Joint Transition-Based Models for Morpho-Syntactic Parsing: Parsing Strategies for MRLs and a Case Study from Modern Hebrew

Amir More
Open University
Ra'anana, Israel
habeanf@gmail.com

Amit Seker
Open University
Ra'anana, Israel
amitse@openu.ac.il

Victoria Basmova
Open University
Ra'anana, Israel
vicbas@openu.ac.il

Reut Tsarfaty
Open University
Ra'anana, Israel
reutts@openu.ac.il

Abstract

In standard NLP pipelines, *morphological analysis and disambiguation* (MA&D) precedes syntactic and semantic downstream tasks. However, for languages with complex and ambiguous word-internal structure, known as *morphologically rich languages* (MRLs), it has been hypothesized that syntactic context may be crucial for accurate MA&D, and vice versa. In this work we empirically confirm this hypothesis for Modern Hebrew, an MRL with complex morphology and severe word-level ambiguity, in a novel transition-based framework. Specifically, we propose a joint morphosyntactic transition-based framework which formally unifies two distinct transition systems, morphological and syntactic, into a single transition-based system with joint training and joint inference. We empirically show that MA&D results obtained in the joint settings outperform MA&D results obtained by the respective standalone components, and that end-to-end parsing results obtained by our joint system present a new state of the art for Hebrew dependency parsing.

1 Introduction

NLP research in recent years has shown increasing interest in parsing typologically different languages, as evident, for instance, by the *universal dependencies*[1] initiative (Nivre et al., 2016). In particular, much attention is drawn to parsing *morphologically rich languages* (MRLs), which differ significantly from English in their structure and characteristics (Tsarfaty et al., 2010).

In MRLs, grammatical information, typically expressed using word order in English, is often manifested in the internally complex structure of the words. Words in MRLs may carry, in addition to lexical content, functional affixes and clitics that correspond to additional pieces of information. In Modern Hebrew, for example, the inflected verb "*ahbtih*"[2] (loved + 1pers.singular.past + 3pers.feminine.singular) corresponds to three different grammatical functions: the subject "I," the predicate "loved," and the direct object "her." Similarly, Spanish *dámelo* corresponds to a predicate, an indirect object, and a direct object, as in "give it to me." Thus, in MRLs, *morphological analysis* (MA) which translates raw space-delimited tokens to syntactically relevant "word" units is a necessary condition for any syntactic or semantic downstream task.

However, raw space-delimited tokens in MRLs are often highly ambiguous. In Hebrew, Arabic, and other Semitic languages, this situation is further complicated by fact that written texts lack diacritics. The Hebrew token *"fmn,"* for instance, may be read as the noun "oil," the adjective "fat," the verb "lubricated," the sequence

[1] http://universaldependencies.org/.
[2] Using the transliteration of Sima'an et al. (2001).

Transactions of the Association for Computational Linguistics, vol. 7, pp. 33–48, 2019. Action Editor: Masaaki Nagata.
Submission batch: 5/2018; Revision batch: 8/2018; Published 3/2019.

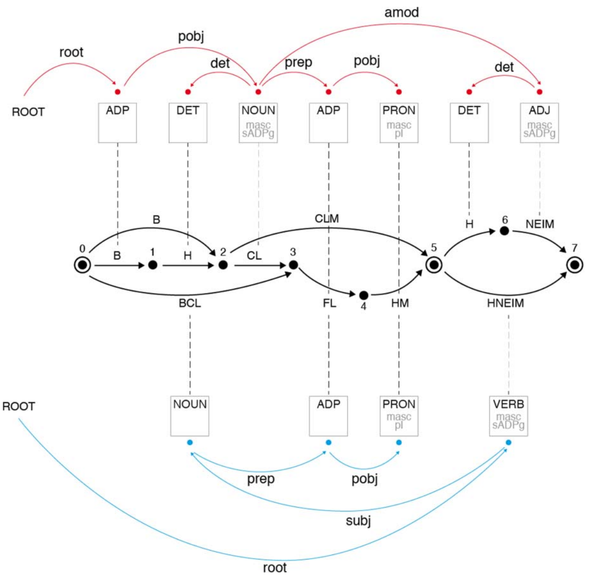

Figure 1: The morphological and syntactic interactions in the analysis of the Hebrew phrase "*bclm hneim*" according to the Hebrew SPMRL annotation.

"that"+"of," or the phrase "their"+"name," only one of which is relevant *in context*. This has clear ramifications for dependency parsing. Figure 1 shows a lattice that captures all possible analyses of the Hebrew phrase *"bclm hneim,"* literally: "in-the-shadow-of-them the-pleasant," translated "in their pleasant shadow." Each lattice arc corresponds to a potential node in a dependency tree. Dark circles mark morpheme boundaries, double circles mark token boundaries. The top tree depicts a correct syntactic analysis. In the bottom tree, incorrectly disambiguated tokens lead to a wrong syntactic analysis.

Previous dependency parsing evaluation campaigns (Buchholz and Marsi, 2006; Nivre et al., 2007) assumed that the correct *morphological analysis and disambiguation* (MA&D) of the input stream is known in advance. In realistic end-to-end parsing scenarios, however, this is of course not so. To overcome this, pipeline architectures where MA&D precedes parsing have been set up. These pipelines are suboptimal since they suffer from error propagation, and since local linear context available for automatic MA&D may be insufficient for accurate morphological disambiguation. For this, actual syntactic context may be required (Tsarfaty, 2006). To resolve this apparent loop, where *morphological analysis is required for syntactic parsing* and *syntactic analysis is required for morphological disambiguation*, Tsarfaty (2006) hypothesised that *joint* morphosyntactic parsing, where morphological information may assist syntactic disambiguation and vice versa, may be better suited.

This *joint morphosyntactic hypothesis* has been taken up and successfully confirmed in the context of phrase–structure parsing for Semitic languages (Goldberg and Tsarfaty, 2008; Cohen and Smith, 2007; Green and Manning, 2010). For dependency parsing, Bohnet and Nivre (2012) and Bohnet et al. (2013) present language-agnostic transition-based frameworks for jointly parsing and tagging input words, though without addressing the complex issue of retokenizing ambiguous input tokens. More recently, Seeker and Centinoglu (2015) presented a graph-based framework for lattice parsing of Turkish also covering morphological segmentation. Their system takes a "product of experts" approach wherein the morphological paths and dependency trees are handled via two distinct models (a linear model over bigrams for MD and an arc-factor model for dependencies), reaching agreement via a dual decomposition setup.

In this work, we present a novel, language-agnostic, transition-based framework for end-to-end morphosyntactic dependency parsing. The framework unifies a morphological and a syntactic component into a joint parser encompassing a single transition system, a single objective function, joint learning, and joint decoding. We apply this system to parsing Modern Hebrew and empirically confirm that predicting MA&D in the joint settings improves upon standalone MA&D, and upon recently reported Hebrew MA&D results. Our system further improves end-to-end dependency parsing results in comparison to existing state-of-the-art parsers in pipeline scenarios, it significantly outperforms the joint parser of Seeker and Centinoglu (2015), and it substantially outperforms the dependency parser of Goldberg and Elhadad (2010), so far considered the de facto standard for Hebrew dependency parsing.

