The Interdisciplinary Center, Herzlia
Efi Arazi School of Computer Science
M.Sc. program - Research Track

Joint Morpho-Syntactic Processing of Morphologically Rich Languages in a Transition-Based Framework

by
Amir More

M.Sc. dissertation, submitted in partial fulfillment of the requirements for the M.Sc. degree, research track, School of Computer Science
The Interdisciplinary Center, Herzliya

September 2016

This work was carried out under the supervision of Dr. Reut Tsarfaty of The Open University of Israel and Prof. Ariel Shamir from the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya.

IDC HERZLIYA

Joint Morpho-Syntactic Processing of Morphologically Rich Languages in a Transition-Based Framework

by

Amir More

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the
IDC Herzliya
Efi Arazi School of Computer Science

February 6, 2017

"Nature is full of infinite causes that have never occurred in experience."

Leonardo Da Vinci

IDC HERZLIYA

Abstract

IDC Herzliya
Efi Arazi School of Computer Science

Master of Science

by Amir More

State-of-the-art results for morph-syntactic analysis of Morphologically Rich Languages (MRLs) such as Hebrew are currently too low to enable real-world applications that are successfully implemented for heavily-studied languages like English. This is because existing frameworks for (morpho)syntactic analysis rely on fundamental structuralists' assumptions, and in particular, assume a strict separation between morphological and syntactic processing. This assumption breaks down in the context of MRLs.

In this work we present a general-purpose transition-based framework that implements standalone lexicon-based and data-driven morphological analyzers, a standalone morphological disambiguator, a standalone dependency parser, and a joint morpho-syntactic dependency parser for MRLs. Each of these tasks is defined via a transition system, a cross-linguistic feature model, and a global scoring function. The learning problem is solved using the structured perceptron, and efficient decoding is achieved via beam-search. We present state-of-the-art results for each of the tasks in isolation, and present the first joint transition-based system for morphological segmentation and dependency parsing for Modern Hebrew.

We illustrate the utility and multilingual coverage of the data-driven morphological analyzer and disambiguator by morphologically analyzing and disambiguating a large set of 48 languages in the Universal Dependency treebanks (http://universaldependencies.org).

Acknowledgements

I would like to express my deep appreciation for my advisor, Dr. Reut Tsarfaty, for sparking my interest in the field of NLP and accompanying me on this journey of curiosity. I am grateful for her dedication to this research and her invaluable insight, advice and support. I thank her and her family for the countless hours and sleepless nights working with me on submissions and presentations.

I would like to thank the organizations that enabled my work. I am grateful for my alma mater IDC Herzliya; for the scholarships I received for my B.A. and M.Sc. in Computer Science, their patience in the pursuit of my thesis, and the supportive faculty who taught me the building blocks. I thank the Weizmann Institute of Science and the Open University for their facilities. I would like to acknowledge the professors of the well thought out online courses I attended, with a special thanks to Michael Collins, whose clarity helped me grasp essential concepts. I extend my gratitude to the countless programmers and engineers of open source projects whose work empowers mine.

I would like to acknowledge the community of researchers who enabled my work; the giants on whose shoulders I stand. I am grateful to the Knowledge Center for Processing Hebrew (MILA) at the Technion who originally developed the Hebrew resources for my work, and the numerous researchers who processed and maintained these resources since. I would like to acknowledge the researchers whose work I have studied in awe, Joakim Nivre and Yue Zhang; for their research that laid the groundwork for mine. I would like to especially thank Yoav Goldberg, whose brilliant insight aided various stages of this work.

I thank the anonymous reviewers of my submissions for their time and thoughtful reviews. Every rejected submission lead to insight and new discoveries that resulted in better performance. In hind sight, I would expect nothing less than the unrelenting demand for perfection from the top echelon of science.

I would like to thank the Public Knowledge Workshop and Karine Nahorn, who caused me to ask the question whose answer resulted in this thesis.

Finally, I would like to express my love and appreciation for all those who supported me throughout this endeavour: my close friends, for enduring me as I worked on this thesis; my parents, whose unwavering love and support are the reason I have come so far; my sister, for being the indomitable cookie-monster with whom I am genetically entwined; my brothers, in-laws, and their families for their support; and for Love.

Contents

List of Figures

List of Tables

List of Algorithms

For my close friends, family, and Love

Chapter 1

Introduction

> "One morning I shot an elephant in my pajamas. How he got into my pajamas, I'll never know."
>
> *Groucho Marx*

1.1 Syntactic Parsing of Natural Language

Natural Language Processing (NLP) is a field of Computer Science concerned with the study of automated analysis and processing of natural language. Automatic analysis allows for a range of applications such as translation, information extraction and information retrieval. Parsing is the task of automatically analyzing the syntactic structure of a sentence. A parser receives as input a phrase or a sentence in natural language and returns its underlying structure, as conformed to a syntactic theory such as dependency grammar (Kübler, McDonald, and J. Nivre, 2009) or phrase-structure grammar (Chomsky, 1965).

For example, figures 1.1 and 1.2 show two types of trees that represent the syntactic analysis of the sentence "I shot an elephant in my pajamas". The two trees show representations of the sentence under different syntactic theories; tree 1.1 conforms to phrase-structure grammar and tree 1.2 conforms to dependency grammar.

In a dependency grammar, a sentence is represented as a directed tree, where a node is a word and an arc is a (labeled) dependency relation between words. A dependency relation is a directed arc from the head to the dependent (or modifier), and the label indicates the type of relation between them.

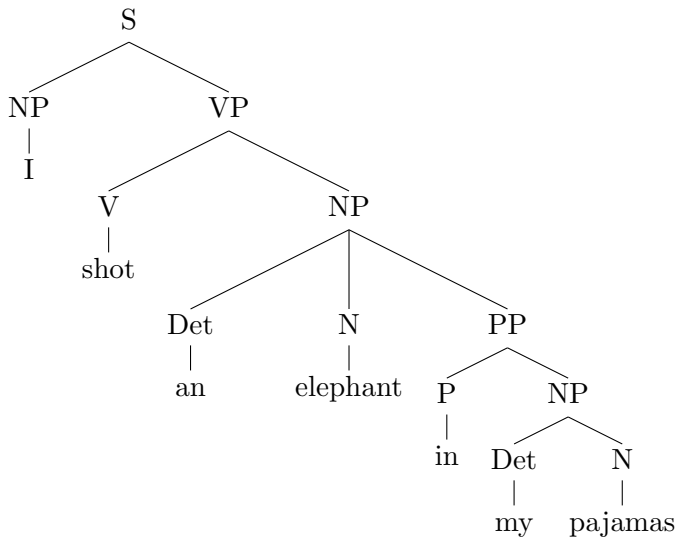

FIGURE 1.1: A constituency tree for the sentence "I shot an elephant in my pajamas"

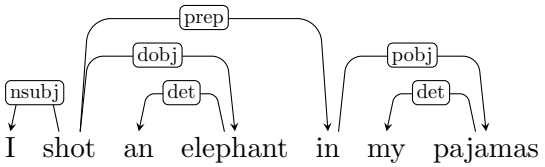

FIGURE 1.2: A dependency tree for the sentence "I shot an elephant in my pajamas"

Dependency parsing is the task of predicting the dependency tree of a given sentence. The approaches to this task can be clustered into (at least) two groups:

- Rule-based - The rules of a formal grammar are used to generate trees.
- Data-driven - A sample set of parsed sentences (corpus) is used to generate a feature model, which is used to predict the tree structure of unseen sentences.

A prevalent challenge in analyzing natural language is ambiguity. In particular, there can be more than one interpretation for a sentence in natural language. For instance, there exist two plausible interpretations of the sentence "I shot an elephant in my pajamas":

- I shot an elephant, and I did the shooting in my pajamas (i.e. wearing pajamas)
- I shot an elephant, and the elephant is literally in my pajamas

The trees of figures 1.1 and 1.2 represent the syntactic structure of option (1). A key task of any parser is to disambiguate the analysis, i.e., pick out the most plausible interpretation as perceived by humans.

Since there may be more than one grammatically correct tree for a sentence, a dependency parser must deal with ambiguities and choose a single tree that represents the most plausible human interpretation (disambiguation).

In data-driven approaches disambiguation is supported by a corpus, which helps to discriminate which of the many alternatives is more likely to be correct under human interpretation. In figure 1.2, we can see the ambiguity of the phrase ".. in my pajamas" which may be modifying the phrase "an elephant" or the phrase "I", since both options are grammatically correct, therefore automatic analysis must disambiguate and choose one. A data-driven approach can discriminate between them, given examples in the corpus that would indicate that "I" is a noun more likely to be in its pajamas, than an elephant being in them.

State-of-the-art dependency parsing results reported for English present high accuracy on community-accepted metrics (Labeled Attachment Score - LAS) (Andor et al., 2016), high enough to enable a variety of applications; e.g. machine translation and information retrieval. However, for many Semitic languages, state-of-the-art results are significantly lower (Seddah, Kübler, and Tsarfaty, 2014), enough so that building applications on current results is not feasible.

Furthermore, parsing models are often customized per language. Developing a cross-linguistic model — that is, achieving language independence — would enable a one-size-fits-all solution, but the various different properties of languages prove to be a formidable barrier because they require the addition of new dimensions to the linguistic model (Tsarfaty, Seddah, et al., 2010), some of which we review in turn.

1.2 Morphological Ambiguity

To demonstrate how the structure of a language relates to its model, let us compare English and Hebrew analyses in the context of dependency grammar. Dependency grammar assumes that each word in a sentence maps to a node in the dependency tree, as defined by Kübler, McDonald, and J. Nivre (2009, Chapter 2, p. 12). This assumption is violated in morphologically rich languages (MRL) such as Modern Hebrew, where each sentence is a series of tokens, and each token is a composition of morphemes (Tsarfaty, Seddah, et al., 2010).

For a sentence in an MRL, the nodes of its dependency tree are morphemes. For example, in figure 1.3 we see the dependency tree for the sentence HLKTI LIM[1] ("I

[1]We use the transliteration scheme of Sima'an et al. (2001)

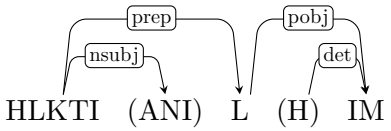

FIGURE 1.3: A dependency tree for the sentence "HLTKI LIM" ("I walked to the sea")

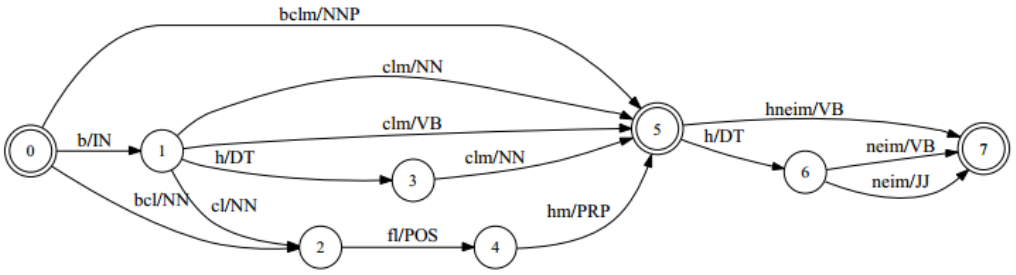

FIGURE 1.4: An example of an MA lattice of the phrase "BCLM HNEIM" ("in their pleasant shadow") in transliterated Hebrew. Edges mark syntactic words, and double circles mark white spaces. Figure taken from Tsarfaty and Goldberg (2008)

walked to the sea"), whose morphemes are HLKTI-(ANI) L-(H)-IM (Walked-(I) to-(the)-sea). The composition rules dictate that ANI (the noun "I") is dissolved into HLKTI, whereas L-(H)-IM (to-(the)-sea) is a complex composition in which H (the) is implicit. The arc $(HLKTI, prep, L)$ $((Walked, prep, to))$ requires that the morpheme L (to) be distinct from the token LIM (to-(the)-sea).

Unfortunately, the task of mapping tokens to morphemes is non-trivial. While the composition of a sequence of morphemes is relatively straightforward, the decomposition (henceforth, *spell-out*, the spelling out of the token into morphemes) has numerous possibilities, leaving ambiguity as to which decomposition is the correct one for the dependency tree. This ambiguity may be represented as a lattice structure, as seen in figure 1.4. The task of choosing which spellout is correct is called *morphological disambiguation* (MD).

There are three separate sub-tasks of morphological disambiguation. First, *segmentation* is the task of decomposing the token into a series of forms. For example, in figure 1.4, the paths "b/IN-clm/NN" and "b/IN-clm/VB" have the same segmentation: "b-clm". Second, *tagging* is the task of choosing a part-of-speech tag for each form, i.e. given the segmentation "b-clm", setting "b"'s tag to "IN", and choosing (tagging) either "NN" or "VB" for "clm". Also, each form/tag pair may have a set of *morphological features*. A morphological feature is usually a binary indicator, such as gender (m/f), person (first/second/third), tense (past/present/future) etc. that represent inflection[2]. Full morphological disambiguation requires correct segmentation, tagging, and choosing

[2]We omit morphological features in the morphological analysis lattice of "BCLM HNEIM" for brevity

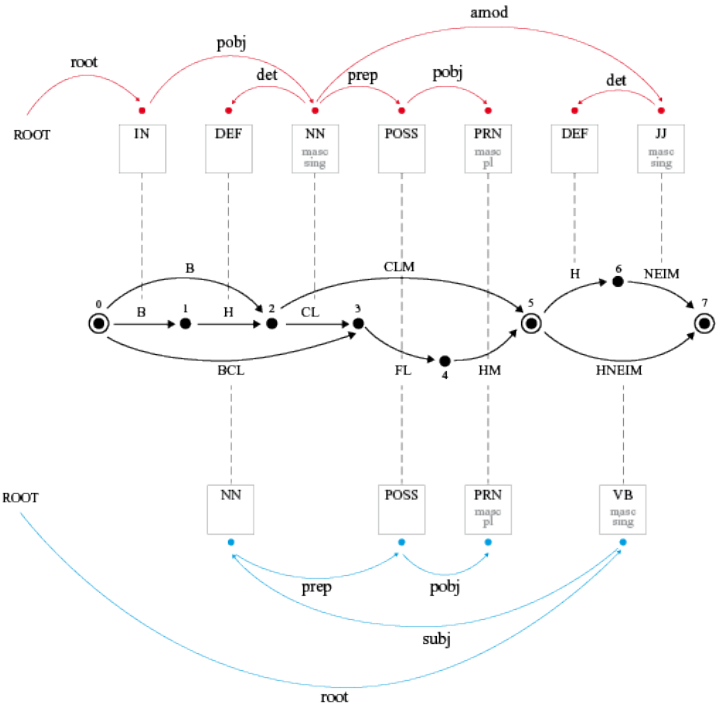

FIGURE 1.5: A visualization of the relationship between morphology and syntax for the phrase "BCLM HNEIM".
This figure is copyrighted to **tsarfatybook** ; it is presented here with permission from the author.
Do not reproduce or distribute without written permission.

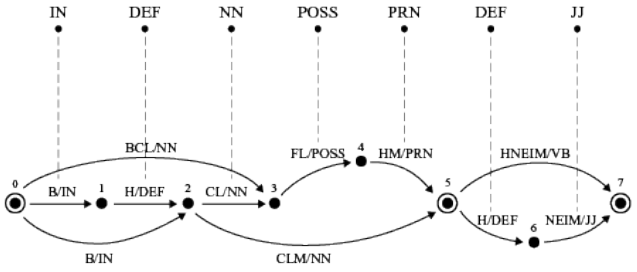

FIGURE 1.6: The morphological disambiguation of "BCLM HNEIM".
This figure is copyrighted to **tsarfatybook** ; it is presented here with permission from the author.
Do not reproduce or distribute without written permission.

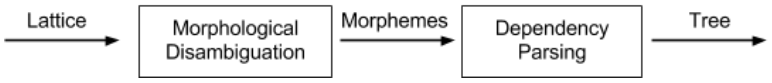

FIGURE 1.7: Pipeline processing - separate stages

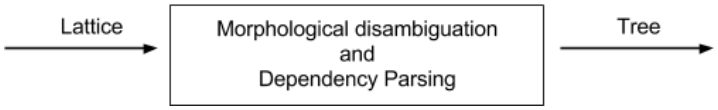

FIGURE 1.8: Joint processing - a single step

the correct set of morphological features. The form, tag, and morphological features, taken together, form the elements of a morpheme — the formal unit represented by an arc in a morphologically ambiguous lattice (3.1).

Figure 1.4 shows a lattice visualizing morphological ambiguity for the phrase BCLM HNEIM.

By itself, the token BCLM is morphologically ambiguous. Its selected path in the lattice (spellout) should depend on the context in which it is used. Figure 1.5 shows different dependency trees, depending on the morphological disambiguation of BCLM HNEIM. In context, the correct spellout is B-CL-FL-HM ("in their shadow", as in Figure 1.6.

To summarize, the complex word structure of MRLs pose numerous problems for dependency parsing, notably:

- A token is not a node in the dependency tree but a composition of morphemes
- A token has multiple possible spellouts, each one admits different sets of trees
- The correct spellout depends on context
- Morphemes can have inflectional properties, which can be the only differentiating feature between ambiguous dependencies

1.3 The Joint Hypothesis

The overall goal of this MSc thesis is to investigate parsing methods in the context of MRLs and to develop an effective and efficient dependency parser for Modern Hebrew texts and for other languages with similar properties.

In principle, in order to solve the problems introduced by MRLs, morphological disambiguation is used to map the tokens of sentences onto their morphemes. In the context of Modern Hebrew, the first approach attempted in Goldberg and Elhadad

(2010) is a pipeline model as shown in Figure 1.7, where morphological disambiguation is separated from syntactic analysis. However, their results indicate that a pipeline model significantly impacts results of dependency parsing. While adequate results are achieved for correctly disambiguated data (gold-standard morphology and segmentation), parsing performance using automatically disambiguated data is low.

A second approach is joint morphological disambiguation and syntactic processing, as shown by Figure 1.8. In this approach, morphological and syntactic disambiguation decisions are interwoven, such that dependency information is made available for morphological disambiguation (choosing a spellout from the lattice), and vice versa. This was shown to be effective for constituency parsing for Semitic languages by Goldberg and Tsarfaty (2008); the purpose of this work is to show that this holds for dependency parsing as well.

We propose to use joint morphological disambiguation and syntactic processing in order to correctly generate a dependency tree for a sentence consisting of morphologically ambiguous tokens. We hypothesize that using joint processing will provide mutual benefits between morphological disambiguation and syntactic decisions: syntactic context made available *during* morphological disambiguation will result in better disambiguation decisions; in turn, better disambiguation will provide the *opportunity* of better syntactic decisions, in the form of tokens' correct spellouts in the context of a sentence.

