD++ is highly overlapping with UCCA, while DM less so, and AMR even less (cf. Figure 4.3).

Comparing the average improvements resulting from adding each of the tasks as auxiliary (see §4.8), we find AMR the least beneficial, UD++ second, and DM the most beneficial, in both in-domain and out-of-domain settings. This trend is weakly correlated with the formal similarity between the tasks (as expressed in Table 4.6), but weakly negatively correlated with the word distribution similarity scores (Table 4.5). We conclude that other factors should be taken into account to fully explain this effect, and propose to address this in future work through controlled experiments, where corpora of the same domain are annotated with the various formalisms and used as training data for MTL.

**AMR, SDP and UD parsing.** Evaluating the full MTL model (All) on the unlabeled auxiliary tasks yielded 64.7% unlabeled Smatch $F_1$ (Cai and Knight, 2013) on the AMR development set, when using oracle concept identification (since the auxiliary model does not predict node labels), 27.2% unlabeled $F_1$ on the DM development set, and 4.9% UAS on the UD development set. These poor results reflect the fact that model selection was based on the score on the UCCA development set, and that the model parameters dedicated to auxiliary tasks were very limited (to encourage using the shared parameters). However, preliminary experiments using our approach produced promising results on each of the tasks’ respective English development sets, when treated as a single task: 67.1% labeled Smatch $F_1$ on AMR (adding a transition for implicit nodes and classifier for node labels), 79.1% labeled $F_1$ on DM, and 80.1% LAS $F_1$ on UD. For comparison, the best results on these datasets are 70.7%, 91.2% and 82.2%, respectively (Foland and Martin, 2017; Peng et al., 2018; Dozat et al., 2017).

### 4.10 Conclusion

We demonstrate that semantic parsers can leverage a range of semantically and syntactically annotated data, to improve their performance. Our experiments show that MTL improves UCCA parsing, using AMR, DM and UD parsing as auxiliaries. We propose a unified DAG representation, construct protocols for converting these schemes into the unified format, and generalize a transition-based DAG parser to support all these tasks, allowing it to be jointly trained on them.

While we focus on UCCA in this work, our parser is capable of parsing any scheme that can be represented in the unified DAG format, and preliminary results on AMR,
DM and UD are promising (see §4.9). Future work will investigate whether a single algorithm and architecture can be competitive on all of these parsing tasks, an important step towards a joint many-task model for semantic parsing.

Acknowledgments

This work was supported by the Israel Science Foundation (grant no. 929/17), by the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister’s Office, and by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI). The first author was supported by a fellowship from the Edmond and Lily Safra Center for Brain Sciences. We thank Roi Reichart, Rotem Dror and the anonymous reviewers for their helpful comments.
After graduation, John moved to Paris.

(a) UCCA

(b) AMR

(c) DM

(d) UD

Figure 4.1: Example graph for each task. Figure 4.1a presents a UCCA graph. The dashed edge is remote, while the blue node and its outgoing edges represent inter-Scene linkage. Pre-terminal nodes and edges are omitted for brevity. Figure 4.1b presents an AMR graph. Text tokens are not part of the graph, and must be matched to concepts and constants by alignment. Variables are represented by their concepts. Figure 4.1c presents a DM semantic dependency graph, containing multiple roots: “After”, “moved” and “to”, of which “moved” is marked as top. Punctuation tokens are excluded from SDP graphs. Figure 4.1d presents a UD tree. Edge labels express syntactic relations.
Figure 4.3: Graphs from Figure 4.1 after conversion to the unified DAG format (with pre-terminals omitted: each terminal drawn in place of its parent). Figure 4.3a presents a converted UCCA graph. Linkage nodes and edges are removed, but the original graph is otherwise preserved. Figure 4.3b presents a converted AMR graph, with text tokens added according to the alignments. Numeric suffixes of op relations are removed, and names collapsed. Figure 4.3c presents a converted SDP graph (in the DM representation), with intermediate non-terminal head nodes introduced. In case of reentrancy, an arbitrary reentrant edge is marked as remote. Figure 4.3d presents a converted UD graph. As in SDP, intermediate non-terminals and head edges are introduced. While converted UD graphs form trees, enhanced++ UD graphs may not.
Chapter 5

Universal Dependency Parsing with a General Transition-Based DAG Parser (Published in CoNLL 2018 Shared Task)

Daniel Hershcovich\textsuperscript{1,2}, Omri Abend\textsuperscript{2} and Ari Rappoport\textsuperscript{2}
\textsuperscript{1}The Edmond and Lily Safra Center for Brain Sciences
\textsuperscript{2}School of Computer Science and Engineering
Hebrew University of Jerusalem
\{danielh,oabend,arir\}@cs.huji.ac.il

Abstract

This paper presents our experiments with applying TUPA to the CoNLL 2018 UD shared task. TUPA is a general neural transition-based DAG parser, which we use to present the first experiments on recovering enhanced dependencies as part of the general parsing task. TUPA was designed for parsing UCCA, a cross-linguistic semantic annotation scheme, exhibiting reentrancy, discontinuity and non-terminal nodes. By converting UD trees and graphs to a UCCA-like DAG format, we train TUPA almost without modification on the UD parsing task. The generic nature of our approach lends itself naturally to multitask learning. Our code is available at \url{https://github.com/CoNLL-UD-2018/HUJI}. 
5.1 Introduction

In this paper, we present the HUJI submission to the CoNLL 2018 shared task on Universal Dependency parsing (Zeman et al. 2018). We focus only on parsing, using the baseline system, UDPipe 1.2 (Straka et al. 2016; Straka and Straková 2017) for tokenization, sentence splitting, part-of-speech tagging and morphological tagging.

Our system is based on TUPA (Hershcovich et al. 2017, 2018a, see §5.3), a transition-based UCCA parser. UCCA (Universal Conceptual Cognitive Annotation; Abend and Rappoport, 2013) is a cross-linguistic semantic annotation scheme, representing events, participants, attributes and relations in a directed acyclic graph (DAG) structure. UCCA allows reentrancy to support argument sharing, discontinuity (corresponding to non-projectivity in dependency formalisms) and non-terminal nodes (as opposed to dependencies, which are bi-lexical). To parse Universal Dependencies (Nivre et al., 2016) using TUPA, we employ a bidirectional conversion protocol to represent UD trees and graphs in a UCCA-like unified DAG format (§5.2).

Enhanced dependencies. Our method treats enhanced dependencies as part of the dependency graph, providing the first approach, to our knowledge, for supervised learning of enhanced UD parsing. Due to the scarcity of enhanced dependencies in UD treebanks, previous approaches (Schuster and Manning, 2016; Reddy et al., 2017) have attempted to recover them using language-specific rules. Our approach attempts to learn them from data: while only a few UD treebanks contain any enhanced dependencies, similar structures are an integral part of UCCA and its annotated corpora (realized as reentrancy by remote edges; see §5.2), and TUPA supports them as a standard feature.

As their annotation in UD is not yet widespread and standardized, enhanced dependencies are not included in the evaluation metrics for UD parsing, and so TUPA’s ability to parse them is not reflected in the official shared task scores. However, we believe these enhancements, representing case information, elided predicates, and shared arguments due to conjunction, control, raising and relative clauses, provide richer information to downstream semantic applications, making UD better suited for text understanding. We propose an evaluation metric specific to enhanced dependencies, enhanced LAS (§5.5.1), and use it to evaluate our method.

\footnote{http://universaldependencies.org/u/overview/enhanced-syntax.html}
Figure 5.1: (a) Example UCCA annotation for the sentence “We were made to feel very welcome.”, containing a control verb, made. The dashed $A$ edge is a remote edge. (b) Bilexical graph annotating the same sentence in UD (reviews-077034-0002 from UD_English-EWT). Enhanced dependencies appear below the sentence. (c) The same UD graph, after conversion to the unified DAG format. Intermediate non-terminals and head edges are introduced, to get a UCCA-like structure.

Figure 5.2: Example line from CoNLL-U file with two enhanced dependencies: 5:nsubj:xsubj and 7:nsubj:xsubj.

5.2 Unified DAG Format

To apply TUPA to UD parsing, we convert UD trees and graphs into a unified DAG format [Herschovich et al., 2018a]. The format consists of a rooted DAG, where the tokens are the terminal nodes. Edges are labeled (but not nodes), and are divided into primary and remote edges, where the primary edges form a tree (all nodes have at most one primary parent, and the root has none). Remote edges (denoted as dashed edges in Figure 5.1) enable reentrancy, and thus form a DAG together with primary edges. Figure 5.1 shows an example UCCA graph, and a UD graph (containing two enhanced dependencies) before and after conversion. Both annotate the same sentence from the
he went straight to work and finished the job efficiently and promptly!

Figure 5.3: UD graph from reviews-341397-0003 (UD_English-EWT), containing conjoined predicates and arguments.

I wish all happy holidays, and moreso, peace on earth.

Figure 5.4: newsgroup-groups.google.com_GuildWars_086f0f64ab633ab3_ENG_20041111_173500-0051 (UD_English-EWT), containing a null node (E9.1) and case information (nmod:on).

English Web Treebank (Silveira et al. 2014)[3]

Conversion protocol. To convert UD into the unified DAG format, we add a pre-terminal for each token, and attach the pre-terminals according to the original dependency edges: traversing the tree from the root down, for each head token we create a non-terminal parent with the edge label head, and add the node’s dependents as children of the created non-terminal node (see Figure 5.1c). This creates a constituency-like structure, which is supported by TUPA’s transition set (see §5.3.1).

Although the enhanced dependency graph is not necessarily a supergraph of the basic dependency tree, the graph we convert to the unified DAG format is their union: any enhanced dependencies that are distinct from the basic dependency of a node (by having a different head or universal dependency relation) are converted to remote edges in the unified DAG format.

To convert graphs in the unified DAG format back into dependency graphs, we collapse all head edges, determining for each terminal what is the highest non-terminal headed

He had a robe that was made back in the '60s.

(a) UD.

(b) UD converted to unified DAG format.

Figure 5.5: (a) reviews-255261-0007 (UD_English-EWT), containing a relative clause, and (b) the same graph after conversion to the unified DAG format. The cycle is removed due to the non-terminal nodes introduced in the conversion.

by it, and then attaching the terminals to each other according to the edges among their headed non-terminals.

**Input format.** Enhanced dependencies are encoded in the 9th column of the CoNLL-U format, by an additional head index, followed by a colon and dependency relation. Multiple enhanced dependencies for the same node are separated by pipes. Figure 5.2 demonstrates this format. Note that if the basic dependency is repeated in the enhanced graph (3:nsubj:pass in the example), we do not treat it as an enhanced dependency, so that the converted graph will only contain each edge once. In addition to the UD relations defined in the basic representations, enhanced dependencies may contain the relation ref, used for relative clauses. In addition, they may contain more specific relation subtypes, and optionally also case information.

**Language-specific extensions and case information.** Dependencies may contain language-specific relation subtypes, encoded as a suffix separated from the universal relation by a colon. These extensions are ignored by the parsing evaluation metrics, so for example, the subtyped relation nsubj:pass (Figure 5.1b) is considered the same as the universal relation nsubj for evaluation purposes. In the enhanced dependencies, these suffixes may also contain case information, which may be represented by the lemma of an adposition. For example, the “peace” → “earth” dependency in Figure 5.4 is augmented as nmod:on in the enhanced graph (not shown in the figure because it shares the universal relation with the basic dependency).

In the conversion process, we strip any language-specific extensions from both basic and enhanced dependencies, leaving only the universal relations. Consequently, case information that might be encoded in the enhanced dependencies is lost, and we do not handle it in our current system.
Ellipsis and null nodes. In addition to enhanced dependencies, the enhanced UD representation adds null nodes to represented elided predicates. These, too, are ignored in the standard evaluation. An example is shown in Figure 5.4 where an elided “wish” is represented by the node E9.1. The elided predicate’s dependents are attached to its argument “peace” in the basic representation, and the argument itself is attached as an orphan. In the enhanced representation, all arguments are attached to the null node as if the elided predicate was present.