The contribution of this paper is thus threefold. First, we define a language-agnostic *joint* morphosyntactic parser in a transition-based framework. Secondly, we empirically confirm that MA&D benefits from syntactic parsing, and in realistic end-to-end parsing scenarios, also vice versa. Finally, we present a new set of strong Hebrew end-to-end parsing results and deliver an open-source, language agnostic implementation of the joint parser, for further investigating joint morphosyntactic parsing strategies. This paper is organized as follows. In Section 2, we present our formal framework (2.1), morphological model (2.2), syntactic model (2.3), and joint framework (2.4). Sections 3 and 4 present our experiments and analysis, respectively. Section 5 discusses related and future work, and Section 6 concludes.

2 The Proposal: Transition-Based Joint Morpho-Syntactic Parsing

2.1 Formal Settings

We cast end-to-end morphosyntactic parsing as a structure prediction function $F : \mathcal{X} \rightarrow \mathcal{Y}$, where $x \in \mathcal{X}$ is a sequence of raw input tokens and $y \in \mathcal{Y}$ is a dependency representation where the nodes in the tree correspond to disambiguated morphosyntactic units we refer to as *morphemes*.[3]

We assume that F is realized in a transition-based framework augmented with the structure prediction method of Zhang and Clark (2011). We start off with a completely general definition of a transition system as a quadruple $S = (C, T, c_s, C_t)$, with C a set of configurations, T a set of transitions, c_s an initialization function, and $C_t \subset C$ a set of terminal configurations. We then define different instantiations of S for the different (*morphological, syntactic, morphosyntactic*) parsing tasks. In each instantiation, a transition sequence y for x is a sequence of configurations that are obtained by applying transitions $t_1 ... t_n \in T$ sequentially. That is, starting with an initial configuration $c_0 = c_s(x)$, we find $y = c_0, ..., c_n$ such that $c_{i+1} = t_{i+1}(c_i)$ and $c_n \in C_t$. Thus, each y depicts a sequence of decisions that constructs a valid analysis for x at the relevant linguistic level.

For each task we employ an objective function $F(x)$ as follows, where $GEN(x)$ holds all the transition sequences that generate relevant candidates:

$$\begin{aligned} F(x) &= argmax_{y \in GEN(x)} Score(y) \\ &= argmax_{y \in GEN(x)} \Phi(y) \cdot \vec{\omega} \\ &= argmax_{y \in GEN(x)} \sum_{c_j \in y} \sum_i \omega_i \phi_i(c_j). \end{aligned}$$

[3]In universal-dependencies terms, these are called *syntactic words* or *tree tokens*. In previous work on Hebrew parsing they are referred to as *morphological segments*.

To compute $Score(y)$, y is mapped to a global feature vector $\Phi(y)$ of size d multiplied by a weights vector $\vec{\omega}$ of the same size. The global feature vector $\Phi(y)$ consists of local feature vectors, each of which is defined via a set of functions $\{\phi_i : C \rightarrow N\}_{i=1}^{d}$ which count the occurrences of a prespecified pattern in a given configuration in y. Following Zhang and Clark (2011), we learn the weights vector $\vec{\omega} \in R^d$ via the *generalized perceptron* using the *early-update* averaged variant of Collins and Roark (2004).

Decoding is based on the *beam search* algorithm, where a number of high-scoring candidate sequences are maintained in the beam in order to mitigate irrecoverable prediction errors that characterize greedy search procedures. At each step, the transition system applies all transitions to all candidates, and keeps the *B* highest-scoring candidates. During learning, the perceptron algorithm iterates through a gold-annotated corpus. Each sentence is parsed (decoded) with the last known weights, and if the parsed result differs from the gold, the weights are updated. The learning is stopped when overfitting begins.

2.2 The Morphological Framework

Our departure point for *morphological disambiguation* (MD) is the transition system of More and Tsarfaty (2016), currently established as the state of the art for Hebrew MA&D.[4] The input to the system is a lattice L that captures the range of valid morphological analyses for the input tokens $x = x_1, \ldots, x_k$, as illustrated in the middle of Figure 1. The goal of the MD system is to select a sequence of contiguous arcs in L which represents the morphological disambiguation of x in context.

Formally, we define for each token x_i its *token-lattice* $L_i = MA(x_i)$ where each *lattice-arc* in L_i corresponds to a potential node in the dependency tree. Each lattice-arc has a *morphosyntactic representation* (MSR) which we define as a tuple $m = (b, e, f, t, g)$ with b and e the beginning and end indices in L, f a form, t a part-of-speech tag, and g a set of attribute:value grammatical properties. $L = MA(x)$ is the sentence lattice obtained by concatenating the token-lattices top to bottom $L = MA(x_1) \circ ... \circ MA(x_k)$. Now,

[4]For exposition of the evaluation, alternatives, and a cross-linguistic application, see More and Tsarfaty (2016).

L represents the full range of valid morphological analyses applicable to x.[5]

A configuration for any input x in the MD system consists of its sentence lattice $L = MA(x)$, an index n representing the internal node (dark circle) in L we are at, and an index i representing the 0-based current-token index (double circle) in L, while M is a set of disambiguated MSRs (selected arcs):

$$C_{md} = (L, n, i, M).$$

The initial configuration function c_s sets $L = MA(x)$, $n = bottom(L)$, $i = 0$, and $M = \emptyset$. For traversing the lattice L from bottom to top we define an open set of transitions using the MD_s transition template, with $s = (\_, \_, \_, t, g)$ specifying the delexicalized projection of (any) lattice arc (b, e, f, t, g).

$$MD_s : (L, p, i, M) \rightarrow (L, q, j, M \cup \{m\}). \quad (1)$$

This transition selects a single lattice arc at a given position. Now, if p is our current position in the lattice and $m = (p, q, f, t, g)$ is the selected arc, then $j = i + 1$ if q is at a token boundary (double circle) and $j = i$ if it is not.

The terminal configuration set is defined to be $C_t = \{(L, top(L), |x|, M)\}$ where $M = \{m_1, m_2, ..., m_l\}$ holds the fully disambiguated path of MSRs (selected arcs) through L.

In order to find this path in a data-driven fashion, we define a parametric model that scores all transitions that can be applied at each step. We define the properties f (form), t (pos tag), g (morphological attribute:value pairs), $path$ (the path in the previously disambiguated token-lattices), and $morphs$ (the set of outgoing morphemes of the current node) and we use unigram, bigram, and trigram combinations of these properties as features for the learning model.[6] Our beam search decoder then applies at each point in the lattice all possible transitions and selects the B-top scoring candidates at this point. Those that don't make the B mark, fall off the beam.