To this end, we introduce a new morphological disambiguator in a framework of dependency parsers for non-MRLs. For dependency parsing, we reproduce an English dependency parser in the same framework and experiment with variations that may better apply to MRLs.

We then test the hypothesis using a novel joint processing system, and compare it to the pipeline approach for Modern Hebrew. We verify our results for morphological disambiguation on 48 languages of the Universal Dependencies (Nivre, Agić, et al., 2016), showcasing the cross-linguistic applicability of the model.

Our results prove the hypothesis in one direction. We show that joint processing results in better morphological disambiguation. However, end-to-end dependency parsing performance of our pipeline-based baseline remain better than those of our joint processing setup.

Chapter 2

Related Work

> "We are like dwarfs sitting on the shoulders of giants. We see more, and things that are more distant, than they did, not because our sight is superior or because we are taller than they, but because they raise us up, and by their great stature add to ours."
>
> *John of Salisbury, Metalogicon*

We present a survey of related work for the morphological disambiguation and dependency parsing tasks, in standalone and end-to-end settings.

We first showcase existing approaches to the standalone task of morphological disambiguation (2.1), in both Hebrew-specific and cross-linguistic MRL settings. We then present a short background and summary of approaches in the literature for standalone dependency parsing, present the latest work for state-of-the-art dependency parsing of English, and discuss its applicability to Modern Hebrew (2.2).

Next, we cover two approaches for the collective task of both morphological disambiguation and dependency parsing: pipeline and joint processing. We present current results for the pipeline approach (2.3.1), and survey recent work advocating for the joint approach in Hebrew in a generative setting, as well as similar tasks in other languages (2.3.2).

2.1 Morphological Disambiguation

Language-specific morphological analysis and disambiguation of Modern Hebrew has been addressed in previous work using Hidden Markov Models (HMM). Bar-haim, Sima'an, and Winter (2008) use standard HMMs, while Adler (2007) offers an unsupervised approach together with HMMs, currently considered the state-of-the-art on the MILA Hebrew Treebank (Itai and Wintner, 2008). Adler bootstraps his model with some pre-set data and an initial supervised corpus, after which learning is achieved through expectation-maximization over a large corpus. While Adler's results are adequate for some downstream applications that require morphological disambiguation, they significantly impede other applications that require higher accuracy. Indeed, using the output of Adler's morphological disambiguator for dependency parsing significantly impacts results, to the extent that dependency parsers' output is rendered sub-par and unusable for practical applications (Goldberg and Elhadad, 2010).

While multilingual/cross-linguistic models exist for morphological tagging and feature disambiguation (Mueller, Schmid, and Schütze, 2013), they all require morphological segmentation before they can be applied. However, in Modern Hebrew, morphological segmentation is virtually inseparable from morphological tagging and feature disambiguation, since segmentation may induce a specific tag and set of morphological features. For example, in the morphological analysis of "BCLM HNEIM" as shown in figure 1.4, note that the segment "BCLM/NNP" of the first token uniquely determines the segment's tag. This exemplifies the reason: requiring pre-segmented Hebrew input just shifts the focus of morphological disambiguation to the segmenter.

For generic morphological segmentation, Morfessor (Smit et al., 2014) uses a max-likelihood in semi-supervised settings, but it cannot handle the rich labeling of morphological segments.

2.2 Dependency Parsing

Automated parsing of natural language into a dependency grammar is an extensively and widely studied subject, with many approaches and subtle variations. We present a short summary of this large body of work, narrowing down to seminal concepts that are relevant to this work. We regard only data-driven approaches. Note that all current approaches assume gold-standard input, pre-segmented and tagged, including morphological features where relevant.

There are two major approaches in recent literature for data-driven dependency parsing: graph-based and transition-based. In graph-based processing, the words of a sentence are represented as nodes in a graph, with weighted edges between them as potential dependency arcs. A search algorithm like Maximum Spanning Tree is used to find a tree thus computing a dependency parse of the input. In contrast, in transition-based processing (to be formalized in chapter 3), a sequence of transitions incrementally generates a dependency tree, simulating a "left-to-right" reading of a sentence, like a human reader.

A third approach to parsing, called Easy-First, introduced by Goldberg and Elhadad (2010), is a parsing method where syntactic decisions are not made left-to-right, but in "easy-first" order, wherein obvious (easy) dependencies are processed first, and possible dependencies can span the entire length of a sentence. This allows for early consideration of dependencies between far placed words, usually attributed to graph-based approaches, while admitting an $O(nlogn)$ run time. This approach was evaluated on gold-standard and predicted, morphologically disambiguated, Modern Hebrew; we report their results and use them as a baseline.

Graph-based systems originally achieved higher accuracy than their transition-based counterparts, albeit at the price of run-time. Graph-based parsing *explicitly* parametrizes models over substructures of a dependency tree, while early transition-based systems relied on local, *indirect* parametrization over transitions to construct a tree (Kübler, McDonald, and J. Nivre, 2009, Chapter 4, p. 41). However, when inquiring into the use of high-order features, embodying the value of deep and complex linguistic relationships, into graph-based systems, McDonald and Pereira (2006) and Mcdonald (2007) found that the resulting complexity would be NP-hard.

Zhang and Clark (2011) introduced a transition-based framework wherein a *global* scoring function and beam-search decoding algorithm mitigates the inherent limitations of transition-based processing. Building on their work, the seminal paper of Zhang and Nivre (2011) introduced a set of features that capture similar high-order relationships, such that transition-based parsing achieved results equal to state-of-the-art graph-based approaches at the time. The transition-based approach runs in linear time, such that with these new features, accuracy and speed need not be a tradeoff.

Since then, much work has gone towards advancing transition-based processing. The concept of word embedding introduced by the seminal work of Mikolov et al. (2013) (a.k.a. word2vec) allowed for words to have vector representations, such that syntactic and semantic similarities are embodied in the vector space. This representation provides a better generalization opportunity for statistical models, explicit and implicit (Goldberg, 2015).

In recent years, the latest approach to machine learning and structured prediction tasks such as dependency parsing seeks to replace engineered representation models entirely, replaced by neural networks that may learn to induce a model automatically (Goldberg, 2015). Most recently, Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN) originally described by Hochreiter and Schmidhuber (1997), evolved and returned to center stage. LSTMs are a representation of context capable of learning long-term dependencies; i.e. for the sentence "I grew up in France.. I speak fluent French", LSTMs can help predict the word 'French', where 'France' can augment the prediction after the trigram "I speak fluent".

At the time of writing, the state-of-the-art English dependency parser uses pre-defined word embeddings, Bi-Directional LSTMs (LSTMs run forward and backward), and a multi-layered perceptron (Kiperwasser and Goldberg, 2016). However, these results are currently inapplicable to Modern Hebrew due mostly to the issue of Hebrew's morphologically rich and ambiguous nature, for which word embeddings and LSTMs (and especially Bi-LSTMs) are as yet undefined and non-trivial. Additionally, using these models would require morphologically disambiguated input, preventing syntax from affecting morphological disambiguation.

In this work, we focus on the *architecture* of the solution, although we may supplant the learning model in the future.

2.3 Morphological and Syntactic Processing

2.3.1 Pipeline Approaches

The first approach to test end-to-end morphological disambiguation and dependency parsing of Modern Hebrew using a modern corpus was presented in Goldberg and Elhadad (2010), where they evaluated the impact of morphology predicted by Adler's MD compared to gold-standard on their Easy-First dependency parsing model, as well as two other prominent models at the time. Table 2.1, as reported in Goldberg and Elhadad (2010), shows the loss of accuracy noted for the best results of all models, with a concluding suggestion that joint processing may alleviate this problem.

Recently, UDPipe (Straka, Hajic, and Straková, 2016) uses a toolkit with multiple independent components for multilingual morphological analysis, segmentation, tagging, morphological features, and dependency parsing. UDPipe underperforms Adler on Hebrew for all morphological disambiguation tasks both independently and cohesively - we attribute this to the underlying components addressing morphological segmentation and tagging as separate tasks and a lack of proper morphological analysis.

Model	Gold Morph.	Predicted Morph.
$MST1$	83.6	75.6
$MST2$	84.4	76.4
$MALT$	80.7	73.4
$EasyFirst$	84.2	76.2

TABLE 2.1: Reported results in Goldberg and Elhadad (2010), showing the impact of predicted morphology on dependency parsing of Modern Hebrew
The measure is UAS - best Unlabeled Attachment Score

In this work we aim to cover all stages of UDPipe (with the exception of tokenization), but in a *joint* architecture.

2.3.2 Joint Approaches

Joint processing of Modern Hebrew has been substantially advocated for in recent years. Tsarfaty and Goldberg (2008) introduced joint morphological and syntactic processing in a generative (PCFG) setting, showing gains in accuracy over previous pipeline-based approaches.

For transition-based dependency parsing in a multilingual setting, Bohnet and J. Nivre (2012) first integrated tagging and syntactic processing improving state-of-the-art accuracy over pipeline approaches. Andor et al. (2016) use the joint transition system proposed in Bohnet and J. Nivre (2012), but improve the model using a globally normalized neural network producing state-of-the-art results for tagging and dependency parsing on untagged input; with parsing accuracy comparable to Kiperwasser and Goldberg (2016) (who require tagged input). Bohnet, J. Nivre, et al. (2013) add morphological feature disambiguation and lexicalization to the joint transition system of Bohnet and J. Nivre (2012) for richly inflected languages, again improving state-of-the-art accuracy for all languages evaluated. However, these systems do not provide a solution for joint morphological analysis and disambiguation of morphologically rich languages such as Modern Hebrew.

For various Chinese parsing tasks, joint systems for tagging and/or segmentation (glyph chunking) together with syntactic parsing have been shown to outperform pipeline settings (Z. Li, Min Zhang, Che, Liu, Chen, and H. Li, 2011; Bohnet and J. Nivre, 2012; Z. Li, Min Zhang, Che, Liu, and Chen, 2014; Meishan Zhang et al., 2014b).

In this work, we present a joint transition system that performs well on Modern Hebrew, yet at the same time is applicable to many other languages, including non-MRLs.

Chapter 3

Approach and Formal Preliminaries

"Le bon Dieu est dans le detail."

Anonymous

We present formal definitions of morphological analysis and disambiguation (3.1), labeled dependency trees (3.2), and the processing framework in which we develop standalone and joint processors (3.3).

We then state our research objectives and the open questions we seek to answer in this work (3.4).

3.1 Morphological Analysis and Disambiguation

In Morphologically Rich Languages (MRLs) such as Modern Hebrew, written sentences are sequences of space-delimited tokens rather than words. A token is a composition of morphemes. For example, in the phrase BCLM HNEIM (In their pleasant shadow), the token BCLM is a composition of the morpheme sequence B-CL-FL-HM (In-shadow-of-(them)).

Formally, a morpheme m is described by its morpho-syntactic representation (MSR) as a triple (f, t, g) with a form f, a part-of-speech tag t, and a set g of attribute:value grammatical properties. In a word-lattice, the MSR is embedded in an edge as the tuple (s, e, f, t, g), where s, e are start and end nodes in the lattice, respectively.

Although the morphemes compose deterministically as a token, there are multiple morphemes sequences that when composed, result in the same token. Therefore, given

a token, its decomposition (or *spellout*, the spelling out of a token into morphemes) is ambiguous and depends on the context in which it is found. To represent this ambiguity, the possible decompositions of a token are represented as a word-lattice, as shown in figure 1.4. The term Morphological Disambiguation (MD) refers to selecting a single path through the morphological analysis (MA) lattice.

Let $x = x_1...x_k$ be an input sentence of k tokens and $L = MA(x_1), ..., MA(x_k)$ be the morphological ambiguity lattice for x, where L is a contiguous series of word-lattices $L_i = MA(x_i)$ connected top to bottom, as illustrated in Figure 1.4. Each word lattice L_i is a set of sequences of morphemes, and each sequence is a single disambiguated analysis for x_i.

3.2 Labeled Dependency Trees

A dependency graph is a directed graph G consisting of a set of V nodes, a set of E arcs, and a linear precedence order $<$ on V. In MRLs, V is a set of *morphemes*, each belonging to a spellout of one of a given sentence's space-delimited tokens. V is therefore a subset of all possible morphemes of a MRL.

A labeled dependency graph has a label function associating a label $l \in L$, a set of dependency labels for each arc in E: $(i, j) \rightarrow l$, which we denote as (i, l, j). Let $i \rightarrow j = (i, j) \in E$ and $i \rightarrow^* j = (i = j \vee \exists k : i \rightarrow k, k \rightarrow^* j)$ There are a few formal conditions for Dependency Graphs:

- G is weakly connected: For every node i there exists a node j such that $i \rightarrow j$ or $j \rightarrow i$
- G is acyclic: if $i \rightarrow j$ then $\neg j \rightarrow i$
- G has a single head: if $i \rightarrow j$, then not $k \rightarrow j$ for any $k \neq j$

A graph G is said to be projective if it maintains the following constraint: if $i \rightarrow j$ then $i \rightarrow^* k$ for any k such that $i < k < j$ or $j < k < i$. In words, projectivity requires that it is possible to draw the dependency arcs above the nodes of a dependency graph such that the arcs never intersect.

Given an arc $i \rightarrow j$, we may call i the head of j, and j a dependent of i.

3.3 The Framework

We formally define the framework employed by Zhang and Nivre (2011), as described by Zhang and Clark (2011), for transition-based syntactic processing using the generalized perceptron and beam search. In the context of this framework, we shall propose a novel transition system for morphological disambiguation of Modern Hebrew such that it may be integrated with existing syntactic processing transitions systems.

The framework contains three relevant parts: a transition system (3.3.1), a statistical feature model (3.3.2) and beam-search (3.3.3).

3.3.1 Transition Systems

A transition system is an abstract machine, consisting of a set of configurations (states) and transitions between configurations. As opposed to a simple finite state automaton, transition systems have complex configurations with internal structure, instead of atomic states, and transitions that correspond to steps in the derivation of the desired output (Kübler, McDonald, and J. Nivre, 2009, Chapter 3).

Formally, a transition system is a quadruple $S = (C, T, c_s, C_t)$, where C is a set of configurations, T is a set of transitions, c_s is an initialization function, and $C_t \subseteq C$ is a set of terminal configurations. A transition sequence y of length n, $y = c_0, t_1(c_0), ..., t_n(c_{n-1})$, starts with an initial configuration $c_0 = c_s(x)$ for the input sentence x and ends with a terminal configuration $c_n \in C_t$, where $t_i \in T$ and $c_n = t_n(c_{n-1}) \in C_t$.

Data:	$c_s(x = x_1, ..., x_n) = ([0], [1, ..., n], 0)$	
Terminal:	$C_t = \{c \in C \mid c = ([0], [], A)\}$	
Transitions:	$(\sigma, [i\|\beta], A) \rightarrow ([\sigma\|i], \beta, A)$	$(SHIFT)$
	$([\sigma\|i], [j\|B], A) \rightarrow (\sigma, [j\|B], A \cup \{(j, l, i)\})$[1]	$(LEFT\text{-}ARC_l)$
	$([\sigma\|i], [j\|B], A) \rightarrow (\sigma, [i\|B], A \cup \{(i, l, j)\})$	$(RIGHT\text{-}ARC_l)$

Algorithm 1: The arc-standard transition system for dependency parsing (Kübler, McDonald, and J. Nivre, 2009)

In Algorithm 1, the arc-standard transition system for dependency parsing presented in section 6.2 is shown, where one of three transitions may be chosen given a configuration.

3.3.2 Transition Prediction with the Generalized Perceptron

"Prêcher le faux pour savoir le vrai."

French Proverb

In a data-driven approach, a parametric model is defined as a means of predicting which transition to apply at each configuration. The parametric model drives the parser by predicting the transition to be made at each step of the sequence.

The framework first defines an objective function F where x is the input sentence and $GEN(x)$ is the set of possible transition sequences for x:

$$F(x) = argmax_{y \in GEN(x)} Score(y) \tag{3.1}$$

To compute $Score(y)$, $y \in GEN(x)$ is mapped to a global feature vector $\Phi(y) \in N_d$, where each feature is a count of occurrences of a pattern defined by a feature function ϕ. The feature vector $\Phi(y)$ is defined via a set of d feature functions $\{\phi_i\}_{i=1}^{d}$. The way Φ is defined effectively determines the quality of the parser, since the feature model captures linguistic information to which the model learns to assign weights. Given this vector, $Score(y)$ is computed by multiplying $\Phi(y)$ with a weights vector $\vec{\omega} \in R^d$.

$$Score(y) = \Phi(y) \cdot \vec{\omega} = \sum_{c_j \in y} \sum_{i=1}^{d} \omega_i \phi_i(c_j) \tag{3.2}$$

Input: training examples (x_i, y_i)
Data: set $\vec{\omega} = 0$
  for $r = 1..P, i = 1..N$ **do**
    calculate $z_i = decode(x_i)$
    if $z_i \neq y_i$ **then**
      $\vec{\omega} = \vec{\omega} + \Phi(y_i) - \Phi(z_i)$
Output: $\vec{\omega}$

Algorithm 2: Generalized perceptron training algorithm

The system learns the weight vector $\vec{\omega} \in R^d$ via the generalized perceptron. The transition system provides an oracle function which, given an expected output for a given input, emits the correct transition for each configuration such that the complete sequence generates the desired output. Algorithm 2 iterates through a gold-annotated corpus, where each sentence is disambiguated (decoded) with the last known weights, and if the decoded result differs from the gold standard, the weights are updated by adding to weights of features of the gold standard and subtracting from weights of features of the incorrect parse. The algorithm iterates over the corpus repeatedly, stopping when overfitting begins to occur.

Overfitting is determined by applying the model at the end of each iteration to a development set, a subset of the gold-annotated corpus set aside such that we may measure the system's performance using a given measure. When the system's performance

decreases twice sequentially, we cease iteration, and use the model that resulted in the best scoring iteration.

3.3.3 Beam-Search Decoding

In a deterministic transition-based parser, only a single sequence of transitions is explored. As a consequence, any mistake made at the beginning of the sequence will necessarily propagate. To overcome this, the framework of Zhang and Clark (2011) utilizes the generic beam-search algorithm 3 for decoding. In beam search, an agenda of B-best partial outputs (sequences) are maintained. At each step, every candidate in the agenda is expanded upon to the next set of possible candidates. At the end of the step, candidates' partial global scores (as defined by $Score$) are incrementally updated, the top B candidates are maintained and the search continues until a candidate with a full tree is produced. By allowing for a number of candidates to be evaluated concurrently, beam search mitigates error propagation as observed with a deterministic parser.

It is important to note the interplay between beam search and the generalized perceptron. A deterministic setting is essentially the same as a beam of size 1. The global score as defined by $Score$ is neutralized because at each step, only one candidate is expanded upon, and only one of the next possible candidates is retained. Therefore, the scores of the next possible candidates are all relative to the current single candidate - it is only the scoring of single transitions that determines the outcome of each step, thus any single error in choosing a next candidate necessarily propagates.