While UCCA supports empty nodes without surface realization in the form of implicit units, previous work on UCCA parsing has removed these from the graphs. We do the same for UD parsing, dropping null nodes and their associated dependencies upon conversion to the unified DAG format. We leave parsing elided predicates for future work.

Propagation of conjuncts. Enhanced dependencies contain dependencies between conjoined predicates and their arguments, and between predicates and their conjoined arguments or modifiers. While these relations can often be inferred from the basic dependencies, in many cases they require semantic knowledge to parse correctly. For example, in Figure 5.3 the enhanced dependencies represent the shared subject (“he”) among the conjoined predicates (“went” and “finished”), and the conjoined modifiers (“efficiently” and “promptly”) for the second predicate (“finished”). However, there are no enhanced dependencies between the first predicate and the second predicate’s modifiers (e.g. “went” → “efficiently”), as semantically only the subject is shared and not the modifiers.

Relative clauses. Finally, enhanced graphs attach predicates of relative clauses directly to the antecedent modified by the relative clause, adding a ref dependency between the antecedent and the relative pronoun. An example is shown in Figure 5.5a. While these graphs may contain cycles (“robe” ↔ “made” in the example), they are removed upon conversion to the unified DAG format by the introduction of non-terminal nodes (see Figure 5.5b).

5.3 General Transition-based DAG Parser

We now turn to describing TUPA (Hershcovich et al., 2017, 2018a), a general transition-based parser (Nivre, 2003). TUPA uses an extended set of transitions and features that supports reentrancies, discontinuities and non-terminal nodes. The parser state is composed of a buffer $B$ of tokens and nodes to be processed, a stack $S$ of nodes currently being processed, and a graph $G = (V, E, \ell)$ of constructed nodes and edges, where $V$ is
The TUPA transition set, shown in Figure 5.6, includes the standard Shift and Reduce operations, Node\textsubscript{$X$} for creating a new non-terminal node and an $X$-labeled edge, Left-Edge\textsubscript{$X$} and Right-Edge\textsubscript{$X$} to create a new primary $X$-labeled edge, Left-Remote\textsubscript{$X$} and Right-Remote\textsubscript{$X$} to create a new remote $X$-labeled edge, Swap to handle discontinuous nodes, and Finish to mark the state as terminal.

The Remote\textsubscript{$X$} transitions are not required for parsing trees, but as we treat the problem as general DAG parsing due to the inclusion of enhanced dependencies, we include these transitions.

### 5.3.2 Transition Classifier

To predict the next transition at each step, TUPA uses a BiLSTM with feature embeddings as inputs, followed by an MLP and a softmax layer for classification. The model is illustrated in Figure 5.7. Inference is performed greedily, and training is done with an oracle that yields the set of all optimal transitions at a given state (those that lead to a state from which the gold graph is still reachable). Out of this set, the actual transition performed in training is the one with the highest score given by the classifier, which is

---

**Table 5.6: The transition set of TUPA.** We write the stack with its top to the right and the buffer with its head to the left. $(\cdot, \cdot)_X$ denotes a primary $X$-labeled edge, and $(\cdot, \cdot)^*_X$ a remote $X$-labeled edge. $i(x)$ is the swap index (see §5.3.3). In addition to the specified conditions, the prospective child in an Edge transition must not already have a primary parent.

<table>
<thead>
<tr>
<th>Before Transition</th>
<th>Transition</th>
<th>After Transition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack ($S$)</td>
<td>Buffer ($B$)</td>
<td>Nodes ($N$)</td>
<td>Edges ($E$)</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$B \mid y$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$B \mid y$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$B \mid y$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$y \mid {x}$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$y \mid {x}$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$y \mid {x}$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$y \mid {x}$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
<tr>
<td>$S \mid x$</td>
<td>$y \mid {x}$</td>
<td>$V \mid z$</td>
<td>$E \mid [\text{root}]$</td>
</tr>
</tbody>
</table>

---

5.3.1 Transition Set

Given a sequence of tokens $w_1, \ldots, w_n$, we predict a rooted graph $G$ whose terminals are the tokens. Parsing starts with the root node on the stack, and the input tokens in the buffer.

The TUPA transition set, shown in Figure 5.6, includes the standard Shift and Reduce operations, Node\textsubscript{$X$} for creating a new non-terminal node and an $X$-labeled edge, Left-Edge\textsubscript{$X$} and Right-Edge\textsubscript{$X$} to create a new primary $X$-labeled edge, Left-Remote\textsubscript{$X$} and Right-Remote\textsubscript{$X$} to create a new remote $X$-labeled edge, Swap to handle discontinuous nodes, and Finish to mark the state as terminal.

The Remote\textsubscript{$X$} transitions are not required for parsing trees, but as we treat the problem as general DAG parsing due to the inclusion of enhanced dependencies, we include these transitions.
We were welcome.

Figure 5.7: Illustration of the TUPA model, adapted from Hershcovitch et al. (2018a). Top: parser state (stack, buffer and intermediate graph). Bottom: BiLSTM architecture. Vector representation for the input tokens is computed by two layers of bidirectional LSTMs. The vectors for specific tokens are concatenated with embedding and numeric features from the parser state (for existing edge labels, number of children, etc.), and fed into the MLP for selecting the next transition.

Features. We use vector embeddings representing the words, lemmas, coarse (universal) POS tags and fine-grained POS tags, provided by UDPipe 1.2 during test. For training, we use the gold-annotated lemmas and POS tags. In addition, we use one-character prefix, three-character suffix, shape (capturing orthographic features, e.g., “Xxxx”) and named entity type, provided by spaCy, punctuation and gap type features (Maier and Lichte, 2016), and previously predicted edge labels and parser actions. These embeddings are initialized randomly, except for the word embeddings, which are initialized with the 250K most frequent word vectors from fastText for each language (Bojanowski et al., 2017) pre-trained over Wikipedia and updated during training. We do not use word embeddings for languages without pre-trained fastText vectors (Ancient Greek, North Sami and Old French).

To the feature embeddings, we concatenate numeric features representing the node height, number of (remote) parents and children, and the ratio between the number of terminals to total number of nodes in the graph $G$.

Table 5.1 lists all feature used for the classifier. Numeric features are taken as they are, whereas categorical features are mapped to real-valued embedding vectors. For each non-terminal node, we select a head terminal for feature extraction, by traversing down the graph according to a priority order on edge labels (otherwise selecting the leftmost child). The priority order is: parataxis, conj, advcl, xcomp

\[\text{http://spacy.io}\]
\[\text{http://fasttext.cc}\]
nodes
s_0
s_1
s_2
s_3
b_0
b_1
b_2
b_3
s_{0l}, s_{0r}, s_{1l}, s_{1r},
s_{0l}, s_{0r}, s_{1l}, s_{1r},
s_{0l}, s_{0r}, s_{1l}, s_{1r},
s_{0l}, s_{0r}, s_{1l}, s_{1r},
s_{0l}, s_{0r}, s_{1l}, s_{1r},
s_{0l}, s_{0r}, s_{1l}, s_{1r},
s_{0l}, s_{0r}, s_{1l}, s_{1r},

edges
s_0 \rightarrow s_1, s_0 \rightarrow b_0,
s_1 \rightarrow s_0, b_0 \rightarrow s_0,
s_0 \rightarrow b_0, b_0 \rightarrow s_0.

past actions
s_0 \rightarrow s_1, s_0 \rightarrow b_0.

Global
node ratio

Table 5.1: Transition classifier features.

s_i: stack node i from the top. b_i: buffer node i.
x_l, x_r (x_L, x_R): x’s leftmost and rightmost children (parents). w: head terminal text.
m: lemma. u: coarse (universal) POS tag. t: fine-grained POS tag. h: node’s height. e: label of its first incoming edge. p: any separator punctuation between s_0 and s_1. q: count of any separator punctuation between s_0 and s_1. x: numeric value of gap type (Maier and Lichte, 2016). y: sum of gap lengths. P, C, I, E, and M: number of parents, children, implicit children, remote children, and remote parents. N: numeric value of the head terminal’s named entity IOB indicator. T: named entity type. #: word shape (capturing orthographic features, e.g. ”Xxxx” or ”dd”). ^: one-character prefix. $: three-character suffix.

x \rightarrow y refers to the existing edge from x to y. x is an indicator feature, taking the value of 1 if the edge exists or 0 otherwise, e refers to the edge label, and a_i to the transition taken i + 1 steps ago.

A refers to the action type (e.g. SHIFT/RIGHT-EDGE/NODE), and e to the edge label created by the action.

node ratio is the ratio between non-terminals and terminals (Hershcovich et al., 2017).

5.3.3 Constraints

During training and parsing, we apply constraints on the parser state to limit the possible transitions to valid ones.

A generic constraint implemented in TUPA is that stack nodes that have been swapped should not be swapped again (Hershcovich et al., 2018a). To implement this constraint, we define a swap index for each node, assigned when the node is created. At initialization, only the root node and terminals exist. We assign the root a swap index of 0, and for each terminal, its position in the text (starting at 1). Whenever a node is created as a result of a NODE transition, its swap index is the arithmetic mean of the swap indices of the stack top and buffer head.

In addition, we enforce UD-specific constraints, resulting from the nature of the converted DAG format: every non-terminal node must have a single outgoing head edge: once it has one, it may not get another, and until it does, the node may not be reduced.
5.4 Training details

The model is implemented using DyNet v2.0.3 \footnote{http://dynet.io} \cite{Neubig2017} Unless otherwise noted, we use the default values provided by the package. We use the same hyperparameters as used in previous experiments on UCCA parsing \cite{Hershcovich2018a}, without any hyperparameter tuning on UD treebanks.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-trained word dim.</td>
<td>300</td>
</tr>
<tr>
<td>Lemma dim.</td>
<td>200</td>
</tr>
<tr>
<td>Coarse (universal) POS tag dim.</td>
<td>20</td>
</tr>
<tr>
<td>Fine-grained POS tag dim.</td>
<td>20</td>
</tr>
<tr>
<td>Named entity dim.</td>
<td>3</td>
</tr>
<tr>
<td>Punctuation dim.</td>
<td>1</td>
</tr>
<tr>
<td>Shape dim.</td>
<td>3</td>
</tr>
<tr>
<td>Prefix dim.</td>
<td>2</td>
</tr>
<tr>
<td>Suffix dim.</td>
<td>3</td>
</tr>
<tr>
<td>Action dim.</td>
<td>3</td>
</tr>
<tr>
<td>Edge label dim.</td>
<td>20</td>
</tr>
<tr>
<td>MLP layers</td>
<td>2</td>
</tr>
<tr>
<td>MLP dimensions</td>
<td>50</td>
</tr>
<tr>
<td>BiLSTM layers</td>
<td>2</td>
</tr>
<tr>
<td>BiLSTM dimensions</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 5.2: Hyperparameter settings.

5.4.1 Hyperparameters

We use dropout \cite{Srivastava2014} between MLP layers, and recurrent dropout \cite{Gal2016} between BiLSTM layers, both with $p = 0.4$. We also use word, lemma, coarse- and fine-grained POS tag dropout with $\alpha = 0.2$ \cite{Kiperwasser2016}: in training, the embedding for a feature value $w$ is replaced with a zero vector with a probability of $\frac{\alpha}{\#(w) + \alpha}$, where $\#(w)$ is the number of occurrences of $w$ observed. In addition, we use node dropout \cite{Hershcovich2018a}: with a probability of 0.1 at each step, all features associated with a single node in the parser state are replaced with zero vectors. For optimization we use a minibatch size of 100, decaying all weights by $10^{-5}$ at each update, and train with stochastic gradient descent for 50 epochs with a learning rate of 0.1, followed by AMSGrad \cite{Sashank2018} for 250 epochs with $\alpha = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We found this training strategy better than using only one of the optimization methods, similar to findings by \cite{Keskar2017}. We select the epoch with the best LAS-F1 on the development set. Other hyperparameter settings are listed in Table 5.2.

