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

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

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

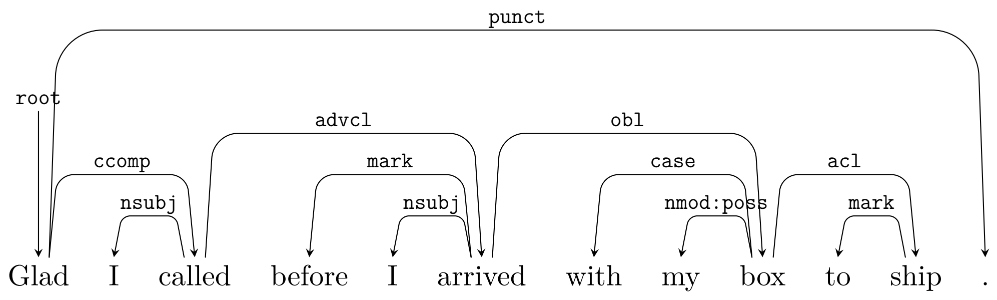


Figure 1.1: Example UD tree (`reviews-372665-0003` from UD_English-EWT), demonstrating practices common in semantic annotation: linking content words to content words, and preference of lexical heads over functional ones.

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[1] 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]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*.

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:

$$\boxed{\text{rest}} \approx \boxed{\text{take a break}}$$
$$\boxed{\text{graduation}} \approx \boxed{\text{סיים את הלימודים}}$$

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 (HPSG; 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 Grammer (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

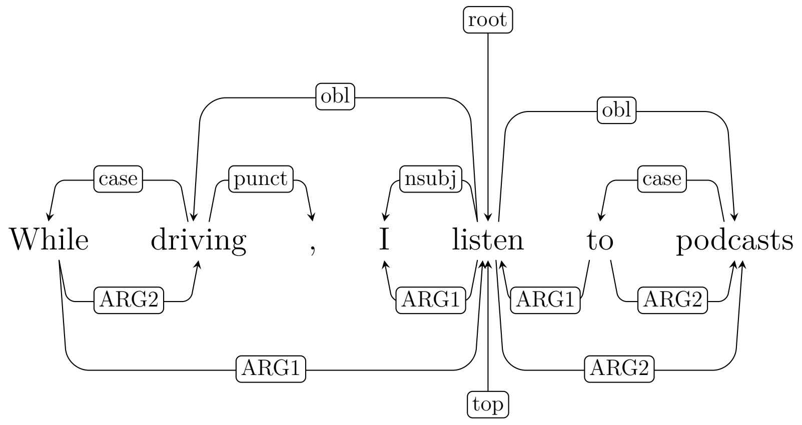


Figure 1.2: Top: syntactic dependencies in the Universal Dependencies framework. Bottom: semantic dependencies in the DM (DELPH-IN MRS) framework.

(DELPH-IN MRS), converted from DeepBank (Flickinger et al., 2012), a corpus of hand-corrected parses from LinGO ERG. It addresses a wide range of semantic phenomena, and supports discontinuous units and multiple parents (Oepen et al., 2016). 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 (Ivanova et al., 2012). 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; Banarescu et al., 2013; Flanigan et al., 2014; Vanderwende et al., 2015; Pust et al., 2015; Artzi et al., 2015; Wang et al., 2015b, 2016, 2015a; Zhou et al., 2016; Damonte et al., 2017; Damonte and Cohen, 2018). 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

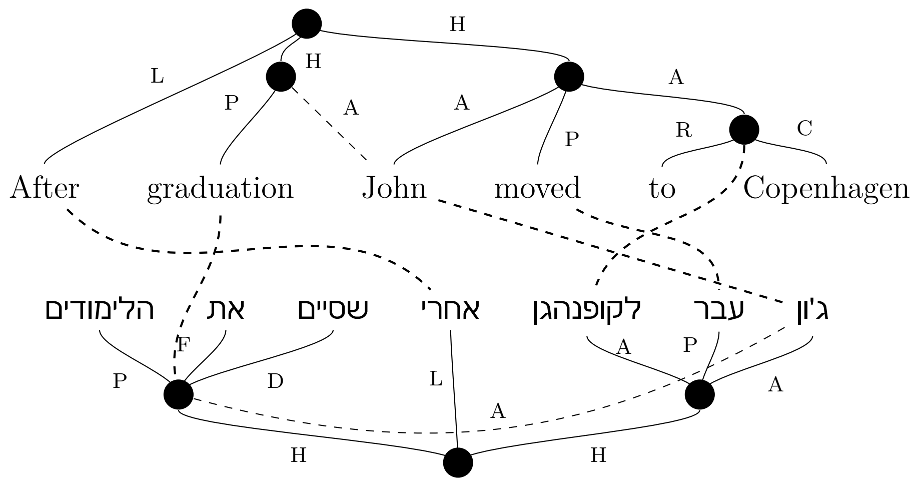


Figure 1.3: Illustration for the cross-linguistic applicability of UCCA.

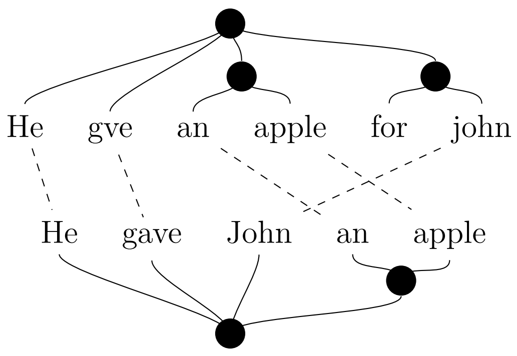


Figure 1.4: UCCA for grammatical error correction evaluation.

no or minimal assumptions about the relation between the syntactic and the semantic structures. This "purist" approach is attractive, as it avoids biases and language-specific properties that might be introduced by using a specific syntactic framework.

1.2 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

	Wiki	20K			EWT
	en	en	fr	de	en
# sentences	5,141	492	492	6,514	3,520
# tokens	158,739	12,638	13,021	144,529	51,042
# non-terminal nodes	62,002	4,699	5,110	51,934	18,156
% discontinuous	1.71	3.19	4.64	8.87	3.87
% reentrant	1.84	0.89	0.65	0.31	0.83
# edges	208,937	16,803	17,520	187,533	60,739
% primary	97.40	96.79	97.02	97.32	97.32
% remote	2.60	3.21	2.98	2.68	2.68

Table 1.1: UCCA data statistics.

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.[2] 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).

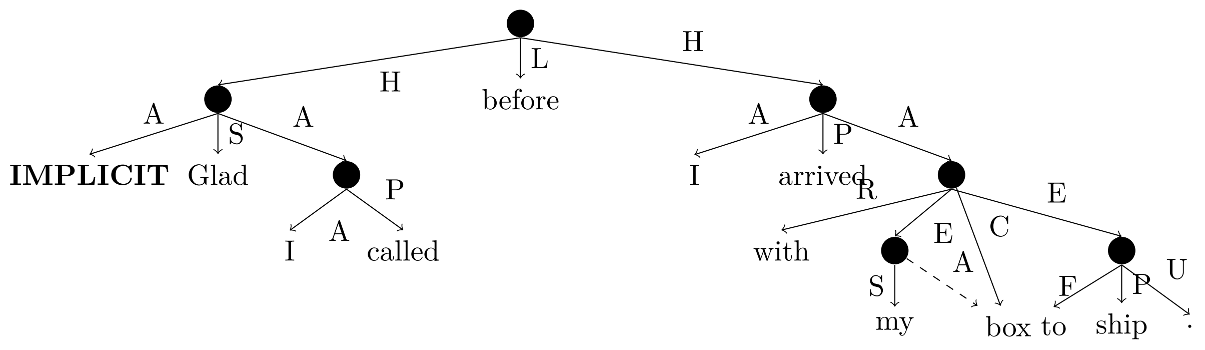


Figure 1.5: UCCA example including an implicit node, where the subject of "Glad" is omitted.

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

[2] `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.

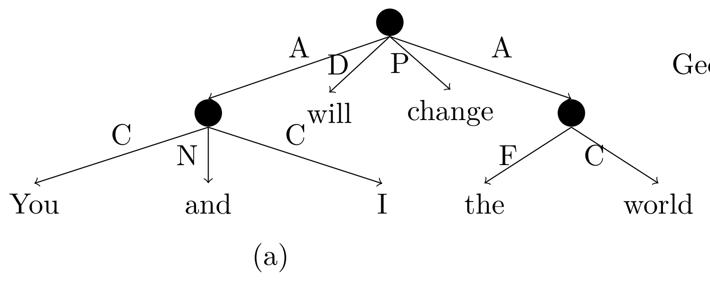
Scenes may be *linked* to one another in several ways. First, a Scene can provide information about some entity, in which case it is marked as an Elaborator. This often occurs in the case of participles or relative clauses. For example, “(child) who went to school” is an Elaborator Scene in “The child 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.



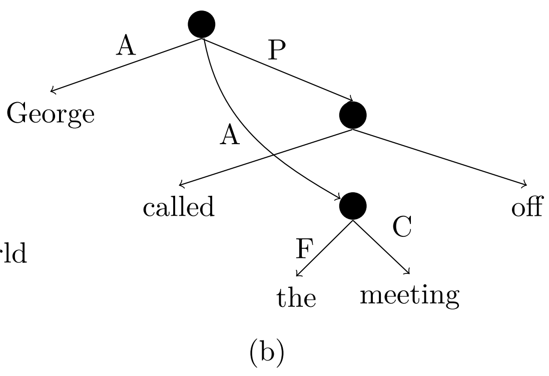


Figure 1.6: UCCA examples. (a) includes a coordination construction ("You and I"). (b) includes a discontinuous unit ("called ... off").

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 multiword expressions.

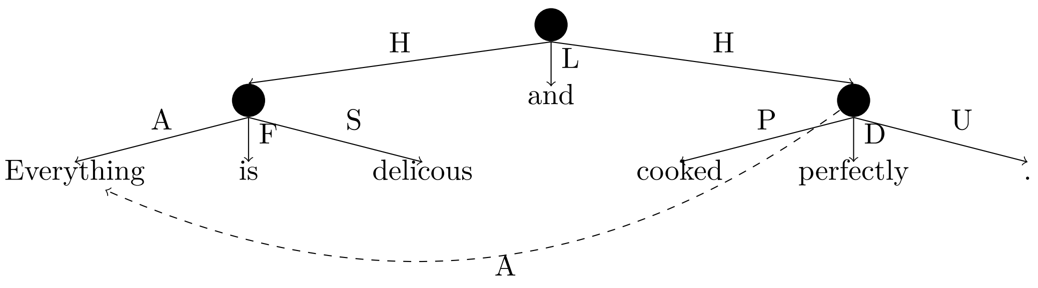


Figure 1.7: UCCA example including a remote edge (dashed), resulting in "Everything" having two parents.

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,

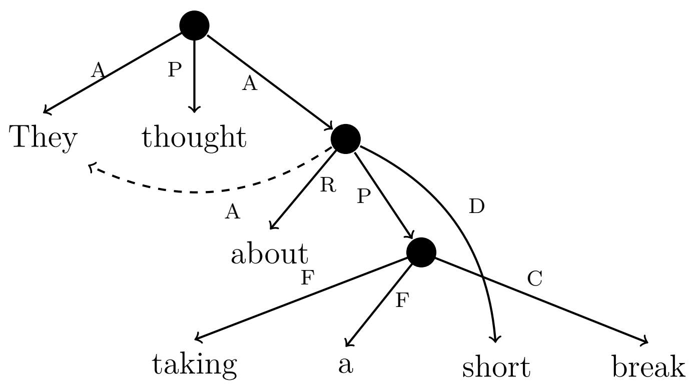


Figure 2.1: UCCA graph for the sentences "They thought about taking a short break".

2017; de Lhoneux et al., 2017). 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 discontiguous 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 discontiguous 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:

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

These transitions enable non-terminal nodes, reentrancy and discontinuity. To parse

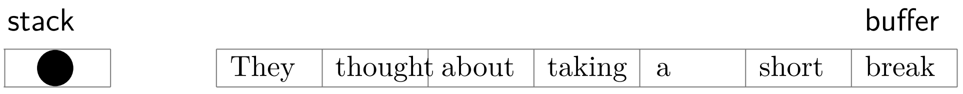


Figure 2.2: Initial state for parsing the sentence from Figure 2.1.

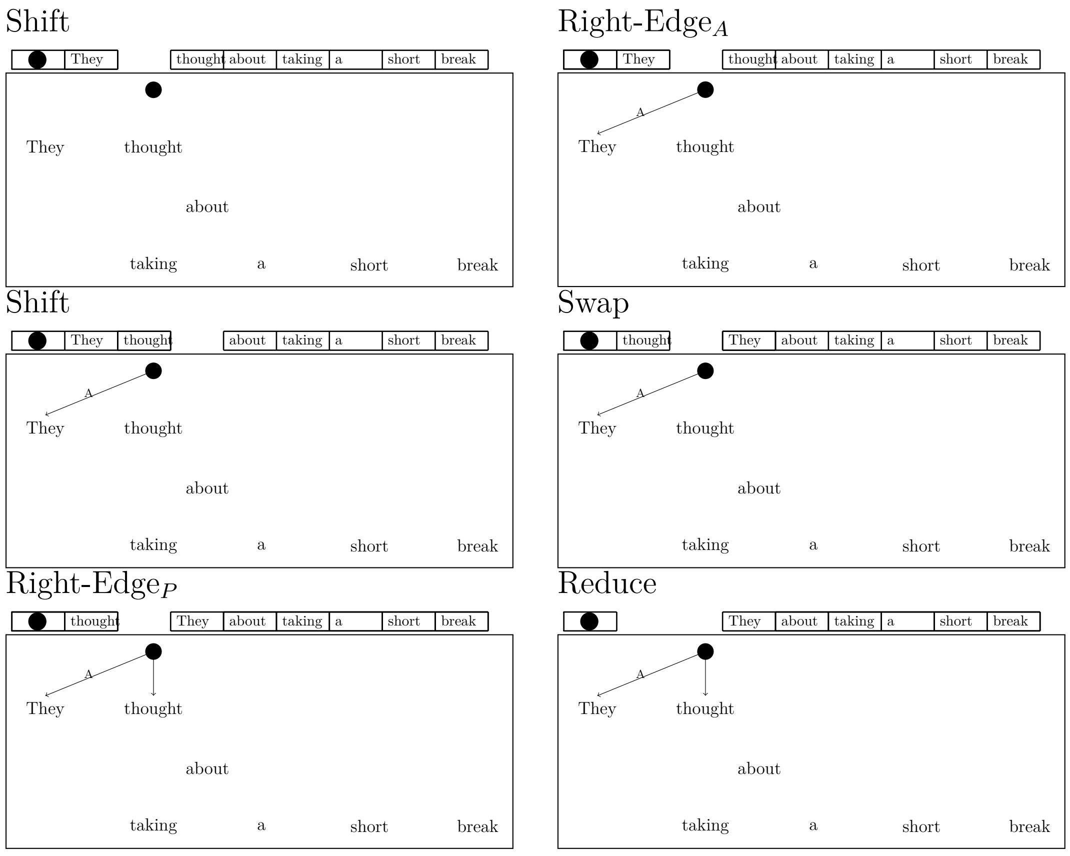


Figure 2.3: A few steps from a TUPA transition sequence.