In contrast, when maintaining multiple candidates in a beam larger than 1, it is possible for a global score to cause an initially lower scoring candidate in the beam to have its next candidates maintained, while the originally higher scoring candidate is dropped altogether due to the global score of the former being higher than the latter. The global score captures a representation of partial output, in the form of a number that is comparable between competing candidates in the beam.

We employ the early-update variant of the globalized perceptron of Collins and Roark (2004). During the learning process, if the gold-standard sequence (as emitted by the oracle function) drops from the beam before the end of processing, learning ceases, and the weights of the best partial sequence are updated.

function *BEAM-SEARCH(problem, agenda, candidates, B)*
$candidates \leftarrow \{STARTITEM(problem)\}$
$agenda \leftarrow CLEAR(agenda)$
loop do
  for each *candidate in candidates*
    $agenda \leftarrow INSERT(EXPAND(candidate, problem), agenda)$
  $best \leftarrow TOP(agenda)$
  if $GOALTEST(problem, best)$ **then**
    then return *best*
  $candidates \leftarrow TOP\text{-}B(agenda, B)$
  $agenda \leftarrow CLEAR(agenda)$

Algorithm 3: The generic beam-search algorithm

3.4 Research Objectives

In this project we use transition-based parsing methods to develop a joint morpho-syntactic parser. This will require us to answer several open research questions:

- Current transition systems are defined for syntactic analysis. How can they be extended to both morphological and syntactic analysis for MRLs such as Modern Hebrew?
- How can we efficiently search through the space of joint morphological / syntactic analyses?
- How can morphological features be used to help syntactic disambiguation and vice-versa?

Chapter 4

Experimental Setup

> "I think that in the discussion of natural problems we ought to begin not with the Scriptures, but with experiments, and demonstrations. "
>
> *Galileo Galilei*

In this chapter, we describe the data sets for our experiments (4.1), the implementation of our baselines and hypotheses with a new parser (4.2), and the evaluation metrics with which we measure the results of our experiments (4.3).

4.1 Data

4.1.1 Modern Hebrew

As a starting point, we use the Modern Hebrew treebank of the SPMRL (Statistical Parsing of Morphologically Rich Languages) 2014 Shared Task (Seddah, Kübler, and Tsarfaty, 2014), derived from the Unified-SD treebank of Tsarfaty (2013). Bugs and inconsistencies in this initial data set impaired the development and experimentation of our models. We therefore decided to homogenize the treebank and relevant tools.

For the purposes of this work, the treebank was updated with consistent theories for treebank annotation and lexicographic resources (Itai and Wintner, 2008). We add lemmas to the treebank and provide a new morphological analyzer whose output is consistent with all treebank train/dev/test sets. We use the standard split, and train on the standard train set (5,000 sentences).

4.1.2 Universal Dependencies

The Universal Dependencies (UD) project (Nivre, Marneffe, et al., 2016) is a NLP community effort to create a cross-linguistically consistent treebank annotation within a dependency-based framework.

We evaluate the cross-linguistic coverage of our MD model on the UD data set. We parse 48 UD treebanks from the UD 1.3 release (Nivre, Agić, et al., 2016), training on the *train* set and evaluating on *test*.[1]

4.2 Implementation

To test our joint hypothesis we implement *yap*, which stands for ***y**et **a**nother **p**arser*. *yap* is a fully integrated, transition-based, multilingual natural language processor, implemented from scratch.[2]

It has two morphological analyzers: HEBLEX, a lexicon-based morphological analyzer backed by the BGU lexicon, and a multilingual data-driven morphological analyzer which induces a lexicon from any treebank in the Universal Dependencies set of corpora. In *yap* we implement our morphological disambiguator and dependency parser which can run in both pipeline and joint architectures.

yap is written in the Go programming language (`https://golang.org`), an open-source language developed by a team at Google. It is a compiled, statically-typed language similar to Algol and C. Go compilers target a wide variety of platforms. The main compiler, gc, targets Linux, Mac OS/OS X, Windows, numerous BSD and Unix variants, and smartphones. It also has a gcc frontend, gccgo, which is part of the standard gcc distribution.

yap is designed to be as modular as possible with the goal of being an adequate platform for research in NLP. *yap* is currently very slow compared to similar parsers, mostly due to our focus on reproducing previous work, developing the pipeline and joint processors, and experimentation with feature models. Additionally, for some setups, *yap* requires an egregiously large amount of memory, on the order of 100GB. This is due to an inefficient implementation of the feature model, which can be eased with some of engineering effort using common techniques in the literature.

[1] We do not present results for 6 languages: cs,kk,es_ancora,en_esl,pt_br,ja_ktc, as some or all required fields are empty.

[2] We intend to add tokenization in the near future such that yap may be provide an end-to-end, self-contained natural language processing pipeline for Modern Hebrew.

4.3 Evaluation

4.3.1 The Morphological Model

To evaluate a morphological disambiguation, we compare the set of predicted morphemes to the set of gold-standard morphemes.

A morpheme $m = (f, t, g)$ of the spellout of the i-th token of a morphologically disambiguated sentence of n tokens is uniquely identified by the quadruple (f, t, g, i), where $0 \leq i < n$. Let M_p, M_g be sets of uniquely identified predicted and gold-standard morphemes of a sentence, respectively. Define $Pr = \frac{|M_p \cap M_g|}{|M_p|}$ and $Re = \frac{|M_p \cap M_g|}{|M_g|}$ as precision and recall, respectively.

We use the standard $F_1 = \frac{2 * Pr * Re}{Pr + Re}$ measure to evaluate morphological disambiguation performance.

To measure the performance of morphological disambiguation tasks individually, we report scores with partially and fully represented morphemes: for segmentation only (F), $m = (f, \_, \_)$; for segmentation and tagging (F+POS), $m = (f, t, \_)$; and full morphological disambiguation including morphological features (ALL), $m = (f, t, g)$.

For comparison with previous work, we also report token-level accuracy. Let S_p, S_g be the sets of predicted and gold-standard *spellouts* of a sentence of n tokens. Define token-level accuracy as $\frac{|S_p \cap S_g|}{|S_g|}$. While F_1 awards partial success on word-lattices, token-level accuracy requires an exact match of a whole path per token.

4.3.2 The Syntactic Model

To evaluate dependency parsing performance in a standalone setting, with gold morphological disambiguation, we report *attachment scores*.

We define the Unlabeled Attachment Score (UAS) of a predicted dependency tree as the percentage of morphemes that have a correct head, and the Labeled Attachment Score (LAS) as the percentage of morphemes that have a correct head *and* dependency label (Kübler, McDonald, and J. Nivre, 2009, Chapter 6, p. 80).

Let A_p, A_g be the arc sets of predicted and gold-standard labeled dependency trees for a given morphologically disambiguated sentence of n morphemes. We assert the nodes (morphemes) of the trees are the same. The elements of A_p and A_g are triples of the form (i, l, j), where $0 \leq i, j < n$ are indexes of morphemes, and $l \in R$ a dependency label. Then, we define $LAS = \frac{|A_p \cap A_g|}{|A_g|}$.

Let A'_p, A'_g be the *unlabeled* predicted and gold-standard projections of A_p, A_g, respectively, such that the label l of each element in A_p, A_g is discarded. Then, we define $UAS = \frac{|A'_p \cap A'_g|}{|A'_g|}$.

We report these results without punctuation, as this is standard practice in the literature and is necessary for comparison to previous work.

4.3.3 The Joint Model

Comparing morphological disambiguation output and dependency graphs jointly is non-trivial due to the possibly varying number of morphemes as a result of morphological disambiguation. For example, consider two spellouts for the token "BCLM": BCLM (Betzelem — a proper noun — an Israeli organization) and B-CL-FL-HM (in-their-shadow). How might one evaluate the dependency graphs of these spellouts relative to one another?

To address this issue we report both the F_1 scores for full morphological disambiguation and the labeled/unlabeled F_1 measures of graph edges and morpheme forms (taken together).

Let $S = x_1, \ldots, x_k$ be an input sentence for which we want to evaluate a *jointly* predicted morphological disambiguation and dependency graph. Let M_p be the predicted morphological disambiguation of S, and let A_p be the predicted dependency graph of S with the morphemes of M_p as nodes. Likewise, let M_g, A_g be the gold-standard morphological disambiguation and dependency tree of S, respectively.

Since the nodes of the arcs of A_p and A_g are not necessarily the same, we can't compare them directly. To solve this, we resolve the indexes of the arcs in A_p, A_g, and *replace* them with the forms of their respective morphemes in M_p and M_G. Let J_p, J_g be the indexed-resolved arcs of a joint predicted and gold-standard morphological disambiguation and dependency graph of S. We evaluate a joint prediction using the standard $F_1 = \frac{2*Pr*Re}{Pr+Re}$, where $Pr = \frac{|J_p \cap J_g|}{|J_p|}$ and $Re = \frac{|J_p \cap J_g|}{|J_g|}$. We report both labeled and unlabeled F_1 scores.

Chapter 5

Transition-Based Morphological Disambiguation

> "Something is elegant if it is two things at once: unusually simple and surprisingly powerful."
>
> *Matthew E. May*

In this chapter, we formally define two transition systems for morphological disambiguation. We start with a configuration definition (5.1) and a parameterization function (5.2), shared by both transition systems. We then define two transitions systems (5.3), (5.4), and present the results of an empirical comparison (5.5).

Using a lexicon-based morphological analyzer, we apply our best model to the Modern Hebrew treebank, and report our results (5.6).

To present the cross-linguistic applicability of the morphological disambiguator, we use a data-driven morphological analyzer and apply our MD to 48 treebanks of the UD 1.3 corpora (Nivre, Agić, et al., 2016). We report our results (5.7), and for non-MRLs we present a comparison with MarMoT, a state-of-the-art multilingual CRF tagger (Mueller, Schmid, and Schütze, 2013).

5.1 Introduction

A configuration for the MD transition system is a quadruple (L, n, i, M) where $L = MA(x)$ is the sentence-lattice, n is a node in L, i is the 0-based index of a word-lattice in L, and M is a set of disambiguated morphemes (i.e., selected arcs). The

set of terminal configurations is defined to be $C_t = \{(L, top(L), tokens(L), M)\}$ for any L, M, where $tokens(L)$ is the number of word-lattices that form L. The initial configuration function c_s concatenates the L_i lattices of the tokens into a single structure $L = MA(x_1) + ... + MA(x_k)$, and sets $n = bottom(L)$, $i = 0$ and $M = \emptyset$.

There are two conceivable ways to make morphological disambiguation decisions: in a *word-based* (WB), and in a *morpheme-based* (MB), fashion, in the terminology of Tsarfaty and Goldberg (2008). In WB models (a.k.a *token-level* in the UD terminology), the disambiguation decision determines a complete path of morphemes between token-boundaries. In the lattice, this refers to selecting a path between two token-boundary nodes (double circles). MB disambiguation decisions (also termed *lexical-level*, or *word-level* in UD) occur at any node in the lattice indicating a morpheme boundary, with more than one outgoing arc, choosing a specific arc m among them.

WB and MB strategies face contradicting, and complementary, challenges. In WB models, disambiguation decisions are complex, and learning how to score them is expected to suffer from data sparseness. MB models, on the other hand, over-generalize in terms of possible morphological combinations, and learning to score combinations may fail to generalize and be prone to over-fitting. On top of that, morpheme sequences are longer than word sequences, which, in a transition-based system, is known to be more error prone. Finally, variable-length sequences introduce length biases which negatively impact performance. Since MA&D is the base for the NLP pipeline, it is critical to settle this debate empirically and establish the basis for downstream tasks.

5.2 Parameterizing Transitions

A transition system is required to distinguish between all possible decisions it can make at a given point. At the same time, the model should be able to generalize from seen decisions to unseen ones, and effectively learn to disambiguate open-class words and out-of-vocabulary items. To satisfy these desiderata, we define a *delexicalization projection* for a pre-defined set of parts-of-speech tags O capturing open-class categories. Simply put, this projection neutralizes the lattice-node specific indices, and, for any tag $t \in O$, it further neutralizes the lexical form. Formally:

$$DLEX_O(m) = \begin{cases} (\_, \_, \_, t, g) & \text{if } t \in O \\ (\_, \_, f, t, g) & \text{otherwise} \end{cases} \tag{5.1}$$

This $DLEX_O(m)$ projection allows the transition system to distinguish between any two arcs or paths of arcs with the same starting node, while providing an opportunity

to generalize the in-context behavior of similar morphemes with different forms. This is possible due to the reasonable assumption that forms of open-class parts-of-speech in a lattice do not appear twice in a word-lattice with the same morpho-syntactic representation (in the rare cases this happens, the two corresponding arcs are collapsed into one). [1]

5.3 Word-Based Transitions

5.3.1 Transition System

For word-based (WB) modeling, a single transition morphologically disambiguates whole word-lattices such that the node n of a configuration is always at a word boundary (a node that is a bottom, top, or both, of word-lattices of L). We define the transitions in the WB system as an open set of transitions termed MD_s, specifying the parameter s as a single path:

$$MD_s : (L, n, i, M) \rightarrow (L, q, i+1, M \cup \{m_0, .., m_j\}) \tag{5.2}$$

Algorithm 4: The Word-Based Morphological Disambiguation Transition System

Here, $\{m_0, ..., m_j\} \in L$ form a contiguous path of arcs, where m_0 starts at node n, and m_j ends at node q (they can be the same arc), and s is the projected paths $s = DLEX_O(m_0), ..., DLEX_O(m_j)$. A terminal configuration will therefore contain the union of contiguous paths of word-lattices in L, together forming a complete morphological disambiguation of the ambiguous tokens of the input sentence.

5.3.2 Learning

We define three types of word-lattice properties: o - the surface form of the token itself, a - the $DLEX$-projected lattice (all MSRs projected by the delexicalization function), and p - a chosen disambiguated path, which only exists for previously processed lattices. Using these properties, we define baseline feature templates modeled after POS tagging: unigram, bigram, and trigram combinations of o and a, and p-based features, which predict the next disambiguation decision based on the previous one(s). See Appendix A for the full word-based feature model.

[1] There are extremely rare cases in Modern Hebrew that violate this assumption, but we can choose alternative linguistic interpretations that transforms these cases into distinguishable representations.

5.4 Morpheme-Based Transitions

5.4.1 Transition System

For morpheme-based (MB) modeling, a single transition chooses an outgoing arc of the current node n in the lattice, requiring a disambiguation decision if (and only if) there is more than one outgoing arc. Again, we define the transitions of the MB transition system as an open set of transitions termed MD_s, now specifying s as a single *arc*: Here, m is a morpheme $(n, q, f, t, g) \in L$, and $s = DLEX_O(m)$. If node q is at a word

$$MD_s : (L, n, i, M) \rightarrow (L, q, j, M \cup \{m\}) \tag{5.3}$$

Algorithm 5: The Morpheme-Based Morphological Disambiguation Transition System

boundary, then $j = i + 1$, otherwise $j = i$. For a terminal configuration, each $m \in M$ is an outgoing arc of the end node of another arc in M (with the exception of the first morpheme, starting at $bottom(L)$) forming a contiguous path that disambiguates x.

5.4.2 Learning

In the MB model we can access specific information concerning the current node inside the word-lattice. We define the properties f, t and g, corresponding to arcs' form, part-of-speech and morphological attribute:value pairs. We use these properties in various unigram, bigram, and trigram combinations, in parallel with the WB model.

We provide properties similar to those of MarMoT (Müller, Schmid, and Schütze, 2013), but we do not employ a hash kernel. We define the prefix/suffix properties e/x, respectively, which generate a feature for each character of lengths 1 to 10 at the start/end of a *token*. Additionally, we define the signature feature g, a set of indicator bits defined in the morpheme-based MD feature appendix (B).[2]

As in the WB model we also define the property p as the path in the previously disambiguated word-lattices. We define the property n to be the set of $DLEX$-projected outgoing morphemes of the current node (this parallels the property a of WB models, but at morpheme granularity). Similarly to the WB case, we use unigram, bigram, and trigram combinations of these properties as well. See Appendix B for the full morpheme-based feature model.

[2]These properties are not applicable to the WB model due to extreme sparsity

5.4.3 Decoding

Since the number of arcs in lattices' paths for x may vary, so do the number of transitions in our morpheme-based transition system; for example, the phrase "BCLM HNEIM" can have disambiguations with 2 to 6 morphemes. This violates a basic assumption of standard beam search decoding — that the number of transitions is a deterministic function of the input.

There are two inherent biases in varied-length transition sequences driven by the general perceptron algorithm. The beam search algorithm tests the best candidate after each step for goal fulfillment. A short sequence may temporarily be the best candidate and fulfill the goal, while longer (and possibly correct) sequences are incomplete and may be lost. This can be easily mitigated by verifying that all beam candidates fulfill the goal before returning the best one.

On the other hand, long sequences have more features, therefore their score may be arbitrarily inflated. So the score may be higher for longer paths, even though a shorter one may be correct and may fall off the beam.

To address these challenges we introduce a special transition we call *ENDTOKEN (ET)*, that explicitly increments i, instead of implicitly in MD_s. So, in equation (4) we set $j = i$ and apply:

$$ET : (L, n, i, M) \rightarrow (L, n, i + 1, M) \tag{5.4}$$

ET is required to occur exactly once at the end of a word-lattice, when n is the top of some word-lattice in L. Set aside from other transitions, ET has its own set of features. Other than incrementing i, ET has no effect on configurations, but it does cause a re-ordering of candidates in the beam during decoding, at each token boundary. Note that ET kicks in only for variable length lattices. On same-length lattices, ET is skipped and equation (4) remains as is — the process essentially falls back on the standard, same-length decoding.

An MD transition sequence thus becomes the union of disjoint sets of configurations $y = y_{md} \cup y_{et}$, and changes $Score$ in Equation (3.2), where $|y_{et}|$ is the # of tokens in L with variable length paths. :

$$\sum_{i=1}^{d} \omega_i^{md} \phi_i^{md}(y_{md}) + \sum_{j=1}^{d} \omega_j^{et} \phi_j^{et}(y_{et}) = \sum_{c_k \in y_{md}} \sum_{i=1}^{d} \omega_i^{md} \phi_i^{md}(c_k) + \sum_{c_l \in y_{et}} \sum_{j=1}^{d'} \omega_j^{et} \phi_j^{et}(c_l) \tag{5.5}$$

While the number of morphemes, and therefore $|y_{md}|$, can vary, $|y_{et}|$ is deterministic per lattice — the number of ET transitions is the number of word-lattices with spellouts of more than one length (variable length). Using this anchor, the features of the ET

transition provide a counter-balance to the effects of varied-length sequences by scoring fully disambiguated paths of each word-lattice individually, occuring a fixed amount of times for all paths.