5.4.2 Small Treebanks

For corpora with less than 100 training sentences, we use 750 epochs of AMSGrad instead of 250. For corpora with no development set, we use 10-fold cross-validation on the
training set, each time splitting it to 80% training, 10% development and 10% validation. We perform the normal training procedure on the training and development subsets, and then select the model from the fold with the best LAS-F1 on the corresponding validation set.

5.4.3 Multilingual Model

For the purpose of parsing languages with no training data, we use a delexicalized multilingual model, trained on the shuffled training sets from all corpora, with no word, lemma, fine-grained tag, prefix and suffix features. We train this model for two epochs using stochastic gradient descent with a learning rate of 0.1 (we only trained this many epochs due to time constraints).

5.4.4 Out-of-domain Evaluation

For test treebanks without corresponding training data, but with training data in the same language, during testing we use the model trained on the largest training treebank in the same language.

5.5 Results

Official evaluation was done on the TIRA online platform [Potthast et al., 2014]. Our system (named “HUJI”) ranked 24th in the LAS-F1 ranking (with an average of 53.69 over all test treebanks), 23rd by MLAS (average of 44.6) and 21st by BLEX (average of 48.05). Since our system only performs dependency parsing and not other pipeline tasks, we henceforth focus on LAS-F1 [Nivre and Fang, 2017] for evaluation.

After the official evaluation period ended, we discovered several bugs in the conversion between the CoNLL-U format and the unified DAG format, which is used by TUPA for training and is output by it (see §5.2). We did not re-train TUPA on the training treebanks after fixing these bugs, but we did re-evaluate the already trained models on all test treebanks, and used the fixed code for converting their output to CoNLL-U. This yielded an unofficial average test LAS-F1 of 58.48, an improvement of 4.79 points over the official average score. In particular, for two test sets, ar_padt and gl_ctg, TUPA got a zero score in the official evaluation due to a bug with the treatment of multi-token words. These went up to 61.9 and 71.42, respectively. We also evaluated the trained TUPA models on all available development treebanks after fixing the bugs.

Table 5.3 presents the averaged scores on the shared task test sets, and Figure 5.8 the (official and unofficial) LAS-F1 scores obtained by TUPA on each of the test and
Table 5.3: Aggregated test LAS-F1 scores for our system (TUPA) and the baseline system (UDPipe 1.2).

deviation treebanks.

5.5.1 Evaluation on Enhanced Dependencies

Since the official evaluation ignores enhanced dependencies, we evaluate them separately using a modified version of the shared task evaluation script: We calculate the enhanced LAS, identical to the standard LAS except that the set of dependencies in both gold and predicted graphs are the enhanced dependencies instead of the basic dependencies: ignoring null nodes and any enhanced dependency sharing a head with a basic one, we align the words in the gold graph and the system’s graph as in the standard LAS, and define

\[ P = \frac{\text{#correct}}{\text{#system}}, \quad R = \frac{\text{#correct}}{\text{#gold}}, \quad F_1 = 2 \cdot \frac{P \cdot R}{P + R} \]

Table 5.4 lists the enhanced LAS precision, recall and F1 score on the test treebanks with any enhanced dependencies, as well as the percentage of enhanced dependencies in each test treebank, calculated as \( \frac{\text{#enhanced}}{\text{#enhanced} + \text{#words}} \).

Just as remote edges in UCCA parsing are more challenging than primary edges [Hershcovitch et al., 2017], parsing enhanced dependencies is a harder task than standard UD parsing, as the scores demonstrate. However, TUPA learns them successfully, getting as much as 56.55 enhanced LAS-F1 (on the Polish LFG test set).

5.5.2 Ablation Experiments

The TUPA transition classifier for some of the languages uses named entity features calculated by spaCy: For German, Spanish, Portuguese, French, Italian, Dutch and Russian, the spaCy named entity recognizer was trained on Wikipedia (Nothman et al., 2013). However, the English model was trained on OntoNotes, which is in fact not among

8 https://spacy.io/api/annotation
9 https://catalog.ldc.upenn.edu/LDC2013T19
Table 5.4: TUPA’s enhanced LAS precision, recall and F1 per test treebank with any enhanced dependencies, and percentage of enhanced dependencies in test treebank.

<table>
<thead>
<tr>
<th>Treebank</th>
<th>Enhanced LAS</th>
<th>% Enhanced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>ar_padt</td>
<td>28.51</td>
<td>16.24</td>
</tr>
<tr>
<td>cs_cac</td>
<td>54.94</td>
<td>35.69</td>
</tr>
<tr>
<td>cs_fictree</td>
<td>48.78</td>
<td>18.53</td>
</tr>
<tr>
<td>cs_pdt</td>
<td>49.46</td>
<td>26.47</td>
</tr>
<tr>
<td>nl_alpino</td>
<td>56.04</td>
<td>50.81</td>
</tr>
<tr>
<td>nl_lassysmall</td>
<td>49.71</td>
<td>51.30</td>
</tr>
<tr>
<td>en_ewt</td>
<td>57.36</td>
<td>52.05</td>
</tr>
<tr>
<td>en_pud</td>
<td>58.99</td>
<td>50.00</td>
</tr>
<tr>
<td>fi_tdt</td>
<td>40.20</td>
<td>31.37</td>
</tr>
<tr>
<td>lv_lvtb</td>
<td>31.76</td>
<td>18.70</td>
</tr>
<tr>
<td>pl_lfg</td>
<td>59.19</td>
<td>54.13</td>
</tr>
<tr>
<td>sk_snk</td>
<td>37.28</td>
<td>21.61</td>
</tr>
<tr>
<td>sv_pud</td>
<td>45.40</td>
<td>39.58</td>
</tr>
<tr>
<td>sv_talbanken</td>
<td>50.15</td>
<td>43.20</td>
</tr>
</tbody>
</table>

the additional resources allowed by the shared task organizers. To get a fair evaluation and to quantify the contribution of the NER features, we re-trained TUPA on the English EWT (en_ewt) training set with the same hyperparameters as in our submitted model, just without these features. As Table 5.5 shows, removing the NER features (−NER) only slightly hurts the performance, by 0.28 LAS-F1 points on the test treebank, and 0.63 on the development treebank.

As further ablation experiments, we tried removing POS features, pre-trained word embeddings, and remote edges (the latter enabling TUPA to parse enhanced dependencies). Removing the POS features does hurt performance to a larger degree, by 2.87 LAS-F1 points on the test set, while removing the pre-trained word embeddings even slightly improves the performance. Removing remote edges and transitions from TUPA causes a very small decrease in LAS-F1, and of course enhanced dependencies can then no longer be produced at all.

5.6 Conclusion

We have presented the HUJI submission to the CoNLL 2018 shared task on parsing Universal Dependencies, based on TUPA, a general transition-based DAG parser. Using a simple conversion protocol to convert UD into a unified DAG format, training TUPA as-is on the UD treebanks yields results close to the UDPipe baseline for most treebanks in the standard evaluation. While other systems ignore enhanced dependencies, TUPA
Table 5.5: Ablation LAS-F1 and Enhanced LAS-F1 on the English EWT development and test set. NER: named entity features. POS: part-of-speech tag features (both universal and fine-grained). Embed.: external pre-trained word embeddings (fastText). Remote: remote edges and transitions in TUPA.

learns to produce them too as part of the general dependency parsing process. We believe that with hyperparameter tuning and more careful handling of cross-lingual and cross-domain parsing, TUPA can be competitive on the standard metrics too.

Furthermore, the generic nature of our parser, which supports many representation schemes, as well as domains and languages, will allow improving performance by multitask learning (cf. Hershcovich et al., 2018a), which we plan to explore in future work.

Acknowledgments

This work was supported by the Israel Science Foundation (grant no. 929/17) and by the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister’s Office.
Development scores (where available).

Figure 5.8: TUPA’s LAS-F1 per treebank: official and unofficial test scores, and development scores (where available).

73
Chapter 6

Content Differences in Syntactic and Semantic Representations
(Published in NAACL-HLT 2019)

Daniel Hershcovich\textsuperscript{1,2}, Omri Abend\textsuperscript{2} and Ari Rappoport\textsuperscript{2}
\textsuperscript{1}The Edmond and Lily Safra Center for Brain Sciences
\textsuperscript{2}School of Computer Science and Engineering
Hebrew University of Jerusalem
{danielh,oabend,arir}@cs.huji.ac.il

Abstract

Syntactic analysis plays an important role in semantic parsing, but the nature of this role remains a topic of ongoing debate. The debate has been constrained by the scarcity of empirical comparative studies between syntactic and semantic schemes, which hinders the development of parsing methods informed by the details of target schemes and constructions. We target this gap, and take Universal Dependencies (UD) and UCCA as a test case. After abstracting away from differences of convention or formalism, we find that most content divergences can be ascribed to: (1) UCCA’s distinction between a Scene and a non-Scene; (2) UCCA’s distinction between primary relations, secondary ones and participants; (3) different treatment of multi-word expressions, and (4) different treatment of inter-clause linkage. We further discuss the long tail of cases where the two schemes take markedly different approaches. Finally, we show that the proposed comparison methodology can be used for fine-grained evaluation of UCCA parsing, highlighting both challenges and potential sources for improvement. The substantial differences between the schemes suggest that semantic parsers are likely to benefit downstream text
understanding applications beyond their syntactic counterparts.

### 6.1 Introduction

Semantic representations hold promise due to their ability to transparently reflect distinctions relevant for text understanding applications. For example, syntactic representations are usually sensitive to distinctions based on POS (part of speech), such as between compounds and possessives. Semantic schemes are less likely to make this distinction since a possessive can often be paraphrased as a compound and vice versa (e.g., “US president”/“president of the US”), but may distinguish different senses of possessives (e.g., “some of the presidents” and “inauguration of the presidents”).

Nevertheless, little empirical study has been done on what distinguishes semantic schemes from syntactic ones, which are still in many cases the backbone of text understanding systems. Such studies are essential for (1) determining whether and to what extent semantic methods should be adopted for text understanding applications; (2) defining better inductive biases for semantic parsers, and allowing better use of information encoded in syntax; (3) pointing at semantic distinctions unlikely to be resolved by syntax.

The importance of such an empirical study is emphasized by the ongoing discussion as to what role syntax should play in semantic parsing, if any (Swayamdipta et al., 2018; Strubell et al., 2018; He et al., 2018; Cai et al., 2018). See §6.8.

This paper aims to address this gap, focusing on content differences. As a test case, we compare relatively similar schemes (§6.2): the syntactic Universal Dependencies (UD; Nivre et al., 2016), and the semantic Universal Conceptual Cognitive Annotation (UCCA; Abend and Rappoport, 2013).

We UCCA-annotate the entire web reviews section of the UD EWT corpus (§6.3), and develop a converter to assimilate UD and UCCA, which use formally different graphs (§6.4). We then align their nodes, and identify which UCCA categories match which UD relations, and which are unmatched.

Most content differences are due to (§6.5):

1. UCCA’s distinction between words and phrases that evoke Scenes (events) and ones that do not. For example, eventive and non-eventive nouns are treated differently in UCCA, but similarly in UD.

2. UCCA’s distinction between primary relations, secondary relations and Participants, in contrast to UD’s core/non-core distinction.
3. Different treatment of multi-word expressions (MWEs), where UCCA has a stronger tendency to explicitly mark them.

4. UCCA’s conflation of several syntactic realizations of inter-clause linkage, and disambiguation of other cases that UD treats similarly.

We show that the differences between the schemes are substantial, and suggest that UCCA parsing in particular and semantic parsing in general are likely to benefit downstream text understanding applications. For example, only 72.9% of UCCA Participants are UD syntactic arguments, i.e., many semantic participants cannot be recovered from UD\textsuperscript{1}. Our findings are relevant to other semantic representations, given their significant overlap in content (Abend and Rappoport\textsuperscript{2} 2017).

