What's Wrong with Hebrew NLP? And How to Make it Right

Reut Tsarfaty **Amit Seker** **Shoval Sadde** **Stav Klein**
Open University of Israel, University Road 1, Ra'anana, Israel
{reutts,shovalsa,amitse,stavkl}@openu.ac.il

Abstract

For languages with simple morphology, such as English, automatic annotation pipelines such as spaCy or Stanford's CoreNLP successfully serve AI/DS projects in academia and the industry. For many morphologically-rich languages (MRLs), similar pipelines show sub-optimal performance that limits their applicability for text analysis in research and commerical use. The suboptimal performance is mainly due to errors in early morphological disambiguation decisions, which cannot be recovered later in the pipeline, yielding incoherent annotations on the whole. In this paper we describe the design and use of the ONLP suite, a joint morpho-syntactic parsing framework for processing Modern Hebrew texts. The joint inference over morphology and syntax substantially limits error propagation, and leads to high accuracy. ONLP provides rich and expressive output which already serves diverse academic and commercial needs. Its accompanying demo further serves educational activities, introducing Hebrew NLP intricacies to researchers and non-researchers alike.

1 Introduction

NLP pipelines for the automatic annotation of unstructured texts are at the core of language technology applications for Data Science, Text Analytics and Artificial Intelligence. For English, annotation pipelines such as spaCy (Honnibal and Montani, 2017) or Stanford's CoreNLP (Manning et al., 2014) successfully deliver the ability to automatically annotate unstructured texts with their underlying linguistic structures, including: Part-of-Speech (POS) Tags, Morphological Features, Dependency Relations, Named Entities, and so on. These annotations serve research labs, non-profit organizations and commercial endeavors in their quest to *make sense* of the vast amount of unstructured data available to them.

Universal processing pipelines such as UDPipe (Straka et al., 2016) aim to serve a range of other languages, but unfortunately, their performance on many morphologically rich languages (MRLs) (Tsarfaty et al., 2010), and in particular Semitic languages, is not on a par with their performance on English. This, in turn, greatly limits their applicability for further research and commercial use. The main reason for this sub-optimal performance on Semitic languages is that the *pipeline* design inherent in these frameworks is inappropriate for languages that exhibit extreme morphological ambiguity in their input stream. This is because errors made in morphological segmentation and disambiguation early on, jeopardize the system accuracy down the pipeline. For Hebrew, this performance gap has long been a show-stopper for advancing Language Technology and Artificial Intelligence for the Hebrew-speaking community. With this contribution, we aim to remedy this situation.

In this paper we describe the design and use of the ONLP system, a *joint* morphological-syntactic parsing framework for processing the Semitic language Modren Hebrew (Henceforth, Hebrew). The system is accurate, efficient, and provides rich and expressive output including: Segmentation, POS tags, Lemmas, Features and Labeled Dependencies. The *joint* training and inference over the different layers substantially limits error propagation, and leads in turn to speed and high accuracy. Among the technical advantages of the ONLP suite are its open license, an easy 3-step installation, and a single package with all elements included — no need to train or maintain individual components separately. The ONLP suite already serves academic and commercial projects in diverse domains. Its accompanying online demo has further proved valuable for educational purposes, exposing CS/NLP and non-CS researchers and engineers to the intricacies of Semitic NLP.

Proceedings of the 2019 EMNLP and the 9th IJCNLP (System Demonstrations), pages 259–264
Hong Kong, China, November 3 – 7, 2019.

2 The Linguistic Challenge

In morphologically-rich languages (MRLs), each input token may consist of multiple lexical and functional units (henceforth, *morphemes*), each of which serves a particular role in the overall syntactic or semantic representation. In Hebrew, for example, the token 'וכשמהמעבדה' corresponds to five word tokens in English, each of which carrying its distinct role: 'ו' (and, CC), 'כש' (when, REL), 'מְ' (from, IN), 'ה' (the, DT), 'מעבדה' (lab, NN).[1] This means that in order to process Hebrew texts, one first needs to segment the Hebrew tokens into their constituting morphemes. At the same time, Hebrew raw tokens are highly ambiguous. A token such as: 'הקפה' may be interpreted as 'הקפה' (orbit, NN), 'ה' + 'קפה' (the+coffee, DT+NN), or 'הקף' + 'של' + 'היא' (perimeter of her, NN+POSS+PRP), etc. This is further complicated by the lack of diacritics in standardized texts, meaning that most vowels are not present, and thus out of context no reading is a-priory more likely than the others. Only *in context* the correct interpretation and segmentation become apparent.