Importantly, $|M|$, the number of lattice arcs in the path at each stage, is unknown in advance, since different disambiguation decisions between token boundaries may end up with different path lengths. This can be seen in the lattice of Figure 1, where path lengths vary between 4–7 arcs. This is a thorny issue, because it violates a basic assumption of beam search decoding—that the number of transitions is a deterministic function of the input and is known in advance. Such length discrepancies may lead to preferring short sequences in the beam due to reaching the end goal early, or preferring long sequences, due to artificial inflation of scores based on the multitude of features.

To address this issue, we adopt the solution proposed by More and Tsarfaty (2016), employing a special transition *ENDTOKEN (ET)* given in (2) which explicitly increments i when reaching a token boundary in L.[7]

$$ET : (L, n, i, M) \rightarrow (L, n, i + 1, M). \quad (2)$$

Set aside from other transitions, ET has its own set of features (of size d'). Other than incrementing i, ET causes a re-ordering of candidates in the beam at each token boundary. More and Tsarfaty (2016) show that when using this anchor, the features of the ET transition provide a counterbalance to the effects of varied-length sequences and improve the accuracy of Hebrew MD.

An MD transition sequence thus becomes a union of disjoint sets of configurations $y = y_{md} \cup y_{et}$, and $Score(y)$ is as follows, where $\omega_j^{md}\phi^{md}$ score configurations resulting from MD transitions and likewise $\omega_j^{et}\phi^{et}$ for ET transitions:

$$Score(y) = \sum_{i=1}^{d} \omega_i^{md}\phi_i^{md}(y_{md}) + \sum_{j=1}^{d'} \omega_j^{et}\phi_j^{et}(y_{et})$$

$$= \sum_{c_k \in y_{md}} \sum_{i=1}^{d} \omega_i^{md}\phi_i^{md}(c_k) + \sum_{c_l \in y_{et}} \sum_{j=1}^{d'} \omega_j^{et}\phi_j^{et}(c_l). \quad (3)$$

2.3 The Syntactic Framework

Given a sequence of selected lattice arcs for the input sequence x, we can define the syntactic dependency representation for x as a dependency tree where each lattice arc corresponds to a node in

[5] As one of our reviewers pointed out, in the general case there may be a single lattice for the entire sequence, with no concatenation needed, as is the case in some Asian languages.

[6] The complete description of the feature model is provided by More (2016).

[7] ET kicks in only for variable length lattices. On token-lattices where all paths are of the same length, ET is skipped.

the dependency tree. Let R be a set of dependency types and let $M = m_1...m_l$ be the sequence of l arcs selected by the MD component.[8] We denote a dependency graph for the sequence $M = m_1...m_l$ as $G_M = (V_M, A_M)$, where V_M is a set of nodes corresponding to the arcs of M and $A_M \subseteq V_M \times R \times V_M$ is a set of labeled arcs between the elements of V_M.

A configuration of an arc system for a morphological sequence $M = m_1...m_n$ is a triplet where σ is a stack of morphemes $m_i \in V_S$, β is a buffer of morphemes $m_i \in V_S$, and A is a set of labeled dependency arcs $(m_i, r, m_j) \in V_S \times R \times V_S$.

$$C_{dep} = (\sigma, \beta, A).$$

A configuration represents a partial analysis of the input sentence, where the morphemes on the stack σ are partially processed morphemes, the morphemes in the buffer β are those waiting to be processed, and the arc set A represents a partially built dependency tree (Kübler et al., 2009, Chapter 3). Unless specified otherwise, the set of terminal configurations is $C_t = \{(\sigma, \beta, A)\}$ where $\beta = []$ and $|\sigma| = 1$.[9]

There are various options for defining transitions over such configurations in the dependency parsing literature for English. In particular, three transition systems have been successfully applied to English as well as other languages (cf. Ballesteros and Nivre [2016]):

Arc Standard: A straightforward method of bottom-up left-to-right incremental parsing as proposed in Nivre (2004). We assume the definition by Kübler et al. (2009).

Arc Eager: Following Abney and Johnson (1991), Arc Eager defines a variant of Arc Standard that allows to eagerly attach a right-dependent to its head while allowing more dependents to attach to it. We assume the definition by Kübler et al. (2009).

Arc (Z)Eager: In our reproduction of the state-of-the-art results presented by Zhang and Nivre (2011) for English, we discovered in the code a variant of Arc Eager that we call Arc (Z)Eager, which has interesting subtle variations from Arc Eager, including a second stack holding head nodes, and certain hard constraints on the application of several transitions.[10]

An empirical study by Nivre (2008) compares the performance of Arc Standard and Arc Eager for 13 languages, amongst them Arabic and Turkish, both considered MRLs with some degree of word-order freedom. For these languages, Arc Standard slightly outperformed Arc Eager. On a different but related note, our preliminary experiments on English and Hebrew show that the Arc ZEager variant always outperforms Arc Eager. However, the question which of the two, Arc-Standard or Arc-ZEager, will be more suited for parsing Hebrew, remains open for our empirical investigation in Section 3.

Defining Features. A significant contribution of Zhang and Nivre (2011) is their proposal of a set of *rich non-local features* (RNF) for Arc ZEager, adding higher-order information previously found only in graph-based parsers. To facilitate a fair comparison of Arc Standard to Arc (Z)Eager, we have to adapt the feature set of Zhang and Nivre (2011) to the different arc system (to the extent that this is possible), and to the different language type. In particular, the RNF set depends on word order, by encoding the arc direction explicitly. We address the order-dependence of RNF by defining a parallel set of features that is suitable for the more *flexible word order* in MRLs, and that is applicable to Arc-Standard. We call this feature set *rich linguistic features* (RLF). The essence of the two feature sets is the same, but we replace features relying on positions of nodes with features relying on the *labeled grammatical functions* of these nodes.[11]

To construct our features, we define properties that capture the linguistic information of *selectional preferences* and *subcategorization frames* (Tesnière, 1959; Chomsky, 1965). To capture the

[8] To avoid confusion between lattice arcs and dependency arcs, we refer to lattice arcs $m_i \in M$ as "morphemes."

[9] A transition system can introduce an artificial root node that can head any partial tree in the sentence. The root node allows for multiple partial trees (a forest) to be related only through the root node. We call transition systems with and without a root node root-full and root-less, respectively. 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 cases.

[10] For the documented list of deviations of Arc (Z)Eager from Arc Eager, consult More (2016).

[11] For the full list and detailed comparison of rich non-local features and rich linguistic features, consult More (2016).

distributional characterization of subcat frames, we define sf_p to be a multiset of part-of-speech tags of the dependents of a given head. To capture the *functional* characterization of subcat frames, we define the sf_f referring to the multiset of function labels of all dependents of a given head. For valency, we define the properties v_{sf}, referring to the number of dependents of a given head. For capturing *selectional preferences* in flexible word order environments, we define order-agnostic bilexical labeled-dependency features, generated separately for each dependent.

Finally, we augment syntactic features with morphological properties. Our augmentation operator allows for creating multiple instances of the same feature, with and without morphological properties.

2.4 The Joint Framework

Given our morphological and syntactic components, we seek an integration such that morphological information aids syntactic disambiguation and vice versa.

We propose to literally embed the two standalone configurations into a single configuration, and to apply transitions via a coherent logic we call a *strategy* that chooses which processor to apply at a given state.