5.4.4 ENDTOKEN vs IDLE

Variable-length sequences in beam search also exist in the structured prediction of constituency trees. Zhu et al. (2013) introduced an IDLE transition (also adopted in Honnibal and Johnson (2014) and Meishan Zhang et al. (2014a)) that, like ET, has no effect on the configuration, but unlike ET, occurs only at the end of the parsing sequence, an arbitrary number of times, until all parsing sequences are complete. While IDLE transitions make sense when applied after a complete hierarchical structure is predicted — where they may learn to rerank candidates based on features that are visible at the top of the structure (the root) — it is futile to use last-seen features that arbitrarily exist at the end of a morphological disambiguation (linear) sequence, to rerank candidates again and again. This is because at the end of the sequence, we can no longer save candidates that were lost earlier on due to length discrepancies.

To mitigate this, one might try to create IDLE padding with global features spanning the entire disambiguated path. Even then, the learned model parameters would not generalize well, since these features will be applied an arbitrary number of times — the maximal length of an occasional word-lattice we are at — which has no linguistic significance, and may arbitrarily inflate certain scores. ET transitions, in contrast, occur right when they are needed — at the boundary of a word-lattice, following a disambiguated token. This position enables the reordering of candidates right after a length discrepancy may have been introduced. Moreover, ET scores are counted against the global score a fixed number of times per lattice, for all, and any length, candidates. This enables a fair comparison of all paths per lattice.

5.5 Empirical Comparison

We empirically evaluate the proposed models, and investigate their strengths, weaknesses, and bounds.

We start with a detailed investigation of MA&D in the Semitic language Modern Hebrew, which is known for its rich morphology and significant ambiguity. We then verify the cross-linguistic coverage of the models on the set of UD treebanks (Nivre, Agić, et al., 2016), to validate their efficacy in MRL and non-MRL settings. Here we provide results and in-depth analysis on *dev* and confirm our findings on *test*.

(a)

	Word-Based				
	unigram	+bigram	+trigram	+next unigram	+next with bigram
	85.73 (86.72)	86.9 (87.88)	86.7 (87.66)	**92.19 (92.76)**	91.98 (92.59)
+ET	89.09 (89.81)	89.93 (90.71)	90 (90.85)	**93.39 (93.94)**	92.84 (93.46)

(b)

	Morpheme-Based				
	unigram	+bigram	+trigram	+next unigram	+next with bigram
	90.67 (91.41)	91.12 (91.88)	90.09 (91.82)	93.56 (94.16)	**93.89 (94.49)**
+ET	92.68 (93.36)	92.74 (93.55)	92.64 (93.47)	94.27 (94.92)	**94.33 (94.9)**

TABLE 5.1: Dev. set results for Word-Based (a) and Morpheme-based (b) MD: F_1 for full morphological disambiguation (form, part of speech, morphological properties). (n parenthesis: F_1 for form and POS only. The +ET lines indicate a variant that employs the ENDTOKEN transition at token boundaries.

A pre-condition for the execution of our MD models is an $MA(x)$ function that generates word-lattices for x. We start off with a morphological analyzer that we implemented, called HEBLEX, which relies on the Ben-Gurion Hebrew Lexicon used by Adler and Elhadad (2006). The lexicon contains full analyses for 567,483 words and 102 prefixes. HEBLEX uses the lexicon to determine the various combinations of prefixes and words that form valid tokens. This process is far from trivial due to morphological *fusion*, as some morphemes are implicit.

Although the lexicon is quite large, there are still tokens which are *out-of-vocabulary (OOV)*. OOV tokens may be of two types: it may be that an entire string is out of the lexicon (mostly proper nouns) or that the affixes and the open class items are seen, but their combination has not yet been encountered. We address OOV by assigning proper noun analyses to entire tokens, as well as to all arcs combined with seen affixes. This adds ambiguity to the lattices, but gives the MD the chance to select a correct path. In our experiments we aim to quantify exactly the effect of lexical coverage of the MA on MD accuracy. To this end, we add an option to *infuse* missing gold analyses into the MA lattices provided by HEBLEX and present two sets of results: once disambiguating lattices with infused gold analyses (ideal MA), and once without infusing gold analyses (realistic MA).

5.6 Results for Modern Hebrew

Tables (5.1a), (5.1b) present our investigation of the WB and MB models on the dev test, respectively, with different feature templates.

Our results show that the MB disambiguation consistently outperforms our WB variant, in the various feature template settings. Moreover, the ET transition consistently improves performance, with best results for Hebrew at F_1 scores of 94.3 (94.9) for full MD (F+POS only). The token-level accuracy for our best results are 93.07 (93.9) for

full MD (F+POS). These results were obtained on *infused* lattices, that include the gold path as one of the alternatives. In order to gauge the effect of incomplete lexical coverage, we disable infusion of the gold analyses into the HEBLEX lattices. We then observe a drop to F_1 scores of 89.62 (92.06) and token accuracy of 87.72(90.85). To set our results in context, we applied our best model in "English-like" settings for tagging, with gold pre-segmented text. F_1 then increases to 96.82 (97.44). That is, in "English-like" settings, our tagging accuracy (97.44) is as high as state-of-the-art English tagging (Manning, 2011). MarMoT (Müller, Schmid, and Schütze, 2013), a state of the art CRF tagger, obtains F_1 93.38 on full MD on this set. Next we aim to verify that ET transitions indeed act as intended.

We classify a sequence length error as either an overshoot (predicted morphological disambiguation sequence is longer than gold) or undershoot (predicted shorter than gold). Without ET, in the infused setting, 36.8% of sentences have incorrect length and the overshoot:undershoot ratio is 6.6:1. Adding ET transitions results in 31.8% length errors, correcting 20.62% of the overshoot errors, resulting in ratio of 4:1.

In the un-infused setting, 41.4% of the sentences have incorrect length with a ratio of 6.39:1. Adding ET results in 36% length errors, correcting 20.67% of the overshoot errors, resulting in ratio of 3.7:1. Hebrew previous results are non-trivial to compare to due to significant changes of the treebank along the way and unavailable code of previous work. The most relevant results to ours are by Adler (2007), who reports state-of-the-art results for Modern Hebrew in the realistic (non-infused) setting, with self-reported token accuracy of 90% (93%) on a different evaluation set. For his prediction on our dev set, F_1 evaluation yields 85.74 (87.95), much lower than ours. Segmentation F_1 for Adler is 96.35, while ours is 97.6. We confirm our findings on the test set, for which Adler F_1 yields 82.91 (85.56). Our best model now yields 86.23 (88.85) and 92.9 (93.73) in realistic and infused settings, respectively.

5.7 Results for Universal Dependencies

The MD requires a lattice structure to describe the ambiguous MSRs of input tokens. The input, as supplied by the UD data set, are tokens with unambiguous gold or predicted MSRs. To generate the required input lattices for MD, we implement a data-driven Morphological Analyzer (MA), that can be trained out-of-the-box along with our MD models for any treebank in the CoNLL-U format.

We generate a dictionary for each language from its train set by collecting all seen analyses of each token in the training data, where an analysis is composed of MSRs that

contain a lemma, POS, and the full set of morphological features. The dictionary maps each token to a set of MSR sequences, which then compose their ambiguous MA lattices.

For out-of-vocabulary (OOV) tokens, the MA pre-computes the cardinality of each coarse POS — the number of unique tokens per coarse POS — and consider the top 5 POS as "open-class". For these top 5 POS, the MA computes the 50 highest-frequency MSRs (POS + morph. properties) to be used as the OOV lattice of an OOV token.

When applying MA to the training set, we add the OOV lattice to tokens whose known analysis contains an open-class POS. The model thus encounters during training a larger space of states than the observed one, and learns to accurately apply transitions in OOV circumstances at test time.

Lang.	sents	5k Sent. Trainset			Full Trainset		
	words	inf.	no inf.	MM	inf.	no inf.	MM
bg	8907	0.933	0.914 (0.959)	0.937 (0.964)	0.946	0.926	0.948
	124474	0.969	0.96 (0.983)	0.976 (0.987)	0.979	0.969	0.982
cu	5077	0.901	0.853 (0.911)	0.893 (0.927)	0.908	0.851	0.897
	46025	0.964	0.931 (0.977)	0.959 (0.978)	0.965	0.929	0.962
da	4868	0.932	0.88 (0.944)	0.943 (0.966)	0.934	0.894	0.943
	88979	0.953	0.91 (0.954)	0.961 (0.972)	0.954	0.922	0.961
el	1929	0.912	0.876 (0.918)	0.921 (0.944)	0.914	0.876	0.921
	47449	0.977	0.965 (0.989)	0.979 (0.99)	0.978	0.965	0.979
en	12543	0.904	0.887 (0.927)	0.904 (0.937)	0.927	0.914	0.93
	204586	0.923	0.911 (0.951)	0.923 (0.954)	0.942	0.932	0.945
en_	3650	0.95	0.944 (0.955)	0.96 (0.969)	0.95	0.944	0.96
lines	66374	0.95	0.944 (0.955)	0.96 (0.969)	0.95	0.944	0.96
et	14510	0.866	0.821 (0.911)	0.883 (0.933)	0.913	0.884	0.928
	187814	0.923	0.895 (0.953)	0.933 (0.961)	0.951	0.932	0.959
eu	5396	0.877	0.79 (0.895)	0.874 (0.93)	0.88	0.797	0.877
	72974	0.948	0.904 (0.964)	0.944 (0.971)	0.948	0.908	0.946
fi	12217	0.884	0.756 (0.942)	0.889 (0.961)	0.925	0.864	0.932
	162721	0.915	0.863 (0.96)	0.928 (0.973)	0.95	0.909	0.958
ga	720	0.781	0.743 (0.87)	0.819 (0.909)	0.791	0.747	0.819
	16701	0.896	0.873 (0.946)	0.924 (0.961)	0.9	0.882	0.924
gl	2276	0.971	0.968 (0.985)	0.971 (0.984)	0.971	0.968	0.971
	79329	0.971	0.968 (0.985)	0.971 (0.984)	0.971	0.968	0.971
got	4360	0.874	0.824 (0.873)	0.877 (0.901)	0.871	0.83	0.877
	44722	0.955	0.927 (0.964)	0.961 (0.972)	0.953	0.924	0.961
grc	13185	0.862	0.762 (0.867)	0.844 (0.883)	0.862	0.776	0.859
	196083	0.926	0.853 (0.937)	0.935 (0.946)	0.92	0.861	0.941
grc_	13306	0.829	0.719 (0.89)	0.774 (0.887)	0.909	0.853	0.9
proiel	166061	0.911	0.827 (0.972)	0.893 (0.966)	0.973	0.939	0.971
hi	13304	0.82	0.799 (0.823)	0.867 (0.891)	0.846	0.832	0.891
	281057	0.944	0.934 (0.955)	0.947 (0.964)	0.953	0.948	0.963
hr	3557	0.86	0.812 (0.877)	0.87 (0.913)	0.86	0.813	0.87
	78817	0.951	0.936 (0.974)	0.954 (0.973)	0.951	0.933	0.954
hu	1433	0.763	0.701 (0.831)	0.751 (0.862)	0.758	0.697	0.751
	33016	0.92	0.895 (0.951)	0.945 (0.973)	0.915	0.896	0.945
id	4477	0.932	0.926 (0.934)	0.937 (0.945)	0.932	0.926	0.937
	97531	0.932	0.926 (0.934)	0.937 (0.945)	0.932	0.926	0.937

TABLE 5.2: MA&D for non-MRLs : F_1 scores of the languages in UD that do not require morphological segmentation, the upper line indicates full MD, the lower line indicates segmentation and POS only.

Lang.	sents	5k Sent. Trainset			Full Trainset		
	words	inf.	no inf.	MM	inf.	no inf.	MM
la	2660	0.797	0.696 (0.814)	0.797 (0.876)	0.787	0.709	0.797
	37819	0.904	0.847 (0.927)	0.935 (0.967)	0.897	0.854	0.935
la_	16258	0.916	0.895 (0.922)	0.922 (0.938)	0.935	0.914	0.945
ittb	276941	0.978	0.965 (0.985)	0.979 (0.986)	0.986	0.977	0.988
la_	11986	0.857	0.773 (0.859)	0.841 (0.886)	0.892	0.843	0.892
proiel	132376	0.949	0.896 (0.959)	0.943 (0.967)	0.969	0.943	0.97
lv	673	0.797	0.745 (0.927)	0.817 (0.931)	0.801	0.739	0.817
	12629	0.876	0.841 (0.971)	0.898 (0.971)	0.88	0.835	0.898
nl	13000	0.836	0.797 (0.889)	0.823 (0.911)	0.88	0.863	0.872
	197134	0.866	0.841 (0.923)	0.861 (0.928)	0.902	0.891	0.897
nl_	6641	0.939	0.934 (0.967)	0.95 (0.971)	0.943	0.94	0.952
lassysmall	88929	0.954	0.95 (0.973)	0.961 (0.974)	0.956	0.953	0.963
no	15696	0.926	0.887 (0.952)	0.93 (0.963)	0.947	0.917	0.952
	244776	0.959	0.944 (0.97)	0.964 (0.976)	0.971	0.958	0.975
pl	6800	0.858	0.772 (0.882)	0.854 (0.916)	0.87	0.778	0.866
	69499	0.955	0.915 (0.976)	0.961 (0.978)	0.961	0.922	0.967
pt	8800	0.913	0.881 (0.919)	0.916 (0.938)	0.927	0.906	0.93
	214812	0.96	0.947 (0.965)	0.967 (0.977)	0.967	0.958	0.972
ro	4759	0.915	0.894 (0.917)	0.927 (0.939)	0.916	0.895	0.927
	108618	0.962	0.946 (0.967)	0.962 (0.973)	0.961	0.947	0.962
ru	4029	0.869	0.791 (0.895)	0.861 (0.925)	0.872	0.791	0.861
	79772	0.957	0.925 (0.976)	0.956 (0.979)	0.956	0.925	0.956
ru_	46750	0.896	0.818 (0.919)	0.89 (0.944)	n/a	n/a	n/a
syntagrus	815485	0.965	0.935 (0.977)	0.968 (0.98)	n/a	n/a	n/a
sl	6471	0.874	0.791 (0.876)	0.884 (0.927)	0.882	0.804	0.892
	112334	0.954	0.923 (0.967)	0.963 (0.976)	0.958	0.927	0.967
sl_	2472	0.83	0.796 (0.86)	0.857 (0.902)	0.832	0.793	0.857
sst	23575	0.896	0.88 (0.92)	0.919 (0.947)	0.895	0.878	0.919
sv	4303	0.928	0.921 (0.951)	0.946 (0.966)	0.931	0.921	0.946
	66645	0.952	0.947 (0.965)	0.966 (0.975)	0.954	0.947	0.966
sv_	3650	0.955	0.952 (0.964)	0.957 (0.966)	0.955	0.952	0.957
lines	63949	0.955	0.952 (0.964)	0.957 (0.966)	0.955	0.952	0.957
zh	3997	0.903	0.889 (0.918)	0.913 (0.932)	0.903	0.889	0.913
	98608	0.914	0.903 (0.933)	0.923 (0.943)	0.914	0.903	0.923

TABLE 5.3: MA&D for non-MRLs : F_1 scores of the languages in UD that do not require morphological segmentation, the upper line indicates full MD, the lower line indicates segmentation and POS only.

Lang.	sents	Gold Segmented 5k Train. Set (non-OOV accuracy)					Gold Segmented Full Trainset					Un-Segmented Full Trainset			
	words	inf.	+ET	no inf.	+ET	MM	inf.	+ET	no inf.	+ET	MM	inf.	+ET	no inf.	+ET
ar	6174	0.887	0.887	0.853	0.853 (0.87)	0.903 (0.924)	0.892	0.892	0.86	0.86	0.907	0.867	0.871	0.8	0.799
	225853	0.956	0.956	0.948	0.948 (0.956)	0.956 (0.972)	0.959	0.959	0.951	0.951	0.959	0.929	0.933	0.882	0.882
ca	13123	0.953	0.953	0.919	0.919 (0.958)	0.957 (0.965)	0.963	0.963	0.939	0.939	0.968	0.961	0.961	0.938	0.937
	429157	0.969	0.969	0.936	0.936 (0.972)	0.973 (0.977)	0.977	0.977	0.954	0.954	0.98	0.975	0.975	0.952	0.951
cs_	23478	0.865	0.857	0.758	0.758 (0.862)	0.86 (0.921)	0.901	0.901	0.831	0.831	0.904	0.897	0.899	0.827	0.827
cac	472608	0.973	0.969	0.93	0.928 (0.984)	0.975 (0.988)	0.983	0.983	0.961	0.961	0.987	0.983	0.983	0.961	0.961
cs_	860	0.845	0.844	0.812	0.801 (0.871)	0.887 (0.922)	0.848	0.847	0.816	0.812	0.887	0.832	0.822	0.804	0.8
cltt	26234	0.956	0.959	0.942	0.938 (0.988)	0.98 (0.991)	0.958	0.957	0.94	0.942	0.98	0.953	0.951	0.937	0.938
de	14118	0.928	0.928	0.921	0.921 (0.936)	0.927 (0.941)	0.929	0.929	0.921	0.921	0.927	0.93	0.928	0.921	0.92
	269626	0.928	0.928	0.921	0.921 (0.936)	0.927 (0.941)	0.929	0.929	0.921	0.921	0.927	0.93	0.928	0.921	0.92
es	14187	0.929	0.928	0.888	0.888 (0.943)	0.927 (0.956)	0.939	0.939	0.908	0.908	0.936	0.93	0.933	0.9	0.903
	382436	0.947	0.947	0.931	0.931 (0.959)	0.945 (0.967)	0.955	0.955	0.943	0.943	0.953	0.948	0.951	0.935	0.938
fa	4798	0.953	0.961	0.958	0.958 (0.972)	0.963 (0.978)	0.954	0.953	0.956	0.957	0.963	0.957	0.956	0.947	0.949
	121020	0.96	0.968	0.964	0.964 (0.974)	0.969 (0.98)	0.96	0.959	0.962	0.963	0.969	0.962	0.962	0.954	0.955
fi_	14981	0.876	0.88	0.794	0.793 (0.921)	0.858 (0.944)	0.856	0.847	0.811	0.809	0.915	0.916	0.929	0.814	0.856
ftb	127602	0.914	0.917	0.856	0.855 (0.953)	0.904 (0.957)	0.883	0.869	0.843	0.844	0.943	0.939	0.95	0.849	0.899
fr	14554	0.931	0.93	0.921	0.918 (0.935)	0.939 (0.954)	0.943	0.951	0.925	0.925	0.949	0.944	0.942	0.92	0.921
	356216	0.949	0.947	0.941	0.939 (0.952)	0.955 (0.967)	0.959	0.965	0.942	0.942	0.964	0.958	0.957	0.937	0.938
he	5241	0.934	0.933	0.888	0.888 (0.907)	0.921 (0.953)	0.934	0.93	0.891	0.886	0.922	0.914	0.917	0.724	0.724
	135496	0.968	0.968	0.937	0.938 (0.947)	0.955 (0.974)	0.967	0.966	0.939	0.937	0.957	0.945	0.947	0.768	0.769
it	11699	0.945	0.946	0.91	0.911 (0.953)	0.943 (0.962)	0.96	0.958	0.945	0.945	0.97	0.953	0.957	0.935	0.937
	249330	0.959	0.96	0.924	0.925 (0.961)	0.958 (0.972)	0.97	0.967	0.956	0.955	0.978	0.961	0.965	0.946	0.947
ta	400	0.761	0.793	0.761	0.732 (0.912)	0.82 (0.929)	0.794	0.797	0.757	0.756	0.82	0.72	0.722	0.659	0.664
	6329	0.835	0.859	0.825	0.813 (0.927)	0.877 (0.938)	0.856	0.861	0.827	0.821	0.877	0.765	0.772	0.714	0.717
tr	3947	0.781	0.873	0.765	0.806 (0.935)	0.855 (0.933)	0.794	0.787	0.768	0.805	0.855	0.84	0.781	0.742	0.78
	40609	0.884	0.938	0.87	0.897 (0.968)	0.934 (0.964)	0.893	0.885	0.879	0.899	0.934	0.907	0.87	0.846	0.871

TABLE 5.4: MA&D in MRLs: F_1 scores of the languages in UD which require morphological segmentation. The upper line indicate full MD, the lower line indicates segmentation and POS only. The left hand side shows results for GOLD segmentation, the right hand side for input lattices.