These facts create an apparent loop in the design of NLP pipelines for Hebrew: *syntactic parsing requires morphological disambiguation – but morphological disambiguation requires syntactic context*. This apparent loop has called for the development of *joint systems* rather than *pipelines*, for Semitic languages processing (Tsarfaty, 2006; Green and Manning, 2010). This joint hypothesis has proven useful for Hebrew and Arabic phrase-structure parsing (Goldberg and Tsarfaty, 2008; Green and Manning, 2010; Goldberg and Elhadad, 2011). The ONLP suite is a *dependency-based* parsing framework implementing this joint hypothesis, over the entire morpho-syntactic search-space, as depicted in Figure 1 (More et al., 2019).

3 The Architectural Design

The core of ONLP is *YAP (Yet Another Parser)*, a morpho-syntactic parser for morphological and syntactic analysis of Hebrew Texts. YAP re-implements and extends the structure-prediction framework of Zhang and Clark (2011). We describe *YAP* in detail in More and Tsarfaty (2016) and More et al. (2019). Here we only provide a bird's eye view of the architecture.

[1] We use the annotation conventions of Sima'an et al. (2001) that underlie the Hebrew SPMRL scheme http://www.spmrl.org/spmrl2013-sharedtask.html.

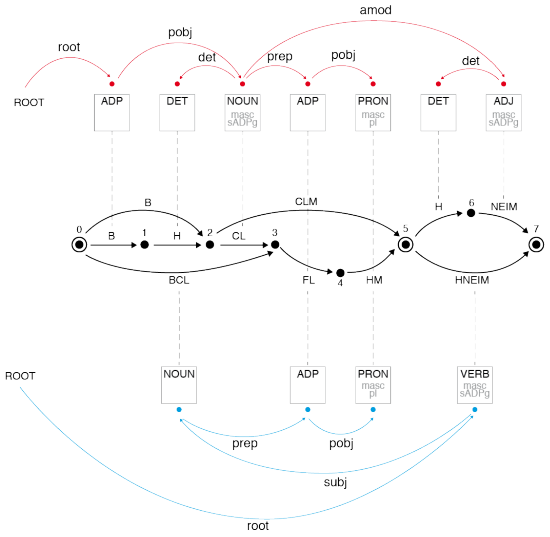

Figure 1: The Joint Morpho-Syntactic Search-Space. Lattice paths vary in length. Each lattice path can be assigned an exponential number of dependency trees.

In YAP we embrace the extreme morphological ambiguity in Hebrew. That is, we do *not* aim to resolve morphological ambiguity via pre-processing. The input to YAP is the complete *Morphological Analysis (MA)* of an input sentence x, termed here $\text{MA}(x)$. $\text{MA}(x)$ is a *lattice* structure, consisting of all possible morphological analysis possibilities of the input sentence, as seen in the middle of Figure 1. Each *lattice arc* is a tuple specifying the *start-index*, *end-index*, the *form* of the segment, its *part-of-speech, lemma, features*, and the *index* of the raw token the arc

originated from. An *arc* in the lattice can serve as a *node* in a syntactic dependency tree. Each contiguous path in the lattice presents one valid morphological segmentation of the sentence, for which a dependency tree can be assigned, as in Figure 1. For each path in the lattice, there is an exponential number of dependency trees that are potentially applicable.

We refer to the task of selecting the most likely lattice-path as *Morphological Disambiguation* (MD), and to the task of selecting the most likely dependency tree for a given path as *Dependency Parsing* (DEP). For an input sentence x, our goal is to *jointly* predict a single pair of $\text{MD}(x)$ and $\text{DEP}(x)$ that are consistent with one another, and form the most-likely analysis of the sentence.

The MD component is the transition-based *morphological parser* of More and Tsarfaty

(2016), which is formally based on the structure-prediction framework of Zhang and Clark (2011). MD accepts a sentence lattice MA(x) as input and delivers a selected sequence of arcs (morphemes) MD(x) as output. The transition-based system for MD selects arcs for MD one at a time. It decodes the lattice using beam-search, and keeps the K-best paths at each step, scored according to morpheme-level and token-level features, weighted via structured-perceptron learning.

The DEP component is a re-implementation of the Zhang and Nivre (2011) dependency parser for English, adapted for Hebrew. We assume an Arc-Eager transition system and beam-search decoding. Feature weights are learned via the structured perceptron. We employ a carefully-designed feature set that reflects linguistic properties of Hebrew such as its rich morphological paradigms, flexible word-order, agreement, etc. This provides SOTA results on Hebrew dependency parsing, albeit in Oracle (i.e., gold morphology) scenario.