Formally, let c_{md} and c_{dep} be MD and dependency parser configurations as defined in Sections 2.2 and 2.3, respectively. We define the joint configuration as follows:

$$c_j = (c_{md}, c_{dep}) = ((L, n, i, M), (\sigma, \beta, A)). \quad (4)$$

We initialize the embedded MD configuration c_{md} with the MD transition system initialization function, as defined in Section 2.2, but leave c_{dep} empty, with $\sigma = \beta = []$ empty stack and buffer. Also, as before in c_{dep}, $A = \emptyset$. A configuration c_j is terminal if and only if c_{md} and c_{dep} are both terminal configurations of their respective systems.

Let $\mathcal{T} = (T_{md}, T_{dep})$ be a pair of transition sets of the MD and dependency parsing transition systems, respectively, let $\mathcal{C} = \{C_{md}, C_{dep}\}$ hold the sets of possible non-terminal configurations, and let $\mathcal{C}_t = \{C_{t_{md}}, C_{t_{dep}}\}$ hold the respective sets of terminal configurations. A joint strategy is a function that, given a non-terminal joint configuration, chooses exactly one transition system to act on:

$$JOINT : \mathcal{C} \rightarrow \mathcal{T}. \quad (5)$$

The Pipeline Strategy. Our baseline morpho-syntactic parsing strategy is simply a pipeline that first applies the morphological component which selects the best output morpheme sequence, and then applies the dependency parser to it.

The MDFirst Strategy. If we seek to improve on the simplistic pipeline strategies, we first need to adjust our MD transition system such that its disambiguation decisions feed into the configuration of the dependency parser. We modify the MD_s transition as follows:

$$\begin{aligned} MD_s: \quad & ((L, n, i, M), (\sigma, \beta, A)) \rightarrow \\ & ((L, q, j, M \cup \{m\}), (\sigma, [m|\beta], A)). \end{aligned} \quad (6)$$

Now, a simple improvement upon the pipeline approach would be, rather than choosing just the top-scoring candidate of the MD component, passing all B candidates in the beam to the dependency component. We refer to this strategy as *MDFirst*:

$$\textit{MDFirst}((c_{md}, c_{dep}) \in \mathcal{C}) = \begin{cases} T_{md} & c_{md} \notin C_{t_{md}} \\ T_{dep} & \textit{otherwise}. \end{cases} \quad (7)$$

This simple extension offers the opportunity to maximize a single objective function, and to “re-rank” initially locally scored candidates if syntactic processing leads to a better MD result.

The ArcGreedy Strategy. 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 $\textit{ArcGreedy}_k$ strategies, in which we greedily choose to apply a syntactic transition if the dependency buffer β is populated by at least k morphemes, so that the syntactic processor may look k morphemes “forward” in order to predict its next transition.

$$\textit{ArcGreedy}_k(c_{md}, (\sigma, \beta, A)) = \begin{cases} T_m & \textit{if } |\beta| < k \\ T_d & \textit{otherwise}. \end{cases} \quad (8)$$

In contrast with the pipeline architecture, both *MDFirst* and *ArcGreedy*$_k$ perform joint morpho-syntactic parsing, in the sense that the framework aims to maximize a joint global score over both morphological and dependency transitions. This is formally depicted as follows in (9), where c_{md} and c_{et} are the resulting configurations of MD and ET transitions respectively, and c_{dep} are the resulting configurations of syntactic transitions:

$$
\begin{aligned}
Score_{Joint}(y) &= Score_{MD}(y) + Score_{Dep}(y) \qquad (9)\\
&= \sum_{i=1}^{d} \omega_i^{md}\phi_i^{md}(y_{md}) + \sum_{j=1}^{d'} \omega_j^{et}\phi_j^{et}(y_{et}) \\
&\quad + \sum_{r=1}^{d''} \omega_r^{dep}\phi_r^{dep}(y_{dep}) \\
&= \sum_{c_k \in y_{md}} \sum_{i=1}^{d} \omega_i^{md}\phi_i^{md}(c_k) + \sum_{c_l \in y_{et}} \sum_{j=1}^{d'} \omega_j^{et}\phi_j^{et}(c_l) \\
&\quad + \sum_{c_d \in y_{dep}} \sum_{r=1}^{d''} \omega_r^{dep}\phi_r^{dep}(c_d).
\end{aligned}
$$

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 information to interact frequently. So, syntax can affect the ordering of candidates during the parsing sequence, correcting local mistakes closer to where they occur.

3 Experiments

Goal: We aim to test the hypothesis that joint syntactic and morphological disambiguation is better than a pipeline by empirically comparing the *Pipeline, MDFirst* and *ArcGreedy*$_3$ parsing strategies in our unified transition-based morpho-syntactic framework.[12]

Data: We use the Modern Hebrew section of the SPMRL shared task (Seddah et al., 2014), derived from the Hebrew Unified-SD version of Tsarfaty (2013). For the purpose of this work, we harmonized the treebank annotation scheme with the annotation scheme of the lexical resources of Itai and Wintner (2008), and in particular the HEBLEX lexicon of Adler and Elhadad (2006). We use the standard *train/dev/test sets* split, train on the *train* set (5,000 sentences) with a detailed investigation on *dev* (500), and confirm our results on *test* (716).

Implementation: We implemented from scratch a fully integrated, transition-based, multilingual natural language processor, written in Go.[13] Our implementation uses a general purpose morphological analyzer, which for Hebrew is backed by the BGU HEBLEX lexicon (Adler and Elhadad, 2006). We implemented the morphological disambiguator, dependency parser, and joint integration strategies defined herein. We implemented and experimented with both the *Arc Standard* and *Arc ZEager* transition systems.[14]

Scenarios: In MRLs, out-of-vocabulary (OOV) tokens pose a great challenge to parsing. A raw token may have not been observed during training, even though all its morphemes have been observed in other contexts. To gauge the effect of such OOV items on the quality of Hebrew parses, we evaluate the system in two different scenarios. In the first, *infused* scenario, we verify that each lattice contains the gold morphological analysis. That is, if the gold path is not present in $L = MA(x)$ (hence, an OOV), we automatically infuse the gold path into L. We contrast this with *uninfused* scenarios, where we use a realistic morphological analyzer with its (incomplete) lexical coverage as is, compliant with Adler and Elhadad (2006).

Settings: In all experiments, we used a beam of size 64, which, in our preliminary experiments on *dev*, gave better results for the joint models than a beam of 32, and in any event no worse results than a beam of 128. To avoid both overfitting and underfitting, we define a stopping condition for the training procedure, which we test in each training iteration. During training, we use a sliding window of three iterations and select the first model that precedes two sequential scores-drop on dev.

For pipeline models, we test distinct stopping conditions for the morphological and the syntactic models, each based on its own standalone scores.

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

[13] `https://golang.org` by Google.

[14] Dispensing with Arc Eager, which underperformed Arc ZEager in all settings in all our preliminary investigations.

For joint models, we test the stopping condition with respect to a single overall dependency F_1 score, which we define shortly.