A methodology for comparing syntactic and semantic treebanks can also support fine-grained error analysis of semantic parsers, as illustrated by Szubert et al.\textsuperscript{3} (2018) for AMR (Banarescu et al. 2013). To demonstrate the utility of our comparison methodology, we perform fine-grained error analysis on UCCA parsing, according to UD relations (§6.6). Results highlight challenges for current parsing technology, and expose cases where UCCA parsers may benefit from modeling syntactic structure more directly\textsuperscript{2}.

6.2 Representations

The conceptual and formal similarity between UD and UCCA can be traced back to their shared design principles: both are designed to be applicable across languages and domains, to enable rapid annotation and to support text understanding applications. This section provides a brief introduction to each of the schemes, whereas the next sections discuss their content in further detail\textsuperscript{3}.

**UCCA** is a semantic annotation scheme rooted in typological and cognitive linguistic theory. It aims to represent the main semantic phenomena in text, abstracting away from syntactic forms. Shown to be preserved remarkably well across translations (Sulem et al. 2015), it has been applied to improve text simplification (Sulem et al. 2018b), and text-to-text generation evaluation (Birch et al. 2016; Choshen and Abend 2018; Sulem et al. 2018a).

Formally, UCCA structures are directed acyclic graphs (DAGs) whose nodes (or *units*) correspond either to words, or to elements viewed as a single entity according to some

---

\textsuperscript{1}This excludes cases of shared argumenthood, which are partially covered by enhanced UD. See §6.4.1.

\textsuperscript{2}Our conversion and analysis code is public available at https://github.com/danielhers/synsem.

\textsuperscript{3}See Supplementary Material for a definition of each category in both schemes, and their abbreviations.
semantic or cognitive consideration. Edges are labeled, indicating the role of a child in the relation the parent represents. Figure 6.1 shows a legend of UCCA abbreviations. A Scene is UCCA’s notion of an event or a frame, and is a description of a movement, an action or a state which persists in time. Every Scene contains one primary relation, which can be either a Process or a State. Scenes may contain any number of Participants, a category which also includes abstract participants and locations. They may also contain temporal relations (Time), and secondary relations (Adverbials), which cover semantic distinctions such as manner, modality and aspect.

Scenes may be linked to one another in several ways. First, a Scene can provide information about some entity, in which case it is marked as an Elaborator. This often occurs in the case of participles or relative clauses. For example, “(child) who went to school” is an Elaborator Scene in “The child who went to school is John”. A Scene may also be a Participant in another Scene. For example, “John went to school” in the sentence: “He said John went to school”. In other cases, Scenes are annotated as Parallel Scenes (H), which are flat structures and may include a Linker (L), as in: “When he arrives, he will call them”.

Non-Scene units are headed by units of the category Center, denoting the type of entity or thing described by the whole unit. Elements in non-Scene units include Quantifiers (such as “dozens of people”) and Connectors (mostly coordinating conjunctions). Other modifiers to the Center are marked as Elaborators.

UCCA distinguishes primary edges, corresponding to explicit relations, from remote edges, which allow for a unit to participate in several super-ordinate relations. See example in Figure 6.1. Primary edges form a tree, whereas remote edges (dashed) enable reentrancy, forming a DAG.

**UD** is a syntactic dependency scheme used in many languages, aiming for cross-linguistically consistent and coarse-grained treebank annotation. Formally, UD uses bi-lexical trees, with edge labels representing syntactic relations.

4Despite the similar terminology, UCCA Adverbials are not necessarily adverbs syntactically.
After graduation, John moved to Paris.

Figure 6.1: UCCA graph. Dashed: remote edge.

One aspect of UD similar to UCCA is its preference of lexical (rather than functional) heads. For example, in auxiliary verb constructions (e.g., “is eating”), UD marks the lexical verb (eating) as the head, while other dependency schemes may select the auxiliary *is* instead. While the approaches are largely inter-translatable (Schwartz et al. 2012), lexical head schemes are more similar in form to semantic schemes, such as UCCA and semantic dependencies (Oepen et al. 2016).

Being a dependency representation, UD is structurally underspecified in an important way: it is not possible in UD to mark the distinction between an element modifying the head of the phrase and the same element modifying the whole phrase (de Marneffe and Nivre 2010).

An example UD tree is given in Figure 6.2. UD relations will be written in typewriter font.

Figure 6.2: UD tree.

### 6.3 Shared Gold-standard Corpus

We annotate 723 English passages (3,813 sentences; 52,721 tokens), comprising the web reviews section of the English Web Treebank (EWT; Bies et al. 2012). Text is annotated by two UCCA annotators according to v2.0 of the UCCA guidelines⁵ and cross-reviewed. As these sentences are included in the UD English_EWT treebank, this is a shared gold-standard UCCA and UD annotated corpus⁶. We use the standard train/development/test split, shown in Table 6.2.

⁶Our data is available at https://github.com/UniversalConceptualCognitiveAnnotation/UCCA_English-EWT
Table 6.2: Data split for the shared gold-standard corpus.

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td># Passages</td>
<td>347</td>
<td>192</td>
<td>184</td>
</tr>
<tr>
<td># Sentences</td>
<td>2,723</td>
<td>554</td>
<td>535</td>
</tr>
<tr>
<td># Tokens</td>
<td>44,804</td>
<td>5,394</td>
<td>5,381</td>
</tr>
</tbody>
</table>

6.4 Comparison Methodology

To facilitate comparison between UCCA and UD, we first assimilate the graphs by abstracting away from formalism differences, obtaining a similar graph format for both schemes. We then match pairs of nodes in the converted UD and UCCA trees if they share all terminals in their yields.

UD annotates bi-lexical dependency trees, while UCCA graphs contain non-terminal nodes. In §6.4.1, we outline the unified DAG converter by Hershcovich et al. (2018a,b), which we use to reach a common format. In §6.4.2, we describe a number of extensions to the converter, which abstract away from further non-content differences.

Figure 6.3: Converted UD tree. Non-terminals and head edges are introduced by the unified DAG converter.

6.4.1 Basic Conversion

Figure 6.3 presents the same tree from Figure 6.2 after conversion. The converter adds one pre-terminal per token, and attaches them according to the original dependency tree: traversing it from the root, for each head it creates a non-terminal parent with the edge label head, and adds the dependents as children of the created non-terminal. Relation subtypes are stripped, leaving only universal relations. For example, the language-specific definite article label det: def is replaced by the universal det.

Reentrancies. Remote edges in UCCA enable reentrancy, forming a DAG together with primary edges. UD allows reentrancy when including enhanced dependencies (Schuster and Manning, 2016) which form (bi-lexical) graphs, representing phenomena such as

7 https://github.com/huji-nlp/semstr
8 https://universaldependencies.org/u/overview/enhanced-syntax.html
as predicate ellipsis (e.g., gapping), and shared arguments due to coordination, control, raising and relative clauses.

UCCA is more inclusive in its use of remote edges, and accounts for the entire class of implicit arguments termed *Constructional Null Instantiation* in FrameNet ([Ruppenhofer et al.](#Ruppenhofer)) (2016). For example, in “The Pentagon is bypassing official US intelligence channels [...] in order to create strife” (from EWT), remote edges mark *Pentagon* as a shared argument of *bypassing* and *create*. Another example is “if you call for an appointment [...] so you can then make one”, where a remote edge in UCCA indicates that *one* refers to *appointment*. Neither is covered by enhanced UD.

In order to facilitate comparison, we remove remote edges and enhanced dependencies in the conversion process. We thus compare basic UD and UCCA trees, deferring a comparison of UCCA and enhanced UD to future work.

### 6.4.2 Extensions to the Converter

We extend the unified DAG converter to remove further non-content differences.

**Unanalyzable units.** An unanalyzable phrase is represented in UCCA as a single unit covering multiple terminals. In multi-word expressions (MWEs) in UD, each word after the first is attached to the previous word, with the flat, fixed or goeswith relations (depending on whether the expression is grammaticalized, or split by error). We remove edges of these relations and join the corresponding pre-terminals to one unit.

**Promotion of conjunctions.** The basic conversion generally preserves terminal yields: the set of terminals spanned by a non-terminal is the same as the original dependency yield of its head terminal (e.g., in Figure 6.3, the yield of the non-terminal headed by *graduation* is “After graduation”, the same as that of “graduation” in Figure 6.2).

Since UD attaches subordinating and coordinating conjunctions to the subsequent conjunct, this results in them being positioned in the same conjunct they relate (e.g., *After* will be included in the first conjunct in “After arriving home, John went to sleep”; *and* will be included in the second conjunct in “John and Mary”). In contrast, UCCA places conjunctions as siblings to their conjuncts (e.g., “[After] [arriving home], [John went to sleep]” and “[John] [and] [Mary]”).

To abstract away from these convention differences, we place coordinating and subordinating conjunctions (i.e., cc-labeled units, and mark-labeled units with an advcl head such as *when*, *if*, *after*) as siblings of their conjuncts.
Table 6.3: UD-UCCA confusion matrix calculated based on EWT gold-standard annotations from the training and development sets (§6.3), after applying our extended converter to UD (§6.4), by matching UD vertices and UCCA units with the same terminal yield. The last column (row), labeled **No Match**, shows the number of edges of each UD (UCCA) category that do not match any UCCA (UD) unit. Zero counts are omitted.

### 6.5 Analysis of Divergences

Using the shared format, we turn to analyzing the content differences between UCCA and UD.

#### 6.5.1 Confusion Matrix

Table 6.3 presents the confusion matrix of categories between the converted UD and UCCA, calculated over all sentences in the training and development sets of the shared EWT reviews corpus. We leave the test set out of this evaluation to avoid contamination for future parsing experiments.

In case of multiple UCCA units with the same terminal yield (i.e., units with a single non-remote child), we take the top category only, to avoid double-counting. Excluding punctuation, this results in 60,434 yields in UCCA and 58,992 in UD. Of these, 52,280 are common, meaning that a UCCA “parser” developed this way would get a very high F1 score of 87.6%, if it is provided with the gold UCCA label for every converted edge.

Some yields still have more than one UCCA category associated with them, due to edges with multiple categories (**A|P** and **A|S**). For presentation reasons, 0.15% of the

---

UCCA units in the data are not presented here, as they belong to rare (< 0.1%) multiple-category combinations.

Only 82.6% of UD’s syntactic arguments (ccomp, csubj, obj, nsubj, subj, obl and xcomp) are UCCA Participants, and only 72.9% of the Participants are syntactic arguments—a difference stemming from the Scene/non-Scene (§6.5.2) and argument/adjunct (§6.5.3) distinctions. Moreover, if we identify predicates as words having at least one argument and Scenes as units with at least one Participant, then only 92.1% of UD’s predicates correspond to Scenes (many are secondary relations within one scene), and only 80% of Scenes correspond to predicates (e.g., eventive nouns, which are not syntactic predicates).

Examining the head row in Table 6.3 allows us to contrast the schemes’ notions of a head. head-labeled units have at least one dependent in UD, or are single-clause sentences (technically, they are non-terminals added by the converter). Of them, 75.7% correspond to Processes, States, Parallel Scenes or Centers, which are UCCA’s notions of semantic heads, and 11.6% are left unmatched, mostly due to MWEs analyzed in UD but not in UCCA (§6.5.4). Another source of unmatched units is inter-Scene linkage, which tends to be flatter in UCCA (§6.5.5). The rest are mostly due to head swap (e.g., “all of Dallas”, where all is a Quantifier of Dallas in UCCA, but the head in UD).

In the following subsections, we review the main content differences between the schemes, as reflected in the confusion matrix, and categorize them according to the UD relations involved.

6.5.2 Scenes vs. Non-Scenes

UCCA distinguishes between Scenes and non-Scenes. This distinction crosses UD categories, as a Scene can be evoked by a verb, an eventive or stative noun (negotiation, fatigue), an adjective or even a preposition (“this is for John”).