Seen that both the MD and DEP realize the same formal framework and computational machinery, we can easily *unify* them and treat the morpho-synactic task as a single objective. The transition systems are combined and the beam-search decoder interleaves morphological and syntactic decisions. Now morphological decisions may be affected by syntactic content, and vice versa. The architecture is depicted in Figure 2. In More et al. (2019) we compared the performance of our joint system to our own pipeline, and to other parsing systems available for Hebrew. Our empirical results in More and Tsarfaty (2016); More et al. (2019) show significant improvements of YAP's joint model for both the morphological and syntactic tasks, over all standalone morphological or syntactic parsers available for Hebrew.

4 The Annotation Scheme

We deliver automatic morpho-syntactic annotation of Hebrew texts based on the scheme of the SPMRL Hebrew dependency treebank. The SPMRL Hebrew scheme employs the labels of Sima'an et al. (2001) for morphology and POS tags, and the Unified-SD scheme of Tsarfaty (2013) for the labeled dependencies.[2] Specifically, we deliver the following annotation layers:

[2]With an eye for future comparability, we further developed a conversion algorithm to convert the dependency tree from Unified-SD to Universal Dependencies (UD).https://universaldependencies.org/

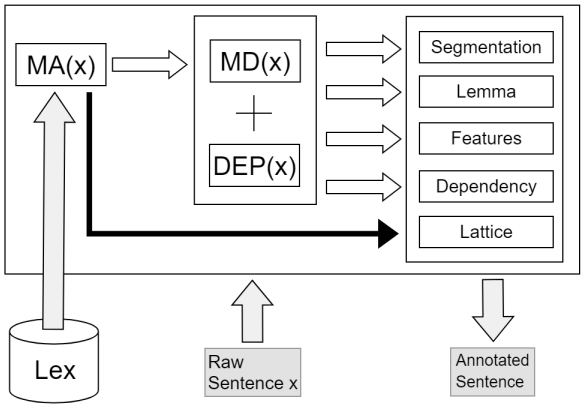

Figure 2: A bird's eye view of the Architecture

Morphological Segmentation The most basic form of analysis of Hebrew texts is the segmentation of raw tokens into multiple meaning-bearing units that we call *morphemes*.[3] Due to orthographic and phonological processes, some morphemes do not appear explicitly in the surface form. Our segmentation recovers all morphemes, both overt and covert. For example, the token 'בבית' (in the house) is segmented as 'ב' + 'ה' + 'בית'.

Part-of-Speech (POS) Tags Each morphological segment is assigned a single Part-of-Speech tag category that indicates its syntactic role. The set of tags used by the system is based on the SPMRL scheme which in turn adopts the POS labels from Sima'an et al. (2001) (detailed in our appendix).

Morphological Features Along with the POS category, we specify for each segment the properties that are signalled by inflectional morphology. The scheme encodes the following properties: **Number** [S (Singular) / P (Plural) / D (Dual)], **Gender** [F (Female) / M (Male) / F,M (both)], **Person** [1 / 2 / 3 / A (All)],[4] and **Tense** [Past, Present, Future, Imperative, Infinitive].[5]

Lemmas Each segment is also assigned a lemma, i.e., the cannonical representation of its core (uninflected) meaning.[6] For Hebrew nouns and adjectives, the lemma is chosen to be the Masculine-Singular form. For verbs, the lemma is in the Masculine-Singular-3per form in Past tense.

[3]In UD they are called *words*. In Hebrew NLP they are called *segments*. We use *morphemes* or *segments* herein.

[4]*A* is used in cases where all analyses are valid, such as in Beinoni form - 'אוכלת' (I/you/she eat.singular.feminine)

[5]Present-tense verbs and participles are tagged 'Beinoni'.

[6]Note that due to high morphological fusion in Hebrew, simple surface-based *stemming* will not suffice.

Dependency Tree The dependency tree is defined over all morphological segments and an artificial root node. It consists of a set of labeled binary relations that indicate the bi-lexical dependencies between segments. Note that the SPMRL dependency scheme, as opposed to UD, always selects *functional* heads, rather than lexical heads. The dependency labeling is based on the scheme from Tsarfaty (2013), repeated in the appendix.

Lattices As explained in section 3 above, a word can be segmented into morphemes in multiple ways, which are constrained by a broad-coverage lexicon. In addition to the parsed output, we make available for each input sentence its sentence lattice, i.e. the set of all possible segmentations for a given sentence, along with all possible morphosyntactic analyses for each arc.