Evaluating Morphology: To evaluate *morphological disambiguation* (MD) results, we report the F_1 scores on the set of predicted morphemes versus gold-standard morphemes in the sentence. Formally, let M_p, M_g be sets of predicted and gold morphemes of the sentence, respectively. We define precision, recall, and F_1-scores as:

$$Pr = \frac{|M_p \cap M_g|}{|M_p|}; Re = \frac{|M_p \cap M_g|}{|M_g|}; F_1 = \frac{2 * Pr * Re}{Pr + Re}.$$

We report two different scores for each MA&D run, one for full MD including segmentation, tagging and morphological features (MD Full), and one for segmentation and tags only (MD POS).

Evaluating Dependencies: Evaluating *joint morpho-syntactic dependency parsing* performance is non-trivial, because the gold and parse trees may have a different number of nodes, which precludes the application of standard attachment scores; it suffices that an incorrect segmentation occurs early in the sequence, then off-by-one indices in the remainder of the sentence deem the rest of the arcs incorrect (Tsarfaty et al., 2012).

Let us illustrate this effect. Consider the Hebrew phrase "*bbit*" (translated "in the house") that appears as a single space-delimited token. Now consider the two following MD alternatives, with and without the Hebrew covert definite article. We also include here the indices of the disambiguated morphemes in their linear order:

> Gold MD: 1.*b*("in") 2.*h*("the") 3.*bit*("house")
> Predicted MD: 1.*b*("in") 2.*bit*("house").

Further assume that both the Gold and Predicted dependency trees contain the *correct* dependency arc between *b* ("in") and *bit* ("house") labeled *pobj*. In simple LAS terms, the arcs that would be compared for the purpose of evaluation are:

> Gold Dep: *pobj(1,3), det(3,2)*
> Predicted Dep: *pobj(1,2).*

So the *pobj* predicted arc will be considered an error, even though the relation between forms is correct, and accordingly both UAS and LAS will be 0.

To address this issue, we define an F_1 accuracy measure with respect to the *forms* of arc edges, rather than their node *indices*. Formally, let M_p be the predicted morphological disambiguation of x, and let A_p be the predicted dependency tree over M_p. Likewise, let M_g, A_g be the gold-standard morphological disambiguation and dependency tree of x. We now *replace* the index of each node in the arcs of A_p, A_g with the form of the corresponding morpheme in M_p and M_g. Let J_p, J_g be the form-based (rather than index-based) arcs of the predicted and gold representations of x. We report both labeled and unlabeled F_1 as:[15]

$$Pr = \frac{|J_p \cap J_g|}{|J_p|}; Re = \frac{|J_p \cap J_g|}{|J_g|}; F_1 = \frac{2 \times Pr \times Re}{Pr + Re}.$$

In our example, the revised arcs will now be:

> Gold Dep: *pobj(b,bit),det(bit,h)*
> Predicted Dep: *pobj(b,bit).*

Now, the parser will be credited for identifying the *pobj* arc correctly, as desired, and the dependency scores will be: $Pr = 1, Re = 0.5$, and $F_1 = 0.67$.

Results: Tables 1–4 present our morpho-syntactic parsing results for each of our different systems in all, *pipeline* and *joint*, strategies. We report F_1 scores, both MD Full and MD POS for morphological disambiguation (MD), and both unlabeled and labeled F_1 scores for the dependency trees (Dep). Tables 1 and 3 present results on the Modern Hebrew *dev set*, and Tables 2 and 4 confirm our results on the *test* set.

Table 1 presents parsing results for *infused* morphological lattices; that is, ambiguous MA lattices that are guaranteed to also include the correct MD path in them. In these experiments, we see that MD results in joint parsing strategies (MDFirst, ArcGreedy) always improve upon the MD standalone/pipeline results. In particular, all MD results across the joint strategies are very close. We observe only a minor advantage for Arc-Zeager over Arc-Standard for both joint strategies. This increase in MD accuracy unfortunately comes at the expense of syntax, where we observe a slight drop (up to 0.5 point in [un]labeled F_1) when switching from pipeline to joint strategies.

We confirm this trend on the test set in Table 2, where we use the same models in the *infused* settings to parse the standard *test set*. For MD, all

15 This is effectively equivalent to the F_1 metric in Goldberg and Elhadad (2011) and Seeker and Centinoglu (2015).

Strategy	System	MD F_1 Full/POS	Dep F_1 Un/labeled
Standalone	M&T 2016	93.32/94.09	n/a / n/a
Pipeline	Standard	93.32/94.09	80.44/73.86
Pipeline	ZEager	93.32/94.09	**80.82/74.28**
MDFirst	Standard	94.39/95.19	80.32/73.22
MDFirst	ZEager	**94.71/95.49**	80.50/73.53
ArcGreedy$_3$	Standard	94.56/95.36	80.60/73.43
ArcGreedy$_3$	ZEager	**94.62/95.45**	80.73/73.89

Table 1: Joint morpho-syntactic parsing of the Modern Hebrew *dev set* with *infused* morphological lattices.

Strategy	System	MD F_1 Full/POS	Dep F_1 Un/labeled
Standalone	M&T 2016	92.09/92.92	n/a / n/a
Pipeline	Standard	92.09/92.92	78.51/73.13
Pipeline	ZEager	92.09/92.92	**78.59/73.22**
MDFirst	Standard	92.70/93.66	77.32/70.57
MDFirst	ZEager	**92.90/93.92**	77.33/70.62
ArcGreedy$_3$	Standard	**92.88/93.85**	77.73/70.69
ArcGreedy$_3$	ZEager	92.60/93.67	77.70/70.96

Table 2: Joint morpho-syntactic parsing of the Modern Hebrew *test set* with *infused* morphological lattices.

Strategy	System	MD F_1 Full/POS	Dep F_1 Un/labeled
Standalone	M&T 2016	88.57/90.83	n/a / n/a
Pipeline	Standard	88.57/90.83	77.45/70.74
Pipeline	ZEager	88.57/90.83	77.56/70.85
MDFirst	Standard	89.48/91.89	78.30/71.21
MDFirst	ZEager	**89.83/92.34**	78.86/71.91
ArcGreedy$_3$	Standard	89.67/92.26	78.76/71.80
ArcGreedy$_3$	ZEager	**89.81/92.36**	**79.07/72.39**

Table 3: Joint morpho-syntactic parsing of the Modern Hebrew *dev set* with *uninfused* (realistic) lattices.

Strategy	System	MD F_1 Full/POS	Dep F_1 Un/labeled
Standalone	M&T 2016	84.89/87.53	n/a / n/a
Pipeline	Standard	84.89/87.53	73.70/67.83
Pipeline	ZEager	84.89/87.53	74.43/68.33
MDFirst	Standard	85.79/88.81	75.49/**69.41**
MDFirst	ZEager	**85.92/89.02**	75.37/69.28
ArcGreedy$_3$	Standard	**85.98/89.08**	**75.73**/69.23
ArcGreedy$_3$	ZEager	85.85/88.92	75.30/69.13

Table 4: Joint morpho-syntactic parsing of the Modern Hebrew *test set* with *uninfused* (realistic) lattices.

joint results are better than the respective pipelines (although now Arc-Standard slightly improves upon Arc-Zeager in the ArcGreedy strategy), while dependency parsing results drop in joint scenarios (a slightly larger drop than on dev).