Tables 5.2 and 5.3 report F_1 accuracy for full MD and seg/POS for UD languages that do not require segmentation. For most large train sets ($> 200k$ tokens), we observe 2-3 points absolute drop from infused (ideal) to uninfused (realistic) setting. This suggests that when large train sets exist, our data-driven MA is a viable economic alternative to costly hand-crafted monolingual lexical resources.

To scrutinize our realistic results we also report non-OOV-only F_1 for 5k limited uninfused setting. Here we see that for 80% of languages, our results are on a par with a state-of-the-art tagger, MarMot (Müller, Schmid, and Schütze, 2013), retrained on these data, within 0.035 (or less) points gap. This demonstrates that our disambiguation capacity is on-par with MarMot, where performance gaps come mostly from our OOV strategy (which is intentionally restrained, to allow handling of MRL segmentation that is handled by MarMot).

Table 5.4 shows results for UD MRLs that require segmentation, contrasting results on gold pre-segmented input and un-segmented raw data. For raw data we see a minor, 0.02, drop in F_1, compared to gold-segmented settings. That is, our model still retains the competitive MA&D performance, in a *single, universal, trainable* model — we attribute this to our *joint* segmentation and tagging strategy, which overcomes error propagation.

Shortly before submitting this thesis, UDPipe (Straka, Hajic, and Straková, 2016), a tool for tokenization, morphological analysis, tagging and parsing, had been released. As its name suggests, UDPipe is a pipeline implementation packaging together separate tools for different tasks. Our approaches and ultimate goals are rather different. We present *joint* morphological segmentation and tagging, as opposed to a pipeline. Moreover, our framework can be extended into a *single* transition-based system performing all tasks jointly, overcoming overheads and error propagation, as we intend to address next.

5.8 Discussion

In this chapter, we defined two transitions systems for morphological disambiguation: word-based (5.3) and morpheme-based (5.4). For the morpheme-based transition, we introduce a novel mitigation of the variable-length transition sequence problem (ET), and argue its advantage over the commonly used variable-length solution, $IDLE$, for our use-case (5.4.4).

We report the results of an empirical comparison (5.5), and conclude that the morpheme-based transition-system out performs the word-based transition-system.

We evaluate the performance of our best model against the Modern Hebrew treebank, in ideal (infused) and realistic (uninfused) settings. We present state-of-the-art results in both settings (5.6).

To showcase the cross-linguistic applicability of our model, we present results for 48 treebanks of the UD 1.3 corpora (5.7).

We present an in-depth discussion of our results, conclusions, and new research questions for future work in Discussion and Conclusion (8.1.1).

Chapter 6

Transition-Based Dependency Parsing

> "Never mistake a clear view for a short distance."
>
> *Paul Saffo*

6.1 Introduction

We first define a general configuration for dependency parsing, followed by major variants of transition systems for dependency parsing found in the literature and in practice. Transition systems designed for dependency parsing are also called Arc Systems (as they aspire to generate the arcs of a dependency tree); we shall use these terms interchangeably.

Originally, transition systems for dependency parsing were defined for non-MRL languages. In non-MRLs, sentences are sequences of words that also serve as the nodes of their respective dependency trees. In contrast, sentences in MRLs are composed of tokens while their respective dependency trees have morphemes for nodes. We describe transition systems given a generic sequence of morphemes rather than words; the reader shall accommodate the discrepancy by considering a word of a non-MRL to be a morpheme.

Let R be a set of dependency types and $S = m_1...m_n$ be a sentence of n morphemes. We denote a dependency graph of S as $G_S = (V_S, A_S)$, where V_S is a set of elements corresponding to the morphemes of S, and $A_S \subseteq V_S \times R \times V_S$ as a set of labeled arcs between the elements of V_S.

Define a configuration of an arc system for a sentence $S = m_1...m_n$ of n morphemes as a triple $c = (\sigma, \beta, A)$, where:

- σ is a stack of morphemes $m_i \in V_S$
- β is a buffer of morphemes $m_i \in V_S$
- A is a set of labeled dependency arcs $(m_i, r, m_j) \in V_S \times R \times V_S$

A configuration represents a partial analysis of the input sentence, where the morphemes on the stack S are partially processed morphemes, the morphemes in the buffer β are the remaining morphemes, and the arc set A represents a partially built dependency tree (Kübler, McDonald, and J. Nivre, 2009, Chapter 3).

There are two aspects for subtle variations on a given arc system: the existence of a root node and projectivity.

A transition system can introduce an artificial so-called "root" node - a morpheme that does not exist in the input sentence made available as the head of any morpheme (and therefore, any partial tree) in the sentence. A root node allows for multiple partial dependency trees (forest) of an input sentence to be unrelated, except through the root node. This facilitates grammatically incorrect sentences or pathological cases. We call transition systems with and without a root node root-full and root-less, respectively.

A projective arc system can only output projective dependency graphs; it is guaranteed to fail given a non-projective input sentence because no transition sequence exists such that the resulting set of arcs represents a non-projective dependency graph. Likewise, a non-projective arc system can result in both projective and non-projective dependency graphs. Although non-projective arc systems exist in the literature, we limit our scope to that of projective arc systems. However, there are no inhibiting properties of our work preventing the applicability to non-projective arc systems.

Unless otherwise specified, the set of terminal configurations is $C_t = \{(\sigma, []_\beta, A)\}$ for any A and $|\sigma| = 1$.[1]

6.2 Arc Standard

Given an input sentence $S = m_1...m_n$, the Arc Standard root-full transition system defines the initial configuration function as $c_0(S) = ([m_0]_\sigma, [m_1, ..., .m_n]_\beta, \emptyset)$, where m_0 is the root node. For the root-less variation, let $\sigma = []$. This transition system's name

[1] In the literature, $\sigma = [m_0]$ is the formal requirement for root-full variations, however $|\sigma| = 1$ is a generalization that applies to both root-full and root-less variations.

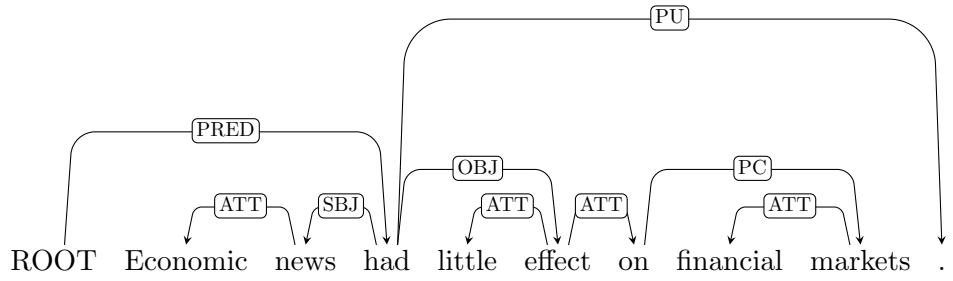

FIGURE 6.1: A dependency tree for the sentence "Economic news had little effect on financial markets."

alludes to it's straightforward, standard method of bottom-up left-to-right incremental parsing (J. Nivre, 2004).

We use the formal definition of the Arc Standard transition system by Kübler, McDonald, and J. Nivre (2009):

Data:	$c_s(x = x_1, ..., x_n) = ([0], [1, ..., n], \emptyset)$	
Terminal:	$C_t = \{c \in C \mid c = ([0], [], A)\}$	
Transitions:	$(\sigma, [i\|\beta], A) \rightarrow ([\sigma\|i], \beta, A)$	$(SHIFT)$
	$([\sigma\|i], [j\|\beta], A) \rightarrow (\sigma, [j\|\beta], A \cup \{(j, l, i)\})$ if $i \neq 0$	$(ArcLeft_l)$
	$([\sigma\|i], [j\|\beta], A) \rightarrow (\sigma, [i\|\beta], A \cup \{(i, l, j)\})$	$(ArcRight_l)$

Algorithm 6: The arc-standard transition system for dependency parsing (Kübler, McDonald, and J. Nivre, 2009, Chapter 3)

The correct (gold) transition sequence for a transition system, given an input sentence and it's respective dependency graph, is produced by an oracle function. The gold transition sequence is used in training by comparing it to the predicted sequence, and may be referenced by the parametric model to learn correct transitions.

In table 6.1 an example of a correct transition sequence with the root-full variant is shown for the English sentence "Economic news had little effect on financial markets.", resulting in its dependency tree as in figure 6.1.

Formally, the oracle is a function that is given a configuration and a reference dependency graph $G_d = (V_d, A_d)$, and it returns a transition. The oracle for Arc Standard is as follows:

$$o(c = (\sigma, \beta, A)) = \begin{cases} ArcLeft_l & \text{if } (\beta[0], l, \sigma[0]) \in A_d \\ ArcRight_l & \text{if } (\sigma[0], l, \beta[0]) \in A_d \text{ and } \forall m, l' \\ & \text{if } (\beta[0], l', m) \in A_d \text{ then } (\beta[0], l', m) \in A \\ SHIFT & \text{otherwise} \end{cases} \tag{6.1}$$

Algorithm 7: The oracle function for the Arc Standard Transition System for Dependency Parsing (Kübler, McDonald, and J. Nivre, 2009, Chapter 3)

Transition	Stack σ	Buffer β	Arcs A
$c_0 =$	([R],	[Economic,...,.],	$\emptyset$)
$SH \rightarrow$	([R,Economic],	[news,...,.],	$\emptyset$)
$LA_{ATT} \rightarrow$	([R],	[news,...,.],	A_1 ={(news,ATT,Economic)})
$SH \rightarrow$	([R,news],	[had,...,.],	A_1)
$LA_{SBJ} \rightarrow$	([R],	[had,...,.],	$A_2 = A_1 \cup$ {(had,SBJ,news)})
$SH \rightarrow$	([R,had],	[little,...,.],	A_2)
$SH \rightarrow$	([R,had,little],	[effect,...,.],	A_2)
$LA_{ATT} \rightarrow$	([R,had],	[effect,...,.],	$A_3 = A_2 \cup$ {(effect,ATT,little)})
$SH \rightarrow$	([R,had,effect],	[on,...,.],	A_3)
$SH \rightarrow$	([R,...,on],	[financial,...,.],	A_3)
$SH \rightarrow$	([R,...,financial],	[markets,.],	A_3)
$LA_{ATT} \rightarrow$	([R,...,on],	[markets,.],	$A_4 = A_3 \cup$ {(markets,ATT,financial)})
$RA_{PC} \rightarrow$	([R,had,effect],	[on,.],	$A_5 = A_4 \cup$ {(on,PC,markets)})
$RA_{ATT} \rightarrow$	([R,had],	[effect,.],	$A_6 = A_5 \cup$ {(effect,ATT,on)})
$RA_{OBJ} \rightarrow$	([R],	[had,.],	$A_7 = A_6 \cup$ {(had,OBJ,effect)})
$SH \rightarrow$	([R, had],	[.],	A_7)
$RA_{PU} \rightarrow$	([R],	[had],	$A_8 = A_7 \cup$ {(had,PU,.)})
$RA_{PRED} \rightarrow$	([],	[R],	$A_9 = A_8 \cup$ {(R,PRED,had)})
$SH \rightarrow$	([R],	[],	A_9)

TABLE 6.1: Example Parsing Sequence with Arc Standard; SH - SHIFT, RA/LA - ArcRight/ArcLeft, R - ROOT (Kübler, McDonald, and J. Nivre, 2009, Chapter 3)

6.3 Arc Eager

In Arc Standard, right dependents can't be attached to their head (i.e. RightArc-ed) until all their dependents have been attached. This constraint is due to dependents of right arcs (head being on the "left" of the dependent) being removed from the buffer as a result of a RightArc transition. Therefore, if a RightArc transition is applied prematurely to a node n, it follows that an eventual error in the transition sequence will arise as an incorrect attachment of a true dependent of n to some incorrect node, $m \neq n$. The parametric model predicting the next transition must take this into account, required to foresee that a RightArc should be delayed due to possible dependents appearing after it in the buffer β.

The RightArc constraint in Arc Standard has the additional effect that even though a RightArc is certain, this information is never stored in the configuration until the RightArc is created. In table 6.1, observe that at the fifth transition, even though "had" is certainly a dependent of the root, it is shifted instead.

Therefore, in following transitions, the parametric model does not have access to an indication that ROOT will be the head of "had", eventually. This limits the depth of inquiry available to the parametric model - perhaps it could better predict a subsequent

transition knowing that "had"s head is the ROOT? This line of question begs a higher-level question — perhaps taking the incremental approach of Arc Standard further, we need to parse bottom-up for left-dependents but top-down for right-dependents (J. Nivre, 2004)?

Data:	$c_s(x = x_1, ..., x_n) = ([0], [1, ..., n], \emptyset)$	
Terminal:	$C_t = \{c \in C \mid c = ([0], [], A)\}$	
Transitions:	$(\sigma, [i\|\beta], A) \rightarrow ([\sigma\|i], \beta, A)$	$(SHIFT)$
	$([\sigma\|i], [j\|\beta], A) \rightarrow ([\sigma\|i\|j], \beta, A \cup \{(i, l, j)\})$	$(ArcRight_l)$
	if $(k, l', i) \notin A$ and $i \neq 0$ then	
	$([\sigma\|i], [j\|\beta], A) \rightarrow ([\sigma], [j\|\beta], A \cup \{(j, l, i)\})$	$(ArcLeft_l)$
	if $(k, l', i) \in A$ then	
	$([\sigma\|i], \beta, A) \rightarrow (\sigma, \beta, A \cup \{(i, l, j)\})$	$(REDUCE)$

Algorithm 8: The arc-eager transition system for dependency parsing J. Nivre (2004)

Following the work of Abney and Johnson (1991), the Arc Eager transition system in figure 6.2, formalized by J. Nivre (2003), is a variant of Arc Standard in which RightArc is redefined so that the dependent is pushed onto the stack, making it possible to eagerly attach a right-dependent to its head while allowing more dependents to attach to it. To accommodate this, a new transition, REDUCE, makes it possible to pop the dependent from the top of the stack at a later point on the condition that it already has a head.

In figure 6.2, the Arc Eager transition sequence, given by its oracle, for the sentence "Economic news had little effect on financial markets." is shown. As opposed to Arc Standard, RightArcs are observed occurring early in the sequence, doubling as replacements for many SHIFT operations and complemented by REDUCE transitions towards the end. Note how both transition systems and their respective oracles output the same dependency tree (figure 6.1). However, in Arc Eager, "had" is attached to its head (ROOT) earlier in the transition sequence.

6.4 Arc ZEager

The Arc Standard and Arc Eager transition systems defined in the previous chapters are often found with slight variations in the literature and in practice. For example, in recent literature Hatori et al. (2011) use a version of Arc Standard where ArcLeft and ArcRight operations (which they call ReduceLeft and ReduceRight) occur on the top two elements of the stack, rather than on the top elements of the stack and the buffer. This specific modification retains the nature of Arc Standard, since the functionality of the top of the buffer is only relocated to the top of the stack; functionally these are equivalent, but using this modification clears the way for using the SHIFT transition as a tagger.

Transition	Stack σ	Buffer β	Arcs A
$c_0 =$	([R],	[Economic,...,.],	$\emptyset$)
$SH \rightarrow$	([R,Economic],	[news,...,.],	$\emptyset$)
$LA_{ATT} \rightarrow$	([R],	[news,...,.],	A_1 ={(news,ATT,Economic)})
$SH \rightarrow$	([R,news],	[had,...,.],	A_1)
$LA_{SBJ} \rightarrow$	([R],	[had,...,.],	$A_2 = A_1 \cup$ {(had,SBJ,news)})
$RA_{PRED} \rightarrow$	([R,had],	[little,...,.],	$A_3 = A_2 \cup$ {(R,PRED,had)})
$SH \rightarrow$	([R,had,little],	[effect,...,.],	A_3)
$LA_{ATT} \rightarrow$	([R,had],	[effect,...,.],	$A_4 = A_3 \cup$ {(effect,ATT,little)})
$RA_{OBJ} \rightarrow$	([R,had,effect],	[on,...,.],	$A_5 = A_4 \cup$ {(had,OBJ,effect)})
$RA_{ATT} \rightarrow$	([R,...,on],	[financial,...,.],	$A_6 = A_5 \cup$ {(effect,ATT,on)})
$SH \rightarrow$	([R,...,financial],	[markets,.],	A_6)
$LA_{ATT} \rightarrow$	([R,...,on],	[markets,.],	$A_7 = A_6 \cup$ {(markets,ATT,financial)})
$RA_{PC} \rightarrow$	([R,...,markets],	[.],	$A_8 = A_7 \cup$ {(on,PC,markets)})
$RE \rightarrow$	([R,...,on],	[.],	A_8)
$RE \rightarrow$	([R,...,effect],	[.],	A_8)
$RE \rightarrow$	([R,had],	[.],	A_8)
$RA_{PU} \rightarrow$	([R,...,.],	[],	$A_9 = A_8 \cup$ {(had,PU,.)})
$RE \rightarrow$	([R,had],	[],	A_9)
$RE \rightarrow$	([R],	[],	A_9)

TABLE 6.2: Example Parsing Sequence with Arc Eager; SH - SHIFT, RA/LA - ArcRight/ArcLeft, RE - REDUCE, R - ROOT (Kübler, McDonald, and J. Nivre, 2009, Chapter 3)

When we set out to test the joint hypothesis, we decided to first reproduce the state-of-the-art (SOA) system for English at the time as presented by Zhang and Nivre (2011). Reproducing SOA served multiple purposes, but first and foremost it provided us the ability to test numerous experimental approaches to joint processing, some of which would require significant engineering effort if we were to build upon the existing implementation - the zpar parser.