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

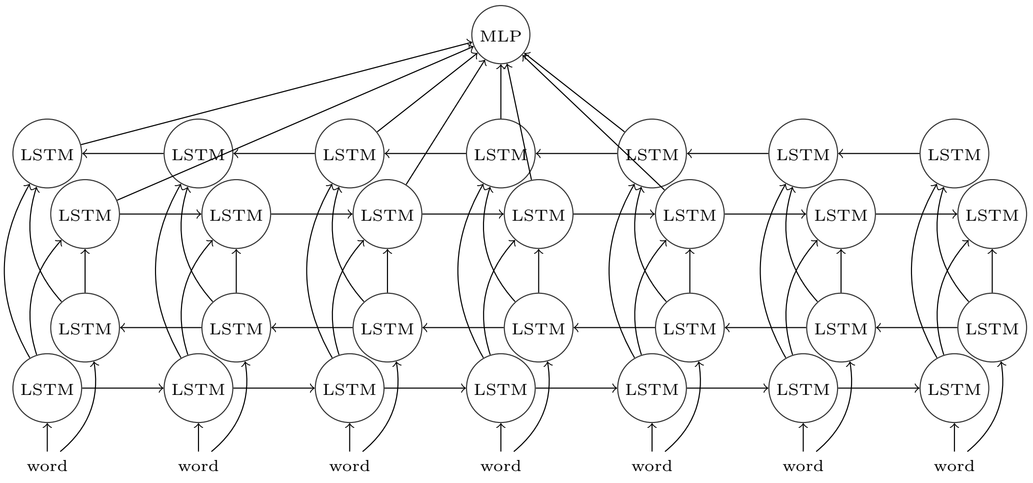


Figure 2.4: BiLSTM encoder with MLP classifier.

complicated function such as an LSTM (Long-Short Term Memory) cell, which supports additive gated updates for alleviating vanishing gradients:

$$\begin{aligned}
c_j &= c_{j-1} \odot f + g \odot i \\
h_j &= \tanh(c_j) \odot o \\
i &= \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
f &= \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
o &= \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \\
g &= \tanh(x_j W^{xg} + h_{j-1} W^{hg})
\end{aligned}$$

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:

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

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.

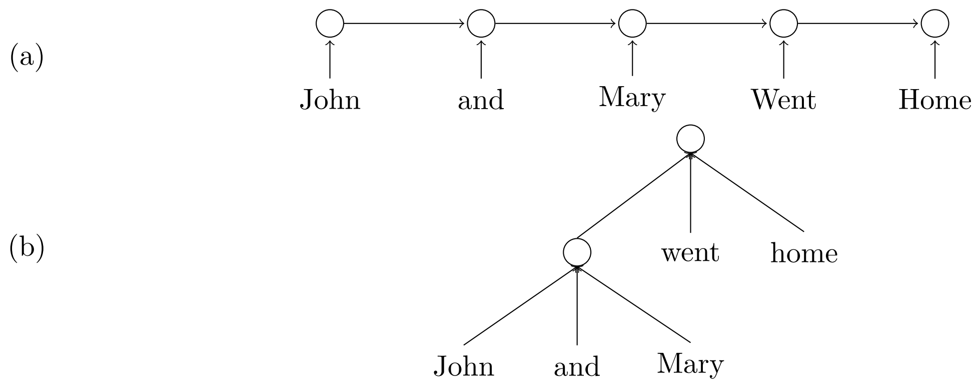


Figure 2.5: A recurrent neural network (a) combines distributed representations along a linear structure, whereas a recursive neural network (b) does so along a pre-determined hierarchical structure.

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).

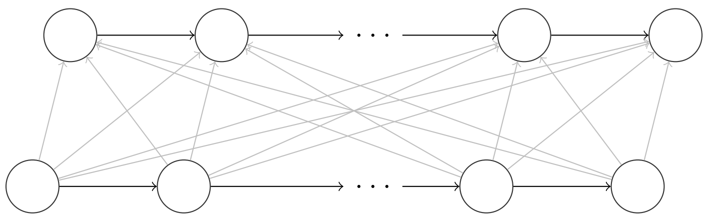
Training neural predictors for independent or sequential predictions is often done with the categorical cross-entropy loss (also referred to as *negative log likelihood*):

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

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 (Zou et al., 2013), or using memory and treating the text as a sequence to be converted to another sequence (Sutskever et al., 2014), 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 (Dagan et al., 2005; Bowman et al., 2015a), 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 (Bowman et al., 2015b, among others), combinatorical generalization in learning inevitably requires an inductive bias on the hypothesis class, realized as the network architecture (Mitchell, 1980; Battaglia et al., 2018). Specifically, scalable learning of meaning and inference requires an inductive bias on the compositional structure of language.



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 (Caruana, 1997) 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:

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

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[1,2]**, Omri Abend**[2] **and Ari Rappoport**[2]
[1]**The Edmond and Lily Safra Center for Brain Sciences**
[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; Tokgöz and Eryiğit, 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.[1]

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.

[1]All parsing and conversion code, as well as trained parser models, are available at `https://github.com/danielhers/tupa`.

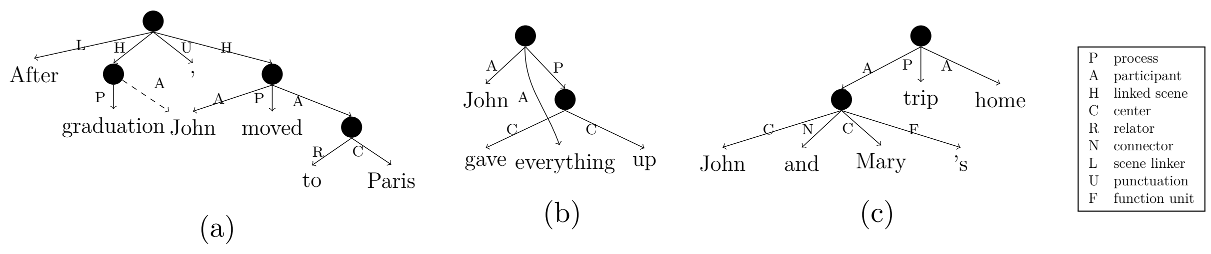


Figure 3.1: UCCA structures demonstrating three structural properties exhibited by the scheme. (a) includes a remote edge (dashed), resulting in "John" having two parents. (b) includes a discontinuous unit ("gave ... up"). (c) includes a coordination construction ("John and Mary"). Pre-terminal nodes 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

Before Transition				**Transition**	**After Transition**					**Condition**
Stack	**Buffer**	**Nodes**	**Edges**		**Stack**	**Buffer**	**Nodes**	**Edges**	**Terminal?**	
S	$x \mid B$	V	E	Shift	$S \mid x$	B	V	E	$-$	
$S \mid x$	B	V	E	Reduce	S	B	V	E	$-$	
$S \mid x$	B	V	E	Node$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \cup \{(y,x)_X\}$	$-$	$x \neq \text{root}$
$S \mid y,x$	B	V	E	Left-Edge$_X$	$S \mid y,x$	B	V	$E \cup \{(x,y)_X\}$	$-$	
$S \mid x,y$	B	V	E	Right-Edge$_X$	$S \mid x,y$	B	V	$E \cup \{(x,y)_X\}$	$-$	$\left\{ \begin{array}{l} x \notin w_{1:n}, \\ y \neq \text{root}, \\ y \not\leadsto_G x \end{array} \right.$
$S \mid y,x$	B	V	E	Left-Remote$_X$	$S \mid y,x$	B	V	$E \cup \{(x,y)_X^*\}$	$-$	
$S \mid x,y$	B	V	E	Right-Remote$_X$	$S \mid x,y$	B	V	$E \cup \{(x,y)_X^*\}$	$-$	
$S \mid x,y$	B	V	E	Swap	$S \mid y$	$x \mid B$	V	E	$-$	$\mathrm{i}(x) < \mathrm{i}(y)$
[root]	$\emptyset$	V	E	Finish	$\emptyset$	$\emptyset$	V	E	$+$	

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. $(\cdot,\cdot)_X$ denotes a primary X-labeled edge, and $(\cdot,\cdot)_X^*$ a remote X-labeled edge. $\mathrm{i}(x)$ 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 $\ell : E \to L$ 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), $\mathbf{TUPA_{Sparse}}$ 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.[2]

2. Changing the model to a feedforward neural network with dense embedding features, $\mathbf{TUPA_{MLP}}$ ("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, $\mathbf{TUPA_{BiLSTM}}$ 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 TUPA_{MLP}. The feedforward layers, BiLSTM and embeddings are all trained jointly.

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

Features. $\text{TUPA}_{\text{Sparse}}$ 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

[2]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.

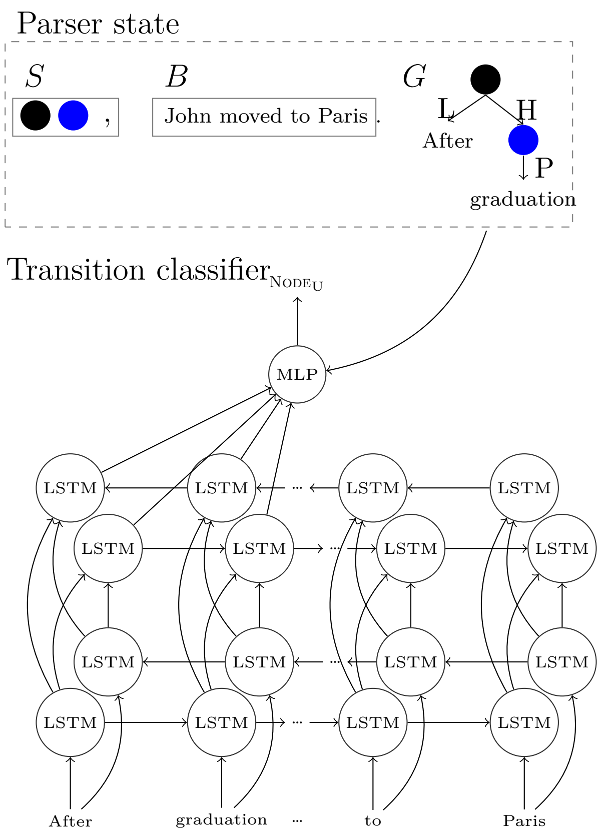


Figure 3.3: Illustration of the TUPA model. Top: parser state (stack, buffer and intermediate graph). Bottom: TUPA$_{\text{BiLTSM}}$ architecture. Vector representation for the input tokens is computed by two layers of bidirectional LSTMs. The vectors for specific tokens are concatenated with embedding 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.[3]

For TUPA$_{\text{MLP}}$ and TUPA$_{\text{BiLSTM}}$, 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),[4] 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.

[3]See Appendix A.1 for a full list of used feature templates.

[4]`https://goo.gl/6ovEhC`

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, **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.

Training. For training the transition classifiers, we use a dynamic oracle (Goldberg and Nivre, 2012), 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 NODE transition if the stack has on its top a parent in the gold graph that has not been created, but would predict a 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 (Honnibal and Johnson, 2015).[5] We use the categorical cross-entropy objective function and optimize the NN classifiers with the Adam optimizer (Kingma and Ba, 2014).

3.4 Experimental Setup

Data. We conduct our experiments on the UCCA Wikipedia corpus (henceforth, *Wiki*), and use the English part of the UCCA *Twenty Thousand Leagues Under the Sea* English-French parallel corpus (henceforth, *20K Leagues*) as out-of-domain data.[6] Table 3.1 presents some statistics for the two corpora. We use passages of indices up to 676 of the *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.[7] In the out-of-domain experiments, we apply the same parsers (trained on the *Wiki* training set) to the *20K Leagues* corpus without parameter re-tuning.

[5] `https://spacy.io`
[6] `http://cs.huji.ac.il/~oabend/ucca.html`
[7] Appendix A.2 further discusses linkage and implicit units.

	Wiki Train	Wiki Dev	Wiki Test	20K Leagues
# passages	300	34	33	154
# sentences	4268	454	503	506
# nodes	298,993	33,704	35,718	29,315
% terminal	42.96	43.54	42.87	42.09
% non-term.	58.33	57.60	58.35	60.01
% discont.	0.54	0.53	0.44	0.81
% reentrant	2.38	1.88	2.15	2.03
# edges	287,914	32,460	34,336	27,749
% primary	98.25	98.75	98.74	97.73
% remote	1.75	1.25	1.26	2.27
Average per non-terminal node				
# children	1.67	1.68	1.66	1.61

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:

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

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

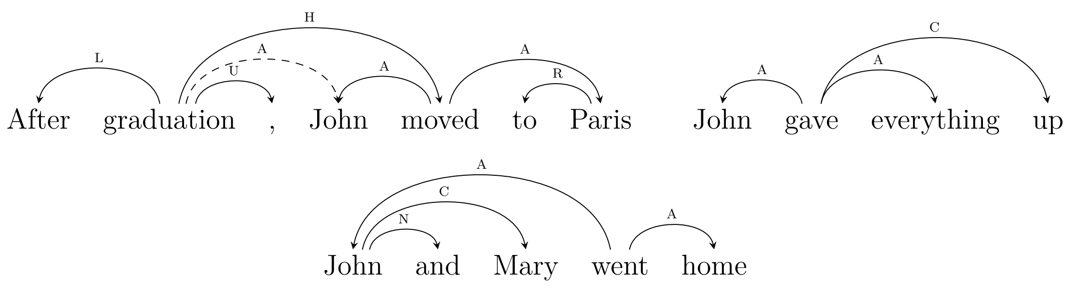


Figure 3.4: Bilexical graph approximation (dependency graph) for the sentences in Figure 3.1.

	Wiki (in-domain)						20K Leagues (out-of-domain)					
	Primary			Remote			Primary			Remote		
	LP	**LR**	**LF**	**LP**	**LR**	**LF**	**LP**	**LR**	**LF**	**LP**	**LR**	**LF**
TUPA$_{\text{Sparse}}$	64.5	63.7	64.1	19.8	13.4	16	59.6	59.9	59.8	22.2	7.7	11.5
TUPA$_{\text{MLP}}$	65.2	64.6	64.9	23.7	13.2	16.9	62.3	62.6	62.5	20.9	6.3	9.7
TUPA$_{\text{BiLSTM}}$	74.4	72.7	**73.5**	47.4	51.6	**49.4**	68.7	68.5	**68.6**	38.6	18.8	**25.3**
Bilexical Approximation (Dependency DAG Parsers)												
Upper Bound			91			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
Tree Approximation (Constituency Tree Parser)												
Upper Bound			100			–			100			–
UPARSE	60.9	61.2	61.1	–	–	–	52.7	52.8	52.8	–	–	–
Bilexical Tree Approximation (Dependency Tree Parsers)												
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. TUPA$_{\text{BiLSTM}}$ obtains the highest F-scores in all metrics, surpassing the bilexical parsers, tree parsers and other classifiers.

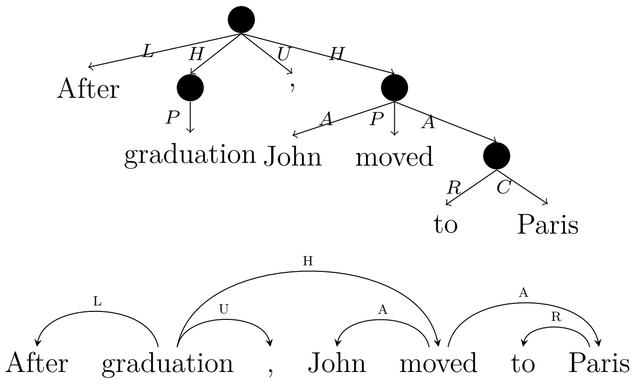


Figure 3.5: Tree approximation (constituency) for the sentence in Figure 3.1a (top), and bilexical tree approximation (dependency) for the same sentence (bottom). These are identical to the original graphs, apart from the removal of remote edges.

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.[8]

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.,

[8]See Appendix A.4 for a detailed description of the conversion procedures.

2014) 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.[9] 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.[10]

DAGParser and UPARSE are most directly comparable to TUPA$_{\text{Sparse}}$, as they also use a perceptron classifier with sparse features. TUPA$_{\text{Sparse}}$ 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$_{\text{MLP}}$ obtains higher scores than TUPA$_{\text{Sparse}}$, 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$_{\text{BiLSTM}}$ obtains substantially higher scores than TUPA$_{\text{MLP}}$ 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.[11] 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$_{\text{BiLSTM}}$.

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

[10]The 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.

[11]We 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[12] 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

[12]By *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.[13]

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

[13]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[1,2], Omri Abend[2] and Ari Rappoport[2]
[1]The Edmond and Lily Safra Center for Brain Sciences
[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.[1]

[1]`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 (Søgaard 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,b, 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 dominant dependency scheme for syntactic annotation in many languages, aiming for cross-linguistically consistent and coarse-grained treebank annotation. Formally, UD uses bilexical 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 English, which are generated by the Stanford CoreNLP converters (Schuster and Manning, 2016).[2] These include additional and augmented relations between content words, partially 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 originally developed for UCCA, as it supports all these structural properties. TUPA's transition 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

[2] `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.