Tables 3 and 4 present parsing results for the more interesting scenario, a realistic parsing scenario where we use *uninfused* lattices—ambigous lattices obtained by an existing broad-coverage morphological analyzer, which are not (and cannot be) guaranteed to always also include the correct path. As expected, on both the *dev set* (Table 3) and *test set* (Table 4), the results drop relative to the respective *infused* scenarios (Tables 1 and 2, respectively), as some elements from the correct path and tree are no longer reachable within the search space. At the same time, it is interesting to observe that for both *dev* and *test*, all MD scores (Full/POS) *as well as* dependency scores (un/labeled) are better in joint parsing. The specific differences between the joint strategies and transition systems do not matter very much—the robust empirical trend is that switching from *pipeline* to *joint* improves both MD and dependency parsing performance.

It is interesting to inquire why in the *infused* scenario, on both *dev* and *test*, dependency parsing results in the joint strategies drop relative to the respective pipelines. At it turns out, in case the correct analysis of a rare (OOV) token has been injected artificially into the lattice, training on these lattices may turn out to be misleading. Injecting a correct but rare MSR may lead to an artificial "certainty" as to its appropriate syntactic context. Then, if the parser does not apply robust statistics on the general behavior of rare/OOV items in different syntactic contexts (as would be the case in *joint uninfused* scenarios), selecting the injected MD may lead to a wrong syntactic decision.

The main message coming out of our experiments is that joint morphological disambiguation and syntactic parsing in this transition-based framework is preferred to pipeline settings, in line with the hypothesis that syntactic information aids morphological disambiguation. Furthermore, it is reassuring to observe that when parsing *uninfused* lattices, as in the more realistic scenario, dependency parsing results improve upon pipeline scenarios, corroborating the findings of Seeker and Centinoglu (2015) in graph-based frameworks and of Cohen and Smith (2007) and Goldberg and Tsarfaty (2008) in phrase–structure parsing.

End-to-End Parsing Performance: To put our end-to-end system performance in context, Tables 5 and 6 present our best results for dependency parsing in a pipeline architecture, assuming gold morphology, on the *dev set* and

	System	UAS/LAS
Previous SOTA	G&E 2010 MST	84.4/—
Previous SOTA	G&E 2010 Malt	80.7/—
Previous SOTA	G&E 2010 EasyFirst	84.2/—
This Work	Pipeline Standard	86.75/80.46
This Work	Pipeline ZEager	**87.22/81.24**

Table 5: **Comparing to previous pipelines:** Parsing results with gold morphology on the Hebrew *dev set*.

	System	UAS/LAS
SPMRL 2013	MALT OPTIMER	84.9/80.0
SPMRL 2013	ALPAGE DYALOG	86.2/80.7
SPMRL 2013	IMS-SZEGED-CIS	**88.9/83.8**
SPMRL 2014	MALT	81.36/76.61
SPMRL 2014	LORIA	82.73/75.24
SPMRL 2014	ICT	88.08/81.37
This Work	Pipeline Standard	85.94/80.70
This Work	Pipeline ZEager	86.05/80.92

Table 6: **Comparing to previous pipelines:** Parsing results with gold morphology on the Hebrew *test set*.

the *test set,* respectively. We compare these results with studies that parsed the same data sets. As Table 5 shows, our parser significantly outperforms the state-of-the-art parser by Goldberg and Elhadad (2011), so far considered the de facto standard for Hebrew parsing.[16] As shown in Table 6, the parser also outperforms the results reported by most (though not all) SPMRL shared tasks participants, using the same data and same split.

Such gold morphology settings are of course not suited for realistic parsing scenarios. So, in Table 7 we compare our best end-to-end parsing results to the most recent dependency parsing results in realistic scenarios on the same data (by (Seeker and Centinoglu 2015). Here our best pipeline and joint systems outperform the previously reported pipeline and joint results, thus presenting a new state of the art for Hebrew dependency parsing. Moreover, these results are obtained within a unified formal framework in a single "all-included" implementation, providing a further practical advantage of not having to maintain and train separate standalone components.[17]

[16]Goldberg and Elhadad (2010) report only UAS, only *dev*.

[17]Our implementation, models, and data are publicly available via `https://github.com/OnlpLab/yap`. We also provide a web demo of Hebrew raw-to-dependency parsing `http://onlp.openu.org.il/`.

	System	Un/labeled F_1
Previous SOTA	S&C 2015 Mate	71.11/65.69
Previous SOTA	S&C 2015 Turbo	70.86/65.66
Previous SOTA	S&C 2015 Pipeline	71.30/66.33
Previous SOTA	S&C 2015 Joint	71.52/66.68
This Work	Pipeline Standard	73.70/67.83
This Work	Pipeline ZEager	74.43/68.33
This Work	Joint Standard	**75.73**/69.23
This Work	Joint ZEager	75.49/**69.41**

Table 7: **Comparing to previous SOTA:** Parsing results in predicted morphology, *uninfused* lattices on *test set*. S&C 2015 refers to Seeker and Centinoglu (2015).

4 Qualitative Error Analysis

To shed more light on the particular ways in which the joint system improves performance over the pipeline, we conducted a qualitative error analysis in 100 sentences from the Modern Hebrew standard dev set, when parsed in the more realistic *uninfused* scenario. More concretely, we sampled 100 sentences from our parsed corpus and a linguist manually assigned each error to one of 10 linguistic categories. We then clustered the categories into four different types.

- TYPE 1 errors include true semantic ambiguity, where additional semantic and world knowledge is required for disambiguation.
- TYPE 2 errors include categories that transcend different levels of linguistic structure, for example, when morphological segmentation errors affect syntactic disambiguation.
- TYPE 3 errors include parsing errors that stem from idiosyncrasies of the data and peculiarities of the SPMRL annotation scheme,
- TYPE 4 (other) errors include parse errors that pertain to linguistic structures that characterize Semitic phenomena.