Core syntactic arguments. Subjects and objects are usually Participants (e.g., “wine was excellent”). However, when describing a Scene, the subject may be a Process/State (e.g., “but service is very poor”). Some wh-pronouns are the subjects or objects of a relative clause, but are Linkers or Relators, depending on whether they link Scenes or non-Scenes, respectively. For example, “who” in “overall, Joe is a happy camper who has found a great spot” is an nsubj, but a Linker. Other arguments are Adverbials or Time (see §6.5.3), and some do not match any UCCA unit, especially when they are parts of MWEs (see §6.5.4).
Adjectival modifiers are Adverbials when modifying Scenes ("romantic dinner"), States when describing non-Scenes ("beautiful hotel") or when semantically predicative ("such a convenient location"), or Elaborators where defining inherent properties of non-Scenes ("medical school").

Nominal and clausal modifiers. Most are Participants or Elaborators, depending on whether they modify a Scene (e.g., “discount on services” and “our decision to buy when we did” are Participants, but “my car’s gears and brakes” and “Some of the younger kids that work there” are Elaborators). Unmatched acl are often free relative clauses (e.g., in “the prices were worth what I got”, what is the obj of worth but a Participant of I got).

Case markers. While mostly Relators modifying non-Scenes (e.g., “the team at Bradley Chevron”), some case markers are Linkers linking Scenes together (e.g., “very informative website with a lot of good work”). Others are Elaborators (e.g., “over a year”) or States when used as the main relation in verbless or copula clauses (e.g., “it is right on Wisconsin Ave”).

Coordination. Coordinating conjunctions (cc) are Connectors where they coordinate non-Scenes (e.g., “Mercedes and Dan”) or Linkers where they coordinate Scenes (e.g., “outdated but not bad”). Similarly, conjuncts and list elements (conj, list) may be Parallel Scenes (H), or Centers when they are non-Scenes.10

Determiners. Articles are Functions, but determiners modifying non-Scenes are Elaborators (e.g., “I will never recommend this gym to any woman”). Where modifying Scenes (mostly negation) they are marked as Adverbials. For example, “no feathers in stock”, “what a mistake”, and “the rear window had some leakage” are all Adverbials.

6.5.3 Primary and Secondary Relations

UD distinguishes core arguments, adverb modifiers, and obliques (in English UD, the latter mostly correspond to prepositional dependents of verbs). UCCA distinguishes Participants, including locations and abstract entities, from secondary relations (Adverbials), which cover manner, aspect and modality. Adverbials can be verbs (e.g., begin, fail), prepositional phrases (with disrespect), as well as modals, adjectives and adverbs.

10While in UD the conjunction cc is attached to the following conjunct, in UCCA coordination is a flat structure. This is a convention difference that we normalize (§6.4.2).
Adverbs and obliques. Most UD adverb modifiers are Adverbials (e.g., “I sometimes go”), but they may be Participants, mostly in the case of semantic arguments describing location (e.g., here). Obliques may be Participants (e.g., “wait for Nick”), Time (e.g., “for over 7 years”) or Adverbials–mostly manner adjuncts (by far).

Clausal arguments are Participant Scenes (e.g., “it was great that they did not charge a service fee”, “did not really know what I wanted” or “I asked them to change it”). However, when serving as complements to a secondary verb, they will not match any unit in UCCA, as it places secondary verbs on the same level as their primary relation. For example, to pay is an xcomp in “they have to pay”, while the UCCA structure is flat: have to is an Adverbial and pay is a Process. Single-worded clausal arguments may correspond to a Process/State, as in “this seems great”.

Auxiliary verbs are Functions (e.g., “do not forget”), or Adverbials when they are modals (e.g., “you can graduate”). Semi-modals in UD are treated as clausal heads, which take a clausal complement. For example, in “able to do well”, UD treats able as the head, which takes do well as an xcomp. UCCA, on the other hand, treats it as an Adverbial, creating a mismatch for xcomp.

6.5.4 Multi-Word Expressions

UD and UCCA treat MWEs differently. In UD they include names, compounds and grammaticalized fixed expressions. UCCA treats names and grammaticalized MWEs as unanalyzable units, but also a range of semantically opaque constructions (e.g., light verbs and idioms). On the other hand, compounds are not necessarily unanalyzable in UCCA, especially if compositional.

Compounds. English compounds are mostly nominal, and are a very heterogeneous category. Most compounds correspond to Elaborators (e.g., “industry standard”), or Elaborator Scenes (e.g., “out-of-place flat-screen TV”), and many are unanalyzable expressions (e.g., “mark up”). Where the head noun evokes a Scene, the dependent is often a Participant (e.g., “food craving”), but can also be an Adverbial (e.g., “first time buyers”) depending on its semantic category. Other compounds in UD are phrasal verbs (e.g., “figure out”, “cleaned up”), which UCCA treats as unanalyzable (leading to unmatched units).

Core arguments. A significant number of subjects and objects are left unmatched as they form parts of MWEs marked in UCCA as unanalyzable. UD annotates MWEs
involving a verb and its argument(s) just like any other clause, and therefore lacks this semantic content. Examples include light verbs (e.g., “give a try”), idioms (“bites the dust”), and figures of speech (e.g., “when it comes to”, “offer a taste (of)”), all are UCCA units.

**Complex prepositions.** Some complex prepositions (e.g., according to or on top of), not encoded as MWEs in UD, are unanalyzable in UCCA.

### 6.5.5 Linkage

**Head selection.** UCCA tends to flatten linkage, where UD, as a dependency scheme, selects a head and dependent per relation. This yields scope ambiguities for coordination, an inherently flat structure. For instance, “unique gifts and cards” is ambiguous in UD as to whether unique applies only to gifts or to the whole phrase—both annotated as in Figure 6.4a. UCCA, allowing non-terminal nodes, disambiguates this case (Figure 6.4b).

![Figure 6.4: Coordination in UD and UCCA.](image)

**Clausal dependents.** UD categorizes clause linkage into coordination, subordination, argumenthood (complementation), and parataxis. UCCA distinguishes argumenthood but conflates the others into the Parallel Scene category. For example, “We called few companies before we decided to hire them” and “Check out The Willow Lounge, you’ll be happy” are Parallel Scenes.

Note that while in UD, mark (e.g., before) is attached to the dependent adverbial clause, a UCCA Linker lies outside the linked Scenes. To reduce unmatched advcl instances, this convention difference is fixed by the converter (§6.4.2). Many remaining unmatched units are due to conjunctions we could not reliably raise. For instance, the marker to introducing an xcomp is ambiguous between Linker (purposive to) and Function (infinitive marker). Similarly, wh-pronouns may be Linkers (“he was willing to budge a little on the price which means a lot to me”), but have other uses in questions and free relative clauses. Other mismatches result from the long tail of differences in how UD and UCCA construe linkage. Consider the sentence in Figure 6.5. While moment is an oblique argument of know in UD, From the moment is analyzed as a Linker in UCCA.
6.5.6 Other Differences

**Appositions** in UD always follow the modified noun, but named entities in them are UCCA Centers, regardless of position (e.g., in “its sister store Peking Garden”, the UD head *its sister store* is an Elaborator, while *Peking Garden* is the Center).

**Copulas.** UCCA distinguishes copular constructions expressing identity (e.g., “This *is* the original Ham’s restaurant”) where the copula is annotated as State, and cases of attribution (e.g., “Mercedes and Dan *are* very thorough”) or location (e.g., “Excellent chefs *are* in the kitchen”), where the copula is a Function.

**Discourse markers and interjections.** Units relating a Scene to the speech event or to the speaker’s opinion are Ground (e.g., “*no*, Warwick in New Jersey” and “*Please* visit my website”). On the other hand, discourse elements that relate one Scene to another are Linkers (e.g., *anyway*).

**Vocatives** are both Ground and Participants if they participate in the Scene and are the party addressed. For example, *Mark* in “Thanks *Mark*” is both the person addressed and the one thanked.\(^\text{11}\)

**Expletives and subjects.** Expletives are generally Functions, but some instances of *it* and *that* are analyzed as *nsubj* in UD and as Function in UCCA (e.g., “*it’s* like driving a new car”).

**Excluded relations.** We exclude the following UD labels, as they are irrelevant to our evaluation: *root* (always matches the entire sentence); *punct* (punctuation is ignored in UCCA evaluation); *dep* (unspecified dependency), *orphan* (used for gapping, which is

\(^{11}\)The A|G column is omitted from Table 6.3 as this category combination occurs in only 0.02% of edges in the corpus.
represented using remote edges in UCCA—see §6.4.1; **fixed, flat and goeswith** (correspond to parts of unanalyzable units in UCCA, and so do not represent units on their own—see §6.4.2); **reparandum** and **dislocated** (too rare in EWT).

6.6 Fine-Grained UCCA Parsing Evaluation

In §6.5 we used our comparison methodology, consisting of the conversion to a shared format and matching units by terminal yield, to compare gold-standard UD and UCCA. In this section we apply the same methodology to parser outputs, using gold-standard UD for fine-grained evaluation.

6.6.1 Experimental Setup

**Data.** In addition to the UCCA EWT data (§6.3), we use the reviews section of the UD v2.3 English_EWT treebank ([Nivre et al. 2018](https://hdl.handle.net/11234/1-2895)) annotated over the exact same sentences. We additionally use UDPipe v1.2 ([Straka et al. 2016](https://hdl.handle.net/11234/1-2898)) [Straka and Straková 2017](https://hdl.handle.net/11234/1-2898), trained on English_EWT for feature extraction. We apply the extended converter to UD as before (§6.4.2).

**Parser.** We train TUPA v1.3 ([Hershcovich et al. 2017](https://hdl.handle.net/11234/1-2895) 2018a) on the UCCA EWT data, with the standard train/development/test split. TUPA uses POS tags and syntactic dependencies as features. We experiment both with using gold UD for feature extraction, and with using UDPipe outputs.

**Evaluation by gold-standard UD.** UCCA evaluation is generally carried out by considering a predicted unit as correct if there is a gold unit that matches it in terminal yield and labels. Precision, Recall and F-score (F1) are computed accordingly. For the fine-grained analysis, we split the gold-standard, predicted and matched UCCA units according to the labels of the UD relations whose dependents have the same terminal yield (if any).

6.6.2 Results

Table 6.4 presents TUPA’s scores on the UCCA EWT development and test sets. Surprisingly, using UDPipe for feature extraction results in better scores than gold syntactic tags and dependencies.
Table 6.4: Labeled precision, recall and F1 (in %) for primary and remote edges output by TUPA on the UCCA EWT development (top) and test (bottom) sets, using either gold-standard UD or UDPipe for TUPA features.

Table 6.5 shows fine-grained evaluation by UD relations. TUPA does best on auxiliaries and determiners, despite the heterogeneity of corresponding UCCA categories (see Table 6.3), possibly by making lexical distinctions (e.g., modals and auxiliary verbs are both UD auxiliaries, but are annotated as Adverbials and Functions, respectively).

Copulas and coordinating conjunctions pose a more difficult distinction, since the same lexical items may have different categories depending on the context: State/Function for copulas, due to the distinction between identity and attribution, and Connector/Linker for conjunctions, due to the distinction between Scenes and non-Scenes. However, the reviews domain imposes a strong prior for both (Function and Linker, respectively), which TUPA learns successfully.

Inter-clause linkage (conj, advcl, xcomp, ccomp, parataxis, acl and csubj) is a common source of error for TUPA. Although the match between UCCA and UD is not perfect in these cases, it is overall better than TUPA’s unlabeled performance, despite using gold-standard syntactic features. Our results thus suggest that encoding syntax more directly, perhaps using syntactic scaffolding (Swayamdipta et al., 2018) or guided attention (Strubell et al., 2018), may assist in predicting unit boundaries. However, TUPA often succeeds at making distinctions that are not even encoded in UD. For example, it does reasonably well (71%) on distinguishing between noun modifiers of Scene-evoking nouns (Participants) and modifiers of other nouns (Elaborators), surpassing a majority baseline based on the UD relation (51%). Lexical resources that distinguish eventive and relational nouns from concrete nouns may allow improving it even further. In the similar case of compounds, lexical resources for light verbs and idioms may increase performance.
Table 6.5: Fine-grained evaluation of TUPA (with gold-standard UD features) on the EWT development set. (a) Columns are sorted by labeled F1, measuring performance on each subset of edges. Unlabeled F1 ignores edge categories, evaluating unit boundaries only. (b) Total number of instances of each UD relation; of them, matching UCCA units in gold-standard and in TUPA’s predictions; their intersection, with/without regard to categories. (c) Percentage of correctly categorized edges; for comparison, percentage of most frequent category (see Table 6.3). (d) Average number of words in corresponding terminal yields.