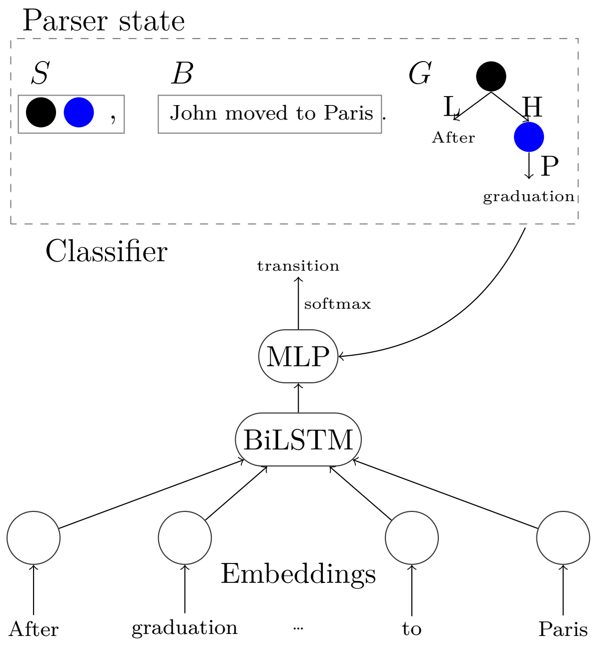


Figure 4.2: Illustration of the TUPA model, adapted from Hershcovich et al. (2017). Top: parser state. Bottom: BiLTSM architecture.

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,[3] all provided by spaCy (Honnibal and Montani, 2018).[4] To the learned word vectors, we concatenate the 250K most frequent word vectors from fastText (Bojanowski 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.[6]

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.[7]

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 *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 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 *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 *root*, where top nodes are labeled *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 *name*-labeled root has a child for each token in the name (see the two *name* nodes in Figure 4.1b). We collapse this subgraph into a single node whose children are the name tokens.

[6]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.

[7]See Supplementary Material for more conversion details.

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).

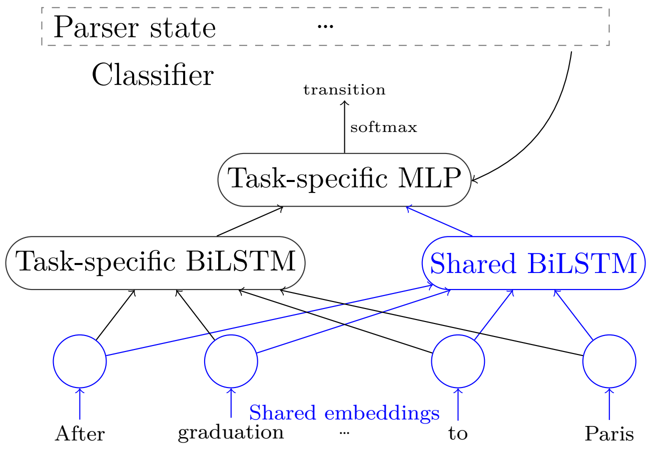


Figure 4.4: MTL model. Token representations are computed both by a task-specific and a shared BiLSTM. Their outputs are concatenated with the parser state embedding, identical to Figure 4.2, and fed into the task-specific MLP for selecting the next transition. Shared parameters are shown in blue.

	English						French						German					
	# tokens			# sentences			# tokens			# sentences			# tokens			# sentences		
	train	dev	test	train	dev	test	train	dev	test	train	dev	test	train	dev	test	train	dev	test
UCCA																		
Wiki	128444	14676	15313	4268	454	503												
20K			12339			506	10047	1558	1324	413	67	67	79894	10059	42366	3429	561	2164
AMR	648950			36521														
DM	765025			33964														
UD	458277			17062			899163			32347			268145			13814		

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.[8] 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).[9] For SDP, we use the DM representation from the SDP 2016 dataset (Oepen et al., 2016).[10] For Universal Dependencies, we use all English, French and German treebanks from UD v2.1 (Nivre et al., 2017).[11] 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.

[8]`http://github.com/huji-nlp/ucca-corpora`
[9]`http://catalog.ldc.upenn.edu/LDC2017T10`
[10]`http://sdp.delph-in.net/osdp-12.tgz`
[11]`http://hdl.handle.net/11234/1-2515`

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.[12] 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).[13]

		Multitask		
Hyperparameter	**Single**	**Main**	**Aux**	**Shared**
Pre-trained word dim.	300			300
Learned word dim.	200			200
POS tag dim.	20			20
Dependency relation dim.	10			10
Named entity dim.	3			3
Punctuation dim.	1			1
Action dim.	3			3
Edge label dim.	20	20		
MLP layers	2	2	1	
MLP dimensions	50	50	50	
BiLSTM layers	2	2		2
BiLSTM dimensions	500	300		300

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)

[12]We did not use AMR, DM or UD^{++} in French and German, as these are only available in English.
[13]`http://dynet.io`

	Primary			**Remote**		
	LP	**LR**	**LF**	**LP**	**LR**	**LF**
English (in-domain)						
HAR17	74.4	72.7	73.5	47.4	51.6	49.4
Single	74.4	72.9	73.6	53	50	51.5
AMR	74.7	72.8	73.7	48.7⋆	51.1	49.9
DM	75.7⋆	73.9⋆	74.8⋆	54.9	53	**53.9**
UD^{++}	75⋆	73.2	74.1⋆	49	52.7	50.8
AMR + DM	75.6⋆	73.9⋆	74.7⋆	49.9	53	51.4
AMR + UD^{++}	74.9	72.7	73.8	47.1	50	48.5
DM + UD^{++}	75.9⋆	73.9⋆	**74.9**⋆	48	54.8	51.2
All	75.6⋆	73.1	74.4⋆	50.9	53.2	52

Table 4.3: Labeled precision, recall and F_1 (in %) for primary and remote edges, on the **Wiki** test set. ⋆ indicates significantly better than *Single*. HAR17: Hershcovich et al. (2017).

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).[14] 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.

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 $p < 0.05$.

	Primary			Remote		
	LP	**LR**	**LF**	**LP**	**LR**	**LF**
English (out-of-domain)						
HAR17	68.7	68.5	68.6	38.6	18.8	25.3
Single	69	69	69	41.2	19.8	26.7
AMR	69.5	69.5	69.5	42.9	20.2	27.5
DM	70.7⋆	70.7⋆	70.7⋆	42.7	18.6	25.9
UD^{++}	69.6	69.8⋆	69.7	41.4	22	28.7
AMR + DM	70.7⋆	70.2⋆	70.5⋆	45.8	19.4	27.3
AMR + UD^{++}	70.2⋆	69.9⋆	70⋆	45.1	21.8	29.4
DM + UD^{++}	70.8⋆	70.3⋆	70.6⋆	41.6	21.6	28.4
All	71.2⋆	70.9⋆	**71**⋆	45.1	22	**29.6**
French (in-domain)						
Single	68.2	67	67.6	26	9.4	13.9
UD	70.3	70⋆	**70.1**⋆	43.8	13.2	20.3
German (in-domain)						
Single	73.3	71.7	72.5	57.1	17.7	27.1
UD	73.7⋆	72.6⋆	**73.2**⋆	61.8	24.9⋆	**35.5**⋆

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: Hershcovich 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; Hershcovich 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.

	20K	AMR	DM	UD
Wiki	1.047	0.895	0.913	0.843
20K		0.949	0.971	0.904
AMR			0.757	0.469
DM				0.754

Table 4.5: L1 distance between dataset word distributions, quantifying domain differences in English (low is similar).

	Primary			**Remote**		
	UP	**UR**	**UF**	**UP**	**UR**	**UF**
AMR	53.8	15.6	24.2	7.3	5.5	6.3
DM	65	49.2	56	7.4	65.9	13.3
UD^{++}	82.7	84.6	83.6	12.5	12.7	12.6

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.

English) and 1% in German, where the training set is comparable in size to the English one, but is noisier (see §4.7). The best MTL models are significantly better than single-task models, demonstrating that even a small training set for the main task may suffice, given enough auxiliary training data (as in French).

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`

[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`.

format.

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.

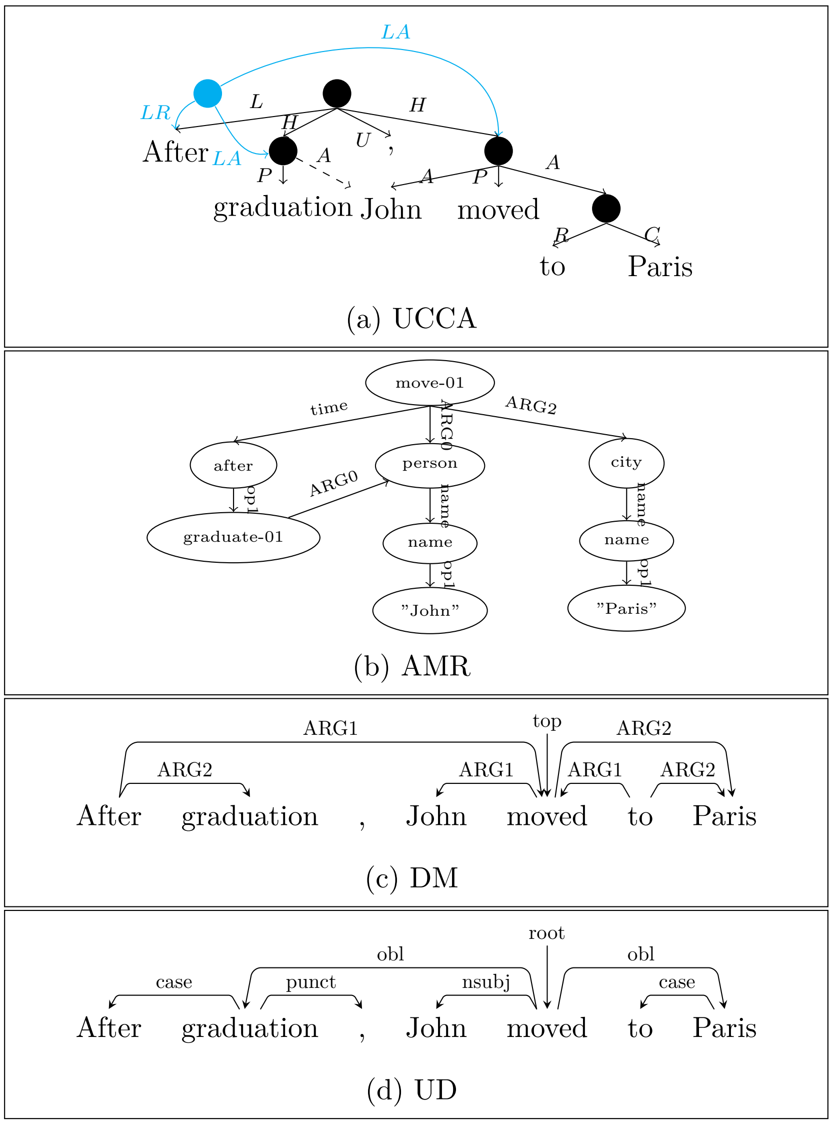


Figure 4.1: Example graph for each task. Figure 4.1a presents a UCCA graph. The dashed edge is remote, while the blue node and its outgoing edges represent inter-Scene linkage. Pre-terminal nodes and edges are omitted for brevity. Figure 4.1b presents an AMR graph. Text tokens are not part of the graph, 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.

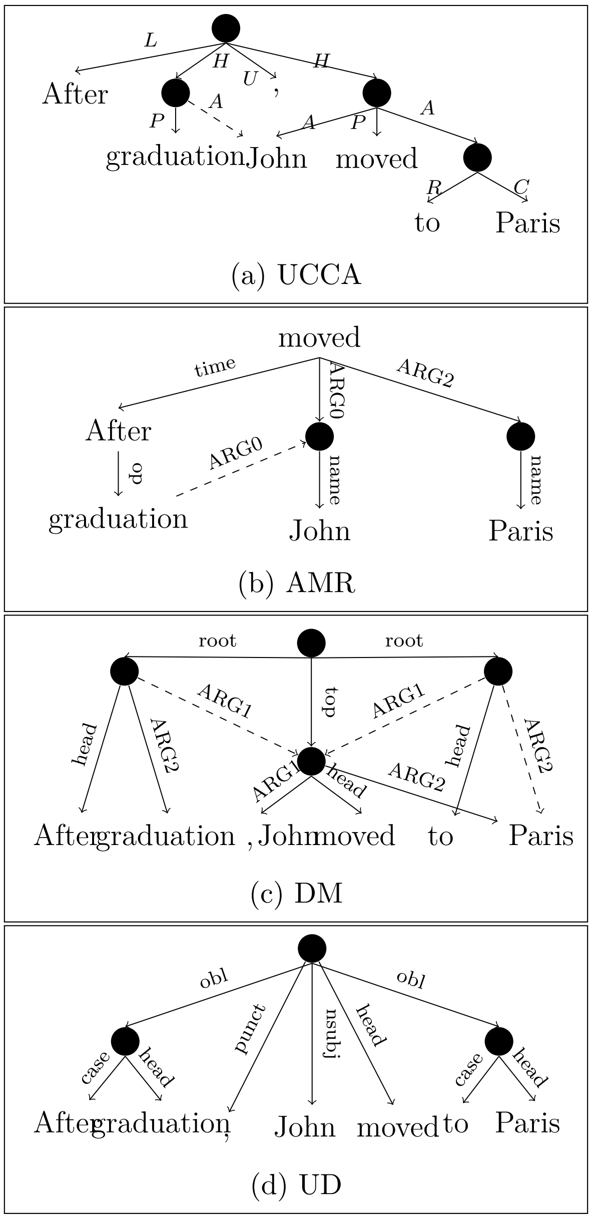


Figure 4.3: Graphs from Figure 4.1, after conversion to the unified DAG format (with pre-terminals omitted: each terminal drawn in place of its parent). Figure 4.3a presents a converted UCCA graph. Linkage nodes and edges are removed, but the original graph is otherwise preserved. Figure 4.3b presents 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[1,2]**, Omri Abend**[2] **and Ari Rappoport**[2]
[1]**The Edmond and Lily Safra Center for Brain Sciences**
[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 `https://github.com/CoNLL-UD-2018/HUJI`.

5.1 Introduction

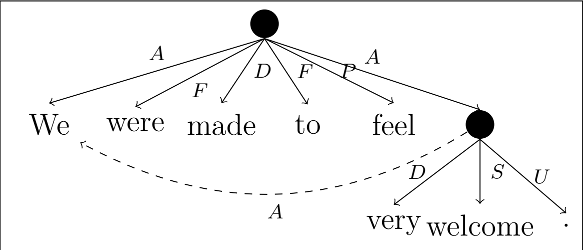
In this paper, we present the HUJI submission to the CoNLL 2018 shared task on Universal Dependency parsing (Zeman et al., 2018). We focus only on parsing, using the baseline system, UDPipe 1.2 (Straka et al., 2016; Straka and Straková, 2017) for tokenization, sentence splitting, part-of-speech tagging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

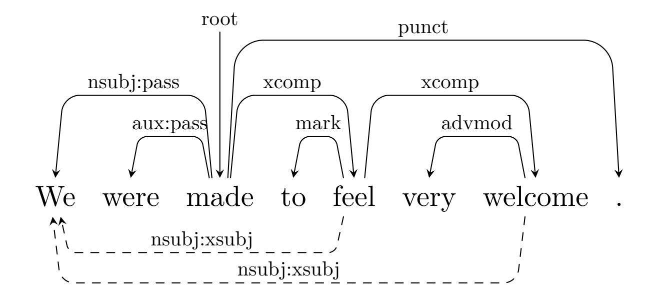
Enhanced dependencies. Our method treats *enhanced dependencies*[1] as part of the dependency graph, providing the first approach, to our knowledge, for supervised learning of enhanced UD parsing. Due to the scarcity of enhanced dependencies in UD treebanks, previous approaches (Schuster and Manning, 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.