Table 8 shows, for each error category, the number (and percentage) of occurrences of that error in the pipeline versus joint settings. The most outcome is that the type that shows the largest decrease in joint scenarios relative to pipeline scenarios belongs to TYPE 2, reflecting phenomena directly related to the morpho-syntactic interface. Moreover, we also see a decrease in the errors concerned with the lexico-syntactic interface (e.g., solving PP attachment

System Strategy Morphology	Zeager Pipeline Gold	Zeager Pipeline Predicted	Zeager ArcGreedy Joint
Total Number of Errors	390	641	546
TYPE 1:			
Could be Considered Correct	64 (16.4)	62 (9.6)	17 (3.1)
Difficult Clause Attachment	62 (15.8)	103 (18.8)	109 (19.9)
Difficult PP Attachment	**27 (6.9)**	**41 (6.4)**	**30 (5.4)**
TYPE 2:			
Wrong arc due to Seg/Tag Error in the focus word	**0**	**106 (16.5)**	**56 (10.2)**
Wrong arc due to Seg/Tag Error in other words	**0**	**58 (9.0)**	**37 (6.7)**
Wrong arc label due to tag error	0	23 (3.6)	31 (5.6)
TYPE 3:			
Gold Standard is Wrong	51 (13.0)	48 (7.5)	44 (8.0)
Trainset is Inconsistent	68 (17.4)	60 (9.3)	69 (12.6)
Prediction is Underspecified	65 (16.6)	76 (11.8)	83 (15.2)
TYPE 4:			
Other	53 (13.6)	64 (9.9)	70 (12.8)

Table 8: **Qualitative error analysis:** The number (percentage) of error patterns of Arc-Zeager in pipeline/joint scenarios, on a sample of 100 sentences from the *dev set*, in *uninfused* lattices settings.

ambiguity), which turn out to also benefit from the joint settings. With the other types of errors, there is no clear advantage for joint parsing, and we would not expect one. TYPE 3 errors have to do with train-set inconsistencies, under-specification, or errors in the gold trees. TYPE 4 errors stem from linguistic phenomena which appear harder to disambiguate, and they are equally difficult across scenarios.

5 Related and Future Work

Monolingual MA&D for Modern Hebrew has been previously addressed in standalone settings using Hidden Markov Models (Bar-haim et al., 2008; Adler, 2007). While these results are adequate for some downstream applications, using Adler's MA&D for dependency parsing, for instance, significantly harms parsing performance (Goldberg and Elhadad, 2010). More recently, More and Tsarfaty (2016) presented a standalone transition-based MA&D which jointly solves morphological segmentation, tagging, and feature assignment, presenting new state-of-the-art Hebrew MA&D, providing the starting point for our study.

In terms of end-to-end dependency parsing for Hebrew, Goldberg and Elhadad (2010) were the first to evaluate the impact of predicted morphology compared to gold morphology across different (transition-based, graph-based, easy-first) frameworks. They demonstrated a significant loss in accuracy for all models in predicted morphology settings, and concluded with a suggestion to attempt joint processing. Recently, Straka et al. (2016) presented UDPipe, a toolkit with standalone components for morphological analysis, segmentation, tagging, features assignment, and dependency parsing—again using a pipeline architecture, with no way of interleaving the different decisions, as we strive to do here. This work aims to cover all stages of UDPipe, but within a *joint* architecture, allowing the use of information from any layer when disambiguating another.

Joint morphological and syntactic processing has been addressed in the context of phrase-structure parsing for Semitic languages, showing empirical advantages over pipeline architectures (Goldberg and Tsarfaty, 2008; Cohen and Smith, 2007; Green and Manning, 2010). In the context of dependency parsing, Bohnet and Nivre (2012) and Bohnet et al. (2013) integrated tagging and dependency parsing, improving state-of-the-art accuracy for a set of typologically different languages. Andor et al. (2016) use the joint transition system proposed by Bohnet and Nivre (2012), and improve it using a globally

TYPE	Error	Explanation
1	Could be considered correct	Cases of true semantic ambiguity. Both analyses could be considered correct. For example, in the phrase *mrkz kwx erbi* the adjective *erbi* ("arab") modifies *mrkz* ("center") in gold. The parser attaches it to *kwx* ("force"). Both could be correct.
	Clause attachment	In complex sentences with multiple clauses or coordinated structures, the parser often identifies the conjunctions and the predicates correctly, but makes mistakes in connecting clauses. Semantic or world knowledge is required for disambiguation.
	PP attachment	Semantic or world knowledge is also often required to determine PP attachment. For example, in the clause *kdi lmnwe hedptm el ewbdim ifralim* the parser attaches the PP *el ewbdim ifralim* ("over Israeli workers") to the verb *lmnew* ("to prevent") rather than to the required noun *hedptm* ("their preference").
2	Seg/Tag err in focus word	Incorrect segmentation of a token may lead to missing or incorrect dependency heads. For example, the parser analyses the token *bqrb* as a single word (a preposition, "near") while in the gold standard it is segmented into three words *b* + *h* + *qrb* (preposition + def + noun, "in the battle"). This leads to missing dependency heads.
	Seg/Tag err in other word	Incorrect segmentation of a token may also lead to an incorrect dependent. For example, in the phrase *bqrb mgnnh* the parser analyses the PP *b* + *qrb* (preposition + noun, "in battle") as a single word *bkrb* (preposition, "near"). As a result, the word *mgnnh* (defence) is labeled object of a preposition (pobj) rather than a genitive object of a construct-state noun (gobj).
	Label err due to tagging err	Incorrect tag prediction may lead to an apropriate yet incorrect arc label. For example, in the phrase *amcei xi lhpgnwt* ("living means for demonstrations") the parser tags the adjective *xi* ("living") as a noun instead of an adjective, which is why it attaches *xi* as gobj (genitive object) to "means" rather than as amod.
3	Gold is wrong	The analysis in gold is wrong, while the analysis provided by the parser is correct. For example, in the phrase *w+b+silwp ewbdwt* ("and in distortion of facts"), the conjunction marker *w* is labeled comp in gold while the parser correctly picks cc.
	Train is inconsistent	(a) Multiple labels are used for the same type of dependencies. For example, prepmod and comp are both used in the train set for prepositional complements and prepositional modifiers without a clear distinction. (b) Identical structures are analyzed in different ways. For example, in the train set there are different structures used for the same type of partitive construction. In both (a) and (b), the predicted analyses might likewise be inconsistent and arbitrary.
	Label underspecified	The label dep is used instead of different types of dependencies in gold. In several cases the test set uses more specific labels where the parser predicts dep, and vice versa.
4	Other	There is a smaller amount of errors that involve linguistic structures that reflect particular Semitic phenomena. For example: (a) Indefinite objects in Hebrew are not case marked, so are sometimes mislabeled as subject due to flexible word order patterns and object pre-posing. (b) Construct-state nouns may be analysed as names and vice versa. Since Hebrew lacks capitalization, Hebrew names very often string-match common nouns. (c) Adjective attachment errors inside construct-state nouns. For example, in the phrase *hjlt qnswt kbdim* the parser attaches the adjective *kbdim* ("heavy") to the construct-state noun *hjlt* ("imposition-of") instead of attaching it to the genitive object *qnswt* ("fines").

Table 9: Qualitative error analysis: Explanation and illustrations.

normalized neural network. These systems address joint morpho-syntactic analysis for disambiguated words, but without addressing the issue of segmenting and disambiguating raw input tokens.

Seeker and Centinoglu (2015) explore the idea of joint morphological and syntactic parsing, including morphological segmentation, in a graph-based framework. Their system integrates two standalone components that reach agreement via a dual-decomposition setup. However, they report suboptimal performance on the standard Hebrew benchmark. For various Chinese parsing tasks, joint systems for word segmentation and syntactic parsing have been shown to outperform pipeline settings (Li et al., 2011; Zhang et al., 2014), but these systems assume transitions over equal-length character-based sequences, and thus they are not applicable to the setup of variable-length lattice paths, as demonstrated in Figure 1.