6.7 Discussion

NLP tasks often require semantic distinctions that are difficult to extract from syntactic representations. Consider the example “after graduation, John moved to Paris” again. While graduation evokes a Scene (Figure 6.1), in UD it is an oblique modifier of moved, just like Paris is (Figure 6.2). The Scene/non-Scene distinction (§6.5.2) would assist structural text simplification systems in paraphrasing this sentence to two sentences, each one containing one Scene [Sulem et al., 2018a].

Another example is machine translation—translating the same sentence into Hebrew, which does not have a word for graduation, would require a clause to convey the same meaning. The mapping would therefore be more direct using a semantic representation, and we would benefit from breaking the utterance into two Scenes.

6.8 Related Work

The use of syntactic parsing as a proxy for semantic structure has a long tradition in NLP. Indeed, semantic parsers have leveraged syntax for output space pruning [Xue and Palmer, 2004], syntactic features [Gildea and Jurafsky, 2002; Hershcovitch et al., 2017], joint modeling [Surdeanu et al., 2008; Hajić et al., 2009], and multi-task learning [Swayamdipta et al., 2016b; 2018; Hershcovitch et al., 2018a]. Empirical comparison be-
 tween syntactic and semantic schemes, however, is still scarce. Rudinger and Van Durme (2014) mapped Stanford Dependencies (precursor to UD) to Hobbsian Logical Form, identifying semantic gaps in the former. PredPatt (White et al., 2016), a framework for extracting predicate-argument structures from UD, was evaluated by Zhang et al. (2017) on a large set of converted PropBank annotations. Szubert et al. (2018) proposed a method for aligning AMR and UD subgraphs, finding that 97% of AMR edges are evoked by one or more words or syntactic relations. Damonte et al. (2017) refined AMR evaluation by UD labels, similar to our fine-grained evaluation of UCCA parsing.

Some syntactic representation approaches, notably CCG (Steedman, 2000), directly reflect the underlying semantics, and have been used to transduce semantic forms using rule-based systems (Basile et al., 2012). A related line of work tackles the transduction of syntactic structures into semantic ones. Reddy et al. (2016) proposed a rule-based method for converting UD to logical forms. Stanovsky et al. (2016) converted Stanford dependency trees into proposition structures (PropS), abstracting away from some syntactic detail.

6.9 Conclusion

We evaluated the similarities and divergences in the content encoded by UD and UCCA. We annotated the reviews section of the English Web Treebank with UCCA, and used an automated methodology to evaluate how well the two schemes align, abstracting away from differences of mere convention. We provided a detailed picture of the content differences between the schemes. Notably, we quantified the differences between the notions of syntactic and semantic heads and arguments, finding substantial divergence between them. Our findings highlight the potential utility of using semantic parsers for text understanding applications (over their syntactic counterparts), but also expose challenges semantic parsers must address, and potential approaches for addressing them.

Acknowledgments

This work was supported by the Israel Science Foundation (grant No. 929/17), and by the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister’s Office. We thank Jakob Prange, Nathan Schneider and the anonymous reviewers for their helpful comments.
Chapter 7

Discussion

In this thesis, I showed that meaning representation is valuable for language understanding, and that TUPA, an accurate UCCA parser, is suited to many meaning representations. Furthermore, multitask learning allows useful shared generalizations to emerge, improving TUPA’s performance by taking advantage of its general transition-based architecture and flexible neural network classifier. As different meaning representations capture many similar distinctions, this approach proved effective, gaining from data annotated in each scheme even though they had been designed separately and with different formal properties. While divergences between the content of the schemes may limit the gain from sharing, they highlight relative strengths, which are meaningful both theoretically and practically.

7.1 Objectives

The objective of this thesis (see Section 1.3) were to develop techniques for graph parsing in general and UCCA in particular, comparing UCCA to other representation schemes, and hybridizing meaning representations and parsers to improve their performance. Indeed, in Chapter 3 an accurate and efficient UCCA parser was presented, and in Chapter 5 its applicability was demonstrated to Universal Dependencies. In Chapter 4 more representation frameworks were shown to be supported by the parser, and furthermore, transfer and multitask learning were shown to effectively improve parsing performance by taking advantage of shared generalizations. Finally, Chapter 6 provided a thorough qualitative and quantitative analysis of UCCA and Universal Dependencies, highlighting the content differences, the semantic divergences and points of similarity.

UCCA’s merits in providing a cross-linguistically applicable, broad-coverage annotation will support ongoing efforts to incorporate deeper semantic structures into a variety of applications, such as machine translation (Jones et al., 2012) and summarization (Liu
The advantage of UCCA as compared to syntactic annotation schemes for machine translation is apparent, as translation tends to preserve semantic structure more than syntactic structure (Sulem et al., 2015). Using UCCA as an intermediate representation is thus likely to achieve outputs that are more semantically similar to the source.

### 7.2 Challenges

At the initial stages of this work, dataset size was a concern. Models with a large number of parameters, such as the neural network models employed in TUPA, typically require very large training sets. Since the UCCA datasets are small in relation to other schemes, training neural network models on them seemed to pose a challenge. However, the current performance of TUPA in UCCA parsing is already quite satisfactory. This is demonstrated by the fact that the parser has been successfully used in various applications (Choshen and Abend, 2018; Sulem et al., 2018a,b). Furthermore, multitask learning proved to be an effective method to overcome the data scarcity issue. The UCCA data has also been slowly growing and extended to more languages by further labeling efforts, and results seem to improve as more and more training data is available.

While many distinctions are shared between UCCA and other semantic and syntactic schemes, they are largely obscured by differences in representation format and convention. A second challenge was thus in developing a generic parsing system that would be able to handle more than one semantic scheme. However, the general architecture of the parser was effective in handling these graphs structure. Conversion to a common graph format and assimilation of superficial structures addressed the generality question effectively to allow multitask learning, and further, allowed deep inspection of content divergences and convergences.

### 7.3 Further Analysis

#### 7.3.1 Benefit of Multitask Learning

To further quantify the improvements due to multitask learning, where the tasks of parsing multiple semantic representations are combined as auxiliary tasks for TUPA to improve UCCA parsing, Figure 7.1 offers a fine-grained analysis of the performance of different multitask models. The benefit of the multitask models over the single-task baseline is especially apparent for Connectors and Linkers, demonstrating the improved capability to parse coordination structures correctly, which are indeed relevant for all
meaning representations. Furthermore, while the labeled F1 for State is uniformly low due to the rarity of this category and its common confusion with Process, the multitask model with all auxiliary task is able to reach 40% labeled F1, showing it is able to generalize this difficult distinction between predicates describing events and attributes. In French and German, the improvement is again apparent for Connectors and Linkers, but now especially also for Parallel Scenes—these seem to be responsible for most of the improvement due to multitask learning in these languages. Again, this shows that the syntactic ability to split phrases and clauses to separate Scenes is greatly boosted by the auxiliary task (UD in this case).

Figure 7.2 shows fine-grained analysis by UD relations. Surprisingly, determiners seem to suffer from multitask learning, as the single-task baseline does best on them. While determiners in UCCA are mostly Elaborators (in the version of the corpora on which the experiment was made), they are mostly treated as vacuous semantically in DM and AMR, which could explain their poor representation in models trained with these auxiliary tasks. UD as an auxiliary greatly helps with this category in French and German, but also seems to deteriorate its treatment in English. Prepositions in German (bearing the case relation) are greatly improved by adding UD as an auxiliary. This is a common relation, spread over multiple UCCA categories. Learning to represent it more accurately improves performance across the board.

7.4 Ongoing Work

The ideas presented in this thesis offer many exciting opportunities for further research. Following are such directions, which I have started pursuing.

7.4.1 Combining Syntax with Lexical Semantics

In the comparison between Universal Dependencies (UD) and UCCA, 88% of edges were found to be common between the schemes (ignoring the label), meaning the linguistic structures annotated by them are very similar. Inspecting the remaining divergences reveals, for example, that only about 82% of UCCA unanalyzable units (i.e., units without a compositional internal structure) are even sub-trees in UD. The remaining cases seem to be almost exclusively multi-word Linkers, such as “even though”, “when it comes to” and “just because”. Furthermore, only 73% of Participants in UCCA Scenes were found to be arguments of syntactic predicates in UD, due to the differences in distinctions between Scenes/non-Scenes and between main relations, secondary relations and participants.

These gaps can perhaps be closed by complementing syntax with lexical semantics to
Figure 7.1: TUPA’s F1 per UCCA category in each single-/multitask setting.
Figure 7.2: TUPA’s F1 per UD relation in each single-/multitask setting.

95
make up for differences corresponding to semantic distinctions that are not expressed in UD. Lexical semantic resources, such as STREUSLE (Schneider, 2014; Walsh et al., 2018; Schneider et al., 2018; Blodgett and Schneider, 2018), may provide the necessary annotation to close the gap between UD and UCCA: for example, it contains labels for various types of multi-word expressions, even ones that are not UD sub-trees; and semantically categorize lexical items according to lexical category and supersense, distinguishing, for example, eventive from non-eventive nouns. This will also address the need for a comparison of UD enhanced dependencies and UCCA remote and implicit units, mentioned in Chapter 6.

7.4.2 Broad-coverage Semantic Parsing

While implicit units (see Section 1.2) are not supported by the original parser presented in Chapter 3, ongoing work, beyond the scope of this thesis, addresses the prediction of implicit arguments in UCCA and other meaning representations (Bender et al., 2011; Cheng and Erk, 2018; Petruck, 2019) with a general mechanism that can be used to augment a UCCA parser to support implicit unit prediction. Similarly, the phenomenon of elided predicates and null nodes in Universal Dependencies, mentioned in Chapter 5, overlaps with the phenomenon of implicit units (where in this case the main relation is implicit rather than an argument), and can be addressed by a similar technique.

7.4.3 Establishing the Meaning Representation Parsing Task

TUPA, the parser presented in this thesis, is the first UCCA parser. In other parsing tasks, years of experiments and progress have yielded very accurate and fine-tuned parsers. While TUPA is quite accurate, there can doubtlessly be countless improvements due to different ways of looking at the problem or a better selection of architecture and parameters. During January 2019, we (Zohar Aizenbud, Leshem Choshen, Elior Sulem, Omri Abend, Ari Rappoport and myself) ran a shared task as part of the International Workshop on Semantic Evaluation, titled Task 1: Cross-lingual Semantic Parsing with UCCA. The task presented participants with UCCA parsing challenges in English, German and French. The shared task has yielded improvements over TUPA in all languages and settings, with various approaches with respect to the parsing system, machine learning architecture, and cross-lingual transfer.

The task results were presented during SemEval 2019.

1 https://competitions.codalab.org/competitions/19160
2 http://alt.qcri.org/semeval2019/
Subsequently, I co-organized another shared task at the SIGNLL Conference on Computational Natural Language Learning\(^3\) (CoNLL 2019), on Cross-Framework Meaning Representation Parsing\(^4\). The task, organized jointly by Stephan Oepen, Omri Abend, Jan Hajič, Tim O’Gorman, Nianwen Xue and myself, involved parsing into a range of different semantic representation schemes (DM, PSD, EDS, UCCA and AMR), differing in both formal structure and linguistic approaches. By combining the different schemes in a single task, we established cross-framework meaning representation parsing as a task, and “blurred the boundaries” between meaning representations, enabling cross-fertilization. Furthermore, the task yielded an improved understanding of the commonalities and differences between the schemes, by systematic contrastive evaluation across frameworks. This shared task also addresses the prospective future work referred to in Chapter \(^4\) since a subset of the evaluation data is annotated in all included meaning representation frameworks, providing a controlled experiment for comparison without confounds of domain and text properties. Additionally, the best-performing parsers submitted to the shared task provide uniform algorithms and architectures competitive on all parsing tasks, rather than focusing on a particular task and treating the others as auxiliary. This is a significant step in the direction of a truly universal model for semantic parsing.