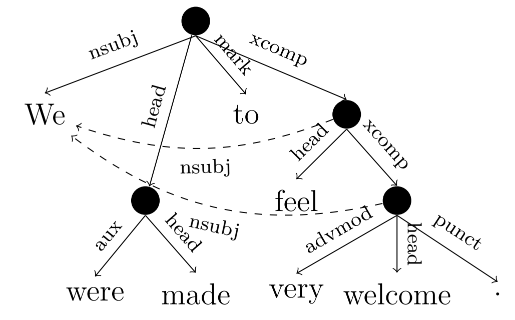
[1]`http://universaldependencies.org/u/overview/enhanced-syntax.html`



(a) Example UCCA graph.



(b) Example UD graph.



(c) UD graph after conversion to unified DAG format.

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.

```
1 We we PRON PRP Case=Nom|Number=Plur|Person=1|PronType=Prs 3 nsubj:pass 3:nsubj:pass|5:nsubj:xsubj|7:nsubj:xsubj _
```

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 (Hershcovich et al., 2018a). The format consists of a rooted DAG, where the tokens are the terminal nodes.[2] 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

[2]Our conversion code supports full conversion between UCCA and UD, among other representation schemes, and is publicly available at `http://github.com/danielhers/semstr/tree/master/semstr/conversion`.

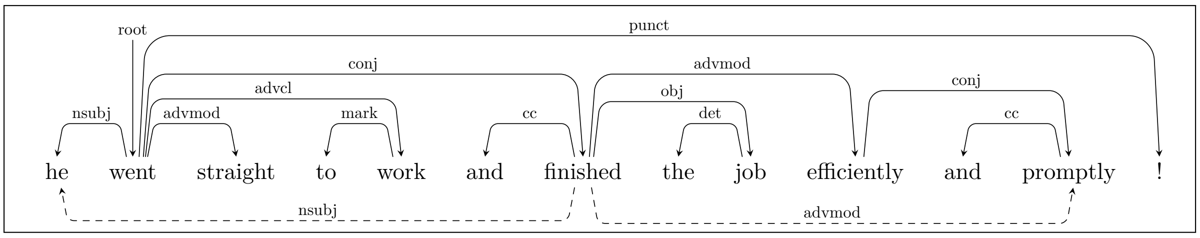


Figure 5.3: UD graph from `reviews-341397-0003` (`UD_English-EWT`), containing conjoined predicates and arguments.

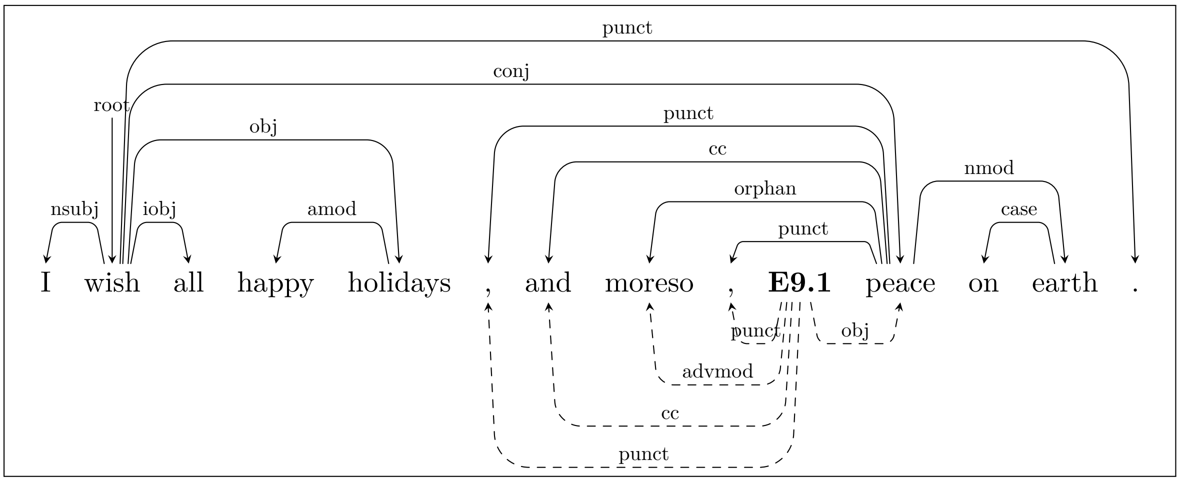


Figure 5.4: `newsgroup-groups.google.com_GuildWars_086f0f64ab633ab3_ENG_20041111_173500-0051` (`UD_English-EWT`), containing a null node (**E9.1**) and case information (nmod:on).

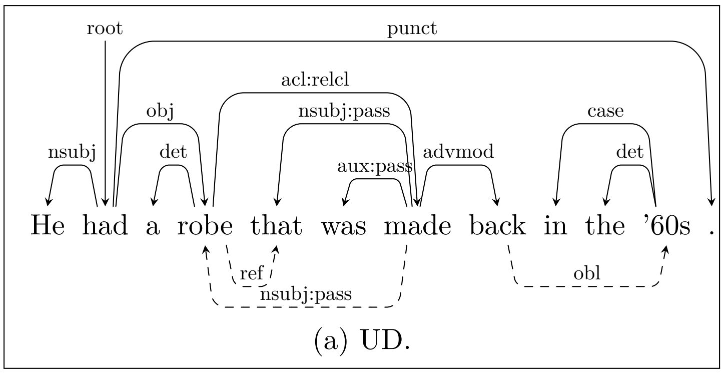
English Web Treebank (Silveira et al., 2014)[3].

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 dependnecies 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

[3]`https://catalog.ldc.upenn.edu/LDC2012T13`



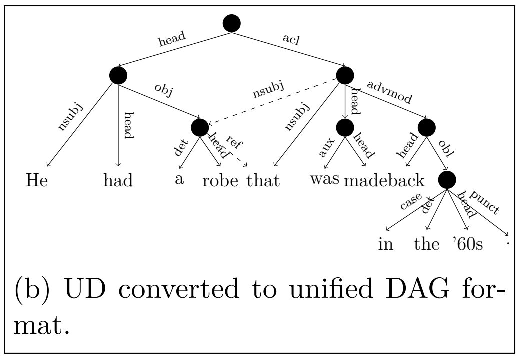


Figure 5.5: (a) `reviews-255261-0007` (`UD_English-EWT`), containing a relative clause, and (b) the same graph after conversion to the unified DAG format. The cycle is removed due to the non-terminal nodes introduced in the conversion.

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" $\rightarrow$ "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" $\leftrightarrow$ "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

Before Transition				Transition	After Transition					Condition
Stack	**Buffer**	**Nodes**	**Edges**		**Stack**	**Buffer**	**Nodes**	**Edges**	**Terminal?**	
S	$x \mid B$	V	E	Shift	$S \mid x$	B	V	E	$-$	
$S \mid x$	B	V	E	Reduce	S	B	V	E	$-$	
$S \mid x$	B	V	E	Node$_X$	$S \mid x$	$y \mid B$	$V \cup \{y\}$	$E \cup \{(y,x)_X\}$	$-$	$x \neq \text{root}$
$S \mid y, x$	B	V	E	Left-Edge$_X$	$S \mid y, x$	B	V	$E \cup \{(x,y)_X\}$	$-$	
$S \mid x, y$	B	V	E	Right-Edge$_X$	$S \mid x, y$	B	V	$E \cup \{(x,y)_X\}$	$-$	$\left\{ x \notin w_{1:n},\right.$
$S \mid y, x$	B	V	E	Left-Remote$_X$	$S \mid y, x$	B	V	$E \cup \{(x,y)^*_X\}$	$-$	$y \neq \text{root},$
$S \mid x, y$	B	V	E	Right-Remote$_X$	$S \mid x, y$	B	V	$E \cup \{(x,y)^*_X\}$	$-$	$y \not\to_G x$
$S \mid x, y$	B	V	E	Swap	$S \mid y$	$x \mid B$	V	E	$-$	$\mathrm{i}(x) < \mathrm{i}(y)$
[root]	$\emptyset$	V	E	Finish	$\emptyset$	$\emptyset$	V	E	$+$	

Figure 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. $\mathrm{i}(x)$ 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.

the set of *nodes*, E is the set of *edges*, and $\ell : E \to L$ 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.

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$_X$ 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.

The Remote$_X$ 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

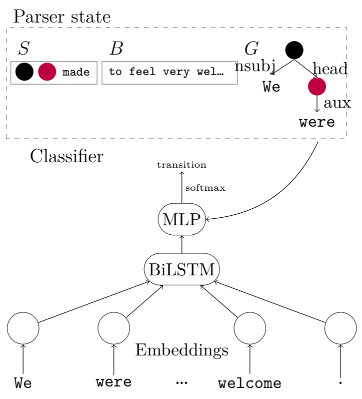


Figure 5.7: Illustration of the TUPA model, adapted from Hershcovich et al. (2018a). Top: parser state (stack, buffer and intermediate graph). Bottom: BiLTSM architecture. Vector representation for the input tokens is computed by two layers of bidirectional LSTMs. The vectors for specific tokens are 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.

trained to maximize the sum of log-likelihoods of all optimal transitions at each step.

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;[4] 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),[5] 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`

[4]`http://spacy.io`
[5]`http://fasttext.cc`

Nodes	
s_0	`wmtuepT#^$xhqyPCIEMN`
s_1	`wmtueT#^$xhyN`
s_2	`wmtueT#^$xhy`
s_3	`wmtueT#^$xhyN`
b_0	`wmtuT#^$hPCIEMN`
b_1, b_2, b_3	`wmtuT#^$`
$s_0l, s_0r, s_1l, s_1r,$ $s_0ll, s_0lr, s_0rl, s_0rr,$ $s_1ll, s_1lr, s_1rl, s_1rr$	`wme#^$`
$s_0L, s_0R, s_1L,$ s_1R, b_0L, b_0R	`wme#^$`
Edges	
$s_0 \to s_1, s_0 \to b_0,$ $s_1 \to s_0, b_0 \to s_0$	`x`
$s_0 \to b_0, b_0 \to s_0$	`e`
Past actions	
a_0, a_1	`eA`
Global	`node ratio`

Table 5.1: Transition classifier features.
s_i: stack node i from the top. b_i: buffer node i.
xl, xr (xL, xR): 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.
$x \to y$ 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 (Neubig et al., 2017).[6] 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 (Hershcovich et al., 2018a), without any hyperparameter tuning on UD treebanks.

Hyperparameter	**Value**
Pre-trained word dim.	300
Lemma dim.	200
Coarse (universal) POS tag dim.	20
Fine-grained POS tag dim.	20
Named entity dim.	3
Punctuation dim.	1
Shape dim.	3
Prefix dim.	2
Suffix dim.	3
Action dim.	3
Edge label dim.	20
MLP layers	2
MLP dimensions	50
BiLSTM layers	2
BiLSTM dimensions	500

Table 5.2: Hyperparameter settings.

5.4.1 Hyperparameters

We use dropout (Srivastava et al., 2014) between MLP layers, and recurrent dropout (Gal and Ghahramani, 2016) between BiLSTM layers, both with $p = 0.4$. We also use word, lemma, coarse- and fine-grained POS tag dropout with $\alpha = 0.2$ (Kiperwasser and Goldberg, 2016): 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* (Hershcovich et al., 2018a): 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 (Sashank J. Reddi, 2018) 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 Keskar and Socher (2017). 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

[6]`http://dynet.io`

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

	TUPA (official)	**TUPA (unofficial)**	**UDPipe (baseline)**
All treebanks	53.69	58.48	65.80
Big treebanks	62.07	67.36	74.14
PUD treebanks	56.35	56.82	66.63
Small treebanks	36.74	41.19	55.01
Low-resource	8.53	12.68	17.17

Table 5.3: Aggregated test LAS-F1 scores for our system (TUPA) and the baseline system (UDPipe 1.2).

development 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[7]. 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

$$P = \frac{\#\text{correct}}{\#\text{system}}, R = \frac{\#\text{correct}}{\#\text{gold}}, F1 = 2 \cdot \frac{P \cdot R}{P + R}.$$

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 $100 \cdot \frac{\#\text{enhanced}}{\#\text{enhanced}+\#\text{words}}$.

Just as remote edges in UCCA parsing are more challenging than primary edges (Hershcovich 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.[8] 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[9], which is in fact not among

[7]`https://github.com/CoNLL-UD-2018/HUJI/blob/master/tupa/scripts/conll18_ud_eval.py`
[8]`https://spacy.io/api/annotation`
[9]`https://catalog.ldc.upenn.edu/LDC2013T19`

Treebank	Enhanced LAS P	R	F1	% Enhanced
ar_padt	28.51	16.24	20.69	5.30
cs_cac	54.94	35.69	43.27	7.57
cs_fictree	48.78	18.53	26.85	4.30
cs_pdt	49.46	26.47	34.48	4.61
nl_alpino	56.04	50.81	53.30	4.80
nl_lassysmall	49.71	51.30	50.49	4.13
en_ewt	57.36	52.05	54.58	4.36
en_pud	58.99	50.00	54.13	5.14
fi_tdt	40.20	31.37	35.24	7.34
lv_lvtb	31.76	18.70	23.54	4.12
pl_lfg	59.19	54.13	56.55	2.61
sk_snk	37.28	21.61	27.36	3.91
sv_pud	45.40	39.58	42.29	6.36
sv_talbanken	50.15	43.20	46.42	6.89

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.

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

	LAS-F1		Enhanced LAS-F1	
Model	**Test**	**Dev**	**Test**	**Dev**
Original	72.10	72.44	54.58	57.13
−NER	71.82	71.81	55.31	54.65
−POS	69.23	69.54	53.78	49.12
−Embed.	72.33	72.55	56.26	54.54
−Remote	72.08	72.32	0.00	0.00

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.

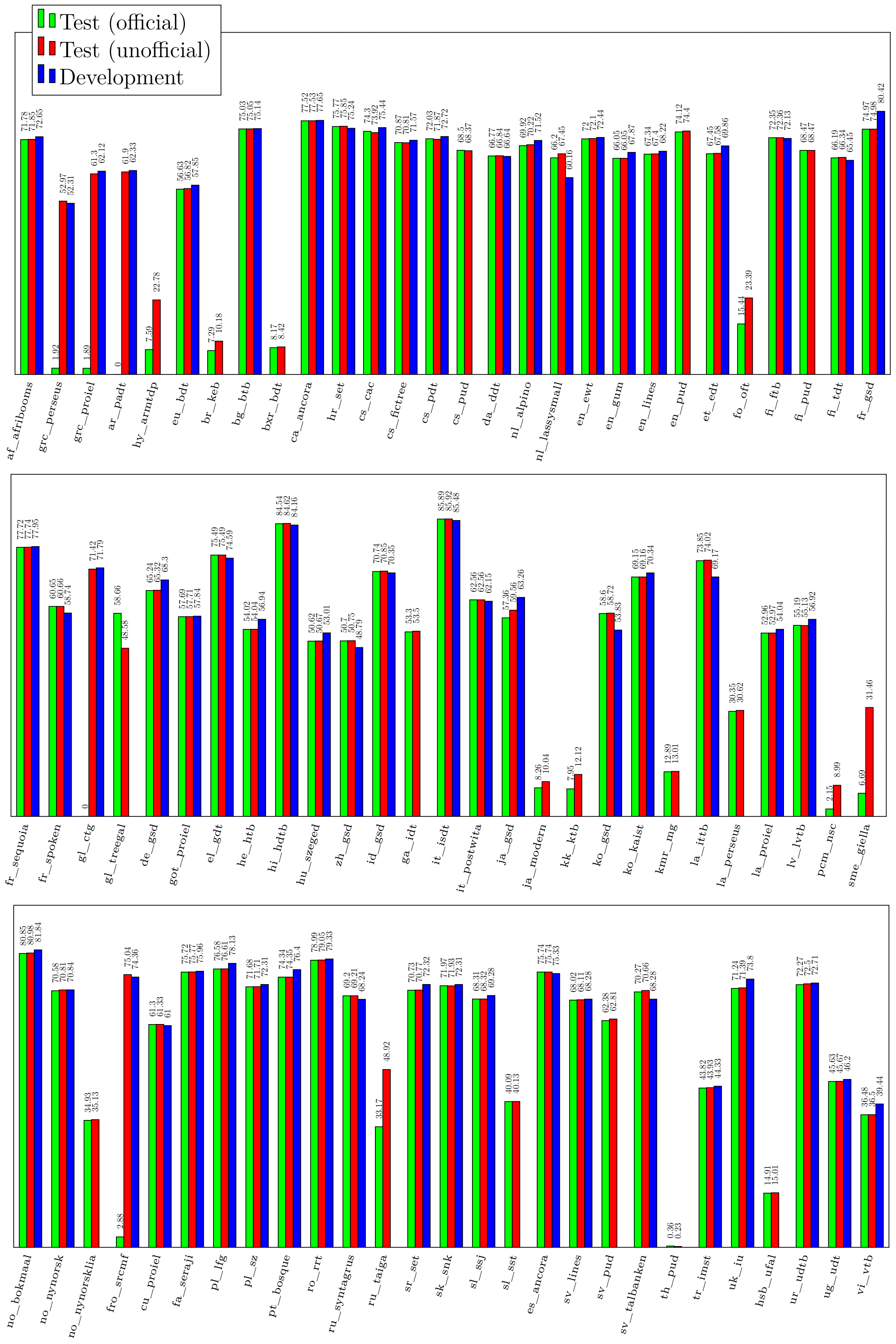


Figure 5.8: TUPA's LAS-F1 per treebank: official and unofficial test scores, and development scores (where available).