Second, while nevertheless requiring a significant engineering effort, implementing a new parser from scratch had the added benefit of helping us acquiring an understanding of the minute intricacies of dependency parsing and our framework.

Third, the often undervalued but important task of reproducing scientific work can result in unexpected insight. This may occur when previously overlooked or unreported assertions and assumptions are brought to light. Also, exposing supposedly minor variations of theories when applied in practice may be of interest to the community.

In our reproduction of Zhang and Nivre (2011), we discovered a variant of Arc Eager in the code that we call Arc (Z)Eager. Arc ZEager has interesting subtle variations on the configuration structure, transition system, its oracle, computation of the parametric model, and surprisingly even the algorithms for the BEST and TOP-B functions of beam

search. One frustrating aspect of these variations is that in attempting reproduction, we learned that deviation from any one of them (including bugs!) results in loss of accuracy.

In this work, we survey the subtle difference between the transition systems.

6.4.1 Configuration and Root Node

The configuration for Arc ZEager is very similar to Arc Eager, except it includes a second stack called a head-stack (sic), which we shall denote as σ_h: $(\sigma, \sigma_h, \beta, A)$. Arc ZEager is root-less; the initial configuration has an empty stack and head-stack.

6.4.2 End of Sequence

The first noticeable difference is in end-of-sequence behavior; where $|\beta| = 0$, and the only operations left to apply are REDUCEs to achieve a sequence of length exactly $2n$ (n=the number of words/morphemes in the input sentence). If $|\beta| = 0$, the parser continually REDUCEs the remainder of the stack, with the exception of the last value in the stack. For the last value, a unique transition named POPROOT (sic), is required to occur as the last transition of the sequence, i.e. when $|\beta| = 0$ and $|\sigma| = 1$. This is interesting because this transition necessarily has unique features, reserved only for the word in the sentence that serves as its root.

6.4.3 SHIFT

One of the most interesting changes introduced by Arc ZEager is the added pre-condition that a SHIFT may not occur after a REDUCE. This condition is followed by the comment: "there are many ways when there are many arcrighted items on the stack and the root need arcleft. force this.". This is an interesting, possibly linguistically motivated, optimization of the transition system itself. Additionally, SHIFT requires that either $|\beta| > 1$ or $|\sigma| = 0$. We have no insight into this condition. As in Arc Eager, SHIFT pushes the top of the buffer onto the stack, but it also pushes it onto the top of σ_h.

6.4.4 ARCLEFT

ArcLeft is the same as in Arc Eager, except it also pops the head-stack. ArcLeft is the only operation to pop the head-stack, yet SHIFT is the only operation to push onto it. This leads one to believe that the head-stack is a stack of head-less words on the stack:

SHIFT-ed words are head-less, and ArcLeft results in attaching a head to head-less words on the stack, therefore they are removed.

6.4.5 ARCRIGHT

ArcRight is the same as in Arc Eager, except it has the additional pre-condition that either $|\beta| > 1$ or $|\sigma_h|$=1. This condition is perhaps the most peculiar. $|\beta| > 0$ and $|\sigma| > 0$ are implicitly required as a pre-condition for Arc Eager, therefore the relevance of $|\sigma_h| = 1$ is only when $|\beta| = 1$. Assuming we accept the plausible explanation that the head-stack is a stack of head-less elements in $|\sigma|$, this added pre-condition means that ArcRight will not occur if the only element in β has a head, meaning the only possible next transitions are either a SHIFT or a REDUCE (ArcLeft is not relevant because it attaches a head onto the head of the stack).

6.4.6 Discussion

The particular conditions of SHIFT and ARCRIGHT in Arc ZEager are important because they result in a limitation of the search space when they apply. Where a limited next transition would otherwise be required to be weighed against all the possible next transitions, Arc ZEager deliberately does away with it, begging the question: Are there some linguistic insights that motivates these limitations? Since using the vanilla Arc Eager transition system results in a performance loss, it appears the community could find utility for such linguistic insights.

We intend to reach out to the authors of Arc ZEager regarding these issues for further discussion.

6.5 Dependency Parsing of Modern Hebrew

We consider the suitability of Arc Standard vs Arc ZEager in the context of Modern Hebrew. The suitability of each can be argued; Arc Eager is legitimized by its logical derivation from the study of Arc Standard, but its applicability may be due to the nature of the English language. Most pointedly, English grammar requires a strict Subject-Verb-Object (SVO) order, but Hebrew grammar allows freedom in this ordering. For example, in English one may only say “Tom ran home” - any other ordering of the subject, verb, or object would be considered grammatically incorrect. In contrast, the equivalent Hebrew sentence TWM RC HBITA, exhibiting SVO order, can also be written HBITA RC TWM, which is OVS order.

An empirical study by J. Nivre (2008) compares the performance of Arc Standard and Arc Eager for 13 languages, amongst them Arabic and Turkish, considered MRLs and both allowing for some degree of freedom in word order. For these languages, Arc Standard performance is observed to be slightly better than Arc Eager, although no conclusive explanation exists for this difference as too many variables may have contributed.

We therefore conduct an empirical evaluation for Modern Hebrew, comparing Arc Standard to Arc ZEager.

6.5.1 Rich Linguistic Features

The foremost contribution of Zhang and Nivre (2011) is the set of Rich Non-Local features, adding high-order feature sets previously found only in graph-based parsers. In order to at least attempt a fair comparison of Arc Standard to Arc ZEager, we adapt the feature set of Zhang and Nivre (2011) to Arc Standard and Modern Hebrew, to the extent that this is possible. Recall that Arc Eager attaches the head of a right-dependent as soon as possible, thus allowing access to a more complete picture of its syntactic surroundings during later processing. This is not possible in Arc Standard, and no feature set could, by itself, overcome this limitation.

However, we address the dependence of Rich Non-Local features on strict word order to the free word order of MRLs. We call our feature set *Rich Linguistic features*. The essence of the two feature sets is the same, but we replace some features relying on strict word order with parallel features that allow for free word order.

To construct our features, we define new properties to capture *linguistic* information of selectional preferences and subcategorization frames (Tesnière, 1959; Chomsky, 1965). To capture the syntactic characterizations of the subcategorization frame, we define f_p as the multi-set of parts-of-speech of dependents of a given head. Likewise, to capture the functional characterizations of subcategorization frames, we define the properties s_f, referring to the multi-set of labels of all dependents in the frame of a given head. For valency, we define the properties v_f, referring to the number of dependents of a given head.

For selectional preferences, and to accommodate free word order, we allow for the definition of order-agnostic bi-lexical dependency features to be generated *for each* dependent of a head, indicated as C_i.

To generalize the head-stack σ_h of Arc ZEager, we introduce the edge potential property $o = |\sigma_h|$, and likewise update Arc Standard such that it maintains σ_h.

Feature Set	Morpheme Form	Arc System	zpar		yap	
			UAS	LAS	UAS	LAS
Non Local (EN 40k)	Form	ZEager	93.1	91.8	93.1	91.8
Non Local (EN 5k)	Form	ZEager	90.4	88.9	90.4	88.9
Non Local (HE)	Form	ZEager	**88.7**	82.5	86.6	79.7
Non Local (HE)	Lemma	ZEager	**88.9**	82.9	86.3	79.7
Linguistic (HE)	Form	ZEager	-	-	86.1	79.3
Linguistic(HE)	Lemma	ZEager	-	-	86.2	79.7
Linguistic (HE)	Form	Standard	-	-	**88.1**	77.5
Linguistic (HE)	Lemma	Standard	-	-	**88.5**	77.7

TABLE 6.3: Dependency Parsing Variants for Modern Hebrew, with English for Reference

Additionally, we augment existing features with morphological properties, providing an augmentation operator that treats features as templates. The augmentation operator allows for creating multiple instances of the same feature with and without morphological features. See Appendix C for a full comparison of rich non-local/linguistic feature models.

6.6 Experiments

In our approach to empirical evaluation, we first verify that we are able to reproduce the results of Zhang and Nivre (2011) given their setup. We therefore first compare the outputs of zpar and yap on the Penn Treebank (PTB), using the same standard train/dev/test split, with predicted POS tags using zpar's implementation of the Collins (2002) tagger.

The training set of the full PTB contains 40k projective sentences, dwarfing the Modern Hebrew treebank of just 5k sentences, of which a handful are non-projective. To enable a comparison of models' performance between English and Modern Hebrew, we also run both parsers on a set of the first 5k sentences of the English training set.

With the appropriate flags set for bug emulation, we verify our implementation's output is at parity with zpar. We then compare the performance with these same settings on our updated Modern Hebrew treebank. To test our feature set, we then compare the performance of the Rich Linguistic feature set, and alternate between surface form and lemma representation of morphemes.

6.7 Results

In table 6.3 we present the results of our experiments on the development sets of English and Hebrew. We first verify that indeed, given the full English corpus of the Penn Treebank, we have successfully reproduced the results of Zhang and Nivre (2011). In addition, we isolate the effect of treebank size, and report the results of Zhang and Nivre (2011) on the PTB when limited to just 5k sentences, resulting in an absolute drop of 2.7% and 2.9% in UAS and LAS, respectively, and enlarging the error rate by about 50% and 30%, respectively. This exemplifies the critical impact of corpus size. We must bear in mind this significant impact on model accuracy when considering our results on Modern Hebrew.

When applied to the Modern Hebrew treebank with lemmas replacing forms where possible, the model of Zhang and Nivre (2011), as implemented by zpar, results in state-of-the-art scores of 88.9 UAS and 82.9 LAS. Compared to our baseline result of Goldberg and Elhadad (2010) of 84.2 UAS (they do not report LAS), this result represents a significant advancement in the state-of-the-art for the dependency parsing task of Modern Hebrew. Of note is the large difference in error rates of UAS and LAS. Unfortunately, due to time and scope limitations, we did not investigate this further.

Bare in mind that the PTB and Hebrew treebank are not directly comparable since their annotation directives are not equivalent - they have different sets of POS tags and dependency labels, resulting in different syntactic theories.[2]

When applying the same models as implemented in yap, we see a noticeable difference in performance: we present UAS/LAS of 86.3/79.7 in the same setting as the state-of-the-art results of zpar. We attribute this to a possible software bug in implementation, noticing that yap ceased the learning process after just 7 iterations, compared to zpar which continued to 34 iterations. While we could continue to debug yap and analyze the difference, we preferred to stop at this point due to time constraints in order to apply our remaining time to the main hypothesis (joint processing).

We can still gain some valuable insight through comparison to our own baseline results - we do see a noticeable performance gain of 2.2 in UAS when switching to Arc Standard from Arc ZEager, however at the same time, we see a drop in LAS. This reversal is quite peculiar, as one expects these two measures to change in the same direction. We did not investigate this further.

[2]Of note is the minute difference that the Modern Hebrew treebank has an annotated root label, whereas the English PTB does not. Since Arc ZEager is root-less, and the POPROOT transition does not label the root, we disregard the root's dependency label during evaluation.

Data	Experiment	UAS	LAS	F1 (unlabeled)	F1 (labeled)
Gold	G & E (2010)	84.2	n/a	n/a	n/a
Gold	yap	86.6	79.7	86.6	79.7
Gold	zpar	88.9	82.9	88.9	82.9
Pred	G & E (2010)	76.2	n/a	n/a	n/a
Pred	yap (This work)	81.7	75.5	82.2	76.0
Pred	zpar (This work)	79.7	73.3	80.3	73.8

TABLE 6.4: Impact of Gold vs Predicted morphology on performance of various standalone dependency parsers; Goldberg and Elhadad (2010) use predicted morphology by Adler (2009), yap and zpar use the best MD setting of this work, and both use the same setting: rich non-local features with Arc ZEager

To illustrate the impact of gold vs predicted morphology, we provide a comparison in table 6.4. We observe a noticeable drop in UAS as reported by Goldberg and Elhadad (2010), as well as in UAS and LAS of our own experiments. We use the form, rather than lemma variant of the input, because the MD does not predict lemmas. UAS/LAS scores, when applied to predicted input, correspond to precision. For a more complete comparison, we also supply F_1 scores.

Chapter 7

Transition-Based Joint Processing

> "The whole is greater than the sum of its parts."
>
> *Aristotle*

Given our morphological disambiguation and dependency parsing processors as transition-based systems in the same framework, we seek an integration such that syntax may affect morphological disambiguation and vice versa. We propose to literally embed the two standalone processors presented in (5, 6), and define coherent logic, called a *joint strategy*, that deterministically chooses which processor to apply given a configuration state. We first define a joint morpho-syntactic configuration that embeds both the MD and dependency parser configurations (7.1). We then formally define the concept of a joint strategy (7.2), propose two such strategies (7.2.1, 7.2.2), carry out experiments to validate our main hypothesis (7.3), and report our results (7.4).

7.1 A Joint Morpho-Syntactic Configuration

Let c_m and c_d be MD and dependency parser configurations as defined in chapters 5 and 6, respectively. We define a Joint Configuration as follows:

$$c_j = (c_m, c_d) = ((L, n, i, M), (\sigma, \beta, A)) \tag{7.1}$$

We initialize the embedded MD configuration c_m with the MD transitions system's initialization function as in chapter 5, but leave c_d empty, with $\sigma = \beta = []$ as an empty stack and buffer, respectively, and as before $A = \emptyset$. c_j is terminal if and only if c_m and c_d are both terminal configurations of their respective transition systems.

7.2 Joint Strategies

Our baseline approach, called the Pipeline strategy (for which we need not supply a formal definition), first applies the morphological disambiguation processor, chooses the best output, and then applies the syntactic processor. The Pipeline strategy does not require a joint framework at all, and is applicable to any two frameworks for MD and dependency parsing processing.

We seek to improve upon the Pipeline strategy/approach, but we first must adjust the MD transition system such that its disambiguations feed into the embedded configuration of the dependency parser. We modify the morpheme-based MD transition such that it accepts a joint configuration and populates the buffer of the dependency configuration:

$$MD_s^* : ((L, n, i, M), (\sigma, \beta, A)) \rightarrow ((L, q, j, M \cup \{m\}), (\sigma, [m|\beta], A)) \tag{7.2}$$

We may now define a proper joint transition system, using a joint strategy. Let $\mathcal{T} = (T_m^*, T_d)$ be the ordered pair of the transitions sets of the MD and dependency parsing transition systems (respectively), let $\mathcal{C} = \{c_j\}$ be the set of all possible non-terminal joint configurations, and C_{t_m}, C_{t_d} be the set of terminal configurations of their respective transition systems. A joint strategy is a deterministic function that, given a non-terminal joint configuration and set of transition systems, chooses exactly one transition system:

$$JOINT : \mathcal{C} \rightarrow \mathcal{T} \tag{7.3}$$

7.2.1 MDFirst

With two integrated processors defined in the same framework, which maintains a set of B-best candidates, we propose the trivial improvement upon the pipeline approach of not choosing just the top-scoring candidate of the MD processor, but passing on all B candidates to the dependency processor. We call this strategy the $MDFirst$ strategy:

$$MDFirst((c_m, c_d) \in \mathcal{C}) = \begin{cases} T_m^* & \text{if } c_m \notin C_{t_m} \\ T_d & \text{otherwise} \end{cases} \tag{7.4}$$

Algorithm 9: The $MDFirst$ Joint Strategy

While $MDFirst$ is trivial, it offers us the opportunity of allowing the syntactic processor to "re-rank" an initially lesser scored disambiguation if its syntactic processing ends up resulting in a better dependency parse.

7.2.2 ArcGreedy

Since both transition systems process their input left-to-right, there is no inherent constraint preventing the application of a syntactic transition as soon as the embedded dependency configuration meets the minimal state required for a dependency transition to be applied. We therefore propose a set of $ArcGreedy_k$ strategies, wherein we greedily choose to apply a syntactic transition if the dependency configuration's buffer β is populated by at least k morphemes:

$$ArcGreedy_k(c_m, c_d = (\sigma, \beta, A)) = \begin{cases} T_m^* & \text{if } |\beta| < k \\ T_d & \text{otherwise} \end{cases} \tag{7.5}$$

Algorithm 10: The $ArcGreedy_k$ Set of Joint Strategies

In our model, we (currently) do not propose altering the learning models of either processors. We require a minimum k morphemes in β so that the feature sets of the syntactic processor may look k morphemes "forward" in order to predict its next syntactic transition.

Thus, both $MDFirst$ and $ArcGreedy_k$ are joint morpho-syntactic transition systems, in that candidates of the framework have a joint global score:

$$Score_{Joint}(y) = Score_{MD}(y) + Score_{Dep}(y) \tag{7.6}$$

$$= \sum_{i=1}^{d} \omega_i^{md} \phi_i^{md}(y_{md}) + \sum_{j=1}^{d} \omega_j^{et} \phi_j^{et}(y_{et}) + \sum_{r=1}^{d} \omega_r^{dep} \phi_r^{dep}(y_{dep}) \tag{7.7}$$

$$= \sum_{c_k \in y_{md}} \sum_{i=1}^{d} \omega_i^{md} \phi_i^{md}(c_k) + \sum_{c_l \in y_{et}} \sum_{j=1}^{d'} \omega_j^{et} \phi_j^{et}(c_l) + \sum_{c_d \in y_{dep}} \sum_{r=1}^{d''} \omega_r^{dep} \phi_r^{dep}(c_m) \tag{7.8}$$

Where c_{m_d} and c_{et} are the (unique) resulting configurations of MD and ET transitions respectively, and c_{dep} are the (unique) resulting configurations of syntactic (arc system) transitions.

The theoretical advantage of $ArcGreedy_k$ compared to $MDFirst$ is that the incremental update of the joint global score by the former alternates between MD and syntactic predictions, allowing for syntax and morphological disambiguation to interact frequently, such that syntax can affect the ordering of candidates many times in the parsing sequence, correcting "local" mistakes close to when they occur in the sequence. This is contrast to $MDFirst$, where syntax may only affect morphological disambiguation once it is complete.