5 Technical Details and Forms of Use

YAP is implemented in the Go language.[7] It requires 6GB of RAM to run, and employs a simple 3-step installation, given in the supplementray material. The input to the system is a tokenized sentence, with tokens appearing one per line, and a line break after every sentence.[8] The output is a dependency tree (where each node in the tree is a lattice arc) provided in the CoNLL-X format (Buchholz and Marsi, 2006). YAP is trained on the Hebrew section of the SPMRL shared task. It also makes use of the broad-coverage lexicon of Itai and Wintner (2008) for finding all potential lattice paths. In case of out-of-vocabulary (OOV) items, we employ a simple heuristic where we suggest the 10 most-likely analyses of rare tokens observed during training.

Simple Use | Command line From the command line, one can process one input file at a time, with a single sentence or more. The input file must be formatted with a single token per line, and an empty line denoting the end of every sentence.

Processing a file is done in 2 steps: First, run Morphological Analysis using `./yap hebma` to generate a sentence lattice containing all possible morphological breakdowns of each token. YAP will save the lattice to the file specified via the `-out` flag.

Now you can run joint Morphological Disambiguation and Dependency Parsing using `./yap joint` to jointly predict the best lattice path and corresponding dependency tree. The input to this command is the output file generated in the previous step, and there are 3 output files: one containing word segments, one containing the disambiguated lattice path, and one containing the complete dependency tree in CoNLL-X format.

Advanced Use | RESTful API YAP can run as a RESTful server that accepts parse requests. To do this simply start the server, listening on localhost port 8000. Now you can call the joint endpoint with a json object containing the list of tokens to process in the HTTP data payload. The response is a json object containing the three output levels (MA, MD and Dep). You can use jq and sed (or any other json and line processing tools) to format the (tab separated value) responses and reassemble the output. Check our appendix for an illustration.

Educational Use | The Online Demo In 2018 we decided to create an online demo of the system, for educational purposes: (i) To expose NLP/AI researchers to NLP capabilities available for Hebrew. (ii) To educate non-CS scientists and engineers who work with Hebrew data (e.g., digital humanities) on text annotations that can potentially be useful for their applications. (iii) To launch outreach activities where we teach *what is NLP* to the local community (e.g., school kids).[9]

To use the demo, simply go to `onlp.openu.ac.il` and type a Hebrew sentence in the textbox. The demo is built with Django and Bootstrap web frameworks. It sends the user's Hebrew text input to the ONLP server, which returns a CoNLL-X formatted parse along with the complete sentence lattice. Pre-processing includes pre-morphological tokenization of the input, where punctuation is being separated from the tokens. Double quotation marks are being separated from the word unless they appear before the last character of the word, to avoid over-segmentation of acronyms.[10] The tokenized sequence is then passed to the ONLP server. The CoNLL-X output is then processed into the following layers: the FORM column is concatenated and presented as "Segmented Text", and the POS, LEMMA, FEATS and DEPS are pre-

[7] https://golang.org/

[8] We assume the tokenization convention of MILA (Itai and Wintner, 2008).

[9] E.g., https://www.youtube.com/watch?v=TFwQeoKpznA&feature=youtu.be

[10] Acronyms in Hebrew are written with a quotation mark before the last letter, e.g. 'ב"ארה' (USA) .

	Tok	MA	MD	POS	Lem	Feats	Deps	Joint
Tasks								
MILA	✓	✓						
NITE	✓	✓	✓					
Hebrew-NLP		✓						
Adler			✓	✓		✓		
Goldberg							✓	
Pipelines								
UDPipe	✓	✓	✓	✓	✓	✓	✓	
CoreNLP	✓	✓	✓	✓	✓	✓	✓	
ONLP	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Existing Coverage for Hebrew NLP Tasks

sented in separate accordion tabs. Furthermore, the demo presents the sentence lattice which is the input to the joint parser. This is useful for debugging, and for analyzing lexical-coverage in out-of-domain scenarios.

Expert Use | Out of Domain Scenarios A bottleneck for the system in out-of-domain parsing scenarios is the coverage of the lexicon. We rely on a general-purpose lexicon containing over 500K entries. OOV words are treated via heuristics we designed, which are suitable for the general case only. However, identifying vocabulary items accurately may be critical when applying the parser to new domains with domain-specific information (medical, financial, political, etc.). Fortunately, we can extend the system with a domain-specific lexicon, thus extending the MA coverage. Due to joint inference, the availability of a better suited *lexical* analysis triggers better *lexico-syntactic* decisions on the whole.[11]

6 Related and Future Work

Hebrew NLP in general and Hebrew parsing in particular are known to be challenging, due to interesting linguistic properties, the scarcity of annotated data, and the small research community around. So, Hebrew has been seriously understudied in NLP. During the early 2000s, the MILA knowledge center was established, where the two of the main Hebrew resources for NLP were developed: the Hebrew treebank (Sima'an et al., 2001) and the Hebrew Lexicon (Itai and Wintner, 2008).