Chapter 6

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

Daniel Hershcovich[1,2], **Omri Abend**[2] **and Ari Rappoport**[2]
[1]**The Edmond and Lily Safra Center for Brain Sciences**
[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.[1] Our findings are relevant to other semantic representations, given their significant overlap in content (Abend and Rappoport, 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. (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.[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.[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

[1]This excludes cases of shared argumenthood, which are partially covered by *enhanced UD*. See §6.4.1.
[2]Our conversion and analysis code is public available at `https://github.com/danielhers/synsem`.
[3]See Supplementary Material for a definition of each category in both schemes, and their abbreviations.

Participant	A	Linker	L
Center	C	Connector	N
Adverbial	D	Process	P
Elaborator	E	Quantifier	Q
Function	F	Relator	R
Ground	G	State	S
Parallel Scene	H	Time	T

Table 6.1: Legend of UCCA categories (edge labels).

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.[4]

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.

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.

[4]Despite the similar terminology, UCCA Adverbials are not necessarily adverbs syntactically.

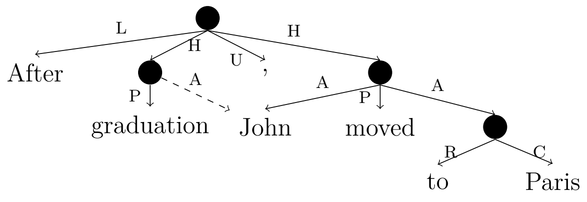


Figure 6.1: UCCA graph. Dashed: remote edge.

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, 2019).

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

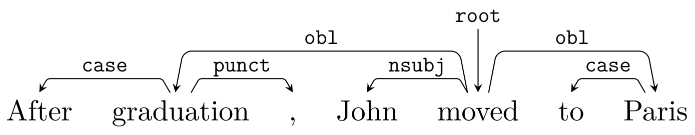


Figure 6.2: UD tree.

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 guidelines[5] and cross-reviewed. As these sentences are included in the UD English_EWT treebank, this is a *shared* gold-standard UCCA and UD annotated corpus.[6] We use the standard train/development/test split, shown in Table 6.2.

[5]`http://bit.ly/ucca_guidelines_v2`

[6]Our data is available at `https://github.com/UniversalConceptualCognitiveAnnotation/UCCA_English-EWT`.

	Train	**Dev**	**Test**
# Passages	347	192	184
# Sentences	2,723	554	535
# Tokens	44,804	5,394	5,381

Table 6.2: Data split for the shared gold-standard corpus.

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),[7] 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.

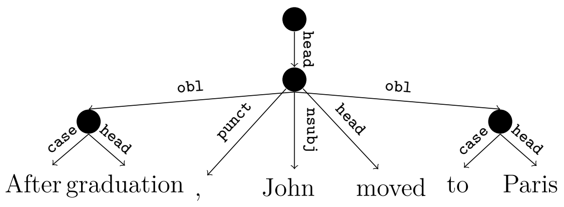


Figure 6.3: Converted UD tree. Non-terminals and *head* edges are introduced by the unified DAG converter.

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),[8] which form (bi-lexical) graphs, representing phenomena such

[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., 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.

	A	A|P	A|S	C	D	E	F	G	H	L	N	P	Q	R	S	T	No Match
`acl`	58			1	4	249	1		48			6			1	1	409
`advcl`	14			12	2	2		6	512	4		11					423
`advmod`	225		1	69	1778	332	27	135	14	258	2	2	15	44	9	368	273
`amod`	25			134	647	837		1	28			7	130	3	269	25	176
`appos`	21			39	2	34			18						8		33
`aux`					384	2	1335			2		1		1			17
`case`	11			31	27	25	123			213	26	11	1	2629	154	1	262
`cc`				8	4	1	4	1	1	1567	381		6	12			52
`ccomp`	345			1		1			36			2			1	1	166
`compound`	225			116	67	586	21		2			32	19	1	12	24	683
`conj`	10			449	4	5		1	1262	1		6	2		10		497
`cop`				1			1312			1		9		10	178		7
`csubj`	13								3								46
`det`	10			17	119	440	2963				1		129	16	1		124
`discourse`	1			2	1		25	29	27	16					5		19
`expl`	21			1			98								17		3
`iobj`	131			1			1										10
`list`	3			7	2	1			27						1		6
`mark`				9	7	1	531	1		654				407	1	5	143
`nmod`	844	1	1	20	9	786	8	4	12	1	1	20	2	2	11	27	488
`nsubj`	4296	7	21	25	3	2	55	1	5	61		58	1	80	14	4	247
`nummod`	2			33	12	17		4		4			334				64
`obj`	1845		1	54	21	6	11	1	4	23		52	1	23	3	11	583
`obl`	1195			19	115	41	1	17	39	34		6	6	26	7	302	611
`parataxis`	6		1	5		4		6	285						3		180
`vocative`	17							8									
`xcomp`	121			4	25				8			38			38		526
head	445	48	159	6388	717	142	564	83	2462	42	1	4163	120	52	1547	32	2235
No Match	1421	37	58	640	417	291	14	33	2291	146	6	802	94	52	369	96	

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.[9]

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

[9]See `http://bit.ly/uccaud` for a detailed explanation of each example in this section.

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`, `iobj`, `nsubj`, `obj`, `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.

[10]While 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).

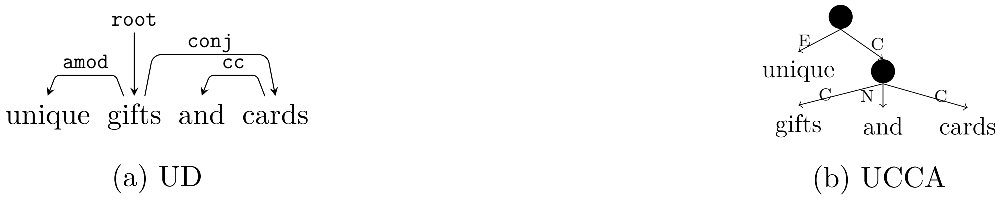
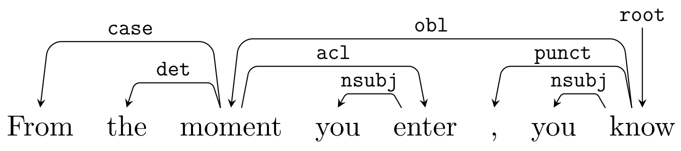


Figure 6.4: Coordination in UD and UCCA.

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.

(a) UD



(b) UCCA

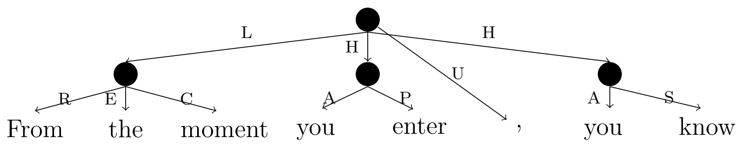


Figure 6.5: Clause linkage in UD and UCCA.

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.[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),[12] annotated over the exact same sentences. We additionally use UDPipe v1.2 (Straka et al., 2016; Straka and Straková, 2017), trained on English_EWT,[13] 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, 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.

[12] `https://hdl.handle.net/11234/1-2895`
[13] `https://hdl.handle.net/11234/1-2898`

	Primary			**Remote**		
Features	**LP**	**LR**	**LF**	**LP**	**LR**	**LF**
Development						
Gold UD	72.1	71.2	71.7	61.2	38.1	47.0
UDPipe	73.0	72.1	72.5	53.7	40.8	46.4
Test						
Gold UD	72.2	71.2	71.7	60.9	36.8	45.9
UDPipe	72.4	71.7	72.1	60.3	38.5	47.0

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.

		aux	det	cop	cc	expl	iobj	nsubj	case	list	advmod	amod	nummod	mark	vocative	compound	obj	nmod	conj	advcl	obl	xcomp	discourse	ccomp	parataxis	appos	acl	csubj
(a)	**Labeled F1 %**	94	93	89	86	83	83	80	76	76	72	71	71	70	62	59	57	55	50	49	48	41	38	29	23	21	20	0
	Unlabeled F1 %	99	99	100	99	100	83	84	95	76	95	95	86	97	92	84	65	77	61	51	61	63	95	29	36	48	37	33
(b)	**Total in UD #**	156	392	187	212	12	8	463	335	15	378	374	38	116	1	219	222	231	244	52	208	1	16	29	52	22	81	5
	Match Gold #	156	385	187	206	12	6	468	305	12	359	361	33	111	7	146	187	198	210	40	162	28	10	20	48	17	56	4
	Match Predicted #	154	388	187	203	12	6	446	313	9	345	339	32	113	6	136	163	183	177	30	147	26	11	15	30	12	36	2
	Labeled Correct #	145	361	166	175	10	5	365	236	8	253	248	23	78	4	83	99	104	96	17	74	11	4	5	9	3	9	0
	Unlabeled Correct #	154	381	187	203	12	5	386	293	8	336	334	28	109	6	118	113	147	119	18	94	17	10	5	14	7	17	1
(c)	**Labeled/Unlabeled %**	94	95	89	86	83	100	95	81	100	75	74	82	72	67	70	88	71	81	94	79	65	40	100	64	43	53	0
	Mode/Match Gold %	79	82	86	75	58	100	91	79	83	51	35	85	45	71	54	91	51	70	92	68	44	30	94	98	41	72	100
(d)	**Average Words #**	1.0	1.0	1.0	1.0	1.0	1.1	1.6	1.0	2.2	1.2	1.2	1.1	1.0	1.6	1.2	3.0	2.4	5.8	6.6	3.8	6.0	1.1	9.0	6.7	4.0	5.6	7.5

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; Hershcovich et al., 2017), joint modeling (Surdeanu et al., 2008; Hajič et al., 2009), and multi-task learning (Swayamdipta et al., 2016b, 2018; Hershcovich et al., 2018a). Empirical comparison between 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 et al., 2015). 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.

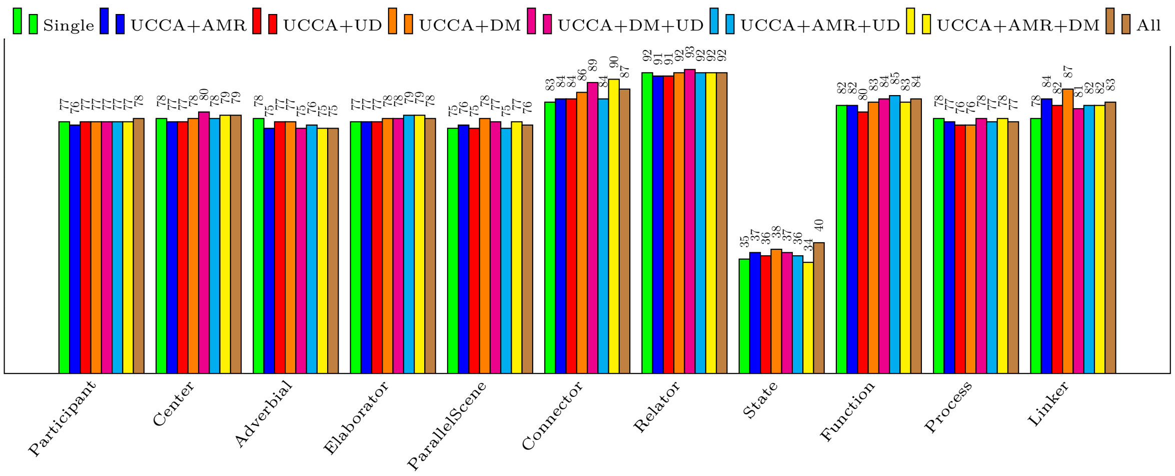
7.4 Ongoing Work

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

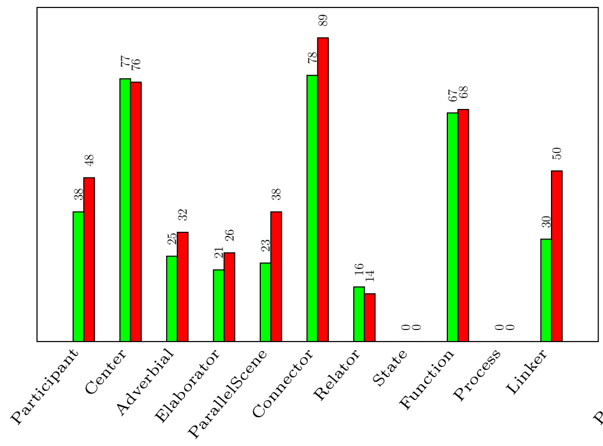
7.4.1 Combining Syntax with Lexical Semantics

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.