Strategy	Arc System	ET	Full/POS MD F_1	Un/labeled F_1
Pipe/Gold (Prev)	G&E 2010 (MST)	n/a	100/100	84.4 (UAS)
Pipe/Gold (new)	ZEager	n/a	100/100	89.0/82.6
Pipe/Pred (Prev)	G&E 2010 (MST)	n/a	n/a	76.4 (UAS)
Pipe/Pred	ZEager (zpar)	+ET	94.3/94.9	80.3/73.8
Pipe/Pred	ZEager (yap)	+ET	94.3/94.9	83.5/**76.6**
Pipe/Pred	Standard (yap)	+ET	94.3/94.9	**83.6**/75.8
Joint/$MDFirst$	ZEager		94.7/95.3	78.2/70.5
Joint/$MDFirst$	ZEager	+ET	94.9/95.6	80.2/72.5
Joint/$ArcGreedy_3$	ZEager		94.4/95.1	80.3/72.5
Joint/$ArcGreedy_3$	ZEager	+ET	94.6/95.3	79.8/72.4
Joint/$MDFirst$	Standard		94.8/95.4	80.9/73.4
Joint/$MDFirst$	Standard	+ET	94.8/95.5	80.7/73.3
Joint/$ArcGreedy_3$	Standard		94.7/95.3	**81.7/74.4**
Joint/$ArcGreedy_3$	Standard	+ET	**95.1/95.8**	81.6/74.3

TABLE 7.1: Joint Morpho-Syntactic Processing Results on the development sets in an ideal setting (infused); Pipe-Gold represents the upper bound (gold morphology), while Pipe-Pred represents the drop incurred by pipeline processing; Prev indicates results as reported by Goldberg and Elhadad (2010), as UAS

7.3 Experiments

To test our hypothesis, we investigate joint parsing with $MDFirst$ and $ArcGreedy_3$ strategies.[1] We experiment with both Arc ZEager and Arc Standard transition systems for dependency parsing. To test the applicability of the ENDTOKEN transition to variable-length sequences in a joint environment, we also run all experiments with and without ENDTOKEN.

7.4 Results

In Table 7.1 we present results for the various joint experiment settings. For comparison, we include the best results with gold MD and pipeline settings.

We report the success of all joint models in surpassing standalone MD for Full and POS MD of Modern Hebrew. We observe that ENDTOKEN generally increases MD results by up to 0.4 absolute points, proving it's applicability (at least to MD) in a joint setting. We report our best Full MD result in the expected setting, $ArcGreedy_3$ with ENDTOKEN, with Full and POS MD scores of 95.1 and 95.8, respectively, on the development set. We verify these results on the test set, where we report 86.26/89.4 and 92.9/93.77 Full/POS MD scores in the realistic and ideal setting, respectively. For

[1] We set $k = 3$ because some features of Zhang and Nivre (2011) require three morphemes in the buffer. See Appendix C

comparison, our best results for standalone MD are 86.23/88.85 and 92.9/93.73 for realistic and ideal settings, respectively. While Full MD scores are only marginally better in the joint approach, we observe a noticeable improvement in segmentation and POS disambiguation, especially in the realistic setting. Thus we prove the utility of joint processing for morphological disambiguation.

Unfortunately, the improvement in MD appears to come at the expense of syntax, where we observe a drop in Unlabeled and Labeled F_1 scores. However, we do observe that our best syntactic results in a joint setting are with the Arc Standard transition system, and that $ArcGreedy_k$ mostly improves over $MDFirst$. This leads us to believe that although dependency parsing results in joint settings are overall less than those of pipeline results, we are headed in the right direction. We further discuss possible issues and future work in Section 8.1.3.

Chapter 8

Discussion and Conclusion

> "In literature and in life we ultimately pursue, not conclusions, but beginnings."
>
> *Sam Tanenhaus*

In this work we present a novel morphological disambiguation transition-system, an adaptation of a transition-based dependency parser to MRLs, and a joint morpho-syntactic parser that unifies them.

We present a discussion of the results for standalone (8.1.1,8.1.2) and joint parsers (8.1.3), followed by our concluding remarks (8.2).

8.1 Discussion

8.1.1 Morphological Disambiguation

In order to test our joint hypothesis, we chose to implement a morphological disambiguator in a proven framework for dependency parsing. We chose the transition-based framework of Zhang and Clark (2011), for which we present an instantiation of a novel, open class morphological disambiguator. We conduct an empirical investigation of a word-based vs. morpheme-based approach, and conclude that the latter performs better. To mitigate the issue of variable-length sequences introduced by the morpheme-based approach we propose a novel ENDTOKEN transition and show it has some effect as a counter-balance to the bias of longer sequences.

Together with a new lexicon-based morphological analyzer and rectified corpus, the performance of the transition-based MD is state-of-the-art for Modern Hebrew, both

in ideal and realistic settings. Using a data-driven morphological analyzer, we apply the MD to 48 treebanks of the Universal Dependencies data set and show multilingual applicability, especially regarding Morphologically Rich Languages.

Looking forward, the transition-based MD would likely profit from a feature model backed by word embeddings and LSTMs on languages for which this is applicable. Unfortunately, to the best of our knowledge, there are no Modern Hebrew word embedding or LSTM models for morphologically ambiguous lattices. An embedding of a morphologically ambiguous lattice must represent the multitude of spellouts that derive from it; how this may be done is an open research question.

8.1.2 Dependency Parsing

For the second part of the joint hypothesis, we start with the transition-based dependency parser of Zhang and Nivre (2011), at the time the state-of-the-art syntactic parser in English. We compare and contrast arc systems, and conclude that in an ideal setting, the approach of Zhang and Nivre (2011) yields the best results on Modern Hebrew too. Although not directly comparable, when controlling for corpus size, performance on Modern Hebrew is less than that of English, although not by much. Given the significant loss of performance noticeable in English when limiting the corpus, we conclude the need for a much larger annotated corpus, but we acknowledge the enormous effort required by such an endeavor.

We did not perform thorough analysis of our reproduced parser's performance results in Hebrew, although such an analysis is wanting. In the future, an immediate improvement could be support for non-projective dependencies using existing methods in the literature.

As with MD, the model could profit significantly from word embeddings and LSTMs could assist (if not replaced entirely) the feature model, but such models do not yet exist for MRLs such as Hebrew.

8.1.3 Joint Morpho-Syntactic Processing

We introduce the novel concept of a joint strategy, and provide a deterministic example that integrates morphological disambiguation and syntactic processing. In doing so, we present the first joint morphological disambiguator and dependency parser in a transition-based framework for MRLs.

We show that syntax indeed improves morphological disambiguation, improving upon the standalone MD. Furthermore, we show that our proposed mitigation of bias in variable-length sequences is applicable to joint processing (at least for MD).

However, we report worse performance for dependency parsing in joint settings when compared to a pipeline process. Also, our mitigation strategy of variable-length sequences appears to be detrimental to syntax. Regardless of variable-length mitigation, the decreased performance in joint processing may be due to the affect of longer parsing sequences resulting in higher error rates. Also, it is possible that while morphology and syntax indeed interact implicitly, they do so with a delay. Recall that we experimented only with $ArcGreedy_3$, we suggest experimenting with other values of k, but advise to provide a solution for the forward-looking features of the dependency parser that require access to unattached disambiguated morphemes.

Due to time constraints, we did not perform analysis of our joint results, although such analysis is necessary to understand why we only see an improvement in MD but not syntactic performance in a joint setting.

Looking forward, we suggest possible improvements to the joint model. For example, the deterministic joint strategy may be too rigid, perhaps a non-deterministic strategy could allow for obvious MD sequences, such as known multi-word expressions and proper nouns; this would require a dynamic oracle (Goldberg and J. Nivre, 2012).

Alternatively, it might be possible to somehow delay full morphological disambiguation, similar to an approach advocated for in Hatori et al. (2011). This would be uniquely complicated for an MRL with segmentation, but may be interesting to apply to tokens that may only form a self-contained dependency tree, such that all dependents share a common head outside the token.

Observe that the joint aspect of processing is implicit, in that syntax and MD interact indirectly through the joint global score, and is afforded by the existence of a beam search allowing multiple candidates to be considered. If the beam is limited to 1 candidate, syntax cannot affect morphological disambiguation. Therefore, another possible future direction is unification of MD and syntactic transitions into a single, truly joint, transition.

Additionally, we suggest adding features to both underlying processors such that they may access the information in each others' configurations.

8.2 Conclusion

We present an MD transition-based system that can effectively cope with extreme morphological ambiguities in MRLs, that is also compatible with transition-based frameworks in which state-of-the-art dependency parsers are implemented.

In the same framework, we apply a transition-based dependency parser for English to the latest Modern Hebrew treebank, with state-of-the-art results.

With these two parsers in the same framework, we propose and implement a method for joint morpho-syntactic parsing.

To the best of our knowledge, this is the first joint framework for MRL segmentation and tagging in a transition-based setup. Moreover, we present the best MA&D results for Modern Hebrew to date in both ideal and realistic settings. The transition-based system provides a first tier for MRLs for dependency parsing in real-world scenarios, dispensing with the need of external pre-processing.

Appendix A

Word-Based MD Feature Model

A.1 Feature Properties

Let (s, e, f, t, g) be a morpheme of a word-lattice L and $c = (L, i, n, M)$ a configuration.

Lattice-based properties are addressed by L_j, where j references the $i + j$-th word-lattice [1] of L (L_j is relative to i).

We define the following feature properties:

- o - lattice's token
- a - lattice's set of all possible paths/spellouts, with projected morphemes
- p - path of a lattice's spellout with projected morphemes - exists only for disambiguated lattices

A.2 Features

Lattice Unigram: L_0a, L_0o

Lattice Bigram: $L_0oL_{-1}o, L_0oL_{-1}a, L_0aL_{-1}o, L_0aL_{-1}a$

Lattice Trigram:
$L_0oL_1oL_{-1}o, L_0oL_1aL_{-1}o, L_0aL_1oL_{-1}o, L_0aL_1aL_{-1}o,$
$L_0oL_1oL_{-1}a, L_0oL_1aL_{-1}a, L_0aL_1oL_{-1}a, L_0aL_1aL_{-1}a$

Previously Disambiguated Lattice Unigram: $L_{-1}p$

[1] Negative values of j address previous word-lattices, so -1 addresses the word-lattice occurring in the input sentence, before the i-th word-lattice in L

Previously Disambiguated Lattice Bigram:

$L_{-1}pL_0a, L_{-1}pL_0o$

$L_{-1}paL_0a, L_{-1}paL_0o, L_{-1}poL_0a, L_{-1}poL_0o$

Previously Disambiguated Lattice Trigram:

$L_{-2}pL_{-1}pL_0a, L_{-2}pL_{-1}pL_0o, L_{-2}pL_{-1}paL_0a,$

$L_{-2}pL_{-1}paL_0o, L_{-2}pL_{-1}ptL_0a, L_{-2}pL_{-1}ptL_0o$

$L_{-2}paL_{-1}pL_0a, L_{-2}paL_{-1}pL_0o, L_{-2}paL_{-1}paL_0a,$

$L_{-2}paL_{-1}paL_0o, L_{-2}paL_{-1}ptL_0a, L_{-2}paL_{-1}ptL_0o$

$L_{-2}poL_{-1}pL_0a, L_{-2}poL_{-1}pL_0o, L_{-2}poL_{-1}paL_0a,$

$L_{-2}poL_{-1}paL_0o, L_{-2}poL_{-1}ptL_0a, L_{-2}poL_{-1}ptL_0o$

ET[2]:$L_{-1}p, L_{-1}po, L_{-1}pa$

[2]ET feature templates are restricted to fire when predicting a ET transition, all other feature templates are not applied.

Appendix B

Morpheme-based MD Feature Model

B.1 Feature Properties

Let (s, e, f, t, g) be a morpheme of a word-lattice L and $c = (L, i, n, M)$ a configuration.

Properties are addressed by M_k, where k indexes the k-th most recently disambiguated morpheme in M of c, and we define the additional feature n, the set of projected ambiguous morphemes (outgoing edges) of the lattice node n.

Define the prefix property e as a *generator* property — it creates a feature for each substring of lengths 1 to 10 of a token, starting at the beginning. Likewise, define the suffix property x as a *generator* of substrings starting at the end of a token.

Define the signature property g as a set of bits, each indicating that at least one character in a token returns true for its respective indicator function. The set of functions is:

- IsDigit — Input is a decimal digit (0-9)
- IsGraphic — Input is a Graphic as defined by the Unicode standard (categories L, M, N, P, S, and Z)
- IsLetter — Input is a Unicode letter (category L)
- IsLower — Input is a lower case letter
- IsMark – Input is a Unicode mark (category M)
- IsNumber — Input is a Unicode number (category N)

- IsPunct — Input is a Unicode punctuation character (category P)
- IsSymbol — Input is a symbolic character
- IsTitle — Input is a title case letter
- IsUpper — Input is an upper case letter

All functions are defined in the Go language unicode library (https://golang.org/pkg/unicode/).

B.2 Features

Disambiguated Morphemes Unigram: $M_0f, M_0t, M_0ft, M_0g, M_0fg, M_0tg, M_0ftg$

Disambiguated Morphemes Bigram:
$M_0fM_1f, M_0fgM_1f, M_0fpM_1f, M_0ftgM_1f$
$M_0fM_1ft, M_0fgM_1ft, M_0ftM_1ft, M_0ftgM_1ft$
$M_0tM_1t, M_0tgM_1tg, M_0tgM_1t, M_0gM_1t$

Disambiguated Morphemes Trigram:
$M_0fM_1fM_2f, M_0tM_1tM_2t, M_0ftM_1ftM_2ft$
$M_0ftgM_1ftM_2ft, M_0ftgM_1ftgM_2ftg, M_0gM_1tM_2t$

Current Ambiguous Morphemes Unigram: $L_0n, L_0na, L_0nt, L_0t, L_0g, L_0e, L_0x$

Next Ambiguous Morphemes with Previous Lattice Disambiguation Bigram:
$L_{-1}pL_0n, L_{-1}pL_0na, L_{-1}qL_0no, L_{-1}paL_0n, L_{-1}paL_0na,$
$L_{-1}paL_0no, L_{-1}poL_0n, L_{-1}poL_0na, L_{-1}poL_0no$

Next Ambiguous Morphemes with Previous Lattice Disambiguation Trigram:
$L_{-2}pL_{-1}pL_0n, L_{-2}pL_{-1}pL_0na, L_{-2}pL_{-1}pL_0no, L_{-2}pL_{-1}paL_0n$
$L_{-2}pL_{-1}paL_0na, L_{-2}pL_{-1}paL_0no, L_{-2}pL_{-1}ptL_0n, L_{-2}pL_{-1}ptL_0na$
$L_{-2}pL_{-1}ptL_0no, L_{-2}paL_{-1}pL_0n, L_{-2}paL_{-1}pL_0na, L_{-2}paL_{-1}pL_0no$
$L_{-2}paL_{-1}paL_0n, L_{-2}paL_{-1}paL_0na, L_{-2}paL_{-1}paL_0no, L_{-2}paL_{-1}ptL_0n$
$L_{-2}paL_{-1}ptL_0na, L_{-2}paL_{-1}ptL_0no, L_{-2}poL_{-1}pL_0n, L_{-2}poL_{-1}pL_0na$
$L_{-2}poL_{-1}pL_0no, L_{-2}poL_{-1}paL_0n, L_{-2}poL_{-1}paL_0na, L_{-2}poL_{-1}paL_0no$
$L_{-2}poL_{-1}ptL_0n, L_{-2}poL_{-1}ptL_0na, L_{-2}poL_{-1}ptL_0no$

ET[1]:$L_{-1}p, L_{-1}po, L_{-1}pa$

[1]ET feature templates are restricted to fire only when predicting a ET transition, all other feature templates are not applied

Appendix C

Rich Linguistic Features for Dependency Parsing of Hebrew

C.1 Feature Addresses

We use the feature description scheme of Zhang and Nivre (2011) for easy comparison.

Let $c = (S, N, A)$ be a configuration where S is the stack, N is the buffer.

We define an *address* as the location of a node in the partial dependencies trees in S and N of configuration c. An address has a structure name S or N, a subscript integer to access a k-deep node, and characters to access the heads or dependents of the node found at S_k or N_k. For example, the address S_{0h} refers to the head (if such exists) of the partial tree found at the top of the stack. The address N_1 refers to the node that is second in the buffer.

C.2 Rich Non-Local Addresses and Feature Types

For Rich Non-Local features, we define the special character sequences for high-order addressing:

- $h2$ - the head-of-the-head
- $rd/r2d/ld/l2d$ - the right-most, 2nd-to-right-most, left-most, and 2nd-to-left most dependents (respectively) of a node

We define *attributes* of nodes found at addresses which may be strung together to form a feature:

- w and t - surface form and part-of-speech tag
- l_p/r_p - the set of labels of dependents to the left/right of a node (respectively)
- l - the dependency label of the current node (as a dependent)
- v_l/v_r - the valency (= number) of dependents to the left/right of a node (respectively)
- d - the distance (in the sentence) between the top of the stack and the top of the buffer (regardless of address)

C.3 Rich Linguistic Feature Types

For Rich Linguistic Features, we define the attributes:

- f_p - the multi-set of parts of speech of the dependents of a node
- s_f - the multi-set of labels of all dependents of a node
- v_f - the valency (= number) of all dependents of a node

Also, we define C_i as an address generator - it generate a feature for each dependent of the addressed node.

C.4 Morphological Augmentation

To allow the inclusion of morphology we add the ability of specifying morphological properties to be added to all features of a feature group. Augmentation of a feature *group* does not cause a replacement of the defined features, it only creates a copy with the addition of morphological properties.

To augment a feature group, all the features to the groups are required to have the same number of addresses. An augmentation specifies a character, either h or x, to specify the host or suffix morphological properties as attributes, respectively. If the group has more than one address, the augmentation must specify an address (a 1-indexed integer offset). Multiple augmentations may be used together.

For example, given the feature group Pairs in table C.1, the first few features are S_wtN_0wt, S_0wtN_0w, S_wN_0wt, etc. All features in the Pairs group have two addresses. An example of a morphological augmentation of the Pairs group is $h1h2$, resulting in the new features $S_0wtm_hN_0wtm_h$, $S_0wtm_hN_0wm_h$, $S_wm_hN_0wtm_h$, etc. where m_h is

the set of key-value pairs of properties of the respective morphemes at the top of the stack (S_0) and buffer (N_0).

C.5 Features

The set of rich non-local features of Zhang and Nivre, 2011 and the new rich linguistic features defined in this work are shown in table C.1 and table C.2. The features are shown side by side to ease the comparison of the two feature sets, along with a column indicating the changes made.

The feature groups are augmented with morphological properties as defined in table C.3.