7.5 Conclusion

The comparison between different traditions in linguistic representation, supported by the technology developed in this research and by ongoing work, may contribute to linguistic theory by helping address questions such as how best to measure the difference in meaning expressed in translations, or differences between child-directed and adult language, or between different domains.

I see learning semantic parsing as a means for computers to learn language. While different representations focus on different distinctions and do so with formally different structures, they share an overall goal, which is to support the development of natural language processing systems that are aware of the meaning expressed in the processed text. The combined datasets annotated in each of these representations are an invaluable resource, which, used effectively, can greatly boost our achievements in language understanding and processing.

\(^3\)\url{http://www.conll.org/}
\(^4\)\url{http://mrp.nlpl.eu}
Bibliography


Leshem Choshen and Omri Abend. 2018. Reference-less measure of faithfulness for grammatical error correction In Proc. of NAACL-HLT.


Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In Proceedings of International Conference on Neural Networks (ICNN’96), volume 1, pages 347–352. IEEE.


Shexia He, Zuchao Li, Hai Zhao, and Hongxiao Bai. 2018. Syntax for semantic role labeling, to be, or not to be. In Proc. of ACL, pages 2061–2071.


Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Walied Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna


Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Atteh, Mohammed Attila, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Çebioghlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Silvie Cinková, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraz, Peter Dirix, Kaja Dobrovoljč, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richard Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Frei-


---

111


Sheng Zhang, Rachel Rudinger, and Benjamin Van Durme. 2017. An evaluation of PredPatt and open IE via stage 1 semantic role labeling. In IWCS.


Appendix A

A Transition-Based Directed Acyclic Graph Parser for UCCA
Supplementary Notes

A.1 Feature Templates

Figure A.1 presents the feature templates used by TUPA Sparse. All feature templates define binary features. The other classifiers use the same elements listed in the feature templates, but all categorical features are replaced by vector embeddings, and all count-based features are replaced by their numeric value.

For some of the features, we used the notion of head word, defined by the $h^*$ function (see Appendix A.4). While head words are not explicitly represented in the UCCA scheme, these features prove useful as means of encoding word-to-word relations.

A.2 Extended Presentation of UCCA

This work does not handle two important constructions in the UCCA foundational layer: Linkage, representing discourse relations, and Implicit, representing covert entities. Table A.1 shows the statistics of linkage nodes and edges and implicit nodes in the corpora.

**Linkage.** Figure A.2 demonstrates a linkage relation, omitted from Figure 3.1a. The linkage relation is represented by the gray node. LA is link argument, and LR is link relation. The relation represents the fact that the linker “After” links the two parallel scenes that are the arguments of the linkage. Linkage relations are another source of multiple parents for a node, which we do not yet handle in parsing and evaluation.
Table A.1: Statistics of linkage and implicit nodes in the Wiki and 20K Leagues UCCA corpora. Cf. Table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>Wiki</th>
<th>20K Leagues</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train Dev</td>
<td>Test</td>
</tr>
<tr>
<td># implicit</td>
<td>899 122 77</td>
<td>241</td>
</tr>
<tr>
<td># linkage</td>
<td>2956 263 359</td>
<td>376</td>
</tr>
<tr>
<td>edges</td>
<td>9276 803 1094</td>
<td>957</td>
</tr>
</tbody>
</table>

Figure A.2: UCCA example with linkage.

**Implicit units.** UCCA graphs may contain implicit units with no correspondent in the text. Figure A.3 shows the annotation for the sentence “A similar technique is almost impossible to apply to other crops, such as cotton, soybeans and rice.”. The sentence was used by Oepen et al. (2015) to compare between different semantic dependency schemes. It includes a single scene, whose main relation is “apply”, a secondary relation “almost impossible”, as well as two complex arguments: “a similar technique” and the coordinated argument “such as cotton, soybeans, and rice.” In addition, the scene includes an implicit argument, which represents the agent of the “apply” relation.

The parsing of these units is deferred to future work, as it is likely to require different methods than those explored in this paper (Roth and Frank, 2015).

### A.3 Hyperparameter Values

Table A.2 lists the hyperparameter values we found for the different classifiers by tuning on the development set. Note that learning rate decay is multiplicative and is applied at each epoch. Mini-batch size is in number of transitions, but a mini-batch must contain only whole sentences.
<table>
<thead>
<tr>
<th>Sparse</th>
<th>MLP</th>
<th>BiLSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding dimensions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>external word</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>word</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>POS tag</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>syntactic dep.</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>edge label</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>punctuation</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gap</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>action</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>training epochs</td>
<td>19</td>
<td>28</td>
</tr>
<tr>
<td>MinUPDATE</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>initial learning rate</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>learning rate decay</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>MLP #layers</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MLP layer dim.</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>LSTM #layers</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>LSTM layer dim.</td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>word dropout</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>dropout</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>weight decay</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td>mini-batch size</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table A.2: Hyperparameters used for the different classifiers.

### A.4 Bilexical Graph Conversion

Here we describe the algorithms used in the conversion referred to in Section 3.4.

**Notation.** Let $L$ be the set of possible edge labels. A UCCA graph over a sequence of tokens $w_1, \ldots, w_n$ is a directed acyclic graph $G = (V, E, \ell)$, where $\ell : E \rightarrow L$ maps edges to labels. For each token $w_i$ there exists a leaf (terminal) $t_i \in V$. A bilexical (dependency) graph over the same text consists of a set $A$ of labeled dependency arcs $(t', l, t)$ between the terminals of $G$, where $t'$ is the head, $t$ is the dependent and $l$ is the edge label.

**Conversion to bilexical graphs.** Let $G = (V, E, \ell)$ be a UCCA graph with labels $\ell : E \rightarrow L$. The conversion to a bilexical graph requires calculating the set $A$. All non-terminals in $G$ are removed.

We define a linear order over possible edge labels $L$ (see Figure A.4b). The priority order generally places core-like categories before adjunct-like ones, and was decided heuristically. For each node $u \in V$, denote by $h(u)$ its child with the highest-priority edge label. The leftmost edge is chosen in case of a tie. Let $h^*(u)$ be the terminal reached by recursively applying $h(\cdot)$ over $u$. For each terminal $t$, we define

$$N(t) = \{(u, v) \in E \mid t = h^*(v) \land t \neq h^*(u)\}$$

For each edge $(u, v) \in N(t)$, we add $h^*(u)$ as a head of $t$ in $A$, with the label $\ell(u, v)$. This procedure is given in Algorithm A.4a.
Data: UCCA graph $G = (V, E, \ell)$

Result: set $A$ of labeled bilexical arcs

$A \leftarrow \emptyset$

foreach $t \in \text{Terminals}(V)$ do

foreach $(u, v) \in N(t)$ do

$A \leftarrow A \cup \{(h^*(u), \ell(u, v), t)\}$

end

end

(a) Conversion to bilexical graphs.

(b) Priority order of edge labels used by $h(u)$.

Figure A.4

Note that this conversion procedure is simpler than the head percolation procedure used for converting syntactic constituency trees to dependency trees \cite{collins1997head}, since $h(u)$ (similar to $u$’s head-containing child) depends only on $\ell(u, h(u))$ and not on the subtree spanned by $u$, because edge labels in UCCA directly express the role of the child in the parent unit, and are thus sufficient for determining which of $u$’s children contains the head node.

Conversion from bilexical graphs. The inverse conversion introduces non-terminal nodes back into the graph. As the distinction between low- and high-attaching nodes is lost in the conversion, we assume that attachments are always low-attaching. Let $A$ be a the labeled arc set of a bilexical graph. Iterating over the terminals in topological order according to $A$, we add its members as terminals to graph and create a pre-terminal parent $u_t$ for each terminal $t$, with an edge labeled as $\text{Terminal}$ between them. The parents of the pre-terminals are determined by the terminal’s parent in the bilexical graph: if $t'$ is a head of $t$ in $A$, then $u_{t'}$ will be a parent of $u_t$. We add an intermediate node in between if $t$ has any dependents in $A$, to allow adding their pre-terminals as children later. Edge labels for the intermediate edges are determined by a rule-based function, denoted by $\text{Label}(t)$. This procedure is given in Algorithm 1.

A.5 Proof Sketch for Completeness of the TUPA Transition Set

Here we sketch a proof for the fact that the transition set defined in Section 3.3 is capable of producing any rooted, labeled, anchored DAG. This proves that the transition set is complete with respect to the class of graphs that comprise UCCA.
Let $G = (V, E, \ell)$ be a graph with labels $\ell : E \to L$ over a sequence of tokens $w_1, \ldots, w_n$. Parsing starts with $w_1, \ldots, w_n$ on the buffer, and the root node on the stack.

First we show that every node can be created, by induction on the node height: every terminal (height zero) already exists at the beginning of the parse (and so does the root node). Let $v \in V$ be of height $k$, and assume all nodes of height less than $k$ can be created. Take any (primary) child $u$ of $v$: its height must be less than $k$. If $u$ is a terminal, apply \textsc{Shift} until it lies at the head of the buffer. Otherwise, by our assumption, $u$ can still be created. Right after $u$ is created, it lies at the head of the buffer. A \textsc{Shift} transition followed by a \textsc{Node}_\ell(v, u)$ transition will move $u$ to the stack and create $v$ on the buffer, with the correct edge label.

Next, we show that every edge can be created. Let $(v, u) \in E$ be any edge with parent $v$ and child $u$. Assume $v$ and $u$ have both been created (we already showed that both are created eventually). If either $v$ or $u$ are in the buffer, apply \textsc{Shift} until both are in the stack. If both are in the stack but neither is at the stack top, apply \textsc{Swap} transitions until either moves to the buffer, and then apply \textsc{Shift}. Now, assume either $v$ or $u$ is at the stack top. If the other is not the second element on the stack, apply \textsc{Swap} transitions until it is. Finally, $v$ and $u$ are the top two elements on the stack. If they are in that order, apply \textsc{Right-Edge}_\ell(v, u)$ (or \textsc{Right-Remote}_\ell(v, u) if the edge between them is remote). Otherwise, apply \textsc{Left-Edge}_\ell(v, u)$ (or \textsc{Left-Remote}_\ell(v, u) if the edge between them is remote). This creates $(v, u)$ with the correct edge label.

Once all nodes and edges have been created, we can apply \textsc{Reduce} until only the root node remains on the stack, and then \textsc{Finish}. This yields exactly the graph $G$.

Note that the distinction we made between primary and remote transitions is suitable for UCCA parsing. For general graph parsing without this distinction, the \textsc{Remote} transitions can be removed, as well as the single-primary-parent restriction on \textsc{Edge} transition.
Figure A.1: Binary feature templates for TUPA\textsubscript{Sparse}. Notation:

- \( s_i, b_i \): \( i \)th stack and buffer items.
- \( w, t, d \): word form, POS tag and syntactic dependency label of the terminal returned by \( h^*() \) (see Appendix A.4).
- \( e \): edge label to the node returned by \( h() \).
- \( l, r (ll, rr) \): leftmost and rightmost (grand)children.
- \( u (uu) \): unary (grand)child, when only one exists.
- \( p \): unique separator punctuation between \( s_0 \) and \( s_1 \).
- \( q \): separator count.
- \( x \): gap type (“none”, “pass” or “gap”) at the sub-graph under the current node.
- \( y \): sum of gap lengths (Maier and Lichter 2009).
- \( P, C \): number of parents and children.
- \( R \): number of remote children.
- \( a_i \): action taken \( i \) steps back.