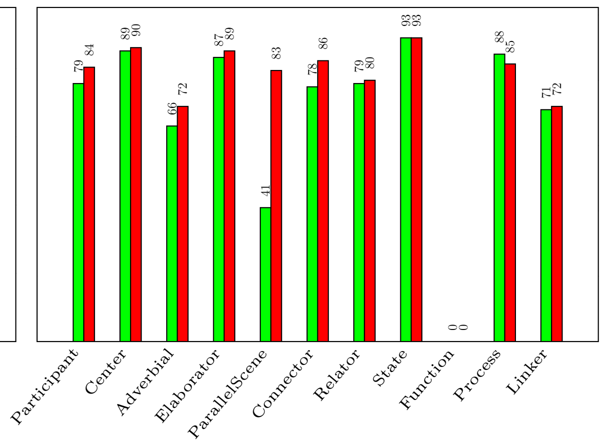
These gaps can perhaps be closed by complementing syntax with *lexical* semantics to



(a) English Wiki

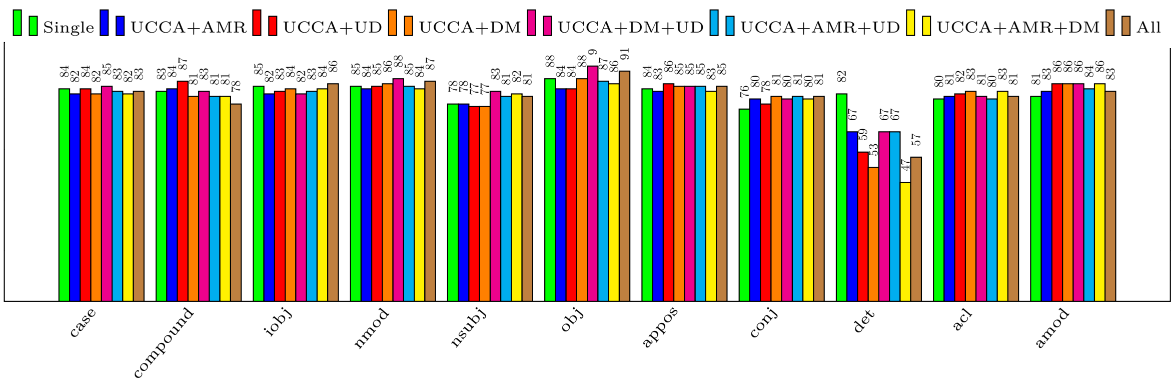


(b) French 20K

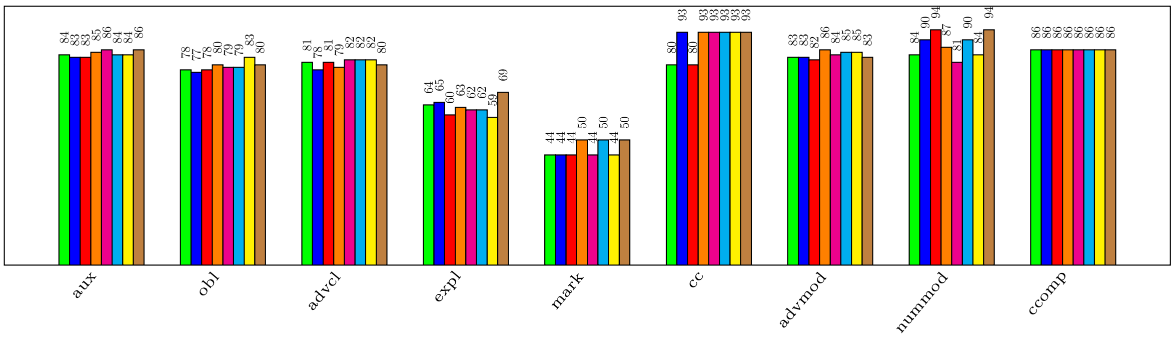


(c) German 20K

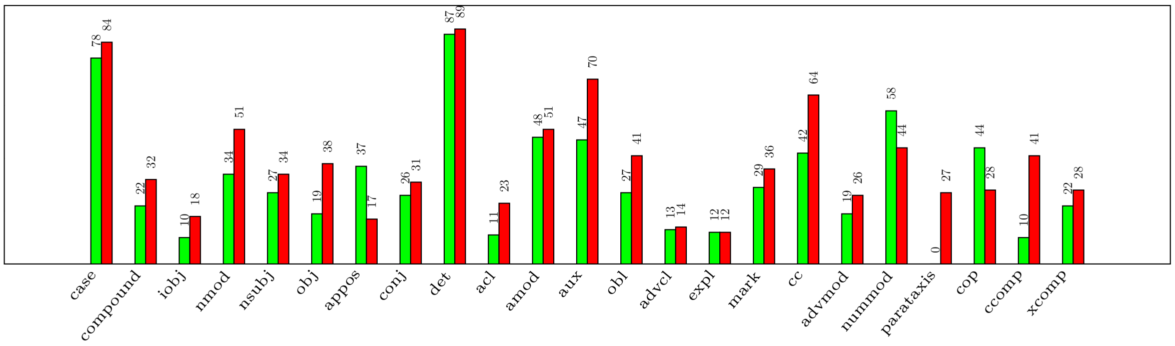
Figure 7.1: TUPA's F1 per UCCA category in each single-/multitask setting.



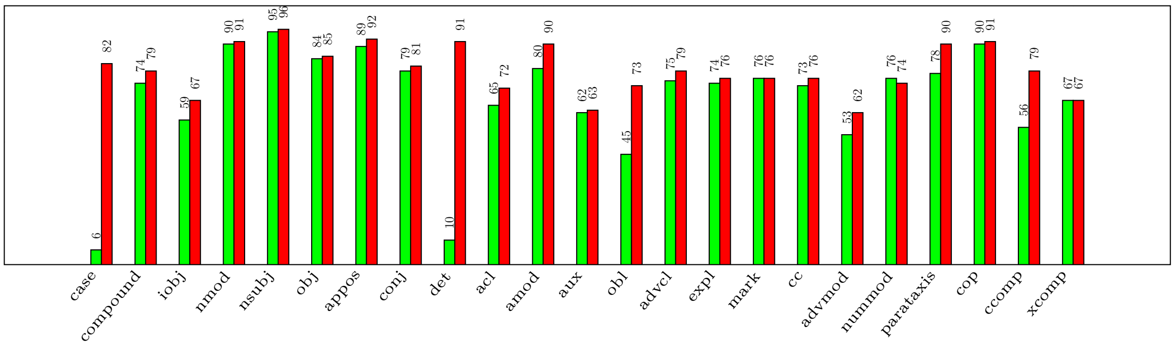
(a) English Wiki



(b) English Wiki (cont.)



(c) French 20K



(d) German 20K

Figure 7.2: TUPA's F1 per UD relation in each single-/multitask setting.

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.[1] The task results were presented during SemEval 2019.[2]

[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]`http://www.conll.org/`
[4]`http://mrp.nlpl.eu`

Bibliography

Omri Abend and Ari Rappoport. 2013. Universal Conceptual Cognitive Annotation (UCCA). In *Proc. of ACL*, pages 228–238.

Omri Abend and Ari Rappoport. 2017. The state of the art in semantic representation. In *Proc. of ACL*, pages 77–89.

Omri Abend, Shai Yerushalmi, and Ari Rappoport. 2017. Uccaapp: Web-application for syntactic and semantic phrase-based annotation. *Proc. of ACL System Demonstrations*, pages 109–114.

Željko Agić and Alexander Koller. 2014. Potsdam: Semantic dependency parsing by bidirectional graph-tree transformations and syntactic parsing. In *Proc. of SemEval*, pages 465–470.

Željko Agić, Alexander Koller, and Stephan Oepen. 2015. Semantic dependency graph parsing using tree approximations. In *Proc. of IWCS*, pages 217–227.

Mariana S. C. Almeida and André F. T. Martins. 2015. Lisbon: Evaluating TurboSemanticParser on multiple languages and out-of-domain data. In *Proc. of SemEval*, pages 970–973.

Bharat Ram Ambati, Tejaswini Deoskar, Mark Johnson, and Mark Steedman. 2015. An incremental algorithm for transition-based CCG parsing. In *Proc. of NAACL*, pages 53–63.

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2016. Shift-reduce CCG parsing using neural network models. In *Proc. of NAACL-HLT*, pages 447–453.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah Smith. 2016. Many languages, one parser. *TACL*, 4:431–444.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proc. of ACL*, pages 2442–2452.

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proc. of EMNLP*, pages 1699–1710.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In *ACL-COLING '98*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. AMR parsing using stack-LSTMs. In *Proc. of EMNLP*, pages 1269–1275.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proc. of EMNLP*, pages 349–359.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proc. of the Linguistic Annotation Workshop*.

Guntis Barzdins and Didzis Gosko. 2016. RIGA at SemEval-2016 task 8: Impact of Smatch extensions and character-level neural translation on AMR parsing accuracy. In *Proc. of SemEval*, pages 1143–1147.

Valerio Basile, Johan Bos, Kilian Evang, and Noortje Venhuizen. 2012. Developing a large semantically annotated corpus. In *Proc. of LREC*, pages 3196–3200.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Eric Baucom, Levi King, and Sandra Kübler. 2013. Domain adaptation for parsing. In *Proc. of RANLP*, pages 56–64.

Emily M Bender, Dan Flickinger, Stephan Oepen, and Yi Zhang. 2011. Parser evaluation over local and non-local deep dependencies in a large corpus. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 397–408. Association for Computational Linguistics.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In *Proc. of EMNLP-CoNLL*, pages 995–1005.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. 2012. English web treebank. *Linguistic Data Consortium, Philadelphia, PA*.

Joachim Bingel and Anders Søgaard. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proc. of EACL*, pages 164–169.

Alexandra Birch, Omri Abend, Ondřej Bojar, and Barry Haddow. 2016. HUME: Human UCCA-based evaluation of machine translation. In *Proc. of EMNLP*, pages 1264–1274.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP*, pages 120–128.

Austin Blodgett and Nathan Schneider. 2018. Semantic supersenses for English possessives. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, pages 1529–1534, Miyazaki, Japan. ELRA.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proc. of EMNLP-CoNLL*, pages 1455–1465.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL*, 5:135–146.

Marcel Bollmann and Anders Søgaard. 2016. Improving historical spelling normalization with bi-directional lstms and multi-task learning. In *Proc. of COLING*, pages 131–139.

Johan Bos. 2005. Towards wide-coverage semantic interpretation. In *Proc. of IWCS*, volume 6, pages 42–53.

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

Johan Bos. 2015. Open-domain semantic parsing with boxer. In *Proceedings of the 20th nordic conference of computational linguistics (NODALIDA 2015)*, pages 301–304.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015a. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*.

Samuel R Bowman, Christopher D Manning, and Christopher Potts. 2015b. Tree-structured composition in neural networks without tree-structured architectures. *arXiv preprint arXiv:1506.04834*.

Samuel R Bowman, Christopher Potts, and Christopher D Manning. 2014. Recursive neural networks can learn logical semantics. *arXiv preprint arXiv:1406.1827*.

Chloé Braud, Barbara Plank, and Anders Søgaard. 2016. Multi-view and multi-task training of RST discourse parsers. In *Proc. of COLING*, pages 1903–1913.

Jan Buys and Phil Blunsom. 2017a. Oxford at SemEval-2017 task 9: Neural AMR parsing with pointer-augmented attention. In *Proc. of SemEval*, pages 914–919.

Jan Buys and Phil Blunsom. 2017b. Robust incremental neural semantic graph parsing. In *Proc. of ACL*, pages 1215–1226.

Jiaxun Cai, Shexia He, Zuchao Li, and Hai Zhao. 2018. A full end-to-end semantic role labeler, syntactic-agnostic over syntactic-aware? In *Proc. of COLING*, pages 2753–2765.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proc. of ACL*, pages 748–752.

Rich Caruana. 1997. Multitask Learning. *Machine Learning*, 28(1):41–75.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. of EMNLP*, pages 740–750.

Pengxiang Cheng and Katrin Erk. 2018. Implicit argument prediction with event knowledge. *arXiv preprint arXiv:1802.07226*.

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

Michael Collins. 1997. Three generative lexicalized models for statistical parsing.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. of ACL*, pages 111–118.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

Matthieu Constant and Joakim Nivre. 2016. A transition-based system for joint lexical and syntactic analysis. In *Proc. of ACL*, pages 161–171.

Ann Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proc. of LREC*, pages 591–600.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2):281–332.

William Croft and D Alan Cruse. 2004. *Cognitive linguistics.* Cambridge University Press.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Marco Damonte and Shay B. Cohen. 2018. Cross-lingual abstract meaning representation parsing. In *Proc. of NAACL-HLT*, pages 1146–1155.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for Abstract Meaning Representation. In *Proc. of EACL*.

Hal Daume III. 2007. Frustratingly easy domain adaptation. In *Proc. of ACL*, pages 256–263.

Robert M. W. Dixon. 2010a. *Basic Linguistic Theory: Grammatical Topics*, volume 2. Oxford University Press.

Robert M. W. Dixon. 2010b. *Basic Linguistic Theory: Methodology*, volume 1. Oxford University Press.

Robert M. W. Dixon. 2012. *Basic Linguistic Theory: Further Grammatical Topics*, volume 3. Oxford University Press.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proc. of CoNLL*, pages 20–30.

Yantao Du, Fan Zhang, Xun Zhang, Weiwei Sun, and Xiaojun Wan. 2015. Peking: Building semantic dependency graphs with a hybrid parser. In *Proc. of SemEval*, pages 927–931.

Long Duong, Hadi Afshar, Dominique Estival, Glen Pink, Philip Cohen, and Mark Johnson. 2017. Multilingual semantic parsing and code-switching. In *Proc. of CoNLL*, pages 379–389.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependeny parsing with stack long short-term memory. In *Proc. of ACL*, pages 334–343.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Xing Fan, Emilio Monti, Lambert Mathias, and Markus Dreyer. 2017. Transfer learning for neural semantic parsing. In *Proc. of Workshop on Representation Learning for NLP*, pages 48–56.

Daniel Fernández-González and André FT Martins. 2015. Parsing as reduction. In *Proc. of ACL*, pages 1523–1533.

Jenny Rose Finkel and Christopher D. Manning. 2009. Joint parsing and named entity recognition. In *Proc. of NAACL-HLT*, pages 326–334.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the Abstract Meaning Representation. In *Proc. of ACL*, pages 1426–1436.

Daniel Flickinger. 2000. On building a more efficient grammar by exploiting types. In *Collaborative Language Engineering*, volume 6, pages 15–28. CLSI, Stanford, CA.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. DeepBank: A dynamically annotated treebank of the Wall Street Journal. In *Proc. of Workshop on Treebanks and Linguistic Theories*, pages 85–96.

William Foland and James H. Martin. 2017. Abstract Meaning Representation parsing using LSTM recurrent neural networks. In *Proc. of ACL*, pages 463–472.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3).

Yoav Goldberg. 2016. A primer on neural network models for natural language processing.

Yoav Goldberg and Michael Elhadad. 2011. Learning sparser perceptron models. Technical report.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proc. of COLING*, pages 959–976.

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.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. Noise reduction and targeted exploration in imitation learning for Abstract Meaning Representation parsing. In *Proc. of ACL*, pages 1–11.

Alex Graves. 2008. Supervised sequence labelling with recurrent neural networks. *Ph. D. thesis*.

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2016. Exploiting multi-typed treebanks for parsing with deep multi-task learning. *CoRR*, abs/1606.01161.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štepánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proc. of CoNLL*, pages 1–18.

Kazuma Hashimoto, caiming xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple NLP tasks. In *Proc. of EMNLP*, pages 1923–1933.

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.

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual joint parsing of syntactic and semantic dependencies with a latent variable model. *Computational Linguistics*, 39(4):949–998.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2017. A transition-based directed acyclic graph parser for UCCA. In *Proc. of ACL*, pages 1127–1138.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018a. Multitask parsing across semantic representations. In *Proc. of ACL*, pages 373–385.

Daniel Hershcovich, Omri Abend, and Ari Rappoport. 2018b. Universal dependency parsing with a general transition-based DAG parser. In *Proc. of CoNLL*. To appear.

Jonathan Herzig and Jonathan Berant. 2017. Neural semantic parsing over multiple knowledge-bases. In *Proc. of ACL*, pages 623–628.

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

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proc. of EMNLP*, pages 1373–1378.

Matthew Honnibal and Ines Montani. 2018. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *To appear.*

Ozan Irsoy and Claire Cardie. 2014. Opinion mining with deep recurrent neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 720–728.

Angelina Ivanova, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? A contrastive study of syntacto-semantic dependencies. In *Proc. of LAW*, pages 2–11.

Bevan Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In *Proc. of COLING*, pages 1359–1376.

Aravind Joshi and Yves Schabes. 1997. Tree-Adjoining Grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin.

Hans Kamp and Uwe Reyle. 2013. *From discourse to logic: Introduction to model theoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer Science & Business Media.

Nitish Shirish Keskar and Richard Socher. 2017. Improving generalization performance by switching from Adam to SGD. *CoRR*, abs/1712.07628.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

Sigrid Klerke, Yoav Goldberg, and Anders Søgaard. 2016. Improving sentence compression by learning to predict gaze. In *Proc. of NAACL-HLT*, pages 1528–1533.

Lingpeng Kong, Alexander M. Rush, and Noah A. Smith. 2015. Transforming dependencies into phrase structures. In *Proc. of NAACL HLT*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proc. of ACL*, pages 146–157.

Marco Kuhlmann and Joakim Nivre. 2010. Transition-based techniques for non-projective dependency parsing. *NEJLT*, 2(1):1–19.

Marco Kuhlmann and Stephan Oepen. 2016. Towards a catalogue of linguistic graph banks. *Computational Linguistics*.