Morphological Taggers for Hebrew using local linear-context have been trained on these data and were made available for free use (Adler and Elhadad, 2006; Bar-haim et al., 2008). However, their performance was not on a par with parallel tools for English and thus insufficient for commercial use. Hebrew dependency parsing was initially provided by Goldberg and Elhadad (2009), but the parser provided *unlabeled* dependency, and the pipeline relied on Adler's morphological tagger. This left the predicted dependency trees inaccurate and unsatisfying. *Joint* morpho-syntactic models for constituency-based parsing based on Tsarfaty (2010) showed good performance on benchmark data, but was never released for open use.

With the development of the UD treebanks collection, general frameworks such as UDPipe (Straka et al., 2016) and CoreNLP (Manning et al., 2014) have been trained on the Hebrew UD treebank, and made the model available. However, these models provide performance that is still far from satisfactory. As we also demonstrate in our screen-cast,[12] these systems make critical mistakes, even with the simplest sentences. We conjecture that this is due to their inherent pipeline assumption: initial layers of processing present many mistakes. due to the extreme morphological ambiguity, and later layers cannot recover.[13] Notably, neural-network models utilizing word embeddings, (e.g., UDPipe) also lag behind.

Table 1 shows the task-coverage of existing tools and toolkits for NLP in Hebrew, academic as well as private initiatives (NITE,Hebrew-NLP). The task-coverage of the ONLP suite we present is on a par with international standards (UDPipe, CoreNLP), and its level of performance was shown to exceed all existing models (More et al., 2019). We are currently working towards Named-Entity Recognition as well as Open Information Extraction, to be added to ONLP in the near future.

7 Conclusion

This paper presents ONLP, a complete language-processing framework for automatic annotation of Modern Hebrew Texts. The framework covers morphological segmentation, POS tags, lemmas and features, and dependency parsing, predicted jointly. The system is easy to install and to use, and we support multiple forms of usage fitting user-personas with different needs. We hope the availability of an open-source, accurate, and easy-to-use system for NLP in Hebrew will benefit the local NLP open-source community and greatly advance Hebrew language technology research and development, in academia and in the industry.

[11] We discuss how exactly this is executed in the appendix.

[12] https://www.youtube.com/watch?v=H6pvh1x20FQ

[13] Our detailed qualitative error analysis in More et al. (2019) indeed confirms this conjecture.

Acknowledgements

We thank the NLPH community, in particular Shay Palachi, Amit Shkolnick and Yuval Feinstein, for discussion and insightful comments. We further thank Avi Bivas (Israel Innovation Authority) and Milo Avisar for promoting NLP initiatives in Israel. This research is supported by an ISF grant (1739/26) and an ERC Starting grant (677352), for which we are grateful.

References

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

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.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of CoNLL*, pages 149–164.

Yoav Goldberg and Michael Elhadad. 2009. Hebrew dependency parsing: Initial results. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 129–133.

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

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

Spence Green and Christopher D. Manning. 2010. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of COLING*.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

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

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings ACL: system demonstrations*, pages 55–60.

Amir More, Amit Seker, Victoria Basmova, and Reut Tsarfaty. 2019. Joint transition-based models for morpho-syntactic parsing: Parsing strategies for MRLs and a case study from modern Hebrew. *Transactions of the Association for Computational Linguistics*, 7:33–48.

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. The COLING 2016 Organizing Committee.

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).

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

Reut Tsarfaty. 2010. *Relational-realizational parsing*. Ph.D. thesis.

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

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 HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, SPMRL '10, pages 1–12, Stroudsburg, PA, USA. Association for Computational Linguistics.

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, Stroudsburg, PA, USA. ACL.

A Resources

1. Paper Appendix and Supplementary Materials:
https://arxiv.org/abs/1908.05453
2. Github:
https://github.com/OnlpLab/yap
3. Demo - Website:
http://onlp.openu.org.il
4. Demo - Screencast:
https://www.youtube.com/watch?v=H6pvh1x20FQ
5. API Docker Image:
https://hub.docker.com/r/onlplab/yap-api
6. SPMRL-to-UD Conversion:
https://github.com/OnlpLab/Hebrew_UD
7. ONLP Lab Website:
http://onlp.openu.org.il/home