N-L Group	N-L Feature	Ling. Feature	Ling. Group	Change
Single	S_0w	S_0w	Single	
Single	S_0t	S_0t	Single	
Single	S_0wt	S_0wt	Single	
Single	N_0w	N_0w	Single	
Single	N_0t	N_0t	Single	
Single	N_0wt	N_0wt	Single	
Single	N_1w	N_1w	Single	
Single	N_1t	N_1t	Single	
Single	N_1wt	N_1wt	Single	
Single	N_2w	N_2w	Single	
Single	N_2t	N_2t	Single	
Single	N_2wt	N_2wt	Single	
Pairs	S_0wtN_0wt	S_0wtN_0wt	Pairs	
Pairs	S_0wtN_0w	S_0wtN_0w	Pairs	
Pairs	S_0wN_0wt	S_0wN_0wt	Pairs	
Pairs	S_0wtN_0t	S_0wtN_0t	Pairs	
Pairs	S_0tN_0wt	S_0tN_0wt	Pairs	
Pairs	S_0wN_0w	S_0wN_0w	Pairs	
Pairs	S_0tN_0t	S_0tN_0t	Pairs	
Pairs	N_0tN_1t	N_0tN_1t	Pairs	
Three Words	$N_0tN_1tN_2t$	$N_0tN_1tN_2t$	Three Words (A)	
Three Words	$S_0tN_0tN_1t$	$S_0tN_0tN_1t$	Three Words (A)	
Three Words	$S_{0h}tS_0tN_0t$	$S_{0h}tS_0tN_0t$	Three Words (A)	
Three Words	$S_0tN_0tN_{0ld}t$	$S_0tN_0tf_p$	Three Words (B)	$N_{0ld}t \rightarrow N_0f_p$
Three Words	$S_0tS_{0ld}tN_0t$	$S_0tf_pN_0t$	Three Words (B)	$ld/rd \rightarrow f_p$
Three Words	$S_0tS_{0rd}tN_0t$		Three Words (B)	
Distance	S_0wd	S_0wd	Distance	
Distance	S_0td	S_0td	Distance	
Distance	N_0wd	N_0wd	Distance	
Distance	N_0td	N_0td	Distance	
Distance	S_0wN_0wd	S_0wN_0wd	Distance	
Distance	S_0tN_0td	S_0tN_0td	Distance	
Valency	S_0wv_r	S_0wv_f	Valency frames	$v_r/v_l \rightarrow v_f$
Valency	S_0wv_l		Valency frames	
Valency	S_0tv_r	S_0tv_f	Valency frames	
Valency	S_0tv_l		Valency frames	
Valency	N_0wv_l	N_0wv_f	Valency frames	
Valency	N_0tv_l	N_0tv_f	Valency frames	

Table C.1: Rich Non-Local and Linguistic Features Table 1/2

N-L Group	N-L Feature	Ling. Feature	Ling. Group	Change
Unigrams	$S_{0h}w$	$S_{0h}w$	Unigrams (A)	
Unigrams	$S_{0h}t$	$S_{0h}t$	Unigrams (A)	
Unigrams	S_0l	S_0l	Unigrams (A)	
Unigrams	$S_{0ld}w$	$S_0wS_0C_iw$	Unigrams (B)	
Unigrams	$S_{0ld}t$	$S_0wS_0C_it$	Unigrams (B)	
Unigrams	$S_{0ld}l$	$S_0wS_0C_il$	Unigrams (B)	Switch to non-directional bi-lexical dependencies, C_i = for each dependent
Unigrams	$S_{0rd}w$	$S_0tS_0C_iw$	Unigrams (B)	
Unigrams	$S_{0rd}t$	$S_0tS_0C_it$	Unigrams (B)	
Unigrams	$S_{0rd}l$	$S_0tS_0C_il$	Unigrams (B)	
Unigrams	$N_{0ld}w$	$N_0wN_0C_iw$	Unigrams (B)	
Unigrams	$N_{0ld}t$	$N_0wN_0C_it$	Unigrams (B)	
Unigrams	$N_{0ld}l$	$N_0wN_0C_il$	Unigrams (B)	
Unigrams		$N_0tN_0C_iw$	Unigrams	
Unigrams		$N_0tN_0C_it$	Unigrams	New
Unigrams		$N_0tN_0C_il$	Unigrams	
Third Order	$S_{0l2d}w$		Third Order	
Third Order	$S_{0l2d}t$		Third Order	
Third Order	$S_{0l2d}l$		Third Order	
Third Order	$S_{0r2d}w$		Third Order	
Third Order	$S_{0r2d}t$		Third Order	Removed
Third Order	$S_{0r2d}l$		Third Order	
Third Order	$N_{0l2d}w$		Third Order	
Third Order	$N_{0l2d}t$		Third Order	
Third Order	$N_{0l2d}l$		Third Order	
Third Order	$N_0tN_{0ld}tN_{0l2d}t$	N_0tf_p	Third Order	$N_{0l2d}t \rightarrow N_0f_p$
Third Order	$S_{0h2}w$	$S_{0h2}w$	Third Order (A)	
Third Order	$S_{0h2}t$	$S_{0h2}t$	Third Order (A)	
Third Order	$S_{0h}l$	$S_{0h}l$	Third Order (A)	
Third Order	$S_0tS_{0ld}tS_{0l2d}t$	S_0tf_p	Third Order (B)	$ld/rd/l2d/r2d \rightarrow f_p$
Third Order	$S_0tS_{0rd}tS_{0r2d}t$		Third Order (B)	
Third Order	$S_{0h2}tS_{0h}tS_0t$	$S_{0h2}tS_{0h}tS_0t$	Third Order (C)	
LabelSet	S_0wl_p	S_0ws_f	Subcat. frames	
LabelSet	S_0wr_p		Subcat. frames	
LabelSet	S_0tr_p	S_0ws_f	Subcat. frames	$l_p/r_p \rightarrow s_f$
LabelSet	S_0tl_p		Subcat. frames	
LabelSet	N_0wl_p	N_0ws_f	Subcat. frames	
LabelSet	N_0tl_p	N_0ts_f	Subcat. frames	
		S_0wS_0o	Edge Potential	New
		S_0tS_0o	Edge Potential	$o = \lvert\sigma_h\rvert$ = edge potential
		N_0tS_0o	Edge Potential	
		N_0wS_0o	Edge Potential	

Table C.2: Rich Non-Local and Linguistic Features Table 2/2

Feature Group	Morphological Augmentations
Single	h x
Pairs	h1h2 h1x2 x1h2
Three Words (A)	h1h2 h1x2 x1h2 h1h3 h1x3 x1h3 h2h3 h2x3 x2h3
Three Words (B)	h1h3 h1x3 x1h3
Valency	h
Unigram (A)	h x
Bigram	h1h2 h1x2 x1h2
Third Order (A)	h x
Third Order (B)	h x
Third Order (C)	h1h2 h1x2 x1h2 h1h3 h1x3 x1h3 h2h3 h2x3 x2h3

Table C.3: Morphological Augmentation of Rich Linguistic Feature Groups

Bibliography

[AJ91] Steven P. Abney and Mark Johnson. "Memory requirements and local ambiguities of parsing strategies". In: *Journal of Psycholinguistic Research* 20.3 (1991), pp. 233–250. ISSN: 1573-6555. DOI: 10.1007/BF01067217. URL: http://dx.doi.org/10.1007/BF01067217.

[Adl07] Meni Adler. "Hebrew Morphological Disambiguation: An Unsupervised Stochastic Word-based Approach". PhD thesis. Ben-Gurion University of the Negev, Beer-Sheva, Israel, 2007.

[AE06] Meni Adler and Michael Elhadad. "An Unsupervised Morpheme-Based HMM for Hebrew Morphological Disambiguation." In: *ACL*. Ed. by Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle. The Association for Computer Linguistics, 2006. URL: http://dblp.uni-trier.de/db/conf/acl/acl2006.html#AdlerE06.

[And+16] Daniel Andor et al. "Globally Normalized Transition-Based Neural Networks". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. 2016. URL: http://aclweb.org/anthology/P/P16/P16-1231.pdf.

[BSW08] Roy Bar-haim, Khalil Sima'an, and Yoad Winter. "Part-of-speech tagging of Modern Hebrew text". In: *Natural Language Engineering* 14.2 (2008), pp. 223–251.

[BN12] Bernd Bohnet and Joakim Nivre. "A Transition-based System for Joint Part-of-speech Tagging and Labeled Non-projective Dependency Parsing". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '12. Jeju Island, Korea: Association for Computational Linguistics, 2012, pp. 1455–1465. URL: http://dl.acm.org/citation.cfm?id=2390948.2391114.

[Boh+13] Bernd Bohnet, Joakim Nivre, et al. "Joint Morphological and Syntactic Analysis for Richly Inflected Languages." In: *TACL* 1 (2013), pp. 415–428.

URL: http://dblp.uni-trier.de/db/journals/tacl/tacl1.html#BohnetNBFGH13.

[Cho65] Noam Chomsky. *Aspects of the Theory of Syntax*. Cambridge: The MIT Press, 1965. URL: http://www.amazon.com/Aspects-Theory-Syntax-Noam-Chomsky/dp/0262530074.

[Col02] Michal Collins. "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms". In: *Proceedings of ACL*. 2002.

[CR04] Michal Collins and Brian Roark. "Incremental Parsing with the Perceptron Algorithm". In: *Proceedings of ACL*. 2004.

[Gol15] Yoav Goldberg. "A Primer on Neural Network Models for Natural Language Processing". In: *CoRR* abs/1510.00726 (2015). URL: http://arxiv.org/abs/1510.00726.

[GE10] Yoav Goldberg and Michael Elhadad. "An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing". In: *Proceedings of ACL*. 2010.

[GN12] Yoav Goldberg and Joakim Nivre. "A Dynamic Oracle for Arc-Eager Dependency Parsing". In: *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India*. 2012, pp. 959–976. URL: http://aclweb.org/anthology/C/C12/C12-1059.pdf.

[GT08] Yoav Goldberg and Reut Tsarfaty. "A Single Framework for Joint Morphological Segmentation and Syntactic Parsing". In: *Proceedings of ACL*. 2008.

[Hat+11] Jun Hatori et al. "Incremental Joint POS Tagging and Dependency Parsing in Chinese". In: *Fifth International Joint Conference on Natural Language Processing, IJCNLP 2011, Chiang Mai, Thailand, November 8-13, 2011*. 2011, pp. 1216–1224. URL: http://aclweb.org/anthology/I/I11/I11-1136.pdf.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[HJ14] Matthew Honnibal and Mark Johnson. "Joint Incremental Disfluency Detection and Dependency Parsing". In: *Transactions of the Association of Computational Linguistics – Volume 2, Issue 1* (2014), pp. 131–142. URL: http://aclweb.org/anthology/Q14-1011.

[IW08] Alon Itai and Shuly Wintner. "Language resources for Hebrew". In: *Language Resources and Evaluation* 42.1 (Mar. 2008), pp. 75–98.

[KG16] Eliyahu Kiperwasser and Yoav Goldberg. "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations". In: *TACL* 4 (2016), pp. 313–327. URL: https://transacl.org/ojs/index.php/tacl/article/view/885.

[KMN09] Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies 2. Morgan & Claypool Publishers, 2009. ISBN: 1598295969, 9781598295962.

[Li+14] Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, and Wenliang Chen. "Joint Optimization for Chinese POS Tagging and Dependency Parsing". In: *IEEE/ACM Trans. Audio, Speech & Language Processing* 22.1 (2014), pp. 274–286. DOI: 10.1109/TASLP.2013.2288081. URL: http://dx.doi.org/10.1109/TASLP.2013.2288081.

[Li+11] Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. "Joint Models for Chinese POS Tagging and Dependency Parsing". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP '11. Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pp. 1180–1191. ISBN: 978-1-937284-11-4. URL: http://dl.acm.org/citation.cfm?id=2145432.2145557.

[Man11] Christopher D. Manning. "Computational Linguistics and Intelligent Text Processing: 12th International Conference, CICLing 2011, Tokyo, Japan, February 20-26, 2011. Proceedings, Part I". In: ed. by Alexander F. Gelbukh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Chap. Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?, pp. 171–189. ISBN: 978-3-642-19400-9. DOI: 10.1007/978-3-642-19400-9_14. URL: http://dx.doi.org/10.1007/978-3-642-19400-9_14.

[Mcd07] Ryan Mcdonald. "On the complexity of non-projective data-driven dependency parsing". In: *In Proc. IWPT*. 2007, pp. 121–132.

[MP06] Ryan McDonald and Fernando Pereira. "Online learning of approximate dependency parsing algorithms". In: *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006))*. Vol. 6. 2006, pp. 81–88.

[Mik+13] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781 (2013). URL: http://arxiv.org/abs/1301.3781.

[MSS13a] Thomas Mueller, Helmut Schmid, and Hinrich Schütze. "Efficient Higher-Order CRFs for Morphological Tagging". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle,

Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 322–332. URL: http://www.aclweb.org/anthology/D13-1032.

[MSS13b] Thomas Müller, Helmut Schmid, and Hinrich Schütze. "Efficient Higher-Order CRFs for Morphological Tagging". In: *In Proceedings of EMNLP* (2013).

[Niv03] Joakim Nivre. "An Efficient Algorithm for Projective Dependency Parsing". In: *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. 2003, pp. 149–160.

[Niv04] Joakim Nivre. "Incrementality in Deterministic Dependency Parsing". In: *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*. IncrementParsing '04. Barcelona, Spain: Association for Computational Linguistics, 2004, pp. 50–57. URL: http://dl.acm.org/citation.cfm?id=1613148.1613156.

[Niv08] Joakim Nivre. "Algorithms for Deterministic Incremental Dependency Parsing". In: *Comput. Linguist.* 34.4 (Dec. 2008), pp. 513–553. ISSN: 0891-2017. DOI: 10.1162/coli.07-056-R1-07-027. URL: http://dx.doi.org/10.1162/coli.07-056-R1-07-027.

[Niv+16a] Joakim Nivre et al. *Universal Dependencies 1.3*. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague. 2016. URL: http://hdl.handle.net/11234/1-1699.

[Niv+16b] Joakim Nivre et al. "Universal Dependencies v1: A Multilingual Treebank Collection". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Ed. by Nicoletta Calzolari (Conference Chair) et al. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016. ISBN: 978-2-9517408-9-1.

[SKT14] Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. "Introducing the spmrl 2014 shared task on parsing morphologically-rich languages". In: 2014, pp. 103–109.

[Sim+01] Khalil Sima'an et al. "Building a Tree-Bank of Modern Hebrew Text". In: *Traitment Automatique des Langues* 42.2 (2001).

[Smi+14] Peter Smit et al. "Morfessor 2.0: Toolkit for statistical morphological segmentation". In: *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, 2014, pp. 21–24. URL: http://aclweb.org/anthology/E14-2006.

[SHS16] Milan Straka, Jan Hajic, and Jana Straková. "UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Ed. by

Nicoletta Calzolari (Conference Chair) et al. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016. ISBN: 978-2-9517408-9-1.

[Tes59] L Tesnière. *Elements de syntaxe structurale*. Ed. by Editions Klincksieck. Editions Klincksieck, 1959.

[Tsa13] Reut Tsarfaty. "A Unified Morphosyntactic Scheme for Stanford Dependencies". In: *Proceedings of ACL*. 2013.

[TG08] Reut Tsarfaty and Yoav Goldberg. "Word-Based or Morpheme-Based? Annotation Strategies for Modern Hebrew Clitics". In: *Proceedings of LREC*. 2008.

[Tsa+10] Reut Tsarfaty, Djamé Seddah, et al. "Statistical Parsing of Morphologically Rich Languages (SPMRL): What, How and Whither". In: *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*. SPMRL '10. Los Angeles, California: Association for Computational Linguistics, 2010, pp. 1–12. URL: http://dl.acm.org/citation.cfm?id=1868771.1868772.

[Zha+14a] Meishan Zhang et al. "Character-Level Chinese Dependency Parsing". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 1326–1336. URL: http://www.aclweb.org/anthology/P14-1125.

[Zha+14b] Meishan Zhang et al. "Character-level chinese dependency parsing". In: *In Proceedings of the ACL*. 2014.

[ZC11] Yue Zhang and Stephen Clark. "Syntactic Processing Using the Generalized Perceptron and Beam Search". In: *Computational Linguistics* 37.1 (2011), pp. 105–151. DOI: 10.1162/coli_a_00037. URL: http://dx.doi.org/10.1162/coli_a_00037.

[ZN11] Yue Zhang and Joakim Nivre. "Transition-based Dependency Parsing with Rich Non-local Features". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 188–193. ISBN: 978-1-932432-88-6. URL: http://dl.acm.org/citation.cfm?id=2002736.2002777.

[Zhu+13] Muhua Zhu et al. "Fast and Accurate Shift-Reduce Constituent Parsing". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. The Association for Computer Linguistics, 2013, pp. 434–443. URL: http://aclweb.org/anthology/P/P13/P13-1043.pdf.

תקציר

תוצאות עדכניות לניתוח מורפולוגי ותחבירי אוטומטי של שפות עשירות מורפולוגית, כמו עברית, נמוכות מכדי לאפשר שימושים מעשיים, כמו אלה המיושמים לשפות שנחקרו רבות כמו אנגלית. מצב זה נובע מכך שתשתיות קיימות לניתוח מורפולוגיה ותחביר בנויות על הנחות יסוד המפרידות בין עיבוד מורפולוגי ותחבירי. אולם, הנחות אלה מופרות בהקשר של שפות עשירות מורפולוגיה.

בעבודה זו אנו מציגים תשתית העבר-צמצם (shift-reduce) כללית עם יישומים לניתוח מורפולוגי עצמאי מבוסס לקסיקון ומונע נתונים, מנתח (מפיג-עמימות) מורפולוגי עצמאי, מנתח תחבירי עצמאי, ומנתח מורפולוגי-תחבירי משולב. כל אחת ממשימות אלו מוגדרת על ידי מערכת מעברים (transition system), מודל תכונות רב-שפתי (cross linguistic feature model), ופונקציית תמחור (scoring function) גלובלית. אנו פותרים את בעיית הלמידה על ידי פרספטרון מוכלל (generalized perceptron), ומבצעים פיענוח יעיל על ידי חיפוש קרן (beam search).

אנו מציגים תוצאות עדכניות למשימות הניתוח המורפולוגי והתחבירי בנפרד, ומציגים לראשונה מערכת מבוססת מעברים לעיבוד מורפולוגי ועיבוד תלויות משותף לטקסטים בעברית. בנוסף, אנו מדגימים את השימושיות ורב-השפתיות של המנתח והמעבד המורפולוגי על ידי הפעלת ניתוח ועיבוד (הפגת עמימות) מורפולוגי על קבוצה של 48 שפות הלקוחה ממאגר בנק התלויות הבינלאומי (http://universaldependencies.org).

עבודה זו בוצעה בהדרכתם של דר' רעות צרפתי מהאוניברסיטה הפתוחה ופרופ' אריאל שמיר מבי"ס אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה.

המרכז הבינתחומי בהרצליה
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

עיבוד מורפולוגי ותחבירי משולב לשפות עשירות מורפולוגית במערכת מעברים

מאת
אמיר מור

עבודת תיזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc. במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

ספטמבר 2016