Mike Lewis, Luheng He, and Luke Zettlemoyer. 2015. Joint A* CCG parsing and semantic role labelling. In *Proc. of EMNLP*, pages 1444–1454.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *Proc. of TLT*, pages 99–110.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward abstractive summarization using semantic representations. In *Proc. of NAACL*, pages 1077–1086.

Xavier Lluís and Lluís Màrquez. 2008. A joint model for parsing syntactic and semantic dependencies. In *Proc. of CoNLL*, pages 188–192.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proc. of ACL*, pages 1202–1212.

Wolfgang Maier and Timm Lichte. 2009. Characterizing discontinuity in constituent treebanks. In *Proc. of Formal Grammar*, number 5591 in Lecture Notes in Artificial Intelligence, pages 167–182, Bordeaux, France. Springer.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proc. of Workshop on Discontinuous Structures in NLP*, pages 47–57.

Marie-Catherine de Marneffe and Joakim Nivre. 2019. Dependency grammar. *Annual Review of Linguistics*, 5(1):197–218.

Héctor Martínez Alonso and Barbara Plank. 2017. When is multitask learning effective? Semantic sequence prediction under varying data conditions. In *Proc. of EACL*, pages 44–53.

Jonathan May. 2016. SemEval-2016 task 8: Meaning representation parsing. In *Proc. of SemEval*, pages 1063–1073.

Jonathan May and Jay Priyadarshi. 2017. SemEval-2017 task 9: Abstract Meaning Representation parsing and generation. In *Proc. of SemEval*, pages 536–545.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *Proc. of NAACL-HLT*, pages 28–36.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

Dipendra K Misra and Yoav Artzi. 2016. Neural shift-reduce CCG semantic parsing. In *Proc. of EMNLP*, pages 1775–1786.

Tom M Mitchell. 1980. *The need for biases in learning generalizations.* Citeseer.

Amir More. 2016. Joint morpho-syntactic processing of morphologically rich languages in a transition-based framework. Master's thesis, The Interdisciplinary Center, Herzliya.

Shashi Narayan and Claire Gardent. 2014. Hybrid simplification using deep semantics and machine translation. In *Proc. of ACL*, pages 435–445.

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

Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*, pages 149–160.

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

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proc. of ACL*, pages 351–359.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor

Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2018. Universal dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, 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 Cebiroğlu 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 Ilarraza, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Radu Ion, Elena Irimia, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shinsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Robert Östling, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jonathan North Washington, Mats Wirén, Tak-sum Wong, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. 2017. Universal dependencies 2.1. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre and Chiao-Ting Fang. 2017. Universal dependency evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 86–95.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proc. of LREC*.

Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R Curran. 2013. Learning multilingual named entity recognition from wikipedia. *Artificial Intelligence*, 194:151–175.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Zdenka Uresova. 2016. Towards comparability of linguistic graph banks for semantic parsing. In *Proc. of LREC*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, pages 915–926.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, pages 63–72.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

Hao Peng, Sam Thomson, and Noah A. Smith. 2017a. Deep multitask learning for semantic dependency parsing. In *Proc. of ACL*, pages 2037–2048.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018. Learning joint semantic parsers from disjoint data. In *Proc. of NAACL-HLT*.

Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. 2017b. Addressing the data sparsity issue in neural AMR parsing. In *Proc. of EACL*, pages 366–375.

Miriam RL Petruck. 2019. Meaning representation of null instantiated semantic roles in FrameNet. In *Proceedings of the First International Workshop on Designing Meaning Representations*, pages 121–127.

Barbara Plank. 2016. Keystroke dynamics as signal for shallow syntactic parsing. In *Proc. of COLING*, pages 609–619.

Barbara Plank and Gertjan van Noord. 2011. Effective measures of domain similarity for parsing. In *Proc. of ACL-HLT*, pages 1566–1576.

Carl Pollard and Ivan Sag. 1994a. *Head Driven Phrase Structure Grammar*. CSLI Publications, Stanford, CA.

Carl Pollard and Ivan A Sag. 1994b. *Head-driven phrase structure grammar*. University of Chicago Press.

Martin Potthast, Tim Gollub, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. 2014. Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*, pages 268–299, Berlin Heidelberg New York. Springer.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English strings with Abstract Meaning Representation graphs. In *Proc. of EMNLP*, pages 425–429.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Parsing English into Abstract Meaning Representation using syntax-based machine translation. In *Proc. of EMNLP*, pages 1143–1154.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *TACL*, 4:127–141.

Siva Reddy, Oscar Täckström, Slav Petrov, Mark Steedman, and Mirella Lapata. 2017. Universal semantic parsing. In *Proc. of EMNLP*, pages 89–101.

Corentin Ribeyre, Eric Villemonte de la Clergerie, and Djamé Seddah. 2014. Alpage: Transition-based semantic graph parsing with syntactic features. In *Proc. of SemEval*, pages 97–103.

Michael Roth and Anette Frank. 2015. Inducing Implicit Arguments from Comparable Texts: A Framework and its Applications. *Computational Linguistics*, 41:625–664.

Rachel Rudinger and Benjamin Van Durme. 2014. Is the Stanford dependency representation semantic? In *Proc. of EVENTS*, pages 54–58.

Josef Ruppenhofer, Michael Ellsworth, Miriam R. L Petruck, Christopher R. Johnson, Collin F. Baker, and Jan Scheffczyk. 2016. *FrameNet II: Extended Theory and Practice.*

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proc. of IWPT*, pages 125–132.

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proc. of COLING*, pages 753–760.

Sanjiv Kumar Sashank J. Reddi, Satyen Kale. 2018. On the convergence of Adam and beyond. *ICLR*.

Natalie Schluter, Anders Søgaard, Jakob Elming, Dirk Hovy, Barbara Plank, Héctor Martínez Alonso, Anders Johanssen, and Sigrid Klerke. 2014. Copenhagen-Malmö: Tree approximations of semantic parsing problems. In *Proc. of SemEval*, pages 213–217.

Nathan Schneider. 2014. *Lexical Semantic Analysis in Natural Language Text*. Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

Nathan Schneider, Emily Danchik, Chris Dyer, and Noah A Smith. 2014. Discriminative lexical semantic segmentation with gaps: running the MWE gamut. *TACL*, 2:193–206.

Nathan Schneider, Jena D. Hwang, Vivek Srikumar, Jakob Prange, Austin Blodgett, Sarah R. Moeller, Aviram Stern, Adi Bitan, and Omri Abend. 2018. Comprehensive supersense disambiguation of English prepositions and possessives. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Melbourne, Australia. Association for Computational Linguistics.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced English Universal Dependencies: An improved representation for natural language understanding tasks. In *Proc. of LREC*. ELRA.

Roy Schwartz, Omri Abend, and Ari Rappoport. 2012. Learnability-based syntactic annotation design. In *Proc. of COLING*, pages 2405–2422.

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Chris Manning. 2014. A gold standard dependency corpus for English. In *Proc. of LREC*.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 deep learning and unsupervised feature learning workshop*, volume 2010, pages 1–9.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proc. of ACL*, pages 231–235.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Gabriel Stanovsky, Jessica Ficler, Ido Dagan, and Yoav Goldberg. 2016. Getting more out of syntax with PropS. *arXiv preprint arXiv:1603.01648*.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia. European Language Resources Association.

Milan Straka and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proc. of EMNLP*, pages 5027–5038.

Elior Sulem, Omri Abend, and Ari Rappoport. 2015. Conceptual annotations preserve structure across translations: A French-English case study. In *Proc. of S2MT*, pages 11–22.

Elior Sulem, Omri Abend, and Ari Rappoport. 2018a. Semantic structural annotation for text simplification. In *Proc. of NAACL*.

Elior Sulem, Omri Abend, and Ari Rappoport. 2018b. Simple and effective text simplification using semantic and neural methods. In *Proc. of ACL*.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proc. of CoNLL*, pages 159–177.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016a. Greedy, joint syntactic-semantic parsing with stack LSTMs. In *Proc. of CoNLL*, pages 187–197.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016b. Greedy, joint syntactic-semantic parsing with stack LSTMs. In *Proc. of CoNLL*, pages 187–197.

Swabha Swayamdipta, Sam Thomson, Chris Dyer, and Noah A. Smith. 2017. Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. *CoRR*, abs/1706.09528.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A. Smith. 2018. Syntactic scaffolds for semantic structures. In *Proc. of EMNLP*, pages 3772–3782.

Ida Szubert, Adam Lopez, and Nathan Schneider. 2018. A structured syntax-semantics interface for English-AMR alignment. In *Proc. of NAACL-HLT*. To appear.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Sam Thomson, Brendan O'Connor, Jeffrey Flanigan, David Bamman, Jesse Dodge, Swabha Swayamdipta, Nathan Schneider, Chris Dyer, and Noah A. Smith. 2014. CMU: Arc-factored, discriminative semantic dependency parsing. In *Proc. of SemEval*, pages 176–180.

Alper Tokgöz and Gülsen Eryiğit. 2015. Transition-based dependency DAG parsing using dynamic oracles. In *Proc. of ACL Student Research Workshop*, pages 22–27.

Kristina Toutanova, Aria Haghighi, and Christopher Manning. 2005. Joint learning improves semantic role labeling. In *Proc. of ACL*, pages 589–596.

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

Lucy Vanderwende, Arul Menezes, and Chris Quirk. 2015. An AMR parser for English, French, German, Spanish and Japanese and a new AMR-annotated corpus. In *Proc. of NAACL*, pages 26–30.

Abigail Walsh, Claire Bonial, Kristina Geeraert, John P. McCrae, Nathan Schneider, and Clarissa Somers. 2018. Constructing an annotated corpus of verbal MWEs for English. In *Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions*, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Chuan Wang, Sameer Pradhan, Xiaoman Pan, Heng Ji, and Nianwen Xue. 2016. CAMR at SemEval-2016 task 8: An extended transition-based AMR parser. In *Proc. of SemEval*, pages 1173–1178.

Chuan Wang and Nianwen Xue. 2017. Getting the most out of AMR parsing. In *Proc. of EMNLP*, pages 1257–1268.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proc. of ACL*, pages 857–862.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for AMR parsing. In *Proc. of NAACL*, pages 366–375.

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. 2013. OntoNotes release 5.0 LDC2013T19. *Linguistic Data Consortium, Philadelphia, PA*.

Keenon Werling, Gabor Angeli, and Christopher D. Manning. 2015. Robust subgraph generation improves Abstract Meaning Representation parsing. In *Proc. of ACL*, pages 982–991.

Aaron Steven White, Drew Reisinger, Keisuke Sakaguchi, Tim Vieira, Sheng Zhang, Rachel Rudinger, Kyle Rawlins, and Benjamin Van Durme. 2016. Universal decompositional semantics on universal dependencies. In *Proc. of EMNLP*, pages 1713–1723.

Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–20, Brussels, Belgium. Association for Computational Linguistics.

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

Yuan Zhang and David Weiss. 2016. Stack-propagation: Improved representation learning for syntax. In *Proc. of ACL*, pages 1557–1566.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proc. of IWPT*, pages 162–171.

Yue Zhang and Stephen Clark. 2011. Shift-reduce CCG parsing. In *Proc. of ACL*, pages 683–692.

Junsheng Zhou, Feiyu Xu, Hans Uszkoreit, Weiguang Qu, Ran Li, and Yanhui Gu. 2016. AMR parsing with an incremental joint model. In *Proc. of EMNLP*, pages 680–689.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. of ACL*, pages 434–443.

Yftah Ziser and Roi Reichart. 2017. Neural structural correspondence learning for domain adaptation. In *Proc. of CoNLL*, pages 400–410.

Will Y Zou, Richard Socher, Daniel Cer, and Christopher D Manning. 2013. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1393–1398.

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 $\text{TUPA}_{\text{Sparse}}$. 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.

	Wiki			20K
	Train	Dev	Test	Leagues
nodes				
# implicit	899	122	77	241
# linkage	2956	263	359	376
edges				
# linkage	9276	803	1094	957

Table A.1: Statistics of linkage and implicit nodes in the *Wiki* and *20K Leagues* UCCA corpora. Cf. Table 3.1.

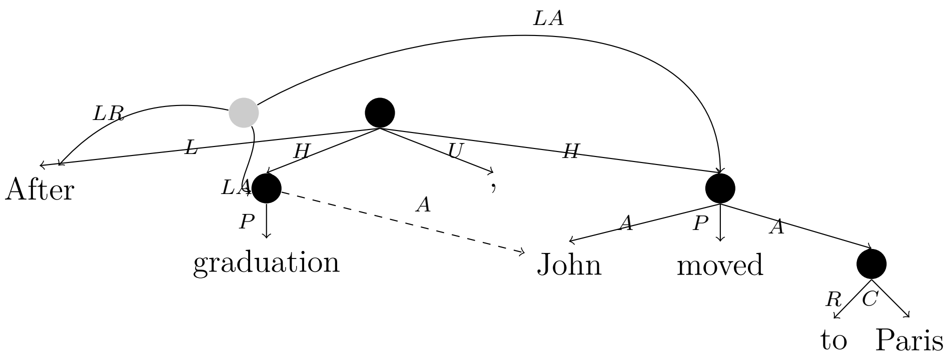


Figure A.2: UCCA example with linkage.

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.

	Sparse	MLP	BiLSTM		Sparse	MLP	BiLSTM
Embedding dimensions				Other parameters			
external word		100	100	training epochs	19	28	59
word		200	200	MinUpdate	5		
POS tag		20	20	initial learning rate	1	1	1
syntactic dep.		10	10	learning rate decay	0.1	1	1
edge label		20	20	MLP #layers		2	2
punctuation		1	1	MLP layer dim.		100	50
gap		3	3	LSTM #layers			2
action		3	3	LSTM layer dim.			500
				word dropout		0.2	0.2
				dropout		0.4	0.4
				weight decay		10^{-5}	10^{-5}
				mini-batch size		100	100

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 $\ell : E \to L$ 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 $\ell : E \to L$. 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

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

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.

1. *C* (Center)
2. *N* (Connector)
3. *H* (ParallelScene)
4. *P* (Process)
5. *S* (State)
6. *A* (Participant)
7. *D* (Adverbial)
8. *T* (Time)
9. *E* (Elaborator)
10. *R* (Relator)
11. *F* (Function)
12. *L* (Linker)
13. *LR* (LinkRelation)
14. *LA* (LinkArgument)
15. *G* (Ground)
16. *Terminal*
17. *U* (Punctuation)

(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 (Collins, 1997), 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 *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 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 $w_1, \dots, w_n$. Parsing starts with $w_1, \dots, w_n$ 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 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 SHIFT transition followed by a $\text{NODE}_{\ell(v,u)}$ 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 SHIFT until both are in the stack. If both are in the stack but neither is at the stack top, apply SWAP transitions until either moves to the buffer, and then apply SHIFT. Now, assume either v or u is at the stack top. If the other is not the second element on the stack, apply SWAP transitions until it is. Finally, v and u are the top two elements on the stack. If they are in that order, apply $\text{RIGHT-EDGE}_{\ell(v,u)}$ (or $\text{RIGHT-REMOTE}_{\ell(v,u)}$ if the edge between them is remote). Otherwise, apply $\text{LEFT-EDGE}_{\ell(v,u)}$ (or $\text{LEFT-REMOTE}_{\ell(v,u)}$ 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 REDUCE until only the root node remains on the stack, and then 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 REMOTE transitions can be removed, as well as the single-primary-parent restriction on EDGE transition.