With the surge of interest in deep learning for NLP (Goldberg, 2016), research in dependency parsing seeks to replace engineered feature models with neural networks that induce a model automatically (Chen and Manning, 2014; Zhou et al., 2015; Andor et al., 2016). Furthermore, the concept of word embedding introduced by Mikolov et al. (2013) allows for words to have vector representations, such that syntactic and semantic similarities are embodied in the vector space. However, these kinds of architectures are not immediately applicable to parsing Hebrew and other MRLs. Pretraining word embeddings is non-trivial for ambiguous input tokens, unless resorting to pipeline "segmentation-first" scenarios. Similarly, parsing architectures based in RNNs require morphologically disambiguated forms as input, which prevents syntax from improving morphological disambiguation, as we argue for here.

In the future, we intend to augment the architecture we present here with neural network models for both the morphological and syntactic models, in a way that would allow them to effectively interact and affect one another, in the hope to lead towards further improvements in both tasks.

6 Conclusion

We present a novel joint transition-based framework for morpho-syntactic parsing, designed to solve end-to-end dependency parsing in realistic scenarios. We consider the properties of MRLs and directly address the disambiguation of raw input tokens exploiting larger syntactic contexts. We apply this system to Modern Hebrew, and our empirical results support the long-standing conjecture that MA&D can greatly benefit from syntactic parsing. We present a new set of state-of-the-art Hebrew parsing results, in both pipeline and joint scenarios, which then serve as a strong baseline for exploring future *neural* joint morpho-syntactic architectures that would potentially improve performance on both tasks.

Acknowledgments

We thank Joakim Nivre, Yue Zhang, and Yoav Goldberg for comments and suggestions, and four anonymous reviewers for their comments on earlier drafts. We further thank Shoval Sadde, Yochay Gurman, and Dan Bareket from the ONLP Lab at the Open University of Israel for critical discussion of the data and the empirical results. This research was supported by a Starting Research Grant from the European Research Council (ERC-StG-677352), and a grant from the Israel Science Foundation (ISF-1739/26), for which we are grateful.

References

Steven P. Abney and Mark Johnson. 1991. Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

Meni Adler. 2007. *Hebrew Morphological Disambiguation: An Unsupervised Stochastic Word-Based Approach*. Ph.D. thesis, Ben-Gurion University of the Negev, Israel.

Meni Adler and Michael Elhadad. 2006. An unsupervised morpheme-based HMM for Hebrew morphological disambiguation. In *Proceedings of ACL*. Association for Computer Linguistics.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the ACL*. Association for Computational Linguistics.

Miguel Ballesteros and Joakim Nivre. 2016. Maltoptimizer: Fast and effective parser optimization. *Natural Language Engineering*, 22(2):187–213.

Roy Bar-haim, Khalil Sima'an, and Yoad Winter. 2008. Part-of-speech tagging of Modern Hebrew text. *Natural Language Engineering*, 14(2): 223–251.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1455–1465. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajic. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL-X*, pages 149–164. Association for Computational Linguistics.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. The MIT Press, Cambridge, MA.

Shay B. Cohen and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proceedings of EMNLP*.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04. Association for Computational Linguistics.

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57(1):345–420.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proceedings of ACL*. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2011. Joint Hebrew segmentation and parsing using a PCFGLA lattice parser. In *Proceedings of ACL*. Association for Computational Linguistics.

Yoav Goldberg and Reut Tsarfaty. 2008. A single framework for joint morphological segmentation and syntactic parsing. In *Proceedings of ACL*. Association for Computational Linguistics.

Spence Green and Christopher D. Manning. 2010. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 394–402. Association for Computational Linguistics.

Alon Itai and Shuly Wintner. 2008. Language resources for Hebrew. *Language Resources and Evaluation*, 42(1):75–98.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Number 2 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for chinese POS tagging and dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1180–1191. Association for Computational Linguistics.

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

Amir More. 2016. Joint morphological and syntactic disambiguation in a transition-based framework. MSc. Thesis. IDC Herzliya Israel.

Amir More and Reut Tsarfaty. 2016. Data-driven morphological analysis and disambiguation for morphologically rich languages and universal dependencies. In *Proceedings of COLING*, pages 337–348.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

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

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Yevgeni Berzak, Riyaz Ahmad Bhat, Cristina Bosco, Gosse Bouma, Sam Bowman, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, ÇağrÄś Çöltekin, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Sebastian Garza, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gokirmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Normunds Grūzītis, Bruno Guillaume, Jan Hajič, Dag Haug, Barbora Hladká, Radu Ion, Elena Irimia, Anders Johannsen, Hüner KaşÄśkara, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Jessica Kenney, Simon Krek, Veronika Laippala, Lucia Lam, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Keiko Sophie Mori, Shunsuke Mori, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Vitaly Nikolaev, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Loganathan Ramasamy, Laura Rituma, Rudolf Rosa, Shadi Saleh, Baiba Saulīte, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Carolyn Spadine, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jing Xian Wang, Jonathan North Washington, Zdeněk, Žabokrtský, Daniel Zeman, and Hanzhi Zhu. 2016. Universal dependencies 1.3. LINDAT/CLARIN Digital Library at Institute of Formal and Applied Linguistics, Charles University in Prague.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932. Association for Computational Linguistics.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109. Dublin City University, Dublin, Ireland.

Wolfgang Seeker and Ozlem Centinoglu. 2015. A graph-based lattice dependency parser for joint morphological segmentation and syntactic analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373.

Khalil Sima'an, Alon Itai, Yoad Winter, Alon Altman, and N. Nativ. 2001. Building a tree-bank of Modern Hebrew text. *Traitment Automatique des Langues*, 42(2).

Milan Straka, Jan Hajic, and Jana Straková. 2016. 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)*. Paris, France. European Language Resources Association (ELRA).

Lucien Tesnière. 1959. *Elements de Syntaxe Structurale*, Editions Klincksieck.

Reut Tsarfaty. 2006. Integrated morphological and syntactic disambiguation for modern Hebrew. In *Proceedings ACL-CoLing Student Research Workshop*, pages 49–54. ACL.

Reut Tsarfaty. 2013. A unified morphosyntactic scheme for stanford dependencies. In *Proceedings of ACL*. Association for Computational Linguistics.

Reut Tsarfaty, Joakim Nivre, and Evelina Andersson. 2012. Joint evaluation of morphological segmentation and syntactic parsing. In *Proceedings of ACL*, ACL '12, pages 6–10.

Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kübler, Marie Candito, Jennifer Foster, Yannick Versley, Ines Rehbein, and Lamia Tounsi. 2010. Statistical parsing of morphologically rich languages (SPMRL): What, how and whither. In *Proceedings of the NAACL SPMRL Workshop*, SPMRL '10, pages 1–12. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level chinese dependency parsing. In *Proceedings of the ACL*.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the ACL*, HLT '11, pages 188–193. ACL.

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