### Features from Zhang and Clark (2009):

**unigrams**

- \( s_{0tde}, s_{0we}, s_{1tde}, s_{1we}, s_{2tde}, s_{2we}, s_{3tde}, s_{3we}, b_{0wtd}, b_{1wtd}, b_{2wtd}, b_{3wtd}, s_{0wte}, s_{0wue}, s_{1wte}, s_{1wue}, s_{2wte}, s_{2wue}, s_{3wte}, s_{3wue} \)

**bigrams**

- \( s_{0ws1w}, s_{0ws1e}, s_{0es1w}, s_{0es1e}, s_{0wbo}, s_{0wbo0}d, s_{0eb0}w, s_{0eb0}td, s_{1wbo}, s_{1wbo0}d, s_{1eb0}w, s_{1eb0}td, b_{0wbo}w, b_{0wbo}td, b_{0db0}w, b_{0db0}td \)

**trigrams**

- \( s_{0es1s2w}, s_{0es1s2e}, s_{0es1eb0}w, s_{0es1eb0}td, s_{0wes1w}, s_{0wes1w0}d, s_{0wes1es}e, s_{0wes1eb0}td \)

**separator**

- \( s_{0wp}, s_{0wep}, s_{0wq}, s_{0wcp}, s_{0es1ep}, s_{0es1eq}, s_{1wp}, s_{1wep}, s_{1wq}, s_{1weq} \)

**extended** (Zhu et al. 2013)

- \( s_{0llwe}, s_{0lrwe}, s_{0lwe}, s_{0rwe}, s_{0rww}, s_{0sue}, s_{0tuwe}, s_{0tuwu}, s_{0tuwe}, s_{0llwe}, s_{1llwe}, s_{1luwe}, s_{1luwe}, s_{1ruwe}, s_{1rww}, s_{1rwe}, s_{1lrwe}, s_{1rww} \)

**disco** (Maier 2015)

- \( s_{0twe}, s_{1txe}, s_{2txe}, s_{3txe} \), \( s_{0xtde}, s_{1xtde}, s_{2xtde}, s_{3xtde} \), \( s_{0xy}, s_{1xy}, s_{2xy}, s_{3xy} \)

**counts**

- \( s_{0P}, s_{0C}, s_{0wP}, s_{0wC}, b_{0P}, b_{0C}, b_{0wP}, b_{0wC} \)

**edges**

- \( s_{0s1}, s_{1s0}, s_{0b0}, s_{0bo0}, s_{0s0e} \)

**history**

- \( a_0, a_1 \)

**remote** (Novel, UCCA-specific features)

- \( s_{0R}, s_{0wR}, b_{0R}, b_{0wR} \)

Figure A.3: UCCA example with an implicit unit.
**Data:** list $T$ of terminals, set $A$ of labeled bilexical arcs

**Result:** UCCA graph $G = (V, E, \ell)$

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$;

foreach $t \in \text{TopologicalSort}(T, A)$ do

$u_t \leftarrow \text{Node}()$;

$V \leftarrow V \cup \{u_t, t\}$, $E \leftarrow E \cup \{(u_t, t)\}$;

$\ell(u_t, t) \leftarrow \text{Terminal}$;

foreach $t' \in T, l \in L$ do

if $(t', l, t) \in A$ then

if $\exists t'' \in T, l' \in L : (t, l', t'') \in A$ then

$u \leftarrow \text{Node}()$;

$V \leftarrow V \cup \{u\}$, $E \leftarrow E \cup \{(u, u_t)\}$;

$\ell(u, u_t) \leftarrow \text{Label}(t)$;

else

$u \leftarrow u_t$;

end

$E \leftarrow E \cup \{(u_{t'}, u)\}$;

$\ell(u_{t'}, u) \leftarrow l$;

end

end

**Algorithm 1:** Conversion from bilexical graphs.

**Function Label**

**Data:** node $t \in T$

**Result:** label $l \in L$

if $\text{IsPunctuation}(t)$ then

return Punctuation;

else if $\exists t' \in T : (t, \text{ParallelScene}, t') \in A$ then

return ParallelScene;

else if $\exists t' \in T : (t, \text{Participant}, t') \in A$ then

return Process;

else

return Center;

end
Appendix B

Multitask Parsing Across Semantic Representations

Supplementary Notes

B.1 Features

Table B.1 lists all features used for the classifier (see §4.4.2). Numeric features are taken as they are, whereas categorical features are mapped to real-valued embedding vectors. For \( w \) features, we concatenate randomly-initialized and pre-trained word embeddings. For each node, we select a head terminal by traversing the graph according to a priority order on edge labels, taken from \cite{Hershcovich2017}.

\( s_i \) refers to stack node \( i \) from the top, and \( b_i \) to buffer node \( i \). \( x_l \) and \( x_r \) refer to a \( x \)'s leftmost and rightmost children, and \( x_L \) and \( x_R \) to its leftmost and rightmost parents.

\( w \) refers to the node’s head terminal text, \( t \) to its POS tag, and \( d \) to its dependency relation. \( h \) refers to the node’s height, \( e \) to the tag of its first incoming edge, \( n \) and \( c \) to the node label and category (used only for AMR), \( p \) to any separator punctuation between \( s_0 \) and \( s_1 \), \( q \) to the count of any separator punctuation between \( s_0 \) and \( s_1 \), \( x \) to the numeric value of gap type \cite{Maier2016}, \( y \) to the sum of gap lengths, \( P \), \( C \), \( I \), \( E \), and \( M \) to the number of parents, children, implicit children, remote children, and remote parents, \( N \) to the numeric value of the head terminal’s named entity IOB indicator, \( T \) to its named entity type, \# to its word shape (capturing orthographic features, e.g. ”Xxxx” or ”dd”), \( ^{\sim} \) to its one-character prefix, and \$ to its three-character suffix.

\( x \rightarrow y \) refers to the existing edge from \( x \) to \( y \). \( x \) is an indicator feature, taking the value of 1 if the edge exists or 0 otherwise, \( e \) refers to the edge label, and \( d \) to the dependency distance between the head terminals of the nodes.

\( a_i \) to the transition taken \( i + 1 \) steps ago. \( A \) refers to the action type label (e.g.
SHIFT/RIGHT-EDGE/NODE), and $e$ to the edge label created by the action (e.g. $C/E/P$).

The node ratio is the ratio between non-terminals and terminals, taken from Hershcovich et al. (2017).

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>wtdencT#*8xhyPCIEMN</td>
</tr>
<tr>
<td>$s_1$</td>
<td>wtdencT#*8hyn</td>
</tr>
<tr>
<td>$s_2$</td>
<td>wtdencT#*8hy</td>
</tr>
<tr>
<td>$b_0$</td>
<td>wtdncT#*8hPCIEMN</td>
</tr>
<tr>
<td>$b_1, b_2, b_3$</td>
<td>wenc#*8</td>
</tr>
<tr>
<td>$s_0, s_0L, s_0R, s_1, s_1l, s_1r, s_0dl, s_0dr, s_0rl, s_0rr, s_1ll, s_1lr, s_1rl, s_1rr$</td>
<td>wenc#*8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edges</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0 \rightarrow s_1, s_1 \rightarrow b_0$</td>
<td>xd</td>
</tr>
<tr>
<td>$s_1 \rightarrow s_0, b_0 \rightarrow s_0$</td>
<td>x</td>
</tr>
<tr>
<td>$s_0 \rightarrow b_0, b_0 \rightarrow s_0$</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Past actions</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0, a_1$</td>
<td>eA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Misc.</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ratio</td>
<td>node ratio</td>
</tr>
</tbody>
</table>

Table B.1: Transition classifier features.

### B.2 Conversion to and from Unified DAG Format

Although all experiments reported in the paper with the auxiliary tasks (AMR, DM and UD) are using unlabeled parsing for these schemes, our conversion code supports full conversion to and from these formats, and is publicly available at [http://github.com/danielhers/semstr/tree/master/semstr/conversion](http://github.com/danielhers/semstr/tree/master/semstr/conversion).

Conversion from AMR to the unifid DAG format and back results in 95% Smatch $F_1$ (Cai and Knight [2013]) when averaged over the LDC2017T10 test set. On SDP, the conversion is lossless and results in identical graphs when converted to UCCA and back. For UD, and conversion results in 98.5% LAS $F_1$ on the UD English test set, due to multi-word tokens, not supported in the unified DAG format.

### B.3 Qualitative evaluation

Figure B.1 shows an example sentence from the English 20K test set, with the outputs of both our single-task model and our best MTL model (using all auxiliaries). While the single-task model obtains an $F_1$ score of 67.9% on this sentence, the MTL model’s output matches the gold-annotates graph perfectly. This example demonstrates how the parser’s ability to identify syntactic constituents, which is important for all tasks we tackled, is improved with MTL.
Figure B.1: Output of single-task model on sentence 53001 from the English 20K test set (top), and of MTL model using all of AMR, DM and UD++ as auxiliaries on the same sentence (bottom).
Appendix C

Content Differences in Syntactic and Semantic Representation
Supplementary Material

C.1 UCCA Category Definitions

Table C.1 provides a concise description of the categories used by the UCCA foundational layer.
Scene Elements

P  Process  The main relation of a Scene that evolves in time (usually an action or movement).
P  State  The main relation of a Scene that does not evolve in time.
A  Participant  A participant in a Scene in a broad sense (including locations, abstract entities and Scenes serving as arguments).
D  Adverbial  A secondary relation in a Scene.
T  Time  A temporal relation in a Scene.

Elements of Non-Scene Units

C  Center  Necessary for the conceptualization of the parent unit.
E  Elaborator  A non-Scene relation which applies to a single Center.
N  Connector  A non-Scene relation which applies to two or more Centers, highlighting a common feature.
R  Relator  All other types of non-Scene relations. Two varieties: (1) Rs that relate a C to some super-ordinate relation, and (2) Rs that relate two Cs pertaining to different aspects of the parent unit.
Q  Quantifier  Describing the quantity or magnitude of something, or defines an entity as a group or a set (e.g., “two” or “a group of”).

Inter-Scene Relations

H  Parallel Scene  A Scene linked to other Scenes by regular linkage (e.g., temporal, logical, purposive).
L  Linker  A relation between two or more Hs (e.g., “when”, “if”, “in order to”).
G  Ground  A relation between the speech event and the uttered Scene (e.g., “surprisingly”, “in my opinion”).

Other

F  Function  Does not introduce a relation or participant. Required by the structural pattern it appears in.

Table C.1: The complete set of categories in UCCA’s foundational layer.

C.2  Universal Dependencies Category Definitions

Table C.2 lists the full names of the relation labels used by Universal Dependencies v2.
acl clausal modifier of noun (adjectival clause)
advc1 adverbia1 clause modifier
advmod adverbia1 modifier
amod adjectival modifier
appos appositional modifier
aux auxiliary
case case marking
cc coordinating conjunction
ccomp clausal complement
clf classifier
compound compound
conj conjunct
cop copula
csubj clausal subject
dep unspecified dependency
det determiner
discourse discourse element
dislocated dislocated elements
expl expletive

fixed fixed multiword expression
flat flat multiword expression
goeswith goes with
iobj indirect object
list list
mark marker
nmod nominal modifier
nsubj nominal subject
nummod numeric modifier
obj object
obl oblique nominal
orphan orphan
parataxis parataxis
punct punctuation
reparandum overridden disfluency
root root
vocative vocative
xcomp open clausal complement

Table C.2: UD v2 relations.
عونודו וז נטעתה חדדרכת של
פרופ' אריא רפואו ד"ר עמר אבנדה
ניהול סמסי אוניברסלי, באמעעוט רשקות קונורימ

חיבר לשם קבלת תואר דוקטור לפילוסופיה

נאות

דריאל הרשקוביץ

הוגש לסמ סמס אוניברסיטת העברית בירושלים

פברואר 2019
גיוון סמטים אוניברסלי, באמצעות עישון ניירום

 nowrap="false"

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>

hiro>