Features from (Zhang and Clark, 2009):

unigrams
$s_0tde, s_0we, s_1tde, s_1we, s_2tde, s_2we, s_3tde, s_3we,$
$b_0wtd, b_1wtd, b_2wtd, b_3wtd,$
$s_0lwe, s_0rwe, s_0uwe, s_1lwe, s_1rwe, s_1uwe$

bigrams
$s_0ws_1w, s_0ws_1e, s_0es_1w, s_0es_1e, s_0wb_0w, s_0wb_0td,$
$s_0eb_0w, s_0eb_0td, s_1wb_0w, s_1wb_0td, s_1eb_0w, s_1eb_0td,$
$b_0wb_1w, b_0wb_1td, b_0tdb_1w, b_0tdb_1td$

trigrams
$s_0es_1es_2w, s_0es_1es_2e, s_0es_1eb_0w, s_0es_1eb_0td,$
$s_0es_1wb_0w, s_0es_1wb_0td, s_0ws_1es_2e, s_0ws_1eb_0td$

separator
$s_0wp, s_0wep, s_0wq, s_0wcq, s_0es_1ep, s_0es_1eq,$
$s_1wp, s_1wep, s_1wq, s_1weq$

extended (Zhu et al., 2013)
$s_0llwe, s_0lrwe, s_0luwe, s_0rlwe, s_0rrwe,$
$s_0ruwe, s_0ulwe, s_0urwe, s_0uuwe, s_1llwe,$
$s_1lrwe, s_1luwe, s_1rlwe, s_1rrwe, s_1ruwe$

disco (Maier, 2015)
$s_0xwe, s_1xwe, s_2xwe, s_3xwe,$
$s_0xtde, s_1xtde, s_2xtde, s_3xtde,$
$s_0xy, s_1xy, s_2xy, s_3xy$
$s_0xs_1e, s_0xs_1w, s_0xs_1x, s_0ws_1x, s_0es_1x,$
$s_0xs_2e, s_0xs_2w, s_0xs_2x, s_0ws_2x, s_0es_2x,$
$s_0ys_1y, s_0ys_2y, s_0xb_0td, s_0xb_0w$

Features from (Tokgöz and Eryiğit, 2015):

counts
$s_0P, s_0C, s_0wP, s_0wC, b_0P, b_0C, b_0wP, b_0wC$

edges
$s_0s_1, s_1s_0, s_0b_0, b_0s_0, s_0b_0e, b_0s_0e$

history
a_0, a_1

remote (Novel, UCCA-specific features)
s_0R, s_0wR, b_0R, b_0wR

Figure A.1: Binary feature templates for TUPA$_{\text{Sparse}}$. Notation:
s_i, b_i: ith stack and buffer items.
w, t, d: word form, POS tag and syntactic dependency label of the terminal returned by $h^*(\cdot)$ (see Appendix A.4).
e: edge label to the node returned by $h(\cdot)$.
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 Lichte, 2009).
P, C: number of parents and children.
R: number of remote children.
a_i: action taken i steps back.

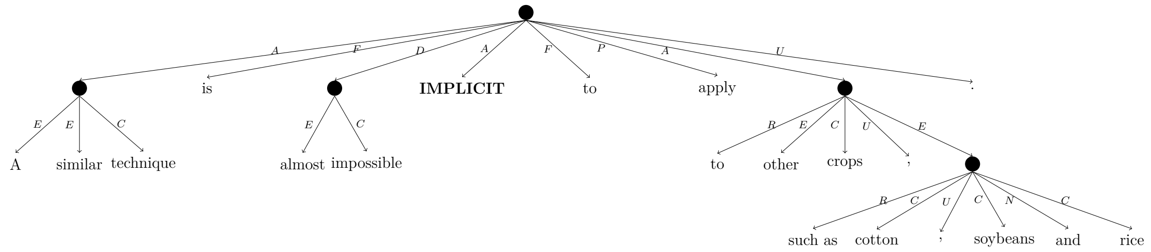


Figure A.3: UCCA example with an implicit unit.

```
**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**
    $u_t \leftarrow \text{Node}()$;
    $V \leftarrow V \cup \{u_t, t\}$, $E \leftarrow E \cup \{(u_t, t)\}$;
    $\ell(u_t, t) \leftarrow \textit{Terminal}$;
    **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**
                $u \leftarrow \text{Node}()$;
                $V \leftarrow V \cup \{u\}$, $E \leftarrow E \cup \{(u, u_t)\}$;
                $\ell(u, u_t) \leftarrow \texttt{Label}(t)$;
            **else**
                $u \leftarrow u_t$;
            **end**
            $E \leftarrow E \cup \{(u_{t'}, u)\}$;
            $\ell(u_{t'}, u) \leftarrow l$;
        **end**
    **end**
**end**
**Function** Label
    **Data:** node $t \in T$
    **Result:** label $l \in L$
    **if** $\text{IsPunctuation}(t)$ **then**
        **return** $\textit{Punctuation}$;
    **else if** $\exists t' \in T : (t, \textit{ParallelScene}, t') \in A$ **then**
        **return** $\textit{ParallelScene}$;
    **else if** $\exists t' \in T : (t, \textit{Participant}, t') \in A$ **then**
        **return** $\textit{Process}$;
    **else**
        **return** $\textit{Center}$;
```

Algorithm 1: Conversion from bilexical graphs.

Appendix B

Multitask Parsing Across Semantic Representations Supplementary Notes

B.1 Features

Table B.1 lists all feature 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 Hershcovich et al. (2017).

s_i refers to stack node i from the top, and b_i to buffer node i. xl and xr refer to a x's leftmost and rightmost children, and xL and xR 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 (Maier and Lichte, 2016), `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"), `^` 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. $C/E/P$).

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

Nodes	**Features**
s_0	`wtdencpT#^$xhqyPCIEMN`
s_1	`wtdencT#^$xhyN`
s_2	`wtdencT#^$xhy`
s_3	`wtdencT#^$xhyN`
b_0	`wtdncT#^$hPCIEMN`
b_1, b_2, b_3	`wtdncT#^$`
$s_0l, s_0r, s_1l, s_1r, s_0ll, s_0lr, s_0rl, s_0rr, s_1ll, s_1lr, s_1rl, s_1rr$	`wenc#^$`
$s_0L, s_0R, s_1L, s_1R, b_0L, b_0R$	`wen#^$`
Edges	
$s_0 \to s_1, s_0 \to b_0$	`xd`
$s_1 \to s_0, b_0 \to s_0$	`x`
$s_0 \to b_0, b_0 \to s_0$	`e`
Past actions	
a_0, a_1	`eA`
Misc.	`node ratio`

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`.

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.

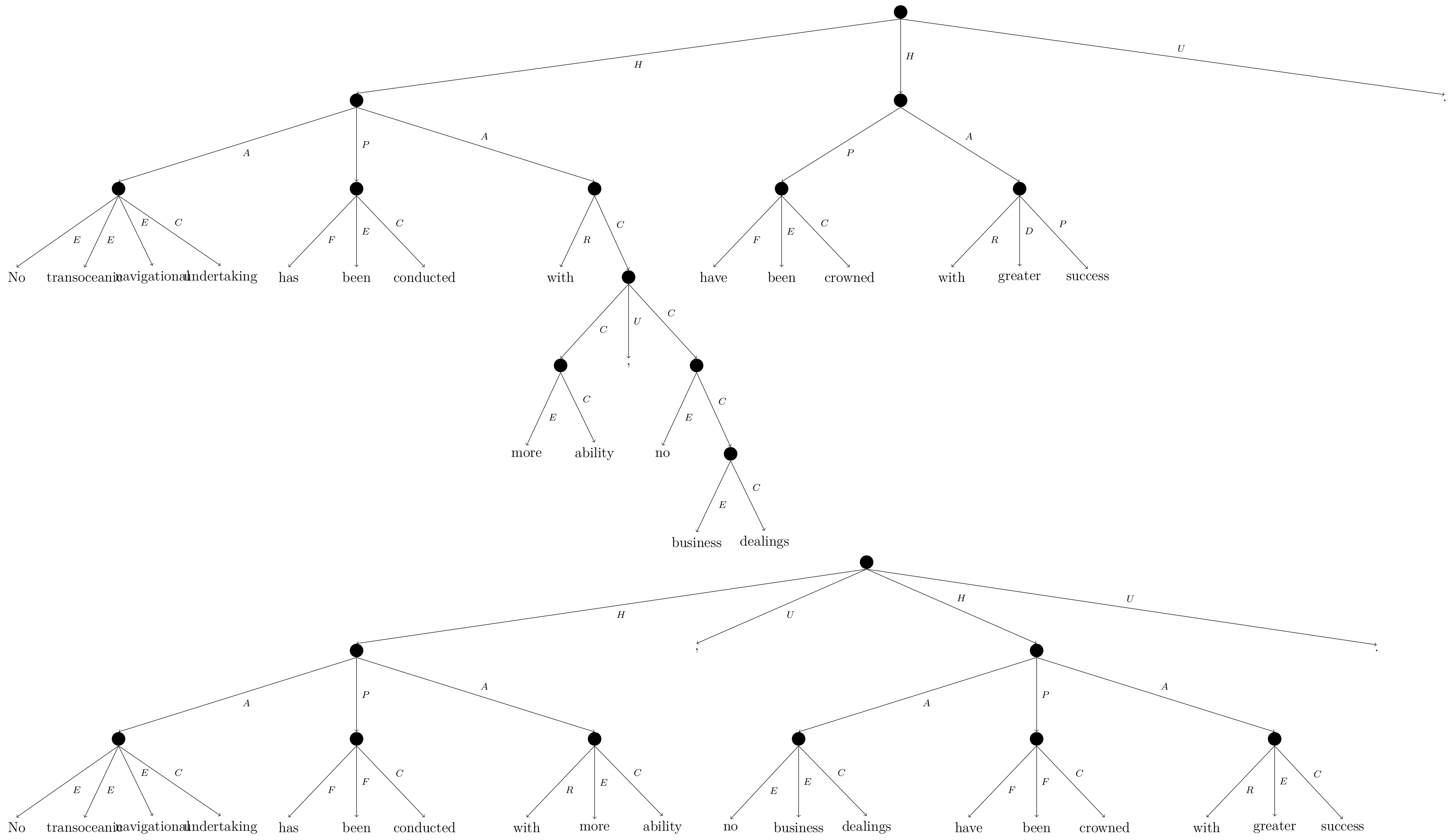


Figure B.1: Output of single-task model on sentence 53001 from the English 20K test set (top), and of MTL model using all of AMR, DM and UD^{++} as auxiliaries on the same sentence (bottom).

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).
S	**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)	`fixed`	fixed multiword expression
`advcl`	adverbial clause modifier	`flat`	flat multiword expression
`advmod`	adverbial modifier	`goeswith`	goes with
`amod`	adjectival modifier	`iobj`	indirect object
`appos`	appositional modifier	`list`	list
`aux`	auxiliary	`mark`	marker
`case`	case marking	`nmod`	nominal modifier
`cc`	coordinating conjunction	`nsubj`	nominal subject
`ccomp`	clausal complement	`nummod`	numeric modifier
`clf`	classifier	`obj`	object
`compound`	compound	`obl`	oblique nominal
`conj`	conjunct	`orphan`	orphan
`cop`	copula	`parataxis`	parataxis
`csubj`	clausal subject	`punct`	punctuation
`dep`	unspecified dependency	`reparandum`	overridden disfluency
`det`	determiner	`root`	root
`discourse`	discourse element	`vocative`	vocative
`dislocated`	dislocated elements	`xcomp`	open clausal complement
`expl`	expletive		

Table C.2: UD v2 relations.

תקציר

מאמץ רב בעיבוד שפה טבעית מוקדש להבנת שפה טבעית, תחום אשר שם לעצמו כמטרה להיות מסוגל להבין טקסט, להקיש ממנו היסקים, ולפעול על פיו באופן מלומד. בעוד לשימושים מסויימים ניתן להשתמש בשיטות פשוטות יחסית, אשר מתעלמות לחלוטין מסדר המלים (מודלים מסוג שק מלים) או מתייחסות אליו כשרשרת פשוטה (כמו שיטת הרצף-לרצף הנפוצה, המאפשרת לרשתות נוירונים ללמוד משימות באופן כולל), הבנת טקסט באופן כללי דורשת ייצוג היררכי של משמעות. בניית ייצוג זה מהטקסט היא מטרתה של סדרת עבודות רחבה בתחום הניתוח הסמנטי. בעוד ייצוגים סמנטיים רבים הוצעו עד כה, יש ביניהם הרבה מן המשותף בנוגע לאבחנות הבסיסיות, כמו למשל בין פרדיקטים (יחסים, מצבים ואירועים) ובין ארגומנטים (משתתפים).

תזה זו מתמקדת בייצוג סמנטי מסויים בשם UCCA, אשר העקרונות המנחים העיקריים שלו הם תמיכה בכל התופעות הלשוניות הסמנטיות, תמיכה ויציבות בין שפות, קלות התיוג (אפילו בידי מי שאינו מומחה בבלשנות), וארכיטקטורה מודולרית התומכת בשכבות שונות של אנוטציה סמנטית. מנתח אוטומטי לחלוטין מוצע במסגרת התזה, ומוערך על פני מספר שפות (אנגלית, צרפתית וגרמנית). המנתח, אשר שמו TUPA, מסוגל ללמוד מבנים גרפיים כללים ביותר: גרפים מכוונים חסרי מעגלים על-פני סדרות של מלים עם צמתים פנימיים עבור יחידות מורכבות, אשר יכולים לכסות סדרות לא רצופות של מלים. המחלקה הכללית הזו של גרפים מכסה את המבנים אשר מתוייגים ב-UCCA, וגם ייצוגים אחרים. TUPA ממומש כמנתח מבוסס מעברים, אשר מערכת המעברים שלו תומכת בתכונות המבניות הללו. מסווג המעברים של TUPA הוא רשת נוירונים בעלת רכיב BiLSTM לחישוב ייצוגים עבור הקלט. בהשוואה נרחבת לעומת שיטות המבוססות על המרה למבנים פשוטים יותר, וגם בהשוואה למימושים אחרים של המסווג, נמצא ש-TUPA מסוגל לנתח בדיוק רב יותר טקסט למבני UCCA, גם במצב שבו אוסף הבדיקה דומה לאוסף האימון וגם כאשר הוא נלקח ממקור שונה. תוצאות אלו מודגמות בשלוש שפות.

יכולתו של המנתח מודגמת גם לשתי שיטות אחרות לניתוח סמנטי, DM ו-AMR, וגם לניתוח תחבירי בשיטת UD. ניסוי זה מדגים את גמישותו של המנתח, ואת יכולתו להתמודד עם משימות נוספות מלבד UCCA. בנוסף, כאשר מאמנים את TUPA על מספר משימות בעת ובעונה אחת, ביצועיו משתפרים על UCCA. שיפור זה נעשה באמצעות למידה של הכללות הנוגעות לכל המשימות וחשובות להן.

לבסוף, בהשוואה אמפירית של התוכן בייצוגים סמנטיים ותחביריים, אנו מגלים פנים שונים של הבדל ביניהם. להבדלים אלו יש משמעות רבה בנוגע לתרומה של תחביר לניתוח סמנטי, ועל השימושיות של כל אחת מהשיטות למשימות סמנטיות בעיבוד שפה טבעית.

אני רואה בניתוח סמנטי אמצעי עבור מחשבים ללמוד שפה אנושית. בעוד ייצוגים שונים מתמקדים באבחנות שונות באמצעות מבנים פורמליים שונים, הם חולקים מטרה משותפת, לתמוך ביישומים של עיבוד שפה טבעית, כמו למשל סיווג טקסט לקטגוריות, תיוג לפי תכונות בלשניות, הסקת מסקנות, וייצור טקסט חדש לפי אילוצים מסויימים (כמו בתרגום מכונה). מאגרי המידע המתויגים בכל הייצוגים הם משאב יקר-ערך, אשר ניתן להשתמש בו להשגת שיפור ניכר בעיבוד והבנה של שפה טבעית.

עבודה זו נעשתה בהדרכתם של
פרופ' ארי רפופורט וד"ר עמרי אבנד

ניתוח סמנטי אוניברסלי באמצעות רשתות נוירונים

חיבור לשם קבלת תואר דוקטור לפילוסופיה

מאת
דניאל הרשקוביץ

הוגש לסנט האוניברסיטה העברית בירושלים
פברואר 2019

ניתוח סמנטי אוניברסלי באמצעות רשתות נוירונים

חיבור לשם קבלת תואר דוקטור לפילוסופיה

מאת
דניאל הרשקוביץ

הוגש לסנט האוניברסיטה העברית בירושלים
פברואר 2019